Realistic FPS Prefab

What is the Realistic FPS Prefab for Unity3D?

The realistic FPS Prefab is an easy way to implement the core features of first person games into your Unity3D projects with a few mouse clicks. Set up is quick and just requires dragging and dropping the FPS object into your scene and assigning a layer to collision geometry. The asset has been designed with a focus on keeping scripts simple and ensuring that motion and effects are smooth and glitch free. The Realistic FPS Prefab is a great learning tool and template for FPS games, so look it up on the Unity Asset Store today!

What features does the Realistic FPS Prefab offer?

Full physics interaction with a rigidbody character controller that has mass in the game world. Pick up and throw objects, ride elevators and horizontal platforms, create physics puzzles, and climb ladders!

Player state handling for sprinting, crouching, swimming, and jumping movement states with seamless transitions, easily configurable speeds, and air manipulation. Player health management with damage from falling, explosives, enemy fire, and elevators is also included.

Advanced weapon positioning using sine bobs, iron sights, weapon sway, position changes based on player movement state, and animation for actions like reloading and readying weapons.

Camera movement effects such as weapon fire, landing, and pain kicks, as well as camera view animation to accompany weapon actions like reloading. Fully configurable sine bobbing based on player movement states is also supported.

Weapon behavior customization allows switchable fire modes, droppable weapons, weapon inventory handling, shotguns, scoped weapons, in-depth accuracy and recoil, single shell reloading, and melee weapons all by modifying the variables of one script instance from the editor - no additional scripting required to make a new weapon.

Item pickups and objects to place in your scene for ammo, health, and weapons that can either be static or moveable (and throwable) rigidbodies. Explosive and breakable objects as well as a variety of other interactive objects are available for placement in scenes.

Weapon effects like bullet impacts per surface type, smoke, tracers, and ejecting casings with simple variables to control size, velocity, and ejection delay of shells.

Ease of use through extensive documentation and commented code, simple drag and drop functionality of game objects into scenes, compatibility with version 3.5 and higher of Unity free and professional, and customization variables accessible from the inspector.

An example sandbox scene with an improved default AI for testing.

Are there any more features planned?

Yes, we plan to continue adding features and improving the asset however we can. The addition of grenades, bows and arrows, and true sniper sights are possible in the future. Follow our progress at http://azulinestudios.blogspot.com

Contents

Version Notes

1.17

-Made weapon sprinting animation return to center sooner after player starts falling.

-Unhid the mouse cursor when paused (or timescale = 0).

-Added collider sweep in front of player to allow better jumping over stationary obstacles.

-Fixed sights not rising if sprint is cancelled, but sprint and forward buttons are still held.

-Added an extra check to make animated camera angle values return to zero when not playing animations. This prevents rare occurrences of the weapon and camera becoming misaligned.

-Added **muzzleLightDelay** to *WeaponBehavior.cs* to define delay before muzzle light begins fading.

-Added **muzzleLightReduction** to *WeaponBehavior.cs* to define speed of muzzle light fading.

-Changed name of "**childNum**" var in *PlayerWeapons.cs* to "**currentWeapon**" for clarity.

-Added **defaultFov** var to *Ironsights.cs* to allow the default camera FOV to be customized.

-Added **sprintFov** var to *Ironsights.cs* to allow the camera FOV while sprinting to be customized.

-Added **weaponCamFovDiff** var to *Ironsights.cs* to allow the weapon camera FOV to be customized. This value is subtracted from the main camera FOV to control the perspective of weapon models.

-New Footsteps.cs script defines different footstep sounds and landing sounds per surface type.

-New WeaponEffects.cs script defines weapon impact effects, impact marks, and weapon impact sound effects per surface type and contains effects related functions that were previously a part of the *WeaponBehavior.cs script*.

-**Pain Screen Kick Amt** of *FPSPlayer.cs* now can be used to set magnitude of screen kicks when player takes damage.

-Added the var **myWaypointGroup** to *AI.js* and added the **waypointGroup** var to *AutoWayPoint.js* script to set waypoint groups for NPC patrols.

-Fixed **doPatrol** behavior in *AI.js* so NPCs stand watch if not patrolling.

-**RandomSpawnChance** of *AI.js* can be used to make NPCs have a random chance of spawning.

-Airborne NPC character controllers now move to the ground when scene is loaded.

-Added **waypointNumber** var to *AutoWayPoint.js* to define the path order of waypoints that the NPC will patrol starting from number 1 and returning from the last numbered waypoint in the waypoint path group.

-Locked mouselook roll in *SmoothMouseLook.cs* to prevent gun rotating with fast mouse movements.

-Player can now drop weapons with the **droppable** value set to true in *WeaponBehavior.cs*.

-Player now drops weapon, falls down, and is affected by physics forces on death.

-Added **addsToTotalWeps** bool to *WeaponBehavior.cs* and **backupWeapon** int to *PlayerWeapons.cs* to allow player's backup weapon to be skipped in auto weapon selection during weapon drops and swaps, to make the backup weapon not be counted toward player's weapon total, and to prevent the backup weapon from being dropped.

-The **maxWeapons** int in *PlayerWeapons.cs* now defines the maximum amount of weapons that can be held at one time.

-*WeaponPickup.cs* will now read its **removeOnUse** bool and the **dropWillDupe** bool in *WeaponBehavior.cs* to determine if the current weapon should be dropped or destroyed when picking up a new weapon.

-The player's crosshair now changes to an "X" reticle if the player can't pick up the weapon under the crosshair, such as when the player has already picked up the weapon from an armory (**dropWillDupe** set to true) and can't drop the weapon due to this allowing ammo duplication exploit.

-Added unique Texture2D vars to item pickup scripts to allow custom pickup reticles/icons.

-Added **showAimingCrosshair** to *WeaponBehavior.cs* to selectively disable aiming crosshair for weapons like sniper rifles.

-Added **useSwapReticle** var to *FPSPlayer.cs* to toggle the display of the swap reticle if the weapon pickup under player's crosshair will be switched or swapped with current weapon.

-Improved canceling of non-magazine reload if fire button is pressed and player has loaded at least 2 shells/bullets.

-Prevented switching weapon during non-magazine reload from causing neutral anim glitch and delay at the start of next reload for that weapon.

-**playerHeightMod** and **crouchHeightPercentage** in *FPSRigidbodyWalker.cs* will now control standing and crouching height of player.

-*WaterZone.cs* script can be attached to a box trigger collider to create a swimmable liquid volume that handles player swimming and water effects.

-Added **MarkDuration** to *FadeOutDecals.cs* to define the time that impact marks should stay before fading.

-*WorldRecenter.cs* now prevents spatial jitter of game objects due to floating point precision loss and allows much larger scenes to be used.

-Made objects that are set to layer 9, the "Ragdoll or Usable" layer, run any *ActivateObject()* functions in their scripts if the player presses the use button when object is in the crosshair.

