

Table of Contents

User Manual

Working With NeoFPS

 Installation

 Getting Started

 Common Tasks

 Wizards

 Game Modes

 Generated Constants

 Layers and Tags

 Game Settings

 Origin Shift

 Extension Packages

 Reference

 MonoBehaviours

 ScriptableObjects

FPS Characters

 Spawning

 First Person Body

 Stamina

 Troubleshooting

 Reference

 MonoBehaviours

 Scriptable Objects

First Person Camera

 Aim Controllers

 Additive Transforms & Effects

 Cinemachine Extension

 Troubleshooting

 Reference

 MonoBehaviours

 ScriptableObjects

The Motion Graph

[NeoCharacterController](#)
[The Motion Graph Editor](#)
[Motion Graph States](#)
[Motion Graph Behaviours](#)
[Motion Graph Conditions](#)
[Motion Graph Parameters And Data](#)
[Ladders](#)
[Moving Platforms](#)
[Swimming](#)
[Motion Debugger](#)
[Troubleshooting](#)
[Reference](#)
 [States](#)
 [Behaviours](#)
 [Conditions](#)
 [MonoBehaviours](#)
 [ScriptableObjects](#)

[NeoFPS Input System](#)
[Input Settings](#)
[Touchscreen Input](#)
[Input System Extension](#)
[Creating Custom Input Handlers](#)
[Troubleshooting](#)
[Reference](#)
 [MonoBehaviours](#)
 [ScriptableObjects](#)

[Interaction With The World](#)
[Interactive Objects](#)
[Doors](#)
[Carry System](#)
[Troubleshooting](#)
[Reference](#)
 [AnimatedDoorHandle](#)
 [Carryable](#)
 [CarrySystemEncumbranceHandler](#)
 [CharacterInteractionHandler](#)
 [CharacterTriggerZone](#)

CharacterTriggerZonePersistant
DoorInteractiveObject
DoorTrigger
ElevatorController
ElevatorMovingPlatform
InteractiveObject
InteractiveObjectCornerMarkers
InteractiveObjectMaterialMarker
KeypadInteractiveObject
KeypadPopup
KeyRing
KinematicHingeDoor
KinematicHingeDoubleDoor
LockedDoorInteractiveObject
LockedDoorTrigger
LockedInteractiveObject
LockedTriggerZone
LockpickPopup3D
LockpickPopupUI
PhysicsHingeDoor
PickableLockedDoorInteractiveObject
PickableLockedInteractiveObject
RigidbodyCarrySystem
SlidingDoor
SoloCharacterTriggerZone
SoloCharacterTriggerZonePersistant
StandardCarrySystem
TriggerZoneColliderCounter

Audio Systems

Footsteps
Troubleshooting
Reference
AnimationEventAudioPlayer
AudioEffectsPreset
AudioTimeScalePitchBend
ClipSetContactAudioHandler
CriticalHitAudioPlayer

[FpsCharacterAudioData](#)
[FpsCharacterAudioEffects](#)
[FpsCharacterAudioHandler](#)
[NeoFpsAudioManager](#)
[SurfaceAudioData](#)
[SurfaceContactAudioHandler](#)

Inventory

[Inventory Examples](#)

[Troubleshooting](#)

[Reference](#)

[MonoBehaviours](#)

[ScriptableObjects](#)

Weapons

[Firearms](#)

[The Modular Firearm System](#)

[Hitscan vs Projectiles](#)

[Scopes & Optics](#)

[Attachments](#)

[Troubleshooting](#)

[Melee Weapons](#)

[Thrown Weapons](#)

[Wieldable Tools](#)

[Explosions](#)

[Troubleshooting](#)

[Reference](#)

[MonoBehaviours](#)

[ScriptableObjects](#)

Health and Damage

[Troubleshooting](#)

[Reference](#)

[ArmouredDamageHandler](#)

[BasicDamageHandler](#)

[BasicHealthManager](#)

[DamageZone](#)

[DeathAnimation](#)

[EventDamageHandler](#)

[GenericDamageTriggerZone](#)

[HealthPickup](#)

[HealZone](#)

[RechargingHealthManager](#)

[ShieldedArmouredDamageHandler](#)

[ShieldedDamageHandler](#)

[ShieldManager](#)

[ShieldPickup](#)

The Player HUD

Reference

[HudAdvancedCrosshair](#)

[HudAdvancedCrosshairStyleStandard](#)

[HudAmmoCounter](#)

[HudCarryObjectPopup](#)

[HudCrosshair](#)

[HudDamageMarkers](#)

[HudDeathPopup](#)

[HudEnemyHealthBar](#)

[HudFirearmMode](#)

[HudFirearmOverheatBar](#)

[HudHealthCounter](#)

[HudHider](#)

[HudHitDamageCounters](#)

[HudHitDamageCounterStandard](#)

[HudInteractionTooltip](#)

[HudInventoryItemCounter](#)

[HudInventoryItemMeter](#)

[HudInventoryStackedPC](#)

[HudInventoryItemStacked](#)

[HudInventoryStackedSlot](#)

[HudInventoryStandardPC](#)

[HudInventoryItemStandard](#)

[HudMotionGraphParameterMeter](#)

[HudMotionGraphParameterReadout](#)

[HudOxygenMeter](#)

[HudProgressBar](#)

[HudScope](#)

[HudShieldMeter](#)

HudShieldMeterStep
HudSlowMoCharge
HudStaminaBar
HudTargetLock
HudTargetLockMarkers
HudToggle
HudWieldableHasFlashlightCondition
HudWieldableHasModeSwitcherCondition
HudWieldableTypeCondition
HudWorldSpacePositionTracker
HudWorldSpaceTarget

Save Games

Serializing Data
Runtime Objects
Multi-Scene Saves
Overrides And Persistence
Troubleshooting
Reference

MonoBehaviours
ScriptableObjects

Graphics & Rendering

Built-In Pipeline Shaders
Universal Render Pipeline
High-Definition Render Pipeline
Troubleshooting
Reference

MonoBehaviours
ScriptableObjects

Samples

AI
UI
Reference

ApplyRandomDamage
CameraSeeker
DemoFacilityGameMode
DemoFacilityTarget
DemoFacilityTargetDamageTracker

[DemoInfoLaptop](#)
[DoorsDemoElevatorReadout](#)
[FiringRangeMovingTarget](#)
[FiringRangeReadout](#)
[FiringRangeSequencer](#)
[FiringRangeTarget](#)
[InfoPopupTrigger](#)
[KeypadPopup](#)
[LoadingScreen](#)
[MinimalDemoCharacter](#)
[OutOfBoundsRespawn](#)
[TurretSeeker](#)
[WaterZoneMover](#)

Utilities

[Reference](#)
[StateMachineBehaviours](#)
[MonoBehaviours](#)
[ScriptableObjects](#)

Surfaces

[Troubleshooting](#)
[Reference](#)
[MonoBehaviours](#)
[ScriptableObjects](#)

Welcome to the NeoFPS documentation

NeoFPS is a first person shooter asset and toolkit for the Unity game engine. Its goal is to enable you to create an FPS that matches your vision without restrictions. Designed to be flexible and extensible, NeoFPS can be the perfect starting point for your game.

Installation

For instructions on getting set up with NeoFPS, see the section titled [NeoFPS Installation](#).

Learning NeoFPS

More comprehensive documentation with more frequent updates can be found on the NeoFPS [documentation website](#).

For information on using NeoFPS in the Unity editor, see the [Manual](#).

For details on writing code for NeoFPS, see the [Scripting Reference](#).

You can find a selection of helpful tutorials [here](#).

Support

NeoFPS is a complex asset and as such there are bound to be occasional problems. If something seems broken or counter-intuitive, or if you have a suggestion that would make NeoFPS better for developers then please get in touch.

You can access the support page on the website at [NeoFPS Support](#).

Alternatively, you can email support@neofps.com.

Further sources of information

- [The NeoFPS Website](#) - tutorials, roadmap and support
- NeoFPS Unity Forum Thread - coming soon

Working With NeoFPS

Overview

NeoFPS is a framework for building PC and console based first person games.

Since it is impossible to cover all of the first person game types in a single asset, instead NeoFPS concentrates on the core mechanics of FPS games. Namely: movement, camera, interaction and shooting. It also provides implementations for features that Unity lacks out of the box, but which are especially important in first person games such as bindable controls, surface based effects, runtime settings and separating physics for shooting and movement.

NeoFPS was designed with the following goals:

- To help raise the quality bar of first person games created with the Unity game engine.
- To make it possible for a developer to achieve their vision of a first person game without placing constraints on how the game plays.
- To make the more complex mechanics of first person shooters accessible with a minimum of coding required.
- To allow developers to choose which features to use and which to replace without having to modify the code.

Installation

NeoFPS is a complex asset that requires a number of custom settings and has dependencies on other packages. Please see the [Installation Instructions](#) for a guide to getting NeoFPS up and running.

Getting Started

If this is your first time using NeoFPS, take a look at the [Getting Started](#) section for details on where to start. The [Common Tasks](#) section runs through some of the early tasks that are frequently raised in the [NeoFPS Discord server](#) and can point you in the right direction for the steps and documentation required.

Generated Constants

To meet the goals of enabling developers to achieve their vision and create complex mechanics without rewriting code, a flexible way of referencing objects and states is required. It would be easy to implement something using strings for keys, but the performance and memory impact of this can be a problem if the system is used in a lot of places or many times a frame. NeoFPS gets around this by providing a simple interface for creating constants similar to Unity's layers. These constants are turned into code that makes the constants easy to use in scripts as well as using them for properties in the inspector.

For more information see [Generated Constants](#).

Layers and Tags

NeoFPS makes heavy use of [Unity layers](#) to enable complex features and to separate out scene elements for better performance.

For more information see [Layers and Tags](#).

Game Settings

NeoFPS exposes a number of runtime settings to the player through text based settings files.

For more information see [Game Settings](#).

See Also

[NeoFPS Installation](#)

[Getting Started](#)

[Generated Constants](#)

[Layers and Tags](#)

[Game Settings](#)

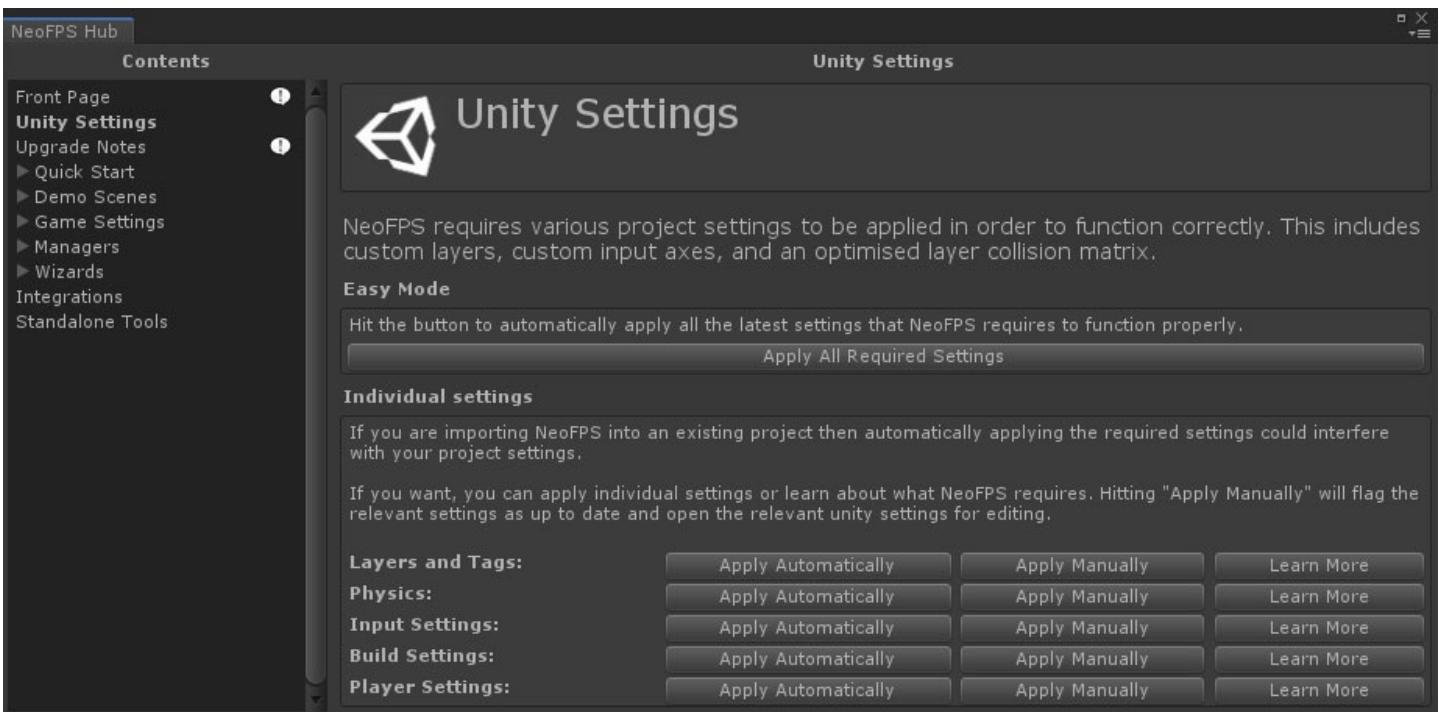
NeoFPS Installation

Overview

On first adding NeoFPS to a project, once the assets have been imported, the NeoFPS hub should appear, guiding you through the installation and providing useful information on [getting started](#). This document covers any extra steps and gives details of what is happening in case you want more manual control of the installation process.

Unity Settings

NeoFPS requires various project settings to be applied in order to function correctly. This includes custom layers, custom input axes, and an optimised layer collision matrix. NeoFPS has an automated system for applying Unity settings built into the NeoFPS Hub. It uses version numbers to track when updates to NeoFPS require new settings to be applied and will open the hub on the **Unity Settings** page if they are not up to date. You can also find the hub in the toolbar via *Tools/NeoFPS/NeoFPS Hub*.



You can apply all required settings automatically here. If you are installing NeoFPS in a fresh project then this is the recommended approach. If you would prefer to apply the correct settings manually then you can do so using the **Individual Settings** section. Clicking the **Apply Manually** button for a section will open the relevant Unity settings editor and flag the settings as up to date in the NeoFPS hub. The following is a breakdown of what is required for each section:

SETTINGS	DETAILS
Layers and Tags	NeoFPS makes extensive use of custom layers. For more information, see the Layers and Tags documentation.
Physics	NeoFPS uses the physics layer collision matrix to define interactions between the custom layers. For more information see the Layer Collision Matrix section of the Layers and Tags documentation. This must be changed after the layers and tags have been set up.
Input	To enable consistent controller mapping across platforms, NeoFPS has some fairly complex input settings. You can find a full list in the Input Settings documentation.
Player Settings	NeoFPS uses the Linear color space (this will be set to "Gamma" by default). Without this change, the sample scenes will look very washed out and bright.

SETTINGS	DETAILS
Build Settings	See the <i>Sample Scenes</i> section below.

Sample Scenes

Sample scenes that demonstrate NeoFPS' features can be found in the project folder: *NeoFPS/Samples/SinglePlayer/Scenes*

The **MainMenu** scene is intended as a jumping off point for the individual demos and needs to be set as the first scene in the build settings (index 0). This means it will be the start-up scene for standalone builds. The **Loading** scene is a loading screen that is displayed when loading demo scenes. By default, this must be the second scene in the build settings (index 1), though this can be changed via the Neo Scene Manager in the hub.

The **Unity Settings** page applies the above settings automatically as part of the **Build Settings** section.

Render Pipeline Selection

NeoFPS is set up with demo assets for the built-in render pipeline out of the box. It also comes with extension packages with shaders and replacement assets that support the universal and high definition render pipelines. For more information, see [Graphics and Rendering](#).

Runtime Managers

By default, NeoFPS will instantiate runtime versions of its managers in the *DontDestroyOnLoad* scene as soon as you enter play mode or run your game. If you want to prevent this from happening (for example, if you have imported NeoFPS into a test project with a number of other assets and you want to try each one in isolation), then you can add the **NEOFPS_LOAD_ON_DEMAND** scripting define to your Unity player settings. This will delay instantiation until the first time the manager is accessed. In some cases this can cause a slight frame hitch as the manager initialises, so it is advised not to use this in a final project.

See Also

[Layers and Tags](#)

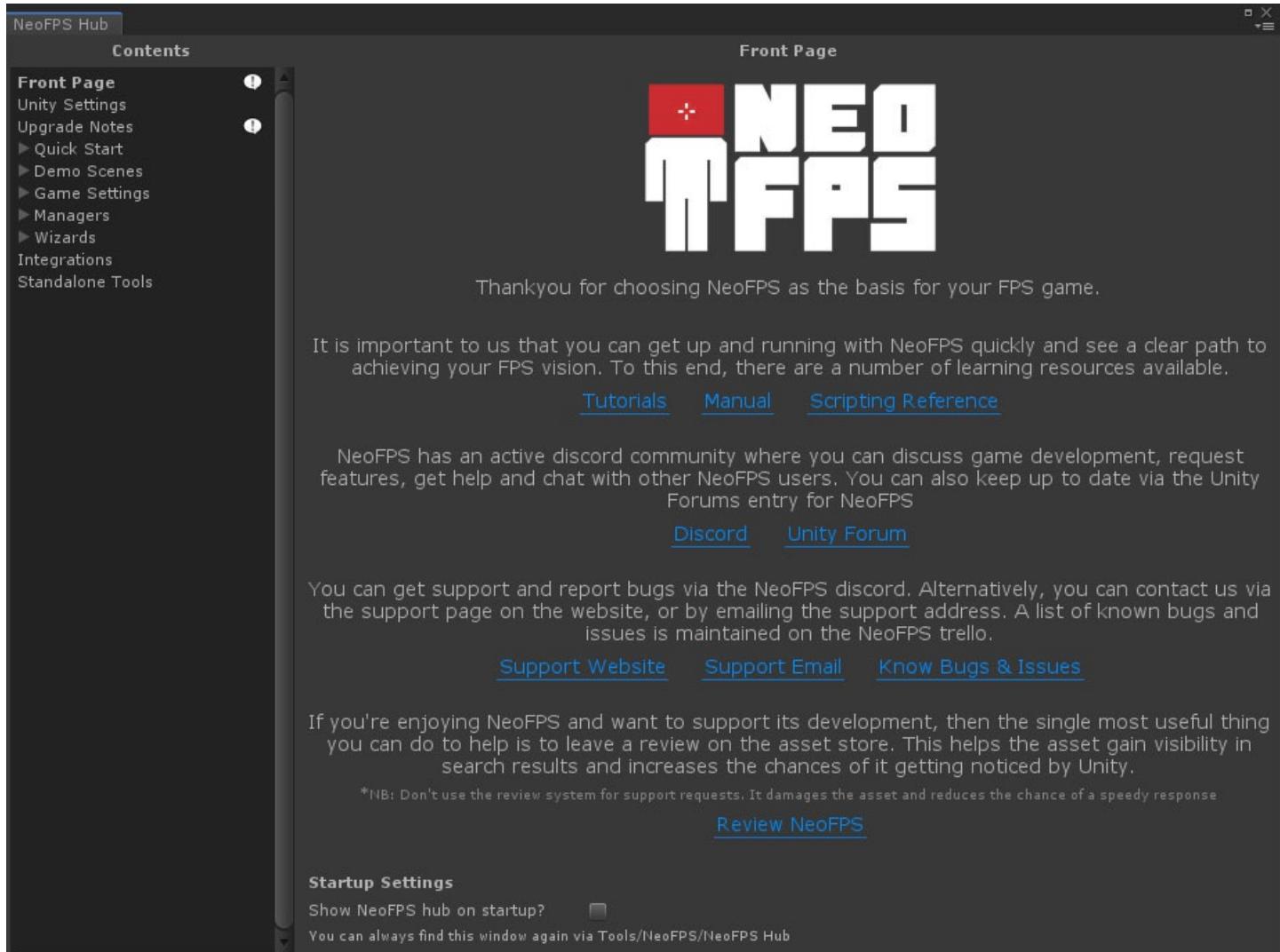
[Input Settings](#)

[Graphics and Rendering](#)

Getting Started

NeoFPS Hub

On first adding NeoFPS to your project, once the assets have been imported, you should see the NeoFPS hub window appear:



The hub will notify you of any changes to the required settings that must be applied (for more information see [NeoFPS Installation](#)). It also provides a quick way of accessing the sample scenes along with helpful links for getting the most from NeoFPS.

The hub pop-up will only be shown automatically this first time. If you want to see it again then it can be found in the toolbar via [Tools/NeoFPS/NeoFPS Hub](#). Alternatively you can check the **Show hub on startup** checkbox at the bottom of the front page, and it will be shown each time the project is loaded.

Down the left-hand side of the hub is a navigation pane that lets you choose the page to view. The hub is split into the following sections:

- The **Front Page** contains useful links to aid with learning NeoFPS, getting support and engaging with the NeoFPS community.
- **Unity Settings** is used to apply the Unity settings that NeoFPS requires.
- **Upgrade Notes** tracks the installed version of NeoFPS and describes the steps required to upgrade to the latest update (both in scripts and components)
- **Quick Start** contains a selection of readme sub-pages which aid in getting up and running in NeoFPS, link to the relevant docs, and highlight useful assets and folders.
- **Demo Scenes** lists the available demo scenes and provides a quick way to find and load them. Once loaded, the **Scene Info** sub-page displays a readme for the scene with links to relevant docs and scene or project items. You can also find an

object in each demo scene called **Readme** which contains this information.

- **Game Settings** organises the player facing game options, allowing you to change default / starting values or delete your local settings files. These are the options the player can access through the in-game menus such as volume settings, mouse sensitivity, etc.
- **Managers** organises the manager settings for the various NeoFPS systems.
- **Wizards** provides a series of tools for quickly creating and setting up items in NeoFPS. For more information, see [NeoFPS Wizards](#).
- **Integrations** lists the available integrations for NeoFPS and provides links to the asset store and GitHub repositories. For the latest integrations, see the [integrations page](#) on the website.
- **Standalone Tools** lists the non-hub based tools available in NeoFPS such as the motion graph editor, the motion debugger, and the save file inspector.

If you would prefer a video guide to the hub and setting up NeoFPS in a project, I have created the following youtube video:

Project Structure

All NeoFPS assets are contained in the **NeoFPS** folder. Inside this there are 4 subfolders:

- **Constants** contains all the [Generated Constants](#) used by NeoFPS and its samples.
- **Core** contains all the code and assets required by NeoFPS.
- **Resources** contains the managers and settings assets that need to be loaded at runtime.
- **Samples** contains a number of sample assets and scenes that demonstrate the different features of NeoFPS and provide a reference for implementing your own contents and mechanics. For more information, see [Samples](#).

Using NeoFPS In Your Game

The **Scene Setup Quick-Start** guide available in the hub details the required scene components and links to the relevant prefabs in the project. For a video guide to the different elements of a NeoFPS scene, you can watch on youtube:

The quickest way to start prototyping in NeoFPS is to use the template scene as a starting point. You can find the template scene at the following location:

`Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Template\FeatureDemo_Template.unity`

You can either duplicate this scene as a starting point, or alternatively, if you already have a scene that you want to add then you can use Unity's multi-scene editing features to open the template scene additively in order to copy the relevant objects across. With your starting scene selected, open the template scene additively by right clicking on the scene asset and selecting the **Open Scene Additive** option. Both scenes will now be shown in the hierarchy and you can copy objects from the template scene and paste them in your own scene. Once you are done, remember to close the template scene by clicking on its options dropdown in the hierarchy and selecting **Remove Scene**.

The important objects in the demo scene are:

- **SimpleSpawnerAndGameMode** is a prefab game setup that handles player spawning and death (explained below). You can also find this prefab at `Assets\NeoFPS\Samples\SinglePlayer\Prefabs\SimpleSpawnerAndGameMode.prefab`
- **GlobalProfile** is a post-processing profile. You can use this or create your own.
- **SceneSaveInfo** is required in any scene that you want to support the NeoFPS save game system. This is not a prefab as its settings are unique to each scene. You will also need to add the scene to the build settings.
- **ScenePoolHandler** is used by the pooling system to manage pooled objects. This will be created automatically if not found, but including one in your scenes leads to a smoother experience.
- **HudAndMenuCanvas** contains the ingame menu and player HUD
- **HudAndMenuEventSystem** contains a Unity UI event system that handles input in the HudAndMenuCanvas object above.

In NeoFPS, the player character is not usually placed directly in a scene. Instead it is spawned at runtime from a spawn point and attached to a player object. This system allows for features such as swapping characters and respawning while persisting player information. The **SimpleSpawnerAndGameMode** prefab listed above contains the relevant spawner, along with a minimal game mode called **FpsSoloGameMinimal**. This component references the character object and player object to spawn. For more information, see [Game Modes](#) and [FPS Characters](#).

The default demo character can be found at `Assets\NeoFPS\Samples\SinglePlayer\Prefabs\NeoFpsSoloPlayerCharacter.prefab`. You can also find a spawnerless character called `PrototypeSpawnerlessCharacter.prefab` at the same location. This can be placed directly in a scene instead of the SimpleSpawnerAndGameMode object, but it is limited in how it can handle player death (it defaults to reloading the scene).

To add environmental features to your scene, see [Layers and Tags](#) for details on what layers to use for environmental geometry, and [Surfaces](#) to add surface audio and impact effects.

For information on how to set up weapons for your game, see [Weapons](#).

To add interaction to your game such as switches, doors and puzzles, see [Interaction](#).

To add your own characters or customise the sample character, see [FPS Characters](#).

NeoFPS is also designed to be extended to suit your vision. Each of the features is implemented in such a way that it can be replaced or expanded. For more information see [Extending NeoFPS](#).

See Also

[NeoFPS Installation](#)

[Samples](#)

Common Tasks

Overview

The following are tasks that are frequently raised by new users on the [NeoFPS Discord server](#). More will be added over time based on user feedback.

How To Interact With Objects In A Scene

NeoFPS lets you walk up to objects in a scene and use them or pick them up with the interact button using the [Interactive Objects](#) system.

There is an [interactive object creation wizard](#) available in the NeoFPS Hub that can create new interactive objects and set them up with render geometry, physics, sound effects and animations. Once you have an object set up, you would use the events on the interactive object behaviour to trigger your own scripts or components. Alternatively you can create a script that inherits the `InteractiveObject` class and extend its `Interact()` method.

One example of an interactive object is the scene switcher buttons in the persistence demo. The use the interactive object events to trigger a `NeoFpsSceneSwitcher` component when pressed, changing the scene.

How To Remove Weapons From A Player Character And Add Your Own

The player character's loadout is specified in the inventory component attached to the root of the prefab. There are 3 inventory types implemented in the demos: `FpsInventoryQuickSwitch`, `FpsInventoryStacked` and `FpsInventorySwappable`. Each of these have a **Starting Items** list in the inspector. You can add items to this as required. The **Backup Item** is the inventory item that is used when all others have been dropped, or when the selected item is holstered. By default, this is the demo hands, but it can be switched to any inventory item you like.

You can also create inventory loadout assets in the project hierarchy using right-click: *Create/NeoFPS/Inventory/Loadout*. These are assigned to the game mode component in your scene (for example the `FpsSoloGameMinimal` component). When the game mode spawns a character, it will assign it that loadout instead of its default starting loadout. The backup item (hands) will remain the same. See [FpsInventoryLoadout](#) for more details.

How To Add New Item Keys To The Inventory

Inventory items are each identified using a unique key. These keys are stored in the [inventory database](#) which is divided into tables. When adding your own inventory item keys, the first thing you should do is create a new `FpsInventoryDbTable` for your project by right-clicking in your project folder and using *Create/NeoFPS/Inventory/Database Table*. You then need to add the table to the inventory database by opening the database in the inspector (you can find it in the NeoFPS Hub managers section, or in the `NeoFPS/Resources` folder) and dragging your new table asset into the **Add Table Asset** field at the top. This means that there is no risk of the keys you add being overwritten when updating the asset (though you might need to re-add your table to the database).

Once you have an inventory database table for your project, there are a number of ways you can add a key. Firstly, you can select the table asset and use the controls at the bottom of the inspector to add a new key. Secondly, you can select the inventory database itself in the hub or resources folder. Clicking on your project table in the database table list will show its contents below. Lastly, the most convenient approach is to add the key at the moment you need it. Whenever a component has an inventory key property that you can set in the inspector, it will pop up the inventory browser when clicked. At the bottom of this is a section to add a new key. You must select your project's table, insert a name for the item, and click create. This will add the key to the selected inventory database table, and assign it to the property that opened the browser.

Renaming a key in the asset database will not break connections. Any items that use that key will use the new name automatically.

For more information see the [Inventory](#) section.

How To Make Player Weapons, Items And Health Persist Across Play

Sessions

NeoFPS has a full [save game system](#) built in which can also be used to save and load persistent data to memory between scene switches.

The save system revolves around the [NeoSerializedGameObject](#) component which must be added to each GameObject with components that need saving (including its transform). You can tell the NeoSerializedGameObject which child objects, components and properties to save as required. You can also add overrides for different save modes such as **persistence**. You would usually save less information for persistence data. For example, you don't want to save the position of the character or the state of the selected weapon's animations. You do want to save information such as their health though. By default, all components and child objects will be saved.

If you have your own scripts or components that need saving then you can do it one of two ways: Firstly, you can modify your script to inherit from the `INeSerializableComponent` interface and implement its `ReadProperties` and `WriteProperties` methods. If you don't have access to modify the scripts then you can write formatters that read/write them to the save system's binary format. For more information see [Serializing Data](#).

How To Make Your Own Stat System Affect Movement And Combat

The NeoFPS [motion graph](#) is the movement system that drives the player characters. Each movement state can be set to use motion data such as movement speed, jump height or directional multipliers. This motion data can be overridden by assigning an object to the motion controller that implements the `IMotionGraphDataOverride` interface. The motion graph has a data override built in that just lets you assign different values in an asset, but you can also use MonoBehaviours that change the values at runtime. This allows you to base speed on character level or attributes, or add buffs and debuffs.

For stat based health effects, you would need to write a behaviour that implements the `IHealthManager` interface, or inherit from one of the existing health managers and add your extra stat based behaviour.

To modify the way firearms work based on stats, you can create your own ammo effect module scripts that deal damage based on the shooter's stats, or inherit from the existing `BulletAmmoEffect` to do the same thing.

For more information on extending NeoFPS, see the [Extending NeoFPS](#) section of the docs (online only).

How To Control Where The Player Spawns

In NeoFPS the player character is spawned in to the scene using [spawn points](#). Spawning is controlled via the [game mode](#) along with what action to perform on the character's death. The demo scenes use a prefab called the *SimpleSpawnerAndGameMode* that has a game mode and a spawn point built in.

There's a few ways that you can control where the player character spawns. By default, spawn points register with the game mode and spawn system on Awake. If multiple spawn points are registered, you can choose if they will be used in the order they were registered or at random. You can also add an [OrderedSpawnPointGroup](#) to enforce the registration order.

To track progress through a level, you can enable and disable spawn points as you reach checkpoints. Combining this with the [save game system](#), you can add autosaves when reaching the checkpoint. If the spawn points each have a [NeoSerializedGameObject](#) attached then their active state will be saved, meaning that the character will spawn at the correct spawn points when the game is re-loaded.

If you want a much simpler system for the sake of prototyping, then you can also create a spawnerless character that can be placed directly in the scene, though this is a much less flexible setup. For more information see [spawning](#).

See Also

[NeoFPS Wizards](#)

[Samples](#)

NeoFPS Wizards

Overview

NeoFPS comes with a number of item creation wizards to simplify the setup of common objects such as characters and weapons. These can be found in the **Wizards** section of the [NeoFPS Hub](#), which is available through Unity's menus at *Tools/NeoFPS/NeoFPS Hub*. Each wizard (with the exception of the script creation wizard) allows you to create an object and/or save a template for use when creating your next weapon.

Available Wizards

The following wizards are available to use now:

Modular Firearm Wizard

The modular firearm wizard runs through the setup of a [modular firearm](#), from assigning a view model to picking the relevant firearm modules, and even adding advanced features such as weapon overheat, bullet penetration and sprint animations. The wizard does not set up an animator controller for the weapon, but if you have one available then you can attach it and hook up the parameters through the wizard.

Player Character Wizard

The player character wizard allows you to build a [playable character](#) prefab. This includes setting up health, inventory and more advanced features such as stamina. The end result can be added to a spawner, and spawned at runtime immediately after setup.

Melee Weapon Wizard

The melee weapon wizard runs you through the creation of a basic [melee weapon](#) prefab for use in NeoFPS. This includes setting up the weapon itself, how it appears in the inventory, and more advanced settings such as sprint animations. The wizard can also create a new animator controller for you based on the weapon's settings, allowing you to pick specific animation clips to use. Alternatively, you can hook up an existing animator controller so the correct parameters are used.

Thrown Weapon wizard

The melee weapon wizard runs you through the creation of a basic [thrown weapon](#) prefab for use in NeoFPS. This includes setting up the weapon itself, how it appears in the inventory, and more advanced settings such as sprint animations. The wizard can also create a new animator controller for you based on the weapon's settings, allowing you to pick specific animation clips to use. Alternatively, you can hook up an existing animator controller so the correct parameters are used.

Pickup Wizards

The pickup wizard is used to create pickups / powerups that the player character can use. The available pickup types are:

- Wieldable items such as melee or thrown weapons.
- Modular firearm drops (these track the ammo in the weapon when it is dropped).
- Inventory item pickups.
- Multi-item inventory pickups.
- Health packs.
- Shield boosters.

Interactive Object Wizard

The [interactive object](#) wizard allows you to add NeoFPS' interaction system to objects to allow the player character to use them. This includes assigning geometry and physics, highlighting when looked at, as well as some preset interactions such as playing animations or audio. You can then use the events on the created prefabs' `InteractiveObject` component to trigger your own code or components.

Script Creation Wizard

The script creation wizard provides a number of template scripts for various NeoFPS features that you can use as a starting point. This includes motion graph elements, firearm modules, save system formatters, and input behaviours. To use the wizard:

- Select the script type from the dropdown.
- Give it a name and a namespace.
- Fill in any script specific properties that appear.
- Select output folders for the generated scripts. If these are not set, then the new scripts will be created in the root of the *Assets* folder, while any editor scripts will be created in *Assets/Editor*.

Coming Soon

The following wizards are coming soon:

- **Firearm AnimatorController Wizard** will take a modular firearm prefab, and create and set up an AnimatorController based on the modules it has attached.

See Also

[Getting Started](#)

Game Modes

Overview

Game modes in NeoFPS are intended to control the flow of a game. This means creating a player object, spawning and respawning characters and attaching them to the player. They are derived from the `FpsGameMode` base behaviour which is an abstract class and therefore cannot be placed in the scene without a specific implementation.

In this version of NeoFPS the majority of the demos use a game mode called `FpsSoloGameMinimal`. This implementation simply spawns both a player and character from prefabs on start. On a character's death, the `FpsSoloGameMinimal` can be set to either spawn a new character, reload the scene, return to the main menu, or load the last valid save game (for more information on save games, see [Save Games](#)). The minimal implementation is designed for testing and development of prototypes and as a template for implementing a custom game mode for your own game.

The game mode also handles persisting data such as player health and inventory between scenes, also using the [save game system](#). This can be toggled on or off.

Customizable Game Modes

With some scripting you are able to customise the behaviour of the game modes at various points. For example:

- You can change where the game mode gets the player or character prefab to spawn
- You could modify the character's starting inventory using a loadout
- You can hook into the player character's death to display a UI or change the respawn behaviour

For full control over the game mode, you can inherit a custom class from the `FpsSoloGameBase` behaviour class. This has a number of virtual and abstract methods and properties that influence it's behaviour. For more information, you can see the scripting reference for this class [here](#).

There is also a version of the game mode called `FpsSoloGameCustomisable` that inherits from this and ties into the sample UI system to show a pre-spawn popup. This allows you to add tabs to the popup for functionality such as selecting a loadout, creating a custom loadout, selecting a character, or picking a spawn point in the scene. This is used as the basis for the `DemoFacilityGameMode`. Working with this system does require you to script the desired behaviour, because a componentised version that can be assembled through the editor and adapted to any game design would be a huge undertaking.

See Also

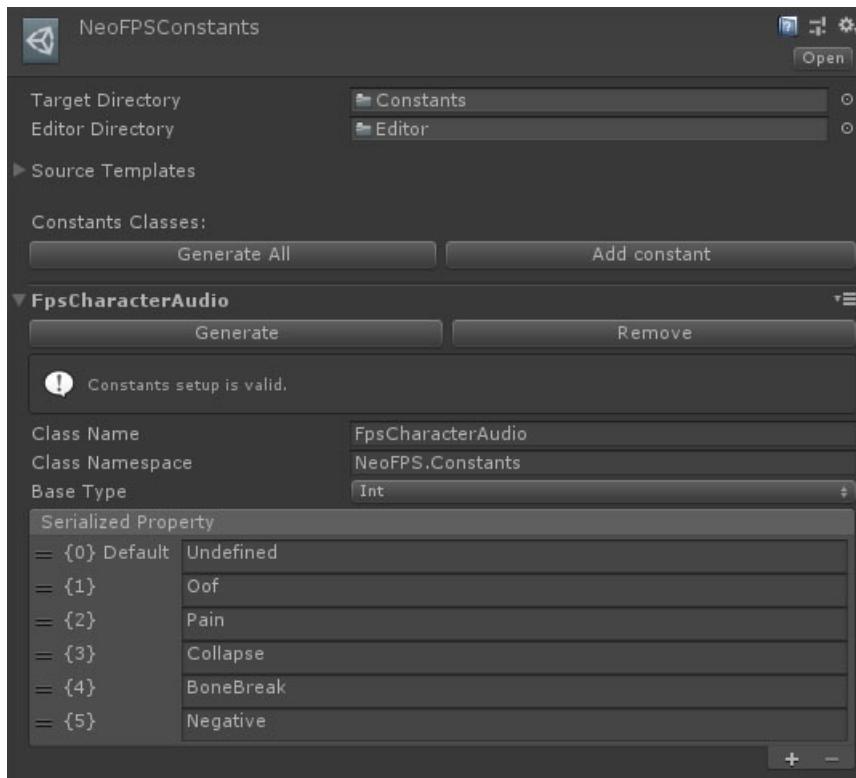
[Working With NeoFPS](#)

[Save Games](#)

Generated Constants

Overview

To meet the goals of enabling developers to achieve their vision and create complex mechanics without rewriting code, a flexible way of referencing objects and states is required. It would be easy to implement something using strings for keys, but the performance and memory impact of this can be a problem if the system is used in a lot of places or many times a frame. NeoFPS gets around this by providing a simple interface for creating constants similar to Unity's layers. These constants are turned into code that makes them easy to use in scripts as well as using them for properties in the inspector.



The [ConstantsSettings](#) scriptable object specifies constants to generate for your game. You can have as many of these settings files as you like and add new constants as required. If you do plan to add your own constants then it is best to create a new constants settings file in case the one that comes with NeoFPS is modified in a future version. It is also important to consider whether it is best to use a constant or a string. If the situation involves a set of keys or IDs that are referenced frequently, but will rarely be changed then generated constants are a good fit. If there is a high chance that new keys will be needed or the keys will be changed regularly, then it might be better to use strings for the ID and a dictionary to store items.

Generated Constants Limitations

The biggest limitation with generated constants is that they are serialized by index. This means that reorganising the values will not be reflected in the inspector.

As an example, let's say you have a **Vehicle** constant with the following values:

1. Bike
2. Car
3. Truck

You also have a monobehaviour that stores a vehicle as a serialized field and in the inspector it is set to **Car**. Later on you decide to add a number of new values as follows:

1. Bike
2. Motorbike
3. Car

4. Van
5. Truck

Looking at the previous monobehaviour in the inspector, it would now be set to **Motorbike**. This is because the monobehaviour only actually references the number 2, not the word "Car". You can work around this by making sure to keep the order the same and only add new values to the end.

If you remove values instead of adding, then any serialized properties that reference values beyond the new limit will be reset to the first constant value. It is best practice to reserve the first value (0) as a default with a name that reflects this.

Generated Constants In Scripts

You can use constants in scripts as though you were working with an enum. For example, the `FpsInputAxis` constant can be used like so:

```
FpsInputAxis axis = FpsInputAxis.MouseX;
```

Constants can also be implicitly cast as in the following examples:

```
FpsInputAxis axis = FpsInputAxis.MouseX;  
var element = inputAxisArray[axis];
```

and:

```
FpsInputAxis axis = 2;
```

This makes them very efficient as keys to store items. If you were using strings, you would need to store and reference objects from a dictionary as follows:

```
var dictionary = new Dictionary<string, MyObject>();  
  
//...  
  
MyObject result = dictionary[myStringKey];
```

This is a relatively expensive operation as the string needs to be hashed, and then the dictionary searched for the key value pair that the key references. If the key is not found then this code would throw an exception so you would also want to add extra error checking for safety. With constants the object can be stored and referenced in an array as follows.

```
var array = new MyObject[MyConstant.count];  
  
//...  
  
MyObject result = array[myConstantKey];
```

This is a much faster operation as you are simply accessing an array by index. By allocating an array that is the size of the constant (count is a preset property in the constants templates included with NeoFPS) you are guaranteed that accessing via a constant key will never be out of bounds. Preallocating like this is a good speed optimisation, but if the constant has a lot of values, and the array is sparsely populated then it would be wasteful to implement in this way. If so then you could either use the string option, or a dictionary with constant keys to save the string hashing step.

How They Are Generated

constants are generated by taking a template script and replacing sections with values in the settings file. The following keys are defined which will be replaced:

KEY	DESCRIPTION
%NAME%	The generated constants container name.
%NAMESPACE%	The namespace for the generated constants container.
%TYPE%	The underlying type for the value (for example int , ushort , byte).
%VALUES%	The values to be added. These will be written as a number of const values of the specified type.
%VALUE_NAMES%	An array of strings to be populated with the value names. This is used for the inspector value dropdown among other things.
%COUNT%	The number of values written.

The generation process requires 2 templates. One for the output constant, and one for an editor drawer that draws the constant in the inspector as a dropdown.

See Also

[ConstantsSettings](#)

Layers and Tags

Overview

NeoFPS has a number of systems that require objects to be filtered using layers. The following is the layer set-up required by NeoFPS, along with an overview of some useful features for working with layers in code.

NeoFPS Layers

Layers	
Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	PostProcessingVolumes
User Layer 9	EnvironmentRough
User Layer 10	EnvironmentDetail
User Layer 11	MovingPlatforms
User Layer 12	DynamicProps
User Layer 13	CharacterControllers
User Layer 14	CharacterFirstPerson
User Layer 15	CharacterExternal
User Layer 16	CharacterPhysics
User Layer 17	CharacterRagdoll
User Layer 18	CharacterNonColliding
User Layer 19	WieldablesFirstPerson
User Layer 20	WieldablesExternal
User Layer 21	TriggerZones
User Layer 22	InteractiveObjects
User Layer 23	Doors
User Layer 24	SmallDynamicObjects
User Layer 25	Effects
User Layer 26	AiVisibility
User Layer 27	
User Layer 28	
User Layer 29	
User Layer 30	
User Layer 31	

NAME	DESCRIPTION
PostProcessingVolumes	Trigger volumes used by the Unity post processing system to define override volumes.
EnvironmentRough	Low detail mesh and primitive colliders used for character motion and traversal.
EnvironmentDetail	High detail mesh and primitive colliders used for weapon impacts.
MovingPlatforms	Low detail mesh and primitive colliders used for character motion on moving platforms.
DynamicProps	Dynamic rigidbody props. Used for larger objects that could affect characters such as barrels.
CharacterControllers	Used for character controller root objects.
CharacterFirstPerson	Character geometry and objects visible from the first person view.
CharacterExternal	Character geometry and objects visible from the external views.

NAME	DESCRIPTION
CharacterPhysics	Character body colliders, used for things like bullet impact detection.
CharacterRagdoll	Character body colliders used for ragdolls, so set up to collide against the ground but not detail physics.
CharacterNonColliding	Used for objects that are tested against using Physics casts, but not the Unity collision system.
WieldablesFirstPerson	First person geometry and colliders for weapons and other wieldable objects.
WieldablesExternal	Geometry and colliders for weapons and other wieldable objects when seen from an external view.
TriggerZones	Trigger volumes that act on character triggers.
InteractiveObjects	Low detail trigger volumes used for detecting interactive objects.
DoorPhysics	Colliders for door objects.
SmallDynamicObjects	Small rigidbody objects that should not noticeably affect characters. A character can push them around, but they can't really push a character.
Effects	A layer used for debris and particle effects.
AIVisibility	Low detail colliders used by AI for visibility checks.

NeoFPS Tags

NAME	DESCRIPTION
AI	Used to tag AI characters.

Layer Collision Matrix

	Default	TransparentFX	Ignore Raycast	Water	UI
Default	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TransparentFX	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ignore Raycast	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Water	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
UI	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
PostProcessingVolumes	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
EnvironmentRough	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
EnvironmentDetail	<input type="checkbox"/>				
MovingPlatforms	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
DynamicProps	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CharacterControllers	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CharacterFirstPerson	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CharacterExternal	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CharacterPhysics	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CharacterRagdoll	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CharacterNonColliding	<input type="checkbox"/>				
WieldablesFirstPerson	<input type="checkbox"/>				
WieldablesExternal	<input type="checkbox"/>				
TriggerZones	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
InteractiveObjects	<input type="checkbox"/>				
Doors	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SmallDynamicObjects	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Effects	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AiVisibility	<input type="checkbox"/>				

The layer collision matrix is used to define which layers can interact with each other in the Unity physics system. Keeping a minimum of layers from colliding with each other is important to maintaining performance as well as for keeping interactions between different systems clean.



In the above example image, the green objects are character blockers on the CharacterRough and EnvironmentRough layers that are used for moving characters around the scene. The red objects are bullet blockers on the EnvironmentDetail layer that are used for bullet hits.

In code you can use the `PhysicsFilter` type to define filters when performing physics operations. This also has a number of preset constants for aiding in writing code.

`PhysicsFilter.LayerIndex` contains the layer indices matching the layers as specified in the layers and tags settings.

`PhysicsFilter.LayerFilter` contains a number of preset physics filters matching the layers as specified in the layers and tags settings.

`PhysicsFilter.Masks` contains a number of filters for commonly used groups of layers as follows:

NAME	DESCRIPTION
BulletBlockers	Layers that bullet raycasts can hit.
CharacterBlockers	Layers that characters traverse. Includes larger props and platforms.
DynamicCharacterBlockers	Layers that characters traverse. Excludes static environment physics.
Interactable	Colliders attached to interactive objects, along with environment colliders that can block them from view.
SpawnBlockers	Any dynamic objects that can block spawn points.
ShowDecals	Objects that can accept decals.
AiVisibilityCheck	AI visibility markers and environmental colliders that can block them from view.

See Also

[Unity Tags and Layers](#)

[Unity Layer Based Collision](#)

Game Settings

Overview

NeoFPS exposes various game settings to players through text based settings files. This provides much more flexibility for players than the existing system of build settings that Unity hides for runtime builds.

The following settings files are created by a NeoFPS project when it is run:

NAME	FILENAME	DESCRIPTION
Audio	Audio.settings	Contains settings for volumes (spatial effect, music and global).
Graphics	Graphics.settings	Contains various graphics and quality settings such as resolution, vsync and antialiasing.
Gameplay	Gameplay.settings	Contains miscellaneous gameplay settings such as crosshair colour.
Input	Input.settings	Contains settings for mouse and keyboard input including sensitivity, smoothing and acceleration.
Gamepad	Gamepad.settings	Contains settings for gamepad input such as profile preset and analog sensitivity.
Key Bindings	KeyBindings.settings	Contains individual key bindings for the different inputs.

If any of these files are found when the game starts, then the game settings will be based off these. The individual settings are mapped to the following scriptable objects:

NAME	BEHAVIOUR
Audio	FpsAudioSettings
Graphics	FpsGraphicsSettings
Gameplay	FpsGameplaySettings
Input	FpsInputSettings
Gamepad	FpsGamepadSettings
Key Bindings	FpsKeyBindings

If no settings file is found for one of the above, then the settings will default to those in the scriptable object asset located in the *NeoFPS/Resources* folder. These assets can be customised to set the default settings. Modifying the settings inn game and via the ".settings" files will not modify the assets.

If any of the settings are changed at runtime then the ".settings" file is saved again and an event is fired. This event allows objects to interact to changes of settings. For example, if the crosshair colour is changed in the gameplay settings is changed then the HUD crosshair can immediately change to reflect this.

These settings behaviours and files are intended as a basis for your own projects. Add any new settings as required and use these files as an example though be aware that these files will change in future versions of NeoFPS to reflect new features such as HDRP and post-processing settings potentially being added to the graphics settings.

See Also

Working With NeoFPS

Origin Shift

NeoFPS contains a system called origin shift that repositions the world and its contents periodically in order to keep the player's viewpoint within a certain distance of the origin. This is used in larger game worlds in order to compensate for visual and physics errors that increase as your distance from the origin increases.

What Is Floating Point Accuracy

In order to store transform positions, Unity uses 32-bit floating point values. These have a major weakness in that the larger the number becomes, the less precise the value it can store. The standard floating point value uses 1 bit to represent the sign of the number (+ or -), 8 bits to represent the exponent, and 23 bits to represent the mantissa. You can imagine the exponent as the position of the decimal point along the digits of the number represented by the mantissa.

A simplified way of visualising this is to imagine a decimal equivalent. For example, a mantissa of 1 and an exponent (in base 10) of 3 means the end result is 1000. There are 3 zeros to the right of the value in the mantissa. A mantissa of 1 and an exponent of -4 means the end result is 0.0001. There are 4 zeros to the left of the value in the mantissa (including the one before the decimal point). If the maximum value the mantissa could hold were 999 then that would mean that if you wanted to represent a value greater than 1000 by using the exponent, then there wouldn't be enough digits left in the mantissa to represent anything to the right of the decimal point. If your number represents meters, then you've lost cm and mm accuracy by this point.

Modern (IEEE 754) floats obviously have much greater precision than the above example, but there does still come a point where you lose sub-mm accuracy, and it can be surprising how early the effects can be seen. The effects of floating point accuracy are also compounded when performing a lot of mathematical operations, such as when calculating a world position from a hierarchy of local positions.

Since in NeoFPS the character's weapon is scaled down to prevent clipping through scenery, and then placed directly in front of the camera, you will see the vertices of your weapon mesh start to "swim" around as soon as 500m from the origin. The further you move the worse it gets. If you have ever fallen out of the map in a game and let your character keep falling, then you might have seen this effect as well, as the character eventually turns into a ball of thrashing spikes.

Unfortunately, due to the way Unity handles its scene hierarchies and represents transforms, this problem is almost unavoidable. However there are a number of ways to mitigate the problem. HDRP introduces a mode that centers the world to the camera when rendering, which can vastly reduce the visual swimming effect, however it doesn't help with physics collisions. Another solution that has been popular is a simple origin shift script that was shared on the now defunct unity wiki. This repositioned all the root objects in your scene when you moved a certain distance from the origin. The NeoFPS solution is an evolution of this.

Origin Shift Subscribers

The original floating origin script from the wiki would scan through all the root objects in a scene when a origin shift was triggered, adding an offset to each one. This approach is simple and doesn't require much setup to get working. However it has the major drawback that it only really works with basic transforms. You could adapt it to check the components on the object and apply specialised behaviours, but that would add a lot of overhead in `GetComponent()` calls and cause other issues with extensibility.

The NeoFPS origin shift solution uses subscribers attached to the objects in the world instead. This allows them to specialise in how they apply the offset, and if you have any new components that require a unique solution then you can simply create a custom `IOriginShiftSubscriber` based component and subscribe to the origin shift system with that. An example reason you might want to specialise is for any components that calculate velocity using position deltas from one frame to the next. They will need to compensate for the fact they just moved hundreds of meters each time the origin point offsets, or gain 1000s of m/s velocity the next frame.

The core of the system is the `OriginShift` component in your scene. This is the component that the subscribers attach to and which tracks the player character.

The following specialised subscribers are provided:

- [OriginShiftTransform](#) will reposition an object transform based on the offset.
- [OriginShiftParticleSystem](#) will reposition all the particles in a particle system based on the offset. This should be used for particle systems that are set to calculate in **world space**. If they are set to calculate in local space, then you should use the [OriginShiftTransform](#) instead.
- [OriginShiftTrailRenderer](#) is used to correct the historic points in a trail based on the offset so that the trail doesn't suddenly streak.

Some NeoFPS components also implement the `IOriginShiftSubscriber` directly, such as:

- The [NeoCharacterController](#).
- [Moving Platforms](#).
- [Firearm Projectiles](#).

Limitations

Unfortunately, there isn't really a single solution for floating point accuracy problems that isn't a huge hack. This means that there will be trade offs to gain this functionality. These are some of the known limitations or problems with the solution:

- You won't be able to mark any objects in your scene for static batching. Static batching can be a big optimisation, depending on your scene complexity, but it essentially bakes the positions of your objects and so changing their positions doesn't have a visual effect.
- Certain scene data such as nav meshes do not have a transform. I will be looking at adding nav mesh offset support as part of the AI work for the 1.2 upgrade.
- There may be Unity features that require a custom origin shift subscriber that I've missed. If you find these then you can report them to me via the [discord](#) or via the [support form on the website](#)

See Also

[NeoCharacterController](#)

Extension Packages

Overview

NeoFPS comes with a number of extension packages that extend the functionality by integrating with optional Unity packages such as Cinemachine. These extension packages are kept in the *NeoFPS/Extensions/* folder, which also contains a readme with the most up to date install instructions.

Cinemachine

The Cinemachine package adds a powerful virtual camera system to Unity. The extension package for NeoFPS adds Cinemachine based first person camera components, allowing you to use the procedural camera spring systems and field of view features of NeoFPS with Cinemachine's virtual camera. This makes it easy to maintain a consistent post processing look and feel, and create camera transitions for cutscenes. It also contains demo prefabs and a demo scene to test the implementation.

For more information, see [Cinemachine Extension](#)

Input System

The Input System extension adds alternative versions of each of the NeoFPS input handlers and the NeoFPS input manager that leverage Unity's newer input system. The Unity input system package adds much better support for gamepad controllers, along with an action map editor that can be used to quickly add new input actions to your game, however its rebinding functionality is under-developed and buggy. The extension package provided with NeoFPS also includes replacement touchscreen control scripts and prefabs, and a replacement rebinding options menu panel.

For more information, see [Input System Extension](#)

Render Pipelines

NeoFPS now comes with packages containing shaders and demo assets to support the scriptable render pipelines: URP and HDRP. You can check the requirements and import the relevant package via the **Unity Settings** page in the NeoFPS Hub.

Since the URP and HDRP packages contain replacement materials for a number of the demos provided with NeoFPS, there is also a package for the built-in pipeline (BIRP) with the original materials intact.

For more information, see [Graphics & Rendering](#)

See Also

[NeoFPS Installation](#)

[Getting Started](#)

[Generated Constants](#)

[Layers and Tags](#)

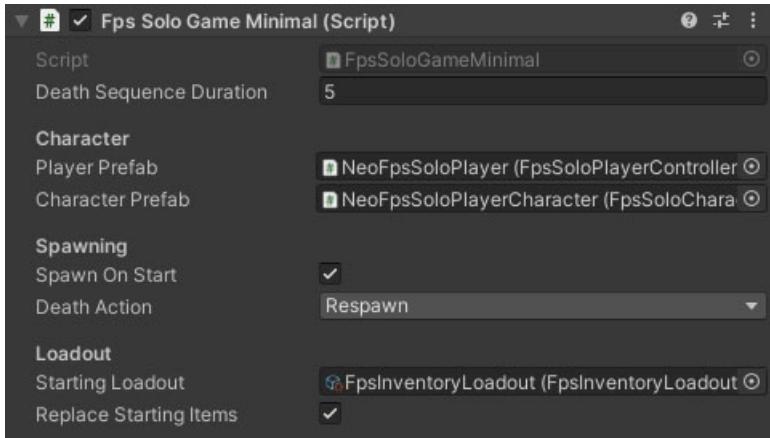
[Game Settings](#)

FpsSoloGameMinimal MonoBehaviour

Overview

The minimal game setup is intended for testing and development purposes. It simply spawns the specified player and character, and can be set to perform various actions on the player character's death.

Inspector



Properties

Name	Type	Description
Death Sequence Duration	Float	The time in seconds between the player character dying and respawning.
Player Prefab	FpsSoloPlayerController	The player prefab to spawn if no player exists (players are persistant).
Character Prefab	FpsSoloCharacter	The character prefab to spawn.
Spawn On Start	Boolean	Should the game mode automatically spawn a player character immediately on start.
Death Action	Dropdown	<p>What to do if the player character is killed. Available options are:</p> <ul style="list-style-type: none">• Respawn will spawn a new player character at the next available spawn point• ReloadScene will reload the scene from fresh, resetting everything• MainMenu will unload the scene and take the player back to the main menu• ContinueFromSave will load the last save (based on the SaveGameManager settings) if available and reload the scene if not.• RespawnWithItems will spawn a new player character at the next available spawn point, but their inventory will match the inventory on death (take care not to drop items on death as well as this)• ReloadScene will reload the scene from fresh, resetting everything
Starting Loadout	FpsInventoryLoadout	An optional inventory loadout for the character on spawn (this will replace their starting items).
Replace Starting Items	Boolean	How should the loadout be applied to the character. Should it replace the character's starting items or be added alongside them.

See Also

[FpsSoloPlayerController](#)

[FpsSoloCharacter](#)

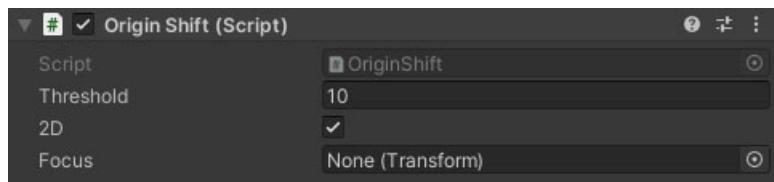
[FpsInventoryLoadout](#)

OriginShift MonoBehaviour

Overview

The OriginShift monobehaviour is the brain of the [origin shift system](#). It is responsible for tracking the position of the target. When the target object moves further than +/- the threshold on any axis, then all subscribers will be repositioned to move everything back towards the world origin.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Threshold	Float	The distance from 0 on each axis before the origin shift system repositions object back towards the center.
Focus	Transform	The transform to track the position of. This can be set at runtime via the API (such as when a character is spawned).

See Also

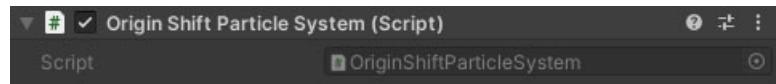
[The Origin Shift System](#)

OriginShiftParticleSystem MonoBehaviour

Overview

The OriginShiftParticleSystem monobehaviour subscribes to the origin shift system, and when a reposition is signalled it applies the offset to all the active particles emitted by the particle system. This should only be used for particle systems that are simulated in world space. If the particles are simulated in local space then it is more efficient to use the [OriginShiftTransform](#) behaviour instead.

Inspector



Properties

The OriginShiftParticleSystem behaviour has no properties exposed in the inspector.

See Also

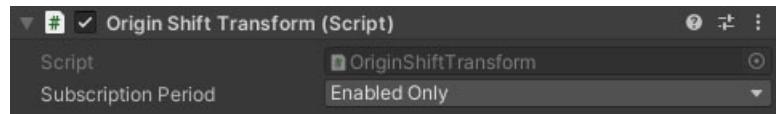
[The Origin Shift System](#)

OriginShiftTransform MonoBehaviour

Overview

The OriginShiftTransform monobehaviour subscribes to the origin shift system, and when a reposition is signalled it moves the object it's attached to based on the offset.

Inspector



Properties

NAME	TITLE	DESCRIPTION
Subscription Period	Dropdown	When should the component subscribe and unsubscribe from the origin shift system. ObjectLifecycle means that the component will subscribe on Awake and unsubscribe when destroyed - the object will be repositioned when inactive. EnabledOnly means the component will subscribe when enabled and unsubscribe when disabled.

See Also

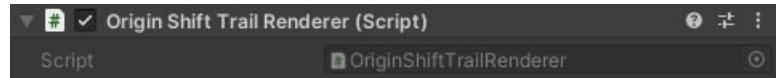
[The Origin Shift System](#)

OriginShiftTrailRenderer MonoBehaviour

Overview

The OriginShiftTrailRenderer monobehaviour subscribes to the origin shift system, and when a reposition is signalled it applies the offset to the points in the trail renderer to prevent the trail from connecting old position to new.

Inspector



Properties

The OriginShiftTrailRenderer behaviour has no properties exposed in the inspector.

See Also

[The Origin Shift System](#)

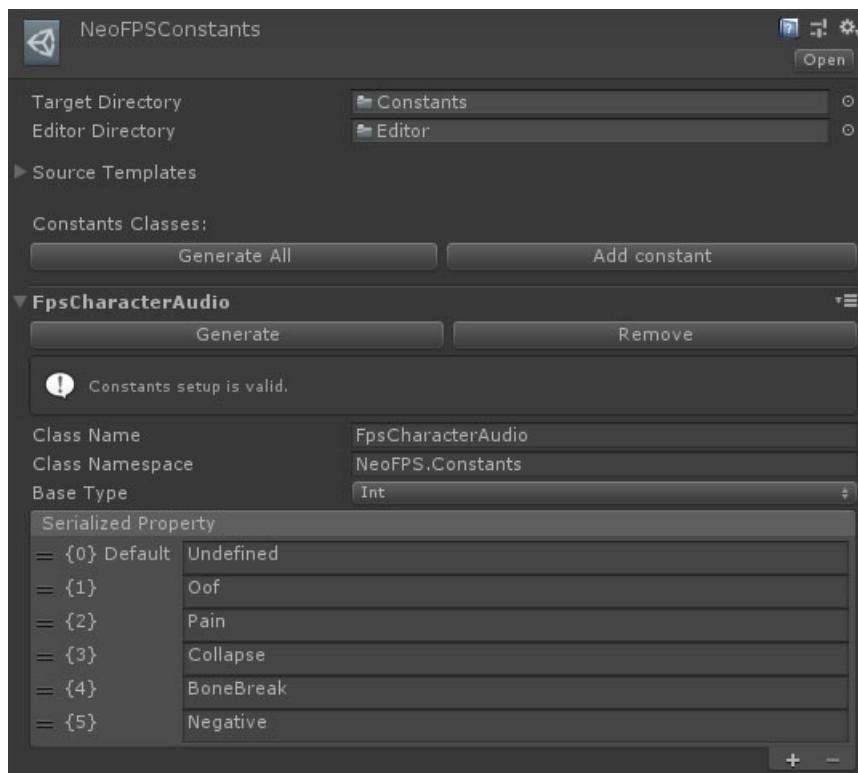
ConstantsSettings ScriptableObject

Overview

The ConstantsSettings asset is used to generate and maintain arbitrary constants for a game. The generated output script contains a wrapped constant that acts in code like an enum, but can intrinsically convert to its base type. A drawer is also generated that shows a dropdown with the constant values when the generated constant is serialized in the inspector. These constants have a number of uses such as object IDs or keys, without the overhead of using strings and dictionaries.

See [Generated Constants](#) for more information.

Inspector



Properties

Output

NAME	TYPE	DESCRIPTION
TargetDirectory	Folder	Where the generated constant script should be output.
EditorDirectory	Folder	Where the generated constant editor script should be output.

Source Templates

NAME	TYPE	DESCRIPTION
ByteConstant	TextFile	The text file to use when generating byte constants.*
ByteDrawer	TextFile	The text file to use when generating a byte constant editor script.*
IntConstant	TextFile	The text file to use when generating integer constants.*

NAME	TYPE	DESCRIPTION
IntDrawer	TextFile	The text file to use when generating an integer constant editor script.*
UIntConstant	TextFile	The text file to use when generating unsigned integer constants.*
UIntDrawer	TextFile	The text file to use when generating an unsigned integer constant editor script.*
ShortConstant	TextFile	The text file to use when generating short constants.*
ShortDrawer	TextFile	The text file to use when generating a short constant editor script.*
UShortConstant	TextFile	The text file to use when generating unsigned short constants.*
UShortDrawer	TextFile	The text file to use when generating an unsigned short constant editor script.*

* For more information on how the template is turned into a script see [Generated Constants](#).

Controls

The **Generate All** button will output a script file and a drawer script file to the specified directories for each constant, overwriting any existing files with the same name.

The **Add Constant** button will add a new constant settings entry to this scriptable object

Constant settings

Each constant has a number of common controls:

The **Generate** button will output a script file and a drawer script file to the specified directories, overwriting any existing files with the same name.

The **Remove** button will remove this constant from the scriptable object. Any existing generated files will not be touched.

Each constant also has the following properties:

NAME	TYPE	DESCRIPTION
Class Name	String	The name for the output constant. This will also be the output script file name, while the output drawer script will be named Drawer.
Class Namespace	String	The namespace for the output scripts.
Base Type	Dropdown	This value specifies which source templates should be used to generate the constant.
Constant Values	String Array	A sequential array of constant value names. These must be valid names, and not duplicated. Use the + and - buttons to add or remove values, or reorder by dragging the handle on the left of the array entry.

See Also

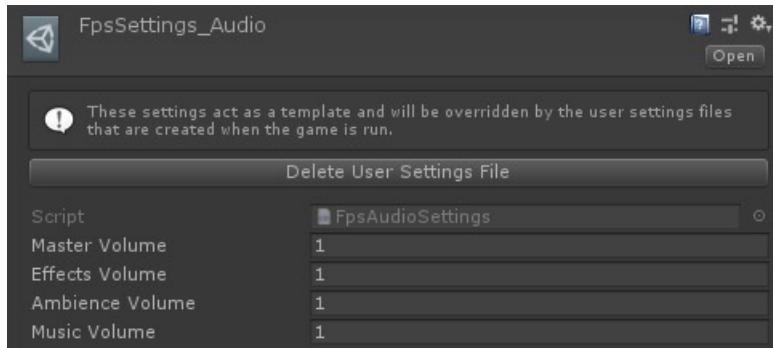
FpsAudioSettings ScriptableObject

Overview

The FpsAudioSettings asset specifies the default audio settings and loads / saves them to a *.settings* file on disk for player editing.

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Master Volume	Float	The overall game volume.
Effects Volume	Float	The volume for in game effects.
Ambience Volume	Float	The volume for ambience effects.
Music Volume	Float	The volume for music.

See Also

[Audio Systems](#)

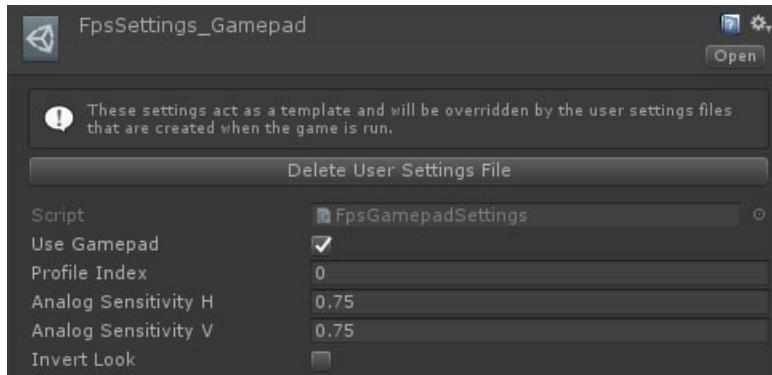
FpsGamepadSettings ScriptableObject

Overview

The FpsGamepadSettings asset specifies the default gamepad settings and loads / saves them to a *.settings* file on disk for player editing.

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Use Gamepad	Boolean	Should gamepad input be registered.
Profile Index	Int	The gamepad profile to use.
Analog Sensitivity H	Float	The horizontal gamepad aim sensitivity.
Analog Sensitivity V	Float	The vertical gamepad aim sensitivity.
Invert Look	Boolean	Invert the gamepad vertical aim.

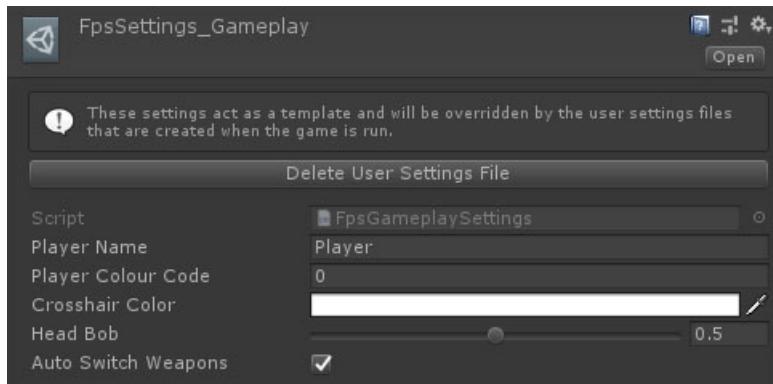
See Also

FpsGameplaySettings ScriptableObject

Overview

The FpsGameplaySettings asset specifies the default gameplay settings and loads / saves them to a `.settings` file on disk for player editing.

Inspector



The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

NAME	TYPE	DESCRIPTION
Player Name	String	The default player name.
Player Colour Code	Int	The default player colour.
Crosshair Colour	Color	The default crosshair colour.
Head Bob	Float	The ratio of head vs item bob. Head bob looks more natural when close to scenery, but some people can find it uncomfortable.
Auto Switch Weapons	Boolean	Should the player character automatically switch to a better weapon when picking it up.

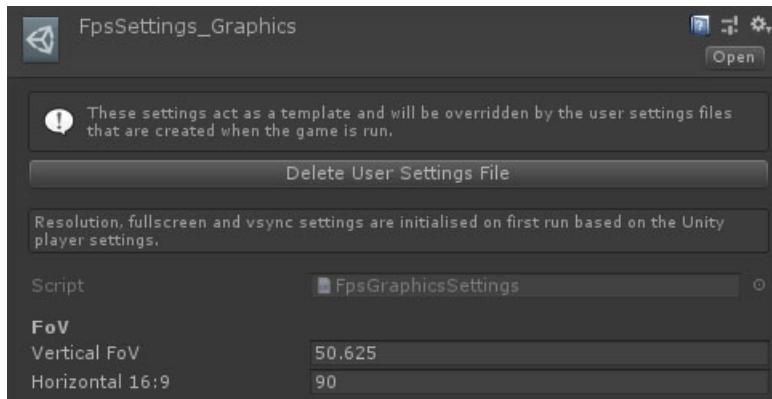
See Also

FpsGraphicsSettings ScriptableObject

Overview

The FpsGraphicsSettings asset specifies the default graphics settings and loads / saves them to a *.settings* file on disk for player editing.

Inspector



The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

NAME	TYPE	DESCRIPTION
Vertical FoV	Float	The field of view vertically of the player camera. This is how field of view is saved.
Horizontal 16:9	Float	The horizontal field of view on a 16:9 monitor. This is actually calculated from the vertical FoV, and when it is changed, the vertical FoV is modified.

See Also

[Unity Graphics](#)

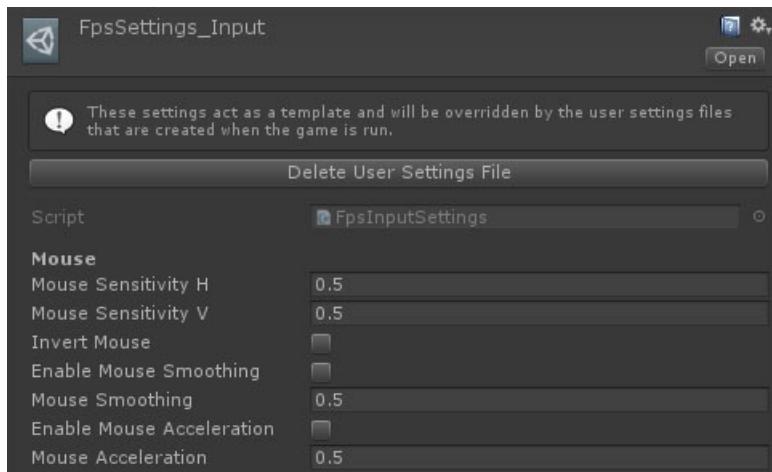
[Unity Quality Settings](#)

FpsInputSettings ScriptableObject

Overview

The FpsInputSettings asset specifies the default input settings and loads / saves them to a *.settings* file on disk for player editing.

Inspector



The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

NAME	TYPE	DESCRIPTION
Mouse Sensitivity H	Float	The horizontal mouse look sensitivity.
Mouse Sensitivity V	Float	The vertical mouse look sensitivity.
Invert Mouse	Boolean	Invert the mouse vertical aim.
Enable Mouse Smoothing	Boolean	Mouse smoothing takes a weighted average of the mouse movement over time for a smoother effect.
Mouse Smoothing	Float	The amount of mouse smoothing to add.
Enable Mouse Acceleration	Boolean	Mouse acceleration amplifies faster mouse movements.
Mouse Acceleration	Float	The amount of mouse acceleration to add.

See Also

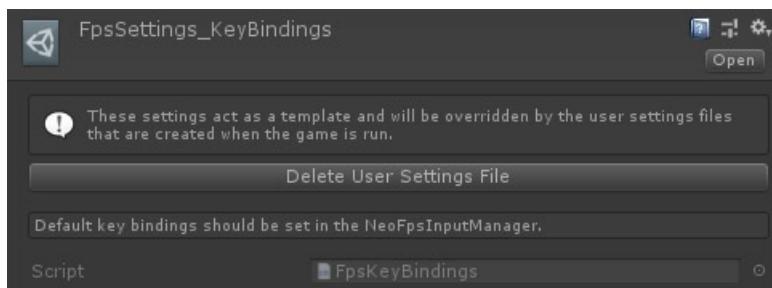
[NeoFPS Input System](#)

FpsKeyBindings ScriptableObject

Overview

The FpsKeyBindings scriptable object specifies the default input key bindings settings and loads / saves them to a *.settings* file on disk for player editing.

Inspector



The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

No properties are exposed in the inspector. The default key bindings are specified in the [NeoFpsInputManager](#) instead.

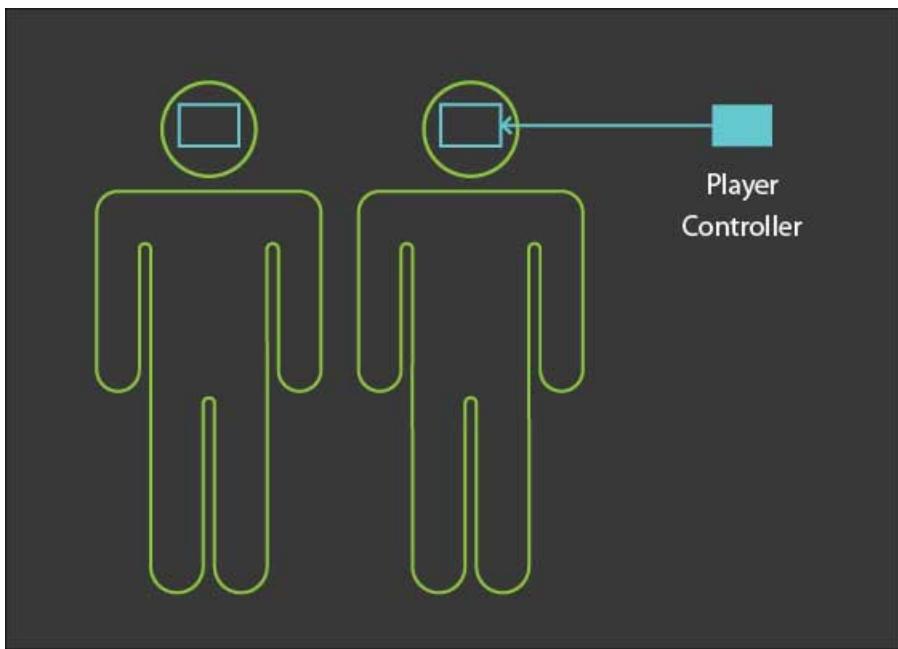
See Also

[Unity Input Settings](#)

FPS Characters

Overview

In NeoFPS, a first person character is split into two parts: the character and a controller. If a character dies then the controller persists and can be attached to a new character. Controllers can also be persisted between scenes if the game design requires.

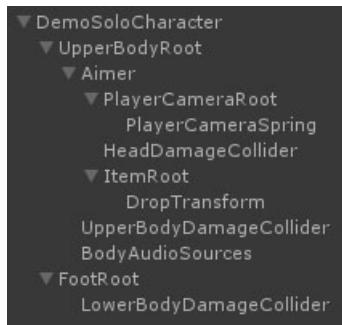


Characters



An FPS character ties together all the systems of NeoFPS into a single unit. Characters implement the `ICharacter` interface, which means you are free to define your own implementations of the character without being fixed to a specific base class.

A typical character hierarchy is as follows:



The root object is the most complex, containing the majority of components for the character:

- The `ICharacter` component
- [Capsule Collider](#)
- [Rigidbody](#)
- [NeoCharacterController](#)
- [MotionController](#)
- [AimController](#)
- [Inventory](#)
- [Interaction handler](#)
- [Character audio handler](#)
- [Input behaviours](#)

The player camera root is another object with a number of components:

- [First person camera](#)
- [Additive transform handler and effects](#)

There are also a number of [damage handlers](#) in the hierarchy for detecting incoming damage. These can be very simple, or more complex such as hit boxes on an animated character skeleton.

Controller

The FPS controller is persistent and can be attached to one character at any time. This system helps decouple the characters' mechanical implementation from their behaviour. In a multiplayer game there would be multiple player controllers or there could be a mix of player and AI controllers if bots were required. It also helps data such as stats persist between character deaths.

Fly-Cam Character

NeoFPS comes with an example fly-cam character prefab that you can use in your scenes (eg as a spectator cam). This is actually a full NeoFPS player character, but with the inventory and a number of other features removed. Its motion graph is also a very simple one with only regular and sprint based flying states. This does also mean that it could be expanded with weapons, scene interaction and other features if desired.

You can find the fly-cam character prefab at: *NeoFPS\Samples\SinglePlayer\Prefabs*

See Also

[FpsSoloCharacter](#)

[FpsSoloPlayerController](#)

Spawning

Overview

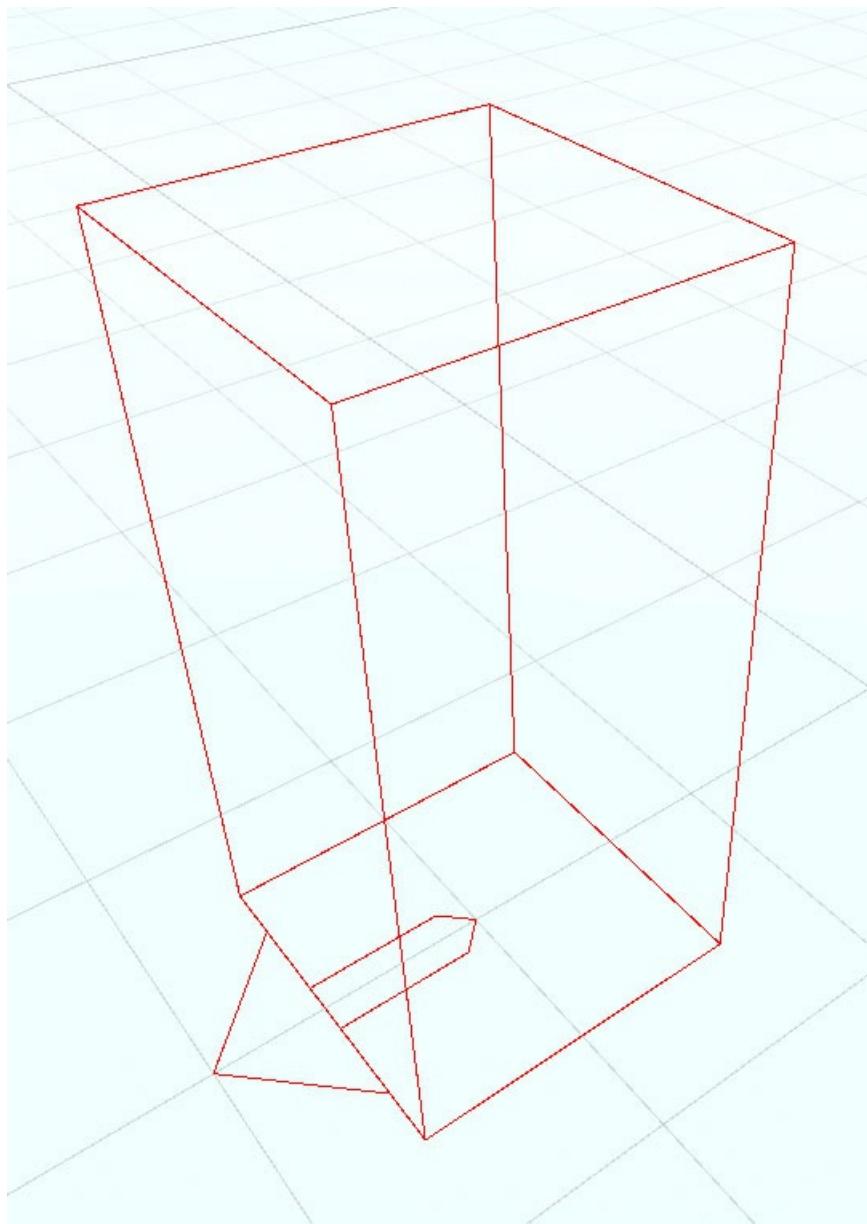
The NeoFPS player character should not be added to your scenes directly. Instead it uses a spawner system that allows you to place a number of spawn points around your map. This allows for greater flexibility in game mechanics such as:

- Spawn at one of a number of random locations
- Respawn on death without resetting the scenes
- Activate and deactivate spawn points as you move through a level so that respawning drops you closer to where you died
- Add a UI menu to choose a character on start
- Use multiple spawn points in case your preferred spawn is blocked

To make getting started easy, the **SimpleSpawnerAndGameMode** prefab contains a [spawn point](#) and a simple [game mode](#) that handles player spawning on start, as well as respawn on death. You can find this prefab at:

`Assets\NeoFPS\Samples\SinglePlayer\Prefabs\SimpleSpawnerAndGameMode.prefab`

Spawn Points



Spawn points are objects that are placed in the scene and positioned as required. By default, they will register themselves with the spawn manager as soon as the object is enabled, and unregister themselves as soon as the object is disabled.

You can also use an [OrderedSpawnPointGroup](#) behaviour to enforce a specific order to the registration of your spawn points. This

is useful when using the **Round-Robin** spawn mode which iterates through the spawn points, spawning in order, or the **First Valid** spawn mode which will iterate through the points in order and spawn at the first one that isn't blocked by an overlapping physics object. To control which is the active spawn mode, you can add a [SpawnManager](#) behaviour to an object in your scene.

Prototype Character

The NeoFPS samples also include a spawnerless player character that can be placed directly in your scenes. If the character dies then the scene is reloaded. This character is intended for rapid prototyping, and not for final production games.

You can find the prototype character prefab at the following location:

`Assets\NeoFPS\Samples\SinglePlayer\Prefabs\PrototypeSpawnerlessCharacter.prefab`

See Also

[FPS Characters](#)

[Game Modes](#)

First Person Body

Overview

NeoFPS now contains a number of features for adding an animated 3D body for your player character. These features allow you to drive animations from the motion graph, as well as syncing weapon and body animations seamlessly.

Alongside the information below, there is also a youtube playlist that gives a deep dive explanation of the different options here:

Demo Packages

The NeoFPS demo assets do not currently include character animations, so you need to add your own animations to make them work. See the included readme text file in the same folder for instructions on installation.

In the first person body folder (`NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\FirstPersonBody`) you will find packages set up with demo controllers and characters for the following animation packs:

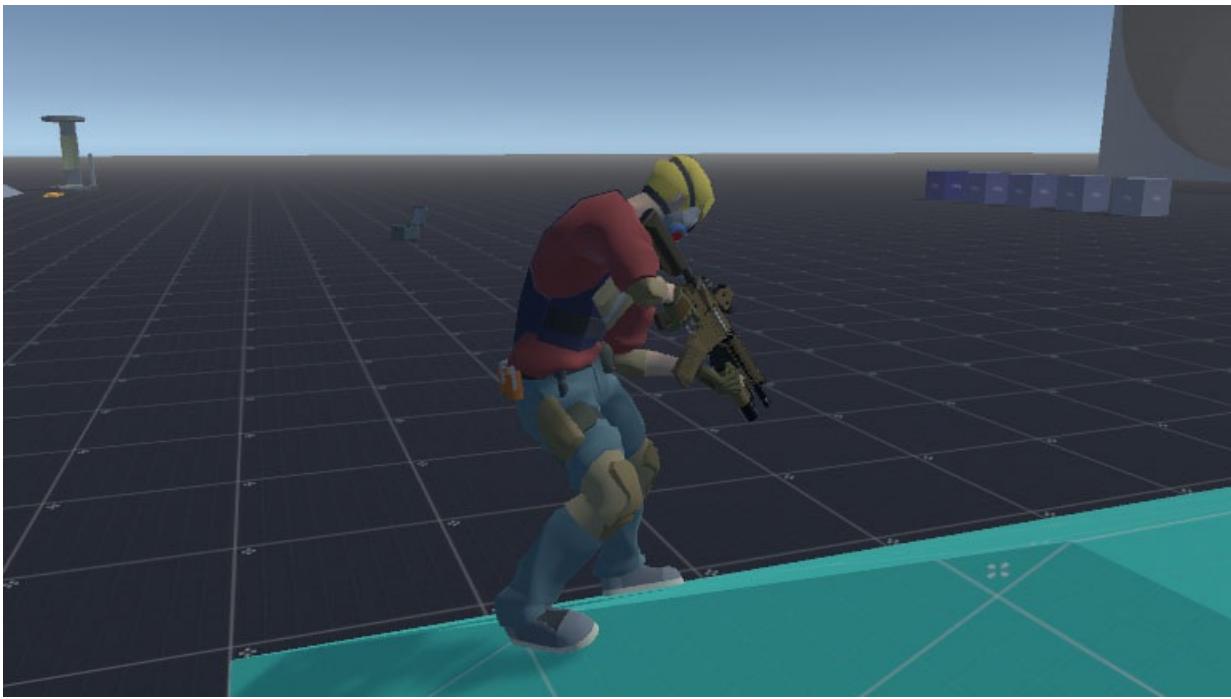
- Kubold's [Movement Animset Pro](#) and [Rifle Animset Pro](#) (both are required).
- MoCapOnline's [RIFLE Basic - Mocap Animation Pack](#).

First Person Body Types

There are 3 main types of first person body that the new systems are designed to support:

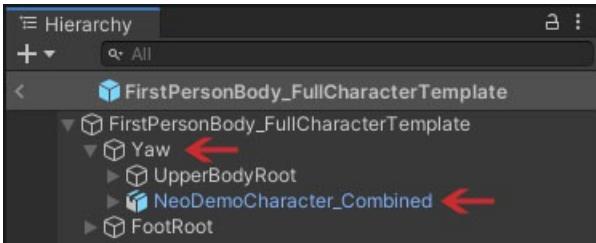
1. A full body with arms, torso and legs. Hand animations are synced to animated proxy objects on the weapons. This gives the most realistic first person body, similar to what you might see in a mil-sim type game.
2. Torso and legs, with the weapons containing animated arms. This allows you to look down and see your feet while keeping the weapon framed on the camera similar to the vast majority of first person games.
3. Split weapon rigs, where the arms and the weapon are rigged separately and have animations that sync up. This does not include torso and legs so you cannot see your feet when you look down. Its main strength is it makes it easier to swap out character arms.

Full Body Setup

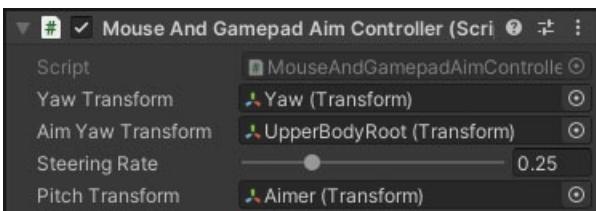


The full body setup means using a single character rig that includes the legs, body, arms and head.

The body uses steering, which means that it needs a separate **Yaw** transform under the root (local position and rotation all zeroed out) as a parent of the **UpperBodyRoot** object. The character body model and animator should be placed under this **Yaw** transform, alongside **UpperBodyRoot**.

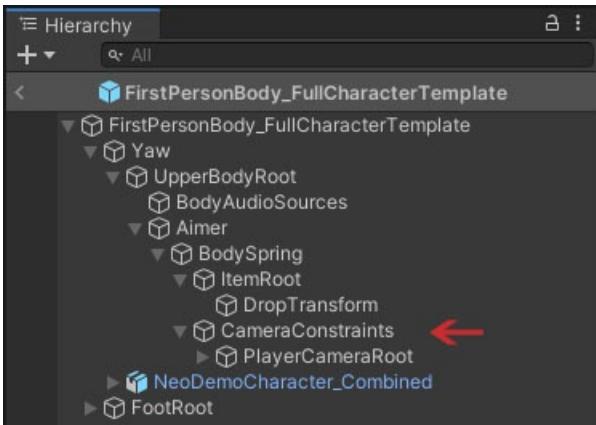


With that object in place, you need to set the [MouseAndGamepadAimController](#) so that its **Yaw Transform** points at the **Yaw** object, and **Aim Yaw Transform** points at **UpperBodyRoot**. The **Steering Rate** setting should then be set to some value above 0 and below 1, such as 0.25.

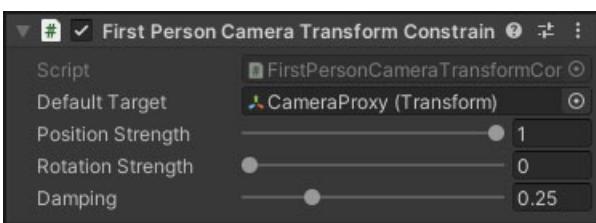


This means that the **UpperBodyRoot** will turn separately to the character heading, and the character will rotate towards this each frame based on the steering rate.

The camera position is constrained to a proxy object attached to the character rig's head bone. This requires a [FirstPersonCameraTransformConstraints](#) component attached to a new parent object of the **PlayerCameraRoot** object.

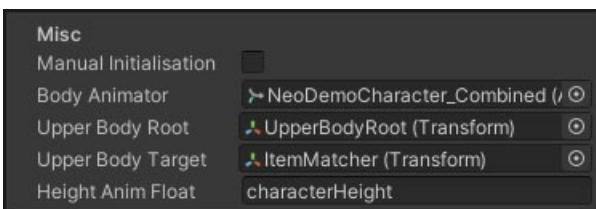


The constraints component should be set so that its **Default Target** is pointing at a proxy object parented to the character skeleton's head bone. This means that the camera will move with the head as the character animates. Set The **Position Strength** to 1, and **Rotation Strength** to zero. You don't want animated rotation of the character's head to affect the camera rotation or else this could cause motion sickness or force the camera to point in random directions. **Damping** smooths out the constraint's position matching and removes jitter from the animation. 0.25 should be fine.



Since the camera position is now controlled by the character skeleton, you should remove any head bob or position bob component from the **PlayerCameraRoot** object.

Wieldable items (weapons and tools) are added to an object called the item root that is set in the [inventory](#). For most FPS, that object would be positioned to match the camera so that their view-models and animations are fitted to the screen. With a full body FPS character, this can cause the weapon to rotate away from the character's body when they look up or down. To get around this, the weapons are aligned to a proxy object parented to the character's upper chest bone instead. To achieve this, first create an object with a name like **ItemMatcher** as a child of the upper chest bone (the last spine bone before the neck) and position it to roughly shoulder height. You will then need to assign this object to the **Upper Body Target** in the [MotionController](#) component. Each frame, the upper body root object will be positioned to match the target object. Rotation will be controlled by the aim controller.



The **Height Anim Float** is a float parameter in the character's [animator controller](#). This is a zero to 1 value, where 1 is the full height of the character. This can be used in a blend tree to blend movement and idle animations between standing and crouched versions. Without this parameter, the character will not be able to crouch. The **Body Animator** property should point at the animator component of the character model.

In order to utilise the various IK features of unity to match character feet to the ground and hands to the weapon, you will need a number of components adding to the character body, along with some tweaks to the character's [Animator](#) component. Firstly, you need to make sure that the **Culling Mode** in the character's [Animator](#) is set to **Always Animate**. If you don't do this, then when you look up the character animation will stop, which means the root motion and head movement will also stop. The additional components you need to add are:

- [FirstPersonBody](#). This component handles the foot IK and the optional aim animation parameters (see "Spine Kinematics")

below).

- [FirstPersonBodyRootMotion](#). This component captures root motion from the character animations and sends it to the [MotionController](#) on the character root.
- [FirstPersonCharacterArms](#). This component handles IK matching the character hands to animated proxy objects on the weapon. In the case of a split weapon rig, this component can realign the arms object to match the weapon view-model so that the animations sync properly.
- [ProceduralSpineAimMatcher](#). This component rotates the various spine bones to bend and twist the character spine as they aim up/down and side to side. See "Spine Kinematics" below for more details.
- [DeathAnimation](#). This is a simple component that just sets a bool parameter in the character's [animator controller](#) when the character dies. In most cases the default properties will work well for all of the above components.

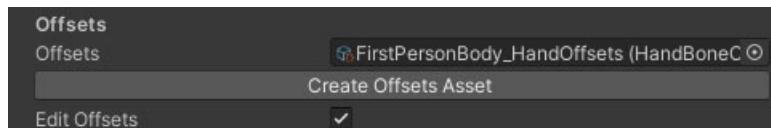
The last things that need doing to separate a full body character from a standard character are replacing the damage colliders and seeker targets. In a standard character, these are just positioned in the hierarchy. In a full body character they should be attached to the character. This means adding child objects to the limb bones of the character, set to the **CharacterPhysics** layer, and adding a collider component and one of the [damage handler](#) components. This allows you to set per-limb damage multipliers and critical flags. The seeker targets require a collider on the **AiVisibility** layer.

Weapon Setup and Hand Matching for Full Body

Matching the character hands and arms to your weapons requires 2 main components. A [WieldableItemKinematics](#) component on your weapon points at the bones or objects in your weapon rig that represent the animated hands for the weapon. The [FirstPersonCharacterArms](#) points at the bones for your character. Each frame, after the character and weapon animations have been updated, the hands of the character are positioned and rotated using IK to match the target objects on the weapon. The fingers are then rotated to match as well.

Since you will often be sourcing weapons and characters from different places (different asset publishers, stores or even different 3D content creation software), it is often the case that the target hand rig on the weapon and the hand rig on the character won't match up. For example, bending a finger might mean a 90 degree rotation on the X axis for one, and a -90 degree rotation on the Z axis for the other. To compensate for this, NeoFPS defines a standardised hand rig, and you can match both the weapon and character to the standardised rig using a [HandBoneOffsets](#) asset each.

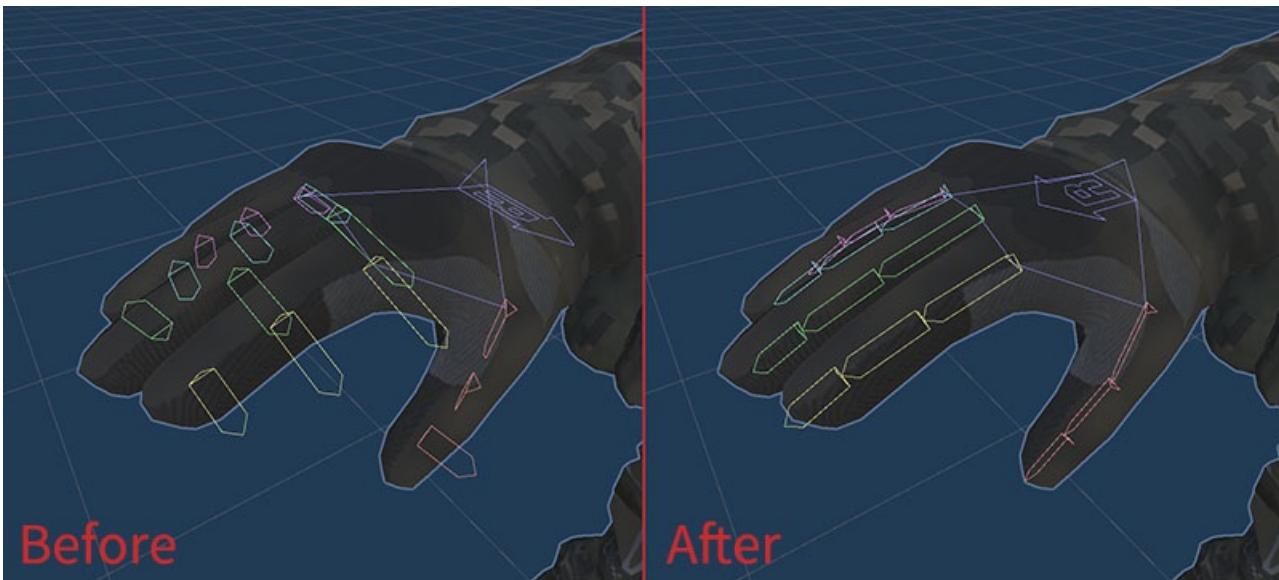
You won't usually edit these offset assets in isolation. Instead, both the [WieldableItemKinematics](#) and [FirstPersonCharacterArms](#) components let you apply an offsets asset or create a new one at the prefab location from within their inspectors:



Clicking the "**Create Offsets Asset**" will create one alongside the prefab with a matching name, plus the "_Offsets" suffix. Alternatively you can share the same offsets asset with multiple weapons or characters with matching hand setups by simply dragging and dropping the asset into the field above. Ticking on "**Edit Offsets**" will show the settings for the hand bone offsets asset directly below the checkbox in the inspector, allowing you to change its settings in place on the weapon or character. If you have the object with the [WieldableItemKinematics](#) or [FirstPersonCharacterArms](#) selected in the scene view (while editing the prefab) then you will see a set of gizmos that represent the bones of the hand and fingers. This is the standardised hand rig. Your goal is to edit the offsets so that the hand gizmo has its large arrow pointing from the wrist to the middle finger knuckle (roughly), and its smaller arrow pointing up. The fingers should be aligned so that the length of each gizmo is pointing directly at the base of the gizmo for the next finger joint, and the small arrow is pointing up from the hand. In both these cases, you can visualise up by imagining placing your hand flat, palm down on a table. The small arrows should point up from the fingers in this situation. For example, the following is the before and after for applying offsets to a weapon:



And the following is the before and after for applying offsets to a character:



With this completed in both cases, when the character equips a weapon, their hands should match up with the animations on the gun. Do bear in mind though, that this only copies across the position and rotation from weapon to character. If the character's hands are a different size and shape than the targets on the weapon, then you might get some intersection and misalignment. Fixing this is well outside of the scope for an asset like NeoFPS.

Once the hand matching is set up, there are 2 more elements to weapon animations with full body to be aware of:

The next thing to tackle is the position of the weapons. Where in a standard FPS setup your weapons will be centered on the camera, here your weapons are aligned to the upper chest instead. This can mean that they sit very low on the screen and often aren't visible. To compensate for this you can offset the view-model upwards. The root object, pose transform (if applicable) and weapon spring / local weapon object are all affected by procedural animation, so if you move these manually their positions will be reset on entering play mode. Instead you need to offset the view model itself (child of the weapon spring in a NeoFPS firearm's hierarchy). If the view model has animations that affect the root position (resetting it on play) then you can either switch on root motion for the view model, or add a new object between it and the weapon spring / local weapon object and offset that instead.

Finally, another side-effect of the camera position relative to the weapon is that the old firearm aimer modules will no longer correctly align to the optics of the weapon. In order to align the weapon correctly when aiming down sights, you will need to use a

[CameraConstraintsAimer](#) on your firearms. This detects the [FirstPersonCameraTransformConstraints](#) component in the camera hierarchy and sets a new constraint when aiming.

Torso and Legs



The torso and legs body uses more traditional hand, arm and weapon animations. It is only intended to give you a body that you can see when you look down. The setup is similar to the full body, with the following differences:

- In the [MotionController](#) component, you should set the **UpperBodyTarget** property to point at the camera proxy you are using for the camera constraints.
- Do not add the [FirstPersonCharacterArms](#) component to the animated character object.
- Weapons are regular first person weapons, with none of the first person body components required. Arms should be built into the weapon instead of using the character arms.

Split Weapon Rig

(Images Coming Soon)

This setup is used with weapon assets that come with separate arm and weapon rigs. If the arms are part of a full body then you will need to use the full body setup above, but if the arms do not have a body attached then they can be added to the character and shared across multiple weapons.

In order to achieve this, you need a much simpler setup similar to the standard characters.

Firstly, alongside the **ItemRoot** object in the hierarchy, you should add your character arms prefab. To this you need to add a [FirstPersonCharacterArms](#) component and set its **Arms Root Transform** to point at the object's transform component.

Next, create a [FirstPersonCharacterAnimationProfile](#) asset in the project folder and point it at the character's [animator controller](#). With this done, you can select which animations you want to expose for overrides, and give them more descriptive names. On each of your weapons, you will need to add a [WieldableItemBodyAnimOverrides](#) component. You can point this at the [FirstPersonCharacterAnimationProfile](#) asset you created for your character and then override specific animation clips for the arms by selecting the named animation from the list and dropping a replacement clip into the field.

You will also need to add a [FirstPersonCharacterInventoryWatcher](#) component to the root of the character. This ties into the [inventory system](#) and detects weapon selection changes. When the selected weapon changes, this component grabs the new weapon's [WieldableItemBodyAnimOverrides](#) component and applies the animation overrides to the character.

If your arms do not perfectly match your weapons (eg you are using a variety of arms from different assets), then you can use the

IK features by adding a [WieldableItemKinematics](#) component to your weapon object and pointing it at animated proxy objects on the weapon you want to match to. You can use a [HandBoneOffsets](#) asset to match the hands and finger rotations precisely. For this to work, your arms rig **must** be humanoid based.

Spine Kinematics



In order to prevent the camera, weapon and character body from intersecting when looking up or down, the character needs to be able to bend their spine based on the look direction. The NeoFPS aim controllers also have a steering system that disconnects the character body direction from the look direction. This allows you to do things like force the body to align with a ladder while still letting the player look around, and provides more physicality and inertia to the body when the player rapidly changes their aim direction while moving. This steering system means that the character needs to be able to twist their upper body left and right.

There are 2 ways to achieve this:

- Firstly, the [FirstPersonBody](#) component allows you to send the aim pitch and aim yaw (angle delta from character heading) to the character body [Animator](#) so that they can be used with an additive blend layer. This requires you to have additive pose animations for each direction that you can use to set up a blend tree.
- Alternatively, you can use the [ProceduralSpineAimMatcher](#) component. This interpolates the pitch and yaw for each bone up the spine and applies that rotation on top of the animation. This gives less scope for controlling weapon pose, etc in the animation, but it tends to give more reliable results that work across different upper body poses with much less effort. The screenshots above use this procedural system.

Character Shadows



The full-body setup is currently the only option that allows you to give your first person character an accurate shadow. To achieve this the head should be split from the rest of the character body, either as a separate skinned mesh, or as a standard mesh that is parented to the bone. You can then set the head up as a shadow caster (does not render visible geometry to the camera) in the **Skinned Mesh Renderer** component by setting **Cast Shadows** to **Shadows Only** and switching off **Receive Shadows**.

The torso and legs setup will be adapted to work as a shadow caster in the future. Due to the way that the weapons are aligned to the camera in this setup, looking up or down too far will cause the weapon and arms to rotate away from the body. This means that the arms will be visibly disconnected in the shadow in a way that using separate shadow casting arms with IK on the body would not be able to overcome.

The solution to this is to have a separate external version of the weapons, along with full body animations for their use. The modular firearm modules would then send animation data to the weapon animator and the character body animator at the same time to sync them up. Since the character animations are only used for shadow casting, they would not need to match exactly or be the same quality as the first person animations. This is exactly the same setup that the vast majority of multiplayer FPS have for displaying remote player characters. When looking at another player character you are not seeing the same weapon and animations they are. Instead you are seeing a lower detail weapon that can lod out at a distance and has much less moving parts, and uses much more generic animations (eg simpler finger rigs, and magazines that don't drop to the floor on reload). It is only very recently that a limited number of multiplayer games have started using the same animations for first person and external views, and this lends them a very distinct look that might not suit all game styles.

See Also

[FPS Characters](#)

Stamina

Overview

The stamina system in NeoFPS allows you to model various fatigue effects on your player character. Fatigue can be used to alter movement speed, to drive weapons' procedural animation (breathing), and to set thresholds where exhaustion sets in.

Movement & Stamina

The stamina system implementation provided can act as a motion graph data override. This means that the [motion data](#) used to specify movement speed will be overridden by values that the stamina system specifies. You can use animation curves on the [StaminaSystem](#) behaviour to blend between difference speeds as the character gets tired.

Animation

The current stamina level is used to specify a breathing rate and strength for the character. NeoFPS uses a number of [procedural animation](#) effects for animating weapons, including the [BreathingEffect](#). This applies position and rotation animation with each breath, making aiming harder as the character gets tired.

Modifying Stamina

There are a number of example ways to modify the character's stamina provided out of the box. The [DrainStamina](#) and [ModifyStamina](#) motion graph behaviours allow you to modify the stamina based on the current state of the [Motion Graph](#). The [FirearmAimFatigue](#) monobehaviour is attached to [modular firearms](#) and applies a stamina drain whilst aiming down sights.

The stamina system also has a setting to recharge stamina over time. An example stamina implementation would be to give both the character's walking state and the firearm aiming a stamina drains that are each smaller than the stamina system's refresh rate, but are larger when combined. This would mean that the player could move or aim without getting tired, but doing both together would cause them to slowly fatigue.

You can add your own systems to modify the character's stamina via a simple API. You can directly alter the stamina and max stamina values on the system by using the following methods:

```
void IStaminaSystem.SetStamina(float amount, bool normalised = false);
void IStaminaSystem.IncrementStamina(float amount, bool isFactor = false);
void IStaminaSystem.DecrementStamina(float amount, bool isFactor = false);
```

The `normalised` and `isFactor` parameters are used to specify if the value you are applying is directly to the stamina value (false), or as a factor of stamina divided by max stamina (true).

You can also add stamina drain delegates to the system that allow you to modify the stamina over time based on any number of outside influences. You do this using the following methods:

```
void IStaminaSystem.AddStaminaDrain(StaminaDrainDelegate drain);
void IStaminaSystem.RemoveStaminaDrain(StaminaDrainDelegate drain);
```

Where the `StaminaDrainDelegate` is defined as:

```
public delegate float StaminaDrainDelegate(IStaminaSystem system, float modifiedStamina);
```

Here the parameters are the stamina system itself, and the current stamina (this might not match the stamina value on the system as this stamina drain delegate might be one of many, and the altered stamina is passed to each delegate in sequence before it is applied to the system). The delegate returns the amount to reduce the stamina by this frame.

See Also

[The Motion Graph](#)

Modular Firearms

Additive Transforms And Effects

Character Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

I Placed The Player Character In My Scene, But It Deactivates On Start

NeoFPS uses a spawning system instead of placing characters directly in your scene. This enables a number of features such as respawning, and persisting character data across scenes. If you place one of the demo characters, and then hit play, that character will not be assigned a player controller by the spawning system, and so it will be deactivated to prevent clashing with spawned characters.

To fix this, remove the character and place a spawner such as the **SimpleSpawnerAndGameMode** prefab that you can find in the folder: `NeoFPS\Samples\SinglePlayer\Prefabs`

If you really don't want to use the spawning system for some reason, you can either use the **PrototypeSpawnerlessCharacter** prefab from the same folder as above, or add a **FpsPrototypePlayerController** component to your character, which will assign itself as the character's player controller on start.

I Get An Error Saying "No valid spawn points found" And No Character Spawns

The NeoFPS spawn system requires you to have one or more spawn points in your scene that are active and unobstructed in order to spawn a player character. Most of the demo scenes use the **SimpleSpawnerAndGameMode** prefab found at `NeoFPS\Samples\SinglePlayer\Prefabs`. This prefab combines a spawn point, a camera (for points where there is no first person character) and a game mode (controls when the character spawns).

Selecting an object with a **Spawn Point** component will show a gizmo in the scene view (make sure you haven't hidden the relevant gizmo in the dropdown at the top of the scene view window). The gizmo will show red if there is a physics object blocking its volume, and blue if it is unobstructed. Check that your spawn point is not overlapping the ground.

If you have a scene with a lot of dynamic rigidbody physics objects, then it is possible that one of them could move to obstruct the spawn point and prevent you from respawning after death. To prevent this, you can create a new object and add a **Spawn Point** component to it to provide an alternative spawn point. The spawning system will respawn your character at each (unobstructed) spawn point in sequence by default. You can also add a **Spawn Manager to one object in your scene** if you want to change the spawn order to random, or to restart the test from the first spawn point each respawn. If you want to specify the spawn order manually, you can use an [OrderedSpawnPointGroup](#).

FirstPersonBody MonoBehaviour

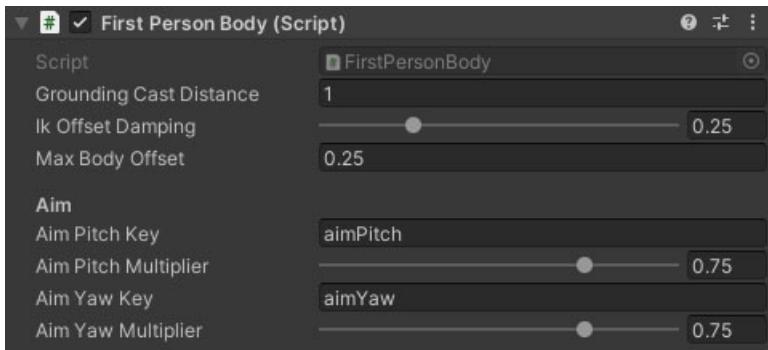
Overview

The FirstPersonBody behaviour is used to apply foot IK and aim parameters to a character's body [Animator](#).

The foot IK strength can be applied using the [FootIk motion graph behaviour](#) on relevant states in the motion graph. It works by getting the height of the foot from a virtual ground plane in the animation, and then getting the offset of the real ground from the foot position at runtime. Foot contacts will be calculated based on the highest offset between the heel and toes bones. If one foot is lower than the virtual ground plane then the whole body will be lowered (with some damping) to prevent over-extending the leg.

Aiming is used with the NeoFPS steering system on the aim controllers (eg the [MouseAndGamepadAimController](#)). As the character looks side to side the body does not immediately match that unless steering is set to 1. The difference between the aim yaw and body yaw, along with the pitch value will be sent to the animator controller so you can use those with a blend tree or animation layers and bend the spine to match. Alternatively you can keep these settings blank, and use the [ProceduralSpineAimMatcher](#) component on your character body for a much simpler setup, though you will have slightly less control this way.

Inspector



Properties

NAME	TYPES	DESCRIPTION
Grounding Cast Distance	Float	The distance to cast for ground detection.
Ik Offset Damping	Float	A smoothing value for foot offsets to prevent popping on stepped or faceted ground.
Max Body Offset	Float	The maximum distance downwards that the body can be shifted to prevent legs overextending on slopes.
Aim Pitch Key	String	The name of a float parameter on the character's animator that should be set with the pitch value. Pitch will be normalised to 90 degrees, so looking straight up would be +1 and down would be -1.
Aim Pitch Multiplier	Float	A multiplier applied to the pitch value to reduce any animation effect.
Aim Yaw Key	String	The name of a float parameter on the character's animator that should be set with the aim yaw value (relative to the body heading). Pitch will be normalised to 90 degrees, so looking right would be +1 and left would be -1.
Aim Yaw Multiplier	Float	A multiplier applied to the yaw value to reduce any animation effect.

See Also

[First Person Body](#)

[Animator Controllers](#)

[FootIkBehaviour](#)

FirstPersonBodyRootMotion MonoBehaviour

Overview

The FirstPersonBodyRootMotion behaviour captures root motion on the character's [animator controller](#) and sends it to the character's [motion controller](#). The motion controller can then blend between root motion and state driven motion based on its motion graph. You can use the [RootMotion motion graph state](#) or [SetRootMotionStrength motion graph behaviour](#) to control the root motion strength.

Inspector



Properties

The FirstPersonBodyRootMotion behaviour has no properties exposed in the inspector.

See Also

[The Motion Graph](#)

[Animator Controller](#)

[RootMotion motion graph state](#)

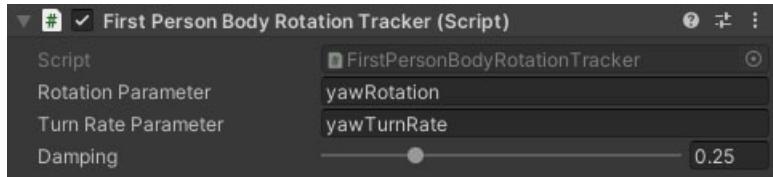
[SetRootMotionStrength motion graph behaviour](#)

FirstPersonBodyRotationTracker MonoBehaviour

Overview

The FirstPersonBodyRotationTracker behaviour tracks the difference between the character heading (body forwards) and aim yaw and sends the result to the character's body animator. This can then be used for things like turning on the spot animations.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rotation Parameter	String	The name of a float parameter in the animator controller to set with the rotation (damped) per frame.
Turn Rate Parameter	String	The name of a float parameter in the animator controller to set with the turn rate / angular velocity (damped) per frame.

See Also

[First Person Body](#)

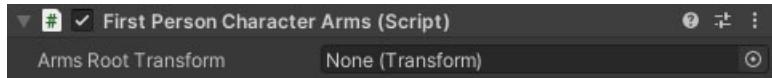
[Animator Controller](#)

FirstPersonCharacterArms MonoBehaviour

Overview

The FirstPersonCharacterArms behaviour is used on a character's arms to match animations to a [WieldableItemKinematics](#) component on the character's selected weapon. There are 2 approaches to this based on whether the arms that are attached to the character have a full body and animations synced with the weapon. If the character has a full body then the **Arms Root Transform** property should be kept empty, and the arms will have their hand and finger positions matched using IK. If the arms do not have a full body and they use their own animations that are synced to the weapon, then you should set the **Arms Root Transform** to point to the root of the arms and it will be repositioned to match an equivalent transform on the weapon each frame.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Arms Root Transform	Transform	An optional arms transform that should be matched to the weapon geometry. Use this to match arm animations to weapon animations after the weapon has been affected by procedural animation effects such as bob or poses, but only if the character does not have a full body and arms and weapon have synced animations .

See Also

[First Person Body](#)

[Animator Controllers](#)

FirstPersonCharacterInventoryWatcher MonoBehaviour

Overview

The FirstPersonCharacterInventoryWatcher is attached to a character's body (the object with the [animator controller](#)) and checks each selected weapon for a [WieldableItemBodyAnimOverrides](#) component and applies any animation overrides to the character body if one is found.

Inspector



Properties

The FirstPersonCharacterInventoryWatcher behaviour has no properties exposed in the inspector.

See Also

[Inventory](#)

[WieldableItemBodyAnimOverrides](#)

FpsPrototypePlayerController MonoBehaviour

Overview

The FpsPrototypePlayerController is a version of the player character controller which can be added to the character object directly to bypass the spawning system.

If the player is killed then the scene will be reloaded.

This controller is intended purely for prototyping and testing. If more complex behaviour is required such as respawning or loading from saves, then use the spawn system as in the demo scenes.

Inspector



Properties

The FpsPrototypePlayerController has no properties exposed in the inspector.

See Also

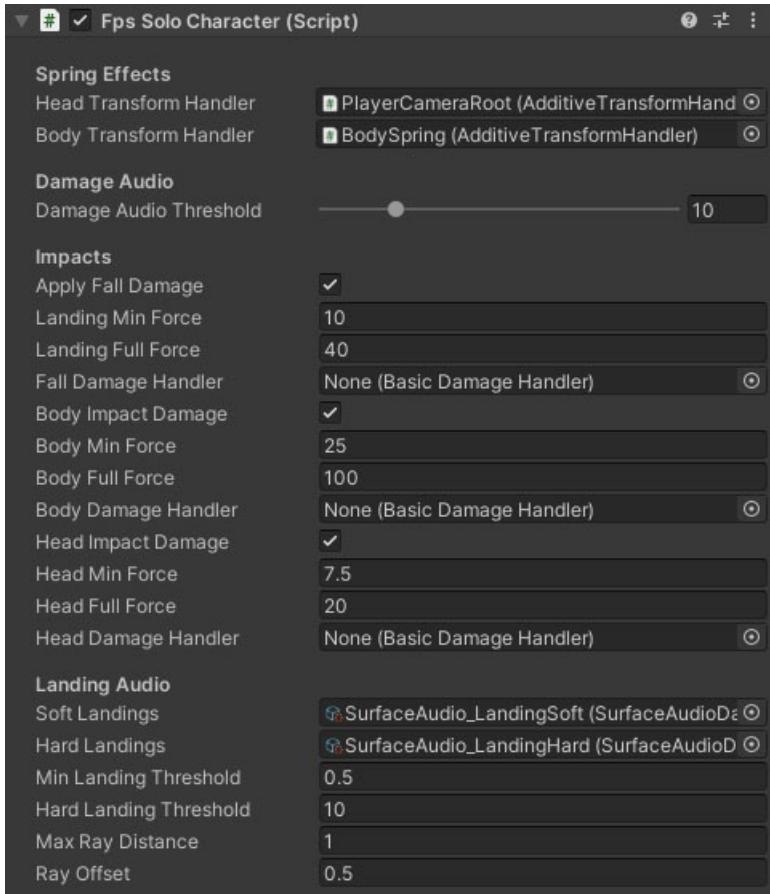
[FPS Characters](#)

FpsSoloCharacter MonoBehaviour

Overview

FpsSoloCharacter is the base character behaviour for single player games. Handles impact and landing damage and audio, and specifies the transform handlers for reacting events.

Inspector



Properties

Name	Type	Description
Head Transform Handler	AdditiveTransformHandler	The additive transform handler attached to the head hierarchy of this character (used for things like weapon recoil and impacts). The spring effects here affect the camera but not the carried items.
Body Transform Handler	AdditiveTransformHandler	The additive transform handler attached to the body hierarchy of this character (used for things like weapon recoil and impacts). The spring effects here affect the camera and the carried items.
Damage Audio Threshold	Float	The amount of damage to take in a single hit before playing a character damage audio clip.
Apply Fall Damage	Boolean	Should the character be subject to damage from landing impacts (impacts where the character capsule is hit in the bottom hemisphere).
Landing Min Force	Float	The minimum landing impact magnitude before any damage is applied.

Name	Type	Description
Landing Full Force	Float	The landing impact magnitude where a full 100 damage will be applied.
Fall Damage Handler	Damage Handler	(Optional) The damage handler to pass fall impact damage to (can apply armour, shields, etc). If this is null then damage will be sent direct to the character's health manager.
Body Impact Damage	Boolean	Should the character be subject to damage from body impacts (impacts where the character capsule is hit in the central cylinder).
Body Min Force	Float	The minimum body impact magnitude before any damage is applied.
Body Full Force	Float	The body impact magnitude where a full 100 damage will be applied.
Body Damage Handler	Damage Handler	(Optional) The damage handler to pass body impact damage to (can apply armour, shields, etc). If this is null then damage will be sent direct to the character's health manager.
Head Impact Damage	Boolean	Should the character be subject to damage from head impacts (impacts where the character capsule is hit in the top hemisphere).
Head Min Force	Float	The minimum head impact magnitude before any damage is applied.
Head Full Force	Float	The head impact magnitude where a full 100 damage will be applied.
Head Damage Handler	Damage Handler	(Optional) The damage handler to pass head impact damage to (can apply armour, shields, etc). If this is null then damage will be sent direct to the character's health manager.
Soft Landings	SurfaceAudioData	Surface audio library used to trigger the correct sound when the character lands below the "hard landing" threshold.
Hard Landings	SurfaceAudioData	Surface audio library used to trigger the correct sound when the character makes a heavy landing.
Min Landing Threshold	Float	The magnitude of the landing force below which no landing sound will be played.
Hard Landing Threshold	Float	The magnitude of the landing force above which to play a hard landing sound.
Max Ray Distance	Float	The maximum downward ray length for a ground test.
Ray Offset	Float	The vertical offset from the absolute bottom of the character to start the ground test raycast.

See Also

AdditiveTransformHandler

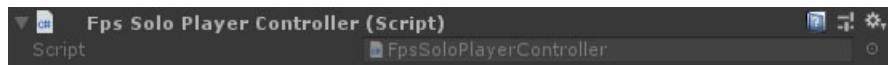
SurfaceAudioData

FpsSoloPlayerController MonoBehaviour

Overview

The FpsSoloPlayerController is the base player character controller for single player games.

Inspector



Properties

The FpsSoloPlayerController has no properties exposed in the inspector.

See Also

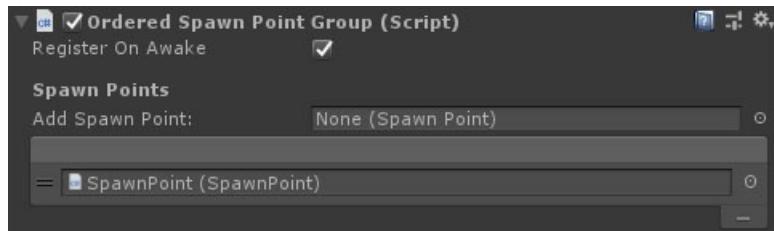
[FPS Characters](#)

OrderedSpawnPointGroup MonoBehaviour

Overview

The OrderedSpawnPointGroup behaviour is used to enforce a spawn order for multiple [spawn points](#). The order that Unity calls Start() on multiple objects can be unpredictable, so this behaviour allows you to add the spawn points to a list and rearrange the order. This works best with the [SpawnManager](#) set to use the "Round Robin" spawn mode.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Register On Awake	Dropdown	Should the spawn points be registered with the manager as soon as this object is awoken.
Spawn Points	SpawnPoint Array	The spawn points to register in order.

See Also

[SpawnManager](#)

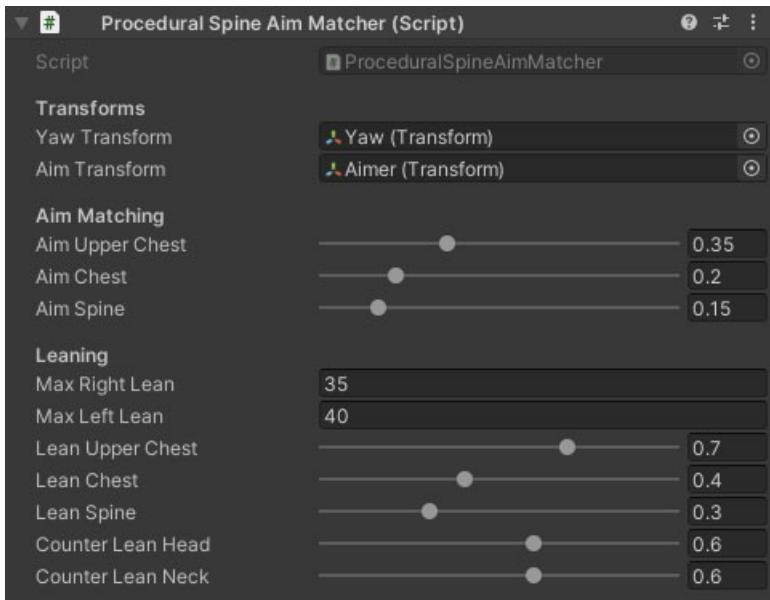
[SpawnPoint](#)

ProceduralSpineAimMatcher MonoBehaviour

Overview

The ProceduralSpineAimMatcher behaviour adapts the 3 spine bones in a player character's humanoid rig to match the aim rotation relative to the body. This means that when you aim up or down the character bends forward or back to prevent the gun and camera clipping into the body.

Inspector



Properties

Name	Type	Description
YawTransform	Transform	The transform to use for the body heading.
AimTransform	Transform	The transform to use for the aim direction.
Aim Upper Chest	Float	The strength of the rotation to apply to the upper chest (usually this should be the strongest and all bones should add up to 1 or less).
Aim Chest	Float	The strength of the rotation to apply to the middle chest bone (usually this should be medium strength, and all bones should add up to 1 or less).
Aim Spine	Float	The strength of the rotation to apply to the lower spine bone (usually this should be the weakest since it is anchored to the hips, and all bones should add up to 1 or less).
Max Right Lean	Float	A target angle for the leaning bone rotation (right). The results will not be exact as the rotation is blended across the bones of the spine.
Max Left Lean	Float	A target angle for the leaning bone rotation (left). The results will not be exact as the rotation is blended across the bones of the spine.
Lean Upper Chest	The strength of the lean rotation to apply to the upper chest (usually this should be the strongest).	

NAME	TYPE	DESCRIPTION
Lean Chest	Float	The strength of the lean rotation to apply to the middle chest bone (usually this should be medium strength).
Lean Spine	Float	The strength of the lean rotation to apply to the lower spine bone (usually this should be the weakest since it is anchored to the hips).
Counter Lean Head	Float	The strength of the lean counter rotation to apply to the head bone (usually this should be medium strength).
Counter Lean Neck	Float	The strength of the lean counter rotation to apply to the neck bone (usually this should be low strength).

See Also

[First Person Body](#)

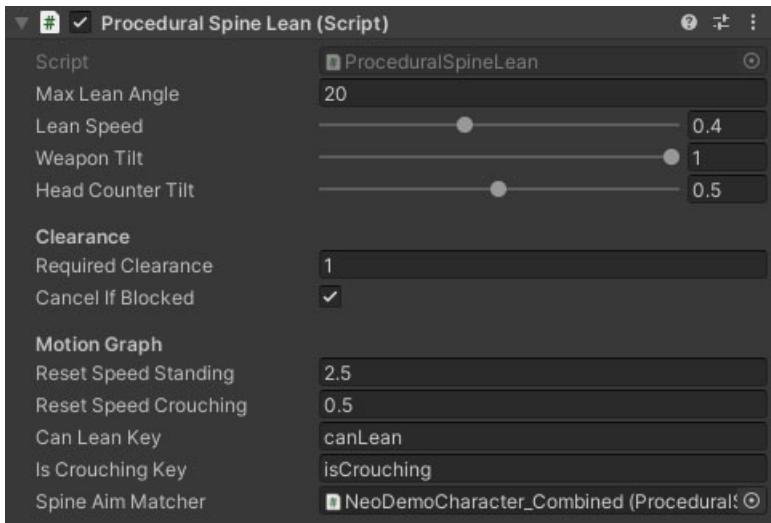
[Animator Controller](#)

ProceduralSpineLean MonoBehaviour

Overview

The ProceduralSpineLean behaviour is a variation of the [BodyLean additive effect](#) that ties into a first person body's [ProceduralSpineAimMatcher](#) to add spine based leaning into the mix.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Lean Angle	Float	The angle to lean to either side when lean power is set to 1 or -1.
Lean Speed	Float	The speed of transitioning from 1 lean value to another. 1 is nearly instant.
Weapon Tilt	Float	How much of the lean rotation is reflected in the weapon.
Head Counter Tilt	Float	A counter rotation of the head compared to the weapon.
Required Clearance	Float	The distance to check from the center line for clearance space to lean. The lean amount will be capped based on the result.
Cancel If Blocked	Boolean	If there is no clearance space then return the lean amount to 0 until manually re-applied.
Reset Speed Standing	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Reset Speed Crouching	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Can Lean Key	String	The key to a motion graph switch parameter that dictates if the character can lean or not.
Is Crouching Key	String	The key to a motion graph switch parameter that dictates if the character is crouching or standing.

NAME	TYPE	DESCRIPTION
Spine Aim Matcher	ProceduralSpineAimMatcher	The ProceduralSpineAimMatcher component that drives the procedural spine rotation for aiming and lean.

See Also

[Additive Transforms](#)

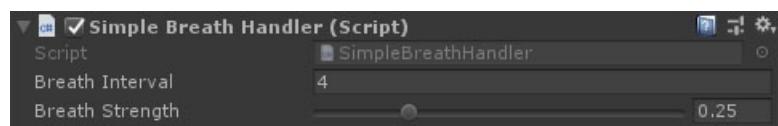
[Motion Graph Parameters And Data](#)

SimpleBreathHandler MonoBehaviour

Overview

The SimpleBreathHandler is attached to a character to give them a constant breathing rate. This is used to drive the [BreathingEffect](#) procedural animation on weapons. You can alternatively use a [StaminaSystem](#) behaviour for a more complex breathing system that adapts to character fatigue.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Breath Interval	Float	The time in seconds between breaths.
Breath Strength	Float	The strength of the character's breathing (0 = non-existent, 1 = heaving/panting).

See Also

[BreathingEffect](#)

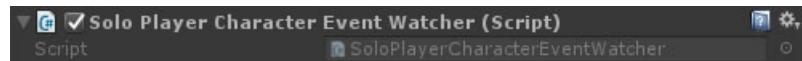
[StaminaSystem](#)

SoloPlayerCharacterEventWatcher MonoBehaviour

Overview

The SoloPlayerCharacterEventWatcher behaviour attaches to an event that is fired when the player character changes. It then passes that change on to its subscribers. An example use is in the player HUD, where it is used to bind various HUD elements such as health and inventory to the player character.

Inspector



Properties

The SoloPlayerCharacterEventWatcher behaviour has no properties exposed in the inspector.

See Also

[AdditiveTransformHandler](#)

[SurfaceAudioData](#)

SpawnManager MonoBehaviour

Overview

The spawn manager is used to spawn characters. Spawn points register with the manager on start and deregister when disabled or destroyed. You do not need this behaviour in your scene, but adding it will allow you to change the spawn behaviour by setting the behaviour properties.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Spawn Mode	Dropdown	<p>How the next spawn point is chosen:</p> <ul style="list-style-type: none">• RoundRobin picks each spawn point in sequence.• FirstValid starting at the first registered spawwn point and iterating until a valid one is found• Random picked at random until a valid point is found.

See Also

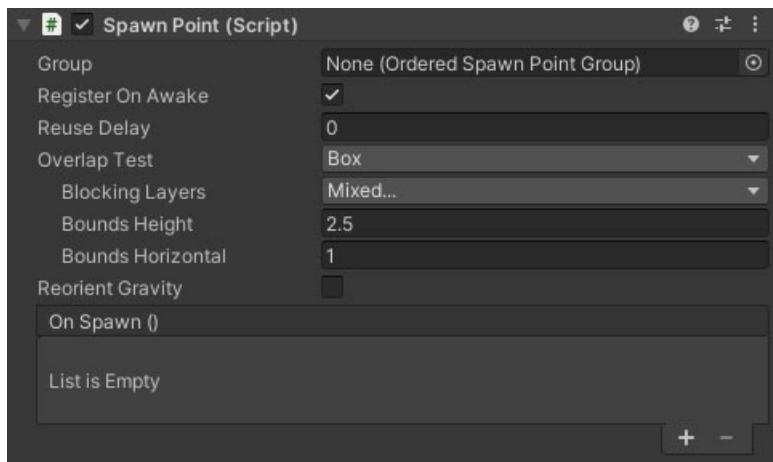
[SpawnPoint][1]

SpawnPoint MonoBehaviour

Overview

The SpawnPoint behaviour specifies where characters are spawned. It will register with the [SpawnManager](#) when active and enabled, and deregister when deactivated or destroyed. It is recommended to have more than one in the scene in case one or more is blocked.

Inspector



Properties

Name	Type	Description
Register On Awake	Boolean	Should the spawn point be registered with the SpawnManager immediately on awake? This option will be disabled if the spawn point is part of a OrderedSpawnPointGroup .
Reuse Delay	Float	How long before the spawn point can be used again.
Overlap Test	Dropdown	The collider volume type for checking if the spawn point is clear or overlapped by another object. Options are Box , Capsule , None .
Blocking Layers	LayerMask	Which layers will block the character from spawning if they overlap the spawn volume.
Bounds Height	Float	The vertical height of the bounding volume for overlap checks.
Bounds Horizontal	Float	The horizontal dimension of the bounding volume for overlap checks.
Reorient Gravity	Boolean	Should the character's gravity be reoriented to match the spawn point. If the spawn is tilted on one side, this will make the character's down direction equal to the spawn point's.
On Spawn	Event	A UnityEvent fired when a character is spawned at this point. Allows for simple triggering of spawn audio and visual effects.

See Also

[SpawnManager](#)

[OrderedSpawnPointGroup](#)

StaminaSystem MonoBehaviour

Overview

The StaminaSystem behaviour is attached to a character and used to model various fatigue and exertion effects.

Stamina regenerates at a preset rate and can be drained or modified via a simple API. NeoFPS includes examples such as the [FirearmAimFatigue](#) behaviour which drains stamina while aiming down sights, or the [DrainStaminaBehaviour](#) and [ModifyStaminaBehaviour](#) motion graph behaviours which can be used to drain stamina over time in a specific motion state such as sprinting, or on entering a state such as jumping.

Stamina can be used to drive the [BreathingEffect](#) procedural animation on weapons. You can specify the breathing rate and strength based on fatigue using animation curves for each.

Stamina can also be used to control movement speed, slowing the character as they get tired. There is an optional exhaustion setting which can be tied to the motion graph to prevent the character from sprinting once they reach the exhaustion threshold, and re-enable sprinting once stamina has recovered to a certain point.

Inspector



Properties

Stamina

NAME	TYPE	DESCRIPTION
Stamina	Float	The current stamina of the character. This acts as the starting stamina and changes at runtime.
Max Stamina	Float	The maximum stamina of the character.
Stamina Refresh Rate	Float	The rate that stamina increases over time when no drains are applied.

Movement Speed

NAME	TYPE	DESCRIPTION
Affect Movement Speed	Boolean	Should the stamina system modify movement speed based on current stamina. The other settings will be hidden if this is false.
Min Walk Multiplier	Float	A multiplier applied to the walking speed at minimum stamina.
Min Sprint Multiplier	Float	A multiplier applied to the sprinting speed at minimum stamina.
Min Crouch Multiplier	Float	A multiplier applied to the crouching speed at minimum stamina.
Move Speed Curve	Unity AnimationCurve	A curve that defines the character speed based on stamina. The X axis is the normalised stamina (stamina / max), while the Y axis is the min to max lerp value (0 = min, 1 = max).
Walk Speed Data	String	The name of the motion data property on the motion graph that defines walk speed.
Aim Walk Speed Data	String	The name of the motion data property on the motion graph that defines walk speed when aiming.
Sprint Speed Data	String	The name of the motion data property on the motion graph that defines sprint speed.
Aim Sprint Speed Data	String	The name of the motion data property on the motion graph that defines sprint speed when aiming.
Crouch Speed Data	String	The name of the motion data property on the motion graph that defines crouch movement speed.
Aim Crouch Speed Data	String	The name of the motion data property on the motion graph that defines crouch movement speed when aiming.

Breathing

NAME	TYPE	DESCRIPTION
Breathe Slow Interval	Float	The time in seconds between breaths (when breathing slow).

NAME	TYPE	DESCRIPTION
Breathe Fast Interval	Float	The time in seconds between breaths (when breathing fast).
Breathing Rate Curve	Unity AnimationCurve	A curve that defines the breathing rate based on stamina. The X-axis is the normalised stamina (stamina / max stamina), and the Y-Axis is a lerp between slow and fast breathing rate (0 = slow, 1 = fast).
Breathing Strength Curve	Unity AnimationCurve	A curve that defines the breathing strength based on stamina. The X-axis is the normalised stamina (stamina / max stamina), and the Y-Axis is the strength of the character's breathing (0 = non-existent, 1 = heaving/panting).

Exhaustion

NAME	TYPE	DESCRIPTION
Use Exhaustion	Bool	Should the character suffer an exhaustion effect on hitting a specific stamina threshold. The other properties will be hidden if this is false.
Exhaustion Threshold	Float	The stamina level below which the character will become exhausted.
Recover Threshold	Float	The character will stop being exhausted once their stamina has recovered above this value.
Exhausted Motion Parameter	String	The name of the switch motion graph parameter that the graph uses as a condition for preventing sprinting.
Sprint Motion Parameter	String	The name of the switch motion graph parameter that the character input handler sets to tell the motion graph to start sprinting.
On Exhausted	UnityEvent	An event which is fired when the character hits the exhaustion threshold.
On Recovered	UnityEvent	An event which is fired when the character recovers from exhaustion.

See Also

[BreathingEffect](#)

[Motion Graph Parameters And Data](#)

[Modular Firearms](#)

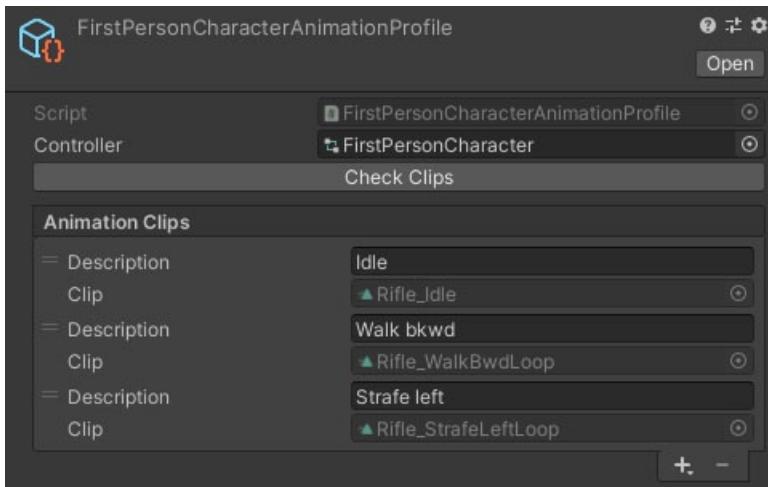
[Unity AnimationCurve](#)

FirstPersonCharacterAnimationProfile Scriptable Object

Overview

The FirstPersonCharacterAnimationProfile scriptable object asset is used to filter which animations on a character are exposed for overriding, as well as allowing more descriptive names. Attach the [animator controller](#) that you want to use and you can then add animation clips to the list below.

Inspector



Properties

Name	Type	Description
Controller	Animator Controller	The animator controller that you want to search through and expose animation clips from.

The **Check Clips** button is used if you are actively modifying the animator controller that you have assigned to this profile. It will re-scan the available animation clips and check the clips you have added to the profile.

The **Animation Clips** list contains each clip that you are exposing to components that consume this profile. To add a new clip, click the **+** button at the bottom. To remove a clip, select the entry in the list and hit the **-** button. Each element in the list has the following properties:

Name	Type	Description
Description	String	The name for the clip that will be shown when choosing overrides in a component that consumes these profiles such as the WieldableItemBodyAnimOverrides component.
Clip	Animation Clip	The animation clip that you are exposing to be overridden. This value cannot be changed.

See Also

[FPS Characters](#)

[First Person Body](#)

[Animator Controllers](#)

[WieldableItemBodyAnimOverrides](#)

HandBoneOffsets Scriptable Object

Overview

The HandBoneOffsets scriptable object contains offsets for the IK targets on a weapon or wieldable item to ensure that the character's hands and fingers align correctly at runtime. This is used alongside a [WieldableItemKinematics](#) component on the weapon, and if the weapon is edited at runtime then the hand offsets properties will be embedded in its inspector.

Inspector



Properties

Each hand in the offsets asset has the following properties:

Name	Type	Description
Hand Position Offset	Vector	The position offset for the IK target of the hand from the transform selected in the WieldableItemKinematics component.
Hand Rotation Offset	Vector	The rotation offset for the IK target of the hand from the transform selected in the WieldableItemKinematics component.
Offset Fingers	Boolean	Do the fingers on the wieldable have a rotation offset compared to the character.
Finger Offsets	Vector Array	An array of finger rotation offsets. These will only be visible if Offset Fingers is set to true. Next to each rotation offset is a copy and paste button that you can use to quickly duplicate offsets across fingers.

See Also

[First Person Body](#)

[Animator Controllers](#)

[WieldableItemKinematics](#)

First Person Camera

Overview

The first person camera in NeoFPS is a companion component for the [Unity camera](#) and adds a number of useful features for first person shooters.

Field of View

The NeoFPS camera adds additional functionality for controlling the field of view.

Firstly, it allows a horizontal field of view to be specified instead of the default vertical field of view. This is considered a standard in first person shooters, and a number of players will have their preferred FoV settings that they apply in and FPS games they play.

Secondly, it allows the field of view to be modified and animated using multipliers for zoom effects such as aiming down sights. You can modify and reset the FoV in your scripts using the following methods:

```
public void FirstPersonCamera.SetFov (float targetFovMult, float aimTime);  
  
public void FirstPersonCamera.ResetFov (float aimTime);
```

The `SetFov` method takes an FoV multiplier that multiplies based on the standard field of view settings, along with an aim time that controls the duration of the animation from the existing FoV to the new FoV. The `ResetFov` method takes the same aim time parameter which controls how long it takes to return to the standard field of view settings.

Camera Offset

The camera offset is a position and rotation offset that can be applied and removed to move and animate the camera within the hierarchy. An example use is the modular firearms' [HeadMoveAimer](#) which actually moves the head down to the weapon instead of the weapon up to the camera. This allows a lean angle to be added and for the camera rotation to be modified making for an interesting aim effect. You can modify and reset the offset in scripts using the following methods:

```
public void FirstPersonCamera.SetOffset (Vector3 posOffset, Quaternion rotOffset, float aimTime);  
  
public void FirstPersonCamera.ResetOffset (float aimTime);
```

These methods take offset parameters, along with an aim time parameter that controls how long it takes to reach the offset and return.

Events

The first person camera also contains the following event:

```
public static event FirstPersonCamera.UnityAction<FirstPersonCamera> onCurrentCameraChanged;
```

Subscribing to this event will notify when the camera changes (when it is being looked through or not). This can be used for things such as enabling a scene or spectator camera if the player dies.

Camera Constraints

To aid in matching the first person camera to an animated character skeleton, whilst still allowing procedural aim-down-sights, NeoFPS includes a simple camera constraints system. The [FirstPersonCameraTransformConstraints](#) component will match position and rotation to one or more targets with priority, weight and damping. The component has properties to set up a default constraint (such as the character's skeleton head bone) and then new constraints can be added as required.

For aiming down sights, the [CameraConstraintsAimer](#) firearm module sets a camera constraint based on the firearm's root transform with an added offset. This allows the procedural animations to move the gun separate to the aim.

To layer in your own constraints you can use the API:

```
public void FirstPersonCameraTransformConstraints.AddPositionConstraint(IFirstPersonCameraPositionConstraint constraint, int priority, float blendDuration);
public void FirstPersonCameraTransformConstraints.AddRotationConstraint(IFirstPersonCameraRotationConstraint constraint, int priority, float blendDuration);
public void
FirstPersonCameraTransformConstraints.RemovePositionConstraint(IFirstPersonCameraPositionConstraint constraint, float blendDuration);
public void
FirstPersonCameraTransformConstraints.RemoveRotationConstraint(IFirstPersonCameraRotationConstraint constraint, float blendDuration);
```

See Also

[FirstPersonCamera Behaviour](#)

[additive-Effects](#)

[Unity Camera](#)

Aim Controllers

Overview

Aim controllers take input and control a character's look direction before additive effects are applied.

Turn Rate

The aimer turn rate can be altered using the `turnRateMultiplier` property. This is used when setting the first person camera field of view, for example when aiming down sights.

Setting Yaw And Pitch

You can modify the aim controller's rotation from scripts using the following methods:

```
void AddYaw (float rotation);  
  
void AddPitch (float rotation);  
  
void AddRotation (float y, float p);
```

Aim Constraints

The aim controller can be constrained to a specific range of rotations on either the vertical or horizontal axes or both. This is used in situations such as climbing ladders or using a mounted turret, where it would be unnatural for the character to be able to turn a full 360 degrees.

If the aim controller rotation is outside of the constraints when they are set, then it uses a damped rotation to smoothly turn into the constraints.

Since the fps character is able to change their up direction, the yaw constraint must use a direction vector for the constraints center. This means that as the character tilts, they can still constrain to the direction. Camera pitch is relative to the character.

In order to set the aim controller constraints from a script you can use the following methods on the aim controller component:

```
void SetYawConstraints(Vector3 forward, float range);  
  
void SetPitchConstraints(float minimum, float maximum);  
  
void ResetYawConstraints();  
  
void ResetPitchConstraints();
```

See Also

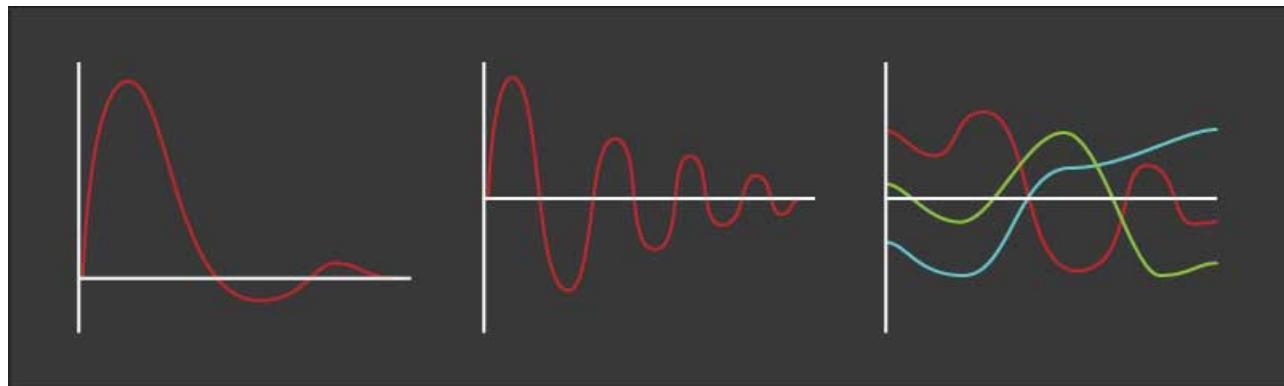
[FirstPersonCamera Behaviour](#)

[additive-Effects](#)

[Unity Camera](#)

Additive Transforms and Effects

Overview



NeoFPS uses a system called additive transforms to manage camera and object transform effects such as recoil and shake. In order to make use of additive transforms, the [AdditiveTransformHandler](#) is added to an object and then various additive effects are attached to it. It is very simple to write custom additive effects, but a number of useful effects are included with NeoFPS including:

NAME	DESCRIPTION	USE
AdditiveKicker	A simple knock and spring return. Triggered by the CharacterEventKickTrigger and ImpactHandlerKickTrigger .	Any
AdditiveJiggle	A simple rotation around the z-axis with a bouncy spring return.	Any
BodyLean	Lean to either side	Body
BodyTilt	Tilt the body in an arbitrary direction	Body
BreathingEffect	Applies a breathing motion to the selected weapon.	Weapon
CameraShake	Apply continuous and one-off shakes to the camera.	Camera
CharacterMovementSway	A position and rotation offset applied to the weapon based on character velocity.	Weapon
CharacterRecoilEffect	Applies recoil spring animation to the character. Driven by a BetterSpringRecoilHandler firearm module.	Body
HeadBob	(Deprecated) Position bob using curves	Camera
HeadBobV2	An enhanced head bob with more control over animation curves, aim compensation, and in-game strength settings.	Body
HeadDuck	A simple lowering of the head used for charged jumps	Camera
FirearmRecoilEffect	Applies recoil spring animation to a modular firearm . Driven by a BetterSpringRecoilHandler firearm module.	Weapon
OverShoulder	Rotate the head to look backwards, while leaving the weapon and body pointing forwards.	Camera
PeekVertical	Peek over or under obstacles by moving the head up or down.	Camera

NAME	DESCRIPTION	USE
PositionBob	Position bob using curves. Syncs and blends bob between head and weapon based on player settings by sharing bob settings with a PositionBobData scriptable object.	Camera And Weapon
ProceduralSpineLean	Similar to the BodyLean effect, but instead of rotating the upper body based on a lowered pivot, it applies rotations to the character spine procedurally via its ProceduralSpineAimMatcher .	Body
RotationBob	Rotation bob using curves.	Weapon
TransformMatcher	Matches the offset of one transform relative to another. Used to blend keyframe animation in with procedural effects.	Camera
WeaponAimAmplifier	Multiplies the movement of the camera back onto the held weapon to add weight and momentum.	Weapon
WeaponBob	(Deprecated) A bob effect for weapons, similar to the head bob but with rotation too.	Weapon
WeaponMomentumSway	Similar to the WeaponAimAmplifier, but changes the position instead of the rotation.	Weapon

The [AdditiveTransformHandler](#) will check each of the effects every tick (the update frequency can be set in the component properties) and then layer them on top of each other in order to create a combined offset. Translation and rotation are handled separately, meaning that the rotation offset of an effect will not alter the position offset of another effect further along the chain. This makes the outcome more predictable when a number of effects are applied at once.

You can also specify a pivot point for the resulting transform if desired.

Footstep driven animation such as head or weapon bob is driven through a step tracking system attached to the character's motion controller. This ensures that any animations like this sync up, are only active while in the correct movement state, and are correctly driven by the character's velocity. The character's stride length (and therefore bob speed while moving) is controlled by adding a [TrackSteps](#) motion graph behaviour to the relevant states or sub-graphs in your character's motion graph.

See Also

[AdditiveTransformHandler](#)

[FirstPersonCamera](#)

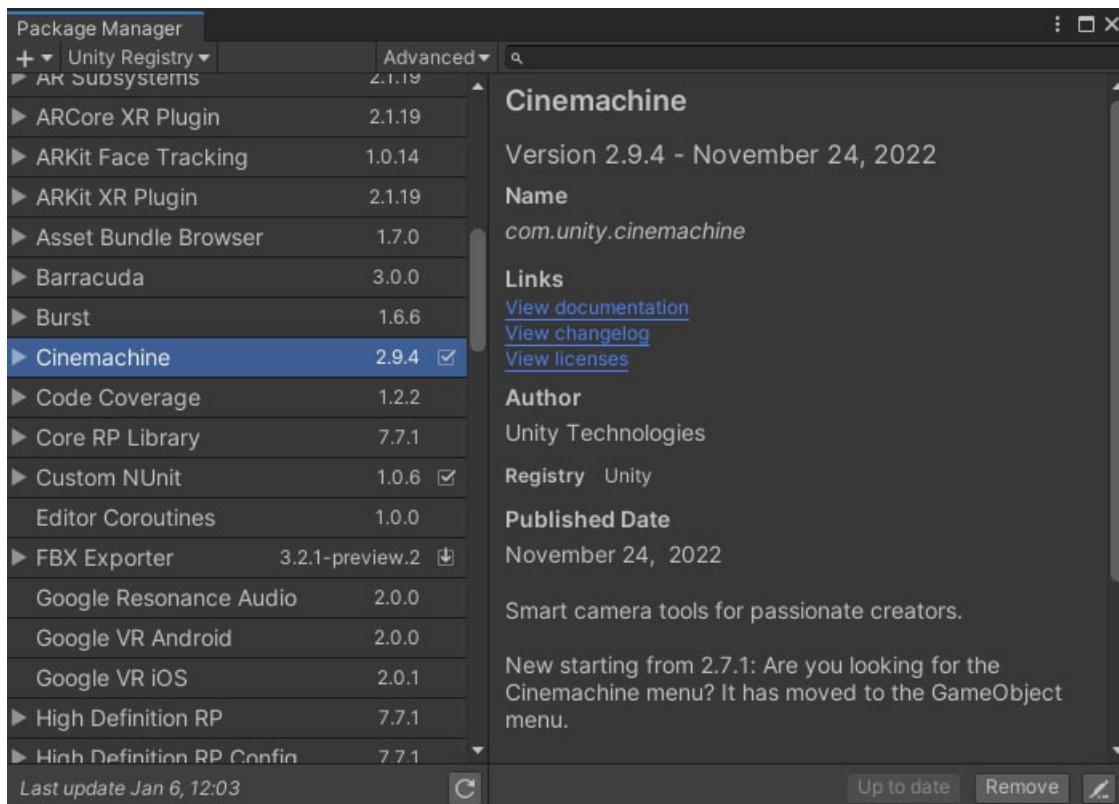
Cinemachine Extension

Overview

The Cinemachine package adds a powerful virtual camera system to Unity. The extension package for NeoFPS adds Cinemachine based first person camera components, allowing you to use the procedural camera spring systems and field of view features of NeoFPS with Cinemachine's virtual camera. This makes it easy to maintain a consistent post processing look and feel, and create camera transitions for cutscenes. It also contains demo prefabs and a demo scene to test the implementation.

Installation

Install the latest Cinemachine package from the package manager before extracting the extension package file or you will get script errors.



Note: The package manager might look different based on your Unity version. Consult Unity's documentation if you have trouble finding the required package

Once the Cinemachine package is imported, you can extract the extension package found in *NeoFPS/Extensions*. This will create a new folder with the scripts and demo assets for the integration at *NeoFPS/Extensions/Cinemachine*.

Adding The Cinemachine Camera To Your Characters

The cinemachine extension replaces the Unity camera in the character's hierarchy with a virtual camera instead, and then replaces the [FirstPersonCamera](#) component in its hierarchy with a [CinemachineFirstPersonCamera](#) instead. To achieve this, follow these steps:

- Find the camera object in your character's hierarchy
- Remove the camera component, along with the audio listener, post processing fix and any camera effects such as flare layers
- Add a [CinemachineVirtualCamera](#) component and set the priority high enough to be the primary camera
- Find the object in the character hierarchy with the [FirstPersonCamera](#) component (usually the parent of the camera object)
- Replace the [FirstPersonCamera](#) component with a [CinemachineFirstPersonCamera](#) component, duplicating the settings and pointing it at the new virtual camera
- Add the [CinemachineCameraController](#) to your scene or use your own camera setup with a Cinemachine brain

Blending Between Virtual Cameras

Switching between cinemachine cameras works best with the **Cut** default transition type. Since the arms and weapon are attached to the virtual camera, a slow blend into the first person view will make the arms and weapons seem detached until the blend is complete. This is particularly noticeable when spawning the player character with move input held down. In this situation, the camera will actually be blending from the static scene virtual camera in the spawn point to the new first person virtual camera, and so the arms and weapon will appear to move forwards independently and the camera will catch up. The **Cut** transition type will instantly switch to the first person camera and prevent this.

For more information see the official Cinemachine documentation section: [Blending Between Virtual Cameras](#).

See Also

[CinemachineFirstPersonCamera Behaviour](#)

[Cinemachine Documentation](#)

Camera Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Camera Bounce/Bob Is Too Extreme

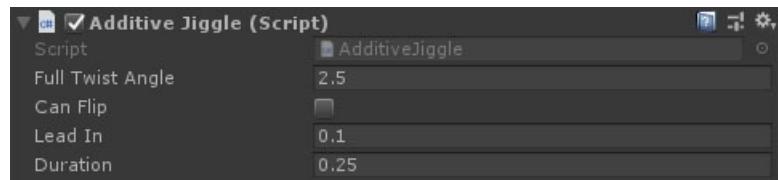
You can find the various camera procedural animation effects on the **PlayerCameraRoot** in the demo characters. Each of these is a component, and has a number of properties that can be modified to increase/decrease the effects.

AdditiveJiggle MonoBehaviour

Overview

The additive jiggle is used to knock the object's rotation (roll) and then spring back.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Full Twist Angle	Float	The angle (either side of the axis) of a full strength jiggle.
Can Flip	Boolean	If true, the jiggle could rotate in either direction.
Lead In	Float	The time taken to ease into the jiggle.
Duration	Float	The time taken for the jiggle spring to ease out.

See Also

[Additive Transforms](#)

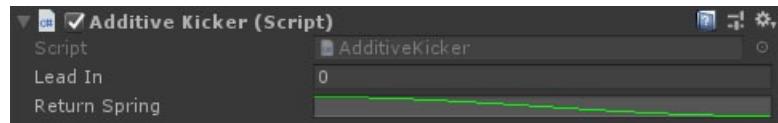
[AdditiveKicker](#)

AdditiveKicker MonoBehaviour

Overview

The additive kicker is used to knock the camera or object and then spring back. Pair it with [CharacterImpactKicker](#) and [KickerImpactHandler](#) behaviours in order to consume character impact events.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lead In	Float	The time taken to ease into the kick.
Return Spring	AnimationCurve	The return spring is an animation curve that dictates how the kicker returns from the kick angle/position (1 on the y-axis) to its original state (0 on the y axis).

See Also

[Additive Transforms](#)

[CharacterImpactKicker](#)

[KickerImpactHandler](#)

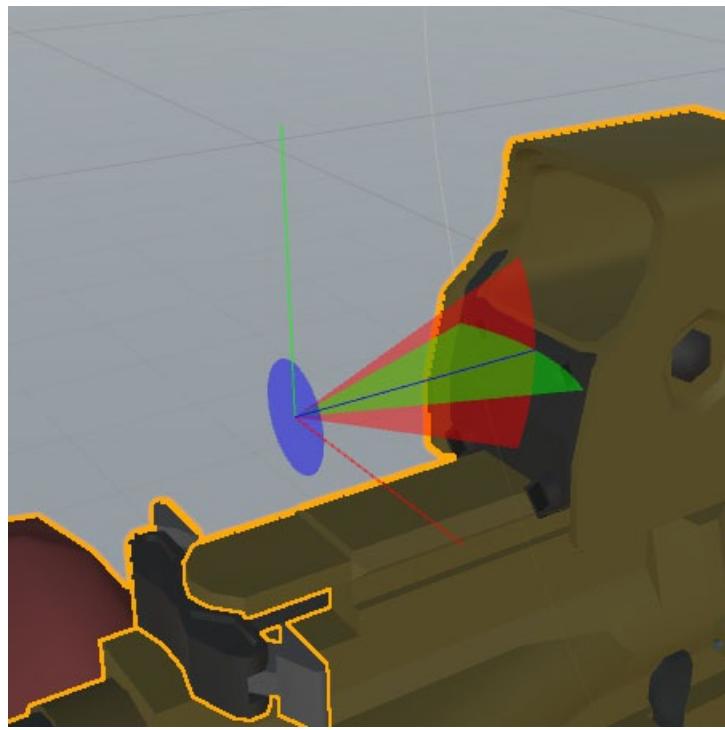
[Unity AnimationCurve](#)

AdditiveTransformHandler MonoBehaviour

Overview

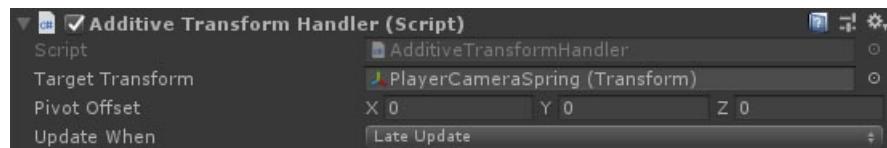
The AdditiveTransformHandler behaviour sums together all the additive transform effects applied to it and uses them to transform an object such as the camera.

In The Scene View



With a gameobject selected that uses an AdditiveTransformHandler component, the guide handle will be visible in the scene view. This represents the point the target transform will rotate around.

Inspector



Properties

Name	Type	Description
Target Transform	Transform	The transform that the handler affects.
Pivot Offset	Vector3	The offset from the transform origin for the pivot point to rotate around.
Update When	Dropdown	When should the additive transform effects be calculated and applied. Options are: Update , LateUpdate , FixedUpdate , FixedAndLerp , FixedAndLateLerp (The "Lerp" options calculate in fixed update and then interpolate between results during Update/LateUpdate for smooth results).

See Also

[Additive Transforms](#)

WeaponBob

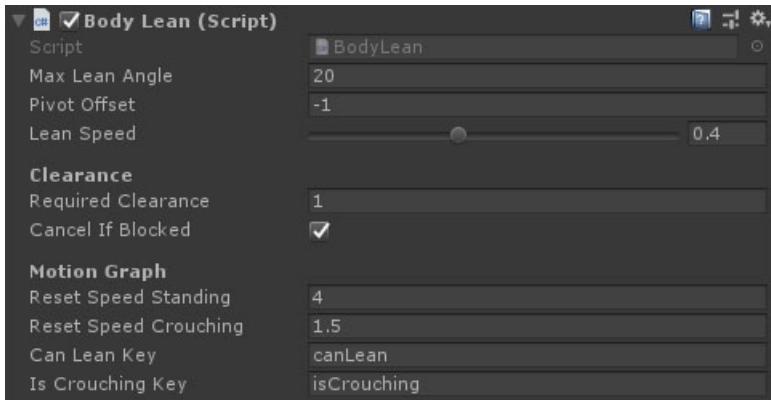
Unity AnimationCurve

BodyLean MonoBehaviour

Overview

The BodyLean behaviour is used to lean the character body to either side.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Lean Angle	Float	The angle to lean to either side when lean power is set to 1 or -1.
Pivot Offset	Float	The downward offset for the pivot point from the transform origin.
Lean Speed	Float	The speed of transitioning from 1 lean value to another. 1 is nearly instant.
Weapon Tilt	Float	How much of the lean rotation is reflected in the weapon.
Head Counter Tilt	Float	A counter rotation of the head compared to the weapon.
Required Clearance	Float	The distance to check from the center line for clearance space to lean. The lean amount will be capped based on the result.
Cancel If Blocked	Boolean	If there is no clearance space then return the lean amount to 0 until manually re-applied.
Reset Speed Standing	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Reset Speed Crouching	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Can Lean Key	String	The key to a motion graph switch parameter that dictates if the character can lean or not.
Is Crouching Key	String	The key to a motion graph switch parameter that dictates if the character is crouching or standing.

See Also

[Additive Transforms](#)

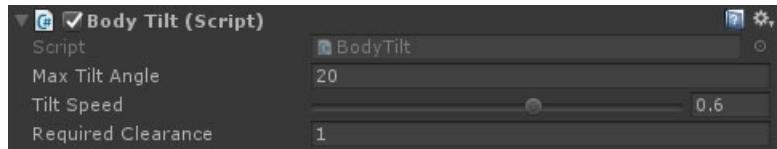
[Motion Graph Parameters And Data](#)

HeadBob MonoBehaviour

Overview

The BodyTilt behaviour is used to lean the character body based on an input vector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Tilt Angle	Float	The maximum angle the character can tilt. Angles above this will be capped.
Tilt Speed	Float	The speed of transitioning from 1 tilt value to another. 1 is nearly instant.
Required Clearance	Float	The distance to check from the center line for clearance space to lean. The lean amount will be capped based on the result.

See Also

[Additive Transforms](#)

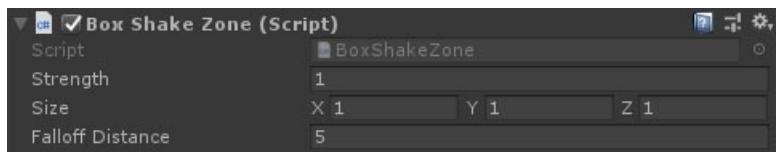
[Motion Graph Parameters And Data](#)

BoxShakeZone MonoBehaviour

Overview

The BoxShakeZone behaviour defines a 3D box area with a constant shake value used by the [CameraShake](#) additive transform effect. The shake zone is not affected by the scale of the object it's attached to or its parent. To change its size, use the properties in the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Size	Vector	The dimensions of the box along each axis (centered on position).
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

[Additive Transforms](#)

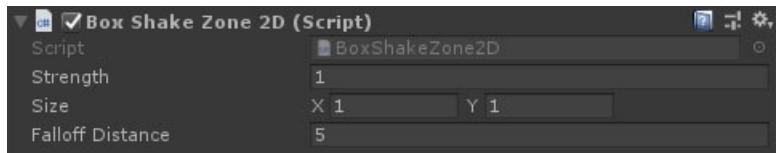
[CameraShake](#)

BoxShakeZone2D MonoBehaviour

Overview

The BoxShakeZone2D behaviour defines a 2D box area (on the x-y plane) with a constant shake value used by the [CameraShake](#) additive transform effect. The shake zone is not affected by the scale of the object it's attached to or its parent. To change its size, use the properties in the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Size	Vector	The dimensions of the box along each axis (centered on position).
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

[Additive Transforms](#)

[CameraShake](#)

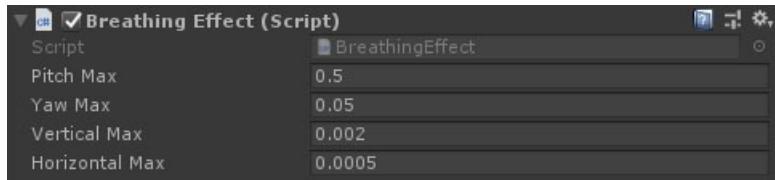
BreathingEffect MonoBehaviour

Overview

The BreathingEffect behaviour applies a procedural breathing animation based on a character's breath handler.

Example breath handlers include the [SimpleBreathHandler](#) which has a constant breathing rate, or the [StaminaSystem](#) which modifies the breathing rate and strength based on fatigue.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Pitch Max	Float	The maximum pitch rotation at breathing strength 1.
Yaw Max	Float	The maximum yaw rotation at breathing strength 1.
Vertical Max	Float	The maximum vertical position offset at breathing strength 1.
Horizontal Max	Float	The maximum horizontal position offset at breathing strength 1.

See Also

[Additive Transforms](#)

[SimpleBreathHandler](#)

[StaminaSystem](#)

CameraShake MonoBehaviour

Overview

The CameraShake behaviour is used to add constant and one-shot shake effects to the first person camera.

Constant shake effects can be applied using shake zones such as the [BoxShakeZone](#), [BoxShakeZone2D](#), [SphereShakeZone](#) and [CircleShakeZone](#). You can also set the global shake value directly with scripts using the following code:

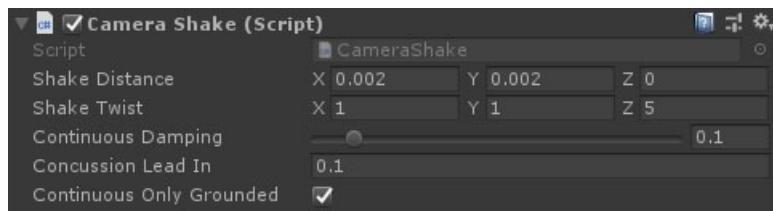
```
ShakeHandler.globalShake = shakeAmount;
```

Where `shakeAmount` is a float value between 0 (no shake) and 1 (maximum shake).

You can apply a one-shot shake using the following method:

```
public static void Shake(Vector3 position, float innerRadius, float falloffDistance, float strength, float duration, bool requiresGrounding = false)
```

Inspector



Properties

NAME	TYPE	DESCRIPTION
Shake Distance	Vector	The distance the camera can move either side of the origin on each axis at a shake strength of 1.
Shake Twist	Vector	The max rotation (in each direction) for a shake strength of 1.
Continuous Damping	Float	Damping smooths the blend between different continuous shake strengths.
Concussion Lead In	Float	The time it takes for the shake to go from 0 to 1.
Continuous Only Grounded	Boolean	Should continuous shake only be applied while the character this is attached to is grounded (if there is a character).

See Also

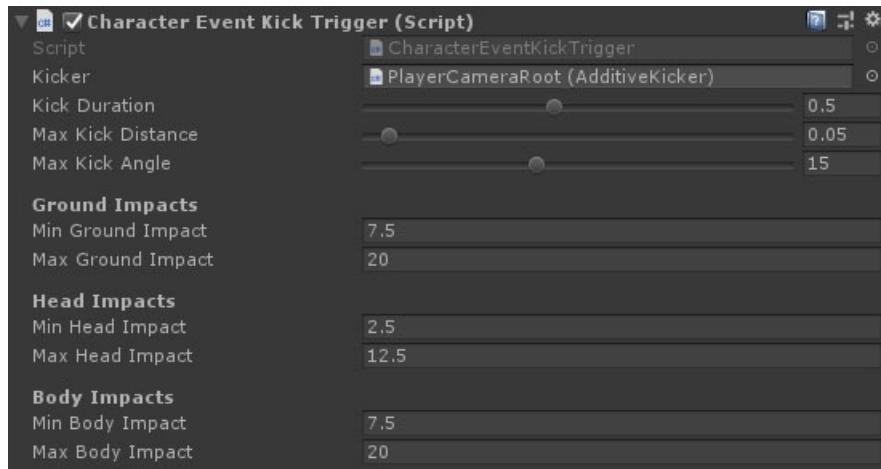
[Additive Transforms](#)

CharacterEventKickTrigger MonoBehaviour

Overview

The CharacterEventKickTrigger consumes character impact events such as landings and bullet hits, and uses them to drive an additive kicker effect.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Kicker	AdditiveKicker	The additive kicker used to react to the force.
Kick Duration	Float	The time taken to recover from the kick.
Max Kick Distance	Float	The downward position kick distance at max impulse.
Max Kick Angle	Float	The forward kick angle at max impulse.
Min Ground Impact	Float	A ground impact impulse with magnitude lower than this will be ignored.
Max Ground Impact	Float	The ground impact impulse magnitude that gives the maximum kick.
Min Head Impact	Float	A head impact impulse with magnitude lower than this will be ignored.
Max Head Impact	Float	The head impact impulse magnitude that gives the maximum kick.
Min Body Impact	Float	A body impact impulse with magnitude lower than this will be ignored.
Max Body Impact	Float	The body impact impulse magnitude that gives the maximum kick.

See Also

[Additive Transforms](#)

[AdditiveKicker](#)

CharacterRecoilEffect MonoBehaviour

Overview

The CharacterRecoilEffect behaviour is a part of the modular firearm recoil system and applies the character (body and head) based component of the recoil animation. This is paired with the [FirearmRecoilEffect](#) behaviour to apply the firearm component of the animation, and controlled by the [BetterSpringRecoil](#) firearm module.

Inspector



Properties

NAME	TYPE	DESCRIPTION

See Also

[Additive Transforms](#)

[FirearmRecoilEffect](#)

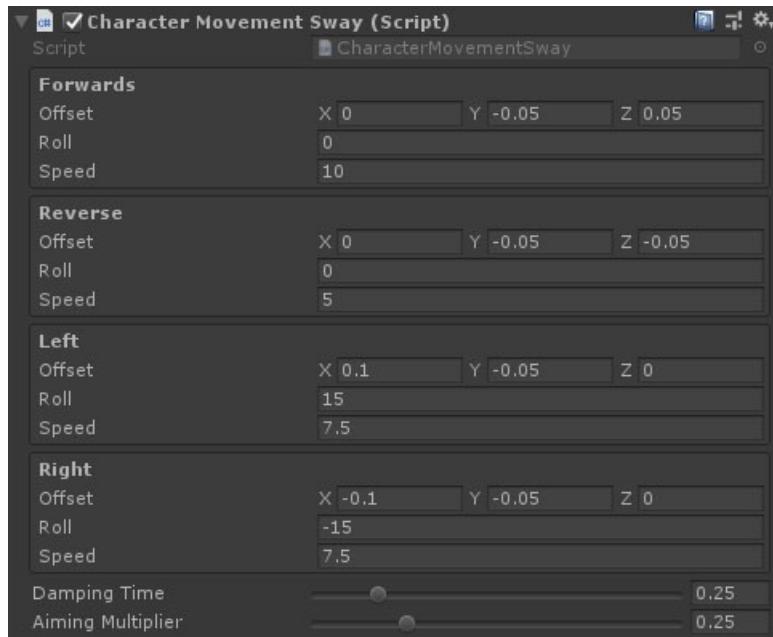
[BetterSpringRecoil](#)

CharacterMovementSway MonoBehaviour

Overview

The CharacterMovementSway behaviour adds an offset to the held weapon based on the character's velocity. This can help exaggerate movement and make the weapon feel more dynamic.