-By default, the Tab button now pauses the game.

-**barrelSmokeShots** in *Weaponbehavior.cs* now defines number of shots required to make smoke rise from gun barrel.

-Glass surface type added that uses *BreakableObject.cs* script and prefab.

-*MineExplosion.cs* can be attached to a game object to create landmines.

-*ExplosiveObject.cs* can be attached to a game object to make it explode after being damaged.

-Made the zoom and sprint button behaviors selectable between toggle, hold, or both (tap to toggle, press to hold).

-*MonsterItemTrap.cs* can be used to spawn enemies and create ambushes when player activates or uses an item.

-*MonsterTrigger.cs* can be used to spawn enemies and create ambushes when player enters a trigger zone.

-*EdgeClimb.cs* can be attached to a trigger near a ledge to allow the player to climb themselves up it.

-*EdgeClimbTriggerActivate.cs* can be used to create a trigger that will reactivate an edge climb trigger that has disabled its collider after player has climbed up it.

-Fixed non magazine reloads with **bulletsToReload** amount greater than 1 not counting reloaded bullets correctly.

-Made jump and crouch buttons move player up and down a ladder trigger if they aren't holding a forward or backward move button already.

-Added Zombie NPC prefab with animations, sound, and ragdoll.

-**targetPlayer** var of *Ai.js* will now determine if NPC should ignore player.

-Made the raycast length for item pickups scale with the **playerHeightMod** amount.

-**reachDistance** var of *DragRigidbody.cs* is now proportionately scaled by **playerHeightMod** amount in FPSRigidBodyWalker.cs

-The attack range of NPCs will now be reduced by the **crouchRangeMod** amount of **AI.js** when player is crouching.

-The **useAxisInput** var of *FPSPlayer.cs* will now allow the player to use Unity’s built in horizontal and vertical axes for movement (for joystick input).

-Changed the shell ejection method to use **Shell Prefab RB** and **Shell Prefab Mesh** object references of *WeaponBehavior.cs* to render physics and interpolated mesh separately for compatibility with greater ranges of fixed timestep and framerate speeds.

-By default, the Q button now enters slow motion/bullet time mode.

-The **useViewClimb**, **viewClimbup**, **viewClimbSide**, and **viewClimbRight** vars of *WeaponBehavior.cs* can now be used to create non-recovering view recoil when firing.

-The **useRecoilIncrease**, **shotsBeforeRecoil**, **viewKickIncrease**, and **aimDirRecoilIncrease** vars of *WeaponBehavior.cs* can now be used to control sustained fire recoil for rapid firing weapons.

-Added **lowerGunForClimb** var to *FPSRigidBodyWalker.cs* to allow lowering weapon when climbing ladders or ledges.

-Added **playClimbingAudio** var to *Ladder.cs* to selectively prevent climbing sounds from playing while climbing a surface.

1.16

-Made changes to several scripts to allow Time.timescale to be set to 0 (for game pausing) and back to a positive value.

-Changed the input smoothing method in SmoothMouseLook.cs.

-Added **shellRldAnimSpeed** var to *WeaponBehavior.cs* to allow per-weapon customization of animation speed for reloading single shells/bullets.

-Added **RemoveOnUse** var to all pickup scripts which can be unchecked to make pickups stay after use (for making armories, ammo crates, etc.)

1.1

-Added a sniper rifle and pickups to demonstrate scoped weapons.

-*Attack Range* variable of *AI.js* is now properly taken into account for determining when to attack the player.

-Fixed raycast calculation in *DragRigidbody.cs* so user won't have to click outside and back in the running game window in the editor to drag rigidbodies.

-Fixed rigidbodies keeping high drag values resulting in objects falling slowly.
-Controls can now be easily set from the inspector in the *FPSPlayer.cs* script on the *FPS Player* object

-Added **swayAmountZoomed** variable to *WeaponBehavior.cs* to control zoomed weapon sway which can be used for reducing the zoomed sway of scoped weapons.

-Added **swayAmountUnzoomed** variable to *WeaponBehavior.cs* to allow customization of unzoomed sway amount for individual weapons.

-Miscellaneous inspector visibility cleanup.

1.05

-Added **CameraRollAmt** variable to CameraKick.cs to allow for animation of camera roll angles (currently used in melee weapon camera swings).

-Changed Ironsights button behavior to tap to toggle and press to hold.

-Decreased default zoom bobbing and removed weapon lowering effect for smoother view weapon model movement when zoomed and walking.

-Improved frame rate independence of bobbing transitions in Ironsights.cs.

-Fixed shotgun last reload anim from using reload speed instead of last reload speed when ammo was 1.

-Increased Melee hit detection area by using a sphere cast instead of ray cast.

-Rotation and scale of hit marks now remain consistent even if parent object is unevenly scaled in editor.

-Updated onscreen F1 help text.

1.0

-Initial Release

Before we begin,

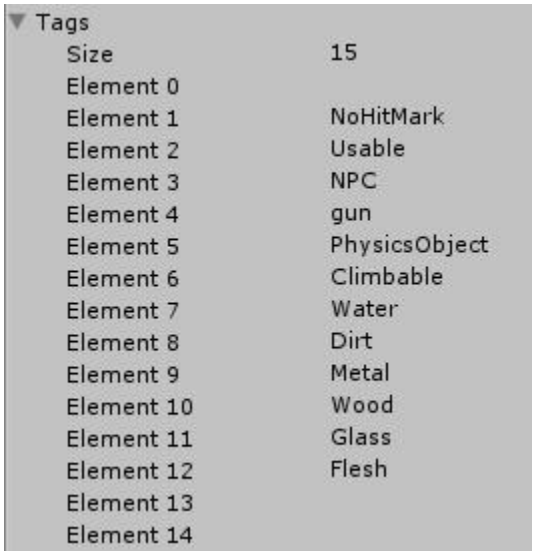
There are a few important pieces of information to take into consideration when importing the FPS Prefab into other projects.

First, this asset is compatible with **Unity 3.5** and **Unity 4.0**. If you start with using 3.5 and then decide to upgrade your project to 4.x, make sure that *all* children underneath the *FPS Weapon* object are active to avoid a mix-up of *activeInHeirarchy* and *activeSelf* attributes that are added in Unity 4.0. Other than this issue, Version interchangeability of this asset between 3.5 and 4.x is automatic.

A note on *FPS Weapon* children: This asset uses an additional weapon camera to render weapons and 2D elements over the game world, but the default weapons are scaled to fit inside the player capsule collider to provide the option of not needing a separate weapon camera. This might make bullet shells look small in the game world. To solve this, your models can be scaled up and repositioned from the default shell and weapon meshes if using the weapon camera.

Also, versions lower than 1.17 of this asset were developed using a Fixed Time Step of 0.01, which was set automatically, along with other physics settings, in the Start() function of the FPSPlayer script. In version 1.17, the Fixed Timestep is **no longer set automatically** in FPSPlayer.cs and may be set to any value due to improved variable framerate and Fixed Timestep compatibility introduced in version 1.17.

And finally, the Realistic FPS Prefab uses layers and tags to identify game objects and automatically configures collision layers in the Start() function of the FPSPlayer script. If more layers are needed for other behaviors while using this asset, please name your layers as shown in the image below, and add any additional layers below layer 18.



Tutorials

Adding the Realistic FPS Prefab to a new scene

Adding the asset to your scenes is easy. To do so, drag the *!!!FPS Player Main* object in the !Realistic FPS Prefab Files folder in the Project Library into the 3D Scene window of the editor and place the FPS prefab where you want the player to start, making sure that the green wireframe of the player's capsule collider is above the ground.

You may also expand the !!!FPS Player Main/FPS Camera objects in the Hierarchy window, select the *Main Camera* object, and rotate the main camera's yaw angle to the view the player will see when starting the level.

The last step is to assign the World Collision layer (layer 10) to any objects that the player will collide with, stand on, or shoot and leave bullet impact effects. This layer should be assigned carefully with efficiency in mind, as it will be checked by a raycast and capsule cast in the FPSRigidbodyWalker script, a raycast in the WeaponBehavior script, and by moveable or draggable rigidbodies. Large objects like sky domes or any other visual effect object should be excluded from this layer for efficiency.

How to add new weapons

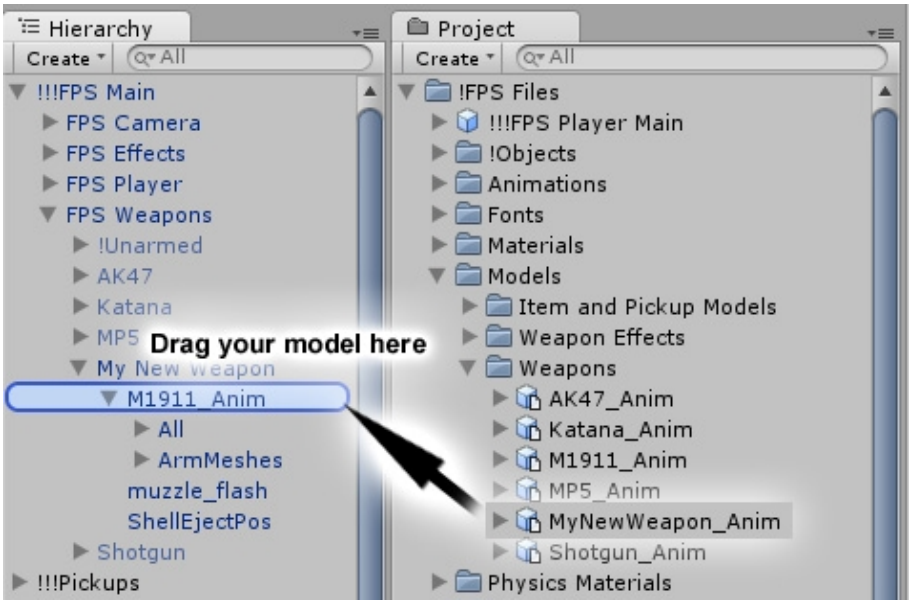
Adding a new weapon to the Realistic FPS Prefab takes only a few steps and requires that you have a weapon model asset imported to your Project Library. The weapon model should have animations to make it move during firing and reloading, though it's possible to add simple weapon animations to a non-animated model through code.

Once you have a weapon model, expand the *!!!FPS Player Main* object in the Hierarchy window and then expand the *FPS Weapons* object. Below it, you will see objects with weapon names. To create a new weapon, find the object that most resembles the type of weapon you want to add, right click that weapon object, and select duplicate. Then rename the duplicated weapon object to your weapon name and select this object with the left mouse. In the inspector window, you should see a script called WeaponBehavior with many editable variables. This script controls nearly all weapon actions and we will refer to it later.

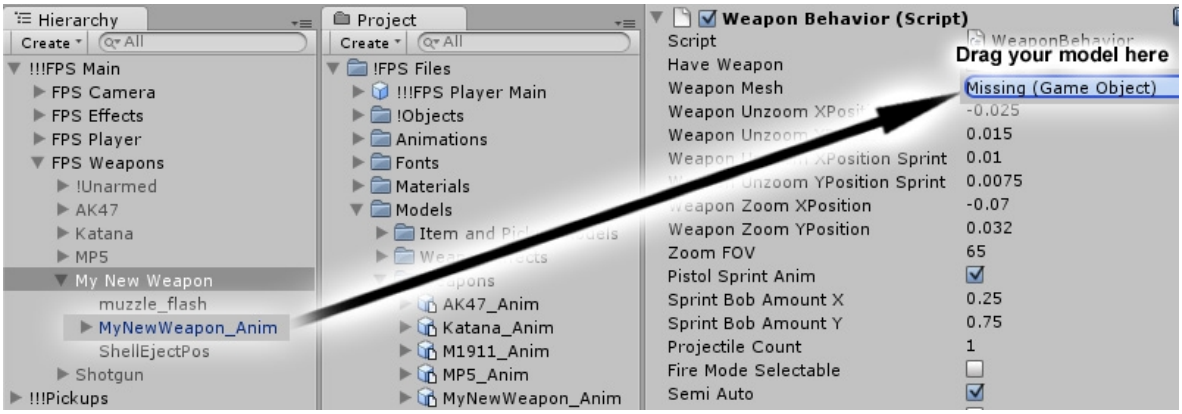
Then, if the other weapon objects are active and their view models are visible and overlapping in the 3D Scene window, hold control and click on all weapon objects except the one you duplicated, disable them by clicking the active check box in the upper left of the inspector window, and select the "Deactivate Children" option (Unity 3.5). You should now see only the model of the weapon object you duplicated near the center-top of the player's green capsule collider wireframe.

Expand your new weapon object in the Hierarchy window. Below it, there are three children; a weapon mesh with an _anim suffix, a muzzle_flash object, and a ShellEjectPos object, unless you duplicated the Katana, which only has a weapon model and a null muzzle_flash stand-in object. The next step is to add your weapon mesh to the FPS Prefab under your duplicated weapon object's hierarchy. To do this, find your weapon model in the Project Library window, and drag it over the weapon mesh with the _anim suffix under your duplicated weapon object in the Hierarchy window. This will import your mesh as a child of the existing mesh, inheriting its scale, rotation, and position.