Inspector



Properties

Name	Type	Description
Damping Time	Float	Approximately the time it will take to reach the target sway. A smaller value will reach the target faster.
Aiming Multiplier	Float	A multiplier applied to the offset when aiming down sights. This helps make the weapon more controlled

There are also 4 sway limit sections - one for each direction. When moving diagonally, the sway will blend between the closest 2. Each of these section has the following properties:

Name	Type	Description
Offset	Vector	The position offset to apply to the weapon when at or above the target speed for this direction.
Roll	Float	The rotation around the weapon's forward axis when at or above the target speed for this direction.
Speed	Float	The speed in this direction at or above which the full offset and roll will be applied.

See Also

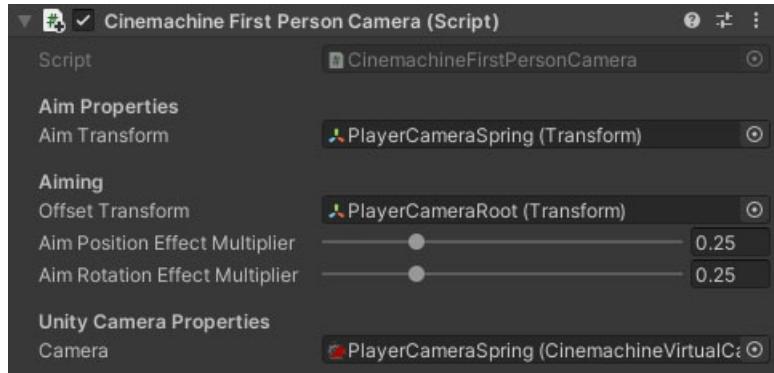
[Additive Transforms](#)

CinemachineFirstPersonCamera MonoBehaviour

Overview

The CinemachineFirstPersonCamera MonoBehaviour is used with the standard Unity [Camera][unity-camera] to add useful features.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Aim Transform	Transform	The transform to use for accurate shooting. If you add extra spring effects to the camera that don't affect the gun, you might want to set this to something higher up the hierarchy.
Default FoV	Float	The default vertical field of view. This will also change the horizontal fov below.
Horizontal 16:9	Float	The default horizontal field of view for a 16:9 screen. This will also change the vertical fov above.
Offset Transform	Transform	The offset from standard upright position for moving the head. Used for aiming down sights, etc.
Aim Position Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.
Aim Rotation Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.
Camera	CinemachineVirtualCamera	The cinemachine camera for the first person view.

See Also

[Additive Effects](#)

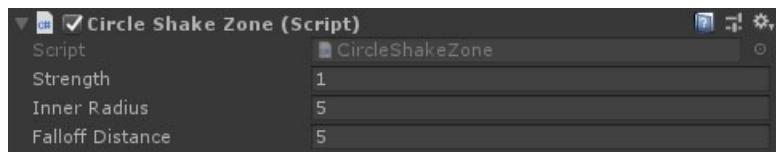
[Cinemachine Documentation](#)

CircleShakeZone MonoBehaviour

Overview

The CircleShakeZone behaviour defines a 2D circular area (on the x-y plane) with a constant shake value used by the [CameraShake](#) additive transform effect. The shake zone is not affected by the scale of the object it's attached to or its parent. To change its size, use the properties in the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Inner Radius	Float	The inner radius of the circle. Inside this, the strength is constant.
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

[Additive Transforms](#)

[CameraShake](#)

CutsceneCamera MonoBehaviour

Overview

The CutsceneCamera behaviour is used to switch camera from a first person character to an externally controlled camera. It handles disabling the player character's input, and hiding the HUD. Once the camera is disabled again, control is restored and the HUD is re-enabled.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Can Skip	Boolean	Can the cutscene be skipped by holding the use button.
Skip Hold	Float	The amount of time the use button must be held to skip the cutscene.
On Skip	UnityEvent	An event fired when the cutscene is skipped.

See Also

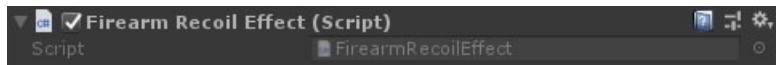
[Additive Transforms](#)

FirearmRecoilEffect MonoBehaviour

Overview

The FirearmRecoilEffect behaviour is a part of the modular firearm recoil system and applies the character (body and head) based component of the recoil animation. This is paired with the [CharacterRecoilEffect](#) behaviour to apply the body and head components of the animation, and controlled by the [BetterSpringRecoil](#) firearm module.

Inspector



Properties

NAME	TYPE	DESCRIPTION

See Also

[Additive Transforms](#)

[CharacterRecoilEffect](#)

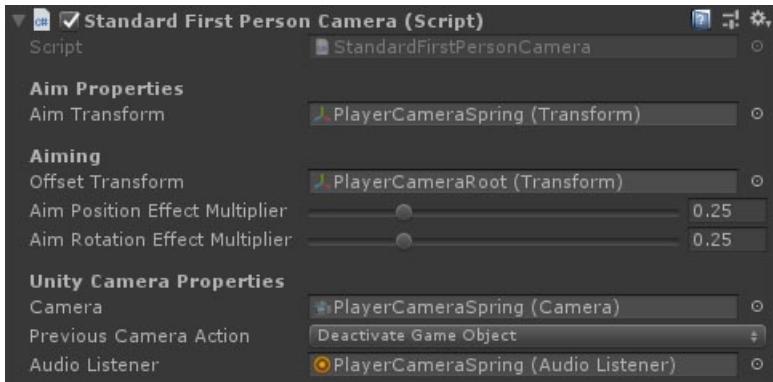
[BetterSpringRecoil](#)

FirstPersonCamera MonoBehaviour

Overview

The FirstPersonCamera MonoBehaviour is used with the standard Unity [Camera](#) to add useful features.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Aim Transform	Transform	The transform to use for accurate shooting. If you add extra spring effects to the camera that don't affect the gun, you might want to set this to something higher up the hierarchy.
Default FoV	Float	The default vertical field of view. This will also change the horizontal fov below.
Horizontal 16:9	Float	The default horizontal field of view for a 16:9 screen. This will also change the vertical fov above.
Offset Transform	Transform	The offset from standard upright position for moving the head. Used for aiming down sights, etc.
Aim Position Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.
Aim Rotation Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.
Camera	Camera	The main camera for the first person view.
Previous Camera Action	Dropdown	What to do with the main camera in the scene? Use this to prevent wasted render cycles and multiple listeners. Options: DeactivateGameObject , DisableComponent , DestroyGameObject , Ignore .
Audio Listener	Audio Listener	The audio listener for the first person view.

See Also

[Additive Effects](#)

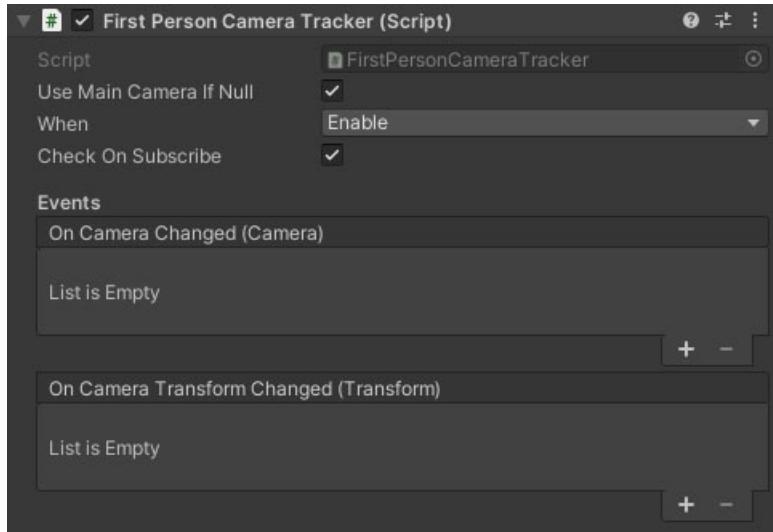
[Unity Camera](#)

FirstPersonCameraTracker MonoBehaviour

Overview

The FirstPersonCameraTracker MonoBehaviour can be placed on any gameobject, and will track the active [FirstPersonCamera](#). You can then use the provided [unity events] to tie into other scripts and behaviours.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the event handler subscribe. Using Enable will only connect the event handler up while the object or component is enabled.
Check On Subscribe	Boolean	Should the camera be checked immediately when subscribing or wait for the first camera changed event.
On Camera Changed	Unity Event	A unity event called when the first person camera changes. Pointing this at a method on your behaviours that takes a Camera component will pass the current camera to that method when the event fires.
On Camera Transform Changed	Unity Event	A unity event called when the first person camera changes. Pointing this at a method on your behaviours that takes a Transform component will pass the current camera's transform to that method when the event fires.

See Also

[FirstPersonCamera](#)

[Unity Events](#)

FirstPersonCameraTransformConstraints MonoBehaviour

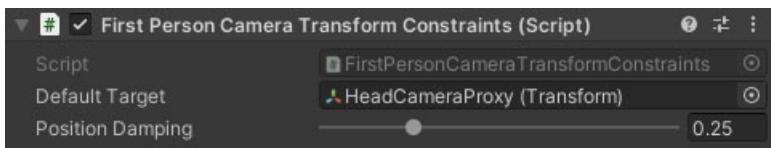
Overview

The FirstPersonCameraTransformConstraints behaviour acts as an enhanced, combined position constraint and rotation constraint. It aligns the camera object with one or more targets based on priority and strength. An example usage is in first person characters with a full animated body. The constraints can be used to align the camera to the head bone in the character skeleton. You can add new constraints at runtime using a priority system, and position and rotation can each be assigned different strengths, meaning that the constraints will blend with the next lowest priority constraint if they have a strength lower than 1. In the full body character example, this allows a firearm using the [CameraConstraintsAimer](#) firearm module to align the camera to the weapon optics by setting runtime constraints.

To add or remove constraints you can use the following methods:

```
public void AddPositionConstraint(IFirstPersonCameraPositionConstraint constraint, int priority, float blendDuration);
public void AddRotationConstraint(IFirstPersonCameraRotationConstraint constraint, int priority, float blendDuration);
public void RemovePositionConstraint(IFirstPersonCameraPositionConstraint constraint, float blendDuration);
public void RemoveRotationConstraint(IFirstPersonCameraRotationConstraint constraint, float blendDuration);
```

Inspector



Properties

NAME	TYPE	DESCRIPTION
Default Target	Transform	The default target will act as a position constraint from start. Use this for full body to match to the head bone. This only constrains position, not rotation.
Position Damping	Float	A damping factor applied to the default target position constraint to prevent excessive jerkiness during animation blends.

See Also

[CameraConstraintsAimer](#)

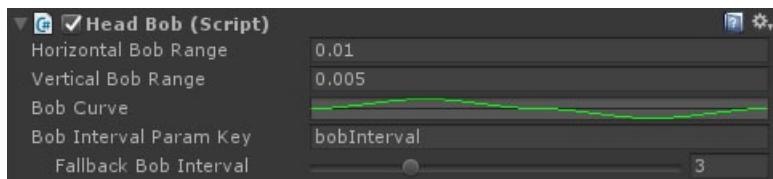
HeadBob MonoBehaviour (Deprecated)

Overview

The HeadBob behaviour is used to move the camera in sync with character movement. This behaviour has been deprecated and replaced with the [PositionBob](#) and [RotationBob](#) systems which sync weapon and head bobbing, and allow player settings to reduce head movement.

The head bob can tie into the motion graph and subscribe to a [float parameter](#) to allow the motion graph elements to control when the bob occurs and the bob rate. Without this, the bob occurs whenever the character is grounded, and the bob rate is based on movement speed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the weapon bob.
Bob Interval Param Key	FloatParameter	The name of a float parameter on the character motion graph that sets the bob interval distance.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle. If the parameter key above is set then this value will only be used if the parameter can not be found.

See Also

[Additive Transforms](#)

[WeaponBob](#)

[Unity AnimationCurve](#)

HeadBobV2 MonoBehaviour (Deprecated)

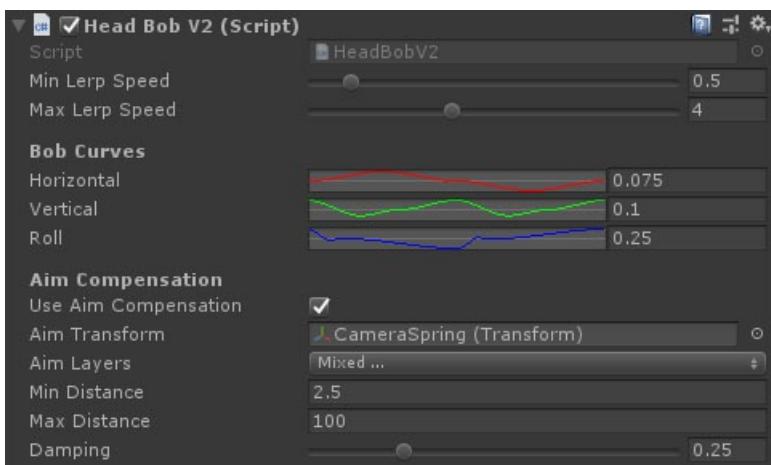
Overview

The HeadBobV2 behaviour is a new spring effect which uses the character step tracking system to sync complex head bob to the character. Alongside simple position bob it adds the following features:

- In-game strength settings via the gameplay options or quick-options popup
- Individual animation curves for horizontal offset, vertical offset and roll
- Aim compensation which adds a counter rotation to keep the crosshair fixed on objects as the head moves

Despite the name, the HeadBobV2 component is intended to be applied to the upper body spring so that the weapon moves with it.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.
Horizontal	AnimationCurve + Float	The maximum position offset along the horizontal axis in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Vertical	AnimationCurve	The maximum position offset along the vertical axis in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Roll	AnimationCurve	The maximum angle to roll the camera in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Use Aim Compensation	Boolean	Aim compensation involves rotating the camera so that the crosshair stays fixed on the same point in space.
Aim Transform	Transform	The transform to use for camera/crosshair casting.

NAME	TYPE	DESCRIPTION
Aim Layers	LayerMask	The layers to check against for aim depth.
Min Distance	Float	The minimum distance to compensate against. At very close distances, the bob will have to rotate larger angles to keep the crosshair fixed on target.
Max Distance	Float	The maximum distance to compensate against.
Damping	Float	A damping against the crosshair target. Prevents objects crossing the camera at close ranges from causing sudden shifts.

See Also

[Additive Transforms](#)

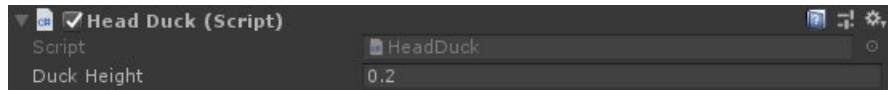
[Unity AnimationCurve](#)

HeadDuck MonoBehaviour

Overview

The HeadDuck effect lowers the head. It is used in charged jumps.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Duck Height	Float	The distance to duck the head downwards.

See Also

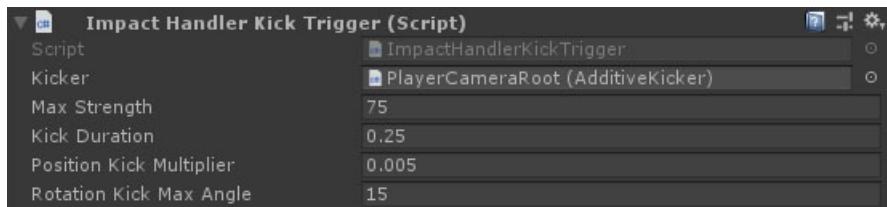
[Additive Transforms](#)

ImpactHandlerKickTrigger MonoBehaviour

Overview

The ImpactHandlerKickTrigger behaviour is used to drive the [AdditiveKicker](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Kicker	AdditiveKicker	The additive kicker used to react to the force.
Max Strength	Float	The strength cap. Above this strength, the kick will not increase.
Kick Duration	Float	The time taken to recover from the kick.
Position Kick Multiplier	Float	The camera moves along the direction of the force by strength * position multiplier.
Rotation Kick Max Angle	Float	The rotation kick angle at full strength.

See Also

[Additive Transforms](#)

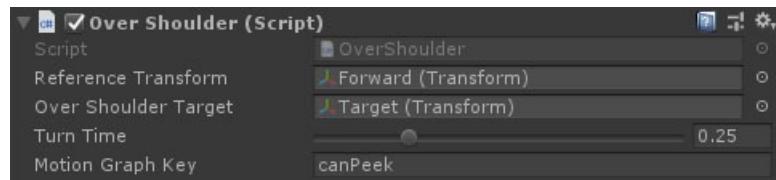
[AdditiveKicker](#)

OverShoulder MonoBehaviour

Overview

The OverShoulder behaviour allows you look over your shoulder, keeping your weapon pointing in its original direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
ReferenceTransform	Transform	The transform to use as the un altered aim direction.
OverShoulderTarget	Transform	The transform to look down (forwards) when looking over shoulder.
TurnTime	Float	The time taken to turn.
MotionGraphKey	String	The key to a motion graph switch parameter that dictates if the character can peek or not

See Also

[Additive Transforms](#)

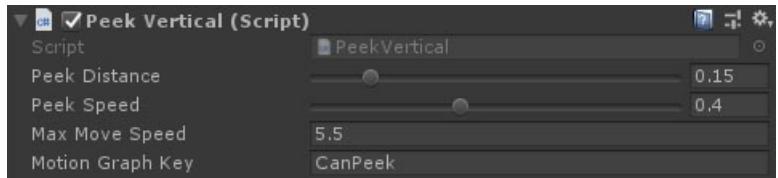
[Motion Graph Parameters And Data](#)

PeekVertical MonoBehaviour

Overview

The PeekVertical behaviour allows you to peek over or under obstacles by moving the head up/down.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Peek Distance	Float	The distance up or down to move the camera when peeking up or down.
Peek Speed	Float	The speed the character can change lean amount.
Max Move Speed	Float	The maximum speed the character can travel before the peek is cancelled. (0 = no max speed).
Motion Graph Key	String	The key to a motion graph switch parameter that dictates if the character can peek or not.

See Also

[Additive Transforms](#)

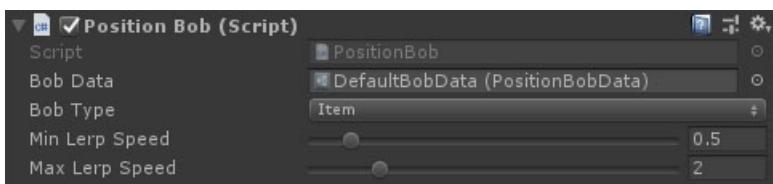
[Motion Graph Parameters And Data](#)

PositionBob MonoBehaviour

Overview

The PositionBob behaviour is used to apply a bob effect to the player character's head and weapon. The bob syncs steps with the [RotationBob](#) effect and the various sprint handlers. It can also blend the bob effect between the head and weapon object to keep a consistent effect while reducing head movement for people that find it causes them motion sickness. This is controlled via the head bob setting in the [FpsGameplaySettings](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bob Data	PositionBobData	The bob animation data, shared between the head and the item/weapon.
Bob Type	Dropdown	Is this bob being applied to the head or the item (allows the effect to blend between the 2 with similar results based on game settings).
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.

See Also

[Additive Transforms](#)

[RotationBob](#)

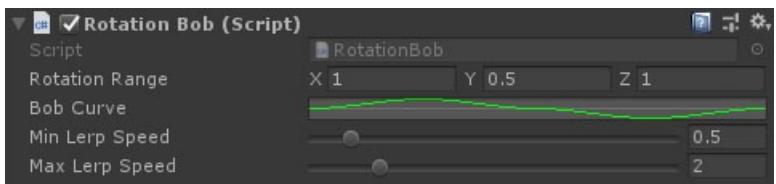
[Gameplay Settings](#)

RotationBob MonoBehaviour

Overview

The RotationBob behaviour is used to apply a bob effect to the player character's weapon. The bob syncs steps with the [PositionBob](#) effect and the various sprint handlers. Rotation bob should be used sparingly and not applied to the character body or head as it can be very disorientating.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rotation Range	Vector	The maximum rotation on each axis at the peak of the bob. Negative values essentially offset the timing for that access to the other foot.
Bob Curve	[AnimationCurve][unity-animationcurve]	The curve over one step cycle for the bob effect.
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.

See Also

[Additive Transforms](#)

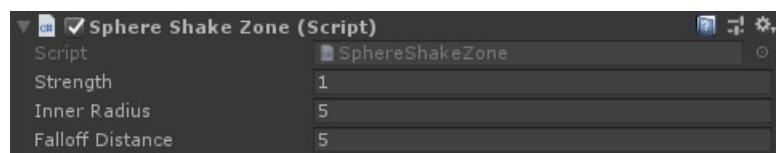
[PositionBob](#)

SphereShakeZone MonoBehaviour

Overview

The SphereShakeZone behaviour defines a spherical area with a constant shake value used by the [CameraShake](#) additive transform effect. The shake zone is not affected by the scale of the object it's attached to or its parent. To change its size, use the properties in the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Inner Radius	Float	The inner radius of the sphere. Inside this, the strength is constant.
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

[Additive Transforms](#)

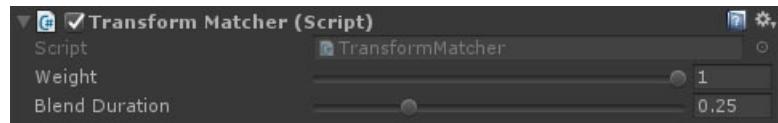
[CameraShake](#)

TransformMatcher MonoBehaviour

Overview

The TransformMatcher behaviour is an additive effect that matches the position and rotation of one transform relative to another. It is useful for layering keyframed animation in with the procedural animation of the other additive effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Weight	Float	The strength of the effect. 1 matches the movement absolutely, while 0 is no movement.
Blend Duration	Float	The time it takes to blend in or out of the movement when the transforms are changed.

See Also

[Additive Transforms](#)

[FirearmTransformMatchSetter](#)

WeaponAimAmplifier MonoBehaviour

Overview

The WeaponAimAmplifier uses the camera rotation to twist and turn the held weapon, adding the illusion of weight and momentum.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Multiplier	Float	The multiplier for the resulting weapon rotation side to side.
Vertical Multiplier	Float	The multiplier for the resulting weapon rotation up and down.
Sensitivity	Float	How sensitive the sway is to camera rotation. Higher sensitivity means the sway approaches its peak with slower rotations
Damping Time	Float	Approximately the time it will take to reach the target rotation. A smaller value will reach the target faster.

See Also

[Additive Transforms](#)

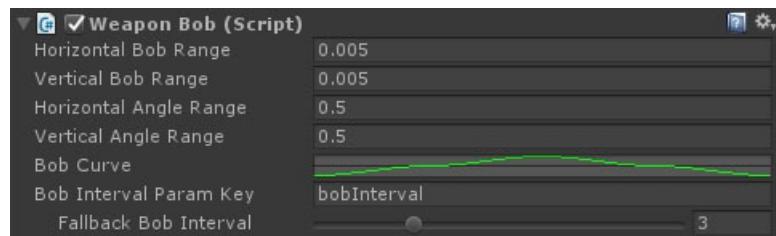
WeaponBob MonoBehaviour (Deprecated)

Overview

The WeaponBob behaviour is used to move the gun or held item in sync with character movement. This behaviour has been deprecated and replaced with the [PositionBob](#) and [RotationBob](#) systems which sync weapon and head bobbing, and allow player settings to reduce head movement.

The weapon bob can tie into the motion graph and subscribe to a [float parameter](#) to allow the motion graph elements to control when the bob occurs and the bob rate. Without this, the bob occurs whenever the character is grounded, and the bob rate is based on movement speed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Horizontal Angle Range	Float	The maximum rotation offset around the x-axis in either direction.
Vertical Angle Range	Float	The maximum rotation offset around the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the weapon bob.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle. If the parameter key above is set then this value will only be used if the parameter can not be found.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle.

See Also

[Additive Transforms](#)

[HeadBob](#)

[Unity AnimationCurve](#)

WeaponMomentumSway MonoBehaviour

Overview

The WeaponMomentumSway behaviour is an alternative to the WeaponAimAmplifier behaviour and moves instead of rotates the weapon. This makes it more suitable to weapons that are set to use the gun transform for the shooting direction as the weapon will still be pointing forwards as it moves.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Multiplier	Float	The multiplier for the position offset side to side.
Vertical Multiplier	Float	The multiplier for the position offset up and down.
Sensitivity	Float	How sensitive the sway is to camera rotation. Higher sensitivity means the sway approaches its peak with slower rotations
Damping Time	Float	Approximately the time it will take to reach the target rotation. A smaller value will reach the target faster.

See Also

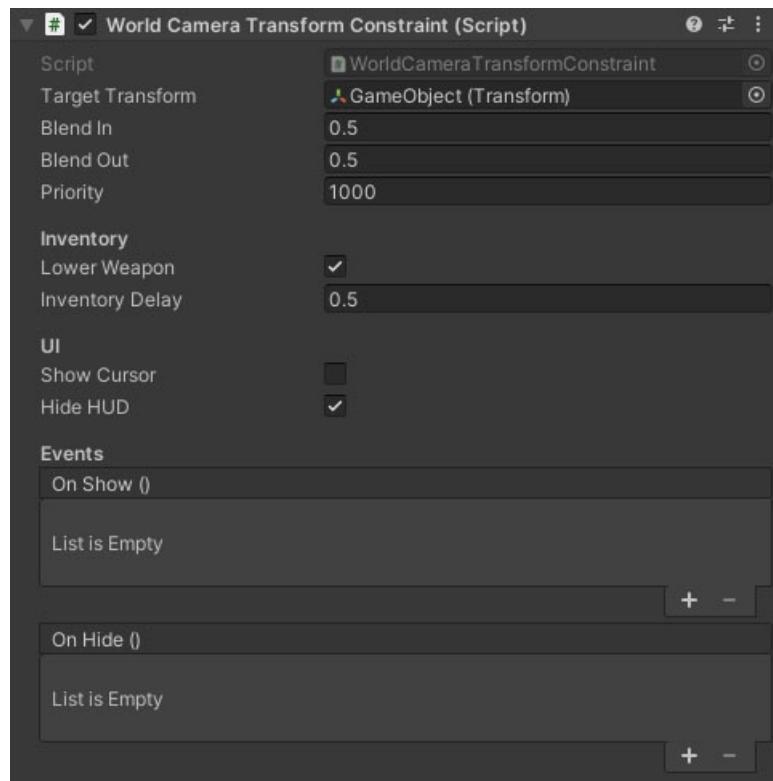
[Additive Transforms](#)

WorldCameraTransformConstraint MonoBehaviour

Overview

The WorldCameraTransformConstraint behaviour can be added to an object in the world to create a constraint target for the player character's camera. When applied using the `ApplyConstraint()` method, the player character camera will snap to the constraint position and rotation. You can use `RemoveConstraint()` or `RemoveConstraintInstant()` to remove the constraint and return the character camera back to its correct position and rotation on the character body. This requires that the player character has the [FirstPersonCameraTransformConstraints](#) behaviour attached, as with the [First Person Body](#).

Inspector



Properties

Name	Type	Description
Target Transform	Transform	The target transform of the constraint.
Blend In	Float	The time taken to blend the character camera to this constraint's target position and rotation.
Blend Out	Float	The time taken to blend the character camera out from this constraint's target position / rotation to its neutral position / rotation on the character.
Priority	Integer	Higher priority constraints will override lower priority ones.
Lower Weapon	Boolean	Should the player character's weapon be lowered while the camera constraint is active.
Inventory Delay	Float	The time taken to blend the character camera out from this constraint's target position / rotation to its neutral position / rotation on the character.

NAME	TYPE	DESCRIPTION
Show Cursor	Boolean	Should the mouse cursor be visible.
Hide HUD	Boolean	Should the in game HUD be hidden while the camera constraint is active.
On Show	UnityEvent	An event fired when the camera constraint is applied.
On Hide	UnityEvent	An event fired when the camera constraint is removed.

See Also

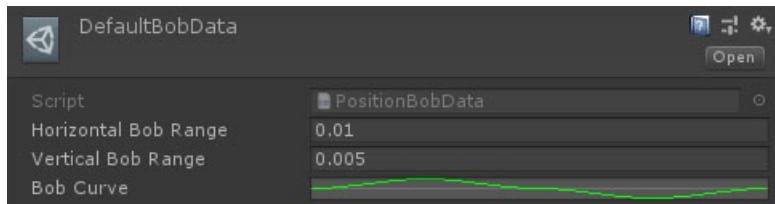
[FirstPersonCameraTransformConstraints](#)

PositionBobData ScriptableObject

Overview

The PositionBobData scriptable object stores bob information to be shared between multiple [PositionBob](#) behaviours. This allows all the bob effects on a character to sync up, and for the player to blend between item and head bob if they find head bob uncomfortable.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the bob effect.

See Also

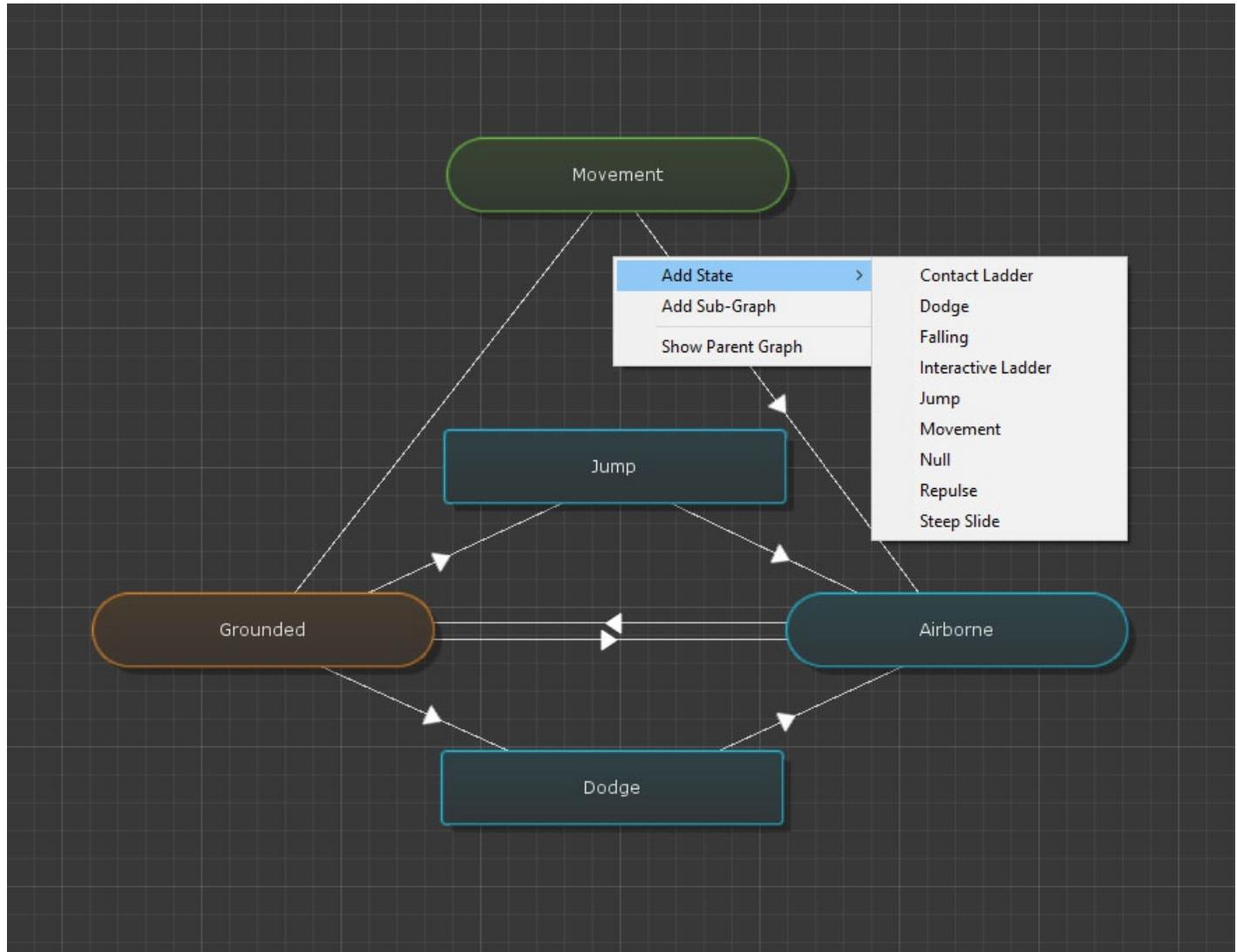
[Additive Transforms](#)

[PositionBob](#)

[Unity AnimationCurve](#)

The Motion Graph

Overview



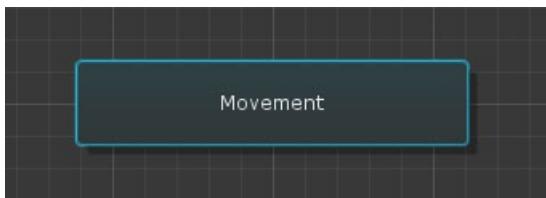
The motion graph is the core movement system of NeoFPS. It provides a powerful but flexible tool for designers to control how the FPS character moves, and how the character's movement state affects other Unity systems such as audio and animation. The motion graph means designers aren't tied to a specific style of movement, and minimises the need for complex custom code when trying to achieve a specific FPS vision (though it is also designed to be highly [extensible](#)).

The motion graph is a state machine that tracks and manages the movement state of the FPS character. It is built up of [States](#), [Connections](#), [Sub-Graphs](#), [Behaviours](#), [Parameters](#) and [Motion Data](#).

The NeoFPS movement system uses a [MotionController](#) behaviour attached to the FPS character, which drives a [NeoCharacterController](#). The [MotionController](#) has parameters to select the desired motion graph ScriptableObject, along with a [MotionControllerData ScriptableObject][7] that the graph queries for parameters such as movement speed and strafe multipliers. By separating the motion controller from the graph and data, NeoFPS allows for a huge amount of flexibility in creating separate characters with unique movement styles and traits in a single game. This can be very useful if the game design involves character classes or RPG style stats.

Graph Elements

States



States are the building blocks of the motion graph. Each state is a style of movement and has complete control of the character's [MotionController](#).

The states base their movement on the motion graph [Parameters and Motion Data](#) as well as a number of properties from the [MotionController](#) and [NeoCharacterController](#) such as ground contact and input.

For more information, see [Motion Graph States](#).

Connections

Connection

Source: Grounded Destination: Mantle

Mute:

Conditions

- = Collision Flags Condition
Include: Mask Front
- = Trigger Condition
jump: triggered
- = Climbable Condition
Output Wall Normal: wallNormal
Check Direction: Yaw Forward
Check Distance: wallCheckDistance
Wall Collision Mask: Mixed...
Max Climb Height: 2
Climb Forward: 0.25
Output Climb Height: climbHeight
- = Float Condition
climbHeight > 0.75
- = Direction Condition
Target Direction: wallNormal
Yaw Vs Horizontal > 150
- = Direction Condition
Target Direction: wallNormal
Input Vs Horizontal > 135
- = Condition Group
Condition Group: CheckVelocity

Transition On: All True

ConditionGroups

! Condition groups are evaluated using "Condition Group" conditions

Create New Condition Group

CheckVelocity

Group Name: CheckVelocity

Conditions

- = Velocity Condition
Horizontal Speed < 5
- = Direction Condition
Target Direction: wallNormal
Velocity Vs Horizontal > 135

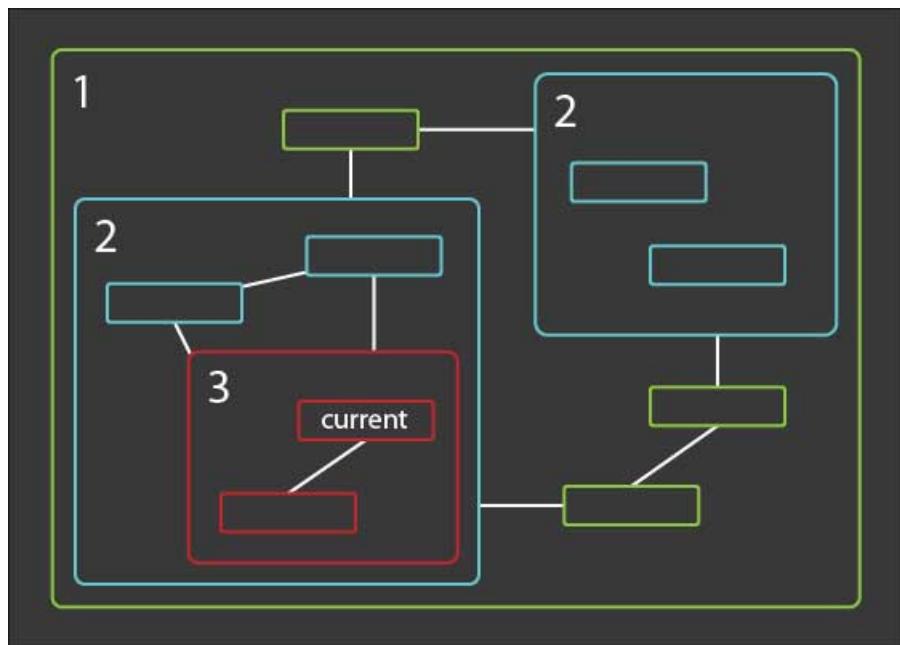
Transition On: Any True

Connections handle the relationship between movement states. Each update tick, all of the outbound connections of the current state are checked in sequence. If an outbound connection is found to be valid, then that state in turn is checked until the final state in the chain is reached. That final state becomes the new "current" state, and is then updated.

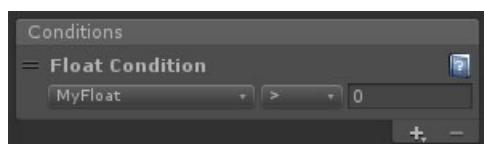
In order for a connection to be valid, the conditions attached to it must return true (all or any depending on the setting).

Connections can also be muted by checking the **Mute** checkbox above the conditions list. A muted connection will never be traversed and will show as red in the viewport. A muted connection will also be greyed out in the out-connections list of the source element.

Connections are evaluated for parent sub-graphs as well to allow for grouping logic. The connection checks are handled in order from the highest level sub-graph containing the current state, down the sub-graph hierarchy, and lastly the current state. An example of how this can be useful is switching between grounded and airborne movement. You can group all of the grounded movement states (running, walking, crouching, etc) and connections together in their own sub-graph, and group all of the airborne states and connections together in another sub-graph. Connecting these 2 sub-graphs using a ground contact check condition means that these connections are much higher priority than the connections inside the respective sub-graphs. It doesn't matter what connections exist and conditions are met for grounded movement if the character is no longer touching the ground. Without this ordering and grouping of nodes into sub-graphs, every grounded movement state would require a connection to the airborne states.



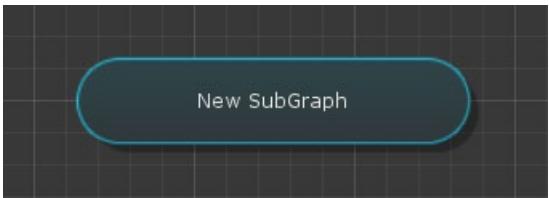
Conditions



Conditions are a one off test that returns either true or false. The tests can be based on the graph state, graph parameters, or external systems such as physics and character health. Connections can be set to traverse when **all** or **any** conditions are true. If you need some combination of all/any then you can use condition groups.

For more information, see [Motion Graph Conditions](#).

Sub-Graphs



Sub-Graphs allow states to be logically grouped together into smaller graphs. They have inbound and outbound connections just like a state and are tested before the current state in order from the graph root down to the current sub-graph and finally the current state. This enables complex logic to be designed through a simple interface. For example, most characters will have an **Airborne** and a **Grounded** sub-graph. Inside the **Grounded** sub-graph will be various states handling movement such as walking, sprinting and sliding, as well as the connections between them. If the character loses ground contact then it doesn't matter which of those states the character is in. The sub-graph connections will handle the connection into the *Airborne* sub-graph and then select the relevant airborne state.

Each sub-graph has a **default** state. On entering a sub-graph, all of the connections to child states are evaluated in sequence. If none of them are valid then the default state automatically becomes the current state.

Parameters And Data



Motion graph parameters are similar to the parameters inside the [Unity AnimatorController](#). Any number of parameters can be added to the motion graph and then referenced by key or hash. You can also store the parameter reference for quicker access.

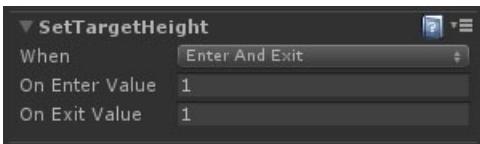
Certain parameters such as triggers and switches can be blocked, either by graph behaviours, or from outside the graph.

The motion graph also allows for an event parameter. This can be subscribed to from outside the graph and triggered by graph behaviours, allowing the graph to interact with components outside of the motion graph system.

Motion data behaves similarly but cannot be changed from outside the graph, except for using an override asset. Motion data are referenced in graph states and behaviors, and are completely optional.

For more information, see [Motion Graph Parameters And Data](#).

Behaviours



Motion graph behaviours add extra functionality to the graph. They can process logic on entering or exiting a state or sub-graph and/or when a state is updated. Certain behaviours can only be attached to states, and others can only be attached to sub-graphs, while most can be attached to both.

Motion graph behaviours allow for simple logic such as resetting or modifying parameters. They can also be complex systems on their own, such as the motion graph behaviour based [footsteps](#).

For more information, see [Motion Graph Behaviours](#).

Root Motion

For characters that have an animated humanoid body, you can also make use of root motion to drive movement using animations. To make this work you need to add a [FirstPersonBodyRootMotion](#) component to the character's body. This intercepts the root motion offsets from the animation and, instead of then moving the character model, it sends those offsets to the motion controller instead. This allows you to blend root motion in and out, and use it selectively on specific movement states.

To leverage the root motion information you can add the [SetRootMotionStrength](#) motion graph behaviour to your states or

subgraphs, or you can use the [RootMotion](#) motion graph state. The latter sends the player input to the character's body [animator](#) for use in a blend tree.

See Also

[Motion Graph Editor](#)

[Motion Graph States](#)

[Motion Graph Conditions](#)

[Motion Graph Parameters And Data](#)

[Motion Graph Behaviours](#)

[MotionController Behaviour](#)

[Unity AnimatorController](#)

[Ladders](#)

[Moving Platforms](#)

[Extending The Motion Graph](#)

NeoCharacterController

Overview

The NeoCharacterController is a replacement for the [Unity Character Controller](#) which adds additional functionality such as:

- [Moving platforms](#)
- Interpolation between fixed framerate physics movement
- Arbitrary gravity and up-vector
- Height change, including jump-crouching
- Built-in pushing of rigidbody objects
- Reaction to impacts from rigidbody objects
- Physical interaction with other character controllers
- Curve driven slope speed modifiers
- Ledge friction property to allow sliding off ledges when overhanging

The NeoCharacterController is a kinematic character controller that uses a collide and slide technique to ensure smooth movement and collisions, while preventing fast movement penetrating through thin objects.

For more information on the properties that are exposed in the inspector see the [NeoCharacterController Behaviour](#).

The other classes within NeoFPS refer to the NeoCharacterController using the `INeoCharacterController` interface. This means that the character controller can be replaced with another implementation, as long as that implements the correct interface.

Friction

The **Slope Friction** property dictates how much of the downward movement is redirected down the slope instead of cancelled out. At a friction level of 1, any vertical movement into the slope will be cancelled out completely. At a friction level of 0, the character will slide down the slope due to gravity.

The **Ledge Friction** property specifies what happens when the character is overhanging a ledge. The NeoCharacterController has basic awareness of ground contacts that lets it know when it is in contact with an edge as opposed to a flat surface. If the character centerpoint is overhanging a ledge, and the drop distance is large enough, then the character will slide off the ledge depending on this setting. It can be used to prevent the character from hovering off the edge of obstacles unrealistically. If this value is higher than the slope friction, then the slope friction will be used instead. This prevents sliding down a slope and then sticking on the bottom edge.

Rigidbody and Character Interaction

The NeoCharacterController can push dynamic rigidbodies, as well as react to impacts from rigidbodies.

The **Low Rigidbody Push Mass** is used to set a mass that the character can easily push. Any rigidbody at or below this mass will have a proportionate force exerted on it that will achieve the same effect. Above this mass, that force will drop away to zero as the rigidbody approaches the **Max Rigidbody Push Mass**. The **Rigidbody Push** property is the push power and can be experimented with to get the correct effect.

Similarly, the NeoCharacterController can also push and be pushed by other NeoCharacterController objects.

Gravity and Up-Vector

The NeoCharacterController operates using its own gravity value instead of the Unity Physics gravity. This allows for more flexibility to achieve a specific game feel. First person shooters often use a higher gravity than normal for character movement as it feels more realistic and less floaty.

The NeoCharacterController can also have its up-vector changed. This allows for features such as walking around on relatively small planets, or localised gravity zones. All features such as step height and slopes work the same as gravity and the up-vector rotate.

By default, the up-vector will be adjusted to face the opposite direction to gravity whenever the character gravity is changed. If the gravity is set to zero then the up-vector can be freely changed to allow for unlimited zero-g movement. A range of motion graph states will be added to take advantage of this in a future update.

See Also

[Moving Platforms](#)

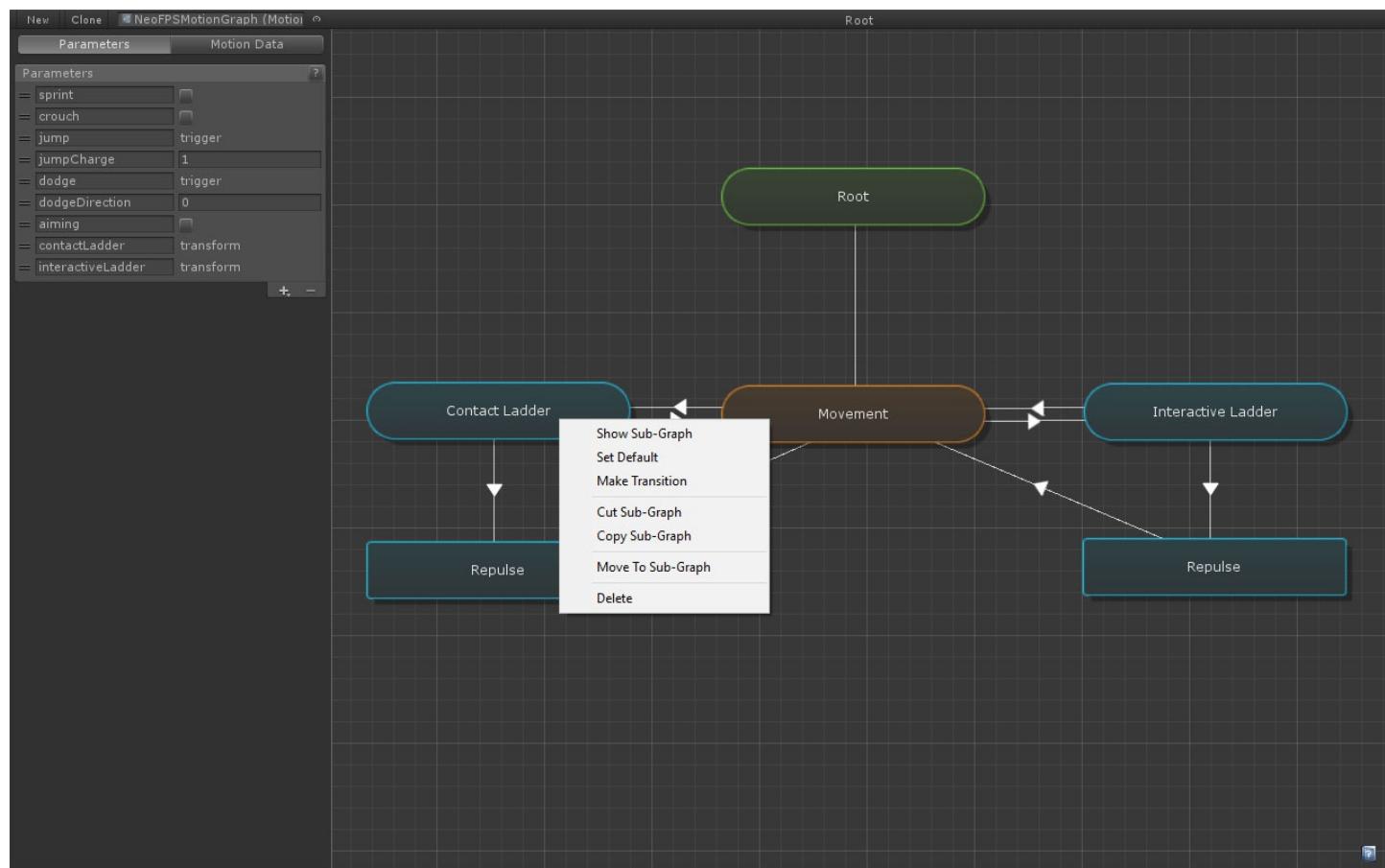
[NeoCharacterController Behaviour](#)

[Unity Character Controller](#)

The Motion Graph Editor

Overview

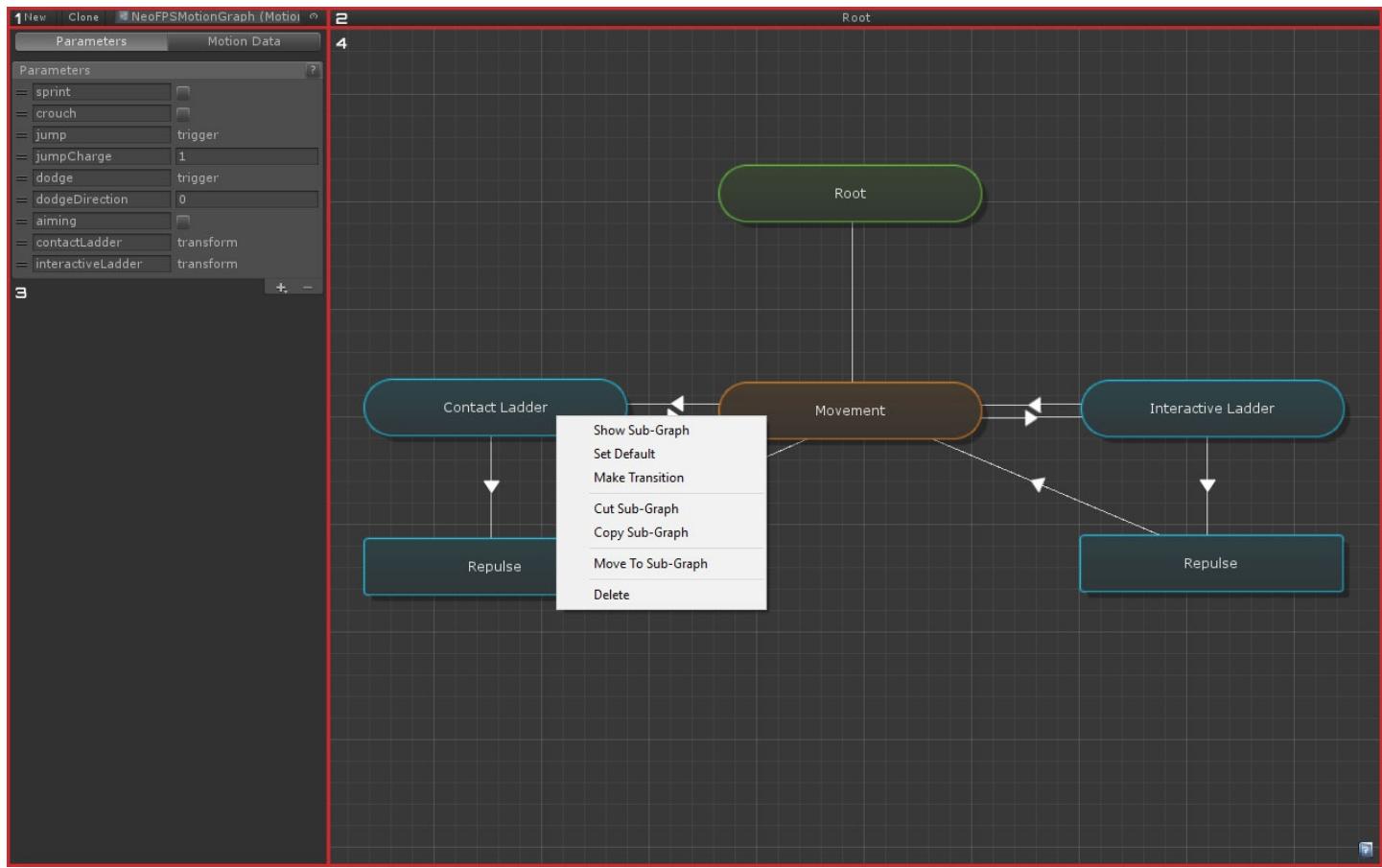
The motion graph editor is a set of tools for creating and editing NeoFPS motion graphs. It contains a viewport for navigating and visualising the graph, along with an inspector for changing the properties of the graph and its components.



You can access the motion graph editor through the Unity menu: **Window/NeoFPS/Motion Graph Editor**. You can also access the editor from the **Show Motion Graph Editor** button that appears in the inspector for a motion graph asset.

You can create a new motion graph from the motion graph editor by clicking the **New** button in the top bar. The new asset will be placed in the project root folder. You can also right-click in the project and create a new motion graph through the create menu using the following command: **Create/NeoFPS/Motion Graph**. Selecting a motion graph asset with the motion graph editor open will update the editor to show the contents of the selected graph.

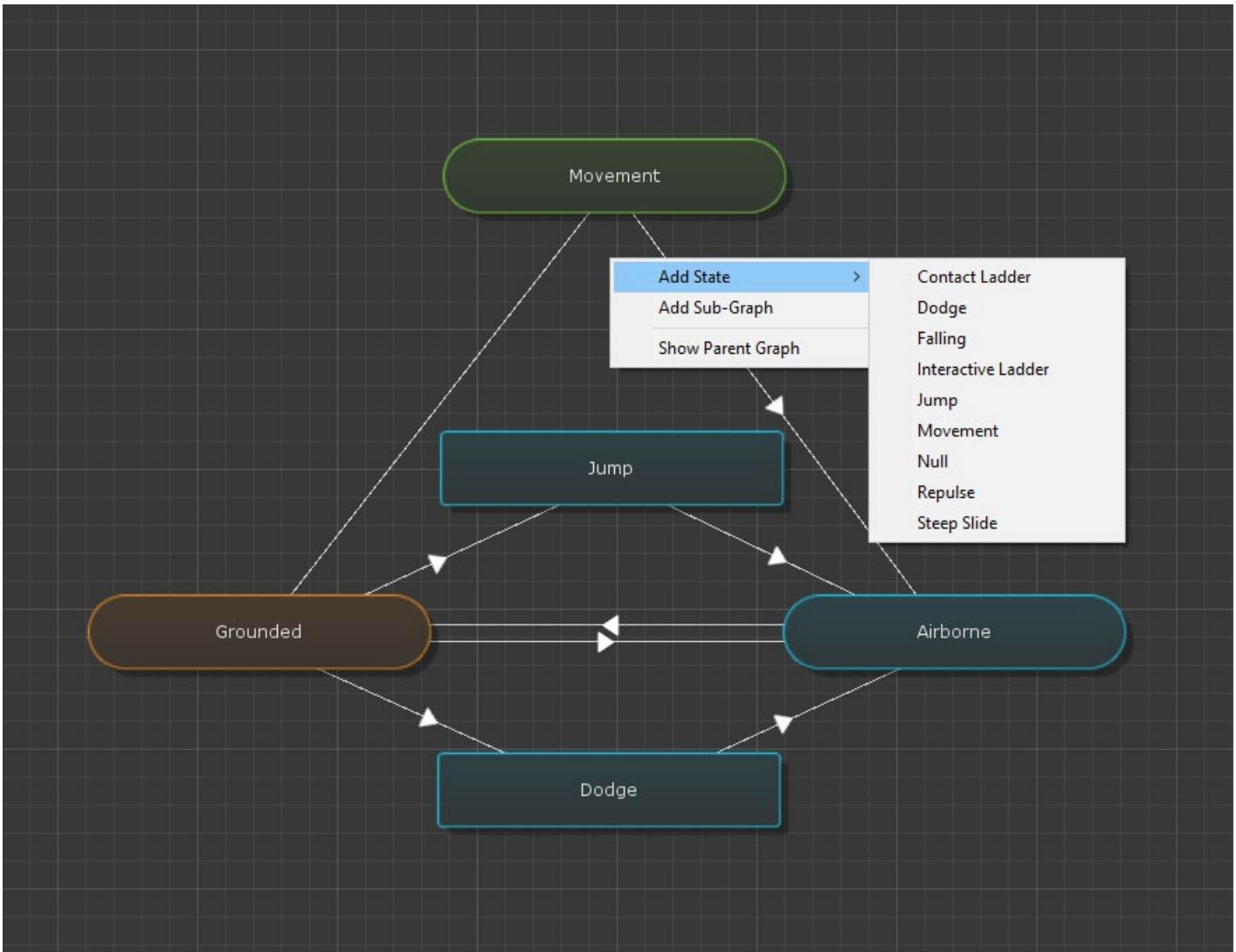
Motion Graph Editor UI



The following is a breakdown of what you will see when you open the editor:

1. The **asset** controls allow you to create and save graph assets as well as select the graph currently being edited.
2. The **breadcrumb** at the top shows the current sub-graph of the hierarchy that is currently visible in the viewport.
3. The **side bar** allows you to add and modify parameters and motion data on the graph.
4. The **viewport** shows the layout of the graph.

Motion Graph Viewport



The motion graph viewport gives a visual representation of the graph, allowing you to navigate the connections, add elements and select and modify existing elements.

To move the viewport hold the **middle mouse button** and drag the cursor. Alternatively, you can hold the **Left Alt** key and click the **left mouse button** on an empty area of the viewport and then hold and drag the mouse button to move.

To open a context menu you can right click anywhere in the viewport. This menu will contain different entries depending on what you clicked and covers actions such as navigating the graph, cut/copy/paste/deleting elements and creating transitions between elements.

Inside the viewport, the visible elements are as follows:

- The rectangular boxes are motion graph states. These handle the movement of the character and the motion controller depending on the motion controller's current state. **Left-click** a state to edit its properties in the editor inspector panel.
- The angled boxes are sub-graphs. These contain their own graph layout and can be transitioned into and out of as if they were a state. **Left-click** a sub-graph to edit its properties in the inspector panel. **Double-click** a sub-graph to replace the viewport contents with that sub-graph's contents.
- The green angled box is the sub-graph that is currently being shown in the viewport. You can **double-click** this element or use the context menus to show its parent graph in the viewport.
- The lines with arrows between different elements are the motion graph connections. These define which states and sub-graphs connect to each other and what conditions must be met to transition between them. **Left-click** the arrow on a connection to edit its properties in the inspector panel.

You can also group select states and sub-graphs in order to perform group operations. To drag-select elements, **Left-click** in an empty area of the viewport and, holding the mouse button down, drag across the desired states and sub-graphs. Releasing the mouse button will select the states and sub-graphs within the area. Connections can not be group selected, but any group

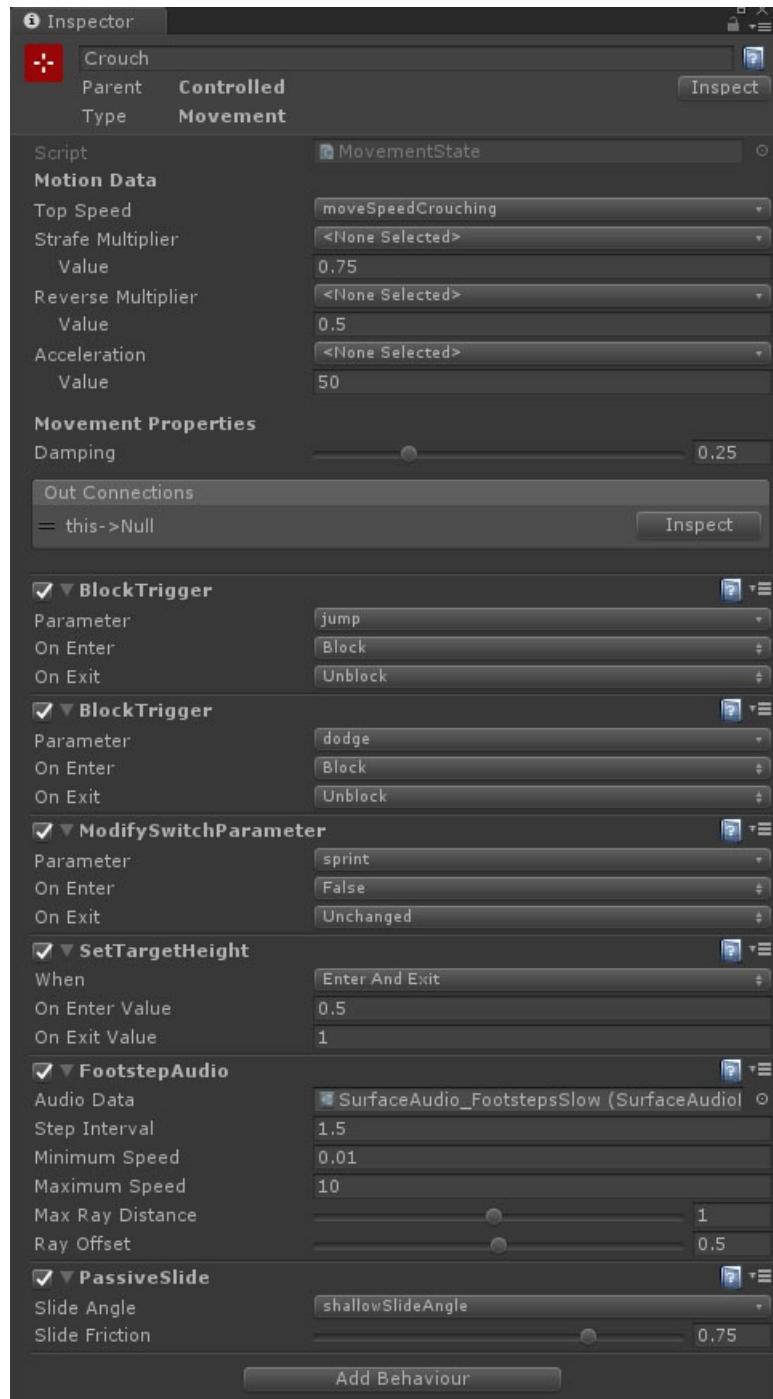
operations will affect the connections into and out of the modified elements. You can also **Control-Left-Click** or **Command-Left-Click** on an element to add it to the group, or **Alt-Left-Click** to remove it from the group.

Any selected states or sub-graphs will be highlighted in the viewport.

Inspecting Motion Graph Elements

Selecting an element in the viewport will show it in the Unity editor inspector.

Connectables (States an Sub-graphs)



The connectable inspector is used for both states and sub-graph elements due to the similarities between them.

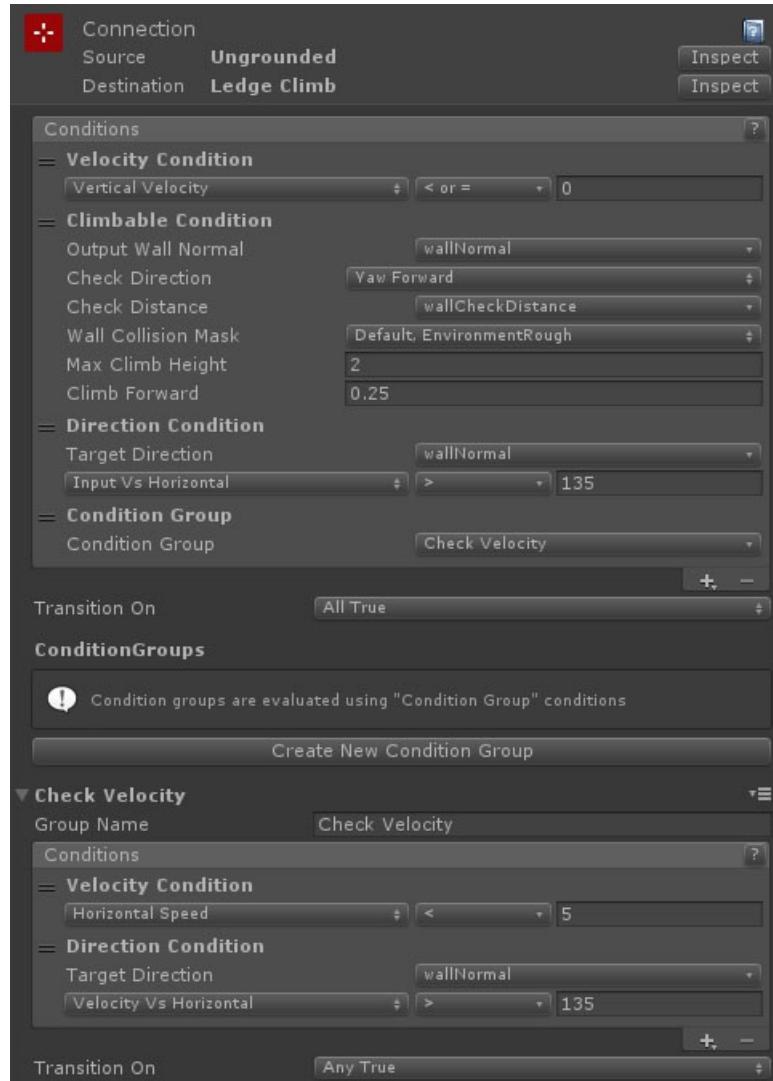
The header of the inspector allows you to change the element's name, see (and inspect) its parent sub-graph, and in the case of states, see its type.

Straight after the header are the connectable properties. States can have any number of properties, and in some cases might have complex editor logic such as you would expect from a MonoBehaviour or ScriptableObject.

After the connectable properties is an **Out Connections** array that shows all of the connections out of the inspected element. Each entry in the array shows the destination, along with an **Inspect** button. Clicking this will select the connection element and show its properties in the inspector for editing.

Both states and sub-graphs can also have a number of behaviours attached to them. These are listed after the out connections. Each behaviour has a foldout title bar that can be used to show or hide its properties. They also have a link to the relevant manual reference for that behaviour, and a dropdown context menu with options for changing order, copy/paste and removal. You can add behaviours to the state by clicking the **Add Behaviour** dropdown button and selecting a behaviour. For more information on behaviours, see [Motion Graph Behaviours](#).

Connections

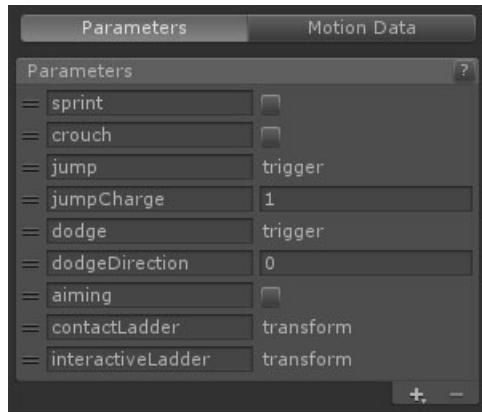


The header of the connection inspector shows the source and destination elements and allows you to quickly inspect either one.

The **Conditions** array shows all the conditions that decide if the connection is valid. For more information on conditions, see [Motion Graph Conditions](#). You can add a new condition by clicking the + button at the bottom of the array and selecting the relevant condition from the dropdown.

The **Transition On** dropdown specifies if all of the conditions must be met for control to be transferred, or if any one of the conditions must be met.

Parameters and Data



The side-bar of the editor has 2 tabs that show all of the parameters attached to the graph, and all of the motion data attached to the graph. Parameters and data can be referenced from any state, behaviour or condition in the graph. Scripts can also get a reference to any of the attached parameters from outside the graph, creating a flexible system for communication back and forth between the graph and other systems.

For more information, see [Motion Graph Parameters And Data](#).

See Also

[The Motion Graph](#)

[Motion Graph States](#)

[Motion Graph Conditions](#)

[Motion Graph Parameters And Data](#)

[Motion Graph Behaviours](#)

[Unity AnimatorController](#)

Motion Graph States

Overview

States are the building blocks of the motion graph. Each state is a style of movement. It takes input and basic physics information from the [MotionController](#) and returns a move vector each tick.

Included States

Ground Movement States

NAME	DESCRIPTION
AnimCurveDash	A basic grounded movement with a dash layered on top. The dash speed is controlled with an animation curve.
Dash	Surge in a direction based on yaw or movement direction.
Movement	Standard ground based movements such as walking or running.
Ski	Maintain ground speed and allow steering based on speed (slower speed is tighter steering).
Steep Slide	Sliding on a steep slope, with some control over direction.

Airborne States

NAME	DESCRIPTION
Controlled Jetpack	Applies an upward force, with some air control.
Falling	Air control based movement with gravity.
Fly	No-clip style movement that can move in all axes.
Jetpack	Applies an upward force and maintains horizontal velocity.

Instantaneous States

NAME	DESCRIPTION
Boost Pad	Instantly sets velocity to match a vector parameter.
Dodge	As jump, but lower and further.
Impulse	Instantly sets velocity based on a vector with options for space and ground alignment.
Jump	An instant upward jump. Completes in one frame.
JumpDirection	A jump, but the direction is rotated from the vertical based on input.
JumpDirectionV2	A jump with a fixed vertical speed and a horizontal speed based on input.
Push Off	Applies velocity based on a vector parameter, with optional upward force.

NAME	DESCRIPTION
Repulse	An instant impulse away from a transform in the scene.

Misc States

NAME	DESCRIPTION
AnimCurveDirectionalDash	A dash movement in the direction of a vector parameter. The dash speed is controlled with an animation curve.
Constant Move	Accelerates to a fixed velocity and maintains it.
Maintain Velocity	Simply returns the velocity from the previous frame.
Match Transform	Attaches the character (position with or without yaw and up-vector) to a transform in the scene.
Move To Point	Moves from the starting point to the position set in a vector parameter.
Null	An empty state used to branch to others.
Root Motion	Sends input to the character's animator, and then uses root motion to move.

Wall Movement States

NAME	DESCRIPTION
AnimCurveWallDash	A wall run movement with a dash layered on top. The dash speed is controlled with an animation curve.
Mantle	Climb onto a ledge.
Vertical Wall Run	Run directly up a wall.
Wall Run	Run along a wall surface.

Swimming States

NAME	DESCRIPTION
Swim Smooth Surface	Swimming state that tries to stick to a water zone's surface. Smooth movement.
Swim Smooth Underwater	Swimming state that can move in all axes. Smooth movement.
Swim Stroke Surface	Swimming state that tries to stick to a water zone's surface. Moves in pulses.
Swim Stroke Underwater	Swimming state that can move in all axes. Moves in pulses.
Swim Submerge	Used to transition from surface to underwater states while taking the movement of the surface into account.
Wading	A variation of the movement state with speed dependent on the amount of a character's body below the water surface.

Ladder States

NAME	DESCRIPTION
Contact Ladder	A basic climb state when touching a contact ladder.
Interactive Ladder	A climb state constrained to an interactive ladder.

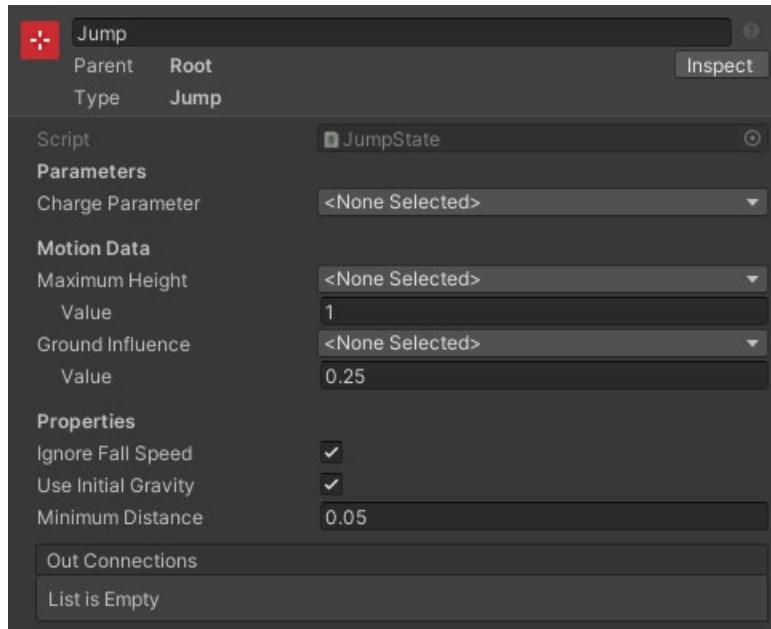
In The Viewport



Context Menu Options

NAME	DESCRIPTION
Set Default	When entering a subgraph, the motion graph checks the connections out of the sub-graph to its child nodes in sequence. If none of the connections are valid then the default state or sub-graph will be chosen as a fallback.
Make Transition	Starts a connection out of this node in the viewport. Clicking on another node will form a connection from this node to the clicked node. Clicking an empty area of the viewport will cancel the connection.
Move To Sub-Graph	This will replace the state with a sub-graph with the same name, and then make this state a child of the new sub-graph.
Cut State	Removes the state from the graph and adds it to the clipboard for pasting later. Any inbound and outbound connections will be removed. Right-click on an empty area of viewport to paste.
Copy State	Copy the state to the clipboard. Right-click on an empty area of viewport to paste.
Delete	Remove the state from the graph entirely, along with any inbound and outbound connections and attached behaviours.

In The Inspector



Each state will have its own properties visible before the **Out Connections** array. This array lists the connections from this state to other connectables, allowing you to jump to inspecting each of those targets. The **Add Behaviour** button allows you to add motion graph behaviours to the state which will be evaluated while the state is active.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

Motion Graph Behaviours

Overview

Motion graph behaviours add extra functionality to the graph. They can process logic on entering or exiting a state or sub-graph and/or when a state is updated. Certain behaviours can only be attached to states, and others can only be attached to sub-graphs, while most can be attached to both.

Motion graph behaviours allow for simple logic such as resetting or modifying parameters. They can also be complex systems on their own, such as the motion graph behaviour based [footsteps](#).

Motion graph behaviours update at the same rate as the motion controller - in FixedUpdate. This should be kept in mind if creating new behaviours for your project.

Included Behaviours

Character Behaviours

Character behaviours act on components on the character object.

These include:

- [BodyTiltBehaviour](#)
- [DrainStaminaBehaviour](#)
- [FootIkBehaviour](#)
- [ImpactDamageBehaviour](#)
- [LockInventorySelectionBehaviour](#)
- [ModifyStaminaBehaviour](#)
- [SetSteeringBehaviour](#)
- [SetWieldableStanceBehaviour](#)
- [SpineAimBehaviour](#)
- [TrackStepsBehaviour](#)
- [UnlockInventorySelectionBehaviour](#)

Camera Behaviours

Camera behaviours apply effects or constraints to the character camera

These include:

- [CameraJiggleSpringBehaviour](#)
- [CameraKickSpringBehaviour](#)
- [CameraPulseFovBehaviour](#)
- [CameraShakeBehaviour](#)
- [ConstrainCameraYawBehaviour](#)
- [ConstrainCameraPitchBehaviour](#)

Graph Parameter Behaviours

Graph parameter behaviours act on [motion graph parameters](#).

These include:

- [ClampFloatParameterBehaviour](#)
- [ClampIntParameterBehaviour](#)
- [ModifyIntParameterBehaviour](#)
- [ModifyFloatParameterBehaviour](#)

- [ModifySwitchParameterBehaviour](#)
- [ModifyTriggerParameterBehaviour](#)
- [ModifyTransformParameterBehaviour](#)
- [ModifyVectorParameterBehaviour](#)
- [BlockSwitchParameterBehaviour](#)
- [BlockTriggerParameterBehaviour](#)
- [InvokeEventBehaviour](#)
- [TimeOpsBehaviour](#)

Animation Behaviours

Animation behaviours interact with the character's animator component by setting animator controller parameters,

These include:

- [AnimatorGroundSlopeBehaviour](#)
- [AnimatorInputVectorBehaviour](#)
- [AnimatorSpeedBehaviour](#)
- [AnimatorVelocityBehaviour](#)
- [SetAnimatorBoolBehaviour](#)
- [SetAnimatorFloatBehaviour](#)
- [SetAnimatorIntBehaviour](#)
- [SetAnimatorTriggerBehaviour](#)
- [SetRootMotionStrengthBehaviour](#)

Audio Behaviours

Audio behaviours trigger audio in the scene and interact with the various [audio systems](#).

These include:

- [FootstepAudioBehaviour](#)
- [SlidingAudioBehaviour](#)
- [LadderAudioBehaviour](#)
- [LoopingAudioBehaviour](#)
- [PlayCharacterAudioBehaviour](#)
- [PlayAudioClipBehaviour](#)
- [SetAudioEffectStrengthBehaviour](#)
- [SurfaceAudioBehaviour](#)
- [SurfaceFootstepAudioBehaviour](#)
- [SwimStrokeAudioBehaviour](#)

Physics Behaviours

Physics behaviours modify the character physics.

These include:

- [AddForceBehaviour](#)
- [AlignWithVelocityBehaviour](#)
- [ConstrainCharacterHeadingBehaviour](#)
- [DisableColliderBehaviour](#)
- [ModifyCharacterVelocityBehaviour](#)
- [PassiveSlideBehaviour](#)
- [SetTargetHeightBehaviour](#)

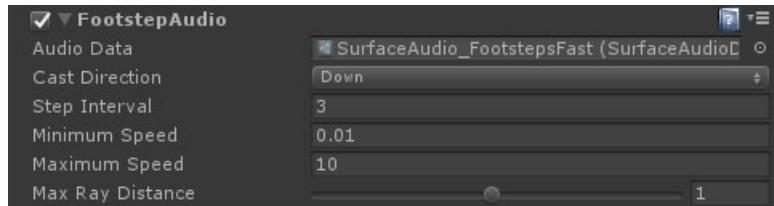
Miscellaneous Behaviours

Miscellaneous behaviours don't fall into the above categories.

These include:

- [SetTimeScaleBehaviour](#)

In The Inspector



Editing behaviours is very straightforward. Each one will have its own unique properties, along with the following controls as highlighted in the above image:

1. A checkmark to enable or disable the behaviour. Disabled behaviours will not be evaluated until enabled.
2. The type of the behaviour. Click this to collapse or expand the behaviour and its properties.
3. A link to the manual reference for the behaviour.
4. A dropdown of actions for the behaviour including changing the order on the state or sub-graph, copy / pasting properties and removing.

See Also

[Motion Graph Parameters And Data](#)

Motion Graph Conditions

Overview



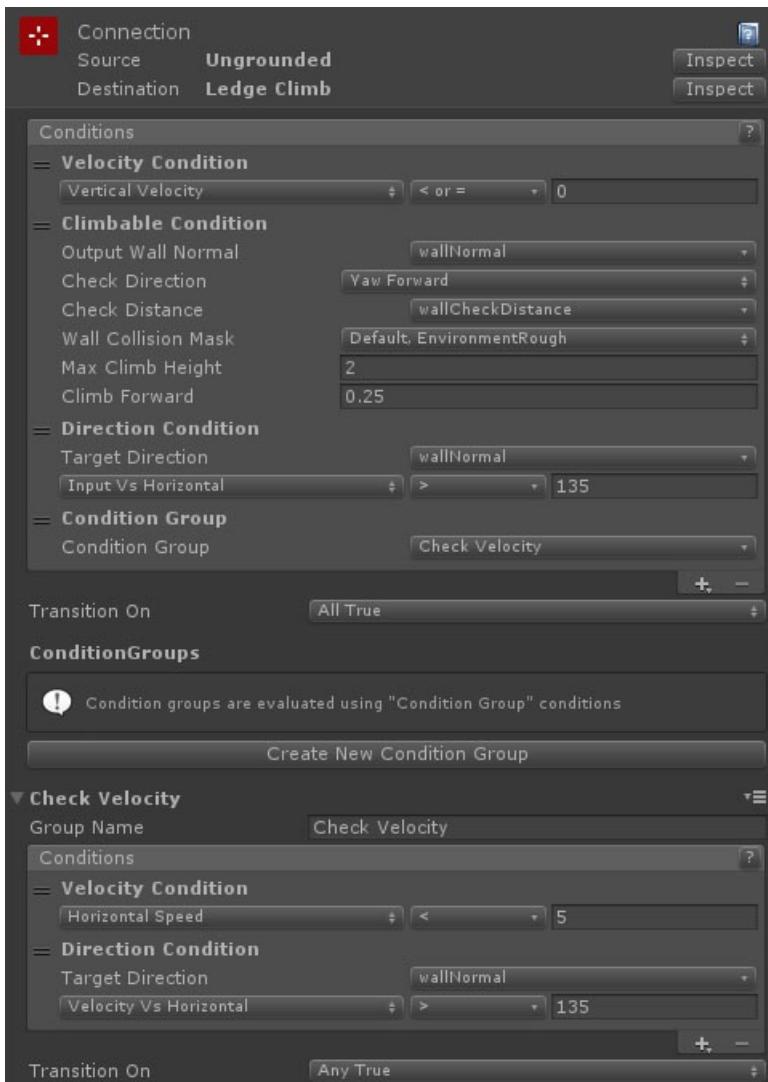
Conditions are simple true/false checks that are used to specify when the motion graph changes its active state. Conditions can be far reaching, checking against the graph, the controller, the wider character, or any external system. Connections can transition when a single condition is met or when all conditions in a connection are met.

A connection can also be muted, which means its conditions will be ignored and it will never be traversed. A muted connection will appear red in the motion graph editor viewport.

Condition Groups

Condition groups are used to evaluate conditions together and allow for more complex rules than just "any" and "all".

For example, when swimming on the surface of a water zone, you might transition to the underwater state if you **either** press crouch **or** look down **and** press forwards. In this situation you would set the main connection to transition on **any** of: crouch is pressed or the *Move Down* condition group is true. The *Move Down* condition group will be set to be true if **all** of: the character is looking down and the player is pressing forward.



The above connection is used for checking if a character can climb a wall. It is set up with a condition group that checks if either the horizontal character speed is low or the direction of movement is directly into the wall (within a certain angle range).

The results of a condition group are recorded when evaluated (reset each frame). This prevents infinite loops where 2 condition

groups reference each other.

Conditions

The following conditions are available:

Graph Conditions

These are conditions that check against the graph state.

NAME	DESCRIPTION
CompletedCondition	Checks if the state it is connected from has its complete flag set.
ConditionGroupCondition	Used to evaluate multiple conditions together (see above).
DebugCondition	Used to debug connections by specifying the value and outputting a message to the debug log.
Elapsed Time	Checks if the state it is connected from has been active for a set period.

Parameter Conditions

These are conditions that check against the [parameters][3] attached to the graph.

NAME	DESCRIPTION
CompareFloatsCondition	Compares two float parameters attached to the graph.
CompareIntsCondition	Compares two int parameters attached to the graph.
CompareSwitchesCondition	Compares two switch parameters attached to the graph.
CompareTime	Compares the current time with the time stored in a float parameter attached to the graph.
FloatCondition	Compares a float parameter attached to the graph to a specific value.
IntCondition	Compares an int parameter attached to the graph to a specific value.
SwitchCondition	Compares a switch parameter attached to the graph to a specific value.
TransformCondition	Compares a transform parameter attached to the graph to a specific value.
TriggerCondition	Compares if a trigger parameter attached to the graph has been triggered.
VectorCondition	Compares a vector parameter or its components attached to the graph to a specific value.
[VectorTypeCondition] [param11]	Checks if a vector parameter attached to the graph matches a certain type of vector such as zero length or wall vector.

Physics Conditions

These conditions use the physics system to check for collisions and contacts.

NAME	DESCRIPTION
CapsuleCast	Casts the character capsule to check for a collision.
CapsuleCast (Enhanced)	Casts the character capsule to check for a collision and can store the results in parameters.
CapsuleLookahead	Casts the character capsule based on its movement.
CapsuleLookahead (Enhanced)	Casts the character capsule based on its movement and can store the results in parameters.
Climbable	Checks if a wall has an unobstructed climbable ledge within a certain height.
RayCast	Performs a ray cast from a point on the character.
RayCast (Enhanced)	Performs a ray cast from a point on the character and can store the results in parameters.
RayLookahead	Performs a ray cast based on the movement of the character.
RayLookahead (Enhanced)	Performs a ray cast based on the movement of the character and can store the results in parameters.
SphereCast	Performs a sphere cast from a point on the character.
SphereCast (Enhanced)	Performs a sphere cast from a point on the character and can store the results in parameters.
SphereLookahead	Performs a sphere cast based on the movement of the character.
SphereLookahead (Enhanced)	Performs a sphere cast based on the movement of the character and can store the results in parameters.

Character Conditions

These conditions interface with the character that is controlled by the graph.

NAME	DESCRIPTION
AirTime	Checks how long the character has been ungrounded.
CharacterHeight	Checks the character capsule height or height multiplier (from standing).
CollisionFlags	Checks the collision flags generated on the last movement frame.
Direction	Checks the character's aim, input or movement direction.
GroundContact	Checks if the character is touching the ground.
GroundNormal	Tests against the normal vector of the ground contact.
GroundSurfaceNormal	Tests against the normal vector of the ground surface at the contact point. If the contact is an edge, this will be the normal of the top face connected to that edge.
HeightRestriction	Checks if the character can reach a specific height (eg trying to stand while crouched in an air vent).
InputVector	Checks against the input vector provided to the motion controller.

NAME	DESCRIPTION
InventoryItem	Checks the number of inventory items with a specific ID in the character's inventory against a certain value.
Pitch	Checks the character's aim pitch.
ScriptedComponent	Paired with a component attached to the character that implements the IScriptedComponentCondition interface.
Velocity	Checks the character's velocity in various directions.
Water	Checks the character's position relative to a water zone.
Wieldable Selected	Checks if the character has a specific wieldable inventory item and whether they have it selected.

See Also

[The Motion Graph](#)

[Motion Graph Parameters And Data](#)

Motion Graph Parameters And Data

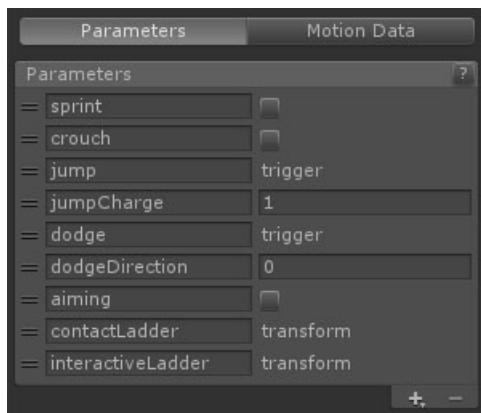
Overview

The Motion Graph allows you to add parameter and data entries to the graph.

Parameters are accessible from elements inside the graph and also externally via the get and set parameters outlined below. This enables various systems and your own scripts to feed data to the motion graph that it can use to drive and influence the character movement.

Motion data are used inside the graph, but cannot be changed directly from outside. These values are used for things like speeds that are considered more constant than parameters. You can, however, create override assets which can be attached to the graph at runtime and then

Parameters



Parameters are accessible through the motion graph editor by clicking the **Parameters** tab in the motion graph editor side-bar.

To add a new parameter click the + button on the list and select the type from the dropdown.

To remove a parameter select the relevant entry and click the - button on the list.

To edit a parameter use the fields in the relevant entry. The text field on the left is the desired name for the parameter. This should be unique to this parameter or only the first entry in the list with that name will be accessible. On the right side of the name is the default starting value. This is the value the parameter will have on initialisation and whenever it is reset.

The following parameter types are available:

NAME	DESCRIPTION
Integer	The integer parameter is a decimal value.
Float	The float parameter is a floating point value.
Switch	The switch parameter is a boolean value. It can be toggled and blocked.
Trigger	The trigger parameter is a one off trigger. It will be reset when it is consumed in the graph. It can also be blocked.
Transform	The transform parameter is a reference to a Transform component.
Vector	The vector parameter is a Vector3 value.
Event	The event parameter is a C# event that can be invoked from inside the graph and subscribed to from outside.

For parameters that can be blocked, they will always return their default value as long as there are blockers. Use `SwitchParameter.AddBlocker()` to add a blocker to a switch parameter, and `SwitchParameter.RemoveBlocker()` to remove a blocker. The parameter will be blocked when the number of blockers is not 0. Be careful to remove any blockers you add when no longer needed.



Parameters are referenced from within the graph using dropdowns. The user can choose to select **None**, **Create New** (another way to add a new parameter to the graph), or any of the relevant parameters attached to the graph.

Events

The event parameter is a useful parameter for interacting with external code from within a motion graph. They can be subscribed and unsubscribed by using the `EventParameter.AddListener (UnityAction listener)` and `EventParameter.RemoveListener (UnityAction listener)` methods.

You can invoke an event parameter from your own code using `EventParameter.Invoke ()` or using a [InvokeEventBehaviour](#) motion graph behaviour.

Motion Data



Motion data are accessible through the motion graph editor by clicking the **Motion Data** tab in the motion graph editor side-bar.

To add a new data entry click the + button on the list and select the type from the dropdown.

To remove a data entry select the relevant entry and click the - button on the list.

The **Create Override Asset** button will create a new [MotionGraphDataOverrideAsset](#) in the same folder as the edited motion graph and with the same name.



Various properties on graph states or behaviours can be assigned a motion data value using a dropdown as above. The user can choose to select **None**, **Create New** (another way to add a new motion data entry to the graph), or any of the relevant motion data entries attached to the graph. If the dropdown is set to **None** then a separate **Value** field will be visible below. In simple cases you can use this value directly, but setting the property to a motion data entry allows you to share a single data entry across multiple properties, and also to override from an asset later.

Accessing Parameters

Parameters are accessed from the motion graph root object. They can be set directly with the following methods:

```
int MotionGraphRoot.GetInt (string parameterName);
int MotionGraphRoot.GetInt (int hash);
void MotionGraphRoot.SetInt (string parameterName, int value);
void MotionGraphRoot.SetInt (int hash, int value);

float MotionGraphRoot.GetFloat (string parameterName);
float MotionGraphRoot.GetFloat (int hash);
void MotionGraphRoot.SetFloat (string parameterName, float value);
void MotionGraphRoot.SetFloat (int hash, float value);

bool MotionGraphRoot.GetSwitch (string parameterName);
bool MotionGraphRoot.GetSwitch (int hash);
void MotionGraphRoot.SetSwitch (string parameterName, bool value);
void MotionGraphRoot.SetSwitch (int hash, bool value);

Transform MotionGraphRoot.GetTransform (string parameterName);
Transform MotionGraphRoot.GetTransform (int hash);
void MotionGraphRoot.SetTransform (string parameterName, Transform value);
void MotionGraphRoot.SetTransform (int hash, Transform value);

void MotionGraphRoot.SetTrigger (string parameterName);
void MotionGraphRoot.SetTrigger (int hash);

void MotionGraphRoot.AddEventListener (string parameterName, UnityAction listener);
void MotionGraphRoot.AddEventListener (int hash, UnityAction listener);
void MotionGraphRoot.RemoveEventListener (string parameterName, UnityAction listener);
void MotionGraphRoot.RemoveEventListener (int hash, UnityAction listener);
```

Passing a string parameter for parameterName is more convenient, but performance can be increased by storing a hash of the string and using that instead. You can generate a hash using Unity's `Animator.StringToHash (string input)` method. This is similar to the way you would access a parameter in an animator controller graph.

If you plan to access a parameter multiple times then the most efficient way to do so is to store a reference to the parameter itself. You can do this using the following methods:

```
IntParameter MotionGraphRoot.GetIntParameter (string parameterName);
IntParameter MotionGraphRoot.GetIntParameter (int hash);

FloatParameter MotionGraphRoot.GetFloatParameter (string parameterName);
FloatParameter MotionGraphRoot.GetFloatParameter (int hash);

SwitchParameter MotionGraphRoot.GetSwitchParameter (string parameterName);
SwitchParameter MotionGraphRoot.GetSwitchParameter (int hash);

TransformParameter MotionGraphRoot.GetTransformParameter (string parameterName);
TransformParameter MotionGraphRoot.GetTransformParameter (int hash);

TriggerParameter MotionGraphRoot.GetTriggerParameter (string parameterName);
TriggerParameter MotionGraphRoot.GetTriggerParameter (int hash);

VectorParameter MotionGraphRoot.GetVectorParameter (string parameterName);
VectorParameter MotionGraphRoot.GetVectorParameter (int hash);

EventParameter MotionGraphRoot.GetEventParameter (string parameterName);
EventParameter MotionGraphRoot.GetEventParameter (int hash);
```

This is also the way that the included motion graph elements reference parameters. If you plan to expand on the motion graph with your own custom elements then there are a number of GUI helpers available that simplify adding custom parameter and data selectors to inspectors. For more information see [Extending the Motion Graph](#).

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Extending the Motion Graph](#)

Motion Graph Ladders

Overview

Ladders have been an ubiquitous part of the first person shooter genre, with a huge range of implementations. Many FPS ladders are hacky workarounds for what is a surprisingly complex problem, and a poor implementation can often stand out.

NeoFPS attempts to set a standard for first person shooter ladders. They come in 2 flavours: contact ladders and interactive ladders, but both share the same underlying system.

Contact Ladders

Contact ladders are climbed automatically as soon as the player character touches them. They often allow the player to climb sideways across the ladder, dropping off if they pass the edge. They also allow the player to turn a full 360 degrees while on the ladder without any look constraints.

Contact ladders are often used in fast paced shooters where flow is much more important than realism.

For more information see the [ContactLadder](#) reference.

Interactive Ladders

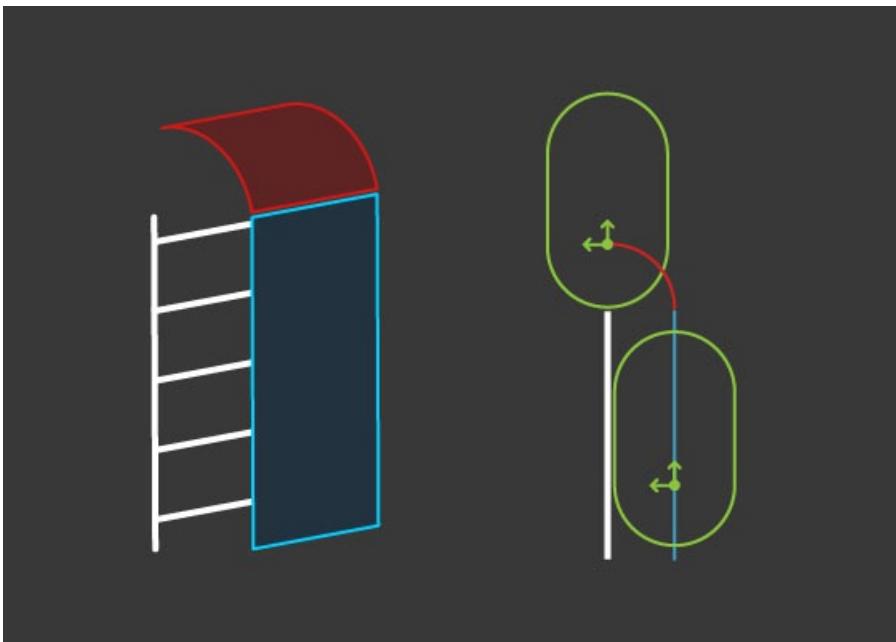
Interactive ladders require the player to look at and use the ladder and attach to it. Once attached, the character is constrained to moving up and down the ladder until they reach the ends, let go (by interacting again) or jump off.

Interactive ladders are more restrictive than contact ladders, keeping the character fixed to one axis of movement. They can also constrain the camera, blocking the camera from turning too far away from the ladder if at all. Because of these features they are often used in more realistic games or in games where pacing needs to be more controlled.

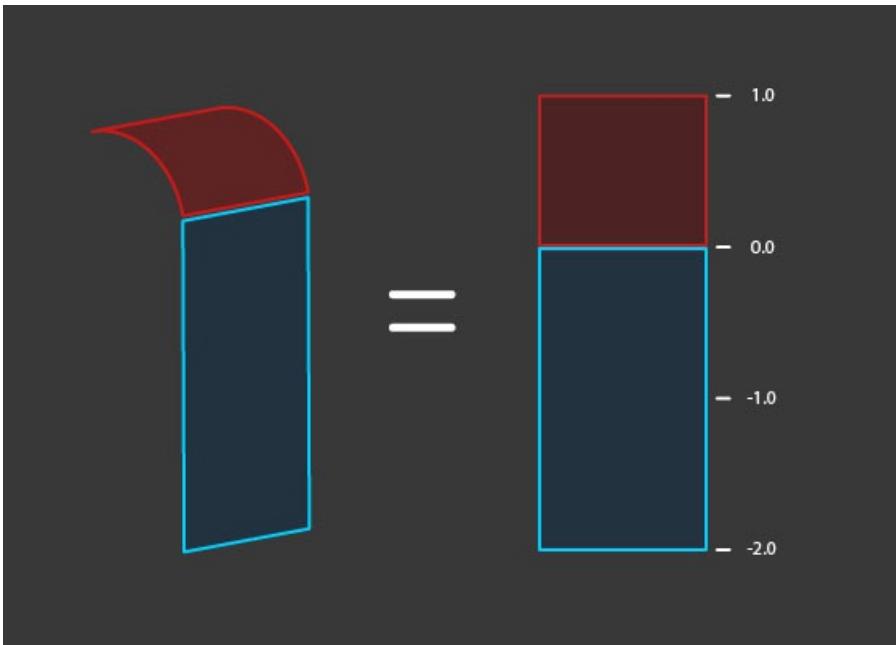
For more information see the [InteractiveLadder](#) reference.

Ladder Implementation

In NeoFPS ladders have 2 elements. The ladder surface itself, and a wrap around at the top of the ladder.



These sections are inflated from the ladder geometry by the character's radius. The system then moves a point at the bottom center of the character collider around in this "ladder space".



This system has 2 benefits: Firstly, it is easy to stick the character to the shape, making climbing consistent both part way up the ladder and also at the top of the ladder where some implementations can have difficulty registering contact. Secondly, the look direction can be judged in ladder space instead of world space.

By calculating look direction in ladder space, it means that a character at the top of a ladder and looking forwards is actually looking **up** the ladder. This makes changing climb direction based on aim much more intuitive than if the character had to look straight up in this situation. Another example is if the player approaches a ladder from the top. Walking forwards while looking at the edge will walk **down** the ladder. Looking away from the edge and slightly down will still be considered looking **up** the ladder in ladder space, so walking backwards will climb **down** the ladder as expected.

See Also

[ContactLadder](#)

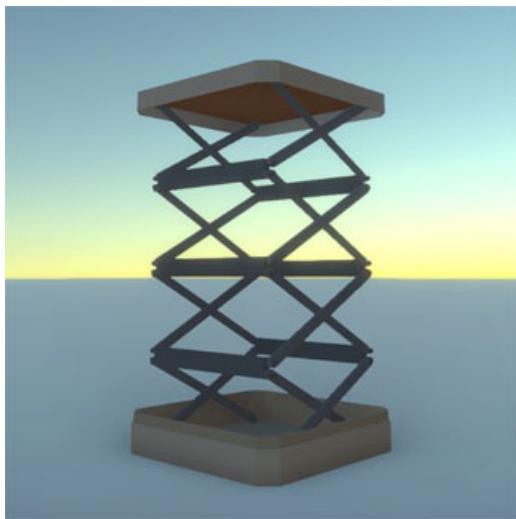
[InteractiveLadder](#)

[Unity CharacterController](#)

Motion Graph Moving Platforms

Overview

A moving platform is a kinematic or dynamic [rigidbody](#) that the player can stand on. The player will move and rotate with the moving platform.



For a demonstration of the moving platforms, you can check out the [sample scene](#).

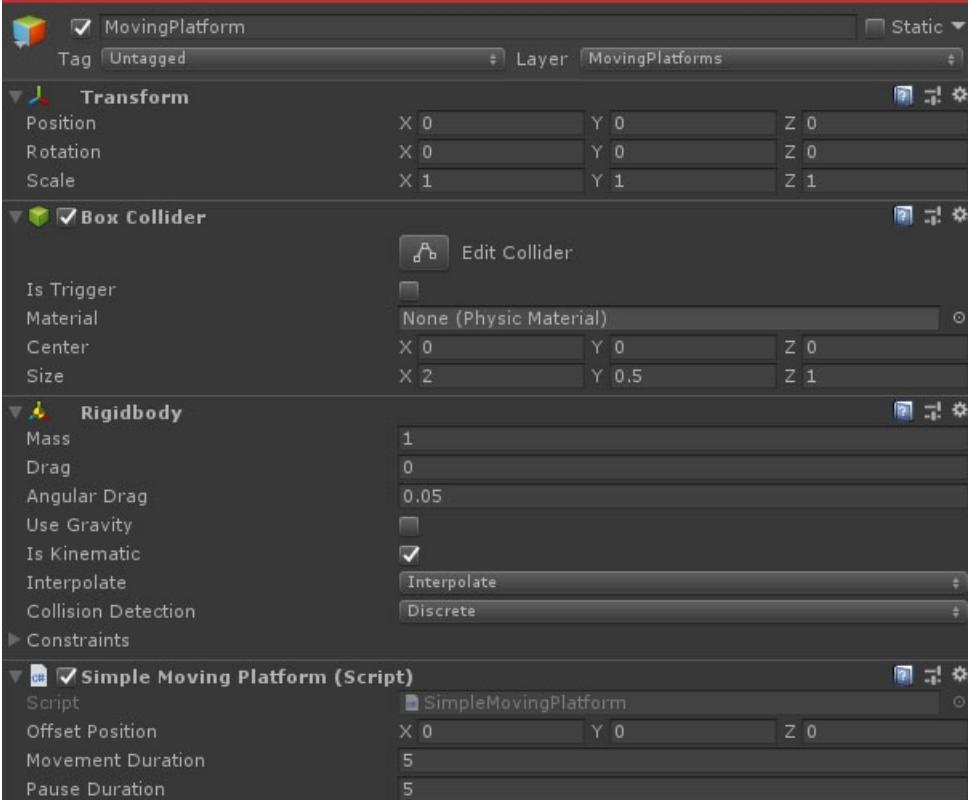
How To Set Up A Moving Platform

Creating a moving platform is relatively simple. The key things that need setting up correctly are the [rigidbody](#) and the object layers.

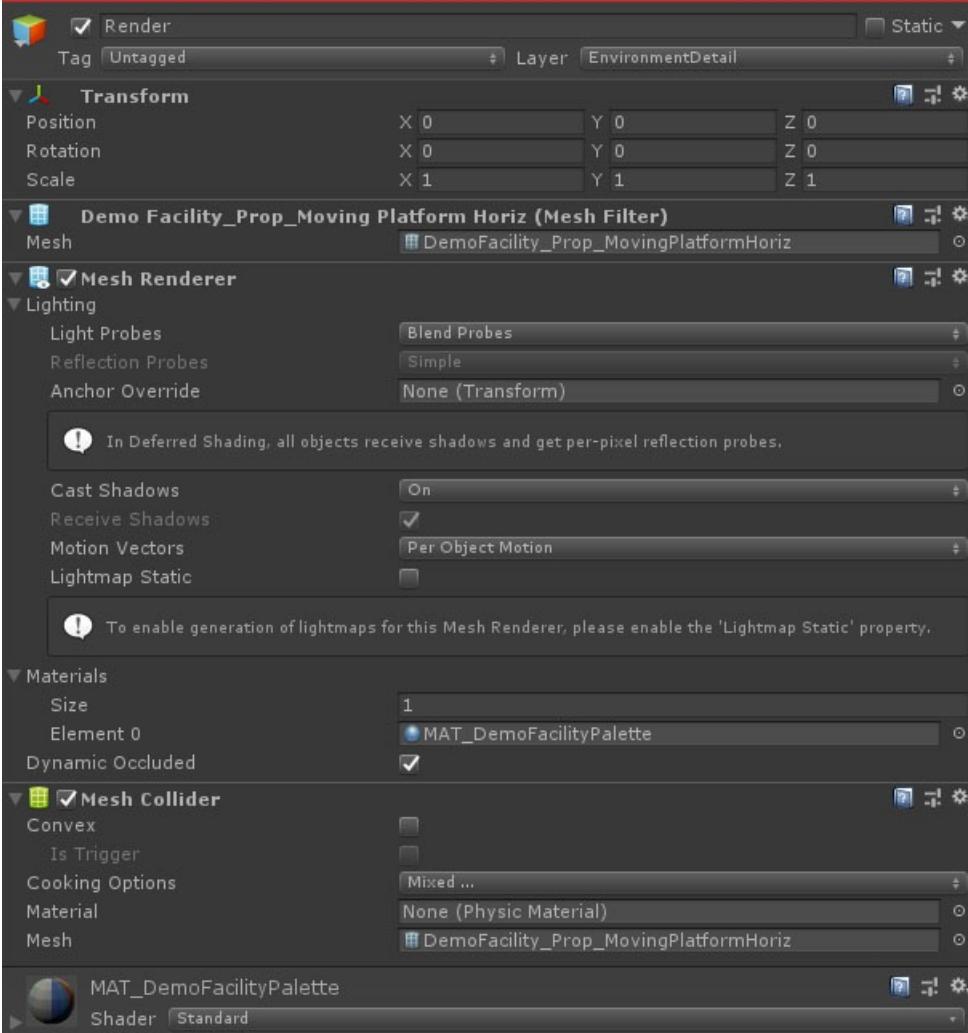
1. Scene Hierarchy

▼ MovingPlatform
 Render

2. Moving Platform Object



3. Render Geometry



In the above image there are 2 objects. The first is the physical platform that the character interacts with. This uses one or more primitive colliders to define a rough shape and should be set to use the **Moving Platform** layer. It also requires an object that derives from the `IMovingPlatform` interface or `BaseMovingPlatform` abstract class. These communicate with the [NeoCharacterController](#) to smoothly move the character. Moving platform movement should always be handled through the **Rigidbody move and rotate methods**. NeoFPS relies on rigidbody interpolation to get smooth interpolation between the fixed update frames. You should use `Rigidbody.MovePosition()` and `Rigidbody.MoveRotation()` instead of setting the position and rotation directly. If you don't do this, then the platforms will appear to stutter when you ride them.

The second object is the render geometry. This should be set to use the **EnvironmentDetail** layer and the collider should match the render geometry as closely as possible.

When a character is in contact with the moving platform it will match any change of position or rotation of the platform each frame. Any character movement will be additive on top of this.

Example Moving Platform Behaviours

NeoFPS comes with the following moving platform types already implemented, though it is easy to add more with scripting:

NAME	DESCRIPTION
SimpleMovingPlatform	The SimpleMovingPlatform behaviour is used to create a platform that moves between 2 points at set intervals.
SimpleRotatingPlatform	The SimpleRotatingPlatform behaviour is used to create a platform that rotates at set intervals.
ConstantRotatingPlatform	The ConstantRotatingPlatform behaviour is used to create a platform that constantly turns at a set rate.
DrivenMovingPlatform	The DrivenMovingPlatform behaviour is added to a rigidbody that is driven by physics or a script and turns it into a moving platform . See the above note on rigidbody interpolation.
WaypointMovingPlatform	The WaypointMovingPlatform behaviour is used to create a platform that moves between a set of waypoints in sequence or directly.
ElevatorMovingPlatform	The ElevatorMovingPlatform behaviour is used to control the movement of an elevator cab. It is attached to the kinematic rigidbody that acts as the elevator cab or platform, and moves the character smoothly with it.

See Also

[NeoCharacterController](#)

[The Motion Graph](#)

[The Motion Graph Editor](#)

Motion Graph Swimming

Overview



Swimming in NeoFPS is broken down into 3 sets of states: surface, underwater, and wading. The motion graph has the flexibility to craft a range of swimming movement styles using any of those states, while controlling things like capsule height and camera effects using the various graph behaviours.

Water Zones

Water zones are used in the scene to define a volume that the character can swim in. The [BasicWaterZone](#) behaviour is placed on an object with a trigger box collider and will notify any character that enters it.

The water zone provides the motion graph with details of the water surface height and normal, as well as the flow velocity, at a specific point.

The [BasicWaterZone](#) simply returns the top face of the attached box collider, along with a constant flow vector. More complex water zone behaviours can be created by inheriting from the [IWaterZone](#) interface. This requires the following methods:

```
Vector3 FlowAtPosition(Vector3 position);  
WaterSurfaceInfo SurfaceInfoAtPosition(Vector3 position);
```

Surface Swimming

The surface swimming states will try to keep the character's head at a certain height above the water surface. If the surface level rises too fast, or the character is already moving up/down too quickly, then the head can breach the water's surface. In this case, you can use the [Water MotionGraphCondition](#) to check the submerged depth of the character and transition into an underwater state.

The available surface swimming states are:

- [SwimSmoothSurfaceState](#) which moves smoothly along the water surface
- [SwimStrokeSurfaceState](#) which moves in pulses to simulate swimming strokes. The pulse frequency is based on movement direction.

Underwater Swimming

The underwater swimming states move in the direction of the character's aim (including pitch), while jump and crouch are used to move straight up and down.

The available underwater swimming states are:

- [SwimSmoothUnderwaterState](#) which moves smoothly
- [SwimStrokeUnderwaterState](#) which moves in pulses to simulate swimming strokes. The pulse frequency is based on movement direction, while jump/crouch driven movement is smooth.

Wading

The [Wading](#) state is a variation of the [Movement](#) state which is used for walking and swimming. It changes the target speed based on how much of the character is below the water surface.

Drowning

Drowning can be implemented simply by recording the time spent in underwater states in a [float parameter](#) on the graph and acting based on its value. The [DrowningMotionGraphWatcher](#) monobehaviour is attached to the character and applies damage once a parameter reaches a certain value and at set increments thereafter.

Future Development

The current implementation of swimming in NeoFPS should be considered version 1. Version 2 will extend this with extra features and behaviour such as:

- Swimming hand animations
 - Options to prevent shooting while underwater
 - Audio effects
 - Waves
 - Better underwater visuals and post-processing
 - Bouyancy and buoyant objects
 - Better handling of multiple overlapping water zones
-

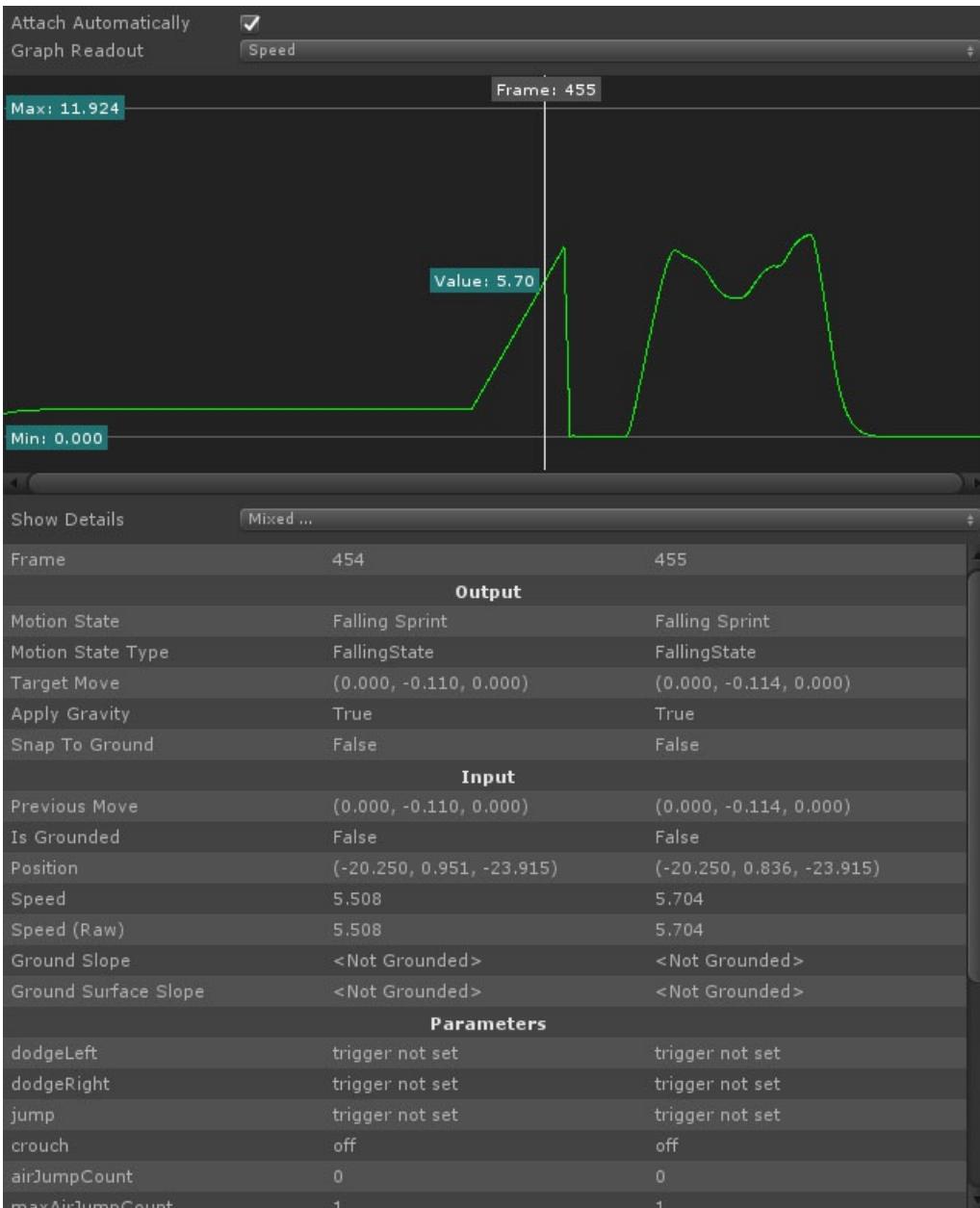
See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

Motion Debugger

Overview



The motion debugger is used to help diagnose unexpected and unwanted movement behaviour by the NeoFPS [Motion Graph](#). It allows you to chart movement details in real time, pause, and focus in on a specific moment to inspect the input and output of the motion graph and [NeoCharacterController](#).

Accessing The Motion Debugger

The motion debugger can be found in the menus at *Tools/NeoFPS/Motion Debugger* or via the **Attach Debugger** button on the [MotionController][3] component attached to the character. If the character is in the scene and the game is playing then this button will attach the selected character to the debugger and start recording its movement data. Alternatively, you can select **Attach Automatically** at the top of the motion debugger, and it will attach to the local player character whenever it spawns.

Each time a new character is attached to the debugger, it resets the recorded data, so if the problem movement results in the death of the character then you will either need to pause the game before the character respawns, or disable the automatic attachment.

Output Graph

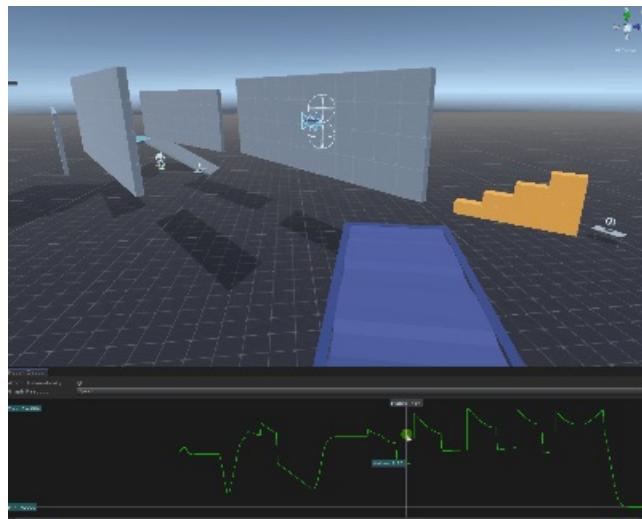
The graph will chart the value selected in the **Graph Readout** dropdown just above. Values are recorded regardless of whether

they are displayed, so changing this value does not reset the graph.

The vertical axes of the graph will resize dynamically based on the minimum and maximum value recorded in the time range. These min and max values are displayed to one side of the graph.

If the game is paused or play mode has ended, then you can click and drag on the graph to inspect a specific frame. The selected frame number along with the charted value at that point will be displayed alongside a vertical marker showing your position on the graph.

Scene View Ghost



Scrubbing the graph while the game is paused will show a representation of the character at that frame in the scene view. You can use this to help find the exact point that the character did something unexpected, and then use the readout values to diagnose the problem. Due to the way that Unity draws object gizmos in the scene, this can only be done in pause mode and not after exiting play mode to editor mode, though the recorded values will persist until a new character is attached to the debugger.

Readout

Underneath the graph is a readout of the various variables for the current frame and previous frame. You can filter which values are visible using the **Show Details** dropdown at the top of the list.

Output

The outputs are the values the motion graph sent to the NeoCharacterController on the inspected frame. This includes the state, target move vector and whether gravity and ground snapping should be applied.

Inputs

These are the variables the NeoCharacterController provided to the motion graph to base its decisions from. Most of these are the result of the previous frame's movement.

Parameters

These are the values of the motion graph parameters at the point the output was sent to the NeoCharacterController. These can help diagnose which of the conditions were valid that frame, though bear in mind that these can be modified by the conditions and behaviours on the graph, as well as external sources.

See Also

[ContactLadder](#)

[InteractiveLadder](#)

Motion Graph Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

I Get "Circular Reference Error" Messages In Console

This means that somewhere in your motion graph, the valid transitions between states or sub-graphs at this moment loop back on themselves. This is almost always caused by a setup error where the conditions for the transition into a state or sub-graph as well as the transition back out can be true at the same time.

An example is if you have a transition into a state with a **Velocity Condition** that is true if **Vertical Velocity >= 0** and transition back out with a **Velocity Condition** that is true if **Vertical Velocity <= 0**. In this situation, if the character has a vertical velocity of *exactly* 0, both of these conditions will be true.

This can also apply to a huge range of logic errors, and also does not need to be transitions in and out of the same node. Each fixed frame, the motion graph checks the transitions out of the current state (and its parent sub-graphs) and switches state based on the valid transitions. If at any point the chain of valid transitions hits a state or sub-graph that has already been tested, you will get this error.

To fix this, you will need to pay attention to the error in the console. This will list the sequence of states / sub-graphs that were tested against. If one of those states appears multiple times, then that is the problem one and you need to check that the transitions out of it and into it (including any jumps in between) cannot be valid at the same time.

My Character Movement Isn't Behaving As Expected

This is a pretty broad issue. The motion graph is a powerful but complex system, so it's only natural that sometimes it doesn't behave as expected. The best way to troubleshoot this is by using the [Motion Debugger](#). You can attach this tool to your character at runtime, and then view graphs of and record your movement as you play. To troubleshoot:

- Open the motion debugger via the Unity menu: *Tools/NeoFPS/Motion Debugger*
- Switch on **Attach Automatically** in the debugger
- Play the game and try to recreate the problem
- Pause the game as soon as you have hit the problem (the debugger records the last 10 seconds)
- Click and drag in the graph readout to scrub through your character movement. You will see a ghost of the character collider in the scene view to help you find the specific frame it went wrong
- Underneath the graph is a readout of inputs and outputs from the character controller and motion graph each frame. On the right is the selected frame in the graph, and on the left is the frame before that
- Find the first frame where the movement went wrong and compare the inputs and outputs to find the property that changed in an unexpected way between them
- Follow that lead to work out the cause (there is some detective work involved)

I'm Trying To Swim But Just Fall Through The Water Zone

This suggests one of two things: that the water zone's trigger physics is not set up correctly, or that the character's motion graph is not reacting to the water zone.

To troubleshoot the water zone physics:

- Select the water zone object in your scene (or prefab)
- Check that the object has a water zone component such as **BasicWaterZone**
- Make sure that object has a collider with **Is Trigger** set to **true**
- Check that the object's layer is set to **TriggerZones**

To troubleshoot the motion graph side of things:

- Ensure that the water zone's **Parameter Key** property is not blank

- Open the motion graph for the character you are using
- Check in the **Parameters** tab that the motion graph has a **Transform Parameter** with the same name as the above parameter key
- Check that your swimmings states / subgraphs are using the correct parameter to transition in or out. You can look at the motion graph from the swimming demo to understand how this should work.

Moving Platform Seems To Jitter When Riding It

The NeoFPS character performs its movement logic in FixedUpdate and interpolates its position and rotation between the last 2 fixed frame results during `Update()`.

Note: this does not apply to aim rotation as this is updated based on mouse input each update frame.

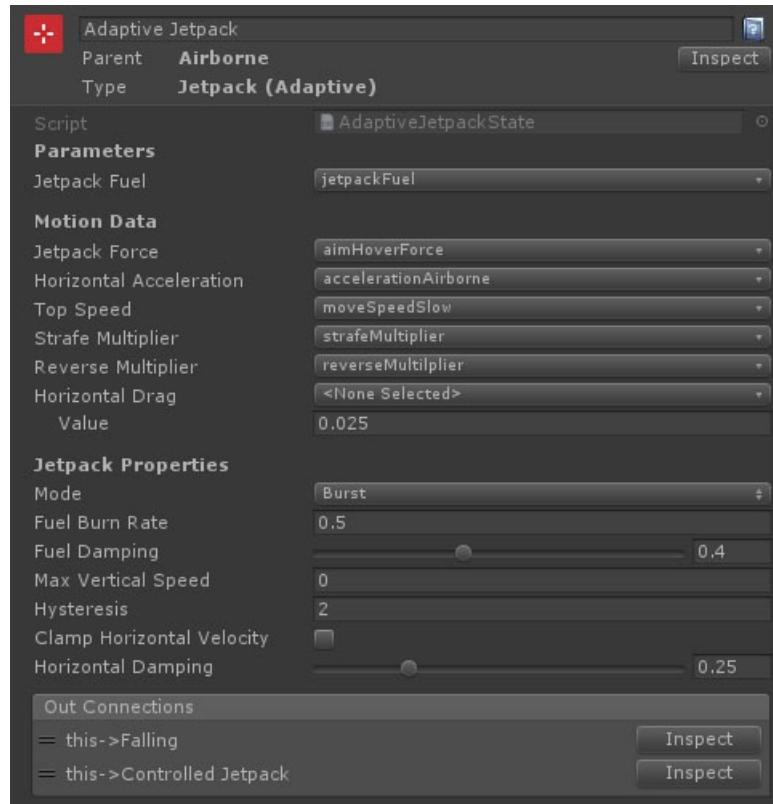
If you are using custom platform components, or the [DrivenMovingPlatform](#) component on a moving Rigidbody object, you must make sure that the **Rigidbody** component's **Interpolate** property is set to **Interpolate**. Without this, the platform and character will appear to move on separate frames to each other and the platform movement will seem to stutter.

AdaptiveJetpack MotionGraphState

Overview

The AdaptiveJetpack state adds an upward force to the character while allowing some control over direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Jetpack Fuel	FloatParameter	An optional parameter for the jetpack fuel. Will be consumed based on output.
Jetpack Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry .
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry .
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Mode	Dropdown	How the jetpack should work. Smooth reduces power as approaching the max vertical speed. Burst sets a minimum jetpack burst duration and gap between bursts to create a bouncier hover.

NAME	TYPE	DESCRIPTION
m_MaxVerticalSpeed	Float	The maximum vertical speed, at which the jetpack stops pushing upwards. A speed of zero is hovering.
m_SpeedFalloff	Float	The speed below the max at which jetpack power (and fuel consumption) starts to fall off. The power will fade out exponentially the closer you get to the max speed. Only shown if the "Mode" dropdown is set to Smooth .
m_Hysteresis	Float	A speed differential where the jetpack will switch on/off. Max + half this = off. Max - half this = on. Only shown if the "Mode" dropdown is set to Burst .
m_FuelBurnRate	Float	The amount of fuel burned per second at full burn. Only shown if the "Jetpack Fuel" parameter is not null.
m_FuelDamping	Float	A damping amount for the fuel consumption to smooth it out. Set to zero for direct feedback. Only shown if the "Jetpack Fuel" parameter is not null.
m_MinFuelBurn	Float	The fuel burn rate when at the target speed, as opposed to accelerating towards it. Only shown if the "Jetpack Fuel" parameter is not null.
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Damping	Float	The amount of damping to apply when changing direction.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

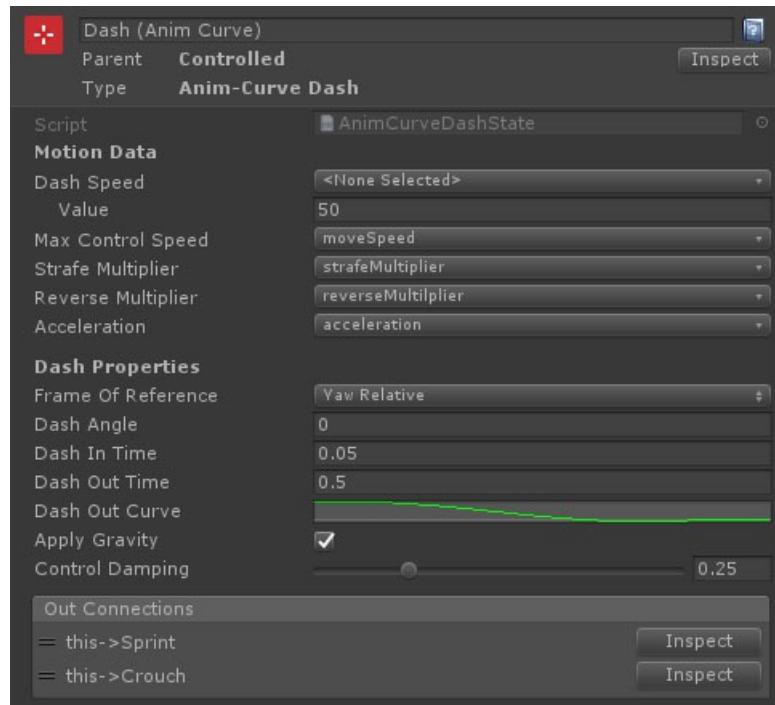
[Motion Graph States](#)

AnimCurveDash MotionGraphState

Overview

The AnimCurveDash state is a simple movement state that layers a directional dash with speed based on an animation curve, on top of a basic movement system. This gives a lot of control over the feel of the dash, and also allows the player to have some control over the exit speed and dash distance.

Inspector



Properties

Name	Type	Description
Dash Speed	FloatData	The target speed for the dash to reach. This will be layered on top of the control speed.
Max Control Speed	FloatData	The maximum speed the character can reach under motor control (driven by input). The dash velocity will be layered on top of this.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Acceleration	Float Data	The input driven acceleration.
Dash Direction	Dropdown	The direction to base the dash off. This can be Yaw Relative or Move Relative .
Dash Angle	Float	The angle offset for the dash direction. For example, yaw relative and an angle of 90 will dash to the right. -90 will dash to the left.
Dash In Time	Float	The amount of time it takes to reach the dash speed. At this point, the animation curve kicks in to ease out of the dash. A Dash In Time of 0 is instant.

NAME	TYPE	DESCRIPTION
Dash Out Time	Float	The amount of time it takes for the animation curve kicks to ease out of the dash.
Dash Out Curve	Animation Curve	The ease out curve for the dash velocity. This should start at 1. Dipping below zero will mean the dash is moving backwards.
Apply Gravity	Boolean	Should the character fall with gravity during the dash.
Control Damping	Float	The amount of damping to apply to the controlled velocity when changing direction.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

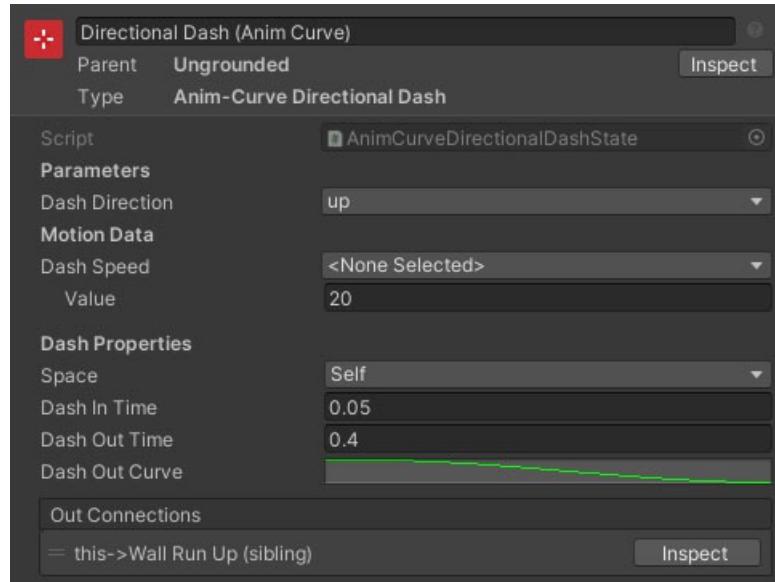
[Motion Graph States](#)

AnimCurveDirectionalDash MotionGraphState

Overview

The AnimCurveDirectionalDash state is a simple movement state that applies a dash in the specified direction, using an animation curve to define the speed. An example usage is dashing while running directly up a wall in the parkour demo scene will perform a dash straight up.

Inspector



Properties

Name	Type	Description
Dash Speed	FloatData	The target speed for the dash to reach.
Dash Direction	VectorParameter	The direction vector parameter to base the dash off.
Space	Dropdown	The space the direction vector exists in.
Dash In Time	Float	The amount of time it takes to reach the dash speed. At this point, the animation curve kicks in to ease out of the dash. A Dash In Time of 0 is instant.
Dash Out Time	Float	The amount of time it takes for the animation curve kicks to ease out of the dash.
Dash Out Curve	Animation Curve	The ease out curve for the dash velocity. This should start at 1. Dipping below zero will mean the dash is moving backwards.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

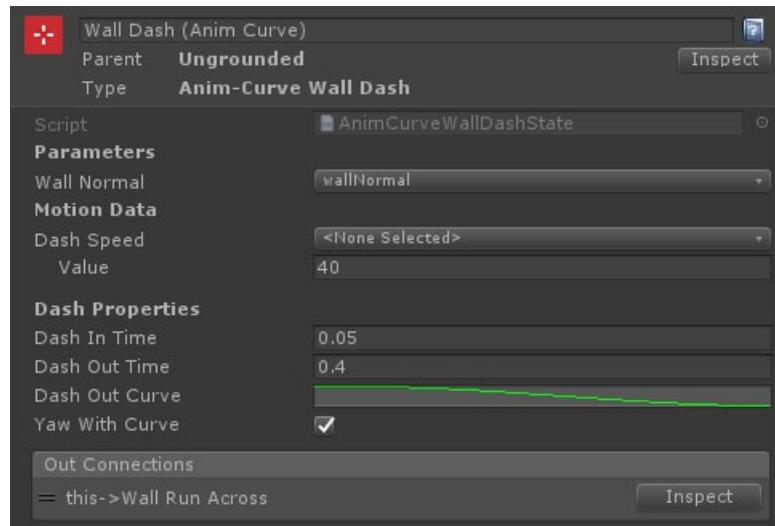
[Motion Graph States](#)

AnimCurveWallDash MotionGraphState

Overview

The AnimCurveWallDash state is a simple movement state that layers a directional dash with speed based on an animation curve, on top of a basic movement system. This gives a lot of control over the feel of the dash, and also allows the player to have some control over the exit speed and dash distance.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Dash Speed	FloatData	The target speed for the dash to reach. This will be layered on top of the control speed.
Wall Normal	Vector Parameter	The wall normal parameter, as used by the wall run states.
Dash In Time	Float	The amount of time it takes to reach the dash speed. At this point, the animation curve kicks in to ease out of the dash. A Dash In Time of 0 is instant.
Dash Out Time	Float	The amount of time it takes for the animation curve kicks to ease out of the dash.
Dash Out Curve	[Animation Curve] [unity-animationcurve]	The ease out curve for the dash velocity. This should start at 1. Dipping below zero will mean the dash is moving backwards.
Yaw With Curve	Boolean	Should the yaw direction of the character be turned if the character must change directions to curve with the wall. It's easy to become disoriented with this disabled.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

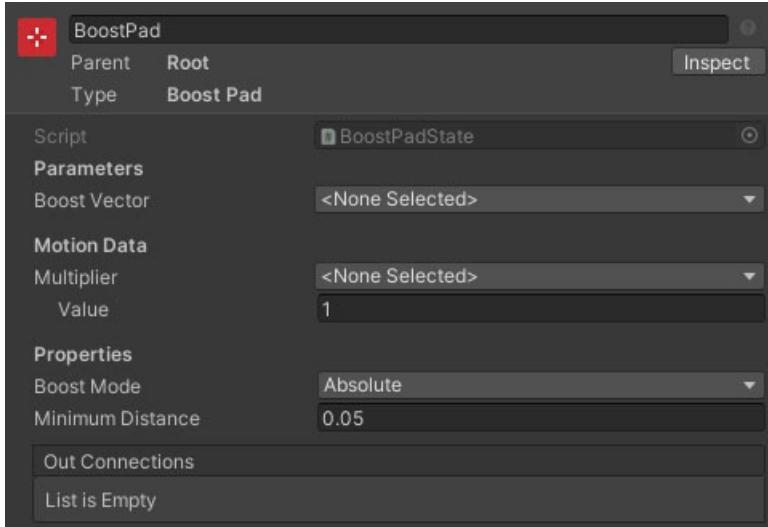
[Motion Graph States](#)

BoostPad MotionGraphState

Overview

The BoostPad state is a simple one frame movement state which sets the character's velocity to the velocity vector scaled.

Inspector



Properties

NAME	TYP	DESCRIPTION
Boost Vector	VectorParameter	The movement velocity to apply.
Boost Mode	Dropdown	How the boost vector is applied to the character. Options are: Absolute sets the character velocity, Additive adds the boost to the character velocity, MaintainPerpendicular sets the velocity in the direction of the boost, but keeps any velocity along the plane perpendicular to the boost.
Multiplier	FloatData	A multiplier for the movement velocity.
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

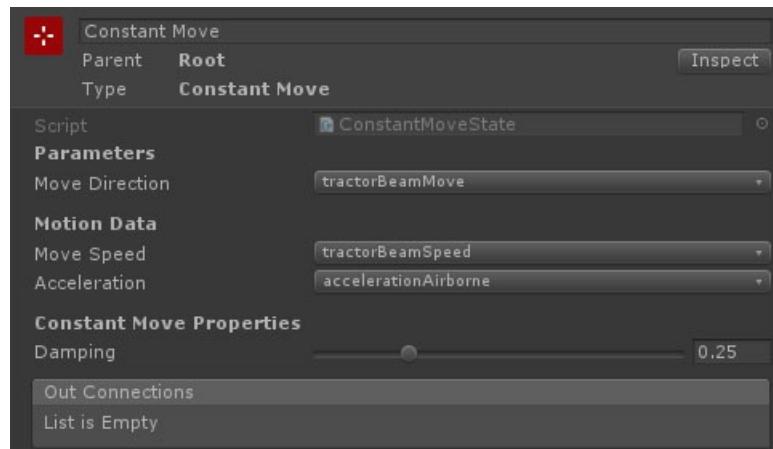
[MotionControllerData ScriptableObject][6]

ConstantMove MotionGraphState

Overview

The ConstantMove state accelerates to a set velocity and then maintains it indefinitely.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Move Direction	VectorParameter	A vector parameter used to define the direction in world space.
Move Speed	FloatData	The target movement speed. This can also be negative to move back along the direction vector.
Acceleration	FloatData	The maximum acceleration.
Damping	Float	The amount of damping to apply when changing direction or speed.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

[MotionControllerData ScriptableObject][6]

ContactLadder MotionGraphState

Overview

The contact ladder state handles climbing up and down physics ladders. As soon as the character walks into one of these ladders, the [ladder TransformProperty](#) will be set. You should set up an in-bound connection to this state with a [TransformCondition](#) that checks if the property is not null, and an outbound connection with another that checks if the property is null.

The character can exit the ladder by jumping or moving past the ladder limits (horizontal and vertical). Doing either of these will set the [ladder TransformProperty](#) to null and the state completed flag to true.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Transform Parameter	TransformParameter	The transform parameter on the graph that is used to attach to ladder transforms in the scene.
Climb Speed	Float Data	The maximum climb speed (based on input and aim). This can either be set as a value, or by referencing a float data entry .
Ground Speed	Float Data	The top movement speed with ground under foot (top and bottom of the ladder). This can either be set as a value, or by referencing a float data entry .
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Acceleration	Float Data	The maximum acceleration. This can either be set as a value, or by referencing a float data entry .

Name	Type	Description
Use Aimer V	Dropdown	<p>Should the direction of the camera aim influence the vertical move direction?</p> <ul style="list-style-type: none"> • IgnoreAimer = No • AimerAbsolute = Direction of aim changes direction of move • AimerSmooth = Direction of aim changes direction and speed of move • AimerHeading = Direction and speed based on yaw only (factors in both input axes) • AimerAllAxes = Direction and speed based on yaw and pitch (factors in both input axes)
Center Zone	Float	<p>For AimerAbsolute, the angle past the horizontal you need to aim to flip directions. For AimerSmooth or AimerAllAxes, the angle past the horizontal that reaches full speed.</p>
Use Aimer H	Dropdown	<p>Should the camera aim influence the horizontal move direction?</p> <ul style="list-style-type: none"> • IgnoreAimer = No • AimerAbsolute = Direction of aim changes direction of move • AimerSmooth = Direction of aim changes direction and speed of move • AimerHeading = Direction and speed based on yaw only (factors in both input axes) • AimerAllAxes = Direction and speed based on yaw and pitch (factors in both input axes)

See Also

[The Motion Graph](#)

[Motion Graph States](#)

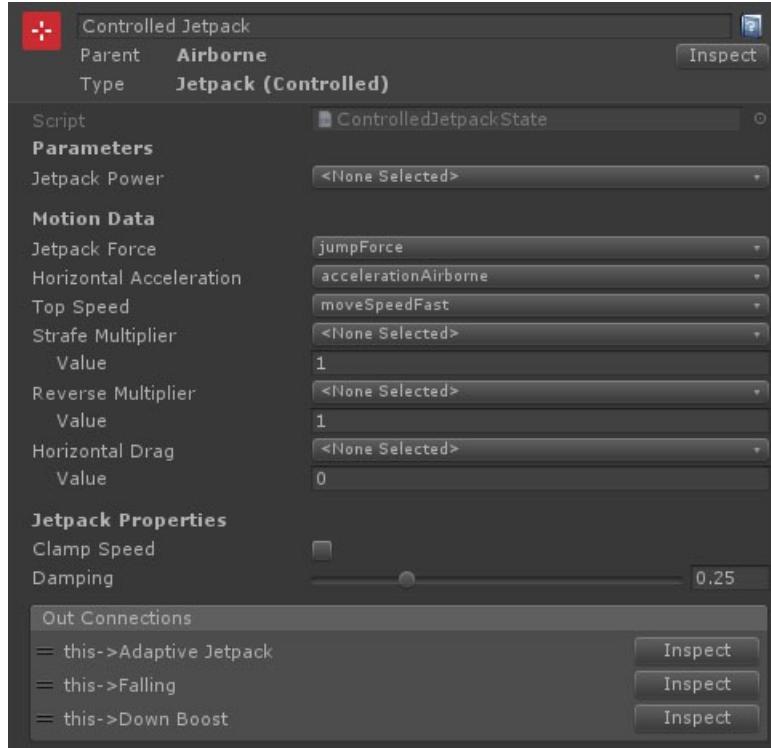
[Motion Graph Parameters And Data](#)

ControlledJetpack MotionGraphState

Overview

The ControlledJetpack state adds an upward force to the character while allowing some control over direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Jetpack Power	FloatParameter	An optional parameter that acts as a multiplier for the jetpack force (0.5 is half power, etc)..
Jetpack Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry .
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry .
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Damping	Float	The amount of damping to apply when changing direction.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

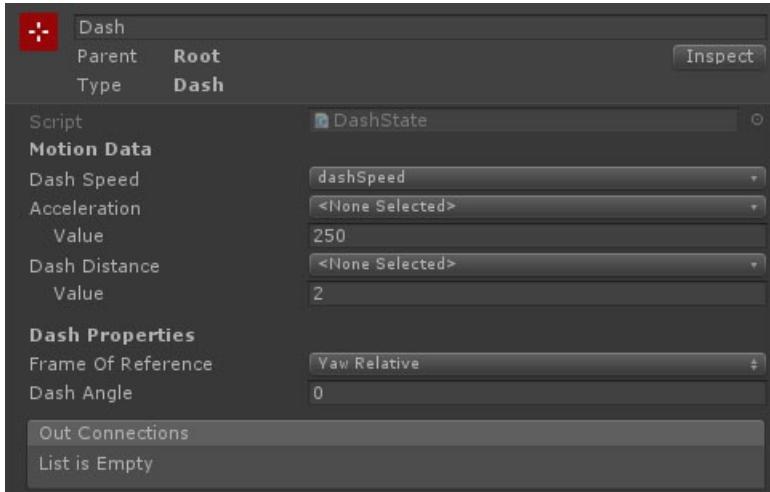
Dash MotionGraphState

Overview

The Dash state is a simple movement state that surges in a specific direction. The direction can be based off the character's yaw forward or the direction it is currently moving, along with an angle offset.

Once the dash has attempted to move a specific distance it will signal completion.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Dash Speed	FloatData	The target speed for the dash to reach.
Dash Acceleration	FloatData	The acceleration to reach the target dash speed.
Dash Distance	FloatData	The distance to dash before the state completes.
Dash Direction	Dropdown	The direction to base the dash off. This can be Yaw Relative or Move Relative .
Dash Angle	Float	The angle offset for the dash direction. For example, yaw relative and an angle of 90 will dash to the right. - 90 will dash to the left.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

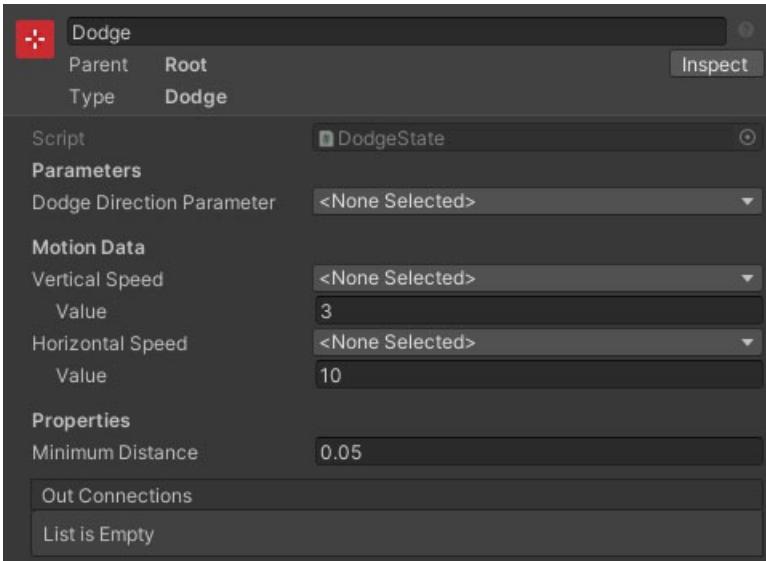
[Motion Graph States](#)

Dodge MotionGraphState

Overview

The dodge state adds a simple directional impulse to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the [CompletedCondition](#), connecting to an airborne state.

Inspector



Properties

Name	Type	Description
Vertical Speed	Float Data	The vertical dodge speed. This can either be set as a value, or by referencing a float data entry .
Horizontal Speed	Float Data	The horizontal dodge speed. This can either be set as a value, or by referencing a float data entry .
Dodge Direction Parameter	IntParameter	<p>The dodge direction is a simple compass heading relative to the character's forward direction. It is set in one of the graph properties and conforms to the following:</p> <ol style="list-style-type: none">1. North2. North East3. East4. South East5. South6. South West7. West8. North West
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

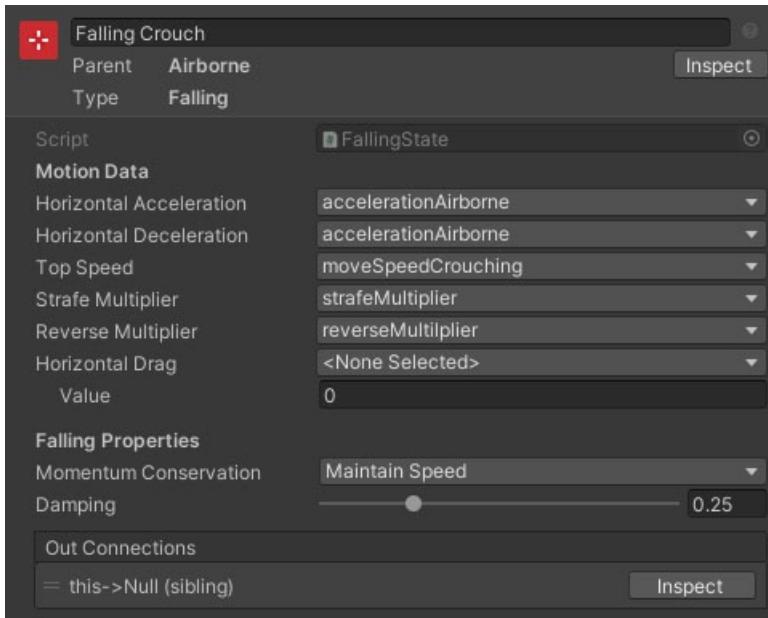
[Motion Graph Parameters And Data](#)

Falling MotionGraphState

Overview

The falling state behaves as the name suggests, but provides some level of control when in the air.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry .
Horizontal Deceleration	Float Data	The input driven deceleration while falling (if input is zero or less than previous). This can either be set as a value, or by referencing a float data entry .
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry .
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Horizontal Drag	Float Data	A drag deceleration applied to horizontal movement. This can either be set as a value, or by referencing a float data entry .
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Momentum Conservation	Dropdown	Should the speed of the character decelerate to top speed if going faster. Options are MaintainSpeed (the speed will not be modified), ClampSpeed (the speed will be clamped to top speed if no input, or the input target speed if there is) and ClampToInput (the speed will decelerate to zero if no input, or be clamped to the input target speed if there is).
Damping	Float	The amount of damping to apply when changing direction.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

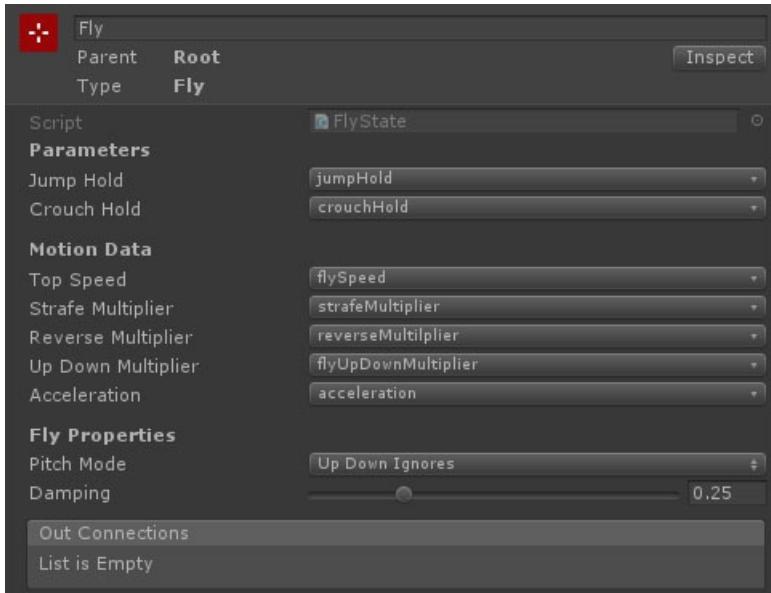
[Motion Graph Parameters And Data](#)

Fly MotionGraphState

Overview

The Fly state is a simple noclip style flying style. The character moves based on their aim direction. Jump moves up, while crouch moves down.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Jump Hold	SwitchParameter	The crouch hold parameter (used for flying up).
Crouch Hold	SwitchParameter	The crouch hold parameter (used for flying down).
Top Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed when moving backwards.
Up Down Multiplier	FloatData	The multiplier applied to the max movement speed when moving up/down.
Acceleration	FloatData	The maximum acceleration.
Pitch Mode	Dropdown	How does the camera pitch affect the movement. The available options are: <ul style="list-style-type: none">• Up Down Ignores means jump/crouch move straight up or down.• Affects All Axes means that up/down are along the camera up axis.• Ignore only takes yaw into account for all movement
Damping	Float	The amount of damping to apply when changing direction or speed.

See Also

[The Motion Graph](#)

The Motion Graph Editor

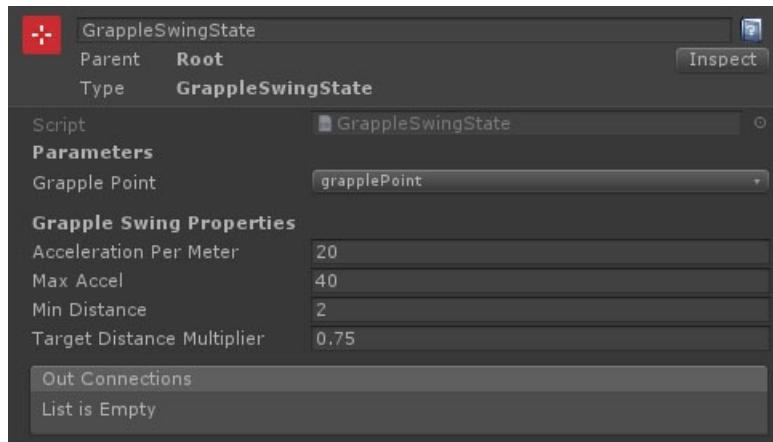
Motion Graph States

GrappleSwing MotionGraphState

Overview

The GrappleSwing state pulls the character towards a grapple point and swings as though tethered to it with a springy rope. One option for setting the grapple point is by using the [GrappleToolModule](#) with the [wieldable tools](#) system

Inspector



Properties

NAME	TYP	DESCRIPTION
Grapple Point	VectorParameter	The point in space that the grapple is tethered to.
Target Distance Multiplier	Float	When entering the state, a target distance will be calculated based on multiplying the current distance to the grapple point by this multiplier.
Min Distance	Float	Below this distance from the grapple point, the character will actually be pushed away.
Acceleration Per Meter	Float	The acceleration towards the grapple point per meter distance above the target distance.
Max Accel	Float	The maximum acceleration towards the grapple point.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

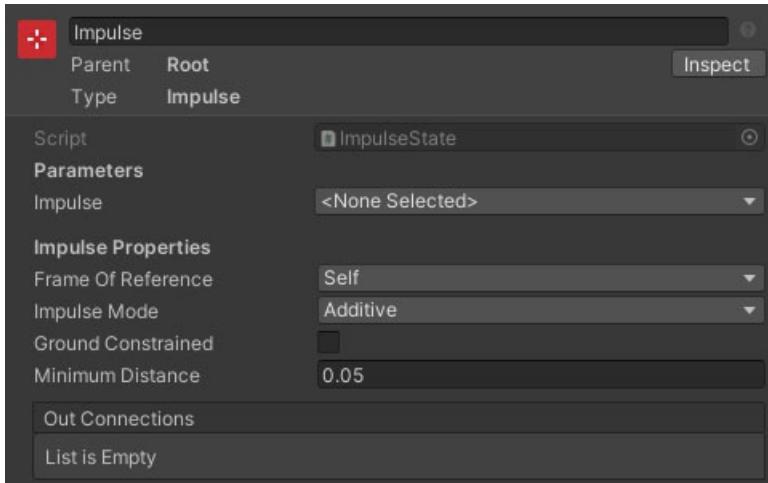
[GrappleToolModule](#)

Impulse MotionGraphState

Overview

The Impulse state is a one frame state that adds or sets the character velocity. An example use is at the start of a crouch-slide to add an instantaneous speed boost.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Impulse	VectorParameter	The velocity impulse to apply.
Frame Of Reference	Dropdown	The coordinate space to apply the impulse in. Options are Self and World .
Impulse Mode	Dropdown	How should the impulse be applied. Additive will add the impulse velocity to the original velocity. Replace Velocity will ignore the original velocity and use the impulse alone.
Ground Constrained	Boolean	If true, the impulse vector will be aligned onto the ground surface normal plane.
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

InteractiveLadder MotionGraphState

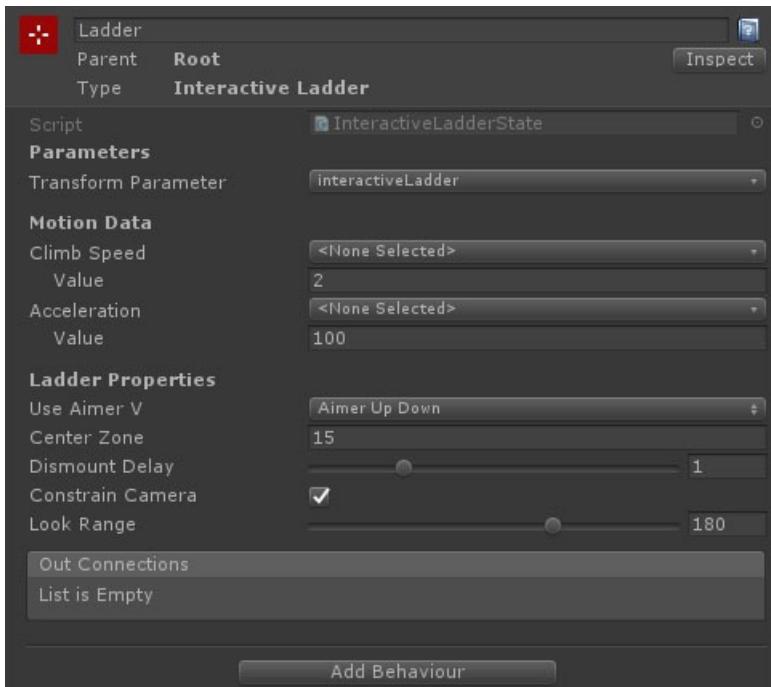
Overview

The InteractiveLadder state handles climbing up and down interactive ladders. These are ladders are simple obstacles until interacted with, at which point the [ladder TransformProperty](#) will be set. You should set up an in-bound connection to this state with a [TransformCondition](#) that checks if the property is not null, and an outbound connection with another that checks if the property is null.

The character can exit the ladder by interacting with it again, jumping, or moving past the ladder limits. Doing any of these will set the [ladder TransformProperty](#) to null and the state completed flag to true.

The state has various properties for how the character climbs the ladder, along with camera constraints.

Inspector



Properties

Name	Type	Description
Transform Parameter	TransformParameter	The transform parameter on the graph that is used to attach to ladder transforms in the scene.
Climb Speed	Float Data	The maximum climb speed (based on input and aim). This can either be set as a value, or by referencing a float data entry .
Acceleration	Float Data	The acceleration when on the ladder or attaching / dismounting. This can either be set as a value, or by referencing a float data entry .
Use Aimer V	Dropdown	Should the direction of the camera aim influence the vertical move direction? <ul style="list-style-type: none">• IgnoreAimer = No• AimerUpDown = Only direction• AimerSmooth = Direction and speed
Center Zone	Float	For AimerUpDown , the angle past the horizontal you need to aim to flip directions. For AimerSmooth , the angle past the horizontal that reaches full speed.

NAME	TYPE	DESCRIPTION
Dismount Delay	Float	How long is the character blocked from stepping straight off the ladder using up/down? For example, if they attach to the top of the ladder then they should not immediately step off by pressing up.
Constrain Camera	Boolean	If true, constrain the camera's horizontal rotation to a maximum range from looking directly at the ladder.
Look Range	Float	The angle range (degrees) that the camera can turn from looking straight at the ladder (180 = 90 degrees to either side).

See Also

[The Motion Graph](#)

[Motion Graph States](#)

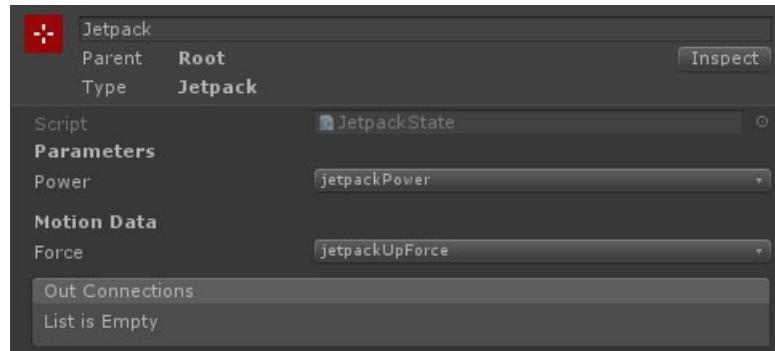
[Motion Graph Parameters And Data](#)

Jetpack MotionGraphState

Overview

The Jetpack state maintains horizontal velocity and adds an upward force.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Power	FloatParameter	An optional parameter that acts as a multiplier for the jetpack force (0.5 is half power, etc).
Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

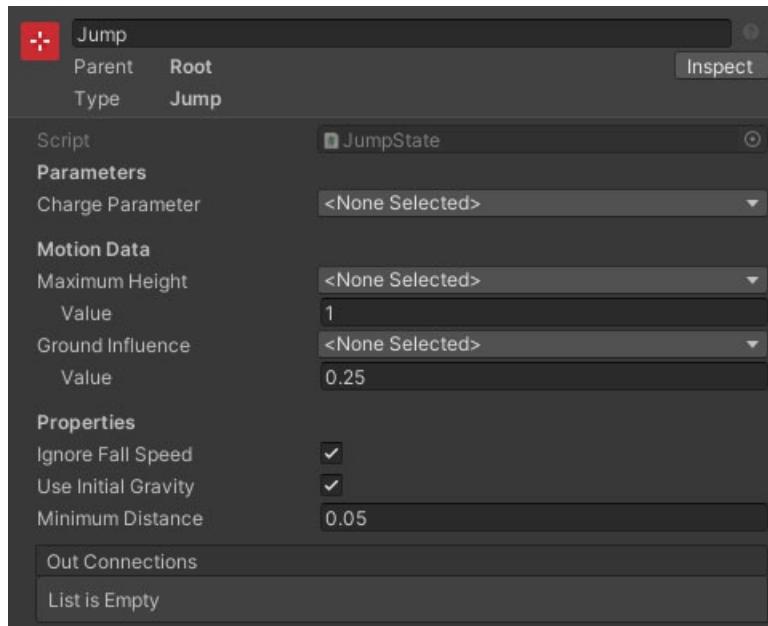
[Motion Graph States](#)

Jump MotionGraphState

Overview

The jump state adds a simple upward impulse to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the [CompletedCondition](#), connecting to an airborne state.

Inspector



Properties

Name	Type	Description
Charge Parameter	FloatParameter	The charge property is a float with value 0 to 1 that defines how high to jump. If no graph property is selected, the jump will always be performed with full strength.
Maximum Height	Float Data	The height the player will jump when fully charged. Note: Changes to gravity or physics multiplier after start will affect this. Also does not take step height into account. This can either be set as a value, or by referencing a float data entry .
Minimum Height	Float Data	The smallest height the player will jump (at the equivalent of a zero tap - actually unattainable). This can either be set as a value, or by referencing a float data entry .
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal. This can either be set as a value, or by referencing a float data entry .
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

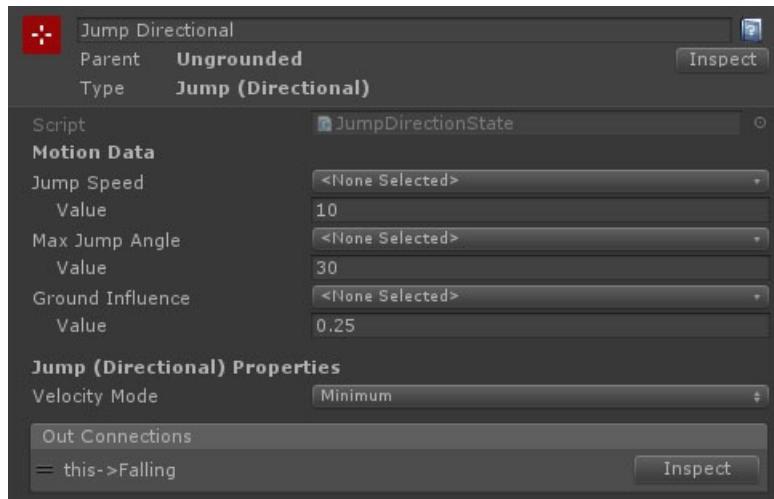
[Motion Graph Parameters And Data](#)

JumpDirection MotionGraphState

Overview

The jump direction state adds a jump direction to the [Jump](#) state. You should add an out-bound connection from this state that uses the [CompletedCondition](#), connecting to an airborne state. This state should be replaced with the [JumpDirectionV2](#) motion graph state instead, as it gives more predictable results with analogue controllers.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Jump Speed	Float Data	The instant speed of the jump.
Maximum Height	Float Data	The angle the jump direction will be tilted in the character input direction when input scale is 1.
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal.
Velocity Mode	Dropdown	How should the velocity be applied. Available options are: <ul style="list-style-type: none">• Additive will add the jump velocity to the original velocity.• Absolute will ignore the original velocity.• Minimum will boost the character velocity if it is less than the jump speed in the jump direction.
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

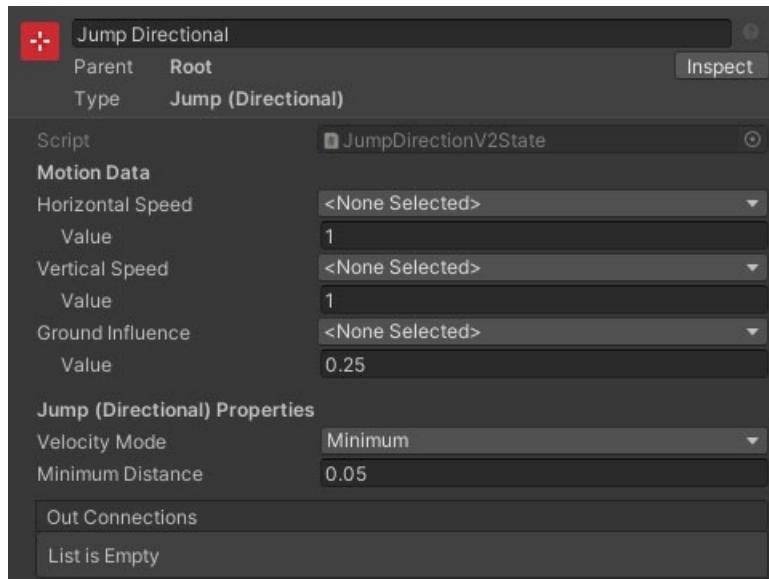
[Motion Graph Parameters And Data](#)

JumpDirectionV2 MotionGraphState

Overview

The JumpDirectionV2 state adds a jump direction to the [Jump](#) state. This version gives a more consistent result with analogue controllers than the original JumpDirection motion graph state and should be used in its place.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Horizontal Speed	Float Data	The horizontal speed of the jump.
Vertical Speed	Float Data	The upward speed of the jump.
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal.
Velocity Mode	Dropdown	How should the velocity be applied. Available options are: <ul style="list-style-type: none">• Additive will add the jump velocity to the original velocity.• Absolute will ignore the original velocity.• Minimum will boost the character velocity if it is less than the jump speed in the jump direction.
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

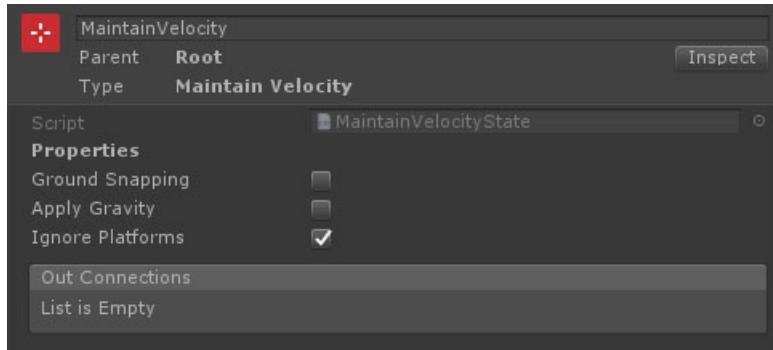
[Motion Graph Parameters And Data](#)

MaintainVelocity MotionGraphState

Overview

The MaintainVelocity state is a utility state that simply moves the character at the same velocity as last frame. This will be affected by collisions.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Ground Snapping	Boolean	Should ground snapping be applied.
Apply Gravity	Boolean	Should gravity force be applied.
Ignore Platforms	Boolean	Should the character inherit movement from platforms it's touching.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

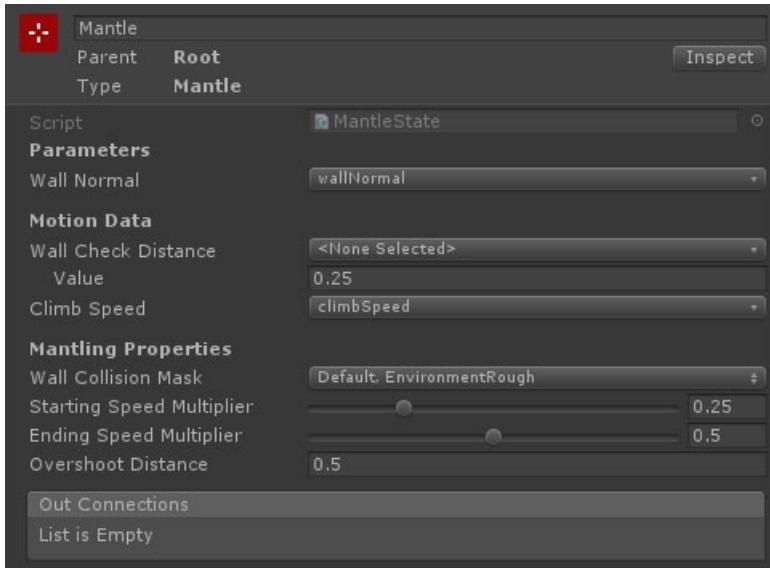
[Motion Graph States](#)

Mantle MotionGraphState

Overview

The Mantle state is used to climb a wall ledge onto its top surface. It is paired with the [Climbable](#) condition to test if a wall impact is climbable.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Wall Normal	VectorParameter	The normal of the wall to climb. This value will be read from and written to each frame.
Wall Check Distance	FloatData	The cast distance for the initial wall check. Using a motion data entry allows the value to be shared with the Climbable condition.
Climb Speed	FloatData	The movement speed while climbing the surface.
Wall Collision Mask	LayerMask	The collision mask to use when checking the wall normal.
Starting Speed Multiplier	Float	The climb speed multiplier (for the data value above) on entering the state.
Ending Speed Multiplier	Float	The climb speed multiplier (for the data value above) on completing the ledge mantle.
Overshoot Distance	Float	The distance to move past the edge onto flat ground before completing.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

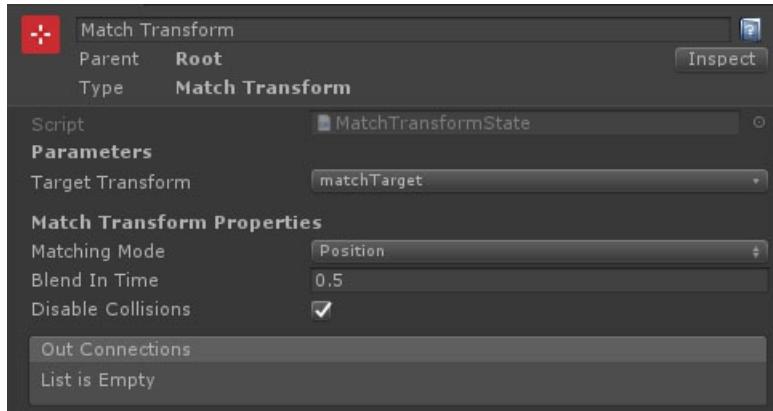
[Motion Graph States](#)

MatchTransform MotionGraphState

Overview

The MatchTransform state makes the character move as though it is attached to a transform in the scene. This allows you to sync the character to animated cut-scenes or create complex on-rails movement systems for certain situations.

Inspector



Properties

Name	Type	Description
Target Transform	Transform Parameter	The transform to match to.
Matching Mode	Dropdown	What transform components to match. Options are: Position (match the position but ignore rotation), PositionAndUp (match the position and keep the character up-vector aligned but ignore yaw), PositionAndDirection (match position and yaw direction or heading, but maintain the character's up vector), and All (match position, up vector and heading).
Blend In Time	Float	The time taken to blend from current position and velocity to match the transform.
Disable Collisions	Boolean	Should collisions be disabled for the duration of the movement.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

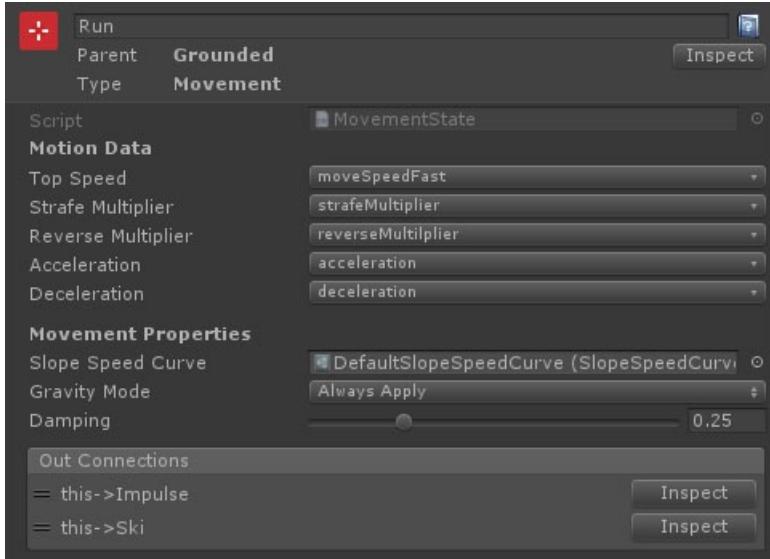
[Motion Graph Parameters And Data](#)

Movement MotionGraphState

Overview

The movement state handles basic ground based movement such as walking, sprinting and sneaking.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Top Speed	Float Data	The top movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry .
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry .
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry .
Acceleration	Float Data	The maximum acceleration. This can either be set as a value, or by referencing a float data entry .
Deceleration	Float Data	The maximum deceleration (when no input is applied). This can either be set as a value, or by referencing a float data entry .
Slope Speed Curve	SlopeSpeedCurve	An optional slope speed curve which controls the movement speed on sloping surfaces. Without this, the character controller will handle the speed change automatically.
Gravity Mode	Dropdown	When to apply gravity to the resulting move vector. Options are: Always Apply , When Not Grounded and Never Apply .
Damping	Float	The amount of damping to apply when changing direction.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

Motion Graph Parameters And Data

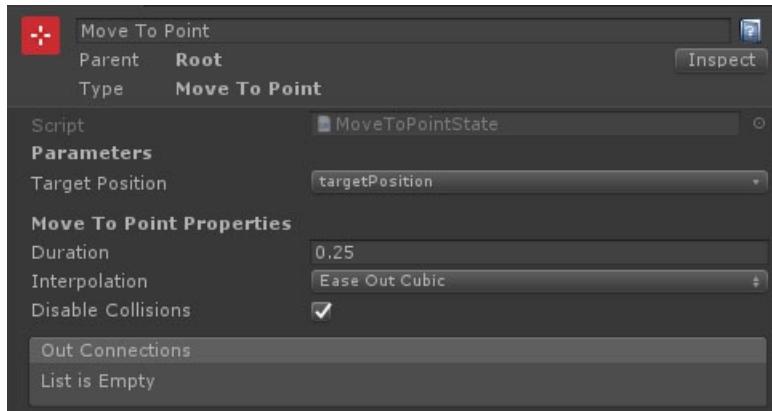
SlopeSpeedCurve ScriptableObject

MoveToPoint MotionGraphState

Overview

The MoveToPoint state is used to move directly from the starting point to a target point with various interpolation options.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Position	Vector Parameter	The position to move to.
Duration	Float	The time required to reach the target.
Interpolation	Dropdown	The interpolation method from start to end. Options are: Linear , EaseOutQuadratic , EaseOutCubic , Spring and Bounce .
Disable Collisions	Boolean	Should collisions be disabled for the duration of the movement.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

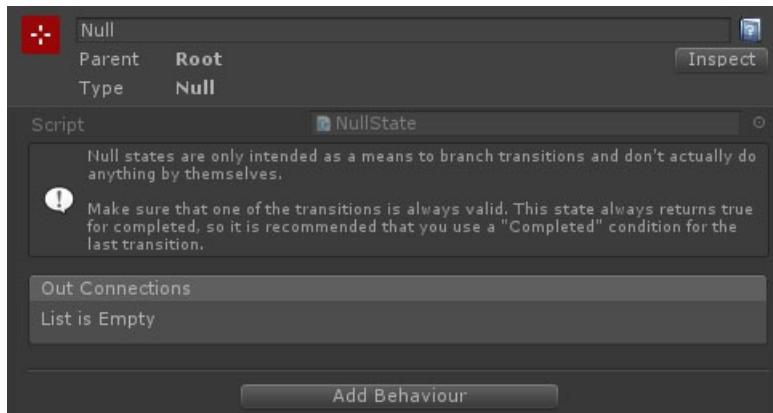
[Motion Graph Parameters And Data](#)

Null MotionGraphState

Overview

The null state is only used to branch to other states in the graph. The state always has the completed flag set, so it is best practice to set the last outbound connection from this state to one that uses the [CompletedCondition](#). This would act as a default connection if none of the others are valid.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Name	String	The name of the state as visible in the graph viewport.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

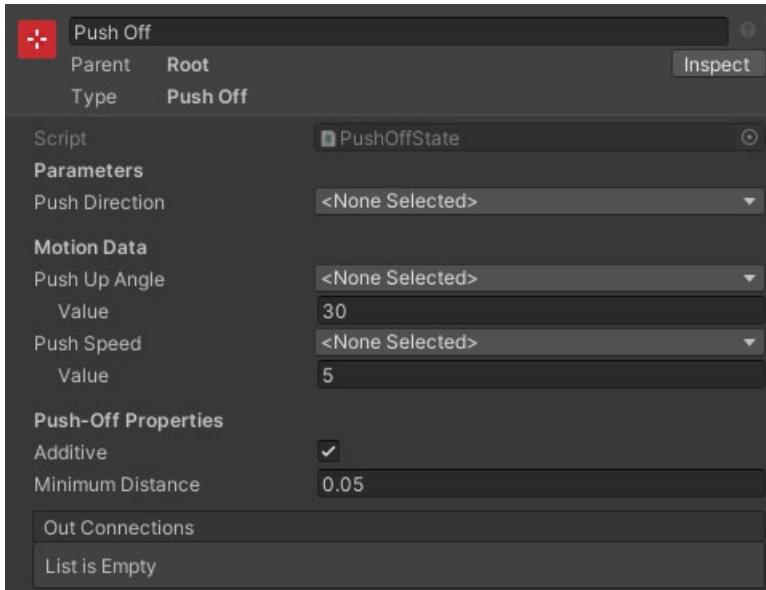
[Motion Graph States](#)

PushOff MotionGraphState

Overview

The PushOff state is a one frame state which adds velocity based on a direction vector and can add an optional extra vertical boost.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Push Direction	VectorParameter	The world direction to push in (you can fill this parameter using enhanced cast conditions).
Push Up Angle	FloatData	An additional upward rotation applied to the push direction. Resulting direction won't rotate past up/down.
Push Speed	FloatData	The speed to along the rotated push direction.
Additive	Boolean	Should the resulting velocity be added to the original character velocity or replace it.
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

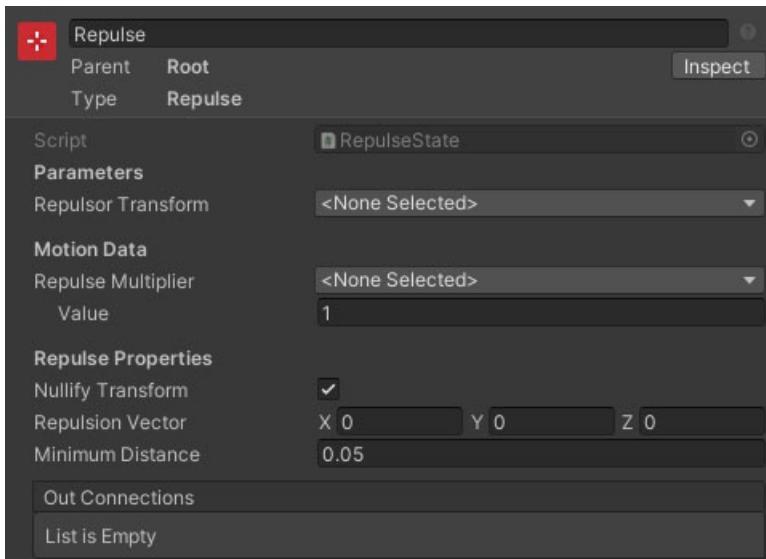
[Motion Graph States](#)

Repulse MotionGraphState

Overview

The repulse state adds a simple directional impulse away from the repulsor transform to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the [CompletedCondition](#), connecting to an airborne state.

Inspector



Properties

Name	Type	Description
Repulsor Transform	TransformParameter	The transform parameter on the graph that is used to specify which transform to repulse from.
Nullify Transform	Boolean	Should the transform be nullified after use?
Repulsion Vector	Vector3	The velocity vector to apply to the character relative to the repulsor transform.
Repulse Multiplier	Float Data	A multiplier for the repulsion vector. This can either be set as a value, or by referencing a float data entry .
Minimum Distance	Float	The minimum distance the state should attempt to move before completing. This prevents a very small fixed time step causing the movement to be too small to overcome ground snapping / detection.

See Also

[The Motion Graph](#)

[Motion Graph States](#)

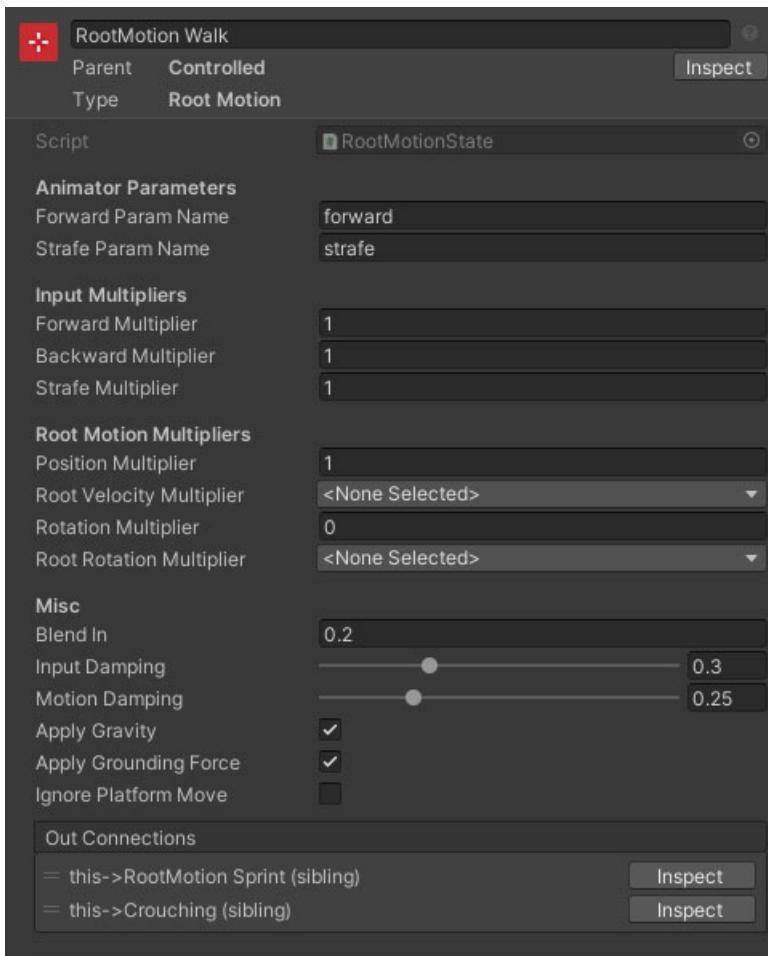
[Motion Graph Parameters And Data](#)

RootMotion MotionGraphState

Overview

The RootMotion state is used when the character has a [first person body](#) and will move the character based on the root motion of its animations. Instead of using the character input directly for movement, it sends them as parameters to the body [Animator](#) to control / blend what animations are playing. To use this, the character must have a body with an animator. It must have animations that use root motion. lastly, it must have a [FirstPersonBodyRootMotion](#) component attached to the character body which catches root motion and sends it to the character's [MotionController](#).

Inspector



Properties

Animator Parameters are the parameters on the character animator to set with the movement values.

NAME	TYPE	DESCRIPTION
Forward Param Name	String	The animator parameter name the forward input value should be written to.
Strafe Param Name	String	The animator parameter name the strafe input value should be written to. (positive = right).

Input Multipliers are multipliers that are applied to the player input before sending to the animator.

| Forward Multiplier | Float | A multiplier applied to the forward input before being sent to the animator parameter. || Backward Multiplier | Float | A multiplier applied to the backwards input before being sent to the animator parameter. || Strafe Multiplier | Float | A multiplier applied to the strafe input before being sent to the animator parameter. |

Root Motion Multipliers are multipliers that are applied to the resulting movement that the blended animations output.

| Position Multiplier | Float | A multiplier applied to the root motion position changes. || Root Velocity Multiplier | [Float Parameter](#)
| An optional parameter that will multiply the root motion movement velocity. This will be applied alongside the value above. ||
Rotation Multiplier | Float | A multiplier applied to the root motion rotation changes. || Root Rotation Multiplier | [Float Parameter](#) |
An optional parameter that will multiply the root motion rotation rate. This will be applied alongside the value above. |

Misc properties are the other properties that affect the resulting movement.

| Blend In | Float | The amount of time it should take to blend into the root motion movement. Blend out is always instant to prevent root motion blocking movements. || Input Damping | Float | Damping smooths out the input values. || Motion Damping | Float | Damping smooths out the root motion values that the animator returns. || Apply Gravity | Boolean | Should gravity be applied while in the state. || Apply Grounding Force | Boolean | Should ground snapping be applied while in the state. || Ignore Platform Move | Boolean | Should the character be affected by moving platforms while in the state. |

See Also

[First Person Body](#)

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

[MotionController](#)

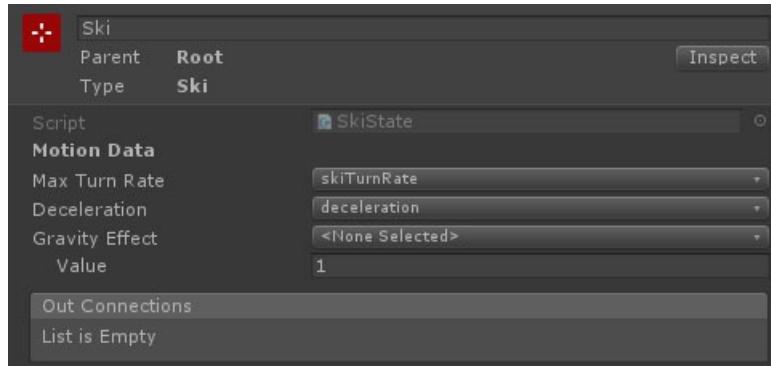
[FirstPersonBodyRootMotion](#)

Ski MotionGraphState

Overview

The Ski state maintains forward velocity and allows the player to steer left and right. Steering turn rate is based on movement speed, with slower movement speed allowing turning in tighter circles.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Turn Rate	FloatData	The maximum turn rate in degrees per second.
Deceleration	FloatData	A constant deceleration (meters per second squared) while skiing. Once the character speed reaches zero, the state completes.
Gravity Effect	FloatData	A multiplier for gravity which affects speed on slopes.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

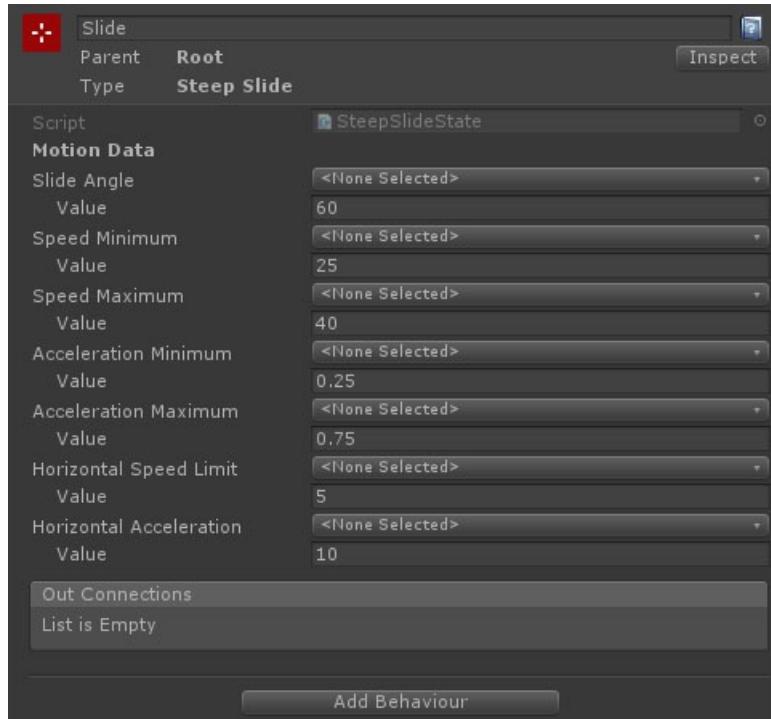
[Motion Graph States](#)

SteepSlide MotionGraphState

Overview

The steep slide state handles sliding down slopes that are too steep to walk on. The character has some degree of control while sliding, but only in the horizontal direction. They can't affect the speed of the slide down the slope.

Inspector



Properties

Name	Type	Description
Slide Angle	Float Data	The angle above which a character loses motor control and is in pure slide mode. This can either be set as a value, or by referencing a float data entry .
Speed Minimum	Float Data	The sliding speed the character will reach (downwards only) at the lowest slope angle for a full slide. This can either be set as a value, or by referencing a float data entry .
Speed Maximum	Float Data	The fastest possible sliding speed the character can reach (downwards only) during a near vertical slide. This can either be set as a value, or by referencing a float data entry .
Acceleration Minimum	Float Data	The down-slope acceleration multiplier applied to a character during a shallow slide. This can either be set as a value, or by referencing a float data entry .
Acceleration Maximum	Float Data	The down-slope acceleration multiplier applied to a character during a near vertical slide. This can either be set as a value, or by referencing a float data entry .
Horizontal Speed Limit	Float Data	The top speed the character can reach against the slide (side to side). This can either be set as a value, or by referencing a float data entry .
Horizontal Acceleration	Float Data	The across slope acceleration when trying to redirect slide sideways (0 is instant). This can either be set as a value, or by referencing a float data entry .

See Also

[The Motion Graph](#)

[Motion Graph States](#)

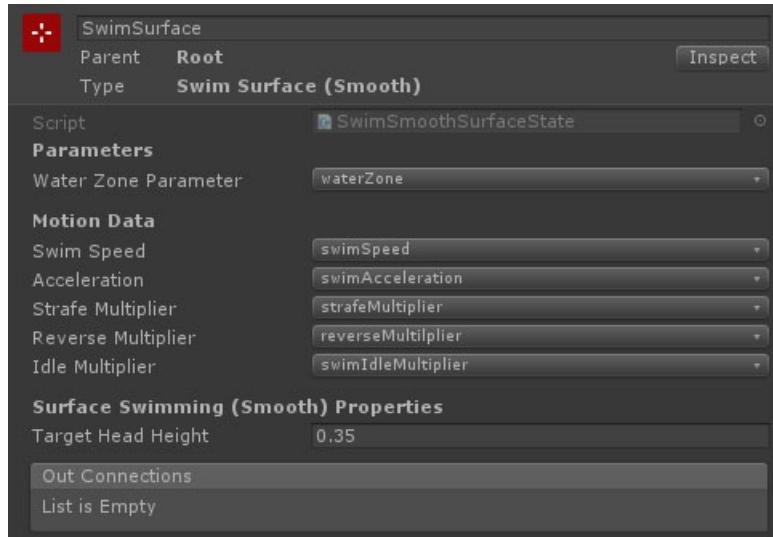
[Motion Graph Parameters And Data](#)

SwimSmoothSurfaceState MotionGraphState

Overview

The SwimSmoothSurfaceState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Swim Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Target Head Height	Float	A target for the distance the character capsule should be above the surface.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

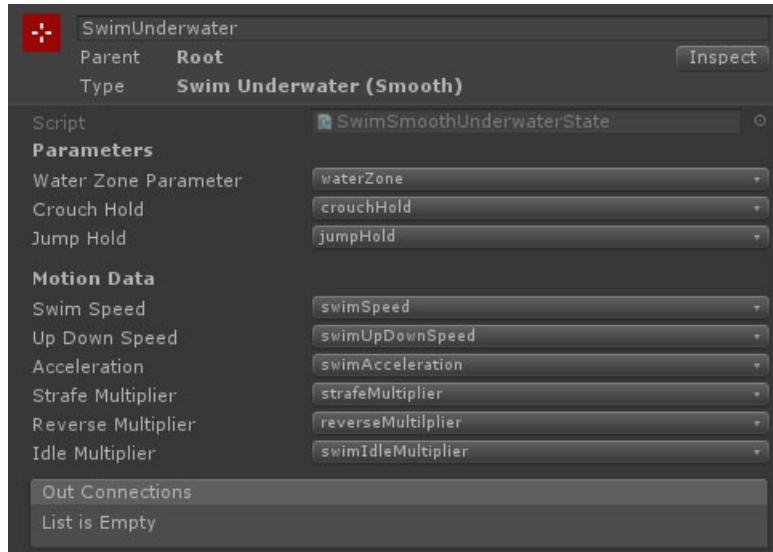
[Motion Graph States](#)

SwimSmoothUnderwaterState MotionGraphState

Overview

The SwimSmoothUnderwaterState state is used to move in all axes when underwater. It moves in strokes, pulling forward and then recovering.

Inspector



Properties

Name	Type	Description
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Jump Hold	SwitchParameter	A switch parameter which tracks if the jump button is held (swim up).
Crouch Hold	SwitchParameter	A switch parameter which tracks if the crouch button is held (swim down).
Swim Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Up Down Speed	FloatData	The maximum movement speed and acceleration due to up or down input.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

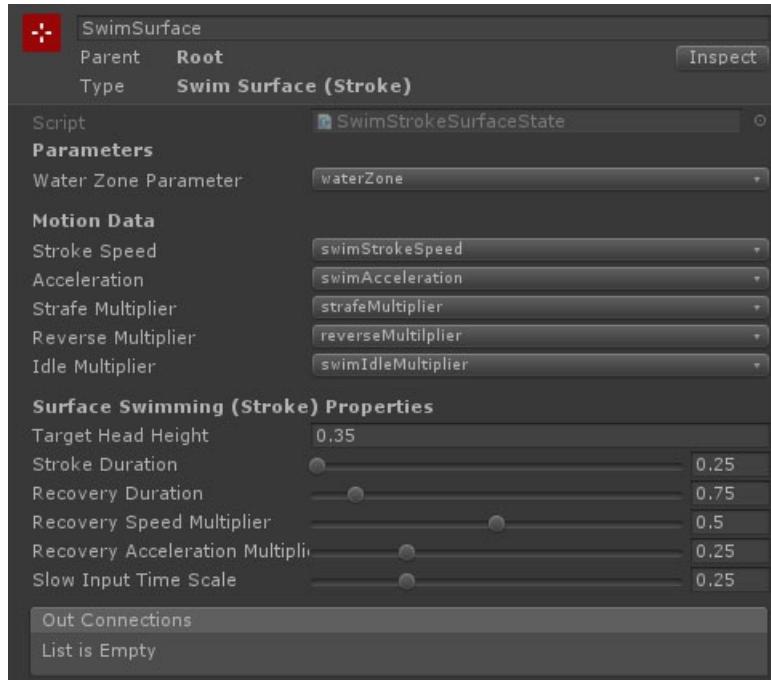
[Motion Graph States](#)

SwimStrokeSurfaceState MotionGraphState

Overview

The SwimStrokeSurfaceState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Stroke Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Target Head Height	Float	A target for the distance the character capsule should be above the surface.
Stroke Duration	Float	The length of time a swimming stroke lasts (will be scaled by strafe / reverse multipliers).
Recovery Duration	Float	The length of time in between swimming strokes (will be scaled by strafe / reverse multipliers).
Recovery Speed Multiplier	Float	A multiplier applied to the speed in between strokes.

NAME	TYPE	DESCRIPTION
Recovery Acceleration Multiplier	Float	A multiplier applied to the acceleration (and deceleration) in between strokes.
Slow Input Time Scale	Float	At the minimum input amount, how much slower are strokes and recovery.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

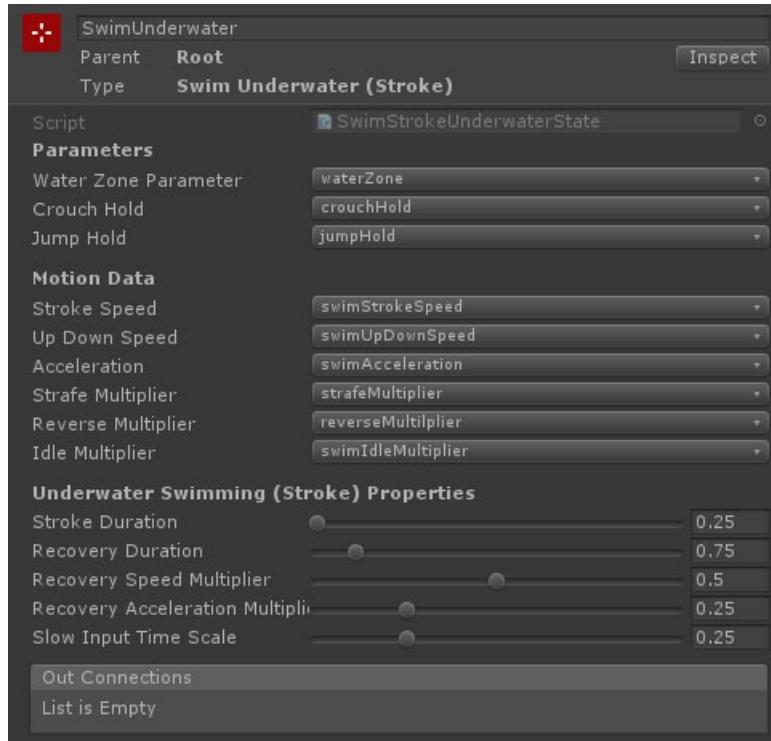
[Motion Graph States](#)

SwimStrokeUnderwaterState MotionGraphState

Overview

The SwimStrokeUnderwaterState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Jump Hold	SwitchParameter	A switch parameter which tracks if the jump button is held (swim up).
Crouch Hold	SwitchParameter	A switch parameter which tracks if the crouch button is held (swim down).
Stroke Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Up Down Speed	FloatData	The maximum movement speed and acceleration due to up or down input.
Stroke Duration	Float	The length of time a swimming stroke lasts (will be scaled by strafe / reverse multipliers).

NAME	TYPE	DESCRIPTION
Recovery Duration	Float	The length of time in between swimming strokes (will be scaled by strafe / reverse multipliers).
Recovery Speed Multiplier	Float	A multiplier applied to the speed in between strokes.
Recovery Acceleration Multiplier	Float	A multiplier applied to the acceleration (and deceleration) in between strokes.
Slow Input Time Scale	Float	At the minimum input amount, how much slower are strokes and recovery.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

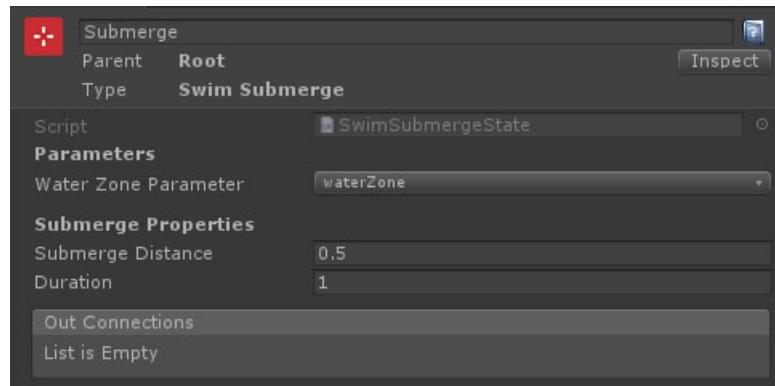
[Motion Graph States](#)

SwimSubmergeState MotionGraphState

Overview

The SwimSubmergeState state is used to transition from surface to underwater swimming while adapting to a moving water surface (eg waves).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone Parameter	TransformParameter	The transform parameter which contains the transform of the water zone object.
Submerge Distance	Float	The distance below the surface of the water to submerge.
Duration	Float	The time to take while submerging (will be instant if already below submerge distance).

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

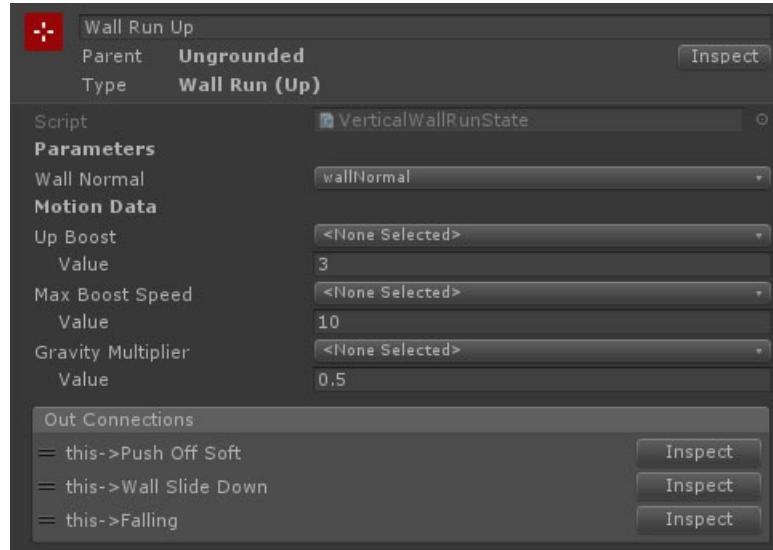
VerticalWallRun MotionGraphState

Overview

The VerticalWallRun state is used to make a character run up a wall when they run and jump into it. The state completes once the upward speed reaches zero.

The VerticalWallRun state is often paired with the [PushOff](#) state, using the wall normal to jump away from the wall.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Up Boost	FloatData	An upward speed boost applied when entering the state.
Max Boost Speed	FloatData	The upward speed can not be boosted above this value (though it can start higher than this).
Gravity Multiplier	FloatData	A multiplier that is used to reduce the effects of gravity when running up the wall.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

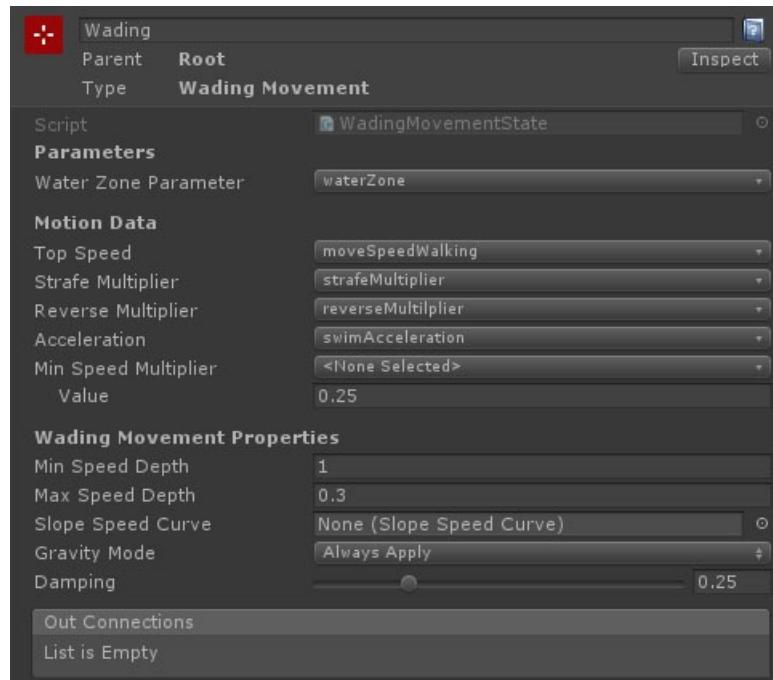
[Motion Graph States](#)

Wading MotionGraphState

Overview

The Wading state is a variation of the [Movement](#) state which slows the character down based on how much of their capsule is below the water line.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Name	FloatData	The multiplier for the standard movement speed when submerged to min speed depth.
Name	Float	The depth the character must be submerged to move at minimum speed.
Name	Float	The submersion depth of the character where their speed is not affected.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

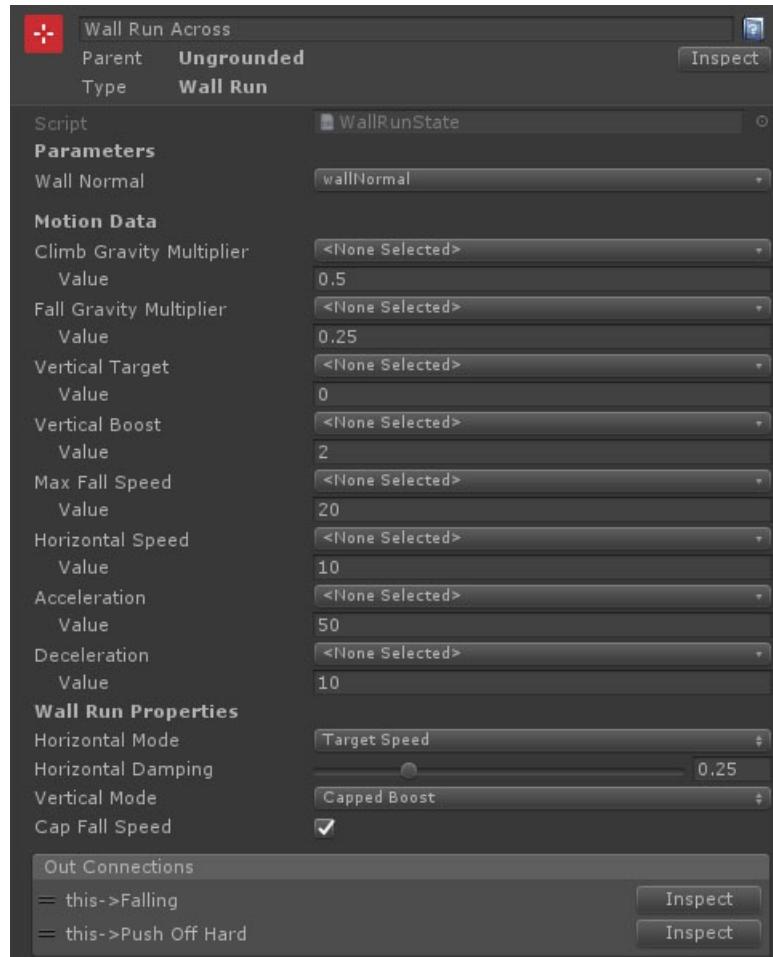
WallRun MotionGraphState

Overview

The WallRun state sticks the character to a wall and runs along it. It maintains the horizontal velocity and reduces gravity so that the character slowly moves down the wall. The state completes when the wall angles away from its initial plane, but you can also use conditions to transition out for things like downwards velocity over a certain threshold or input away from the wall.

The WallRun state is often paired with the [PushOff](#) state, using the wall normal to jump away from the wall.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Wall Normal	VectorParameter	The vector parameter containing the wall normal. This will be read and written to each frame.
Climb Gravity Multiplier	FloatData	A multiplier applied to gravity acceleration when moving up the wall.
Fall Gravity Multiplier	FloatData	A multiplier applied to gravity acceleration when moving down the wall.
Vertical Target	FloatData	The target vertical speed. Visible if Horizontal Mode is set to CappedBoost , Minimum or FixedSpeed .

NAME	TYPE	DESCRIPTION
Vertical Boost	FloatData	An upward speed boost when first entering the state. Visible if Horizontal Mode is set to Vertical Boost or CappedBoost .
Max Fall Speed	FloatData	The maximum downward speed the character can reach while wall running. This property is only visible if Cap Fall Speed is set to true .
Horizontal Speed	FloatData	The target horizontal speed. Only visible if Horizontal Mode is set to TargetSpeed or MinimumSpeed .
Acceleration	FloatData	The acceleration up to the target speed. Only visible if Horizontal Mode is set to TargetSpeed or MinimumSpeed .
Deceleration	FloatData	The deceleration down to the target speed. Only visible if Horizontal Mode is set to TargetSpeed .
Horizontal Mode	Dropdown	How the horizontal wall run speed is calculated. MaintainExisting keeps the horizontal speed from the previous frame, TargetSpeed accelerates / decelerates to a set horizontal speed, MinimumSpeed accelerates up to the minimum speed if falling below it, but does not decelerate if faster.
Horizontal Damping	Float	The amount of damping to apply when changing direction or speed. Only visible if Horizontal Mode is set to TargetSpeed or MinimumSpeed .
Vertical Mode	Dropdown	How the vertical speed is calculated when entering the wall run. VerticalBoost adds an upward speed boost, CappedBoost adds an upward boost up to a maximum vertical speed, Minimum raises the vertical speed up to the minimum, MaintainExisting uses the vertical entry speed with no changes, FixedSpeed sets the vertical speed to a specific value
Cap Fall Speed	Boolean	Should the downwards fall speed be limited .

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

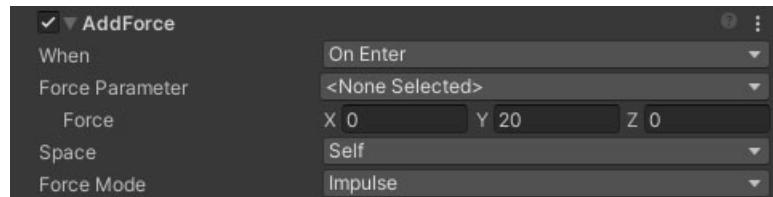
[Motion Graph States](#)

AddForce MotionGraphBehaviour

Overview

The AddForce behaviour adds a force to the character at specific points. This can have a variety of uses, but one key one is as an alternative way to handle jumping that doesn't require interrupting other states like the jump motion state does.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the force be added to the character. Options are OnEnter , OnExit and WhenTriggered .
Trigger Parameter	Trigger Parameter	A trigger that will be checked each frame, and the force applied if the trigger is set. This property is only visible if "When" is set to WhenTriggered .
Force Parameter	Vector Parameter	The force to apply to the character.
Force	Vector	The force to apply to the character. This property is shown if a "Force Parameter" has not been chosen.
Force Mode	ForceMode	How should the force be applied. Options are Force (takes mass and time into account), Impulse (takes mass into account), VelocityChange (mass and time are irrelevant), Acceleration (takes time into account).
Space	Dropdown	What coordinate space should the force be applied in. Options are World and Self .

See Also

[NeoCharacterController](#)

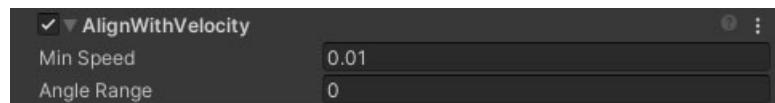
[Motion Graph Parameters And Data](#)

AlignWithVelocity MotionGraphBehaviour

Overview

The AlignWithVelocity behaviour is used with the steering system built into the aim controllers (eg the [MouseAndGamepadAimController](#) component) to lock the character's body direction based on its movement. This is useful for things like wall running or sliding when the character has a visible body, as you want the body to be aligned to the wall or travel direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character should be travelling before being aligned with the velocity.
Angle Range	Float	The angle range to constrain to.

See Also

[MouseAndGamepadAimController](#)

[Motion Graph Parameters And Data](#)

AnimatorGroundSlope MotionGraphBehaviour

Overview

The AnimatorGroundSlope behaviour outputs the current ground slope under the character, along with the down-slope direction relative to the player to [animator controller](#) parameters for use in blend trees.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Slope Param Name	String	The animator parameter name the ground slope angle should be written to. 0 is flat, 90 is vertical.
Direction Param Name	String	The animator parameter name the down-slope direction should be written to. 0 means that the ground slopes straight down in the direction the character is facing. 90 means the ground slopes down to the character's right, and -90 means to their left.

See Also

[NeoCharacterController](#)

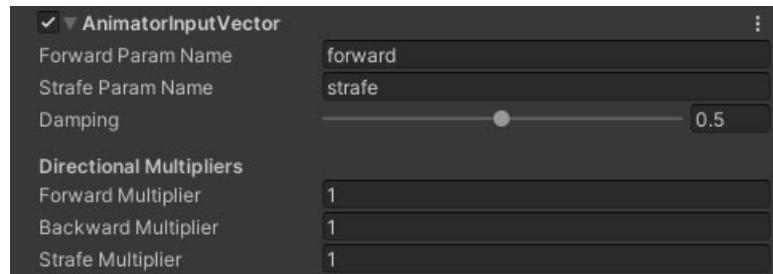
[Motion Graph Parameters And Data](#)

AnimatorInputVector MotionGraphBehaviour

Overview

The AnimatorInputVector behaviour outputs the current move input of the character to [animator controller](#) parameters for use in blend trees.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Forward Param Name	String	The animator parameter name the forward input value should be written to.
Strafe Param Name	String	The animator parameter name the strafe input value should be written to. (positive = right).
Damping	Float	Damping smooths out the input values.
Forward Multiplier	Float	A multiplier applied to the forward input before being sent to the animator parameter.
Backward Multiplier	Float	A multiplier applied to the backwards input before being sent to the animator parameter.
Strafe Multiplier	Float	A multiplier applied to the strafe input before being sent to the animator parameter.

See Also

[NeoCharacterController](#)

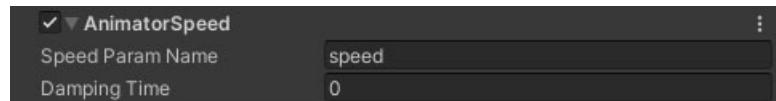
[Motion Graph Parameters And Data](#)

AnimatorSpeed MotionGraphBehaviour

Overview

The AnimatorSpeed behaviour outputs the current move speed of the character to an [animator controller](#) parameter for use in blend trees or clip speeds.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Speed Param Name	String	The animator parameter name the speed value should be written to.
Damping Time	Float	The duration used for smooth damping the velocity.

See Also

[NeoCharacterController](#)

[Motion Graph Parameters And Data](#)

AnimatorVelocity MotionGraphBehaviour

Overview

The AnimatorVelocity behaviour outputs the current velocity of the character (in local space) to [animator controller](#) parameters for use in blend trees.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Forward Param Name	String	The animator parameter name the forward input value should be written to.
Strafe Param Name	String	The animator parameter name the strafe input value should be written to. (positive = right).

See Also

[NeoCharacterController](#)

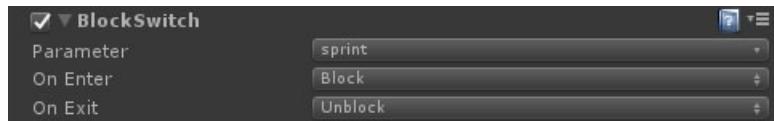
[Motion Graph Parameters And Data](#)

BlockSwitchParameter MotionGraphBehaviour

Overview

The BlockSwitchParameter behaviour will block the specified switch parameter, meaning that it can only return false for the duration of the block.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to modify.
OnEnter	Dropdown	Whether to block or unblock the parameter on entering the state. Options are Block , Unblock , Nothing .
OnExit	Dropdown	Whether to block or unblock the parameter on exiting the state. Options are Block , Unblock , Nothing .

See Also

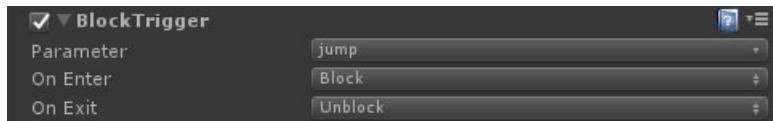
[Motion Graph Parameters And Data](#)

BlockTriggerParameter MotionGraphBehaviour

Overview

The BlockTriggerParameter behaviour will block the specified trigger parameter, meaning that it cannot fire. This is useful for situations like preventing jump triggers while crouched.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The property to modify.
OnEnter	Dropdown	Whether to block or unblock the parameter on entering the state. Options are Block , Unblock , Nothing .
OnExit	Dropdown	Whether to block or unblock the parameter on exiting the state. Options are Block , Unblock , Nothing .

See Also

[Motion Graph Parameters And Data](#)

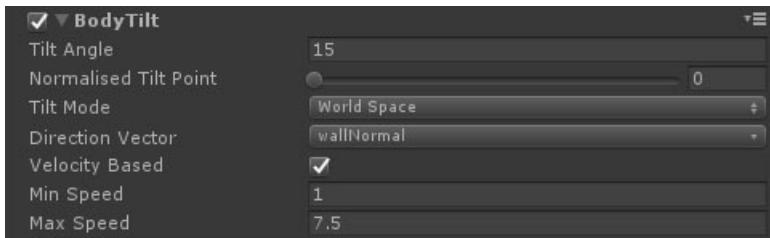
BodyTilt MotionGraphBehaviour

Overview

The BodyTilt behaviour tilts the character based on the specified tilt mode. Some example uses are:

- It is used in the parkour demo to tilt away from the wall by using the wall normal
- It is also used in the parkour demo to tilt back when crouch sliding by setting the mode to velocity, and the tilt angle to negative
- It is used to add a bob to the swimming in the swimming demo by setting the mode to velocity

Inspector



Properties

Name	Type	Description
Tilt Angle	Float	The angle to tilt.
Normalised Tilt Point	Float	The point on the character to tilt from. 0 is the bottom, 1 is the top.
Tilt Mode	Dropdown	How the tilt direction is calculated: <ul style="list-style-type: none">• Character Relative uses a vector parameter to specify a tilt vector relative to the character.• World Space does the same but in world space.• Velocity tilts in the direction of movement of the character.• Velocity Lateral tilts left or right based on the speed of the character in those directions.• Input tilts based on the player input and the direction the character is facing.• Input Lateral tilts left or right based on the player input.
Direction Vector	Vector Parameter	The direction to tilt in. This property is only shown if the tilt mode is set to Character Relative or World Space .
Velocity Based	Boolean	If set then the tilt angle will be based on the speed of the character.
Min Speed	Float	The speed below which the tilt angle will be 0.
Max Speed	Float	The speed above which the tilt will reach the full tilt angle specified above.

See Also

[NeoCharacterController](#)

[Motion Graph Parameters And Data](#)

CameraJiggleSpring MotionGraphBehaviour

Overview

The CameraJiggleSpring behaviour triggers the player character's camera jiggle spring additive effect with the supplied strength.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Switch Condition	SwitchParameter	An optional switch condition that defines if the jiggle should be triggered.
Trigger Condition	TriggerParameter	An optional trigger condition that defines if the jiggle should be triggered.
When	Dropdown	When should the camera jiggle spring be triggered.
Angle	Float	The strength of the jiggle effect (max angle is set in the Additive Jiggle component on the camera spring transform).
Angle	Float	Should the CW/CCW direction of the jiggle be chosen at random each time.

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

[Additive Transforms and Effects](#)

CameraKickSpring MotionGraphBehaviour

Overview

The CameraKickSpring behaviour triggers the player character's camera kick spring additive effect with the supplied offset and rotation.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Switch Condition	SwitchParameter	An optional switch condition that defines if the kick should be triggered.
Trigger Condition	TriggerParameter	An optional trigger condition that defines if the kick should be triggered.
When	Dropdown	When should the camera kick spring be triggered.
Position Kick	Vector3	The position offset for the camera at the strongest point of the kick. Keep these values small (cm, not meters) or you risk clipping scenery or weapon geometry.
Rotation Kick	Vector3	The rotation offset for the camera at the strongest point of the kick. Positive X nods forwards. Positive Y turns right. Positive Z tilts counter-clockwise.
Kick Duration	Float	The amount of time the kick effect should last.

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

[Additive Transforms and Effects](#)

CameraPulseFoV MotionGraphBehaviour

Overview

The CameraPulseFoV behaviour can apply a brief pulsed multiplier to the player camera's field of view. This is great for quick acceleration and impact effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the camera FoV pulse be triggered.
FovMultiplier	Float	The FoV multiplier to apply to the camera when the animation curve Y-axis is at 1.
PulseDuration	Float	The duration in seconds for the pulse to last.
PulseCurve	Animation Curve	A curve for the strength of the pulse. X is normalised time. Y = 0 means the FoV is 1x (no effect), Y = 1 means the FoV is the target FoV multiplier.

See Also

[MotionController](#)

[Additive Transforms and Effects](#)

CameraShake MotionGraphBehaviour

Overview

The CameraShake behaviour adds a constant shake to the camera whilst in the current state or sub-graph. You can also use the shake multiplier to fade in or out the effect over time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Shake Multiplier	FloatParameter	An optional float parameter to multiply the shake value by. This allows for increasing shake while falling, etc.
Shake Strength	Float	The strength of the shake effect (how this translates to camera movement is set in the CameraShake component on the character).

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

[Additive Transforms and Effects](#)

CameraShakeOneShot MotionGraphBehaviour

Overview

The CameraShakeOneShot behaviour triggers a one-off camera shake on entering or exiting the current state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Shake Multiplier	FloatParameter	An optional float parameter to multiply the shake value by. This allows for increasing shake while falling, etc.
Shake Strength	Float	The strength of the shake effect (how this translates to camera movement is set in the CameraShake component on the character).
Shake Duration	Float	How long should the shake last.
When	Dropdown	When should the camera shake. Options are OnEnter , OnExit and Both .

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

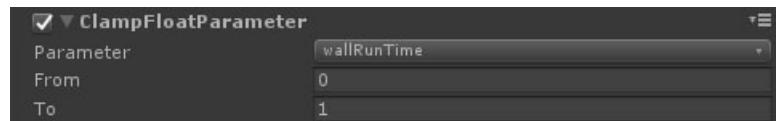
[Additive Transforms and Effects](#)

ClampFloat MotionGraphBehaviour

Overview

The ClampFloat behaviour can be used to clamp a [float parameter](#) within a specific range. The clamp is applied every tick.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to modify.
From	Float	The minimum parameter value.
To	Float	The maximum parameter value.

See Also

[Motion Graph Parameters And Data](#)

ClampInt MotionGraphBehaviour

Overview

The ClampInt behaviour can be used to clamp a [int parameter](#) within a specific range. The clamp is applied every tick.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to modify.
From	Float	The minimum parameter value.
To	Float	The maximum parameter value.

See Also

[Motion Graph Parameters And Data](#)

ConstrainCameraPitch MotionGraphBehaviour

Overview

The ConstrainCameraPitch behaviour is used to constrain the player character's camera to a specific pitch range direction while inside a motion state.

Inspector



Properties

Name	Type	Description
Min Pitch Constraint	Dropdown	What is the minimum pitch based on. Value is a specific angle from the horizontal plane. Parameter is an angle set in a float parameter. GroundSlope is an angle from the ground plane in the current direction.
Minimum Pitch	Float	The minimum angle the camera can look down. This property will only be visible if the constraint is set to Value .
Min Pitch Parameter	FloatParameter	A float parameter representing the minimum angle the camera can look down. This property will only be visible if the constraint is set to Parameter .
Min Ground Offset	Float	The angle from the ground plane (based on the current ground contact) that will be the minimum angle the camera can look down. This property will only be visible if the constraint is set to GroundSlope .
Max Pitch Constraint	Dropdown	What is the maximum pitch based on. Value is a specific angle from the horizontal plane. Parameter is an angle set in a float parameter. GroundSlope is an angle from the ground plane in the current direction.
Maximum Pitch	Float	The maximum angle the camera can look up. This property will only be visible if the constraint is set to Value .
Max Pitch Parameter	FloatParameter	A float parameter representing the maximum angle the camera can look down. This property will only be visible if the constraint is set to Parameter .
Max Horizon Offset	Float	The angle from the ground plane (based on the current ground contact) that will be the maximum angle the camera can look down. This property will only be visible if the constraint is set to GroundSlope .

See Also

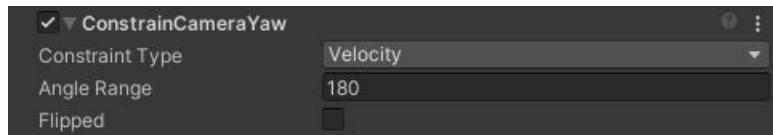
[MotionController](#)

ConstrainCameraYaw MotionGraphBehaviour

Overview

The ConstrainCameraYaw behaviour is used to constrain the player character's camera to a specific yaw direction while inside a motion state. An example is using the wall normal vector during a wall run to prevent the character turning into the wall (and to turn when running along curving walls).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Constraint Type	Dropdown	What to constrain the yaw to. The available options are: DirectionVector , TargetVector , TransformForward , TransformDirection , Velocity .
Direction	VectorParameter	The vector parameter to use as the constraint direction. This will only appear if the constraint type is set to DirectionVector .
Target	VectorParameter	The vector parameter containing the target point. This will only appear if the constraint type is set to TargetVector .
Transform	TransformParameter	The transform parameter to use as the constraint target. This will only appear if the constraint type is set to TransformForward or TransformDirection .
Angle Range	Float	The angle range to constrain to.
Flipped	Boolean	Flip the direction vector.
Continuous	Boolean	Should the constraints be updated each frame (if the vector parameter changes). This will not appear if the constraint type is set to Velocity as the constraint will always be continuous in that situation.

See Also

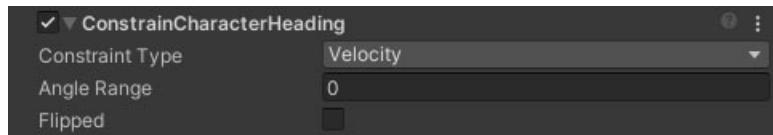
[MotionController](#)

ConstrainCharacterHeading MotionGraphBehaviour

Overview

The ConstrainCharacterHeading behaviour is used with the steering system built into the aim controllers (eg the [MouseAndGamepadAimController](#) component) to lock the character's body direction. They will still be able to look in any direction (you can use the [ConstrainCameraYaw](#) and [ConstrainCameraPitch](#) if you also want to constrain the aim), but the body will be locked to the set direction. This is useful for things like climbing ladders or mantling when the character has a visible body.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Constraint Type	Dropdown	What to constrain the heading to. The available options are: DirectionVector , TargetVector , TransformForward , TransformDirection , Velocity .
Direction	VectorParameter	The vector parameter to use as the constraint direction. This will only appear if the constraint type is set to DirectionVector .
Target	VectorParameter	The vector parameter containing the target point. This will only appear if the constraint type is set to TargetVector .
Transform	TransformParameter	The transform parameter to use as the constraint target. This will only appear if the constraint type is set to TransformForward or TransformDirection .
Angle Range	Float	The angle range to constrain to.
Flipped	Boolean	Flip the direction vector.
Continuous	Boolean	Should the constraints be updated each frame (if the vector parameter changes). This will not appear if the constraint type is set to Velocity as the constraint will always be continuous in that situation.

See Also

[MouseAndGamepadAimController](#)

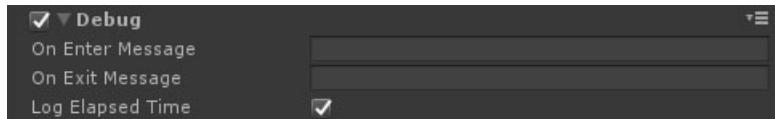
[Motion Graph Parameters And Data](#)

Debug MotionGraphBehaviour

Overview

The Debug behaviour can be used to help understand when the motion graph enters or exits a specific state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter Message	String	A message to print to the console on entering the state or sub-graph.
On Exit Message	String	A message to print to the console on exiting the state or sub-graph.
Log Elapsed Time	Boolean	If set, prints the time spent in the state or sub-graph to the console on exiting it.

DisableCollider MotionGraphBehaviour

Overview

The DisableCollider behaviour will completely disable the character's [CharacterController](#). This is useful in situations such as scripted or animated movements where environment collisions could interfere.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter	Dropdown	What to do to the character collider on entering the state. Options are: Enable , Disable , Nothing .
On Exit	Dropdown	What to do to the character collider on exiting the state. Options are: Enable , Disable , Nothing .

See Also

[Unity CharacterController](#)

DisableDamage MotionGraphBehaviour

Overview

The DisableDamage behaviour can be used to make the character invincible during specific movements.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter	Dropdown	What should happen to character damage on entering the state or sub-graph. Options are Disable , Enable and Ignore .
On Exit	Dropdown	What should happen to character damage on exiting the state or sub-graph. Options are Disable , Enable and Ignore .

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

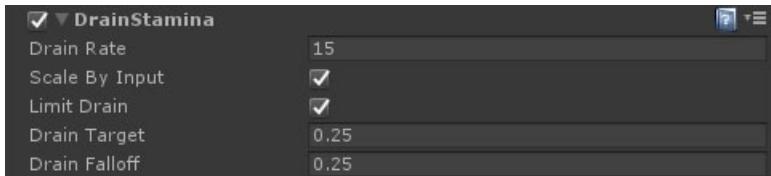
[Additive Transforms and Effects](#)

DrainStamina MotionGraphBehaviour

Overview

The DrainStamina behaviour applies a constant stamina drain to a [StaminaSystem](#) attached to the character.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Drain Rate	Float	The rate to drain the stamina at (bear in mind the stamina system also refreshes at a certain rate too, so these can cancel out).
Scale By Input	Boolean	Should the controller's move input scale also scale the stamina drain.
Limit Drain	Boolean	Is there a lower limit that the behaviour will drain stamina to before the drain rate falls off. The following properties will be exposed if this is true.
Drain Target	Float	The minimum level that the behaviour can drain stamina to.
Drain Falloff	Float	The stamina drain falls away to 0 as it approaches the target level, starting at this falloff value above it.

See Also

[StaminaSystem](#)

FootIk MotionGraphBehaviour

Overview

The FootIk behaviour enables or disables the foot IK on the character's body for the duration of this state or sub-graph. This requires that the character have a body with an [animator](#), as well as a [FirstPersonBody](#) component on that object.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter	Float	How the character's foot IK should be set on entering the state or sub-graph.
In/Out Time	Float	The time taken to blend in or out the IK on entry.
On Exit	Float	How the character's foot IK should be set on exiting the state or sub-graph.
In/Out Time	Float	The time taken to blend in or out the IK on exit.

See Also

[First Person Body](#)

[Animator](#)

FootstepAudio MotionGraphBehaviour

Overview

The FootstepAudio behaviour plays footsteps based on the character speed and the ground surface below them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Cast Direction	Dropdown	<p>The direction to perform the footstep surface check. Available options are:</p> <ul style="list-style-type: none">• Down casts downwards (based on the character up vector).• LocalVector casts based on the <i>Cast Vector</i> property in local space.• WorldVector casts based on the <i>Cast Vector</i> property in world space.• WorldParameter casts based on the vector value of the <i>Vector Parameter</i> property in world space.• WorldParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in world space.• LocalParameter casts based on the vector value of the <i>Vector Parameter</i> property in local space.• LocalParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Step Interval	Float	The interval between steps. Higher numbers mean the steps are further apart.
Minimum Speed	Float	The speed below which no footstep audio will be played.
Maximum Speed	Float	The maximum speed that the actual speed will be clamped to. Prevents rapid fire footsteps.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Ray Offset	Float	The vertical offset above the absolute bottom of the character collider to start the downward surface test.

See Also

[SurfaceAudioData](#)

Motion Graph Parameters And Data

ImpactDamage MotionGraphBehaviour

Overview

The ImpactDamage behaviour enables or disables character damage on impact when entering or exiting a state.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Fall Damage On Enter	Dropdown	Should fall damage be enabled, disabled or unchanged on entering the state.
Fall Damage On Exit	Dropdown	Should fall damage be enabled, disabled or unchanged on exiting the state.
Body Impact Damage On Enter	Dropdown	Should body impact damage be enabled, disabled or unchanged on entering the state.
Body Impact Damage On Exit	Dropdown	Should body impact damage be enabled, disabled or unchanged on exiting the state.
Head Impact Damage On Enter	Dropdown	Should head impact damage be enabled, disabled or unchanged on entering the state.
Head Impact Damage On Exit	Dropdown	Should head impact damage be enabled, disabled or unchanged on exiting the state.

See Also

[Motion Graph Parameters And Data](#)

[Health and Damage](#)

InvokeEvent MotionGraphBehaviour

Overview

The InvokeEvent behaviour will invoke the specified event on entering the state, on exiting or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	EventParameter	The event parameter to invoke.
When	Dropdown	When should event be invoked. Options are EnterAndExit , EnterOnly , ExitOnly .

See Also

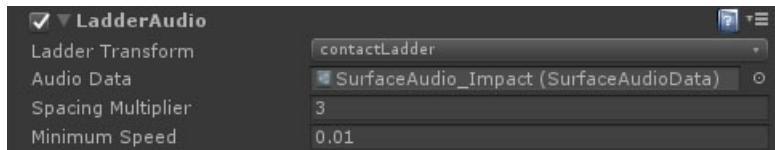
[Motion Graph Parameters And Data](#)

LadderAudio MotionGraphBehaviour

Overview

The LadderAudio behaviour is used to trigger audio cues when climbing a ladder. This works in a similar way to the footsteps system, but based on movement along the ladder's up axis.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Ladder Transform	TransformProperty	The transform property holding the ladder transform.
Spacing Multiplier	Float	How many rungs apart to play a sound. This is based on the ladder spacing property.
Minimum Speed	Float	The speed below which no audio will be played.

See Also

[Ladders](#)

[SurfaceAudioData](#)

LockInventorySelection MotionGraphBehaviour

Overview

The LockInventorySelection behaviour will lock the character's [inventory](#) selection to the specified item until unlocked by an [UnlockInventorySelection](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the inventory selection be set. Options are: OnEnter and OnExit .
What	Dropdown	What to lock the inventory selection to. Options are: Nothing , BackupItem and SlotIndex
Slot Index	Float	The quick slot to lock the selection to.

See Also

[Inventory](#)

[UnlockInventorySelection](#)

LoopingAudio MotionGraphBehaviour

Overview

The LoopingAudio behaviour plays a looping audio clip from the specified source.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Clip	[AudioClip][unity-audioclip]	The looping audio clip to play.
Source	FpsCharacter AudioSource	The source ID to play from (generated constant).
Pitch	Float	The pitch of the loop.

See Also

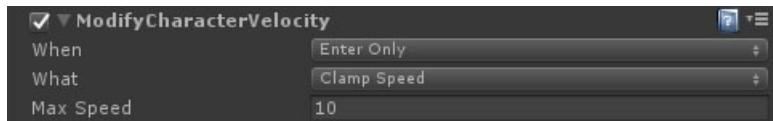
[Generated Constants](#)

ModifyCharacterVelocity MotionGraphBehaviour

Overview

The ModifyCharacterVelocity behaviour can change the character controller's velocity in various ways on entering or exiting the state.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the velocity be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	How should the velocity be modified. SetLocal sets the velocity relative to the character's current heading, SetWorld applies the new velocity in world space, ClampSpeed reduces the speed to the limit value if it is above it, and Multiply multiplies the current velocity by a value.
Local Velocity	Vector3	The target velocity of the character controller relative to its direction. Only visible if "What" is set to SetLocal .
World Velocity	Vector3	The target velocity of the character controller in world space. Only visible if "What" is set to SetWorld .
Max Speed	Float	The maximum speed the character can travel at. Only visible if "What" is set to ClampSpeed .
Multiplier	Float	A multiplier to apply to the character's velocity. Only visible if "What" is set to Multiply .

See Also

[NeoCharacterController](#)

[Motion Graph Parameters And Data](#)

ModifyFloatParameter MotionGraphBehaviour

Overview

The ModifyFloatParameter behaviour either sets or modifies the specified float parameter on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	How should the parameter be modified. Options are Set , Reset , Add , Subtract , Floor (round down to the nearest whole number) and Ceiling (round up to the nearest whole number).
Value	Float	The value to set to, add or subtract based on the What parameter.

See Also

[Motion Graph Parameters And Data](#)

ModifyIntParameter MotionGraphBehaviour

Overview

The ModifyIntParameter behaviour either sets or modifies the specified int parameter on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	How should the parameter be modified. Options are Set , Reset , Add , Subtract .
Value	Int	The value to set to, add or subtract based on the What parameter.

See Also

[Motion Graph Parameters And Data](#)

ModifyStamina MotionGraphBehaviour

Overview

The ModifyStamina behaviour performs an operation on the [StaminaSystem](#) attached to the character.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should stamina be modified. Options are: OnEnter , OnExit and Both .
What	Dropdown	What should the modification be. Options are: Increment , Decrement , IncrementNormalised , DecrementNormalised , SetToValue , SetToValueNormalised , SetToMax , SetToZero . The normalised options act on the stamina as a factor of max stamina.
Amount	Float	Value to use for modifying the stamina.

See Also

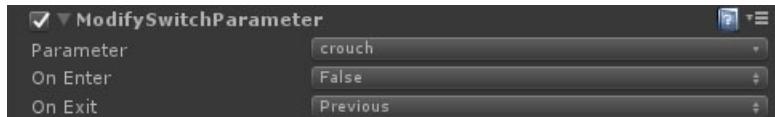
[StaminaSystem](#)

ModifySwitchParameter MotionGraphBehaviour

Overview

The ModifySwitchParameter behaviour sets the specified switch on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to set.
On Enter	Dropdown	How should the parameter be modified on entering the state. Options are Unchanged, True, False, Toggle, Reset, Previous .*
On Exit	Dropdown	How should the parameter be modified on exiting the state. Options are Unchanged, True, False, Toggle, Reset, Previous .*

* Previous is only valid on exit, and will set the switch to its state before entering.

See Also

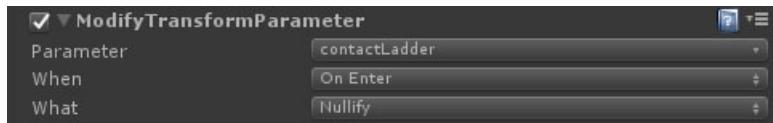
[Motion Graph Parameters And Data](#)

ModifyTransformParameter MotionGraphBehaviour

Overview

The ModifyTransformParameter behaviour either sets or resets the specified transform parameter on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	TransformParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	What is the action to do. Nullify clears the value and Find will set the value using GameObject.Find .

See Also

[Motion Graph Parameters And Data](#)

[GameObject.Find](#)

ModifyTriggerParameter MotionGraphBehaviour

Overview

The ModifyTriggerParameter behaviour either sets or resets the specified trigger on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	What is the action to do. Options are Set and Reset .

See Also

[Motion Graph Parameters And Data](#)

ModifyVectorParameter MotionGraphBehaviour

Overview

The ModifyVectorParameter behaviour either sets or modifies the specified vector parameter on entering the state, on exit or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are EnterAndExit , EnterOnly , ExitOnly .
What	Dropdown	How should the parameter be modified. The available options are: <ul style="list-style-type: none">• Set will set the vector to the specified value.• Reset resets the vector to its starting value.• Add adds the specified vector to the parameter.• Subtract subtracts the specified vector from the parameter.• Multiply multiplies the parameter by the specified multiplier.• Normalize normalizes the parameter vector to magnitude. 1

- **Flatten** flattens the parameter vector onto the horizontal plane of the character
- **ClampMagnitude** clamps the magnitude to a specified maximum length || Value | Vector3 | The value to set to, add or subtract based on the **What** property. || Multiplier | Float | A multiplier to apply to the vector if the **What** property is set to **Multiply**. || Clamp | Float | The magnitude to clamp the vector to if the **What** property is set to **ClampMagnitude**. |

See Also

[Motion Graph Parameters And Data](#)

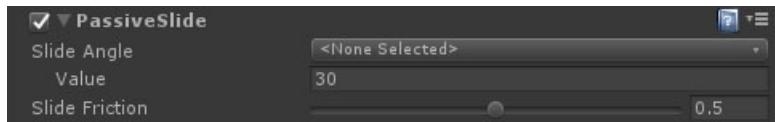
PassiveSlide MotionGraphBehaviour

Overview

The PassiveSlide behaviour lowers ground friction when the character is on a steep enough slope, and is not attempting to move. If the motion controller receives move input then the [NeoCharacterController](#) slope friction will be set to its value on entering the state or sub-graph. If not, then the slope friction will be set to **Slide Friction** setting as long as this is lower than the friction already set and the angle is steep enough.

By default, the character must be stood on a slope steeper than 30 degrees in order to slide. If the **Slope Data Key** corresponds to a valid [Slope MotionControllerDataEntry](#) then the slide will be triggered when the angle is greater than its **Slope Slide Angle** property.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Slope Angle	Float Data	A slope angle that chooses when to start sliding. This can either be set as a value, or by referencing a float data entry .
Slide Friction	Float	The slope friction to apply when sliding.

See Also

[NeoCharacterController](#)

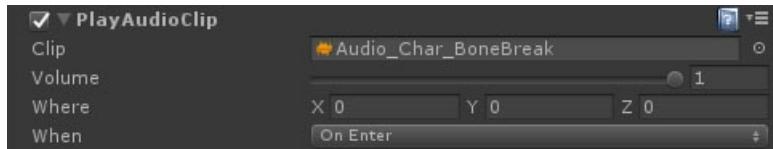
[Motion Graph Parameters And Data](#)

PlayAudioClip MotionGraphBehaviour

Overview

The PlayAudioClip behaviour plays an audio clip on entering the state, on exiting, or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Clip	AudioClip	The audio clip to play.
Volume	Float	The volume to play the clip at.
Where	Vector3	The offset from the character controller transform position to play the clip at.
When	Dropdown	When should the clip be played. Options are EnterAndExit , EnterOnly , ExitOnly .

See Also

[Unity AudioClip](#)

PlayCharacterAudio MotionGraphBehaviour

Overview

The PlayCharacterAudio behaviour will play an audio clip on entering the state, exiting the state, or both.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio	FpsCharacterAudio	The audio ID to play (generated constant).
When	Dropdown	When should the audio be played. Options are EnterAndExit , EnterOnly , ExitOnly .

See Also

[Generated Constants](#)

RecordVelocity MotionGraphBehaviour

Overview

The RecordVelocity behaviour outputs the current velocity of the character to a vector parameter on the graph for use in other conditions or states.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	Vector Parameter	The parameter to modify.
When	Dropdown	At what point should the velocity be recorded. Options are OnEnter , OnExit and Always . The latter records the velocity each frame while in the state or subgraph it's attached to.

See Also

[NeoCharacterController](#)

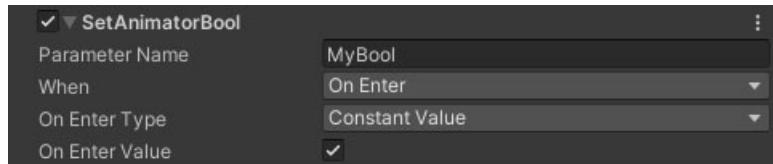
[Motion Graph Parameters And Data](#)

SetAnimatorBool MotionGraphBehaviour

Overview

The SetAnimatorBool behaviour writes to an [animator controller](#) parameter on entering or exiting the state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Type	Dropdown	What type of value to set the animator parameter to on entering the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Enter Value	Boolean	The value to write to the parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to ConstantValue .
On Enter Parameter	SwitchParameter	The parameter that contains the value to write to the animator parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .
On Exit Type	Dropdown	What type of value to set the animator parameter to on exiting the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Exit Value	Boolean	The value to write to the parameter on exiting the state / subgraph. *This property will only be visible if On Exit Type is set to ConstantValue .
On Exit Parameter	SwitchParameter	The parameter that contains the value to write to the animator parameter on exiting the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

SetAnimatorFloat MotionGraphBehaviour

Overview

The SetAnimatorFloat behaviour writes to an [animator controller](#) parameter on entering or exiting the state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Type	Dropdown	What type of value to set the animator parameter to on entering the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Enter Value	Boolean	The value to write to the parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to ConstantValue .
On Enter Parameter	FloatParameter	The parameter that contains the value to write to the animator parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .
On Exit Type	Dropdown	What type of value to set the animator parameter to on exiting the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Exit Value	Boolean	The value to write to the parameter on exiting the state / subgraph. *This property will only be visible if On Exit Type is set to ConstantValue .
On Exit Parameter	FloatParameter	The parameter that contains the value to write to the animator parameter on exiting the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

SetAnimatorInt MotionGraphBehaviour

Overview

The SetAnimatorInt behaviour writes to an [animator controller](#) parameter on entering or exiting the state or sub-graph.

Inspector



Properties

NAME	TYPe	DESCRIPTION
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Type	Dropdown	What type of value to set the animator parameter to on entering the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Enter Value	Boolean	The value to write to the parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to ConstantValue .
On Enter Parameter	IntParameter	The parameter that contains the value to write to the animator parameter on entering the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .
On Exit Type	Dropdown	What type of value to set the animator parameter to on exiting the state / subgraph. Options are ConstantValue and MotionGraphParameter .
On Exit Value	Boolean	The value to write to the parameter on exiting the state / subgraph. *This property will only be visible if On Exit Type is set to ConstantValue .
On Exit Parameter	IntParameter	The parameter that contains the value to write to the animator parameter on exiting the state / subgraph. *This property will only be visible if On Enter Type is set to MotionGraphParameter .

See Also

[MotionController](#)

[Motion Graph Parameters And Data](#)

SetAnimatorBool MotionGraphBehaviour

Overview

The SetAnimatorTrigger behaviour writes to an [animator controller](#) parameter on entering or exiting the state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Name	String	The name of the animator parameter to write to.
On Enter	Dropdown	The action to perform on entering the state / subgraph.
On Exit	Dropdown	The action to perform on exiting the state / subgraph.

See Also

[MotionController](#)

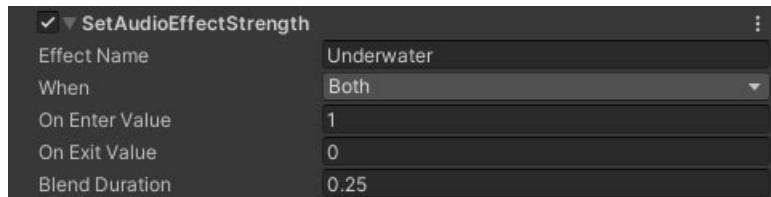
[Motion Graph Parameters And Data](#)

SetAudioEffectStrength MotionGraphBehaviour

Overview

The SetAudioEffectStrength behaviour connects to an [FpsCharacterAudioEffects](#) behaviour on the character's camera object. It will set the strength of one of the effects presets on entering or exiting the state or sub-graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Effect Name	String	The name of the audio effect on the character camera's FpsCharacterAudioEffects component.
When	Dropdown	When should the parameter be modified (OnEnter , OnExit or Both).
On Enter Value	Float	The strength the effect should be set to on entering the state / subgraph.
On Exit Value	Float	The strength the effect should be set to on exiting the state / subgraph.
Blend Duration	Float	The time taken to blend from the current strength to the target strength.

See Also

[MotionController](#)

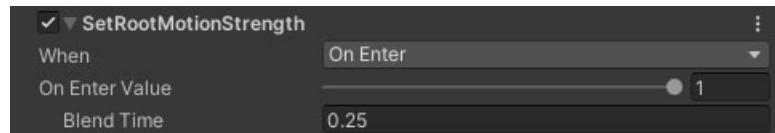
[Motion Graph Parameters And Data](#)

SetRootMotionStrength MotionGraphBehaviour

Overview

The SetRootMotionStrength behaviour is used when the character has a [first person body](#) and will move the character based on the root motion of its animations based on the strength you set. To use this, the character must have a body with an animator. It must have animations that use root motion. lastly, it must have a [FirstPersonBodyRootMotion](#) component attached to the character body which catches root motion and sends it to the character's [MotionController](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the parameter be modified. Options are OnEnter , OnExit and Both .
On Enter Value	Float	The root motion strength to set on entering this state / subgraph.
Blend Time	Float	The blend time for the new root motion value on entry.
On Exit Value	Float	The root motion strength to set on exiting this state / subgraph.
Blend Time	Float	The blend time for the new root motion value on exit.

See Also

[First Person Body](#)

[MotionController](#)

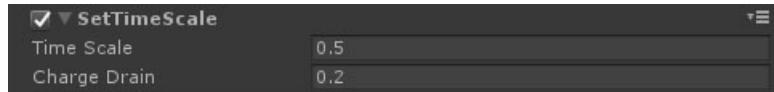
[FirstPersonBodyRootMotion](#)

SetSteering MotionGraphBehaviour

Overview

The [NeoCharacterController](#) that NeoFPS is based on has a steering system that can decouple the aim direction from the character's move direction. The SetSteering behaviour is used set a speed at which the body rotates to match the aim direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter	Dropdown	What to do to the steering rate on entering the state.
On Exit	Dropdown	What to do to the steering rate on exiting the state.
Entry Value	Float	The value to set the steering rate to on entering the state.
Exit Value	Float	The value to set the steering rate to on exiting the state.

See Also

[NeoCharacterController](#)

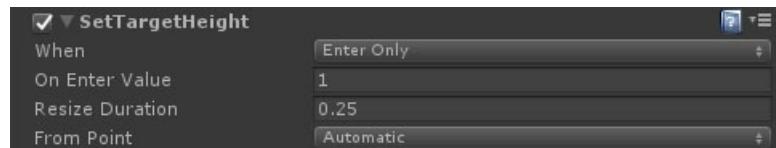
[MotionController](#)

SetTargetHeight MotionGraphBehaviour

Overview

The SetTargetHeight behaviour signals the desired character height multiplier for the controller to lerp to.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the target height be set. Options are EnterAndExit , EnterOnly , ExitOnly .
On Enter Value	Float	The character height multiplier (standing height) to set on entering this state (if <i>when</i> is set to EnterandExit or EnterOnly).
On Exit Value	Float	The character height multiplier (standing height) to set on exiting this state (if <i>when</i> is set to EnterandExit or ExitOnly).
Resize Duration	Float	The time taken to change heights.
From Point	Dropdown	Where is the character resized from. Available options are: <ul style="list-style-type: none">• Automatic will scale from the bottom if grounded and the top if airborne and crouch jumping is enabled.• Bottom always resizes from the bottom of the character.• Top always resizes from the top of the character.

See Also

[MotionController](#)

SetTimeScale MotionGraphBehaviour

Overview

The SetTimeScale behaviour is used to trigger slow motion effects when entering a motion state.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The target timescale to set. Will be reset to 1 on exit.
Charge Drain	Float	The amount of charge drained per (real, unscaled) second.

See Also

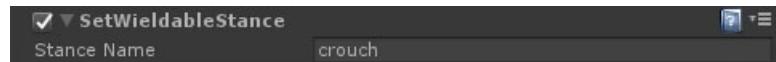
[MotionController](#)

SetWieldableStance MotionGraphBehaviour

Overview

The SetWieldableStance behaviour is used to choose the pose that a wieldable [inventory](#) item such as a firearm or melee weapon is held in. This can be used to telegraph when the character is crouching or falling, for example.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Stance Name	String	The name of the stance to use. The wieldable item needs a WieldableStanceManager component, with a stance that has this name.

See Also

[Inventory](#)

SlidingAudio MotionGraphBehaviour

Overview

The SlidingAudio behaviour plays a looping audio clip based on the ground contact surface, pitch shifted based on the character speed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Surface Test Interval	Int	Every n-th frame, the behaviour will check what the ground surface is and switch the sliding audio if required.
Minimum Speed	Float	The speed below which the pitch will be at its minimum.
Maximum Speed	Float	The speed above which the pitch will be at its maximum.
Minimum Pitch	Float	The minimum pitch for the slide loop.
Maximum Pitch	Float	The maximum pitch for the slide loop.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Ray Offset	Float	The vertical offset above the absolute bottom of the character collider to start the downward surface test.

See Also

[SurfaceAudioData](#)

SpineAim MotionGraphBehaviour

Overview

The SpineAim behaviour connects to a character's [ProceduralSpineAimMatcher](#) component and enables or disables the spine rotation offsets that are used to match the aim direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Enter	Float	How the character's spine aim matching should be set on entering the state or sub-graph.
In/Out Time	Float	The time taken to blend in or out the spine rotations on entry.
On Exit	Float	How the character's spine aim matching should be set on exiting the state or sub-graph.
In/Out Time	Float	The time taken to blend in or out the spine rotations on exit.

See Also

[First Person Body](#)

[ProceduralSpineAimMatcher](#)

SurfaceAudio MotionGraphBehaviour

Overview

The SurfaceAudio behaviour plays an audio clip based on the current ground surface.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the audio clips.
When	Dropdown	When should the audio be played. Options are EnterAndExit , EnterOnly , ExitOnly .
Cast Direction	Dropdown	<p>The direction to perform the footstep surface check. Available options are:</p> <ul style="list-style-type: none">• Down casts downwards (based on the character up vector).• LocalVector casts based on the <i>Cast Vector</i> property in local space.• WorldVector casts based on the <i>Cast Vector</i> property in world space.• WorldParameter casts based on the vector value of the <i>Vector Parameter</i> property in world space.• WorldParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in world space.• LocalParameter casts based on the vector value of the <i>Vector Parameter</i> property in local space.• LocalParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Max Ray Distance	Float	The downward raycast length for the ground surface test.

See Also

[Surfaces](#)

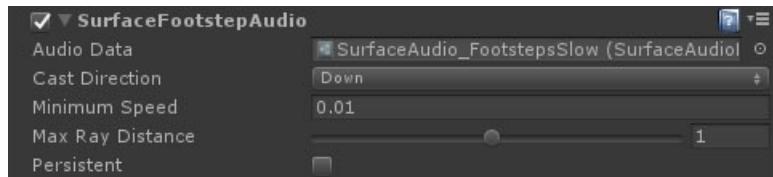
[Audio Systems](#)

SurfaceFootstepAudio MotionGraphBehaviour

Overview

The SurfaceFootstepAudio behaviour interacts with the [SurfaceFootstepAudioSystem][3] monobehaviour on the root of the character to control how it processes footsteps.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Cast Direction	Dropdown	<p>The direction to perform the footstep surface check. Available options are:</p> <ul style="list-style-type: none">• Down casts downwards (based on the character up vector).• LocalVector casts based on the <i>Cast Vector</i> property in local space.• WorldVector casts based on the <i>Cast Vector</i> property in world space.• WorldParameter casts based on the vector value of the <i>Vector Parameter</i> property in world space.• WorldParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in world space.• LocalParameter casts based on the vector value of the <i>Vector Parameter</i> property in local space.• LocalParameterInverse casts based on the flipped vector value of the <i>Vector Parameter</i> property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Transform Parameter	TransformParameter	A parameter containing the transform to use for the cast's space. If none is selected or the parameter value is null then the cast is performed in world space.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Minimum Speed	Float	The speed below which no footstep audio will be played.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Persistent	Boolean	If persistent is true, then exiting this state or sub-graph will keep the footstep settings until they are explicitly set from elsewhere.

See Also

[SurfaceFootstepAudioSystem][3]

[SurfaceAudioData](#)

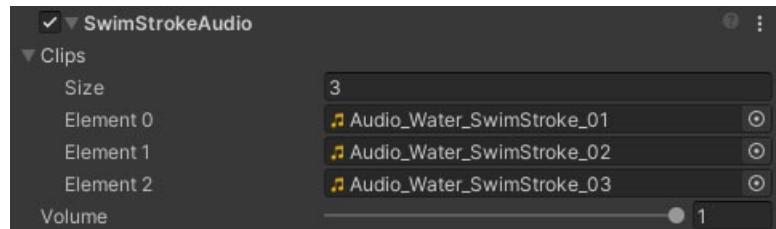
Motion Graph Parameters And Data

SwimStrokeAudio MotionGraphBehaviour

Overview

The SwimStrokeAudio behaviour must be added to one of the stroke based swimming motion graph states. It will subscribe to the state's events and play a swimming audio clip each stroke as the character moves.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Clips	AudioClip	A selection of audio clips to play. One will be chosen at random each time.
Volume	Float	The volume to play the clip at.

See Also

[MotionController](#)

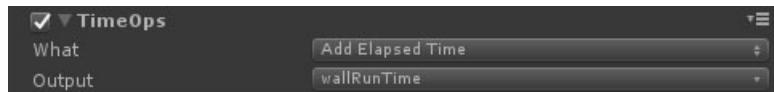
[Motion Graph Parameters And Data](#)

TimeOps MotionGraphBehaviour

Overview

The TimeOps behaviour modifies a float parameter on the graph based on the Time. It can be used to record the time that specific events occur or to track the time spent in one or more states or sub-graphs.

Inspector



Properties

NAME	TYPE	DESCRIPTION
What	Dropdown	<p>How should the parameter be modified. Available options are:</p> <ul style="list-style-type: none">• Add Elapsed Time adds elapsed time to the output parameter.• Add Elapsed Time Scaled adds elapsed time, multiplier by a scale factor to the output parameter.• Record Entry Time sets the output parameter to the current time on entering the state or sub-graph.• Record Exit Time sets the output parameter to the current time on exiting the state or sub-graph.• Record Time sets the output parameter to the current time every frame.
Output	FloatParameter	The parameter to modify.
Multiplier	Float	A multiplier to apply to the output if What is set to Add Elapsed Time Scaled .

See Also

[Motion Graph Parameters And Data](#)

TrackStepsBehaviour MotionGraphBehaviour

Overview

The TrackStepsBehaviour behaviour is used to provide a consistent step count for use by the various bob effects such as [PositionBob](#) and [RotationBob](#). Stride length will be reset to the previous value on exiting the behaviour, so these behaviours can be nested.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Stride Length	Float	The travel distance for one stride.
Maximum Rate	Float	The maximum number of steps per second.

See Also

[PositionBob](#)

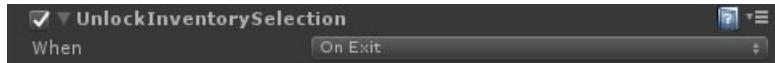
[RotationBob](#)

UnlockInventorySelection MotionGraphBehaviour

Overview

The UnlockInventorySelection behaviour will unlock the character's [inventory](#) selection after being locked by a [LockInventorySelection](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the inventory selection be set. Options are: OnEnter and OnExit .

See Also

[Inventory](#)

[LockInventorySelection](#)

AirTime MotionGraphCondition

Overview

The AirTime condition checks against the airtime value of the character's [NeoCharacterController](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type between the airtime and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

[NeoCharacterController](#)

CapsuleCast MotionGraphCondition

Overview

The CapsuleCast condition performs a cast of the [NeoCharacterController](#) capsule in the specified direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

See Also

[Layers And Tags](#)

CapsuleLookahead MotionGraphCondition

Overview

The CapsuleLookahead condition performs a cast of the [CharacterController](#) capsule based on either its movement, the direction it's facing, or the input direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

See Also

[Layers And Tags](#)

CharacterHeight MotionGraphCondition

Overview

The CharacterHeight condition checks the height of the character capsule, or its multiplier compared to its standing height.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Compare	Dropdown	What to check. Available options are Multiplier and Actual Height .
Comparison	Dropdown	The comparison type between the character height and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

[NeoCharacterController](#)

Climbable MotionGraphCondition

Overview

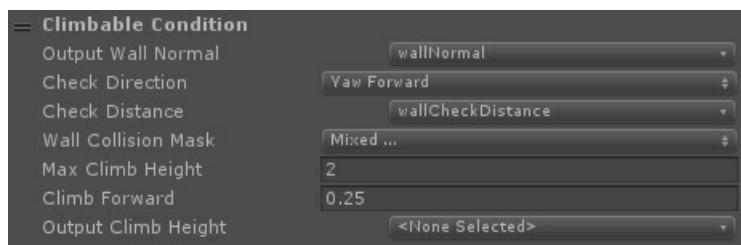
The Climbable condition performs a complex condition which checks if a character can climb onto a ledge. It has a number of steps:

1. It capsule casts to find the wall surface to climb
2. It sphere casts up to check available head height
3. It casts forward into the wall plane at either the maximum climb height or the point the head cast hit in order to check that the space is clear to climb

The condition outputs the wall normal found in step 1 to a [Vector Parameter](#) which can then be used by the [Mantle state](#) to actually perform the climbing action. The mantle state performs its own checks each frame so you do not need to keep testing if a wall is climbable once the mantle state has control.

Since the climbable condition uses a number of complex checks, it is worth putting any cheaper conditions ahead of it in the transition in order to narrow down the situations where it will be checked.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Output Wall Normal	Vector Parameter	The vector parameter used to store the wall normal.
Check Direction	Dropdown	The direction to perform the initial wall check in Yaw Forward checks straight ahead of the character. Inverse Wall Normal reads the value of the wall normal parameter above and flips it, casting back into the wall.
Check Distance	Float Data	The distance of the initial capsule cast to check wall contact. It is advised to set up a motion data entry and share it with the Mantle state to ensure they have the same value.
Wall Collision Mask	Layermask	The layers to check against.
Max Climb Height	Float	The maximum height the character can pull itself up to reach the top surface.
Climb Forward	Float	After reaching the top surface, this is the distance to move in past the edge before the state completes.
Output Climb Height	Float Parameter	An optional parameter to store the climb height.

See Also

[Motion Graph Parameters And Data](#)

[Layers And Tags](#)

CollisionFlags MotionGraphCondition

Overview

The CollisionFlags condition checks the [NeoCharacterController](#) collision flags that resulted from the last movement frame.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	Check if the NeoCharacterController collision flags Include or Exclude the "Compare To" flags.
Compare To	NeoCharacterControllerHit	The collision flags to check against.

Note

If you want to check against individual sides, then use the **Mask** values. Using the non-mask versions also sets the **Sides** flag which is shared between all of them.

See Also

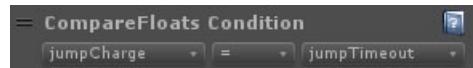
[NeoCharacterController](#)

CompareFloats CapsuleCastCondition

Overview

The CompareFloats condition compares two float parameters.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter A	FloatParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are Equal To (=) , Not Equal To (!=) , Greater Than (>) , Greater Or Equal To (>=) , Less Than (<) , Less Than Or Equal To (<=) .
Parameter B	FloatParameter	The right hand side parameter of the comparison

See Also

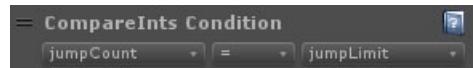
[Motion Graph Parameters And Data](#)

CompareInts CapsuleCastCondition

Overview

The CompareInts condition compares two int parameters.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter A	IntParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are Equal To (=) , Not Equal To (!=) , Greater Than (>) , Greater Or Equal To (>=) , Less Than (<) , Less Than Or Equal To (<=) .
Parameter B	IntParameter	The right hand side parameter of the comparison

See Also

[Motion Graph Parameters And Data](#)

CompareSwitches CapsuleCastCondition

Overview

The CompareSwitches condition compares two switch parameters.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter A	SwitchParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are Equal To (=) , Not Equal To (!=) .
Parameter B	SwitchParameter	The right hand side parameter of the comparison

See Also

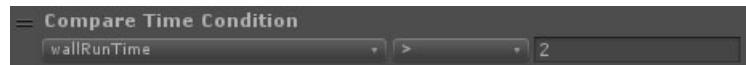
[Motion Graph Parameters And Data](#)

CompareTime MotionGraphCondition

Overview

The CompareTime condition is used in conjunction with the [TimeOps](#) motion graph behaviour to check the current time against the parameter values it outputs.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Parameter	FloatParameter	The float parameter with the time stored in it byt TimeOps .
Comparison	Dropdown	The comparison type between the parameter and value. Options are GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

[Motion Graph Parameters And Data](#)

[TimeOps MotionGraphBehaviour](#)

Completed MotionGraphCondition

Overview

The Completed condition checks the current state's completed flag. If the state has completed, the condition is true.

Inspector



Properties

The Completed condition has no properties exposed in the motion graph inspector.

See Also

ConditionGroup MotionGraphCondition

Overview

The ConditionGroup condition is used to group conditions together to allow for more complex rules than any and all. For example, when swimming on the surface of a water zone, you might transition to the underwater state if you **either** press crouch **or** look down **and** press forwards.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Condition Group	Dropdown	A dropdown that lists the available groups on the connection or allows you to create a new one.

Note

The result of a condition group is recorded when it is evaluated, so you will not be able to create infinite loops where group A checks group B which checks group A again.

See Also

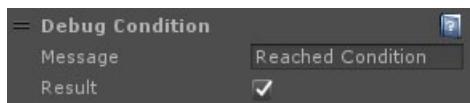
[Motion Graph Conditions](#)

Debug MotionGraphCondition

Overview

The Debug condition is only used to check the float of a motion graph. Add it to connections that are not behaving as expected, before and after the problem condition with a message to help check when the condition is hit. Do not leave debug conditions in a complete motion graph.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Message	String	A message to send to the debug console when this condition is checked.
Result	Boolean	The result the condition should give.

See Also

Direction MotionGraphCondition

Overview

The Direction condition checks the specified [vector parameter](#) against the character's direction

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The vector parameter to check against.
Compare	Dropdown	<p>What direction to compare against the parameter. Available options are:</p> <ul style="list-style-type: none">• YawVsVector checks the character's yaw forward direction against the parameter.• YawVsHorizontal checks the character's yaw forward direction against the parameter aligned to the character's horizontal plane.• AimVsVector checks the character's aim direction against the parameter.• VelocityVsVector checks the character's move direction against the parameter.• VelocityVsHorizontal checks the character's movement direction against the horizontal aligned parameter.• InputVsHorizontal checks the character's input direction (aligned to the character) against the horizontal aligned parameter.
Comparison	Dropdown	The comparison type between the parameter and value. This can be less than "<" or greater than ">".
Angle	Float	The angle value to check against.

See Also

[NeoCharacterController](#)

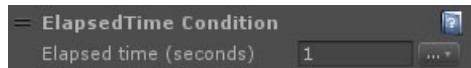
[Motion Graph Parameters And Data](#)

ElapsedTime MotionGraphCondition

Overview

The Elapsed Time condition checks how long the motion graph has been in the current state.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timeout Value	Float	The time after which the condition will be valid. If the parameter is set, this will be ignored.
Timeout Property	FloatProperty	An optional parameter containing a value for the time after which the condition will be valid. Selecting a parameter will remove the value field until this parameter is removed again.

See Also

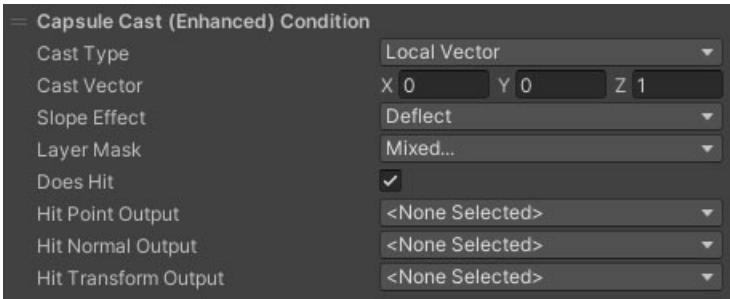
[Motion Graph Parameters And Data](#)

EnhancedCapsuleCast MotionGraphCondition

Overview

The EnhancedCapsuleCast condition performs a cast of the [NeoCharacterController](#) capsule in the specified direction and then outputs the results to [motion graph parameters](#) for use in other conditions and states.

Inspector



Properties

Name	Type	Description
Cast Type	Dropdown	<p>What to use for the cast vector. Available options are:</p> <ul style="list-style-type: none">• LocalVector uses a preset vector relative to the character. Magnitude is distance.• WorldVector uses a preset vector in world space. Magnitude is distance.• LocalParameter uses a vector parameter to get the direction relative to the character and casts a set distance.• LocalParameterInverse uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance.• WorldParameter uses a vector parameter to get the direction in world space and casts a set distance.• WorldParameterInverse uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to World Vector or Local Vector .
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the Parameter settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the Parameter settings.
Slope Effect	Dropdown	How does the cast vector react to slopes. Deflect will deflect the cast vector up if it intersects with the ground plane. Project will project the vector onto the ground slope from above regardless whether the vector collides with the slope. Both only work if the character is grounded. None ignores the ground slope.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.

NAME	TYPE	DESCRIPTION
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

[Layers And Tags](#)

EnhancedCapsuleLookahead MotionGraphCondition

Overview

The EnhancedCapsuleLookahead condition performs a cast of the [NeoCharacterController](#) capsule based on either its movement, the direction it's facing, or the input direction. The cast results are output to [motion graph parameters](#) for later use.

Inspector



Properties

Name	Type	Description
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

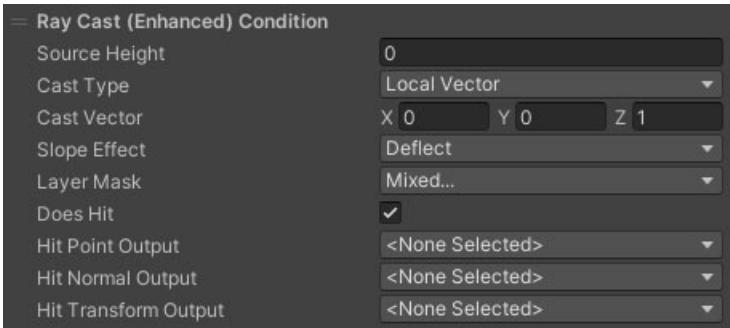
[Layers And Tags](#)

EnhancedRayCast MotionGraphCondition

Overview

The EnhancedRayCast condition performs a raycast from the specified point on the character. The cast results are output to [motion graph parameters](#) for later use.

Inspector



Properties

Name	Type	Description
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Type	Dropdown	<p>What to use for the cast vector. Available options are:</p> <ul style="list-style-type: none">• LocalVector uses a preset vector relative to the character. Magnitude is distance.• WorldVector uses a preset vector in world space. Magnitude is distance.• LocalParameter uses a vector parameter to get the direction relative to the character and casts a set distance.• LocalParameterInverse uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance.• WorldParameter uses a vector parameter to get the direction in world space and casts a set distance.• WorldParameterInverse uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to World Vector or Local Vector .
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the Parameter settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the Parameter settings.
Slope Effect	Dropdown	How does the cast vector react to slopes. Deflect will deflect the cast vector up if it intersects with the ground plane. Project will project the vector onto the ground slope from above regardless whether the vector collides with the slope. Both only work if the character is grounded. None ignores the ground slope.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

NAME	TYPE	DESCRIPTION
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

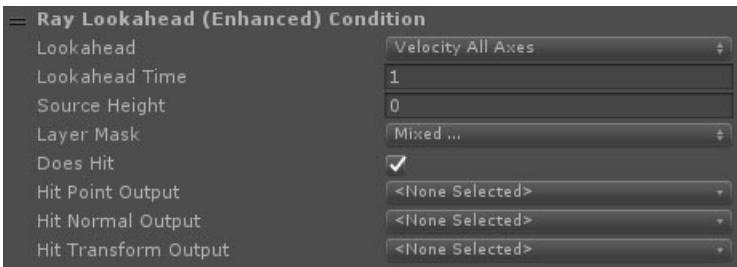
[Layers And Tags](#)

EnhancedRayLookahead MotionGraphCondition

Overview

The EnhancedRayLookahead condition performs a raycast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction. The cast results are output to [motion graph parameters](#) for later use.

Inspector



Properties

Name	Type	Description
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

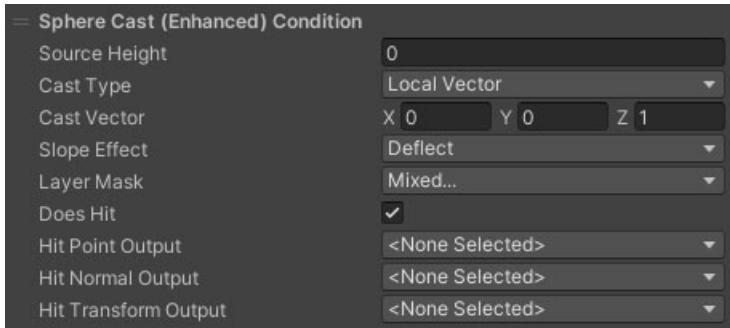
Layers And Tags

EnhancedSphereCast MotionGraphCondition

Overview

The EnhancedSphereCast condition performs a spherecast from the specified point on the character. The cast results are output to [motion graph parameters](#) for later use.

Inspector



Properties

Name	Type	Description
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Type	Dropdown	What to use for the cast vector. Available options are: <ul style="list-style-type: none">• LocalVector uses a preset vector relative to the character. Magnitude is distance.• WorldVector uses a preset vector in world space. Magnitude is distance.• LocalParameter uses a vector parameter to get the direction relative to the character and casts a set distance.• LocalParameterInverse uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance.• WorldParameter uses a vector parameter to get the direction in world space and casts a set distance.• WorldParameterInverse uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to World Vector or Local Vector .
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the Parameter settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the Parameter settings.
Slope Effect	Dropdown	How does the cast vector react to slopes. Deflect will deflect the cast vector up if it intersects with the ground plane. Project will project the vector onto the ground slope from above regardless whether the vector collides with the slope. Both only work if the character is grounded. None ignores the ground slope.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

NAME	TYPE	DESCRIPTION
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

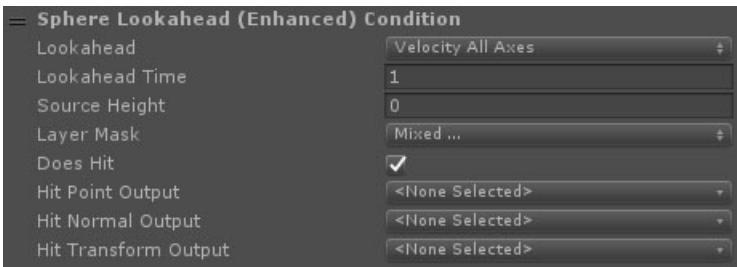
[Layers And Tags](#)

EnhancedSphereLookahead MotionGraphCondition

Overview

The EnhancedSphereLookahead condition performs a spherecast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction. The cast results are output to [motion graph parameters](#) for later use.

Inspector



Properties

Name	Type	Description
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

[Motion Graph Parameters And Data](#)

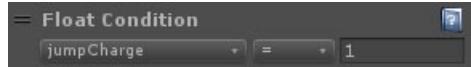
Layers And Tags

Float MotionGraphCondition

Overview

The Float condition checks the specified parameter against a value.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to check.
Comparison	Dropdown	The comparison type between the parameter and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

[Motion Graph Parameters And Data](#)

GroundContact MotionGraphCondition

Overview

The GroundContact condition checks if the character is airborne or touching the ground.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Equals	Boolean	Is the condition true if the character is touching the ground or not.

See Also

[MotionController](#)

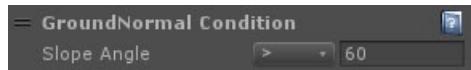
GroundNormal MotionGraphCondition

Overview

The GroundNormal condition is used to check against the angle of the character ground contact.

This condition differs from the [GroundSurfaceNormal](#) condition in that it is only aware of the point of contact and does not take the shape of the collision geometry into account. As a character walks off the edge of a flat surface, this angle will slope down, whereas the ground surface normal will stay pointing straight up until the character loses contact with the surface.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type. Options are GreaterThan () , GreaterOrEqual () , LessThan () , LessOrEqual () , EqualTo () .
Angle	Float	The ground slope angle in degrees from horizontal.

See Also

[MotionController](#)

[GroundSurfaceNormal Condition](#)

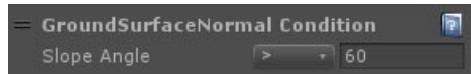
GroundSurfaceNormal MotionGraphCondition

Overview

The GroundSurfaceNormal condition is used to check against the slope of the ground surface the character is currently standing on. If the character is in contact with a flat surface then the resulting angle is the angle of the slope. If the character is in contact with an edge then the resulting angle is the slope of the top face the edge belongs to.

This condition differs from the [GroundNormal](#) condition which only takes into account the point of contact and not the shape of the collision geometry.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type. Options are GreaterThan () , GreaterOrEqual () , LessThan () , LessOrEqual () , EqualTo () .
Angle	Float	The ground slope angle in degrees from horizontal.

See Also

[MotionController](#)

[GroundNormal Condition](#)

HeightRestriction MotionGraphCondition

Overview

The HeightRestriction condition checks if the character height is restricted by a ceiling. This is useful for situations such as preventing the character moving to a sprint state while stuck in an air vent.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Height	Float	The target height multiplier for the character to check against.
Is Blocked	Boolean	Is the condition true if the character height is blocked or not blocked.

See Also

[MotionController](#)

InputVector MotionGraphCondition

Overview

The InputVector condition checks against the input vector provided by the [motion controller](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
InputComponent	Dropdown	<p>What component of the input to check. Available options are:</p> <ul style="list-style-type: none">• Magnitude 0 is no input, while 1 is one of the direction keys, or full tilt on an analog stick.• InputY the forward or back amount (1 is forward, -1 is back).• InputX the left or right amount (1 is right, -1 is left).• AbsoluteY the absolute forward or back amount.• AbsoluteX the absolute left or right amount.
Comparison	Dropdown	The comparison type between the actual input vector magnitude and the value specified. Options are greater than (>), less than (<).
Value	Float	The value to compare against.

See Also

Int MotionGraphCondition

Overview

The Int condition checks the specified parameter against a value.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to check.
Comparison	Dropdown	The comparison type between the property and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Int	The value to check against.

See Also

[Motion Graph Parameters And Data](#)

InventoryItem MotionGraphCondition

Overview

The InventoryItem condition checks how many of the selected inventory item the character is currently carrying.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Item Key	ID Picker	The inventory item key you are checking. Clicking this button will open the inventory database item picker.
Comparison Type	Dropdown	How are you comparing the carried quantity to the compare value (equal or not equal, greater or less than, etc).
Compare Value	Integer	The quantity to compare against.

See Also

[Inventory](#)

Pitch MotionGraphCondition

Overview

The Pitch condition checks pitch (up/down rotation) of the character's [aim controller](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type between the parameter and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Angle	Float	The pitch angle value to check against. 90 is straight up and -90 is straight down.

See Also

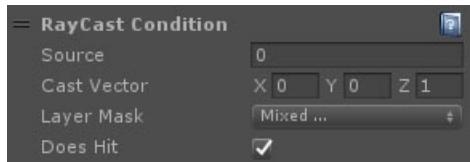
[Aim Controllers](#)

RayCast MotionGraphCondition

Overview

The RayCast condition performs a cast in the specified direction and from the specified source.

Inspector



Properties

Properties

Name	Type	Description
Normalise Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Vector	Vector3	The direction and distance to cast relative to the character. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or if it does not.

See Also

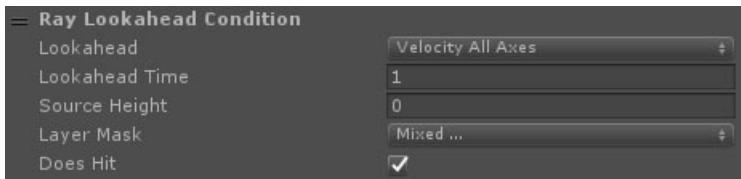
[Layers And Tags](#)

RayLookahead MotionGraphCondition

Overview

The RayLookahead condition performs a raycast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

See Also

[Motion Graph Parameters And Data](#)

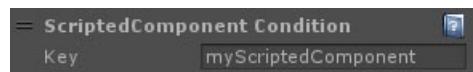
[Layers And Tags](#)

ScriptedComponent MotionGraphCondition

Overview

The ScriptedComponent condition attaches to a component on the controller's game object. Each time the condition is checked, the component is queried and the result returned. Valid components must implement the [IScriptedComponentCondition](#) interface.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Key	String	The name of the specific component. This is used to distinguish between multiple components on the same game object.

See Also

SphereCast MotionGraphCondition

Overview

The SphereCast condition performs a cast in the specified direction and from the specified source.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Normalise Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule. 1 is the top of the capsule.
Cast Vector	Vector3	The direction and distance to cast relative to the character. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or if it does not.

See Also

[Layers And Tags](#)

SphereLookahead MotionGraphCondition

Overview

The SphereLookahead condition performs a spherecast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction.

Inspector



Properties

Name	Type	Description
Lookahead	Dropdown	<p>How the lookahead direction is calculated. The available options are:</p> <ul style="list-style-type: none">• Velocity All Axes uses the current velocity of the character.• Velocity Horizontal Plane uses the character velocity constrained to its horizontal plane.• Velocity Vertical uses the character velocity constrained to its up-vector.• Direction All Axes uses the direction the character is moving in, but checks a fixed distance ahead.• Direction Horizontal Plane uses the character movement direction constrained to its horizontal plane.• Direction Vertical uses the character movement direction constrained to its up-vector.• Input Direction uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the Velocity lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the Direction or Input lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or does not.

See Also

[Motion Graph Parameters And Data](#)

[Layers And Tags](#)

Switch MotionGraphCondition

Overview

The Switch condition checks the value of the specified switch parameter.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to check.
Equals	Boolean	The value of the switch that the condition should be true for.

See Also

[Motion Graph Parameters And Data](#)

Transform MotionGraphCondition

Overview

The Transform condition checks if the specified transform parameter is null or not.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	TransformParameter	The parameter to check.
Is Null	Boolean	Is the condition true if the transform is null or not null.

See Also

[Motion Graph Parameters And Data](#)

Trigger MotionGraphCondition

Overview

The Trigger condition checks the specified trigger parameter and then resets it if triggered.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The trigger to check.

See Also

[Motion Graph Parameters And Data](#)

Vector MotionGraphCondition

Overview

The Vector condition checks the specified parameter against a value.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The parameter to check.
Compare	Dropdown	<p>What element of the vector parameter to check against. The available options are:</p> <ul style="list-style-type: none">• Magnitude checks the value against the magnitude of the vector.• X checks against the vector's x axis.• Y checks against the vector's y axis.• Z checks against the vector's z axis.• Character Horizontal checks against the magnitude of the vector after it has been projected onto the character's horizontal plane.• Character Up checks against the magnitude of the vector after it has been projected onto the character's up vector.
Comparison	Dropdown	The comparison type between the parameter and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

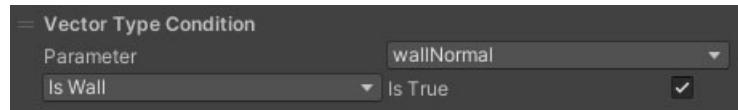
[Motion Graph Parameters And Data](#)

VectorType MotionGraphCondition

Overview

The VectorType condition checks the specified parameter against a value.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The parameter to check.
What	Dropdown	What condition to apply to the vector. Options are IsEmpty , IsUnitLength , IsWall , IsFloor , IsFlat .
Is True	Boolean	Does the condition pass if the check above is true or false.

See Also

[Motion Graph Parameters And Data](#)

Velocity MotionGraphCondition

Overview

The Velocity condition checks the velocity output of the character's [NeoCharacterController](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Compare	Dropdown	<p>What element of the velocity to check against. The available options are:</p> <ul style="list-style-type: none">• Character Speed checks the magnitude of the velocity vector against the value.• Horizontal Speed checks the velocity after it has been projected on the character's horizontal plane.• Vertical Velocity checks the velocity after it has been projected on the character's up vector.• Ground Speed checks the velocity after it has been projected on the ground normal plane.• Ground Surface Speed checks the velocity after it has been projected on the ground surface normal plane.• Yaw Velocity checks the velocity of the character along its forward vector.• Yaw Ground Velocity checks the velocity of the character along its forward vector projected on the ground normal plane.• Yaw Ground Surface Velocity checks the velocity of the character along its forward vector projected on the ground surface normal plane.
Comparison	Dropdown	The comparison type between the parameter and value. Options are EqualTo (=) , NotEqualTo (!=) , GreaterThan (>) , GreaterOrEqual (>=) , LessThan (<) , LessOrEqual (<=) .
Value	Float	The value to check against.

See Also

[NeoCharacterController](#)

Water MotionGraphCondition

Overview

The Water condition checks the character's position relative to a [water zone](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter with the transform of the water zone object. If this is null, the condition returns false.
Check	Dropdown	<p>The comparison type between the character and the water zone. Available options are:</p> <ul style="list-style-type: none">• Above Water Greater Than checks if the height of the character above the water line against the value.• Above Water Less Than checks if the height of the character above the water line against the value.• Below Water Greater Than checks if the height of the character below the water line against the value.• Below Water Less Than checks if the height of the character below the water line against the value.
Value	Float	The value to check against.

See Also

[Motion Graph Parameters And Data](#)

WieldableSelected MotionGraphCondition

Overview

The WieldableSelected condition is valid/invalid if the character has the specified wieldable item in their inventory and currently selected.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Item Key	ID Picker	The inventory item key for the wieldable you are checking. Clicking this button will open the inventory database item picker.
Selected	Dropdown	Does the condition return true if the wieldable is Selected or NotSelected .

See Also

[Inventory](#)

BasicWaterZone MonoBehaviour

Overview

The BasicWaterZone behaviour is used to communicate water properties to the motion graph, including surface height and normal, and flow vectors.

The water zone uses a [box collider](#) to detect when the character enters or leaves the zone.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Key	string	The name of the transform parameter on the motion graph for any character that enters the water zone. This will be set with the water zone's transform.
Flow	Vector3	A flow velocity (m/s) for the water zone.

See Also

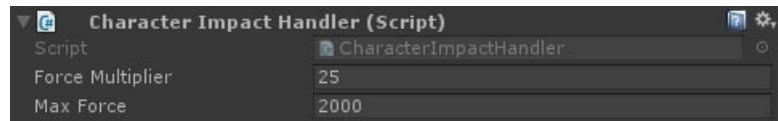
[Motion Graph Parameters And Data](#)

CharacterEventKickTrigger MonoBehaviour

Overview

The CharacterImpactHandler consumes impact events such as bullet hits, and uses them to apply a force to the character's NeoCharacterController.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Force Multiplier	Float	A multiplier to apply to inbound forces for a more exaggerated effect.
Max Force	Float	The maximum force that can be applied to the character.

See Also

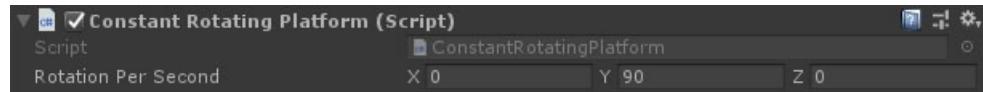
[NeoCharacterController](#)

ConstantRotatingPlatform MonoBehaviour

Overview

The ConstantRotatingPlatform behaviour is used to create a [moving platform](#) that constantly turns at a set rate.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rotation Per Second	Vector3	The rotation in degrees on each axis per second.

See Also

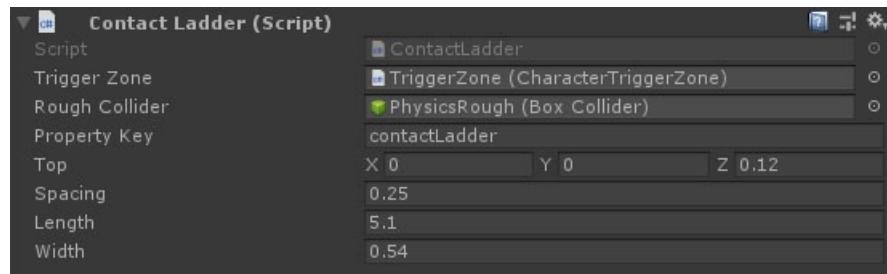
[Moving Platforms](#)

ContactLadder MonoBehaviour

Overview

The ContactLadder behaviour defines a [ladder](#) in the scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Trigger Zone	TriggerZone	The trigger area for detecting contact with the ladder.
Rough Collider	BoxCollider	The box collider for the ladder geometry.
Property Key	String	The motion graph parameter name to set the ladder to.
Top	Vector3	The top of the ladder surface relative to the transform position.
Spacing	Float	The spacing between rungs on the ladder.
Length	Float	The length of the ladder along the ladder transform down axis from the top offset.
Width	Float	The width of the ladder surface.

See Also

[Ladders](#)

[InteractiveLadder](#)

[Motion Graph Parameters And Data](#)

DrivenMovingPlatform MonoBehaviour

Overview

The DrivenMovingPlatform behaviour is added to a [rigidbody](#) that is driven by physics or a script and turns it into a [moving platform](#). It tracks the position and rotation of the object in a way that the [NeoCharacterController](#) can use.

Inspector



Properties

The DrivenMovingPlatform has no properties exposed in the inspector;

See Also

[Moving Platforms](#)

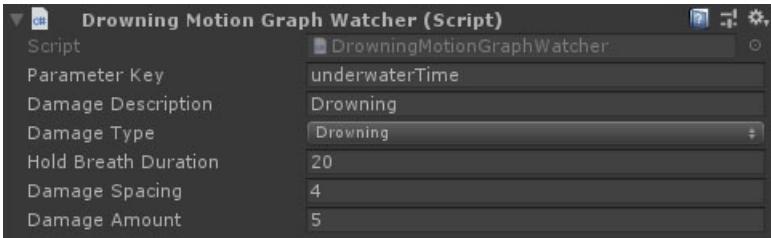
[NeoCharacterController](#)

DrowningMotionGraphWatcher MonoBehaviour

Overview

The DrowningMotionGraphWatcher watches a [float parameter](#) on the character motion graph and if the float value exceeds the specified amount then it starts to apply damage to the player's [health manager](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Key	String	The name of the float parameter on the motion graph that the watcher should track.
Damage Description	String	A descriptive name for the damage that can be used in any HUD logs.
Damage Type	Dropdown	The damage type the drowning should be classed as. Used for filtering the damage.
Hold Breath Duration	Float	The value the parameter needs to reach before damage is applied.
Damage Spacing	Float	Damage will be applied every time the parameter passes an increment of this amount above the drown duration.
Damage Amount	Float	The amount of damage to apply to the character.

See Also

[Health and Damage](#)

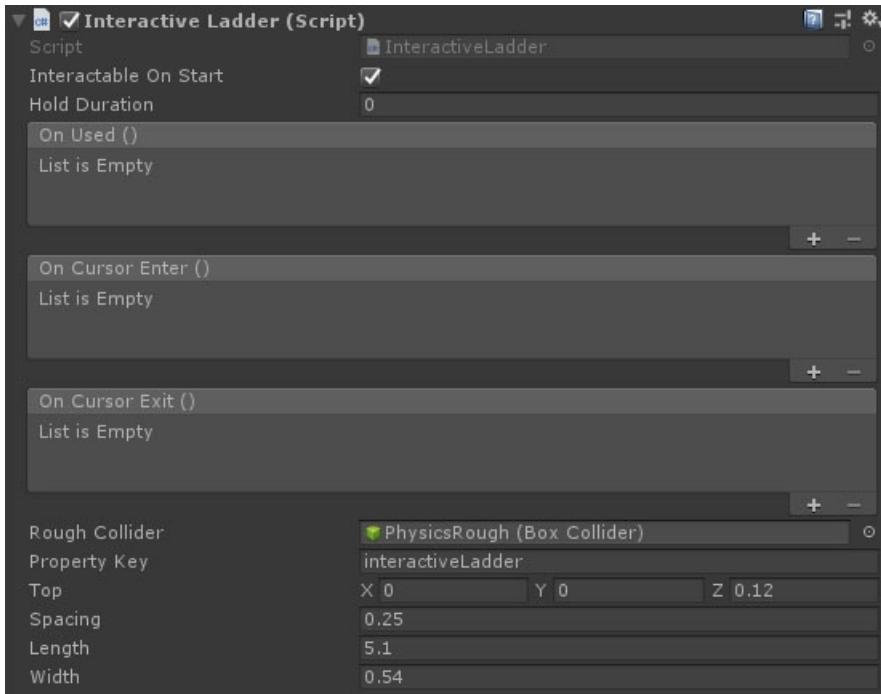
[Motion Graph Parameters And Data](#)

InteractiveLadder MonoBehaviour

Overview

The InteractiveLadder behaviour defines a [ladder](#) in the scene.

Inspector



Properties

The InteractiveLadder inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Rough Collider	BoxCollider	The box collider for the ladder geometry.
Property Key	String	The motion graph parameter name to set the ladder to.
Top	Vector3	The top of the ladder surface relative to the transform position.
Spacing	Float	The spacing between rungs on the ladder.
Length	Float	The length of the ladder along the ladder transform down axis from the top offset.
Width	Float	The width of the ladder surface.

See Also

[Ladders](#)

[Interaction](#)

[ContactLadder](#)

[Motion Graph Parameters And Data](#)

JumpPad MonoBehaviour

Overview

The JumpPad behaviour detects when a character touches it and passes a move vector to its motion graph.

Inspector



Properties

NAME	TYP	DESCRIPTION
Parameter Name	String	The motion graph parameter name to set with the jump vector.
Boost Vector	Vector3	The velocity vector to give to the character.
Boost Relative To	Float	The space in which to apply the vector. If this is set to self, the vector will be relative to this object's transform.
Cooldown	Float	The cooldown time to prevent the jumppad registering multiple times.

See Also

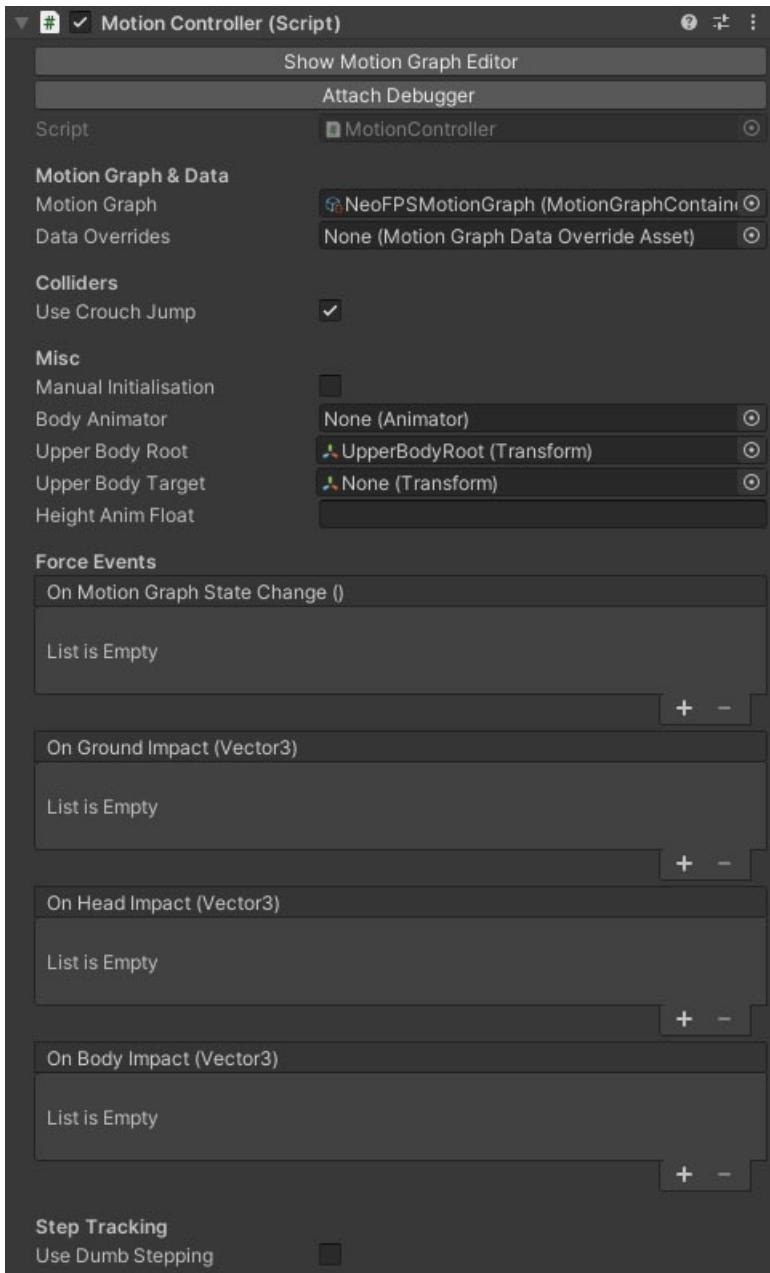
[Motion Graph Parameters And Data](#)

MotionController Behaviour

Overview

The MotionController is the workhorse of the MotionGraph system. It handles the actual movement of the character, ticks the motion graph, and sends input and information to the motion graph that it needs to react to the player and environment.

Inspector



The **Show Motion Graph Editor** button will show the motion graph editor, with this controller's motion graph opened for editing. If the controller is attached to a character in the scene and the game in play mode, then the opened graph will update at runtime to show the current state.

The **Attach Debugger** button will open the motion debugger and attach the character to start recording its movement details.

Properties

Motion Graph & Data

NAME	TYPE	DESCRIPTION
Motion Graph	MotionGraph	The motion graph for the controller to use (a unique instance will be instantiated from this).
Override	MotionGraphDataOverrideAsset	The motion data override for the controller to use.

Colliders

NAME	TYPE	DESCRIPTION
Use Crouch Jump	Boolean	If this is enabled, then the collider will provide an offset that can be used to give extra height to a jump so it appears the legs are tucked up instead of the head ducked down.

Misc

NAME	TYPE	DESCRIPTION
Manual Initialisation	Boolean	Should the component be initialised manually or automatically in Awake and Start? Switch this on for things like networked players.
Body Animator	Animator	An optional animator component for the character body . This allows motion graph states and behaviours to easily drive animations.
Upper Body Root	Transform	(Required) The root transform of the head hierarchy (height interpolated when crouching, or matched to animator hierarchy). This property was formerly called <i>UpperBodyRoot</i> .
Upper Body Target	Transform	(Required if the character has a body) A bone in the character's skeleton that's used as a target for the head root.
Height Anim Float	Transform	(Required if the character has a body) The name of the float parameter in the character body's animator controller that is used to control the character height blend.

Force Events

NAME	TYPE	DESCRIPTION
On Motion Graph State Change	UnityEvent	This event is called whenever the controller graph state changes (only includes the end state when traversing multiple states in the graph).
On Ground Impact	UnityEvent	This event is called when the controller first contacts the ground after being airborne. Parameters = <code>Vector3 impulse, float mass</code>
On Head Impact	UnityEvent	This event is called whenever the top of the controller capsule makes initial contact with a collider. Parameters = <code>Vector3 impulse, float mass</code>
On Body Impact	UnityEvent	This event is called whenever the sides of the controller capsule makes initial contact with a collider. Parameters = <code>Vector3 impulse, float mass</code>

Step Tracking

NAME	TYPE	DESCRIPTION
Use Dumb Stepping	Boolean	Switch this to true if you aren't tracking steps in the motion graph, and they will simply be counted at a default rate whenever the character is grounded.

See Also

[The Motion Graph](#)

[Motion Controller Data](#)

[NeoCharacterController](#)

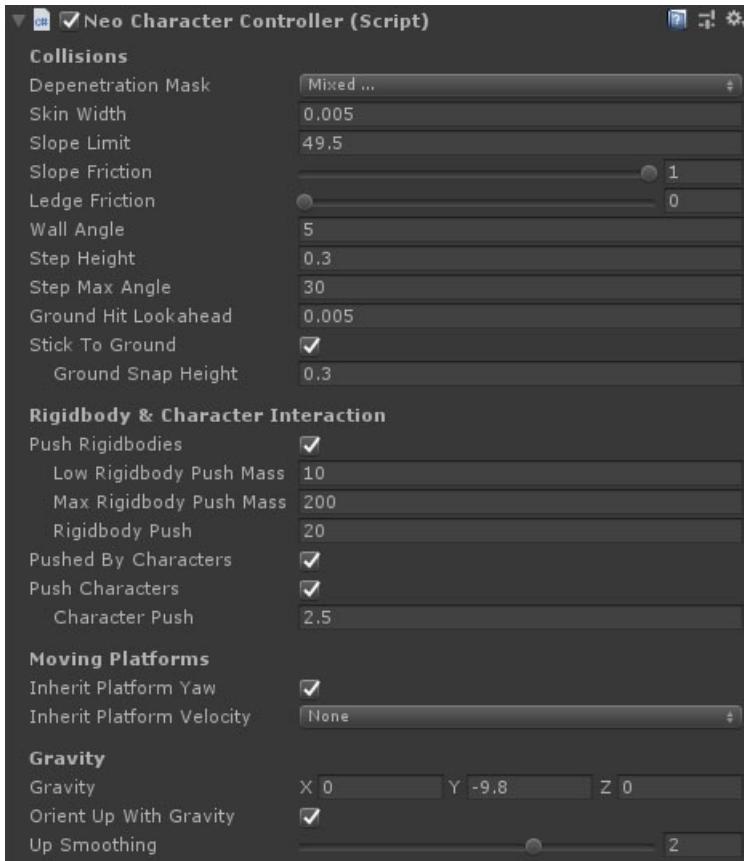
[First Person Body](#)

NeoCharacterController Behaviour

Overview

The NeoCharacterController handles movement of characters and their interaction with the physical environment.

Inspector



Properties

Collisions

Name	Type	Description
Depenetration Mask	LayerMask	The physics layers that the character will depenetrate from. This cannot include anything outside of the collision matrix for the gameobject layer. Use this to filter out small dynamic props that should not influence the player.
Skin Width	Float	When performing the move loop, the capsule is shrunk by this amount. When testing for contacts it is grown by this amount.
Slope Limit	Float	The maximum distance above the ground to apply a "sticky" downforce on the frame after leaving the ground.
Slope Friction	Float	The friction of ground contacts when standing on a slope. At 1, all downward velocity will be cancelled out. At 0, the character will slide down the slope.
Ledge Friction	Float	The friction of ground contacts when overhanging a ledge. At 1, the character will not slide off the ledge.
Wall Angle	Float	The angle (in degrees) from the vertical for a surface to be considered a wall.

NAME	TYPE	DESCRIPTION
Deflection Curve	AnimationCurve	A curve that defines the deflection drop off based on angle from normal. Y-axis is the deflection multiplier, X-axis is the normalised angle (0 = 0 degrees, 1 = 90 degrees)
Step Height	Float	The character will traverse any ledge up to their radius in height. If the step is equal to or below the step height then the character will not lose any horizontal speed when stepping up, and any vertical movement does not count to the character's velocity calculations.
Ground Snap Height	Float	The maximum distance above the ground to apply a "sticky" downforce on the frame after leaving the ground in certain conditions. This prevents leaving the ground when stepping onto down-slopes or off low steps.
Stick To Ground	Boolean	Should the character stick to the ground when walking down steep slopes or over the top of ramps.
Ground Hit Lookahead	Float	The distance to check ahead of a contact (based on contact normal) to see if it was a slope or a step. Set this higher if you are using physics with bevelled corners instead of primitives (naughty).

Rigidbody and Character Interaction

NAME	TYPE	DESCRIPTION
Push Rigidbodies	Boolean	Do not apply forces to non-kinematic rigidbodies if false.
Low Rigidbody Push Mass	Float	Any rigidbodies this mass or below will be pushed with the full push multiplier. Above this and it drops off to zero at max mass.
Max Rigidbody Push Mass	Float	Any rigidbodies above this mass will have zero force applied to them.
Rigidbody Push	Float	A multiplier for the push force at or below the minimum push mass. At normal gravity with no physics materials applied, a 1m box will be on the threshold of moving when this is set to 10. Higher will push the box up to the character's velocity with greater acceleration.
Pushed By Characters	Boolean	Can this character be pushed by other INeoCharacterControllers.
Push Characters	Boolean	Can this character push other INeoCharacterControllers.
Character Push	Float	A multiplier for the push force when pushing characters at or below this characters mass. Drops to 0 when approaching max push mass.

Moving Platforms

NAME	TYPE	DESCRIPTION
Inherit Platform Yaw	Boolean	Does the character inherit yaw changes from moving platforms.

NAME	TYPE	DESCRIPTION
Inherit Platform Velocity	Dropdown	<p>What component of the platform velocity should be included in the character velocity. Options are as follows:</p> <ul style="list-style-type: none"> • None - any movement of the platform is ignored when calculating the character velocity. This prevents exponential acceleration if the character tracks momentum. • Full - the movement of the platform is included in character velocity calculations. • Horizontal Only - only the horizontal movement of the platform is included in character velocity calculations. Vertical movement is ignored. • Vertical Only - only the vertical movement of the platform is included in character velocity calculations. Horizontal movement is ignored.

Gravity

NAME	TYPE	DESCRIPTION
Gravity	Vector3	The gravity vector (direction and acceleration) for the character.
OrientUpWithGravity	Boolean	If this is true, then adjusting the gravity direction will reorient the character so that down is in the direction of gravity, and up is opposed.
UpSmoothing	Float	The duration (in seconds) it takes to rotate the character up vector a whole 180 degrees.

Note

Due to a change in the way ground slope collisions are resolved, the slope speed curve settings have been removed and are now applied to the movement states in the motion graph directly.

See Also

[NeoCharacterController](#)

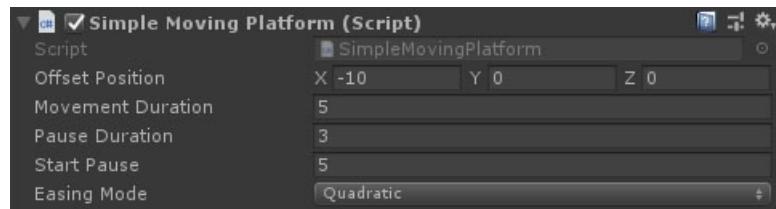
[Unity CharacterController](#)

SimpleMovingPlatform MonoBehaviour

Overview

The SimpleMovingPlatform behaviour is used to create a [moving platform](#) that moves between 2 points at set intervals.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Offset Position	Vector3	The position to move to (offset from current position in world space).
Movement Duration	Float	The time it takes to move.
Pause Duration	Float	The pause before returning to original position or moving again.
Start Pause	Float	The delay before the first move.
Easing Mode	Dropdown	The easing mode for the movement. Options are: Linear , Quadratic , Cubic , Quartic .

See Also

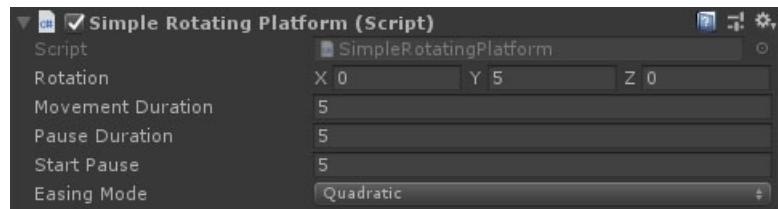
[Moving Platforms](#)

SimpleRotatingPlatform MonoBehaviour

Overview

The SimpleRotatingPlatform behaviour is used to create a [moving platform](#) that rotates at set intervals.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rotation	Vector3	The total rotation for each rotation phase (relative to the starting rotation of the phase, in world space).
Movement Duration	Float	The time it takes to move.
Pause Duration	Float	The pause before returning to original position or moving again.
Start Pause	Float	The delay before the first move.
Easing Mode	Dropdown	The easing mode for the movement. Options are: Linear , Quadratic , Cubic , Quartic .

See Also

[Moving Platforms](#)

WaypointMovingPlatform MonoBehaviour

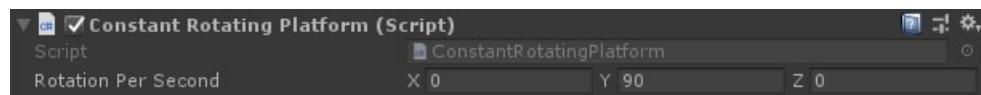
Overview

The WaypointMovingPlatform behaviour is used to create a [moving platform](#) that moves between a set of waypoints.

The waypoints include position and rotation and can be moved to in sequence or directly. The waypoints can also be set to loop round from last back to first, or to form a broken chain.

Movement can be triggered on startup so the platform loops through waypoints, or it can be triggered via scripts and [UnityEvents](#). Using scripting and events you can set the platform to move to a specific waypoint, either via the intermediate waypoints or directly, or you can start and stop the platform looping through waypoints.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Starting Waypoint	Integer	The waypoint the platform starts at (will be repositioned on start).
Speed Curve	AnimationCurve	An animation curve to apply easing to movement between waypoints.
Delay	Float	The delay between waypoints when moving through a sequence. The platform will stop at a waypoint for this duration.
On Start	Dropdown	What to do on start. Options are as follows: <ul style="list-style-type: none">• Nothing - the platform will not do anything until another script or component triggers it.• Loop Forwards - the platform will keep looping through all of the waypoints in order until stopped. If the waypoints are not circular, it will change directions when it reaches the last waypoint.• Loop Backwards - the platform will keep looping through all of the waypoints in reverse order until stopped. If the waypoints are not circular, it will change directions when it reaches the first waypoint.
Circular	Boolean	If the waypoints are circular then there is a direct route from the first to last waypoints without going through the others.
Interruptable	Boolean	Defines how the platform behaves when setting the target waypoint while already moving. If interruptable, the platform will immediately switch to the new target waypoint, reversing direction if necessary. If not interruptable, then the platform will move to the next waypoint and then change directions if required.

The inspector then shows a list of waypoints with the following properties:

NAME	TYPE	DESCRIPTION
Position	Vector3	The position of the platform at this waypoint.
Rotation	Vector3	The rotation of the platform at this waypoint.

NAME	TYPE	DESCRIPTION
Time To Waypoint X	Float	The journey time to reach the next platform in the sequence.
Use Current Transform	Button	Set the position and rotation to match the platform transform in the scene.
Move To Waypoint	Button	Instantly move the platform transform to the position and rotation of the waypoint in the scene.
Up	Button	Move this waypoint up in the sequence.
Down	Button	Move this waypoint down in the sequence.
Insert After	Button	This will insert a new waypoint immediately after this one.
Remove	Button	This will remove the waypoint from the list.

See Also

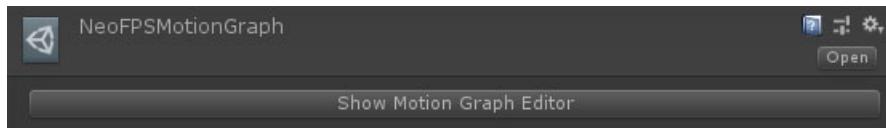
[Moving Platforms](#)

MotionGraph ScriptableObject

Overview

The MotionGraph scriptable object contains the motion graph layout and components. During runtime, a unique instance of the graph will be instantiated so that any modifications to the properties are unique to the character in question.

Inspector



Properties

The MotionGraph has no properties exposed in the inspector.

The **Show Motion Graph Editor** button will open the motion graph editor window, editing this graph.

See Also

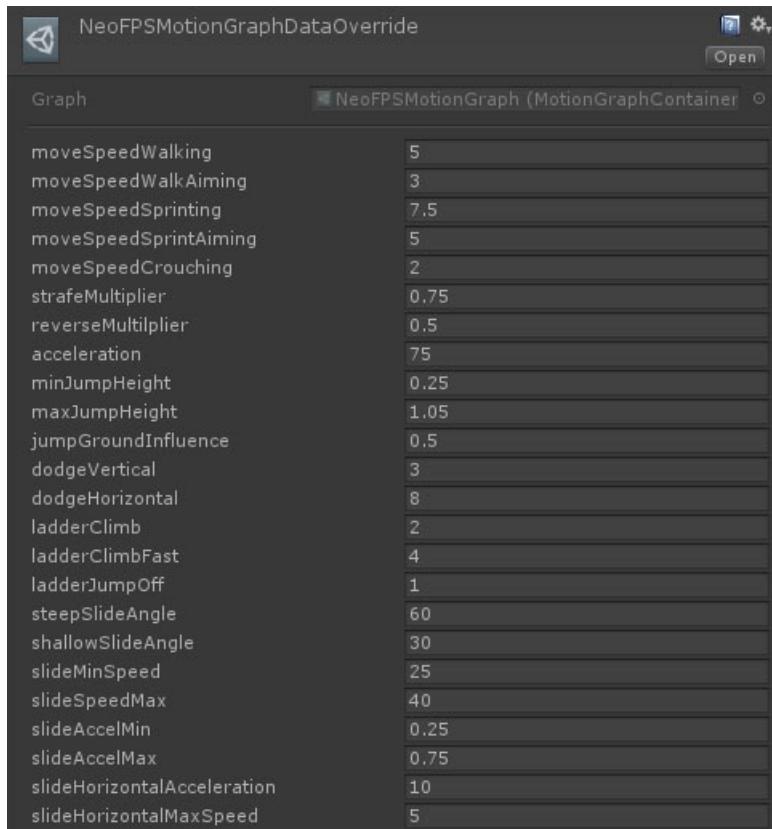
[The Motion Graph](#)

MotionGraphDataOverrideAsset ScriptableObject

Overview

The MotionGraphDataOverrideAsset scriptable object is created from the motion graph editor in the **Motion Data** section. It is permanently attached to the graph you were editing when creating the asset, and will allow you to override each of the data entries on the graph.

Inspector



Properties

The MotionGraphDataOverrideAsset displays an override entry for each of the motion data entries on the motion graph it is attached to. For more information see [Motion Graph Parameters And Data](#).

See Also

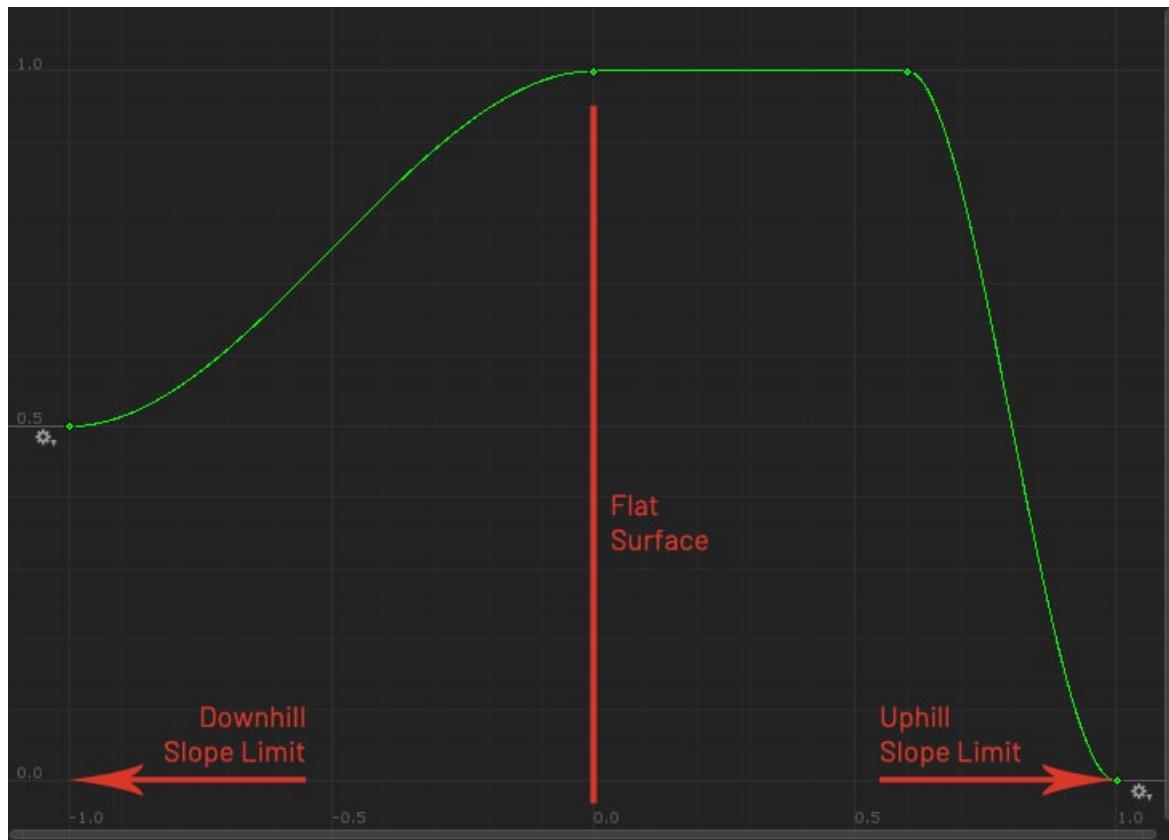
[The Motion Graph](#)

[Motion Graph Parameters And Data](#)

SlopeSpeedCurve ScriptableObject

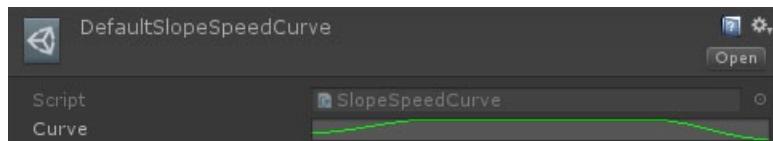
Overview

The SlopeSpeedCurve scriptable object is used by the grounded [Movement](#) state to allow more control over the character's speed on slopes. The X axis of the curve represents the slope, from slope limit downhill at -1 to completely flat at 0 and up to the character slope limit uphill at 1. The y value is the amount of horizontal speed that is maintained when deflected by the slope.



The [Movement](#) state aligns its target velocity to the ground surface, clamps the new move vector to the original movement speed and then multiplies by the result of the slope speed curve at the relevant angle. This means that the character can maintain their speed up to a set angle and then it drops off slowly as they approach the limit instead of hitting an invisible wall as the slope angle increases.

Inspector



Properties

Name	Type	Description
Curve	AnimationCurve	The desired speed curve as specified above. Ensure that the x-axis extends from -1 to 1 and the y-axis does not go below 0.

See Also

[Movement MotionGraphState](#)

NeoFPS Input System

Overview

NeoFPS is built from the ground up to facilitate developing first person shooters for PC and console.

In order for NeoFPS' input system to work it requires a number of custom project settings to be applied. For more information see the [Input Settings](#) page.

Unity's existing input system has a number of shortcomings when designing a control scheme. This has led Unity to develop a new input system which is currently in testing.

NeoFPS extends the current input system to address a number of its problems. It adds features such as:

- Runtime bindable keys
- Consistent gamepad profiles
- Mouse smoothing and acceleration
- Input contexts (character, menu, etc).

Once Unity's new input system is in full release the NeoFPS system will be adapted to that. You can also completely replace the NeoFPS input system if you have an input solution that you prefer. Look at the existing input handlers to see which properties and methods they access and use these with your preferred system.

Input Contexts

In order to categorise inputs and control which inputs are active at any time, NeoFPS makes use of input contexts. These contexts are defined in the `FpsInputContext` [generated constant](#) through the `[NeoFpsInputManager][6]` asset. Each input handler defines its input context in script and its inputs will only be processed if its context is either **None** or if no higher priority context is active. Context priority is simply based on the constant value, with higher numbers being higher priority. A context becomes active when one or more input handlers with that context are active. For example, if a UI element has a `InputMenu` handler attached to it, then when the UI element is active all character input will be blocked.

Keyboard

NeoFPS is preset with a full WASD control scheme, but any keys can be remapped from the in-game menu. The available buttons and their default key bindings can be specified in the `[NeoFpsInputManager][6]` asset.

Movement		
Forward	W	-
Backward	S	-
Left	A	-
Right	D	-
Jump	Space	-
Sprint (Hold)	-	-
Sprint (Toggle)	LeftShift	-
Crouch (Hold)	H <small>+</small> T <small>+</small>	-
Crouch (Toggle)	LeftControl	-
Interaction		
Use	E	-
Pick Up	Z	-
Combat		
Primary Fire	Mouse0	-
Secondary Fire	Mouse2	-

At the top of the input bindings menu is a drop down to reset to defaults. This also allows you to select a keyboard layout. Resetting to a different keyboard layout will map the keys so that the hand positions match a Qwerty keyboard. Available keyboard layouts currently include:

- Qwerty
- Azerty
- Qwertz
- Dvorak
- Colemak

To request more keyboard layouts, please contact support via the discord, or using support@neofps.com

Mouse

NeoFPS features detailed mouse support with inverse look, smoothing and acceleration.

Horizontal Mouse Sensitivity	-	25	+
Vertical Mouse Sensitivity	-	25	+
Invert Mouse	◀	No	▶
Mouse Acceleration	◀	Enabled	▶
Mouse Acceleration Amount	-	50	+
Mouse Smoothing	◀	Enabled	▶
Mouse Smoothing Amount	-	50	+

Smoothing

Mouse smoothing is performed with a weighted average system. You can customise the number of frames to look back as well as the weighting to apply in the [MouseAndGamepadAimController](#) behaviour.

Acceleration

Acceleration can be customised via the [MouseAndGamepadAimController](#) behaviour, including setting limits and whether the acceleration is linear or quadratic.

Gamepads

Gamepad support in Unity can be quite complicated, with the same gamepad being mapped differently across platforms.

For more information see the [Unity Wiki](#)

NeoFPS maps all of the available axes and buttons of the gamepad and wraps them up into preset profiles. The player can then choose between these profiles from the in-game menus.

Use Gamepad	◀	Yes	▶
Gamepad Profile	◀	XBox 360	▶
Invert Gamepad Look	◀	No	▶
Horizontal Analog Sensitivity	-	100	+
Vertical Analog Sensitivity	-	100	+

On console those profiles might be fairly minimal such as *standard* and *south paw*. On standalone builds there might be a number more options as more individual controllers are supported. You can set up your own gamepad profiles via the [\[NeoFpsInputManager\]](#)[6].

See Also

[Input Settings](#)

[Creating Custom Input Handlers](#)

Input Settings

Overview

NeoFPS uses controller profiles to achieve consistent mapping for game controllers across multiple platforms. Since the button layouts provided to Unity by the various controller drivers are not consistent, NeoFPS requires every available axis to be mapped in the input settings and then builds the profiles in code using conditional compilation for the different platforms. Examples of the different controller mappings can be found on the [Unify Community Wiki](#).

For more details of how Unity handles input out of the box, see the [Unity Input Settings](#).

A new input system is in development at Unity and available to use through the package manager in the editor. Once this is in full release, a NeoFPS implementation will be added that makes use of it.

It should also be noted that you can use an alternative input system in NeoFPS by replacing the included input handlers with your own implementations.

Required Axes

The following are the axes that NeoFPS uses:

Movement

PROPERTIES
Name = Horizontal
Negative Button = left
Positive Button = right
Alt Negative Button = a
Alt Negative Button = d
Gravity = 3.0
Dead = 0.001
Sensitivity = 3.0
Snap = true
Invert = false
Type = Key or Mouse Button
PROPERTIES
Name = Horizontal
Gravity = 0.0
Dead = 0.3

PROPERTIES

Sensitivity = 1.0

Snap = false

Invert = false

Type = Joystick Axis

Axis = X Axis

Joy Num = Get Motion from all Joysticks

PROPERTIES

Name = Vertical

Negative Button = down

Positive Button = up

Alt Negative Button = s

Alt Negative Button = w

Gravity = 3.0

Dead = 0.001

Sensitivity = 3.0

Snap = true

Invert = false

Type = Key or Mouse Button

PROPERTIES

Name = Vertical

Gravity = 0.0

Dead = 0.3

Sensitivity = 1.0

Snap = false

Invert = true

PROPERTIES**Type** = Joystick Axis**Axis** = Y Axis**Joy Num** = Get Motion from all Joysticks

Mouse Controls

PROPERTIES**Name** = Mouse X**Gravity** = 0.0**Dead** = 0.0**Sensitivity** = 0.1**Snap** = false**Invert** = false**Type** = Mouse Movement**Axis** = X Axis**PROPERTIES****Name** = Mouse Y**Gravity** = 0.0**Dead** = 0.0**Sensitivity** = 0.1**Snap** = false**Invert** = false**Type** = Mouse Movement**Axis** = Y Axis**PROPERTIES****Name** = Mouse ScrollWheel**Gravity** = 0.0

PROPERTIES

Dead = 0.0

Sensitivity = 0.1

Snap = false

Invert = false

Type = Mouse Movement

Axis = 3rd Axis (Joysticks and Scrollwheel)

UI

PROPERTIES

Name = Submit

Positive Button = return

Alt Positive Button = enter

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = false

Invert = false

Type = Key or Mouse Button

PROPERTIES

Name = Submit

Positive Button = joystick button 0

Alt Positive Button = joystick button 16

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = false

PROPERTIES

Invert = false

Type = Key or Mouse Button

PROPERTIES

Name = Cancel

Positive Button = escape

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = false

Invert = false

Type = Key or Mouse Button

Individual Axes

Axes 1 to 5 are mapped to generic axes with the names **Gamepad Axis 1** to **Gamepad Axis 5** and the following settings. The **Axis** property is set to the relevant entry (1-5).

PROPERTIES

Gravity = 0.0

Dead = 0.2

Sensitivity = 1.0

Snap = false

Type = Joystick Axis

Each axis is also mapped with and inverted version called **Gamepad AxisInv 1** to **Gamepad AxisInv 5**. As you would expect, the non-inverse axes have the **invert** property set to *false*, while the inverse axes have the property set to *true*.

Individual Buttons

Gamepad buttons are mapped with the names **Gamepad Button 0** to **Gamepad Button 19** and the following settings. The **Positive Button** property is set to *joystick button*, where *is* replaced with the corresponding button number.

PROPERTIES

Gravity = 1000.0

PROPERTIES

Dead = 0.001

Sensitivity = 1000.0

Snap = true

Invert = false

Type = Key or Mouse Button

Button Axes

On some platforms, some controller buttons are actually presented as axes. To handle these cases the buttons are mapped with the names **Gamepad AxisBtn 3** to **Gamepad AxisBtn 10** with the following settings. The **Axis** property is set to the relevant entry (3-10).

PROPERTIES

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = true

Invert = false

Type = Joystick Axis

There are also 2 inverse button axes called **Gamepad AxisBtnInv 6** and **Gamepad AxisBtnInv 7** where the **Invert** property is set to *true*.

See Also

[NeoFPS Input System](#)

[Unity Input Settings](#)

[Unify Community Wiki - Gamepads](#)

Touchscreen Input

Overview



NeoFPS comes with a custom touchscreen setup for games targeting mobile devices. There are input manager and input system versions of the touch controls, along with replacement UI prefabs to demo them.

The touchscreen controls system works by attaching a "virtual input" device to the NeoFPS input manager. When the input handlers (such as for firearms or character movement) check for input each frame, they will also check the virtual input. In the case of the default input manager setup, this means asking the virtual input the [FpsInputButton](#) or [FpsInputAxis](#) values. In the case of the input system, this means asking the virtual input for a value corresponding with the relevant input action's GUID.

The touchscreen controller component checks for taps each frame. For any new touch inputs it detects all the controls under that point, and assigns a "touch processor" that stores those controls. Each frame until the touch is released the processor will work through the controls in priority order and ask them to handle the input. Each control can then consume the input or allow it to fall through to the control underneath it. This allows for behaviour such as in the demo HUD prefab, where the whole screen is a giant virtual trackball for aiming that falls through to a shoot button. Any other controls on top of that such as the movement or buttons consume their input so it never reaches those aim or shoot controls.

Monobehaviours

The touchscreen system utilises the following behaviours:

INPUT MANAGER BEHAVIOUR	INPUT SYSTEM BEHAVIOUR	DESCRIPTION
NeoFpsTouchScreenController	NeoFpsInputSystemTouchScreenController	The controller is the behaviour that actually detects touches and passes them on to the attached controls.
NeoFpsTouchButton	NeoFpsInputSystemTouchButton	The touch button emulates tapping or holding a key or gamepad button.

INPUT MANAGER BEHAVIOUR	INPUT SYSTEM BEHAVIOUR	DESCRIPTION
NeoFpsTouchInventoryButton	NeoFpsInputSystemTouchInventoryButton	The inventory touch button is meant to be added to the inventory HUD element's slot items. It gets the slot index from the item and uses that to detect which input button / action ID to use.
NeoFpsTouchVirtualAnalog	NeoFpsInputSystemTouchVirtualAnalog	The virtual analog emulates an analog stick on a gamepad. Pressing near the middle will only give slight results, while moving further from the center will give a stronger input value.
NeoFpsTouchVirtualTrackball	NeoFpsInputSystemTouchVirtualTrackball	The virtual trackball emulates mouse style input as you would see with a trackball or trackpad. This is usually used for look/aim input as it can allow for very fine and precise control, or very quick large movements.

There is a demo UI prefab called **HudAndMenuCanvas_Touch** in the folder `Assets/NeoFPS/Samples/SinglePlayer/Prefabs` that is fully set up with a touch based control scheme for a mobile FPS (**HudAndMenuCanvas_InputSystemTouch** in `Assets/InputSystem/Prefabs` for the input system version).

Mobile Performance

The NeoFPS demo scenes target standalone platforms and are not optimised for mobile devices. The following suggestions are intended to help get the best performance you can:

- The NeoFPS demo scenes use 2 layers for environment physics: **EnvironmentDetail** for high detail, bullet blocking colliders, and **EnvironmentRough** for low detail character collisions. For mobile you should drop the high detail physics and just use the **Default** layer for your environment objects. This will also mean that decals no longer show up, so if you want to re-enable these then you can switch decals on for the default layer via the surface manager in the NeoFPS hub.
- The demo prefabs use post-processing attached to the character camera. This requires your lighting be set to **Linear** in the player settings, and on Android this means you must manually set the OpenGL API compatibility to *disable* OpenGL ES 2. Post processing can be very expensive on mobile, as can many rendering features, so it is advised to switch to the [Universal Render Pipeline](#).
- The way you build your games can have a drastic impact on performance. For example, switching the scripting backend from **Mono** to **IL2CPP** in your project's player settings can have a big impact. It's worth experimenting with the different settings here and following Unity's best practice advice for mobile platforms.
- If you have a lot of physics objects in your scenes, or environments with more complex physics, then it can be worth dropping your physics framerate in your project settings **Time** settings, and changing **Fixed Timestep**. This is how long each fixed frame lasts, so a larger value is a lower frame rate. The default of 0.02 means 50fps, but 30fps would be very playable for many mobile games.
- Due to the physical size of mobile phone screens, you can often drop the render resolution to have a big positive impact on performance, without drastically affecting the visual quality of your game. For example, the switch handheld games console runs at 720p when not docked, while many phones have 1080p screens.

See Also

[NeoFPS Input](#)

[Input System Extension](#)

Input System Extension

Overview

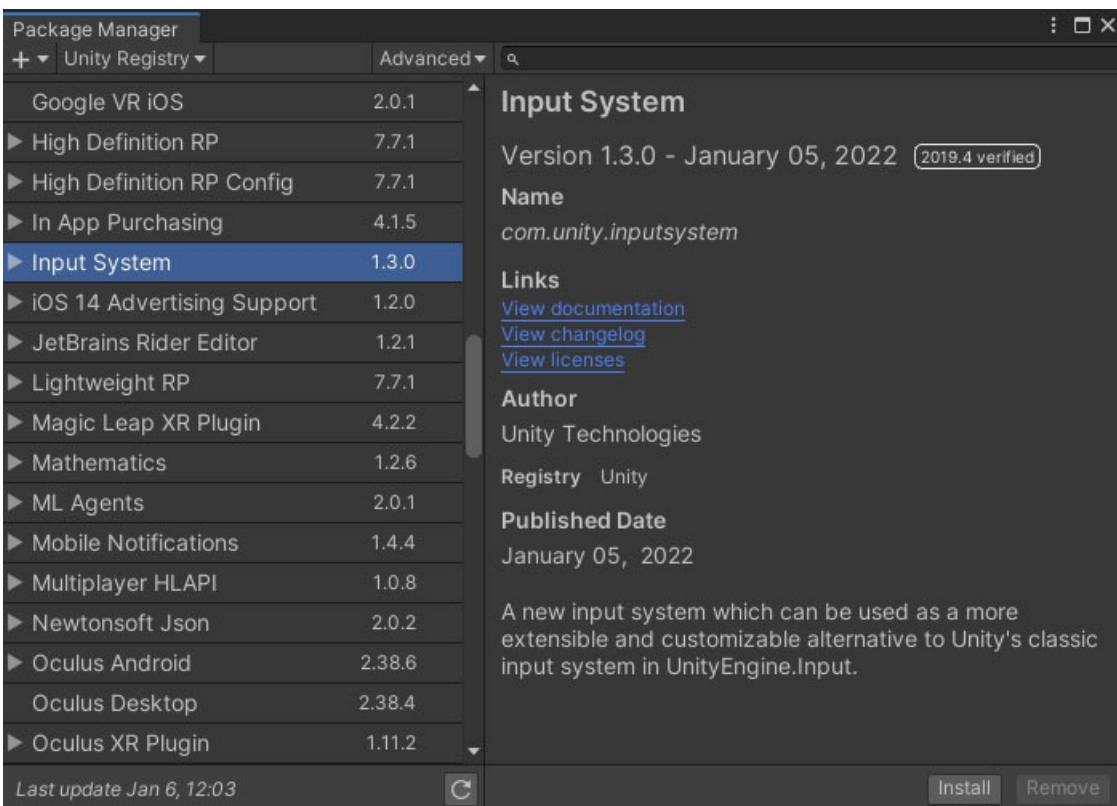
The Input System extension adds alternative versions of each of the NeoFPS input handlers and the input manager which use the input system instead of old input manager. The input system adds much better support for gamepad controllers, along with an action map editor that can be used to quickly add new input actions to your game, however its keyboard rebinding functionality is buggy when it comes to UI for key names.

Installation

Installing the Input System

Before extracting the extension you will need to make sure that the input system is installed and activated:

- Make sure that the **Active Input Handling** property in the Player Settings/Other Settings section is set to **Both**.
- Import the Input System package in the package manager (**minimum version 1.1.1**).



Note: The package manager might look different based on your Unity version. Consult Unity's documentation if you have trouble finding the required package

Importing the Extension Package

Setup and Use

Note: make sure to take a backup of your project before making major changes like these

Once the extension package has been imported you should follow these steps to start using the input system with NeoFPS:

1. Find the old input manager at *NeoFPS/Resources/FpsManager_Input*, and move it out of the Resources folder or delete it.
 - You will need to close and reopen the NeoFPS hub if you had it open at this point as it will be pointing at the old input manager.
2. Replace all input handler components (you can see a list below) on your weapons, characters and other prefabs with the equivalent input system versions.

- In **1.1.26** a tool was added to automatically convert input handlers on prefabs in the project. You can access this in the Unity menus at: *Tools/NeoFPS/Input Handler*. There is one option to convert input handlers to the input system, and one to convert them back to the legacy input manager. This will also transfer any settings. **Warning:** this tool modifies your project files and can not be undone. Always back up your project before making changes like this (or better yet, use version control such as Git).
3. Modify your menu prefabs to use the new input system based rebinding options panel
- Open your menu prefabs (eg **InGameMenu** and **MainMenu**) and under the **Panels** object delete the **OptionsPanel_Bindings** object
 - Replace that object with the **OptionsPanel_InputSystemBindings** from the *NeoFPS/Extensions/InputSystem/SampleUI* folder
 - In the menu hierarchy, expand *MenuObject/NavControls/NavControls_Options* and select the **Button_Bindings** object
 - In the **MultilInputButton** component, in the **On Click()** event, drag the new **OptionsPanel_InputSystemBindings** object onto the parameter that currently says **None (Menu Panel)**

Input Handlers

The following input system based input handlers are provided, which replace the input handlers used in the demo assets:

NAME	REPLACES	DESCRIPTION
InputSystemAbilityFirearm	InputAbilityFirearm	This is attached to a (non-inventory based) firearm in the character's hierarchy to trigger it with the ability button. An example is the shoulder mounted missile launchers in the jetpacks and guided missiles demo.
InputSystemCharacterMotion	InputCharacterMotion	This is attached to the root of the player character and sends input to the motion controller.
InputSystemCharacterSlowMo	InputCharacterSlowMo	This is attached to the root of the player character and triggers slow motion effects using the ability button.
InputSystemFirearm	InputFirearm	This is attached to the root of a firearm prefab and sends it the required weapon input.
InputSystemFirearmWithMelee	InputFirearmWithMelee	This is attached to the root of a firearm prefab that also has a Melee Weapon behaviour attached. It acts just as the firearm input above, but adds an extra control to trigger the melee attack.
InputSystemGame	InputGame	This is a placeholder input handler that can be extended to add game specific functionality such as showing a scoreboard. An example is placed on the player controller (not character) prefab.
InputSystemInventory	InputInventory	This is attached to the root of the player character and handles weapon selection and dropping items.
InputSystemLockpick	InputLockpick	This is attached to a lockpick popup prefab such as the <i>InteractableDoors_LockpickPopup3D</i> prefab found in the interactable doors demo folder.
InputSystemMeleeWeapon	InputMeleeWeapon	This is attached to the root of a melee weapon prefab and sends it the required weapon input.
InputSystemMenu	InputMenu	This is attached to a custom (non-NeoFPS) UI/menu object and disables character input when activated and restores it when deactivated.

NAME	REPLACES	DESCRIPTION
InputSystemThrownWeapon	InputThrownWeapon	This is attached to the root of a thrown weapon prefab and sends it the required weapon input.
InputSystemVehicle	InputVehicle	This is a placeholder input handler that can be attached to a vehicle to block character input.
InputSystemWieldableTool	InputWieldableTool	This is attached to the root of a wieldable tool prefab and sends it the required tool input.
MenuInputSystemToggle	UiInputToggle	This is extended from the InputSystemMenu component above and adds properties and methods to manually toggle the UI visibility instead of reacting to the UI object's state.

You can find the existing input handlers attached to the following objects:

- The root object of any weapon or tool (InputFirearm, InputMeleeWeapon, InputThrownWeapon, InputWieldableTool)
- The root of the character (InputCharacterMotion, InputInventory, InputCharacterSlowMo, etc)
- The player prefab (InputGame)
- Ability weapon prefabs (InputAbilityWeapon)
- Custom UI menus, not including the demo UIs (InputMenu)
- Lockpick UI prefabs (InputLockpick)

Touch controls



A number of touch controls are available for touchscreen input. The input system contains replacement versions of each of them. Inside the `Assets/InputSystem/Touchscreen` folder you will find the scripts that are equivalent to the input manager based touch controls. Inside the `Assets/InputSystem/Prefs` folder you will find touchscreen versions of the HUD and menu prefabs. Replacing the **HudAndMenuCanvas** or **HudAndMenuCanvas_Touch** in your scene with **HudAndMenuCanvas_InputSystemTouch** should be all you need to get started.

For more information on using touch controls in NeoFPS, see [Touchscreen Input](#).

Known Issues

The input system is a vast improvement on the old input manager with regards to gamepad controls and device agnostic input. However it does have some longstanding bugs, and the rebinding system is not as well rounded as it could be. There are 3 main areas where I feel it causes problems that might not be solveable until the input system itself is changed:

- The main visible issue that you will see is that the display name for keyboard keys looks good for the generic keyboard (as specified in the actions asset when defining the controls for actions), but when switching to a specific keyboard (ie when rebinding), the display name will just return the enum for the key. This means that on first setup a key such as left-shift will

appear as "Left Shift" in the rebinding UI, but if you try to map that to another key it will appear as "leftShift".

- There is no in-built system for gamepad profiles, and the NeoFPS solution works by using the rebinding system to rebind each button individually. This means a lot of complicated code that overlaps with the keyboard rebinding quite heavily and might be more fragile than it should be.
- The way the code generation works for the input asset locks the architecture down in a way that makes it very hard to extend (eg when using NeoFPS alongside other assets). In a future update (probably the 1.2 upgrade as mentioned below) the system will be switched away from using the code generation, and just tie into the action IDs on the asset directly (in a similar way to the Unity UI InputSystemEventHandler).

Future Development

For the upcoming 1.2 upgrade, NeoFPS will switch over to the input system by default, with an input manager fallback. There will also be a redesign of the rebinding setup to try and work around some of the limitations and weaknesses of the system.

See Also

[Touchscreen Input](#)

Creating Custom Input Handlers

Overview

NeoFPS includes a number of example input handlers:

- [InputCharacterMotion](#)
- [InputFirearm](#)
- [InputGame](#)
- [InputMeleeWeapon](#)
- [InputThrownWeapon](#)
- [InputInventory](#)

Handlers are attached to objects that require a unique control scheme. For example, each firearm has an input handler that is different from the handlers attached to melee weapons.

Handlers are context based and will not return any input when the context is not current.

FpsInputButton

NeoFPS uses a [generated constant](#) to define the available button/key inputs. This constant can be expanded based on your game requirements by changing the [ConstantsSettings](#) and regenerating the constant.

Take care that reordering entries in the constant will break those values in the inspector as they are stored based on index. Code will adapt to the changes in order, but inspector properties will need checking.

FpsInput Base Class

All input handlers derive from the `FpsInput` class. This controls context and exposes functions similar to Unity's `Input` class but uses the `FpsInputButton` constant instead of key codes and button IDs in order to separate the input from the key bindings.

The following is a simple example input handler:

```
public class InputExample : FpsInput
{
    public override FpsInputContext inputContext
    {
        // This input will only work when the "character" input context is active
        get { return FpsInputContext.Character; }
    }

    protected override void OnGainFocus()
    {
        // This method is called when the input context the handler uses gains focus
    }

    protected override void OnLoseFocus()
    {
        // This method is called when the input context the handler uses loses focus
    }

    protected override void UpdateInput()
    {
        if (GetButtonDown (FpsInputButton.PrimaryFire))
        {
            // React to the fire button
        }
    }
}
```

See Also

Input Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Character Is Looking Around And Shooting While In UI/Menu

NeoFPS input uses a context system to check which inputs should be active at any time. The context system is priority based. If an input handler with a higher priority context is active, then lower priority contexts won't do anything. One of the highest priority contexts is **InputContext.Menu**. This is used in the **InputMenu** component, meaning that if this component is active then character input will be disabled. If you have a number of UIs in your game, then add this component to the canvas UI objects that are made active when your UI is shown and it will disable character input automatically.

Gamepad Controls Are Broken

The main probable cause of this is that the Unity Input Manager settings have changed from what NeoFPS expects. When you first import NeoFPS you will see the NeoFPS Hub pop up, open on the Unity Settings page. From there you can apply the required settings (either all, or just input). You can access this again via the Unity menu: *Tools/NeoFPS/NeoFPS Hub*.

Some other assets have custom input settings that they require and will apply them automatically (especially if the asset type is "Complete Project", which will overwrite all Unity settings on import). If you need to combine settings with those then you can find the input requirements for NeoFPS [here](#).

Another possible cause is that the gamepad profile included in NeoFPS is incorrect. This would be unusual, since unknown controllers won't be detected via the current system. However, the input axes and buttons for gamepads in Unity can vary wildly based on the platform you are running on, and even whether the controllers are attached via cable or wirelessly. This problem is largely solved by using Unity's new(er) input system (NeoFPS integration coming soon).

I Changed Default Key Bindings In The NeoFPS Input Manager But They Haven't Changed In Game

The NeoFPS settings system checks for a relevant `.settings` file on start each time the game runs. The settings that you apply in the NeoFPS Hub should be considered defaults that will be used when no settings file is found. The easiest way to delete the keybindings settings is to open the **Game Settings** section of the hub, and select **Key Bindings**. In there you will see a button called **Delete User Settings File**. Clicking that will delete the settings file and effectively restore the game settings to your new defaults the next time the game is run.

InputAbilityFirearm MonoBehaviour

Overview

The InputAbilityFirearm behaviour reads the **Ability** button and uses it to fire the [ModularFirearm](#) that it's attached to. These firearms can be placed in the character hierarchy outside of the inventory system to create things like shoulder mounter launchers.

Inspector



Properties

The InputAbilityFirearm behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Unity Input](#)

[Modular Firearms](#)

InputAbilityMelee MonoBehaviour

Overview

The InputAbilityMelee behaviour reads the **Ability** button and uses it to trigger the [melee weapon](#) that it's attached to. These melee weapons can be placed in the character hierarchy outside of the inventory system and so used in parallel with a held weapon.

Inspector



Properties

The InputAbilityMelee behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Unity Input](#)

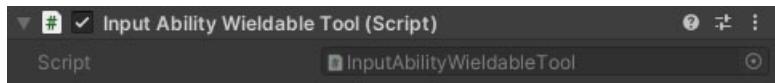
[Melee Weapons](#)

InputAbilityWieldableTool MonoBehaviour

Overview

The InputAbilityWieldableTool behaviour reads the **Ability** button and uses it to trigger the [wieldable tool](#) that it's attached to. These wieldable tools can be placed in the character hierarchy outside of the inventory system to create things like instant heals or shield boosts.

Inspector



Properties

The InputAbilityWieldableTool behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Unity Input](#)

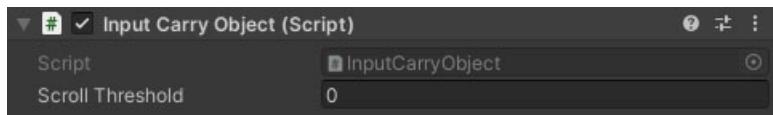
[Wieldable Tools](#)

InputCarryObject MonoBehaviour

Overview

The InputCarryObject behaviour communicates player input to the character's [carry system](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Scroll Threshold	Float	The minimum scroll wheel movement per frame for the scroll input to be registered.