Then select your model and drag it over the parent weapon object you duplicated to get the new model out of the original model's hierarchy. Unity might warn you that you are losing the link to the FPS Prefab, but we can apply our changes to the prefab in the library later. You should now see your weapon model in roughly the same place as the original model of the weapon object in the 3D Scene view, but don't delete the old model yet. You can move your newly imported weapon model so its iron sights line up with the previous model's. Once you are satisfied with your weapon model's position, you can delete the original model with the _anim suffix. At this point, you should also check that your newly imported weapon mesh has the correct animations assigned to its animation component as well.

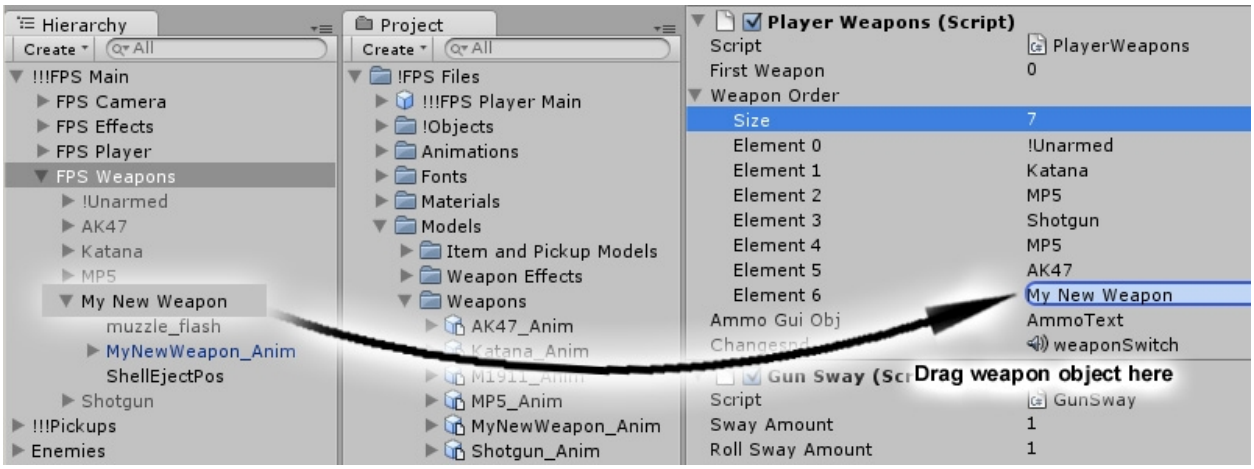


Now we need to tell the weapon script which model to use for your weapon. Near the top of the WeaponBehavior script's variable list, there is a variable named Weapon Mesh. With your weapon object selected in the Hierarchy window and the WeaponBehavior variables visible in the Inspector window, Find your weapon model in the Hierarchy window under your weapon object and drag it to the Weapon Mesh variable of the WeaponBehavior script so your weapon model's name appears in the Weapon Mesh field.



Next, in the 3D view, we move the muzzle_flash object to the end of your weapon model's barrel and the ShellEjectPos object to the position that shell casings should be ejected from the weapon.

Now we have to update the weapon order list in the PlayerWeapons script to include your weapon. This list tells the PlayerWeapons script which weapons to select before others and assigns the weapon an array index number that is referenced by the PlayerWeapons, WeaponBehavior, and item pickup scripts. Select the *FPS Weapons* object in the Hierarchy window and expand the Weapon Order array to display the weapon list. Increase the array length by one and drag the weapon object you duplicated (the parent of the weapon model, muzzle_flash and ShellEjectPos objects) to the new spot on the weapon order list.



If you want your weapon to have different characteristics than the original weapon you duplicated, you can select the weapon object and change the variable values listed for the WeaponBehavior script to your liking.

To test the weapon, you must first decide if you want the player to have to pick up the weapon from the game world, or if they should spawn with the weapon already in their inventory. To try your weapon out right away, all you have to do is select the weapon object and set the "Have Weapon" value of the WeaponBehavior script to true (checked box), set "ammo" to a value greater than zero, and start the game. Your weapon should be selectable with the mouse wheel. To let the player also select the weapon with a number key, you must uncomment a few lines of code in the PlayerWeapons script. This video http://youtu.be/oh3GzWoceY4 also demonstrates the process of adding new weapons.

Item Pickups

If you want to have the player pick up your weapon from the game world, you need to create weapon and ammo pickups. First of all, you should have the weapon and ammo pickup models you want to use imported into your Project Library. Then, in the Project Library window, expand or view the *!Realistic FPS Prefab Files/Objects/!!Pickups* folder and duplicate a Pickup_weaponname object and an Ammo_weaponname object. Then rename the "weaponname" part of the names to your weapon's name. Then you need to set the “Weapon Number” value of the Weapon Pickup and Ammo Pickup scripts to the number of the weapon in the “Weapon Order” array in the PlayerWeapons script (the MP5’s weapon number is 2 and the katana’s number is 1, for example). Also, the existing item objects' mesh should be swapped with your new weapon or ammo mesh and its material. And finally, you just need to update the “Ammo To Add” amount of the Ammo Pickup script.

You can choose to have the pickups be moveable or static. For the static pickups, just drag the pickup object into the scene from the library and uncheck to deactivate the object's rigidbody component. Both pickups can be dragged from the library into the 3D scene view and moved and rotated as needed.

Adding New NPCs

The first step to adding a new NPC is dragging an existing NPC object from the project library from either *Assets\!Realistic FPS Prefab Files\~Demo Scene Assets\~ZombieNPC\Objects* for the zombie NPC or from *Assets\!Realistic FPS Prefab Files\~Demo Scene Assets\~AlienNPC\!*Objects for the robot NPC. Then you can click on the RobotNPC or ZombieNPC object in the hierarchy view and select Break Prefab Instance from the GameObject menu so you won't accidentally overwrite the original.

Then you can expand the hierarchy of the duplicated NPC in the Hierarchy window and delete all of its child mesh rendering objects. After you've done this, you can locate the new NPC character mesh in the project library, and drag it over the duplicated root NPC object. The animation component of the duplicated NPC object can also be deactivated by checking the box in the upper left of its component window.

Now we want the AI.js component of the root/parent NPC object to use the animations of the child character mesh you just added to this object. To do this, expand the root NPC object's hierarchy and drag the child mesh object over the Object With Anims field of the AI.js component. Now, looking in the 3d scene view, make sure that the new NPC character mesh is the right height and fits roughly inside the bounds of the character controller capsule of the root NPC object. If the NPC mesh is too small, you can try increasing its Scale Factor in its Animation Import Settings. You'll also want to make sure that the imported character mesh has these animations defined in the Animations section of the mesh's Animation Import Settings: shoot, idle, walk, run (case sensitive). When you test the scene, your new NPC should animate correctly and chase after the player.

Overview

The *!!!FPS Player Main* object is the parent of four child objects; *FPS Camera*, *FPS effects*, *FPS player*, and *FPS weapons*, that organize the workload of the FPS Prefab. In this section, we will review each of these four objects and their related scripts and child game objects. For more detail on how these scripts operate, please refer to the scripts themselves for detailed comments and descriptions.