See Also

[Carry System](#)

[NeoFPS Input](#)

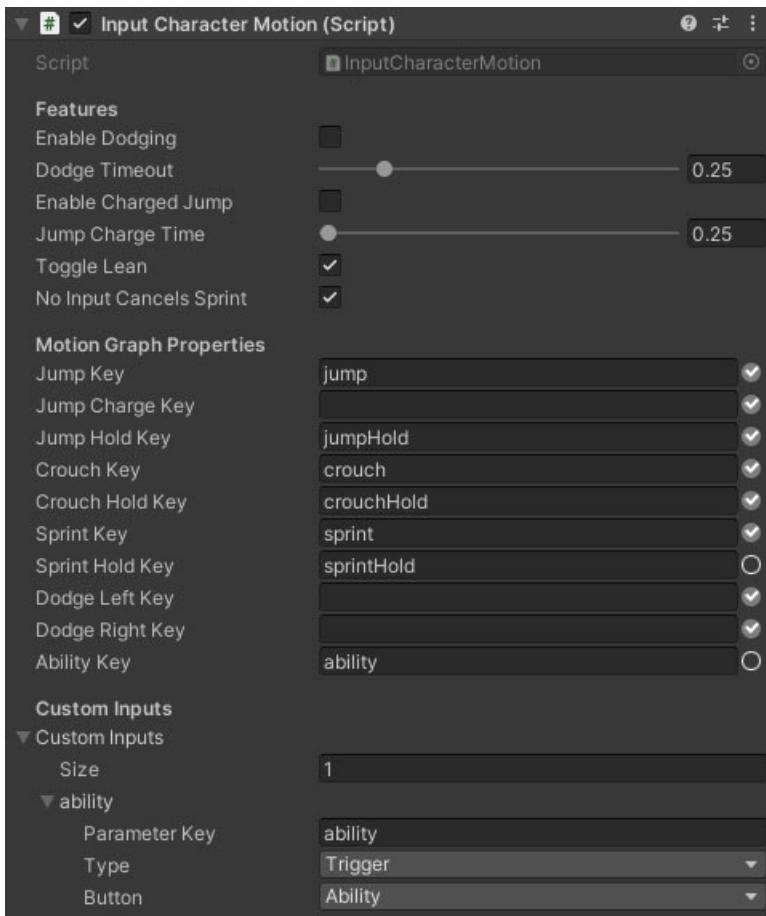
[Unity Input](#)

InputCharacterMotion MonoBehaviour

Overview

The InputCharacterMotion behaviour passes character input to the character [motion controller](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Enable Dodging	Boolean	Should double tapping a move direction dodge the character.
Dodge Timeout	Float	Multiple taps of a direction within this time range register as a double tap.
Enable Charged Jump	Boolean	Does holding the jump button charge up a jump or does the character dodge as soon as the button is pressed.
Jump Charge Time	Float	The time it takes to charge up a full power jump if charged jumps are enabled.
Toggle Lean	Boolean	Toggle leaning or hold to lean.
Jump Key	String	The key to the jump trigger parameter in the character motion graph .

NAME	TYPE	DESCRIPTION
Jump Charge Key	String	The optional key to the jump charge float parameter in the character motion graph . This is used for charging stronger jumps.
Jump Hold Key	String	The key to the jump hold switch parameter in the character motion graph . This is used for movement like swimming or flying where holding jump moves up.
Crouch Key	String	The key to the crouch switch parameter in the character motion graph .
Crouch Hold Key	String	The optional key to the crouch hold switch parameter in the character motion graph . This is used for movement like swimming or flying where holding jump moves down.
Sprint Key	String	The key to the sprint switch parameter in the character motion graph .
Sprint Hold Key	String	The key to the sprint hold switch parameter in the character motion graph .
Dodge Left Key	String	The optional key to the dodge left trigger parameter in the character motion graph .
Dodge Right Key	String	The optional key to the dodge right trigger parameter in the character motion graph .
Ability Key	String	The optional key to the "ability" trigger parameter in the character motion graph . This can be used for a variety of uses such as dashes, teleports, etc.
Custom Inputs	Array	Map input buttons to motion graph parameters here. motion graph .

Custom Input Entries

Each entry in the **Custom Inputs** array has the following properties:

NAME	TYPE	DESCRIPTION
Parameter Key	Boolean	The name of the parameter in the motion graph .
Type	Dropdown	The motion graph parameter type of the target parameter.
Button	Boolean	Which FpsInputButton to map to the parameter.

See Also

[NeoFPS Input](#)

[Unity Input](#)

[The Motion Graph](#)

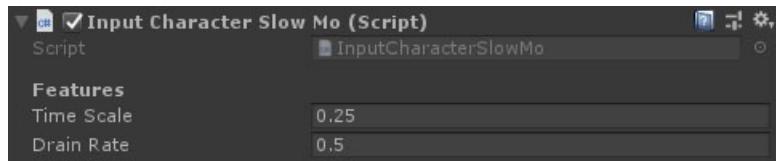
[Generated Constants](#)

InputCharacterSlowMo MonoBehaviour

Overview

The InputCharacterSlowMo behaviour reads the **Ability** button input and toggles slow-motion effects via the character's [SlowMoSystem](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time-scale to use for ability based slow-mo.
Drain Rate	Float	The rate to drain slow-mo charge (time scale will return to normal when charge reaches zero).

See Also

[NeoFPS Input](#)

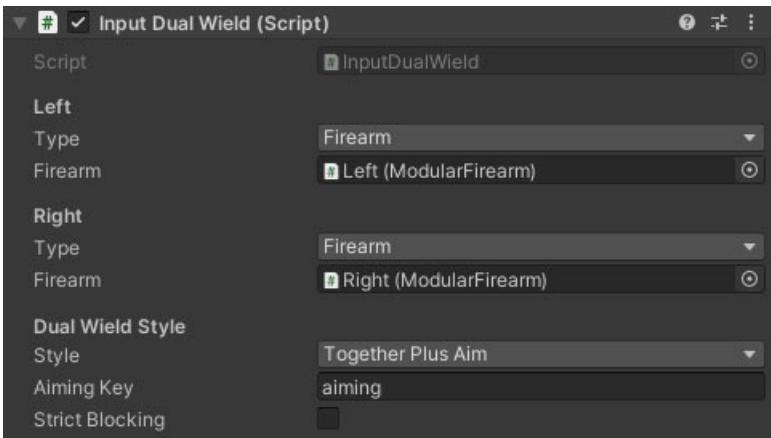
[Unity Input](#)

InputDualWield MonoBehaviour

Overview

The InputDualWield behaviour handles input for a dual wield weapon. This is actually 2 separate weapons parented to one object, and each one can be either a [modular firearm](#), [wieldable tool](#), [melee weapon](#) or [thrown weapon](#). The weapons can be triggered independently using the primary and secondary fire buttons, or set to both trigger on primary fire and allow aiming down sights instead.

Inspector



Properties

Each of the **Left** and **Right** weapon sections has the following properties:

NAME	TYPE	DESCRIPTION
Type	Dropdown	The type of weapon to use.
Firearm	ModularFirearm	The modular firearm behaviour for this weapon. This property will only be visible if the Type property is set to this type.
WieldableTool	WieldableTool	The wieldable tool behaviour for this weapon. This property will only be visible if the Type property is set to this type.
Melee	MeleeWeapon	The melee weapon behaviour for this weapon. This property will only be visible if the Type property is set to this type.
Thrown	ThrownWeapon	The thrown weapon behaviour for this weapon. This property will only be visible if the Type property is set to this type.

The InputDualWield behaviour also has the following shared properties:

NAME	TYPE	DESCRIPTION
Style	Dropdown	How should the dual wielding work. PrimarySecondary fires one weapon with primary fire and the other with secondary. TogetherPlusAim fires both weapons at the same time and can be aimed (see the docs for more info).
Aiming Key	String	The property key for the character motion graph (switch parameter).

NAME	TYPE	DESCRIPTION
Strict Blocking	Boolean	Either weapon blocks the other. For example, when one weapon reloads, the other won't be able to do anything.

See Also

[NeoFPS Input](#)

[Unity Input](#)

[Modular Firearms](#)

[Wieldable Tools](#)

[Melee Weapons](#)

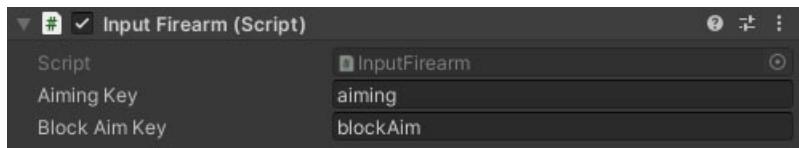
[Thrown Weapons](#)

InputFirearm MonoBehaviour

Overview

The InputFirearm behaviour handles input for the modular firearms

Inspector



Properties

NAME	TYPES	DESCRIPTION
Aiming Key	String	The key for a switch parameter on the character's motion graph to set when aiming. This allows slowing movement, preventing jumping, etc.
Block Aim Key	String	The key for a switch parameter on the character's motion graph to read which prevents the character from aiming down sights (eg when falling or sprinting).

See Also

[NeoFPS Input](#)

[Unity Input](#)

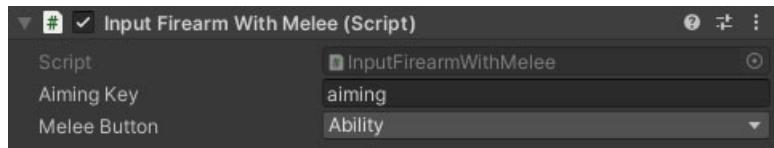
[Modular Firearms](#)

InputFirearmWithMelee MonoBehaviour

Overview

The InputFirearmWithMelee behaviour extends the [InputFirearm](#) behaviour to allow triggering a melee weapon attached to the firearm.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Melee Button	Fps Input Button	The input button that will be checked to trigger the melee attack.

See Also

[NeoFPS Input](#)

[Unity Input](#)

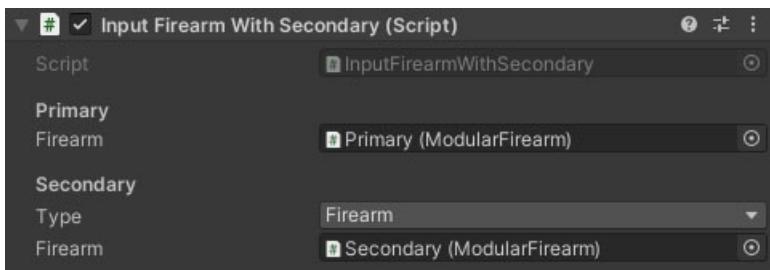
[Modular Firearms](#)

InputFirearmWithSecondary MonoBehaviour

Overview

The InputFirearmWithSecondary behaviour handles input for a modular firearm with another weapon as a secondary behaviour. This secondary weapon can be either a [modular firearm](#), [wieldable tool](#), [melee weapon](#) or [thrown weapon](#). The secondary weapon is triggered via the secondary fire button (meaning that the firearm cannot aim down sights), and when either weapon is blocked (for example when the firearm is reloading) the other will also be blocked at the same time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Firearm	String	The primary firearm.

The secondary weapon section has the following properties:

NAME	TYPE	DESCRIPTION
Type	Dropdown	The type of weapon to use for the secondary.
Firearm	ModularFirearm	The modular firearm behaviour for the secondary. This property will only be visible if the Type property is set to this type.
WieldableTool	WieldableTool	The wieldable tool behaviour for the secondary. This property will only be visible if the Type property is set to this type.
Melee	MeleeWeapon	The melee weapon behaviour for the secondary. This property will only be visible if the Type property is set to this type.
Thrown	ThrownWeapon	The thrown weapon behaviour for the secondary. This property will only be visible if the Type property is set to this type.

See Also

[NeoFPS Input](#)

[Unity Input](#)

[Modular Firearms](#)

[Wieldable Tools](#)

[Melee Weapons](#)

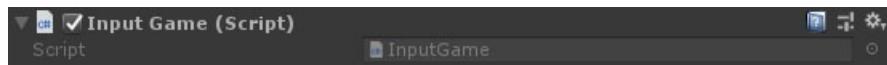
[Thrown Weapons](#)

InputGame MonoBehaviour

Overview

The InputGame behaviour is a placeholder input handler for game specific input (currently does nothing).

Inspector



Properties

The InputGame behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

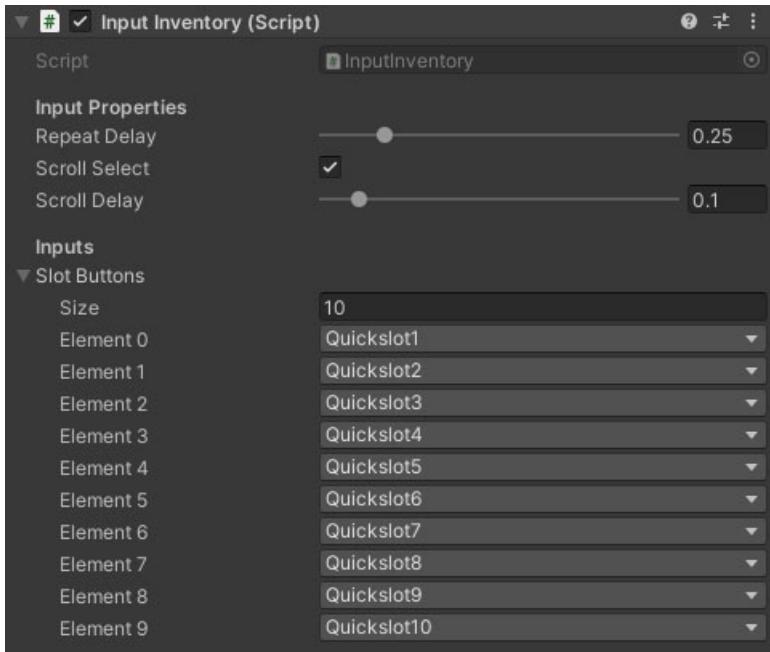
[Unity Input](#)

InputInventory MonoBehaviour

Overview

The InputInventory manages the character inventory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Repeat Delay	Float	The delay between repeating input when holding the next or previous weapon buttons.
Scroll Select	Boolean	Should the mouse scroll wheel switch between weapons.
Scroll Delay	Float	The delay between repeating input when rolling the mouse scroll wheel.
Slot Buttons	Fps Input Button Array	The input buttons corresponding to each slot. If you have quick-melee / thrown inputs then you can map them to specific slots here.

See Also

[NeoFPS Input](#)

[Unity Input](#)

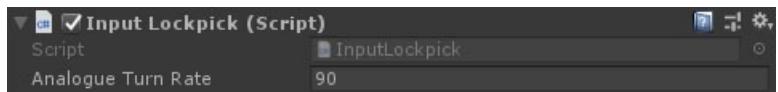
[The Inventory](#)

InputLockpick MonoBehaviour

Overview

The InputLockpick behaviour uses the NeoFPS input system to control a simple [lock-picking mini-game](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Analogue Turn Rate	Float	The maximum turn rate of the pick object in degrees per second.

See Also

[NeoFPS Input](#)

[Unity Input](#)

[LockPickPopup3D Behaviour](#)

InputMeleeWeapon MonoBehaviour

Overview

The InputMeleeWeapon behaviour sends input to a [MeleeWeapon](#) behaviour on the same object as this.

Inspector



Properties

The InputMeleeWeapon behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

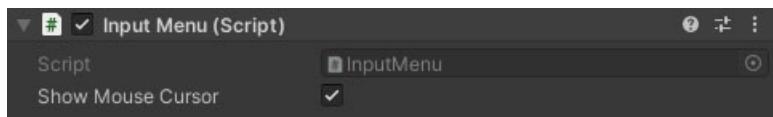
[Unity Input](#)

InputMenu MonoBehaviour

Overview

The InputMenu behaviour is added to UI elements that should block character input when visible. It pushes the **FpsInputModule.Menu** context when enabled and removes it when disabled. An example would be adding this to the root of the dialogue UI of another asset that handles conversations with NPCs. When the UI is activated, the character input will stop and allow you to chat to the NPC without accidentally shooting them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Show Mouse Cursor	Boolean	Should the mouse cursor be released when this object is activated (or component enabled). Set to false for things like loading screens.

See Also

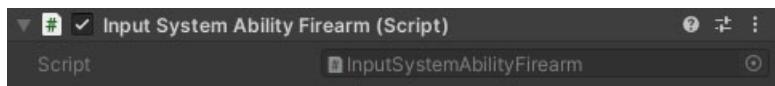
[NeoFPS Input](#)

InputSystemAbilityFirearm MonoBehaviour

Overview

The InputSystemAbilityFirearm behaviour reads the **Ability** button and uses it to fire the [ModularFirearm](#) that it's attached to. These firearms can be placed in the character hierarchy outside of the inventory system to create things like shoulder mounter launchers. This component is the [input system](#) equivalent of the `[InputAbilityFirearm][inputref-mb-inputabilityfirearm.md]` input handler.

Inspector



Properties

The InputSystemAbilityFirearm behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Modular Firearms](#)

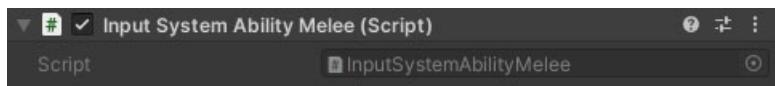
[Input System Extension](#)

InputSystemAbilityMelee MonoBehaviour

Overview

The InputSystemAbilityMelee behaviour reads the **Ability** button and uses it to trigger the [melee weapon](#) that it's attached to. These melee weapons can be placed in the character hierarchy outside of the inventory system and then used in parallel with a held weapon. This component is the [input system](#) equivalent of the [\[InputAbilityMelee\]\[inputref-mb-inputabilitymelee.md\]](#) input handler.

Inspector



Properties

The InputSystemAbilityMelee behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Melee Weapons](#)

[Input System Extension](#)

InputSystemAbilityWieldableTool MonoBehaviour

Overview

The InputSystemAbilityWieldableTool behaviour reads the **Ability** button and uses it to trigger the [wieldable tool](#) that it's attached to. These wieldable tools can be placed in the character hierarchy outside of the inventory system to create things like instant heals or shield recharges. This component is the [input system](#) equivalent of the [!InputAbilityWieldableTool][inputref-mb-inputabilitywieldabletool.md] input handler.

Inspector



Properties

The InputSystemAbilityWieldableTool behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Wieldable Tools](#)

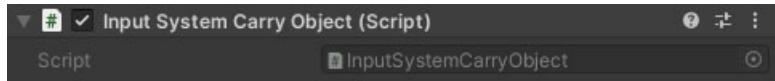
[Input System Extension](#)

InputSystemCarryObject MonoBehaviour

Overview

The InputSystemCarryObject behaviour communicates player input to the character's [carry system](#). It is the [input system](#) equivalent of the [InputCarryObject][inputref-mb-inputcarryobject.md] input handler.

Inspector



Properties

The InputSystemCarryObject behaviour has no properties exposed in the inspector.

See Also

[Carry System](#)

[NeoFPS Input](#)

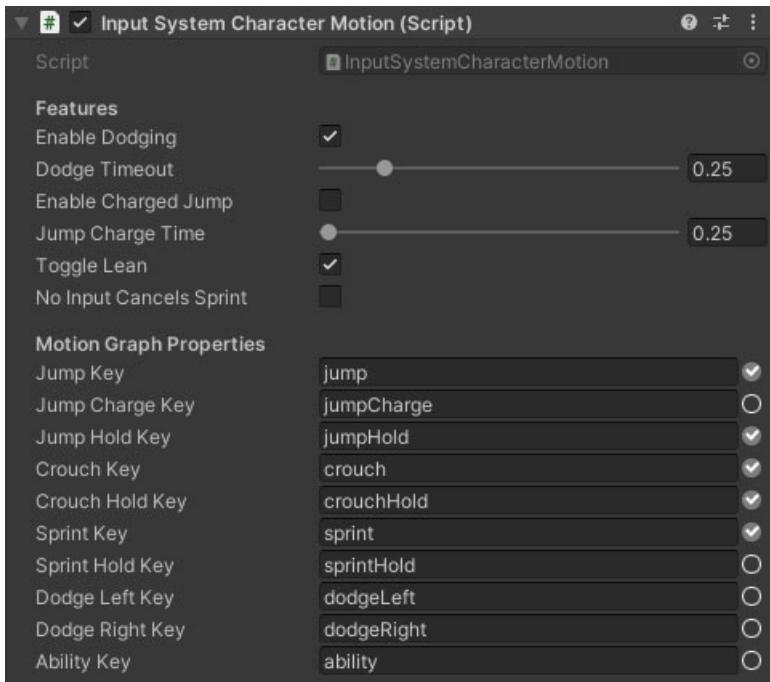
[Input System Extension](#)

InputSystemCharacterMotion MonoBehaviour

Overview

The InputSystemCharacterMotion behaviour passes character input to the character [motion controller](#). It is the [input system](#) equivalent of the [InputCharacterMotion][inputref-mb-inputchartermotion.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Enable Dodging	Boolean	Should double tapping a move direction dodge the character.
Dodge Timeout	Float	Multiple taps of a direction within this time range register as a double tap.
Enable Charged Jump	Boolean	Does holding the jump button charge up a jump or does the character dodge as soon as the button is pressed.
Jump Charge Time	Float	The time it takes to charge up a full power jump if charged jumps are enabled.
Toggle Lean	Boolean	Toggle leaning or hold to lean.
Jump Key	String	The key to the jump trigger parameter in the character motion graph .
Jump Charge Key	String	The optional key to the jump charge float parameter in the character motion graph . This is used for charging stronger jumps.
Jump Hold Key	String	The key to the jump hold switch parameter in the character motion graph . This is used for movement like swimming or flying where holding jump moves up.

NAME	TYPE	DESCRIPTION
Crouch Key	String	The key to the crouch switch parameter in the character motion graph .
Crouch Hold Key	String	The optional key to the crouch hold switch parameter in the character motion graph . This is used for movement like swimming or flying where holding jump moves down.
Sprint Key	String	The key to the sprint switch parameter in the character motion graph .
Sprint Hold Key	String	The key to the sprint hold switch parameter in the character motion graph .
Dodge Left Key	String	The optional key to the dodge left trigger parameter in the character motion graph .
Dodge Right Key	String	The optional key to the dodge right trigger parameter in the character motion graph .
Ability Key	String	The optional key to the "ability" trigger parameter in the character motion graph . This can be used for a variety of uses such as dashes, teleports, etc.
Custom Inputs	Array	Map input buttons to motion graph parameters here. motion graph .

Custom Input Entries

Each entry in the **Custom Inputs** array has the following properties:

NAME	TYPE	DESCRIPTION
Parameter Key	Boolean	The name of the parameter in the motion graph .
Type	Dropdown	The motion graph parameter type of the target parameter.
Button	Boolean	Which FpsInputButton to map to the parameter.

See Also

[NeoFPS Input](#)

[The Motion Graph](#)

[Input System Extension](#)

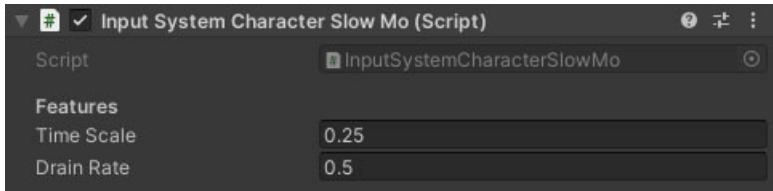
[Generated Constants](#)

InputSystemCharacterSlowMo MonoBehaviour

Overview

The InputSystemCharacterSlowMo behaviour reads the **Ability** button input and toggles slow-motion effects via the character's **SlowMoSystem**. It is the [input system](#) equivalent of the [InputCharacterSlowMo][inputref-mb-inputcharacterslowmo.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time-scale to use for ability based slow-mo.
Drain Rate	Float	The rate to drain slow-mo charge (time scale will return to normal when charge reaches zero).

See Also

[NeoFPS Input](#)

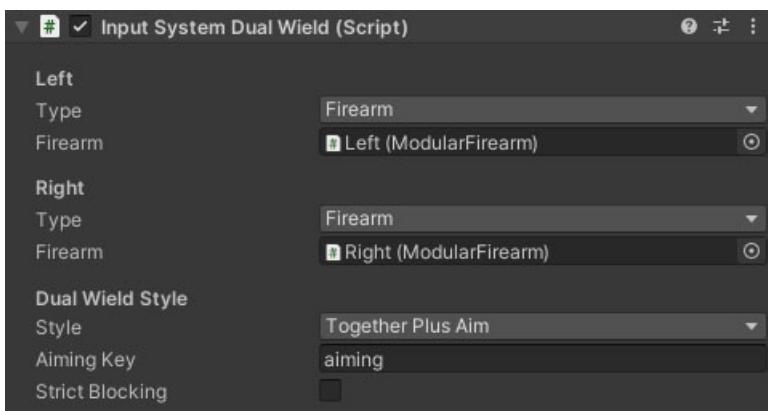
[Input System Extension](#)

InputSystemDualWield MonoBehaviour

Overview

The InputSystemDualWield behaviour handles input for a dual wield weapon. This is actually 2 separate weapons parented to one object, and each one can be either a [modular firearm](#), [wieldable tool](#), [melee weapon](#) or [thrown weapon](#). The weapons can be triggered independently using the primary and secondary fire buttons, or set to both trigger on primary fire and allow aiming down sights instead. It is the [input system](#) equivalent of the [InputDualWield][inputref-mb-inputdualwield.md] input handler.

Inspector



Properties

Each of the **Left** and **Right** weapon sections has the following properties:

NAME	TYPE	DESCRIPTION
Type	Dropdown	The type of weapon to use.
Firearm	ModularFirearm	The modular firearm behaviour for this weapon. This property will only be visible if the Type property is set to this type.
WieldableTool	WieldableTool	The wieldable tool behaviour for this weapon. This property will only be visible if the Type property is set to this type.
Melee	MeleeWeapon	The melee weapon behaviour for this weapon. This property will only be visible if the Type property is set to this type.
Thrown	ThrownWeapon	The thrown weapon behaviour for this weapon. This property will only be visible if the Type property is set to this type.

The InputSystemDualWield behaviour also has the following shared properties:

NAME	TYPE	DESCRIPTION
Style	Dropdown	How should the dual wielding work. PrimarySecondary fires one weapon with primary fire and the other with secondary. TogetherPlusAim fires both weapons at the same time and can be aimed (see the docs for more info).
Aiming Key	String	The property key for the character motion graph (switch parameter).

NAME	TYPE	DESCRIPTION
Strict Blocking	Boolean	Either weapon blocks the other. For example, when one weapon reloads, the other won't be able to do anything.

See Also

[NeoFPS Input](#)

[Input System Extension](#)

[Modular Firearms](#)

[Wieldable Tools](#)

[Melee Weapons](#)

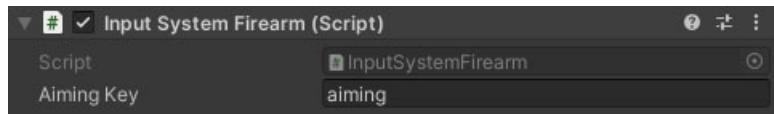
[Thrown Weapons](#)

InputSystemFirearm MonoBehaviour

Overview

The InputSystemFirearm behaviour handles input for the modular firearms. It is the [input system](#) equivalent of the [InputFirearm] [inputref-mb-inputfirearm.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Aiming Key	String	The property key for the character motion graph (switch parameter).

See Also

[NeoFPS Input](#)

[Modular Firearms](#)

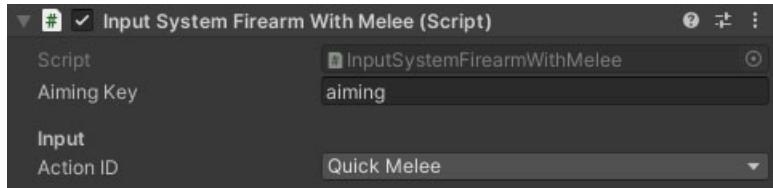
[Input System Extension](#)

InputSystemFirearmWithMelee MonoBehaviour

Overview

The InputSystemFirearmWithMelee behaviour extends the [InputSystemFirearm](#) behaviour to allow triggering a melee weapon attached to the firearm.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Melee Button	Input Action	The input action that will be checked to trigger the melee attack.

See Also

[NeoFPS Input](#)

[Modular Firearms](#)

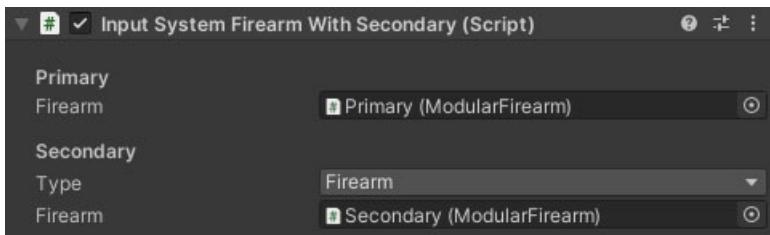
[Input System Extension](#)

InputSystemFirearmWithSecondary MonoBehaviour

Overview

The InputSystemFirearmWithSecondary behaviour handles input for a modular firearm with another weapon as a secondary behaviour. This secondary weapon can be either a [modular firearm](#), [wieldable tool](#), [melee weapon](#) or [thrown weapon](#). The secondary weapon is triggered via the secondary fire button (meaning that the firearm cannot aim down sights), and when either weapon is blocked (for example when the firearm is reloading) the other will also be blocked at the same time. This is the [input system](#) equivalent of the [InputFirearmWithSecondary][inputref-mb-inputfirearmwithsecondary.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Firearm	String	The primary firearm.

The secondary weapon section has the following properties:

NAME	TYPE	DESCRIPTION
Type	Dropdown	The type of weapon to use for the secondary.
Firearm	ModularFirearm	The modular firearm behaviour for the secondary. This property will only be visible if the Type property is set to this type.
WieldableTool	WieldableTool	The wieldable tool behaviour for the secondary. This property will only be visible if the Type property is set to this type.
Melee	MeleeWeapon	The melee weapon behaviour for the secondary. This property will only be visible if the Type property is set to this type.
Thrown	ThrownWeapon	The thrown weapon behaviour for the secondary. This property will only be visible if the Type property is set to this type.

See Also

[NeoFPS Input](#)

[Input System Extension](#)

[Modular Firearms](#)

[Wieldable Tools](#)

[Melee Weapons](#)

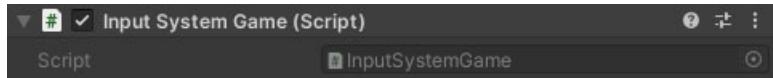
[Thrown Weapons](#)

InputSystemGame MonoBehaviour

Overview

The InputSystemGame behaviour is a placeholder input handler for game specific input (currently does nothing). It is the [input system](#) equivalent of the [InputGame][inputref-mb-inputgame.md] input handler.

Inspector



Properties

The InputSystemGame behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

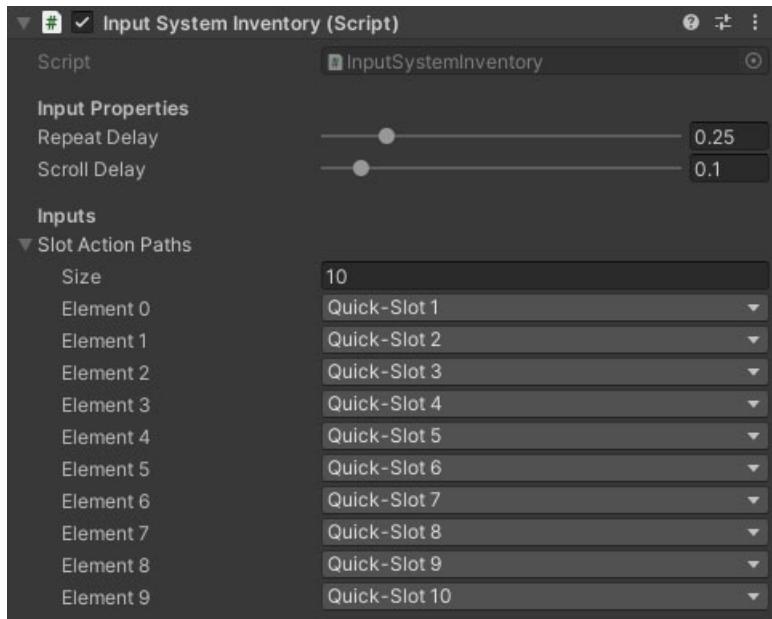
[Input System Extension](#)

InputSystemInventory MonoBehaviour

Overview

The InputSystemInventory manages the character inventory. It is the [input system](#) equivalent of the [InputInventory][inputref-mb-inputinventory.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Repeat Delay	Float	The delay between repeating input when holding the next or previous weapon buttons.
Scroll Delay	Float	The delay between repeating input when rolling the mouse scroll wheel.
Slot Buttons	Input Action Array	The input actions corresponding to each slot. If you have quick-melee / thrown inputs then you can map them to specific slots here.

See Also

[NeoFPS Input](#)

[The Inventory](#)

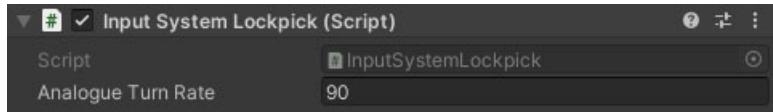
[Input System Extension](#)

InputSystemLockpick MonoBehaviour

Overview

The InputSystemLockpick behaviour uses the NeoFPS input system to control a simple [lock-picking mini-game](#). It is the [input system](#) equivalent of the [InputLockpick][inputref-mb-inputlockpick.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Analogue Turn Rate	Float	The maximum turn rate of the pick object in degrees per second.

See Also

[NeoFPS Input](#)

[LockPickPopup3D Behaviour](#)

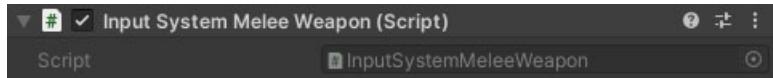
[Input System Extension](#)

InputSystemMeleeWeapon MonoBehaviour

Overview

The InputSystemMeleeWeapon behaviour sends input to a [MeleeWeapon](#) behaviour on the same object as this. It is the [input system][4] equivalent of the [InputMeleeWeapon][inputref-mb-inputmeleeweapon.md] input handler.

Inspector



Properties

The InputSystemMeleeWeapon behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Melee Weapons](#)

[\[Input System Extension\]\[4\]](#)

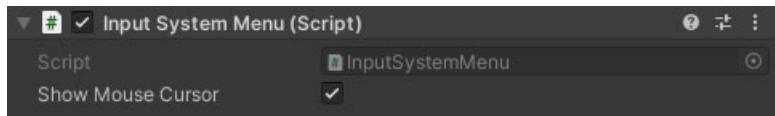
InputSystemMenu MonoBehaviour

Overview

The InputSystemMenu behaviour is added to UI elements that should block character input when visible. It pushes the **FpsInputContext.Menu** context when enabled and removes it when disabled. An example would be adding this to the root of the dialogue UI of another asset that handles conversations with NPCs. When the UI is activated, the character input will stop and allow you to chat to the NPC without accidentally shooting them.

This component is the [input system](#) equivalent of the [InputMenu][inputref-mb-inputmenu.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Show Mouse Cursor	Boolean	Should the mouse cursor be released when this object is activated (or component enabled). Set to false for things like loading screens.

See Also

[NeoFPS Input](#)

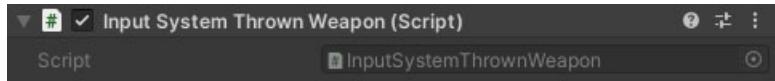
[Input System Extension](#)

InputSystemThrownWeapon MonoBehaviour

Overview

The InputSystemThrownWeapon behaviour sends input to a [ThrownWeapon](#) behaviour on the same game object. It is the [input system](#) equivalent of the `[InputThrownWeapon][inputref-mb-inputsystemthrownweapon.md]` input handler.

Inspector



Properties

The InputSystemThrownWeapon behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Thrown Weapons](#)

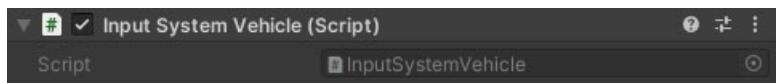
[Input System Extension](#)

InputSystemVehicle MonoBehaviour

Overview

The InputSystemVehicle behaviour is a placeholder input handler for vehicle input. When activated it will set the input context, disabling character input. Its implementation is currently blank, ready to be modified or inherited based on your needs. It is the [input system](#) equivalent of the [InputVehicle][inputref-mb-inputvehicle.md] input handler.

Inspector



Properties

The InputSystemVehicle behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

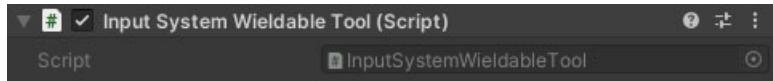
[Input System Extension](#)

InputSystemWieldableTool MonoBehaviour

Overview

The InputSystemWieldableTool behaviour sends input to a [WieldableTool](#) behaviour on the same game object. It is the [input system](#) equivalent of the [InputWieldableTool][inputref-mb-inputwieldabletool.md] input handler.

Inspector



Properties

The InputWieldableTool behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Wieldable Tools](#)

[Input System Extension](#)

InputThrownWeapon MonoBehaviour

Overview

The InputThrownWeapon behaviour sends input to a [ThrownWeapon](#) behaviour on the same game object.

Inspector



Properties

The InputThrownWeapon behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Thrown Weapons](#)

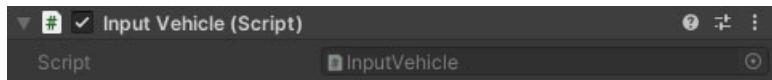
[Unity Input](#)

InputVehicle MonoBehaviour

Overview

The InputVehicle behaviour is a placeholder input handler for vehicle input. When activated it will set the input context, disabling character input. Its implementation is currently blank, ready to be modified on inherited based on your needs.

Inspector



Properties

The InputVehicle behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

InputWieldableTool MonoBehaviour

Overview

The InputWieldableTool behaviour sends input to a [WieldableTool](#) behaviour on the same game object.

Inspector



Properties

The InputWieldableTool behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

[Wieldable Tools](#)

[Unity Input](#)

MouseAndGamepadAimController MonoBehaviour

Overview

The MouseAndGamepadAimController behaviour handles mouse and gamepad input for moving the camera, including smoothing and acceleration.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Yaw Transform	Transform	The transform to yaw when aiming. This should be a parent of the pitch transform.
Aim Yaw Transform	Transform	This optional transform detaches the character direction from the aim direction.
Steering Rate	Float	The time taken to turn the character to the aim-yaw direction (if Aim Yaw Transform is set). 0 = call LerpYawToAim() manually, 1 = instant.
Pitch Transform	Transform	The transform to pitch when aiming. This should be a child of the yaw transform.
Max Pitch	Float	The maximum pitch from horizontal the aimer can rotate.
Constraints Damping	Float	The amount of damping applied when rotating the camera to match constraints.
Constraints Tolerance	Float	Once the angle outside constraints goes below this value, the camera will snap to the constraints. Larger values will have a visible effect.
YawConstraintsFalloff	Float	An angle range from the yaw constraint limits where the input falls off. This gives the effect of softer constraint limits instead of hitting an invisible wall.

Name	Type	Description
Mouse Turn Angle Min	Float	Number of degrees for 1 unit of mouse movement if sensitivity is set to 0.
Mouse Turn Angle Max	Float	Number of degrees for 1 unit of mouse movement if sensitivity is set to 1.
Relative To	Transform	The transform to calculate the input relative to. If the character can tilt left or right then this transform is required to prevent tilt messing up the yaw calculations.
Mouse Smoothing Buffer Size	Float	The number of frames to store and use for the mouse smoothing history.
Mouse Smoothing Multiplier Min	Float	The weight multiplier for the previous frame when averaging if the smoothing is set to minimum.
Mouse Smoothing Multiplier Max	Float	The weight multiplier for the previous frame when averaging if the smoothing is set to maximum.
Mouse Accel Speed Multiply Min	Float	The base acceleration multiplier when acceleration is set to the minimum.
Mouse Accel Speed Multiply Max	Float	The base acceleration multiplier when acceleration is set to the maximum.
Mouse Acceleration Max	Float	The maximum multiplier acceleration can apply to the mouse input (0 means no maximum).
Mouse Acceleration Type	Dropdown	Does mouse speed affect the input linearly or based on the square of the speed. Options are Linear , Quadratic .
Analog Turn Angle Min	Float	Number of degrees per second for the gamepad analog at its limit, if sensitivity is set to 0.
Analog Turn Angle Max	Float	Number of degrees per second for the gamepad analog at its limit, if sensitivity is set to 1.
Analog Curve	AnimationCurve	The input curve for analog input. This can be used to define a deadzone, and damp smaller movements.

See Also

[NeoFPS Input](#)

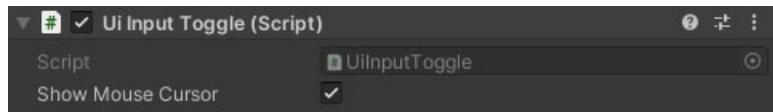
[Unity Input](#)

UiInputToggle MonoBehaviour

Overview

The UiInputToggle behaviour is used to toggle the mouse cursor and character controls on hitting the escape/menu button so that you can use different UIs and menus.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Show Mouse Cursor	Boolean	Should the mouse cursor be released when this object is activated (or component enabled). Set to false for things like loading screens.

See Also

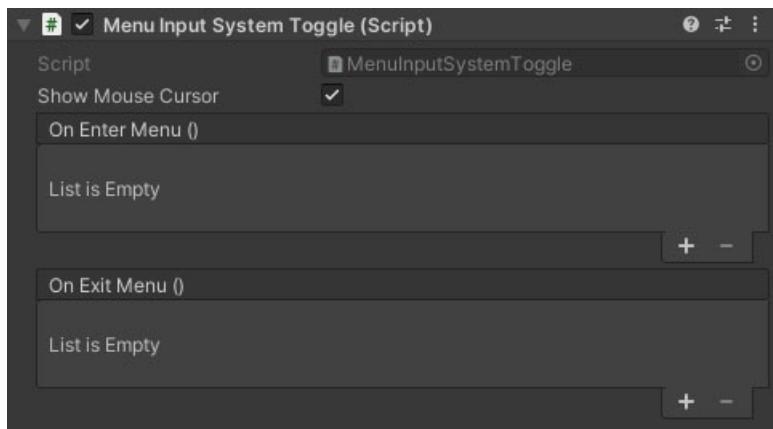
[NeoFPS Input](#)

MenuInputSystemToggle MonoBehaviour

Overview

The MenuInputSystemToggle behaviour is used to toggle the mouse cursor and character controls on hitting the escape/menu button so that you can use different UIs and menus. It is the [input system](#) equivalent of the [UiInputToggle][inputref-mb-uiinputtoggle.md] input handler.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Show Mouse Cursor	Boolean	Should the mouse cursor be released when this object is activated (or component enabled). Set to false for things like loading screens.

See Also

[NeoFPS Input](#)

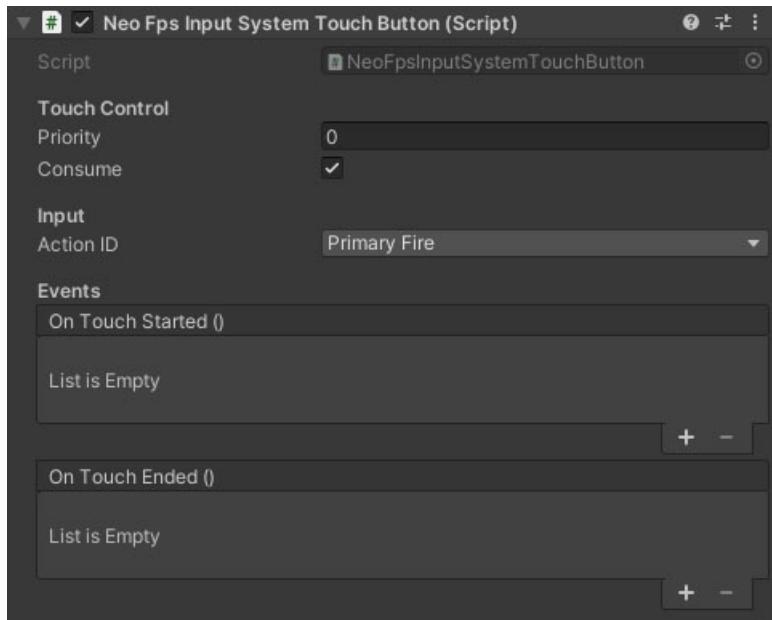
[Input System Extension](#)

NeoFpsInputSystemTouchButton MonoBehaviour

Overview

The NeoFpsInputSystemTouchButton behaviour is used to define the activation region on the HUD for a button style touch control, along with its response when tapped or held. This behaviour attaches to [NeoFpsInputSystemTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Action ID	Dropdown	The input action the control should affect. The dropdown will list all the available actions.
On Touch Started	UnityEvent	An event fired when the player first touches this control.
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[Input System Extension](#)

[Touchscreen Controls](#)

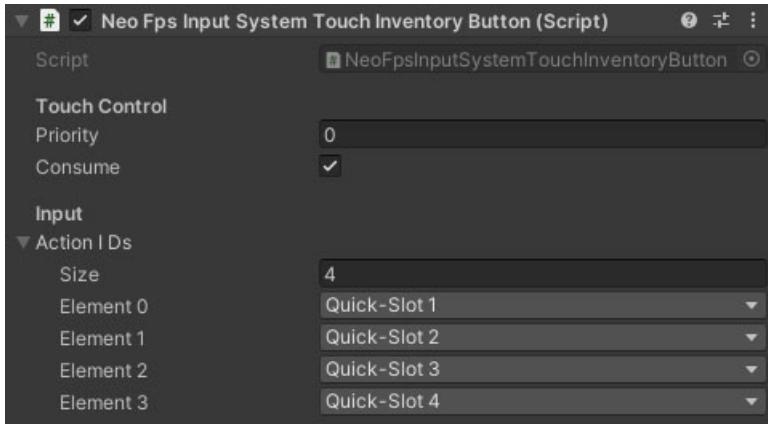
NeoFpsInputSystemTouchInventoryButton MonoBehaviour

Overview

The NeoFpsInputSystemTouchInventoryButton behaviour is used to define the activation region on the HUD for a button style touch control. This should be placed on the individual slot elements of a HUD inventory popup that is set to persist (not fade out). When tapped, it will signal the player character to switch to the inventory item in the tapped slot.

This behaviour attaches to [NeoFpsInputSystemTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.

See Also

[Input System Extension](#)

[Touchscreen Controls](#)

[Inventory](#)

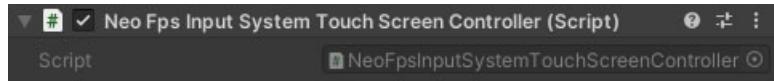
[The Player HUD](#)

NeoFpsInputSystemTouchScreenController MonoBehaviour

Overview

The NeoFpsInputSystemTouchScreenController behaviour is placed on the root object of your HUD touch controls. This is what actually detects and manages touches, whilst the child touch-controls define their activation regions and touch response.

Inspector



Properties

The NeoFpsInputSystemTouchScreenController behaviour has no properties exposed in the inspector.

See Also

[Input System Extension](#)

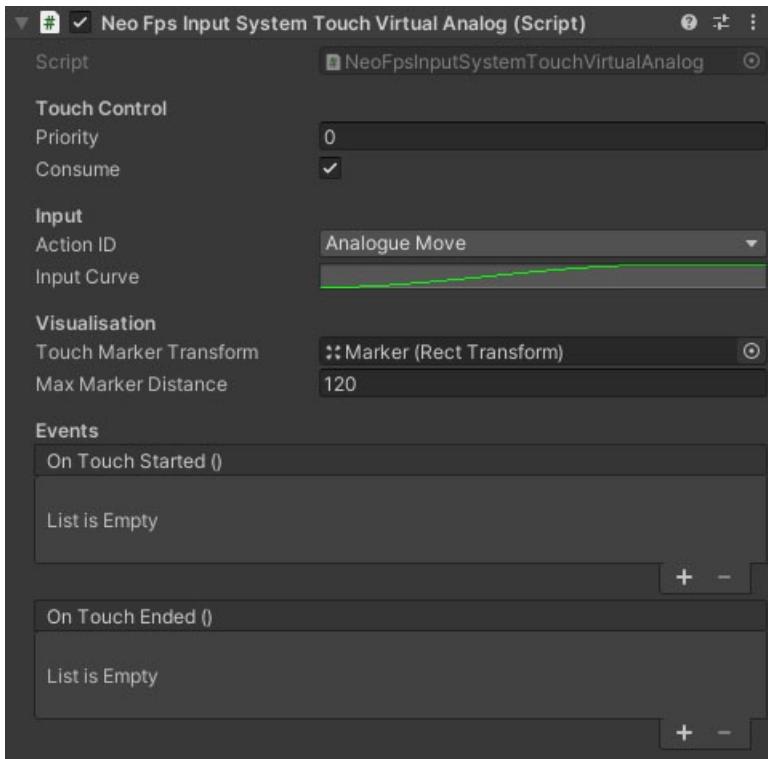
[Touchscreen Controls](#)

NeoFpsInputSystemTouchVirtualAnalog MonoBehaviour

Overview

The NeoFpsInputSystemTouchVirtualAnalog behaviour is used to define the activation region on the HUD for a virtual analog stick which emulates the sticks on a gamepad. This behaviour attaches to [NeoFpsInputSystemTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Action ID	Dropdown	The input action the control should affect. The dropdown will list all the available actions.
Input Curve	AnimationCurve	A falloff curve for the input strength. The horizontal axis is the normalised distance of the touch from the center of the control. The vertical axis is the output strength.
Touch Marker Transform	RectTransform	A visual marker for the analog position that appears while touch is held.
Max Marker Distance	Float	The distance the marker should move from the center of the analog control when at full strength.
On Touch Started	UnityEvent	An event fired when the player first touches this control.

NAME	TYPE	DESCRIPTION
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[Input System Extension](#)

[Touchscreen Controls](#)

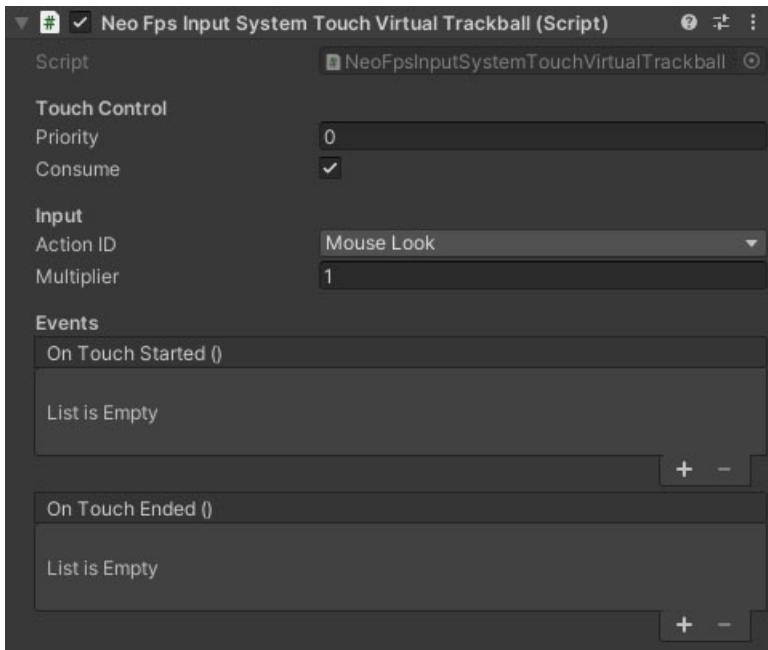
NeoFpsInputSystemTouchVirtualTrackball MonoBehaviour

Overview

The NeoFpsInputSystemTouchVirtualTrackball behaviour is used to define the activation region on the HUD for a virtual trackball / trackpad style input that you can use to emulate mouse input. This behaviour attaches to

[NeoFpsInputSystemTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Action ID	Dropdown	The input action the control should affect. The dropdown will list all the available actions.
Multiplier	Float	A multiplier applied to the touch delta.
On Touch Started	UnityEvent	An event fired when the player first touches this control.
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[Input System Extension](#)

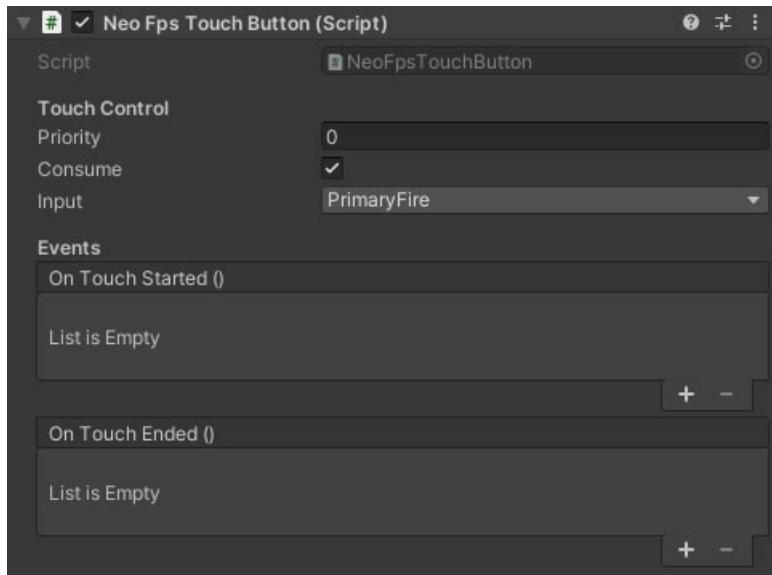
[Touchscreen Controls](#)

NeoFpsTouchButton MonoBehaviour

Overview

The NeoFpsTouchButton behaviour is used to define the activation region on the HUD for a button style touch control, along with its response when tapped or held. This behaviour attaches to [NeoFpsTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Input	FpsInputButton	The button to emulate.
On Touch Started	UnityEvent	An event fired when the player first touches this control.
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[NeoFPS Input](#)

[Touchscreen Controls](#)

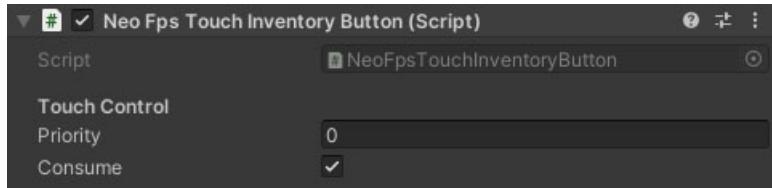
NeoFpsTouchInventoryButton MonoBehaviour

Overview

The NeoFpsTouchInventoryButton behaviour is used to define the activation region on the HUD for a button style touch control. This should be placed on the individual slot elements of a HUD inventory popup that is set to persist (not fade out). When tapped, it will signal the player character to switch to the inventory item in the tapped slot.

This behaviour attaches to [NeoFpsTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.

See Also

[NeoFPS Input](#)

[Touchscreen Controls](#)

[Inventory](#)

[The Player HUD](#)

NeoFpsTouchScreenController MonoBehaviour

Overview

The NeoFpsTouchScreenController behaviour is placed on the root object of your HUD touch controls. This is what actually detects and manages touches, whilst the child touch-controls define their activation regions and touch response.

Inspector



Properties

The NeoFpsTouchScreenController behaviour has no properties exposed in the inspector.

See Also

[NeoFPS Input](#)

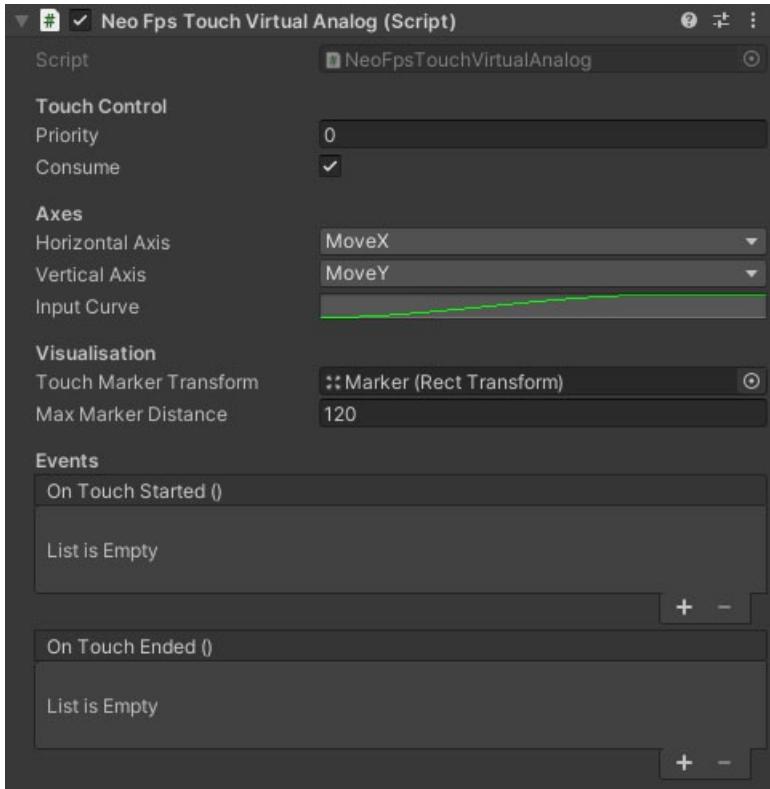
[Touchscreen Controls](#)

NeoFpsTouchVirtualAnalog MonoBehaviour

Overview

The NeoFpsTouchVirtualAnalog behaviour is used to define the activation region on the HUD for a virtual analog stick which emulates the sticks on a gamepad. This behaviour attaches to [NeoFpsTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Horizontal Axis	FpsInputAxis	The axis to apply the virtual analog stick's horizontal axis to.
Vertical Axis	FpsInputAxis	The axis to apply the virtual analog stick's vertical axis to.
Input Curve	AnimationCurve	A falloff curve for the input strength. The horizontal axis is the normalised distance of the touch from the center of the control. The vertical axis is the output strength.
Touch Marker Transform	RectTransform	A visual marker for the analog position that appears while touch is held.
Max Marker Distance	Float	The distance the marker should move from the center of the analog control when at full strength.

NAME	TYPE	DESCRIPTION
On Touch Started	UnityEvent	An event fired when the player first touches this control.
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[NeoFPS Input](#)

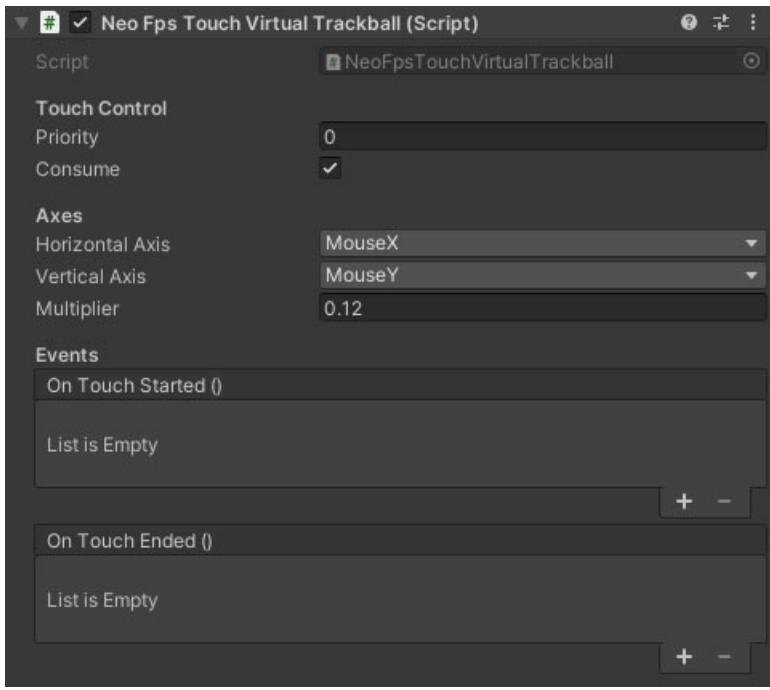
[Touchscreen Controls](#)

NeoFpsTouchVirtualTrackball MonoBehaviour

Overview

The NeoFpsTouchVirtualTrackball behaviour is used to define the activation region on the HUD for a virtual trackball / trackpad style input that you can use to emulate mouse input. This behaviour attaches to [NeoFpsTouchScreenController](#) behaviour in its parent hierarchy whilst enabled, which is what actually detects the touches and picks which control should handle them.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Priority	Integer	The priority this control should have in terms of appearing over another. Higher numbers will cover controls with lower priority.
Consume	Boolean	Should the touch input be consumed or fall through to the control underneath this.
Horizontal Axis	FpsInputAxis	The axis to apply the horizontal touch delta to.
Vertical Axis	FpsInputAxis	The axis to apply the vertical touch delta to.
Multiplier	Float	A multiplier applied to the touch delta.
On Touch Started	UnityEvent	An event fired when the player first touches this control.
On Touch Ended	UnityEvent	An event fired when a touch that started on this control is released.

See Also

[NeoFPS Input](#)

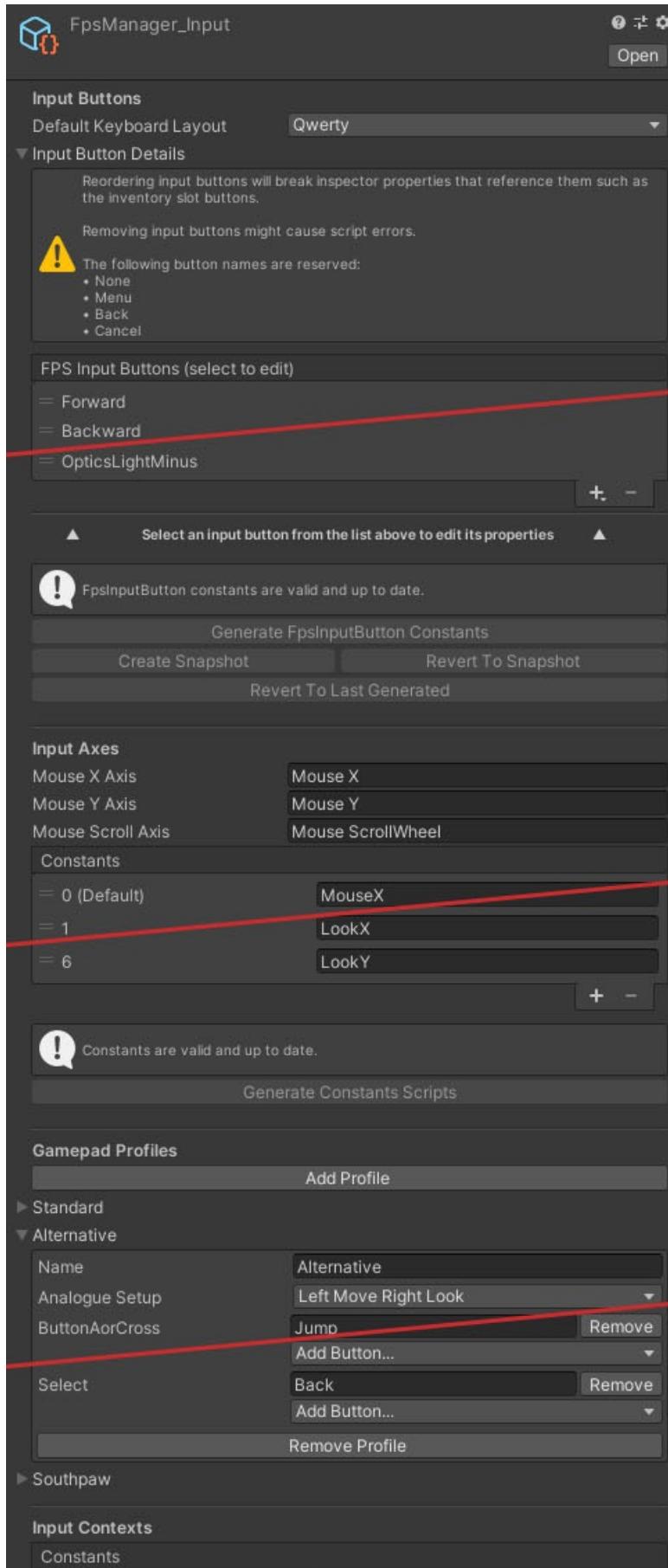
Touchscreen Controls

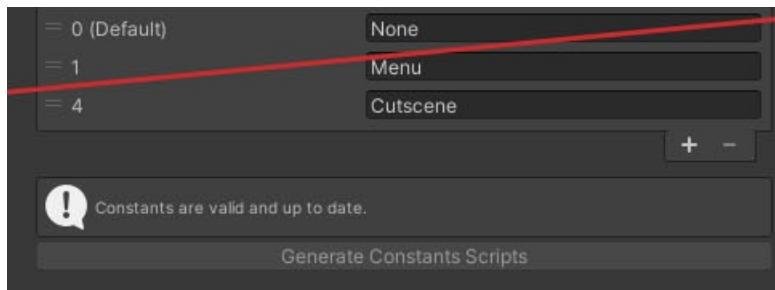
NeoFpsInputManager ScriptableObject

Overview

The NeoFpsInputManager scriptable object handles the

Inspector





Properties

The NeoFpsInputManager inspector is split into multiple sections:

Input Buttons

The input buttons section defines the button or key style controls for the game, as well as their default key bindings.

NAME	TYPE	DESCRIPTION
Default Keyboard Layout	Dropdown	The keyboard layout that the buttons below are set up for (the system converts to qwerty and then on to the required layout when resetting to defaults. Available options are: Qwerty, Azerty, Qwertz, Dvorak, Colemak .

Each button has the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The input button name, as it was referenced in code.
Display Name	String	The name as shown in the key binding game options.
Category	Dropdown	The category of the button, used to organise the buttons in the key binding game options.
Context	Dropdown	The context is used to define when key bindings can be shared by multiple input buttons. An example is weapons that can be aimed, vs weapons that have a secondary function.
Default Keys	Dropdown(s)	The primary and secondary key bindings for the input button. These will be overridden by the player's key binding settings.

Changing any of the buttons in the array or adding / removing constants requires the script to be regenerated. There are also a number of predefined buttons that will also be added for use in menus (None, Menu, Back and Cancel).

The **Generate FpsInputButton Constants** button will generate the button constants script. **Revert To Last Generated** will undo any changes to the buttons to the values when the script was last generated. **Create Snapshot** will store the current button setup, allowing you to revert back to this state using the **Revert To Snapshot** button. The snapshot will be reset when the script is generated.

Input Axes

The input axes are used to track analogue and variable axes in input handlers. The input axes are a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
Mouse X Axis	String	The name of the mouse x axis in the Unity input settings.
Mouse Y Axis	String	The name of the mouse y axis in the Unity input settings.
Mouse Scroll Axis	String	The name of the mouse scroll wheel axis in the Unity input settings.
Constants	String Array	The names of the input axes.

Gamepad Profiles

Each gamepad profile under this section can be expanded to map its controls. The **Name** property is the profile name as shown in the game's gamepad options. The **Analogue Setup** property defines what each analogue stick on the gamepad controls. The rest of the properties are mappings for each of the gamepad buttons based on an XBox or PS4 style controller (2 analogues, triggers, d-pad and buttons). Gamepads are detected when connected, and the primary gamepad will be mapped to the profile for player character control. Each gamepad button can be mapped to multiple inputs. The **Add Button...** dropdown will show all of the valid inputs that you can apply. If the gamepad button already has an input then this will only show inputs that are able to clash with this one.

Input Contexts

Input contexts are used by input handlers to define which handler should process input and which should be ignored at any one time. Pushing a context to the stack will only allow input handlers with that context to process input. The input context is a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION
Constants	String Array	The names of the input contexts.

See Also

[NeoFPS Input](#)

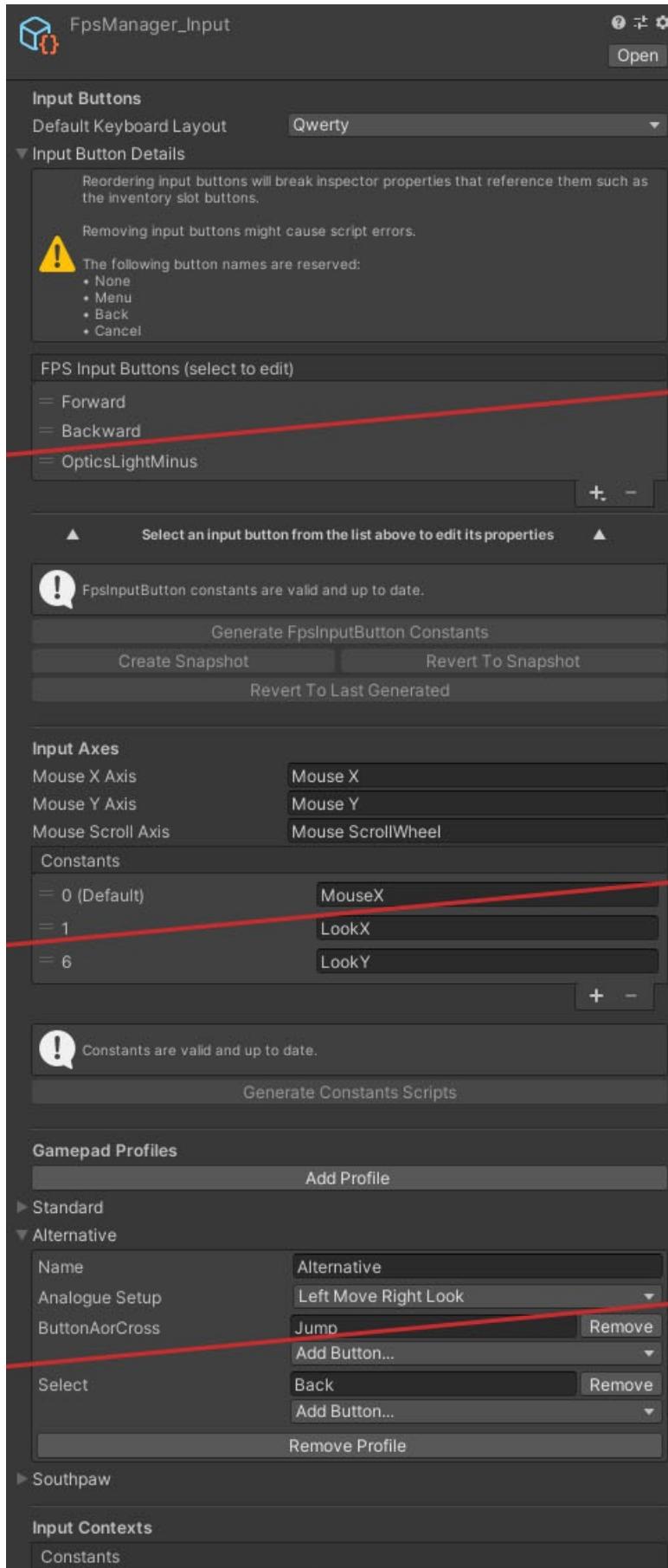
[Unity Input](#)

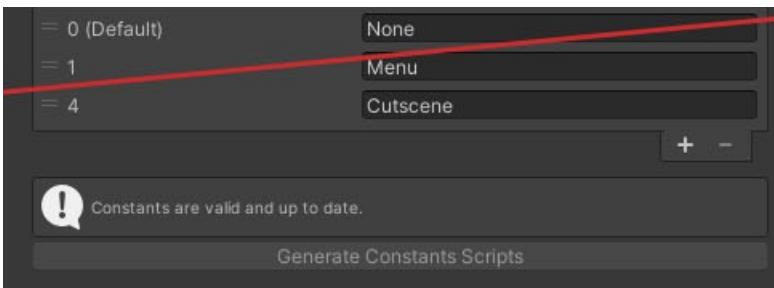
NeoFpsInputManager ScriptableObject

Overview

The NeoFpsInputManager scriptable object handles the

Inspector





Properties

The NeoFpsInputManager inspector is split into multiple sections:

Input Buttons

The input buttons section defines the button or key style controls for the game, as well as their default key bindings.

NAME	TYPE	DESCRIPTION
Default Keyboard Layout	Dropdown	The keyboard layout that the buttons below are set up for (the system converts to qwerty and then on to the required layout when resetting to defaults. Available options are: Qwerty, Azerty, Qwertz, Dvorak, Colemak .

Each button has the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The input button name, as it was referenced in code.
Display Name	String	The name as shown in the key binding game options.
Category	Dropdown	The category of the button, used to organise the buttons in the key binding game options.
Context	Dropdown	The context is used to define when key bindings can be shared by multiple input buttons. An example is weapons that can be aimed, vs weapons that have a secondary function.
Default Keys	Dropdown(s)	The primary and secondary key bindings for the input button. These will be overridden by the player's key binding settings.

Changing any of the buttons in the array or adding / removing constants requires the script to be regenerated. There are also a number of predefined buttons that will also be added for use in menus (None, Menu, Back and Cancel).

The **Generate FpsInputButton Constants** button will generate the button constants script. **Revert To Last Generated** will undo any changes to the buttons to the values when the script was last generated. **Create Snapshot** will store the current button setup, allowing you to revert back to this state using the **Revert To Snapshot** button. The snapshot will be reset when the script is generated.

Input Axes

The input axes are used to track analogue and variable axes in input handlers. The input axes are a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
Mouse X Axis	String	The name of the mouse x axis in the Unity input settings.
Mouse Y Axis	String	The name of the mouse y axis in the Unity input settings.
Mouse Scroll Axis	String	The name of the mouse scroll wheel axis in the Unity input settings.
Constants	String Array	The names of the input axes.

Gamepad Profiles

Each gamepad profile under this section can be expanded to map its controls. The **Name** property is the profile name as shown in the game's gamepad options. The **Analogue Setup** property defines what each analogue stick on the gamepad controls. The rest of the properties are mappings for each of the gamepad buttons based on an XBox or PS4 style controller (2 analogues, triggers, d-pad and buttons). Gamepads are detected when connected, and the primary gamepad will be mapped to the profile for player character control. Each gamepad button can be mapped to multiple inputs. The **Add Button...** dropdown will show all of the valid inputs that you can apply. If the gamepad button already has an input then this will only show inputs that are able to clash with this one.

Input Contexts

Input contexts are used by input handlers to define which handler should process input and which should be ignored at any one time. Pushing a context to the stack will only allow input handlers with that context to process input. The input context is a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION
Constants	String Array	The names of the input contexts.

See Also

[NeoFPS Input](#)

[Unity Input](#)

Interaction With The World

Overview

Unity and NeoFPS provide a number of ways for characters to interact with the world.

Interactive Objects

NeoFPS also has its own system of [interactive objects](#). These allow the player to use items in order to trigger actions.

[Doors](#) are a good example of interactive objects, and NeoFPS includes a number of examples to use as reference.

Trigger Zones

Trigger zones are a fundamental concept for level design and game design as a whole. For information on how to make use of them, please see the [Unity documentation](#).

NeoFPS has a number of helper behaviours that simplify using triggers to drive game mechanics.

The [CharacterZoneTrigger](#) and [CharacterZoneTriggerPersistant](#) fire code events when a character enters. These events are not tied to a specific implementation of the NeoFPS character and the only requirement is that they inherit from the `ICharacter` interface. Some scripting is required to use these behaviours.

The [SoloCharacterZoneTrigger](#) and [SoloCharacterZoneTriggerPersistant](#) fire [Unity events](#) when an [FpsSoloCharacter](#) enters. As such, they require less scripting knowledge to use, but they are constrained to the specific character implementation or characters that derive from this.

See Also

[Interactive Objects](#)

[Doors and Locks](#)

[Unity Colliders](#)

Interactive Objects

Overview

Interactive objects are objects in the scene that the player can approach and *use* to perform some action.

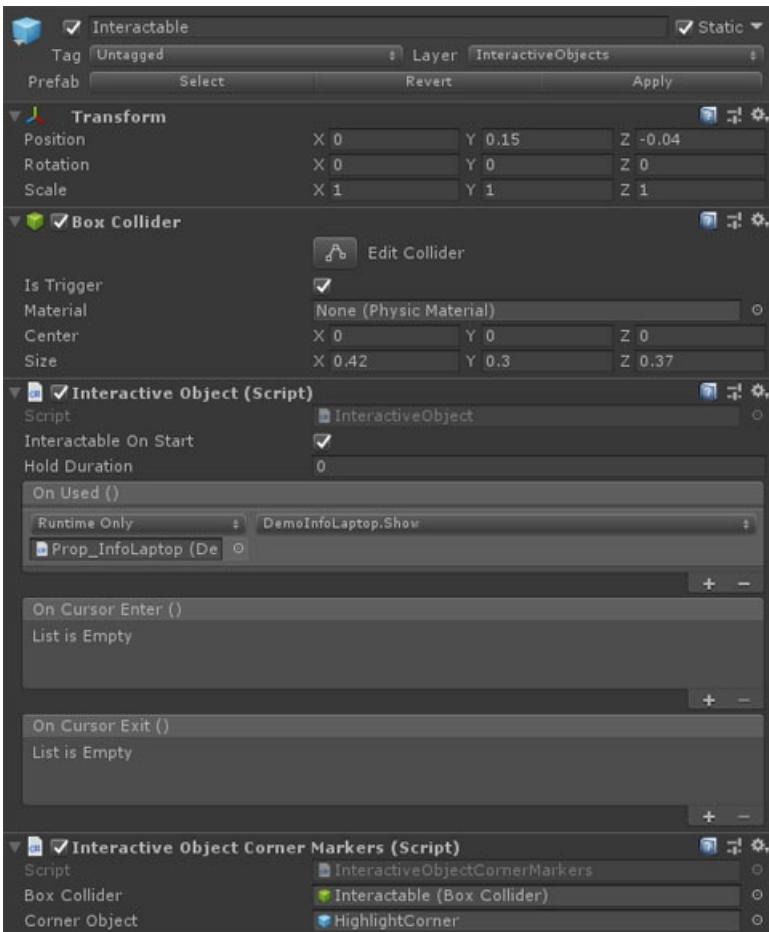
Objects can specify whether to react instantly or once the use button has been held for a set period of time.

As the player looks at an interactive object, events are fired that allow custom highlights and effects to be added. This can be utilised to show markers as in the sample assets, to change a shader setting, or to flag up an object with a HUD element.

Example interactive objects include doors, buttons and weapon pickups.

For more information see the [CharacterInteractionHandler](#) and [InteractiveObject](#) references.

Creating An Interactive Object



Interactive objects use events to call methods on attached components and perform some function.

To make an object interactive, first add a `GameObject` to its hierarchy and set its layer to **InteractiveObjects**. Next, add a primitive collider to the new object and set its **IsTrigger** property to true. With the trigger collider set up, add an `InteractiveObject` component to the object and set its hold duration (0 means the object will be used instantly when the player hits the use button, anything else means the player needs to hold the use button down for that duration). Finally, set the **OnUse** event on the component to point to a relevant method on one of the original object's components.

Alternatively, you can write a script that inherits from `InteractiveObject` and override the `Interact (ICharacter character)` method to add implement the required behaviour.

See Also

[CharacterInteractionHandler](#)

InteractiveObject

Doors and Locks

Overview

NeoFPS includes a number of example doors. Each of which can be opened and closed using either trigger zones or interactive objects.

Kinematic Hinge Doors



These are swing doors that use simple procedural animation to open or close. They use an optional double hinge system to allow for opening in both directions without overlapping into the frame.

Kinematic doors will push the player back and do not react to physics in the scene.

For more information see the [KinematicHingeDoor](#) reference.

Physics Hinge Doors



These are swing doors that use a Unity HingeJoint to drive the door's animation. These doors react physically to the environment and to player actions.

Physics hinge doors can only open in one direction and will automatically latch when closed.

For more information see the [PhysicsHingeDoor](#) reference.

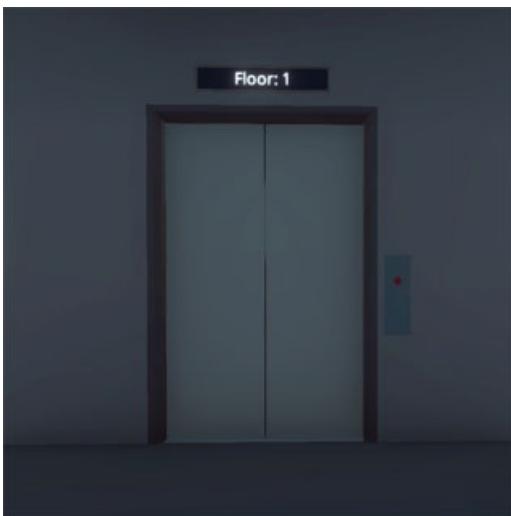
Sliding Doors



Sliding doors involve one or more sliding components and are animated procedurally.

For more information see the [SlidingDoor](#) reference.

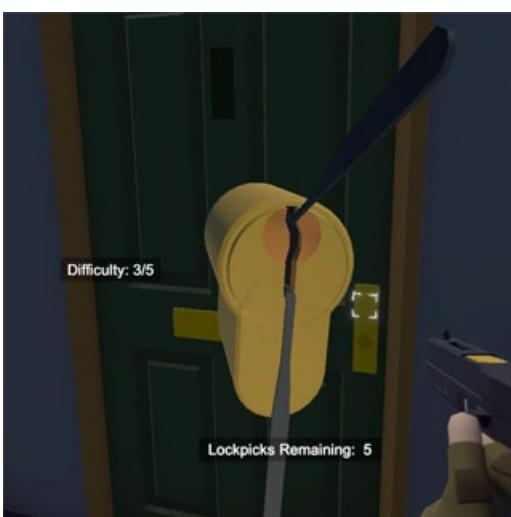
Elevators Doors



Elevators are a complex system of [interactive objects](#), [moving platforms](#) and sliding doors. The elevator cab moves up and down between floors. Only the doors for the floor the cab is currently on can open and only when the cab has stopped moving. The cab inner door sections are shared with each of the outer doors, making them sync up as any of those are opened.

For more information see the [ElevatorController](#) reference.

Locked Doors



NeoFPS also features a system for locking and unlocking doors. The majority of locks rely on the keyring system. The [KeyRing](#) is an inventory object which tracks the key IDs that a character knows. This can include keycodes for entering into keypads, as well as keys for physical locks. Picking up a keyring will merge the contents with the one in the character inventory.

When you set up a locked door, you specify a key ID. If no ID is set, then the door must be unlocked via an event or the scripting API. As an example, events can be used to unlock a door on destroying an object. This is how the destructible padlock demo is set up in the doors demo scene. The API is used by the lockpicking mini-game to unlock the door if the player has lockpicks in their inventory but does not have the door's key.

The following components are available for working with locks and locked doors:

NAME	DESCRIPTION
KeyRing	The keyring stores multiple key IDs.
LockedDoorInteractiveObject	A version of the door component that lets you specify a lock ID.
LockedDoorTrigger	A trigger zone that is used to open an attached door, but only if the character entering has a key with the correct ID.
LockedTriggerZone	A trigger zone that will fire events if a character with the correct key enters.
KeypadInteractiveObject	An interactive object that is attached to a locked door, and displays a UI based KeypadPopup when interacted with. Entering the correct code will unlock the door.
KeypadPopup	The UI based keypad popup that is shown by the KeypadInteractiveObject .
PickableLockedDoorInteractiveObject	A locked door that displays a lockpicking mini-game when interacted with if the character does not have the correct key to unlock it.
LockpickPopup3D	One implementation of the lockpicking mini-game that is used to unlock a locked door.
LockpickPopupUI	A UI overlay for the LockpickPopup3D mini-game that displays remaining pick count and lock difficulty.

See Also

[Interactive Objects](#)

[Moving Platforms](#)

Carry System

Overview

The carry systems in NeoFPS allow you to pick up and carry rigidbody physics objects in the scene.



There are 2 versions of the carry system: the [Rigidbody Carry System](#) and the [Standard Carry System](#). The main difference is that the Rigidbody carry system allows you to pick up any rigidbody below the mass limit, whereas the standard carry system only allows you to pick up rigidbodies with a [Carryable](#) behaviour attached. This carryable behaviour lets you specify some properties per object, such as whether it should be reoriented when picked up, and whether you can manipulate the object while holding it. It also exposes [unity events](#) that fire when the object is picked up or dropped, which you can tie into to make the object you carry react.

Encumbrance

You can add the [CarrySystemEncumbranceHandler](#) behaviour alongside your carry system of choice to make the object you're carrying affect the character's movement. This works by overriding the [data](#) that the character's [motion graph](#) uses to dictate speed or acceleration with a multiplier based on the carried object's mass. As the object approaches the mass limit for the carry system, the multiplier will shrink down to the minimum you set, and your character will move slower. Note that you need to make sure that the movement states are referencing the motion data that you are modifying.

Optionally, you can also set a [switch parameter](#) on the motion graph when the carried object's mass is over a certain threshold. You can then use that switch in your motion graph [conditions](#) to prevent the character from entering certain motion states when carrying a heavy object (for example, jumping).

See Also

[Rigidbody Carry System](#)

[Standard Carry System](#)

Interaction Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

I Can't Interact With An Object

The interactive object system raycasts from the camera and detects any objects with an `InteractiveObject` component. If you can't interact with an object that has the component attached then that suggests that the physics for the object is wrong. Check the following:

- The interactive object has one or more colliders attached
- Each collider has the **Is Trigger** property set to true
- The object is on the **InteractiveObjects** layer

Trigger Zone Isn't Working

Similar to the interactive objects, the trigger zones require a specific setup. Check that your trigger object:

- Has one or more colliders attached
- Each collider has the **Is Trigger** property set to true
- The object is on the **TriggerZones** layer

Door Opens The Wrong Way

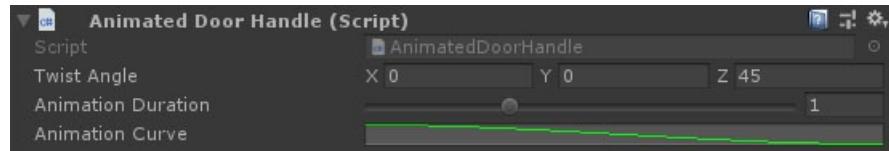
The `KinematicHingeDoor` (or a door component based on this) will detect which side the character that interacts with it is on and open away from that character. To flip the opening direction you can either rotate the transform that is used for the **Direction Transform** property by 180 degrees, or you can switch **Positive Rotation Transform** and **Negative Rotation Transform** properties.

AnimatedDoorHandle MonoBehaviour

Overview

The AnimatedDoorHandle behaviour is attached to a handle object and makes it twist and release when triggered.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Twist Angle	Vector3	The desired euler angles when twisting the handle.
Twist Duration	Float	How long does the twist and release last.
Twist Curve	AnimationCurve	The animation curve for the twist.
Jiggle Angle	Vector3	The maximum euler angles when shaking the handle (if the door is locked).
Jiggle Duration	Float	How long does the locked door jiggle last.
Jiggle Curve	AnimationCurve	The animation curve for the locked door handle jiggle.

See Also

[Doors](#)

[Unity AnimationCurve](#)

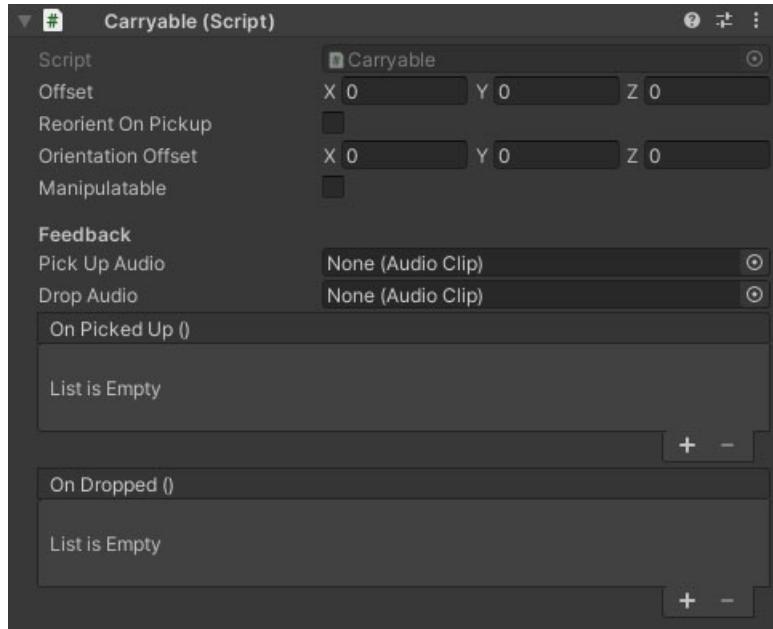
Carryable MonoBehaviour

Overview

The Carryable behaviour is used to represent a physics object that can be picked up by a character with the [StandardCarrySystem](#). It allows you to specify how the object can be manipulated and to expose [unity events](#) that you can tie into to react to the object being picked up or dropped. An example would be using the **ONPICKEDUP** event to fold up an automated turret when you pick it up, and the **ONDROPPED** event to re-deploy the turret when you place it.

You do not need to use the Carryable behaviour if your character is using the [RigidbodyCarrySystem](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Offset	Vector	An offset applied between the transform position of this object and the character's carry anchor point. Use this to prevent larger objects filling the screen.
Reorient On Pickup	Boolean	Should the object be rotated so it is the correct way up when picked up.
Orientation Offset	Vector	An orientation offset that should be applied to the carryable when first picked up.
Manipulatable	Boolean	Can the object be manually rotated.
Pick Up Audio	Audio Clip	An audio clip played when the object is picked up.
Drop Audio	Audio Clip	An audio clip played when the object is dropped.
On Picked Up	UnityEvent	An event that is triggered when the object is picked up. Example uses are to disable an automated turret, or fold up a machine.
On Dropped	UnityEvent	An event that is triggered when the object is dropped. An example use would be to deploy an automated turret when dropped on stable ground.

See Also

[Carry System](#)

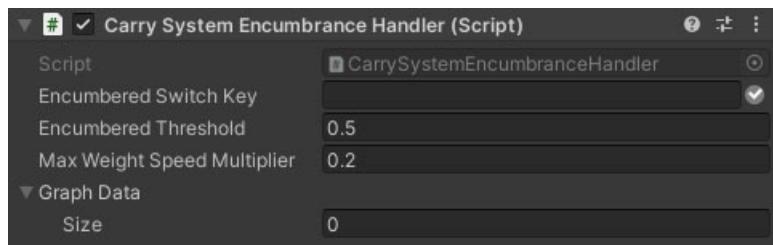
[Unity Events](#)

CarrySystemEncumbranceHandler MonoBehaviour

Overview

The CarrySystemEncumbranceHandler behaviour is used alongside one of the [carry system](#) behaviours to override data on the character's [motion graph](#). This allows you to slow the character down or prevent them performing certain movements while jumping when they are carrying a heavy object.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Encumbered SwitchKey	String	The optional motion graph parameter to set when encumbered.
Encumbered Threshold	Float	The normalised mass (mass / mass limit) of the object before the encumbered switch should be set to true.
Max Weight Speed Multiplier	Float	The multiplier applied to movement speed when carrying something with a mass at the character's limit. For smaller objects, the multiplier will lerp to 1 as their mass approaches zero.
Graph Data	String Array	The motion data parameters to apply the multiplier to.

See Also

[Carry System](#)

[Motion Graph Parameters And Data](#)

CharacterInteractionHandler MonoBehaviour

Overview

The CharacterInteractionHandler behaviour is attached to a character and handles interaction with [interactive objects](#) in the scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Distance	Float	The maximum distance from the camera to trigger interactions.
Tick Rate	Int	How frequently does the handler cast forward to check for an object. Smaller numbers mean more responsive but more wasted calculations.
Layers	LayerMask	The layers that will be checked against when casting for valid interaction targets.
Error Audio	FpsCharacterAudio	The character audio clip to play for an invalid interaction.

See Also

[Interactive Objects](#)

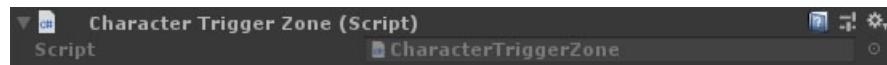
[FpsCharacterAudioHandler](#)

CharacterTriggerZone MonoBehaviour

Overview

The CharacterTriggerZone behaviour fires script events when a character enters and exits the collider.

Inspector



Properties

The CharacterTriggerZone behaviour has no properties exposed in the inspector.

See Also

[Unity BoxCollider](#)

[Unity SphereCollider](#)

[Unity CapsuleCollider](#)

CharacterTriggerZonePersistant MonoBehaviour

Overview

The CharacterTriggerZonePersistant behaviour fires script events when characters enter and exit.

Inspector



Properties

The CharacterTriggerZonePersistant behaviour has no properties exposed in the inspector.

See Also

[Layers And Tags](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

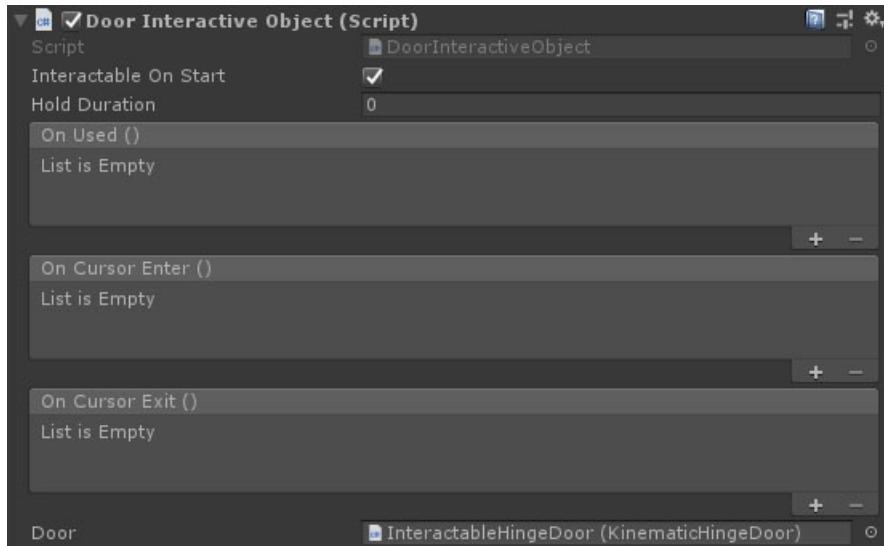
[Unity CapsuleCollider](#)

DoorInteractiveObject MonoBehaviour

Overview

The DoorInteractiveObject behaviour is an extension of the [InteractiveObject](#) used to trigger door open and close.

Inspector



Properties

The DoorInteractiveObject inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from DoorBase).

See Also

[InteractiveObject](#)

DoorTrigger MonoBehaviour

Overview

The DoorTrigger behaviour is attached to an object with a trigger collider. It will open the specified door when a character enters, and close it when they leave.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from <code>DoorBase</code>).
Characters Only	Boolean	Should the door only open for characters, not any collider.

See Also

[Doors](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

[Unity CapsuleCollider](#)

ElevatorController MonoBehaviour

Overview

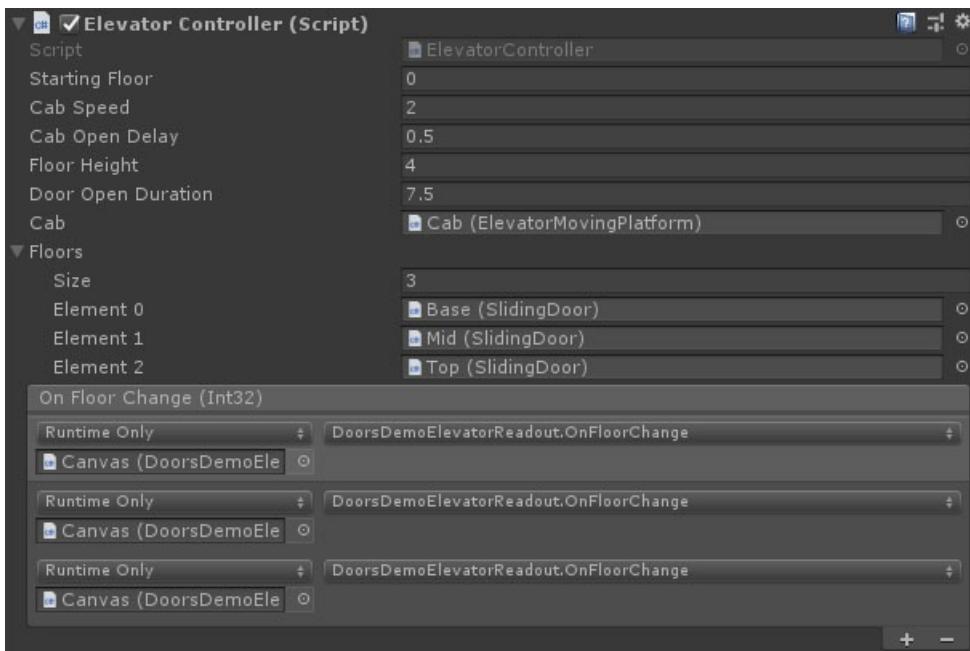
The ElevatorController behaviour manages a single elevator.

An elevator also requires an [ElevatorMovingPlatform](#) to function. This acts as the cab or platform and actually moves between floors.

Each floor has a door that will be opened when the elevator cab arrives at that floor. If that door is a [SlidingDoor](#) behaviour, then the cab's door geometry can be added to every floor as a door section. This means that the cab's inner doors will open in sync with the relevant floor's outer doors.

The ElevatorController exposes the `PressFloorButton(int floorIndex)` function. This can be attached to [InteractiveObject](#) events for the exterior call buttons and the cab interior floor buttons. If the current floor button is pressed the doors will open for a set time. If another floor button is pressed the doors will close, the cab will move to the correct floor, and then the doors will open for a set time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Starting Floor	Int	The floor the elevator cab starts on. It is best to move the cab to this position in the editor to prevent it jumping there instantly which can cause problems if there are dynamic objects in the cab.
Cab Speed	Float	The movement speed of the cab.
Cab Open Delay	Float	The delay between reaching a floor and opening the doors.
Floor Height	Float	The distance between floors.

NAME	TYPE	DESCRIPTION
Door Open Duration	Float	The duration the elevator doors will remain open unless interrupted.
Cab	ElevatorMovingPlatform	The moving platform of the elevator cab.
Floors	Door Array	The doors for each floor (will accept any door that inherits from DoorBase).
On Floor Change	UnityEvent	An event that is invoked every time the cab switches floors. Useful for any elevator readout or chime.

See Also

[Doors](#)

[ElevatorMovingPlatform](#)

[SlidingDoor](#)

[InteractiveObject](#)

[Unity Events](#)

ElevatorMovingPlatform MonoBehaviour

Overview

The ElevatorMovingPlatform behaviour is a [moving platform](#) which works with the [ElevatorController](#). It is attached to the kinematic rigidbody that acts as the elevator cab or platform, and moves the character smoothly with it.

Inspector



Properties

The ElevatorMovingPlatform behaviour has no properties exposed in the inspector.

See Also

[ElevatorController](#)

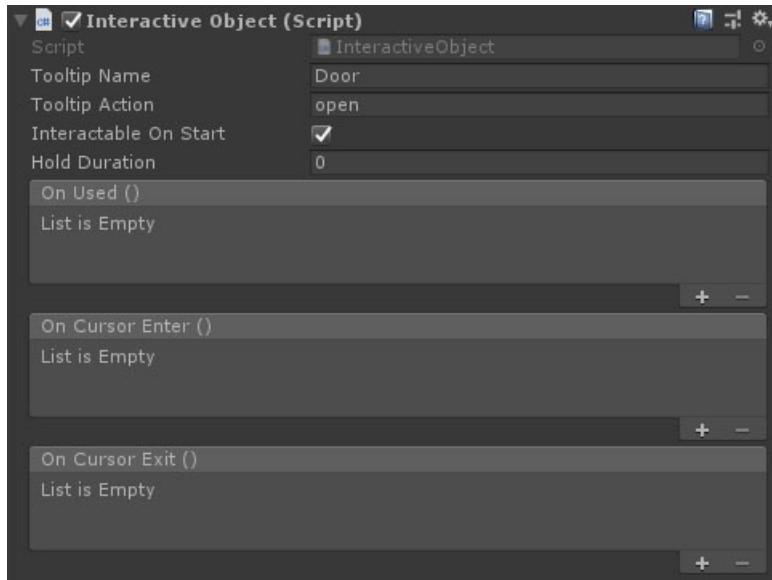
[Moving Platforms](#)

InteractiveObject MonoBehaviour

Overview

The InteractiveObject behaviour is used to represent any object in the world that the player can interact with via the **use** button.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

See Also

[Interactive Objects](#)

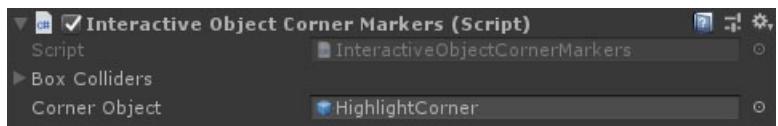
[Unity Events](#)

InteractiveObjectCornerMarkerrs MonoBehaviour

Overview

The InteractiveObjectCornerMarkerrs behaviour is used to highlight [interactive objects](#) when the player looks at them. It spawns markers on the bounding box corners of the object. These are then displayed when the object is highlighted and hidden when it is not.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Box Colliders	BoxCollider Array	The box colliders of the InteractiveObject to attach markers to.
Corner Object	GameObject	The prefab to use for the corner objects. 8 instances of this will be instantiated and placed at the corners of the box.

See Also

[InteractiveObject](#)

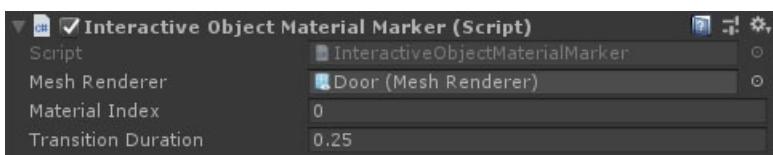
[Unity BoxCollider](#)

InteractiveObjectMaterialMarker MonoBehaviour

Overview

The InteractiveObjectMaterialMarker behaviour is used to highlight [interactive objects](#) when the player looks at them. It fades in a glowing pulse effect on the object as long as it has a material that uses the *NeoFPS/Standard/InteractiveHighlightMetallic* or *NeoFPS/Standard/InteractiveHighlightSpecular* shaders.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Mesh Renderer	MeshRenderer	The mesh renderer of the object (used to set the relevant material property blocks).
Material Index	Integer	The index of the highlight material on the mesh renderer.
Transition Duration	Float	A fade time between he highlighted state and the non-highlighted state.

See Also

[InteractiveObject](#)

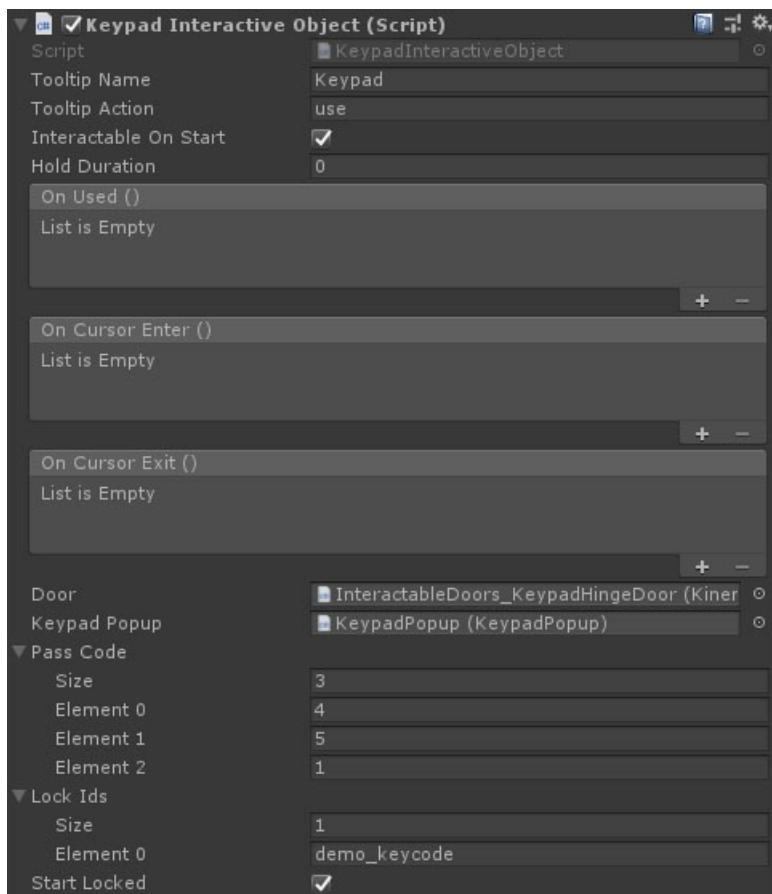
[Unity BoxCollider][unity-boxcollider]

KeypadInteractiveObject MonoBehaviour

Overview

The KeypadInteractiveObject behaviour is an extension of the [InteractiveObject](#) that can be used alongside locked doors. On using the object it displays a [keypad popup](#), and if the player enters the correct key code then the door will be unlocked.

Inspector



Properties

The DoorInteractiveObject inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from DoorBase).
m_KeypadPopup	KeypadPopup	The keypad UI popup to show.
m_PassCode	Integer Array	The key code to unlock the door.
Lock Ids	String Array	A unique ID for this lock. IF the player has an equivalent key in their inventory key ring then the digits will be shown with the popup.
m_StartLocked	Boolean	Should the door be locked on start.

See Also

[InteractiveObject](#)

[KeypadPopup Behaviour](#)

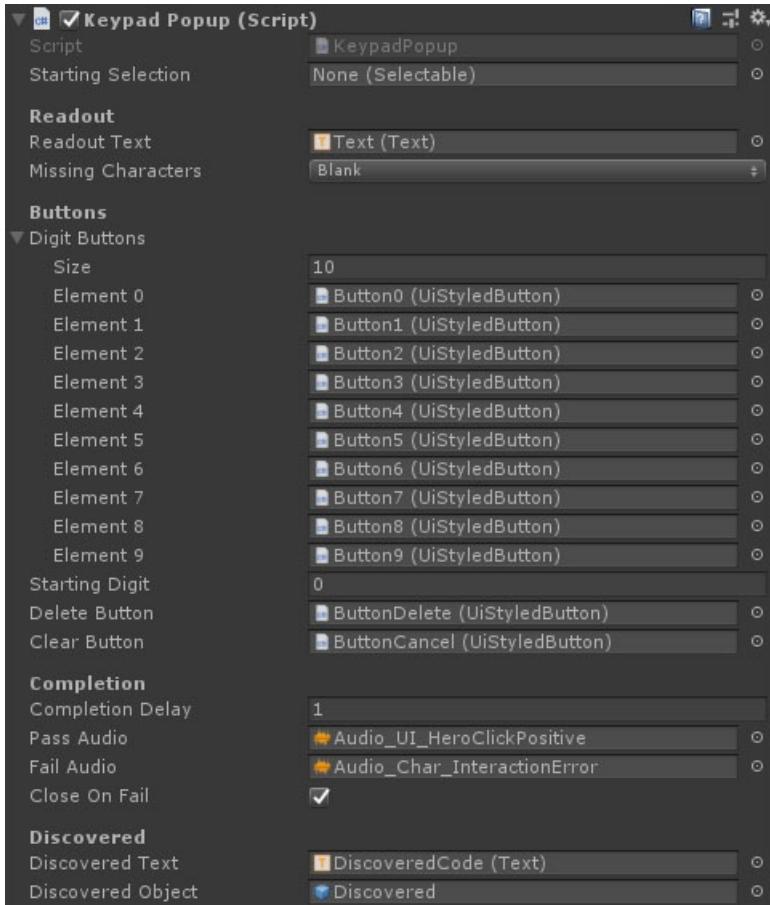
[KeyRing Behaviour](#)

KeypadPopup MonoBehaviour

Overview

The KeypadPopup behaviour is a UI popup which can be displayed for locked doors and items. Events are fired for correct and incorrect passcodes. If the passcode is known, then it will be displayed alongside the keypad.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Starting Selection	Selectable	The initial UI element to select when the popup is shown.
Readout Text	Text	Output text when typing code.
Missing Characters	Dropdown	The character to use for missing digits in the code.
Digit Buttons	Button Array	The number buttons of the keypad in numeric order (eg 0,1,2,3,4...).
Starting Digit	Integer	The lowest numbered button.
Delete Button	Button	The delete button is used to delete the last input digit.
Clear Button	Button	The clear button clears all typed digits.
Completion Delay	Float	The time after the last digit is input before the popup closes.

NAME	TYPE	DESCRIPTION
Pass Audio	AudioClip	The audio clip to play if the correct code is input.
Fail Audio	AudioClip	The audio clip to play if an incorrect code is input.
Close On Fail	Boolean	Should the popup close after the wrong code is input or allow the player to enter another.
Discovered Text	Text	If the keycode is known it will be shown here.
Discovered Object	GameObject	If the keycode is known then this object will be activated.

See Also

[Doors](#)

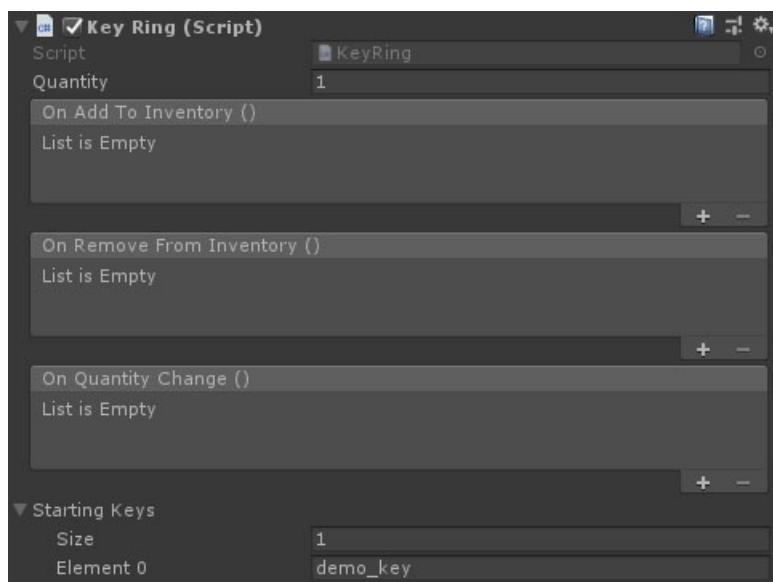
[Sample UI](#)

KeyRing MonoBehaviour

Overview

The KeyRing behaviour is an [inventory](#) item which stores keys and keycodes. Picking up a keyring will add it to the character's inventory if they do not already have one, or merge the contents with the existing one if they do.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Starting Keys	String Array	A list of key IDs this keyring starts with.

See Also

[Doors](#)

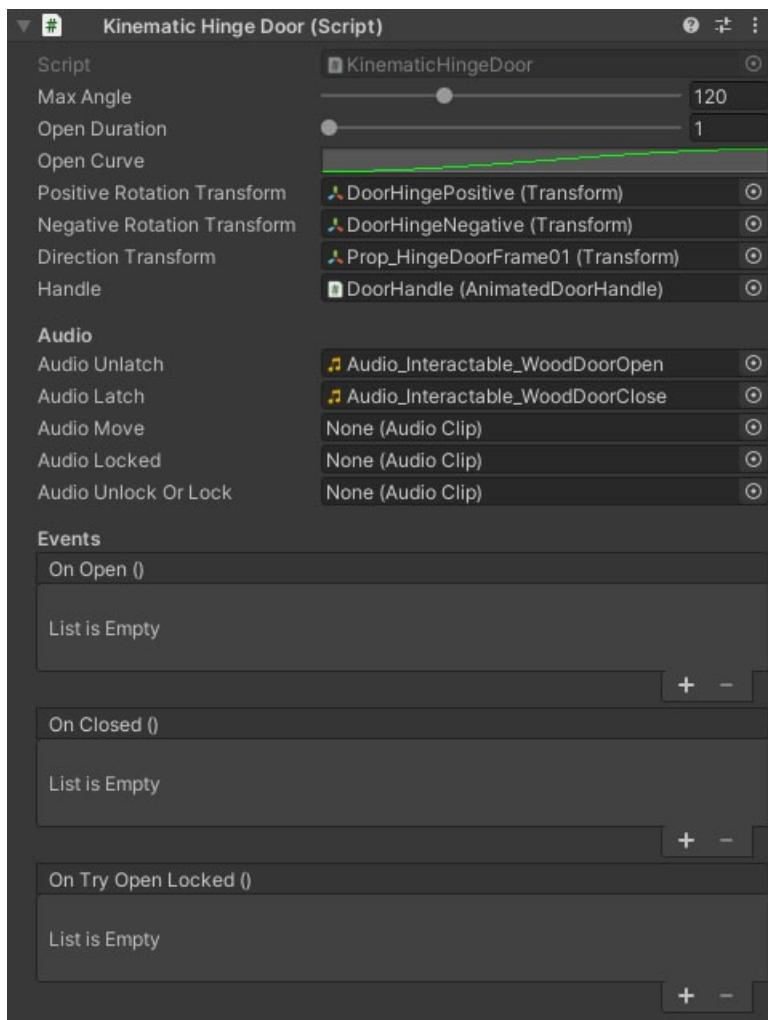
[Inventory](#)

KinematicHingeDoor MonoBehaviour

Overview

The KinematicHingeDoor behaviour handles opening, closing and animating a basic hinge door. The door rotates around 2 points depending on which direction it opens to prevent the door object overlapping with its frame.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Angle	Float	The maximum open angle of the door.
Open Duration	Float	The time it takes to go from fully closed to fully open and vice versa.
Open Curve	AnimationCurve	The interpolation curve for opening and closing the door.
Positive Rotation Transform	Transform	The transform point for the positive rotation of the door (defaults to the transform for the object this behaviour is attached to).
Negative Rotation Transform	Transform	The transform point for the negative rotation of the door. This allows hinges at both edges of a door to prevent overlap. If this is null the door will only open on one side.

NAME	TYPE	DESCRIPTION
Direction Transform	Transform	A fixed transform used to check which side of the doorway a character is on (defaults to the transform for the object this behaviour is attached to).
Handle	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.
Audio Unlatch	AudioClip	The audio to play when the door unlatches and starts opening.
Audio Latch	AudioClip	The audio to play when the door latches closed.
Audio Move	AudioClip	The audio to play when the starts moving.
Audio Locked	AudioClip	The audio to play when attempting to open the door while locked.
Audio Unlock Or Lock	AudioClip	The audio to play when unlocking or locking the door.
On Open	UnityEvent	An event fired when the door unlatches and starts opening.
On Closed	UnityEvent	An event fired when the door is finished closing and latches.
On Try Open Locked	UnityEvent	An event fired when attempting to open the door while locked.

See Also

[Doors](#)

[AnimatedDoorHandle](#)

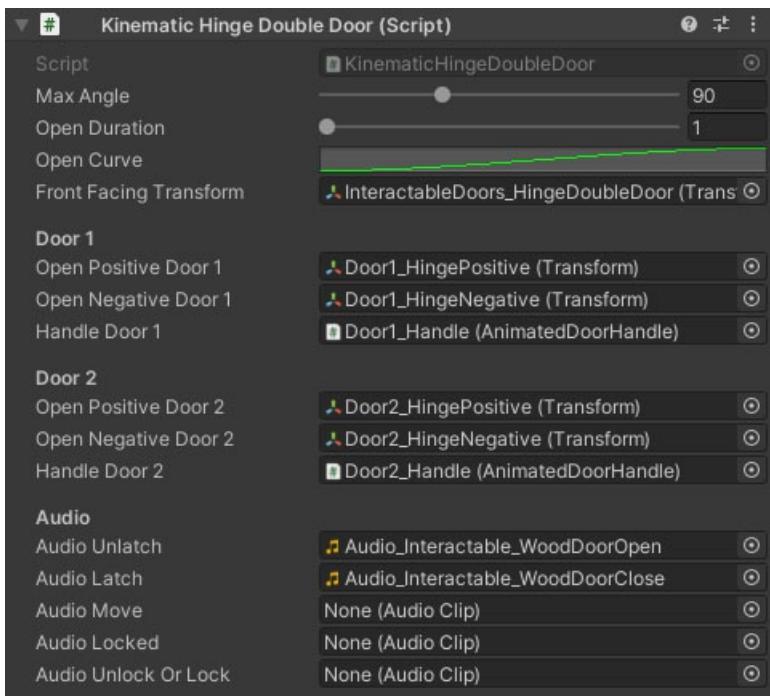
[Unity AnimationCurve](#)

KinematicHingeDoubleDoor MonoBehaviour

Overview

The KinematicHingeDoubleDoor behaviour handles opening, closing and animating a basic hinged double door. Each door rotates around 2 points depending on which direction it opens to prevent the door object overlapping with its frame. Both doors will open and close together.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Angle	Float	The maximum open angle of the door.
Open Duration	Float	The time it takes to go from fully closed to fully open and vice versa.
Open Curve	AnimationCurve	The interpolation curve for opening and closing the door.
Front Facing Transform	Transform	A fixed transform used to check which side of the doorway a character is on (defaults to the transform for the object this behaviour is attached to).
Open Positive Door 1	Transform	The hinge transform for the positive rotation of the door. This allows hinges at both edges of a door to prevent overlap.
Open Negative Door 1	Transform	The transform point for the negative rotation of the door. This allows hinges at both edges of a door to prevent overlap. If this is null the door will only open on one side.
Handle Door 1	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.

NAME	TYPE	DESCRIPTION
Open Positive Door 2	Transform	The hinge transform for the positive rotation of the door. This allows hinges at both edges of a door to prevent overlap.
Open Negative Door 2	Transform	The transform point for the negative rotation of the door. This allows hinges at both edges of a door to prevent overlap. If this is null the door will only open on one side.
Handle Door 2	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.
Audio Unlatch	AudioClip	The audio to play when the door unlatches and starts opening.
Audio Latch	AudioClip	The audio to play when the door latches closed.
Audio Move	AudioClip	The audio to play when the starts moving.
Audio Locked	AudioClip	The audio to play when attempting to open the door while locked.
Audio Unlock Or Lock	AudioClip	The audio to play when unlocking or locking the door.
On Open	UnityEvent	An event fired when the door unlatches and starts opening.
On Closed	UnityEvent	An event fired when the door is finished closing and latches.
On Try Open Locked	UnityEvent	An event fired when attempting to open the door while locked.

See Also

[Doors](#)

[AnimatedDoorHandle](#)

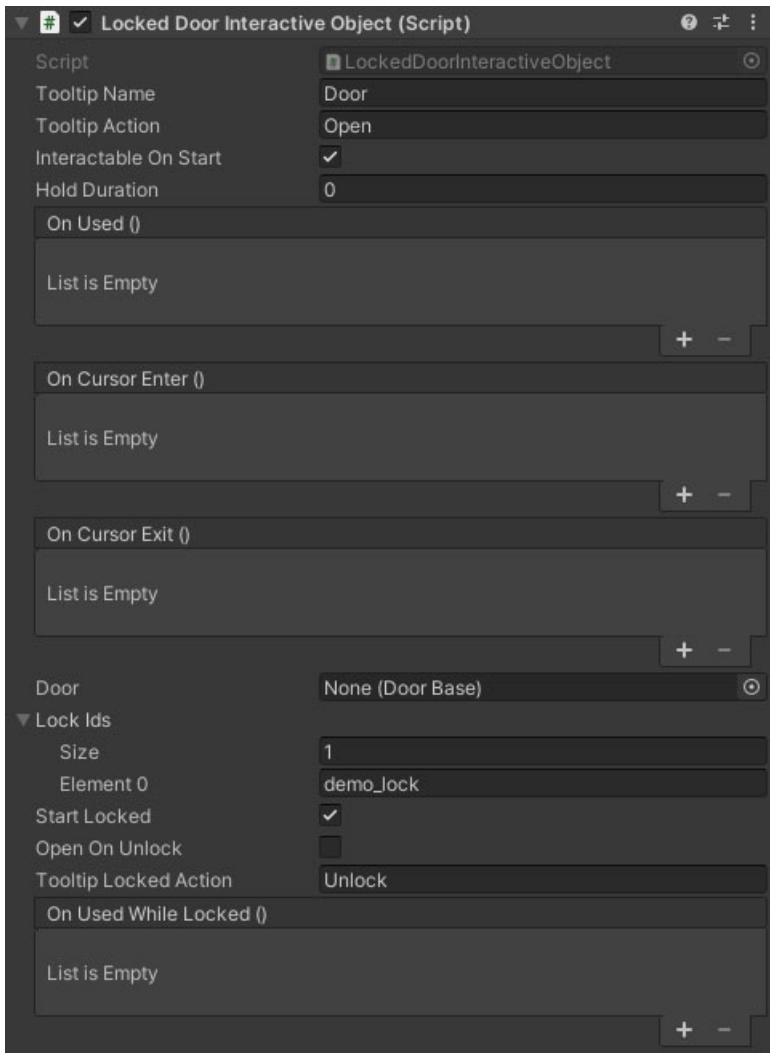
[Unity AnimationCurve](#)

LockedDoorInteractiveObject MonoBehaviour

Overview

The LockedDoorInteractiveObject behaviour is an extension of the [InteractiveObject](#) used to trigger door open and close and managed the lock state of a door. If the door is locked then the character must have a [keyring](#) inventory item with the correct keycode to unlock the door before it can be opened.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.

NAME	TYPE	DESCRIPTION
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.
Door	Door	The door to open (will accept any door that inherits from DoorBase).
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
Start Locked	Boolean	Should the door be locked on start.
Open On Unlock	Boolean	Should the door be opened when it's unlocked.
Tooltip Locked Action	String	The tooltip action to use when the door is locked. Use the open action tooltip for the other tooltip action.
On Used While Locked	UnityEvent	An event that is triggered when the object is used while locked (does not fire when unlocking).

See Also

[InteractiveObject](#)

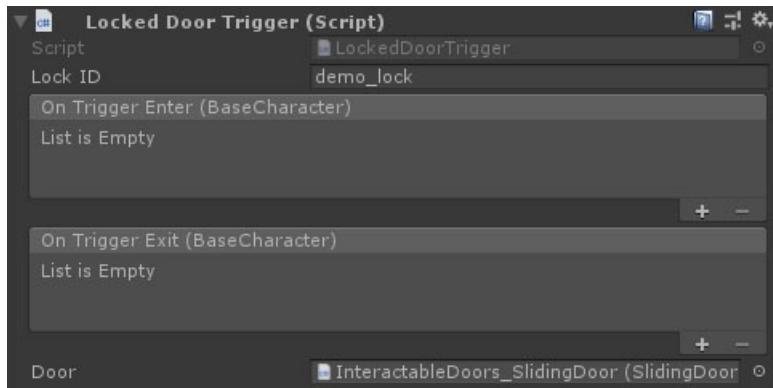
[KeyRing Behaviour](#)

LockedDoorTrigger MonoBehaviour

Overview

The LockedDoorTrigger behaviour is attached to an object with a trigger collider. It will open the specified door when a character with the correct key code in their [inventory key ring](#) enters, and close it when they leave.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory [key ring][3] to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.
Door	Door	The door to open (will accept any door that inherits from <code>DoorBase</code>).

See Also

[Doors](#)

[KeyRing Behaviour](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

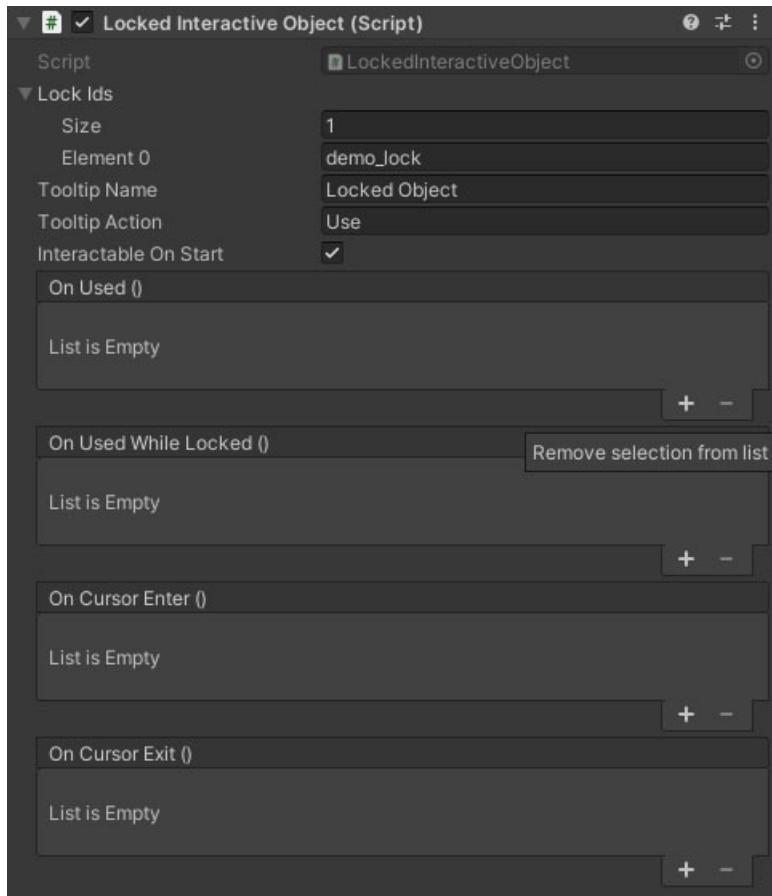
[Unity CapsuleCollider](#)

LockedInteractiveObject MonoBehaviour

Overview

The LockedInteractiveObject behaviour is a version of an [interactive object](#) that the player character can only interact with when unlocked, or when the character has a [keyring](#) inventory item with the correct keycode in their inventory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
Interactable On Start	Boolean	Can the object be interacted with immediately.
On Used	UnityEvent	An event that is triggered when the object is used and unlocked.
On Used While Locked	UnityEvent	An event that is triggered when the character attempts to use the object but it's locked.

NAME	TYPE	DESCRIPTION
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

See Also

[Interactive Objects](#)

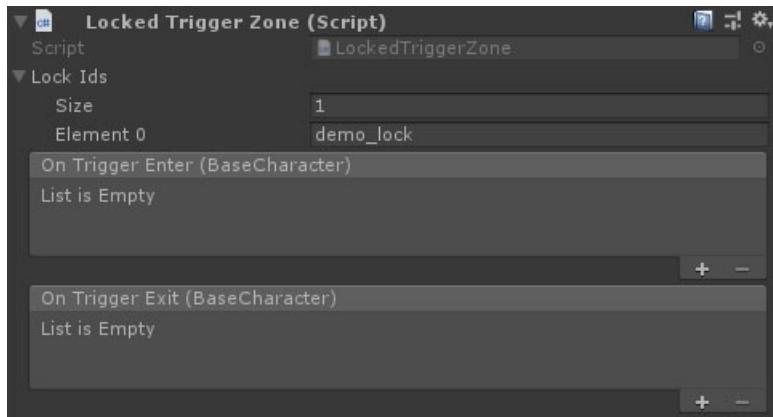
[Unity Events](#)

LockedTriggerZone MonoBehaviour

Overview

The LockedTriggerZone behaviour is a trigger zone that only activates if the character has a specific key code in their [inventory key ring](#).

Inspector



Properties

Name	Type	Description
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to activate the trigger zone. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.

See Also

[KeyRing Behaviour](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

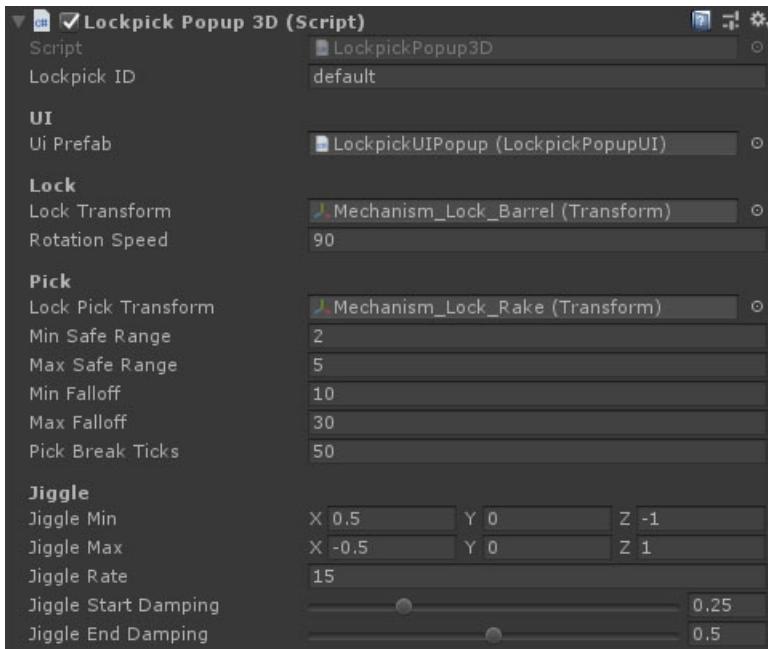
[Unity CapsuleCollider](#)

LockPickPopup3D MonoBehaviour

Overview

The LockPickPopup3D behaviour is a specific lockpicking mini-game in the style of fallout or dying light. Your scene can contain multiple lock-picking minigames, all differentiated by their IDs, as long as their scripts inherit from the `LockPickPopup` base class.

Inspector



Properties

NAME	TYPE	DESCRIPTION
UI Prefab	LockpickPopupUI	The UI popup prefab to be drawn over the top of this minigame using the prefab popup container system.
Lock Transform	Transform	The transform of the lock barrel.
Rotation Speed	Float	The rotation speed of the lock barrel when tensioned (in degrees per second).
Lock Pick Transform	Transform	The transform of the lock pick object. Its pivot should be lined up with the hole of the lock.
Min Safe Range	Float	The smallest size the safe range for the pick can be (at highest difficulty).
Max Safe Range	Float	The maximum size the safe range for the pick can be (at lowest difficulty).
Min Falloff	Float	The smallest falloff outside the safe range, where the lock can rotate but will still snag (at highest difficulty).
Max Falloff	Float	The largest falloff outside the safe range, where the lock can rotate but will still snag (at lowest difficulty).
Pick Break Ticks	Integer	The number of fixed update ticks where the pick is catching before it will break.
Jiggle Min	Vector3	The minimum jiggle angle when the pick catches. It will bounce between this and max.

NAME	TYPE	DESCRIPTION
Jiggle Max	Vector3	The maximum jiggle angle when the pick catches. It will bounce between this and min.
Jiggle Rate	Float	The number of shakes per second when catching.
Jiggle Start Damping	Float	The amount of time it takes for the jiggle to fade in when the lock catches.
Jiggle End Damping	Float	The amount of time it takes for the jiggle to fade out once tension is released.

See Also

[Doors](#)

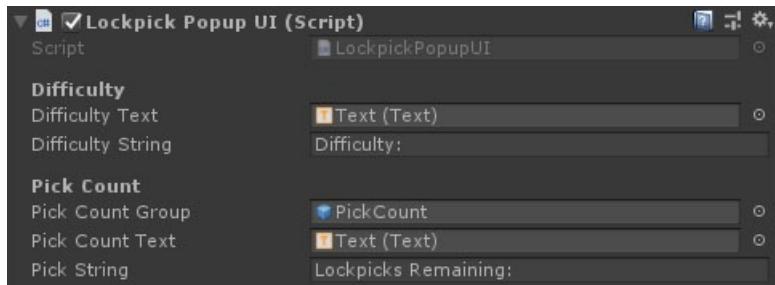
[LockpickPopupUI Behaviour](#)

LockPickPopupUI MonoBehaviour

Overview

The LockPickPopupUI behaviour is attached to an object with a trigger collider. It will open the specified door when a character enters, and close it when they leave.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Difficulty Text	Text	The UI text element that will show the difficulty rating of the lock.
Difficulty String	String	The pick difficulty prefix.
Pick Count Group	GameObject	The parent object of the UI elements that show the pick count. If the lock does not use inventory picks, this object and its children will be hidden.
Pick Count Text	Text	The text readout for the remaining pick count.
Pick String	String	The pick count prefix string.

See Also

[Doors](#)

[Sample UI](#)

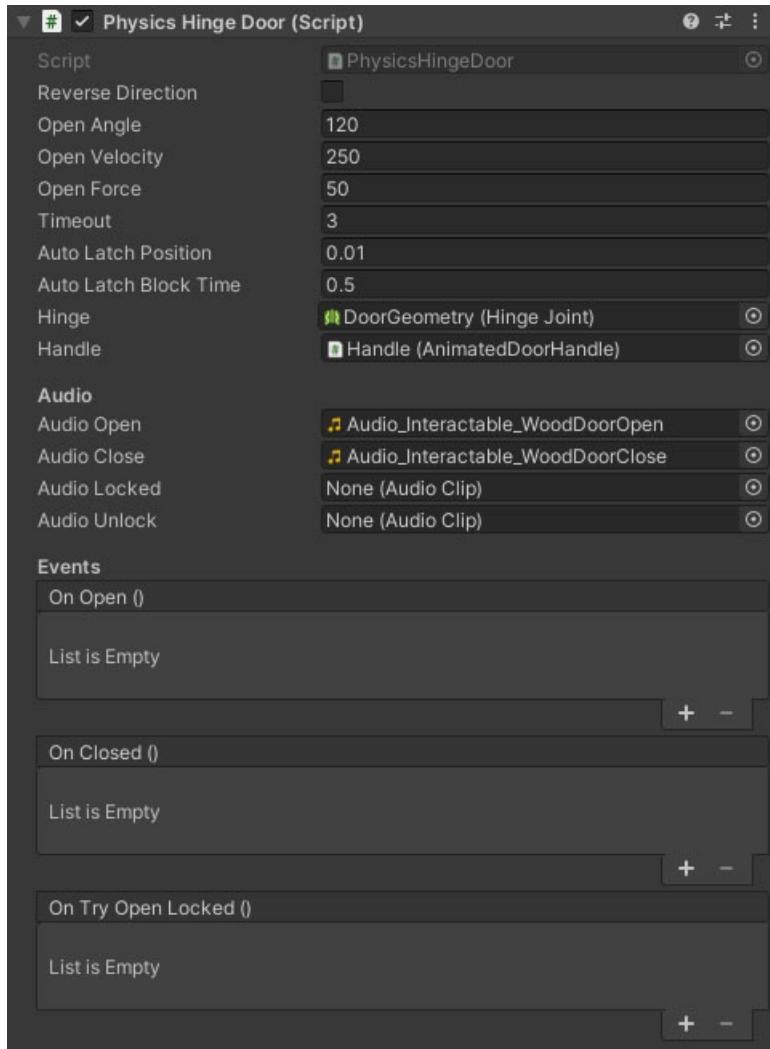
[LockpickPopup3D Behaviour](#)

PhysicsHingeDoor MonoBehaviour

Overview

The PhysicsHingeDoor behaviour handles latching and release of a physical hinge based door.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Reverse Direction	Boolean	Reverses the door opening direction.
Open Angle	Float	The open limit of the door. When opening, force will be applied until the door reaches this angle.
Open Velocity	Float	The target speed to move the door.
Open Force	Float	The force applied to the door when opening or closing.
Timeout	Float	A time limit for force to be applied. If the door is blocked, it would never reach full open or closed and this value prevents it trying forever.

NAME	TYPE	DESCRIPTION
Auto Latch Position	Float	The normalised position (0 to 1 translates to closed to full open angle) at which the door will latch when closing.
Auto Latch Block Time	Float	Prevent latching for a short period when opened.
Hinge	HingeJoint	The hinge joint of the door.
Handle	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.
Audio Open	AudioClip	The audio to play when the door is unlatched and opened.
Audio Close	AudioClip	The audio to play when the door closes and latches.
Audio Locked	AudioClip	The audio to play when attempting to open the door while locked.
Audio Unlock	AudioClip	The audio to play when unlocking or locking the door.
On Open	UnityEvent	An event fired when the door unlatches and starts opening.
On Closed	UnityEvent	An event fired when the door is finished closing and latches.
On Try Open Locked	UnityEvent	An event fired when attempting to open the door while locked.

See Also

[Doors](#)

[AnimatedDoorHandle](#)

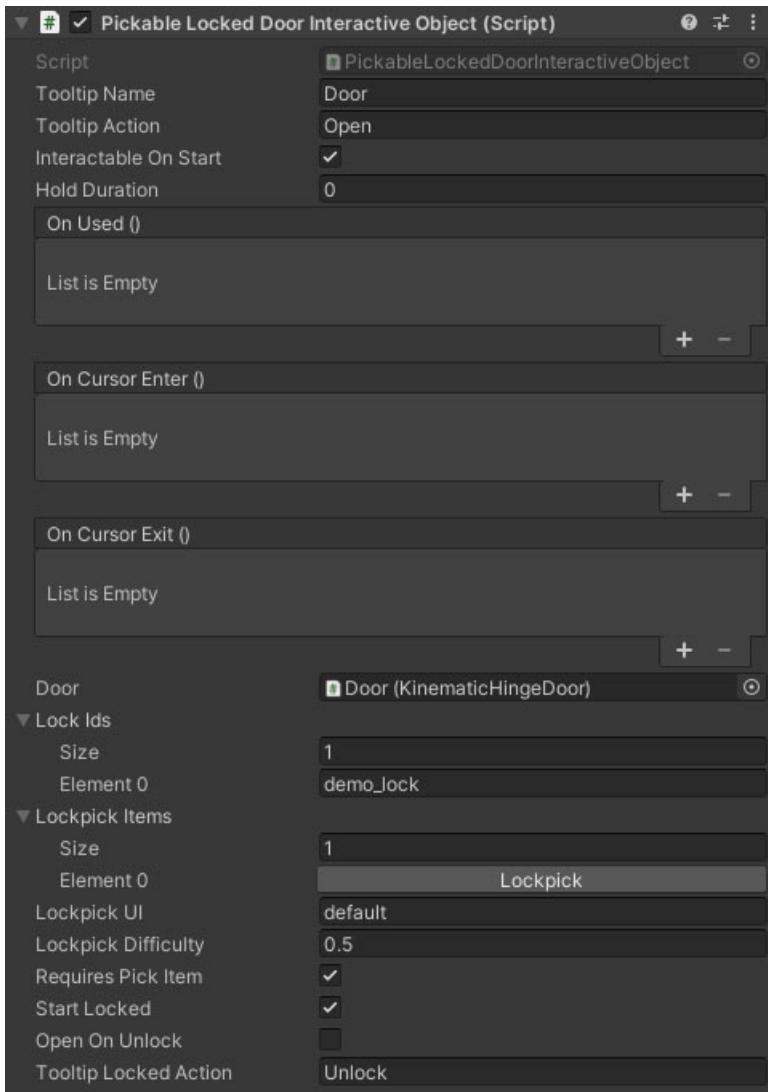
[Unity HingeJoint](#)

PickableLockedDoorInteractiveObject MonoBehaviour

Overview

The PickableLockedDoorInteractiveObject behaviour is an extension of the [InteractiveObject](#) used to trigger door open and close. If the door is locked then there are 2 ways to unlock it. Firstly, if the character has a [keyring](#) inventory item with the correct keycode, then they will unlock the door instantly. Secondly, if they do not have the correct keycode but do have lockpick items in their inventory, then using the door will show a lock-picking minigame popup.

Inspector



Properties

The PickableLockedDoorInteractiveObject inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.

Name	Type	Description
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.
Door	Door	The door to open (will accept any door that inherits from <code>DoorBase</code>).
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
Lockpick Items	Inventory ID Array	A range of inventory IDs for lockpicks in the character inventory. If you have one of these then you can pick the lock.
Lockpick UI	String	The ID for the lockpick UI to use. This allows for multiple lockpick styles in a single scene.
Lockpick Difficulty	Float	The difficulty of this specific lock.
Requires Pick Item	Boolean	Does the character require a lockpick item in their inventory.
Start Locked	Boolean	Should the door be locked on start.
Open On Unlock	Boolean	Should the door be opened when it's unlocked.
Tooltip Locked Action	String	The tooltip action to use when the door is locked. Use the open action tooltip for the other tooltip action.

See Also

[Doors](#)

[InteractiveObject](#)

[KeyRing Behaviour](#)

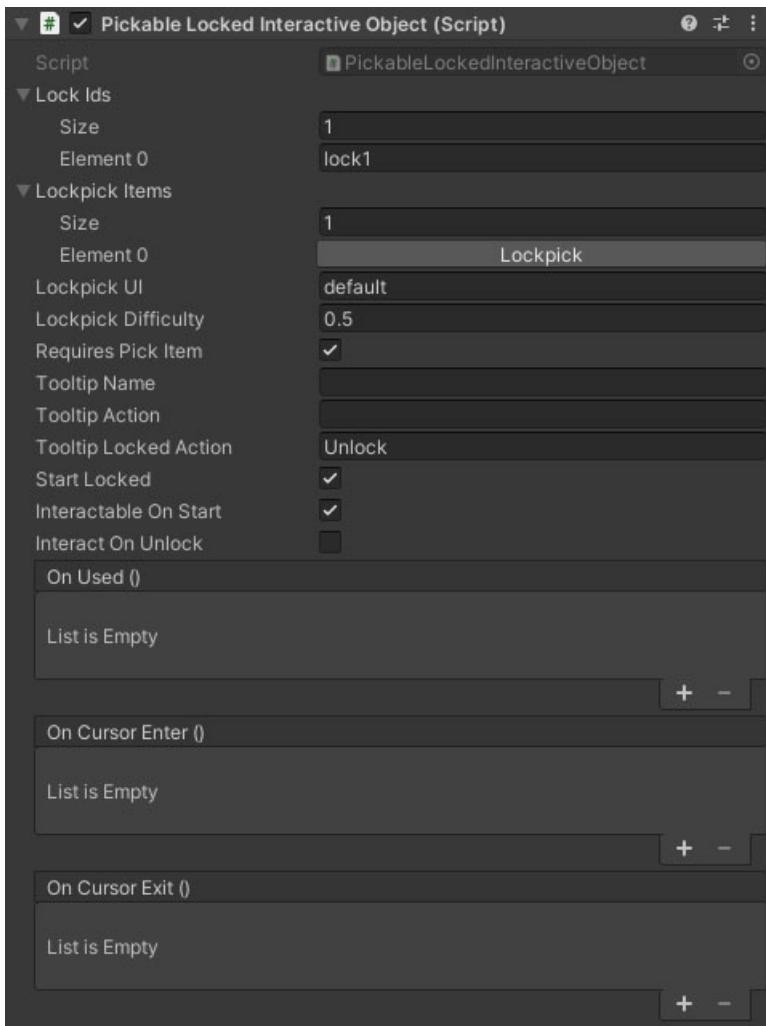
[LockPickPopup3D Behaviour](#)

PickableLockedInteractiveObject MonoBehaviour

Overview

The PickableLockedInteractiveObject behaviour is an extension of the [InteractiveObject](#) that must be unlocked before it will fire the OnUsed event. This can mean picking the lock, using a key, or unlocking via the API. If the inventory object is locked then there are 2 ways to unlock it. Firstly, if the character has a [keyring](#) inventory item with the correct keycode, then they will unlock the interactive object instantly. Secondly, if they do not have the correct keycode but do have lockpick items in their inventory, then using the interactive object will show a lock-picking minigame popup. Once the PickableLockedInteractiveObject has been unlocked it will act like a regular [InteractiveObject](#)

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the lock must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
Lockpick Items	Inventory ID Array	A range of inventory IDs for lockpicks in the character inventory. If you have one of these then you can pick the lock.
Lockpick UI	String	The ID for the lockpick UI to use. This allows for multiple lockpick styles in a single scene.
Lockpick Difficulty	Float	The difficulty of this specific lock.

NAME	TYPE	DESCRIPTION
Requires Pick Item	Boolean	Does the character require a lockpick item in their inventory.
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
TooltipLockedAction	String	The tooltip action (verb) to show when the object is locked.
Start Locked	Boolean	Should the object be locked on start.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Interact On Unlock	Boolean	Should the object be interacted with immediately when it's unlocked.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

See Also

[Doors](#)

[InteractiveObject](#)

[KeyRing Behaviour](#)

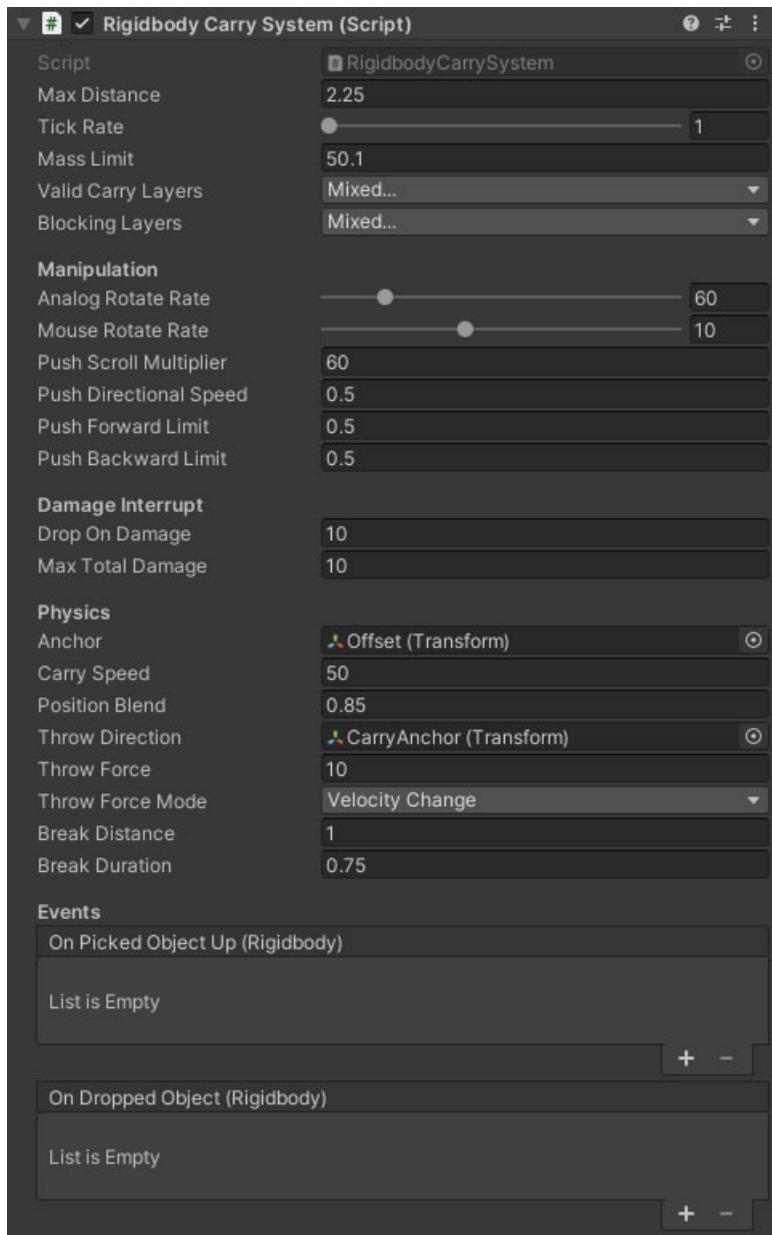
[LockPickPopup3D Behaviour](#)

RigidbodyCarrySystem MonoBehaviour

Overview

The RigidbodyCarrySystem behaviour allows the character it's attached to to pick up any [rigidbody](#) object below a certain mass.

Inspector



Properties

Name	Type	Description			
Max Distance	Float	The maximum cast distance from the camera when checking for a carryable item.			
Tick Rate	Integer	How frequently (in fixed frames) does the carry system cast forward to check for an object. Smaller numbers mean more responsive but more wasted calculations.			
Mass Limit	Float	The maximum mass of the object you can carry.			

NAME	TYPE	DESCRIPTION			
Valid Carry Layers	LayerMask	The valid layers for carryable objects. An object can not be picked up if it is not on one of these layers.			
Blocking Layers	LayerMask	An extra set of layers that can block the raycasts to detect carryable objects. This is used to prevent picking up objects through walls or doors.			
Analog Rotate Rate	Float	The number of degrees per second to turn the anchor at full analog turn. Be warned that setting this too high might turn the anchor faster than the object can turn, so causing it to bounce back and forth as the closest direction flips.			
Mouse Rotate Rate	Float	A multiplier applied to mouse movement when turning the anchor. Be warned that setting this too high might turn the anchor faster than the object can turn, so causing it to bounce back and forth as the closest direction flips.			
Push Scroll Multiplier	Float	A multiplier applied to mouse scroll to determine movement speed when pushing the carried object forwards/backwards.			
Push Directional Speed	Float	The movement speed when pushing the carried object forwards/backwards.			
Push Forward Limit	Float	The maximum forward distance the carried object can be pushed from its default starting position.			
Push Backward Limit	Float	The maximum backwards distance the carried object can be pushed from its default starting position.	Drop On Damage	Float	If the character receives damage higher than this value in one go, then they will drop the object.
Max Total Damage	Float	If the character receives damage totalling this value since picking up the object, they will drop it.			
Anchor	Transform	The anchor transform that carried objects will snap to. This should be a child of an object with the default carry position so that it can move and rotate relative.			
Carry Speed	Float	The max movement speed for the carried object to match the anchor position.			
Position Blend	Float	An event fired when the character picks an object up.			
Throw Direction	Transform	A transform representing the throw direction (forwards / Z-axis). This would usually be the parent of the anchor transform.			

NAME	TYPE	DESCRIPTION			
Throw Force	Float	The force to apply in the throw direction to the carried object when throwing it.			
Throw ForceMode	Dropdown	How the throw force should be applied. This uses the standard [Unity force mode][unity-forcemode].			
Break Distance	Float	A distance from the anchor where the break timer will start. Once the break timer hits the break duration, the object will be dropped. If the object moves back within the break distance then the timer resets.			
Break Duration	Float	If the carried object is further from the anchor than the break distance for this amount of time then it will be dropped.			
On Picked Object Up	UnityEvent	An event fired when the character picks an object up. Pointing this at a method that takes a Rigidbody parameter will pass the object that was picked up when the event is fired.			
On Dropped Object	UnityEvent	An event fired when the character drops an object. Pointing this at a method that takes a Rigidbody parameter will pass the object that was dropped when the event is fired.			

See Also

[Carry System](#)

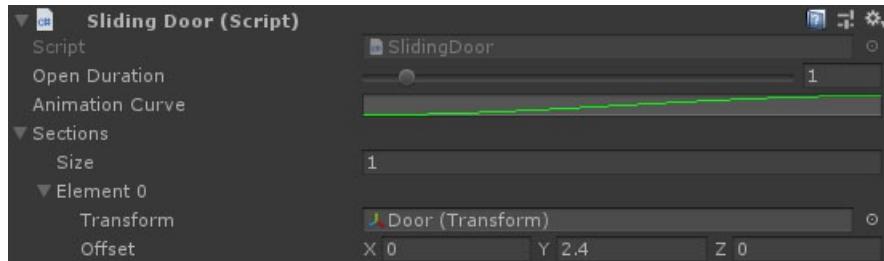
[Unity Events](#)

SlidingDoor MonoBehaviour

Overview

The SlidingDoor behaviour is used to control and animate moving door sections.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Open Duration	Float	The time it takes to go from fully closed to fully open and vice versa.
Animation Curve	AnimationCurve	The interpolation curve for animating the door sections.
Sections	Door Section Array	One or more door sections. All will move when the door opens and closes.
Audio Open	[AudioClip][unity-audioclip]	The audio to play when the door is unlatched and opened.
Audio Close	[AudioClip][unity-audioclip]	The audio to play when the door closes and latches.
Audio Locked	[AudioClip][unity-audioclip]	The audio to play when attempting to open the door while locked.
Audio Unlock	[AudioClip][unity-audioclip]	The audio to play when unlocking or locking the door.

Door Section

NAME	TYPE	DESCRIPTION
Transform	Transform	The door section transform.
Offset	Vector3	The offset from starting (closed) position when opened.

See Also

[Doors](#)

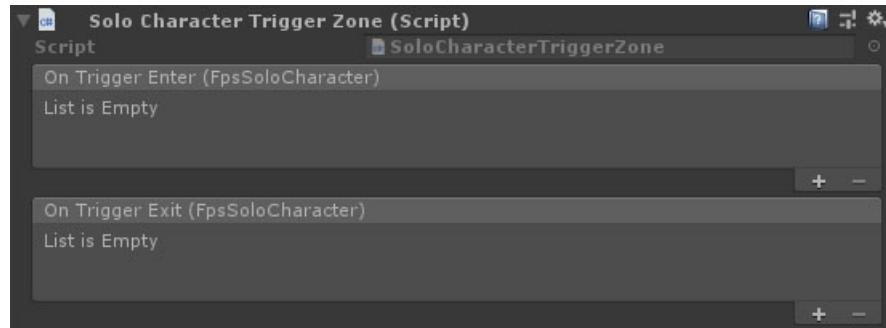
[Unity AnimationCurve](#)

SoloCharacterTriggerZone MonoBehaviour

Overview

The SoloCharacterTriggerZone behaviour fires script events when a character enters and exits the collider.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.

See Also

[Layers And Tags](#)

[Unity Events](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

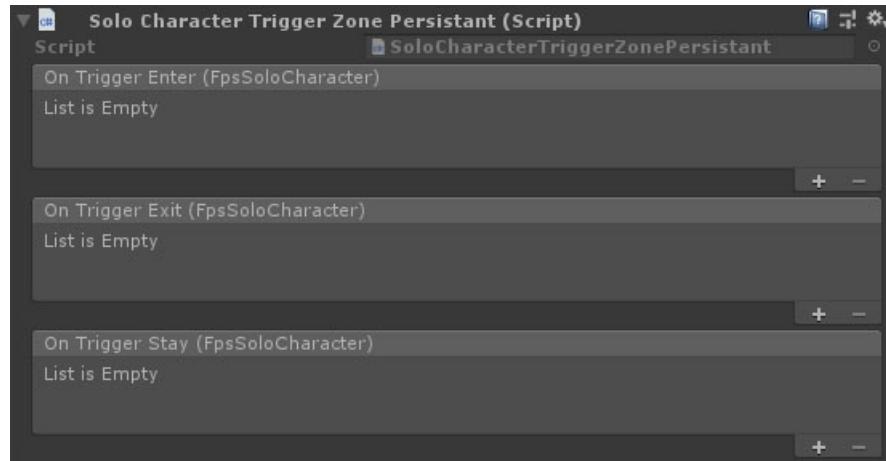
[Unity CapsuleCollider](#)

SoloCharacterTriggerZonePersistant MonoBehaviour

Overview

The SoloCharacterTriggerZonePersistant behaviour fires unity events when characters enter and exit.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.
On Trigger Stay	UnityEvent	The event that is fired each frame a character stays inside the trigger collider.

See Also

[Layers And Tags](#)

[Unity Events](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

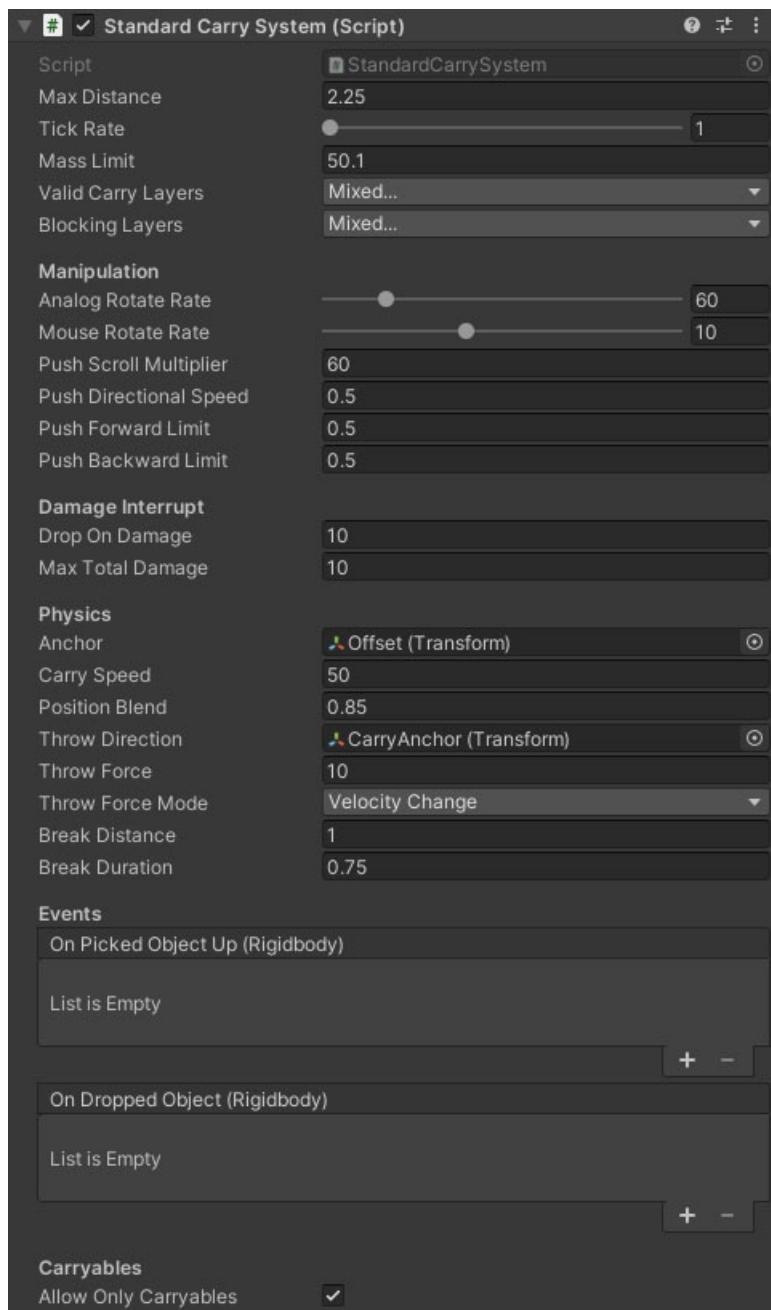
[Unity CapsuleCollider](#)

StandardCarrySystem MonoBehaviour

Overview

The StandardCarrySystem behaviour allows the character it's attached to to pick up any [rigidbody](#) object that has a [Carryable](#) behaviour attached.

Inspector



Properties

Name	Type	Description
Max Distance	Float	The maximum cast distance from the camera when checking for a carryable item.
Tick Rate	Integer	How frequently (in fixed frames) does the carry system cast forward to check for an object. Smaller numbers mean more responsive but more wasted calculations.

Name	Type	Description
Mass Limit	Float	The maximum mass of the object you can carry.
Valid Carry Layers	LayerMask	The valid layers for carryable objects. An object can not be picked up if it is not on one of these layers.
Blocking Layers	LayerMask	An extra set of layers that can block the raycasts to detect carryable objects. This is used to prevent picking up objects through walls or doors.
Analog Rotate Rate	Float	The number of degrees per second to turn the anchor at full analog turn. Be warned that setting this too high might turn the anchor faster than the object can turn, so causing it to bounce back and forth as the closest direction flips.
Mouse Rotate Rate	Float	A multiplier applied to mouse movement when turning the anchor. Be warned that setting this too high might turn the anchor faster than the object can turn, so causing it to bounce back and forth as the closest direction flips.
Push Scroll Multiplier	Float	A multiplier applied to mouse scroll to determine movement speed when pushing the carried object forwards/backwards.
Push Directional Speed	Float	The movement speed when pushing the carried object forwards/backwards.
Push Forward Limit	Float	The maximum forward distance the carried object can be pushed from its default starting position.
Push Backward Limit	Float	The maximum backwards distance the carried object can be pushed from its default starting position.
Drop On Damage	Float	If the character receives damage higher than this value in one go, then they will drop the object.
Max Total Damage	Float	If the character receives damage totalling this value since picking up the object, they will drop it.
Anchor	Transform	The anchor transform that carried objects will snap to. This should be a child of an object with the default carry position so that it can move and rotate relative.
Carry Speed	Float	The max movement speed for the carried object to match the anchor position.
Position Blend	Float	An event fired when the character picks an object up.
Throw Direction	Transform	A transform representing the throw direction (forwards / Z-axis). This would usually be the parent of the anchor transform.
Throw Force	Float	The force to apply in the throw direction to the carried object when throwing it.
Throw ForceMode	Dropdown	How the throw force should be applied. This uses the standard Unity force mode .

NAME	TYPE	DESCRIPTION
Break Distance	Float	A distance from the anchor where the break timer will start. Once the break timer hits the break duration, the object will be dropped. If the object moves back within the break distance then the timer resets.
Break Duration	Float	If the carried object is further from the anchor than the break distance for this amount of time then it will be dropped.
On Picked Object Up	UnityEvent	An event fired when the character picks an object up. Pointing this at a method that takes a Rigidbody parameter will pass the object that was picked up when the event is fired.
On Dropped Object	UnityEvent	An event fired when the character drops an object. Pointing this at a method that takes a Rigidbody parameter will pass the object that was dropped when the event is fired.
Allow Only Carryables	Boolean	With this enabled, you will only be able to pick up rigidbodies with a Carryable component attached. With it disabled you will be able to pick up any rigidbody.

See Also

[Carry System](#)

[Unity Events](#)

TriggerZoneColliderCounter MonoBehaviour

Overview

The TriggerZoneColliderCounter behaviour tracks and counts all the colliders that enter it with the specified layer mask.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Valid Layers	LayerMask	The valid layers for objects to track. Colliders on another layer will be ignored.

See Also

[Layers And Tags](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

[Unity CapsuleCollider](#)

Audio Systems

Overview

NeoFPS has a number of audio systems for specific cases. Some are specific or important to first person shooter games, while others are features that are missing from Unity and have had simple versions implemented in order to provide a more complete game experience out of the box.

Systems

Footsteps

In first person games, more than any other genre, [footsteps](#) are an important part of the immersive experience. NeoFPS uses a number of tools that tie footsteps to the character motion states and give designers full control over how the character's feet interact with the world.

Character Audio

NeoFPS has a customisable system that allows code and events to trigger character specific audio clips. The clips are identified using [generated constants](#), which allows developers to expand the available clips with ease.

Character audio is implemented through a combination of the [FpsCharacterAudioHandler MonoBehaviour](#) and the [FpsCharacterAudioData ScriptableObject](#).

Surface Audio

NeoFPS has a simple in-development system for identifying surfaces (LINK). The [SurfaceAudioData ScriptableObject](#) is used in a number of places throughout NeoFPS to specify audio for things such as footsteps, bullet hits and slides.

Contact Audio

The [ClipSetContactAudioHandler MonoBehaviour](#) is a simple component that can be added to physics objects in order to play audio on collisions. The audio is picked at random from a set of valid clips. You can also use a [SurfaceContactAudioHandler MonoBehaviour](#) which uses the [surface system](#) to pick the correct audio to play.

Audio Effects

NeoFPS uses a behaviour attached to the character camera that allows you to apply audio effects to its listener. The [FpsCharacterAudioEffects](#) behaviour can be assigned a number of [AudioEffectsPreset](#) assets which each represent a named preset such as "Underwater" or "Injured". You can then set the strength of each preset via the behaviour's API. An example use of this is the [SetAudioEffectStrength](#) motion graph behaviour (used to apply the underwater effect in the swimming demo).

See Also

[Surfaces](#)

[Generated Constants](#)

Footsteps

Overview

Footsteps in NeoFPS are handled using a combination of MonoBehaviours and [motion graph behaviours](#). This allows a great deal of flexibility and control over when and how footsteps are triggered. For more information on the specific behaviours, follow the links below.

Footstep Implementation

Basic Footsteps

The motion graph [SurfaceFootstepAudioSystem](#) is component added to the character that is used for basic footsteps on various surfaces. It is controlled via the [SurfaceFootstepAudioBehaviour](#) motion graph behaviour. This has properties for adjusting the frequency of the steps, the [surface audio library](#) to use, and parameters such as the cast direction and distance for surface checks. For each step taken it will choose an audio clip at random from the surface audio library to play. The behaviour can be attached to multiple motion graph states with different properties, so the system can be used to trigger different audio clips based on the character's motion state such as running, sneaking, walking, etc. The component on the character also fires a C# event on steps that can be used to tie it in to other systems.

Sliding

The motion graph [SlidingAudioBehaviour](#) is used to play looping audio when the character is sliding on various surfaces. Properties are available to change the [surface audio library](#), and to pitch shift the audio based on the speed of the character.

Lift-offs

The motion graph [SurfaceAudioBehaviour](#) can be used to trigger surface based audio clips in various situations, including jump lift-offs. It can be combined with the motion graph [PlayCharacterAudioBehaviour](#) to trigger character grunts and exertion noises when jumping.

Ladders

The motion graph [LadderAudioBehaviour](#) is used to play audio clips as the character ascends and descends a ladder. It has properties to control the frequency of the audio clips based on the climb speed, as well selecting the relevant [surface audio library](#).

Landings

Landings are handled slightly differently, as they need to react to the force of the impact in order to add flavour such as bone crunches or grunts for very heavy landings. When the character lands, the motion controller fires impact events that can be handled by any behaviour. The [demo character](#) implements this feature and has properties for soft and hard landings.

See Also

[The Motion Graph](#)

[Motion Graph Behaviours](#)

[Motion Graph FootstepAudioBehaviour](#)

[Motion Graph SlidingAudioBehaviour](#)

[Motion Graph SurfaceAudioBehaviour](#)

[Motion Graph PlayCharacterAudioBehaviour](#)

[Motion Graph LadderAudioBehaviour](#)

[SurfaceAudioData](#)

[FPS Characters](#)

Audio Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Sound Effects Ignore Volume Settings

If the sound effects you are triggering in your scenes don't seem to be affected by the in-game volume settings, please check the following:

- Your audio source's **Output** property is pointed at the relevant mixer group. You can check which mixer group is attached to the volume controls via the **Audio Manager** in the NeoFPS hub.
- The relevant **Volume Key** property for the output mixer group in the **Audio Manager** is correct. You can check this by double clicking the mixer group in the audio manager to be taken to the mixer group in the project hierarchy. Double click the parent audio mixer to open Unity's audio mixer editor, and then check the **Exposed Parameters** dropdown in the top right.

Footsteps or Impact Sounds Are Wrong

The footsteps and audio impacts use the [surface system](#) to detect what type of surface is being interacted with. Per-surface audio clips are specified in [SurfaceAudioData](#) assets that are consumed by the **Surface Manager** found in the NeoFPS hub, and by the [FootstepAudio](#) and [SurfaceFootstepAudio](#) motion graph behaviours that are attached to the relevant movement states in the motion graph [MotionGraph](#).

To troubleshoot incorrect audio, please check the following:

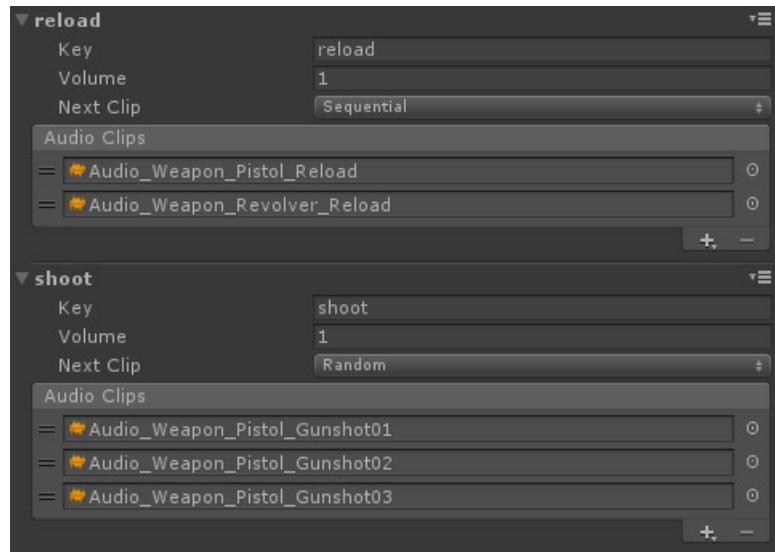
- If your bullet impacts are playing incorrect audio, please check that the **Impact Audio** property in the **Surface Manager** is pointed to the correct audio data asset.
- If your footsteps are playing incorrect audio when in a specific movement state, please open the motion graph for your character, navigate to the relevant state (remember you can double click to enter sub-graphs), and check the **Audio Data** property is pointed at the correct asset.
- If both footsteps and impacts are consistently playing the incorrect audio for certain surface types, this suggests that the order of the surface types in the **FpsSurfaceMaterial GeneratedConstant** has changed. Generated constants are index based, so adding a new one in the middle of the list will offset all constants that come after it. If you do this you will need to update your collider objects to check that their **SimpleSurface** component is using the correct surface type. You will also need to check your **SurfaceAudioData** assets as mentioned above to check that the surfaces are using the correct audio clips (you can copy/paste between surface types using the context menu for each surface).

AnimationEventAudioPlayer MonoBehaviour

Overview

The AnimationEventAudioPlayer behaviour is attached to objects with an [Animator](#) component and handles any animation events sent.

Inspector



Properties

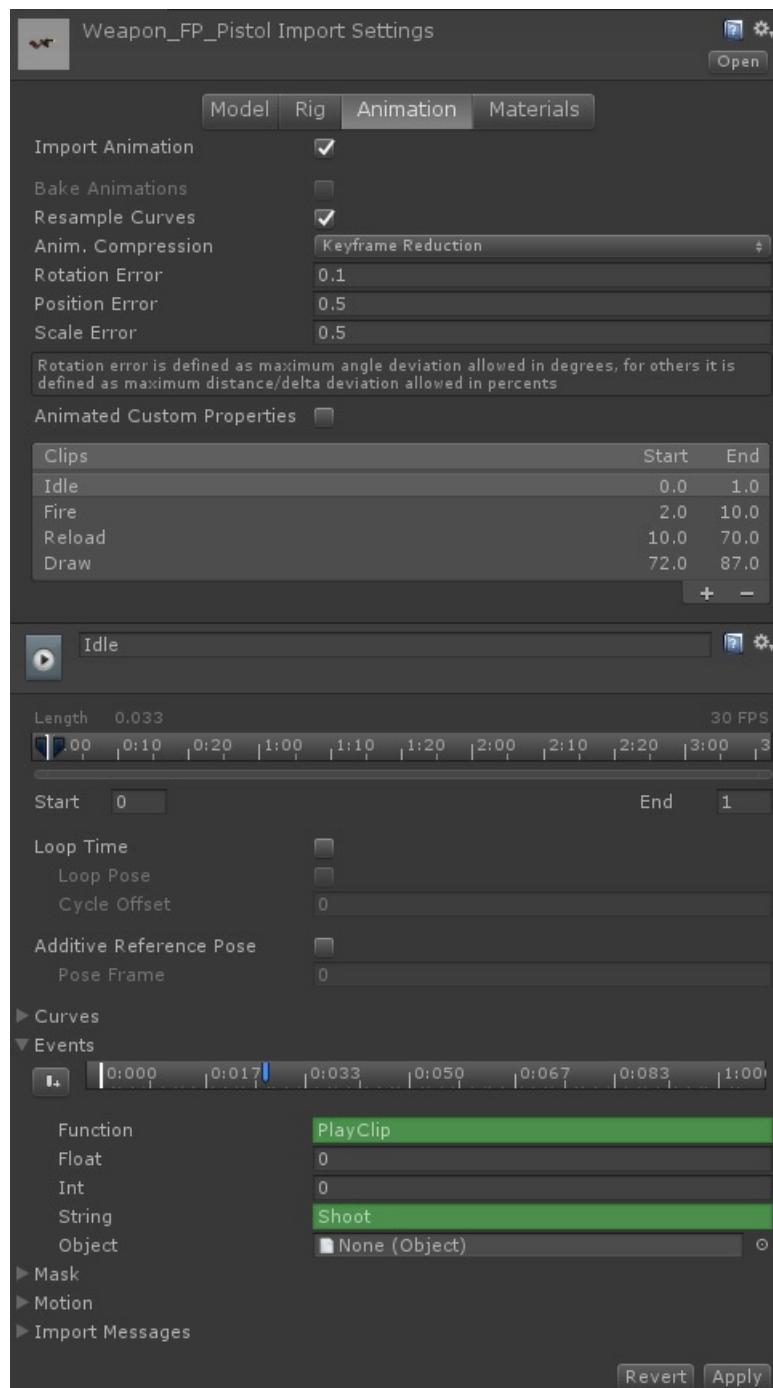
NAME	TYPE	DESCRIPTION
Audio Source	AudioSource	The audio source to play from.

The **Add New Set** button will add a new audio clip set to the component with the following properties:

NAME	TYPE	DESCRIPTION
Key	String	The name of the clip set, used as the parameter of the animation events
Next Clip	Dropdown	How the next clip should be selected. Available options are: Sequential loops through the clips in order while Random selects a clip at random.
Volume	Float	The volume to play the clip at.
Clips	AudioClip Array	The audio clips to choose from.

Triggering Audio Clips

The audio clips are triggered by adding [animation events](#) to the animation clips of the animated object. This is usually done through the import settings in the inspector.



The above example has an event set up that plays an audio clip from the "Shoot" set. The important properties are the **Function** property which must be set to **Play Clip**, and the **String** property which should match the **Key** in the AnimationEventAudioPlayer.

See Also

[Animation Events](#)

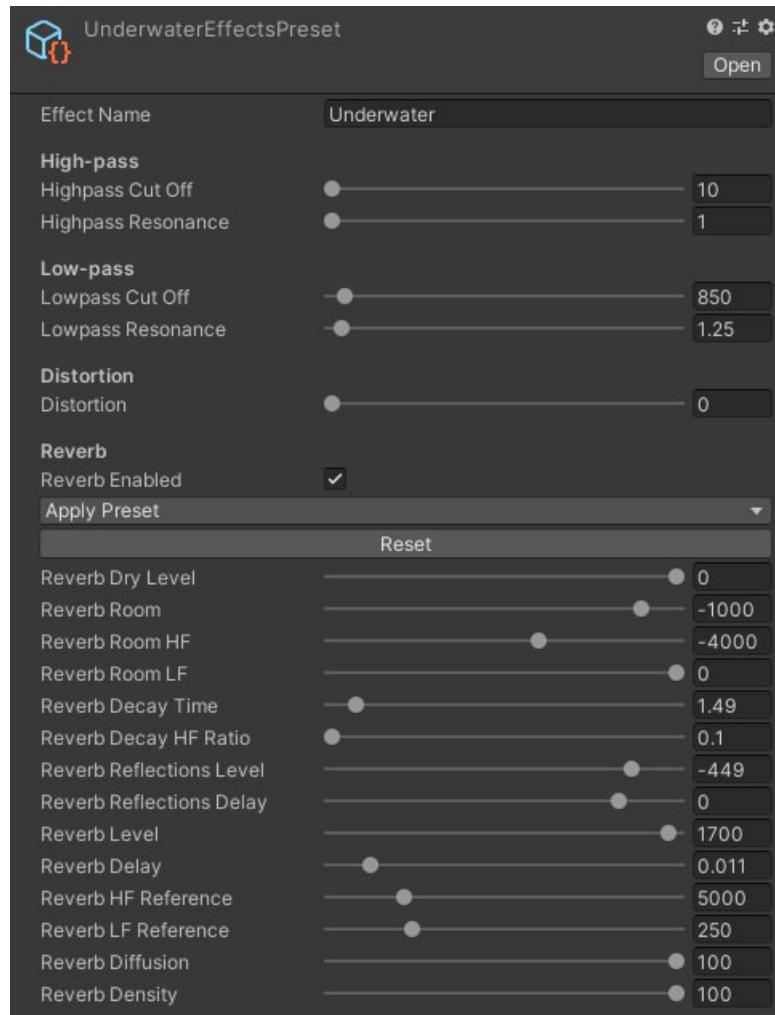
AudioEffectsPreset ScriptableObject

Overview

The AudioEffectsPreset scriptable object specifies a preset name and then a range of audio effect settings that will be applied to the effects on the character listener. This is managed with the [FpsCharacterAudioEffects](#) behaviour. The available effects include:

- Low-pass filter
- High-pass filter
- Distortion
- Reverb

Inspector



Properties

Name	Type	Description
Effect Name	String	The name of this effects preset.
Highpass Cut Off	Float	The cut-off frequency for the high pass effect. Only frequencies above this will pass through to the final mix.
Highpass Resonance	Float	The resonance determines how much the filter's self-resonance is damped. A higher value means more resonance.

NAME	TYPE	DESCRIPTION
Lowpass Cut Off	Float	The cut-off frequency for the low pass effect. Only frequencies below this will pass through to the final mix.
Lowpass Resonance	Float	The resonance determines how much the filter's self-resonance is damped. A higher value means more resonance.
Distortion	Float	The amount of distortion to apply to the sound.
Reverb Enabled	Float	Is reverb enabled with this preset.
Reverb Dry Level	Float	Mix level of dry signal in output in mB. Ranges from -10000.0 to 0.0.
Reverb Room	Float	Room effect level at low frequencies in mB. Ranges from -10000.0 to 0.0.
Reverb Room HF	Float	Room effect high-frequency level in mB. Ranges from -10000.0 to 0.0.
Reverb Room LF	Float	Room effect low-frequency level in mB. Ranges from -10000.0 to 0.0.
Reverb Decay Time	Float	Reverberation decay time at low-frequencies in seconds. Ranges from 0.1 to 20.0.
Reverb Decay HF Ratio	Float	High-frequency to low-frequency decay time ratio. Ranges from 0.1 to 2.0.
Reverb Reflections Level	Float	Early reflections level relative to room effect in mB. Ranges from -10000.0 to 1000.0.
Reverb Reflections Delay	Float	Early reflections delay time relative to room effect in mB. Ranges from -10000.0 to 2000.0.
Reverb Level	Float	Late reverberation level relative to room effect in mB. Ranges from -10000.0 to 2000.0.
Reverb Delay	Float	Late reverberation delay time relative to first reflection in seconds. Ranges from 0.0 to 0.1.
Reverb HF Reference	Float	Reference high frequency in Hz. Ranges from 20.0 to 20000.0. Default is 5000.0 Hz.
Reverb LF Reference	Float	Reference low-frequency in Hz. Ranges from 20.0 to 1000.0.
Reverb Diffusion	Float	Reverberation diffusion (echo density) in percent. Ranges from 0.0 to 100.0.
Reverb Density	Float	Reverberation density (modal density) in percent. Ranges from 0.0 to 100.0.

See Also

[FpsCharacterAudioEffects](#)

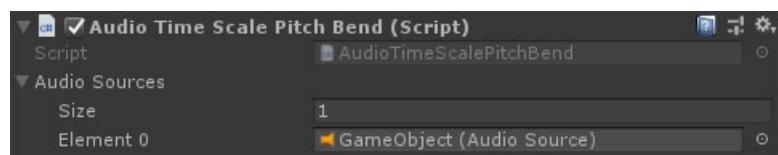
[Audio Effects][unity-audioeffects]

AudioTimeScalePitchBend MonoBehaviour

Overview

The AudioTimeScalePitchBend behaviour alters the pitch of an audio source to match changes to time scale. Pitch also affects the duration of an audio clip, so halving the pitch will double the duration. By default, Unity does not slow audio to match the time scale, so this behaviour compensates for that.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Sources	<audiosource> Array</audiosource>	The audio sources to modify based on time scale.

ClipSetContactAudioHandler MonoBehaviour

Overview

The ClipSetContactAudioHandler behaviour is attached to physics objects and triggers audio clips when they collide with other objects or the environment.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Impulse	Float	The minimum impulse between this object and the object it collides with for an impact noise to be played.
Min Delay	Float	The minimum time between impact sounds.
Clips	AudioClip Array	The audio clips to choose from on impact.

CriticalHitAudioPlayer MonoBehaviour

Overview

The CriticalHitAudioPlayer behaviour is will play an audio clip whenever the local player character inflicts a critical hit on a target. This can be attached to the game UI / HUD since you should only have one in your scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Impulse	Float	The minimum damage required to trigger the audio (still requires a critical hit).
Min Delay	Float	The minimum time between critical hit sounds.
Clips	AudioClip Array	The audio clips to choose from on a critical hit being detected.

See Also

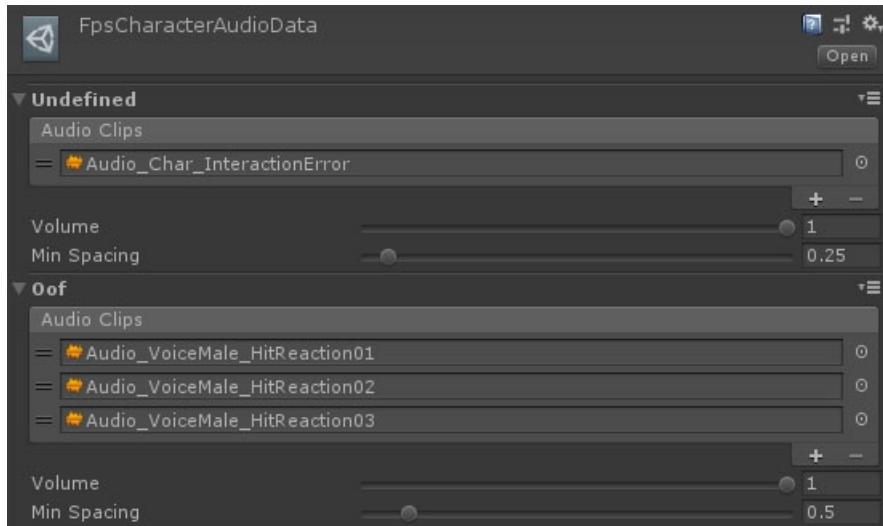
[Generated Constants](#)

FpsCharacterAudioData ScriptableObject

Overview

The FpsCharacterAudioData scriptable object specifies audio to play for the different character audio keys.

Inspector



Properties

The data is divided into sections for each key in the FpsCharacterAudio [generated constant](#). Each section contains the following properties:

NAME	TYPE	DESCRIPTION
Audio Clips	AudioClip Array	A selection of audio clips to pick from. Will be selected at random to prevent repetition.
Volume	Float	The volume to play the clip at.
Min Spacing	Float	New clips will be blocked from playing for this duration after a clip plays. Prevents rapid fire audio.

See Also

[Generated Constants](#)

[Unity AudioClip](#)

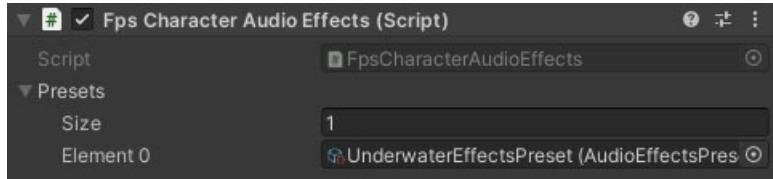
FpsCharacterAudioEffects MonoBehaviour

Overview

The FpsCharacterAudioEffects behaviour manages the different audio effects available to the character and blends between them. These are used for things like underwater or damage effects and can alter the way the character hears scene audio via the following [effects](#):

- Low-pass filter
- High-pass filter
- Distortion
- Reverb

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	AudioEffectsPreset Array	The different audio effect presets available to this character.

See Also

[AudioEffectsPreset](#)

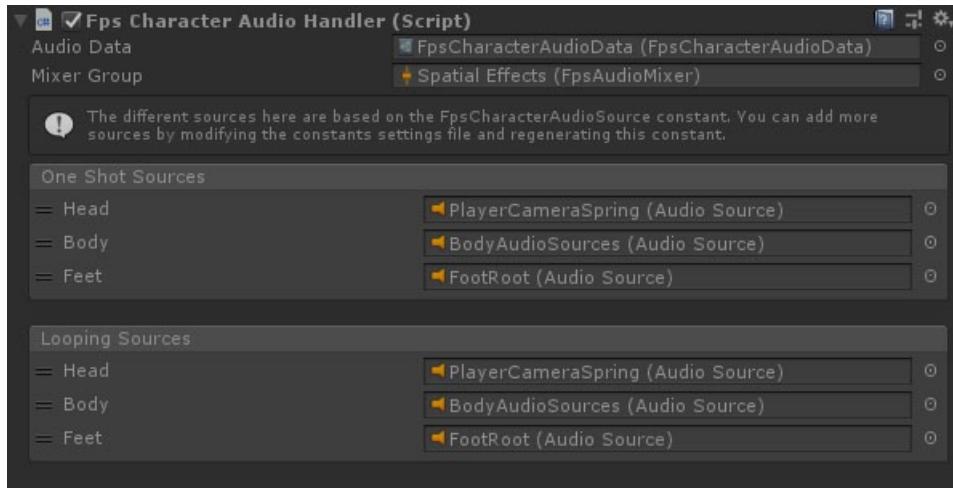
[Audio Effects](#)

FpsCharacterAudioHandler MonoBehaviour

Overview

The FpsCharacterAudioHandler plays audio clips from a library of character audio files.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Audio Data	FpsCharacterAudioData	The character audio library to use.
Mixer Group	MixerGroup	The mixer group for character sound effects.
Sources	<audiosource> Array</audiosource>	The sources to use for playing audio. One for each FpsCharacter AudioSource generated constant.

See Also

[FpsCharacterAudioData](#)

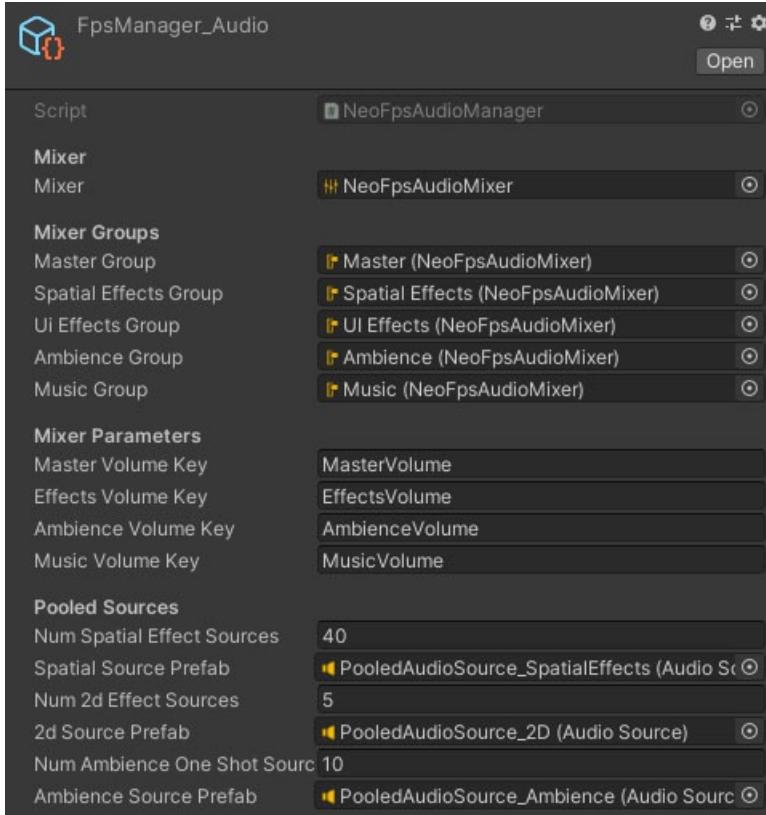
[Generated Constants](#)

NeoFpsAudioManager ScriptableObject

Overview

The NeoFpsAudioManager scriptable object specifies the audio mixer and outputs for audio in a NeoFPS project, and provides pooled audio sources for different effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Mixer	[AudioMixer][unity-audiomixer]	The audio mixer for the project.
Master Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group that controls the overall volume.
Spatial Effects Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for spatial sound effects.
Ui Effects Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for UI sound effects.
Ambience Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for ambient sound effects and looping ambient audio.
Music Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for music.
Master Volume Key	String	The name of the master volume parameter on the audio mixer.

NAME	TYPE	DESCRIPTION
Effects Volume Key	String	The name of the volume parameter that controls sound effects volume.
Ambience Volume Key	String	The name of the volume parameter on the audio mixer for ambient loops and effects.
Music Volume Key	String	The name of the volume parameter on the audio mixer for the music audio.
Num Spatial Effect Sources	Integer	The number of pooled audio sources for spatial sound effects.
Spatial Source Prefab	[AudioSource][unity-audioplayer]	An optional prefab for the spatial effects sources. If not provided then the objects will be created from scratch.
Num 2D Effect Sources	Integer	The number of pooled audio sources for 2D sound effects.
2D Source Prefab	[AudioSource][unity-audioplayer]	An optional prefab for the 2D effects sources. If not provided then the objects will be created from scratch.
Num Ambience One Shot Sources	Integer	The number of pooled audio sources for ambient sound effects.
Ambience Source Prefab	[AudioSource][unity-audioplayer]	An optional prefab for the ambience effects sources. If not provided then the objects will be created from scratch.

See Also

[Surfaces](#)

[Generated Constants](#)

[\[Unity AudioClip\]\[unity-audioclip\]](#)

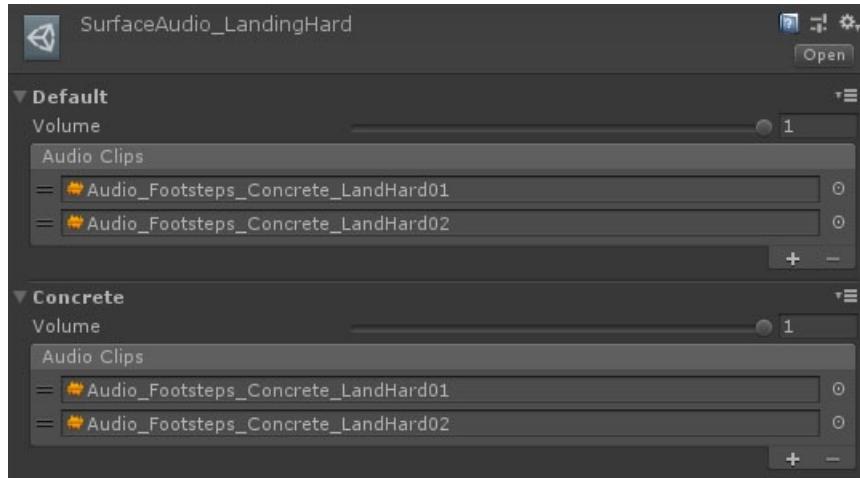
[\[AudioMixer\]\[unity-audiomixer\]: https://docs.unity3d.com/Manual/class-AudioMixer.html](#) [\[unity-audioclip\]: https://docs.unity3d.com/Manual/class- AudioClip.html](#) [\[unity-audioplayer\]: https://docs.unity3d.com/Manual/class- AudioSource.html](#)

SurfaceAudioData ScriptableObject

Overview

The SurfaceAudioData scriptable object specifies audio clips for each surface type. It is used to organise clip sets such as impact sounds, footsteps and slides.

Inspector



Properties

The SurfaceAudioData entries are grouped based on the Surface [generated constant](#). Each surface has the following properties.

NAME	TYPE	DESCRIPTION
Volume	Float	The volume to play the clips at.
Clips	AudioClip Array	Audio clips (will be picked at random).

See Also

[Surfaces](#)

[Generated Constants](#)

[Unity AudioClip](#)

SurfaceContactAudioHandler MonoBehaviour

Overview

The SurfaceContactAudioHandler behaviour is attached to physics objects triggers sound effects when they collide with other objects or the environment. Playing the audio itself is handled by the [SurfaceManager](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Impulse	Float	The minimum impulse between this object and the object it collides with for an impact noise to be played.
Min Delay	Float	The minimum time between impact sounds.

See Also

[Surface Manager](#)

Inventory

Overview

NeoFPS has its own inventory system that it uses for all of its weapons and items. The inventory is implemented as separate components to the itemas themselves so it should be easy to replace with an alternative system if desired.

The NeoFPS inventory system is implemented in 2 parts: inventory and quickslots.

Inventory



The inventory is a container of items. It manages item ownership and quantities, and invokes events to react to changes. All of the example inventories inherit from a single base class `FpsInventoryBase`. This base class handles both the inventory storage and the quick-slots. It also provides inspector properties for initialising the inventory's starting contents and the priority order for the slot items.

The demo implementations of the inventory are not fixed. The inventory is referenced externally via the `IInventory` interface, meaning that it can be swapped out with another implementation and the system as a whole will adapt.

Inventory items are objects that implement the `IInventoryItem` interface. The interface has properties for quantity and ownership, along with methods that are called when added or removed from an inventory. The example inventory items inherit from a single base class `FpsInventoryItemBase` which provides a standard implementation for the above properties and methods.

Quick Slots / Hotbar



The quick-slots system binds items to slots that can be mapped to inputs. In a simple, traditional FPS system that would mean each slot being mapped to a number key on the keyboard. It could also mean a slot for each direction on a game controller d-pad. You can change what inputs are used for each slot via the inventory input handler. For example, you could add an input for "quick melee" and use that to trigger a slot with a quick-use melee item on it.

All quick-Slot items implement the `IQuickSlotItem` interface. This has properties and methods for selection and for dropping.

NeoFPS comes with a number of example inventories that implement quick slots in different ways to model popular first person shooters. For more information see [Inventory Examples](#).

Wieldables

Wieldables are weapons and items that are equipped and held in the hands and then triggered on command. An example is a gun, and all of the demo weapons are set up as wieldables.

To be equipped, your weapon prefabs need either an `FpsInventoryWieldable` or an `FpsInventoryWieldableSwappable` behaviour attached depending on the inventory type.

Quick-Use Items

Alongside equipping wieldable weapons, you can also create quick-use weapons and items. When the quick slot for these items is selected, the weapon is equipped and then immediately performs its action and then unequips again. This can mean a gun that draws, immediately fires and then is re-holstered. It can also mean quick throwing a grenade or swinging a melee weapon. You can also use this in combination with wieldable tools for actions like applying healing bandages.

There are a number of options for what to do with the previously equipped weapon when performing a quick action. The simplest option unequips the active weapon before starting the quick action and then immediately re-equips it once the action is completed. You can also send an animator bool to the equipped weapon and block its input. This can be used to show an animation such as lowering the left hand out of view so that it can be used in the quick action weapon. If you wanted you could also keep using the equipped weapon as though nothing was happening. This might be useful if the quick action is something that doesn't require hands such as a kick.

A quick use item is created by attaching either an `FpsInventoryQuickUseItem` or an `FpsInventoryQuickUseSwappableItem` behaviour to a weapon instead of the wieldable behaviours listed above.

Instant Use Items / Consumables

Lastly, you can also use quick-slots to trigger instant use items. Instant use items will apply their effects immediately and do nothing to interrupt the equipped weapon. By adding a `ConsumeItemInstantAction` to the item, the item essentially becomes a

consumable, and each use will reduce the count in the player's inventory.

An instant use item is created by adding an [FpsInventoryInstantUseItem](#) or an [FpsInventoryInstantUseSwappableItem](#) behaviour to a prefab and then attaching one or more actions. The following instant actions are provided:

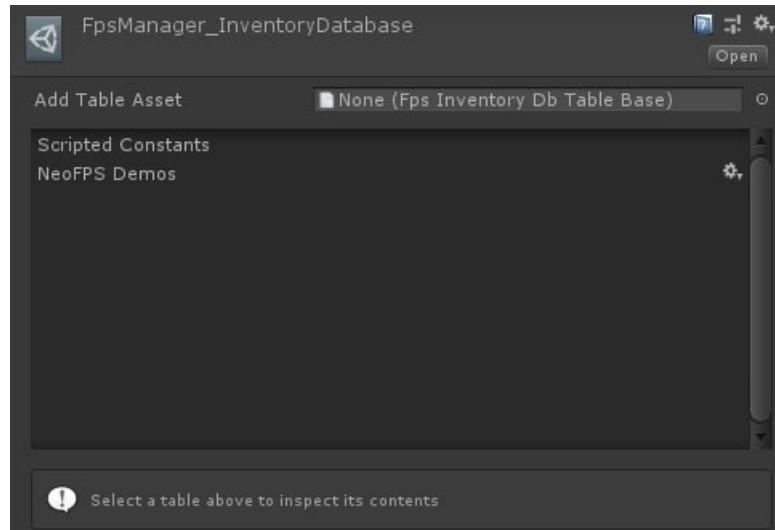
NAME	DESCRIPTION
ConsumeItemInstantAction	Will reduce the quantity of the item in the character's inventory.
HealPlayerInstantAction	Heals the player character that uses the item.
PlayAudioInstantAction	Plays an audio clip.
RechargeShieldsInstantAction	Recharges one or more shield bars.
UnityEventInstantAction	Fires a unity event that you can subscribe to via the inspector.

Inventory Pickups

Inventory items pickups can be interactive, meaning you need to look at them and hit use, or contact based, meaning you need to walk over them.

The simplest way to create an inventory pickup is using the [Pickup Wizard](#) in the NeoFPS Hub. There are also specialised pickups for modular firearms that track ammo count, and for other wieldable items such as melee and thrown weapons. These can all be created from the wizard.

The Inventory Database



The inventory database is a system for managing the inventory keys used to identify each unique item. The database is made up of tables, which are used to gather inventory IDs together. The inventory system used to use [GeneratedConstant](#) based keys. The new database system has one fixed table that references the old **FpsInventoryKey** constants and uses them as keys. This allows you to use the constants to reference inventory item IDs directly in code - useful for items like the keyring, which is a unique, but constant item.

You can create an inventory database table for your project by right-clicking in the project browser and selecting *Create/NeoFPS/Inventory/Database Table*. This allows you to add your own keys without the risk of clashing with any changes to the NeoFPS demo inventory keys in future updates. Once you have created a table, add it to the database via the NeoFPS Hub: *Managers/Inventory Database*.

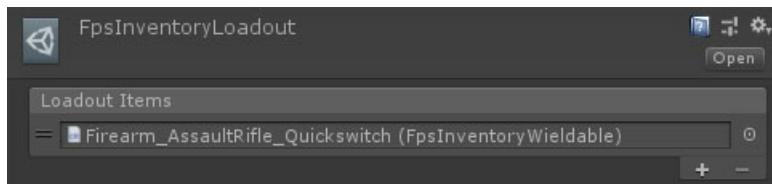
To assign an ID to an object, clicking the ID button will show the inventory database browser.



From here you can browse the different tables to select an ID. You can filter by name using the filter field at the top. You can also add new keys from the browser itself. In the NewEntry section, select the table you want to add to, input a name, and hit **Add New Entry** button. This will add the a new entry to the selected table, and apply its ID to the object you are editing.

Inventory Loadouts

Inventory loadouts are a simple list of inventory objects that should be added to a character's inventory.



You can assign this to a game mode component, and it will apply the loadout on spawning a player character instead of using the character's starting items.

See Also

[Inventory Examples](#)

Inventory Examples

Overview

NeoFPS comes with a number of example inventories that are modelled after popular first person shooters.

Base Classes

All of the example inventories inherit from a single base class `FpsInventoryBase`. This base class handles both the inventory storage and the quick-slots. It also provides inspector properties for initialising the inventory's starting contents and the priority order for the quick slots.

The example inventory items inherit from a single base class `FpsInventoryItemBase`. For simple items such as ammo or armour, the items use the `FpsInventoryItem` behaviour which extends this base class and specifies the ID and max quantity.

If an item is meant to be selected or used by the player via an input such as the number keys, then the items must also implement the `IQuickSlotItem` interface. There are variants for these based on the inventory type, so the options are listed below.

Standard PC Inventory



The "standard" quick-switch inventory emulates old-school FPS games. Each weapon has a set slot corresponding to the keyboard keys 1 to 0. You can also cycle through the weapons using the mouse wheel and inputs for previous and next weapon. Lastly, the inventory also implements a quick-switch system so you can switch between your current and last weapon as in games such as Counter Strike.

Most of the NeoFPS demos use a character with the standard inventory. You can find weapon and item prefabs at the following locations:

- `Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\QuickSwitchInventory` for the first person weapons and pickups/drops.
- `Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\Ammo` for the ammo pickups

To use the standard inventory, your player character should have the `FpsInventoryQuickSwitch` behaviour attached to its root.

The standard inventory accepts the basic `FpsInventoryItem` objects. For wieldables such as weapons, the prefabs should use the `FpsInventoryWieldable` behaviour. For quick use items such as quick throw weapons, your prefabs should use the `FpsInventoryQuickUselItem` behaviour. For instant use items such as instant heals you should use the `FpsInventoryInstantUselItem` behaviour. Each of the above will have a **Quick Slot** property that will correspond to their slot number on the hot bar (be aware that the first item will be slot 0, not slot 1, since arrays start at 0).

Stacked Inventory



The stacked inventory emulates games like half-life with larger, more structured inventories than the older FPS games could

manage. Weapons and items are organised into stacks. Each stack holds a specific type of weapon such as melee, pistols, heavy, thrown. Selecting the same slot multiple times cycles through the weapons in that group. The mouse wheel and previous / next weapon inputs cycle through each of the weapons in a stack before moving onto the previous / next stack slots. The order of the weapons within a stack is fixed.

The slot index of an item added to a stacked inventory must take the stack into account. The maximum stack size is 10. Slot indices 0-9 fall in the first stack, 10-19 in the second, 20-29 in the third, and so on.

The stacked inventory requires a different HUD setup to the others due to how it organises the items.

This inventory is also well suited to console based FPS games where, for example, each direction on the D-Pad is a stack.

The relevant assets for the stacked inventory can be found at the following locations:

- `Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\FeatureDemo_InventoryStacked.unity` for a demo scene using the inventory
- `Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_CharacterStackedInventory.prefab` for a player character that uses the stacked inventory
- `Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\StackedInventory` for the first person weapons and pickups/drops.
- `Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_HUDStacked.prefab` for a replacement inventory HUD that is set up for stacked inventories

To use the stacked inventory, your player character should have the `FpsInventoryStacked` behaviour attached to its root.

The stacked inventory accepts the basic `FpsInventoryItem` objects. For wieldables such as weapons, the prefabs should use the `FpsInventoryWieldable` behaviour. For quick use items such as quick throw weapons, your prefabs should use the `FpsInventoryQuickUselItem` behaviour. For instant use items such as instant heals you should use the `FpsInventoryInstantUselItem` behaviour. Each of the above will have a **Quick Slot** property that will correspond to their slot and stack position. Since stacked slots can contain ten items, a **Quick Slot** setting of 0-9 will mean the item sits in the first slot. 10-19 will sit in the second, and so on.

Swappable Inventory



The swappable inventory emulates FPS games that keep a constrained inventory such as 2 primary weapons, one melee and one throwable. This is a common inventory setup since more FPS games have started launching on console alongside or even ahead of PC.

The swappable inventory groups weapons and items into types similarly to the stacked inventory. A group can hold a fixed amount of weapons. If the group is full then the last selected weapon from that group is dropped when the character picks up another weapon of that type. In simple cases a group will only be able to hold a single weapon or item. A group can also be larger, and will encompass multiple quick slots. In this case, the order of weapons within the group is defined by the order they are dropped and picked up only.

The relevant assets for the swappable inventory can be found at the following locations:

- `Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\FeatureDemo_InventorySwappable.unity` for a demo scene using the inventory
- `Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_CharacterSwappableInventory.prefab` for a player character that uses the swappable inventory
- `Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\SwappableInventory` for the first person weapons and pickups/drops.

To use the swappable inventory, your player character should have the `FpsInventorySwappable` behaviour attached to its root.

The swappable inventory accepts the basic [FpsInventoryItem](#) objects. For wieldables such as weapons, the prefabs should use the [FpsInventoryWieldableSwappable](#) behaviour. For quick use items such as quick throw weapons, your prefabs should use the [FpsInventoryQuickUseSwappableItem](#) behaviour. For instant use items such as instant heals you should use the [FpsInventoryInstantUseSwappableItem](#) behaviour. Each of the above will have a **Category** property that dictates which quick slots it can sit in. If all the slots for that category are full then by default, the last selected item from that category will be dropped to make room for the new one.

See Also

[Inventory](#)

Inventory Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Weapon Doesn't Appear In Hot-Bar But No Errors In Console

This usually means that the quick-slot index of the item is incorrect. There are a few possible causes for this. Please check the following:

- If your character is using an **FpsInventorySwappable** inventory, then your weapon **must** use the **FpsInventoryWieldableSwappable** wieldable component. If not, then it should use the **FpsInventoryWieldable** component.
- When using the quick-switch inventory, the maximum quick-slot index is 9.
- Make sure that you do not have multiple wieldable components on your weapon. The inventory will only detect the first one.

Player Character Has Different Loadout On Start To Expected

If your character is spawning with a very different loadout to the one specified on the character's inventory (as opposed to some of the objects aren't being correctly added) then check the game mode / spawner (in the demos this is the object called **SimpleSpawnerAndGameMode**). The **FpsSoloGameMinimal** component has a property called **Starting Loadout** that is used to override the inventory loadout of the character on spawn.

Can't Drop Weapon

In order to drop a weapon from your hands, the weapon's **FpsInventoryWieldable** or **FpsInventoryWieldableSwappable** wieldable component must have an object set for its **Drop Object** property. This should be a prefab with a **ModularFirearmDrop** component in the case of firearms (this correctly sets the ammo in the dropped weapon based on the held one's reload module) or a **FpsInventoryWieldableDrop** for other types of weapons / items.

When I Drop A Weapon And Pick It Up Again It's Broken

The drop objects mentioned above are essentially inventory pickups that point to a specific wieldable prefab. If the drop object points to a different wieldable prefab than the one the held weapon was created from, then you will get a mismatch. To troubleshoot this, follow these steps:

- Select the root object of the weapon prefab and find its wieldable component (**FpsInventoryWieldable** or **FpsInventoryWieldableSwappable**).
- Select the value for the **Drop Object** property to highlight that prefab in the project view.
- Select the drop object prefab, and find its **InteractivePickup** component.
- Make sure that the **Item** property on this component points at the intended weapon.

Be sure to check that you're not using the same drop object for multiple weapons, as fixing it for one will break it for the other.

When you create a new weapon, you can quickly create a new drop object prefab specifically for it using the pickup wizard. To do this:

- Open the hub via the Unity menu: *Tools/NeoFPS/NeoFPS Hub*
- Expand the **Wizards** section and select the **Pickup Wizard**
- Hit the **Create From Scratch** button
- Under the **Pickup Type** option, select either **Wieldable Item Drop** or **Modular Firearm Drop** depending on the weapon you created
- Follow the wizards instructions to create a drop object
- (Optional) there is an **Add To Gun / Add To Wieldable** option on the second page of the wizard. You can set this to true and select the weapon here, or you can add the drop object to the wieldable component on your weapon manually later on

Multiple Weapons Appear On Start

This is something that can happen if one or more of the weapons in the character's starting loadout use the wrong wieldable type, or have invalid inventory keys. At start, the inventory/spawner instantiate each of the prefabs and then add them to the inventory. If they are rejected due to one of those errors, then the object will still exist, but not be tied into the inventory system correctly. To fix this:

- Check if your scene's spawner and game mode is using a loadout asset to override the character's starting items
- Select either the character prefab or the loadout asset based on the above result, and find the **Starting Items** array (this will be on the inventory component for the character)
- Work your way through the starting items, checking each weapon against the following:
 - If the character is using an **FpsInventorySwappable** component, then the weapon *must* use an **FpsInventoryWieldableSwappable** component. If not, it should be using an **FpsInventoryWieldable** component
 - Check that the wieldable component's **Inventory ID** property is not set to **Not Set** and pick an ID if it is
 - Check that the weapon only has a single wieldable component

Contact Pickup Isn't Working

There are 2 possible causes for this. The first is that the physics is not set up correctly. To check this:

- Open the pickup prefab
- Make sure that the pickup has the following:
 - The relevant pickup component, such as **InventoryItemPickup**
 - A collider that has **Is Trigger** set to **true**
 - A **PickupTriggerZone** component
- Check that the layer of the object is set to **TriggerZones**

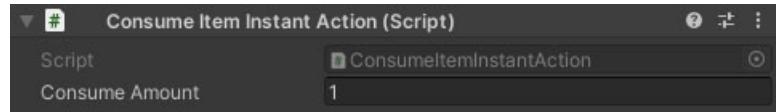
The other possible cause for inventory pickups is that the **InventoryItemPickup** component on the pickup object has nothing set for its **Item Prefab** property. Make sure that this is pointing at the item prefab you want the player to pick up.

ConsumeItemInstantAction MonoBehaviour

Overview

The ConsumeItemInstantAction behaviour will decrement the number of the [FpsInventoryInstantuseItem](#) or [FpsInventoryInstantuseSwappableItem](#) that it's attached to in the player's inventory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Consume Amount	Integer	How many of the object to consume each time it's used.

See Also

[Inventory](#)

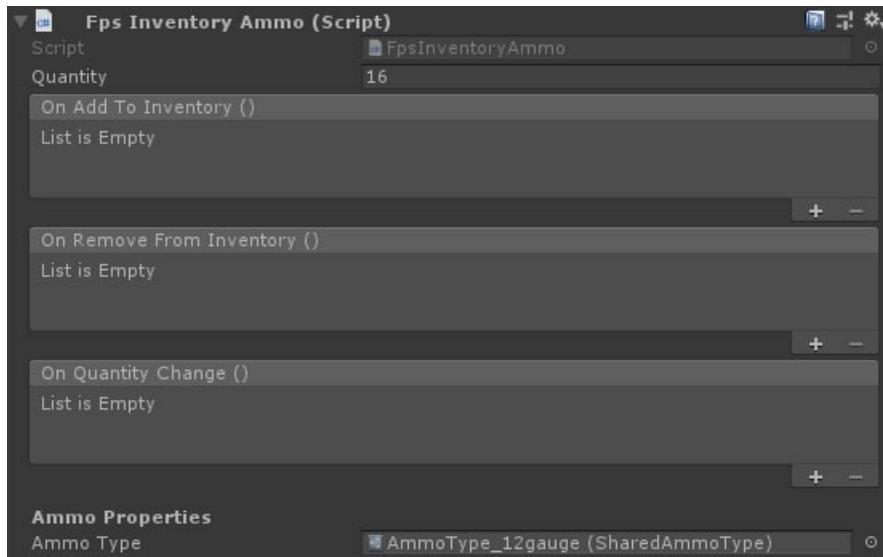
[Unity Rigidbody](#)

FpsInventoryAmmo MonoBehaviour

Overview

The FpsInventoryAmmo behaviour is an ammo type for the [modular firearms](#) that is stored in a character's inventory. It allows the ammo to be shared between different weapons.

Inspector



Properties

The FpsInventoryAmmo behaviour inherits the properties from [FpsInventoryItem](#). It also adds the following:

NAME	TYPE	DESCRIPTION
Ammo Type	SharedAmmoType	The type of ammo.

See Also

[FpsInventoryItem](#)

[Modular Firearms](#)

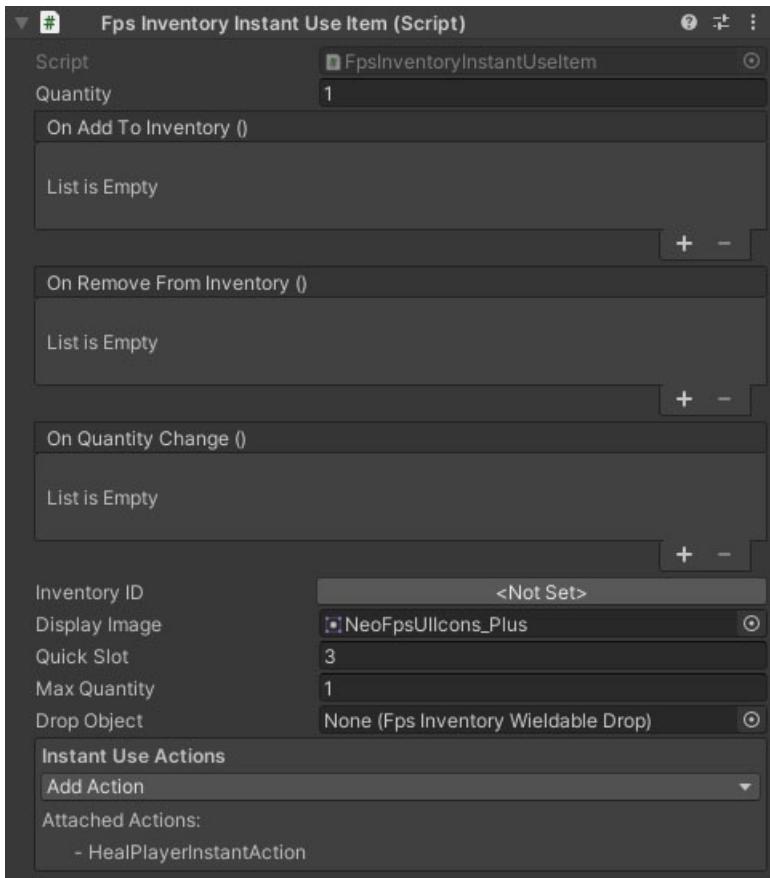
[SharedAmmoType](#)

FpsInventoryInstantUseItem MonoBehaviour

Overview

The FpsInventoryInstantUseItem behaviour is an inventory item that will be used immediately when its quick-slot is selected, and perform the attached actions.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Quick Slot	Int	The quick slot the item should be placed in. If you are using a stacked inventory, remember that each stack is 10 slots (0-9 = stack 1, 10-19 = stack 2, etc).
Max Quantity	Int	The maximum quantity you can hold.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.

NAME	TYPE	DESCRIPTION
Actions	Array	The actions to perform when the instant use item is triggered. You can add new actions from the dropdown here, or via the Add Component menu for the gameobject.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

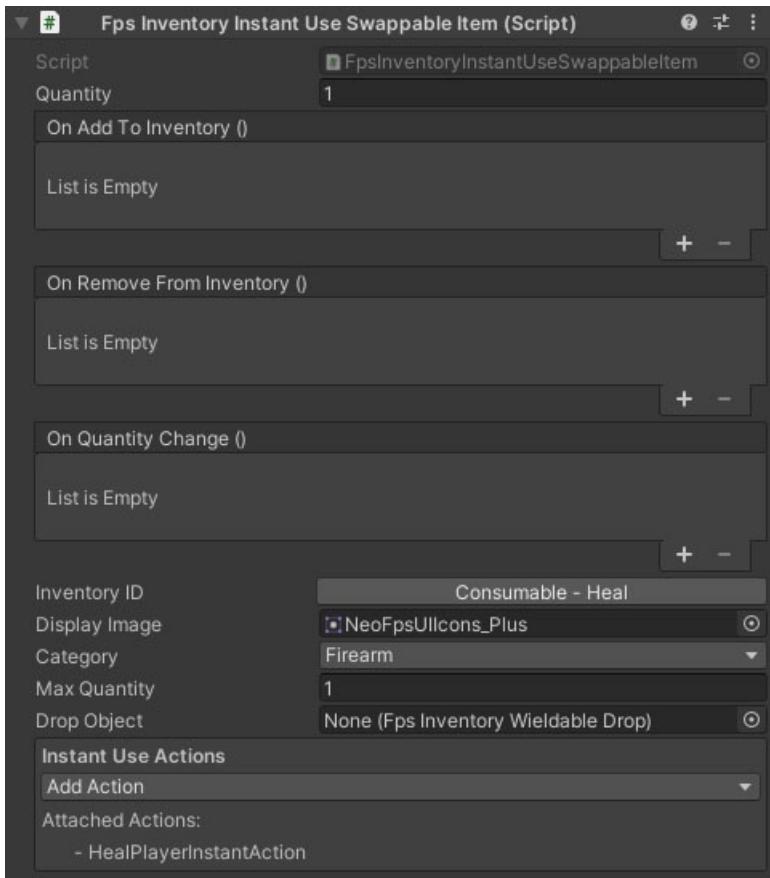
[FpsInventoryWieldableDrop](#)

FpsInventoryInstantUseSwappableItem MonoBehaviour

Overview

The FpsInventoryInstantUseSwappableItem behaviour is an inventory item that will be used immediately when its quick-slot is selected, and perform the attached actions.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Max Quantity	Int	The maximum quantity you can hold.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.
Category	FpsSwappableCategory	The wieldable category.
Actions	Array	The actions to perform when the instant use item is triggered. You can add new actions from the dropdown here, or via the Add Component menu for the gameobject.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

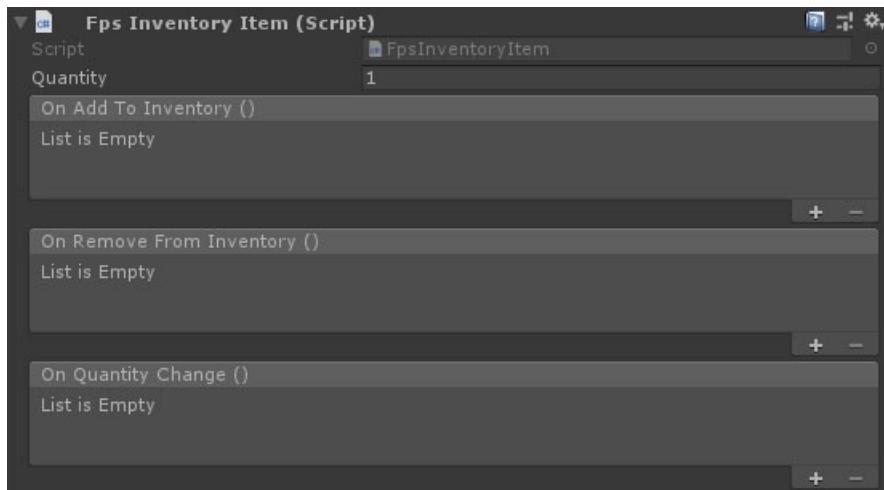
[FpsInventoryWieldableDrop](#)

FpsInventoryItem MonoBehaviour

Overview

The FpsInventoryItem behaviour is an object that can be stored in a character inventory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The item key for this object (from the inventory database).
Quantity	Int	The quantity of items in the stack.
Max Quantity	Int	The maximum amount of this item that the inventory can contain.
On Add To Inventory	UnityEvent	An event that is invoked when the object is first added to the character inventory.
On Remove From Inventory	UnityEvent	An event that is invoked when the object is completely removed from the character inventory.
On Quantity Change	UnityEvent	An event that is invoked when the quantity of objects in the stack changes.

See Also

[Inventories](#)

[Unity Events](#)

FpsInventoryMultiDrop MonoBehaviour

Overview

The FpsInventoryMultiDrop behaviour is a multi-item pickup that is spawned when a character dies. It can contain all of the non-weapon items from a character's inventory such as ammo, armour and keys. The multi-drop will not contain weapons, as picking up the multi-drop could force the character to switch weapons, depending on their inventory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rigidbody	Rigidbody	The object rigidbody for the drop. This will be thrown away from the character that drops it.

See Also

[Inventory](#)

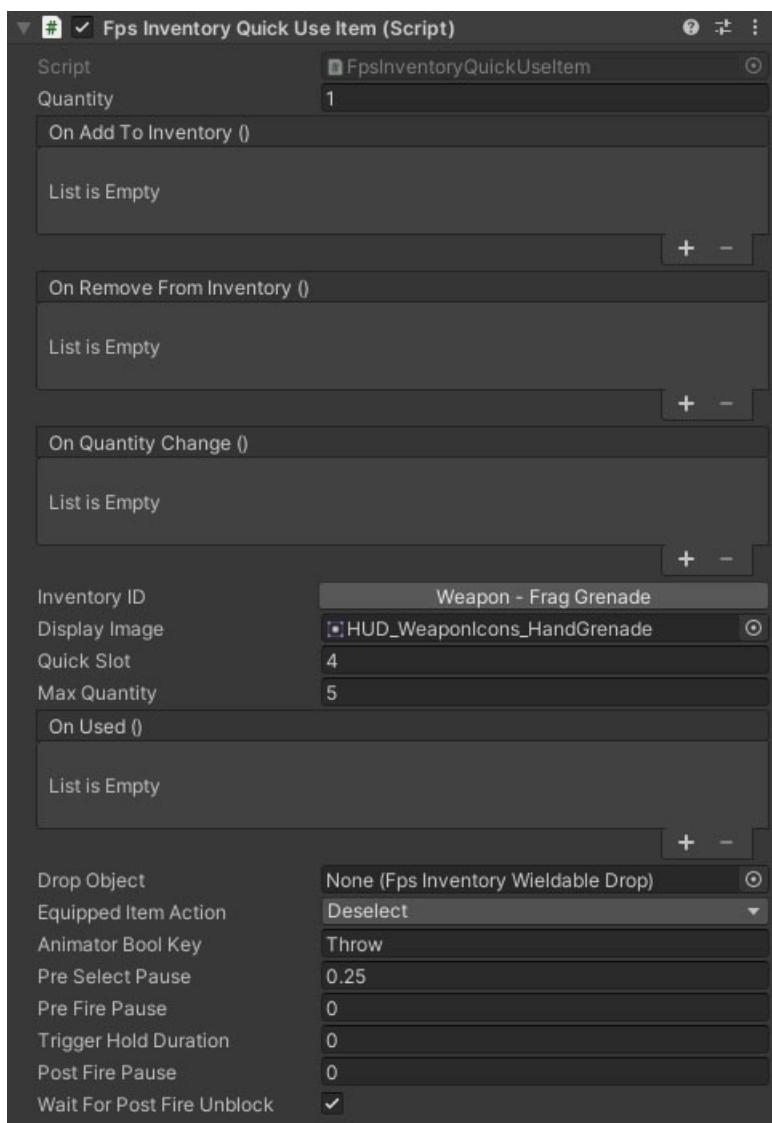
[Unity Rigidbody](#)

FpsInventoryQuickUseItem MonoBehaviour

Overview

The FpsInventoryQuickUseItem behaviour is an inventory item that is used when selected and performs an action based on the attached wieldable. This can be a firearm, melee weapon, thrown weapon or wieldable tool. It is used for behaviours like quick-melee, quick-throw, quick-shoot or quick actions such as applying healing as a wieldable tool.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

Name	Type	Description
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Quick Slot	Int	The quick slot the item should be placed in. If you are using a stacked inventory, remember that each stack is 10 slots (0-9 = stack 1, 10-19 = stack 2, etc).

NAME	TYPE	DESCRIPTION
Max Quantity	Int	The maximum quantity you can hold.
On Used	Unity Event	An event called when the item is used.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.
Equipped Item Action	Dropdown	What to do with the equipped item while performing the quick action. Deselect will deselect it before the action starts, and reselect after. AnimatorBool will block it and set a bool on its animator.
Animator Bool Key	String	The animator bool to set on the previously equipped item. You can use this to do things like lower one hand to use as part of this item.
Pre Select Pause	Float	The amount of time to wait before raising the weapon (this gives time for the currently equipped weapon to deselect).
Pre Fire Pause	Float	The amount of time after raising the weapon before starting the primary fire.
Trigger Hold Duration	Float	The length of time to hold the primary fire trigger for.
Post Fire Pause	Float	The time to wait after the weapon fires or performs its action before deselecting.
Wait For Post Fire Unblock	Boolean	Should the quick use item also wait for the weapon to register that it has completed its action before deselecting.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

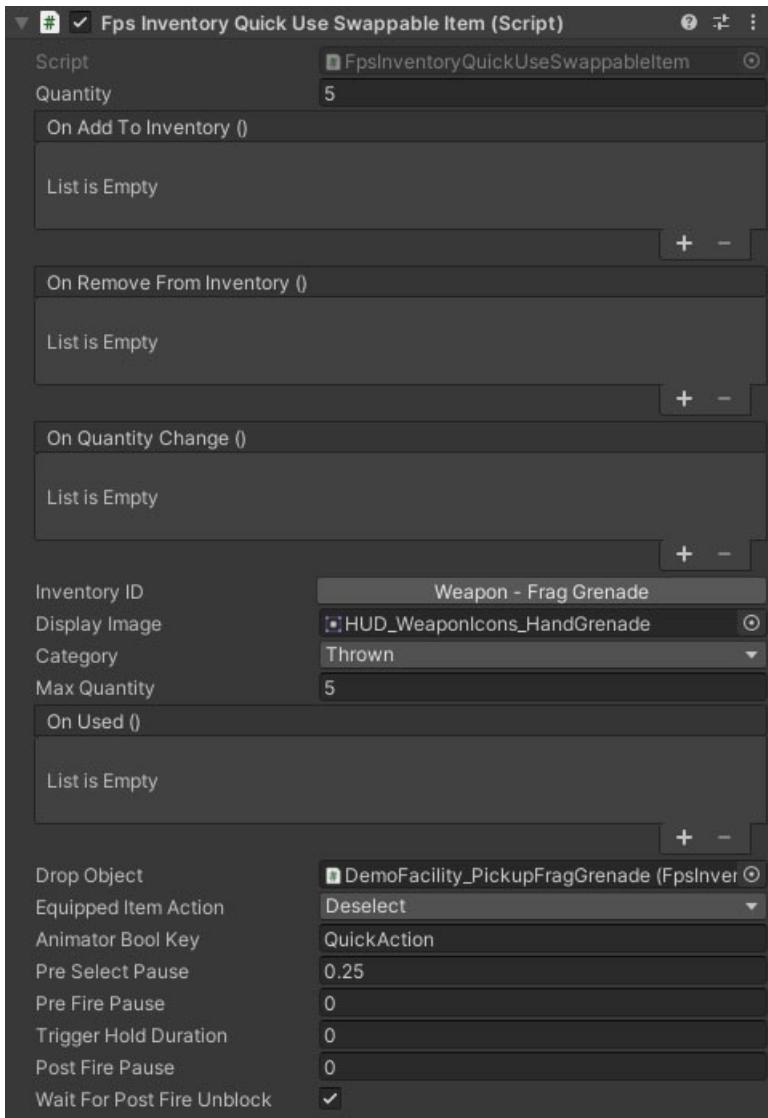
[FpsInventoryWieldableDrop](#)

FpsInventoryQuickUseSwappableItem MonoBehaviour

Overview

The FpsInventoryQuickUseSwappableItem behaviour is an inventory item that is used when selected and performs an action based on the attached wieldable. This can be a firearm, melee weapon, thrown weapon or wieldable tool. It is used for behaviours like quick-melee, quick-throw, quick-shoot or quick actions such as applying healing as a wieldable tool.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Category	FpsSwappableCategory	The wieldable category.

NAME	TYPE	DESCRIPTION
Max Quantity	Int	The maximum quantity you can hold.
On Used	Unity Event	An event called when the item is used.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.
Equipped Item Action	Dropdown	What to do with the equipped item while performing the quick action. Deselect will deselect it before the action starts, and reselect after. AnimatorBool will block it and set a bool on its animator.
Animator Bool Key	String	The animator bool to set on the previously equipped item. You can use this to do things like lower one hand to use as part of this item.
Pre Select Pause	Float	The amount of time to wait before raising the weapon (this gives time for the currently equipped weapon to deselect).
Pre Fire Pause	Float	The amount of time after raising the weapon before starting the primary fire.
Trigger Hold Duration	Float	The length of time to hold the primary fire trigger for.
Post Fire Pause	Float	The time to wait after the weapon fires or performs its action before deselecting.
Wait For Post Fire Unblock	Boolean	Should the quick use item also wait for the weapon to register that it has completed its action before deselecting.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

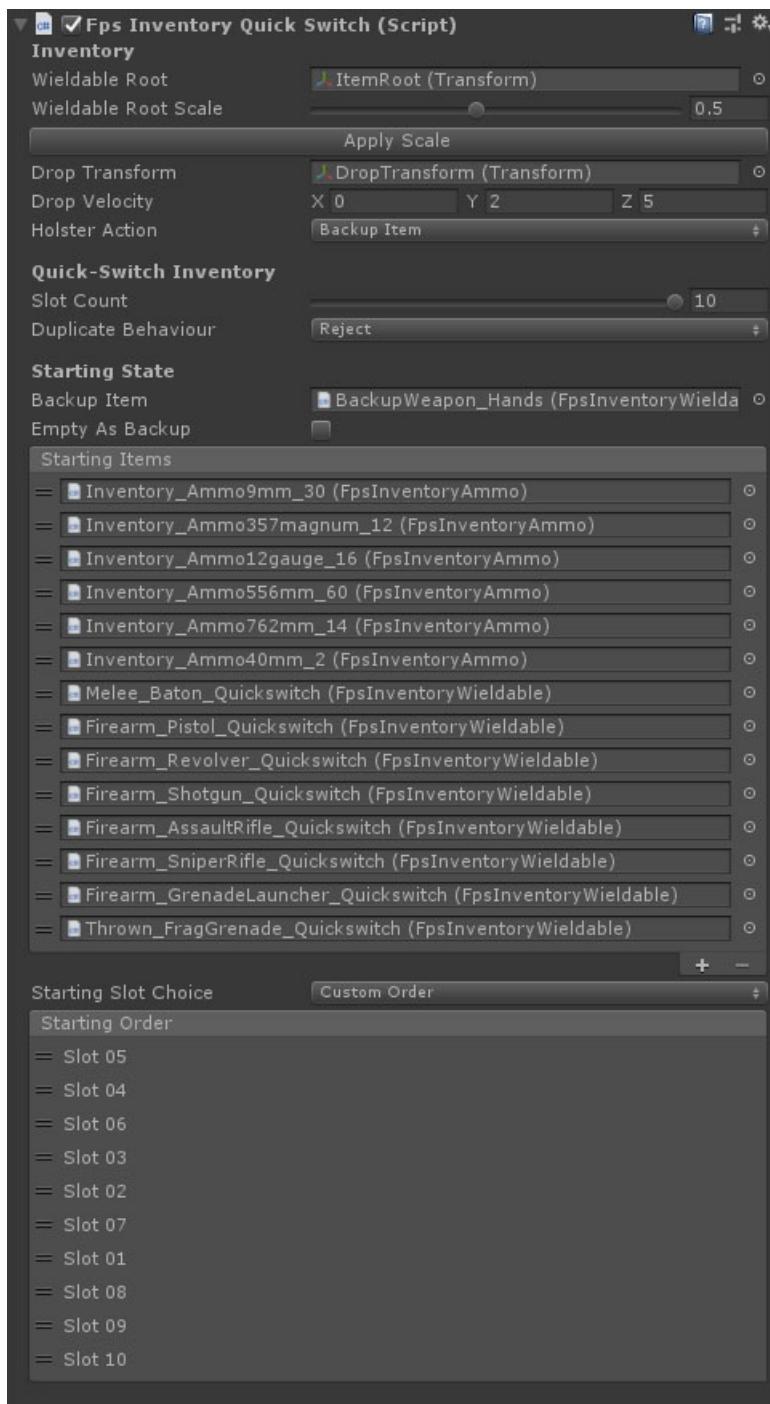
[FpsInventoryWieldableDrop](#)

FpsInventoryQuickSwitch MonoBehaviour

Overview

The quick-switch inventory is a standard PC FPS inventory. Pressing the appropriate number button selects the item in that quick slot. Pressing the *switch weapons* button will switch to the previous selected item.

Inspector



Properties

Name	Type	Description
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.

NAME	TYPE	DESCRIPTION
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are Backup Item , Nothing .
Slot Count	Int	The number of item quick slots.
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are Ascending , Descending , Custom Order .
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	If this is true, then selecting an empty slot will switch to the backup item.
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are Reject , DestroyOld , DropOld .

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

[Inventory Examples](#)

[FpsInventoryItem](#)

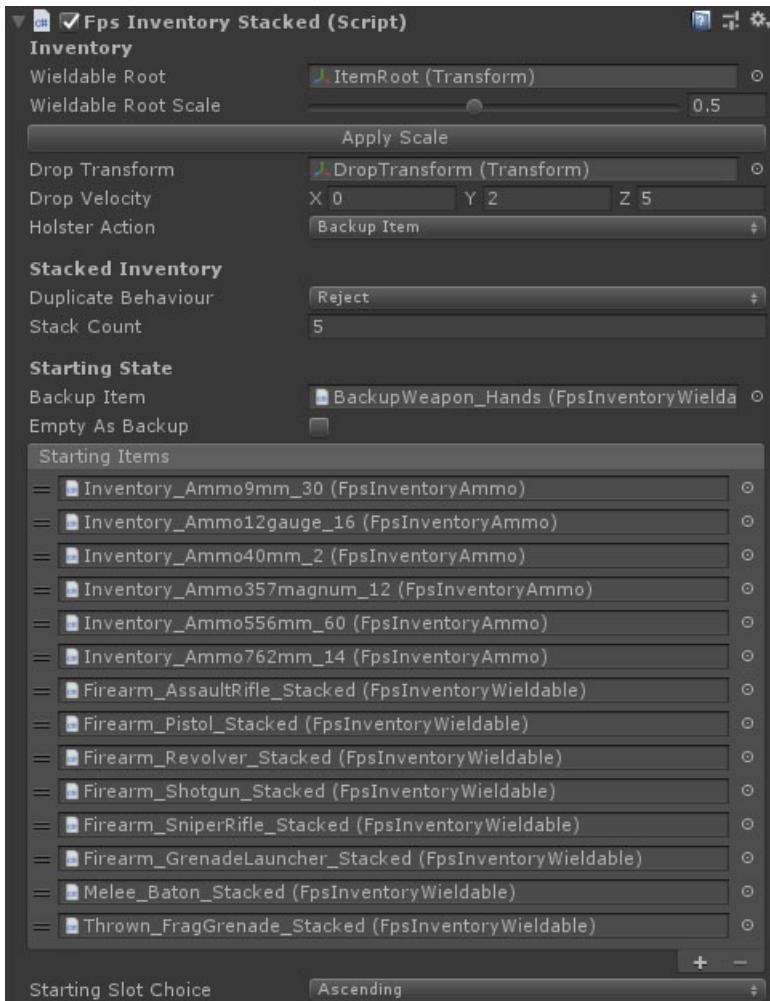
[FpsInventoryWieldable](#)

FpsInventoryStacked MonoBehaviour

Overview

The stacked inventory groups items together into stacks. Selecting a stack multiple times cycles through the items in it. Similar to the inventory system in Half-Life.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are Backup Item , Nothing .
Stack Count	Int	The number of available stacks.

Name	Type	Description
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are Ascending , Descending , Custom Order .
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	<i>This setting does not currently work with the stacked inventory.</i>
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are Reject , DestroyOld , DropOld .

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

[Inventory Examples](#)

[FpsInventoryItem](#)

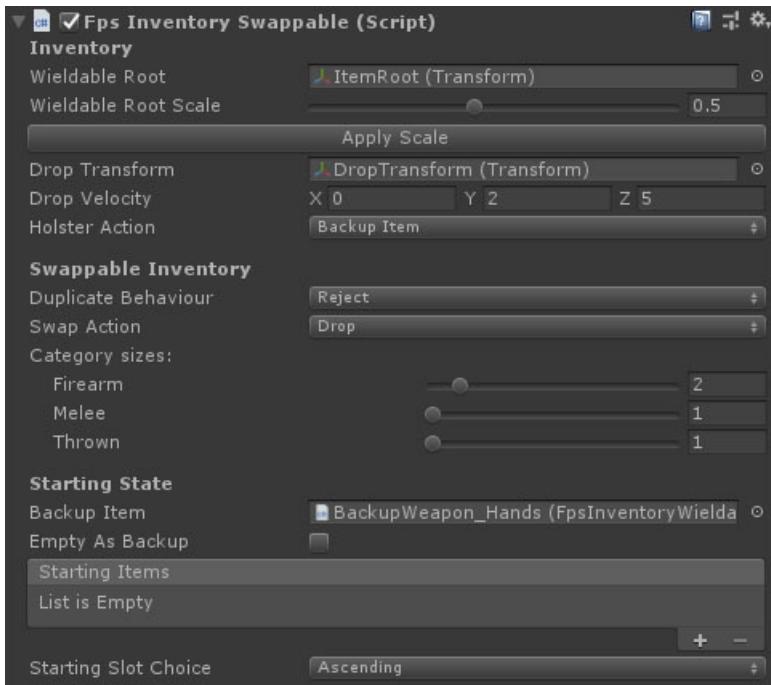
[FpsInventoryWieldable](#)

FpsInventorySwappable MonoBehaviour

Overview

The swappable inventory groups items into categories. If the quick slots for a category are all full then picking up a new item of that category will drop the current or last accessed item of that category.

Inspector



Properties

Name	Type	Description
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Swap Action	Dropdown	What to do when replacing an old item with a new one. Options are Drop , Destroy .
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are Backup Item , Nothing .
Group Sizes	Int	The number of quick slots available for each category. The categories are defined in a generated constant called FpsSwappableCategory . Extending this constant will add new categories in the <code>FpsInventorySwappable</code> editor.

Name	Type	Description
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are Ascending , Descending , Custom Order .
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	If this is true, then selecting an empty slot will switch to the backup item.
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are Reject , DestroyOld , DropOld , Allow . The last option will allow you to pick up multiple weapons of the same type.

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

[Inventory Examples](#)

[FpsInventoryItem](#)

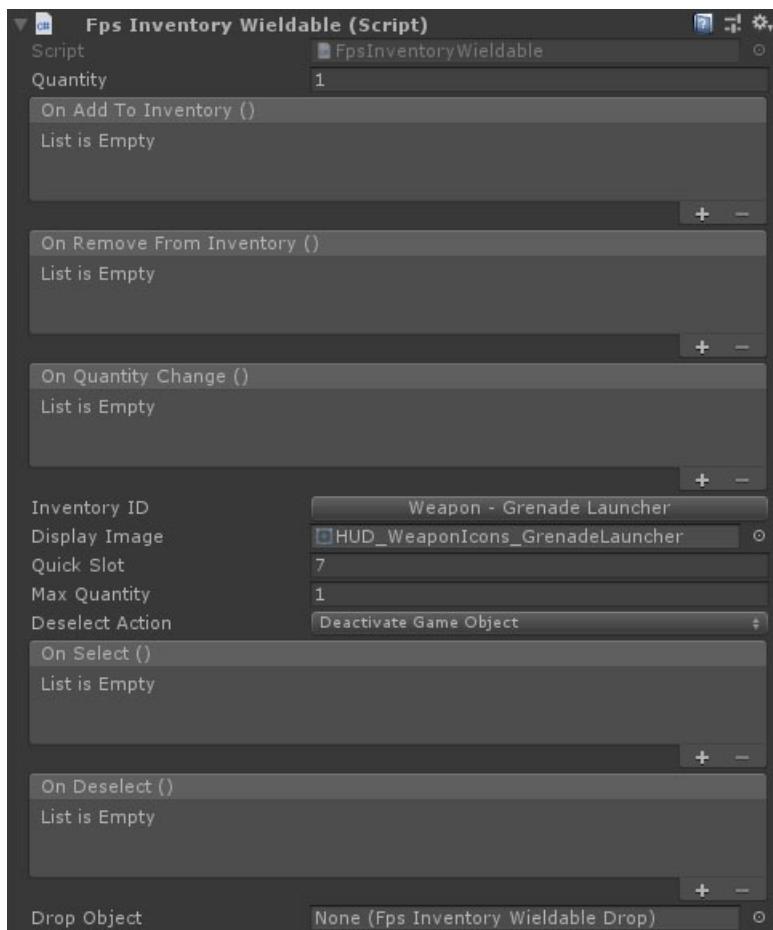
[FpsInventoryWieldable](#)

FpsInventoryWieldable MonoBehaviour

Overview

The FpsInventoryWieldable behaviour is an inventory item that can be held in the hands and used, such as a weapon.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Quick Slot	Int	The quick slot the item should be placed in. If you are using a stacked inventory, remember that each stack is 10 slots (0-9 = stack 1, 10-19 = stack 2, etc).
Max Quantity	Int	The maximum quantity you can hold.
Deselect Action	Dropdown	What to do when the weapon is deselected. Available options are: Disable Game Object , Disable Wieldable Component and Nothing .
On Select	Unity Event	An event called when the wieldable is selected. Use this to enable components, etc.

NAME	TYPE	DESCRIPTION
On Deselect	Unity Event	An event called when the wieldable is deselected. Use this to disable components, etc.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

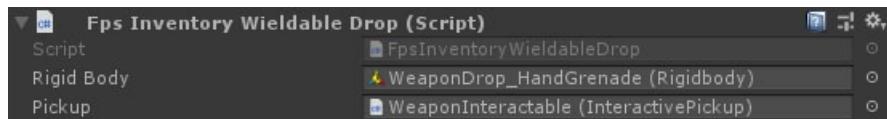
[FpsInventoryWieldableDrop](#)

FpsInventoryWieldableDrop MonoBehaviour

Overview

The FpsInventoryWieldableDrop behaviour is a pickup that is spawned when a character drops a wieldable item.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Rigidbody	Rigidbody	The object rigidbody for the drop. This will be thrown away from the character that drops it.
Pickup	InteractablePickup	The pickup for the item. This will be initialised with the correct quantity based on the dropper's inventory.

See Also

[Inventory](#)

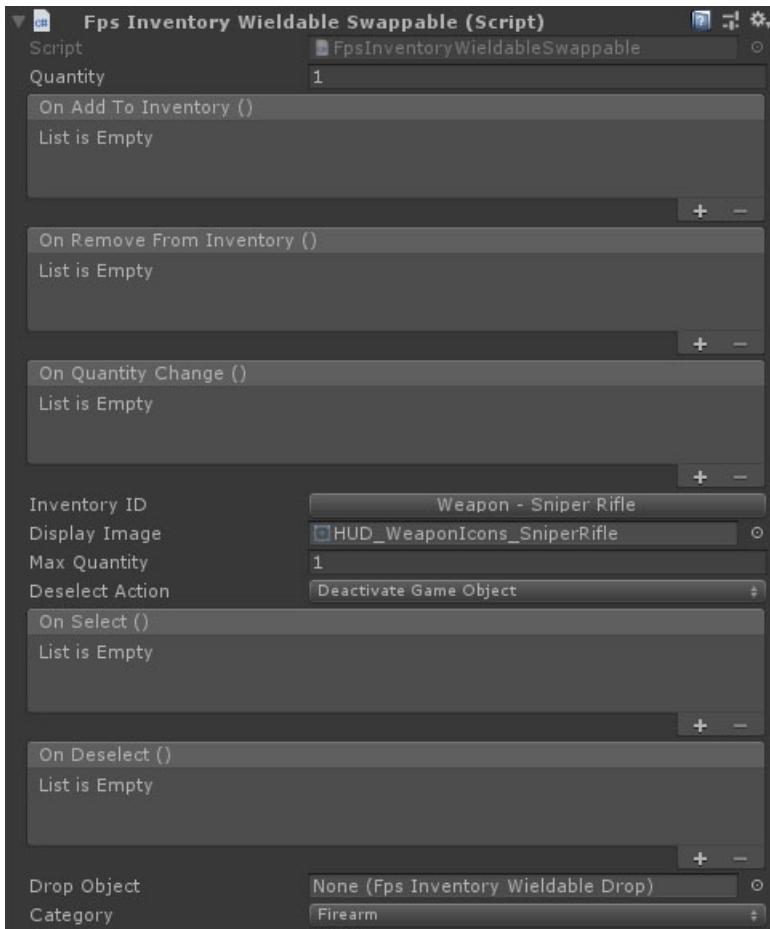
[Unity Rigidbody](#)

FpsInventoryWieldableSwappable MonoBehaviour

Overview

The FpsInventoryWieldableSwappable behaviour is a variant of the [FpsInventoryWieldable](#) for the swappable inventory. It has a property for the wieldable category instead of a specific quick slot.

Inspector



Properties

The FpsInventoryWieldable behaviour inherits from the [FpsInventoryItem](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Max Quantity	Int	The maximum quantity you can hold.
Deselect Action	Dropdown	What to do when the weapon is deselected. Available options are: Disable Game Object , Disable Wieldable Component and Nothing .
On Select	Unity Event	An event called when the wieldable is selected. Use this to enable components, etc.

NAME	TYPE	DESCRIPTION
On Deselect	Unity Event	An event called when the wieldable is deselected. Use this to disable components, etc.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.
Category	FpsSwappableCategory	The wieldable category.

See Also

[Inventory Examples](#)

[Generated Constants](#)

[FpsInventoryItem](#)

[FpsInventoryWieldableDrop](#)

HealPlayerInstantAction MonoBehaviour

Overview

The HealPlayerInstantAction behaviour is attached to an [FpsInventoryInstantuseItem](#) or [FpsInventoryInstantuseSwappableItem](#) and will heal the player for the specified amount when the item is triggered.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Heal Amount	Float	The (max) amount of health to add to the wielder's health manager.

See Also

[Inventory](#)

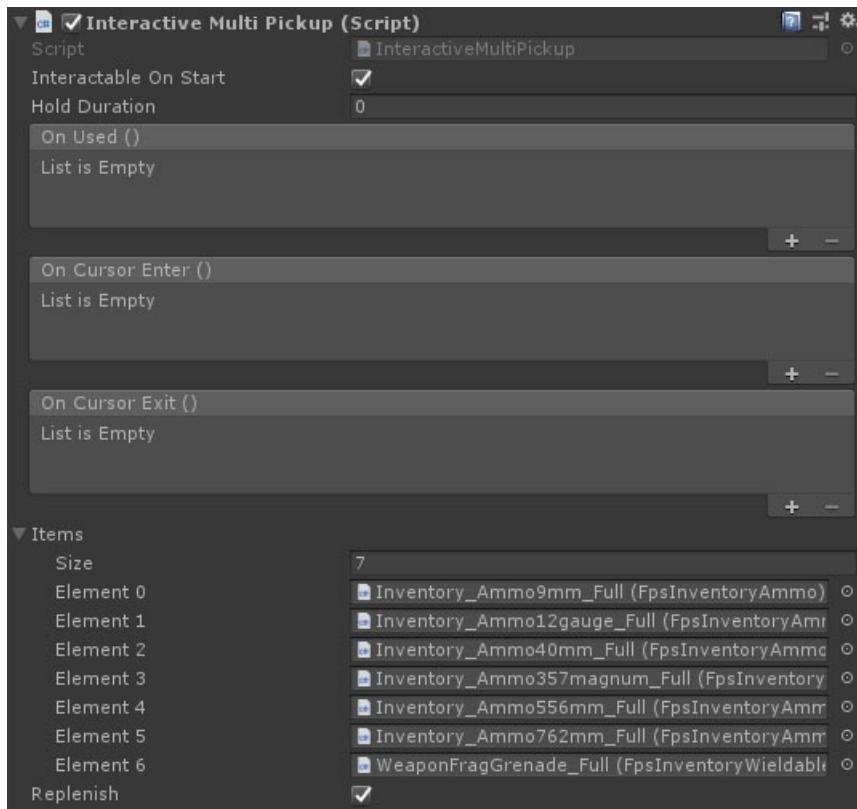
[Health and Damage](#)

InteractiveMultiPickup MonoBehaviour

Overview

The InteractiveMultiPickup behaviour is an object that will give a number of inventory items to the character that interacts with it.

Inspector



Properties

The InteractiveMultiPickup inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Items	FpsInventoryItem Array	The item prefabs to add to the character inventory.
Replenish	Boolean	Do the inventory items replenish. If the items are removed new items will be instantiated. If they are partially removed (to top up an existing item) the quantity will be reset afterwards.

See Also

[InteractiveObject](#)

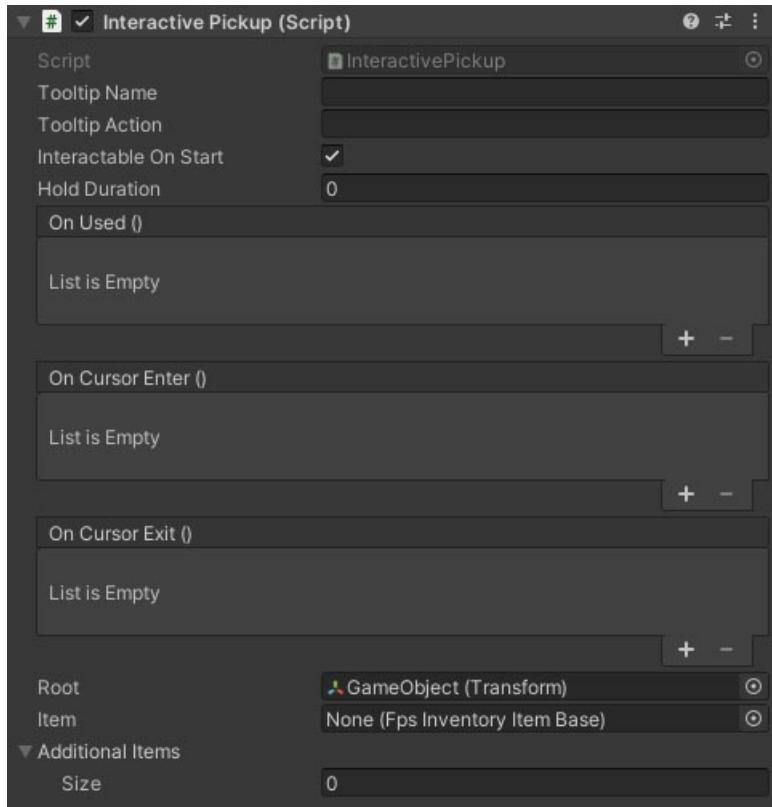
[FpsInventoryItem](#)

InteractivePickup MonoBehaviour

Overview

The InteractivePickup behaviour is an object that must be interacted with to pick up and add an item to the character inventory.

Inspector



Properties

The InteractiveMultiPickup inherits from the [InteractiveObject](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Root	Transform	The root object (destroyed when the item is picked up).
Item	FpsInventoryItem	The item prefab to add to the character inventory.
Additional Items	FpsInventoryItem Array	Optional items that will only be picked up if the main item is. These do not affect whether the pickup is destroyed or deactivated once the main item is.

See Also

[InteractiveObject](#)

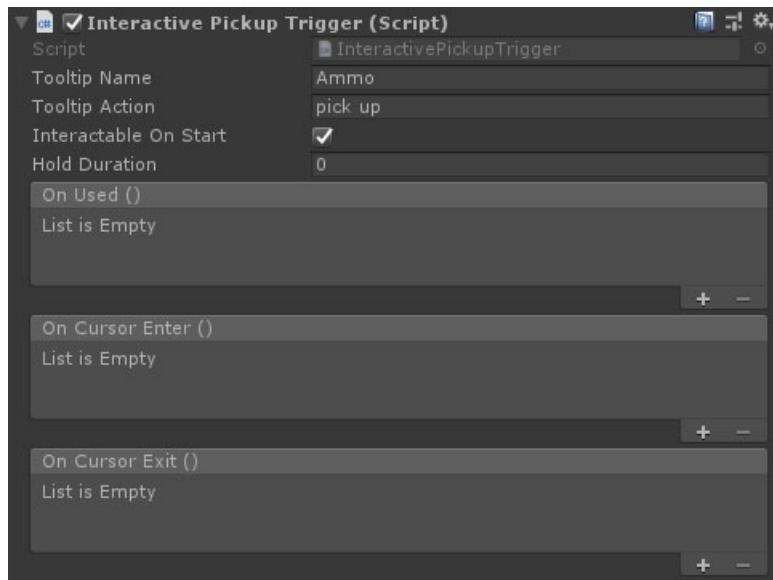
[FpsInventoryItem](#)

InteractivePickupTrigger MonoBehaviour

Overview

The InteractivePickupTrigger behaviour is used to create interactive item pickups or powerups that must be used to consume.

Inspector



Properties

The InteractivePickupTrigger behaviour has no properties exposed in the inspector.

See Also

[Inventory](#)

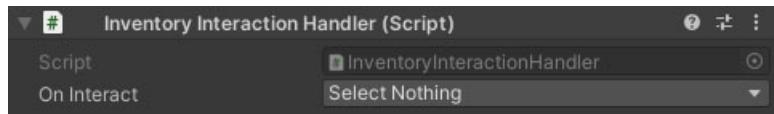
[InteractiveObject](#)

InventoryInteractionHandler MonoBehaviour

Overview

The InventoryInteractionHandler behaviour controls what the player character does with their weapon selection while interacting with an object in the scene.

Inspector



Properties

NAME	TITLE	DESCRIPTION
On Interact	Dropdown	What to do when the character interacts with an object. Options are SelectNothing and SelectHands .

See Also

[InteractiveObject](#)

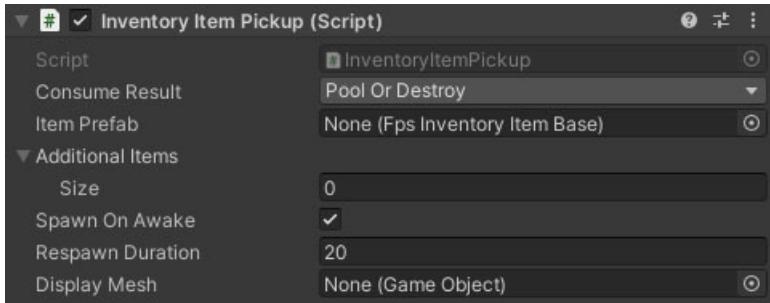
[FpsInventoryItem](#)

InventoryItemPickup MonoBehaviour

Overview

The InventoryItemPickup behaviour is combined with a [PickupTriggerZone](#) to create a pickup that is added to the character inventory when walked over.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Consume Result	Dropdown	What to do to the pickup object once its item has been transferred to the character inventory. Options are Destroy , Disable , Respawn , DisableOnPartial , RespawnOnPartial , Infinite . The partial options will treat any amount consumed as consuming the full object, while Infinite will never consume the item.
Item Prefab	FpsInventoryItem	The inventory item prefab to give to the character.
Additional Items	FpsInventoryItem Array	Optional items that will only be picked up if the main item is. These do not affect whether the pickup is destroyed or deactivated once the main item is.
Spawn On Awake	Boolean	Should the pickup be spawned immediately, or triggered externally.
Respawn Duration	Float	How long to wait before respawning if the consume result is set to Respawn or RespawnOnPartial .
Display Mesh	GameObject	The display mesh of the pickup. This should not be the same game object as this, so that if this is disabled the pickup will still respawn if required.

See Also

[InteractiveObject](#)

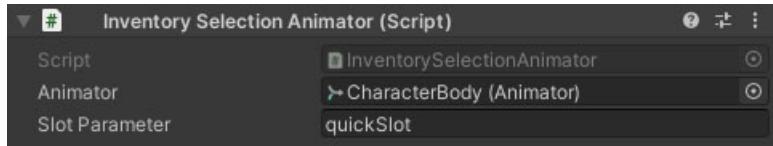
[FpsInventoryItem](#)

InventorySelectionAnimator MonoBehaviour

Overview

The InventorySelectionAnimator behaviour is attached to the character object and sets an integer parameter on the character's [animator controller](#) whenever the character changes its weapon selection.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The character's animator.
Slot Parameter	String	The name of an integer parameter in the character's animator controller that should be set with the quick slot number on selection changed.

See Also

[Inventory Examples](#)

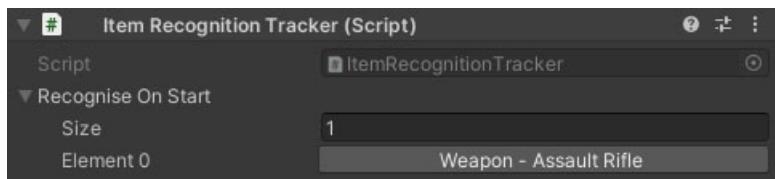
[FpsInventoryItem](#)

ItemRecognitionTracker MonoBehaviour

Overview

The ItemRecognitionTracker behaviour is attached to the player **controller** (so that it persists between respawns) and is used to track inventory items by ID that the character has checked before. This can be used alongside an [InspectOnFirstUse](#) behaviour attached to each of your weapons to show an inspect animation when the weapon is first picked up.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Recognise On Start	ID Array	The inventory items that the player should already be aware of on starting the game.

See Also

[InteractiveObject](#)

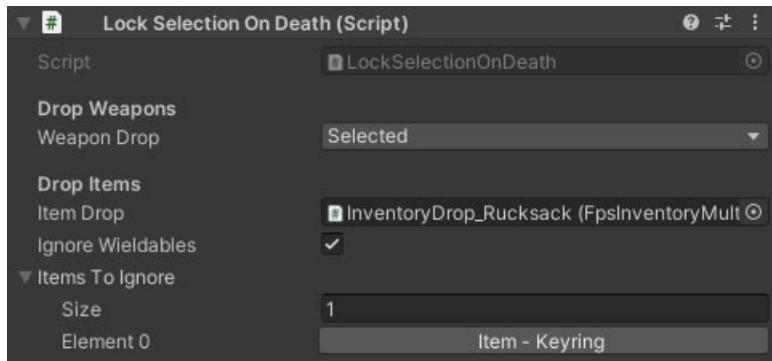
[FpsInventoryItem](#)

LockSelectionOnDeath MonoBehaviour

Overview

The LockSelectionOnDeath behaviour is added to a character, and when that character dies it deselects their current weapon (optionally dropping it on the ground) and locks their selection to nothing.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Weapon Drop	Dropdown	Should the selected weapon be dropped on death. Options are Selected drops the selected weapon, All drops all weapons, None doesn't drop any weapons.
Item Drop	FpsInventoryMultiDrop	A multi-drop object that will be spawned containing the non-wieldable contents of the character's inventory.
Ignore Wieldables	Boolean	Should wieldable items be ignored by the drop object? Since picking up some types of wieldables can cause you to replace the one you hold, you might not want to add them to the drops.
Items To Ignore	Inventory Key Array	Specific items to ignore.

See Also

[InteractiveObject](#)

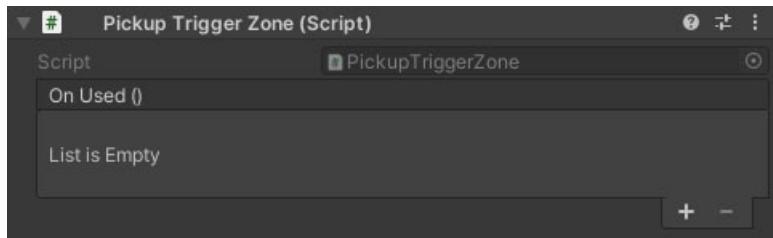
[FpsInventoryItem](#)

PickupTriggerZone MonoBehaviour

Overview

The PickupTriggerZone behaviour is added to objects that also contain a [pickup behaviour](#), and a collider set to act as a trigger. It captures trigger events from the collider and uses them to activate the pickup.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Used	UnityEvent	An event that is triggered when the zone is entered is used.

See Also

[Inventory](#)

[Unity BoxCollider](#)

[Unity SphereCollider](#)

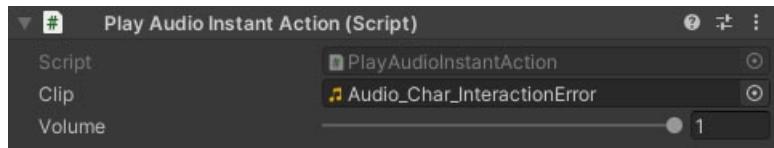
[Unity CapsuleCollider](#)

PlayAudioInstantAction MonoBehaviour

Overview

The PlayAudioInstantAction behaviour is attached to an [FpsInventoryInstantuseItem](#) or [FpsInventoryInstantuseSwappableItem](#) and will play an audio clip when triggered.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Clip	AudioClip	The clip to play.
Volume	Float	The volume for the clip.

See Also

[Inventory](#)

RechargeShieldsInstantAction MonoBehaviour

Overview

The RechargeShieldsInstantAction behaviour is attached to an [FpsInventoryInstantuseItem](#) or [FpsInventoryInstantuseSwappableItem](#) and will restore one or more shield bars on the player character when triggered.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Step Count	Integer	The number of shield steps to recharge.

See Also

[Inventory](#)

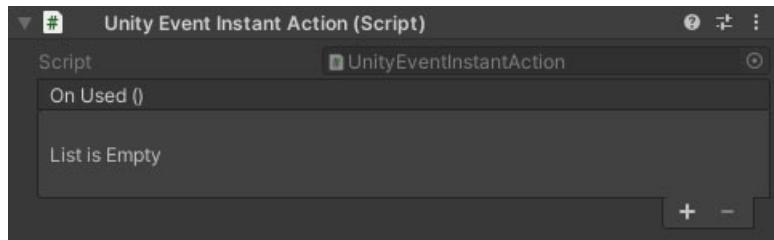
[Shield Manager](#)

UnityEventInstantAction MonoBehaviour

Overview

The UnityEventInstantAction behaviour is attached to an [FpsInventoryInstantuseItem](#) or [FpsInventoryInstantuseSwappableItem](#) and will fire a [Unity event](#) that you can hook into via the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Used	Unity event	The event to fire on using the instant use item.

See Also

[Inventory](#)

WieldableItemBodyAnimOverrides MonoBehaviour

Overview

The WieldableItemBodyAnimOverrides behaviour is added to wieldable items, such as weapons, and overrides the animations on the wielding character's body when the item is selected. The animations to override must be exposed using a [FirstPersonCharacterAnimationProfile](#) asset.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Character Profile	FirstPersonCharacterAnimationProfile	The animation profile for the character body. This is a specific set of animations that have been exposed to override.

Below the character profile property will be a list of [Animation Clip](#) properties with names matching the animations in the profile. A different profile will likely have different animation names.

See Also

[Inventory](#)

[Animator Controllers](#)

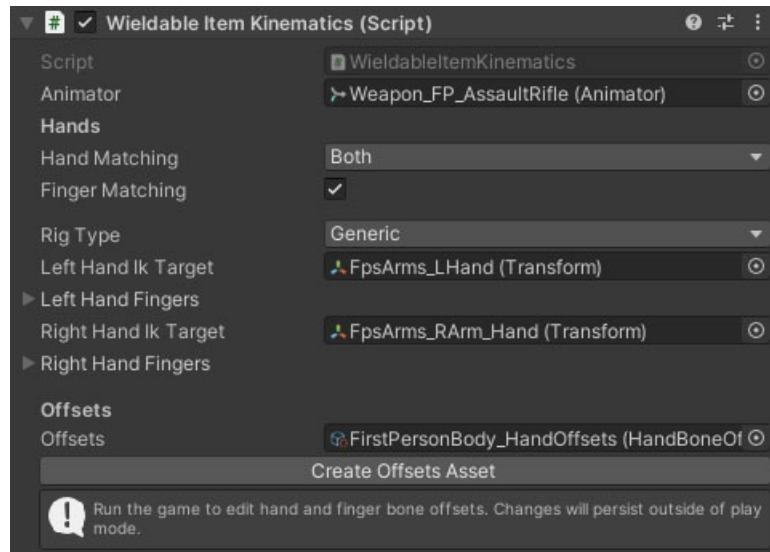
[FirstPersonCharacterAnimationProfile](#)

WieldableItemKinematics MonoBehaviour

Overview

The WieldableItemKinematics behaviour is used to specify hand and finger properties on a weapon or wieldable item that the character's [FirstPersonCharacterArms](#) should IK match character hands and fingers to. It can be used with either generic or humanoid rigs on the wieldable, but the character must use a humanoid rig for IK to work.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The wieldable's animator component.
Hand Matching	Dropdown	Which hand positions and rotations should be matched (using IK). Options are None , Left , Right , Both .
Finger Matching	Boolean	Should finger rotations be matched.
Rig Type	Dropdown	What type of animation rig does the weapon use. Options are Generic and Humanoid . A generic rig requires you to specify which objects represent the hand and finger bones.
Left Hand Ik Target	Transform	The target transform the character's left hand should align to.
(Left Hand Fingers)	Transform Array	A list of finger transforms for the left hand.
Right Hand Ik Target	Dropdown	The target transform the character's right hand should align to.
(Right Hand Fingers)	Transform Array	A list of finger transforms for the right hand.
Offsets	HandBoneOffsets	An offsets asset to help align character hands and fingers to weapon targets. Offsets can be edited below when in play mode and changes will persist to edit mode.

The **Create Offsets** button will create a [HandBoneOffsets](#) at the location of the weapon prefab using its name as a basis. It will then assign the offsets asset to the **Offsets** property above.

If you have a weapon selected in the player character hierarchy at runtime then the inspector will embed the properties of the attached [HandBoneOffsets](#) asset for you to edit in place. Changing its properties will change the hand position and rotation, and finger rotations in the scene view, and changes will persist when exiting play mode. You can also modify the properties on the asset directly to see the changes in the scene view. An offset asset can be shared between multiple weapons if they all use the same hand rig, or created per weapon if not.

See Also

[First Person Body](#)

[Inventory](#)

[HandBoneOffsets](#)

[FirstPersonCharacterArms](#)

FpsInventoryDbTable Scriptable Object

Overview

The FpsInventoryDbTable scriptable object contains a list of inventory items and manages their IDs. You can add new items to the inventory database from this object, or directly from the inventory item components themselves.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Table Name	String	A unique name for this table, to help differentiate them in the database and key picker.

The entries table shows each of the current entries in this table. You can click on the options button to the right of each entry for a list of context sensitive options, or you can click on the name to rename the item. Each item uses a unique ID code, so renaming items will not break connections.

You can sort the table using the **Sort By Name (Ascending)** and **Sort By Name (Descending)** buttons. You can also quickly clear all items with the **Clear All Items** button. Any objects that reference that item will show their ID as "Missing: (ID code)".

You can add new items by entering a name in the item name field and hitting the *Add Entry* button.

See Also

Inventory

[NeoFpsInventoryDatabase Scriptable Object](#)

FpsInventoryKeyDbTable Scriptable Object

Overview

The FpsInventoryKeyDbTable scriptable object is an inventory database table that takes its keys and names directly from the `FpsInventoryKey` constant. This allows the relevant keys to be accessed directly from code (useful for things like keyrings).

Inspector

The FpsInventoryKeyDbTable behaviour is only visible through the [NeoFpsInventoryDatabase](#) inspector.

Properties

The FpsInventoryKeyDbTable behaviour has no properties exposed in the inspector.

See Also

[Inventory](#)

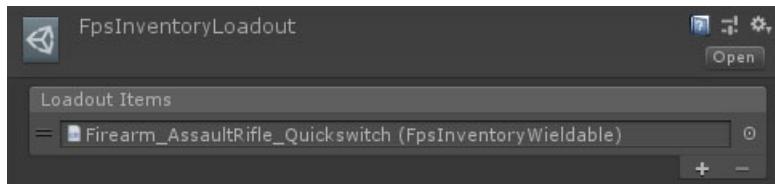
[NeoFpsInventoryDatabase Scriptable Object](#)

FpsInventoryLoadout Scriptable Object

Overview

The FpsInventoryLoadout scriptable object is used to replace a character's starting inventory on spawn. These are assigned to the loadout property on the [FpsSoloGameMinimal](#) component.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Items	FpsInventoryItem Array	A selection of inventory items to be added to the character inventory on start.

See Also

[Inventory Examples](#)

[FpsInventoryItem](#)

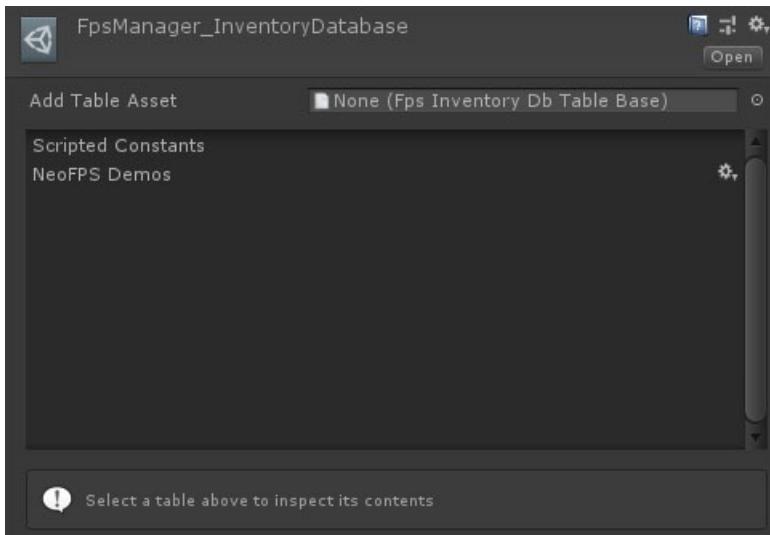
[FpsInventoryWieldable](#)

NeoFpsInventoryDatabase Scriptable Object

Overview

The NeoFpsInventoryDatabase scriptable object is a global inventory database that can be accessed from scripts at runtime and in editor code. It contains a list of database tables, allowing you to divide items up for projects and demos. The first table is always the [FpsInventoryKeyDbTable](#). The other table type that is provided is the [FpsInventoryDbTable](#), though more can be added by inheriting from the common base class.

Inspector



Properties

You can add new tables to the database by dragging and dropping it into the **Add Table Asset** field at the top of the database inspector.

Underneath this is a list of the current tables in use by the database. The options button at the side of each table will show a list of context options. Selecting a table will show its inspector below the list of tables.

See Also

[Inventory](#)

[FpsInventoryKeyDbTable](#)

Weapons

Overview

NeoFPS currently supports three types of weapons: melee, thrown and modular firearms.

Weapon Types

Melee Weapons

Melee weapons are weapons such as clubs and swords that are held onto and swung at the enemy. For more information see [Melee Weapons](#).

Thrown Weapons

Thrown weapons are items such as grenades and ninja stars that are stored in the character inventory and thrown by hand. For more information see [Thrown Weapons](#).

Modular Firearms

Modular firearms are gun type weapons. They are assembled from a number of swappable modules that, together, model the behaviour of a firearm. For more information see [Modular Firearms](#).

Wieldable Tools

Wieldable tools are items that are carried in the player character's hands and used with the primary or secondary fire buttons. They are assembled from a number of tool modules and actions. For more information see [Wieldable Tools](#).

Explosions

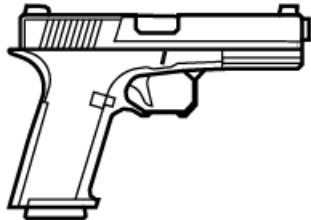
The NeoFPS weapons can make use of a simple explosions system which deals damage and adds a repelling force to any objects in an explosion's blast radius. For more information see [Explosions](#).

Demo Weapons

The following weapons are included in the samples in order to show off the different weapon features:



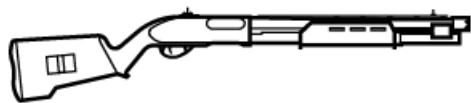
The baton is a simple demonstration of the melee weapons.



The pistol is a low caliber firearm with plenty of ammunition. It uses the basic features of the modular firearms.



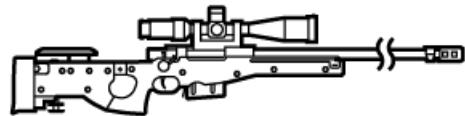
The revolver is a high caliber, heavy hitting firearm. The trigger pull is quite heavy so firing can have a delay.



The shotgun is a pump action weapon with a conical spread. Shells are loaded into the weapon individually and the process can be aborted after the current round is inserted by pressing the fire button.



The assault rifle is a 5.56mm carbine with 3 fire modes: full auto, 3 round burst and semi-automatic.



The sniper rifle is a heavy caliber bolt-action rifle with a long range scope.



The grenade launcher is a single-shot break open launcher that fires 40mm high explosive grenades.



The fragmentation grenade is a thrown weapon. Pull the pin and wait 5 seconds for an explosion.

More demo weapons will be added in future updates.

Dual Wield Weapons

A dual wield weapon is a single prefab with 2 weapons as children. Each of the 2 weapons can be a [modular firearm](#), a [wieldable tool](#), a [melee weapon](#) or a [thrown weapon](#). The root of the dual wield weapon has the inventory wieldable component, along with a dual wield input handler (either [InputDualWield](#) or [InputSystemDualWield](#)) and a [CompoundCrosshairDriver](#). The latter tracks the accuracy of the 2 sub-weapons and uses those to control the spread of the combined crosshair.

When assembling a dual wield weapon, the 2 sub weapons should look very similar to a regular weapon. That means there is an object with the weapon behaviours and modules. As a child of that there is a weapon spring object to handle the procedural animation. As a child of that there is then the view model. Each of the dual wield weapons should have these. If you want to instead use a single view model then you should be looking at the firearm with secondary weapon mentioned below.

Dual wield weapons can be set up to act in one of 2 ways. The first is that one weapon is triggered by the primary fire button. The other is triggered by the secondary fire button. On PC the secondary fire and aim buttons are shared, so this means you can't aim down sights on a weapon set up this way. The other style of dual wield is for both weapon to be triggered simultaneously with the primary fire button, and then you will still be able to aim down sights.

Aiming with a dual wield weapon can restrict the valid aimer modules. For example, the head move aimer and camera constraints aimer will both cause problems as the 2 weapons clash. It is best to stick with the weapon move aimer, and set the zoom FoV multiplier to the same value on both weapons for predictable results.

A number of example dual wield weapons are provided in the folder:

NeoFPS\Samples\Shared\Prefabs\Weapons\SwappableInventory

Firearms With Secondary Weapons

In a similar way to the dual wielding, you can also add a secondary weapon to a firearm. This secondary weapon can be a [modular firearm](#), a [wieldable tool](#), a [melee weapon](#) or a [thrown weapon](#). Both of these weapons should share a view model, however this can add some quirks to the setup. An example would be that the firearm modules attach to the first firearm in their parent hierarchy. Therefore, it is best to place the secondary weapon behaviour and any modules on a child object of the root firearm, but **alongside** the weapon spring for the view model.

In order to make the second weapon a secondary fire mode for the firearm, you will need to use the correct input handler on the weapon's root (either [InputFirearmWithSecondary](#) or [InputSystemFirearmWithSecondary](#)). In the case of a secondary firearm, it is also a good idea to add a [CompoundCrosshairDriver](#) behaviour to the root object and move it above the ModularFirearm in the inspector. This will ensure that the HUD crosshair expands and shrinks based on both firearms' accuracy.

In the case of a secondary firearm, the ModularFirearm behaviour will see that it has no children and show the quick setup mode in the inspector. Assigning an animator controller to the firearm's animator controller property at the top of the quick setup will tell the firearm that it has an external view model, and the inspector will switch to the standard modules mode. A number of the modules use properties that expect to point at child objects (eg the shooter modules expect a muzzle tip transform to be in their child hierarchy). When you assign those properties an object/behaviour reference from outside their hierarchy they will show up as orange in the inspector as a warning. This is purely visual and is meant to aid in troubleshooting your setups.

Weapon-Camera Alignment

One of the most commonly asked support questions that people have is why their weapon is not aligned correctly to the camera.

The key thing to bear in mind with the NeoFPS weapons is that they will all be instantiated and stored under a single transform in the character hierarchy. For most characters this transform will be in the same position as the camera. This means that you should treat the root of the weapon hierarchy as the camera position.

The exception to this is when you have a full character body with arms that use IK to match the weapons. In this situation the item

root that acts as parent to all the weapons should be attached to the upper chest bone (it can be raised to shoulder level or higher). If you do not do this then pitching the camera too far up or down will cause the weapon to swing away from the body and out of reach of the hands. For this situation it is usually best to treat the weapon root as the camera position for the initial setup and then offset the view-model to bring it into view at runtime and re-apply the offset once you re-enter edit mode.

In order to offset the view model to align it with the root of the character you need to move the view-model object itself. The root transform, pose transform (if it's not set to the root) and "WeaponSpring" transform will all have their position and rotation overwritten by the procedural animation systems. The view model usually sits under that "WeaponSpring" object.

Depending on how the weapon view-model animations are set up, this object might also have any offset you apply overwritten at runtime. If the root of the view-model has keyframes on its position or rotation then these will be applied as soon as the animation plays. Sometimes you can get around this simply by checking on "Root Motion" on the weapon [Animator component](#). If the root object actually moves though, then it can be easier to add a new object in between the "Weapon Spring" and the view-model, and then offset that instead.

If your weapon is set up with an object in its view-model hierarchy that represents the camera position then there is a quick solution to aligning the weapon. On the weapon's main component inspector ([MeleeWeapon](#), [\[ModularFirearm\]\[ref2\]](#), [\[ThrownWeapon\]\[ref3\]](#), [\[WieldableTool\]\[ref4\]](#)) you will find a section called **Origin Point** with a property **Match Transform**. If you drag the camera object onto that field, then the view model will be realigned so that object is aligned with the weapon root. The value is not stored for later. It is used once to realign the objects and then forgotten.

See Also

[Melee Weapons](#)

[Thrown Weapons](#)

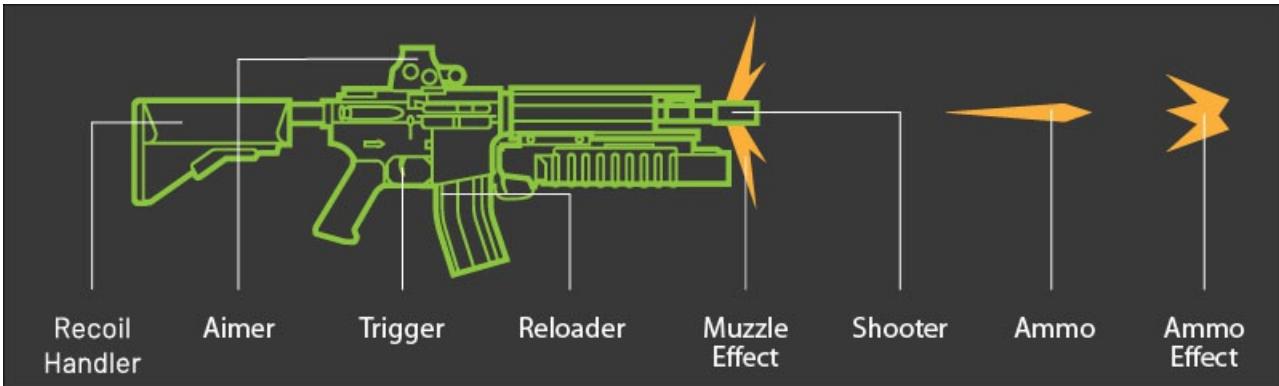
[Modular Firearms](#)

[Wieldable Tools](#)

[Explosions](#)

Modular Firearms

Overview



Gun style weapons in NeoFPS use a system of modules that work together with a central [ModularFirearm behaviour](#) to model the behaviour of a firearm. Multiple modules of each type can exist on a firearm at any time, but only one of each can be active. When another is enabled it automatically disables the previous. Modules can be attached to a child gameobject of the firearm, or on the same gameobject. This allows for weapon behaviour to be built on attachments that can be swapped in and out at runtime.

Alongside the information below, we also have a youtube playlist that deep dives into the modular firearm systems:

Creating a Modular Firearm

There are 2 ways to create a new modular firearm. The quickest and easiest is to use the Modular Firearm Wizard in the NeoFPS Hub. You can find out more about this on the [NeoFPS Wizards](#) page. The more flexible approach is to use the Modular Firearm component itself.

You can create a new modular firearm using the [ModularFirearm](#) behaviour's quick-setup controls in the inspector. Create a new GameObject in the scene and add the [ModularFirearm](#) behaviour to it. To start off with, the behaviour will be in quick-setup mode. This allows you to select a model to use, along with a few basic choices on input and inventory type. Once you have completed quick-setup you will see a "modules" section, which allows you to pick from the available firearm modules as listed below. Any errors in the setup of the modules will be shown in this section, and the module will be highlighted. For more information, see the [ModularFirearm](#) behaviour reference.

Wieldables vs Quick-Fire

The demo firearms are all set up as wieldable. If a firearm is wieldable, that means it is equipped when selected, and then you have to hit the primary fire input to shoot. You can achieve this with your firearms by adding an [FpsInventoryWieldable](#) or [FpsInventoryWieldableSwappable](#) behaviour to the root of your weapon prefab depending on your choice of inventory. These handle selection and deselection, and also define which quick-slot they apply to.

Alternatively, you can set up your firearms as quick-use items. This means that when you select them, the weapon will raise and then immediately shoot before lowering again. This can be used alongside another equipped weapon (optionally interrupting its use). To achieve this, you would add an [FpsInventoryQuickUseItem](#) or [FpsInventoryQuickUseSwappableItem](#) behaviour to the root of your prefab instead of the wieldable behaviours above.

Modules

The modules that work together to model the firearm are as follows:

Shooters

Shooter modules are responsible for the actual gunshot. This could mean checking a raycast or spawning a projectile. The shooter is provided with a source position and direction, an accuracy value (0 to 1) and an ammo effect. NeoFPS comes with the following examples:

NAME	DESCRIPTION
HitscanShooter	The hitscan shooter uses a physics raycast to detect a hit.
BallisticShooter	The ballistic shooter spawns a BallisticProjectile object and passes the ammo effect to it to handle any impacts.
SimpleBallisticShooter	The simple ballistic shooter spawns a BallisticProjectile just like the ballistic shooter, but isn't affected by accuracy or the camera aim.
SpreadHitscanShooter	The spread shooter acts as a cone of hitscan shooters. It is used for weapons like shotguns.
SpreadBallisticShooter	The ballistic shooter spawns multiple BallisticProjectile objects in a cone and passes the ammo effect to them to handle any impacts.
PatternHitscanShooter	The pattern hitscan shooter fires multiple hitscan shots in a preset pattern.
PatternBallisticShooter	The pattern ballistic shooter spawns multiple BallisticProjectile objects in a preset pattern.

The modular firearm shooters allow the choice of either hitscan or projectile versions. Hitscan shooters cast an instant ray from the gun barrel. This helps them feel very responsive, especially at close quarters. Projectile shooters spawn a projectile at the weapon's muzzle tip. These projectiles have full control of their movement and hit detection, and then pass the hit info back to the relevant ammo effect. Some example projectiles include the [BallisticProjectile](#), [BallisticProjectileWithSimpleDrag](#) and [BallisticProjectileWithParticles](#).

Triggers

Trigger modules handle the rate of fire of the firearm and react to input. NeoFPS comes with the following examples:

NAME	DESCRIPTION
AutomaticTrigger	The automatic trigger models a machinegun with a set rate of fire. It will keep firing as long as the trigger is held.
SemiAutoTrigger	The semi-auto trigger fires one shot for each press of the trigger. It can optionally repeat fire after a set time.
BurstFireTrigger	The burst fire trigger fires a set number of bullets at a set rate of fire. It has options to repeat after a set time and to allow / disallow cancelling the burst.
ChargedTrigger	The charged trigger must charge up before firing. If the trigger input is released the charge drops. Once the trigger has been charging for a set time the gun will fire.

NAME	DESCRIPTION
ContinuousTrigger	The continuous trigger fires every frame and is used for weapons like flamethrowers or beam weapons.
QueuedTrigger	The queued trigger will queue up shots up to a maximum count while the trigger is held, and fire once it's released. You can optional cancel with the reload button.
TargetLockTrigger	The target lock trigger will lock onto any valid targets within the firearm's detection cone. Holding the trigger will build the lock. Releasing the trigger at full lock will fire. Releasing early or hitting reload will cancel the lock.
MultiTargetLockTrigger	The multi-target lock trigger is essentially a queued trigger that locks onto a unique target for each shot.

Ammo Effects

Ammo effect modules specify what happens when a shot actually connects with something. This includes the impact visuals and audio, dealing [damage](#) and adding impact forces. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BulletAmmoEffect	The bullet ammo effect uses the Surfaces system to react to a hit.
AdvancedBulletAmmoEffect	The advance bullet ammo effect expands on the BulletAmmoEffect to add randomised damage and damage drop off over range.
PooledExplosionAmmoEffect	The pooled explosion ammo effect spawns an explosion at the impact point.
ParticleAmmoEffect	The particle ammo effect spawns a pooled ParticleImpactEffect particle system at the impact point.
PenetratingHitscanAmmoEffect	Penetrating ammo effects allow shots to penetrate through surfaces up to a specific depth. This version performs a hitscan shot on the other side of the surface.
PenetratingProjectileAmmoEffect	Penetrating ammo effects allow shots to penetrate through surfaces up to a specific depth. This version spawns a new projectile on the other side of the surface.
RicochetHitscanAmmoEffect	Ricochet ammo effects allow shots to bounce off surfaces based on the hit angle. This version performs a hitscan shot from the point of impact.
RicochetProjectileAmmoEffect	Ricochet ammo effects allow shots to bounce off surfaces based on the hit angle. This version spawns a new projectile at the point of impact.
SurfaceBulletPhysicsAmmoEffect	The surface based bullet physics ammo effect gives detailed per-surface type control over a projectile bullet's penetration and ricochet behaviour. Surface info is specified in a SurfaceBulletPhysicsInfo asset.
TargetTrackingAmmoEffect	The target tracking ammo effect is a targeting system. The last object hit with this ammo effect will become a target for guided projectiles.
DelegatedTargetTrackingAmmoEffect	The last object hit with this ammo effect will be assigned as the target in the connected TransformTargetingSystem .

Ammo

The ammo module handles the ammo type of the weapon. This includes the total and current amount. NeoFPS comes with the following examples:

NAME	DESCRIPTION
SharedPoolAmmo	The shared ammo pool references an ammo inventory item. This allows ammo to be shared between multiple firearms.
CustomAmmo	Custom ammo is a self contained ammo component unique to the firearm it is attached to.
InfiniteAmmo	The infinite ammo module is actually both an ammo module and a reloader module. It completely removes the need to reload.
RechargingAmmo	The recharging ammo will build up its ammo pool up to a maximum over time.

Reloaders

Reloader modules act like the magazine attached to the firearm. It reloads using ammo from the attached ammo module. NeoFPS comes with the following examples:

NAME	DESCRIPTION
SimpleReloader	The simple reloader holds a set amount of ammo and reloads it all in one go.
ChamberedReloader	The chambered reloader uses a different delay and animation when reloading from empty vs with a round chambered.
IncrementalReloader	The incremental reloader is reloaded in chunks. It can be interrupted by firing and resumed later. It is used in the example shotgun to reload one shell at a time.
CustomRevolverReloader	This reloader is a variant of the simple reloader that displays the correct number of bullets in the revolver drum and swaps them for empties. It is used to demonstrate how to interact with animations to create complex effects.
PassthroughReloader	The passthrough reloader does away with the magazine and fires bullets directly from the ammo module attached to the weapon.

You can also add the [ReloaderCountdown](#) behaviour to a firearm to play audio clips as the magazine count approaches zero. This can be useful for adding a warning sound that the magazine is almost empty, or for effects such as the iconic ping of a garand rifle.

Aimer

Aimer modules describe what happens when the player tries to aim down the weapon sights. A sniper rifle would hide the weapon geometry and show a UI scope. A pistol would raise the weapon to align its sights to the camera. NeoFPS comes with the following examples:

NAME	DESCRIPTION
AnimOnlyAimer	Sets an animation bool parameter, but does not affect the camera or crosshair. It is mainly intended for using the modular firearm system with AI.
CameraConstraintsAimer	Applies a constraint to the character camera's FirstPersonCameraTransformConstraints component.
HeadMoveAimer	Offsets the first person camera to align it to the weapon sights (allows tilt).
InstantScopedAimer	Instantly hides the weapon geometry and shows a UI scope.

NAME	DESCRIPTION
ScopedAimer	Raises the weapon to align with the camera over a set duration. Once the weapon is raised the its geometry is hidden and a UI scope is shown.
WeaponMoveAimer	Raises the weapon to align its sights with the camera.

Recoil Handlers

Recoil handlers define how recoil affects the firearm object when it shoots. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BetterSpringRecoilHandler	Uses the additive transform system to add procedural animation when the firearm recoils, along with modifying its accuracy. This new module is more intuitive and easier to control than the old SpringRecoilHandler module.
AccuracyOnlyRecoil Handler	Modifies the accuracy of the weapon with each shot, recovering over time.
SpringRecoilHandler	(Deprecated) Uses the additive transform system to add procedural animation when the firearm recoils, along with modifying its accuracy. This has been replaced by the BetterSpringRecoilHandler module.

Muzzle Effects

Muzzle effect modules display the muzzle flash of the weapon and plays gunshot audio. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BasicGameObjectMuzzleEffect	Activates a game object for a brief period and then deactivates it again. Picks audio at random from an array of audio clips.
SimpleParticleMuzzleEffect	Triggers a particle system to emit with each shot.
AdvancedParticleMuzzleEffect	Triggers one or more particle systems to emit with each shot. The particle systems are reparented to the character on start so they can persist between weapon switches, and to allow simulation in character space.
RandomObjectMuzzleEffect	Activates a game object chosen at random from a pool, and then deactivates it again after a brief delay. Picks audio at random from an array of audio clips.

Ejectors

Ejector modules optionally throw empty shells out of the weapon when it fires. NeoFPS comes with the following examples:

NAME	DESCRIPTION
StandardShellEject	Spawns a pooled shell object and throws it out from the weapon. Can be triggered instantly on firing, after a delay, or with an animation event.
ObjectSwapEjector	Swaps the specified object with a pooled physics object. Used to swap animated shell casings with physics based ones at a certain point in the weapon animation.
MultiObjectSwapEjector	As the object swap ejector, but can swap multiple items at once.

NAME	DESCRIPTION
ParticleSystemShellEject	Triggers play on one or more particle systems whenever a shell is ejected.

Utilities

The modular firearms also come with various utilities to extend them with new features:

NAME	DESCRIPTION
AnimatedFirearmSprintHandler	Models firearm sprint behaviour using keyframed sprint animations on an Animator .
AttachedAmmoCounter	This is connected to a UI text element to display the firearm's magazine ammo on a world-space UI.
FirearmAimFatigue	Applies a stamina drain to the wielder's StaminaSystem when aiming down sights.
FirearmOverheat	Adds a barrel glow and haze effect on firing. You can also enable overheating, meaning that once the heat limit is reached, the firearm trigger will be blocked until it cools down enough. This can be paired with a HudFirearmOverheatBar to show the heat status in the player HUD.
FirearmTransformMatchSetter	Communicates with a TransformMatcher additive transform behaviour to sync the wielder's head animation to the firearm.
FirearmWieldableStanceManager	Used to specify procedural or keyframed weapon stances such as crouching or falling.
HolographicSight	Controls the colour and brightness of a holographic projection weapon optic.
LaserPointerAimerSwitch	Combined with a laser pointer, switches the aimer module while the laser is on.
ProceduralFirearmSprintHandler	Models firearm sprint behaviour using procedural animation.
RecoilPushback	Add this to a firearm to apply a force directly away from the shot direction when firing.
RenderTextureScope	Controls the camera, reticule and material for a render texture scope to add parallax and off-axis fade.

Animations

Animation of weapons is based in part on the [ModularFirearm](#) behaviour and also on the attached modules as well. As different modules behave in different ways they often need to be animated differently to match. For example, the semi-auto trigger uses an animator trigger property to tell the animator to play the trigger press and release animation. The full auto trigger uses a boolean property instead to tell the animator how long to hold the trigger, and the charged trigger uses a float property to tell the animator how much to blend the trigger pull / hammer cock animation.

Each animation key can be changed in the firearm or module's inspector and is set by default to match the example assets provided with NeoFPS.

In some cases, the animation can also define the timing for individual actions such as incrementing the magazine ammo count with the incremental reloader. NeoFPS comes with an example animation event handler called the [FirearmAnimEventHandler](#) which catches [Unity animation events](#). It is very simple to implement your own animation event handlers and provides a great deal of flexibility beyond using timers, however in the example modules any potentially timed actions have an option to set a specific wait duration or to wait for an animation event before triggering.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using [Additive Transforms and Effects](#). These can be very simple, such as the spring effects driven by the recoil modules. They can also be more

complicated, such as the procedural sprinting animation using the [ProceduralFirearmSprintHandler](#). For larger scale movements, the main modular firearm behaviour has a pose system built in which allows for seamless transitions between poses. This is used by a number of modules such as the [WeaponMoveAimer](#) or the [FirearmWieldableStanceManager](#).

Footstep driven animation such as sprinting or weapon bob is driven through a step tracking system attached to the character's motion controller. This ensures that any animations like this sync up, are only active while in the correct movement state, and are correctly driven by the character's velocity. The character's stride length (and therefore bob speed while moving) is controlled by adding a [TrackSteps](#) motion graph behaviour to the relevant states or sub-graphs in your character's motion graph.

*Note: If you have a weapon lower animation in your animator controller, then it is a good idea to transition to weapon raise at the end of it. The Unity animator can sometimes fail to transition if too many are triggered in very quick succession (eg tapping between 2 weapon slots), and this will prevent it from causing a problem.

For a guide to creating and using animator controllers for NeoFPS firearms, I have put together a youtube playlist that breaks down some techniques and features here:

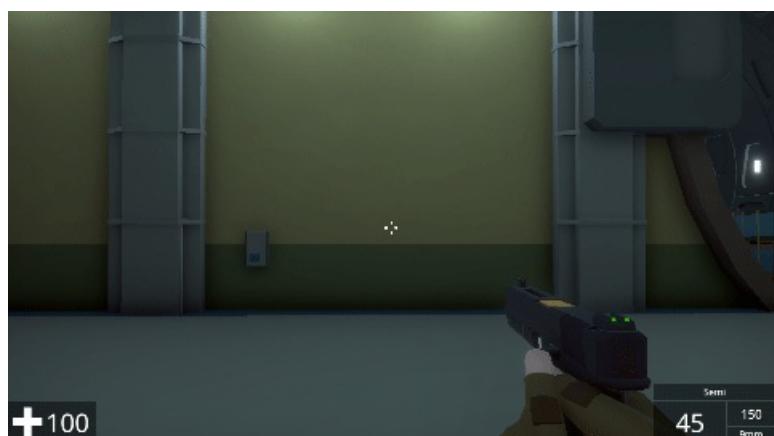
Drops and Pickups

Firearms have their own "drop" system that allows a player or character to drop them from their hands. The firearms need their own system to account for ammo usage and additional data such as attachments. When a firearm drop is spawned by dropping the weapon from your inventory, then the attached ammo is set to the amount in the firearm's magazine, and any additional payload data such as attachments are set to match the weapon that was dropped. If the drop is placed in a scene to be used as a pickup, then the ammo count and payload data will be set based on the values in the [ModularFirearmDrop](#) inspector.

In the sample assets the ammo can be picked up by walking over the weapon drop, while the firearm itself needs to be interacted with to pick back up. For more information, see [ModularFirearmDrop](#) and [ModularFirearmAmmoPickup](#).

The easiest way to create a firearm drop is using the [PickupWizard](#) in the NeoFPS hub.

Switching Modes



The NeoFPS firearms have a simple system for switching weapon modes. Mode switching is delegated to the [ModularFirearmModeSwitcher](#) behaviour, which should be attached to the GameObject with the [ModularFirearm](#) behaviour. This works by grouping firearm modules into weapon modes and then enabling them once the mode is switched. Since only one of each type of module can be enabled at any one time, the [ModularFirearm](#) behaviour will automatically disable the old modules when new ones are activated.

Some examples of things you can do with the firearm mode switcher include:

- Switching trigger groups to cycle between semi-auto, burst fire and full auto
- Switching ammo types in a grenade launcher
- Swapping aimer and trigger when switching a rifle from close combat to long range mode
- Switching muzzle effect (audio and visual) once a silencer is attached to the gun

Visual Effects

The modular firearms feature various options and modules for controlling their visual effects. The samples are set up with a low poly style that uses simple geometry and objects for muzzle effects and ejected shells. You can also use particle systems to create more realistic effects, and the asset ships with more realistic drop-in replacements for each of the demo firearms. These muzzle effect prefabs can be found at: *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\MuzzleFlashes*

To use the new muzzle effects, you will need to remove the [Basic Game Object Muzzle Effect](#) from the firearm prefab. Navigate through the weapon hierarchy until you find the **MuzzleFlash** game object. Delete this, and drag & drop the relevant realistic muzzle effect from the above folder into the same place in the hierarchy. Each of these realistic effects is ready set up with an [Advanced Particle Muzzle Effect](#) behaviour and positioned based on the main weapon mesh position.

Alongside the muzzle effects, NeoFPS also comes with a variety of hitscan bullet trail prefabs, and projectile prefabs. These can be found in the following folders: `Assets/NeoFPS/Samples/Shared/Prefabs/Weapons/Trails/` and `Assets/NeoFPS/Samples/Shared/Prefabs/Weapons/Projectiles/`

These are dropped onto the relevant property on the shooter module. For hitscan trails, you can set the size/radius and the duration of the trail (for example, the distortion bullet effect works best with a much larger size and longer duration than the additive trails).

Lastly, the NeoFPS samples include a number of shaders to achieve the above effects. These are currently only available for the standard pipeline and located at: `E:\Projects\neofps\unity\neofps_stable\Assets\NeoFPS\Samples\Shared\Effects\Shaders`

Each of the shaders has been created using the [Amplify Shader Editor](#). If you own this asset, then you will be able to tweak them for your needs or use them as a starting point for your own shaders. Shaders that are intended for use with particle systems, such as the shockwave or fireball sheets make use of vertex colours to control intensity. A number of shaders also expose properties through material property blocks such as the glow amount on the glow shaders or highlight amount on the interactive object highlight shaders.

See Also

[Health and Damage](#)

[Explosions](#)

[Inventory](#)

[First Person Camera](#)

[ModularFirearm Reference](#)

[BallisticProjectile Reference](#)

[FirearmAnimEventHandler Reference](#)

[ModularFirearmDrop Reference](#)

[ModularFirearmAmmoPickup Reference](#)

[Unity Animation Events](#)

Hitscan vs Projectiles

One of the most important choices you can make when defining the feel of your guns is whether to use a hitscan or projectile based shooter.

Hitscan

Hitscan uses a raycast to detect what a shot hits instantaneously. The shooter module picks a direction for the shot based on the camera direction, the weapon direction, and the current accuracy of the firearm and draws a straight line from the muzzle tip to the first thing it hits.

The instantaneous nature of hitscan makes it well suited to fast paced, arcade style games where responsiveness is key. You don't need to lead your targets or compensate for bullet drop.

In NeoFPS, the visual aspects of hitscan shots are represented by the hitscan trail behaviours and a range of shaders. The hitscan trails available are:

- [Line Renderer Hitscan Trail](#) uses a Unity line renderer to draw a line from the muzzle tip to the hit point. The tracer size property on the shooter modules acts as a multiplier on the width set in the line renderer.
- [Particle System Hitscan Trail](#) emits a set number of particles per meter along the line from muzzle tip to hit point. |
- [Line And Particle Hitscan Trail](#) combines a line renderer and particle system in one. This is useful for effects like tracer using the line renderer, followed by smoke or dust from the particle system.
- [Noisy Line Hitscan Trail](#) uses a particle system with noise to drive the points in a line renderer. This creates an effect where the tracer starts as a straight line and then curls and spirals apart.

The following shaders are also available to create more interesting hitscan trail effects:

- The **NeoFPS_DissolveTrail** shaders start out solid and then dissolve away based on the tracer duration setting on the firearm's shooter module. There are additive and alpha shaded versions, along with edge blend versions which fade out the material at the outside edges of the trail.
- The **NeoFPS_DistortionTrail** adds a refraction style distortion effect similar to that scene in games like fear. This works especially well with a larger tracer size and longer duration. The shaders are available at `Assets\NeoFPS\Samples\Shared\Effects\Shaders`.

Projectiles

Projectile based shooters spawn a projectile at the firearm's muzzle tip with the specified velocity. A projectile travels over time until it hits something. They can also optionally be affected by gravity or even driven by complex movement logic. A large and slow projectile with gravity can be used to model grenade launchers and similar weapons, but you can also use much higher velocities to create standard bullets. For example, the muzzle velocity of a .45" pistol is in the region of 250m/s, a grenade launcher is more like 75m/s, while an assault rifle would be more like 750-900m/s. In games with smaller environments, you might want to keep your muzzle velocities smaller than real life to exaggerate the visuals and bullet drop.

A number of example projectiles can be found at `Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\Projectiles`.

Projectiles are usually comprised of a root object with the projectile component attached, a mesh, which will be enabled after the projectile has travelled a set distance, and a trail renderer. Projectiles are implemented as a [PooledObject](#) and will be returned to the pool on impact. They can also have a [NeoSerializedGameObject](#) component attached, allowing the NeoFPS [save system](#) to save and load projectiles in flight.

Guided projectiles



NeoFPS has a system that lets you build guided projectiles from a combination of a tracker component and a motor component. The tracker component is responsible for detecting and choosing a target, while the motor component is responsible for flying towards it. Currently there are 2 motor components available:

- The [Simple Steering Motor](#) simply turns towards the target each frame with a maximum steering rate that increases over time (this prevents the projectile from orbiting targets).
- The [Drunk Missile Motor](#) has a much more erratic flight path. At brief intervals it picks a random direction (tending towards the target) and turns towards that. Once the projectile gets close enough to the target it adopts a more standard steering system. The projectile can also boost its speed at random intervals as well. Together these give an effect similar to the missiles seen in anime.

There are a range of tracking options for guided projectiles in NeoFPS:

- The [NearestObjectWithTagTracker](#) component checks for colliders with a specific layer and tag in its vicinity. It will target the closest valid target.
- The [PlayerTracker](#) component homes in on the player character.
- The [TargetingSystemTracker](#) component defers its target selection to a separate targeting system. There are multiple targeting systems available, which are attached to the firearm. The ballistic shooter firearm modules will check for a targeting system on the firearm, as well as a [TargetingSystemTracker](#) on the projectile, and register the projectile with the targeting system if both are found. Targeting system options available are:
- [LaserTargetingSystem](#) which is attached to a [WieldableLaserPointer](#). If the laser is on, the projectile will home in on the point the laser hits.
- [RaycastTargetingSystem](#) casts a ray from either the weapon or the camera, and passes the hit point to the projectiles. This can be a one-shot or continuous cast.
- [TargetLockTrigger](#) is a firearm trigger that checks for valid targets within a cone in front of the weapon. Holding the trigger will lock onto the target over time. Releasing the trigger before the lock completes will cancel the shot, while releasing the trigger after will fire the gun and pass the locked target to the projectile. The target is tracked continuously from that point until the projectile hits or is destroyed. You can cancel firing with the reload button.
- [MultiTargetLockTrigger](#) is a firearm trigger which locks onto multiple targets one by one as you hold the trigger. Releasing the trigger will fire as many shots as there were target locks in a burst. You can cancel firing with the reload button.
- [TargetTrackingAmmoEffect](#) allows you to tag a target with a shot, which the guided projectiles will then home in on. This can track moving targets, and will track the exact point on the target that the bullet hit. You can use this in conjunction with a the [ModularFirearmModeSwitcher](#) to allow you to switch weapon modes between tagging and missiles.

See Also

[Modular Firearms](#)

Scopes & Optics

Overview

NeoFPS provides a number of ways to implement different scopes and optics for your weapons. These include close range and long range optics in different styles.

Iron Sights



Iron sights are the simplest form of ADS optics. You simply need to make sure that your iron sights geometry is a sensible shape and layout, and then use one of the firearm aimer modules to align the weapon to the camera when aiming.

Holographic Sights and Red-Dots



Holographic and red dot sights project a reticule at a distance in front of the weapon that is only visible through the sight. This makes it easy to quickly acquire a target and adjust to the movement of the weapon during recoil.

You can increase or decrease the brightness of the reticule using the **Optics Brightness +/-** input buttons in game.

For more information see the [HolographicSight Reference](#). The Demo Facility assault rifle uses a holographic sight by default.

Render Texture Scopes



Render texture based scopes are used in games such as Escape From Tarkov that emphasise realism. They provide a zoomed in view within the scope lens, without zooming in the rest of the frame. This means that the player can maintain awareness of their surroundings much easier than they can with a HUD or stencil based scope where their whole view is zoomed in.

The NeoFPS RT scopes also have a parallax effect when moving. This involves moving the reticule and a blurred scope ring as the scope goes off the player camera's axis. It can also rotate the scope camera slightly to compensate for this off axis viewing so that the scope image feels more dynamic.

When the render texture scope gets far enough from the camera axis it starts to fade out to an opaque, reflective material. Once fully opaque, the scope camera will stop rendering.

Scopes implemented in this way incur a performance penalty since they involve rendering the scene twice. This can be mitigated slightly by showing the scope smaller on the screen (and therefore using a smaller render texture), but you should be aware that the increased realism comes at a cost.

For more information see the [RenderTextureScope Reference](#). The Demo Facility sniper rifle uses a render texture scope by default.

Stencil Based Scopes



Stencil based scopes use stencil shaders to define what parts of the weapon are visible through the lens of the scope. They provide no zoom themselves, so the entire player camera must be zoomed in. The stencil shaders are used to cut out the scope housing and mount behind the lens so that your view is unobstructed. They are also used in the opposite way to display the scope reticule and inner tube, providing a slight parallax effect, while cutting out any of the geometry that extends outside your view of the lens.

Stencil based scopes are much more performant than render texture scopes, with the trade off that the view around the scope is also zoomed in.

The Demo Facility sniper rifle has stencil based scopes disabled in its hierarchy.

Note: The stencil based scopes are currently not working with deferred rendering. A solution is being looked into.

HUD Scopes



HUD based scopes essentially hide all weapon geometry, and overlay a crosshair over the entire screen. This can be instant, or triggered at the end points of an aim-down-sights animation.

These scopes tend to have a much more arcadey / responsive feel than the other options, though it can be harder to make them look good.

You should also make sure that the shooter module on the firearm is set to use the camera aim when aiming so that the bullet

hits where the crosshair is pointing every time.

The sniper rifle in the feature demos uses a HUD based scope.

See Also

[RenderTextureScope Reference](#)

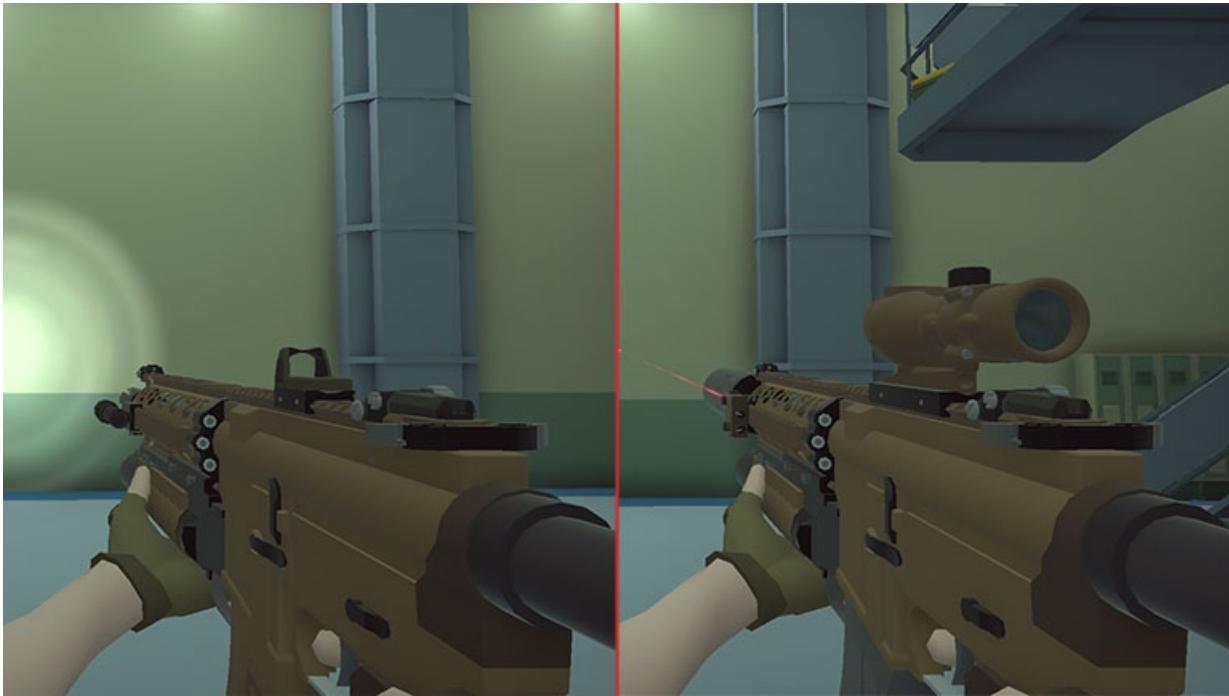
[HolographicSight Reference](#)

Firearm Attachments

Overview

NeoFPS provides a flexible attachments system with its modular firearms that can be leveraged in a number of ways. At its core, creating attachments with NeoFPS firearms simply means moving the firearm modules off the root object and into child objects that are activated/deactivated or instantiated/destroyed as required. You can create your own systems to do this, based on your needs, or you can use the example system that is provided with NeoFPS.

Firearm Attachment System



The example attachment system works through a combination of **sockets** and **attachments**. These are tracked via the [ModularFirearmAttachmentSystem](#) behaviour that must be added to the root of the gun alongside the firearm behaviour.

Attachments are the actual object that is swapped out on the weapon, and it can include things like [scopes and optics](#), magazines, muzzle tips, etc. The only requirement is that they have a behaviour attached that implements the [ModularFirearmAttachment](#) base class. NeoFPS provides an example called the [StandardModularFirearmAttachment](#), which allows you to (optionally) specify an inventory item that is required in order to use the attachment. You can create your own classes that inherit from the base class to add new restrictions, or react to being added/removed.

A socket is an object with the [ModularFirearmAttachmentSocket](#) behaviour added. This behaviour allows you to specify its socket name (eg, "Optics" or "Muzzle") and which attachments it accepts, along with its default attachment. The default attachment can be nothing, a specific attachment, or a random attachment from the available options. You can also inherit a custom class from this socket behaviour to add new default attachment logic.

Sockets are usually parented to the gun bone of the firearm, but they can also be parented to attachments themselves, creating nested sockets.



If you switch the parent attachment out to another one with a child socket of the same name, then the attachment from the original child socket will be transferred and realigned to the new one. An example would be switching the barrel on a shotgun with a shorter one, and the muzzle attachment transferring to the new socket position.

To change the attachments on a gun, an API is provided that allows you to get the sockets registered with the system, and then apply an attachment via prefab, index or ID. An example popup UI is also provided that provides a multi-choice picker for each socket, indenting any nested sockets under their parent. To use the UI you need to add a [ModularFirearmAttachmentUIHandler](#) behaviour alongside the attachment system. This is then pointed at a [ModularFirearmAttachmentUIPopup](#) prefab such as the one provided at: `Assets\NeoFPS\Samples\Shared\Prefabs\Popups\FirearmAttachmentsPopup.prefab`

You can also create a new popup style by inheriting from the base class of the example popup.

Another way to change attachments more directly is to create attachment pickups using the [ModularFirearmAttachmentPickup](#) behaviour. This can be attached to an [interactive object](#) or an [interactive pickup](#). Use the latter if the attachment has an inventory object requirement, and you want the character to pick up the object at the same time.

The Demo Facility folder contains a selection of weapons that use the attachment system in the folder:
`Assets\NeoFPS\Samples\SinglePlayer\Scenes\DemoFacility\Weapons\ModdableWeapons`

Optics Alignment

In order to align the camera to your weapon's optics when aiming down sights, an offset is calculated from the root node on start. This causes an issue with the attachment system as adding an optics attachment during play would mean that offset will be calculated based on the current position and rotation of the weapon. On start, the weapon will be in its default pose. During runtime, the weapon could be mid-way through an animation or influenced by any number of procedural animation effects.

To fix this problem, the attachments update added a new setting to the [ModularFirearm](#) behaviour called **Aim Relative Transform**. If you are using the attachment system with your optics, then you need to assign this property to a transform in the weapon hierarchy that moves with the gun itself. This will usually be the weapon bone that the firearm mesh is parented or skinned to. Once that is done, then instead of calculating the offset from the root on start for each optics / aimer, then the offset will be calculated relative to this transform. Since the optics socket will be a child of this transform, the offset for any optics / aimers attached to this socket will be correct no matter what the weapon is doing at the time.

Note: the firearm aimer modules can have trouble calculating the offsets if the socket they're attached to is not scaled at (1,1,1). If you need to change the attachment sizes then it is better to do that in the attachments themselves.

Flashlights and Laser Pointers



NeoFPS flashlights just use a spotlight object which is toggled on or off with the **Flashlight** input button. There is also an API for setting the flashlight's brightness.

For more information, see the [WieldableFlashlight Reference](#). The Demo Facility moddable pistol, shotgun and assault rifle each let you attach a flashlight.



The laser pointer also uses the flashlight system to toggle on or off using the **Flashlight** input button. It uses a line renderer with a custom shader for the beam and a generated billboard quad for the flare at the point of impact.

Using laser pointers does have some gameplay implications. By default, the NeoFPS firearms use an accuracy system for determining where shots hit, and also use the camera / HUD crosshair to determine the direction to fire. As accuracy decreases due to recoil, movement or other modifiers, the bullets spread out from the aim point. This can feel wrong when you have a laser pointer attached to the weapon which shows you where each of your bullets *should* hit. You can help with this by changing some

of the default settings on the weapon:

- Set the shooter modules to **never** use the camera aim. This means that the weapon will always fire directly in front of it.
- Reduced the max spread / max aim offset to a very low number.

These 2 changes will mean that bullets should always hit where the laser is pointing, though it also means that your weapons are now incredibly accurate. You can mitigate this by emphasising the procedural animation systems and keyframed animation on the model. For example:

- Add a [WeaponAimAmplifier](#) additive transform effect to rotate the weapon as you move your view
- Use a [BreathingEffect](#) additive transform effect along with either a [SimpleBreathHandler](#) or [StaminaSystem](#) on the player character to add breathing motion.
- Add a [CameraShake](#) additive transform effect to the weapon itself with a subtle amount of twist / rotation shake (no position shake).
- Add a [RotationBob](#) additive transform effect to the weapon to layer in rotation as you move instead of just the default position bob.
- Use the **wander** settings on the weapon's recoil handler module to increase the "spread" of the weapon when firing rapidly.

You can use the **On Toggle On** and **On Toggle Off** events on the laser pointer component to enable/disable the above components, or to change their settings while the laser is switched on.

Another component that can be used to modify the feel of a weapon with a laser pointer is the [LaserPointerAimerSwitch](#). When a laser pointer is present, this behaviour detects when it is switched on or off, and switches the firearm's aimer module based on its state. This means that you can switch to a canted pose that obstructs less of your view with the laser on, and switch back to optics when it's off. Each time the laser is switched on it will record the firearm's current aimer module, and then subscribes to its `onAimerChange` event in case you switch the aimer attachment while the laser is switched on. You can place this behaviour on the root of the weapon alongside an aimer module that is set **not** to start active. This will then be used with the laser.

For more information, see the [WieldableLaserPointer Reference](#). The Demo Facility moddable pistol, shotgun and assault rifle each let you attach a laser pointer, and each use the optional aimer switch behaviour mentioned above to provide a laser aim pose.

See Also

[Modular Firearms](#)

[Scopes & Optics](#)

[Additive Transforms & Effects](#)

Firearms Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

My Gun Is In A Weird Position

NeoFPS assumes that weapons treat the origin as the camera position. This means that the camera would be at 0,0,0 relative to the root object of the weapon. Some weapon model assets use a different setup, such as a humanoid rig where the origin is at the character's feet. In these situations you can use the **Origin Point - Match Transform** property on the [ModularFirearm](#) component. You will need an object in the weapon view-model's hierarchy that represents the position of the camera. This would usually be called **Camera** or something similar. Check that this object is correctly oriented, with the Z-axis facing forwards and Y-axis facing up. **Drag the camera object onto the Match Transform** property and it will align the view model so that the camera object is at 0,0,0 relative to the root of the weapon, just as the NeoFPS camera would be.

Note: The Match Transform property does not store the transform. It is used once when you drag and drop it in to apply the offset and then forgotten.

If you have done the above and the weapon seems to be aligned properly in the prefab, but not in game then it is probably the view-model's Animator that is resetting the object's position. Depending on the import settings of the animation clips, they may have keyframes for the root object position that are overriding the offset. You can usually fix this by switching the **Apply Root Motion** property on the animator to **true**.

When I Aim Down Sights The Gun Doesn't Align To The Camera

The [WeaponMoveAimer](#) firearm module (along with a number of others) have a number of properties to define the aim offset.

Firstly, there is the **Aim Offset** property which takes a transform. If this is set then you should check that the selected transform is positioned correctly relative to the sights, and that it is pointing its Z-axis (blue arrow) forwards. If both these are true then it could be that the weapon's starting pose is not its idle pose. The aimer module uses the aim offset transform to calculate an offset position and rotation at start, not continuously, so if the weapon idle differs from the default pose of the weapon then these values can be wrong. It is good practice to have first frame of an animated FBX be a neutral pose such as a T-pose for a character. If that is not the case then you will need to either add a new transform to use as the aim proxy that is correctly aligned to the idle pose, or you will need to use the aim position and aim rotation settings instead.

If no aim offset transform is set, then you will also see **Aim Position** and **Aim Rotation** properties. These allow you to set manual offsets from the idle pose to align to when aiming. You can tweak these to get the correct alignment.

Gun Doesn't Shoot Forwards When Aiming Down Sights

The firearm's shooter module has a setting called **Use Camera Aim** that defaults to using the camera for the initial shot raycast when hip firing, and the muzzle tip transform when aiming down sights. If your gun is shooting off to the side or up/down when ADS, that suggests that the transform you have set as the **Muzzle Tip** in the shooter module is facing the wrong direction. Check that this has the **Z-axis (blue arrow) facing forwards**. You should also check that you have the tool handles rotation set to local and not global using the toggles at the top of the Unity editor (next to the transform tools). If not, then the muzzle tip will look as though it is facing forwards regardless of its actual direction.

My Gun Shoots Through Objects

Each of the firearm's shooter modules has a **Layers** setting. Make sure that the layer the object's collider is using is added to this property.

My Bullets Don't Leave A Mark

The decal system in NeoFPS is quite simplistic and can't attach decals to moving objects. To reduce the chance of this happening, the "Default" layer has decals disabled by default. You can re-enable decals for this layer (or any other) by going to the NeoFPS Hub, selecting the **Surface Manager** from the Managers section, and adding the layer to the **Decal Layers** property.

I Can't Select My New Gun

This usually means that the quick-slot index of the item is incorrect. There are a few possible causes for this. Please check the following:

- If your character is using an **FpsInventorySwappable** inventory, then your weapon **must** use the **FpsInventoryWieldableSwappable** wieldable component. If not, then it should use the **FpsInventoryWieldable** component.
- When using the quick-switch inventory, the maximum quick-slot index is 9.
- Make sure that you do not have multiple wieldable components on your weapon. The inventory will only detect the first one.

Muzzle Flash Keeps Repeating After Shooting

If you are using one of the particle system based muzzle flash modules, then there are a couple of things that you need to be aware of when setting up your particle systems. Firstly, make sure that **Looping** is set to **false** in the main settings. If this is true then the particle system will repeat indefinitely. Secondly, check that **Play On Awake** is set to **false**. If this is true then the particle system will play the first time the weapon is selected.

Shooting Interrupts My Aim Up/Down Animations

When using animations to transition between aiming and hip fire states, depending on how your animator controller is set up, you might find that shooting interrupts the raise/lower animations. This is usually because the shooting animation is set to transition instantly from *Any State*. You can get around this by either blocking the trigger while transitioning, or by switching to procedural aim animations.

- If you are OK with the player not being able to shoot while changing aim state, you can switch the **Block Trigger** setting on the aimer module on. This property will only appear if the animation key **Aim Anim Bool** is not empty.
- To use procedural animations, clear the **Aim Anim Bool** so that no animation is used, and use an **Aim Offset** transform instead.

Switching to procedural aiming has other advantages as well, such as being able to adjust the aim offset for different optics. The best quality is to combine the 2, using an animation with some form of jiggle or hand movement, but which doesn't actually move the weapon, and then using the procedural system to handle the offset.

My Weapon Idle Animation Goes Weird After Shooting/Drawing/Reloading

If you have created or altered your own animations using Unity's animation tools then it is possible that the issue here is due to which objects you have keyframed. If an animation clip does not have any keyframes for an object then it will not change from the position / rotation it enters the animation state with in the animator controller. For example, if you have an idle animation that does not have any keyframes on the weapon object, you have a draw animation that ends in the idle pose, but your reload ends on a different pose, then your weapon will look fine until the reload animation completes. At that point the weapon's idle will be in the reload end pose until you next deselect and reselect the gun. The fix is to edit your idle animation (and any others affected) to make sure that all relevant objects have keyframes set at the beginning and end frame, **even if they do not change from the default pose**.

The Firearm Wizard Created A Gun In The Scene But No Prefab

This suggests that something is broken with the wizard. Please contact support via the [Discord](#) or via [e-mail](#). If you can copy/paste any errors from the console (doing this provides script files and line numbers for the error) that would help track down the problem.

Melee Weapons

Overview



Melee weapons in NeoFPS are a limited range weapon that inflicts damage and impact force directly ahead when the animation is at the striking point.

The NeoFPS melee system is currently quite basic, and will be expanded with more options in future updates. You can extend it with your own custom behaviour via scripting as outlined below.

Wieldables vs Quick-Fire

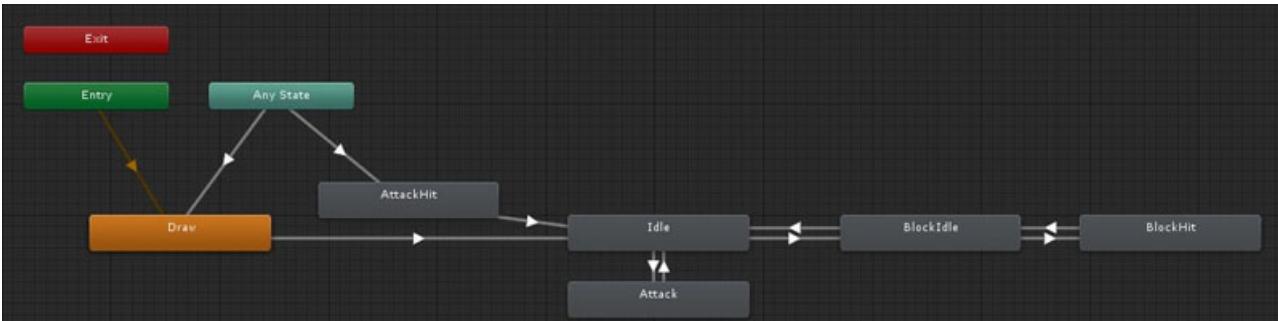
The demo melee weapon is set up as wieldable. If a melee weapon is wieldable, that means it is equipped when selected, and then you have to hit the primary fire input to attack. You can achieve this with your melee weapons by adding an [FpsInventoryWieldable](#) or [FpsInventoryWieldableSwappable](#) behaviour to the root of your weapon prefab depending on your choice of inventory. These handle selection and deselection, and also define which quick-slot they apply to.

Alternatively, you can set up your melee weapons as quick-use items. This means that when you select them, the weapon will raise and then immediately attack before lowering again. This can be used alongside another equipped weapon (optionally interrupting its use). To achieve this, you would add an [FpsInventoryQuickUseItem](#) or [FpsInventoryQuickUseSwappableItem](#) behaviour to the root of your prefab instead of the wieldable behaviours above.

Weapon Pickups

You can create a pickup object for your melee weapon using the [Pickup Wizard](#) in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later.

Melee Animations



The melee weapon animations are relatively simple. See the sample assets for details on how the graph is set up.

The relevant animator controller properties are:

Name	Type	Description
Draw	Trigger	Used when the weapon is drawn to raise it from off the screen. This is also the entry state of the controller.
Attack	Trigger	Signals the start of an attack.
Attack Hit	Trigger	Signals that the attack connected. This is used to trigger a bounce back reaction animation and will interrupt the attack animation that would otherwise follow through.
Block	Boolean	Used to specify when the weapon should be raised in a blocking state.
Block Hit	Trigger	A hit reaction knock while blocking.

The property names can be changed in the [MeleeWeapon](#) behaviour.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using [Additive Transforms and Effects](#). These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the [ProceduralMeleeSprintHandler](#). For larger scale movements, the main melee weapon behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the [MeleeWieldableStanceManager](#).

Adding Custom Melee Behaviour

If you have custom behaviour that you want to be applied when you use a melee weapon, the easiest way is to add a "melee hit extension" component. This is simply a Monobehaviour that implements the [IMeleeHitExtension](#) interface. This interface has a single [OnMeleeHit\(RaycastHit hit\)](#) method. Any extension components that are found on the melee weapon's root object on start will be triggered every time you hit an object with the weapon.

If you actually want to change the hit behaviour, you can also create your own melee weapon class that inherits from the [MeleeWeapon](#) class. You can override the virtual [OnMeleeHit\(RaycastHit hit, Vector3 attackDirection\)](#) method here to either add or replace the functionality.

For more complex behaviour, you can also use the [BaseMeleeWeapon](#) class as a base for your own. This removes all the hit and block functionality, and exposes the following methods

```
public abstract void PrimaryPress();
public abstract void PrimaryRelease();
public abstract void SecondaryPress();
public abstract void SecondaryRelease();
```

See Also

[MeleeWeapon](#)

[Unity AnimatorController](#)

[Unity State Machine Behaviours](#)

Thrown Weapons

Overview



Thrown weapons in NeoFPS are a weapon that will spawn a [PooledObject](#) at a specified point with a set velocity. They can be stacked so that the character can carry a number of them. Each throw the stack is reduced until all have been used, at which point the item is removed from the character inventory.

Thrown weapons have a strong throw and a weak throw. They can also inherit velocity from the character as it moves in a similar style to games like Counter Strike.

The NeoFPS thrown weapons system will be expanded with more options in future updates.

Wieldables vs Quick-Throw

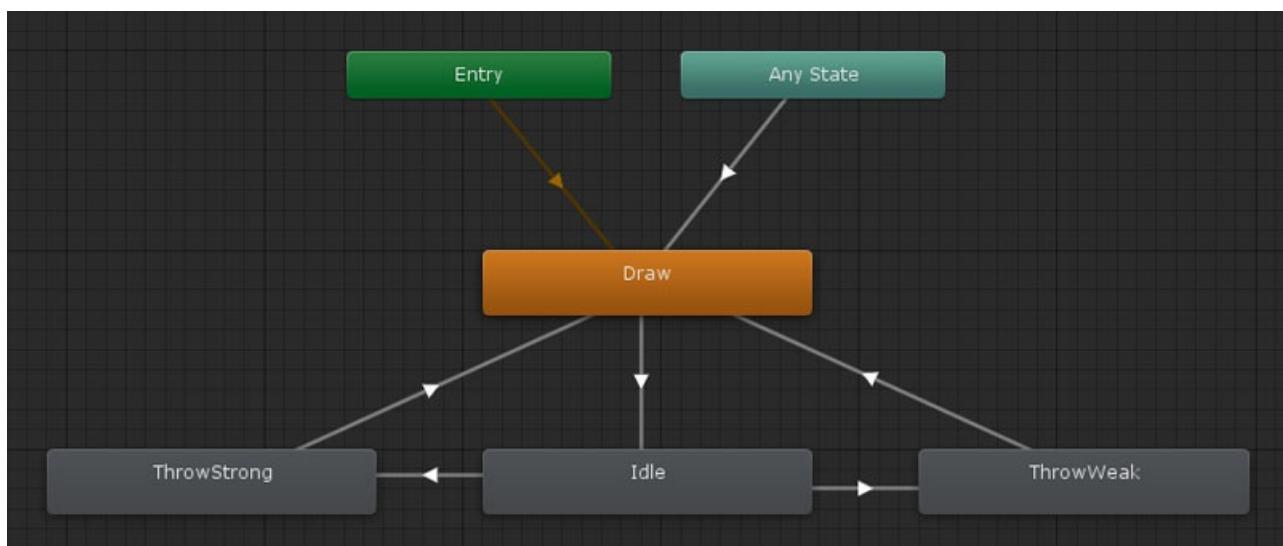
The demo frag grenade is set up as a wieldable. That means that it is equipped when selected, and then you have to hit fire to throw the grenade. You can achieve this with your weapons by adding an [FpsInventoryWieldable](#) or [FpsInventoryWieldableSwappable](#) behaviour to the root of your weapon prefab depending on your choice of inventory. These handle selection and deselection, and also define which quick-slot they apply to.

Alternatively, you can set up your thrown weapons as quick-use items. This means that when you select them, the weapon will raise and then immediately throw before lowering again. This can be used alongside another equipped weapon (optionally interrupting its use). To achieve this, you would add an [FpsInventoryQuickUseItem](#) or [FpsInventoryQuickUseSwappableItem](#) behaviour to the root of your prefab instead of the wieldable behaviours above.

Weapon Pickups

You can create a pickup object for your thrown weapon using the [Pickup Wizard](#) in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later.

Thrown Weapon Animations



The thrown weapon animations are relatively simple. See the sample assets for details on how the graph is set up.

The relevant animator controller properties are:

NAME	TYPE	DESCRIPTION
Draw	Trigger	Used when the weapon is drawn to raise it from off the screen. This is also the entry state of the controller.
Throw Light	Trigger	Signals the start of a weak throw attack.
Throw Heavy	Trigger	Signals the start of a strong throw attack.

The property names can be changed in the [ThrownWeapon](#) behaviour.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using [Additive Transforms and Effects](#). These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the [ProceduralThrownSprintHandler](#). For larger scale movements, the main thrown weapon behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the [ThrownWieldableStanceManager](#).

See Also

[ThrownWeapon](#)

[Unity AnimatorController](#)

[Unity State Machine Behaviours](#)

Wieldable Tools

Overview

The wieldable tools feature is a new **early access** feature that allows you to create wieldable items to perform a range of actions that don't fit into the categories of firearm, melee or thrown weapon. Some examples are healing stim packs, shield boosters and grapple hooks.

Each wieldable tool has a primary and secondary fire that you assign actions and tool modules to through the inspector. Each of these actions and modules is a separate MonoBehaviour component that is added to the object, similarly to the modular firearm modules.

You can find a pair of example tool prefabs in the following folder: *NeoFPS/Samples/Shared/Prefabs/Tools/*

The flashlight tool can be toggled via the primary fire button or the flashlight button. The grapple tool requires that the character you use it with have a vector parameter on their motion graph that the tool can assign the grapple point to. The parkour demo motion graph is set up with this. Note that the demo tools use the quick-switch inventory type and have specific quick slot indices that might clash with existing demo weapons. You will need to remove those weapons from the inventory before adding these tools in order for them to work (in the demos these are the baton and pistol).

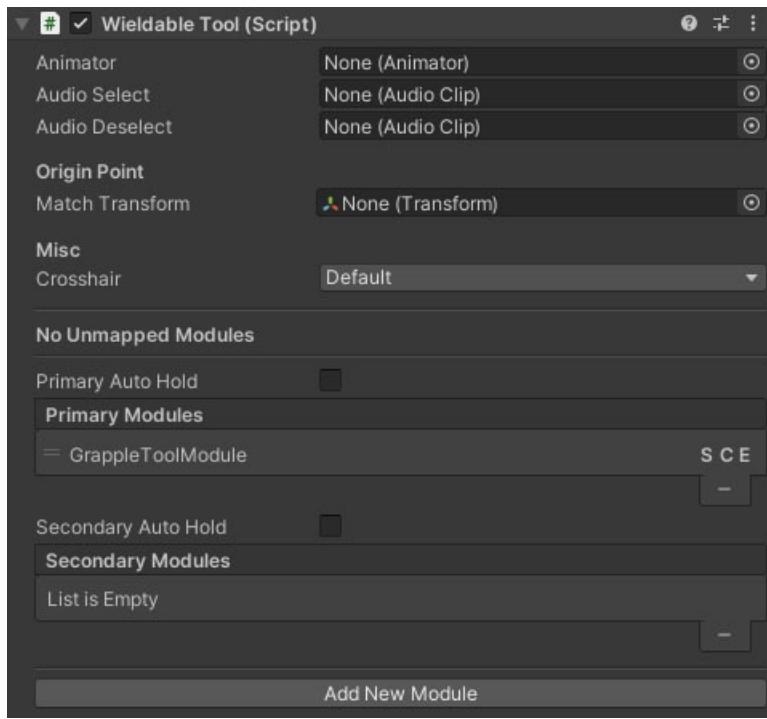
Note: The wieldable tools system does not currently come with any art assets to demo it, but art and animations work in the same way as for melee or thrown weapons. There is also no wizard currently set up to create wieldable tools through the NeoFPS Hub. Once enough feedback has been provided and the systems finalised, the wizard will be added.

Wieldables vs Quick-Fire

The original setup for a wieldable tool is as a "wieldable" item. That means it is equipped when selected, and then you have to hit the primary or secondary fire inputs to perform the tool's actions. You can achieve this with your tools by adding an [FpsInventoryWieldable](#) or [FpsInventoryWieldableSwappable](#) behaviour to the root of your tool prefab depending on your choice of inventory. These handle selection and deselection, and also define which quick-slot they apply to.

Alternatively, you can set up your tools as quick-use items. This means that when you select them, the tool will raise and then immediately perform its actions before lowering again. This can be used alongside another equipped weapon (optionally interrupting its use). To achieve this, you would add an [FpsInventoryQuickUseItem](#) or [FpsInventoryQuickUseSwappableItem](#) behaviour to the root of your prefab instead of the wieldable behaviours above.

Actions And The WieldableTool Component



The [WieldableTool](#) component checks for new modules and actions on the object and displays them in the **Unmapped Modules** section. You can then choose if they should be assigned to the primary or secondary fire for the tool (or removed from the object).

Each module has three letters next to it: **S**, **C**, **E**. These represent when the actions fire: start, continuous and end. Start actions fire as soon as the relevant fire button is pressed. Continuous actions fire each fixed update frame as long as the fire button is held. End actions fire as soon as the fire button is released, or when the firing is interrupted. These end actions can often be set to only fire on "success", which is defined by the other actions. For example, the [ChargeToolAction](#) uses a continuous action to build up a charge. If the fire button is released before the charge reaches 100%, then the end actions are called with a success value of *false*. If the charge reaches 100%, then the primary/secondary fire is interrupted and the end actions are called with a success value of *true*. An example use for this would be recreating the shield boosters from Apex Legends. You would use a [ChargeToolAction](#) along with a [ShieldBoosterToolAction](#) set to apply the shield boost as an end action that requires success.

A number of modules or actions allow you to specify whether they fire on start or end (including success / fail), or continuously.

If a tool has no continuous actions then the end actions will fire immediately after the start actions.

The currently available wieldable tool modules are:

NAME	DESCRIPTION
AddInventoryItemToolAction	Adds an inventory item on start and/or end, or in increments as the relevant trigger is held.
AnimatorBoolToolAction	Sets a bool parameter on the tool's animator controller as long as the trigger is held.
AnimatorTriggerToolAction	Sets a trigger parameter on the tool's animator controller as a start and/or end action.
ChargeToolAction	Builds a charge as a continuous action, setting the end actions' success parameter to true if full charge is reached and false if not. Charging can also be set up to play a looping sound with a pitch shift as charge builds, and play animations based on charge level.
BlinkToolModule	Used to create an Arkane style blink ability. The tool checks for a valid blink target and sends it to a motion graph parameter on releasing the trigger. Used with a MoveToPointState motion graph state.
ConsumeInventoryItemToolAction	Consumes an inventory item on start and/or end, or in increments as the relevant trigger is held.

NAME	DESCRIPTION
FlashlightToolModule	A flashlight that can be toggled on or off.
GrappleToolModule	Casts a ray based on the wielder's aim, and uses it to set a motion graph parameter for the hit point. Releasing the trigger resets the motion graph parameter. When paired with the GrappleSwingState motion graph state, this allows for grappling and swinging around scenes.
HealToolAction	Can be set to heal a set amount instantly on start and/or end, or to heal in increments continuously as long as the trigger is held.
PlayAudioToolAction	Plays a specific sound on start and/or end.
ShieldBoosterToolAction	Restores shield steps on start and/or end.
UnityEventToolAction	Fires a unity event on start and/or end that you can use to call methods on other components.

Wieldable Tool Pickups

You can create a pickup object for your wieldable tool using the [Pickup Wizard](#) in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later. This is done in the same way as for a melee or thrown weapon pickup (you can use the melee baton pickup template as a starting point), by selecting the "Wieldable Item Drop" pickup type.

Wieldable Tool Animations

Wieldable tool animations are generally dictated by the modules that are attached, though the raise and lower animations are specified on the wieldable tool component and behave the same as for all other wieldables (firearms, melee and thrown weapons).

The [AnimatorTriggerToolAction][tools-anim] can be used to add animations at the start and/or end actions for each fire mode by specifying trigger parameters in the tool's [animator controller](#).

You can also add keyframed sprinting animations using the [AnimatedToolSprintHandler](#)

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using [Additive Transforms and Effects](#). These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the [ProceduralToolSprintHandler](#). For larger scale movements, the wieldable tool behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the [WieldableToolStanceManager](#).

See Also

[WieldableTool](#)

[Unity AnimatorController](#)

Explosions

Overview



NeoFPS has a simple system for spawning explosions that deal damage and repel physics objects based on their distance from the explosion center.

The sample explosions are built from a series of particle effects, though you can turn any object into an explosion by adding a [PooledExplosion](#) behaviour.

An explosion can be assigned a source and a damage filter for complex damage effects. For more information, see [Health and Damage](#).

To spawn an explosion from a script, first you need to grab an explosion from the pool manager:

```
var explosion = GetPooledObject<PooledExplosion> (prototype, position, rotation);
```

To apply damage, call the `Explode()` method on the pooled explosion:

```
explosion.Explode(damage, maxForce, sourceController, transformToIgnore);
```

There are a number of scripts already available to work with explosions, such as:

NAME	DESCRIPTION
PooledExplosionSpawner	Simply spawns an explosion on command. Connect it to events to control execution.
ExplosiveObject	Represents a destructible object that spawns an explosion when killed.
PooledExplosionAmmoEffect	A firearm module that spawns an explosion when shooting something.

Explosive Barrels



Explosive barrels can be implemented using the above [ExplosiveObject](#) behaviour. This allows you to set a health amount for the barrel, specify a [PooledExplosion](#) to spawn once health reaches zero, and specify explosion damage and forces. A Unity event is also provided which you can use for actions like spawning debris objects.

An example can be found in the Demo Facility sample scene, and is available as a prefab at `Assets\NeoFPS\Samples\Shared\Prefabs\Props\Prop_Barrel01.prefab`.

See Also

[PooledExplosion](#)

[Health and Damage](#)

[Pooling](#)

[Unity Particle System Explosions](#)

Weapon & Tool Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Note: There is a separate section for troubleshooting the modular firearm system [here](#).

My Weapon / Tool Is In A Weird Position

NeoFPS assumes that weapons treat the origin as the camera position. This means that the camera would be at 0,0,0 relative to the root object of the weapon. Some weapon model assets use a different setup, such as a humanoid rig where the origin is at the character's feet. In these situations you can use the **Origin Point - Match Transform** property on the weapon / tool component. You will need an object in the weapon view-model's hierarchy that represents the position of the camera. This would usually be called **Camera** or something similar. Check that this object is correctly oriented, with the Z-axis facing forwards and Y-axis facing up. **Drag the camera object onto the Match Transform** property and it will align the view model so that the camera object is at 0,0,0 relative to the root of the weapon, just as the NeoFPS camera would be.

Note: The Match Transform property does not store the transform. It is used once when you drag and drop it in to apply the offset and then forgotten.

If you have done the above and the weapon seems to be aligned properly in the prefab, but not in game then it is probably the view-model's Animator that is resetting the object's position. Depending on the import settings of the animation clips, they may have keyframes for the root object position that are overriding the offset. You can usually fix this by switching the **Apply Root Motion** property on the animator to **true**.

I Can't Select My Weapon / Tool

This usually means that the quick-slot index of the item is incorrect. There are a few possible causes for this. Please check the following:

- If your character is using an **FpsInventorySwappable** inventory, then your weapon **must** use the **FpsInventoryWieldableSwappable** wieldable component. If not, then it should use the **FpsInventoryWieldable** component.
- When using the quick-switch inventory, the maximum quick-slot index is 9.
- Make sure that you do not have multiple wieldable components on your weapon. The inventory will only detect the first one.

My Grenades Fall Out Of The World

This suggests that the prefab you are using as your thrown projectile is using a Rigidbody, and is set to use a physics layer that does not collide with the environment. Check that your projectile prefab is using a layer such as **SmallDynamicObjects** or **Default** instead.

AccuracyOnlyRecoilHandler MonoBehaviour

Overview

The AccuracyOnlyRecoilHandler module modifies the accuracy of the weapon with each shot.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
On Recoil	UnityEvent	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).

See Also

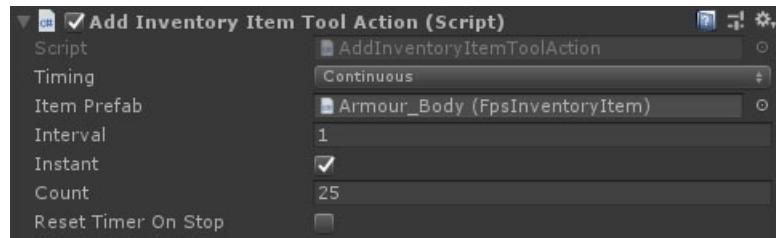
[Modular Firearms](#)

AddInventoryItemToolAction MonoBehaviour

Overview

The AddInventoryItemToolAction behaviour is used to add inventory items to the tool wielder's inventory when used. This is useful for things like repairing armour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the item be consumed.
Item Prefab	FpsInventoryItem	An inventory item prefab to add to the character inventory if none is found.
Interval	Integer	The items should be added every nth fixed update frame in continuous mode.
Instant	Boolean	Should the item be added on the first frame of the continuous action, or should it wait for the first interval to elapse.
Count	Integer	The number of times to add the item.
Reset Timer On Stop	Boolean	Should the countdown between adding items be reset when you stop using the item.

See Also

[Wieldable Tools](#)

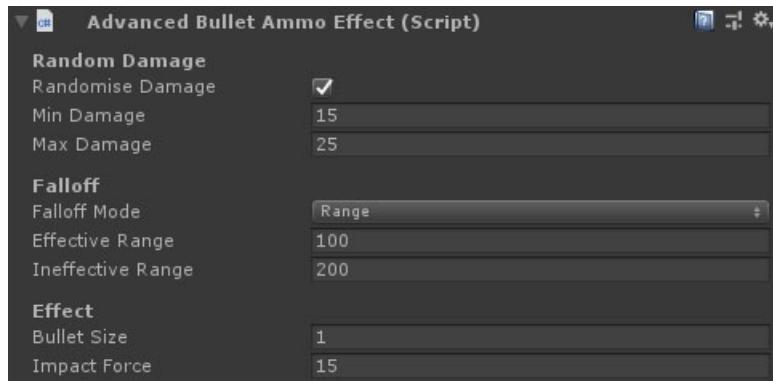
[Inventory](#)

AdvancedBulletAmmoEffect MonoBehaviour

Overview

The AdvancedBulletAmmoEffect module uses the [surfaces](#) system to show impact effects, and imparts damage and force to whatever it hits. It is similar to the [BulletAmmoEffect][4] but adds damage falloff over range and damage randomisation.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Randomise Damage	Boolean	Should the damage value be randomised or not.
Damage	Float	The damage the bullet does before falloff is applied. This property is only visible if Randomise Damage is set to False .
Min Damage	Float	The minimum damage the bullet does before falloff is applied. This property is only visible if Randomise Damage is set to True .
Max Damage	Float	The maximum damage the bullet does before falloff is applied. This property is only visible if Randomise Damage is set to True .
Falloff Mode	Dropdown	How to apply damage falloff. None means damage will not fall off over distance or speed, Range means that damage will be reduced the further the target, Speed is used with projectiles to reduce damage as they slow down.
Effective Range	Float	The max range where the bullet does the full damage (no falloff applied). This property is only visible if Falloff Mode is set to Range .
Ineffective Range	Float	The range where the bullet does 0 damage. This property is only visible if Falloff Mode is set to Range .
Effective Speed	Float	The speed above which the bullet does full damage. This property is only visible if Falloff Mode is set to Speed .
Minimum Speed	Float	The speed below which the bullet does zero damage. This property is only visible if Falloff Mode is set to Speed .
Bullet Size	Float	The size of the bullet. Used to size decals.