FPS Player Main

This object serves as the main parent/root of the prefab which simplifies prefab placement in the scene and is deleted by the RemovePrefabRoot.cs script when loading a scene. This is done to allow the FPS Player's local and world position coordinates to be synchronized.

RemovePrefabRoot.cs

Deletes the FPS Player Main object after deparenting the gameObjects included in the children array.

Children : an array of the child gameObjects that should be deparented from this object before the original parent is removed.

FPS Camera

The FPS Camera object is parent to all cameras used by the asset, including the main camera and the weapon camera, which is used for rendering the weapon and 2D layer on top of the world and everything else rendered by the main camera.

SmoothMouseLook.cs

This script smoothes mouse input, manages non-recovering recoil/climb from weapons, and applies framerate independent rotation of the FPS Camera object.

Sensitivity : the sensitivity of the mouse input.
Smooth Speed : speed that the smoothed camera position follows mouse input.

CameraKick.cs

Performs final calculation of camera position and angles and is responsible for the independent movement and animation of the main camera angles from the mouse input rotation of the main camera parent. The three hidden variables which are animated manually using keyframe animations to perform camera movement effects are **CameraYawAmt**, **CameraRollAmt**, and, **CameraPitchAmt**. A Quaternion.Slerp to smoothly return camera angles to center and landing gun bounce is also performed in this script.

LevelLoadFade.cs and PainFade.cs

Are used to fade the screen to and from black when loading a level and for red pain fades.

Animation Component

The animation component of the FPS Camera object contains references to all the keyframe animations for camera rotation effects in its Animations array.

FPS Effects

This group is parent to a variety of particle effects used by the prefab. There are no scripts or components needed for this object as it is only for organization. Here are some example particle emitter descriptions:

Debris : opaque debris that emit from impact point and bounce against world colliders
FastSmoke : medium size smoke puff that quickly moves upwards and dissipates
HitSpark : bright circular sparks around hit location with very short duration
SlowSmoke : large puff of smoke that moves upwards slowly and lingers
Sparks : fast, bright sparks that bounce against world colliders
Tracer : a stretched spark effect that shoots from the gun muzzle for tracer effect

FPS Player

The FPS Player object holds the player and movement related components, such as the player's capsule collider, rigidbody, and an audio source for sound effects like jumping, landing, and player voices.

FPSPlayer.cs

Handles player hitpoints, pain and death functions, spawning, weapon pickup mechanics, and stores player control keyCodes.

Hit Points : player hit point amount
Maximum Hit Points : maximum amount of player hit points
Pain Color : color of pain screen flash that can be selected in editor
Pain Screen Kick Amt : magnitude of the screen kicks when player takes damage
Bullet Time Speed : percentage to reduce time to in bullet time mode
Crosshair Enabled : to enable or disable the aiming reticle
Use Swap Reticle : to enable or disable the swap reticle when picking up a weapon when the player already has the maximum weapon amount
Aiming Reticle : the texture used for the aiming crosshair (crosshair)
Pickup Reticle : the texture used for the pickup crosshair (hand)
Swap Reticle : the texture used for the weapon swap crosshair (two round arrows)
No Pickup Reticle : the texture used for the "can't pickup' crosshair (circle with line)
Pain Little : audio clip of player taking light damage
Pain Big : audio clip of player taking damage when low on health
Pain Drown : audio clip of player taking damage underwater
Gasp : audio clip of player gasping when surfacing from dive and low on air
Catch Breath : audio clip of player catching their breath after sprint
Die : audio clip of player death
DieDrown : audio clip of underwater player death
Enter Bullet Time Fx : audio clip of entering bullet time
Exit Bullet Time Fx : audio clip of exiting bullet time
Use Axis Input : if true, the player will use Unity's built in horizontal and vertical axes for movement. This allows the player's movement speed and bobbing effects to be gradually applied as player presses the joystick configured in the input manager.
Player Controls : inspector references to buttons and keys to assign to player actions

HorizontalBob.cs

Controls the horizontal aspect of view and weapon position bobbing. Other scripts pass speed and magnitude amounts to this script to change bobbing behavior.

VerticalBob.cs

Controls the vertical aspect of view and weapon position and angle bobbing. Other scripts pass speed and magnitude amounts to this script to change bobbing behavior. Also plays footstep and ladder climbing sound effects from the audioClip arrays in the Footsteps.cs component.

Ironsights.cs

Calculates and smoothes weapon position and bobbing based on movement states, passes bobbing speed and magnitudes to bobbing scripts, and zooms camera FOV.

Default Fov : default camera field of view value
Sprint Fov : camera field of view value while sprinting

Weapon Cam Fov Diff : amount main camera FOV differs from weapon camera FOV
Zoom Mode : sets zooming button behavior to toggle, hold, or both
Zoom Sensitivity : percentage to reduce mouse sensitivity when zoomed
Sights Up Snd : sound effect for raising sights
Sights Down Snd : sound effect for lowering sights
Walk Horizontal Bob Amount : magnitude of horizontal bobbing while walking
Walk Vertical Bob Amount : magnitude of vertical bobbing while walking
Walk Bob Pitch Amount : magnitude of pitch bobbing while walking
Walk Bob Roll Amount : magnitude of roll bobbing while walking
Walk Bob Speed : speed of bobbing while walking
Crouch Horizontal Bob Amount : magnitude of horizontal bobbing while crouching
Crouch Vertical Bob Amount : magnitude of vertical bobbing while crouching
Crouch Bob Pitch Amount : magnitude of pitch bobbing while crouching
Crouch Bob Roll Amount : magnitude of roll bobbing while crouching
Crouch Bob Speed : speed of bobbing while crouching
Zoom Horizontal Bob Amount : magnitude of horizontal bobbing while zooming
Zoom Vertical Bob Amount : magnitude of vertical bobbing while zooming
Zoom Bob Pitch Amount : magnitude of pitch bobbing while zooming
Zoom Bob Roll Amount : magnitude of roll bobbing while zooming
Zoom Bob Speed : speed of bobbing while zooming
Sprint Horizontal Bob Amount : magnitude of horizontal bobbing while sprinting
Sprint Vertical Bob Amount : magnitude of vertical bobbing while sprinting
Sprint Bob Pitch Amount : magnitude of pitch bobbing while sprinting
Sprint Bob Roll Amount : magnitude of roll bobbing while sprinting
Sprint Bob Speed : speed of bobbing while sprinting

FPSRigidBodyWalker.cs

This script controls the player's rigidbody velocity based on control inputs, jumping, sprinting, swimming, climbing, and crouching states, movement speeds, collisions with platforms and elevators, vertical movement angle limits, falling damage, air manipulation, and grounded calculations.