NAME	TYPE	DESCRIPTION
Impact Force	Float	The force to be imparted onto the hit object before falloff is applied. Requires either a Rigidbody or an impact handler.

See Also

[Modular Firearms](#)

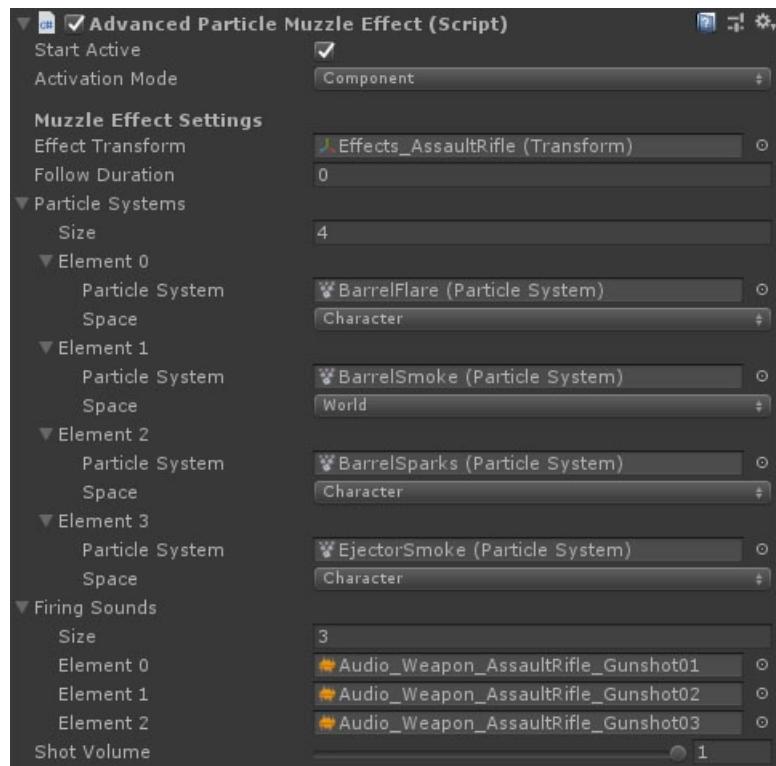
[Surfaces](#)

AdvancedParticleMuzzleEffect MonoBehaviour

Overview

The ParticleMuzzleEffect behaviour uses one or more particle systems which are played for each shot. On start, the systems are moved from the weapon to a specific transform in the character hierarchy so that the particles can persist when weapons are switched. This also allows you to simulate the particles in the character's coordinate space.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Effect Transform	Transform	The root of the muzzle effect hierarchy.
Particle Systems	ParticleSystem Info Array	The particle system to play.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

Particle System Info

NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The particle system to play.
Space	Dropdown	The simulation space for the particles. World means that emitted particles move freely in the world, Weapon means that particles move relative to the weapon (if it moves, they move with it), Character moves with the character (handles rapid character movement better than world space), NoChange leaves the simulation space as defined in the particle system

See Also

[Modular Firearms](#)

AnimatedFirearmObstacleHandler MonoBehaviour

Overview

The AnimatedFirearmObstacleHandler behaviour sets a bool animator parameter and blocks the firearm trigger whenever the player character's weapon is obstructed by an obstacle in the world. This can be used to move the weapon out of the way.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Cast Distance	Float	The distance in front of the weapon to check for obstacles.
Cast Type	Dropdown	The cast type to use. Options are Raycast and Spherecast .
Cast Source	Dropdown	What to use as the source of the cast. Options are WieldableParent , CharacterAim and Camera .
Layer Mask	LayerMask	What layers are counted as obstacles and make the character move the weapon.
Cast Radius	Float	The radius of the sphere cast (if cast type is set to Spherecast).
Min Blocked Frames	Integer	The minimum number of frames the weapon can be blocked. This prevents the weapon from rapidly switching between blocked and not.
Obstructed Key	String	The name of the animator parameter to set when the weapon's obstructed state changes.

See Also

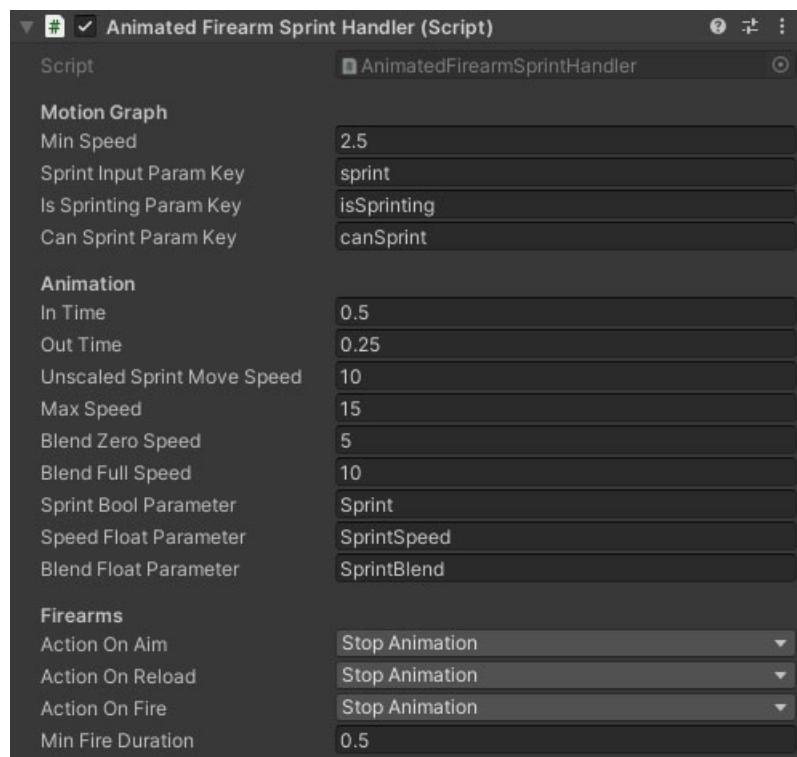
[Modular Firearms](#)

AnimatedFirearmSprintHandler MonoBehaviour

Overview

The AnimatedFirearmSprintHandler behaviour is attached to a [modular firearm](#) to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an [Animator](#) component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the [AnimatorController](#), or to drive a blend between multiple animation clips based on speed.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.

NAME	TYPE	DESCRIPTION
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the heavy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

Firearms

NAME	TYPE	DESCRIPTION
Action On Aim	Dropdown	What to do when the firearm enters / exits ADS. Stop Sprinting completely stops the character from sprinting until they leave ADS. Stop Animation stops the firearm sprint animation without affecting the character movement. Cannot Aim blocks the player from aiming down sights while sprinting.
Action On Reload	Dropdown	What to do when the firearm is reloaded. Stop Sprinting completely stops the character from sprinting until the reload animation completes. Stop Animation stops the firearm sprint animation without affecting the character movement.
Action On Fire	Dropdown	What to do when the firearm trigger is pulled while sprinting. Stop Sprinting completely stops the character from sprinting until the weapon stops firing. Stop Animation stops the firearm sprint animation without affecting the character movement (both of these have a slight delay before firing to allow the weapon to be aligned). Cannot Fire means that the firearm trigger is blocked until the character stops sprinting.
Min Fire Duration	Float	The minimum amount of time the firearm sprint animation will be paused or sprinting blocked when the trigger is pulled. Prevents rapid tapping of the trigger popping in and out of sprint.

See Also

[Modular Firearms](#)

[The Motion Graph](#)

[Unity Animator](#)

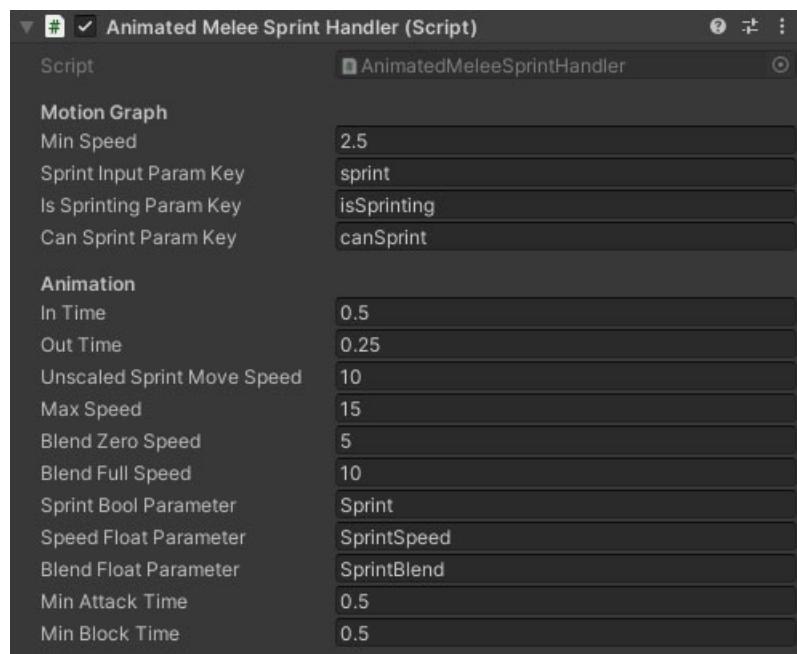
[Unity AnimatorController](#)

AnimatedMeleeSprintHandler MonoBehaviour

Overview

The AnimatedMeleeSprintHandler behaviour is attached to a [melee weapon](#) to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an [Animator](#) component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the [AnimatorController](#), or to drive a blend between multiple animation clips based on speed.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.

NAME	TYPE	DESCRIPTION
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the heavy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

Melee Weapon

NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the sprint animation will be paused after an attack.
Min Block Time	Float	The minimum amount of time the sprint animation will be paused when blocking (prevents tapping block).

See Also

[Melee Weapons](#)

[The Motion Graph](#)

[Unity Animator](#)

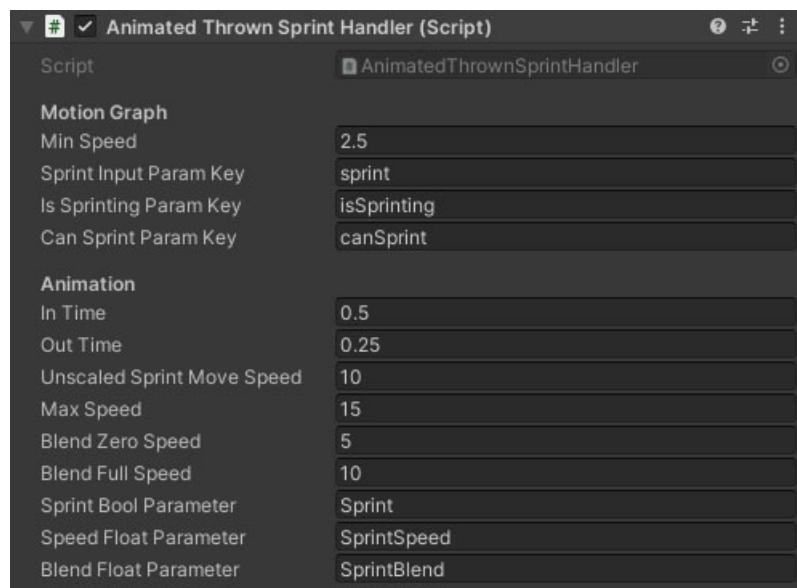
[Unity AnimatorController](#)

AnimatedThrownSprintHandler MonoBehaviour

Overview

The AnimatedThrownSprintHandler behaviour is attached to a [thrown weapon](#) to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an [Animator](#) component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the [AnimatorController](#), or to drive a blend between multiple animation clips based on speed.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..

NAME	TYPE	DESCRIPTION
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the heavy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

See Also

[Thrown Weapons](#)

[The Motion Graph](#)

[Unity Animator](#)

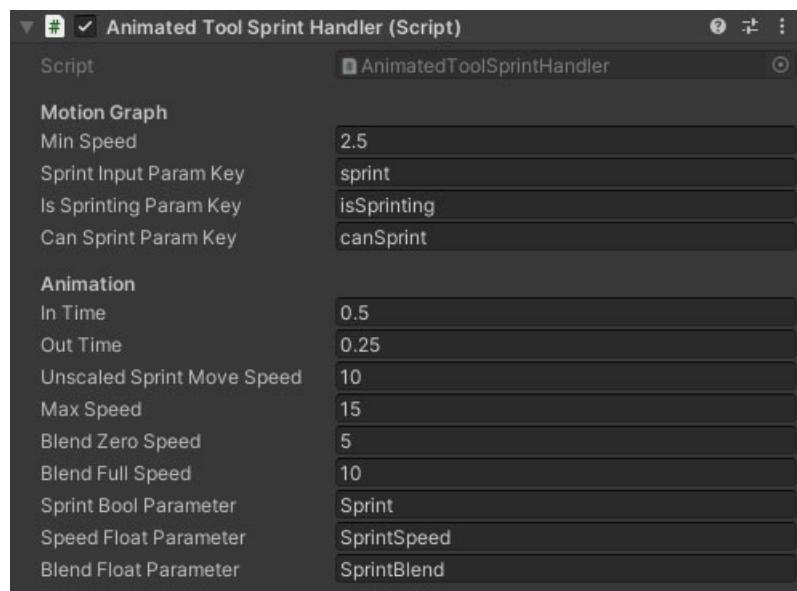
[Unity AnimatorController](#)

AnimatedToolSprintHandler MonoBehaviour

Overview

The AnimatedToolSprintHandler behaviour is attached to a [wieldable tool](#) to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your tools by setting a bool parameter on an [Animator](#) component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the [AnimatorController](#), or to drive a blend between multiple animation clips based on speed.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..

NAME	TYPE	DESCRIPTION
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the heavy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

See Also

[Wieldable Tools](#)

[The Motion Graph](#)

[Unity Animator](#)

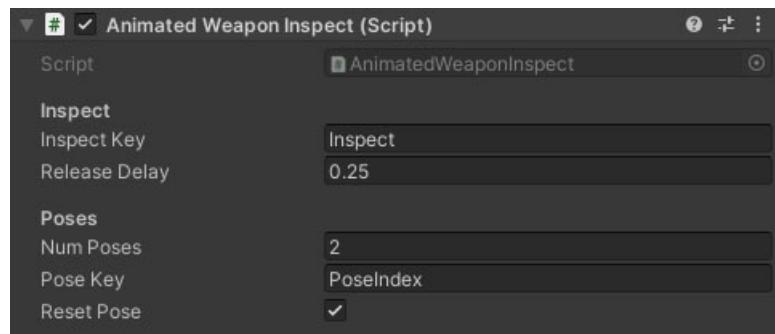
[Unity AnimatorController](#)

AnimatedWeaponInspect MonoBehaviour

Overview

The AnimatedWeaponInspect behaviour sets a bool animator parameter and blocks the weapon or item from functioning while inspecting. You can also use an int animator parameter to cycle between different inspect poses, for example when looking at attachments or trigger mode.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Inspect Key	String	The name of the bool parameter to set on the weapon's animator while inspecting.
Release Delay	Float	How long after releasing the inspect key, should the weapon be able to function again.
Num Poses	Integer	How many inspect poses does the weapon have.
Pose Key	String	The name of the integer parameter to set on the weapon's animator that controls the animation pose.
Reset Pose	Boolean	Should the pose index be reset when inspecting state changes.

See Also

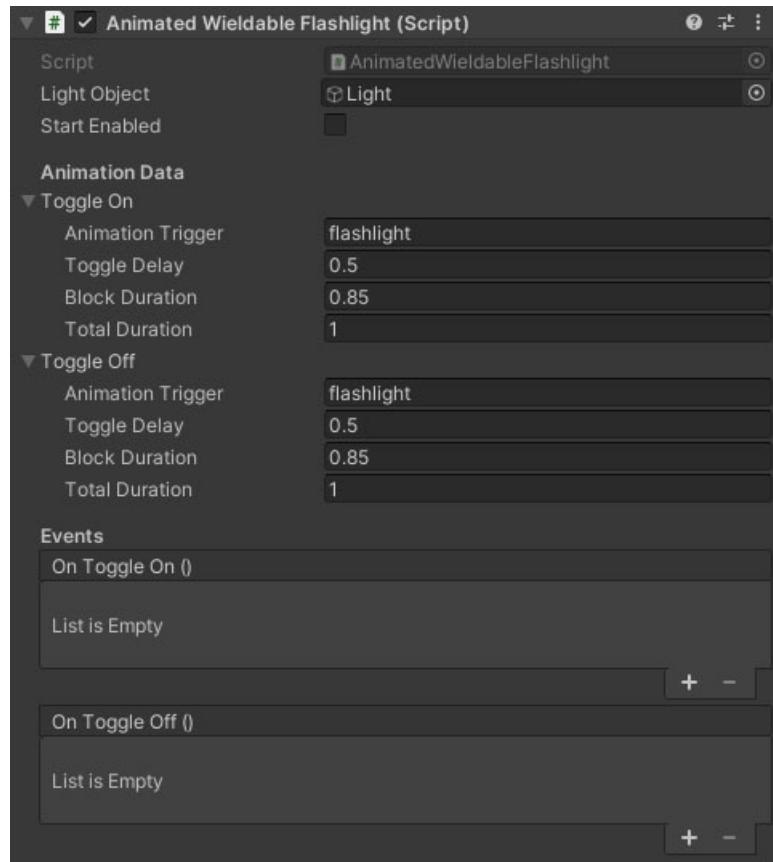
[Modular Firearms](#)

AnimatedWieldableFlashlight MonoBehaviour

Overview

The AnimatedWieldableFlashlight behaviour is used to add a toggleable flashlight to firearms. This behaviour differs from the [WieldableFlashlight](#) behaviour by playing an animation for the toggle on and off, and locking the weapon from firing or reloading while the animation plays.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Light Object	GameObject	A child object with a light component attached.
Start Enabled	Boolean	Should the flashlight be on from the start.
On Toggle On	UnityEvent	An event fired when the laser is switched on.
On Toggle Off	UnityEvent	An event fired when the laser is switched off.

The **Animation Data** section has the following properties for both **Toggle On** and **Toggle Off**.

NAME	TYPE	DESCRIPTION
Animation Trigger	String	The name of the trigger parameter to fire on the weapon's animator when switching the flashlight on or off.
Toggle Delay	Boolean	The delay from the start of the animation after which the light should actually be switched on or off.

NAME	TYPE	DESCRIPTION
Block Duration	Boolean	The amount of time from the start of the animation where it should not be possible to shoot or reload the weapon.
Total Duration	Boolean	The total duration of the animation.

See Also

[Modular Firearms](#)

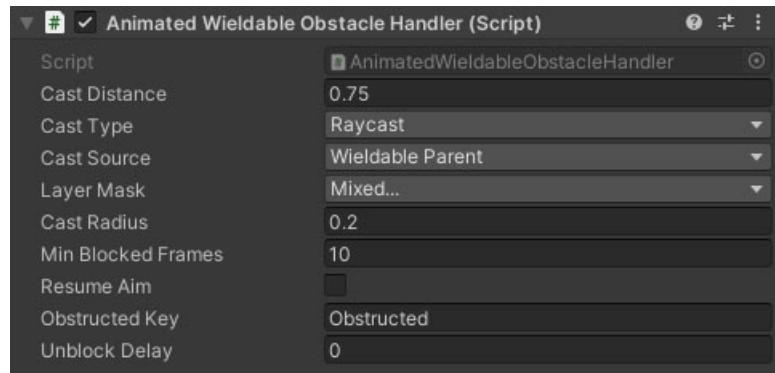
[Attachments](#)

AnimatedWieldableObstacleHandler MonoBehaviour

Overview

The AnimatedWieldableObstacleHandler behaviour sets a bool animator parameter on the player character's weapon animator whenever the weapon is obstructed by an obstacle in the world. This can be used to pull the weapon back. It will also block the use of the weapon as long as it is obstructed.

Inspector



Properties

Name	Type	Description
Cast Distance	Float	The distance in front of the weapon to check for obstacles.
Cast Type	Dropdown	The cast type to use. Options are Raycast and Spherecast .
Cast Source	Dropdown	What to use as the source of the cast. Options are WieldableParent , CharacterAim and Camera .
Layer Mask	LayerMask	What layers are counted as obstacles and make the character move the weapon.
Cast Radius	Float	The radius of the sphere cast (if cast type is set to Spherecast).
Min Blocked Frames	Integer	The minimum number of frames the weapon can be blocked. This prevents the weapon from rapidly switching between blocked and not.
Resume Aim	Boolean	If the weapon is a firearm, and it was aimed when the obstacle was hit, should it resume aiming once the obstacle is cleared.
Obstructed Key	String	The name of the animator parameter to set when the weapon's obstructed state changes.
Unblock Delay	Float	The amount of time the weapon should remain blocked from use after the obstruction is removed. This allows the weapon to move back to its idle or aimed pose before allowing use again.

See Also

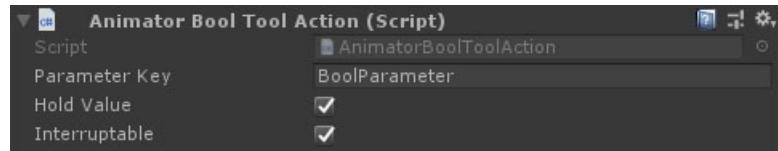
[Modular Firearms](#)

AnimatorBoolToolAction MonoBehaviour

Overview

The AnimatorBoolToolAction sets a bool parameter on the tool's [Animator](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Parameter Key	String	The name of the animator bool parameter to set.
Hold Value	Boolean	The value to set the parameter to while the trigger is held. The value will be reset on release.
Interruptable	Boolean	If the tool is interrupted (reload) the bool paramater will be reset.

See Also

[Wieldable Tools](#)

[Unity Animator](#)

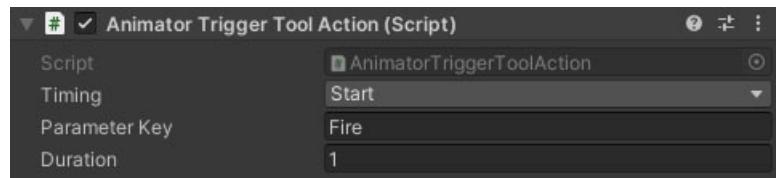
[Unity AnimatorController](#)

AnimatorTriggerToolAction MonoBehaviour

Overview

The AnimatorTriggerToolAction sets a trigger parameter on the tool's [Animator](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the trigger fire.
Parameter Key	String	The name of the animator trigger parameter to fire.
Duration	Float	The tool will be prevented from re-triggering or deselecting for this duration.

See Also

[Wieldable Tools](#)

[Unity Animator](#)

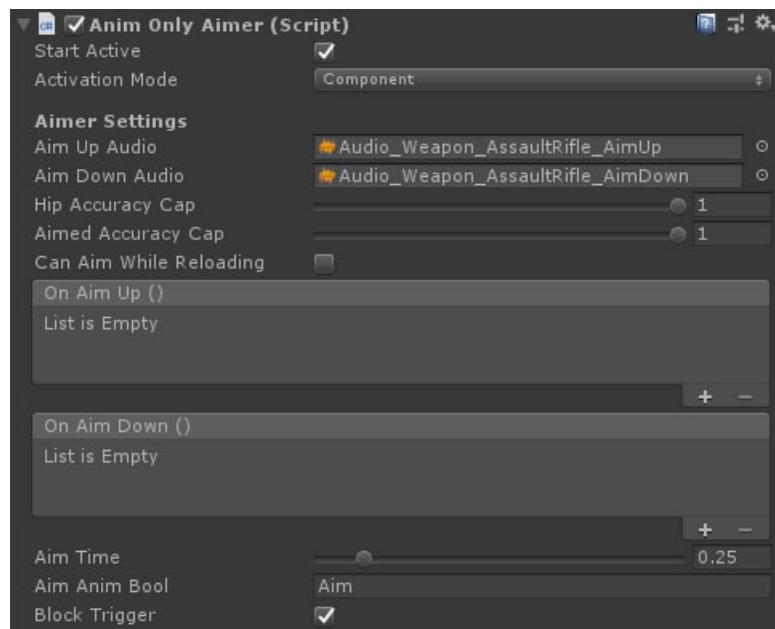
[Unity AnimatorController](#)

AnimOnlyAimer MonoBehaviour

Overview

The AnimOnlyAimer module triggers an animation, without moving the camera or weapon, or changing the crosshair. Its main use is when using the modular firearm system with AI.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.

NAME	TYPE	DESCRIPTION
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.

See Also

[Modular Firearms](#)

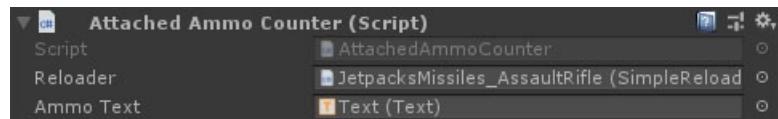
[First Person Camera](#)

AttachedAmmoCounter MonoBehaviour

Overview

The AttachedAmmoCounter behaviour is added to a world space canvas and used to track the ammo in a gun's magazine.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Reloader	Float	The reloader module to track the current magazine from.
Ammo Text	[Text][unity-uitext]	The text output for the current magazine count.

See Also

[Modular Firearms](#)

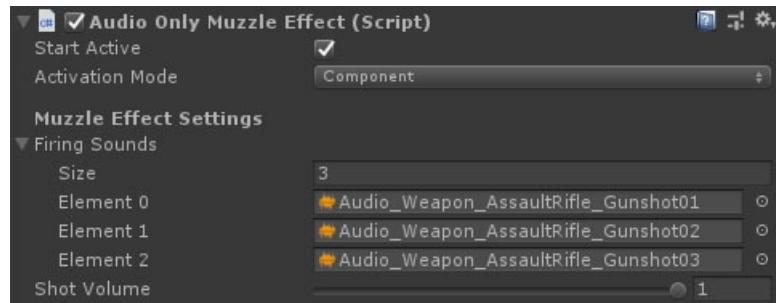
[StaminaSystem](#)

AudioOnlyMuzzleEffect MonoBehaviour

Overview

The AudioOnlyMuzzleEffect module plays a sound when the firearm is fired, but has no visual effect. This can be useful for weapons like crossbows.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

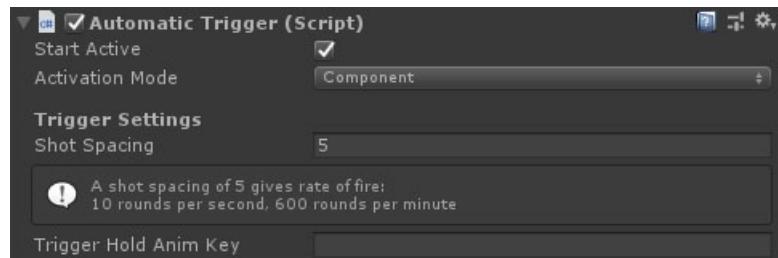
[Modular Firearms](#)

AutomaticTrigger MonoBehaviour

Overview

The AutomaticTrigger module fires continuously

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Shot Spacing	Int	How many fixed update frames between shots. The info box below this show the rate of fire based on the shot spacing.
Trigger Hold Anim Key	String	The AnimatorController bool property key to set while the trigger is pressed.

See Also

[Modular Firearms](#)

[Unity Animator](#)

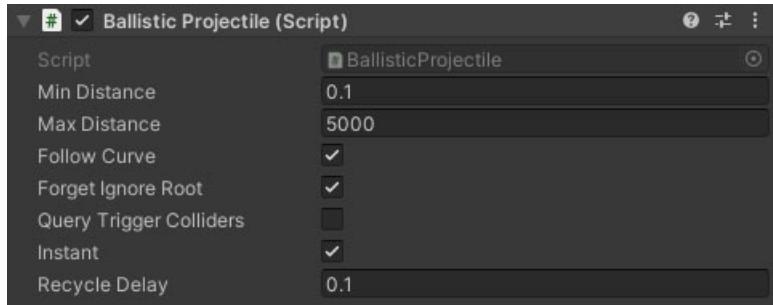
[Unity AnimatorController](#)

BallisticProjectile MonoBehaviour

Overview

The BallisticProjectile is a basic projectile that follows gravity. It will be given an ammo effect by the firearm that it uses when it impacts.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Max Distance	Float	The maximum distance, after the projectile will disappear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Query Trigger Colliders	Boolean	Should the shot be tested against trigger colliders.
Instant	Boolean	This will cause the projectile logic to be calculated instantly on firing. This will be more responsive, but bullet trails, etc may start far ahead of the gun depending on muzzle speed. Works best in first person.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).

See Also

[Modular Firearms](#)

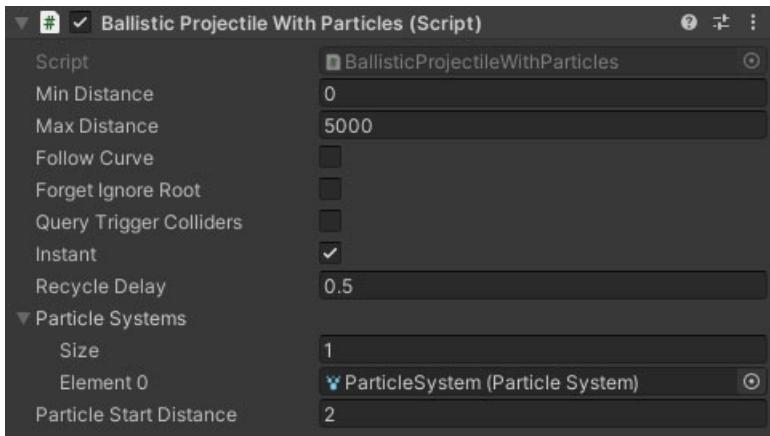
[BallisticShooter](#)

BallisticProjectileWithParticles MonoBehaviour

Overview

The BallisticProjectileWithParticles behaviour is a basic projectile that follows gravity. It will be given an ammo effect by the firearm that it uses when it impacts. This projectile uses a particle system to leave a bullet trail which is started a short distance after firing, and ended as soon as the projectile hits its target.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Max Distance	Float	The maximum distance, after the projectile will disappear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Query Trigger Colliders	Boolean	Should the shot be tested against trigger colliders.
Instant	Boolean	This will cause the projectile logic to be calculated instantly on firing. This will be more responsive, but bullet trails, etc may start far ahead of the gun depending on muzzle speed. Works best in first person.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).
Particle Systems	Particle System Array	The particle systems to play.
Particle Start Distance	Float	The distance the projectile must travel before it starts emitting particles.

See Also

[Modular Firearms](#)

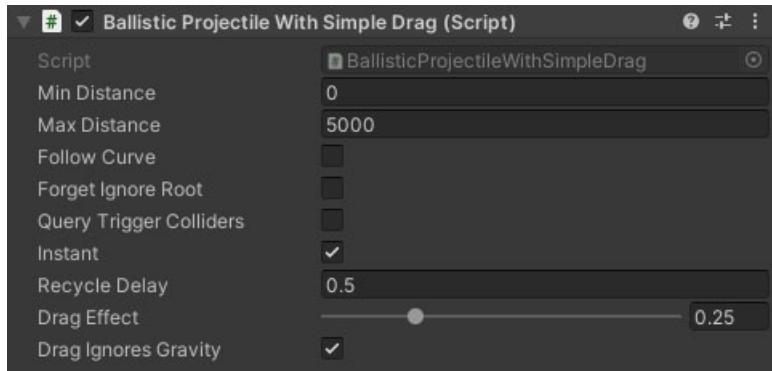
[BallisticShooter](#)

BallisticProjectileWithSimpleDrag MonoBehaviour

Overview

The BallisticProjectileWithSimpleDrag adds a simple air resistance / drag option to the [BallisticProjectile](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance, before the projectile will appear.
Max Distance	Float	The maximum distance, after the projectile will disappear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Query Trigger Colliders	Boolean	Should the shot be tested against trigger colliders.
Instant	Boolean	This will cause the projectile logic to be calculated instantly on firing. This will be more responsive, but bullet trails, etc may start far ahead of the gun depending on muzzle speed. Works best in first person.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).
Drag Effect	Float	The strength of the drag on the projectile (uses a simplified multiplier).
Drag Ignores Gravity	Float	If true, then drag will only be applied to the vertical velocity while the projectile is climbing.

See Also

[Modular Firearms](#)

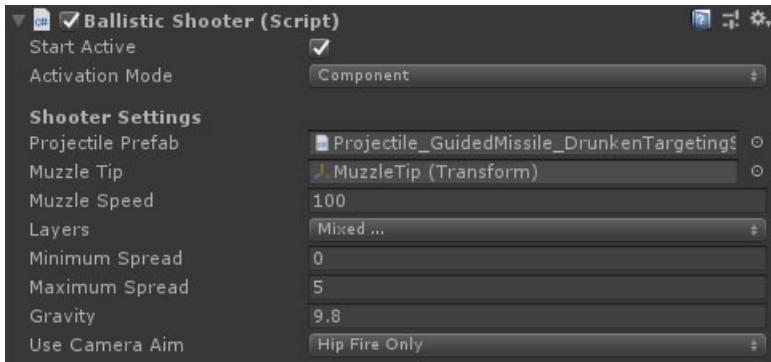
[BallisticShooter](#)

BallisticShooter MonoBehaviour

Overview

The BallisticShooter module spawns an [BallisticProjectile](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The position and direction the projectile is spawned.
Muzzle Speed	Float	The speed of the projectile.
Layers	LayerMask	The physics collision layers the shot can hit.
Minimum Spread	Float	The minimum accuracy spread (in degrees) of the weapon when accuracy is at 1.
Maximum Spread	Float	The maximum accuracy spread (in degrees) of the weapon when accuracy is at 0.
Gravity	Float	The gravity for the projectile.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .

See Also

[Modular Firearms](#)

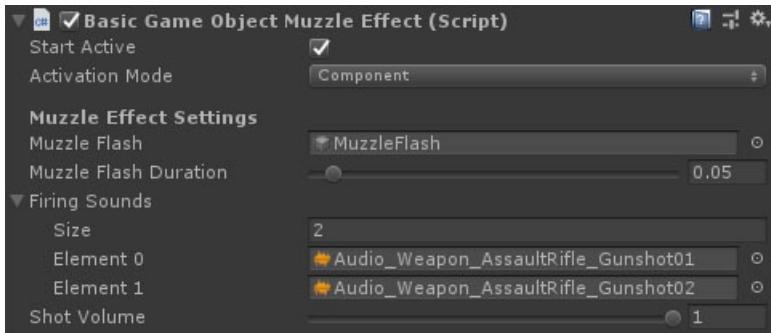
BallisticProjectile

BasicGameObjectMuzzleEffect MonoBehaviour

Overview

The BasicGameObjectMuzzleEffect module simple enables a specified game object, and then disables it again after a brief wait.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Muzzle Flash	GameObject	The muzzle flash game object.
Muzzle Flash Duration	Float	The duration the flash should remain visible in seconds.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

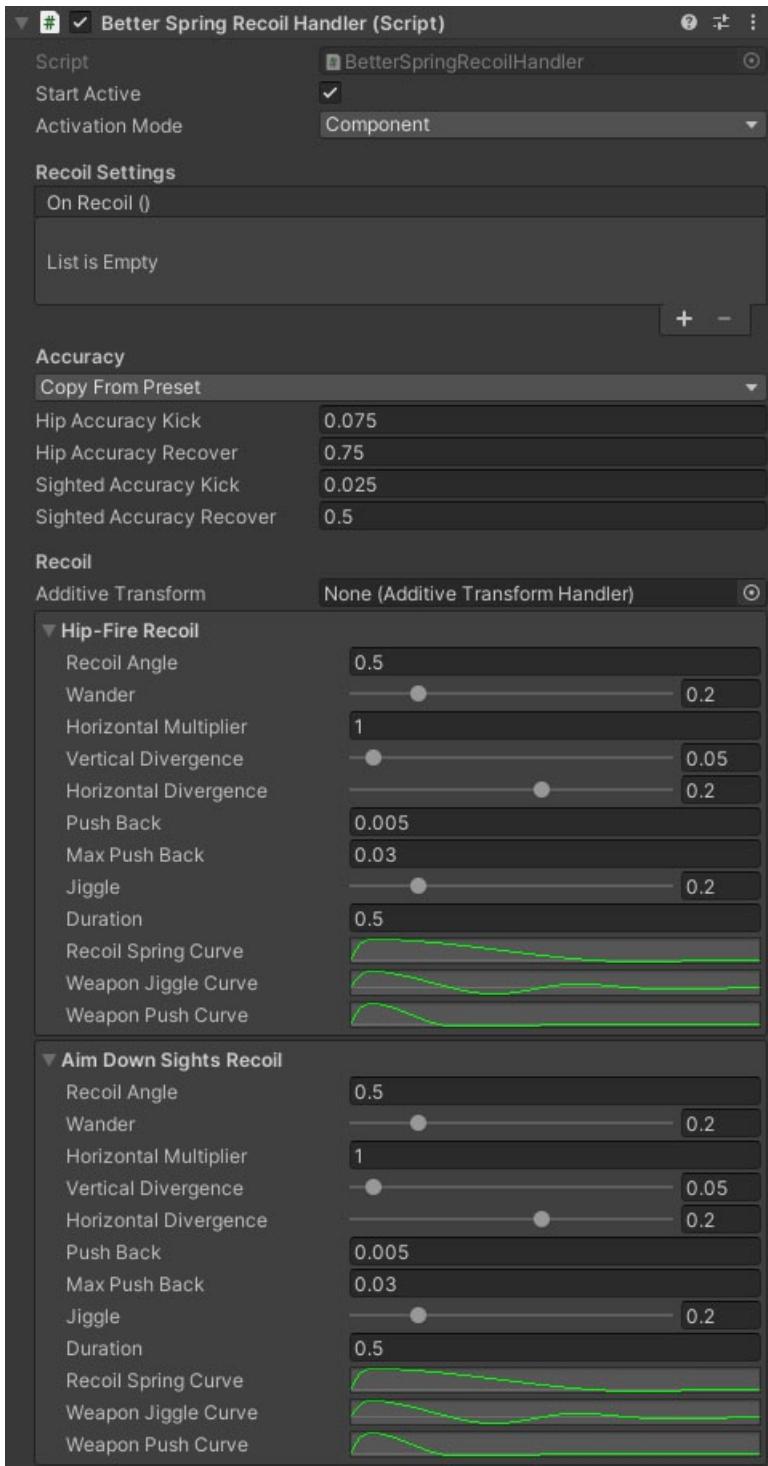
[Modular Firearms](#)

BetterSpringRecoilHandler MonoBehaviour

Overview

The BetterSpringRecoilHandler module controls a [FirearmRecoilEffect](#) and a [CharacterRecoilEffect](#) behaviour to apply procedural recoil animation to the weapon and player character.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.

NAME	TYPE	DESCRIPTION
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
On Recoil	[UnityEvent][unity-event]	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).
Additive Transform	AdditiveTransformHandler	The additive transform handler which manages the firearm spring effects. If this is not set then it will be grabbed from the child hierarchy in awake.

Recoil Profiles

The BetterSpringRecoilHandler module has a recoil profile for aiming down sights and for hip fire. Each of these have the following properties:

NAME	TYPE	DESCRIPTION
RecoilAngle	Float	The angle to rotate the gun on recoil (before modifiers).
Wander	Float	The maximum amount of side to side movement during a recoil. At wander = 1, the gun can rotate anything up to 90 degrees from up. At 0, the gun will only rotate upwards.
HorizontalMultiplier	Float	A multiplier applied to any horizontal rotation of the recoil.
VerticalDivergence	Float	The split between head and weapon recoil. At 0, the head/body will rotate, and the firearm will move with it. At 1, the firearm will rotate and the head/body will remain still.
HorizontalDivergence	Float	The split between head and weapon recoil. At 0, the head/body will rotate, and the firearm will move with it. At 1, the firearm will rotate and the head/body will remain still.
PushBack	Float	The distance the firearm will be pushed backwards each shot.
MaxPushBack	Float	The maximum distance the firearm can be pushed back.
Jiggle	Float	The amount of jiggle (spring rotation around the z-axis) to be applied to the firearm.
Duration	Float	The time taken for the head and firearm to return to their pre-recoil state.

NAME	TYPE	DESCRIPTION
RecoilSpringCurve	AnimationCurve	The animation curve used to drive the amount of recoil rotation over the duration of the recoil effect.
WeaponJiggleCurve	AnimationCurve	The animation curve used to drive the firearm's jiggle rotation over the duration of the recoil effect.
WeaponPushCurve	AnimationCurve	The animation curve used to drive the firearm's push-back over the duration of the recoil effect.

See Also

[Modular Firearms](#)

[FirearmRecoilEffect](#)

[CharacterRecoilEffect](#)

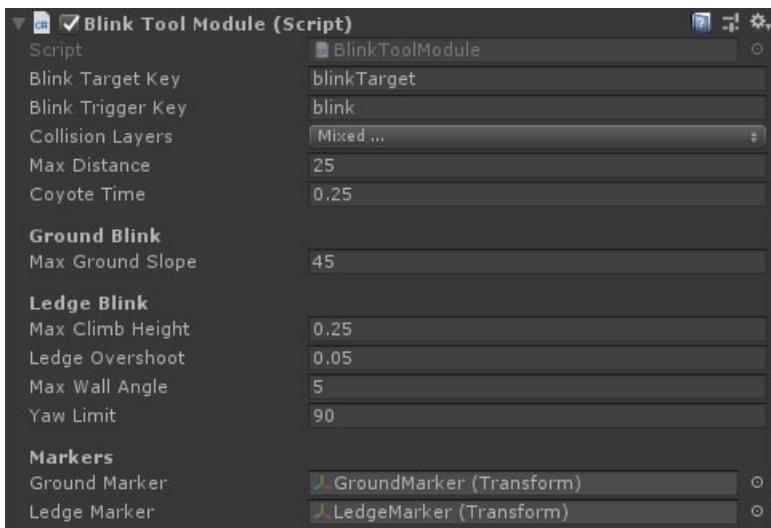
[Additive Transforms and Effects](#)

BlinkToolModule MonoBehaviour

Overview

The BlinkToolModule behaviour is used to model an Arkane style blink (Dishonoured, Deathloop, Prey) as a tool action. Aim at an area of flat ground or a ledge, hold the fire button until the markers are shown. Release to blink to that location. This is intended to be used alongside the [MoveToPointState](#) motion graph state.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Blink Target Key	String	The key to a vector parameter in the wielder's motion graph that will be set with the target position.
Blink Trigger Key	String	The key to a trigger parameter in the wielder's motion graph that will be set when the blink is performed (trigger released on a valid target location).
Collision Layers	Layer Mask	The physics layers the tool can blink onto.
Max Distance	Float	The maximum distance that the character can blink.
Coyote Time	Float	The last valid blink target will be stored this long and used. Makes the target selection more forgiving.
Max Ground Slope	Float	The maximum slope that a ground plane can have for you to blink onto it.
Max Climb Height	Float	The maximum distance down a wall you can look and it will blink to the top ledge.
Ledge Overshoot	Float	If looking at the top of a wall, this is the distance past the edge to overshoot when blinking onto the ledge.

NAME	TYPE	DESCRIPTION
Max Wall Angle	Float	The maximum angle away from vertical that the blink tool can register as a wall with a ledge.
Yaw Limit	Float	An angle range centered on the wall normal that you can blink within. At 180 degrees you can blink to any ledge. At 0 degrees you would have to be looking perfectly flat on to the wall to blink to its ledge.
Ground Marker	Transform	An object in the tool's hierarchy to use as the ground position marker for the blink target (the object will be moved out of the tool's hierarchy).
Ledge Marker	Transform	An object in the tool's hierarchy to use as the ledge position marker for the blink target (the object will be moved out of the tool's hierarchy).

See Also

[Wieldable Tools](#)

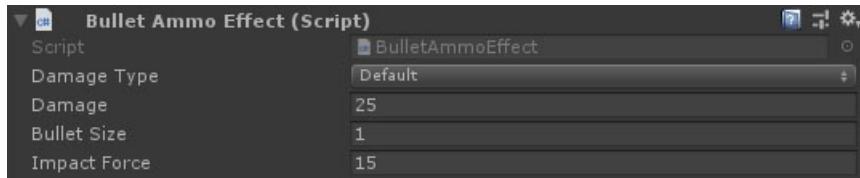
[Motion Graph Parameters And Data](#)

BulletAmmoEffect MonoBehaviour

Overview

The BulletAmmoEffect module uses the [surfaces](#) system to show impact effects, and imparts damage and force to whatever it hits.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Damage	Float	The damage the bullet does.
Bullet Size	Float	The size of the bullet. Used to size decals.
Impact Force	Float	The force to be imparted onto the hit object. Requires either a Rigidbody or an impact handler.

See Also

[Modular Firearms](#)

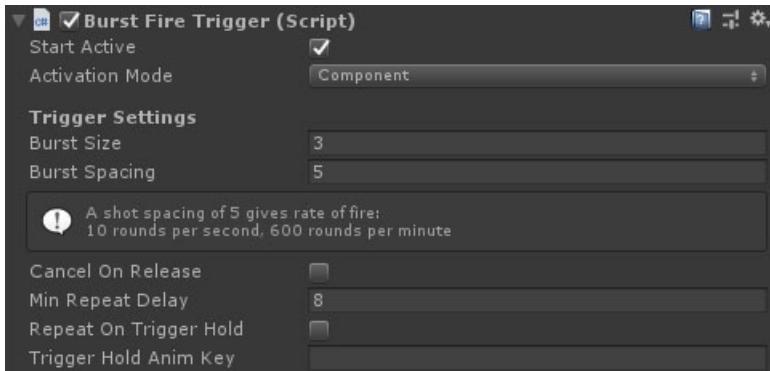
[Surfaces](#)

BurstFireTrigger MonoBehaviour

Overview

The BurstFireTrigger module fires a set number of shots in rapid succession.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Burst Size	Int	The number of shots in a burst.
Burst Spacing	Int	Cooldown between shots in a burst (number of fixed update frames). The info box below this show the rate of fire based on the shot spacing.
Repeat Delay	Int	How many fixed update frames from last shot of a burst before starting another (0 = requires fresh trigger press).
Cancel On Release	Boolean	Is the burst cancelled when the trigger is released.

See Also

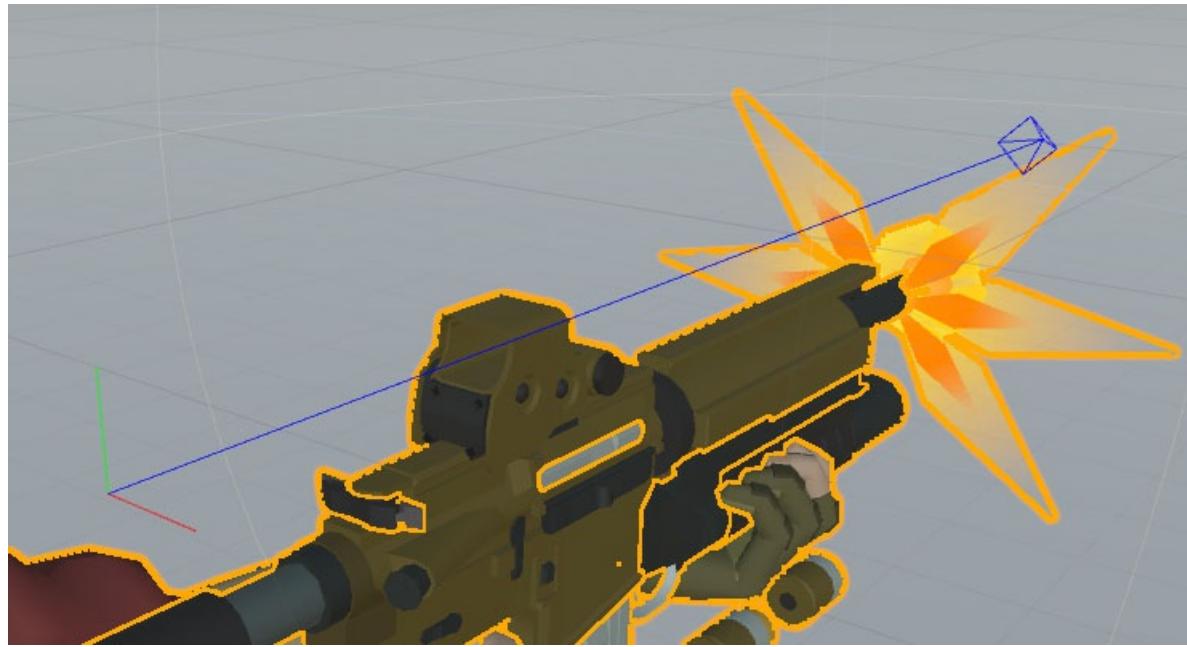
[Modular Firearms](#)

CameraConstraintsAimer MonoBehaviour

Overview

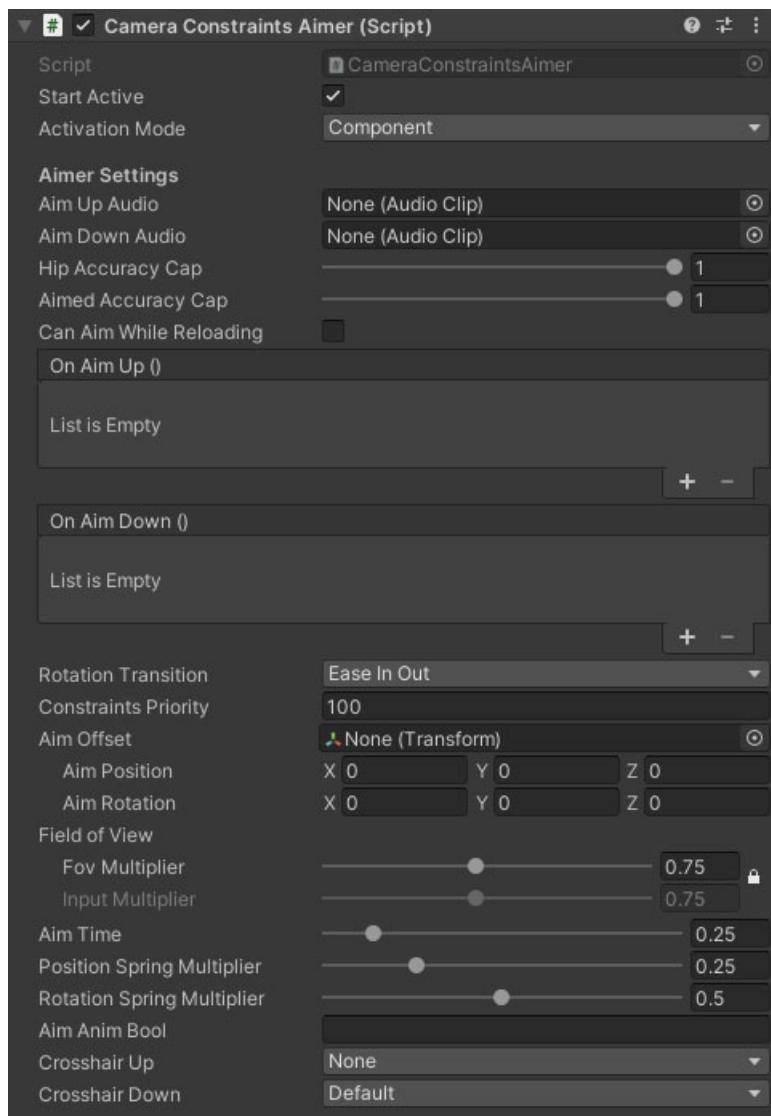
The CameraConstraintsAimer module is intended for weapons that are used with characters with a full first person body. It connects to the [FirstPersonCameraTransformConstraints](#) component in the character hierarchy and sets a camera constraint based on the aim offsets. The player camera will then align to the weapon instead of the character's head bone.

In The Scene View



With a gameobject selected that uses a CameraConstraintsAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.

Name	Type	Description
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Offset	Transform	A target aim transform. The weapon will be moved to align this transform to the camera when aiming down sights. Offsets are calculated on Awake, so moving this transform at after this point has no effect.
Aim Position	Vector3	The offset for the weapon transform to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Aim Rotation	Vector3	An euler angle offset for the weapon to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Input Multiplier	Float	A multiplier for the camera input when aiming down sights. By default, this value is locked to the FoV multiplier. Clicking the lock icon next to the 2 multipliers allows you to modify it separately for situations such as render texture scopes.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.
Rotation Transition	Dropdown	The transition easing to apply to rotation when entering / exiting ADS.
Constraints Priority	Integer	The priority to use for this camera constraint. The camera will align to the highest priority active constraint. You can usually leave this as default.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

[Modular Firearms](#)

[First Person Camera](#)

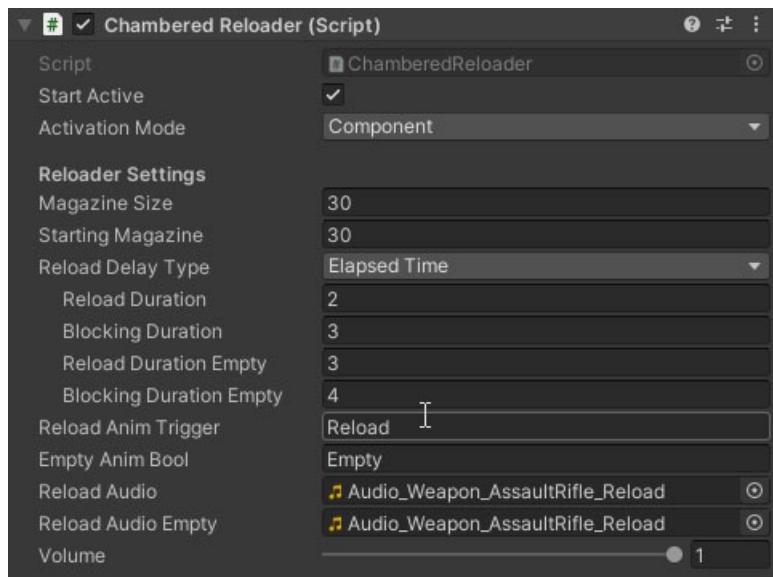
[FirstPersonCameraTransformConstraints](#)

ChamberedReloader MonoBehaviour

Overview

The ChamberedReloader behaviour triggers a reload animation and waits until it completes before taking ammo from the firearm's ammo module and adding it to the magazine. The ChamberedReloader will set a bool parameter in the firearm's [AnimatorController][unity-animatorcontroller] so that it can change the animations it plays, and it can have a different wait duration and audio clip to play if reloading from an empty magazine compared to with a round in the chamber.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Reload Delay Type	Dropdown	The delay type between starting and completing a reload. The options are None , Elapsed Time , External Trigger .
Reload Duration	Float	The time taken to reload.
Reload Duration Empty	Float	The time taken to reload if reloading from empty.

NAME	TYPE	DESCRIPTION
Blocking Duration	Float	The time taken before you can use the weapon again after the reload starts.
Blocking Duration Empty	Float	The time taken before you can use the weapon again after the reload from empty starts.
Reload Anim Trigger	String	The [AnimatorController][unity-animatorcontroller] trigger parameter key for the reload animation.
Empty Anim Bool	String	The key to an [AnimatorController][unity-animatorcontroller] bool parameter to set when the weapon is empty.
Reload Audio	AudioClip	The audio clip to play while reloading.
Reload Audio Empty	AudioClip	The audio clip to play if reloading from empty.

See Also

[Modular Firearms](#)

[FirearmAnimEventsHandler](#)

[\[Unity Animator\]\[unity-animator\]](#)

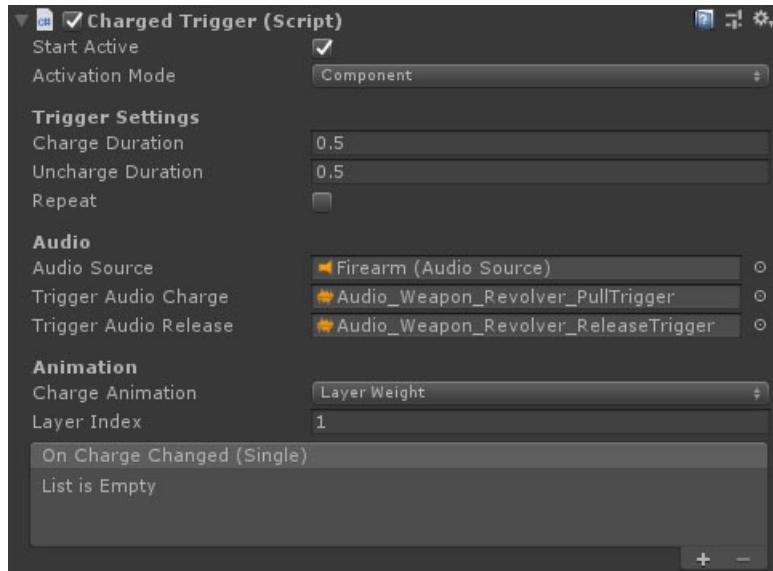
[\[Unity AnimatorController\]\[unity-animatorcontroller\]](#)

ChargedTrigger MonoBehaviour

Overview

The ChargedTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	<audiosource></audiosource>	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

Name	Type	Description
Charge Animation	Dropdown	How should the charge be animated. LayerWeight blends in a layer in the weapon's animator as charge increases. FloatParameter sets a float parameter value on the animator. EventsOnly fires an event when the charge changes, but does not affect the attached animator.
Layer Index	Integer	The animator layer index to blend in for the charge effect. This option will only be visible if "Charge Animation" is set to LayerWeight .
Charge Anim Key	String	The float animator parameter key to set when the trigger charge changes. This option will only be visible if "Charge Animation" is set to FloatParameter .
On Charge Changed	UnityEvent	An event fired each frame the charge changes.

See Also

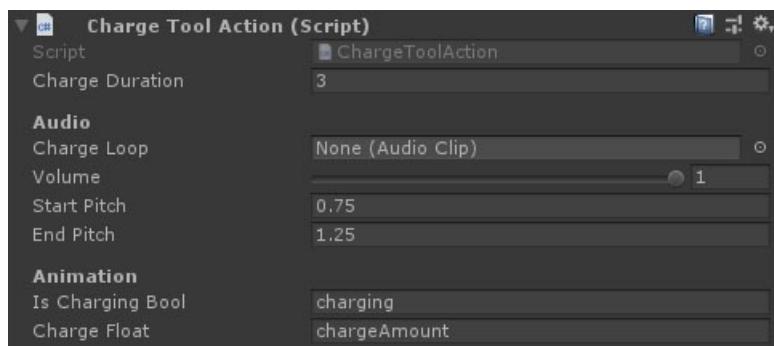
[Modular Firearms](#)

ChargeToolAction MonoBehaviour

Overview

The ChargeToolAction behaviour is used to add a charge up effect to a [wieldable tool](#). Charging all the way to full will interrupt the tool and send the `success = true` signal to the end actions. Releasing the trigger early will send the `success = false` signal.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Charge Duration	Float	The time it takes to reach full charge.
Charge Loop	AudioClip	The audio to play while charging.
Volume	Float	The volume of the charging audio loop.
Start Pitch	Float	The pitch of the audio loop at the start of the charge.
End Pitch	Float	The pitch of the audio loop when charge hits 100%.
Is Charging Bool	String	A bool parameter in the tool's animator that should be set while charging.
Charge Float	String	A float parameter in the tool's animator used to show charge progress (0-1).

See Also

[Wieldable Tools](#)

[Unity Animator](#)

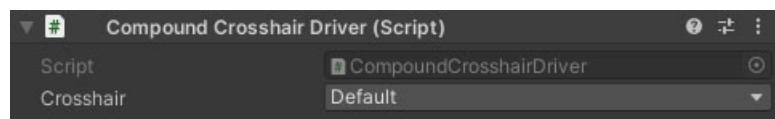
[Unity AnimatorController](#)

CompoundCrosshairDriver MonoBehaviour

Overview

The CompoundCrosshairDriver behaviour is used when you have multiple child weapons that each impact the crosshair accuracy. An example is a dual wield pair of machine guns where you want each weapon to affect the spread of the crosshair as they fire. Placing this behaviour on the root of the weapon will mean the HUD crosshair picks this up and uses the combined crosshair. If you also have a weapon behaviour such as a [ModularFirearm](#) on the root then make sure to move this behaviour above it in the inspector.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Crosshair	FpsCrosshair	The default crosshair to show.

See Also

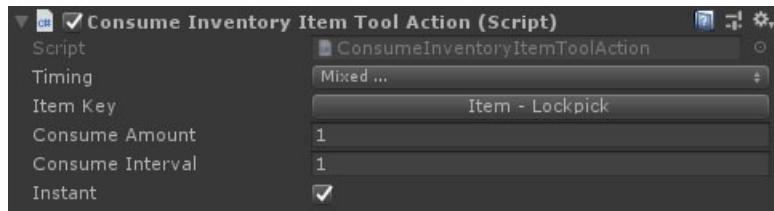
[Modular Firearms](#)

ConsumeInventoryItemToolAction MonoBehaviour

Overview

The ConsumeInventoryItemToolAction behaviour is used to add the effect of using an inventory item to a [wieldable tool](#) either on start or end, or continuously over time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the item be consumed.
Item Key	ID Picker	The inventory ID of the object to consume. Clicking this button will open the inventory database item picker.
Consume Amount	Integer	How many of the item should be consumed when triggered or ticked.
Consume Interval	Integer	The items should be consumed every nth fixed update frame in continuous mode.
Instant	Boolean	Should the item be consumed on the first frame of the continuous action, or should it wait for the first interval to elapse.

See Also

[Wieldable Tools](#)

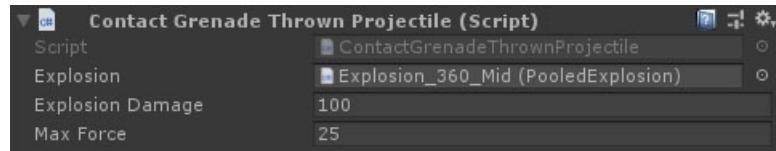
[Inventory](#)

ContactGrenadeThrownProjectile MonoBehaviour

Overview

The ContactGrenadeThrownProjectile behaviour is used to create grenades which explode the moment they hit a target.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn on contact with an object.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody] [unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.

See Also

[Explosions](#)

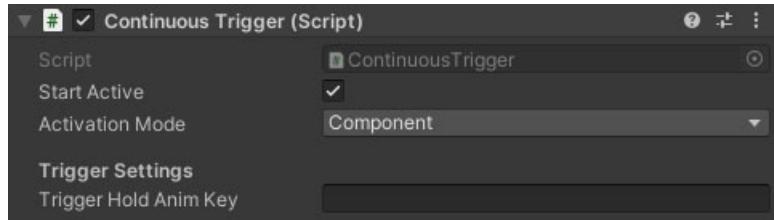
[Health and Damage](#)

ContinuousTrigger MonoBehaviour

Overview

The ContinuousTrigger module fires each frame. This is useful for modelling weapons such as flamethrowers, or beam weapons that fire until you release the trigger again.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Trigger Hold Anim Key	String	The AnimatorController bool property key to set while the trigger is pressed.

See Also

[Modular Firearms](#)

[Unity Animator](#)

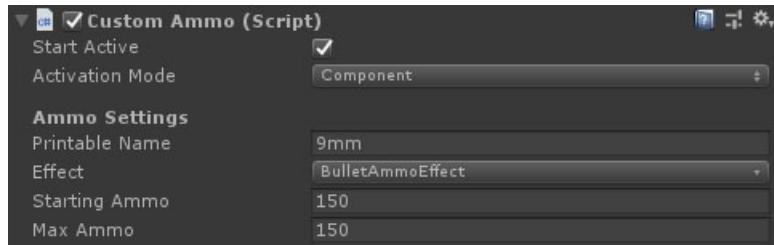
[Unity AnimatorController](#)

CustomAmmo MonoBehaviour

Overview

The CustomAmmo module is a unique ammo for the firearm it is attached to.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Starting Ammo	Int	The amount of ammo the weapon starts with.
Max Ammo	Int	The maximum amount of ammo the weapon can carry.

See Also

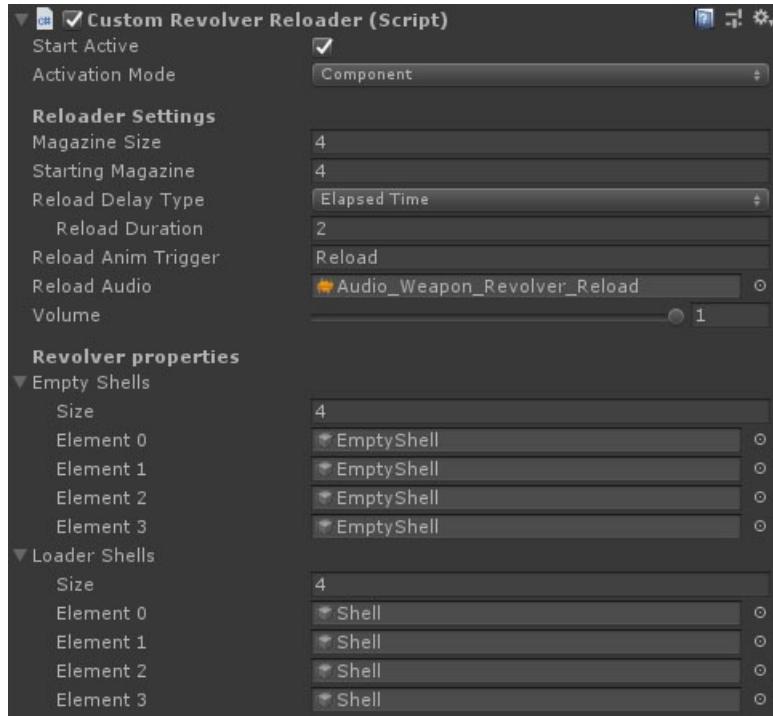
[Modular Firearms](#)

CustomRevolverReloader MonoBehaviour

Overview

The CustomRevolverReloader module is adapted from the [SimpleReloader](#). It adds functionality to make the spent and reloaded rounds visible and the visible count correct based on how many bullets are loaded into the gun.

Inspector



Properties

The CustomRevolverReloader module inherits most of its properties from the [SimpleReloader](#). It adds the following properties:

Name	Type	Description
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Empty Shells	GameObject Array	The empty shells in the revolver cylinder.
Loader Shells	GameObject Array	The unused shells in the revolver speed loader.

See Also

Modular Firearms

[Unity Animator][unity-animator]

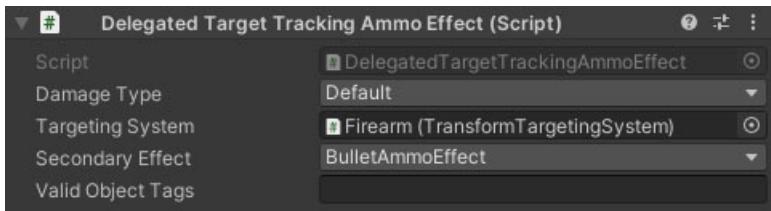
[Unity AnimatorController][unity-animatorcontroller]

DelegatedTargetTrackingAmmoEffect MonoBehaviour

Overview

The DelegatedTargetTrackingAmmoEffect behaviour is used to tag targets for [guided projectiles](#) to home in on. The tagged target is assigned to the [TransformTargetingSystem](#) assigned to its **Targeting System** property. Projectiles must use a [TargetingSystemTracker](#) component.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	Dropdown	The type of damage the impact counts as. This is a common property for all ammo effects and actually has no effect in this case.
TargetingSystem	TargetingSystemTracker	The targeting system to assign the tagged transform to.
Secondary Effect	AmmoEffect	An optional secondary ammo effect to allow the tracking bullet to deal damage, etc.
Valid Object Tag	Float	Only objects with this tag can be targeted. If this is empty then any hit object will be assigned as a target.

See Also

[Modular Firearms](#)

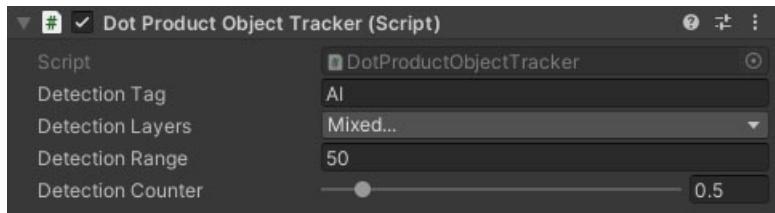
[Hitscan vs Projectiles](#)

DotProductObjectTracker MonoBehaviour

Overview

The DotProductObjectTracker behaviour is a guided projectile tracker that tracks the object that is closest to its forward axis from its point of view (as it requires the least course correction to hit).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Detection Tag	String	The object tag to home in on.
Detection Layers	LayerMask	The layers to check for colliders on.
Detection Range	Float	The max distance for targets to home in on.
Detection Counter	Float	The time between searching for targets.

See Also

[Modular Firearms](#)

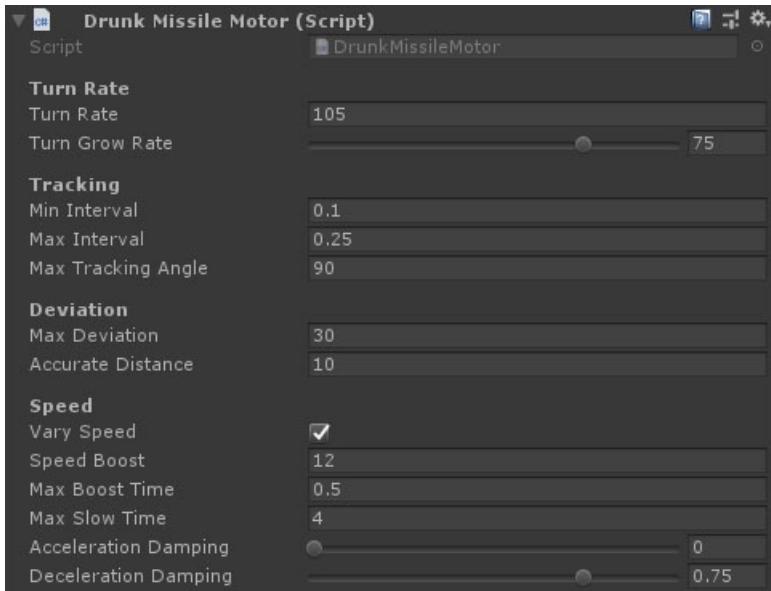
[Hitscan vs Projectiles](#)

DrunkMissileMotor MonoBehaviour

Overview

The DrunkMissileMotor behaviour is added to a guided projectile to make it follow an erratic path towards its target.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Turn Rate	Float	The turn rate of the projectile (degrees per second).
Turn Grow Rate	Float	An increase to the turn rate based on elapsed time in flight (base turn rate + turn grow rate * elapsed time).
Min Interval	Float	The minimum amount of time between tracking ticks (when the projectile updates its target location).
Max Interval	Float	The maximum amount of time between tracking ticks (when the projectile updates its target location).
Max Tracking Angle	Float	The maximum angle off the forwards direction that the tracker can swivel to look at the target each tick.
Max Deviation	Float	The maximum deviation from the tracker angle that the projectile will steer to add the drunken effect.
Accurate Distance	Float	The distance from the target below which the projectile will switch to a more consistent tracking (reduces projectiles overshooting).
Vary Speed	Boolean	Should the projectile's speed vary with occasional boosts.
Speed Boost	Float	The target boost speed (added to the base speed of the projectile when fired).
Max Boost Time	Float	The maximum amount of time a boost will last.
Max Slow Time	Float	The maximum amount of time between boosts.

NAME	TYPE	DESCRIPTION
Acceleration Damping	Float	The damping when accelerating up to boost speed.
Deceleration Damping	Float	The damping when decelerating from boost back down to base speed.

See Also

[Modular Firearms](#)

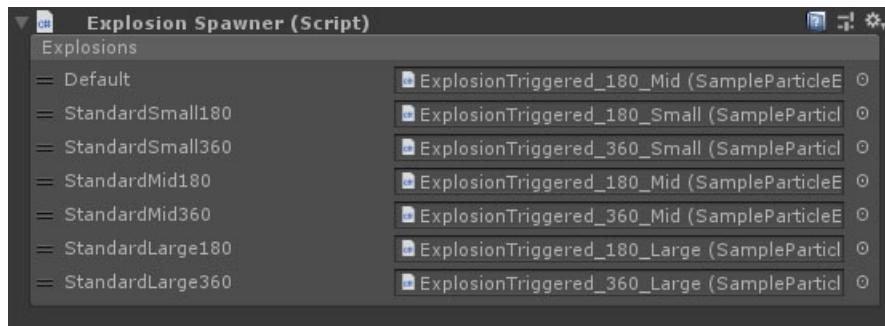
[Hitscan vs Projectiles](#)

ExplosionSpawner MonoBehaviour

Overview

The explosion spawner will spawn an explosion at a point in space on request. This is a demo implementation.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Exploder	Explosion	The explosion to spawn. Must inherit from <code>BaseExplosionBehaviour</code>

See Also

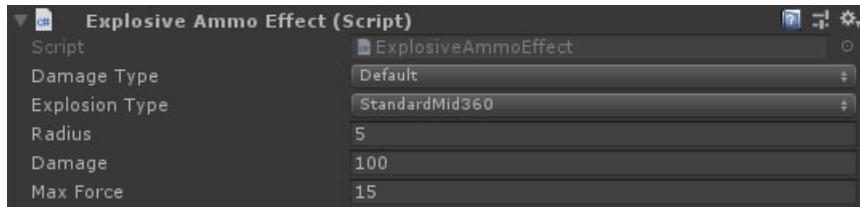
[Explosions](#)

ExplosiveAmmoEffect MonoBehaviour

Overview

The ExplosiveAmmoEffect module spawns an explosion on impact, dealing damage and repelling objects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Explosion Type	ExplosionType	The id of the explosion to spawn.
Radius	Float	The radius of the explosion.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The max force to be imparted onto any objects in the explosion radius. The force falls off as distance from the center increases. Requires either a Rigidbody or an impact handler.

See Also

[Modular Firearms](#)

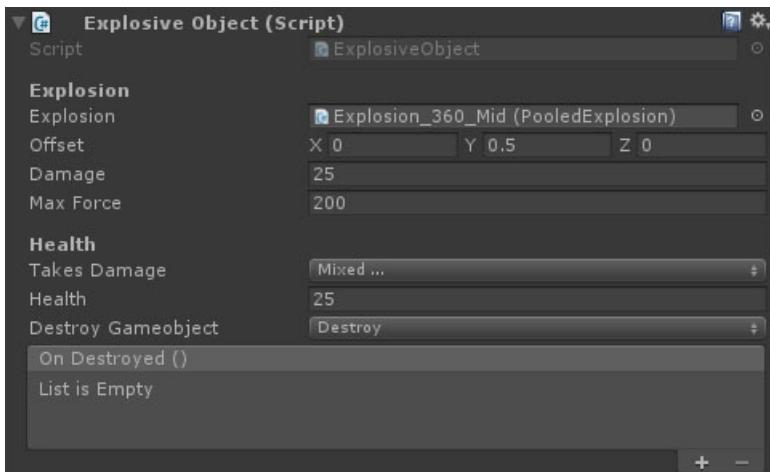
[Explosions](#)

ExplosiveObject MonoBehaviour

Overview

The ExplosiveObject behaviour is used to create objects like barrels that spawn a [PooledExplosion](#) when killed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn.
Offset	Vector3	An offset from the object's origin that the explosion will spawn.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody][unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.
Takes Damage	DamageType	The type of damage the object can take.
Health	Float	The amount of health the object has.
Destroy Gameobject	Dropdown	What to do to the object when it's killed. Available options are: Destroy , Disable , Nothing .
On Destroyed	UnityEvent	An event invoked when the item's health reaches zero

See Also

[Explosions](#)

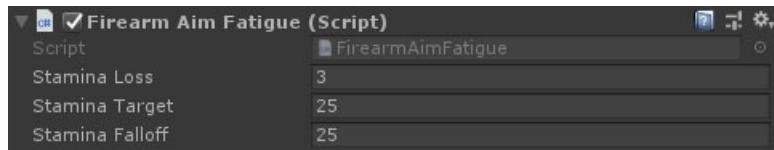
[Health and Damage](#)

FirearmAimFatigue MonoBehaviour

Overview

The FirearmAimFatigue behaviour is used to drain a character's stamina while aiming down sights using the. This requires the character to have a [StaminaSystem](#) behaviour attached.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Stamina Loss	Float	The stamina loss per second when aiming down sights.
Stamina Target	Float	The stamina level to drain down to.
Stamina Falloff	Float	Stamina drain fades when approaching the target, starting at this falloff value above it.

See Also

[Modular Firearms](#)

[StaminaSystem](#)

FirearmAnimEventsHandler MonoBehaviour

Overview

The FirearmAnimEventsHandler behaviour is used to catch events fired from the [Animator](#) attached to this game object. This enables specific triggers to be synced with animation and then feed back to the firearm.

The currently captured events are:

```
public void WeaponRaised ()  
  
public void FirearmReloadPartial (int count)  
  
public void FirearmReloadComplete ()  
  
public void FirearmEjectShell ()
```

Inspector



Properties

The FirearmAnimEventsHandler behaviour has no properties exposed in the inspector.

See Also

[Modular Firearms](#)

[Unity Animator](#)

[Unity Animation Events](#)

FirearmOverheat MonoBehaviour

Overview

The FirearmOverheat behaviour is used to model a simple heat accumulation in a gun as it shoots. It can also optionally cause the weapon to overheat and lock the trigger until it cools to a certain threshold.

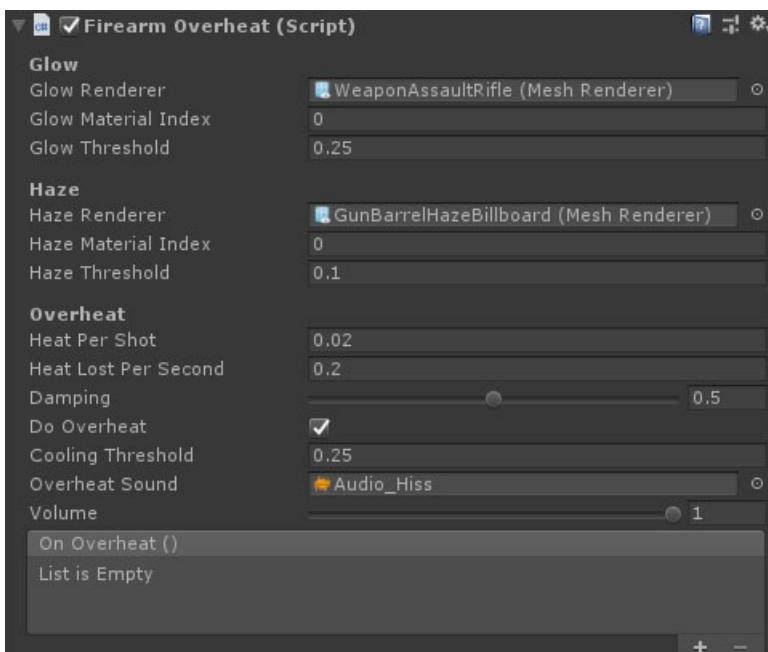


The glow effect requires that one of the NeoFPS glow shaders be used. These are found under the **NeoFPS/Standard** shader path, and are named based on their PBR surface type and glow type: **GlowMetallic** or **GlowSpecular** and **Alpha Masked**, **Distance Masked** and **Position Masked**.

- **Alpha Masked** uses a greyscale glow mask. Glow will be shown wherever the texture is not black, with full strength where it is white.
- **Distance Masked** specifies a 3D point (in object space), and the glow fades out based on the distance from that point.
- **Position Masked** sets start and end points on the object space z-axis, and the glow fades in and out between these points.

The haze effect requires a mesh renderer with a material using the **NeoFPS/Standard/HeatHaze** shader. This shader uses a property block to set the haze strength as the heat increases, and the object is disabled when the heat is zero. A simple haze effect can be created by combining a quad 3D geometry object with a **BillboardOrientation Monobehaviour** to keep it oriented at the first person camera. An example prefab can be found at *NeoFPS/Samples/Shared/Effects/Prefabs/GunBarrelHazeBillboard*.

Inspector



Properties

Name	Type	Description
Glow Renderer	MeshRenderer	The mesh renderer of the glow material.
Glow Material Index	Integer	The index of the glow material in the mesh renderer.
Glow Threshold	Float	The heat level (0 to 1) before the glow starts to appear.
Haze Renderer	MeshRenderer	The mesh renderer of the haze material.
Haze Material Index	Integer	The index of the haze material in the mesh renderer.
Haze Threshold	Float	The heat level (0 to 1) before the haze starts to appear.
Heat Per Shot	Float	The amount of heat to add with each shot of the weapon. When this reaches 1, the gun must cool down before it can fire again.
Heat Lost Per Second	Float	The amount of heat that is dissipated per second. The weapon will never overheat if this is higher than the heat per shot multiplied by rate of fire (rounds per second).
Do Overheat	Float	If true, then once the weapon reaches max heat the weapon will overheat, blocking the trigger until it has cooled down to a set threshold.
Cooling Threshold	Float	Once overheated, the weapon must cool to this heat level before it can fire again.
Overheat Sound	AudioClip	The audio clip to play once max heat is hit and the trigger is blocked.
Volume	Float	The volume to play the overheat sound at.
On Overheat	Float	An event that is fired when the heat hits the max level.

See Also

[Modular Firearms](#)

[BillboardOrientation Monobehaviour](#)

FirearmTransformMatchSetter MonoBehaviour

Overview

The FirearmTransformMatchSetter behaviour is attached to a firearm object. When that firearm is added to a character and equipped it will set the transforms of the [TransformMatcher](#) additive effect attached to the character head. This can be used to sync the character head movement to animations attached to the head, while still allowing additive effects such as recoil and shake.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target	Transform	The target transform to match.
Relative To	Transform	The transform to use when calculating the offset of the target transform.
Weight	Float	The strength of the effect. 1 matches the movement absolutely, while 0 is no movement.

See Also

[Modular Firearms](#)

[TransformMatcher](#)

FirearmWieldableStanceManager MonoBehaviour

Overview

The FirearmWieldableStanceManager behaviour is used to specify poses or stances for a modular firearm. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be temporarily exited when aiming and reloading.

Inspector



Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator .
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
Out Time	Float	The time taken to enter the stance.

See Also

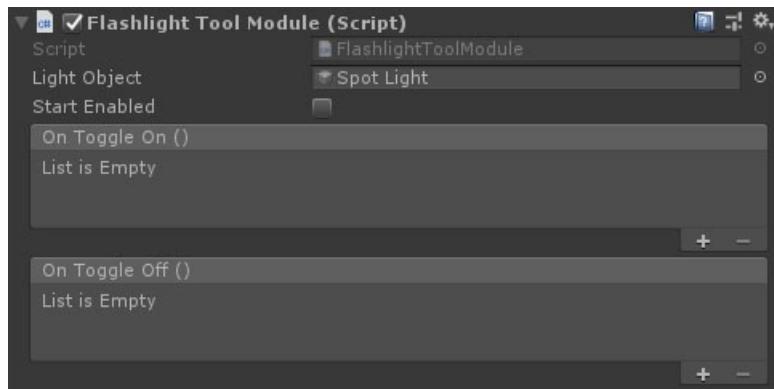
[Modular Firearms](#)

FlashlightToolModule MonoBehaviour

Overview

The FlashlightToolModule behaviour is used to add a toggleable flashlight to [wieldable tools](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
m_LightObject	GameObject	A child object with a light component attached.
m_StartEnabled	Boolean	Should the flashlight be enabled on start.

See Also

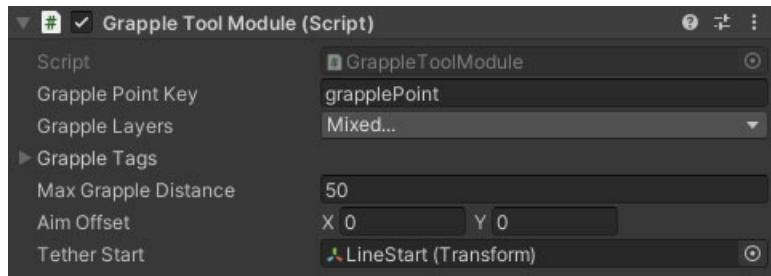
[Wieldable Tools](#)

GrappleToolModule MonoBehaviour

Overview

The GrappleToolModule behaviour is used to set a [vector parameter](#) on the wielding character's [motion graph](#) with an anchor point to grapple to. This can be combined with the [GrappleSwingState](#) motion graph state to add grapple mechanics to the game.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Grapple Point Key	String	The name of the vector parameter on the character's motion graph that you want to set the grapple point to.
Grapple Layers	LayerMask	The physics layers the tool can grapple onto.
Max Grapple Distance	Float	The maximum distance that a grapple can connect.
Aim Offset	Vector	A rotation offset to the aim rotation for the grapple. +x is right, +y is up.
Tether Start	Transform	The transform to use as the start point of the tether line renderer.

See Also

[Modular Firearms](#)

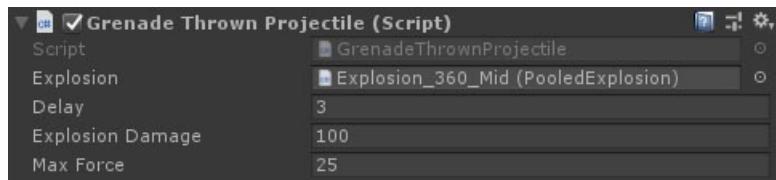
[The Motion Graph](#)

GrenadeThrownProjectile MonoBehaviour

Overview

The GrenadeThrownProjectile behaviour is used to create grenades which explode after a set amount of time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn once the timer expires.
Delay	Float	The delay before exploding.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody] [unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.

See Also

[Explosions](#)

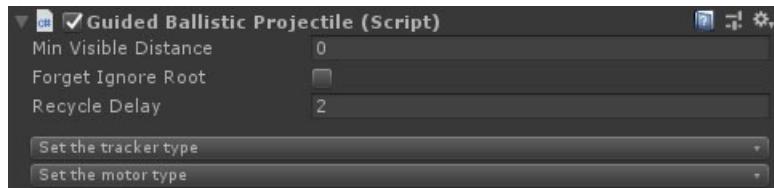
[Health and Damage](#)

GuidedBallisticProjectile MonoBehaviour

Overview

The GuidedBallisticProjectile is a projectile that uses a tracker component and a motor component to control its flight path. The tracker component is used to define what the projectile is tracking, while the motor defines its flight characteristics.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Visible Distance	Float	The minimum distance before the projectile will appear.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).

See Also

[Modular Firearms](#)

[BallisticShooter](#)

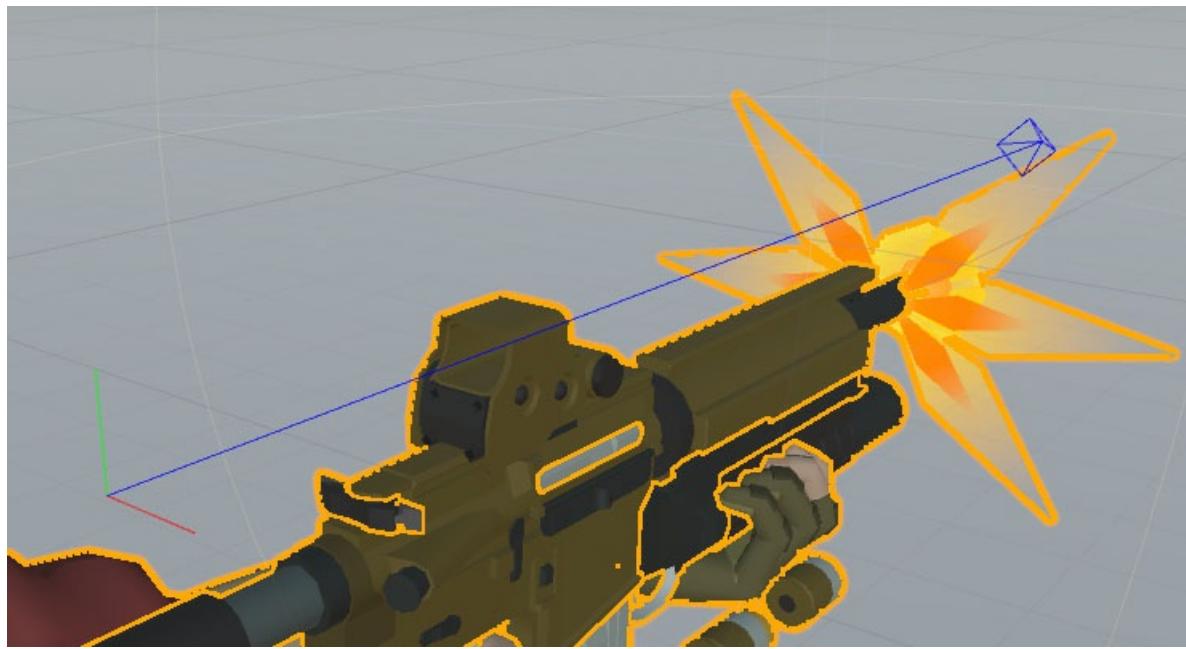
[Hitscan vs Projectiles](#)

HeadMoveAimer MonoBehaviour

Overview

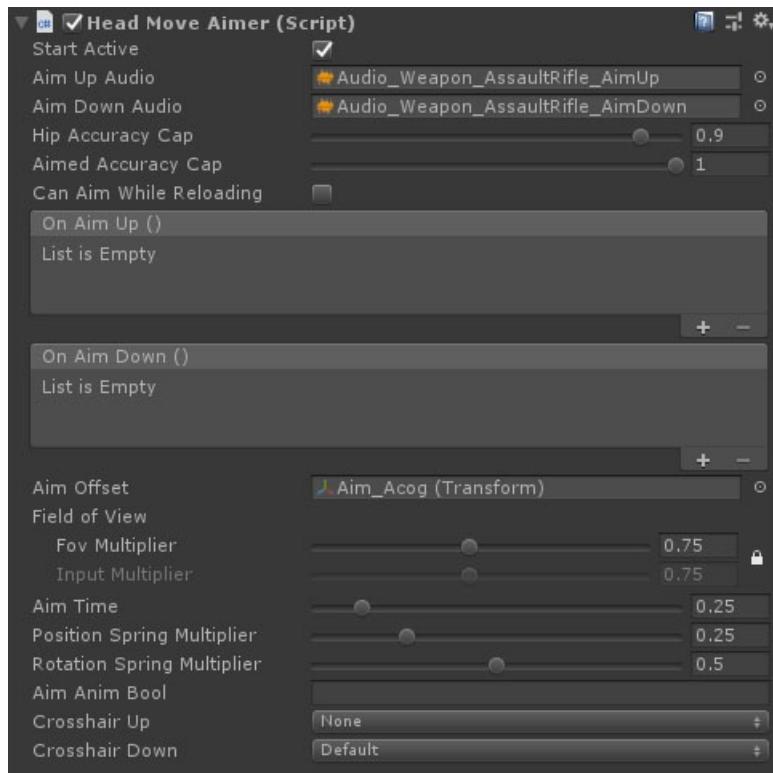
The HeadMoveAimer module moves the character head to align the center of the camera with the weapon sights. This can actually tilt the camera, providing a different effect to the standard [WeaponMoveAimer](#)

In The Scene View



With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Root Transform	Transform	The root transform of the hierarchy (should align with the camera).
Aim Offset	Transform	A target aim transform. The camera will be moved to align it to this transform when aiming down sights. Offsets are calculated on Awake, so moving this transform at after this point has no effect.
Aim Position Offset	Vector3	The offset for the camera transform to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Aim Rotation Offset	Vector3	An euler angle offset for the camera to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Input Multiplier	Float	A multiplier for the camera input when aiming down sights. By default, this value is locked to the FoV multiplier. Clicking the lock icon next to the 2 multipliers allows you to modify it separately for situations such as render texture scopes.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.

NAME	TYPE	DESCRIPTION
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

[Modular Firearms](#)

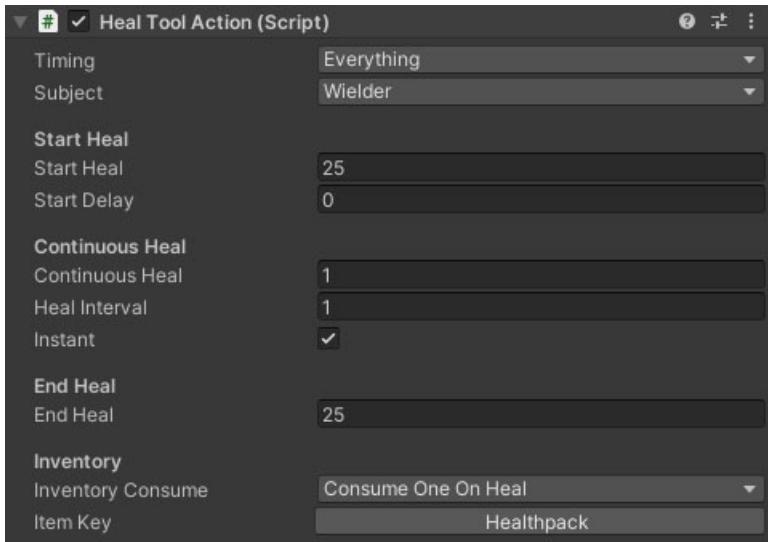
[First Person Camera](#)

HealToolAction MonoBehaviour

Overview

The HealToolAction behaviour adds the ability to [heal](#) the wielder or another subject to a [wieldable tool](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the tool apply the heal.
Subject	Dropdown	Who/what to heal. Wielder means the character using the tool will heal themselves. Target means the tool will be used on the health manager in front of the user.
Target Layers	LayerMask	The physics layers that the tool should check against to get a valid heal subject.
Max Range	Float	The maximum distance in front of the character that the tool should cast to check for valid heal subjects.

Start Heal

NAME	TYPE	DESCRIPTION
Start Heal	Integer	How many points to heal the subject for in the start action.
Start Delay	Integer	How many fixed frames since the start of the action before the heal is applied.

Continuous Heal

NAME	TYPE	DESCRIPTION
Continuous Heal	Integer	How many points to heal the subject for the continuous action.
Heal Interval	Integer	The heal will be applied every nth fixed update tick for continuous heals.

NAME	TYPE	DESCRIPTION
Instant	Boolean	Should the heal be applied on the first frame of the continuous action, or should it wait for the first interval to elapse.

End Heal

NAME	TYPE	DESCRIPTION
End Heal	Integer	How many points to heal the subject for in the end action.

Inventory

NAME	TYPE	DESCRIPTION
Inventory Consume	Dropdown	Should the heal consume an inventory item, and how should that work. Options are NoInventoryItem , ConsumeOneOnHeal and ConsumeHealAmount .
Item Key	Inventory ID Picker	The inventory ID of the object to consume on healing. Clicking this button will show the inventory database browser .

See Also

[Wieldable Tools](#)

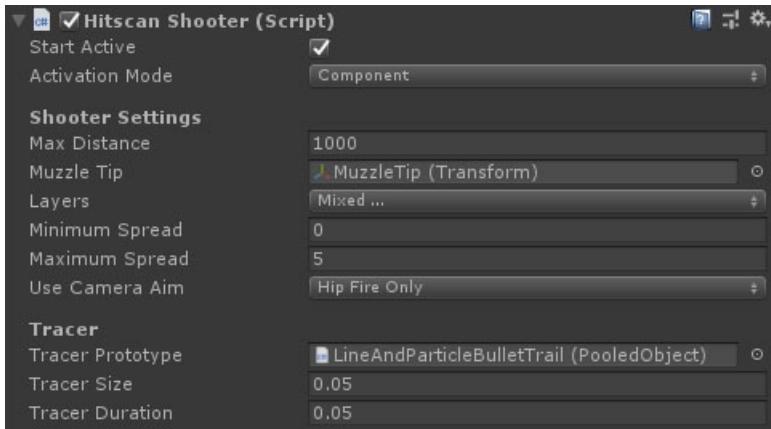
[Health and Damage](#)

HitscanShooter MonoBehaviour

Overview

The HitscanShooter module raycasts directly out from the muzzle of the gun based on the current accuracy.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Minimum Spread	Float	The minimum accuracy spread (in degrees) of the weapon.
Maximum Spread	Float	The maximum accuracy spread (in degrees) of the weapon
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).
Tracer Size	Float	The size (thickness/radius) of the tracer line.

NAME	TYPE	DESCRIPTION
Tracer Duration	Float	How long should the tracer object stay visible.

See Also

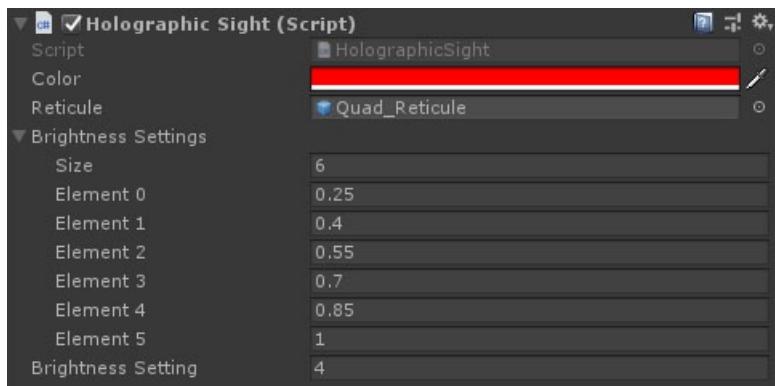
[Modular Firearms](#)

HolographicSight MonoBehaviour

Overview

The HolographicSight behaviour is used to define weapon optics that project a reticule in front of the weapon when viewed through the sight. This is achieved using stencil shaders, and the holographic sight behaviour's main use is to control the colour and the brightness of the reticule.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Color	Color	The base colour of the reticule.
Reticule	GameObject	The game object with the reticule mesh attached. This should be placed directly in front of the weapon at the desired distance.
Brightness Settings	Float Array	A series of brightness values that can be cycled through with the Optics Brightness +/- inputs.
Brightness Setting	int	The index of the starting brightness setting from the above array.

See Also

[Modular Firearms](#)

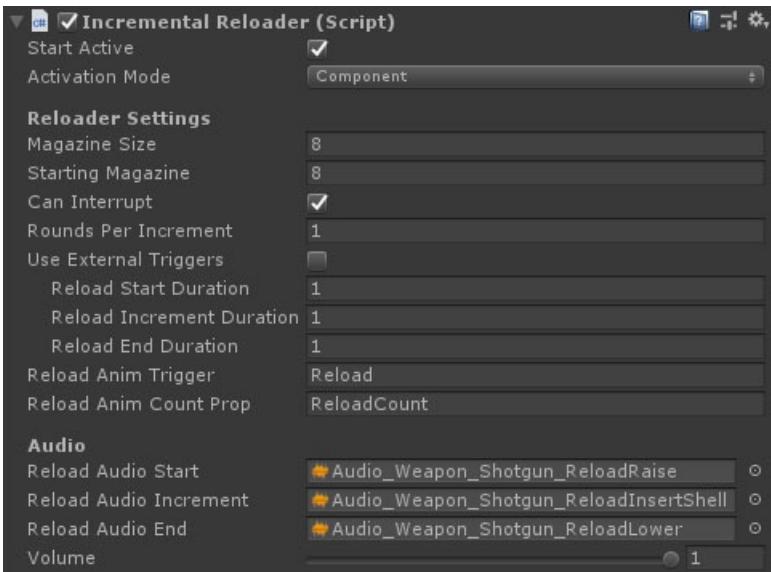
[Scopes & Optics](#)

IncrementalReloader MonoBehaviour

Overview

The IncrementalReloader module is a more complex reloader that increments the ammo count bit by bit. It will increment for each bullet required and can be interrupted. It is used by the demo shotgun.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Rounds Per Increment	Integer	The number of rounds to load into the gun each increment.
Use External Triggers	Boolean	If true, requires an external trigger such as a script or animation event to increment and complete the reload. If false, uses a timer.
Reload Start Duration*	Float	The duration of the reload start animation.

NAME	TYPE	DESCRIPTION
Reload Increment Duration*	Float	The duration of a single increment of the reload.
Reload End Duration*	Float	The duration of the reload end.
Reload Anim Trigger	String	The AnimatorController trigger key for the reload animation.
Reload Anim Count Prop	String	The AnimatorController property key for the reload count.
Reload Audio Start	AudioClip	The audio clip to play when the reload starts.
Reload Audio Increment	AudioClip	The audio clip to play during an increment of the reload.
Reload Audio End	AudioClip	The audio clip to play when the reload ends.

* This property is not visible unless the Use External Triggers property is set to False.

See Also

[Modular Firearms](#)

[FirearmAnimEventsHandler](#)

[Unity Animator](#)

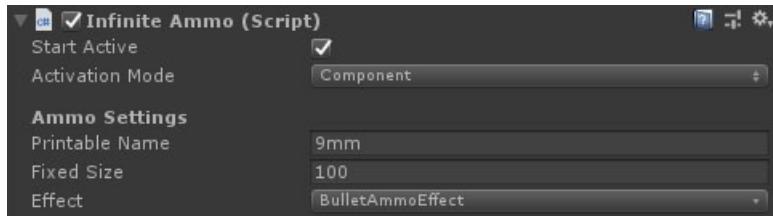
[Unity AnimatorController](#)

InfiniteAmmo MonoBehaviour

Overview

The InfiniteAmmo module provides a limitless pool of ammo for reloader modules to reload from.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Fixed Size	Integer	The ammo quantity available to any reloaders - must be \geq to the magazine size.

See Also

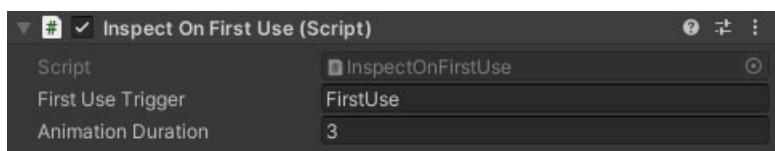
[Modular Firearms](#)

InspectOnFirstUse MonoBehaviour

Overview

The InspectOnFirstUse behaviour is used to play a one-shot inspect animation the very first time you pick up a weapon. It is used alongside an [ItemRecognitionTracker](#) attached to the wielder's player controller object which tracks the items that have been checked and prevents the animation being played twice.

Inspector



Properties

NAME	TYPE	DESCRIPTION
First Use Trigger	String	The name of the trigger parameter to fire on the weapon's animator when picking up the item for the first time.
Animation Duration	Float	The time the inspect animation takes, to prevent shooting and reloading during this time.

See Also

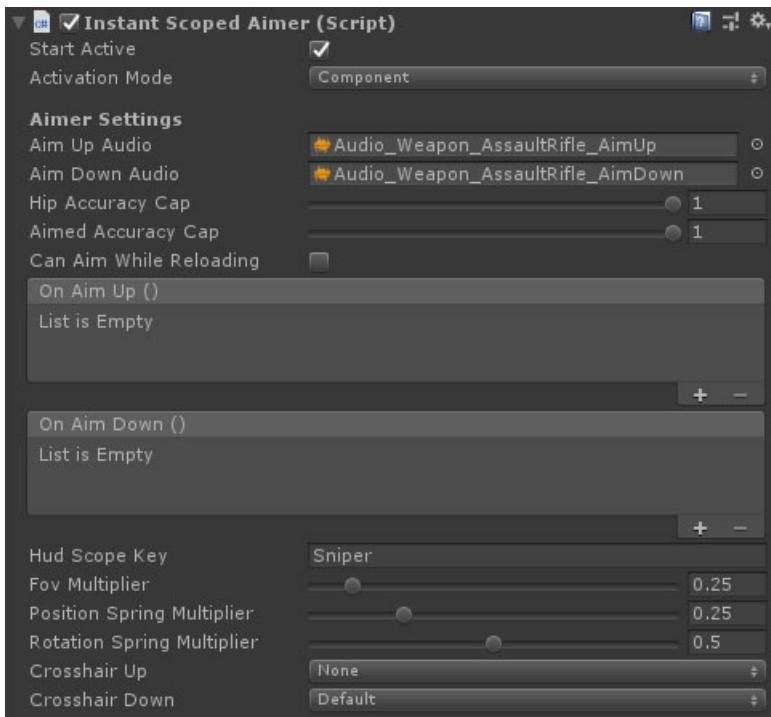
[Modular Firearms](#)

InstantScopedAimer MonoBehaviour

Overview

The InstantScopedAimer behaviour instantly zooms the camera and switches to a [HUD scope](#)

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.

NAME	TYPE	DESCRIPTION
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Hud Scope Key	String	The key for the specific scope to show.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

[Modular Firearms](#)

[First Person Camera](#)

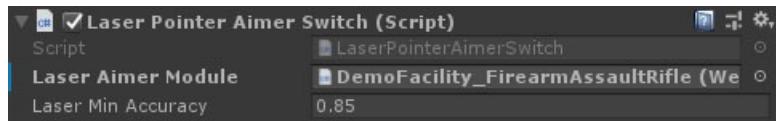
[HudScope](#)

LaserPointerAimerSwitch MonoBehaviour

Overview

The LaserPointerAimerSwitch behaviour can be added to a [laser pointer](#) to give a simplified system for switching a weapon's aimer module when the laser is active. This can be used for holding weapons tilted and using the laser to aim.

Inspector



Properties

NAME	TYPE	DESCRIPTION
LaserAimerModule	Aimer	The aimer module associated with this laser pointer. This should be disabled on start.
LaserMinAccuracy	Float	The minimum accuracy of the firearm when the laser is switched on.

See Also

[Modular Firearms](#)

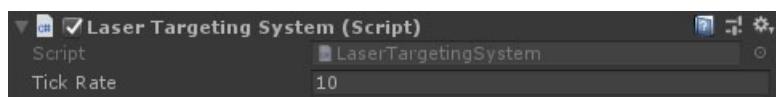
[WieldableLaserPointer Behaviour](#)

LaserTargetingSystem MonoBehaviour

Overview

The LaserTargetingSystem behaviour is paired with a [WieldableLaserPointer](#) to send the laser hit point to the registered guided projectiles as a target point. Switching the laser off or pointing it at the sky will clear the projectile's target. The guided projectile must use a [TargetingSystemTracker](#) component.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tick Rate	Integer	The laser pointer hit location will be updated every nth fixed update frame.

See Also

[Modular Firearms](#)

[TargetingSystemTracker Behaviour](#)

[WieldableLaserPointer Behaviour](#)

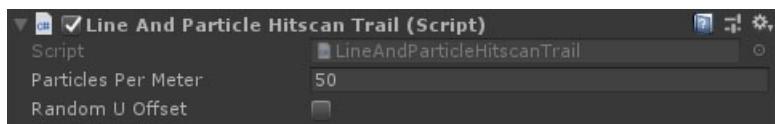
[Hitscan vs Projectiles](#)

LineAndParticleHitscanTrail MonoBehaviour

Overview

The LineAndParticleHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using both a [line renderer](#) and a [particle system](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Particles Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.
Random U Offset	Boolean	Randomise the line renderer texture's U mapping 0-1. Requires the material to have an " _OffsetU " property accessible via property block.

See Also

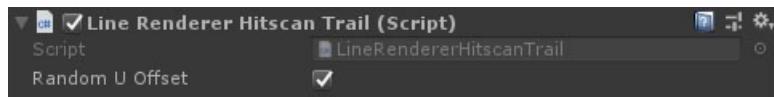
[Modular Firearms](#)

LineRendererHitscanTrail MonoBehaviour

Overview

The LineRendererHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using a [line renderer](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Random U Offset	Boolean	Randomise the texture's U mapping 0-1. Requires the material to have an " _OffsetU " property accessible via property block

See Also

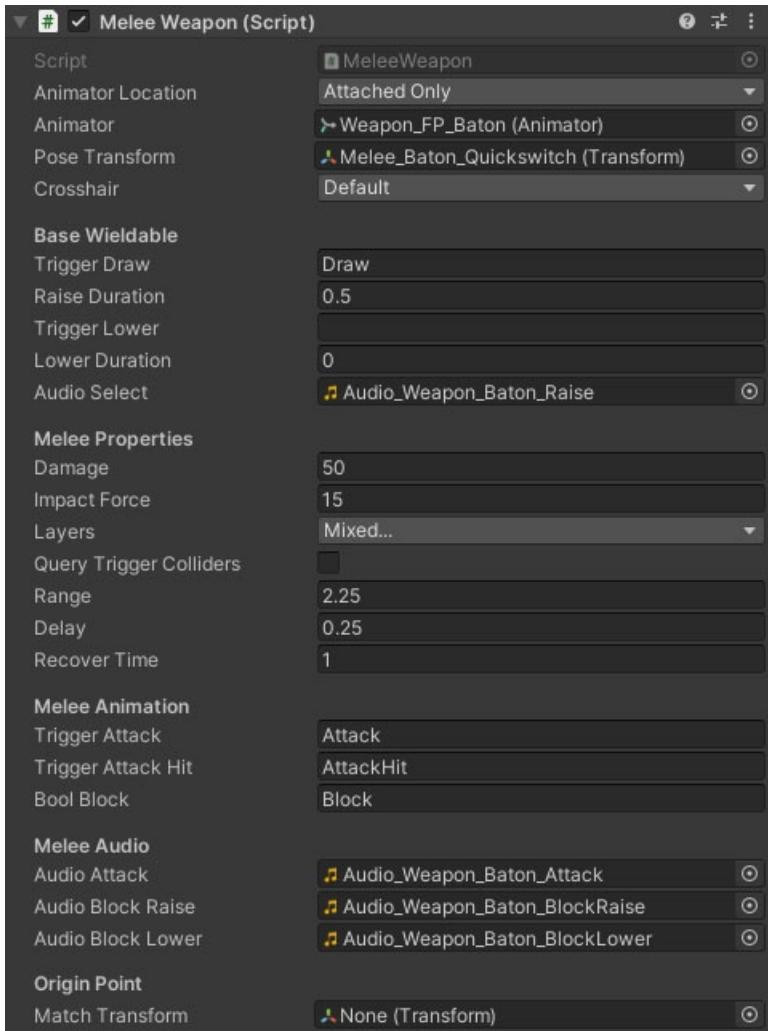
[Modular Firearms](#)

MeleeWeapon MonoBehaviour

Overview

The MeleeWeapon behaviour is a hand held melee weapon like a club or knife.

Inspector



Properties

Name	Type	Description
Animator Location	Dropdown	What animators should be exposed to the weapon components. Options are None , AttachedOnly , AttachedAndCharacter , MultipleAttached , MultipleAttachedAndCharacter and CharacterOnly .
Animator	Animator	The animator component of the weapon.
Pose Transform	Transform	The transform that should be offset for weapon poses.
Damage	Float	The damage the weapon does.
Impact Force	Float	The force to impart on the hit object. Requires either a Rigidbody or an impact handler on the hit object.
Layers	Float	The layers the attack will collide with.

Name	Type	Description
Query Trigger Colliders	Boolean	Should the attack be tested against trigger colliders.
Range	Float	The range that the melee weapon can reach.
Delay	Float	The delay from starting the attack to checking for an impact. Should be synced with the striking point in the animation.
Recover Time	Float	The recovery time after a hit.
Crosshair	FpsCrosshair	The crosshair to show when the weapon is drawn.
Trigger Draw	String	The AnimatorController trigger key for the raise animation.
Raise Duration	Float	The time taken to raise the item on selection.
Trigger Lower	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection.
Trigger Attack	String	The AnimatorController trigger key for the attack animation.
Trigger Attack Hit	String	The AnimatorController trigger key for the attack hit animation.
Bool Block	String	The AnimatorController bool key for the block animation.
Audio Select	AudioClip	The audio clip when raising the weapon.
Audio Attack	AudioClip	The audio clip when attacking.
Audio Block Raise	AudioClip	The audio clip when bringing the weapon into block position.
Audio Block Lower	AudioClip	The audio clip when bringing the weapon out of block position.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

[Melee Weapons](#)

[Unity Animator](#)

[Unity AnimatorController](#)

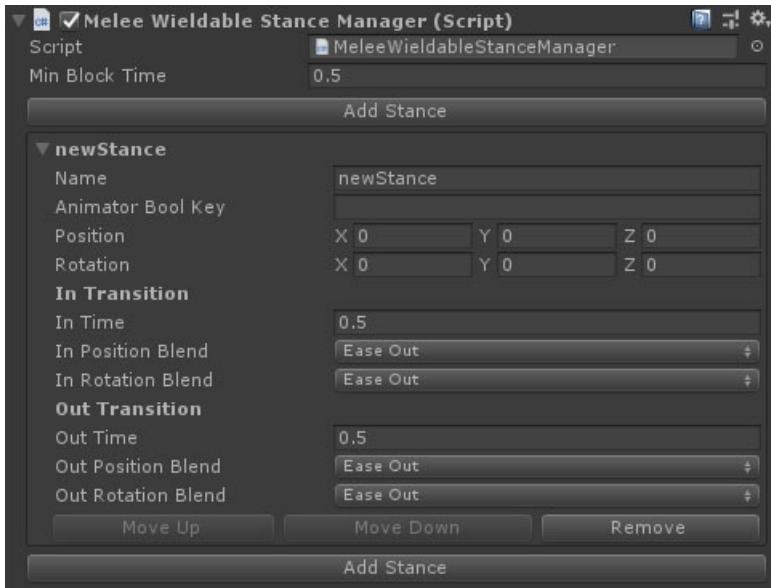
MeleeWieldableStanceManager MonoBehaviour

Overview

The MeleeWieldableStanceManager behaviour is used to specify poses or stances for a melee weapon. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The weapon will exit the stance temporarily while attacking or blocking.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the stance will be temporarily exited for an attack.
Min Block Time	Float	The minimum amount of time the stance will be temporarily exited when blocking (prevents tapping block).

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator .
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .

Name	Type	Description
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
In Time	Float	The time taken to enter the stance.
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
Out Time	Float	The time taken to enter the stance.

See Also

[Melee Weapons](#)

ModularFirearm MonoBehaviour

Overview

The ModularFirearm behaviour manages all of the modules that make up a firearm.

Inspector

Modular Firearm (Script)

Script ModularFirearm

Requires Wielder ✓

Animation

Animator Location Attached Only

Animator Weapon_FP_AssaultRifle (Animator)

Fire Anim Trigger Fire

Pose Transform Firearm_AssaultRifle_Quickswitch (Transform)

Accuracy

Zero Accuracy Speed 15

Move Accuracy Damping 0.5

Air Accuracy 0

Raise / Lower

Raise Delay Type Elapsed Time

Raise Duration 0.4

Raise Anim Trigger Draw

Lower Delay Type Elapsed Time

Lower Anim Trigger

Deselect Duration 0

Aiming

Aim Relative Transform None (Transform)

Reload

Persist Mag Count

Audio

Dry Fire Sound Audio_World_Shared_DryFire

Dry Fire Once Only

Weapon Raise Sound Audio_World_AssaultRifle_Draw

Weapon Volume 1

Origin Point

Match Transform None (Transform)

Modular Firearm Details

Modular Firearms Documentation

Firearm has all the required modules set up correctly

Add Shooter Existing: HitscanShooter

Add Trigger Existing: AutomaticTrigger

SemiAutoTrigger

BurstFireTrigger

Add Aimer Existing: WeaponMoveAimer

Add Reloader Existing: SimpleReloader

Add Ammo Existing: SharedPoolAmmo

Add Ammo Effect Existing: BulletAmmoEffect

Add Recoil Handler Existing: BetterSpringRecoilHandler

Add Muzzle Effect Existing: BasicGameObjectMuzzleEffect

Add Shell Ejector Existing: StandardShellEject

Properties

Name	Type	Description
Animator Location	Dropdown	What animators should be exposed to the weapon components and modules. Options are None , AttachedOnly , AttachedAndCharacter , MultipleAttached , MultipleAttachedAndCharacter and CharacterOnly .
Animator	Animator	The weapon geometry animator (optional).
Pose Transform	Transform	The transform that should be offset for weapon poses.
Fire Anim Trigger	String	The AnimatorController trigger key for the fire animation (blank = no animation).
Zero Accuracy Speed	Float	The speed above which your accuracy hits zero.
Move Accuracy Damping	Float	Smoothes changes in accuracy during sudden speed changes.
Air Accuracy	Float	A multiplier for move accuracy when airborne.
Raise Delay Type	Dropdown	The delay type for raising the weapon (you can use FirearmAnimEventsHandler to tie delays to animation). Options are None , Elapsed Time , External Trigger .
Raise Duration	Float	The duration in seconds for raising the weapon.
Raise Anim Trigger	String	The AnimatorController trigger key for the weapon raise animation (blank = no animation).
Lower Delay Type	Dropdown	The delay type for lowering the weapon (you can use FirearmAnimEventsHandler to tie delays to animation). Options are None , Elapsed Time , External Trigger .
Lower Anim Trigger	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Deselect Duration	Float	The time taken to lower the item on deselection.
Aim Relative Transform	Transform	The aim relative transform is the transform of the gun (mesh or bone) itself, which the optics are parented to. This allows a consistent offset to be calculated, even as the weapon is animated by keyframed or procedural spring animation.
Persist Mag Count	Boolean	When switching reloader modules, should the ammo in the magazine be transferred from one to the other?

NAME	TYPE	DESCRIPTION
Dry Fire Sound	AudioClip	The audio clip to play if attempting to fire while empty.
Dry Fire Once Only	Boolean	Should the dry fire only be played once until the weapon is reloaded.
Weapon Raise Sound	AudioClip	The audio clip to play when the weapon is drawn.
Weapon Volume	Slider	The volume of all weapon sounds (includes gunshots and reload - the muzzle effects and reloader modules have their own volume sliders that stack with this).

Origin Point

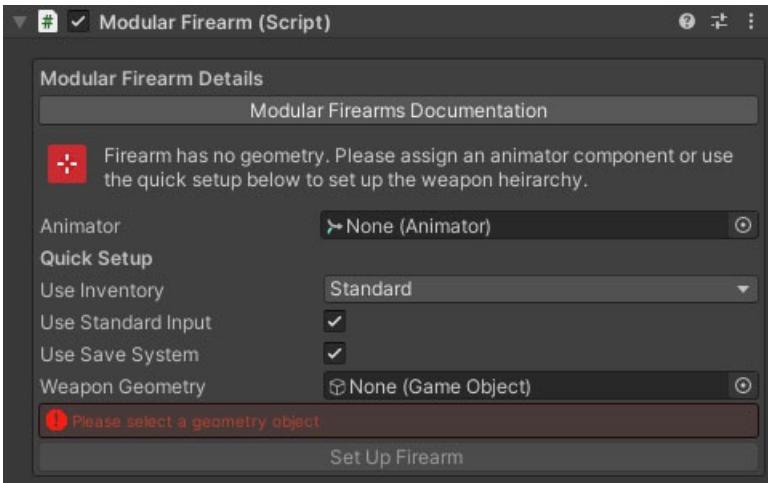
The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

Modular Firearm Details

The modular firearm details section assists in setting up a modular firearm. It displays the current state of the firearm along with the attached modules, and allows you to quickly add modules to the firearm.

Quick Setup

If the modular firearm object does not have any child objects, then the firearm inspector will show the quick-setup mode. This allows you to create a default firearm setup using a view model prefab from the project. This is only available if the object is placed in a scene. If it is accessed as a prefab in the project hierarchy, then these options will not appear because of issues with modifying prefab object heirarchies via scripts.



This is used to set up the firearm hierarchy and attach render geometry. It has the following properties:

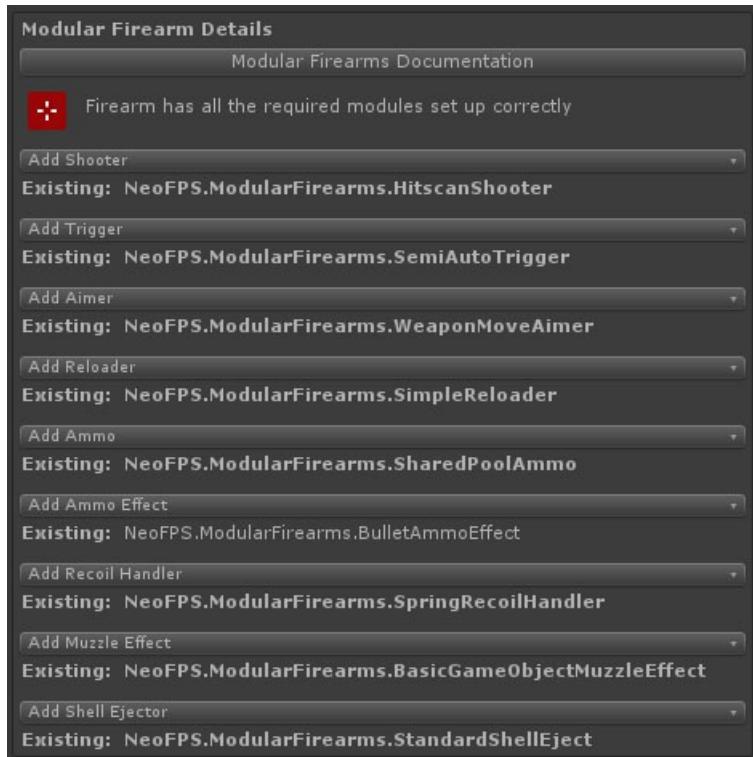
NAME	TYPE	DESCRIPTION
Animator	Animator	The weapon geometry animator. If this is set then the inspector will exit quick setup and move to module management mode as below. Use this if your view model is not a child of the firearm (eg if the firearm is the secondary fire mode of another weapon).

Name	Type	Description
Use Inventory	Dropdown	Which of the inventory implementations do you want the firearm to use. Choosing the Custom option does not add an inventory wieldable behaviour, meaning that you will need to add your own solution.
Use Standard Input	Boolean	Should the firearm use the standard NeoFPS input handler. Switch this off if you wan to add your own custom input handler.
Weapon Geometry	GameObject	This value is a prefab or model that you want to use to represent the first person weapon. If the object does not contain a mesh renderer or skinned mesh renderer in its hierarchy then it will cause an error. There is also a warning if the object has no animator in its hierarchy, though this is not compulsory.

With a valid weapon geometry object selected, the **Set Up Firearm** button will be available. Pressing this will create a spring object set up with procedural animation behaviours for handling things like recoil and weapon bob, and then will copy the weapon geometry under this.

Module Management

If the firearm already has child objects then the details section will instead show the modules options:



To add a firearm module click the relevant dropdown button and select the desired module from the menu. If a module is required for a firearm to operate then the info section at the top of the modular firearm details will display an error if that module is not attached. The firearm can also have multiple modules of each type, though only one should be set to start active.

Each of the attached modules can also report back any errors to the firearm they are attached to, and broken modules will be displayed as red in this list.

See Also

[FirearmAnimEventsHandler](#)

[Additive Transform And Effects](#)

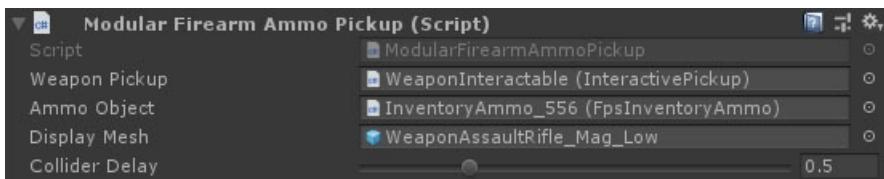
[Unity Animator](#)

ModularFirearmAmmoPickup MonoBehaviour

Overview

The ModularFirearmAmmoPickup behaviour is used in tandem with the [ModularFirearmDrop](#) when a character drops the weapon they are holding. It takes the magazine ammo count from the weapon as it is dropped, so that no extra ammo is spawned or lost.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Weapon Pickup	InteractablePickup	The pickup attached to the weapon drop.
Ammo Object	FpsInventoryAmmo	The ammo inventory item.
Display Mesh	GameObject	The display mesh for the weapon magazine. Will be hidden when the ammo is collected.
Collider Delay	Float	The delay before the ammo collider is enabled. Prevents the character dropping the weapon collecting the ammo immediately.

See Also

[Modular Firearms](#)

[ModularFirearmDrop](#)

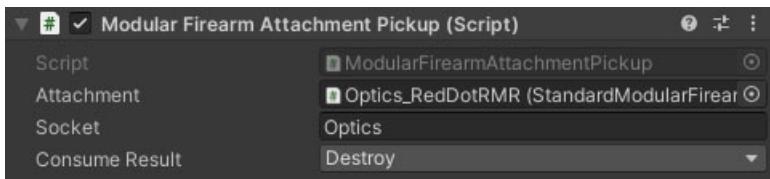
[FpsInventoryAmmo](#)

ModularFirearmAttachmentPickup MonoBehaviour

Overview

The ModularFirearmAttachmentPickup behaviour is used alongside an [InteractiveObject](#) or [InteractivePickup](#) to apply an attachment to the selected weapon. If the attachment requires an inventory item, then you can use the [InteractivePickup](#) component to pick up the attachment before this behaviour tries to apply it to the weapon. In this situation, if the selected weapon cannot accept the attachment then the inventory item will still be added to the player inventory, allowing them to select the attachment using the [attachment popup](#) when using a weapon that can accept the attachment.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Attachment	ModularFirearmAttachment	The attachment to apply to the firearm when picked up. This can be any attachment that inherits from the base <code>ModularFirearmAttachment</code> class.
Socket	String	The name of the socket the attachment should be applied to.
Consume Result	Dropdown	What to do to the pickup object once its attachment has been applied to the firearm. Options are Destroy , Disable , Nothing .

See Also

[Modular Firearms](#)

[Modular Firearm Attachments](#)

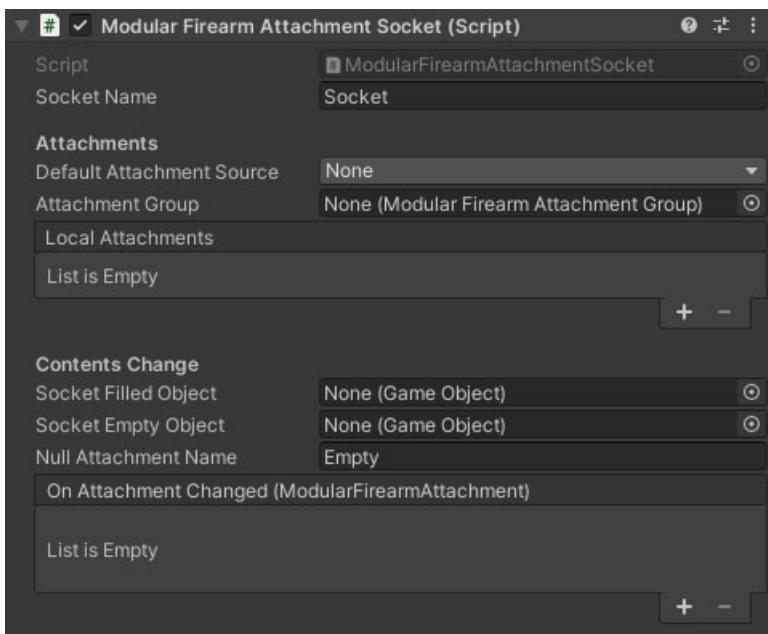
ModularFirearmAttachmentSocket MonoBehaviour

Overview

The ModularFirearmAttachmentSocket behaviour represents a socket into which an attachment can be installed. It allows you to install attachments via their prefab or ID.

Sockets can be nested within attachments. An example would be a shotgun with different length barrel attachments. Each of those could have a muzzle tip socket that can store suppressors or chokes. If the barrel attachment is swapped, then the attachment in the muzzle tip socket will be transferred into the socket on the new barrel and realigned, as long as the socket names match.

Inspector



Properties

Name	Type	Description
Socket Name	String	The name for this socket (eg Optics or Muzzle). This will be used to identify the socket to the attachment system, but could also be used in any attachment UI.
Default Attachment Source	Dropdown	Where is the default attachment taken from. None means the socket will start empty. This means you can select one of the attachments specified in this component. Group allows you to select one of the attachments from the provided group asset. Random means the socket will start with a random attachment from the combined options on this component and in the group.
Default Attachment	Dropdown	Allows you to select which attachment to pick from the default source specified above.
Attachment Group	ModularFirearmAttachmentGroup	A reusable group of attachments that can be attached to this socket. Its contents will be appended to the attachments listed below.
Attachments	Attachment Array	An array of available attachments. The attachments from the attachment group above will be appended to this.
Socket Filled Object	GameObject	(Optional) An object that will be active when the attachment slot is filled. Examples would be a mounting rail or folded iron sights.

NAME	TYPE	DESCRIPTION
Socket Empty Object	GameObject	(Optional) An object that will be active when the attachment slot is empty. This could be a fallback attachment if required.
Null Attachment Name	String	The display name that should be used for the attachment when the socket is empty (null attachment assigned).
On Attachment Changed	[UnityEvent][unity-events]	An event fired when the attachment is changed. If you point this at a method that takes a <code>ModularFirearmAttachment</code> parameter, then the new attachment will be sent to the method whenever the event is fired.

See Also

[Modular Firearms](#)

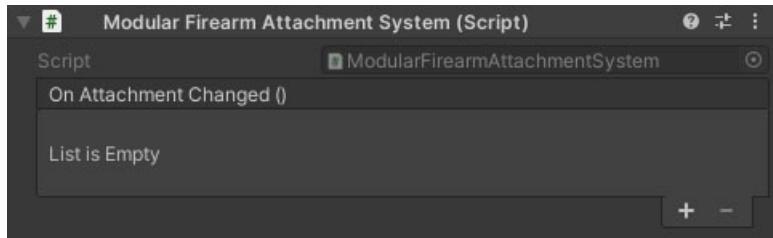
[Modular Firearm Attachments](#)

ModularFirearmAttachmentSystem MonoBehaviour

Overview

The ModularFirearmAttachmentSystem behaviour manages the sockets and attachments on a firearm.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Attachment Changed	[UnityEvent][unity- events]	An event fired whenever one of the attachment system's sockets change attachment.

See Also

[Modular Firearms](#)

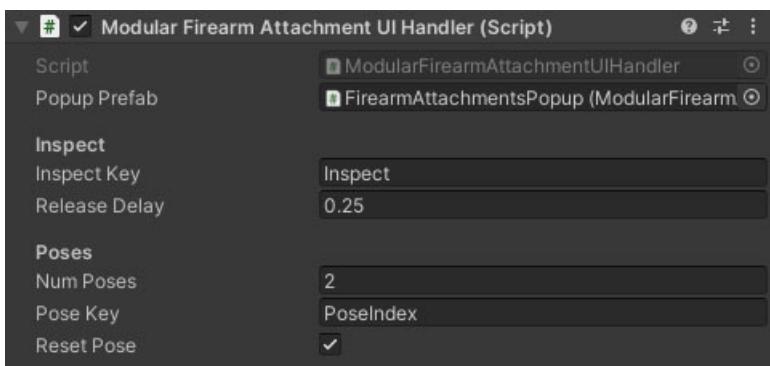
[Modular Firearm Attachments](#)

ModularFirearmAttachmentUIHandler MonoBehaviour

Overview

The ModularFirearmAttachmentUIHandler behaviour inherits from the [AnimatedWeaponInspect](#) behaviour. When the firearm is inspected, an attachment UI popup will be shown, allowing the player to modify the attachments on the gun. A simple example popup called [ModularFirearmAttachmentUIPopup](#) is provided with NeoFPS, but you can add your own by extending from its base class.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Popup Prefab	Attachment UI Popup	The attachments UI popup to show when inspecting the weapon.
Inspect Key	String	The name of the bool parameter to set on the weapon's animator while inspecting.
Release Delay	Float	How long after releasing the inspect key, should the weapon be able to function again.
Num Poses	Integer	How many inspect poses does the weapon have.
Pose Key	String	The name of the integer parameter to set on the weapon's animator that controls the animation pose.
Reset Pose	Boolean	Should the pose index be reset when inspecting state changes.

See Also

[Modular Firearms](#)

[Modular Firearm Attachments](#)

[ModularFirearmAttachmentUIPopup](#)

[AnimatedWeaponInspect](#)

ModularFirearmAttachmentUIPopup MonoBehaviour

Overview

The ModularFirearmAttachmentUIPopup behaviour is an example implementation of a UI for changing firearm attachments. It uses simple multi-choice entries for each [socket](#) that cycles through the available [attachments](#). Nested sockets will be indented.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Socket Picker Prototype	MultilnputMultiChoice	The picker UI element used to select the attachment (should be a child of this object).
Completed Button	MultilnputButton	The button UI element used to close the popup (should be a child of this object).

See Also

[Modular Firearms](#)

[Modular Firearm Attachments](#)

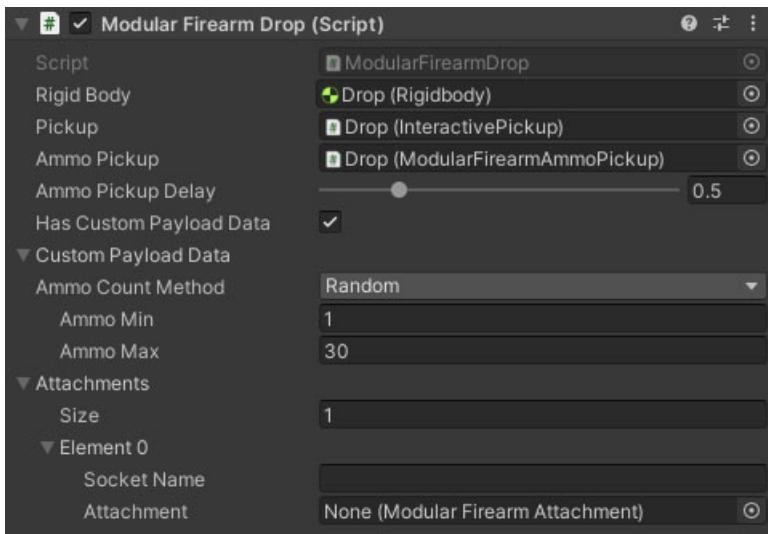
[ModularFirearmAttachmentUIHandler](#)

ModularFirearmDrop MonoBehaviour

Overview

The ModularFirearmDrop is spawned when a character drops their current firearm. It acts as an [interactive pickup](#) for other characters to pick it up, and manages the magazine ammo as a separate pickup. You can optionally add custom payload data to the drop and specify the magazine ammo count and attachments for the firearm. If the drop is spawned by the player character dropping the item from their inventory, then the payload data will be based on the setup of the firearm that was dropped, not the settings in the inspector.

Inspector



Properties

The ModularFirearmDrop inherits from the [FpsInventoryWieldableDrop](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Ammo Pickup	ModularFirearmAmmoPickup	The ammo pickup for this weapon's magazine.
Ammo Pickup Delay	Float	The delay from dropping before the ammo pickup becomes active (prevents the dropper from instantly grabbing ammo).
Has Custom Payload Data	Boolean	Does this drop have custom payload data for specifying ammo and/or attachments. The properties below will only be visible if this is true.
Ammo Count Method	Float	How should the ammo count be calculated.
Ammo Count	Float	The amount of ammo to set for the gun's magazine count on pick up. This property will only be visible if Ammo Count Method is set to Fixed Value .
Ammo Min	Float	The minimum amount (inclusive) of ammo to set for the gun's magazine count on pick up. This property will only be visible if Ammo Count Method is set to Random .
Ammo Max	Float	The maximum amount (inclusive) of ammo to set for the gun's magazine count on pick up. This property will only be visible if Ammo Count Method is set to Random .
Attachments	Attachments Array	The attachments that should be set on the firearm on pickup.

The entries in the **Attachments** array have the following properties.

NAME	TYPE	DESCRIPTION
Socket Name	String	The name of the socket the attachment should be applied to. This must match the socket name assigned in the ModularFirearmAttachmentSocket component on the gun.
Attachment	Attachment	The attachment prefab to attach to this socket. This must have a component that inherits from the ModularFirearmAttachment base class, such as the StandardModularFirearmAttachment behaviour.

See Also

[Modular Firearms](#)

[ModularFirearmAmmoPickup](#)

[InteractivePickup](#)

[Modular Firearm Attachments](#)

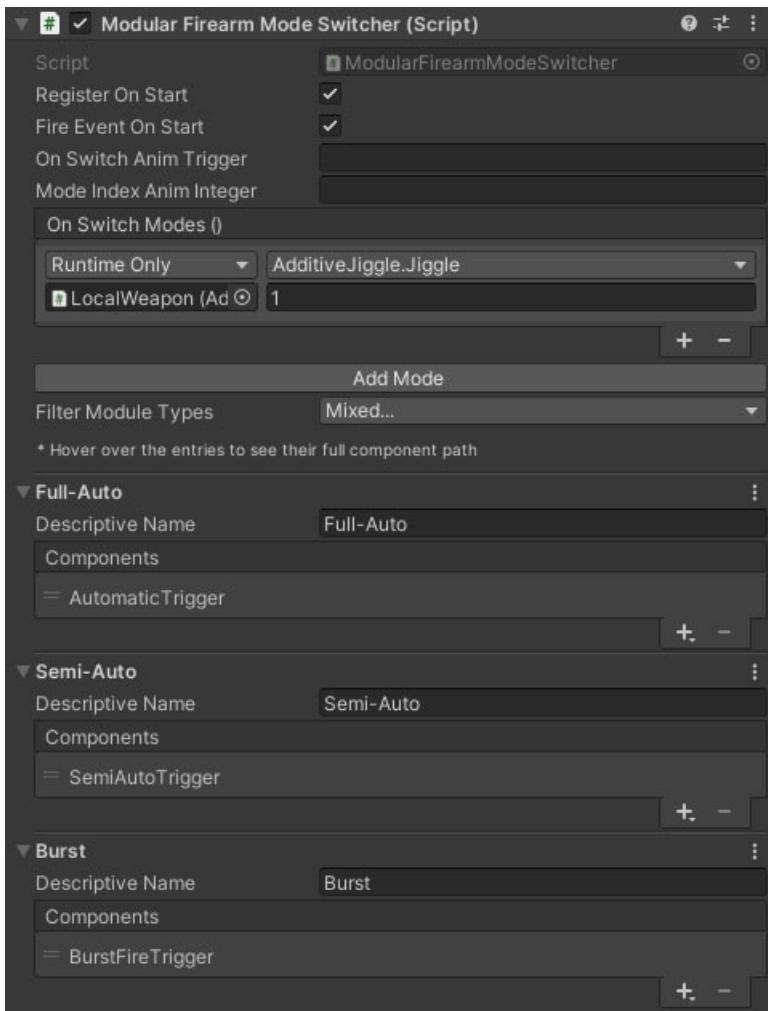
ModularFirearmModeSwitcher MonoBehaviour

Overview

The ModularFirearmModeSwitcher behaviour is attached to modular firearms to handle mode switch commands by enabling specific firearm modules or components.

Since only one of each module type can be enabled on a firearm at any one time, when the mode switcher enables one, the previous is automatically disabled. This does not apply to components which are not firearm modules.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Switch Modes	UnityEvent	An event fired whenever the firearm mode is switched.

The **Add Mode** button adds a new blank mode to the component. Each mode has the following properties:

NAME	TYPE	DESCRIPTION
Descriptive Name	String	The name displayed on the HUD for this mode. Leave this blank if you do not want the mode name displayed.
Components	Component Array	The components to enable for this mode (for firearm module components, this will disable the old component automatically).

See Also

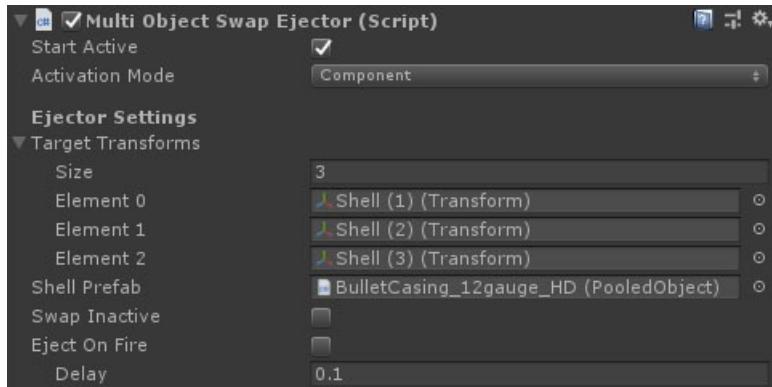
[Modular Firearms](#)

MultiObjectSwapEjector MonoBehaviour

Overview

The MultiObjectSwapEjector behaviour replaces and disables multiple animated objects with pooled bullet casing objects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Target Transforms	Transform Array	The transforms of the objects in the firearm heirarchy to swap out.
Shell Prefab	PooledObject	The bullet casing prefab object to spawn.
Eject On Fire	Boolean	Should the ejector start the ejection process immediately when the gun is fired or wait until triggered by an animation event or other.
Swap Inactive	Boolean	If any of the target transforms are inactive, should they be swapped or ignored.
Delay	Float	The delay time between starting the ejection process and the objects being swapped

See Also

[Modular Firearms](#)

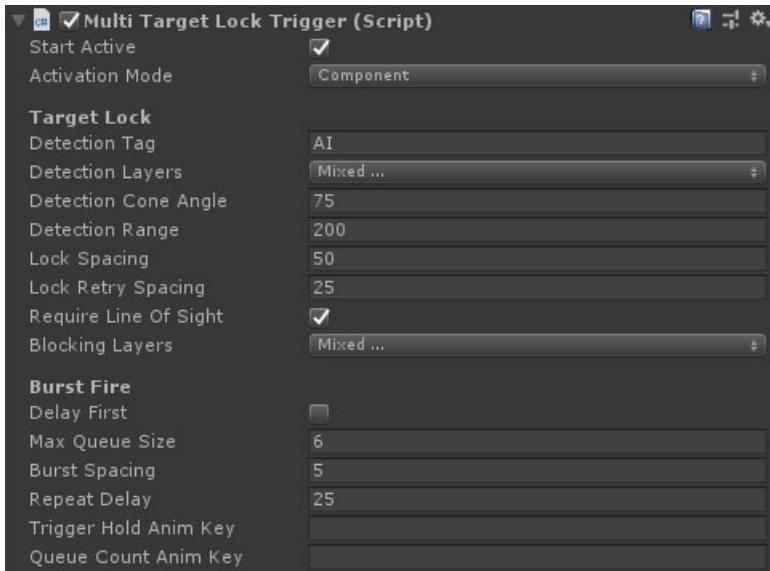
[PooledObject](#)

MultiTargetLockTrigger MonoBehaviour

Overview

The MultiTargetLockTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

[Modular Firearms](#)

NearestObjectWithTagTracker MonoBehaviour

Overview

The NearestObjectWithTagTracker behaviour is a guided projectile tracker that checks its vicinity for objects with the correct layer and tag, then tracks the closest one.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Detection Tag	String	The object tag to home in on.
Detection Layers	LayerMask	The layers to check for colliders on.
Detection Range	Float	The max distance for targets to home in on.
Detection Counter	Float	The time between searching for targets.

See Also

[Modular Firearms](#)

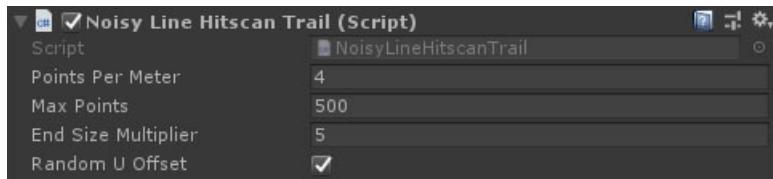
[Hitscan vs Projectiles](#)

NoisyLineHitscanTrail MonoBehaviour

Overview

The NoisyLineHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using a [line renderer](#). The points of the line are driven by a [particle system](#) which applies noise over time, making the line drift over time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Points Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.
Max Points	Integer	The maximum number of points for the line.
End Size Multiplier	Float	The line width once the trail dies.
Random U Offset	Float	Randomise the texture's U mapping 0-1. Requires the material to have an " <code>_OffsetU</code> " property accessible via property block

See Also

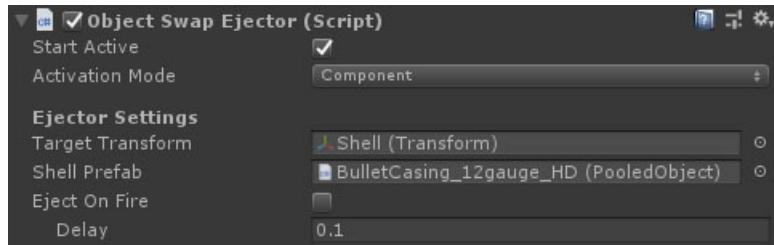
[Modular Firearms](#)

ObjectSwapEjector MonoBehaviour

Overview

The ObjectSwapEjector behaviour replaces and disables an animated object with a pooled bullet casing object.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Target Transform	Transform	The transform of an object in the firearm heirachy to swap out.
Shell Prefab	PooledObject	The bullet casing prefab object to spawn.
Eject On Fire	Boolean	Should the ejector start the ejection process immediately when the gun is fired or wait until triggered by an animation event or other.
Delay	Float	The delay time between starting the ejection process and the object being swapped

See Also

[Modular Firearms](#)

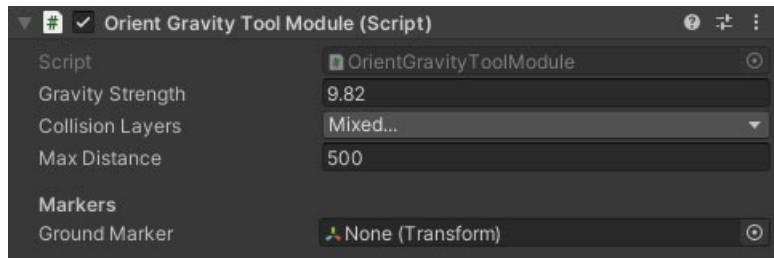
[PooledObject](#)

OrientGravityToolModule MonoBehaviour

Overview

The OrientGravityToolModule behaviour is used to change the wielder's gravity based on the surface they are aiming at. Holding the primary fire button will show a surface normal marker. Releasing the fire button will change the gravity direction to point into that surface. Pressing reload cancels the tool.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Gravity Strength	Float	The gravity acceleration strength (meters per second squared).
Collision Layers	Layer Mask	The physics layers the tool can blink onto.
Max Distance	Float	The maximum distance that the character can blink.
Ground Marker	Transform	An object in the tool's hierarchy to use as the ground position marker for the blink target (the object will be moved out of the tool's hierarchy).

See Also

[Wieldable Tools](#)

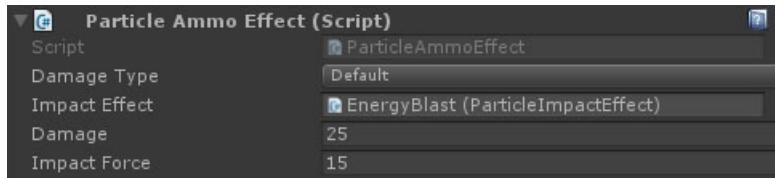
[The NeoCharacterController](#)

ParticleAmmoEffect MonoBehaviour

Overview

The ParticleAmmoEffect behaviour spawns a pooled particle system object on impact.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Impact Effect	ParticleImpactEffect	The object to spawn at the impact location.
Damage	Float	The damage the bullet does.
Impact Force	Float	The force to be imparted onto the hit object. Requires either a Rigidbody or an impact handler.

See Also

[Modular Firearms](#)

[ParticleImpactEffect](#)

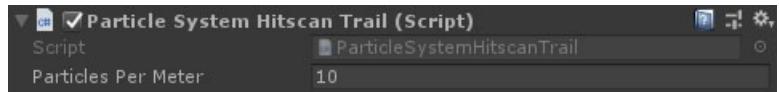
[Unity Rigidbody](#)

ParticleSystemHitscanTrail MonoBehaviour

Overview

The ParticleSystemHitscanTrail behaviour uses a [particle system](#) to spawn particles along a hitscan bullet trajectory.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Particles Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.

See Also

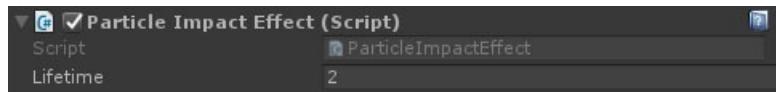
[Modular Firearms](#)

ParticleImpactEffect MonoBehaviour

Overview

The ParticleImpactEffect behaviour manages the lifecycle of a pooled particle object that is spawned on a bullet hit by the [ParticleAmmoEffect](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lifetime	Float	Duration the object should remain active before being returned to the pool.

See Also

[Modular Firearms](#)

[ParticleAmmoEffect](#)

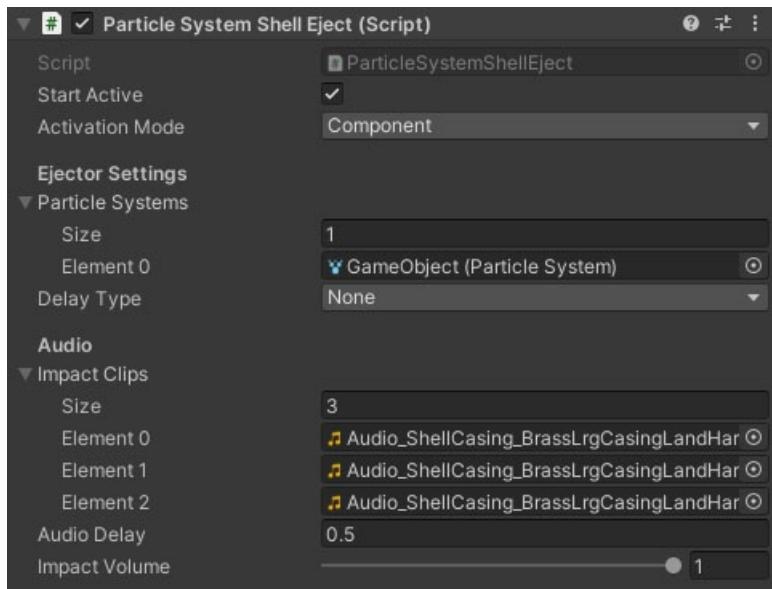
[Unity Rigidbody](#)

ParticleSystemShellEject MonoBehaviour

Overview

The ParticleSystemShellEject module triggers one or more particle systems on each shot.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Particle Systems	ParticleSystem Array	The particle systems to play when a shell is ejected.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are None , Elapsed Time , External Trigger .
Delay*	Float	The delay time between firing and ejecting a shell.
Impact Clips	AudioClip	The audio clips to play for an empty shell hitting the ground.
Audio Delay	Float	The time between a shell ejecting, and an impact audio clip being played.
Impact Volume	Float	The volume to play the empty shell impact audio.

* This property will only be visible if the delay type is set to Elapsed Time

See Also

Modular Firearms

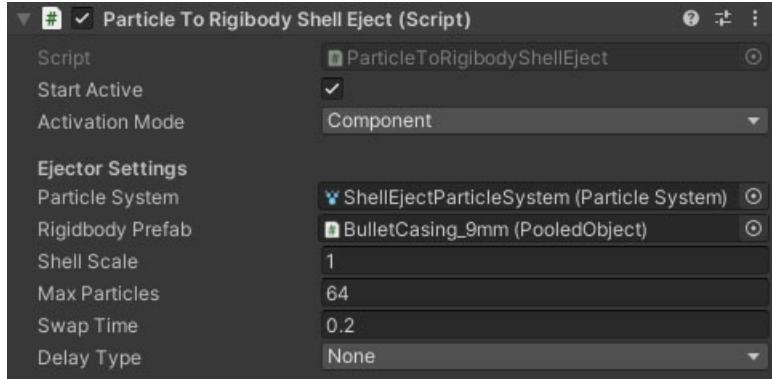
PooledObject

ParticleToRigidbodyShellEject MonoBehaviour

Overview

The ParticleToRigidbodyShellEject module triggers one or more particle systems on each shot.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Particle System	ParticleSystem	The particle system to play when a shell is ejected.
Rigidbody Prefab	PooledObject	The pooled rigidbody to swap the particles for once out of view.
Max Particles	Integer	The maximum number of particles that can be visible.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are None , Elapsed Time , External Trigger .
Delay	Float	The delay time between firing and ejecting a shell. This property will only be visible if the delay type is set to Elapsed Time
Shell Scale	Float	The scale to be applied to the rigidbody shell casings (particles are controlled in the particle system).
Swap Time	Float	The amount of time elapsed from a shell particle spawning before it is switched with a rigidbody version.

See Also

[Modular Firearms](#)

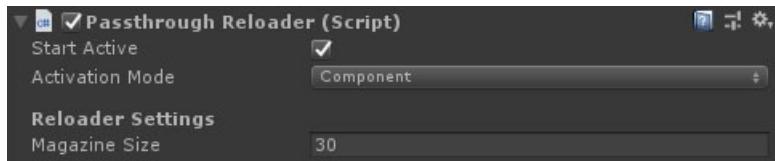
[PooledObject](#)

PassthroughReloader MonoBehaviour

Overview

The PassthroughReloader module has no magazine size, but pulls ammo directly from the firearm's ammo module with every shot.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Integer	The value it should report for its magazine size if any modules query it.

See Also

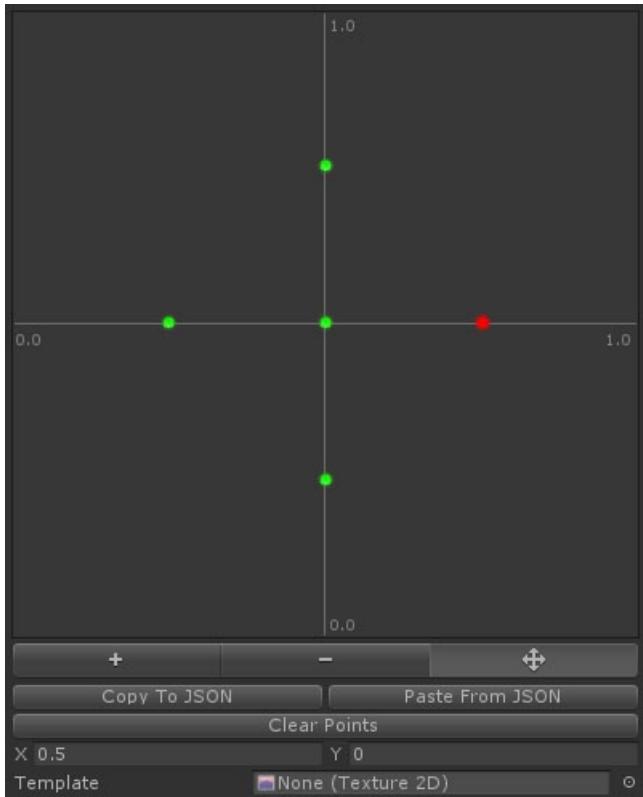
[Modular Firearms](#)

PatternBallisticShooter MonoBehaviour

Overview

The PatternBallisticShooter module instantly fires in a preset pattern. The pattern is defined in 2D with a max distance of 1m from the origin on each axis. The distance setting specifies how far from the firearm's muzzle the full pattern size is achieved.

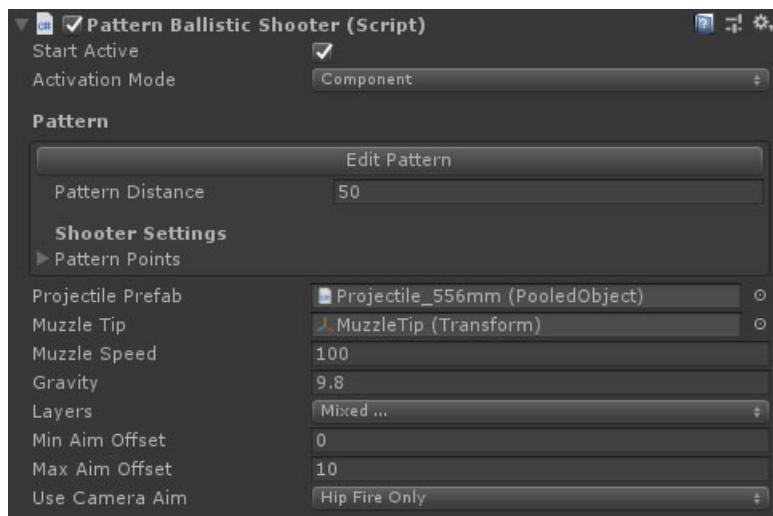
Pattern Editor



The pattern editor allows you to specify the gunshot pattern. The +, - and move buttons switch the editor mode. In + mode, clicking will add a new point. In - mode, clicking will remove the clicked point. In **move** mode, you can select a point by clicking it, and move a point by click-dragging it. The editor also has the following controls:

CONTROL	DESCRIPTION
Copy To JSON	Copy all points to the clipboard in JSON format. You can paste this to any text editor to make changes.
Paste From JSON	If the clipboard contains valid JSON, replace all the points with those on the clipboard.
Clear Points	Remove all points from the pattern.
Position	The position of the selected point (clamped to a -1 to 1 range on each axis).
Template	Load a background image to use as a template for placing points.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Pattern Distance	Float	The distance from the muzzle tip at which the pattern will match the diagram.
Pattern Points	Vector2 Array	The points that make up the pattern.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Muzzle Speed	Float	The speed of the projectile.
Gravity	Float	The gravity for the projectile.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .

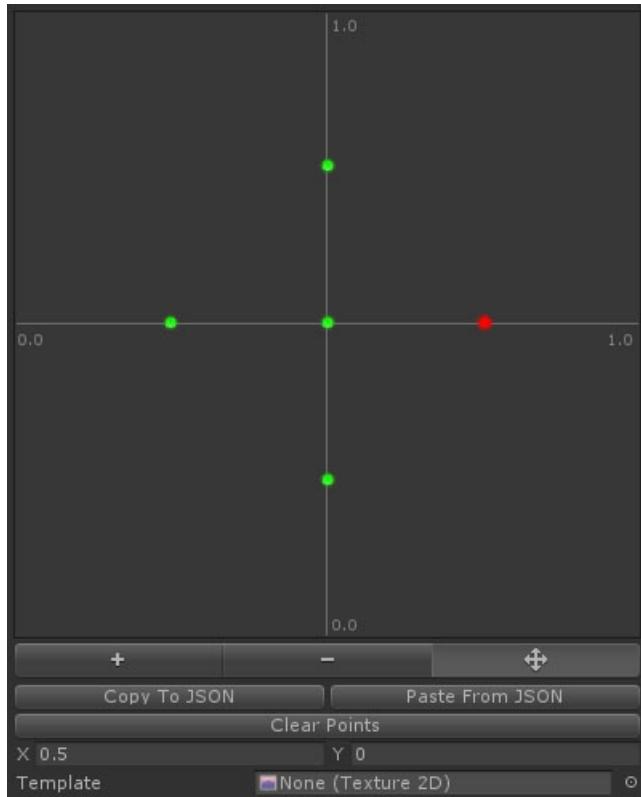
See Also

PatternHitscanShooter MonoBehaviour

Overview

The PatternHitscanShooter module instantly fires in a preset pattern. The pattern is defined in 2D with a max distance of 1m from the origin on each axis. The distance setting specifies how far from the firearm's muzzle the full pattern size is achieved.

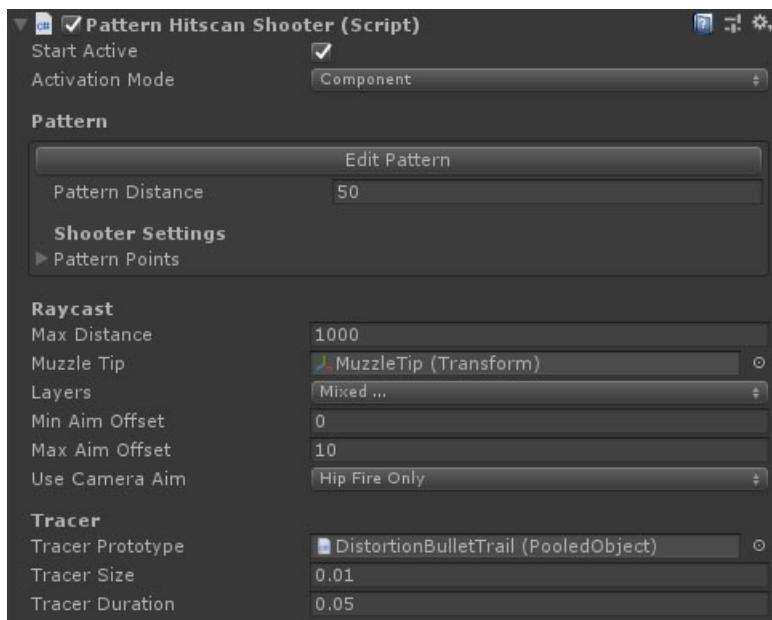
Pattern Editor



The pattern editor allows you to specify the gunshot pattern. You can select a point by left clicking and move a point by left-click dragging.

CONTROL	DESCRIPTION
Template	Load a background image to use as a template for placing points.
Copy To JSON	Copy all points to the clipboard in JSON format. You can paste this to any text editor to make changes.
Paste From JSON	If the clipboard contains valid JSON, replace all the points with those on the clipboard.
Add New Point	Add a new point to the pattern (the new points will be placed at the center and selected automatically).
Remove Point	Remove the selected point.
Position	The position of the selected point (clamped to a -1 to 1 range on each axis).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Pattern Distance	Float	The distance from the muzzle tip at which the pattern will match the diagram.
Pattern Points	Vector2 Array	The points that make up the pattern.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).

NAME	TYPE	DESCRIPTION
Tracer Size	Float	The size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long should the tracer object stay visible.

See Also

[Modular Firearms](#)

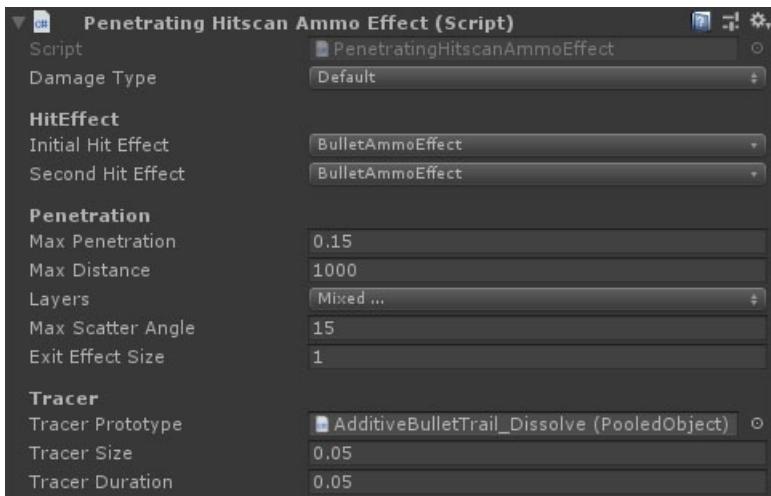
PenetratingHitscanAmmoEffect MonoBehaviour

Overview

The PenetratingHitscanAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the distance travelled. If you are using a projectile shooter, then there is a separate [PenetratingProjectileAmmoEffect](#) that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector



Properties

Name	Type	Description
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Penetration	Float	The maximum thickness of object the bullet can penetrate.
Max Distance	Float	The maximum distance that the weapon will register a hit (includes the distance travelled up to the penetration).
Layers	LayerMask	The layers bullets will collide with.
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Tracer Prototype	PooledObject	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface)
Tracer Size	Float	How size (thickness/radius) of the tracer line.

NAME	TYPE	DESCRIPTION
Tracer Duration	Float	How long the tracer line will last.

See Also

[Modular Firearms](#)

PenetratingProjectileAmmoEffect MonoBehaviour

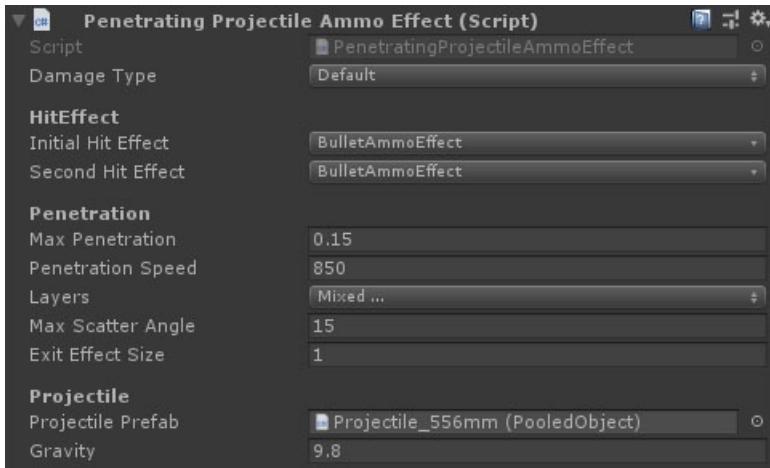
Overview

The PenetratingProjectileAmmoEffect module allows projectiles to penetrate through surfaces based on the impact angle and the speed the projectile is travelling. A new projectile will be spawned on the other side of the surface, with its speed and direction altered based on the impact.

If you are using a hitscan shooter, then there is a separate [PenetratingHitscanAmmoEffect](#) that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Penetration	Float	The maximum thickness of object the bullet can penetrate.
Penetration Speed	Float	The speed at which max penetration will be reached. The speed will also be clamped to this so that (for example) hitscan weapons don't have infinite speed.
Layers	LayerMask	The layers bullets will collide with.
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Projectile Prefab	PooledObject	The projectile to spawn.

NAME	TYPE	DESCRIPTION
Gravity	Float	The gravity for the projectile.

See Also

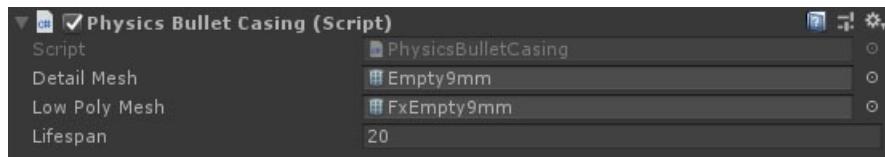
[Modular Firearms](#)

PhysicsBulletCasing MonoBehaviour

Overview

The PhysicsBulletCasing behaviour is a bullet casing that can be ejected from firearms and will physically react with the world.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Detail Mesh	Mesh	The detail mesh to show while the bullet is in the first person view.
Low Poly Mesh	Mesh	The low poly mesh to switch to when not in the first person view.
Lifespan	Float	How long should the casing remain before being returned to the pool.

See Also

[Modular Firearms](#)

[PooledObject](#)

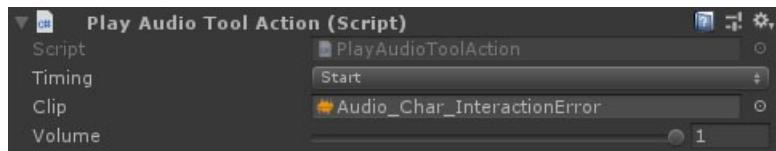
[Unity Rigidbody](#)

PlayAudioToolAction MonoBehaviour

Overview

The PlayAudioToolAction behaviour is used to play an audio clip on the start or end of a [wieldable tool's](#) actions.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the clip be played.
Clip	AudioClip	The clip to play.
Volume	Float	The volume for the clip.

See Also

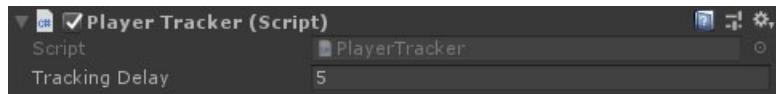
[Wieldable Tools](#)

PlayerTracker MonoBehaviour

Overview

The PlayerTracker behaviour is a guided projectile tracker component which directs the projectile towards the current player character. Definitely more useful on enemy weapons than player weapons.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tracking Delay	Float	The time from firing to starting to steer towards the player character.

See Also

[Modular Firearms](#)

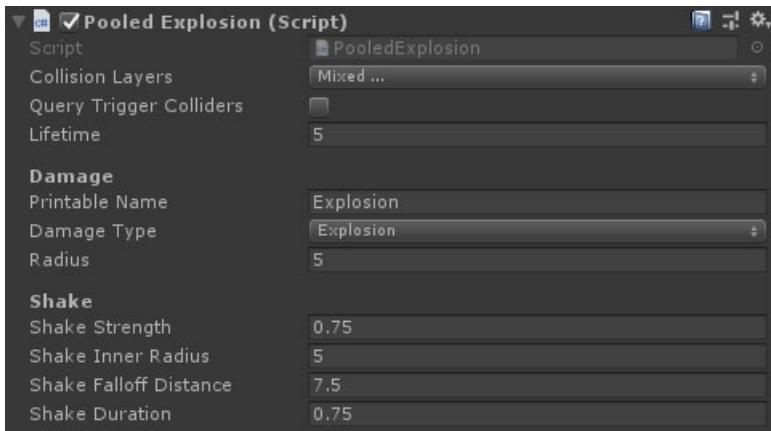
[Hitscan vs Projectiles](#)

PooledExplosion MonoBehaviour

Overview

The PooledExplosion behaviour manages the lifecycle of a pooled particle object, and handles applying damage and force to items in its area of effect.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Collision Layers	Float	The valid collision layers the explosion will affect.
Query Trigger Colliders	Boolean	Should the explosion be tested against trigger colliders.
Lifetime	Float	Duration the object should remain active before being returned to the pool.
Printable Name	Float	A description of the damage, used when logging and displaying game events.
Damage Type	Float	The damage type the explosion applies (enables filtering damage types).
Radius	Float	The radius of the explosion.
Shake Strength	Float	The strength of the camera (and other) shake due to the explosion.
Shake Inner Radius	Float	The inner shake radius of the explosion. Any shake handlers within this radius will be affected at full strength, falling off to 0 outside this based on the falloff distance.
Shake Falloff Distance	Float	The distance beyond the inner radius where the shake effect drops off to 0.
Shake Duration	Float	The duration of the shake effect.

See Also

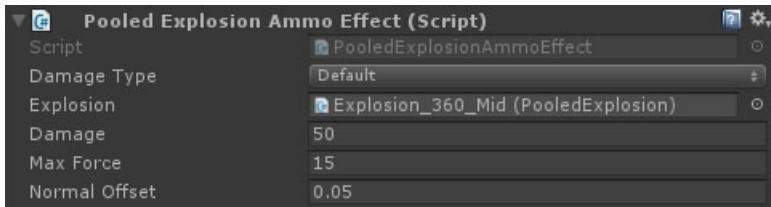
[Explosions](#)

PooledExplosionAmmoEffect MonoBehaviour

Overview

The PooledExplosionAmmoEffect spawns a [PooledExplosion](#) and passes it the relevant data to inflict damage and add force.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn at the impact location.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a Rigidbody or an impact handler and drops off to zero the further from the center the object is.
Normal Offset	Float	An offset from the hit point along its normal to spawn the explosion. Prevents the explosion from appearing embedded in the surface.

See Also

[Modular Firearms](#)

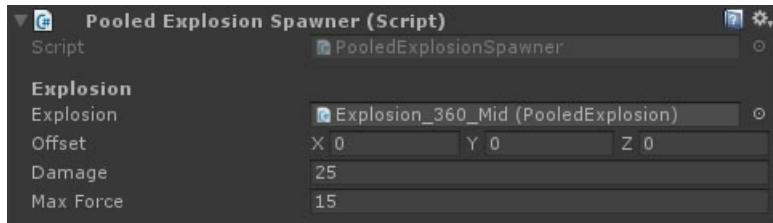
[Pooled Explosion](#)

PooledExplosionSpawner MonoBehaviour

Overview

The PooledExplosionSpawner .