Walk Speed : movement speed while walking
Sprint Speed : movement speed while sprinting
Sprint Mode : sets sprinting button behavior to toggle, hold, or both
Limited Sprint : set to true if player should run only when they have stamina remaining
Stamina For Sprint : decreases when sprinting and must regenerate
Catch Breath Sound : play sound of player catching breath when stamina is depleted
Jump Speed : movement speed while jumping
Climb Speed : speed that player moves upward when climbing
Lower Gun For Climb : true if player should lower their weapon when climbing
Swim Speed : speed that player moves when diving or swimming on water's surface
Hold Breath Duration : time that player can remain submerged underwater
Backward Speed Percentage : percentage to decrease movement speed while moving backwards
Crouch Speed Percentage : percentage to decrease movement speed while crouching
Strafe Speed Percentage : percentage to decrease movement speed while strafing directly left or right
Zoom Speed Percentage : percentage to decrease movement speed while zooming
Player Height Mod : value to add to player height and radius to make player taller
Crouch Height Percentage : percentage of total height where view will be positioned when crouching
Gravity : additional gravity that is manually applied to the player rigidbody

Slope Limit : the maximum allowed ground surface/normal angle that the player is allowed to climb
Falling Damage Threshold : units that player can fall before taking damage
Anti Bunny Hop Factor : to limit the time between player jumps
Clip Mask : mask for reducing the amount of objects that ray and capsule casts have to check

DragRigidbody.cs

Allows the player to pick up and throw rigidbodies in the game world with additional checks for reach range and prevention of rigidbodies that player has picked up from pushing the player unrealistically.

Reach Distance : max distance to drag objects
Throw Force : physics force to apply to thrown objects
Layers To Drag : objects assigned to layers in this layer mask will be draggable

Footsteps.cs

Plays footstep and landing sound effects during player movement. The FootstepSfx() function of this script is called by VerticalBob.cs and also controls the volume of the footstep sound effects.

Dirt Steps Vol : volume of the footsteps on dirt surfaces
Dirt Steps : array containing dirt footstep sound effects that are selected randomly
Wood Steps Vol : volume of the footsteps on wood surfaces
Wood Steps : array containing wood footstep sound effects that are selected randomly
Metal Steps Vol : volume of the footsteps on metal surfaces
Metal Steps : array containing metal footstep sound effects that are selected randomly
Water Steps Vol : volume of the footsteps on water surfaces or while swimming
Water Steps : array containing water footstep sound effects that are selected randomly
Climb Steps Vol : volume of the footsteps on climbable surfaces
Climb Steps : array containing climbing footstep sound effects that are selected randomly
Dirt Land : sound effect for landing on a dirt surface
Metal Land : sound effect for landing on a metal surface
Wood Land : sound effect for landing on a wood surface
Water Land : sound effect for landing on a water surface

WorldRecenter.cs

Orients all game objects to the scene origin if player travels beyond the threshold to correct floating point precision loss. Having this script enabled allows much larger scenes, but also means that the world coordinates of an object will change when the scene is re-centered, unless set to layer 14, the GUICamera layer, which is ignored by this script because the GUI elements need to stay at the world coordinates (0, 0, 0). It is important to take this information into account if you have a system that requires **permanent/absolute world coordinate references**, because this script will align game object positions to the player's relative position in the scene.

With this script active, it is also important to make sure that large rigidbodies don't overlap too much with other colliders because re-centering the world can sometimes cause these objects to overlap and fly across the scene. Physics joints also need to be **rooted in a parent rigidbody** (can be a kinematic rigidbody) and not the world, so the connected objects do not jitter after a world-center due to the initial position of the world-rooted joint becoming unaligned. It is advisable to test your scene to make sure the colliders stay in position after one or more world re-centers.

Threshold : re-center objects if player moves farther than this distance from scene origin
Refresh Terrain : true/checked if terrain should be refreshed when re-centering world to update terrain tree colliders (can cause momentary hiccup on large terrains)

FPS Weapons

The FPS Weapons object is the parent of the separate weapon objects and controls weapon fire, switching, viewmodel animation, particle effects, swaying, reloading, and weapon related camera animations.

PlayerWeapons.cs

This script initializes weapons on level load, calculates weapon switching, handles backup weapons, drops weapons, plays switching sound effects, and syncs the weapon object's position with the main camera's position.

First Weapon : the Weapon Order index of the first weapon that will be selected when the map loads
Backup Weapon : the Weapon Order index of the weapon that will be treated as the player's non-droppable backup weapon (like fists or a melee weapon).
Max Weapons : the maximum amount of weapons the player can carry at one time.
Weapon Order : array for storing order of weapons. This array is created in the inspector by dragging and dropping weapons from under the FPSWeapons branch in the FPS Prefab. Weapon 0 should always be the unarmed/null weapon.
Changesnd : audio clip that plays when switching weapons.

GunSway.cs

Handles weapon sway and weapon angle bobbing.

Sway Amount : percentage of weapon position sway
Roll Sway Amount : percentage of weapon roll sway
Walk Bob Yaw Amount : percentage of weapon yaw bobbing while walking
Walk Bob Roll Amount : percentage of weapon roll bobbing while walking
Sprint Bob Yaw Amount : percentage of weapon yaw bobbing while sprinting
Sprint Bob Roll Amount : percentage of weapon roll bobbing while sprinting

WeaponEffects.cs

This script organizes weapon effects and contains functions to play weapon impact audioClips, emit weapon impact particle effects, and instantiate bullet mark objects in the scene based on the tag assigned to the hit game object (Dirt, Metal, Wood, Flesh, Water, Glass).

Dirt Impact : the game object containing particle effects for dirt impacts
Metal Impact : the game object containing particle effects for metal impacts
Wood Impact : the game object containing particle effects for wood impacts
Water Impact : the game object containing particle effects for water impacts
Glass Impact : the game object containing particle effects for glass impacts
Flesh Impact : the game object containing particle effects for flesh impacts
Dirt Impact Melee : the game object containing particle effects for melee dirt impacts
Metal Impact Melee : the game object containing particle effects for melee metal impacts
Wood Impact Melee : the game object containing particle effects for melee wood impacts
Tracer Particles : the particle emitter for weapon tracer particles

Bubble Particles : the particle emitter for underwater weapon tracer particles
Dirt Marks : an array containing object references to use for dirt impact marks
Metal Marks : an array containing object references to use for metal impact marks
Wood Marks : an array containing object references to use for wood impact marks
Glass Marks : an array containing object references to use for glass impact marks
Dirt Marks Melee : an array containing object references to use for dirt melee impact marks
Metal Marks Melee : an array containing object references to use for metal melee impact marks
Wood Marks Melee : an array containing references to objects to use for wood melee impact marks
Default Impact Sounds : an array containing references to audioClips to play for weapon impacts with the default/dirt surface type
Metal Impact Sounds : an array containing references to audioClips to play for weapon impacts with the metal surface type
Wood Impact Sounds : an array containing references to audioClips to play for weapon impacts with the wood surface type
Water Impact Sounds : an array containing references to audioClips to play for weapon impacts with the water surface type
Glass Impact Sounds : an array containing references to audioClips to play for weapon impacts with the glass surface type
Flesh Impact Sounds : an array containing references to audioClips to play for weapon impacts with the flesh surface type
Default Impact Sounds Melee : an array containing references to audioClips to play for weapon impacts with the default/dirt surface type
Metal Impact Sounds Melee: an array containing references to audioClips to play for weapon impacts with the metal surface type
Wood Impact Sounds Melee : an array containing references to audioClips to play for melee weapon impacts with the wood surface type
Flesh Impact Sounds Melee : an array containing references to audioClips to play for melee weapon impacts with the flesh surface type

WeaponBehavior.cs

This is a comprehensive script that defines most weapon actions such as weapon timers, firing and reloading states, ammo management, shell ejection, weapon effects, weapon animation, view kicks and camera animations, weapon position fine tuning, and weapon sound effects.

Have Weapon : true if player has this weapon in their inventory
Weapon Mesh : the weapon viewmodel mesh to animate
Weapon Drop Obj : reference to weapon pickup object to drop for this weapon
Show Aiming Crosshair : true if aiming crosshair should be used with this weapon
Droppable : true if this weapon can be dropped by pressing drop key, or swapped for weapon pickup on the ground
Adds To Total Weaps : true if this weapon should be counted toward the weapon total (the backup weapon and special weapons may have this set to false)
Weapon Unzoom X Position : horizontal modifier of gun position when not zoomed
Weapon Unzoom Y Position : vertical modifier of gun position when not zoomed
Weapon Unzoom X Position Sprint : horizontal modifier of gun position when sprinting
Weapon Unzoom Y Position : vertical modifier of gun position when sprinting
Weapon Zoom X Position : horizontal modifier of gun position when zoomed
Weapon Zoom Y Position : vertical modifier of gun position when zoomed
Can Zoom : true if this weapon can zoom and use iron sights

Zoom FOV : FOV amount that this weapon zooms to when using ironsights
Sway Amount Unzoomed : sway amount for this weapon when not zoomed
Sway Amount Zoomed: sway amount for this weapon when zoomed

Shooting

Pistol Sprint Anim : set to true to use alternate sprinting animation with pistols
Sprint Bob Amount X : to fine tune horizontal weapon sprint bobbing amounts
Sprint Bob Amount Y : to fine tune vertical weapon sprint bobbing amounts
Projectile Count : amount of projectiles to be fired per shot (> 1 for shotguns)
Fire Mode Selectable : true if weapon can switch between burst and semi-auto
Semi Auto : true when weapon is in semi-auto mode
Fireable Underwater : true if this weapon can be fired underwater
Unarmed : should this weapon be null/unarmed?
Melee Swing Delay : this weapon will be treated as a melee weapon when this value is > 0
Range : range that weapon can hit targets
Fire Rate : minimum time between shots
Fire Anim Speed : speed to play the weapon firing animation
Force : amount of physics push to apply to rigidbodies on contact
Damage : damage to inflict on objects with ApplyDamage(); function
Bullet Mask : only layers to include in bullet hit detection (for efficiency)

Ammo and Reloading

Bullets Per Clip : maximum number of bullets per magazine
Bullets To Reload : number of bullets to reload (single bullets or full magazines)
Bullets Left : bullets left in magazine
Ammo : ammo amount for this weapon in player's inventory
Max Ammo : maximum ammo amount player's inventory can hold for this weapon
Reload Time : time per reload cycle
Reload Anim Speed : speed of reload animation playback
Shell Rld Anim Speed : speed of single shell reload animation playback
Ready Anim Speed : speed of ready animation playback
Ready Time : amount of time needed to finish the ready anim

Effects

Muzzle Flash : the game object that will be used as a muzzle flash
Barrel Smoke Particles : reference to particle emitter for barrel smoke effect
Barrel Smoke Shots : number of consecutive shots required to emit barrel smoke
Muzzle Smoke Particles : reference to particle emitter for muzzle smoke effect
Muzzle Smoke Alpha : opacity of muzzle smoke particles
Muzzle Light Obj : object with light component used for muzzle light effect
Muzzle Light Delay : time to wait until the muzzle light starts fading out
Muzzle Light Reduction : rate of muzzle light fading

Recoil

Shot Spread : defines accuracy cone of fired bullets
Kick Up : amount to kick view up when firing
Kick Side : amount to kick view sideways when firing
Use View Climb : true if weapon should add non-recovering recoil to view angles/input
View Climb Up : vertical amount of non-recovering view recoil
View Climb Side : horizontal amount of non-recovering view recoil
View Climb Right : amount the non-recovering view recoil moves to the right per shot
Use Recoil Increase : true if weapon accuracy should decrease with sustained fire
Shots Before Recoil : number of shots before weapon recoil increases with sustained fire
View Kick Increase : amount of sustained fire recoil for view angles/input (smaller values result in more recoil)

Aim Dir Recoil Increase : amount of sustained fire recoil for weapon accuracy (smaller values result in more recoil)

Shell Ejection

Shell Prefab RB : the rigidbody-only game object to use as empty shell casing and eject from shellEjectPosition
Shell Prefab Mesh : the mesh-only game object to use as empty shell casing that ejects from shellEjectPosition, and lerps to Shell Prefab RB's position and rotation
Shell Eject Direction : direction of ejected shell casing
Shell Scale : can be used to make different shapes and sizes of shells from one model
Shell Eject Delay : delay before ejecting shell (used for bolt action rifles and pump shotguns)
Shell Force : overall movement force of ejected shell
Shell Up : random vertical direction to apply to shellForce
Shell Side : random horizontal direction to apply to shellForce
Shell Forward : random forward direction to apply to shellForce
Shell Rotate Up : amount of vertical shell rotation
Shell Rotate Side : amount of horizontal shell rotation
Shell Duration : time in seconds that shells persist in the world before being removed

Audio clips used for weapon sound effects

Fire Snd : sound of weapon firing
Reload Snd : reloading audio clip that is synchronized with weapon animation
Reload Last Snd : usually shell reload sound + shotgun pump or rifle chambering sound
No Ammo Snd : dry fire of gun when out of ammo
Ready Snd : played when weapon is selected

ShellEjection.cs

This script is attached to the bullet shell objects that are instantiated by WeaponBehavior.cs and controls the movement of the ejected shell, the sound effects played when the shell hits the ground or other surface, and the removal of the shell after the shellDuration time of WeaponBehavior.cs has elapsed.