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn.
Offset	Vector3	An offset from the object's origin that the explosion will spawn.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a Rigidbody or an impact handler and drops off to zero the further from the center the object is.

See Also

[Explosions](#)

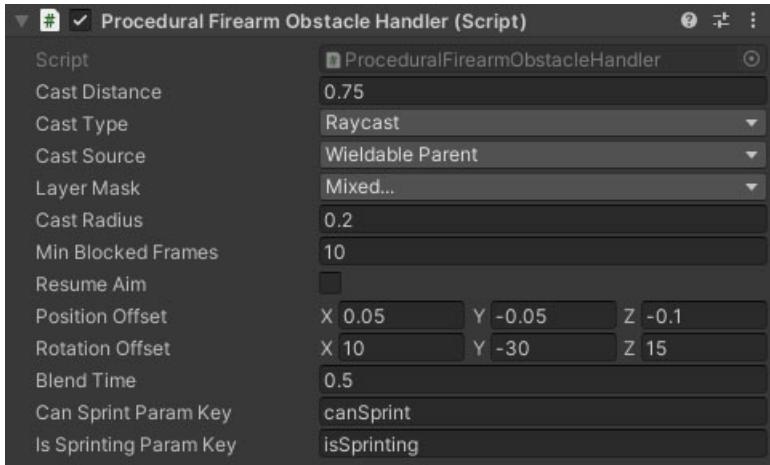
[PooledExplosion](#)

ProceduralFirearmObstacleHandler MonoBehaviour

Overview

The ProceduralFirearmObstacleHandler behaviour uses the stance system to set an obstructed pose, and blocks the firearm trigger whenever the player character's weapon is obstructed by an obstacle in the world.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Cast Distance	Float	The distance in front of the weapon to check for obstacles.
Cast Type	Dropdown	The cast type to use. Options are Raycast and Spherecast .
Cast Source	Dropdown	What to use as the source of the cast. Options are WieldableParent , CharacterAim and Camera .
Layer Mask	LayerMask	What layers are counted as obstacles and make the character move the weapon.
Cast Radius	Float	The radius of the sphere cast (if cast type is set to Spherecast).
Min Blocked Frames	Integer	The minimum number of frames the weapon can be blocked. This prevents the weapon from rapidly switching between blocked and not.
Resume Aim	Boolean	Should the aim be resumed when the firearm is no longer blocked (this will always happen if the aim hold button is pressed).
Position Offset	Vector	The target position offset for the obstructed pose.
Rotation Offset	Vector	The target rotation offset for the obstructed pose.
Blend Time	Float	The time taken to blend in and out of the obstructed pose.
Can Sprint Param Key	String	The
Is Sprinting Param Key	String	The

See Also

[Modular Firearms](#)

[The Motion Graph](#)

ProceduralFirearmSprintHandler MonoBehaviour

Overview

The AnimatedFirearmSprintHandler behaviour is attached to a [modular firearm](#) to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an [additive transform](#) bob effect.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.

NAME	TYPE	DESCRIPTION
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Offset Range	Vector	A position offset range either side of the sprint offset that will be randomised with each step to add variety.
Rotation Range	Vector	A rotation offset range either side of the sprint sprint that will be randomised with each step to add variety.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

Firearms

NAME	TYPE	DESCRIPTION
Action On Aim	Dropdown	What to do when the firearm enters / exits ADS. Stop Sprinting completely stops the character from sprinting until they leave ADS. Stop Animation stops the firearm sprint animation without affecting the character movement. Cannot Aim blocks the player from aiming down sights while sprinting.
Action On Reload	Dropdown	What to do when the firearm is reloaded. Stop Sprinting completely stops the character from sprinting until the reload animation completes. Stop Animation stops the firearm sprint animation without affecting the character movement.
Action On Fire	Dropdown	What to do when the firearm trigger is pulled while sprinting. Stop Sprinting completely stops the character from sprinting until the weapon stops firing. Stop Animation stops the firearm sprint animation without affecting the character movement (both of these have a slight delay before firing to allow the weapon to be aligned). Cannot Fire means that the firearm trigger is blocked until the character stops sprinting.
Min Fire Duration	Float	The minimum amount of time the firearm sprint animation will be paused or sprinting blocked when the trigger is pulled. Prevents rapid tapping of the trigger popping in and out of sprint.

See Also

[Modular Firearms](#)

[The Motion Graph](#)

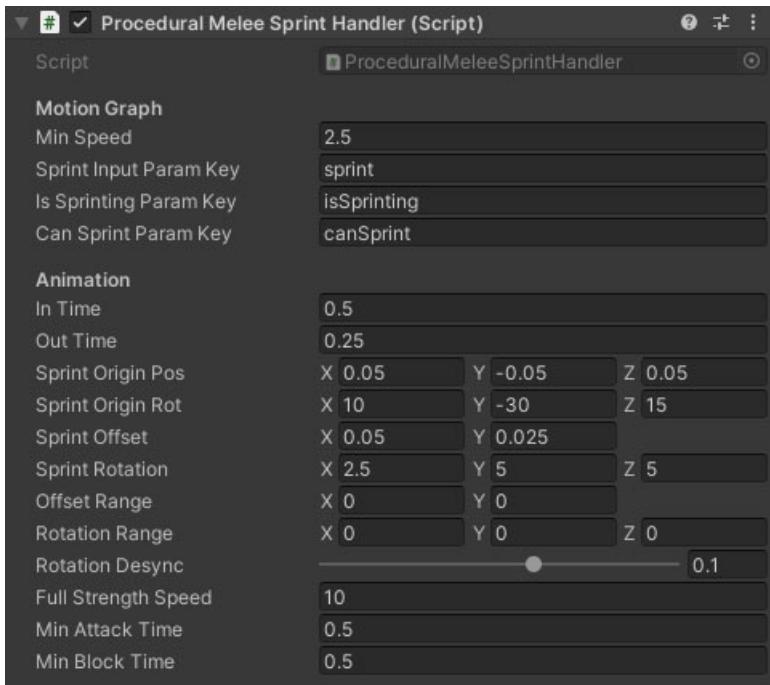
[Additive Transforms & Effects](#)

ProceduralMeleeSprintHandler MonoBehaviour

Overview

The ProceduralMeleeSprintHandler behaviour is attached to a [melee weapon](#) to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an [additive transform](#) bob effect.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.

NAME	TYPE	DESCRIPTION
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Offset Range	Vector	A position offset range either side of the sprint offset that will be randomised with each step to add variety.
Rotation Range	Vector	A rotation offset range either side of the sprint sprint that will be randomised with each step to add variety.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

Melee Weapon

NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the sprint animation will be paused after an attack.
Min Block Time	Float	The minimum amount of time the sprint animation will be paused when blocking (prevents tapping block).

See Also

[Melee Weapons](#)

[The Motion Graph](#)

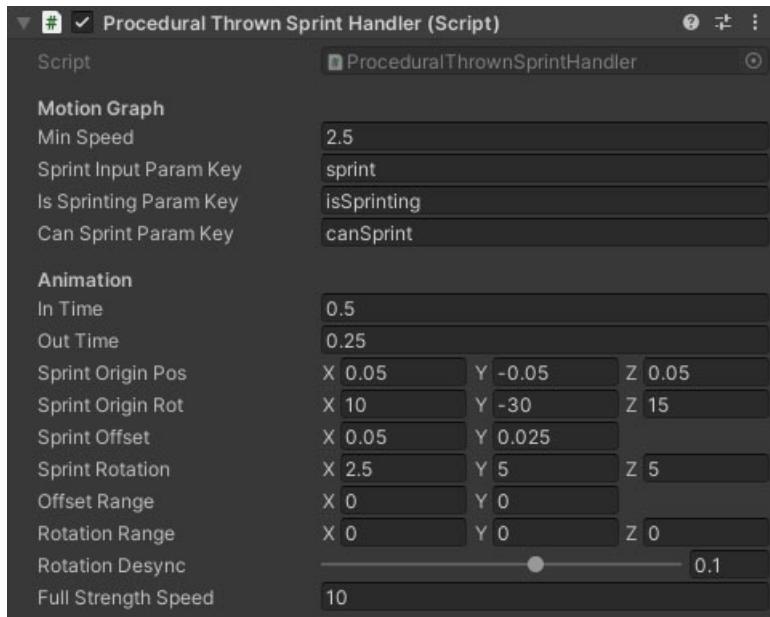
[Additive Transforms & Effects](#)

ProceduralThrownSprintHandler MonoBehaviour

Overview

The ProceduralThrownSprintHandler behaviour is attached to a [thrown weapon](#) to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an [additive transform](#) bob effect.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.

NAME	TYPE	DESCRIPTION
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Offset Range	Vector	A position offset range either side of the sprint offset that will be randomised with each step to add variety.
Rotation Range	Vector	A rotation offset range either side of the sprint sprint that will be randomised with each step to add variety.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

See Also

[Thrown Weapons](#)

[The Motion Graph](#)

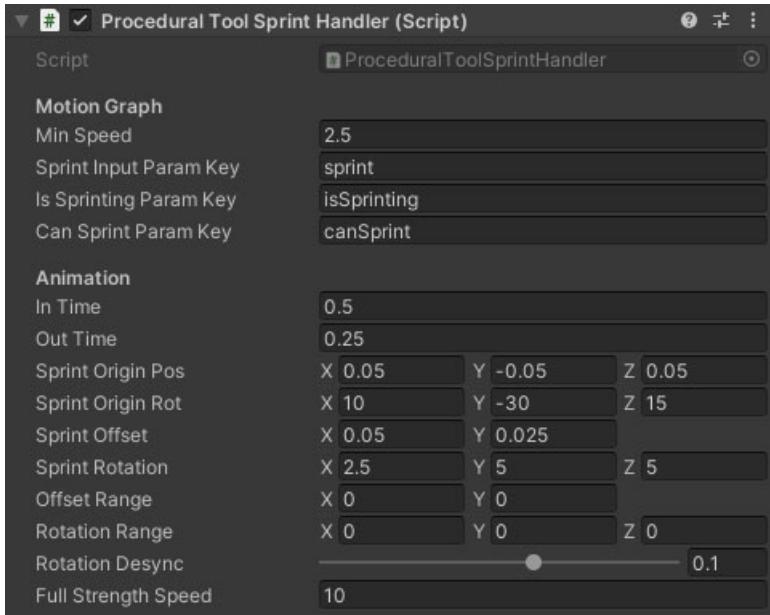
[Additive Transforms & Effects](#)

ProceduralToolSprintHandler MonoBehaviour

Overview

The ProceduralToolSprintHandler behaviour is attached to a [wieldable tool](#) to model different sprinting mechanics. This version uses procedural animation to move the tool into a sprint pose and add an [additive transform](#) bob effect.

Inspector



Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.

NAME	TYPE	DESCRIPTION
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Offset Range	Vector	A position offset range either side of the sprint offset that will be randomised with each step to add variety.
Rotation Range	Vector	A rotation offset range either side of the sprint sprint that will be randomised with each step to add variety.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

See Also

[Wieldable Tools](#)

[The Motion Graph](#)

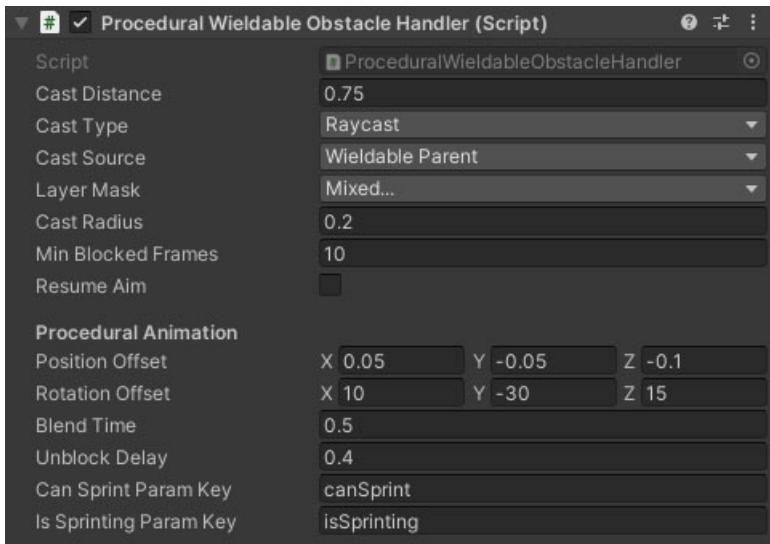
[Additive Transforms & Effects](#)

ProceduralWieldableObstacleHandler MonoBehaviour

Overview

The ProceduralFirearmObstacleHandler behaviour uses the stance system to set an obstructed pose on the player character's weapon whenever it is obstructed by an obstacle in the world. This can be used to pull the weapon back. It will also block the use of the weapon as long as it is obstructed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Cast Distance	Float	The distance in front of the weapon to check for obstacles.
Cast Type	Dropdown	The cast type to use. Options are Raycast and Spherecast .
Cast Source	Dropdown	What to use as the source of the cast. Options are WieldableParent , CharacterAim and Camera .
Layer Mask	LayerMask	What layers are counted as obstacles and make the character move the weapon.
Cast Radius	Float	The radius of the sphere cast (if cast type is set to Spherecast).
Min Blocked Frames	Integer	The minimum number of frames the weapon can be blocked. This prevents the weapon from rapidly switching between blocked and not.
Resume Aim	Boolean	Should the aim be resumed when the firearm is no longer blocked (this will always happen if the aim hold button is pressed).
Position Offset	Vector	The target position offset for the obstructed pose.
Rotation Offset	Vector	The target rotation offset for the obstructed pose.
Blend Time	Float	The time taken to blend in and out of the obstructed pose.

NAME	TYPE	DESCRIPTION
Unblock Delay	Float	The amount of time the weapon should remain blocked from use after the obstruction is removed. This allows the weapon to move back to its idle or aimed pose before allowing use again.
Can Sprint Param Key	String	The
Is Sprinting Param Key	String	The

See Also

[Modular Firearms](#)

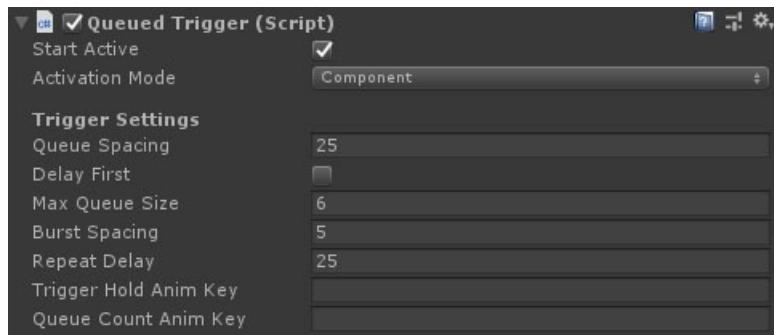
[The Motion Graph](#)

QueuedTrigger MonoBehaviour

Overview

The QueuedTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	<audiosource></audiosource>	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

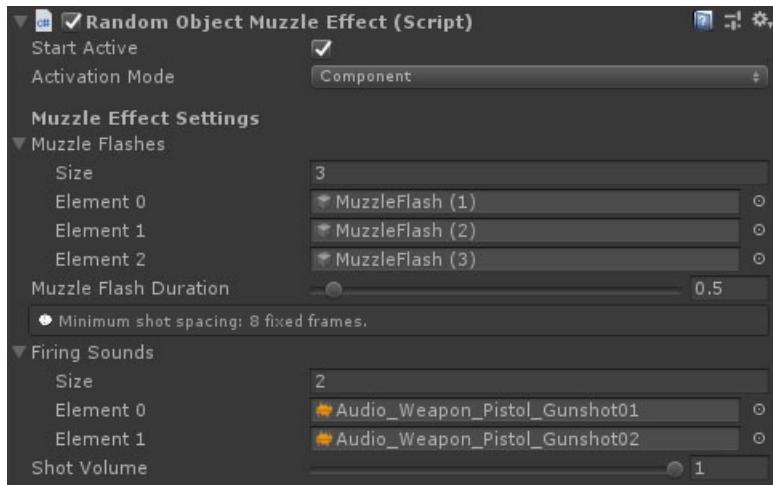
[Modular Firearms](#)

RandomObjectMuzzleEffect MonoBehaviour

Overview

The RandomObjectMuzzleEffect module enables one out of a pool of game objects each shot, and then disables it again after a brief wait.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Muzzle Flashes	GameObject Array	The muzzle flash game objects.
Muzzle Flash Duration	Float	The duration the flash should remain active. Keep this longer for objects with particle effects.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

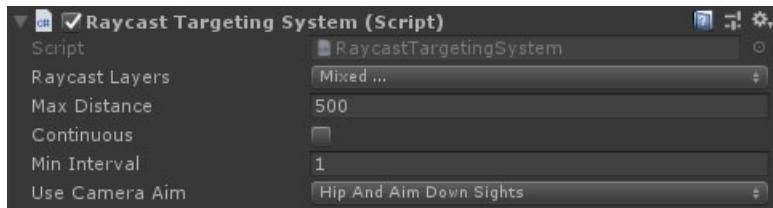
[Modular Firearms](#)

RaycastTargetingSystem MonoBehaviour

Overview

The RaycastTargetingSystem behaviour is a targeting system that passes the current raycast hit point to any guided projectiles that have been registered with it. Those projectiles require a [TargetingSystemTracker](#) component attached.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Raycast Layers	LayerMask	The layers to cast against.
Max Distance	Float	The max distance for targets to home in on.
Continuous	Boolean	Should the targeting system update the projectiles each tick or just pass the initial hit point.
Min Interval	Float	The targeting system will only check for new targets after the min interval has elapsed since the last time, passing the last target to any new trackers before that. This is useful for burst launchers so that each missile tracks the same target.
Use Camera Aim	Dropdown	When set to use camera aim, the targeting system casts forward from the FirstPersonCamera's aim transform. If not then it casts from the transform the targeting system is attached to.

See Also

[Modular Firearms](#)

[Hitscan vs Projectiles](#)

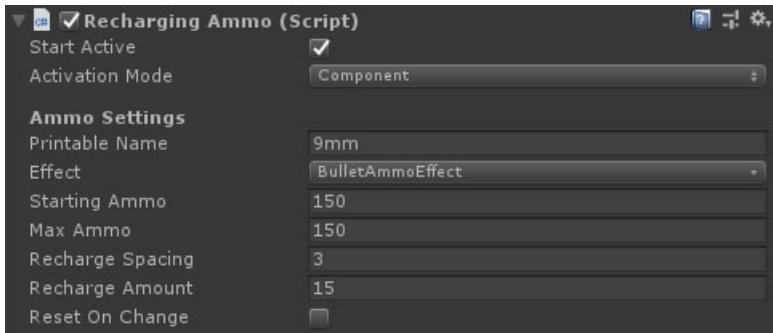
[TargetingSystemTracker Behaviour](#)

RechargingAmmo MonoBehaviour

Overview

The RechargingAmmo module is a unique ammo for the firearm it is attached to.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Starting Ammo	Int	The amount of ammo the weapon starts with.
Max Ammo	Int	The maximum amount of ammo the weapon can carry.

See Also

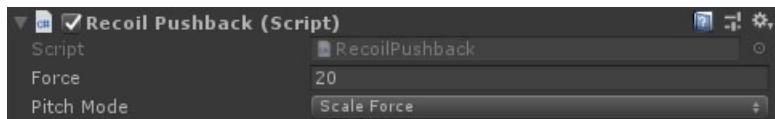
[Modular Firearms](#)

RecoilPushback MonoBehaviour

Overview

The RecoilPushback behaviour connects to a firearm's [recoil](#) module and pushes the wielding character backwards when they fire.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Force	Float	The force to push the character back with.
Pitch Mode	Dropdown	How character aim pitch affects the pushback. Ignore will push the character backwards horizontally with the set force, ScaleForce will push the character back horizontally based on pitch, while Full3D will push the character directly away from their aim direction (including pitch).
Airborne Only	Boolean	Should the recoil only push back when the character is not grounded.

See Also

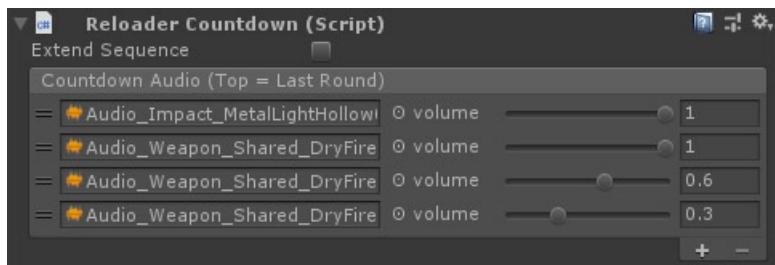
[Modular Firearms](#)

ReloaderCountdown MonoBehaviour

Overview

The ReloaderCountdown behaviour attaches to the firearm's [reloader](#) and tracks the magazine size. As it approaches zero then it plays audio in sequence. This can be used to add a warning that the magazine is almost empty, or for stylistic audio such as an M1 Garand's iconic ping on the last round.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Countdown Audio	Countdown Audio Array	The audio clips to play as ammo is consumed. First element is for the last round in the magazine, second = penultimate, and so on.
Extend Sequence	Boolean	If set, then the last clip will be used for all ammo until the count is within the sequence.

Countdown Audio Element

NAME	TYPE	DESCRIPTION
Clip	AudioClip	The audio clip to play.
Volume	Float	The volume to play the clip at.

See Also

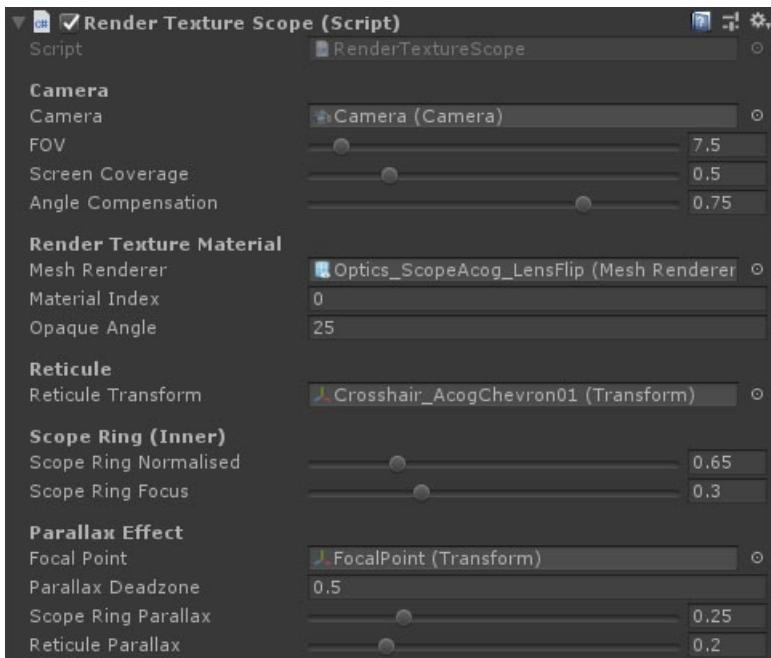
[Modular Firearms](#)

RenderTextureScope MonoBehaviour

Overview

The RenderTextureScope is used to model a weapon scope using a separate camera attached to the weapon. The shader provided with NeoFPS provides parallax effects that move the scope ring and reticule as the camera viewing the scope lens goes off the scope's axis, and fades the lens to an opaque glass at the extremes (disabling the scope camera when fully opaque). Render texture based scopes incur a performance penalty because they require rendering the scene twice, along with communicating back and forth with the GPU each frame.

Inspector



Properties

The RenderTextureScope properties are split into a number of sections:

Camera

Name	Type	Description
Camera	Camera	The scope camera.
FOV	Float	The field of view for the scope camera.
Screen Coverage	Float	The amount of the screen height the scope takes up (multiplier). Smaller screen coverage needs a smaller render texture and therefore performs better.
Angle Compensation	Float	Rotate the camera to adapt to the parallax effect. At 1, the image will track (roughly) with the scope ring. At 0, the scope ring will be completely detached from the image.

Render Texture Material

Name	Type	Description
Mesh Renderer	MeshRenderer	The mesh renderer of the scope lens which will receive the render texture.

NAME	TYPE	DESCRIPTION
Material Index	Integer	The material index is used to specify which material to modify from a multi-part mesh.
Opaque Angle	Float	The angle in degrees off the axis of the scope where the camera stops rendering and the material becomes completely opaque.

Reticule

NAME	TYPE	DESCRIPTION
Reticule Transform	Transform	The transform of the reticule geometry. Local position (0,0,0) should be centered on the camera and slightly in front.

Scope Ring (Inner)

NAME	TYPE	DESCRIPTION
Scope Ring Normalised	Float	The start of the scope ring in terms of the normalised radius. At 1, the ring is a circle that touches the edges of the texture.
Scope Ring Focus	Float	The focus level of the scope ring. Setting this lower will expand the blur of the ring out from the normalised start radius.

Parallax Effect

NAME	TYPE	DESCRIPTION
Focal Point	Transform	A transform that represents the lens of the scope that the player looks into. Used to determine the off-axis angle for parallax and fade. This should be placed near the lens of the scope.
Parallax Deadzone	Float	An angle range from the scope axis where there is no parallax. This prevents the reticule from going off center during small movements like idle animations or procedural breathing.
Scope Ring Parallax	Float	The amount of parallax to add to the scope ring as the eye-line moves away from the scope axis.
Reticule Parallax	Float	The amount of parallax to add to the reticule as the eye-line moves away from the scope axis. Values above the scope ring parallax will make the reticule seem like it is further forward than the ring.

See Also

[Modular Firearms](#)

[Scopes & Optics](#)

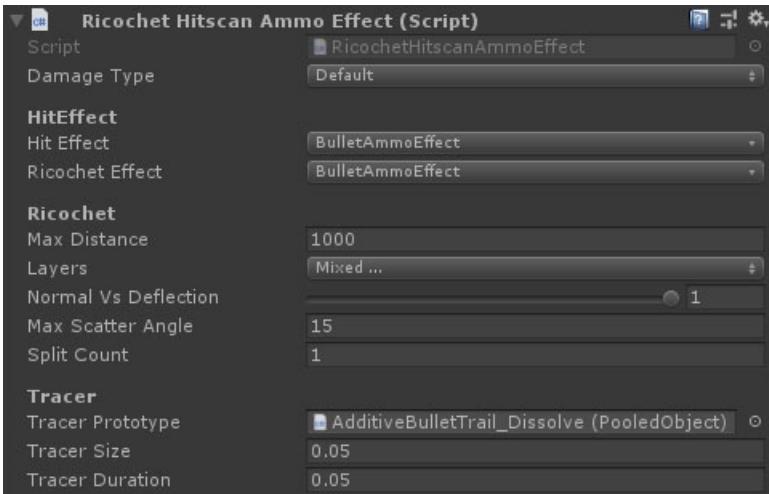
RicochetHitscanAmmoEffect MonoBehaviour

Overview

The RicochetHitscanAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the distance travelled. If you are using a projectile shooter, then there is a separate [RicochetProjectileAmmoEffect](#) that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector



Properties

Name	Type	Description
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Distance	Float	The maximum distance that the weapon will register a hit (includes the distance travelled up to the penetration).
Layers	LayerMask	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Split Count	Integer	The number of shots to split the ricochet into (1 is no split).
Tracer Prototype	[PooledObject] [4]	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface)
Tracer Size	Float	How size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long the tracer line will last.

See Also

[Modular Firearms](#)

[Surfaces](#)

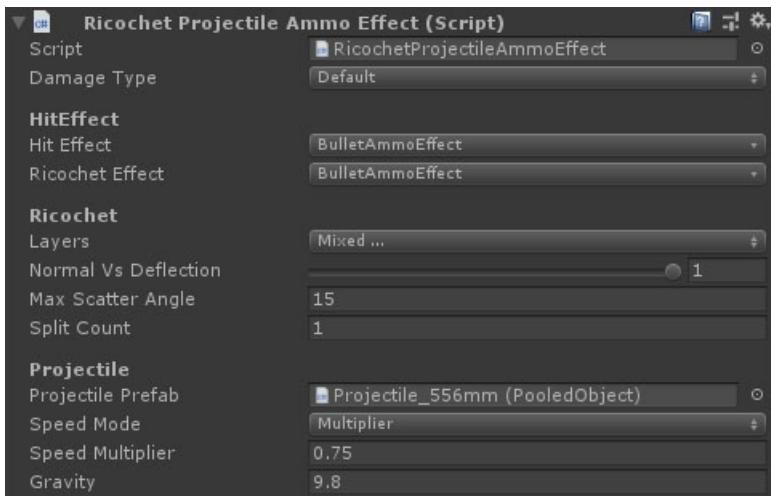
RicochetProjectileAmmoEffect MonoBehaviour

Overview

The RicochetProjectileAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the speed the projectile is travelling. If you are using a hitscan shooter, then there is a separate [RicochetHitscanAmmoEffect](#) that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector



Properties

Name	Type	Description
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Layers	LayerMask	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Split Count	Integer	The number of shots to split the ricochet into (1 is no split).
Projectile Prefab	PooledObject	The projectile to spawn.

Name	Type	Description
Speed Mode	Dropdown	How the ricochet speed of the projectile is calculated. Multiplier multiplies the contact speed by a set value, FixedSpeed uses a set speed, AngleBasedMultiplier multiplies the contact speed by a value based on the contact angle, AngleBasedSpeed uses a set speed based on the contact angle.
Speed Multiplier	Float	A multiplier applied to the bullet speed after ricochet. Only visible when Speed Mode is set to Multiplier .
Fixed Speed	Float	The speed of the projectile after ricochet. Only visible when Speed Mode is set to FixedSpeed .
Straight On Multiplier	Float	A multiplier applied to the bullet speed after ricochet when the entry velocity was perpendicular to the surface. Only visible when Speed Mode is set to AngleBasedMultiplier .
Glancing Multiplier	Float	A multiplier applied to the bullet speed after ricochet if the initial bullet was travelling parallel to the surface. Only visible when Speed Mode is set to AngleBasedMultiplier .
Straight On Speed	Float	The speed of the projectile after ricochet when the entry velocity was perpendicular to the surface. Only visible when Speed Mode is set to AngleBasedSpeed .
Glancing Speed	Float	The speed of the projectile after ricochet if the initial bullet was travelling parallel to the surface. Only visible when Speed Mode is set to AngleBasedSpeed .
Gravity	Float	The gravity for the projectile.

See Also

[Modular Firearms](#)

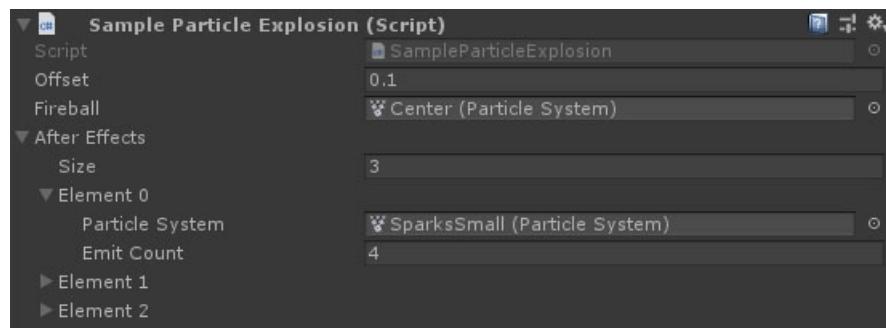
[Surfaces](#)

SampleParticleExplosion MonoBehaviour

Overview

The SampleParticleExplosion behaviour is a basic implementation of an explosion effect involving a central fireball and a number of after effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Offset	Float	The distance along the normal to offset the explosion (for raising from a surface).
Fireball	ParticleSystem	The central fireball of the explosion.
After Effects	After Effects Array	Any extra emitters for explosion after effects (smoke, sparks, etc).

After Effects

NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The after effect particle system.
Emit Count	Int	The amount of particles to emit.

See Also

[Explosions](#)

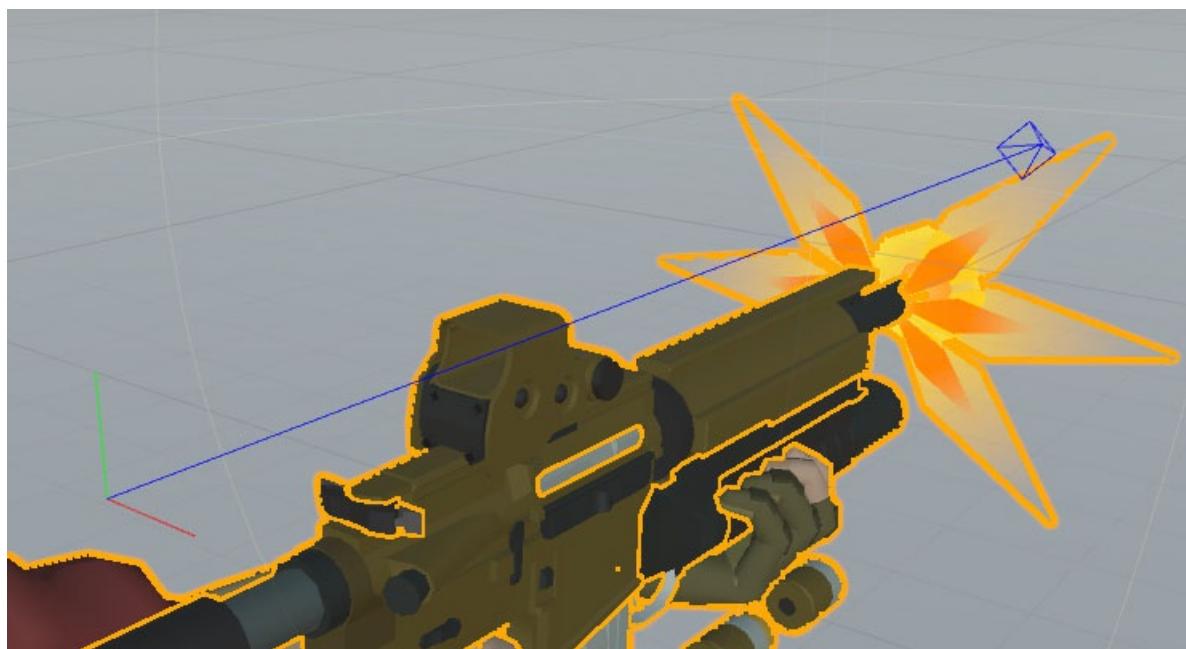
[Unity ParticleSystem](#)

ScopedAimer MonoBehaviour

Overview

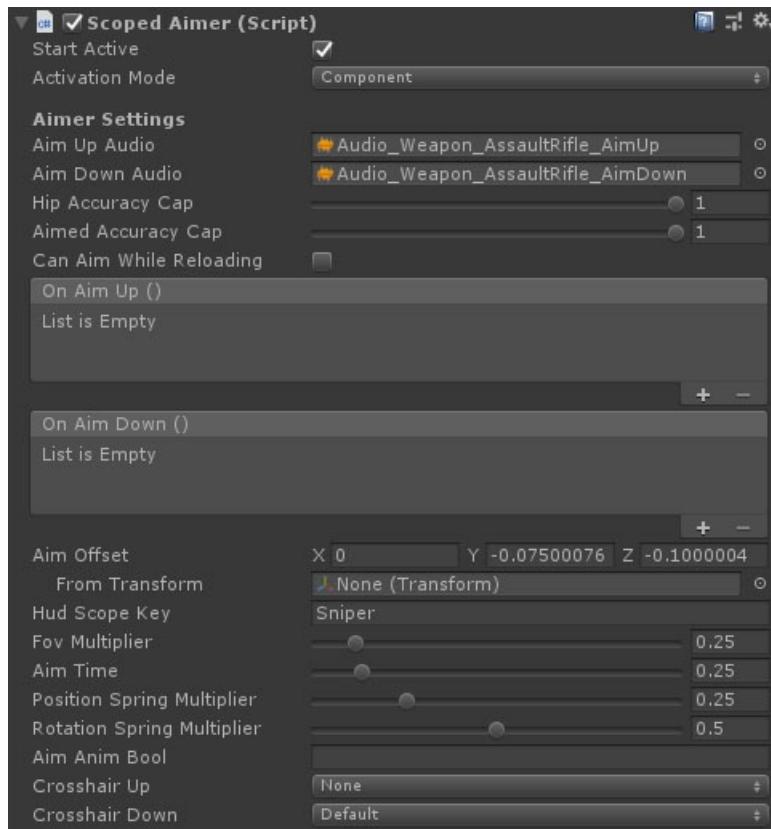
The ScopedAimer module will raise the weapon and then switch to a [HUD Scope](#) when the weapon is fully raised.

In The Scene View



With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimmer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Offset	Vector3	The offset for the root transform to move to align the weapon sights with the camera.
From Transform	Transform	Dragging a transform in here will calculate the offset from the root transform to this transform, and set the Aim Offset property with the result
Hud Scope Key	String	The key for the specific scope to show.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.

NAME	TYPE	DESCRIPTION
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

[Modular Firearms](#)

[First Person Camera](#)

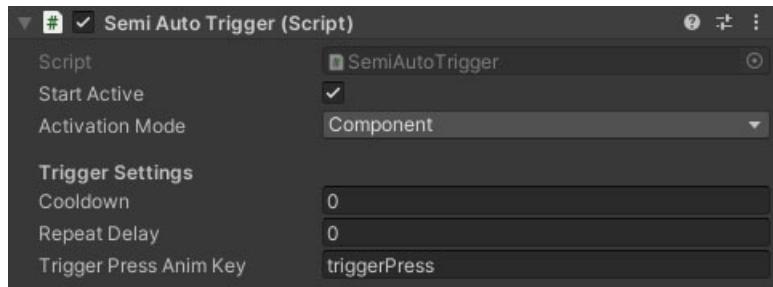
[HudScope](#)

SemiAutoTrigger MonoBehaviour

Overview

The SemiAutoTrigger fires a single shot as fast as the player can press the trigger. If the trigger is held down it will also repeat at a slow rate.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Cooldown	Int	Cooldown between trigger pulls (number of fixed update frames).
Repeat Delay	Int	How many fixed update frames before firing again (0 = requires fresh trigger press).
Trigger Press Anim Key	String	The trigger animator property key to set when the trigger is pressed.

See Also

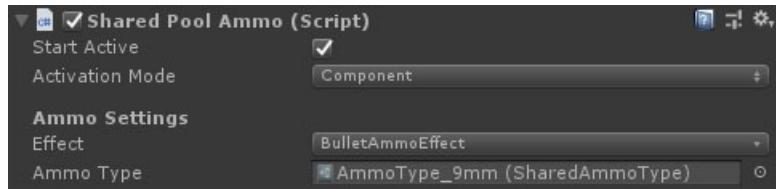
[Modular Firearms](#)

SharedPoolAmmo MonoBehaviour

Overview

The SharedPoolAmmo module uses ammo that is stored in the character inventory and can be shared between weapons.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Ammo Type	SharedAmmoType	The ammo type to use.

See Also

[Modular Firearms](#)

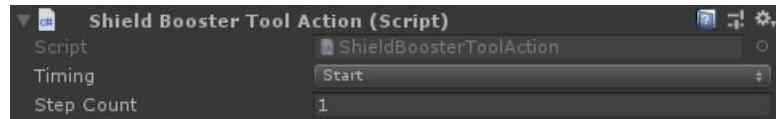
[SharedAmmoType](#)

ShieldBoosterToolAction MonoBehaviour

Overview

The ShieldBoosterToolAction behaviour is used to add an action that restores a character's shield slots to a [wieldable tool](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the tool apply the shield boost.
Step Count	Integer	The number of shield steps to recharge.

See Also

[Wieldable Tools](#)

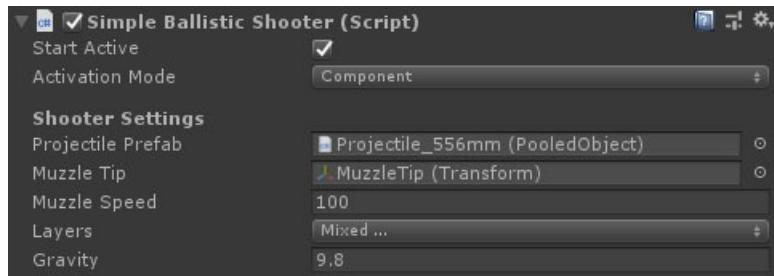
[Health and Damage](#)

SimpleBallisticShooter MonoBehaviour

Overview

The SimpleBallisticShooter module spawns an [BallisticProjectile](#). It is not affected by accuracy, and has no options to use the camera aim target. If you want these extra features then you can alternatively use the [BallisticShooter][4]

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The position and direction the projectile is spawned.
Layers	LayerMask	The physics collision layers the shot can hit.
Muzzle Speed	Float	The speed of the projectile.
Layers	LayerMask	The layers that will be checked against when casting for valid interaction targets.
Gravity	Float	The gravity for the projectile.

See Also

[Modular Firearms](#)

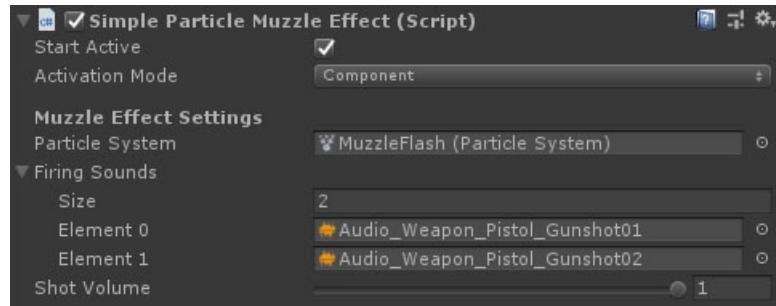
[BallisticProjectile](#)

SimpleParticleMuzzleEffect MonoBehaviour

Overview

The SimpleParticleMuzzleEffect behaviour spawns triggers a particle system to emit each shot.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The particle system to play.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

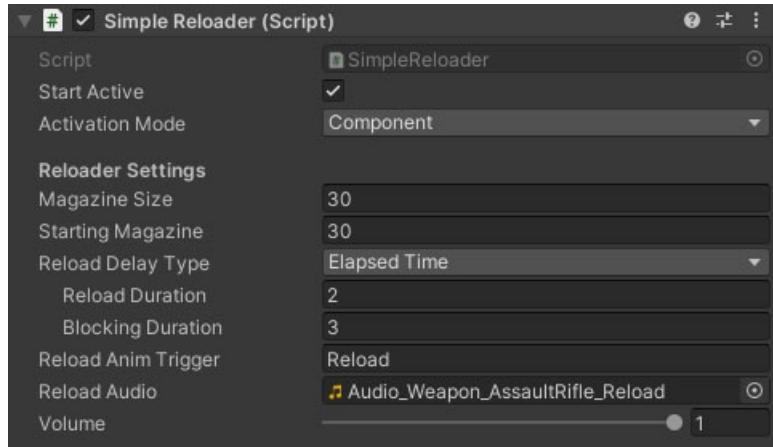
[Modular Firearms](#)

SimpleReloader MonoBehaviour

Overview

The SimpleReloader behaviour triggers a reload animation and waits until it completes before taking ammo from the firearm's ammo module and adding it to the magazine.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Reload Delay Type	Dropdown	The delay type between starting and completing a reload. The options are None , Elapsed Time , External Trigger .
Reload Duration*	Float	The time taken to reload.
Blocking Duration*	Float	The time taken before you can use the weapon again after the reload starts.
Reload Anim Trigger	String	The AnimatorController trigger key for the reload animation.
Reload Audio	AudioClip	The audio clip to play while reloading.

* This property is only visible if the reload delay type is set to ElapsedTime.

See Also

[Modular Firearms](#)

[FirearmAnimEventsHandler](#)

[Unity Animator](#)

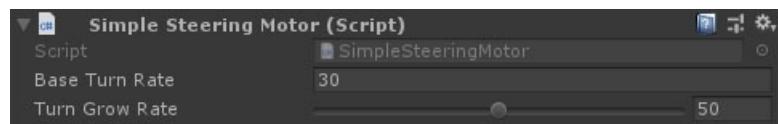
[Unity AnimatorController](#)

SimpleSteeringMotor MonoBehaviour

Overview

The SimpleSteeringMotor behaviour is used as part of a guided projectile to steer and move it towards its target. As time goes on its turning circle improves to prevent it getting stuck orbiting its target.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Base Turn Rate	Float	The turn rate of the projectile (degrees per second).
Turn Grow Rate	Float	An increase to the turn rate based on elapsed time in flight (base turn rate + turn grow rate * elapsed time).

See Also

[Modular Firearms](#)

[Hitscan vs Projectiles](#)

SpreadBallisticShooter MonoBehaviour

Overview

The SpreadShooter module is in effect a number of [BallisticShooter](#) modules in one.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Speed	Float	The speed of the projectile.
Gravity	Float	The gravity for the projectile.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .

NAME	TYPE	DESCRIPTION
Bullet Count	Int	How many pellets are fired each shot.
Cone	Float	The spread of the cone in degrees.

See Also

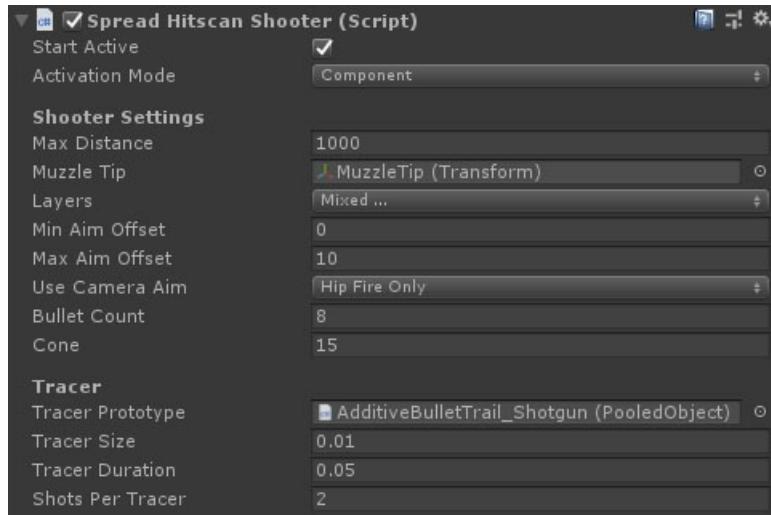
[Modular Firearms](#)

SpreadHitscanShooter MonoBehaviour

Overview

The SpreadHitscanShooter module is in effect a number of [HitscanShooter](#) modules in one.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: HipFireOnly , HipAndAimDownSights , AimDownSightsOnly , Never .
Bullet Count	Int	How many pellets are fired each shot.
Cone	Float	The spread of the cone in degrees.

NAME	TYPE	DESCRIPTION
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).
Tracer Size	Float	The size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long should the tracer object stay visible.
Shots Per Tracer	Int	How many pellets are required per tracer line.

See Also

[Modular Firearms](#)

SpringRecoilHandler MonoBehaviour

Overview

The SpringRecoilHandler module uses the [additive transform](#) system to move and rotate the firearm as it recoils.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
On Recoil	UnityEvent	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).

NAME	TYPE	DESCRIPTION
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).
Weapon Kicker	AdditiveKicker	The additive kicker behaviour on the object.
Weapon Wander	Float	How much of the weapon recoil rotation is directed sideways instead of up.
Weapon Rotation	Float	The rotation angle of the weapon recoil while firing from the hip. At 0 wander, this is how many degrees the weapon pitches up.
Weapon Rotation Aimed	Float	The rotation angle of the weapon recoil while aimed. At 0 wander, this is how many degrees the weapon pitches up.
Weapon Knock Back	Float	Knock-back is movement backwards, towards the camera.
Head Wander	Float	How much of the head recoil rotation is directed sideways instead of up. At negative values this will be the opposite of the guns sideways rotation.
Head Rotation	Float	The head rotation angle of the weapon recoil when firing from the hip. At 0 wander, this is how many degrees the weapon pitches up.
Head Rotation Aimed	Float	The head rotation angle of the weapon recoil when aimed. At 0 wander, this is how many degrees the weapon pitches up.
Bypass Move Multiplier	Boolean	Should the movement recoil effect be affected by the weapon and head spring multipliers? These affect everything, from bob to impacts to shake. If the multiplier is at zero then the recoil will be disabled either way.
Bypass Rotate Multiplier	Boolean	Should the rotation recoil effect be affected by the weapon and head spring multipliers? These affect everything, from bob to impacts to shake. If the multiplier is at zero then the recoil will be disabled either way.
Rotation Duration	Float	The amount of time the angle recoil effect takes to return to zero.
KnockBack Duration	Float	The amount of time the knockback effect takes to return to zero.

See Also

[Modular Firearms](#)

[Additive Transforms and Effects](#)

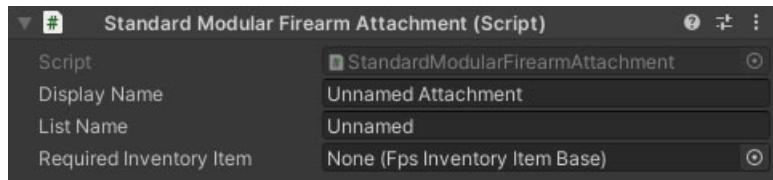
StandardModularFirearmAttachment MonoBehaviour

Overview

The StandardModularFirearmAttachment behaviour is an example implementation of the `ModularFirearmAttachment` base class which allows you to specify the attachment's name. You can also optionally specify a required inventory item. If this is specified, then the character will need to have an inventory item with the same ID in their inventory in order to use this attachment.

An attachment can be installed into a [socket](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Display Name	String	The name that will be used in an attachments UI where the type is unspecified (eg "High Capacity Magazine").
List Name	String	A shorter name that might be used in a list where the type is already known (eg "High Capacity" when the list is all magazines).
Required Inventory Item	Inventory Item	(Optional) If this is set then the character must have an instance of this item (or an item with matching inventory ID) in their inventory before attaching to their weapon.

See Also

[Modular Firearms](#)

[Modular Firearm Attachments](#)

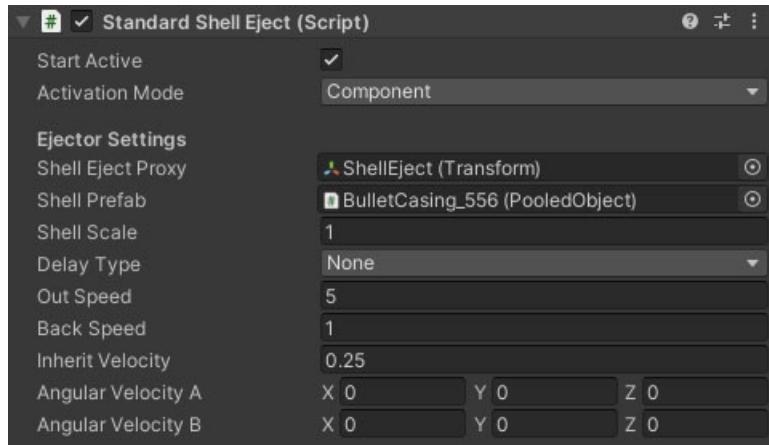
[Inventory](#)

StandardShellEject MonoBehaviour

Overview

The StandardShellEject behaviour launches a pooled bullet casing object away from the specified transform.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Shell Eject Proxy	Transform	A proxy transform where ejected shells will be spawned.
Shell Prefab	PooledObject	The shell prefab object to spawn.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are None , Elapsed Time , External Trigger .
Delay*	Float	The delay time between firing and ejecting a shell.
Out Speed	Float	The ejected shell speed directly out from the ejector.
Back Speed	Float	The ejected shell speed back over the wielder's shoulder.
Inherit Velocity	Float	How much of the character's velocity should be added to the ejected shells.
Angular Velocity A	Vector3	The minimum angular velocity on each axis (will be picked at random between this an B).

NAME	TYPE	DESCRIPTION
Angular Velocity B	Vector3	The maximum angular velocity on each axis (will be picked at random between this an A).
Shell Scale	Float	The scale to be applied to the rigidbody shell casings (particles are controlled in the particle system).
Direction Spread	Float	The maximum angle variation from the out direction that the shell will be ejected (prevents empty shells landing in the same place).

* This property will only be visible if the delay type is set to Elapsed Time

See Also

[Modular Firearms](#)

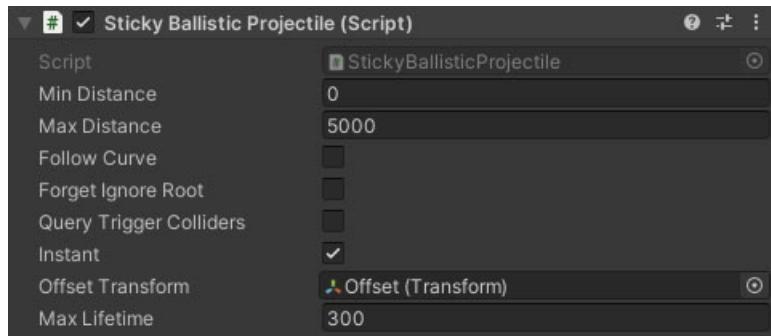
[PooledObject](#)

StickyBallisticProjectile MonoBehaviour

Overview

The StickyBallisticProjectile is a basic projectile that follows gravity. It will be given an ammo effect by the firearm that it uses when it impacts.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Max Distance	Float	The maximum distance, after the projectile will disappear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Query Trigger Colliders	Boolean	Should the shot be tested against trigger colliders.
Instant	Boolean	This will cause the projectile logic to be calculated instantly on firing. This will be more responsive, but bullet trails, etc may start far ahead of the gun depending on muzzle speed. Works best in first person.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).

See Also

[Modular Firearms](#)

[BallisticShooter](#)

SurfaceBulletPhysicsAmmoEffect MonoBehaviour

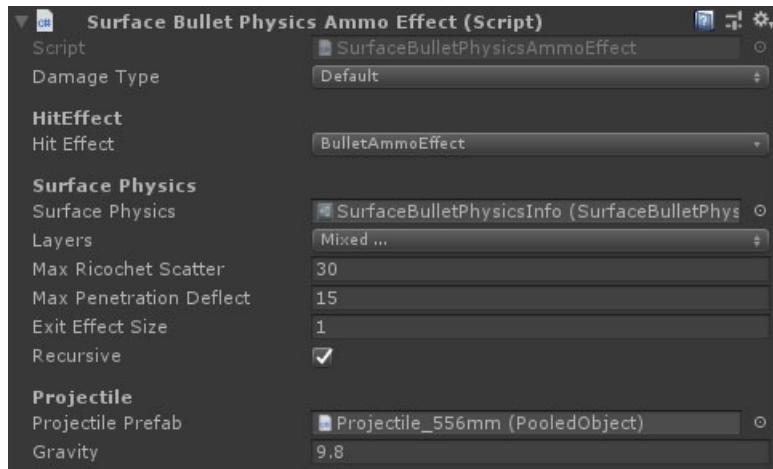
Overview

The SurfaceBulletPhysicsAmmoEffect module allows bullets to ricochet and penetrate objects based on the [surface](#). This ammo effect is used alongside another effect to apply damage and visual elements. Since both ricochets and penetrations reduce the speed of the resulting projectile, this effect works well when paired with the [AdvancedBulletAmmoEffect](#) which allows damage to fall-off with speed.

The penetration and ricochet settings for each surface are specified using a [SurfaceBulletPhysicsInfo](#) scriptable object.

This module only works with projectile based shooters.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo (does nothing for this effect).
Hit Effect	BaseAmmoEffect	The effect of the ammo when it hits something.
Surface Physics	SurfaceBulletPhysicsInfo	The per-surface bullet physics info
Layers	[LayerMask][unity-layers]	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Ricochet Scatter	Float	Randomises the deflected bullet direction within this cone angle, dependent on surface settings.
Max Penetration Deflect	Float	Randomises the penetrating bullet direction within this cone angle, dependent on surface settings.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Recursive	Boolean	Should the bullet keep ricocheting / penetrating after the first time until it has slowed or travelled far enough.

NAME	TYPE	DESCRIPTION
Projectile Prefab	PooledObject	The projectile to spawn.
Gravity	Float	The gravity for the projectile.

See Also

[Modular Firearms](#)

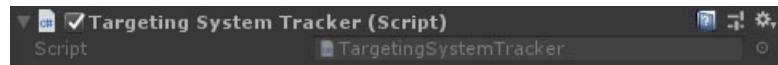
[Surfaces](#)

TargetingSystemTracker MonoBehaviour

Overview

The TargetingSystemTracker behaviour is a guided projectile tracking system which defers its target tracking to a targeting system such as a [TargetLockTrigger](#) or [LaserTargetingSystem](#).

Inspector



Properties

The TargetingSystemTracker behaviour has no properties exposed in the inspector.

See Also

[Modular Firearms](#)

[Hitscan vs Projectiles](#)

TargetLockTrigger MonoBehaviour

Overview

The TargetLockTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector



Properties

Name	Type	Description
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

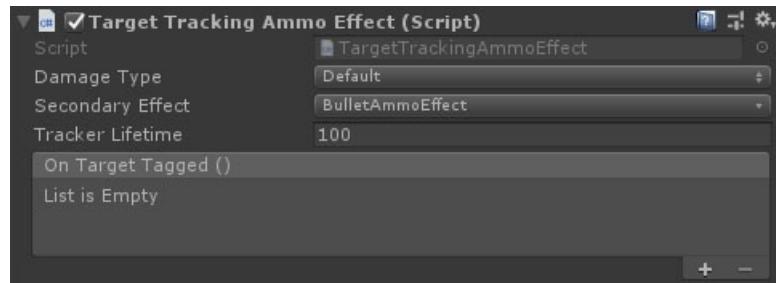
[Modular Firearms](#)

TargetTrackingAmmoEffect MonoBehaviour

Overview

The TargetTrackingAmmoEffect behaviour is used to tag targets for [guided projectiles](#) to home in on. The projectiles must use a [TargetingSystemTracker](#) component.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Type	Dropdown	The type of damage the impact counts as. This is a common property for all ammo effects and actually has no effect in this case.
Secondary Effect	AmmoEffect	An optional secondary ammo effect to allow the tracking bullet to deal damage, etc.
Tracker Lifetime	Float	The amount of time (seconds) the tracking effect will last for.
On Target Tagged	UnityEvent	An event fired when an object is successfully tagged. You can use this to perform actions like switching ammo effects.

See Also

[Modular Firearms](#)

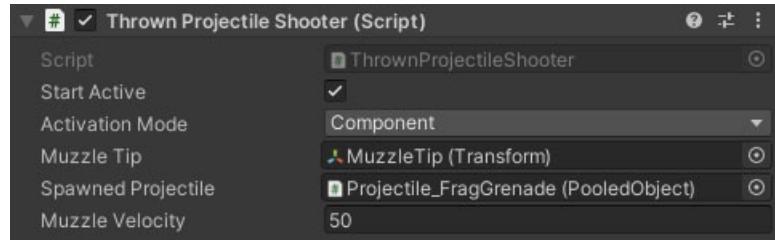
[Hitscan vs Projectiles](#)

ThrownProjectileShooter MonoBehaviour

Overview

The ThrownProjectileShooter module allows a firearm to shoot the kind of projectiles usually used in [thrown weapons](#). This makes it easier to use rigidbodies and bouncing or timed projectiles such as frag grenades.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Muzzle Tip	Transform	A proxy transform for setting the position and rotation of the spawned projectile.
Spawned Projectile	PooledObject	The projectile prefab to spawn. For examples, see the thrown weapons reference .
Muzzle Velocity	Float	The starting speed of the projectile (in the forward direction of the muzzle tip).

See Also

[Modular Firearms](#)

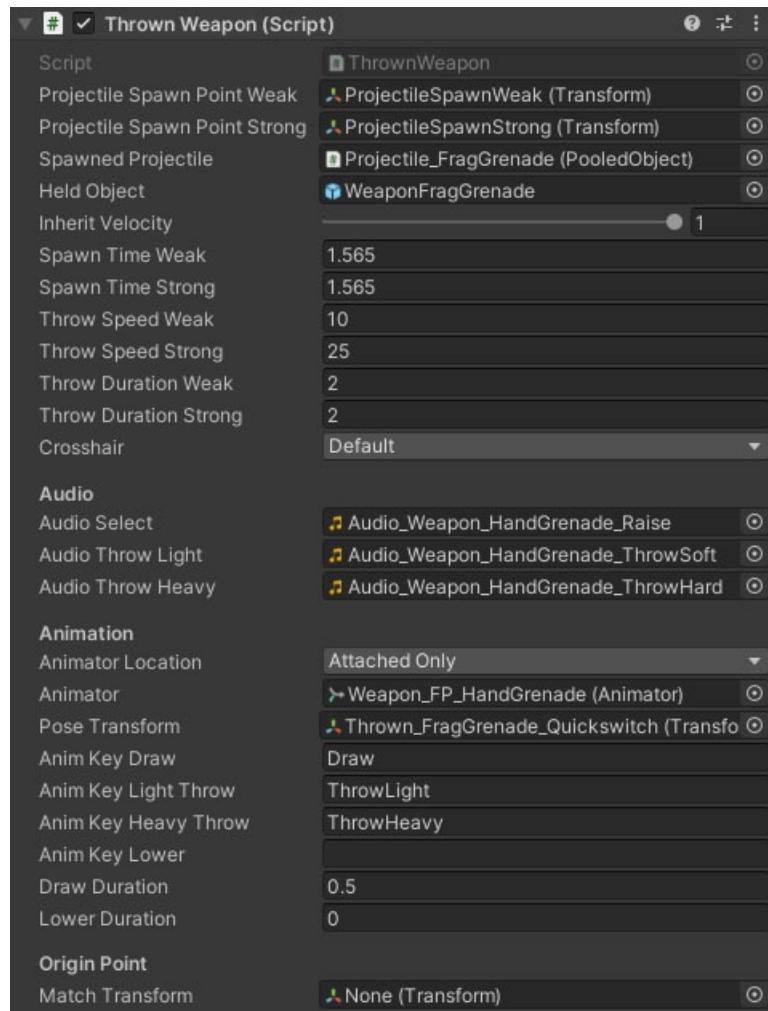
[Thrown Weapons](#)

ThrownWeapon MonoBehaviour

Overview

The ThrownWeapon behavior is used for weapons such as hand grenades.

Inspector



Properties

Name	Type	Description
Animator Location	Dropdown	What animators should be exposed to the weapon components. Options are None , AttachedOnly , AttachedAndCharacter , MultipleAttached , MultipleAttachedAndCharacter and CharacterOnly .
Animator	Animator	The animator component of the weapon.
Pose Transform	Transform	The transform that should be offset for weapon poses.
Projectile Spawn Point Weak	Transform	A proxy transform for setting the position and rotation of the spawned projectile (weak throw).
Projectile Spawn Point Strong	Transform	A proxy transform for setting the position and rotation of the spawned projectile (strong throw).

Name	Type	Description
Spawned Projectile	PooledObject	The prefab to throw.
Held Object	GameObject	The weapon game object. This is deactivated and swapped with the pooled object during the throw animation.
Inherit Velocity	Float	How much of the character velocity does the thrown weapon inherit (think Counter Strike).
Spawn Time Weak	Float	The point in the animation (seconds) to swap the animated weapon with the pooled physics weapon (weak throw).
Spawn Time Strong	Float	The point in the animation (seconds) to swap the animated weapon with the pooled physics weapon (strong throw).
Throw Speed Weak	Float	The throw speed of the projectile (weak throw).
Throw Speed Strong	Float	The throw speed of the projectile (strong throw).
Throw Duration Weak	Float	The full duration of the weak throw animation.
Throw Duration Strong	Float	The full duration of the strong throw animation.
Draw Duration	Float	The time it takes to raise the weapon.
Crosshair	FpsCrosshair	The crosshair to show when the weapon is drawn.
Audio Rotator	AudioRotator	An audio rotator for weapon noises.
Audio Select	AudioClip	The audio clip when raising the weapon.
Audio Throw Light	AudioClip	The audio clip for a weak throw.
Audio Throw Heavy	AudioClip	The audio clip for a strong throw.
Animator	Animator	The animator component of the weapon.
Anim Key Draw	String	The key for the AnimatorController trigger property that triggers the draw animation.
Anim Key Light Throw	String	The key for the AnimatorController trigger property that triggers the light throw animation.

NAME	TYPE	DESCRIPTION
Anim Key Heavy Throw	String	The key for the AnimatorController trigger property that triggers the heavy throw animation.
Anim Key Lower	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

[Thrown Weapons](#)

[PooledObject](#)

[Unity Animator](#)

[Unity AnimatorController](#)

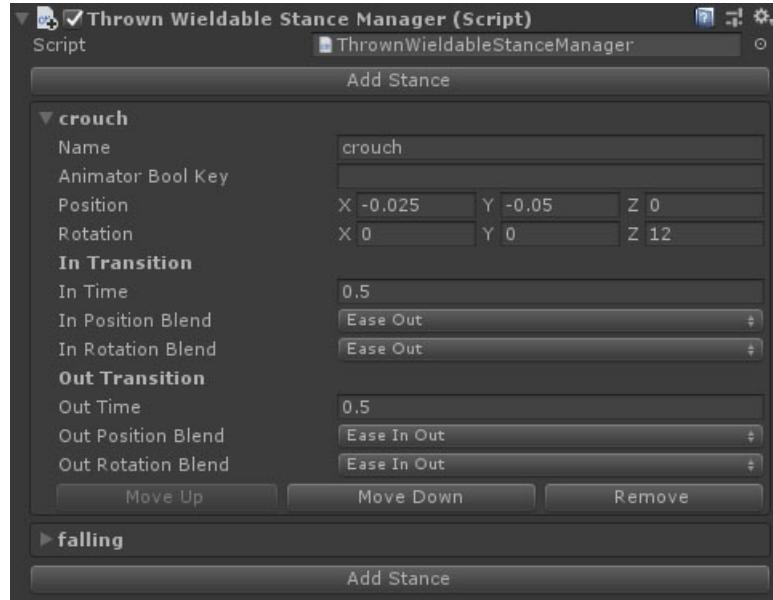
ThrownnWieldableStanceManager MonoBehaviour

Overview

The ThrownnWieldableStanceManager behaviour is used to specify poses or stances for a thrown weapon. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be exited temporarily whenever the weapon is thrown.

Inspector



Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator .
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
Out Time	Float	The time taken to enter the stance.

See Also

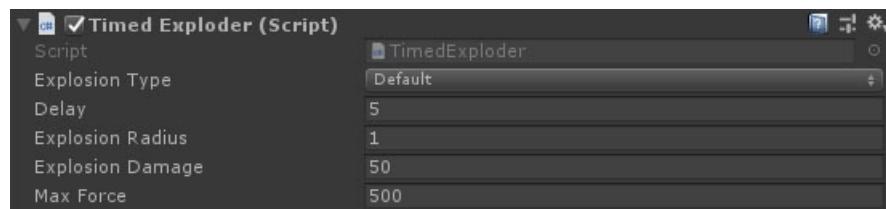
[Thrown Weapons](#)

TimedExploder MonoBehaviour

Overview

The TimedExploder behaviour is attached to such as hand grenades, spawning an explosion after a set time. The item should be a [PooledObject](#) and will be returned to the pool when the explosion is spawned.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Explosion Type	ExplosionType	The id of the explosion to spawn.
Delay		The delay before exploding.
Explosion Radius		The radius of the explosion.
Explosion Damage		The damage the explosion does at its center.
Max Force		The max force to be imparted onto any objects in the explosion radius. The force falls off as distance from the center increases. Requires either a Rigidbody or an impact handler.

See Also

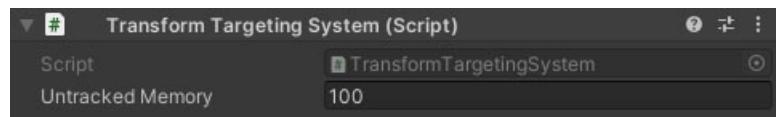
[Explosions](#)

TransformTargetingSystem MonoBehaviour

Overview

The TransformTargetingSystem behaviour is a tracking system behaviour that is used by guided projectiles to pick a target. The target transform will need to be assigned to this behaviour via the API. An example of this in use is the [DelegatedTargetTrackingAmmoEffect](#) which will assign the target it hits to the TransformTargetingSystem it is connected to. This allows one firearm to set the tracking target on another.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Untracked Memory	Float	The amount of time (seconds) the target will be remembered for when there is nothing tracking it.

See Also

[Modular Firearms](#)

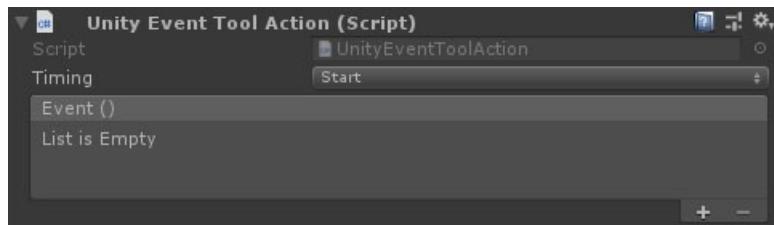
[Hitscan vs Projectiles](#)

UnityEventToolAction MonoBehaviour

Overview

The UnityEventToolAction behaviour is used to add a [Unity event](#) to a [wieldable tool](#) to aid in connecting to other systems.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the event be triggered.
Event	UnityEvent	The event to fire.

See Also

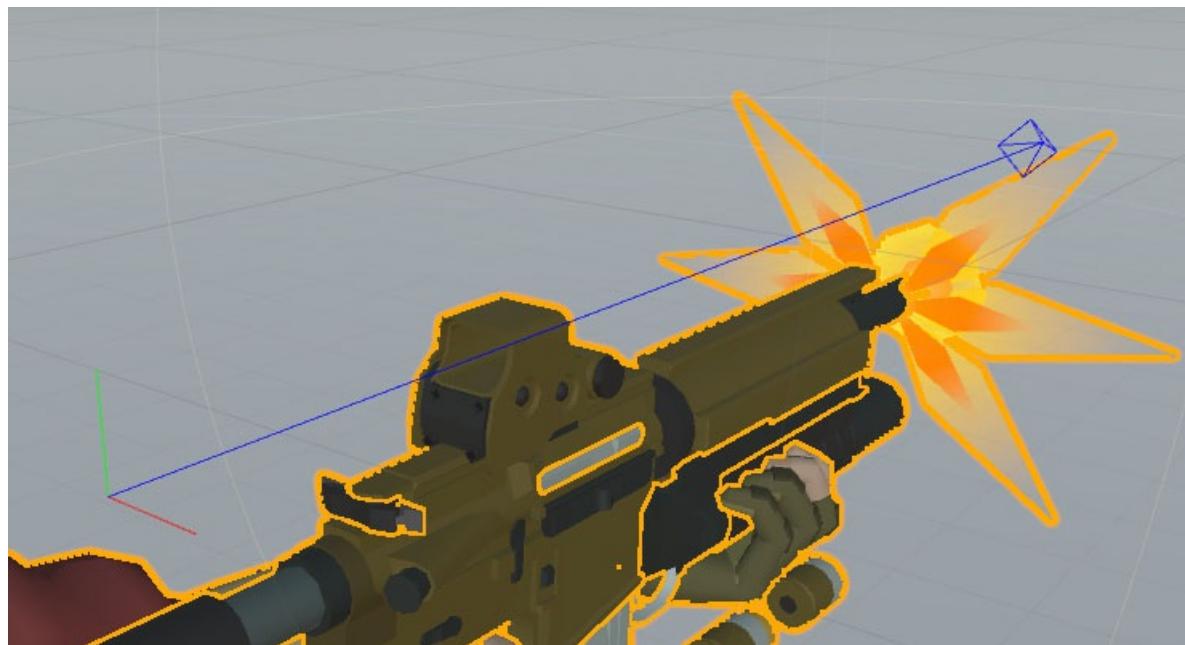
[Wieldable Tools](#)

WeaponMoveAimer MonoBehaviour

Overview

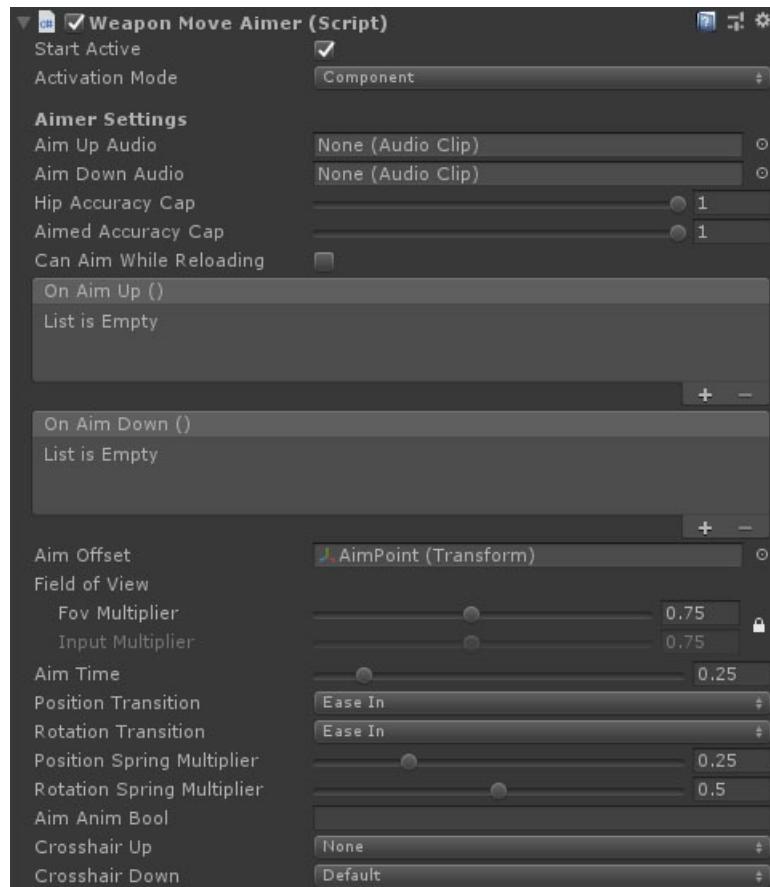
The WeaponMoveAimer module moves the entire weapon up to align the center of the camera with the weapon sights.

In The Scene View



With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aim immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: Component means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. GameObject activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Offset	Transform	A target aim transform. The weapon will be moved to align this transform to the camera when aiming down sights. Offsets are calculated on Awake, so moving this transform after this point has no effect.
Aim Position	Vector3	The offset for the weapon transform to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Aim Rotation	Vector3	An euler angle offset for the weapon to move to align the weapon sights with the camera. This property will only be visible if no AimOffset transform is set.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Input Multiplier	Float	A multiplier for the camera input when aiming down sights. By default, this value is locked to the FoV multiplier. Clicking the lock icon next to the 2 multipliers allows you to modify it separately for situations such as render texture scopes.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.

Name	Type	Description
Transition	Dropdown	The transitions easing to apply. Note: custom transition requires inheriting a new class and overriding the custom transition methods. Lerp moves between states with uniform speed, SwingUp moves across and then up, SwingAcross moves up and then across, EaseInOut smoothes the beginning and end of the transition for a more natural feel, Overshoot moves past the target position and then back, OvershootIn overshoots when raising the weapon, but with a smoother return to lowered, Spring overshoots and wobbles back to position, SpringIn has a smoother return to lowered position, Bounce bounces off the target pose briefly, BounceIn has a smoother return to lowered, Custom uses virtual methods which you can override in a custom script.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

[Modular Firearms](#)

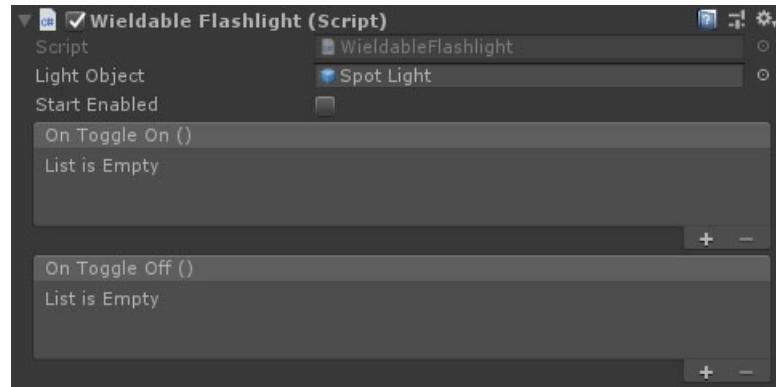
[First Person Camera](#)

WieldableFlashlight MonoBehaviour

Overview

The WieldableFlashlight behaviour is used to add a toggleable flashlight to firearms.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Light Object	GameObject	A child object with a light component attached.
Start Enabled	Boolean	Should the flashlight be on from the start.
On Toggle On	UnityEvent	An event fired when the laser is switched on.
On Toggle Off	UnityEvent	An event fired when the laser is switched off.

See Also

[Modular Firearms](#)

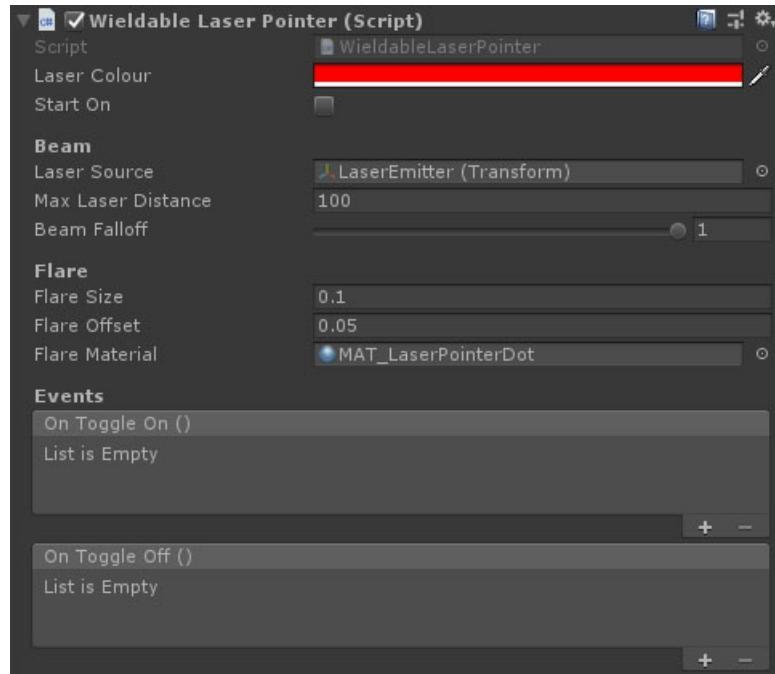
[Attachments](#)

WieldableLaserPointer MonoBehaviour

Overview

The WieldableLaserPointer behaviour is used to add a toggleable laser pointer to firearms with configurable colour and intensity.

Inspector



Properties

The WieldableLaserPointer properties are split into a number of sections:

NAME	TYPE	DESCRIPTION
Laser Colour	Color	The colour of the laser.
Start On	Boolean	Should the laser be on from start.

Beam

NAME	TYPE	DESCRIPTION
Laser Source	Transform	The transform the laser will be projected from (forwards).
Max Laser Distance	Float	The furthest distance the laser pointer will be visible.
Beam Falloff	Float	The alpha falloff of the beam over its length.

Flare

NAME	TYPE	DESCRIPTION
Flare Size	Float	The size of the laser hit point flare.
Flare Offset	Float	The distance the flare should be pushed forward towards the camera from the point of impact (prevents intersection with walls).

NAME	TYPE	DESCRIPTION
Flare Material	Material	The material to use for the laser impact flare. NeoFPS includes a shader for laser flares with variable intensity and colour.

Events

NAME	TYPE	DESCRIPTION
On Toggle On	UnityEvent	An event fired when the laser is switched on.
On Toggle Off	UnityEvent	An event fired when the laser is switched off.

See Also

[Modular Firearms](#)

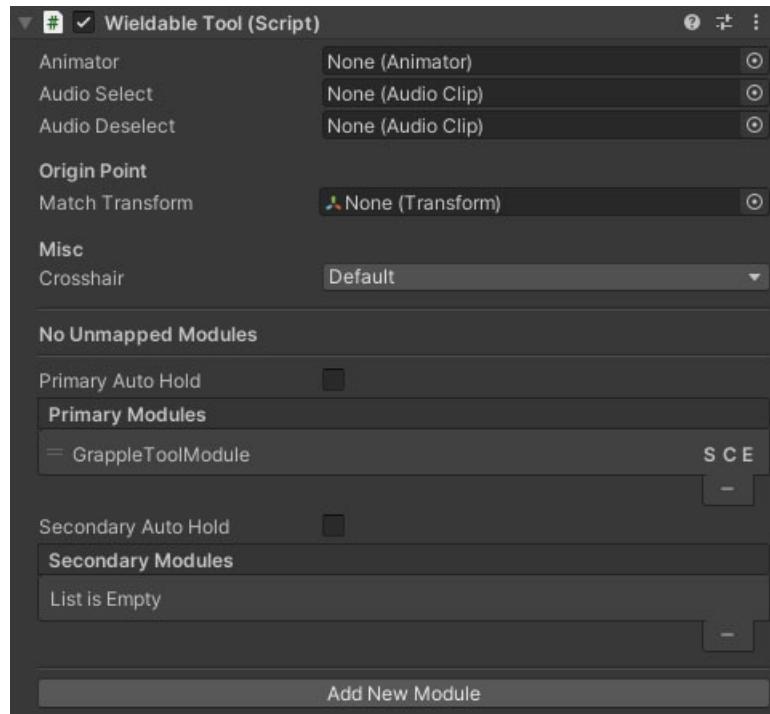
[Attachments](#)

WieldableTool MonoBehaviour

Overview

The WieldableTool behaviour is used to create tools and items that are equipped and used in a character's hands. They have both a primary and secondary fire, and are built out of modules and actions.

Inspector



Properties

Name	Type	Description
Animator Location	Dropdown	What animators should be exposed to the weapon components. Options are None , AttachedOnly , AttachedAndCharacter , MultipleAttached , MultipleAttachedAndCharacter and CharacterOnly .
Animator	Animator	The animator component of the tool.
Pose Transform	Transform	The transform that should be offset for weapon poses.
Anim Key Draw	String	The key for the AnimatorController trigger property that triggers the draw animation.
Draw Duration	Float	The time it takes to raise the tool. This option will only be visible when the anim key is set.
Anim Key Lower	String	The AnimatorController trigger key for the tool lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection. This option will only be visible when the anim key is set.
Audio Select	AudioClip	The audio clip when raising the tool.

Name	Type	Description
Audio Deselect	AudioClip	The audio clip when lowering the tool.
Crosshair	[FpsCrosshair] [3]	The crosshair to show when the weapon is drawn.
Primary Auto Hold	Boolean	Does tapping the primary fire button have the effect of holding it until interrupted (useful for timed sequences).
Secondary Auto Hold	Boolean	Does tapping the secondary fire button have the effect of holding it until interrupted (useful for timed sequences).

Unmapped Modules

The unmapped modules are the tool modules and actions that exist on this object but have not yet been assigned to either the primary fire. The buttons will assign the module to the relevant fire mode, or remove it from the object.

Primary and Secondary Modules

The primary and secondary modules array assign each of the attached modules to either the primary or secondary fire modes, including specifying the order they trigger. The **S**, **C** and **E** characters specify whether the module's actions are triggered on the fire mode's start (S), end (E) or continuously while the trigger is held (C). A number of modules have options you can set for which they should use.

Add New Module

The "Add New Module" button is a fast way to add a module component to the tool. Clicking it will list all the available module types for you to choose from.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

[Wieldable Tools](#)

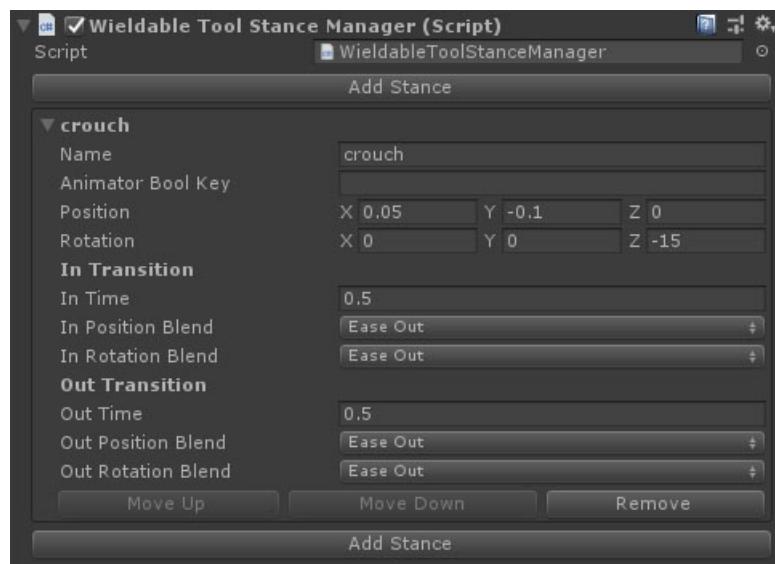
WieldableToolStanceManager MonoBehaviour

Overview

The WieldableToolStanceManager behaviour is used to specify poses or stances for a [wieldable tool](#). The entire tool will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be temporarily exited when aiming and reloading.

Inspector



Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator .
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp , EaseIn , EaseOut , EaseInOut , SwingAcross , SwingUp , Spring , Bounce and Overshoot .
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp , Slerp , EaseIn , EaseOut , EaseInOut , Spring , Bounce , Overshoot .
Out Time	Float	The time taken to enter the stance.

See Also

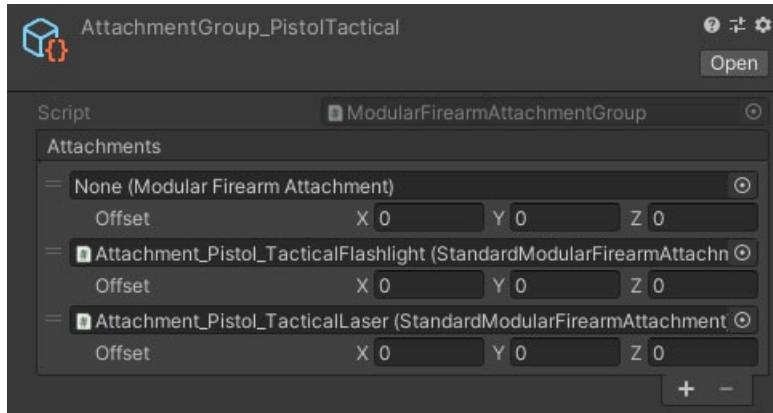
[Wieldable Tools](#)

ModularFirearmAttachmentGroup ScriptableObject

Overview

The ModularFirearmAttachmentGroup scriptable object is used to store a list of [firearm attachments](#) (optics, magazines, etc) that can be attached to a [socket](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Attachments	Attachment Array	The attachment prefabs in this group along with position offsets from their socket (local space).

See Also

[Modular Firearms](#)

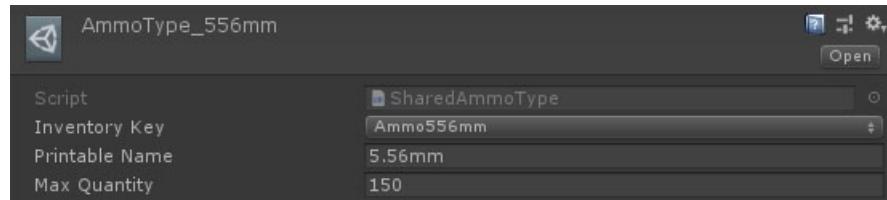
[Modular Firearm Attachments](#)

SharedAmmoType ScriptableObject

Overview

The SharedAmmoType scriptable object is an ammo type that can be stored in the inventory and used by multiple firearms.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Printable Name	String	The name to be printed on the HUD.
Max Quantity	Int	The maximum quantity a character can carry.

See Also

[Modular Firearms](#)

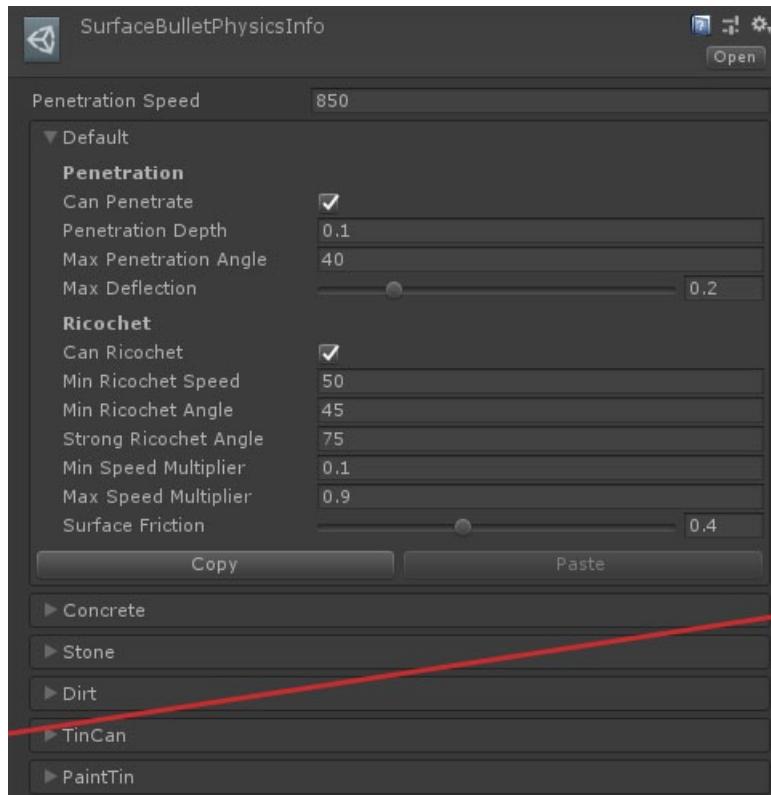
[FpsInventoryAmmo](#)

SurfaceBulletPhysicsInfo ScriptableObject

Overview

The SurfaceBulletPhysicsInfo scriptable object specifies how bullets react to different surface types using a [SurfaceBulletPhysicsAmmoEffect](#). Individual surfaces can have different settings for ricochet and penetration.

Inspector



Properties

Name	Type	Description
Penetration Speed	Float	The speed at which the penetration values below are accurate. As speed drops away, projectiles cannot penetrate as far.

Per-Surface Properties

Name	Type	Description
Can Penetrate	Boolean	Can the projectile penetrate this surface type.
Penetration Depth	Float	The maximum depth the projectile can penetrate this surface when travelling at the Penetration Speed specified above.
Max Penetration Angle	Float	The maximum angle of incidence with the surface where a projectile can penetrate.
Max Deflection	Float	A 0-1 value representing how much the projectile direction can be altered by moving through the surface.
Can Ricochet	Boolean	Can the projectile ricochet off this surface.

NAME	TYPE	DESCRIPTION
Min Ricochet Speed	Float	The speed below which a projectile cannot ricochet.
Min Ricochet Angle	Float	The minimum angle of incidence that a projectile will ricochet. Below this and the projectile will penetrate or be destroyed.
Strong Ricochet Angle	Float	The angle of incidence where the projectile is deflected with the least speed loss.
Min Speed Multiplier	Float	The multiplier applied to the projectile speed when it hits at the minimum angle (as close to straight on as it can ricochet).
Max Speed Multiplier	Float	The multiplier applied to the projectile speed when it hits at the maximum glancing angle.
Surface Friction	Float	Surface friction is used to add randomness to the ricocheted projectile's direction. Lower friction means a more predictable ricochet.

See Also

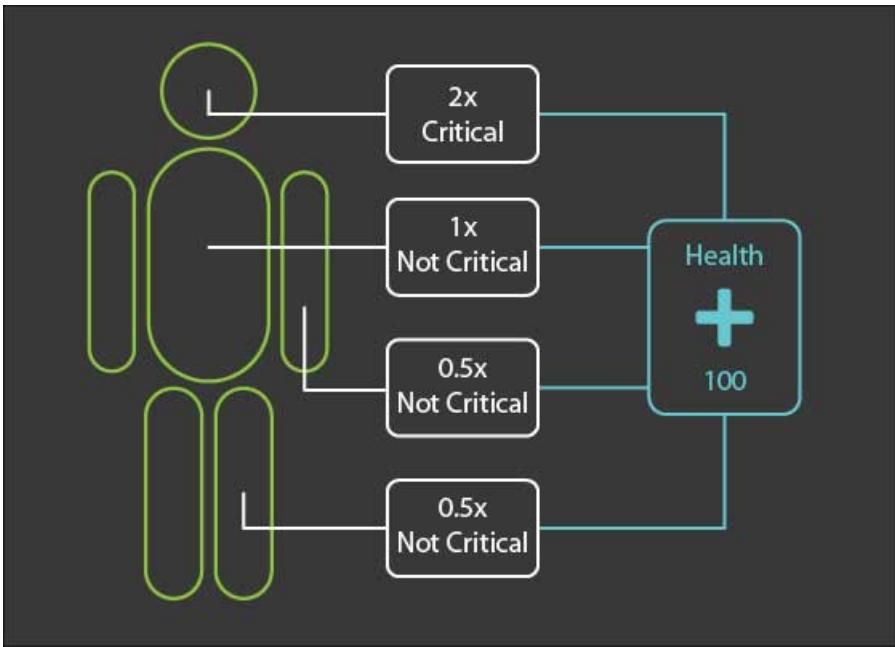
[Modular Firearms](#)

[EpsInventoryAmmo](#)

Health and Damage

Overview

Character health and damage are handled by 2 elements in NeoFPS: A [health manager](#) and a number of damage handlers. Damage handlers react to damage from weapons and other scripts, modify that damage, and then pass it on to the health manager. For more information, see the [BasicDamageHandler](#) and [EventDamageHandler](#).



Incoming damage can have a source attached, which can be used to work out the position or direction damage came from as well as the type of damage.

Health Pickups

Characters can regain health by collecting health pickups. These have a number of options for how they heal, and can be applied to either interactive objects, or trigger based pickups. The simplest way to create a health pickup is using the [Pickup Wizard](#) in the NeoFPS Hub.

Shields & Armour

Alongside health, NeoFPS has systems for simple energy shields and inventory based armour. When the character or object takes damage, first shields will absorb that damage until they are depleted, then armour will mitigate some of the damage (and be destroyed in the process).

Shields require the character to have a [ShieldSystem](#) behaviour attached, and the damage handlers are replaced with [ShieldedDamageHandlers](#). You can then specify a number of shield steps, each with a set charge. Damage is taken from each step in order, and if any step is not fully broken it will recharge after a set period. To restore broken steps you can use a [ShieldPickup](#).

Armour is implemented by using [ArmouredDamageHandler](#) components. These allow you to specify the inventory key for an armour item, and then any damage will be (partially) absorbed and taken away from that inventory item instead.

If you want to use both shields and armour, you can use the [ShieldedArmouredDamageHandler](#).

The simplest way to create shield booster pickups or armour (inventory item) pickups is using the [Pickup Wizard](#) in the NeoFPS Hub.

Healing and Damage Zones

You can define areas that heal or damage characters over time by using the [HealZone](#) and [DamageZone](#) behaviours.

Damage Filters

Damage in NeoFPS can be filtered by team and by type. This allows for gameplay mechanics such as ignoring friendly fire, or blocking damage from explosions but not bullets.

The damage filter can be used for up to 8 teams and 8 damage types.

Damage handlers have an incoming damage filter, while damage sources have an outgoing damage filter. In code a collision can be checked via the following code:

```
if (inDamageFilter.CollidesWith (sourceDamageFilter, friendlyFire))  
{  
    // React to damage here  
}
```

The friendly fire parameter is a `bool`. If this is true then the team section of either filter will be ignored. If not then the team has to be valid as well as the damage type.

Damage Source

The damage source allows for feedback on the location of the source and, when combined with damage filters, effects such as returning damage to melee attackers.

The damage source specifies outgoing damage filter and the [character controller](#) if relevant.

See Also

Health Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

My Character Isn't Taking Damage

There are a number of possible causes here. Please check the following:

- The character has a health manager component on its root, and one or more damage handlers on child objects. A damage handler is a component that implements the `IDamageHandler` interface such as [BasicDamageHandler](#), [EventDamageHandler](#), [ArmouredDamageHandler](#) or [ShieldedArmouredDamageHandler](#).
- Check that any objects that have damage handler components **also have a collider component** and are on a layer that weapons should hit such as **CharacterPhysics**.
- Check your weapons shooter modules or projectiles are set to affect the layers that your damage colliders are on. Each of these will have a property names something like "Collision Layers".

I Want To Disable Fall Damage

The **FpsSoloCharacter** component on the root of your character has a number of settings for impact based damage. The **Apply Fall Damage** property dictates if impacts will be translated to damage based on force, but there are also settings for body impacts and head impacts. These do not include gun shots - only impacts due to character movement.

Armour Isn't Showing Up

The armour system has a number of components that need setting up that work together. The [ArmouredDamageHandler](#) or [ShieldedArmouredDamageHandler](#) apply armour modifiers to any incoming damage for a character. The character needs to have an inventory item for the armour itself, while the [HudInventoryItemCounter](#) and [HudInventoryItemMeter](#) components are used on HUD elements to show the inventory item count (there are prefabs in the sample folders that you can use as a starting point).

All of the above components use an inventory key to define what inventory object is used as the armour. If the key does not match between the damage handlers and the inventory item then the character will take full damage and act as though they have no armour. If the key does not match on the HUD components, then the character will take reduced damage but you will get no feedback on screen to show the current armour amount.

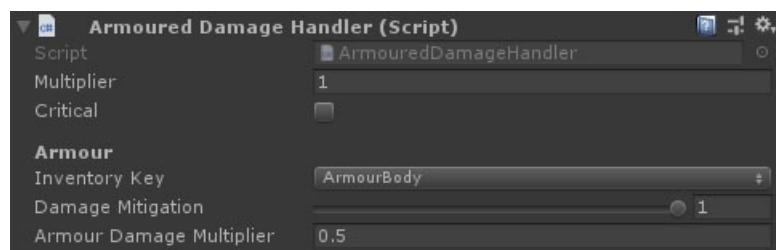
If you are sure that the inventory keys are the same, but the HUD elements aren't showing and the character is taking full damage, then that suggests that the armour item isn't being added to the character inventory. You can open the character hierarchy in play mode and check in the hierarchy under the **ItemRoot** object to check what inventory items they have on them (this is the default name - you can find the object via the **Wieldable Root** property at the top of the inventory component on the character's root). If the object is not there then check the [Inventory Troubleshooting](#) page for assistance in tracking down why.

ArmouredDamageHandler MonoBehaviour

Overview

The ArmouredDamageHandler behaviour adds an armour system to the [BasicDamageHandler](#). Armour will mitigate some or all of the damage received by the damage handler, but it is consumed in the process. Armour is an [inventory item](#) that is picked up in the scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
Inventory Key	FpsInventoryKey	The inventory key of the armour type.
Damage Mitigation	Float	The amount of damage the armour should nullify.
Armour Damage Multiplier	Float	A multiplier used to modify how much armour is destroyed by the incoming damage.

See Also

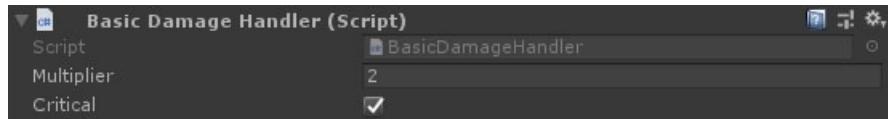
[Health and Damage](#)

BasicDamageHandler MonoBehaviour

Overview

The BasicDamageHandler behaviour takes damage and passes it to a parent [HealthManager](#)

Inspector



Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.

See Also

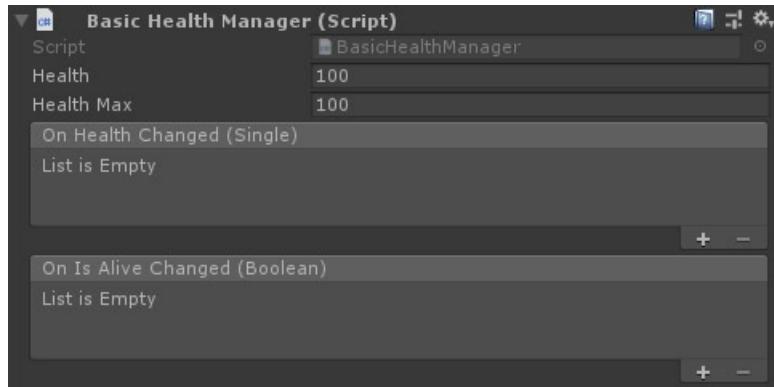
[Health and Damage](#)

BasicHealthManager MonoBehaviour

Overview

The BasicHealthManager behaviour manages a character's health, firing events when health changes and on death.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Health	Float	The starting health of the character.
Health Max	Float	The maximum health of the character.
Can Damage Self	Boolean	Can the character damage itself (eg with explosives).
On Health Changed	Unity Event	An event called whenever the health changes.
On Is Alive Changed	Unity Event	An event called whenever the alive state of the health manager changes.

See Also

[Health and Damage](#)

DamageZone MonoBehaviour

Overview

The DamageZone behaviour applies damage to a character over time whilst they are inside it. It is used alongside a [CharacterTriggerZone](#), so it should be placed on a GameObject on the **TriggerZones** layer with a collider with **IsTrigger** set to true.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Per Second	Float	The amount of damage to apply to the player character per second.
Damage Type	DamageType	The type of damage to apply.
Damage Description	String	A description of the damage to use in logs, etc.

See Also

[Health and Damage](#)

DeathAnimation MonoBehaviour

Overview

The DeathAnimation behaviour sets a bool parameter on the character's [Animator](#) component when the character dies.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Dead Parameter	String	The bool parameter on the animator to set when the character dies.

See Also

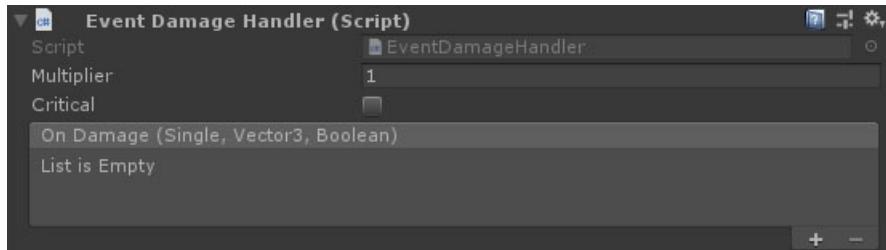
[Health and Damage](#)

EventDamageHandler MonoBehaviour

Overview

The EventDamageHandler behaviour takes damage and passes it to a parent [HealthManager](#) as well as invoking a damage event.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
On Damage	UnityEvent	An event that is invoked when damage is taken. The parameters are: <code>float damage, Vector3 direction, bool critical</code>

See Also

[Health and Damage](#)

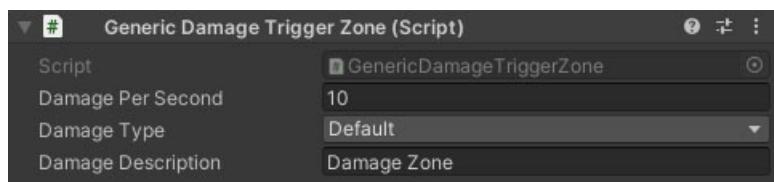
[Unity Events](#)

GenericDamageTriggerZone MonoBehaviour

Overview

The GenericDamageTriggerZone behaviour applies damage to any collider that overlaps with it that contains an behaviour that implements `IDamageHandler`. It is used alongside a trigger zone based collider such as a [Sphere Collider](#). In order to correctly detect damage handler colliders it should be placed on a GameObject on the **TriggerZones** layer, along with a collider with **IsTrigger** set to true.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Per Second	Float	The amount of damage to apply to the player character per second.
Damage Type	DamageType	The type of damage to apply.
Damage Description	String	A description of the damage to use in logs, etc.

See Also

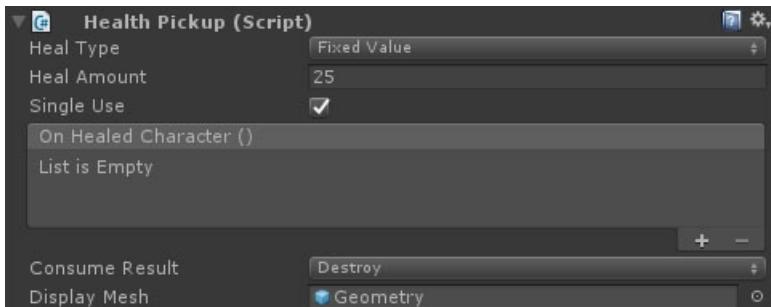
[Health and Damage](#)

HealthPickup MonoBehaviour

Overview

The HealthPickup behaviour restores health to the character that picks it up. It can be set to provide a set amount of healing, or some factor of the character's total or missing health.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Heal Type	Dropdown	How the heal is applied. FixedValue adds the health. Factor adds amount * max health. MissingFactor adds amount * missing health.
Heal Amount	Float	The amount to heal by. If heal type is Factor or MissingFactor , this will be a value between 0 and 1.
On Healed Character	UnityEvent	An event called when the pickup heals a character.
Single Use	Float	If the character needs less healing than the pickup amount, should the pickup still be destroyed or should the remainder be available to use again?
Consume Result	Dropdown	What to do to the pickup object once its item has been used (fully, or single use). Available options are Destroy , Disable and Respawn .
Respawn Duration*	Float	How long to wait before respawning if the consume result is set to Respawn .
Display Mesh	GameObject	The object containing the display mesh of the pickup. This should be a child of the object this behaviour is applied to, so that if this is disabled the pickup will still respawn if required.

* This property is only visible if the consume result is set to **Respawn**.

See Also

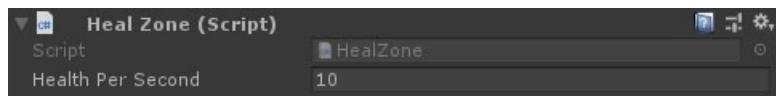
[Health and Damage](#)

HealZone MonoBehaviour

Overview

The HealZone behaviour restores health to a character over time whilst they are inside it. It is used alongside a [CharacterTriggerZone](#), so it should be placed on a GameObject on the **TriggerZones** layer with a collider with **IsTrigger** set to true.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Health Per Second	Float	The amount of health to apply to the player character per second.

See Also

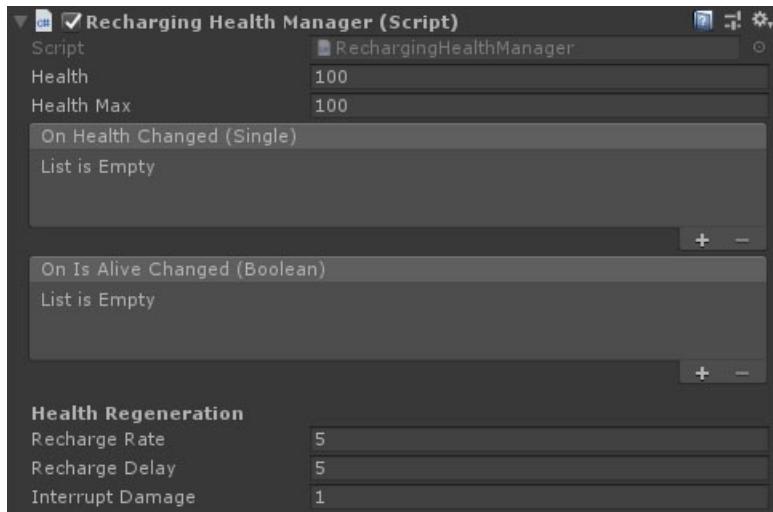
[Health and Damage](#)

RechargingHealthManager MonoBehaviour

Overview

The RechargingHealthManager behaviour will recharge to full health, with a set delay after being damaged. New damage interrupts the healing and resets the timer.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Health	Float	The starting health of the character.
Health Max	Float	The maximum health of the character.
On Health Changed	Unity Event	An event called whenever the health changes.
On Is Alive Changed	Unity Event	An event called whenever the alive state of the health manager changes.
Recharge Rate	Float	The recharge speed for health regeneration.
Recharge Delay	Float	The delay between taking damage and starting health regen.
Interrupt Damage	Float	Health recharge will be interrupted if damage greater than this amount is received.

See Also

[Health and Damage](#)

ShieldedArmouredDamageHandler MonoBehaviour

Overview

The ShieldedArmouredDamageHandler behaviour combines the [ShieldedDamageHandler](#) and [ArmouredDamageHandler](#) systems. Damage is first absorbed by the character's shields. Any damage that is not shielded (because the shields have been depleted) is then mitigated by the armour, before being passed to the character's health manager.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
Inventory Key	[FpsInventoryKey] [4]	The inventory key of the armour type.
Damage Mitigation	Float	The amount of damage the armour should nullify.
Armour Damage Multiplier	Float	A multiplier used to modify how much armour is destroyed by the incoming damage.

See Also

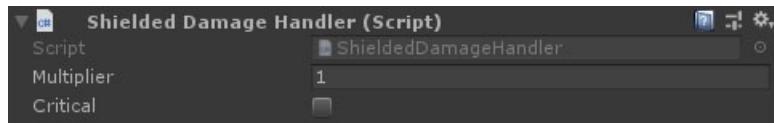
[Health and Damage](#)

ShieldedDamageHandler MonoBehaviour

Overview

The ShieldedDamageHandler behaviour adds an energy shield system to the [BasicDamageHandler](#). Energy shields completely absorb damage until they are broken. The shield capacity and charge is handled by a [ShieldManager](#) behaviour on the character.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.

See Also

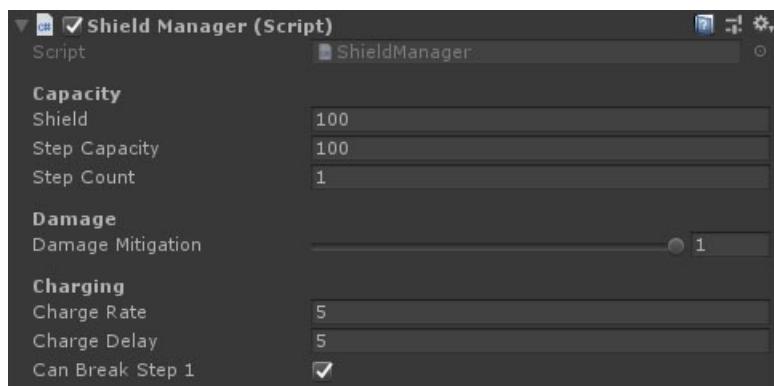
[Health and Damage](#)

ShieldManager MonoBehaviour

Overview

The ShieldManager behaviour is used to track a character's energy shields. Shields completely absorb damage via the [ShieldedDamageHandler](#) and [ShieldedArmouredDamageHandler](#), but are depleted in the process. A character's shields are split into steps. The steps automatically recharge if partially depleted after a short delay. If a shield step is completely depleted, then that step is broken and will not recharge. You will need to collect a [ShieldPickup](#) to restore broken steps.

Inspector



Properties

NAME	TYPe	DESCRIPTION
Shield	Float	The starting shield amount.
Step Capacity	Float	The shield capacity of each shield step / block.
Step Count	Integer	The number of shield steps / blocks.
Damage Mitigation	Float	The amount of damage (multiplier) that the shield negates.
Charge Rate	Float	The recharge speed for shield regeneration.
Charge Delay	Float	The delay between taking damage and starting shield regen.
Can Break Step 1	Boolean	Shield steps only recharge if the shield value is greater than their starting level. If this property is false, step 1 will always recharge, even if it hits zero.

See Also

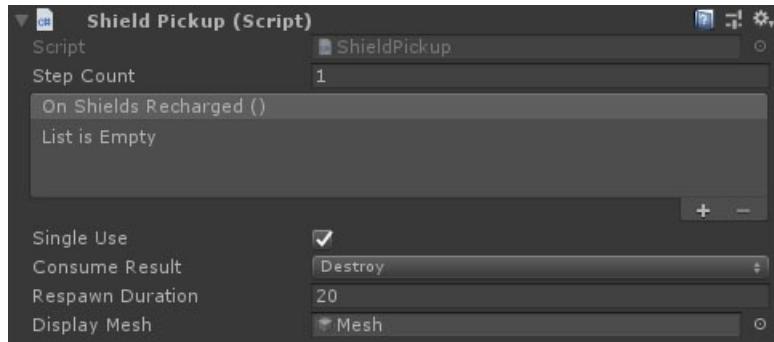
[Health and Damage](#)

ShieldPickup MonoBehaviour

Overview

The ShieldPickup behaviour is used to restore broken shield steps to a [ShieldManager](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Step Count	Integer	The number of shield steps to recharge.
On Shields Recharged	UnityEvent	An event called when the pickup heals a character.
Single Use	Boolean	If the character needs less recharging than the pickup amount, should the pickup still be destroyed or should the remainder be available to use again?
Consume Result	Dropdown	What to do to the pickup object once its item has been used (fully, or single use). Options are Destroy to destroy the whole object, Disable to disable the render mesh and trigger so it can be reset later, and Respawn to disable and the reset automatically after a brief wait.
Respawn Duration	Float	How long to wait before respawning if the consume result is set to Respawn .
Display Mesh	GameObject	The display mesh of the pickup. This should not be the same game object as this, so that if this is disabled the pickup will still respawn if required.

See Also

[Health and Damage](#)

The Player HUD

Overview

The player heads up display is made up of a number of UI elements that each present specific information about the player character or their inventory.

The Player Character Watcher

Most of the HUD elements subscribe to a player character watcher behaviour that is added to an object in their parent hierarchy. This is a behaviour that implements the `IPlayerCharacterWatcher` interface and tells the HUD elements when the player character changes so that they can attach to the relevant property or system attached to the character. If a behaviour that implements this interface is not found, then the HUD elements will log an error to the console on start. The NeoFPS samples use the `SoloPlayerCharacterEventWatcher` behaviour, though you are free to implement your own if required.

HUD Layout

The demo HUD makes use of Unity's UI `auto layout` components to properly order items. This means that when a player character is spawned that does not have a certain feature attached or enabled, then the relevant HUD item will be hidden, and the other HUD elements in its group will be positioned to fill the gap.

HUD Hider

The `HudHider` is used to hide the HUD elements when the player is not looking through a first person camera. This can be used for things like cutscenes.

Included HUD elements

The following HUD elements are included with NeoFPS for direct use or for reference when designing your own player HUD.

Health Counter



The health counter shows how much health your character currently has. For more information see [HudHealthCounter](#).

Shield Meter

![Shield Meter](./images/screenshot-hud-shield meter.jpg)

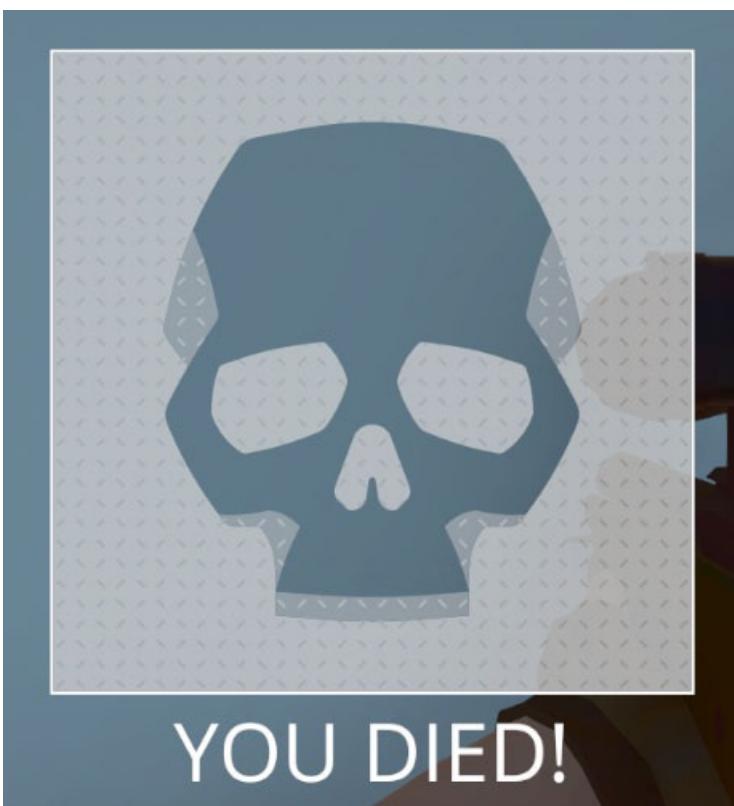
The shield meter shows the status of the player character's energy shields. These can be split into multiple shield steps which recharge unless broken. For more information see [HudShieldMeter](#).

Damage Markers



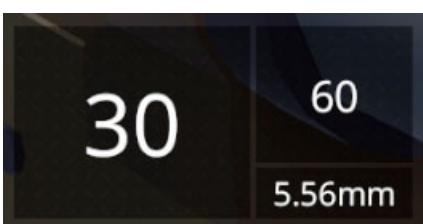
The damage markers highlight which direction damage comes from. The relevant edge of the screen is highlighted and then fades out over time. For more information see [HudDamageMarkers](#).

Death Popup



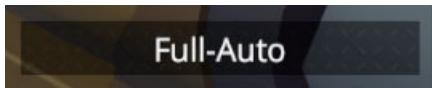
The death popup appears when the player character is killed and before they respawn. For more information see [HudDeathPopup](#).

Ammo Counter



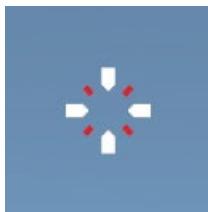
The ammo counter shows the current and available ammo along with the ammo type and weapon mode. For more information see [HudAmmoCounter](#).

Firearm Mode



The firearm mode readout will appear when the player selects a firearm that has multiple modes. It displays the current mode. For more information see [HudAmmoCounter](#).

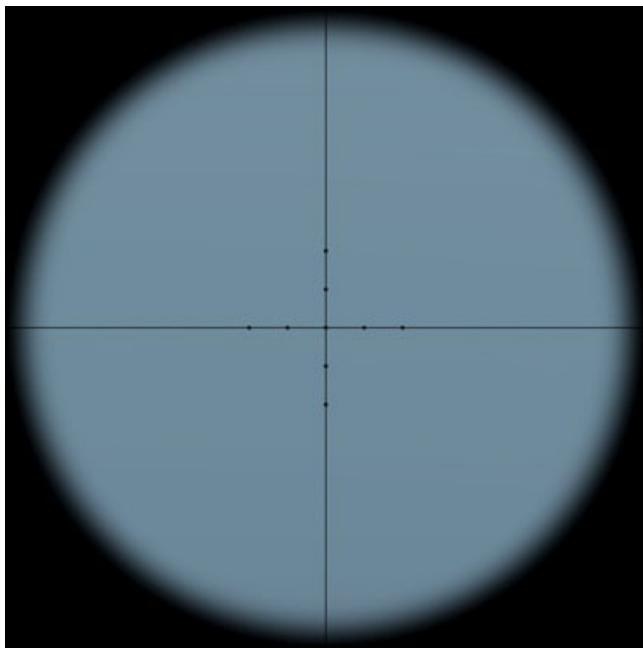
Crosshair



The crosshair is used for aiming and expands and contracts based on the current accuracy of the weapon. Multiple crosshairs can be used and referenced by ID. For more information see [HudCrosshair](#).

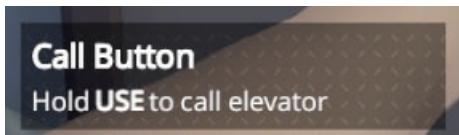
You can also use the advanced crosshair system which has a more extensible system for drawing individual crosshairs, and adds hit and critical hit markers. For more information see [HudAdvancedCrosshair](#).

Scopes



The scope is an image overlay that is used for certain weapons. Multiple scope images can be used and are referenced by ID. For more information see [HudScope](#).

Interaction Tooltip



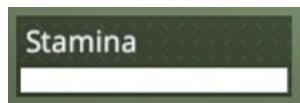
The interaction tooltip displays the name of the interactive object you are currently aiming at, along with the action performed when you interact with it. For more information see [HudInteractionTooltip](#).

Progress Bar



The progress bar is used to show the interaction progress for items that require you to hold down the interact control. For more information see [HudProgressBar](#).

Stamina Bar



The stamina bar shows how much health your character currently has. For more information see [HudStaminaBar](#).

Inventory - Standard



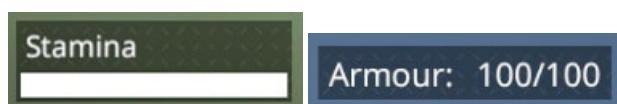
The standard inventory is the kind of inventory you would see in a classic PC FPS. Each weapon has a set slot in a sequence that matches the number keys. This HUD inventory is also used for the swappable inventory. For more information see [HudInventoryStandardPC](#) and [HudInventoryItemStandard](#).

Inventory - Stacked



The stacked inventory groups items together into stacks of a similar type. This is the inventory used in FPS games such as Half-Life. For more information see [HudInventoryStackedPC](#), [HudInventoryItemStacked](#) and [HudInventoryStackedSlot](#).

Meters and Counters



A number of different HUD meters and counters are provided to track various properties. These include:

BEHAVIOUR	DESCRIPTION
HudInventoryItemCounter	Shows the quantity of a specific item in the player character's inventory.
HudInventoryItemMeter	Shows the quantity vs max quantity of a specific item in the player character's inventory.

BEHAVIOUR	DESCRIPTION
HudMotionGraphParameterMeter	Shows the value in the relevant motion graph parameter on the character as a meter based on a set range.
HudMotionGraphParameterReadout	Shows the value in the relevant motion graph parameter on the character.
HudOxygenMeter	Shows the amount of oxygen remaining while swimming.
HudSlowMoCharge	Shows the charge of the slow-mo system. Sometimes called focus or adrenaline.
HudStaminaBar	Shows the player character's stamina / fatigue levels.

Target Lock



The target lock HUD shows icons over objects in the world that are locked onto. Locks can be partial (a second icon overlays and shows the lock progress) and full (only a location marker is shown). For more information see [HudTargetLockMarkers](#) and [HudTargetLock](#)

Touch controls



A number of touch controls are available for touchscreen input. For more information see [\[Touchscreen Input\]](#)[\[input-touchscreen.md\]](#).

See Also

[Health and Damage](#)

[Modular Firearms](#)

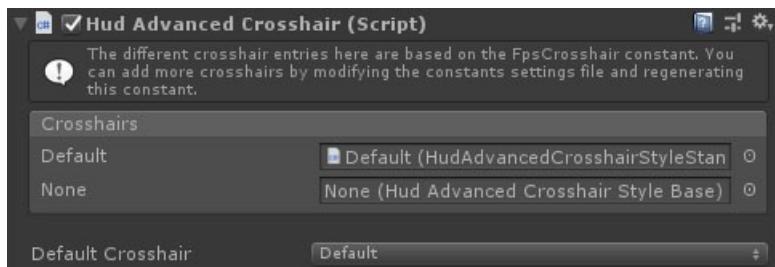
[Inventory](#)

HudAdvancedCrosshair MonoBehaviour

Overview

The HudAdvancedCrosshair behaviour displays a [Unity UI](#) based crosshair when an appropriate wieldable is selected, expanding based on accuracy. It can also show a hit marker briefly when the player deals damage to something, including highlighting the marker for critical hits.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Crosshairs	HudAdvancedCrosshairStyleStandard Array	The individual crosshairs.
Default Crosshair	FpsCrosshair	The default crosshair to use (when no valid crosshair driver is being wielded).

See Also

[Weapons](#)

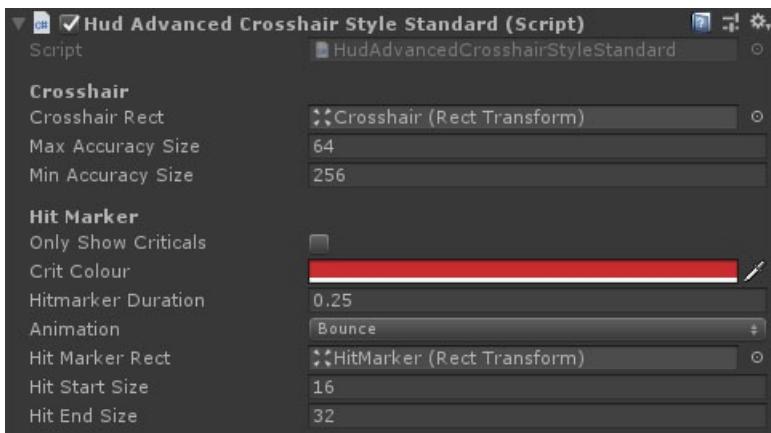
[Unity UI](#)

HudAdvancedCrosshairStyleStandard MonoBehaviour

Overview

The HudAdvancedCrosshairStyleStandard behaviour represents a specific crosshair style for the [HudAdvancedCrosshair](#) behaviour. It handles animating the hit marker and changing its size based on accuracy. You can also write your own crosshair styles for more complex effects if desired.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Crosshair Rect	RectTransform	The parent rect transform of the crosshairs (will be expanded and contracted based on accuracy).
Max Accuracy Size	Float	The size the UI element will reach at 100% accuracy.
Min Accuracy Size	Float	The size the UI element will reach at 0% accuracy.
m_OnlyShowCriticals	Boolean	Should the hit marker only show critical hits, or any hit that dealt damage.
m_CritColour	Color	The colour of the hit markers if for critical hits. Non-critical will use the crosshair colour.
m_HitmarkerDuration	Float	The amount of time the hit marker will be visible.
m_Animation	Dropdown	The animation easing function of the hit marker size. Available options are: Lerp , EaseIn , EaseOut , EaseInOut , Spring and Bounce .
m_HitMarkerRect	RectTransform	The parent rect transform of the hit marker.
m_HitStartSize	Float	The starting size of the hit marker.
m_HitEndSize	Float	The size of the hit marker just before it vanishes.

See Also

[HudAdvancedCrosshair](#)

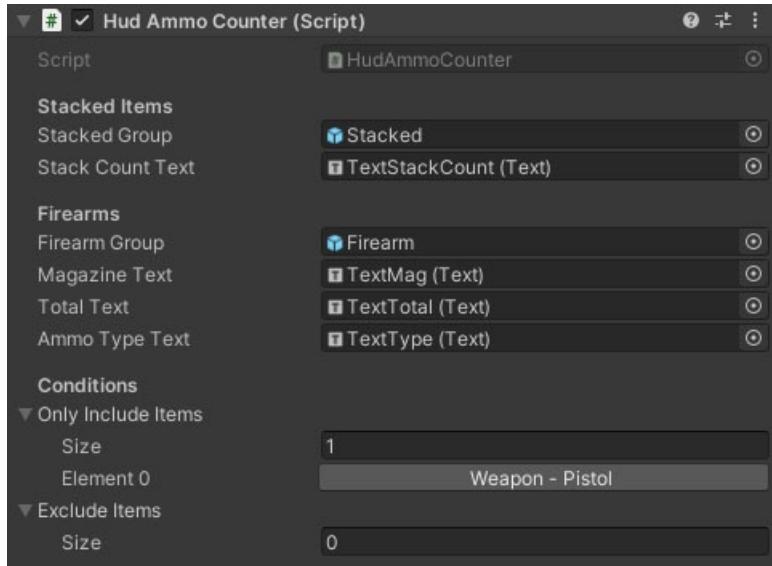
[Unity UI](#)

HudAmmoCounter MonoBehaviour

Overview

The HudAmmoCounter behaviour is a simple [Unity UI](#) based counter for firearm ammo or item stack counts. It also displays a description of the firearm's ammo type.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Stacked Group	GameObject	The group used for wieldable item stacks such as hand grenades.
Stack Count Text	Text	The stack count text entry for displaying count and total.
Firearm Group	GameObject	The group used for firearms.
Magazine Text	Text	The text entry for the current ammo in the firearm magazine.
Total Text	Text	The text entry for the total ammo the character is carrying.
Ammo Type Text	Text	The text entry for displaying the ammo type.
Only Include Items	Item ID Array	A set of IDs this ammo counter should be visible for. If none are specified, then the counter will be visible for all valid.
Exclude Items	Item ID Array	A set of IDs the ammo counter should be hidden for.

See Also

[Modular Firearms](#)

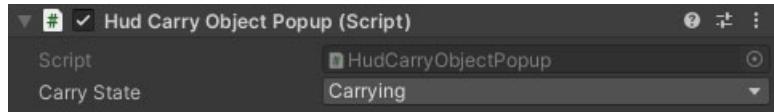
[Unity UI](#)

HudCarryObjectPopup MonoBehaviour

Overview

The HudCarryObjectPopup behaviour detects changes to the player character's [carry system](#) state and then displays or hides the object it's attached to. This can be used to provide prompts such as "Press **Pickup** to pick up the object".

Inspector



Properties

NAME	TYPE	DESCRIPTION
Carry State	Dropdown	The carry state to show this popup for.

See Also

[Carry System](#)

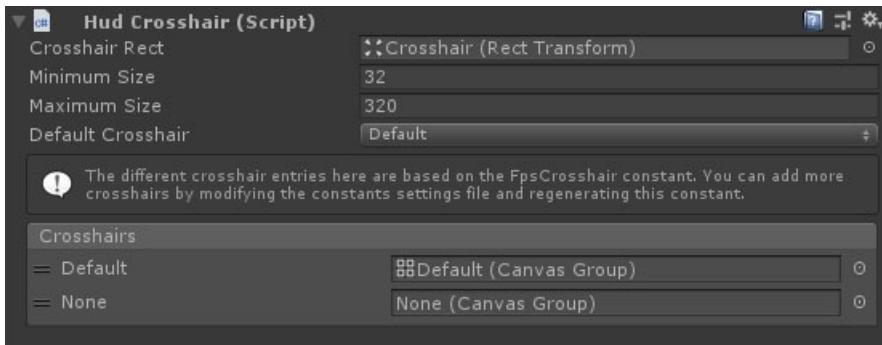
[Unity UI](#)

HudCrosshair MonoBehaviour

Overview

The HudCrosshair behaviour displays a [Unity UI](#) based crosshair when an appropriate wieldable is selected, expanding based on accuracy.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Crosshair Rect	RectTransform	The parent rect transform of the crosshairs (will be expanded and contracted based on accuracy).
Minimum Size	Float	The size the UI element will reach at 100% accuracy.
Maximum Size	Float	The size the UI element will reach at 0% accuracy.
Default Crosshair	FpsCrosshair	The crosshair to show by default.
Crosshairs	RectTransform Array	The individual crosshairs to match the FpsCrosshair generated constants values.

See Also

[Modular Firearms](#)

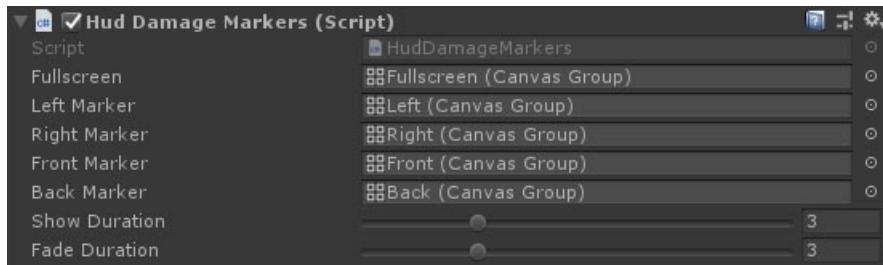
[Unity UI](#)

HudDamageMarkers MonoBehaviour

Overview

The HudDamageMarkers behaviour displays images at the edge of the screen based on the direction the player character takes damage from and then fades out over time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Fullscreen	CanvasGroup	The group for that encompasses all of the markers.
Left Marker	CanvasGroup	The group for fading out the left hand marker.
Right Marker	CanvasGroup	The group for fading out the right hand marker.
Front Marker	CanvasGroup	The group for fading out the front marker (top of the screen).
Back Marker	CanvasGroup	The group for fading out the back marker (bottom of the screen).
Show Duration	Float	How long should the markers remain fully visible.
Fade Duration	Float	How long do the markers take to fade out.

See Also

[Health and Damage](#)

[Unity UI](#)

HudDeathPopup MonoBehaviour

Overview

The HudDeathPopup behaviour is a simple Unity UI [Image](#) popup that displays when the player character dies.

Inspector



Properties

The HudDeathPopup behaviour has no properties exposed in the inspector.

See Also

[Health and Damage](#)

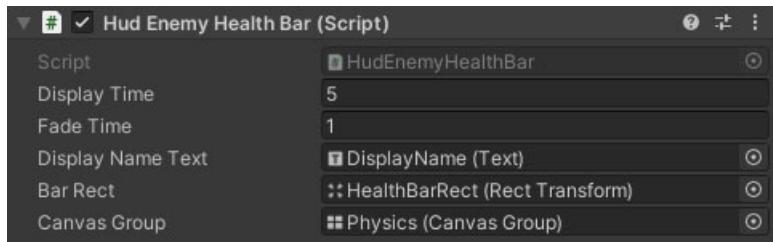
[Unity UI](#)

HudEnemyHealthBar MonoBehaviour

Overview

The HudEnemyHealthBar behaviour is a simple health bar using [Unity UI](#) that appears when you damage something with a [health manager](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Display Time	Float	The amount of time to display the health bar for when an enemy is hit.
Fade Time	Float	The time it takes to fade from full visibility to invisible after the display time has elapsed.
Display Name Text	Text	The text component that should display the hit target's name.
Bar Rect	RectTransform	The rect transform of the filled bar.
Canvas Group	CanvasGroup	The canvas group to fade out after enough time has elapsed.

See Also

[Health and Damage](#)

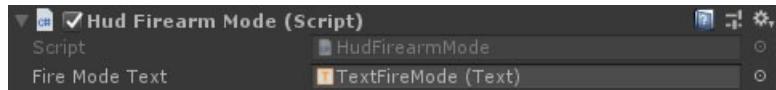
[Unity UI](#)

HudFirearmMode MonoBehaviour

Overview

The HudFirearmMode behaviour is used to display a description of the equipped firearm's current mode if appropriate.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Fire Mode Text	Text	The text entry for displaying the weapon fire mode.

See Also

[Modular Firearms](#)

[Unity UI](#)

HudFirearmOverheatBar MonoBehaviour

Overview

The HudFirearmOverheatBar behaviour displays a bar showing the current heat of the selected weapon. If the weapon overheats, then it will show a warning. If the selected weapon does not have a [FirearmOverheat](#) behaviour attached then this HUD element will be disabled.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The heat bar rect transform. This will be scaled along the x-axis based on the heat.
Cooldown Marker Rect	RectTransform	The rect transform of the cooldown marker. The cooldown marker shows the cooling threshold once a weapon has overheated. It will be hidden if the weapon cannot overheat, or if the threshold is 0.
Overheated Warning	GameObject	An object that is displayed while the weapon is overheating.

See Also

[FirearmOverheat Monobehaviour](#)

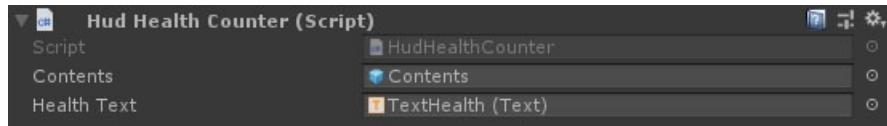
[Unity UI](#)

HudHealthCounter MonoBehaviour

Overview

The HudHealthCounter behaviour is a basic numeric [Unity UI](#) based health counter.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Contents	GameObject	The health counter game object (hidden when the player is not controlling a character).
Health Text	Text	The text readout for the current character health.

See Also

[Health and Damage](#)

[Unity UI](#)

HudHider MonoBehaviour

Overview

The HudHider behaviour is used to hide the HUD canvas when there is no first person camera active.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Hide Root	GameObject	The root object to disable when hiding the HUD.

See Also

[Health and Damage](#)

[Unity UI](#)

HudHitDamageCounters MonoBehaviour

Overview

The HudHitDamageCounters behaviour displays a damage counter whenever you damage something with a damage handler. The damage counter will be positioned on the UI so that it tracks the damage location in world space. An example damage counter is the [HudHitDamageCounterStandard](#)

Inspector



Properties

NAME	TYPE	DESCRIPTION
Max Markers	Integer	The number of markers that will be pooled on start.
Expand Amount	Integer	How many markers to add if the pool limit is reached.

See Also

[The Player HUD](#)

[Unity UI](#)

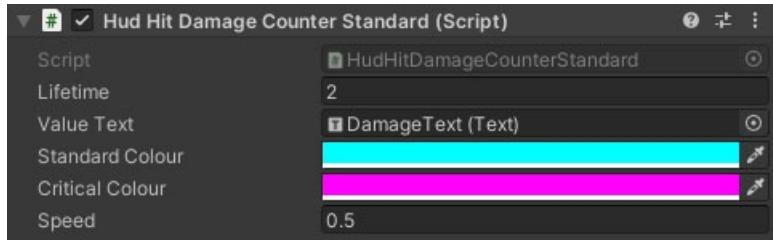
[HudHitDamageCounterStandard](#)

HudHitDamageCounterStandard MonoBehaviour

Overview

The HudHitDamageCounterStandard behaviour is a damage counter that displays the damage value using a [Unity UI Text](#) element, setting the colour depending on the damage type.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Lifetime	Float	The amount of time the counter should remain visible.
Value Text	Text	The text component used to display the damage amount.
Standard Colour	Color	The text colour used to represent standard damage.
Critical Colour	Color	The text colour used to represent critical hits.
Speed	Float	The speed (m/s) the counter appears to move upwards after spawning.

See Also

[The Player HUD](#)

[Unity UI](#)

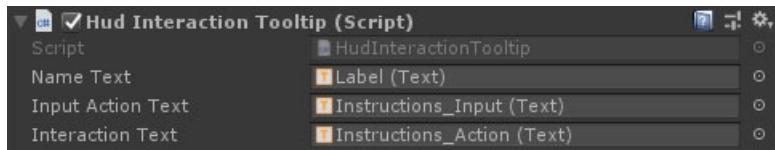
[HudWorldSpaceTarget](#)

HudInteractionTooltip MonoBehaviour

Overview

The HudInteractionTooltip behaviour is used to display the name of the [interactive object](#) that the player character is aiming at, along with a simple description of what interacting with it does.

Inspector



Properties

NAME	TITLE	DESCRIPTION
Name Text	Text	The UI text element to show the highlighted object's name.
Input Action Text	Text	The UI text element to show the input action required (press or hold).
Interaction Text	Text	The UI text element to show the interaction result (eg pick up).
Hold Option	String	The verb to use for holding the use button.
Press Option	String	The verb to use for pressing/tapping the use button.
Default Action	String	The verb to use for the action when that's performed when using the item.

See Also

[Interactive Objects](#)

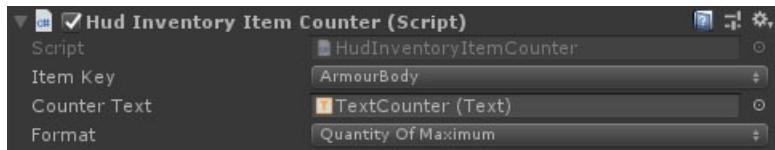
[Unity UI](#)

HudInventoryItemCounter MonoBehaviour

Overview

The HudInventoryItemCounter behaviour displays a simple text counter showing the quantity of a specific item in the player character's [inventory](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Counter Text	Text	The UI text element to output the quantity to.
Format	Dropdown	How to format the quantity. Available options are: Quantity prints the number of items, but hides itself when at 0; QuantityAlwaysVisible prints the number of items, including at 0; QuantityOfMaximum prints the stored number and maximum, eg. 1/10; Percent displays the quantity as a percentage of the maximum.

See Also

[Inventory](#)

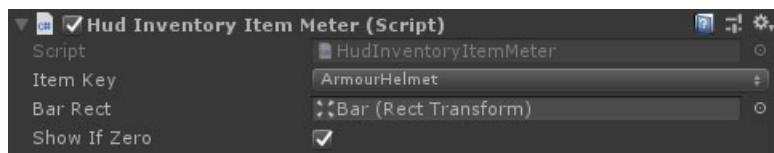
[Unity UI](#)

HudInventoryItemMeter MonoBehaviour

Overview

The HudInventoryItemMeter behaviour displays a simple meter bar that represents the quantity vs max quantity of a specific item in the player character's [inventory](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

[Inventory](#)

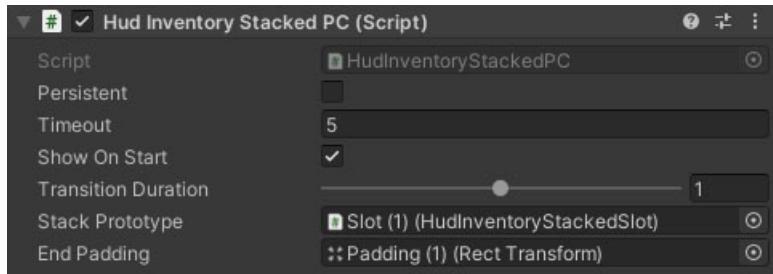
[Unity UI](#)

HudInventoryStackedPC MonoBehaviour

Overview

The HudInventoryStackedPC behaviour represents the stacked inventory on the HUD. Layed out for keyboard users.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Persistent	Boolean	Does the HUD inventory stay visible at all times, or fade out?
Timeout	Float	The duration the HUD inventory stays fully visible before fading out.
Show On Start	Boolean	If the HUD inventory is not persistent, should it appear when the player first assumes control of the character?
Transition Duration	Float	The duration the fade out lasts.
Stack Prototype	HudInventoryStackedSlot	A prototype of a single quick-slot stack for duplicating.
End Padding	Transform	The padding transform to pad the layout group and push the item slots together.

See Also

[HudInventoryStackedSlot](#)

[Inventory Examples](#)

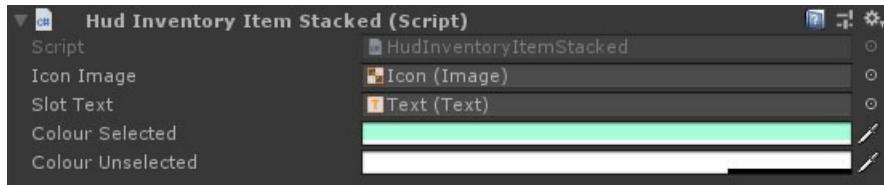
[Unity UI](#)

HudInventoryItemStacked MonoBehaviour

Overview

The HudInventoryStackedSlot behaviour represents an individual quick-slot item in the [HudInventoryStackedPC](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Icon Image	Image	The UI image that is used to display the correct UI icon for the item.
Slot Text	Text	The UI text to display the slot number.
Colour Selected	Color	A colour for the item in the HUD when selected.
Colour Unselected	Color	A colour for the item in the HUD when not selected.

See Also

[HudInventoryStackedPC](#)

[Inventory Examples](#)

[Unity UI](#)

HudInventoryStackedSlot MonoBehaviour

Overview

The HudInventoryStackedSlot behaviour represents a quick-slot stack in the [HudInventoryStackedPC](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Item Prototype	HudInventoryItemStacked	The item slot prototype to duplicate for the stack.
Padding Transform	RectTransform	The padding transform to pad the layout group and push the item slots together.

See Also

[HudInventoryItemStacked](#)

[HudInventoryStackedPC](#)

[Inventory Examples](#)

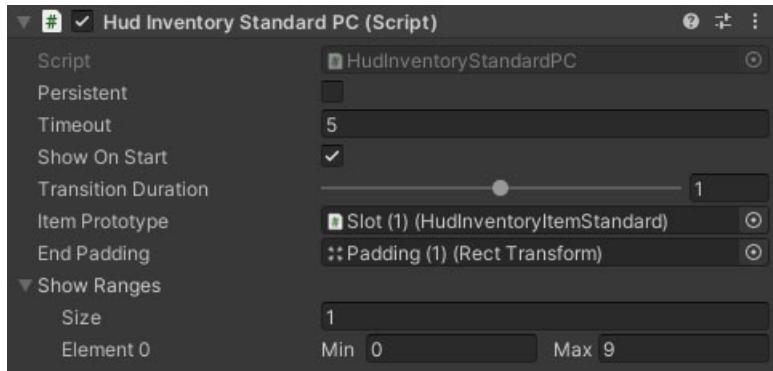
[Unity UI](#)

HudInventoryStandardPC MonoBehaviour

Overview

The HudInventoryStandardPC behaviour represents the standard inventory on the HUD. Layed out for keyboard users.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Persistent	Boolean	Does the HUD inventory stay visible at all times, or fade out?
Timeout	Float	The duration the HUD inventory stays fully visible before fading out.
Show On Start	Boolean	If the HUD inventory is not persistent, should it appear when the player first assumes control of the character?
Transition Duration	Float	The duration the fade out lasts.
Item Prototype	HudInventoryItemStandard	A prototype of a single quick-slot item for duplicating.
End Padding	Transform	The padding transform to pad the layout group and push the item slots together.
Show Ranges	Int Range Array	Ranges of quick slot numbers that will be visible in this HUD inventory.

See Also

[HudInventoryItemStandard](#)

[Inventory Examples](#)

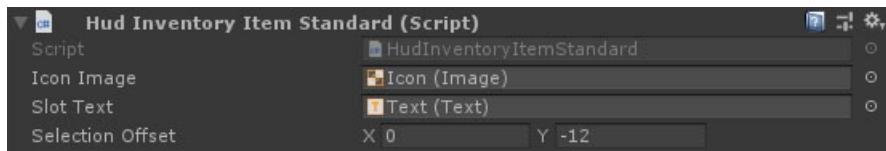
[Unity UI](#)

HudInventoryItemStandard MonoBehaviour

Overview

The HudInventoryItemStandard represents a quick-slot item in the [HudInventoryStandardPC](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Icon Image	Image	The UI image that is used to display the correct UI icon for the item.
Slot Text	Text	The UI text to display the slot number.
Selection Offset	Vector2	An offset to highlight quick slot items when selected.

See Also

[Inventory Examples](#)

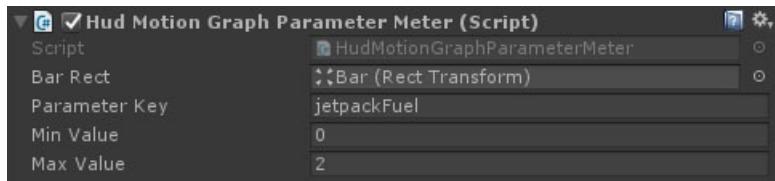
[Unity UI](#)

HudCrosshair MonoBehaviour

Overview

The HudMotionGraphParameterMeter behaviour displays is used to track a [parameter](#) on the player character's motion graph and display it as a meter bar on the [Unity UI](#). An example usage is for jetpack fuel on the jetpack character.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	[RectTransform][unity-recttransform]	The rect transform of the meter's bar. This will be scaled based on the value.
Parameter Key	String	The name of the motion graph float parameter to watch.
Min Value	Float	The minimum value at which the meter bar will have zero width.
Max Value	Float	The maximum value at which the meter bar will have reached its full width.

See Also

[Unity UI](#)

[Motion Graph Parameters And Data](#)

HudCrosshair MonoBehaviour

Overview

The HudMotionGraphParameterReadout behaviour displays is used to track a [parameter](#) on the player character's motion graph and display it as a text readout on the [Unity UI](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	[RectTransform][unity-recttransform]	The rect transform of the meter's bar. This will be scaled based on the value.
Parameter Key	String	The name of the motion graph float parameter to watch.

See Also

[Unity UI](#)

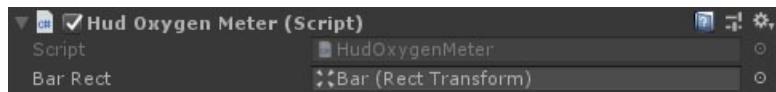
[Motion Graph Parameters And Data](#)

HudOxygenMeter MonoBehaviour

Overview

The HudOxygenMeter behaviour displays a simple meter bar for the player character's oxygen level when swimming.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

[SlowMoSystem](#)

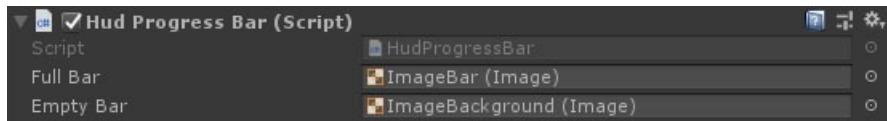
[Unity UI](#)

HudProgressBar MonoBehaviour

Overview

The HudProgressBar behaviour is used to display a progress bar when the character interacts with an [interactive object](#) in the world.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Full Bar	Image	The image for the completed progress (overlaps the empty bar. Should be scaled so 100 wide fills the empty bar.)
Empty Bar	Image	The image for the empty bar.

See Also

[Interactive Objects](#)

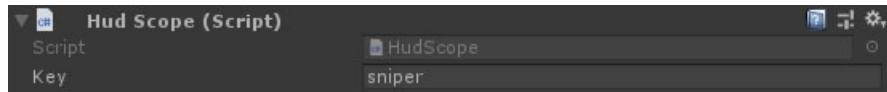
[Unity UI](#)

HudScope MonoBehaviour

Overview

The HudScope behaviour displays a UI based scope.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Key	String	The scope name. Used to allow different weapons to use different scopes.

See Also

[Modular Firearms](#)

[Unity UI](#)

HudShieldMeter MonoBehaviour

Overview

The HudShieldMeter behaviour displays a meter representing the player character's shield strength if it has a [ShieldManager](#) attached. Shields are broken into multiple recharging steps and the current step can be broken if fully depleted. The steps are represented in the HUD by the [HudShieldMeterStep](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Step Prototype	HudShieldMeterStep	A shield meter step (in the object hierarchy) that can be duplicated for multiple steps.

See Also

[ShieldManager](#)

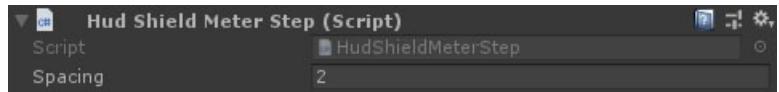
[Unity UI](#)

HudShieldMeterStep MonoBehaviour

Overview

The HudShieldMeterStep behaviour represents a single shield step for the [HudShieldMeter](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Spacing	Integer	The spacing between steps.

See Also

[ShieldManager](#)

[Unity UI](#)

HudSlowMoCharge MonoBehaviour

Overview

The HudSlowMoCharge behaviour displays a simple meter bar for the player character's slow-mo charge (sometimes called focus or adrenaline in games) if it detects they have a [SlowMoSystem](#) attached.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

[SlowMoSystem](#)

[Unity UI](#)

HudStaminaBar MonoBehaviour

Overview

The HudStaminaBar behaviour is used to display the player character's stamina levels if it detects they have a [StaminaSystem](#) attached.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

[StaminaSystem](#)

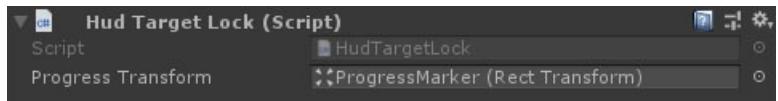
[Unity UI](#)

HudTargetLock MonoBehaviour

Overview

The HudTargetLock behaviour represents a single target lock marker. It can be used for full and partial target locks. The markers are managed by the [HudTargetLockMarkers](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Progress Transform	RectTransform	The transform used to show the progress of the target lock. This size will reduce as the lock strength increases.

See Also

[The Player HUD]

[Unity UI](#)

HudTargetLockMarkers MonoBehaviour

Overview

The HudTargetLockMarkers behaviour connects to a player character or weapon's targeting system and displays target markers for each lock. Each target lock marker must have a script implementing the `ITargetLock` interface, such as the [HudTargetLock](#) behaviour.

Inspector



Properties

Name	Type	Description
Source	Dropdown	Where to get find the targeting system on the player character. Weapon checks the selected weapon. Character checks the root of the player character. CharacterChildren checks any child objects of the child character. If CharacterChildren is selected, then be careful not to use a targeting system on any weapons as these could conflict.

See Also

[The Player HUD]

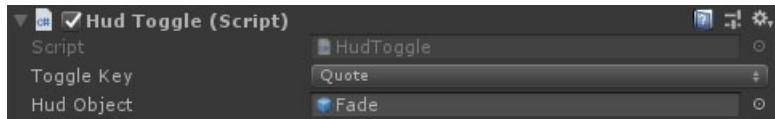
[Unity UI](#)

HudToggle MonoBehaviour

Overview

The HudToggle behaviour is used to hide or show the HUD (useful for taking screenshots).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Toggle Key	Dropdown	The keyboard key which toggles the HUD visibility.
HUD Object	GameObject	The gameobject containing the HUD UI. Must be a child of this object.

See Also

[The Player HUD]¹

[Unity UI](#)

HudWieldableHasFlashlightCondition MonoBehaviour

Overview

The HudWieldableHasFlashlightCondition behaviour is used to activate / deactivate objects in the HUD when the player selects a weapon with a flashlight attached (or a component such as the laser pointer which uses the same underlying system). The main use for this is hiding / showing touchscreen controls based on the selected weapon.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Conditional Objects	GameObject Array	The objects to show when the selected wieldable has a flashlight attached.

See Also

[Unity UI](#)

HudWieldableHasModeSwitcherCondition MonoBehaviour

Overview

The HudWieldableHasModeSwitcherCondition behaviour is used to activate / deactivate objects in the HUD when the player selects a weapon with a [ModularFirearmModeSwitcher](#) component. The main use for this is hiding / showing touchscreen controls based on the selected weapon.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Conditional Objects	GameObject Array	The objects to show when the selected wieldable has a flashlight attached.

See Also

[Unity UI](#)

HudWieldableTypeCondition MonoBehaviour

Overview

The HudWieldableTypeCondition behaviour is used to activate / deactivate objects in the HUD based on the weapon they have selected. The main use for this is hiding / showing touchscreen controls.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Firearm Object	GameObject	The object to show when the selected wieldable is a firearm.
Melee Weapon Object	GameObject	The object to show when the selected wieldable is a melee weapon.
Thrown Weapon Object	GameObject	The object to show when the selected wieldable is a thrown weapon.
Wieldable Tool Object	GameObject	The object to show when the selected wieldable is a wieldable tool.

See Also

[Unity UI](#)

HudWorldSpacePositionTracker MonoBehaviour

Overview

The HudWorldSpacePositionTracker behaviour is used to track the position of a [HudWorldSpaceTarget](#) in the player character's hierarchy and use it to align an item on the [HUD](#) to match. An example would be attaching this to the HUD crosshair to allow gun movement to affect the crosshair position.

Inspector



Properties

The MenulInputToggle behaviour has no properties exposed in the inspector.

See Also

[The Player HUD](#)

[Unity UI](#)

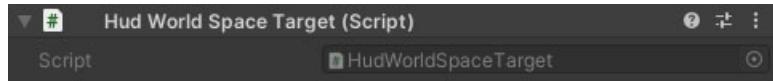
[HudWorldSpaceTarget](#)

HudWorldSpaceTarget MonoBehaviour

Overview

The HudWorldSpaceTarget behaviour is used to identify a target object in the player character hierarchy for a [HudWorldSpacePositionTracker](#) to align to.

Inspector



Properties

The MenuInputToggle behaviour has no properties exposed in the inspector.

See Also

[The Player HUD](#)

[Unity UI](#)

[HudWorldSpacePositionTracker](#)

Save Games

Overview

NeoFPS uses a comprehensive but flexible system for saving game state which is designed to achieve the same effect you see in top single-player first person shooters. The player should be able to quick-save the game at any point without a perceptible freeze, and then re-load to the exact same state down to the level of bullets in flight and rigidbodies mid-fall.

The system can also be set up to save as much or as little information as desired. The mentioned quick-saves might be the desired effect for a narrative game, but a rogue-lite style FPS might want to save limited information on inventory and character skills only, while a first person platformer might want to use checkpoints that save the active spawn point and score.

Quick, Auto and Manual Saves

The save system has 2 user controlled save types that are enabled by default. On standalone builds, the player can quick-save using the **F5** key, and quick-load using the **F9** key (these can be remapped in the game options). The player can also manually save the game from the in-game menu and give the save a specific name to help understand what point in the game they had reached (commonly called a hard save).

In most modern FPS games it is also common to save the game state at specific points in a scene or after specific actions. The save system can trigger auto-saves via scripts, and includes an example behaviour called [AutoSaveBehaviour](#).

Each of the above save types can be enabled or disabled in the [SaveGameManager](#) settings.

Save Browser

The NeoFPS samples provides a save game browser in the main menu, implemented in the NeoFPS sample UI system. The same functionality can be recreated fairly simply in another UI, though it requires some scripting knowledge.

The save browser lists all the of the available saves, along with the following details:

- **Name** - For manual saves, this can be set by the player. For quick and auto saves, this is set in the scene save info.
- **Save Type** - Quick, auto or manual save.
- **Save Date & Time** - The moment the save button was hit.
- **Thumbnail** - An image that represents the save. For quick and auto saves, the save system can take a screenshot on save, get a texture from the scene save info, or get a texture from the save game manager. Manual saves cannot use a screenshot (it would just be a menu).

There is also a version of the save browser in the in-game menu which only lists manual saves, and allows you to create a new manual save or choose one of the existing ones to overwrite.

From the main menu, you can also use the **Continue** button to load from the latest save (this can be set to specific types or all types in the manager).

Overrides and Persistence

Besides the standard game saves, the save game system can also be used for other purposes where serializing and deserializing scene or object data is required (either to disk or to a memory buffer). One example of this is persisting character data between scenes as the player progresses through the game. In this situation, the game mode tells the save manager to save the data for the player and character objects, and then reloads them when the new scene has loaded.

In the case of persistence, it is only desired to save and load specific data such as the character health and inventory. Information such as character position, aim and animation states could actually cause some major problems if they did not match the new scene. To achieve this, the NeoFPS save system supports a feature called overrides. You can specify different save modes, and then each object can override what information is saved for each mode.

For more information see [Overrides and Persistence](#).

Controlling What Data Is Saved

The NeoFPS save system requires some setup for objects to control what information is saved and loaded. You can specify individual child objects and components to either include in or exclude from the save.

Components can come in 2 forms. Neo-serialized components are MonoBehaviours which implement the `INeoSerializableComponent` interface. This is the best method of serialization. If you want to serialize a component that you do not have control over the script for (such as an inbuilt Unity component, or a script from another asset), there must be a formatter created for it which understands what data to save and load.

For more information see [Serializing Data](#).

Objects that are instantiated at runtime can also be recreated from their prefabs as long as that prefab is registered with either the save game manager or the scene save info for the scene it is instantiated in. Unity does not provide information on whether an object is being created as part of a scene load, or a runtime instantiation. This means that in order for the objects to be correctly registered in the scene hierarchy, there are some rules about how to instantiate prefabs and where.

For more information see [Runtime Objects](#).

Troubleshooting

In the event that data is not being saved properly there are a few common mistakes to look out for, and the save game inspector can be used to examine saved data and check where the problem lies.

For more information see [Troubleshooting Saves](#).

See Also

[Serializing Data](#)

[Runtime Objects](#)

[Overrides And Persistence](#)

[Troubleshooting Saves](#)

Serializing Data

Overview

The core of the NeoFPS save games system is the pairing of the [SceneSaveInfo](#) and [NeoSerializedGameObject](#). The latter is used to identify which objects, along with which of their components, to save and load. The former is required for each saved scene and inside the save files it acts as the container for all the root serialized objects in the scene.

NeoSerializedGameObject

The [NeoSerializedGameObject](#) is placed on every object that needs to be saved, or that has components that need saving. On its own it can save the transform properties of the object along with its name. It is also used to filter all of the child objects, neo-serialized components and non-neo-serialized components.

Child Objects

By default, a [NeoSerializedGameObject](#) has its child object filter set to **Exclude**. This means that it **will save all** child [NeoSerializedGameObjects](#), and to block a child from being saved, it must be added to the filter list. You can also set the filter to **Include**. This means that **none** of the child objects will be saved, unless they are specifically added to the filter list.

Child objects do not need to be direct children of the parent [NeoSerializedGameObject](#). Other [GameObjects](#) can sit inbetween them in the hierarchy. The exception to this is when objects are instantiated at runtime. When the hierarchy is rebuilt while loading from save, any runtime instantiated objects will be recreated as direct child objects of their parent [NeoSerializedGameObject](#). Therefore it is advisable to make sure that this relationship is in place from the start. For more information on the instantiation of objects at runtime, see [Runtime Objects](#).

Neo-Serialized Components

Neo-serialized components are any monobehaviour which implements the [INeoSerializableComponent](#) interface, requiring the following 2 methods:

```
public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode);
public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo);
```

The **nsgo** parameter is the object the component is attached to (useful if you want to store references to other components), while the **saveMode** parameter is used for overrides (see [Overrides And Persistence](#)).

Other Components

Components which do not implement the [INeoSerializableComponent](#) interface require a "formatter" to handle writing and reading their data. The following is an example implementation:

```

public class RigidbodyFormatter : NeoSerializationFormatter<Rigidbody>
{
    // Register the formatter with the save manager.
    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
#if UNITY_EDITOR // Required for the formatter to be recognised when selecting components to serialize
    [UnityEditor.InitializeOnLoadMethod]
#endif
    static void Register()
    {
        NeoSerializationFormatters.RegisterFormatter(new RigidbodyFormatter());
    }

    private static readonly NeoSerializationKey k_VelocityKey = new NeoSerializationKey("velocity");

    protected override void WriteProperties(INeoSerializer writer, Rigidbody from)
    {
        writer.writeValue(k_VelocityKey, from.velocity);

        // Write other properties here...
    }

    protected override void ReadProperties(INeoDeserializer reader, Rigidbody to)
    {
        Vector3 v;
        if (reader.tryReadValue(k_VelocityKey, out v, Vector3.zero))
            to.velocity = v;

        // Read other properties here...
    }
}

```

Inside the `ReadProperties()` method you can also perform logic such as simulating a particle system or starting a coroutine.

Formatters allow you to save components which you do not have source control over, such as in-built Unity components or the scripts from other assets.

SceneSaveInfo

The [SceneSaveInfo](#) behaviour must be placed in each scene that needs to be saved. It allows you to specify the scene name, along with a thumbnail to show in the save browser if the manager is not set to use a screenshot. It also acts as a root container for each of the `NeoSerializedGameObjects` in the scene (though they should not be its child in the scene hierarchy).

The `SceneSaveInfo` component can also be used to register prefabs that can be instantiated at runtime and rebuilt by the save system. For more information on this see [Runtime Objects](#).

The Binary Serializer / Deserializer

NeoFPS uses a custom written binary serializer and deserializer for save games. It is designed with the following principles in mind: it should be very fast to gather and write the save data. It should be very robust when reading the data.

Speed is important when writing saves so that the save system does not cause large frame hitches and interfere with the flow of the game. To achieve this, the serializer uses pre-allocated buffers and keeps the save size small by storing the values using a hash instead of a name. This is very different from save systems which prioritise human readability by saving in JSON or XML formats, but makes a huge difference to performance. The actual file writing from the buffers to the save file is performed on a separate thread to the rest of the game.

Robustness is most important when reading because there is no guarantee that the saved data will be up to date with the game when it is run. The player might save the game and then not return to it until a long time has passed and the game has received multiple content updates. The deserializer builds a map of all the properties that is queried by the gameobjects and components in the scene immediately after the scene itself has loaded. Firstly, any objects are destroyed or instantiated as required to recreate the hierarchy in the save file (objects are only destroyed here if they were destroyed in the save so new content is not affected).

After that, all the objects and their components are asked to take the data they need from the save data in turn. This allows the components to handle missing data gracefully while redundant data will just be ignored. Speed is also less important at the point of deserialization as it is usually behind a loading screen and not during ongoing play.

The custom serializer uses what's called "unsafe" code to achieve high performance. This means that it uses pointers to move around the buffers and break values into individual bytes. This requires the code to be placed in an [assembly definition](#) which enables unsafe code for its contents. This also means that the serializer cannot work for WebGL builds. For this situation, another serializer / deserializer combo has been created called the SafeSerializer. This relies on the .NET binary formatter and streams. It has a higher performance overhead than the custom version but is able to run on all platforms.

Property Keys And Hashes

For the best performance, when writing your own formatter and `INeoSerializableComponent` scripts, you can store the property key hashes instead of saving and loading using the key names. If you save or load a value using a string then it will be hashed there and then. If this is done thousands of times in a frame then it can be a very expensive process. To make storing the hash easier, you can use the `NeoSerializationKey` helper class. Some examples of writing a value are as follows:

```
public class ExampleNeoComponent : MonoBehaviour, INeoSerializableComponent
{
    // Create a static readonly key helper once which can be used multiple times
    private static readonly NeoSerializationKey k_PropertyKey = new NeoSerializationKey("propertyName");

    public int propertyValue = 5;

    public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode)
    {
        // Use the property name directly. Easiest, and good for prototyping, but poor performance
        writer.writeValue("propertyName", v);

        // Hash the property name. This is what the serializer does behind the scenes in the previous case
        writer.writeValue(NeoSerializationUtilities.StringToHash("propertyName"), v);

        // Use the key that was hashed once on creation. Best performance
        writer.writeValue(k_PropertyKey, v);
    }

    // ...
}
```

Component And Object References

The save system can serialize references to other NeoSerializedGameObjects or the components on them. It cannot serialize references to components on an object that does not have a NeoSerializedGameObject component attached and is not also serialized.

The following helper functions exist to aid in saving or loading references:

```
public class ExampleNeoComponent : MonoBehaviour, INeoSerializableComponent
{
    public GameObject testGameObject;
    public NeoSerializedGameObject testNsgo;
    public Transform testTransform;
    public NeoComponent testComponent;

    public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode)
    {
        writer.WriteGameObjectReference("testGameObject", testGameObject, nsgo);
        writer.WriteNeoSerializedGameObjectReference("testNsgo", testNsgo, nsgo);
        writer.WriteTransformReference("testTransform", testTransform, nsgo);
        writer.WriteComponentReference("testComponent", testComponent, nsgo);
    }

    public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo)
    {
        reader.TryReadGameObjectReference("testGameObject", out testGameObject, nsgo);
        reader.TryReadNeoSerializedGameObjectReference("testNsgo", out testNsgo, nsgo);
        reader.TryReadTransformReference("testTransform", out testTransform, nsgo);
        reader.TryReadComponentReference("testComponent", out testComponent, nsgo);
    }
}
```

See Also

[NeoSerializedGameObject MonoBehaviour](#)

[SceneSaveInfo MonoBehaviour](#)

Runtime Objects

Overview

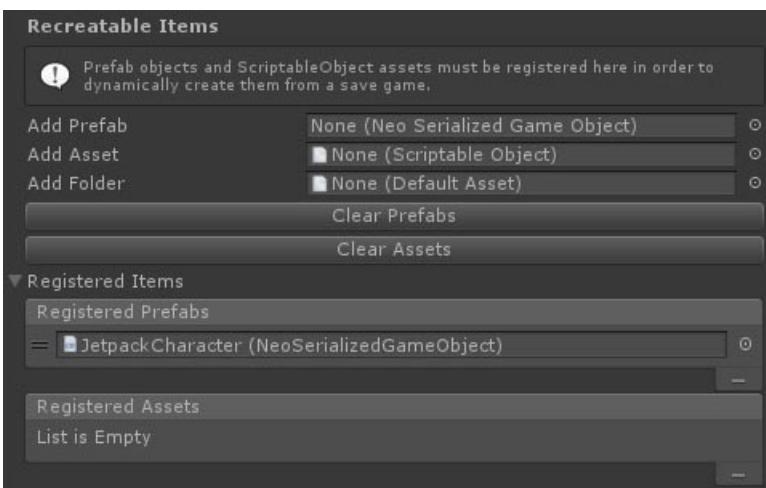
The NeoFPS save system is able to save and load objects that are instantiated at runtime, providing they adhere to the following rules:

1. The prefab must have a [NeoSerializedGameObject](#) on its root object.
2. The prefab must be registered with the save system
3. The object has to be instantiated by the the scene's [SceneSaveInfo](#) object or a parent [NeoSerializedGameObject](#) that is correctly set up.

Registering Prefabs

Prefabs can be registered in 2 places: on the [SaveGameManager](#) asset (*NeoFPS/Resources/FpsSettings_SaveGames.asset* by default), or on the scene's [SceneSaveInfo](#) object. Prefabs registered on the save game manager asset will be available at all times, however they (and all the assets such as textures and audio clips they reference) will be loaded into memory at all times. If the prefab is only supposed to be available in a few scenes then it should be registered in those scenes only.

The process for registering a prefab is the same for both situations. The [SaveGameManager](#) asset and the [SceneSaveInfo](#) monobehaviour both have the following section in the inspector:



To register a prefab, you can drag it from the project view into the **Add Prefab** field. Alternatively you can drag a folder from the project view onto the **Add Folder** field and every prefab in that folder or its subfolders that has a [NeoSerializedGameObject](#) on its root will be registered. Each item will only be registered once, and duplicate items will be skipped.

To view the registered prefabs, you can expand the **Registered Items** foldout at the bottom of the inspector. The order of the items has no effect. You can remove an item by selecting the marker on the left (=) and hitting the - button at the bottom of the list. You can also click the **Clear Prefabs** button to remove all registered prefabs from the list.

Instantiating Prefabs

Instantiating a serializable prefab must be done via the following methods on a [NeoSerializedGameObject](#) or [SceneSaveInfo](#) monobehaviour:

```
public T InstantiatePrefab<T>(T prototype);
public T InstantiatePrefab<T>(T prototype, Vector3 position, Quaternion rotation);
public T InstantiatePrefab<T>(int prefabID, int serializationKey);
```

For the first two methods, if the prefab that's instantiated does not have a [NeoSerializedGameObject](#) at its root then it will be instantiated the standard Unity way. Otherwise it will be given unique serialization ID, registered with the parent object or scene and flagged as a runtime object.

The second method is usually used by the save game manager when rebuilding the scene hierarchy, but it can also be used in special situations where you don't want the save system to generate unique IDs.

Destroying Objects

You can destroy an object using the standard `Destroy()` method. If the object was instantiated at runtime then it will simply be unregistered by the parent object or scene. If the object was in the scene hierarchy on load, then it will be flagged for destruction by the scene or parent object.

Changing Hierarchy

Each [NeoSerializedGameObject](#) is attached to its parent or the scene on load or instantiation. If you want to move an object from one [NeoSerializedGameObject](#) parent to another, then you can use the `NeoSerializedGameObject.SetParent()` method. Using the `transform.SetParent()` method as standard will cause unexpected results as the object will still be registered with its original parent. This means that on saving and loading using the save system, the object will be recreated as though it was still attached to the original.

Runtime instantiated objects must be a direct child of their parent [NeoSerializedGameObject](#). The save game `InstantiatePrefab()` and `SetParent()` methods will enforce this, but the object hierarchy must not be changed separately.

Loading Process

When a scene is loaded from a save game, the system starts by loading the scene as standard. As soon as the scene is available, the [NeoSerializedGameObject](#) hierarchy is traversed and any changes to the hierarchy made. In order from the root [NeoSerializedGameObject](#) objects to the children, any objects that are flagged for destruction in the save are destroyed. Any runtime instantiated objects registered with the parents are instantiated via their prefab ID and assigned their original serialization ID. Once the hierarchy has been rebuilt, it is traversed again to load all the stored component data.

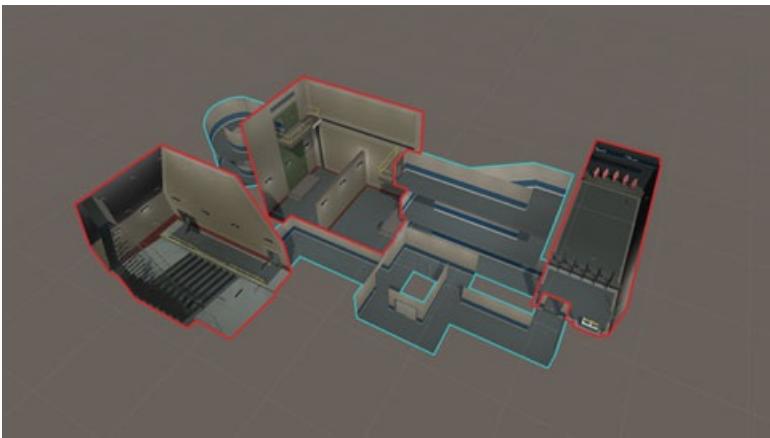
See Also

[Troubleshooting](#)

Multi-Scene Saves

Overview

NeoFPS features a simple system to break levels into separate sub-scenes that are streamed in and out as you move around. This allows you to create much larger environments than you would otherwise be able to, since areas that are not visible or accessible do not need to be loaded into memory and simulated.



The multi-scene save system sits on top of that and uses the NeoFPS save game system to save the contents of each sub-scene to a memory buffer when the scene is unloaded. This memory buffer is stored in the main scene, and included with your regular save game data (such as quick-saves). When the sub-scene is streamed back in later, the contents will be recreated using the save data in the buffer. This means that sub-scenes can be persistent and maintain any changes.

The Main Scene

The main scene is responsible for the persistent systems when playing the game. This includes the player character, the game mode, the pooling system, UI and the main game save info.

Saving your scene data is the responsibility of the [SceneSaveInfo](#) behaviour that should be attached to an object at the root of the main scene.

This main scene is the scene which is loaded first to start the level. The game mode object in this scene should have a [SubSceneManager](#) component attached, which is then responsible for loading the starting sub-scene. This is also where the memory buffers for the individual sub-scenes are stored and managed.

Sub-Scenes

The sub-scenes are loaded in additively as you progress through the level. Note that all sub-scenes and the main scene *must be added to your project build settings*.

The available sub-scenes are referenced in a [SubSceneCollection](#) asset and loaded / unloaded using the index they appear in this collection. This is done to prevent the sub-scene save data breaking if the scene index changes in the project's build settings. Behind the scenes, the scenes in this collection are referenced by name. This means that *renaming a scene will still break its connection to the sub-scene save data*. However, changing it's build index or parent folder will not.

In order to stream a sub-scene in or out, you can use the [SubSceneLoadTrigger](#) and [SubSceneUnloadTrigger](#) behaviours. These are attached to trigger colliders, and when the player character enters the trigger they will load or unload the scene. They should reference the same [SubSceneCollection](#) asset that the [SubSceneManager](#) behaviour does, and then you specify which scene index to use. You can also call the following methods on the manager:

```
SubSceneManager.LoadScene(int index);  
SubSceneManager.UnloadScene(int index);
```

In order for each sub-scene to be saved, it must have an object at its root with an [AdditiveSceneSaveInfo](#) behaviour attached. This

acts as the root of the scene in the save game data and handles recreating and traversing the hierarchies. Your scene contents are then serialized using the [standard NeoFPS save game components](#).

For testing your scenes, it is fine to add an [FpsSoloGameMinimal](#) to your sub-scene. Since the main scene is loaded before any sub-scenes, this (and the object it's attached to) will be destroyed on start if the sub-scene is played as part of the whole level. If it is instead loaded on its own, then this will allow you to spawn and run around the sub-scene. However, neighbouring sub-scenes will not be loaded in this situation.

If you open the main scene and a number of the sub-scenes at the same time (using load additive) in the editor, then hitting play will unload all the sub-scenes except the default starting sub-scene as set in the [SubSceneManager](#).

See Also

[Save Games](#)

[Serializing Data](#)

Overrides and Persistence

Overview

The NeoFPS save system has multiple uses. In the samples it is also set up to save character data for persistence when changing scenes, but it can also be extended to for other purposes.

Persistence

When progressing through a game you often have levels which continue on from each other. In this situation, you need certain data to carry over into the new scene, while other data can be discarded. For example, you would want the character to spawn at the position of the spawn point in the new scene, in a neutral state, but you would probably want them to retain their health and the contents of their inventory from the previous scene.

The actual persistence of the data is handled by the [game mode](#). This tells the save game manager to save the player object and character object to a byte buffer using the persistence save mode. If you want to extend the persistence system or create your own, then it can be done with the following methods:

```
public static bool SaveGameManager.SaveGameObjectsToBuffer(NeoSerializedGameObject[] objects, SaveMode saveMode);  
public static bool SaveGameManager.LoadGameObjectsFromBuffer(NeoSerializedGameObject[] objects);
```

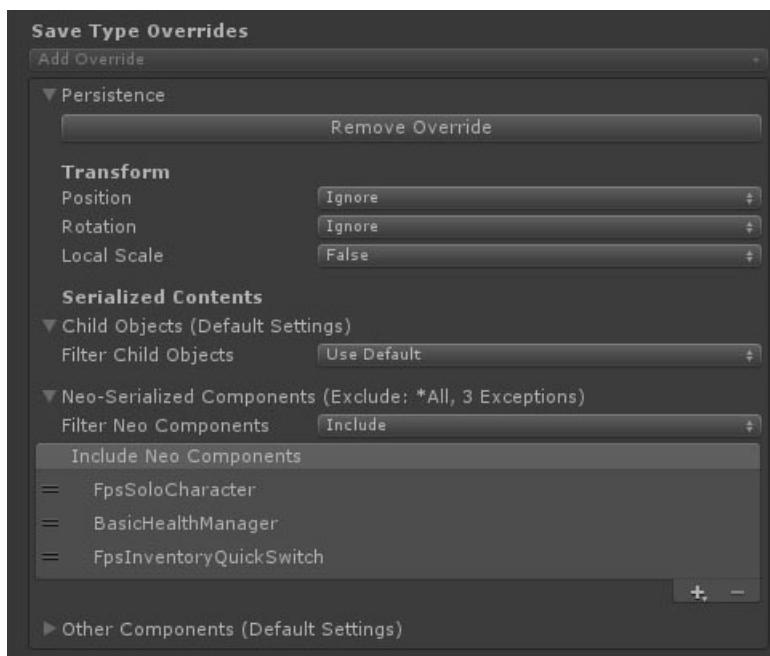
These both take an array of objects and will serialize them in order. As long as the objects in the save method match the objects in the load method (in terms of source prefab and order), then the requested data will be transferred between them.

Adding New Save Modes

The available save modes are set using the NeoFPS [generated constants](#) system. The first option is always the default save mode, as used by the standard quick, auto and manual saves. It is advised to keep the second option as persistence and add any new modes after this unless you're sure it isn't required for your project.

Using Overrides To Customise Saved Data

Each [NeoSerializedGameObject](#) can be set up to save different data for each game mode. The main filters and transform options are used for the default mode.



At the bottom of its inspector, you can see a section called **Save Type Overrides**. The dropdown button below this (**Add Override**) allows you to add a new override based on the available modes.

The override settings will appear below this and mirror the save settings and filters in the main inspector. Each property has a **Use Default** option so that not every property needs to be overriden. It can be limited to just transform properties, children or components as required.

See Also

[Inventory Examples](#)

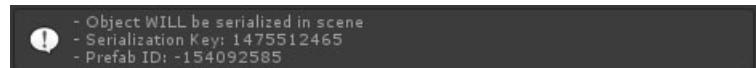
Troubleshooting The Save System

Overview

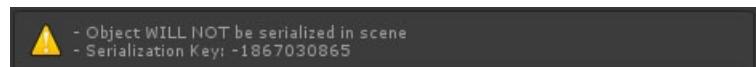
Serializing and deserializing game state is a complex topic, and it can be tricky to work out why something isn't being saved the way you would expect it to. The NeoFPS save system tries to give the visual feedback in the inspector to help diagnose issues, along with a tool called the NeoSave File Inspector, which you can use to check data that has been saved to a file.

Troubleshooting NeoSerializedGameObjects

The following is an example of the serialization info shown in the inspector for a [NeoSerializedGameObject](#):



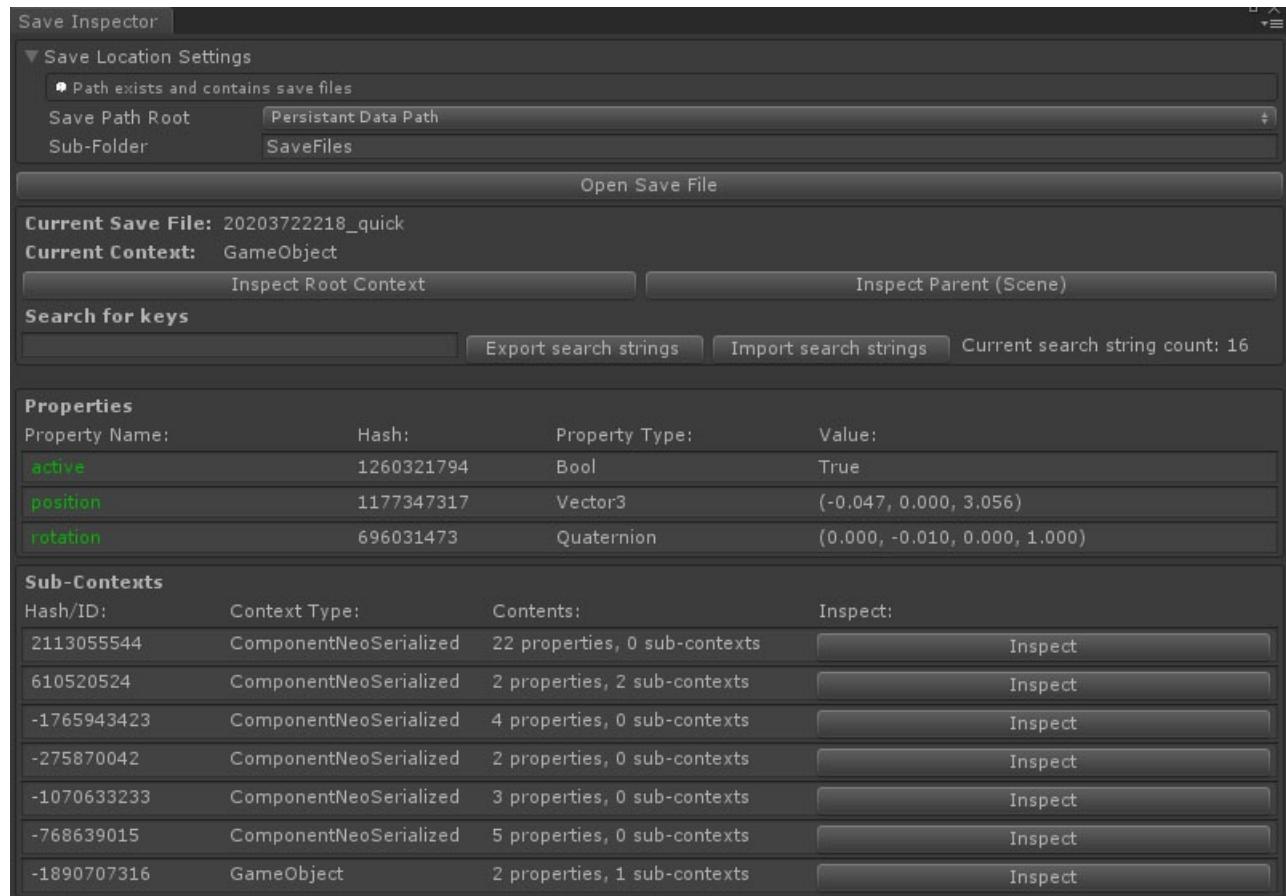
While this is an example of an object that will not be correctly saved by the save system:



If an object says it will not be saved then that is generally down to one of four reasons:

1. The scene does not have a valid [SceneSaveInfo](#).
2. The object's heirarchy does not have a [NeoSerializedGameObject](#) at its root.
3. The object or one of its parents is blocked from saving by [NeoSerializedGameObject](#) child object filters higher up the hierarchy.
4. The object was instantiated at runtime, but using default Unity instantiation and not via the NeoFPS save system.

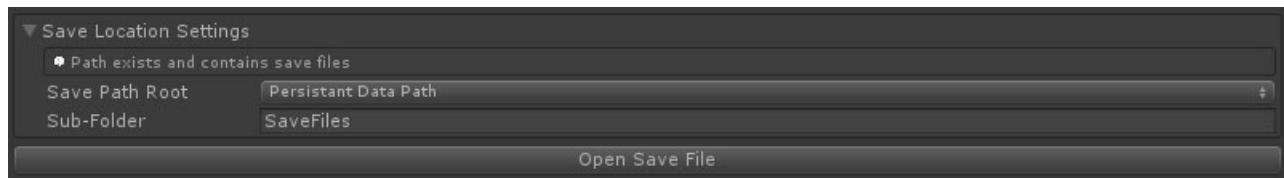
NeoSave File Inspector



The NeoSave File Inspector can be used to open save files and inspect their contents. It can be found in the menus at [Tools/NeoFPS/NeoSave File Inspector](#), or by clicking the **Save Game Inspector** button from within the [SaveGameManager](#)

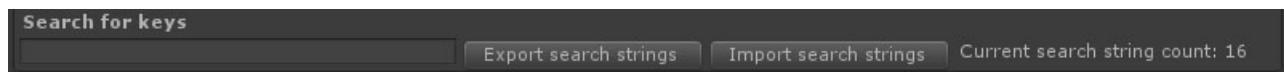
settings.

Save Location Settings



At the top of the editor is a foldout section for exploring to the project's save files. This matches the [SaveGameManager](#) settings and will display an info box that shows if the folder exists and contains save files.

Searching For Strings



NeoFPS save files are designed to use a low footprint and fast format, and not to be human readable. Data is not organised using property names, but hashes instead. This can make it tricky to understand what values in the save file inspector correspond to what properties in a component or object. To aid in this, you can register property names with the inspector, and when a hash matches one of those property names, then the name will be displayed in green besides the hash value.

You can add new property names via the **Search For Keys** text field. This will add the name and its hash to the inspector. The inspector automatically registers a number of common property names that are used in the base [NeoSerializedGameObject](#) and save file metadata. You can also export and import the property names via the **Export Search Strings** and **Import Search Strings** buttons. This is useful if you are making frequent use of the save file inspector and have specific classes you want to focus on.

Navigating The File



Near the top of the inspector is a section which displays information on your current location in the save file, and allows you to navigate up the save hierarchy.

NeoFPS save files group data into contexts. These contexts are things like *Scene*, *MetaData*, *GameObject* and *ComponentNeoSerialized* or *ComponentNeoFormatted*. These contexts are stored in a tree hierarchy, similar to the scene hierarchy. At the root of the save file is always the *Root* context. At any time while inspecting the save file contents, you can return to the save file root by using the **Inspect Root Context** button. You can also move back up the hierarchy towards the root one node at a time using the **Inspect Parent** button. The brackets next to this display the context type of the parent.

The **Current Context** readout tells you what context you are currently inspecting.

Properties			
Property Name:	Hash:	Property Type:	Value:
active	1260321794	Bool	True
position	1177347317	Vector3	(-0.047, 0.000, 3.056)
rotation	696031473	Quaternion	(0.000, -0.010, 0.000, 1.000)

The **Properties** section displays the values that are currently saved for the inspected context and contains the following info:

- **Property Name** displays the name of the property if a property name has been registered with the inspector that matches the property's hash (see above).
- **Hash** is the hashed property name that was used to save the data.
- **Property Type** is the type of the value (it will also say if it is an array).

- **Value** is the saved value. In the case of arrays, you can expand this section to see the individual values (large arrays will show the first 50 values, followed by a "..." to signify there is more data in the file).

Sub-Contexts			
Hash/ID:	Context Type:	Contents:	Inspect:
2113055544	ComponentNeoSerialized	22 properties, 0 sub-contexts	<input type="button" value="Inspect"/>
610520524	ComponentNeoSerialized	2 properties, 2 sub-contexts	<input type="button" value="Inspect"/>
-1765943423	ComponentNeoSerialized	4 properties, 0 sub-contexts	<input type="button" value="Inspect"/>
-275870042	ComponentNeoSerialized	2 properties, 0 sub-contexts	<input type="button" value="Inspect"/>
-1070633233	ComponentNeoSerialized	3 properties, 0 sub-contexts	<input type="button" value="Inspect"/>
-768639015	ComponentNeoSerialized	5 properties, 0 sub-contexts	<input type="button" value="Inspect"/>
-1890707316	GameObject	2 properties, 1 sub-contexts	<input type="button" value="Inspect"/>

The **Sub-Contexts** section shows the child contexts that the current context contains and shows the following info:

- **Hash/ID** is the ID value that was used to identify the context when saved. This could be a generated ID, a hashed name, or a sequential number based on the context type. The only rule is that it is unique within the current parent context.
- **Context Type** is the type of the context.
- **Contents** shows the number of properties and child contexts for this context.
- The **Inspect** button will change the currently inspected context to this one. You can reach the previous context using the **Inspect Parent** button.

Loading vs Initialisation Order

Another area where it can be complicated to enforce correct behaviour is in the execution order of loading objects from save data vs initialising them using the standard Unity `Awake()` and `Start()` methods.

Unfortunately Unity only provides limited functionality to detect when a scene has been loaded, and events are only fired once Unity has initialised all objects with `Awake()`. It is possible to halt async scene loads once all objects have been created, but the scene state is not sufficient for searching for or instantiating objects.

The NeoFPS save system attempts to get around this by using [Script Execution Order](#) with the [SceneSaveInfo](#) behaviour to ensure that it is the first object to be awakened in the scene, which in turn triggers the loading from save data before the rest of the scene objects are awake. The reasoning behind this is that the save system can recreate the scene hierarchy and destroy any objects that require it before they are initialised. Runtime objects that are instantiated will call `Awake()` immediately, while scene objects will call `Awake()` once the hierarchy has been fully rebuilt.

If you find that your components are being loaded after they have already been initialised then check in the [Script Execution Order](#) settings that the top value is **NeoSaveGames.SceneSaveInfo**.

It is also a good idea to move any common initialisation logic out of `Awake()` and into a separate method that can be called from inside both `Awake()` and `ReadProperties()` with a check that it does not occur twice. For example:

```
public class ExampleBehaviour : MonoBehaviour, INeoSerializableComponent
{
    private bool m_Initialised = false;

    void Awake()
    {
        // Do not initialise if already loaded by save system
        if (!m_Initialised)
        {
            // Perform non-shared initialisation logic here

            Initialise();
        }
    }

    void Initialise()
    {
        // Perform shared initialisation logic here

        m_Initialised = true;
    }

    public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo)
    {
        if (!m_Initialised)
        {
            // Read the relevant data here

            Initialise();
        }
        else
            Debug.LogError("Reading properties after component was initialised. Check script execution order of SceneSaveInfo.cs");
    }
}
```

See Also

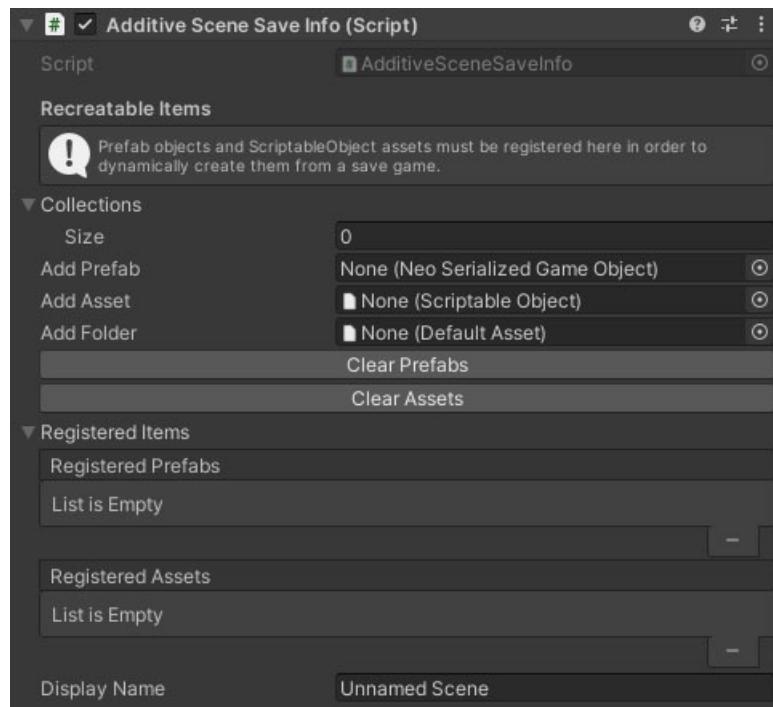
[Inventory Examples][2]

AdditiveSceneSaveInfo MonoBehaviour

Overview

The AdditiveSceneSaveInfo behaviour stores information about an additive scene for the save system. This includes the objects registered for runtime instantiation and serialization while this scene is loaded. Additive scenes are intended to be used alongside a master scene containing a [SceneSaveInfo](#).

Inspector



Properties

Name	Type	Description
Display Name	String	The title of the scene to help identify it.
Collections	[SaveGamePrefabCollection Array][3]	Predefined collections of prefabs and assets that should be available for this scene.

Recreatable Items

This section in the inspector is used to register objects with the save system so that they can be instantiated at runtime. This is only required for objects which need to be instantiated for this scene only. For more information see [Runtime Objects](#).

See Also

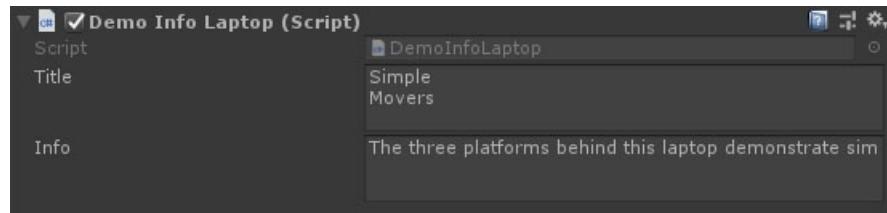
[Runtime Objects](#)

AutoSaveBehaviour MonoBehaviour

Overview

The AutoSaveBehaviour behaviour is used to trigger auto-saves or checkpoint saves via the [NeoFPS save system](#). It can be triggered via [unity events](#), and used with the NeoFPS [contact triggers and interactive objects](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
One Shot	Boolean	Can the autosave be triggered multiple times by this behaviour.
Cooldown	Float	The time in seconds before the autosave can be triggered again. This property will only appear if One Shot is set to false.
Retry Attempts	Integer	If the save fails (because another save or load event is in progress), the behaviour will try every second this many times.

See Also

[Save Games](#)

[Unity Events](#)

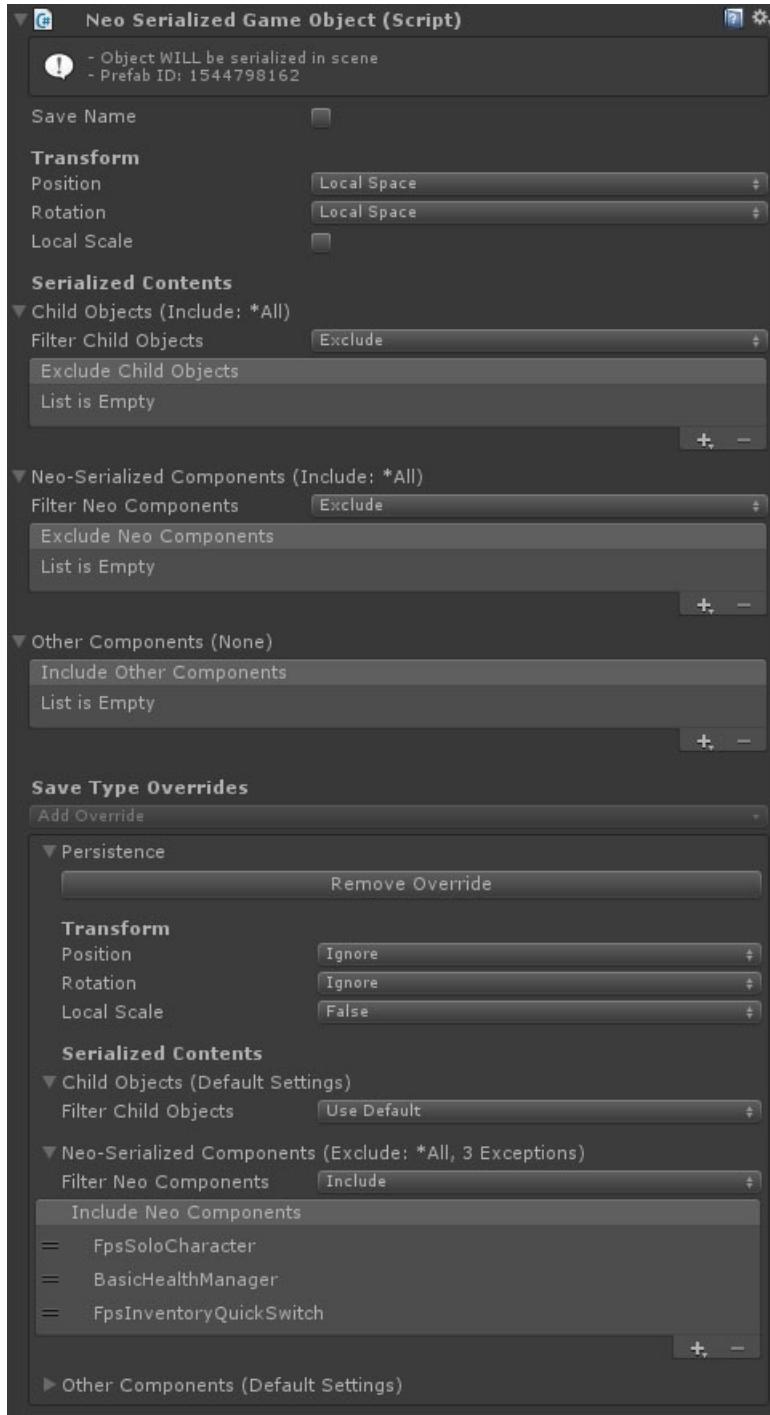
[Interaction With The World](#)

NeoSerializedGameObject MonoBehaviour

Overview

The NeoSerializedGameObject behaviour is a key component of the NeoSave system. This component must be added to any objects that need to be saved and is used to specify which of its components and child objects are serialized.

Inspector



The information box at the top of the inspector gives details on the state of the object. It will list the object's serialization key, if the object was instantiated at run-time and whether the object is correctly set up to be saved. This requires that the root object has a valid and enabled NeoSerializedGameObject attached and none of its NeoSerializedGameObject parents are disabled or filtered out.

Properties

NAME	TYPE	DESCRIPTION
Save Name	Boolean	Save and load the object's name. This is not required in most cases.
Position	Dropdown	How to save and load the object's transform position. The available options are Local Space , World Space and Ignore .
Rotation	Dropdown	How to save and load the object's transform rotation. The available options are Local Space , World Space and Ignore .
Local Scale	Boolean	Should the object transform local scale be saved and loaded or ignored. Saving scale in world space is not an option .

Child Objects Foldout

NAME	TYPE	DESCRIPTION
Filter Child Objects	Dropdown	How to filter child objects for saving and loading. If set to Exclude then any objects added to the list below will not be saved. If set to Include then only objects in the list below will be saved.

To add objects to the list, click the + button and the available objects will be shown in a dropdown.

Neo-Serialized Components Foldout

NAME	TYPE	DESCRIPTION
Filter Neo Components	Dropdown	How to filter the neo-serialized components attached to this game object. If set to Exclude then any objects added to the list below will not be saved. If set to Include then only objects in the list below will be saved.

To add objects to the list, click the + button and the available components will be shown in a dropdown. A neo-serialized component is any behaviour that implements the `INeoSerializedComponent` interface. For more information see [Serializing Data](#).

Other Components Foldout

Components that do not implement the `INeoSerializedComponent` interface can be serialized by adding them to this list. In order for a component (that is not a neo-serialized component) to appear in this list, it must be attached to this gameobject and have a valid formatter registered with the save system. For more information see [Serializing Data](#).

Save Type Overrides

The NeoSave system allows you to define save modes using the [generated constants](#) system. By default, the samples use the save modes **Default** for standard game saves and **Persistence** for persisting certain data such as character health and inventory between scenes. You can add overrides here by clicking the **Add Override** button. This will show the available modes except for **Default** and any that have already been overridden. You can then override the filters and properties for that save mode as required or set them to use default. For example, the characters override the persistence save mode to only save the health manager and inventory components while not saving data like position, aim and motion graph states.

See Also

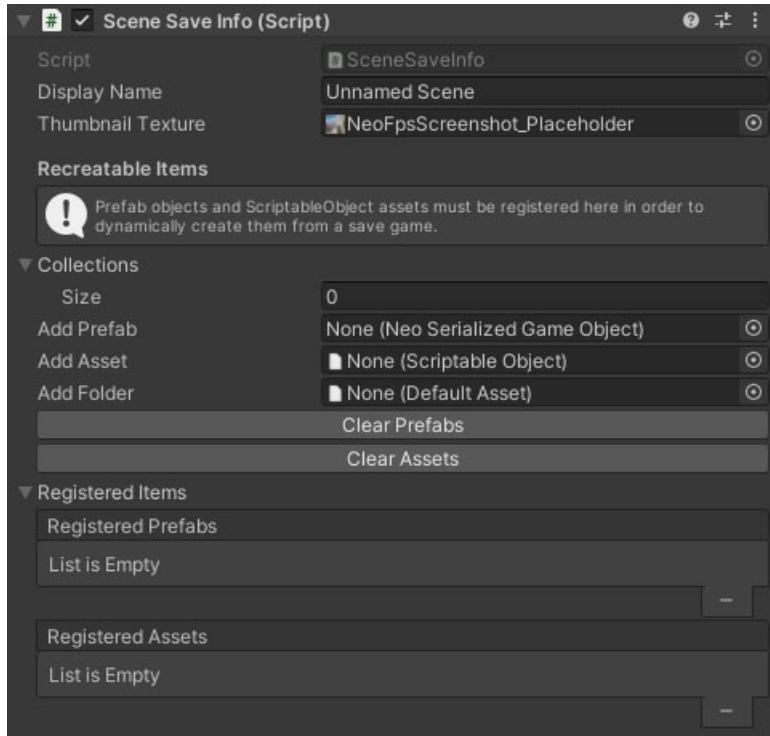
[Serializing Data](#)

SceneSaveInfo MonoBehaviour

Overview

The SceneSaveInfo behaviour stores information about the current scene for the save system. This includes the display name and thumbnail for the [save game browser][2] along with the objects registered for runtime instantiation and serialization.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Display Name	String	The title of the scene to display in the save browser.
Thumbnail Texture	Texture2D	An image to display in the save browser, depending on how the thumbnails are set up.
Collections	[SaveGamePrefabCollection Array] [3]	Predefined collections of prefabs and assets that should be available for this scene.

Recreatable Items

This section in the inspector is used to register objects with the save system so that they can be instantiated at runtime. This is only required for objects which need to be instantiated for this scene only. For more information see [Runtime Objects](#).

See Also

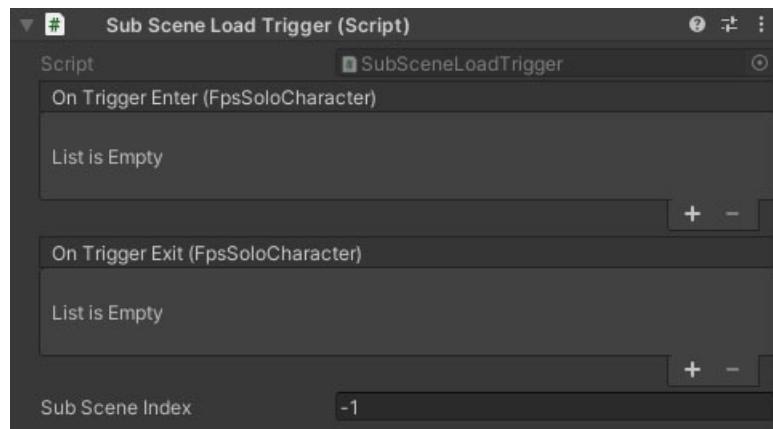
[Runtime Objects](#)

SubSceneLoadTrigger MonoBehaviour

Overview

The SubSceneLoadTrigger behaviour is attached to a trigger collider in your scenes, and tells the [SubSceneManager](#) when to load an additive sub-scene. This can be used to split your levels up into smaller chunks to save memory and processing. It is paired with a [SubSceneUnloadTrigger](#) to unload the sub-scenes.

Inspector



Properties

NAME	TYPE	DESCRIPTION
m_TooltipName	String	The name of the item in the HUD tooltip.
m_TooltipAction	String	A description of the action for use in the HUD tooltip, eg pick up.
m_Interactable.onStart	Boolean	Can the object be interacted with immediately.
m_HoldDuration	Float	How long does the use button have to be held for interaction.
m_OnUsed	UnityEvent	An event that is triggered when the object is used.
m_OnCursorEnter	UnityEvent	An event that is triggered when the player looks directly at the object.
m_OnCursorExit	UnityEvent	An event that is triggered when the player looks away from the object.
m_SubSceneIndex	Integer	The index of the scene within the SubSceneCollection to load.

See Also

[SubSceneManager](#)

[SubSceneUnloadTrigger](#)

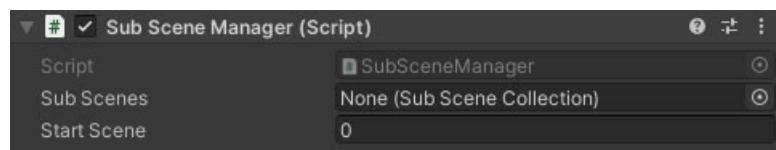
[SubSceneCollection](#)

SubSceneManager MonoBehaviour

Overview

The SubSceneManager behaviour is placed in a master scene and handles the additive loading and unloading of sub-scenes as you move around the level. It ties into the save system to save the sub-scene contents when it is unloaded, and then recreate the sub-scene based on the save-data when it is loaded back in. It is used alongside [SubSceneLoadTrigger](#) and [SubSceneUnloadTrigger](#) behaviours in the scenes to control when their neighbours are loaded and unloaded.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Sub Scenes	SubSceneCollection	The sub-scenes to load/unload while progressing through this level.
Start Scene	Integer	The index (within the sub-scene collection to load by default (if not loading from a save game).

See Also

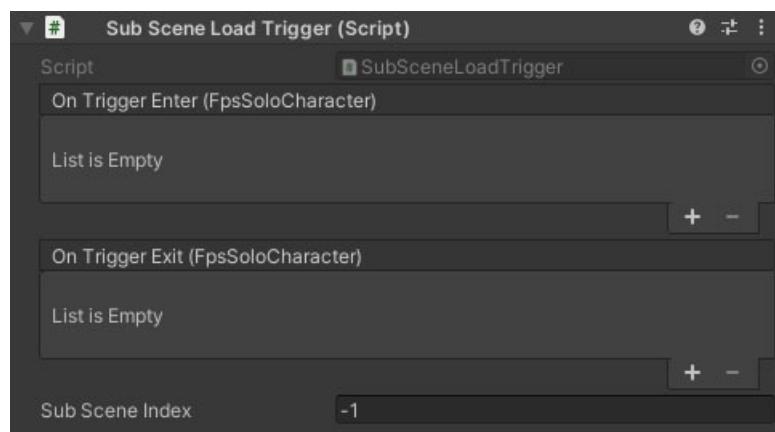
[Serializing Data](#)

SubSceneOperation MonoBehaviour

Overview

The SubSceneOperation behaviour is a simple behaviour that is used in conjunction with others such as [interactive objects](#) to enable loading and unloading subscenes. Its inspector simply specifies a sub-scene index, and then you can use [unity events](#) on other components to trigger the operations.

Inspector



Properties

NAME	TYPE	DESCRIPTION
m_SubSceneIndex	Integer	The index of the scene within the SubSceneCollection to load.

See Also

[Interactive Objects](#)

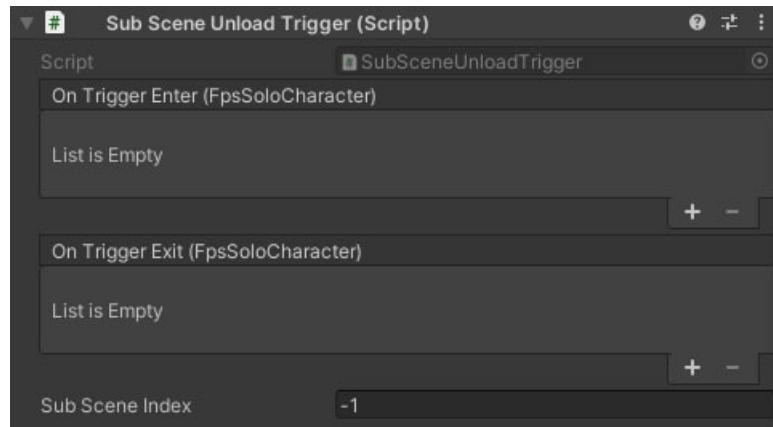
[Unity Events](#)

SubSceneUnloadTrigger MonoBehaviour

Overview

The SubSceneUnloadTrigger behaviour is attached to a trigger collider in your scenes, and tells the [SubSceneManager](#) when to unload an additive sub-scene. This can be used to split your levels up into smaller chunks to save memory and processing. It is paired with a [SubSceneLoadTrigger](#) to load the sub-scenes.

Inspector



Properties

NAME	TYPE	DESCRIPTION
m_TooltipName	String	The name of the item in the HUD tooltip.
m_TooltipAction	String	A description of the action for use in the HUD tooltip, eg pick up.
m_Interactable.onStart	Boolean	Can the object be interacted with immediately.
m_HoldDuration	Float	How long does the use button have to be held for interaction.
m_OnUsed	UnityEvent	An event that is triggered when the object is used.
m_OnCursorEnter	UnityEvent	An event that is triggered when the player looks directly at the object.
m_OnCursorExit	UnityEvent	An event that is triggered when the player looks away from the object.
m_SubSceneIndex	Integer	The index of the scene within the SubSceneCollection to unload.

See Also

[SubSceneManager](#)

[SubSceneLoadTrigger](#)

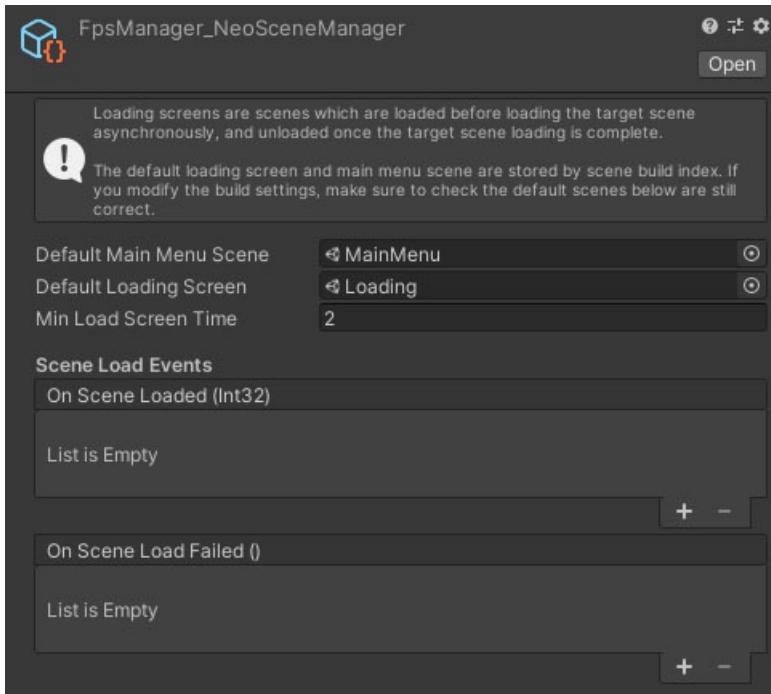
[SubSceneCollection](#)

NeoSceneManager ScriptableObject

Overview

The NeoSceneManager asset specifies the default loading screen scene and exposes events to the inspector.

Inspector



Properties

NAME	TYP	DESCRIPTION
Default Main Menu Scene	Scene	The main menu scene to use by default if none are specified. This is stored by index, so if changing the build settings, make sure to check this after.
Default Loading Screen	Scene	The loading screen scene to use by default if none are specified. This is stored by index, so if changing the build settings, make sure to check this after.
Min Load Time	Float	The minimum amount of time the loading screen should stay visible before the scene is fully activated.
On Scene Loaded	Unity Event	An event called when the new scene is correctly loaded.
On Scene Load Failed	Unity Event	An event called when the new scene fails to load.

See Also

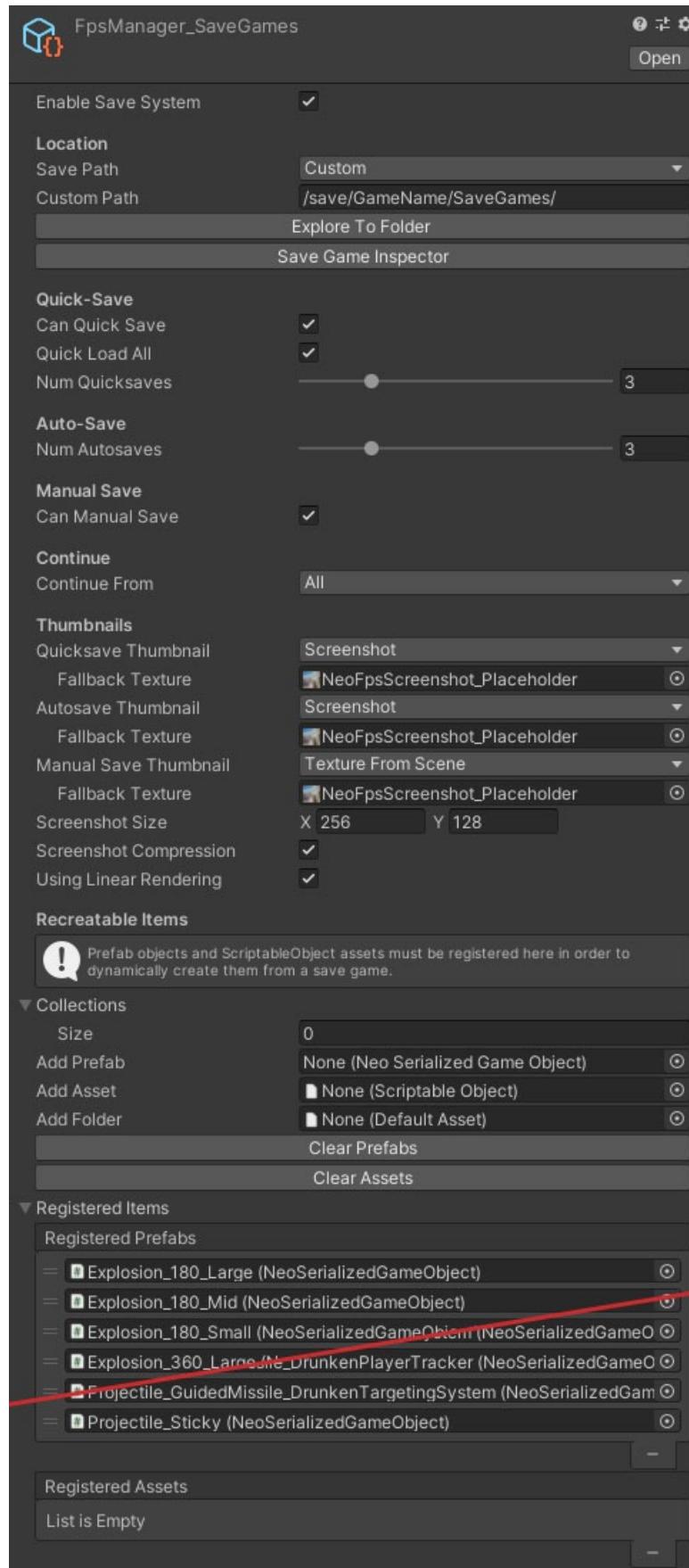
[Unity Events](#)

SaveGameManager ScriptableObject

Overview

The SaveGameManager asset is the central access point for the save games system. It specifies the various save game settings as well as registering objects for instantiation at runtime.

Inspector



Properties

Location

NAME	TYPE	DESCRIPTION
Save Path	Dropdown	The available save file locations.
Sub-Folder	String	A sub-folder within the save path folder to place save files in.

You can use the **Explore To Folder** button to open the save games folder in the OS file explorer.

You can also use the **Save Game Inspector** button to show the save game inspector (also available via *Tools/NeoFPS/Save Game Inspector*). This allows you to open a save file and browse its contents. For more information see [Troubleshooting](#)

Quick-Save

NAME	TYPE	DESCRIPTION
Can Quick Save	Boolean	Sets whether the quick-save system is enabled in this project.
Quick Load All	Boolean	If true, quick loading will load the latest quick/auto/manual save. If not then it will only load the latest quick-save.
Num Quicksaves	Integer	The number of quicksaves to maintain. If the number exceeds this value, the oldest saves will be deleted.

Auto-Save

NAME	TYPE	DESCRIPTION
Num Autosaves	Integer	The number of autosaves to maintain. If the number exceeds this value, the oldest saves will be deleted.

Manual Save

NAME	TYPE	DESCRIPTION
Can Manual Save	Boolean	Sets whether the manual system is enabled in this project.

Continue

NAME	TYPE	DESCRIPTION
Continue From	Dropdown	What type of saves the game can be continued from. Available options are None , All and Auto Save Only .

Thumbnails

NAME	TYPE	DESCRIPTION
Quicksave Thumbnail	Dropdown	Where to get the thumbnail texture for quick-saves. Options are None , Texture , TextureFromScene , Screenshot .

Name	Type	Description
(Fallback) Texture	Texture2D	If the thumbnail type is set to Texture then this is the texture that will be serialized to save files. If the thumbnail type is set to Texture From Scene or Screenshot then this texture will be used in the event those methods fail.
Autosave Thumbnail	Dropdown	Where to get the thumbnail texture for auto-saves. Options are None , Texture , TextureFromScene , Screenshot . The Texture / Fallback Texture property will appear below this if required.
Manual Save Thumbnail	Dropdown	Where to get the thumbnail texture for manual saves. Options are None , Texture , TextureFromScene , Screenshot . The Texture / Fallback Texture property will appear below this if required.
Screenshot Size	Vector2Int	If any of the thumbnail types are set to Screenshot then this property defines the resulting size of the saved screenshot texture.
Screenshot Compression	Boolean	This property defines whether the resulting texture will be compressed before serialization.
Using Linear Rendering	Boolean	Set this property to true if the project is set to use linear color space rendering. Screenshots need adapting to the correct color space or they can appear washed out.

Recreatable Items

This section in the inspector is used to register objects with the save system so that they can be instantiated at runtime. This is only required for objects which need to be instantiated for this scene only. For more information see [Runtime Objects][4].

See Also

[Serializing Data](#)

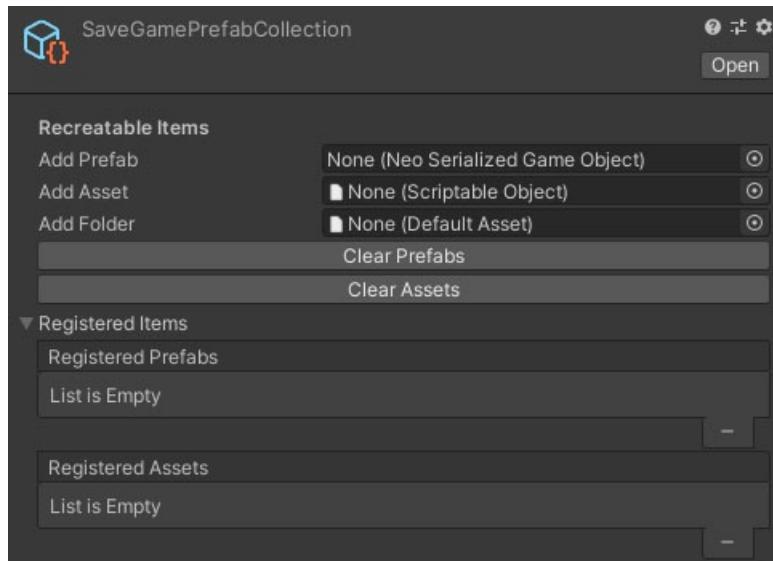
[Troubleshooting](#)

SaveGamePrefabCollection ScriptableObject

Overview

The SaveGamePrefabCollection asset is used to group together prefabs for runtime instantiation in a single collection that can be added to [SceneSaveInfo](#) or [AdditiveSceneSaveInfo](#) components, and the [SaveGameManager](#) instead of adding the prefabs directly.

Inspector



Properties

Recreatable Items

This section in the inspector is used to register objects with the collection so that they can be instantiated at runtime. For more information see [Runtime Objects][4].

See Also

[SaveGameManager](#)

[SceneSaveInfo](#)

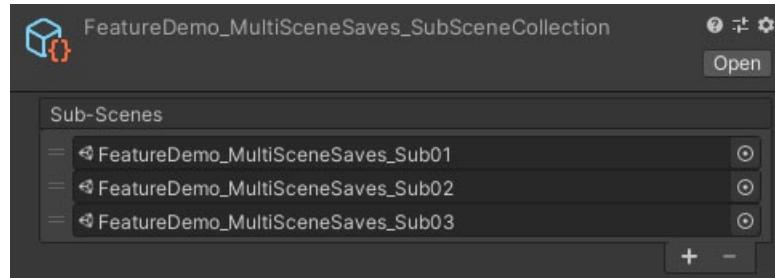
[AdditiveSceneSaveInfo](#)

SubSceneCollection ScriptableObject

Overview

The SubSceneCollection asset is used to specify the scenes that should be additively loaded and unloaded using the [SubSceneManager](#) system.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Sub Scenes	Scene Array	The sub-scenes to load/unload in the background while progressing through the level.

See Also

[SubSceneManager](#)

Graphics and Rendering

Render Pipelines

Unity provides 3 main options for rendering your in-game graphics: the built-in render pipeline (BIRP), universal render pipeline (URP) and the high-definition render pipeline (HDRP). The URP and HDRP pipelines together are called the scriptable render pipelines (SRPs). Each of the render pipelines has its own pros and cons and should be chosen carefully.

NeoFPS is set up for the built-in render pipeline out of the box, but it also comes with unity packages containing alternative shaders, prefabs, scripts and materials. These packages also replace a number of the demo materials with SRP specific version where the tools that come with the SRP would not be able to convert them for you. The best way to import the relevant package is through the **Unity Settings** page in the **NeoFPS Hub**. This also provides some simple tools for checking your project's compatibility before importing.

Built-In

The built-in render pipeline has been around for a long time and so it has the broadest support from asset publishers along with the most learning resources available. It is also the most feature complete, with the SRPs lagging behind in terms of features (though they are now catching up). Unfortunately, the way the built-in pipeline handles lighting limits its ability to compete with the latest generation of AAA titles and performance can be a problem. Despite this, the majority of new projects created in Unity use the built-in pipeline according to Unity's stats (late 2022). For more information on using the built-in render pipeline with NeoFPS see [Built-In Render Pipeline](#)

URP

The universal render pipeline is intended to become the default rendering pipeline once the built-in pipeline is eventually deprecated. It started out targeting mobile devices (as the light-weight render pipeline), but it eventually expanded out in terms of target platforms. This should be considered the performance option, though it also looks great. For more information on using the universal render pipeline with NeoFPS see [Universal Render Pipeline](#)

HDRP

The high-definition pipeline is aimed at high performance computers and latest or next generation consoles. It provides the best visual fidelity and most realistic lighting, but many devices will struggle to run at decent frame rates. For more information on using the high-definition render pipeline with NeoFPS see [High-Definition Render Pipeline](#)

Shaders

NeoFPS comes with a number of shaders for different purposes. Some, such as the water shader are purely for demo purposes and you will be able to find much better quality options from dedicated shader assets. Others, such as the scopes and optics, are more specialised for FPS games and are intended for use in complete games. The following is a rough outline of the shaders available:

- Projection optics such as holographic and red-dot sights
- Render texture scopes
- Bullet trails
- Laser pointers
- Overheat glow
- Shockwave and heat-haze distortions
- Powerup and interactive object glows

Post-Processing

Post-processing is a system where effects are applied to the final image based on the state of the render buffers after the various meshes in the scene have been rendered. This allows you to add screen-space effects such as ambient occlusion, reflections and colour correction. You can also use volumes placed in your scene to control post-processing as you move around the scene. For

example, you might want a different colour balance and exposure range when indoors compared to outside.

With the built-in pipeline, Unity requires the Post Processing Stack V2 package to add post-processing effects. This can be imported via the NeoFPS Hub.

The scriptable render pipelines each come with their own post processing systems. If you import the URP or HDRP unity packages, then you won't have to do anything else to gain access to the volumes or profile assets.

See Also

[Built-In Render Pipeline](#) [Universal Render Pipeline](#) [High-Definition Render Pipeline](#)

Built-In Render Pipeline

Overview

NeoFPS contains a number of shaders that have been created for the built-in (standard) render pipeline. These include various weapon optics shaders, effects, bullet trails and more.

The NeoFPS shaders were created using [Amplify Shader Editor](#).

If you own this asset then you should be able to load the shaders up and modify them to suit your needs.

Available Shaders

All of the NeoFPS shaders are currently located in the folder: `Assets\NeoFPS\Samples\Shared\Effects\Shaders`

They include the following:

- **NeoFPS_DissolveTrail...** are used for hitscan bullet trail. They start as a solid tracer line and then dissipate. The edge blend shaders fade off the tracer texture at the edges of the trail's line renderer.
- **NeoFPS_DistortionTrail** is another bullet trail which provides a shockwave style distortion.
- **NeoFPS_FireballSheet...** can be used for particle effects and pull flipbook style greyscale fireballs from the individual channels of a texture and use a gradient to add colour.
- **NeoFPS_FirstPersonWeapon_Stencil...** are used for the first person weapon models that you need to be stencilled out when looking through stencil scopes (such as the scope mounting geometry).
- **NeoFPS_HeatHaze** is an animated distortion which is used to add a rippling heat haze effect.
- **NeoFPS_PowerupGlow** overlays a colourful animated shimmer over a standard shader to highlight objects in the scene.
- **NeoFPS_Shockwave_Standard** provides a noisy distortion effect around the edges of a shockwave. The shockwave strength is influenced by vertex colour, making it useful for particle effects (start white and fade to black over lifetime).
- **NeoFPS_Water** is a simple water surface shader used for the swimming demo scenes.
- **NeoFPS_WaterCaustics** is a caustics volume shader that can be used to add a caustics lighting effect when underwater.
This shader is only compatible with deferred rendering.
- **Glow/NeoFPS_Glow...** shaders add a heat glow to first person weapons when overheating. The glow can be masked by position ranges, distance from a point (both object space) or using an alpha mask.
- **Highlight/NeoFPS_InteractiveHighlight** adds a shimmer effect to a standard shader to help highlight interactive objects in the scene.
- **Optics/NeoFPS_HoloSightReticule** is used for stencil based reticles that can only be seen through the glass of a holographic or red-dot sight. It also adds brightness and colour controls through the material.
- **Optics/NeoFPS_HoloSightStencil** is used to define the area you can see the reticle through. It is not otherwise visible.
- **Optics/NeoFPS_HoloSightStencilGlass** is used to define the area you can see the reticle through. It also uses a standard transparent base.
- **Optics/NeoFPS_LaserPointerBeam** defines an animated noisy like with a stronger central line for a laser look.
- **Optics/NeoFPS_RedDotReticule** is similar to the holosight reticule, but without the need for a reticule mask texture.
- **Optics/NeoFPS_RenderTextureScope** displays a render texture, adding a parallax scope ring and fading to an opaque, reflective glass based on how far off the scope object axis the camera is.
- **Optics/NeoFPS_StencilScopeInner...** are used for the parallax effect when looking through stencil scopes. Objects using this shader will only be visible through the lens of the scope.
- **Optics/NeoFPS_StencilScopeLens** defines the stencil that the scope inner surfaces can be seen through, and which the **NeoFPS_FirstPersonWeapon_Stencil...** cannot.

NB: The stencil scope shaders are currently not working properly with deferred rendering. A solution is being investigated

Post-Processing

The NeoFPS demos take advantage of Unity's Post-Processing Stack V2. This can be added to your project via the package manager, but it will also be added automatically (with a confirmation prompt) when applying the "Built-In" render pipeline setting in the **Unity Settings** page of the NeoFPS Hub.

Post processing allows for effects and corrections to be applied to the rendered image each frame after the camera has been rendered. This includes effects such as bloom on bright lights and objects, screen-space ambient occlusion which darkens the image in recessed areas and colour correction. Post processing is controlled by placing volumes in your scene that point at a post-processing profile. Volumes can be set to **Global**, which means that the effects will always be applied, or **Local** which means that the camera must enter the collider that's attached to the volume. This can be used for situations such as tinting the screen blue when under water in the swimming demo scene. Any volume gameobjects should be placed on the **PostProcessing layer**.

For more information on how to use post-processing, see [Unity Post-Processing and Full-Screen Effects](#).

In order for the post-processing effects to be shown on the camera, then the camera object must have a **PostProcessLayer** component attached. One quirk of the post processing stack package is that post processing layer component points at a data scriptable object that is generated when the package is installed. If you import an asset that uses post processing *before* you have the post-processing package installed then it will break that connection, and after installing post-processing Unity will constantly spew errors into the console until you close down and restart the editor. To get around this NeoFPS uses a component attached to its cameras called [PostProcessLayerFix](#). This adds the PostProcessLayer component on entering play mode instead. If you want then you can remove the [PostProcessLayerFix](#) component and swap it with a **PostProcessLayer** after installing the post-processing stack. That way you will be able to see the post processing results in the game view while editing your scene.

Extension Package

Since the URP and HDRP extension packages that come with NeoFPS replace a number of the demo materials with SRP compatible alternatives, a built-in piping package is included. This can be found in `Assets/NeoFPS/Extensions`, called **NeoFpsRenderPipeline_BIRP_v1.unitypackage**. Importing this package will restore the demo materials back to their original state.

See Also

[Modular Firearms][8]

[Amplify Shader Editor](#)

[Unity Post-Processing and Full-Screen Effects](#)

Universal Render Pipeline



Installation

In order to use the high-definition render pipeline with NeoFPS, the following requirements must be met:

- You must be using **Unity 2020.3 or higher**
- You must import the Universal RP package **version 10.9 or greater**

To import the universal render pipeline Unity package, you must use Unity's [Package Manager](#). The layout and wording of the package manager can vary between Unity versions, but the first thing required is to set the package manager to show the packages in the **Unity Registry**. This will show all the available Unity add-on packages that you can add to your project, as opposed to the "My Assets" or "In Project" options.

While looking at the unity registry, you should be able to find a package called **Universal RP (com.unity.render-pipelines.universal)**. Make sure to download **version 10.9** or greater and then import it into your project.

Once the URP package is imported, you will need to assign a Universal Render Pipeline Asset to the **Scriptable Render Pipeline Settings** in **Edit/Project Settings/Graphics**. An alternative to the above process is using the **Universal Render Pipeline Project Template** when first setting up your project via the Unity Hub. This will import the latest package and apply it in the graphics settings for you so that you can skip to the next step.

With the URP package imported and set up, you can now import the NeoFPS URP extension package. The best way to do this is via the **Unity Settings** page in the **NeoFPS Hub**. This will track the selected pipeline and notify you of any changes to the package with each NeoFPS update.

While the NeoFPS URP extension package comes with a number of replacement materials for the NeoFPS demos, this only covers the materials that cannot be auto-converted due to using custom NeoFPS shaders. To fix the rest of your materials you can convert them using the tools provided with URP.

For earlier versions of URP, this includes an option somewhere under **Edit/Render Pipeline/Universal Render Pipeline** to **Upgrade Project Materials to UniversalRP Materials**. This will convert all materials using built-in shaders to use their equivalent shader graph options. This option might have a slightly different name or location based on the URP version you have imported.

For the latest versions of URP, you will need to use the pipeline converter tool found under **Window/Rendering/Render Pipeline Converter**. Select the **Material Upgrade** section, then click on the **Initialize And Convert** button at the bottom to convert the materials.

You may also need to tweak and re-bake lighting in your scenes once the render pipeline has been changed.

Post-Processing

The universal render pipeline comes with an integrated post processing solution, however its volumes and profiles are *not* compatible with the Post Processing Stack V2 that Unity uses with the built-in render pipeline. It is advised to remove the Post Processing Stack via the package manager to prevent getting them mixed up.

To get post-processing properly working with the NeoFPS demos there are a few things you need to do:

- Modify the cameras for your characters and spawners, setting their **Volume Mask** setting to include **Default** and **PostProcessingVolumes**
- Replace any **PostProcessingVolume** or **Missing Script** components on the post processing volume objects in the scenes with the new **Volume** components.
- Set the volume components to **Local** for collider based volumes such as the underwater zones in the swimming demo.0.
- Assign one of the provided profile assets to the volume or create a new one as desired.

If you are moving from a built-in render pipeline project to URP then you might already have the old post processing stack package imported in your project. When this package is imported it adds the scripting define `UNITY_POST_PROCESSING_STACK_V2` to your project. NeoFPS uses this to compile code specifically for the post processing package. Unfortunately, removing the package does not remove the scripting define and can cause errors with code that requires the package to be present. You will need to do this manually. To do this, open the project's **Player Settings** and navigate to the bottom to find the **Script Compilation** section. Here you will find a setting called **Scripting Define Symbols**. This might be an array, or it might be a single string with multiple entries each separated using the semicolon (';') character. Delete the entry called `UNITY_POST_PROCESSING_STACK_V2` and the code that requires that will no longer be compiled.

Holographic and Red Dot Sights

Due to the changes in the way stencils work in the SRPs, the holographic sights have been modified in the NeoFPS SRP extension packages to use a new single shader, single mesh approach. The old shaders used a separate quad far in front of the weapon that had the reticule texture on it, and then used stencils to make sure that it was only visible through the glass of the holographic sight. The new system projects the reticule into the uv space of the glass itself, so does not need multiple objects or stencils.

You can find replacement versions of the demo optics in the `Assets/NeoFPS/Extensions/RenderPipelines/Prefabs/` folder. The replacement optics work for both HDRP and URP.

To manually switch a holographic or red dot sight, you can use the following steps:

1. Find the holographic sight object in the weapon or attachment hierarchy
2. Delete the reticule mesh from the hierarchy. This will be a quad that is parented to the sight object, and is referenced by the [HolographicSight](#) behaviour's **Reticule** property
3. Replace the [HolographicSight](#) behaviour with a [HolographicSightSRP](#) behaviour, copying over the settings. A simple way to do this is to switch the inspector to debug mode and then change the **Script** property to point at the SRP version of the script.
4. Find the glass mesh for the sight and replace its material with one that uses the [Optics_HolographicSight](#) or [Optics_RedDot](#) shader graph depending on the scope type. The NeoFPS URP and HDRP packages come with an example for each, with a material name matching the shader name.

Note: The built in shader version used separate shaders for the reticule materials instead of the glass materials to differentiate between holographic and red dot sights. This also means that extracting the NeoFPS URP or HDRP package will overwrite the existing weapon sight glass material with the new holographic sight version, changing the demo optics to all use a holographic sight reticule. To switch your red dot sights back to using a dot reticule, you can either swap the scope with one of the prefabs provided in the `RenderPipelines/Prefabs` folder, or change the scope's glass material to the correct type to fix this.

Render Texture Scopes

Similarly to the holographic and red dot scopes, the render texture scopes also require some manual steps in order to work with URP or HDRP. The way the shader properties are set has changed, and so the scopes require a different behaviour script to

control the fade, reticule and parallax. To switch your render texture scopes to URP/HDRP, replace the [RenderTextureScope](#) behaviour with a [RenderTextureScopeSRP](#) behaviour. A simple way to do this is to switch the inspector to debug mode and then change the **Script** property to point at the SRP version of the script. If you are using the scope lens material from the NeoFPS demos then extracting the NeoFPS URP or HDRP package should replace the material with one that uses the new SRP shader. If not then you will need to switch your material to use the [Optics_RenderTextureScope](#) shader graph.

Stencil Scopes

Stencil scopes are not currently working with the SRPs at this time. Currently, working with stencils in URP or HDRP requires the use of the [Render Objects Feature](#) within the renderer asset. To recreate the stencil scopes from NeoFPS this requires the addition of 4 Unity layers dedicated to the different parts of the scope, along with a complex setup.

Hitscan and Projectile Trails

Each of the bullet trail materials from the NeoFPS demos will have been replaced with alternatives that use the SRP shader graphs when you extracted the NeoFPS URP or HDRP package. If you have your own custom trail materials then you will need to replace their shaders with the shadergraph equivalents. Most of these can be found in the *RenderPipelines\ShaderGraphs\Shared* folder and will start with **BulletTrail_**. The distortion bullet trails are SRP specific, so you will find the URP version in *RenderPipelines\ShaderGraphs\URP\BulletTrail_Distortion.URP.shadergraph*.

See Also

[Modular Firearms](#)

[Amplify Shader Editor](#)

High-Definition Render Pipeline



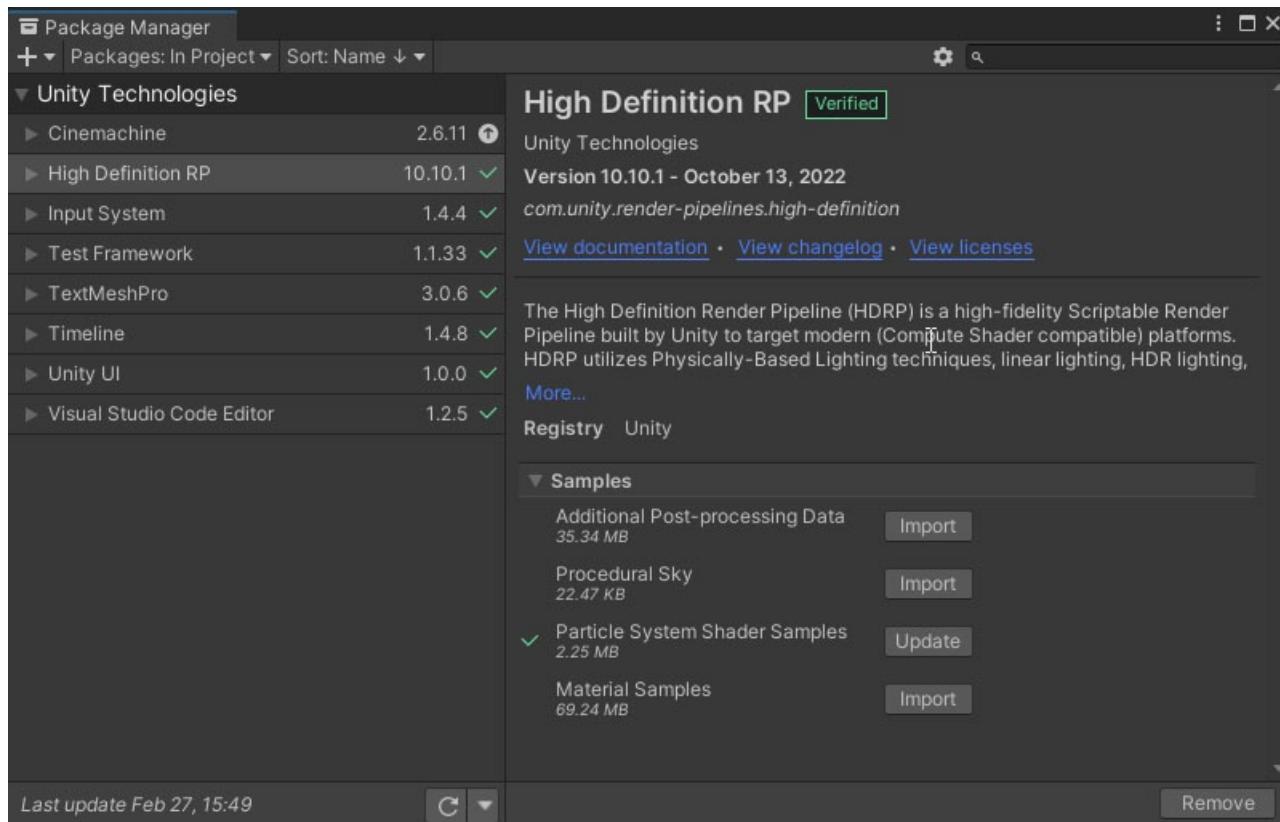
Installation

In order to use the high-definition render pipeline with NeoFPS, the following requirements must be met:

- You must be using **Unity 2020.3 or higher**
- You must import the High Definition RP package **version 10.9 or greater**
- You must import the **Particle System Shader Samples** that are provided with the HDRP package

To import the HD render pipeline Unity package, you must use Unity's [Package Manager](#). The layout and wording of the package manager can vary between Unity versions, but the first thing required is to set the package manager to show the packages in the **Unity Registry**. This will show all the available Unity add-on packages that you can add to your project, as opposed to the "My Assets" or "In Project" options.

While looking at the unity registry, you should be able to find a package called **High Definition RP (com.unity.render-pipelines.high-definition)**. Make sure to download **version 10.9** or greater and then import it into your project. Once the package is imported, stay on the High Definition RP entry in the package manager, and expand it **Samples** dropdown. There should be an option in here called **Particle System Shader Samples**. Add this to your project too, as it's required to convert the particle effects such as muzzle flare, explosions and smoke. The following is how the entry in the package manager looks in Unity 2020.3.



Once the HDRP package and particle sample shaders are imported, you can import the NeoFPS HDRP extension package. The best way to do this is via the **Unity Settings** page in the **NeoFPS Hub**. This will track the selected pipeline and notify you of any changes to the package with each NeoFPS update.

After importing the HDRP package, you will likely see a **HDRP Wizard** window pop up. This lets you check for any problems with your project setup and HDRP. Fix all the issues here before continuing. If required, NeoFPS comes with a number of settings assets to speed up the setup process. If you already have your project's HDRP settings set up then you can ignore the NeoFPS versions, as they are entirely default.

An alternative to the above process is using the **High-Definition RP Project Template** when first setting up your project via the Unity Hub. However, you will still need to import the sample particle shaders, since this won't be done by default.

While the NeoFPS HDRP extension package comes with a number of replacement materials for the NeoFPS demos, this only covers the materials that cannot be auto-converted due to using custom NeoFPS shaders. To fix the rest of your materials you can convert them using the tools provided with HDRP. This includes an option somewhere under **Edit/Render Pipelines to Upgrade Project Materials to High Definition Materials**. This will convert all materials using built-in shaders to use their equivalent shader graph options. This option might have a slightly different name or location based on the HDRP version you have imported.

Post-Processing

The high-definition render pipeline comes with an integrated post processing solution, however its volumes and profiles are *not* compatible with the Post Processing Stack V2 that Unity uses with the built-in render pipeline. It is advised to remove the Post Processing Stack via the package manager to prevent getting them mixed up.

To get post-processing properly working with the NeoFPS demos there are a few things you need to do:

- Modify the cameras for your characters and spawners, setting their **Volume Mask** setting to include **Default** and **PostProcessingVolumes**
- Replace any **PostProcessingVolume** or **Missing Script** components on the post processing volume objects in the scenes with the new **Volume** components.
- Set the volume components to **Local** for collider based volumes such as the underwater zones in the swimming demo0.
- Assign one of the provided profile assets to the volume or create a new one as desired.

If you are moving from a built-in render pipeline project to HDRP then you might already have the old post processing stack package imported in your project. When this package is imported it adds the scripting define `UNITY_POST_PROCESSING_STACK_V2` to your project. NeoFPS uses this to compile code specifically for the post processing package. Unfortunately, removing the package does not remove the scripting define and can cause errors with code that requires the package to be present. You will need to do this manually. To do this, open the project's **Player Settings** and navigate to the bottom to find the **Script Compilation** section. Here you will find a setting called **Scripting Define Symbols**. This might be an array, or it might be a single string with multiple entries each separated using the semicolon (';') character. Delete the entry called `UNITY_POST_PROCESSING_STACK_V2` and the code that requires that will no longer be compiled.

One of the major differences between HDRP and the other render pipelines is the physical camera system. This can give much more natural and realistic looking lighting, but it also means that it heavily relies on post-processing exposure settings to keep the lighting within a controlled visible range. You may find that the lighting in your levels looks very wrong when making the jump to HDRP from built-in. This is because the old light settings of strength and colour don't always map well to real world light values in the conversion process. In this situation you will need to inspect each of the lights in your scene and modify their settings to achieve the look you're after.

Holographic and Red Dot Sights

Due to the changes in the way stencils work in the SRPs, the holographic sights have been modified in the NeoFPS SRP extension packages to use a new single shader, single mesh approach. The old shaders used a separate quad far in front of the weapon that had the reticule texture on it, and then used stencils to make sure that it was only visible through the glass of the holographic sight. The new system projects the reticule into the uv space of the glass itself, so does not need multiple objects or stencils.

You can find replacement versions of the demo optics in the `Assets/NeoFPS/Extensions/RenderPipelines/Prefabs/` folder. The replacement optics work for both HDRP and URP.

To manually switch a holographic or red dot sight, you can use the following steps:

1. Find the holographic sight object in the weapon or attachment hierarchy
2. Delete the reticule mesh from the hierarchy. This will be a quad that is parented to the sight object, and is referenced by the [HolographicSight](#) behaviour's **Reticule** property
3. Replace the [HolographicSight](#) behaviour with a [HolographicSightSRP](#) behaviour, copying over the settings. A simple way to do this is to switch the inspector to debug mode and then change the **Script** property to point at the SRP version of the script.
4. Find the glass mesh for the sight and replace its material with one that uses the [Optics_HolographicSight](#) or [Optics_RedDot](#) shader graph depending on the scope type. The NeoFPS URP and HDRP packages come with an example for each, with a material name matching the shader name.

Note: The built in shader version used separate shaders for the reticule materials instead of the glass materials to differentiate between holographic and red dot sights. This also means that extracting the NeoFPS URP or HDRP package will overwrite the existing weapon sight glass material with the new holographic sight version, changing the demo optics to all use a holographic sight reticule. To switch your red dot sights back to using a dot reticule, you can either swap the scope with one of the prefabs provided in the `RenderPipelines/Prefabs` folder, or change the scope's glass material to the correct type to fix this.

Render Texture Scopes

Similarly to the holographic and red dot scopes, the render texture scopes also require some manual steps in order to work with URP or HDRP. The way the shader properties are set has changed, and so the scopes require a different behaviour script to control the fade, reticule and parallax. To switch your render texture scopes to URP/HDRP, replace the [RenderTextureScope](#) behaviour with a [RenderTextureScopeSRP](#) behaviour. A simple way to do this is to switch the inspector to debug mode and then change the **Script** property to point at the SRP version of the script. If you are using the scope lens material from the NeoFPS demos then extracting the NeoFPS URP or HDRP package should replace the material with one that uses the new SRP shader. If not then you will need to switch your material to use the [Optics_RenderTextureScope](#) shader graph.

Stencil Scopes

Stencil scopes are not currently working with the SRPs at this time. Currently, working with stencils in URP or HDRP requires the use of the **Render Objects Feature** within the renderer asset. To recreate the stencil scopes from NeoFPS this requires the addition of 4 Unity layers dedicated to the different parts of the scope, along with a complex setup.

Hitscan and Projectile Trails

Each of the bullet trail materials from the NeoFPS demos will have been replaced with alternatives that use the SRP shader graphs when you extracted the NeoFPS URP or HDRP package. If you have your own custom trail materials then you will need to replace their shaders with the shadergraph equivalents. Most of these can be found in the *RenderPipelines\ShaderGraphs\Shared* folder and will start with **BulletTrail_**. The distortion bullet trails are SRP specific, so you will find the HDRP version in *RenderPipelines\ShaderGraphs\HDRP\BulletTrail_Distortion_HDRP.shadergraph*.

See Also

[Modular Firearms](#)

Graphics Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Post-Processing Isn't Working (Built-In)

The Unity post-processing package (V2) has a long-standing bug that means installing the package when you already have objects that reference its components (eg an asset like NeoFPS) will break those connections and cause errors on importing the PPSv2 package. To get around this, NeoFPS uses the [PostProcessLayerFix](#) component on its cameras, which add the relevant post processing layer components at runtime.

To check why your post-processing isn't working, please check the following:

- You have the latest post processing package installed in the Unity package manager (com.unity.postprocessing)
- You have either the **PostProcessLayerFix** component, or you have added your own **PostProcessLayer** components to the character and spawner prefab camera objects (once both NeoFPS and the post processing package have been imported, you are fine to replace the PostProcessLayerFix components with PostProcessLayer components if you want)
- You have an object in your scene with a **PostProcessVolume** component on it. Most of the time, you will want one in the scene that has **Is Global** set to true, which will act as the default settings for that scene. You can then add others that have global set to false that override those defaults for specific regions.

I Want To Remove Unity Post Processing But I Get Script Errors

All of the NeoFPS code that references the post-processing API is wrapped in `#if UNITY_POST_PROCESSING_STACK_V2` blocks. This scripting define is added to your project automatically when you import the Unity post processing package. If you remove that package, you will also need to remove the scripting define to prevent errors. You can find this by going to your project's **Player Settings**. Under the **Other Settings** section, you will find a **Scripting Define Symbols** property. In older versions of Unity, this is a single string that is split with semicolon (`;`) characters. You only want to remove the "`UNITY_POST_PROCESSING_STACK_V2;`" from this string, and not the whole thing. In newer versions, this will be an array of strings. You can simply select and remove the relevant entry.

I Switched To URP/HDRP And Now My Materials Are Pink

The scriptable render pipelines each come with automated material conversion tools, however they must be run on your project or selected materials to have an effect. These can be found somewhere in the *Edit/Render Pipeline* menus, but their name and location within that varies across different versions of the render pipeline. The automated material upgrade can only upgrade materials that use the standard Unity shaders. Anything that uses one of the NeoFPS custom shaders must be manually switched to use the shader graph alternative that is included in the relevant extension package. The packages also come with direct replacements for the NeoFPS demo materials, however this will not affect any materials you've made yourself.

The high-definition render pipeline also does not come with replacement shaders for the Unity particle systems out of the box. You must install the **Particle System Shader Samples** via the **High Definition RP** entry in the package manager. This will allow you to convert the materials via the HDRP automatic upgrade tools.

I Switched to HDRP And Now Everything Is White

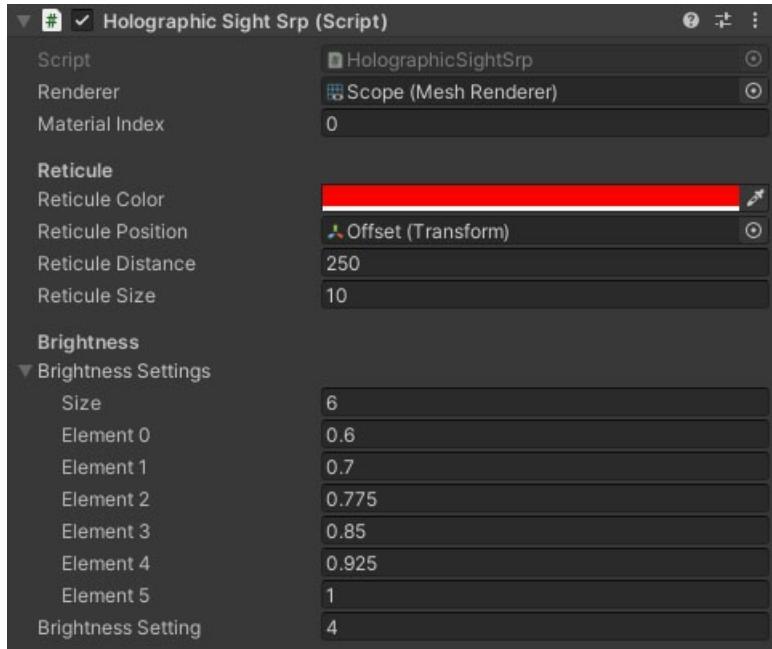
HDRP handles lighting very differently to the other render pipelines. You need to make sure that your post-processing is set up correctly in order for the camera exposure settings to work properly. For more information, see the post-processing section in [High Definition Render Pipeline](#). You might also find that having large areas of pure black in your scene will affect the exposure calculations, and cause the lit objects in those areas to blow out. This is especially noticeable if you use black for your sky colour.

HolographicSightSRP MonoBehaviour

Overview

The HolographicSightSRP behaviour is a modified version of the [HolographicSight](#) that works with the new URP and HDRP shaders. These don't use stencils like the built-in pipeline version, but project the reticule as part of the glass itself.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Renderer	Renderer	The renderer with the holographic sight or red dot sight material (from the shader graph) attached.
Material Index	Integer	The index of the material with the holographic sight or red dot sight shader.
Reticule Color	Color	The base colour of the reticule.
Reticule Position	Transform	A proxy transform that specifies where the reticule will appear in the object's space.
Reticule Distance	Float	A forwards (Z axis in the object's local space) offset for the reticule to project it in front of the sight.
Reticule Size	Float	The size of the reticule.
Brightness Settings	Float Array	A series of brightness values that can be cycled through with the Optics Brightness +/- inputs.
Brightness Setting	int	The index of the starting brightness setting from the above array.

See Also

[Modular Firearms](#)

[Scopes & Optics](#)

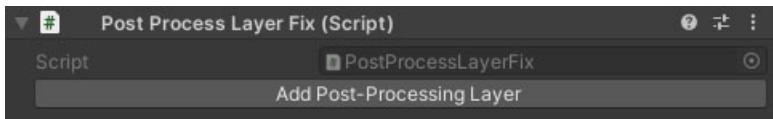
PostProcessLayerFix MonoBehaviour

Overview

The PostProcessLayerFix behaviour adds the **PostProcessLayer** component to any cameras on start at runtime.

The Unity [Post Processing Stack V2](#) has a number of bugs relating to how it serializes and deserializes references its shared resources. This makes it very difficult to use in an asset because on first import it can lead to bugs ranging from post-processing effects not being applied, to errors being spammed to the console. The fixes for this involve manually editing every camera object, and so this behaviour was added in an attempt to shift the requirement for this away from the end user.

Inspector



Properties

The PostProcessLayerFix behaviour has no properties exposed in the inspector.

Controls

The **Add Post-Processing Layer** button will add a post-processing layer component to the current object and then remove this behaviour. This is the same process it performs on start at runtime and can simplify your project if you know you are going to be using the built-in render pipeline.

See Also

[Post Processing Stack V2](#)

[PostProcessLayerSettings](#)

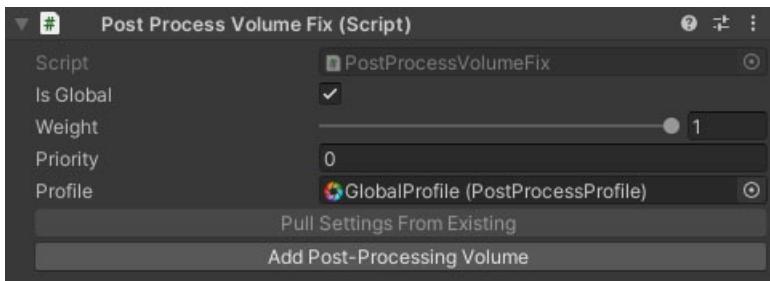
PostProcessVolumeFix MonoBehaviour

Overview

The PostProcessVolumeFix behaviour adds the **PostProcessVolume** component to the object it's attached to at runtime.

Due to URP and HDRP using a different GUID for their volume scripts, switching SRP away from built-in and removing the post-processing stack package will cause a missing script error. This behaviour gets around this by adding the volume behaviour at runtime, but only if the post processing stack package exists within the project.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Is Global	Boolean	Is the volume global or trigger zone based.
Weight	Float	The strength of this volume when combined with other volumes.
Priority	Float	The priority of the volume compared to others.
Profile	Post-Process Profile	The post processing profile asset the volume should use.

Controls

The **Pull Settings From Existing** button will be enabled if you add this behaviour to an object with an existing post-processing volume component. It will read the settings from the volume component and then remove it from the object so that it can be rebuilt at runtime in its original state.

The **Add Post-Processing Volume** button will add a post-processing volume component, assign the properties as set in this behaviour, and then remove this behaviour. This is the same process it performs on start at runtime and can simplify your project if you know you are going to be using the built-in render pipeline.

See Also

[Post Processing Stack V2](#)

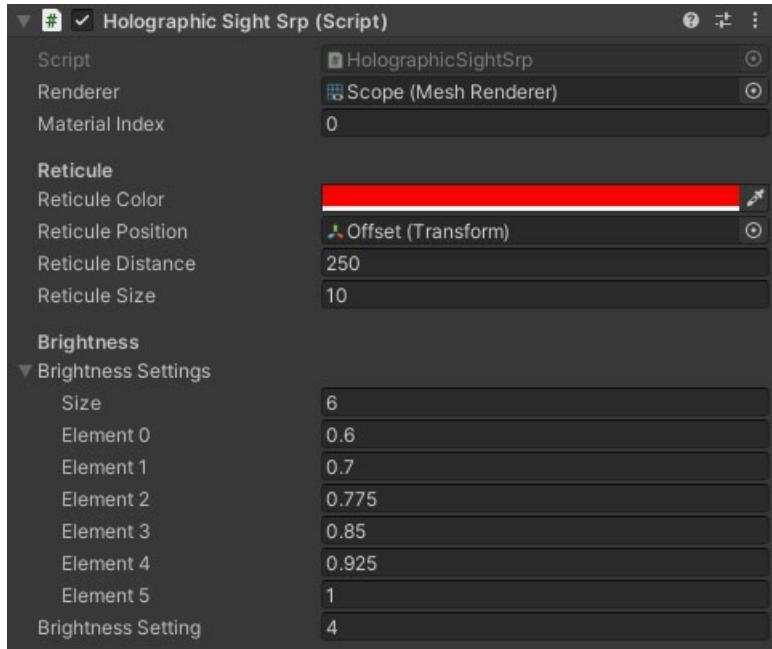
[PostProcessLayerSettings](#)

HolographicSightSRP MonoBehaviour

Overview

The HolographicSightSRP behaviour is a modified version of the [HolographicSight](#) that works with the new URP and HDRP shaders. These don't use stencils like the built-in pipeline version, but project the reticule as part of the glass itself.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Renderer	Renderer	The renderer with the holographic sight or red dot sight material (from the shader graph) attached.
Material Index	Integer	The index of the material with the holographic sight or red dot sight shader.
Reticule Color	Color	The base colour of the reticule.
Reticule Position	Transform	A proxy transform that specifies where the reticule will appear in the object's space.
Reticule Distance	Float	A forwards (Z axis in the object's local space) offset for the reticule to project it in front of the sight.
Reticule Size	Float	The size of the reticule.
Brightness Settings	Float Array	A series of brightness values that can be cycled through with the Optics Brightness +/- inputs.
Brightness Setting	int	The index of the starting brightness setting from the above array.

See Also

[Modular Firearms](#)

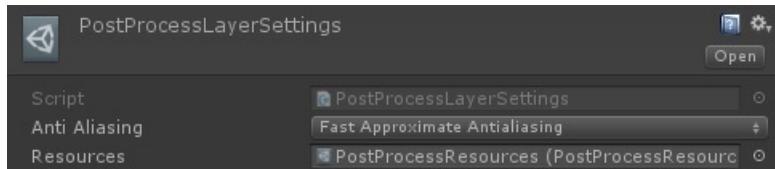
[Scopes & Optics](#)

PostProcessLayerSettings ScriptableObject

Overview

The PostProcessLayerSettings asset is used to store common [post processing](#) settings that are applied to any cameras on start.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Anti Aliasing	Dropdown	The anti-aliasing option to use when adding post processing to a camera.
Resources	PostProcessResources	The post processing resources to apply to the camera. Note: on first import this property will be null. The resources are created when the post processing package is imported and not accessible via the inspector. When you first access play mode, this component will be retrieved and applied, and the property will persist outside of play mode.

See Also

[Post Processing Stack V2](#)

[PostProcessLayerFix](#)

Samples

Overview

NeoFPS comes with a number of sample assets to demonstrate features and provide reference for your own implementations.

Demo Facility



The demo facility is an early stages demo that is intended to grow with NeoFPS. As more features are added it will be expanded into a small demonstration of a story driven FPS' game mechanics.

In this version it contains a firing range and a number of environment elements.

Feature Demos

Firing Range



The firing range demo features 4 ranges to experiment with the weapons in NeoFPS:

- A quick reaction target to test reflexes
- An occupied building with popup targets inside and out
- Popup targets over a large area to test accuracy at different ranges
- Multi-range sliding targets to test scopes and aimers

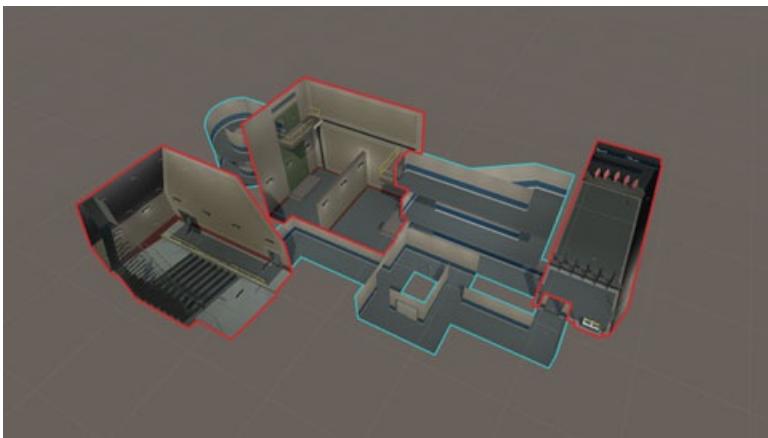
Parkour



The parkour demo uses a custom motion graph that is set up to show off the parkour features of NeoFPS, and provides an obstacle course to try it out in. The scene has obstacles to demo:

- Wall Running
- Crouch Slides
- Jumping Between Walls
- Climbing
- Vertical Wall Run

Multi-Scene Saves



The multi-scene saves demo uses multiple scenes to split a level up into parts that are streamed in/out on demand. The save game system is then used to store the contents of each sub-scene when unloaded and recreate them when the sub-scene is loaded back in.

First Person Body



The first person body demo is a simple scene to show off two of the first person body character types: full body, or torso and legs body. NeoFPS does not come with character animations, so there is a demo package that uses Kubold's Movement Animset Pro

and Rifle Animset Pro assets from the store, alongside the NeoFPS demo characters.

Jetpacks & Guided Missiles



The jetpacks and guided missiles puts you in control of a character with tribes inspired movement including jetpacks, skiing and aim-hover. You also have a shoulder mounted missile launcher which can lock onto up to 6 targets at once using the ability button.

Inventories



The inventories demo comprises 3 scenes with a different inventory type in each. Use the scene switch buttons to switch to the other scenes and compare.

Swimming



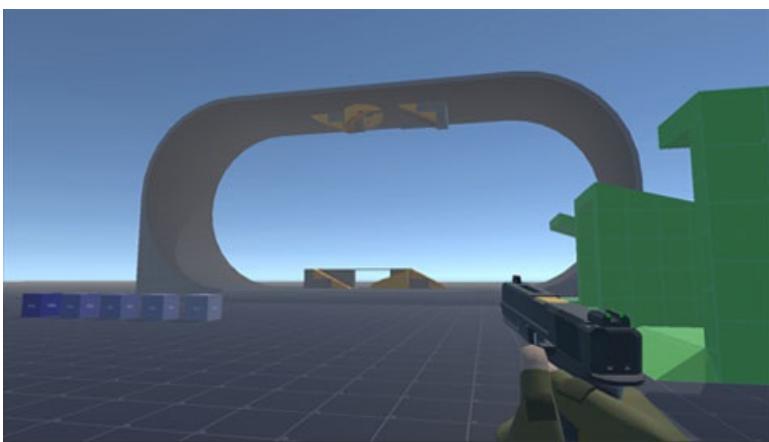
The swimming demo uses a custom motion graph that is set up to show off one implementation of swimming in NeoFPS. It has a number of water zones to demonstrate features like moving surface heights, flow, and changing depths.

Moving Platforms



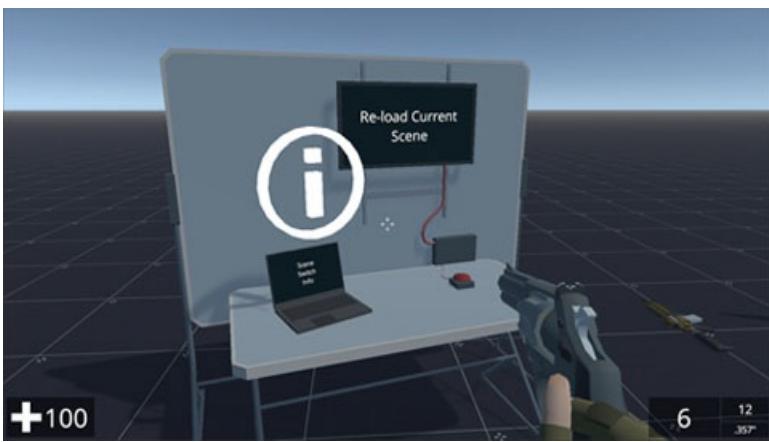
The moving platforms demo features a number of moving and rotating platforms, including complex rotations and waypoint based platforms.

NeoCharacterController



The NeoCharacterController demo shows some of the features of the NeoFPS character controller including pushing rigidbodies, variable gravity, and slope effects.

Persistence



The persistence demo shows how the NeoFPS save system can be used to persist data like character health and inventory across scene changes.

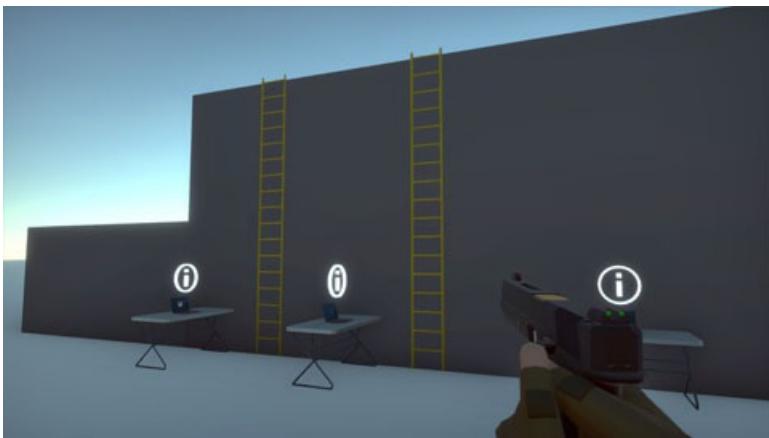
Doors



The doors demo features the main door types in NeoFPS. It is worth exploring the scene hierarchy and tweaking values in the door behaviours to see what happens. The demo also contains a number of locked doors to demonstrate how the locks system works. This includes:

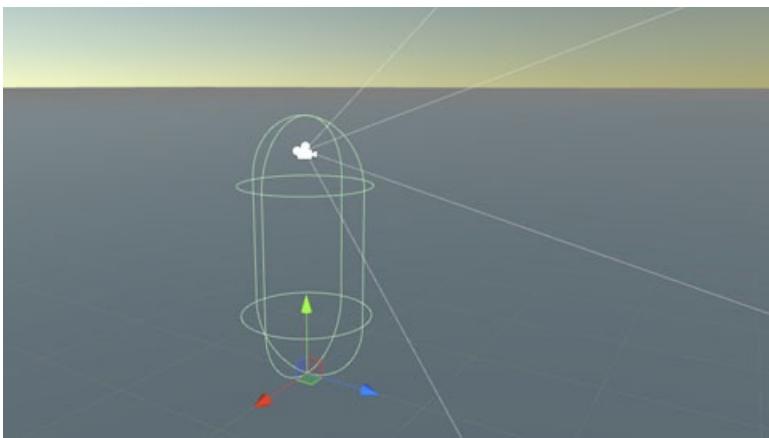
- Inventory keys
- Keypad locks
- Lockpicking
- Destructible locks

Ladders



The ladders demo features the main ladder types for comparison and experimentation. For consistency across a project, a number of the properties that affect ladder climbing are accessible through the relevant [motion graph states](#) instead of the ladders themselves.

Minimal Character



The minimal character demo uses a character that is stripped down to only the fundamental features of NeoFPS: movement and aim. It serves as an example of how you can pick and choose which features of NeoFPS to make use of and acts as a bare

minimum foundation to build on using either NeoFPS' solutions, third party solutions, or your own custom solutions.

Since the minimal character demo uses default Unity input instead of NeoFPS' bindable input, it does not handle the sample menus and UI system. Because of this, the minimal character demo will not be visible in the "Select Level" section of the game main menu, and is intended only as an example in the editor.

See Also

AI

Overview



The NeoFPS Demo Facility sample includes a number of simple AI turrets and cameras. This is a very limited AI system that was mainly created to test out using the [Modular Firearm](#) system to create non-player weapons. The turret is set up with similar modules to the sample assault rifle, with some modifications such as a larger magazine size.

Both the cameras and turrets work by tracking side to side. If the player enters the AI's detection range and view area, then the AI will enter a suspicious state. After a brief period, it will upgrade to an alerted state and the turret will start shooting at the player. If the player leaves AI's view, then it will seek randomly from point to point until it times out and returns to an idle state, or detects the player again and re-enters the alerted state.

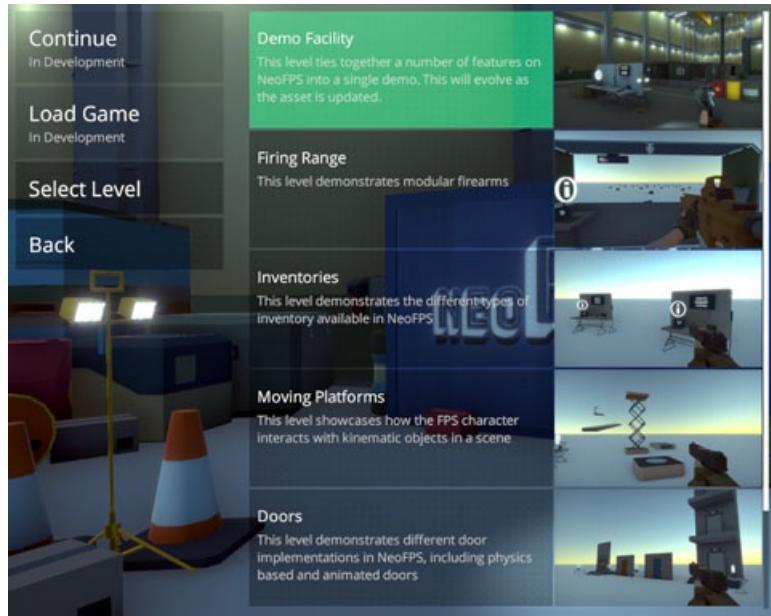
The turret and camera systems are not considered optimised for use in a complete game. For alternative AI solutions for NeoFPS you can check the [integrations](#) page on the NeoFPS website.

See Also

[Modular Firearms](#)

Sample UI

Overview



The sample UI is a tweaked implementation of the main [Unity UI](#) controls for more consistent menus that work with mouse navigation as well as keyboard or gamepad.

The original Unity UI source can be found [here](#).

The main reasoning is to create a set of menu controls with multiple elements that can work as one. For example, a slider menu item in the options menu could comprise of a slider, a + button, a - button and a text input field. These all work together to drive and display the correct value. When using a mouse, dragging on the slider or clicking the buttons will change the value and that will be shown in the input field. Clicking the input field will capture keyboard input, which will set the new value and position the slider to match. When using a gamepad, highlighting the menu entry and hitting submit will give it focus. Until the user cancels or hits submit again the direction controls will update the value (eg. right is increase, left is decrease) instead of navigating to the next item.

The sample UI also has basic layout functionality using menu panels and navigation bars.

Lastly, the sample UI also implements popups that can be displayed from any script. This is used with the [DemolInfoLaptop](#) to provide info popups when used.

Using the sample UI is not a requirement for any of the features of NeoFPS. It is only intended for use in the NeoFPS samples and needs extra work to be considered a complete solution for use in other games. It can be useful, however, in demonstrating how to block character input whilst accessing UI elements

UI Styles

The sample UI uses `UiStyle` scriptable object assets to define a consistent style for the UI elements. The examples include the following assets:

- **UiStyleMenus** asset is used for menu buttons and items
- **UiStyleMenuGroups** asset is used for the foldouts and dropdowns of menu items
- **UiStyleNavigation** asset is used for the navigation menus (SinglePlayer, Options, etc)

See Also

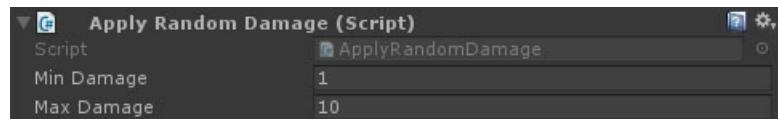
[Unity UI](#)

ApplyRandomDamage MonoBehaviour

Overview

The ApplyRandomDamage behaviour applies a random amount of damage between the min and max provided.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Min Damage	Float	The minimum damage to apply each time.
Max Damage	Float	The maximum damage to apply each time.

See Also

[Health and Damage](#)

CameraSeeker MonoBehaviour

Overview

The CameraSeeker behaviour is a very simple fixed position AI which scans back and forth looking for a player, and enters an alert mode when it finds them.

Inspector



Properties

Name	Type	Description
Look Transform	Transform	The transform used to detect look angles (should be the last in the chain, eg. camera body).

NAME	TYPE	DESCRIPTION
Horizontal Servo Transform	Transform	The transform used to rotate on the horizontal axis.
Vertical Servo Transform	Transform	The transform used to rotate on the vertical axis.
Rotation Speed Idle	Float	The camera rotation speed when idle (degrees per second).
Pause Idle	Float	The length of the pause at each extreme of rotation while idle.
Idle Rotations	Vector2	The rotation points for the camera when idling.
Rotation Speed Hostile	Float	The camera rotation speed when hostile (degrees per second).
Pause Hostile	Float	The length of the pause at each extreme of rotation when hostile (degrees per second).
Suspicious Time	Float	The duration the camera will be suspicious before engaging.
Hunting Time	Float	The duration the camera will stay in hunting mode before going idle.
Detection Range	Float	The maximum range of the camera in meters.
Min Angles	Vector2	The minimum angles the camera can reach (negative y is down).
Max Angles	Vector2	The maximum angles the camera can reach (negative y is down).
Camera Light	Light	The halo glow for the camera.
Colour Idle	Color	The halo colour for the idle state.
Colour Suspicious	Color	The halo colour for the suspicious state.
Colour Engaged	Color	The halo colour for the engaged state.
On Idle	UnityEvent	An event invoked when the seeker enters the idle state.
On Suspicious	UnityEvent	An event invoked when the seeker enters the suspicious state.
On Engaged	UnityEvent	An event invoked when the seeker enters the engaged state.
On Hunting	UnityEvent	An event invoked when the seeker enters the hunting state.
On Killed	UnityEvent	An event invoked when the seeker is killed.
Starting Health	Float	The health of the seeker.

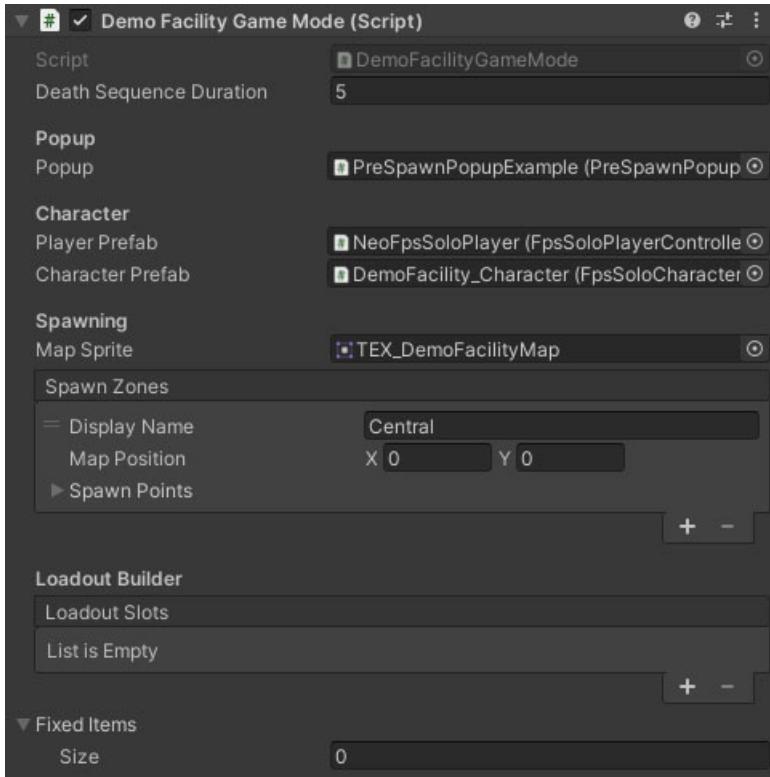
See Also

DemoFacilityGameMode MonoBehaviour

Overview

The DemoFacilityTarget is a [customised game mode](#) that lets you choose a spawn point and weapon loadout each spawn as an example for how you can set that up yourself.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Death Sequence Duration	Float	How long after dying does the game react (gives time for visual feedback).
Popup	PreSpawnPopup	The pre-spawn popup prefab to use.
Player Prefab	FpsSoloPlayerController	The player prefab to instantiate if none exists.
Character Prefab	FpsSoloCharacter	The character prefab to use.

See Also

[Game Modes](#)

DemoFacilityTarget MonoBehaviour

Overview

The DemoFacilityTarget is a simple target that sends damage to a tracker for displaying to the player.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Tracker	DemoDamageTracker	The target damage tracker.
Target Index	Int	The target index in the tracker.

See Also

[DemoFacilityTargetDamageTracker](#)

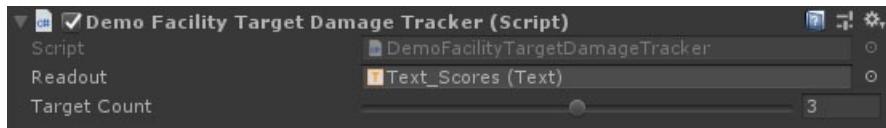
[Health and Damage](#)

DemoFacilityTargetDamageTracker MonoBehaviour

Overview

The DemoFacilityTargetDamageTracker behaviour tracks damage to a number of [DemoFacilityTarget](#) objects and prints it to a [Unity UI] readout.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Readout	Text	The text readout for target damage.
Target Count	Int	The number of targets to track.

See Also

[DemoFacilityTarget](#)

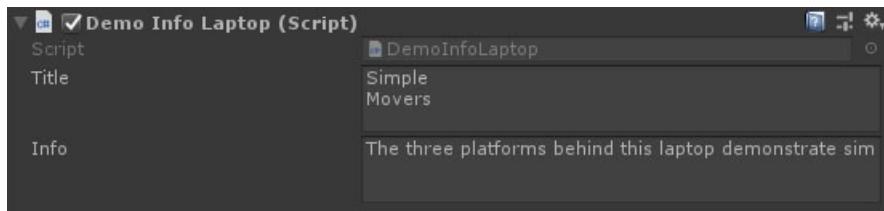
[Unity UI](#)

DemoInfoLaptop MonoBehaviour

Overview

The DemoInfoLaptop behaviour is an info point that can be interacted with and displays a text popup using the [demo UI](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interacted with.

See Also

[Sample UI](#)

[Interactive Objects](#)

[Unity UI](#)

DoorsDemoElevatorReadout MonoBehaviour

Overview

The DoorsDemoElevatorReadout behaviour subscribes to the events on an [ElevatorController](#), and prints it to a [Unity UI](#).

Inspector



Properties

The DoorsDemoElevatorReadout behaviour has no properties exposed in the inspector.

See Also

[ElevatorController](#)

[Unity UI](#)

FiringRangeMovingTarget MonoBehaviour

Overview

The FiringRangeMovingTarget behaviour is a variant of the [FiringRangeTarget](#) that also moves.

Inspector



Properties

The FiringRangeMovingTarget inherits from the [FiringRangeTarget](#). Check the [reference](#) for information on its properties.

NAME	TYPE	DESCRIPTION
Move Offset	Vector3	The offset to move to.
Move Duration	Float	The time taken to reach the offset.

See Also

[Health and Damage](#)

[FiringRangeTarget](#)

[FiringRangeSequencer](#)

FiringRangeReadout MonoBehaviour

Overview

The FiringRangeReadout behaviour receives hit and miss events from the targets in the firing range demo and prints them to a Unity UI.

Inspector



Properties

The FiringRangeReadout behaviour has no properties exposed in the inspector.

See Also

[FiringRangeSequencer](#)

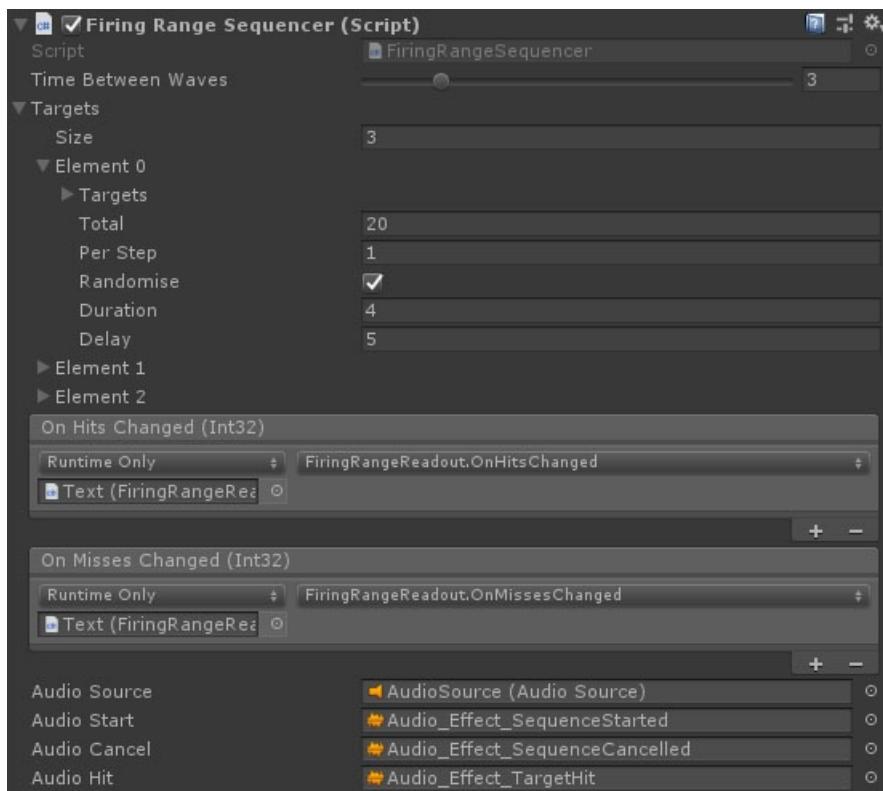
[Unity UI](#)

FiringRangeSequencer MonoBehaviour

Overview

The FiringRangeSequencer behaviour manages all the targets in a demo firing range. It works in waves, controlling the number of targets that pop up at once along with the spacing.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Between Waves	Float	The pause in between each wave.
Targets	Target Group Array	The targets for each wave.
On Hits Changed	UnityEvent	An event that is invoked when a target is hit.
On Misses Changed	UnityEvent	An event that is invoked when a target is missed.
Audio Transform	Transform	The transform to attach the audio rotator to.
Audio Source	<audiosource< td=""><td>The audio source for playing one shot firing range audio clips.</td></audiosource<>	The audio source for playing one shot firing range audio clips.
Audio Start	AudioClip	The audio clip to play when the sequence starts.
Audio Cancel	AudioClip	The audio clip to play when the sequence is cancelled.
Audio Hit	AudioClip	The audio clip to play when a target is hit.

Target Group

NAME	TYPE	DESCRIPTION
Targets	FiringRangeTarget Array	The targets for this wave.
Total	Int	The total number of targets to pop up this wave.
Per Step	Int	The number of targets to pop up for each step of the wave.
Randomise	Boolean	Should the targets be chosen at random or in sequence.
Duration	Float	The duration a target should stay up.
Delay	Float	The delay between steps.

See Also

[FiringRangeTarget](#)

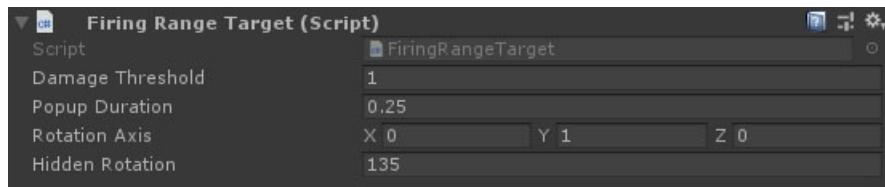
[FiringRangeMovingTarget](#)

FiringRangeTarget MonoBehaviour

Overview

The FiringRangeTarget behaviour is a pop-up target that sends hit or miss events to the [sequencer](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Damage Threshold	Float	The damage threshold for the target to drop and register as a hit.
Popup Duration	Float	The duration the target will be visible. If the target is not hit in this time it registers as a miss.
Rotation Axis	Vector3	The axis to rotate the target around when it pops up.
Hidden Rotation	Float	The rotation of the target around the specified axis when it is fully hidden.

See Also

[Health and Damage](#)

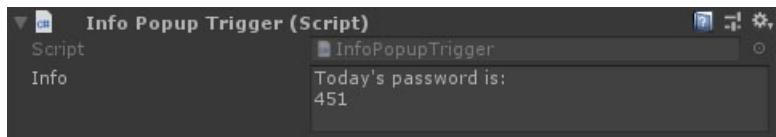
[FiringRangeSequencer](#)

InfoPopupTrigger MonoBehaviour

Overview

The InfoPopupTrigger behaviour is an info point that can be interacted with and displays a text popup using the [demo UI](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interacted with.

See Also

[Sample UI](#)

[Interactive Objects](#)

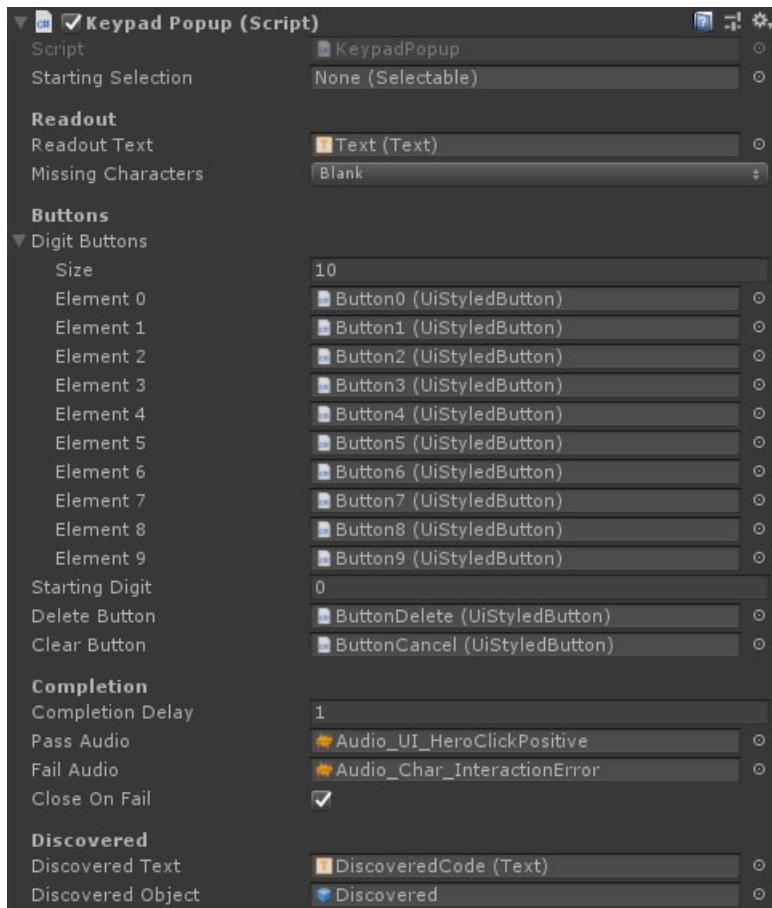
[Unity UI](#)

KeypadPopup MonoBehaviour

Overview

The KeypadPopup behaviour is an info point that can be interacted with and displays a text popup using the [demo UI](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interacted with.

See Also

[Sample UI](#)

[Interactive Objects](#)

[Unity UI](#)

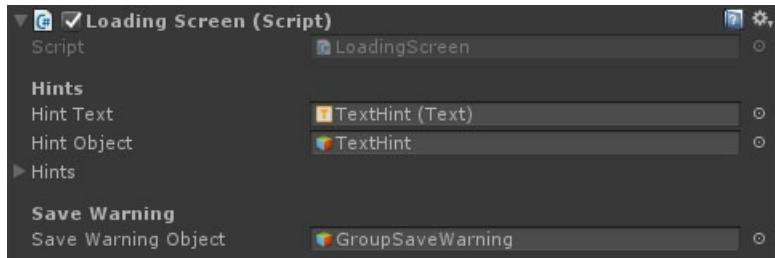
LoadingScreen MonoBehaviour

Overview

The LoadingScreen behaviour is a very simple example implementation of a loading screen with gameplay hints and instructions.

The first time the loading screen is shown, it will display a warning about exiting while performing a save. After this, a random hint will be displayed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Hint Text	Text	The text output for the gameplay hints.
Hint Object	GameObject	The object that contains the hints UI. The first time the screen is shown, this object will be hidden.
Hints	String Array	An array of hints to display, chosen at random.
Save Warning Object	GameObject	The object that contains a warning about the "saving" icon. This will only be shown the first time.

See Also

[Unity UI](#)

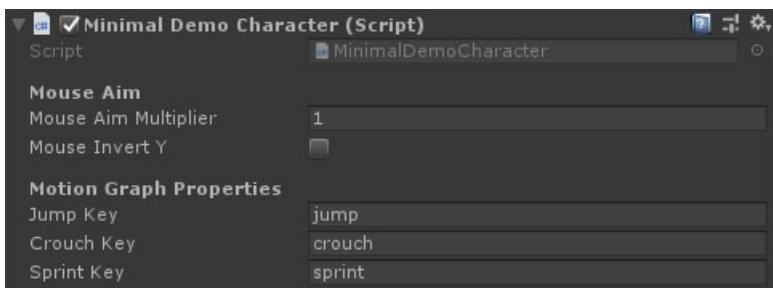
MinimalDemoCharacter MonoBehaviour

Overview

The MinimalDemoCharacter behaviour ties together the [NeoCharacterController](#), [MotionController](#) and [MouseAndGamepadAimController](#) to create a simplified playable character with all the movement and aim features of the main NeoFPS demo character.

The minimal character uses Unity's input system to control the character instead of NeoFPS' bindable input system, while all of NeoFPS' features outside movement and aiming are removed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Mouse Aim Multiplier	Float	The mouse aim sensitivity.
Mouse Invert Y	Boolean	Is the mouse vertical aim rotation flipped.
Jump Key	String	The key to the jump trigger property in the character motion graph.
Crouch Key	String	The key to the crouch switch property in the character motion graph.
Sprint Key	String	The key to the sprint switch property in the character motion graph.

See Also

[NeoCharacterController](#)

[MotionController](#)

[MouseAndGamepadAimController](#)

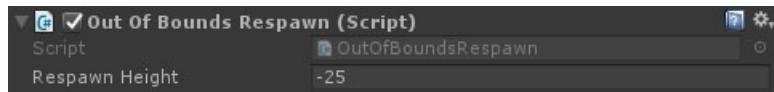
[Samples](#)

OutOfBoundsRespawn MonoBehaviour

Overview

The OutOfBoundsRespawn behaviour simply watches the player character, and if their height is lower than a set value, it respawns them.

Inspector



Properties

NAME	TITLE	DESCRIPTION
Respawn Height	Float	If the player character's position is lower than this height then it will trigger a respawn.

TurretSeeker MonoBehaviour

Overview

The TurretSeeker behaviour is a very simple fixed position AI which scans back and forth looking for a player, similarly to the [CameraSeeker](#). Once it spots a player it will enter a suspicious mode, and then start shooting after a brief pause.

The TurretSeeker uses the [Modular Firearm](#) system to model gun behaviour using the same modules as the player's first person weapons.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Look Transform	Transform	The transform used to detect look angles (should be the last in the chain, eg. camera body).
Horizontal Servo Transform	Transform	The transform used to rotate on the horizontal axis.
Vertical Servo Transform	Transform	The transform used to rotate on the vertical axis.
Rotation Speed Idle	Float	The camera rotation speed when idle (degrees per second).
Pause Idle	Float	The length of the pause at each extreme of rotation while idle.
Idle Rotations	Vector2	The rotation points for the camera when idling.
Rotation Speed Hostile	Float	The camera rotation speed when hostile (degrees per second).
Pause Hostile	Float	The length of the pause at each extreme of rotation when hostile (degrees per second).
Suspicious Time	Float	The duration the camera will be suspicious before engaging.
Hunting Time	Float	The duration the camera will stay in hunting mode before going idle.
Detection Range	Float	The maximum range of the camera in meters.
Min Angles	Vector2	The minimum angles the camera can reach (negative y is down).
Max Angles	Vector2	The maximum angles the camera can reach (negative y is down).
Camera Light	Light	The halo glow for the camera.
Colour Idle	Color	The halo colour for the idle state.
Colour Suspicious	Color	The halo colour for the suspicious state.
Colour Engaged	Color	The halo colour for the engaged state.
On Idle	UnityEvent	An event invoked when the seeker enters the idle state.
On Suspicious	UnityEvent	An event invoked when the seeker enters the suspicious state.
On Engaged	UnityEvent	An event invoked when the seeker enters the engaged state.
On Hunting	UnityEvent	An event invoked when the seeker enters the hunting state.
On Killed	UnityEvent	An event invoked when the seeker is killed.
Starting Health	Float	The health of the turret.

See Also

[Modular Firearms](#)

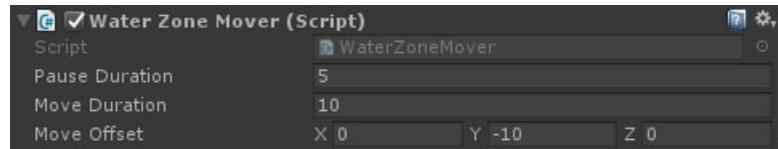
[CameraSeeker](#)

WaterZoneMover MonoBehaviour

Overview

The WaterZoneMover behaviour is used in the swimming demo to raise and lower one of the water zones.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Pause Duration	Float	The pause between moving up and down.
Move Duration	Float	The time it takes to go from the original position to the offset position or back again.
Move Offset	Vector3	The offset from the original position to move in local space.

Utilities

Animation Behaviours

Animation graph behaviours are attached to states in the unity animator controller graphs. NeoFPS uses 2 behaviours that allow animations to sync the active state of objects to specific clips, and to swap animated objects with dynamic physics objects. These are used in the modular firearms during reload animations but can have a number of uses besides.

For more information, see the [AnimStateObjectActivator](#), [AnimStateObjectSwapper](#), and [AnimStateObjectSwapperTarget](#) references.

Pooling

Pooling is an important optimisation in any game. It means pre-instantiating a pool of objects before they are needed that can be spawned and recycled at any time. Instantiation and the garbage collection resulting from object destruction are relatively expensive operations and it is best practice not to instantiate or destroy objects during gameplay. The NeoFPS pooling utilities can specify pool sizes for individual prefabs that will be instantiated at startup. These can then be retrieved and returned as needed.

The [PoolManager](#) specifies the pooled objects to be made available across all scenes, along with a number of default settings, while the [ScenePoolHandler](#) is responsible for preallocating and accessing pools within the active scene. Retrieve a [PooledObject](#) with the following function:

```
public T GetPooledObject<T> (PooledObject prototype) where T : MonoBehaviour
```

The [PooledObject](#) can be returned to the pool with the following function:

```
void PooledObject.ReturnToPool ()
```

If a prototype [PooledObject](#) is not managed by the [PoolManager](#) already then a new pool of the default size will be instantiated. This is an expensive operation so it is best to try and anticipate the required objects in advance.

Object Lifecycle

NeoFPS uses a number of simple utilities that manage the active state and lifecycle of an object. See the [TimedDisabler](#), [ObjectLifecycleManager](#) and [ObjectLifecycleMgrDictionary](#) for more information. You can also use a [TemporaryPooledObject](#) which has the lifecycle built in for objects that should only be visible for a brief period.

Teleporters

There are 2 teleporter setups provided in NeoFPS:

- The [Teleporter](#) behaviour is used in the NeoFPS demos to quickly get around the scenes. It is used for physical teleporter objects that you step into and interact with to teleport to another location. The target location needs to be another teleporter, and that teleporter will be disabled for a brief period on teleporting into it.
- The [TeleportZone1Way](#) behaviour is used for simpler navigation around scenes. On stepping into the teleport zone a timer will start and an *onEntered* [UnityEvent](#) will fire that you can use to trigger particle and audio effects. Once the timer completes, the character is teleported to the target transform and an *onTeleported* event is fired. If the character leaves the zone before the timer completes, then the timer resets and an *onCancelled* event is fired.

Slow-Motion System

NeoFPS implements a simple slow-motion system that allows you to create slow-mo effects in response to various triggers. The [SlowMoSystem](#) behaviour can be added to the player character to create an adrenaline or focus style system as used in games like F.E.A.R. This system has a slow-mo charge which is drained while in slow motion and the recharges when back in normal time. You can also zero the drain or recharge rates for more control over when the effect starts or stops.

To trigger the slow-mo effect, NeoFPS implements a number of sample mechanisms. These include:

- The [SetTimeScale Motion Graph Behaviour](#) triggers slow-mo when entering a specific motion state.
- The [SlowMoZone](#) behaviour is used to define a trigger zone. When the player character enters, time is slowed. When they exit, time speed is restored.
- The [TimedSlowMoZone](#) behaviour is similar, but the slow-mo effect only lasts for a short period (based on the drain rate).

The slow-mo system works by modifying the following script property: `NeoFpsTimeScale.timeScale`. This is similar to Unity's `Time.timeScale`, but fires events that allow other systems to monitor the time scale. This allows things like [audio sources][unity-audio] to react by modifying their pitch (and therefore speed) via the [AudioTimeScalePitchBend](#) behaviour. If you want to create extremely simple slow-mo effects (such as when liberating an outpost in far cry), you can modify this property directly.

See Also

[Unity AnimatorController](#)

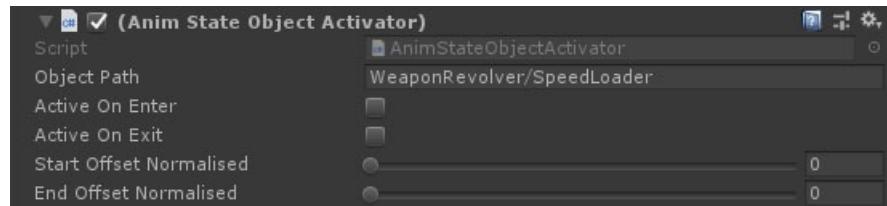
[Unity State Machine Behaviours](#)

AnimStateObjectActivator StateMachineBehaviour

Overview

This is used to control when gameobjects are active from within an animation graph. For example, controlling a shotgun shell's visibility during a reload animation.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Object Path	String	The path to the relevant object in the object hierarchy the animator is attached to.
Active On Enter	Boolean	Should the object be active when the animation state is entered.
Active On Exit	Boolean	Should the object be active when the animation state is exited.
Start Offset Normalised	Float	The normalised offset (0 to 1) from entry of the state for the object state to be changed (0 is the start, 1 is the end).
End Offset Normalised	Float	The normalised offset (0 to 1) from the end of the state for the object state to be changed (0 is the end, 1 is the start).

See Also

[Unity Animator](#)

[Unity AnimatorController](#)

AnimStateObjectSwapper StateMachineBehaviour

Overview

The AnimStateObjectSwapper is a [StateMachineBehaviour](#). It is attached to animator controller graph states and will trigger an [AnimatedObjectSwapper](#) to swap an object out.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Object Path	String	The path to the object containing the AnimatedObjectSwapper relative to the object hierarchy the animator is attached to.
Normalised Swap Time	Float	The normalised offset (0 to 1) from entry of the state for the object swap to occur. (0 is the start, 1 is the end).

See Also

[AnimatedObjectSwapper](#)

[Unity Animator](#)

[Unity AnimatorController](#)

[Unity StateMachineBehaviour](#)

AbilityBasedSlowMoSystem MonoBehaviour

Overview

The AbilityBasedSlowMoSystem behaviour swaps out an object with a pooled physics object. It is paired with an [AnimStateObjectSwapper](#) state machine behaviour that triggers the swap.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Transform	Transform	The object to swap.
Pooled Object	PooledObject	The pooled object to swap the target object with.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

See Also

[AnimStateObjectSwapper](#)

[PooledObject](#)

[Unity Animator](#)

[Unity AnimatorController](#)

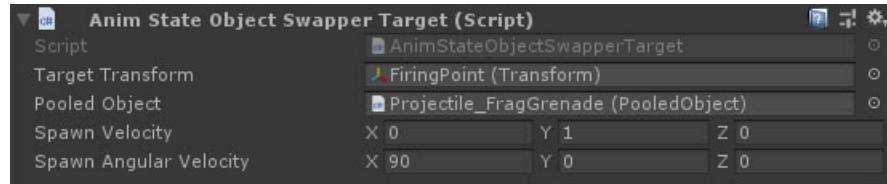
[Unity Rigidbody](#)

AnimStateObjectSwapperTarget MonoBehaviour

Overview

The AnimStateObjectSwapperTarget behaviour swaps out an object with a pooled physics object. It is paired with an [AnimStateObjectSwapper](#) state machine behaviour that triggers the swap.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Transform	Transform	The object to swap.
Pooled Object	PooledObject	The pooled object to swap the target object with.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

See Also

[AnimStateObjectSwapper](#)

[PooledObject](#)

[Unity Animator](#)

[Unity AnimatorController](#)

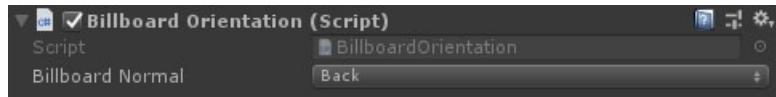
[Unity Rigidbody](#)

BillboardOrientation MonoBehaviour

Overview

The BillboardOrientation behaviour is used to orient an object towards the currently active camera.

Inspector



Properties

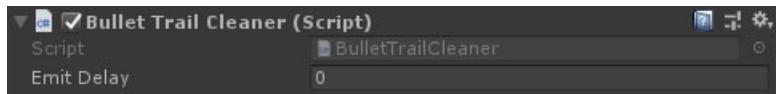
NAME	TYPE	DESCRIPTION
Billboard Normal	Dropdown	The billboard surface normal direction (this will be turned towards the camera).

BulletTrailCleaner MonoBehaviour

Overview

The BulletTrailCleaner behaviour is attached to pooled projectiles with a trail renderer and clears the trail when the projectile is disabled so that it can be recycled without old trail data bleeding into new.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Emit Delay	Float	The delay before enabling the trail render.

See Also

[Interactive Objects](#)

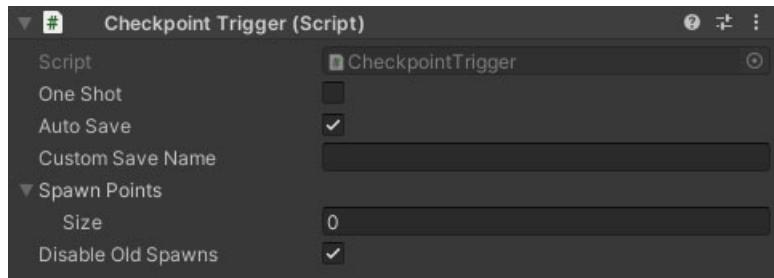
[Unity Events](#)

CheckpointTrigger MonoBehaviour

Overview

The CheckpointTrigger behaviour is used to signify progression checkpoints in your levels. Crossing a checkpoint can enable the connected spawn points and disable the old spawns, as well as auto-saving the game.

Inspector



Properties

NAME	TYPE	DESCRIPTION
One Shot	Boolean	Should the checkpoint trigger fire multiple times (eg allow back-tracking).
Auto Save	Boolean	Save the game progress using the auto save feature.
Custom Save Name	String	A custom name to use on the load screen for this autosave. If this is blank then the scene's display name will be used.
Spawn Points	SpawnPoint Array	A list of spawn points to enable at this checkpoint.
Disable Old Spawns	Boolean	Should previous spawn points be disabled (guaranteeing that the player spawns here).

See Also

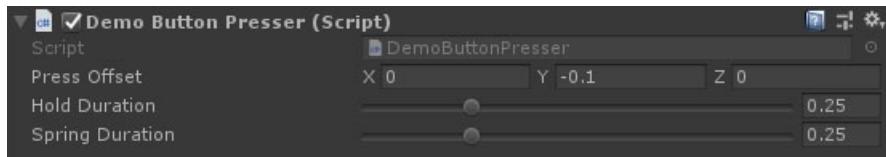
[Save Games](#)

DemoButtonPresser MonoBehaviour

Overview

The DemoButtonPresser behaviour is used to procedurally animate a simple button. It has a public `Press()` function that can be attached to [unity events](#), for example on the [InteractiveObject](#) **OnUsed** event.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Press Offset	Vector3	The button position offset when pressed.
Hold Duration	Float	The duration to hold the button down.
Spring Duration	Float	The duration to spring back to the original position.

See Also

[Interactive Objects](#)

[Unity Events](#)

GravityModifier MonoBehaviour

Overview

The GravityModifier behaviour can be used to reorient the player character and world gravity. It can be triggered via [unity events](#) or the API.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Gravity	Vector	The gravity vector to apply to the player character (and optionally Unity physics).
Set Physics Gravity	Float	Should this behaviour also set the Unity physics vector on top of the character gravity vector.

See Also

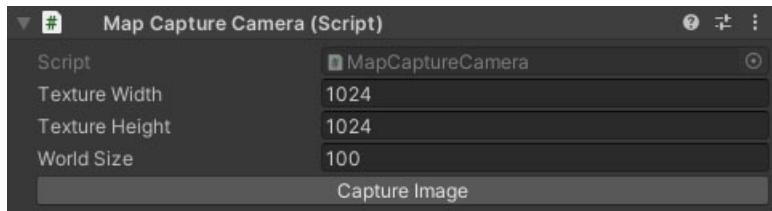
[The NeoCharacterController](#)

MapCaptureCamera MonoBehaviour

Overview

The MapCaptureCamera behaviour is a simple helper tool that can be used to capture a screenshot of your scene and output it to disk as an image. The image will be placed in your project folder. This is useful when creating map images for spawn selection or map selection.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Texture Width	Integer	The width of the output texture in pixels.
Texture Height	Integer	The height of the output texture in pixels.
World Size	Float	The size in meters of the capture volume (along the height axis of the texture).

See Also

[Inventory](#)

[Unity Rigidbody](#)

NeoFpsSceneSwitcher MonoBehaviour

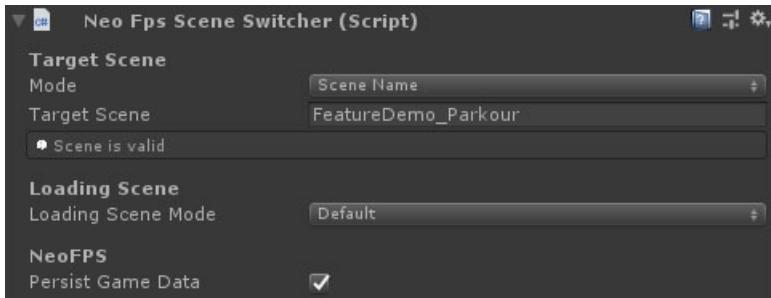
Overview

The NeoFpsSceneSwitcher behaviour is used to trigger loading a new scene.

The NeoFPS scene management system loads a scene with the following steps:

1. Load the loading screen scene which displays a loading screen over the game view.
2. If the switcher is set to persist data, then ask the current [game mode](#) to save the relevant game data to a memory buffer.
3. Unload the current scene.
4. Load the new scene asynchronously in the background.
5. If data was persisted, ask the new scene's [game mode](#) to load it.
6. Unload the loading screen scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Mode	Dropdown	How the scene to load is specified. The options are Scene Name , Scene Index , Next Scene Index . If the last option is used then the next scene listed in the build options will be loaded.
Target Scene	String/Integer	The name or index of the scene to load, matching the entry in the project build options.
Loading Scene Mode	Dropdown	How to choose the loading screen scene to load. The options are Default (specified in the NeoSceneManager options), Scene Name and Scene Index .
Loading Scene	String/Integer	The name or index of the loading screen scene to load, matching the entry in the project build options.
Persist Game Data	Boolean	Should game data (eg, character health and inventory) be persisted to the new scene?

See Also

[Runtime Objects](#)

[NeoSceneManager](#)

[Game Modes](#)

NeoMainMenuSceneSwitcher MonoBehaviour

Overview

The NeoMainMenuSceneSwitcher behaviour is used to trigger exiting the game and loading into the main menu scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Menu Scene Mode	Dropdown	How the menu scene to load is specified. The options are Default , Scene Name , Scene Index . The Default setting will load the menu scene specified in the NeoSceneManager .
Menu Scene	String/Integer	The name or index of the menu scene to load, matching the entry in the project build options. This property will only appear if the Menu Scene Mode is not set to Default

See Also

[Runtime Objects](#)

[NeoSceneManager](#)

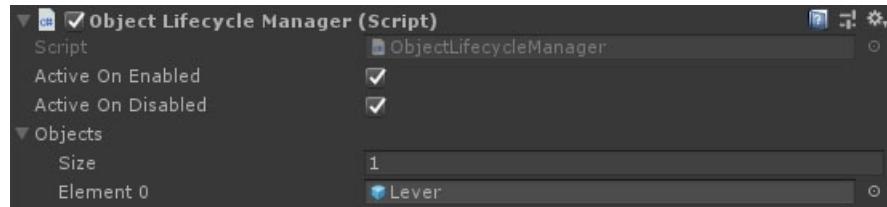
[Game Modes](#)

ObjectLifecycleManager MonoBehaviour

Overview

The ObjectLifecycleManager is used to manage the active state of attached objects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Active On Enabled	Boolean	Should all the attached objects be enabled when this is.
Active On Disabled	Boolean	Should all the attached objects be disabled when this is.
Objects	GameObject Array	The objects being managed.

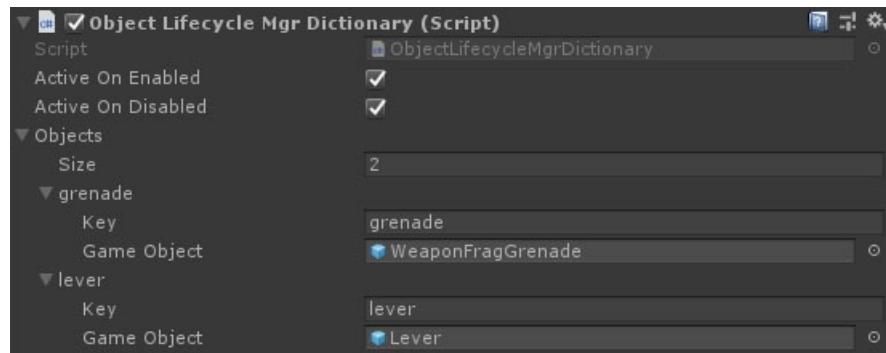
See Also

ObjectLifecycleMgrDictionary MonoBehaviour

Overview

The ObjectLifecycleMgrDictionary behaviour allows scripts to manage the active state of attached objects by name.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Active On Enabled	Boolean	Should all the attached objects be enabled when this is.
Active On Disabled	Boolean	Should all the attached objects be disabled when this is.
Objects	Entry Array	The objects being managed.

Entry

NAME	TYPE	DESCRIPTION
Key	String	A string key for the object.
Game Object	GameObject	The object to manage.

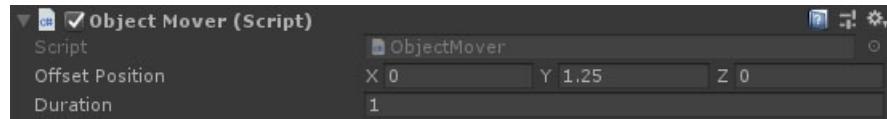
See Also

ObjectMover MonoBehaviour

Overview

The ObjectMover behaviour is attached to an object to interpolate from one position to another.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Offset Position	Vector3	The offset from the starting position to move to.
Duration	Float	The time taken to move from position to position.

See Also

ParticleSystemPlayOnEnable MonoBehaviour

Overview

The ParticleSystemPlayOnEnable behaviour is a simple utility component that is used to get the effect of "Play On Awake" on a [particle system](#) that is recycled and reused multiple times by the pooling system.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Particle Systems	ParticleSystem Array	The particle systems to play on enabling the object.

See Also

[PoolManager](#)

[Utilities](#)

PooledObject MonoBehaviour

Overview

The PooledObject behaviour is used to manage object lifecycle for a reusable object. It is managed by the [PoolManager](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Overflow	Dropdown	What should happen if you request an object from the pool when all of its items are in use. Grow means the pool will expand and add new items. Recycle means that the pool will return the oldest active pooled object (make sure to reset it). ReturnNull means the pool will return null as the result (make sure to handle this case as it could cause errors).

See Also

[PoolManager](#)

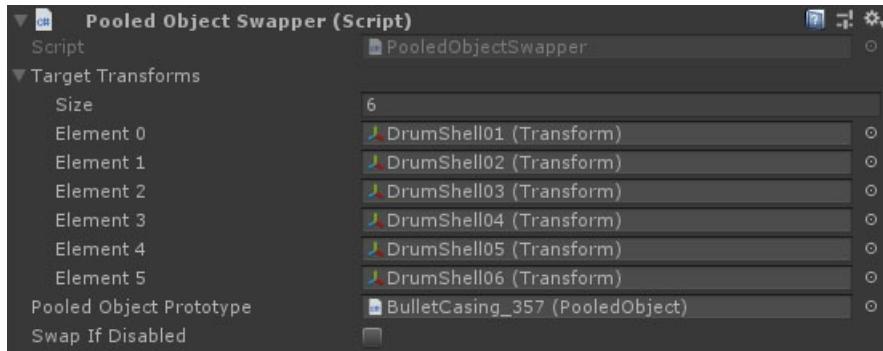
[Utilities](#)

PooledObjectSwapper MonoBehaviour

Overview

The PooledPhysicsObjectSwapper behaviour swaps out a number of objects with pooled objects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Transforms	Transform Array	The objects to swap.
Pooled Object Prototype	PooledObject	The pooled object to swap the target objects with.
Swap If Disabled	Boolean	Swap the target objects if they are disabled.

See Also

[PooledObject](#)

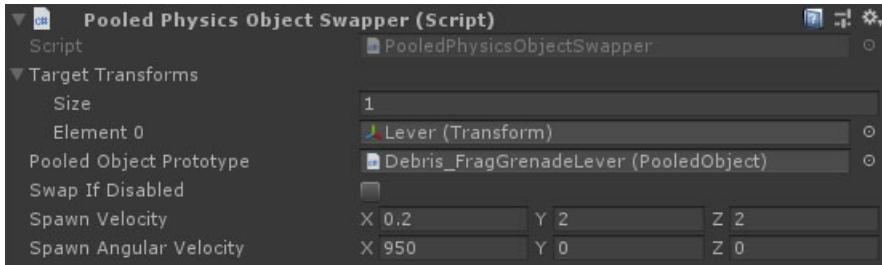
[PooledPhysicsObjectSwapper](#)

PooledPhysicsObjectSwapper MonoBehaviour

Overview

The PooledPhysicsObjectSwapper behaviour swaps out a number of objects with pooled physics objects. An example use is in the revolver where it swaps the empty shells for physics objects during a reload.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Transforms	Transform Array	The objects to swap.
Pooled Object Prototype	PooledObject	The pooled object to swap the target objects with.
Swap If Disabled	Boolean	Swap the target objects if they are disabled.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

See Also

[PooledObject](#)

[PooledObjectSwapper](#)

[Unity Rigidbody](#)

ReplaceObject MonoBehaviour

Overview

The ReplaceObject behaviour simply replaces the object it's attached to with the prefab in its settings. It needs triggering via code or using an event on something like an interactive object or trigger zone.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Swap Prefab	GameObject	The object to replace this one with.

See Also

[SlowMoSystem](#)

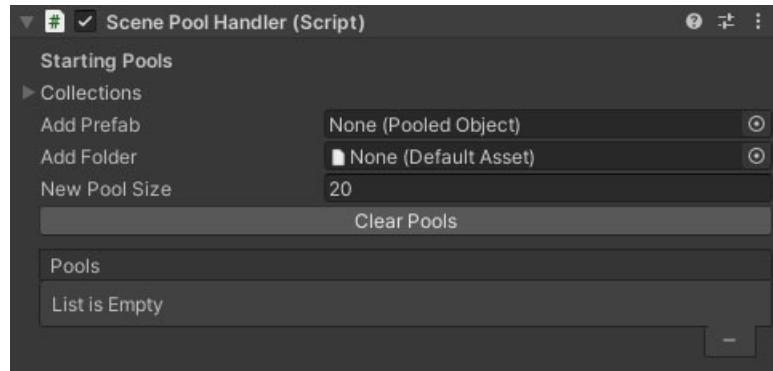
[CharacterTriggerZone](#)

ScenePoolHandler ScriptableObject

Overview

The ScenePoolHandler behaviour manages [PooledObject](#) prefab spawning and reuse within a scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Add Prefab	PooledObject	Adds a prefab to the pools list below, using the new pool size setting.
Add Folder	Folder	Adds all the prefabs within the folder and all its sub-folders to the pools list below, using the new pool size setting.
New Pool Size	Integer	The pool size to use for any new tools added to the pools list below.
Collections	PooledObjectCollection Array	The collections of pooled objects to set up at initialisation.
Pools	Pool Info Array	The pools to set up at initialisation.

Pool Info

NAME	TYPE	DESCRIPTION
Prototype	PooledObject	The prefab object to spawn.
Count	Int	The number of objects to instantiate for the pool.

See Also

[PooledObject](#)

[PoolManager](#)

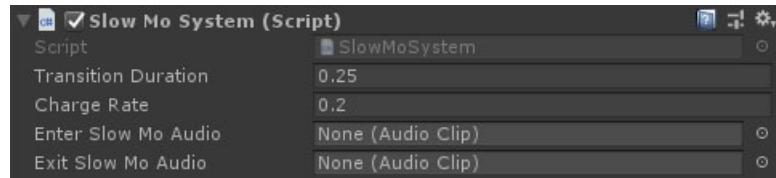
[Utilities](#)

SlowMoSystem MonoBehaviour

Overview

The SlowMoSystem behaviour is added to a the player character and exposes a simple API to allow slow-motion effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Transition Duration	Float	The (real, unscaled) time taken to transition time scales.
Charge Rate	Float	The amount of charge added per (real, unscaled) second.
Enter Slow Mo Audio	AudioClip	The sound to play on entering slow-mo.
Exit Slow Mo Audio	AudioClip	The sound to play on exiting slow-mo.

See Also

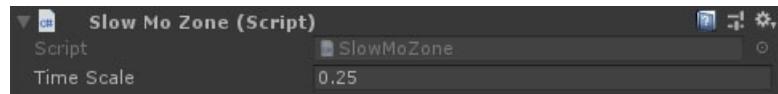
[InteractiveObject](#)

SlowMoZone MonoBehaviour

Overview

The SlowMoZone behaviour derives from [CharacterTriggerZone](#) and is used to define an area that will cause time to slow down when entered, and return to normal once exited.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time scale inside this zone.

See Also

[SlowMoSystem](#)

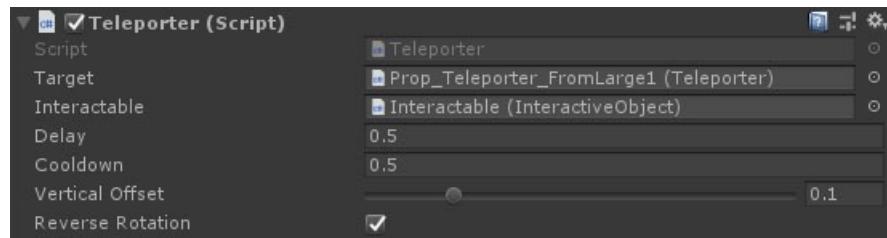
[CharacterTriggerZone](#)

Teleporter MonoBehaviour

Overview

The Teleporter behaviour is used to teleport a character from one point to another. It is triggered via an [InteractiveObject](#).

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target	Teleporter	The target teleporter to teleport to.
Interactable	InteractiveObject	The interactive object that triggers the teleport. Is disabled for cooldown.
Delay	Float	The delay between triggering and teleport.
Cooldown	Float	The cooldown blocks teleport until it has completed.
Vertical Offset	Float	An amount to raise the character on teleport to prevent ground overlap.
Reverse Rotation	Boolean	Should the relative rotation be reversed on teleport. For example walking in to teleporter A translates to walking out of teleporter B.

See Also

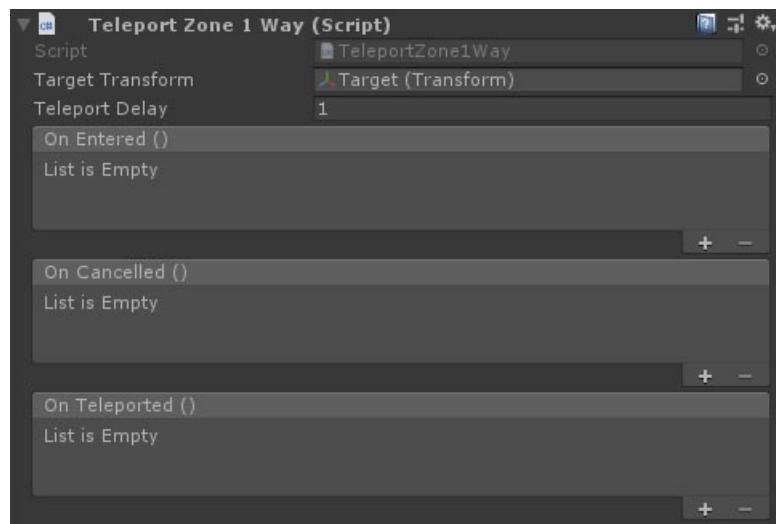
[InteractiveObject](#)

TeleportZone1Way MonoBehaviour

Overview

The TeleportZone1Way behaviour is used to teleport a character to a target transform position and rotation. It can be set with an optional delay and exposes events to control charge up and teleport / cancel effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Target Transform	Transform	The transform to teleport to. The character will match its position and orientation.
Teleport Delay	Float	The time between entering the trigger zone and teleporting. Allows for effects and feedback to communicate what's going on.
On Entered	UnityEvent	An event fired as soon as a character enters the teleport trigger zone.
On Cancelled	UnityEvent	An event fired if the character leaves before the teleport delay is over.
On Teleported	UnityEvent	An event fired after the character is teleported.

See Also

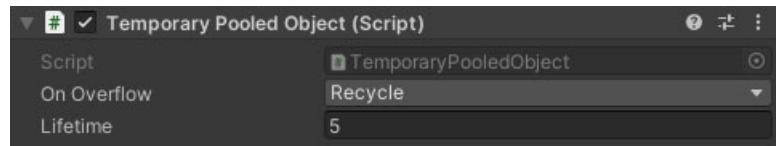
[InteractiveObject](#)

TemporaryPooledObject MonoBehaviour

Overview

The TemporaryPooledObject behaviour is used to manage object lifecycle for a reusable object. It is spawned from the [PoolManager](#), and will be returned after the specified time has elapsed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
On Overflow	Dropdown	What should happen if you request an object from the pool when all of its items are in use. Grow means the pool will expand and add new items. Recycle means that the pool will return the oldest active pooled object (make sure to reset it). ReturnNull means the pool will return null as the result (make sure to handle this case as it could cause errors).
Lifetime	Float	The duration the object will stay active before returning to the pool.

See Also

[PoolManager](#)

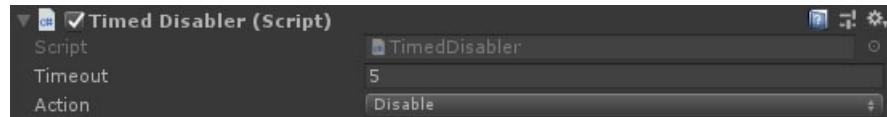
[Utilities](#)

TimedDisabler MonoBehaviour

Overview

The TimedDisabler behaviour is attached to an object to disable or destroy it after a set time.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Timeout	Float	The time after starting that the action will be performed.
Action	Dropdown	The action to performm on timeout. Options are Disable and Destroy

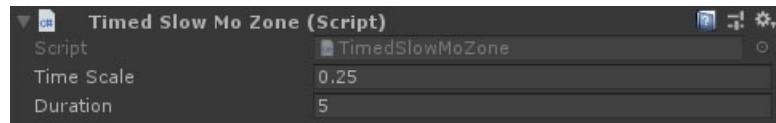
See Also

TimedSlowMoZone MonoBehaviour

Overview

The TimedSlowMoZone behaviour derives from [CharacterTriggerZone](#) and is used to define an area that will cause time to slow down when entered, and return to normal after a set period of (unscaled) time has passed.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time scale inside this zone.
Duration	Float	The duration the slow-mo effect should last.

See Also

[SlowMoSystem](#)

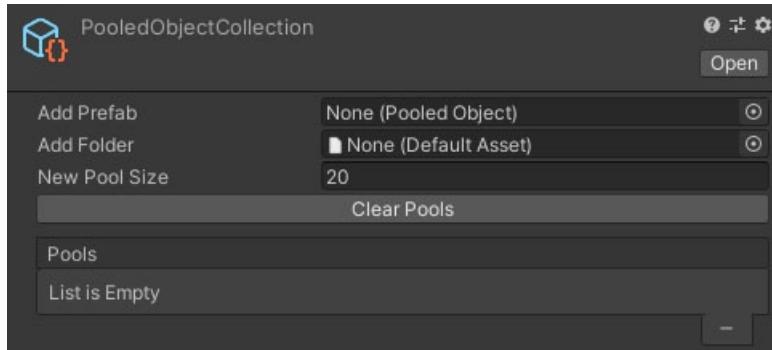
[CharacterTriggerZone](#)

PooledObjectCollection ScriptableObject

Overview

The PooledObjectCollection object is used to group together [PooledObject](#) prefabs into a collection that can be referenced by the [\[PoolManager\]\[1\]](#) or a [ScenePoolHandler](#) instead of having to reference the prefabs individually.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Add Prefab	PooledObject	Adds a prefab to the pools list below, using the new pool size setting.
Add Folder	Folder	Adds all the prefabs within the folder and all its sub-folders to the pools list below, using the new pool size setting.

Pool Info

NAME	TYPE	DESCRIPTION
Prototype	PooledObject	The prefab object to spawn.
Count	Int	The number of objects to instantiate for the pool.

See Also

[PooledObject](#)

[ScenePoolHandler](#)

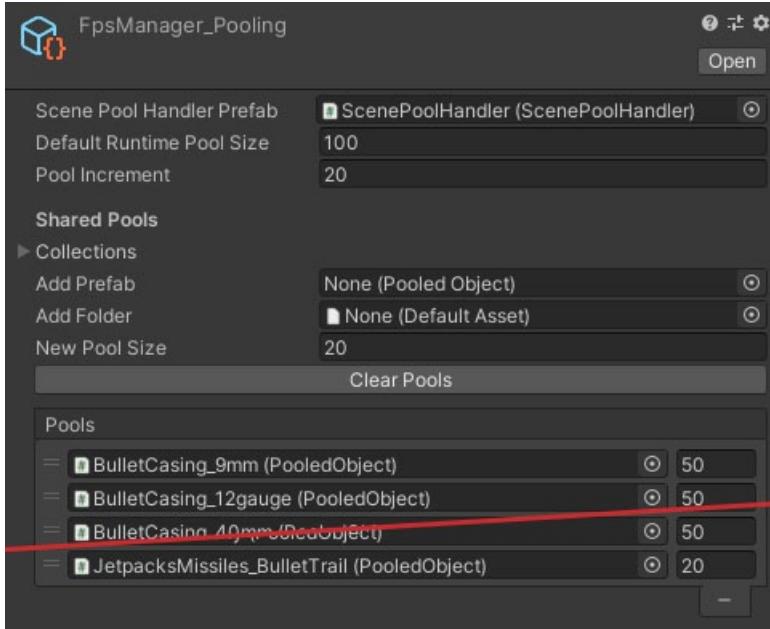
[\[PoolManager\]\[1\]](#)

PoolManager ScriptableObject

Overview

The PoolManager scriptable object specifies the [PooledObject](#) prefabs available across all scenes, and handles creation or registration of the [ScenePoolHandler](#) after scene loads.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Scene Pool Handler Prefab	ScenePoolHandler	A scene pool handler prefab to be instantiated in the scene if none already exists.
Default Runtime Pool Size	Int	The size of any new pools that are added to the array after initialisation.
Add Prefab	PooledObject	Adds a prefab to the pools list below, using the new pool size setting.
Add Folder	Folder	Adds all the prefabs within the folder and all its sub-folders to the pools list below, using the new pool size setting.
New Pool Size	Integer	The pool size to use for any new tools added to the pools list below.
Collections	PooledObjectCollection Array	The collections of pooled objects to set up at initialisation.
Pools	Pool Info Array	The pools to set up at initialisation.

Pool Info

NAME	TYPE	DESCRIPTION
Prototype	PooledObject	The prefab object to spawn.

NAME	TYPE	DESCRIPTION
Count	Int	The number of objects to instantiate for the pool.

See Also

[PooledObject](#)

[ScenePoolHandler](#)

[Utilities](#)

Surfaces

Overview

The NeoFPS surfaces system is an in-development system for querying impact surface materials. It is used in a number of places in the NeoFPS framework, including [Footsteps](#) and [Bullet Hit Effects](#).

Surface Manager

The [SurfaceManager](#) scriptable object is located in the NeoFPS/Resources folder, and accessible through the NeoFPS hub. The surface manager associates specific surfaces with effects and audio clips for use in bullet hits, etc. You can also place a [SurfaceFxOverrides](#) behaviour in the scene in order to override the surface effects on a scene by scene basis.

Adding New Surface Types

The `Surface` identifier is a [generated constant](#) that lists the available surfaces in the game. Adding new surfaces is as simple as editing the [ConstantsSettings](#) and regenerating the constant. Keep in mind that in the inspector generated constants are stored by index. This means that reordering or removing existing surfaces will not be correctly picked up in inspector properties. This means that it is best to try and decide on the available surfaces early in a project if possible.

Once you have added new surface types, you can then access these when assigning audio clips in the [SurfaceAudioData](#) assets used for things like footprint and bullet hit audio, and the [SurfaceHitFxData](#) asset used to specify impact visual effects among others.

Surface Behaviours

Surface behaviours (currently just [SimpleSurface](#)) are attached to physics objects that need a surface ID such as environment or prop colliders. This means that if the object is shot, hit or stepped on the surface behaviour will tell the relevant system the surface they need to react to with audio or graphics effects.

New surface behaviours will be added in future updates.

Adding New Bullet Hit VFX

The [SurfaceManager](#) holds a reference to the [SurfaceHitFxData](#) asset that specifies all the bullet hit VFX for different surface types. This is currently shared for all bullet and melee impacts, but in the future you will be able to specify separate hit FX data for different attack types. The architecture of this has not been decided yet.

Each of the entries in the [SurfaceHitFxData](#) asset references a prefab with a component that implements the `BaseHitFxBehaviour` base class. The demos all use the [ParticleSystemHitFxBehaviour](#) which works by moving a group of particle systems to align with the hit point, and calling `Emit()` on them. This means that all of the particles per surface type reuse the same particle systems instead of having hundreds of separate systems all working at once. Another option that helps when replacing the impact FX with other assets is the [PooledObjectHitFxBehaviour](#). This uses the pooling system to recycle objects for efficiency. This can be combined with the [ParticleSystemPlayOnEnable](#) component to trigger particle systems each time a pooled object is activated.

You can also implement your own FX by writing a script with a class that inherits from the `BaseHitFxBehaviour` base class.

See Also

[SurfaceManager](#)

[SimpleSurface](#)

Surfaces Troubleshooting

The following are issues that have been raised multiple times on the [NeoFPS Discord](#) or are common symptoms for errors with the setup of NeoFPS components.

Impact VFX Point In The Wrong Direction

The `ParticleSystemHitFxBehaviour` component orients the particle system so that the Z axis is facing along the normal of the impact hit. If a hit effect is spraying in the wrong direction, you should open its prefab and check that each of the particle system objects is set up to emit along the Z-axis of the root object, and make sure that the particle system's **Simulation Space** is set to **World**.

When I Shoot A Wall, Existing Bullet Hit FX Vanish

This will happen if the hit effect particle systems are set to simulate in local space. The surface system does not spawn multiple particle systems for multiple hits. Instead, for efficiency & performance reasons, it positions a single set of particle systems at the impact point, and calls `Emit()` on them with a count based on the impact size. This means that the particle systems need to be simulated in world space, or they will move with the particle systems to their new position.

To fix this, open the hit effect prefab and, for each of the particle systems, check that their **Simulation Space** properties are set to **World**.

Impact Effects Keep Triggering After I Stop Shooting

This implies that the particle systems have been set to loop instead of fire once and then stop. To fix this, open the hit effect prefab and, for each of the particle systems, check that their **Looping** properties are set to **false**.

There Are No Decals When I Shoot An Object

NeoFPS currently uses a very simplistic decal system that simply emits a decal particle at the hit point using a particle system. This means that decals cannot move with dynamic objects - they should only be shown when shooting static objects. When you shoot an object, the surface system checks whether the object has a rigidbody, and will not show decals in that case. *It also checks the layer of the object and only shows decals for specific layers.* The "Default" layer is not included in the decal layers because I didn't want to assume that all objects on that layer would be static (or use a rigidbody if not). If you are happy with those restrictions, you can enable decals on the "Default" layer from the [Surface Manager](#).

To access this, go to the Unity menu: *Tools/NeoFPS/NeoFPS Hub*, and in the **Managers** section, select the **Surfaces** entry. In the surface manager, set the **Decal Layers** property to include the **Default** layer.

Footsteps or Impact Sounds Are Wrong

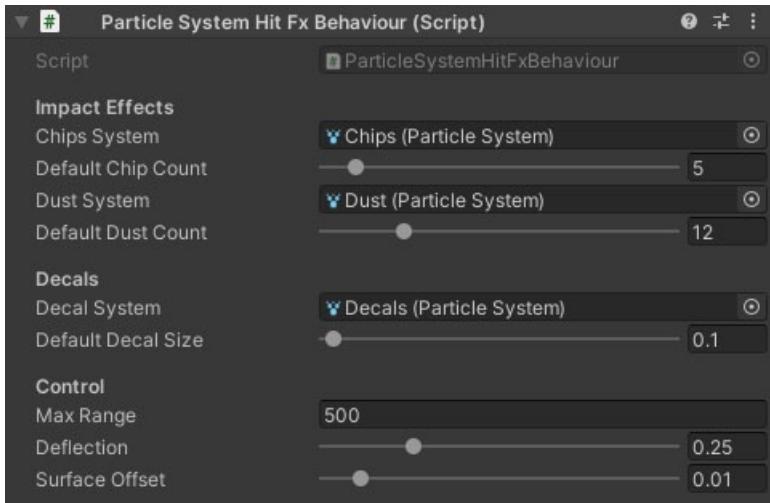
You can find steps for troubleshooting this problem in [Audio Troubleshooting](#).

ParticleSystemHitFxBehaviour MonoBehaviour

Overview

The ParticleSystemHitFxBehaviour ties together a number of [particle systems](#) to create surface impact effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Chips System	ParticleSystem	The particle system for chunks of surface material (eg. wood splinters, paint chips).
Default Chip Count	Int	The number of chip particles for a bullet hit with size 1.
Dust System	ParticleSystem	The particle system for fine dust spray or similar.
Default Dust Count	Int	The number of dust particles for a bullet hit with size 1.
Decal System	ParticleSystem	The particle system used to place a decal.
Default Decal Size	Float	The amount of deflection for the hit system (0 follows the impact normal, 1 follows the reflected impact ray).
Max Range	Float	The maximum distance from the player camera that the effect will be shown.
Deflection	Float	The amount of deflection for the hit system (0 follows the impact normal, 1 follows the reflected impact ray).
Surface Offset	Float	The distance from the surface to place the decal (prevents z-fighting).

See Also

[Surfaces](#)

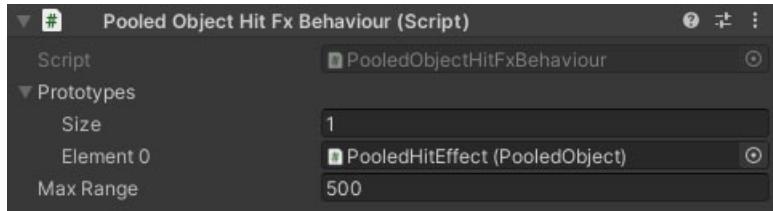
[SurfaceHitFxData](#)

PooledObjectHitFxBehaviour MonoBehaviour

Overview

The PooledObjectHitFxBehaviour spawns an object at the hit location & rotation using the NeoFPS pooling system. This can be used with a [TemporaryPooledObject](#) to ensure the spawned object is returned to the pool after a brief delay.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Prototypes	PooledObject	The pooled objects to choose from.
Max Range	Float	The maximum distance from the player camera that the effect will be shown.

See Also

[Surfaces](#)

[SurfaceHitFxData](#)

SimpleSurface MonoBehaviour

Overview

The SimpleSurface behaviour is attached to objects in the scene and assigns them a [surface tag](#) for responding to effects.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Surface	FpsSurfaceMaterial	The surface material ID.

See Also

[Surfaces](#)

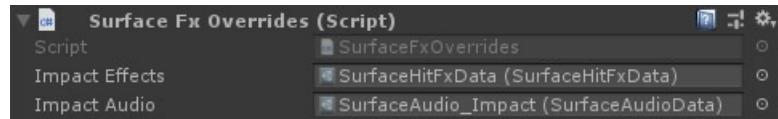
[Generated Constants](#)

SurfaceFxOverrides MonoBehaviour

Overview

The SurfaceFxOverrides behaviour is used to override the surface effects of the [SurfaceManager](#) for a specific scene.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Impact Effects	SurfaceHitFxData	The impact special effects for things like bullet hits.
Impact Audio	SurfaceAudioData	The audio library for impact audio, eg. bullet hits.

See Also

[Surfaces](#)

[SurfaceManager](#)

[SurfaceHitFxData](#)

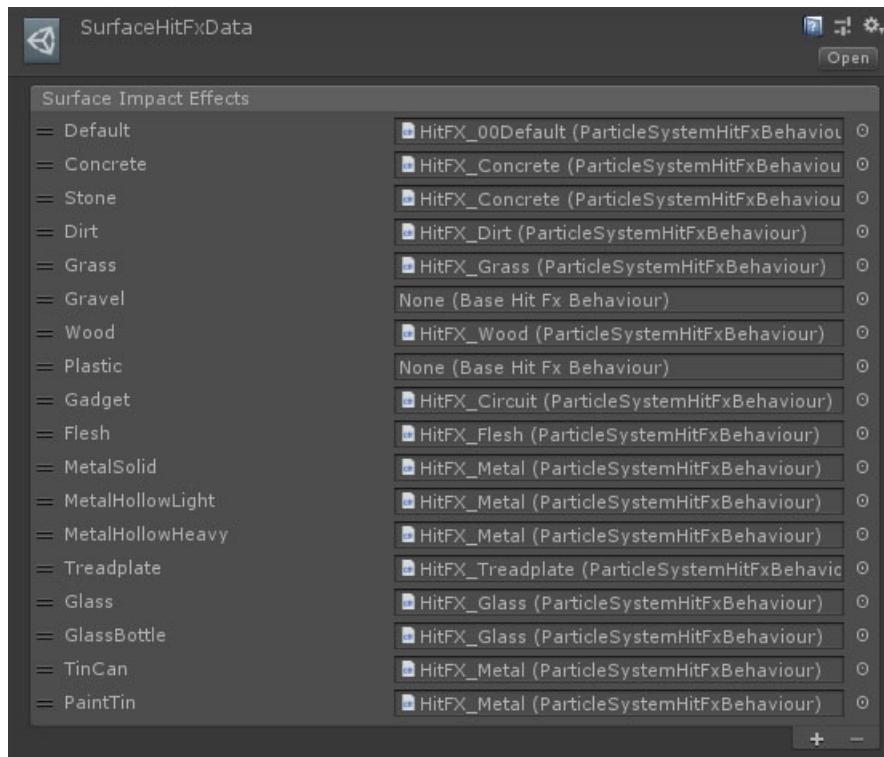
[SurfaceAudioData](#)

SurfaceHitFxData ScriptableObject

Overview

The SurfaceHitFxData scriptable object specifies per-surface visual effects for impacts.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Data	HitFx Array	Per-surface hit effects. Must inherit from the <code>BaseHitFxBehaviour</code> class. Entries correspond to the values in the <code>FpsSurfaceMaterial</code> generated constant.

See Also

[Surfaces](#)

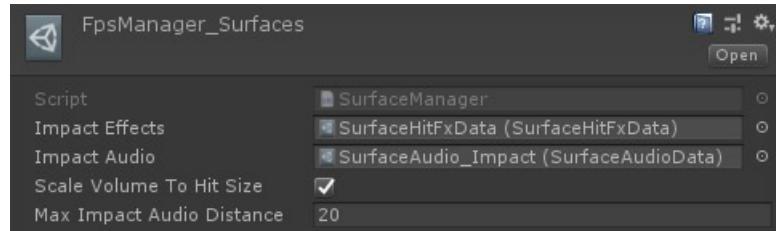
[ParticleSystemHitFxBehaviour](#)

SurfaceManager ScriptableObject

Overview

The SurfaceManager scriptable object is a central manager that handles visual and audio impact effects with various surfaces. You can also override this on a scene by scene basis by using a [SurfaceFxOverrides](#) behaviour.

Inspector



Properties

NAME	TYPE	DESCRIPTION
Impact Effects	SurfaceHitFxData	The impact special effects for things like bullet hits.
Impact Audio	SurfaceAudioData	The audio library for impact audio, eg. bullet hits.
Scale Volume To Hit Size	Bool	Should impact audio be louder for heavier hits.
Max Impact Audio Distance	Float	The maximum distance from the player character to play impact audio.

See Also

[Surfaces](#)

[SurfaceHitFxData](#)

[SurfaceAudioData](#)