Shell Sounds : an array of audio clip references to play when shell contacts surface

NPC Objects

Most of the scripts involved with managing NPC behavior are attached to a root NPC object that also has a Character Controller, an Audio Source, and sometimes an Animation Component. The child/children of this NPC root object is usually the NPC character mesh which has been dragged over the NPC root object from the Project Library.

AI.js

This script controls most of the NPC's behavior including movement speed, target acquisition, waypoint path patrolling, initialization, and character animation.

Object With Anims : if set to nothing, the Animation Component attached to this object (NPC root) will be used for animations. If the whole character mesh object has been made a child of the NPC root (not just the skeleton and skinned mesh) dragging that child object to this field will make this script use the child object's Animation Component and avoid having to set up the NPC root from scratch on the child mesh object.
Walk Anim Speed : speed to play the NPC's walking animation
Run Anim Speed : speed to play the NPC's running animation

Speed : movement speed of the NPC
Push Power : physics force to apply to rigidbodies in the NPC's movement path
Rotation Speed : speed that the NPC rotates when changing direction during a patrol
Target Player : if false, NPC will ignore player and patrol waypoints or stand watch
Shoot Range : range to player that the NPC will need to reach before attacking
Attack Range : range to player that NPC will stop patrolling or idling and pursue player
Sneak Range Mod : percentage to decrease the NPC's attack range when player is crouching to allow for player sneaking past enemies
Don't Come Closer Range : NPC will not attempt to move closer than this range to player
Delay Shoot Time : time between NPC attacks
Do Patrol : if true, NPC will patrol waypoints, if false, NPC will stand guard
Walk On Patrol : if true, NPC will walk on patrol, if false, NPC will run
My Waypoint Group : waypoint group number that this NPC should follow
Search Mask : layers that this NPC should check in its target detection raycast
Eye Height : height to add to origin of target detection raycast (used if the NPC character mesh or root origin is at their feet and needs to be raised to eye height)
Random Spawn Chance : chance between 1 and 0 that NPC will be spawned at runtime

CharacterDamage.js

Any calls or messages to this script's ApplyDamage(damage) function, will subtract from this NPC's hitpoints, play death sound effects, replace this NPC object with the Dead Replacement (usually a ragdoll), and remove the NPC root when this NPC's hitpoints are depleted.

Hit Points : NPC dies when this value reaches zero
Dead Replacement : the object from the Project Library that will be instantiated in the NPC root's position on death (usually a ragdoll)
Die Sound : audio clip to play when NPC dies
Remove Body : if true, this NPC's Dead Replacement object will be removed after a time
Body Stay Time : time NPC's Dead Replacement object remains in scene after death

NPCAttack.js

Manages attack behavior of this NPC, such as attack range, physics force applied by attacks, muzzle flash effects, and sound effects.

Range : max range of this NPC's attack
Fire Rate : rate of fire for this NPC's attacks
Force : physics force to apply to rigidbodies hit by NPC's attacks
Damage : damage inflicted to other objects by this NPC's attacks
Bullets Per Clip : amount of bullets per clip for this NPC's weapon
Ammo : ammo for this NPC's weapon
Reload Time : time required to reload this NPC's weapon
Muzzle Flash : object reference for the muzzle flash effect of this NPC's weapon
Firesnd : audioClip of the firing sound for this NPC's attack
Realoadsnd : audioClip of the reloading sound of this NPC's weapon
Noammosnd : audioClip of the out of ammo sound effect for this NPC's weapon

RemoveBody.js

This script is attached to the Dead Replacement object of this NPC's AI component and removes the NPC's body from the scene after the Body Stay Time value passed from the AI component has elapsed.

AutoWayPoint.js

This script is used by attaching it to a parent waypoint game object with a child cube object that has the EditorOnly tag. These waypoint objects can be arranged in a path that can be referenced by the “My Waypoint Group” value of an NPC’s AI component to have the NPC patrol waypoint paths.

Waypoint Group : the number of the waypoint group that this waypoint belongs to
Waypoint Number : the order number of this waypoint in the waypoint group (the first waypoint to be followed should be set to “1” the second to “2” and so on)

MonsterItemTrap.cs

This script can be attached to any object with the “Usable” tag that is assigned to layer 9, the Ragdolls and Pickups layer. When the player uses this item, this script’s ActivateObject(); function is called and the NPC’s in the NPC’s To Trigger array will be activated.

Npcs To Trigger : NPC objects in this array will be deactivated on level load, and reactivated again when the player uses the trapped item. This array should not have any empty spaces between NPC objects.

MonsterTrigger.cs

This script can be attached to an object with a box collider (or other type of collider) that has Is Trigger set to true. When the player enters the trigger, the NPC’s in the NPC’s To Trigger array will be activated.

Npcs To Trigger : NPC objects in this array will be deactivated on level load, and reactivated again when the player enters the trigger. This array should not have any empty spaces between NPC objects.

Effects

Several scripts are included with the asset which manage visual effects such as the fade in and fade out when loading a game and the fading of bullet mark objects in the scene.

FadeOutDecals.cs

This script fades the opacity of the bullet mark objects made by weapons after a time.

Fade Out Decals : seconds that the bullet mark decals should stay before fading

LevelLoadFade.cs

This script fades in or out the screen to a color based on the true or false value of the fadeIn argument called with this script’s FadeAndLoadLevel(color, fadeLength, fadeIn) function.

FadeAndLoadLevel(color, fadeLength, fadeIn) : Function Argument Descriptions
color : the color to fade in or out the screen to
fadeLength : seconds to perform the screen fade in or out
fadeIn : if true, screen will fade in from color, if false, screen will fade out to color and restart scene

PainFade.cs

This script fades in and out the screen to a color using this script's FadeIn(color, fadeLength) function for effects like flashing the screen red when player is damaged.

FadeIn(color, fadeLength) : Function Argument Descriptions
color : the color to fade in and out the screen to
fadeLength : seconds to perform the screen fade in or out

HUD and GUI objects

HUD and GUI elements are rendered on the screen using a variety of scripts and game objects with either GUIText or GUITexture components. The GUIText objects have a main color rendering object as the parent and a child object for rendering the GUIText shadow with slightly skewed Horizontal Offset and Vertical Offset values of their script than the object rendering the main color text.

AmmoText.cs

This script displays the total ammo amount and ammo amount remaining in weapon clip on screen for player weapons.

Ammo Gui : the ammo amount remaining in the weapon's clip
Ammo Gui 2 : the total ammo amount for this weapon
Text Color : the color to render the text of the GUIText component
Horizontal Offset : the offset from the screen width to position the GUIText
Vertical Offset : the offset from the screen height to position the GUIText

HealthText.cs

This script displays the player health amount on the screen.

Health Gui : the player's health amount
Text Color : the color to render the text of the GUIText component
Horizontal Offset : the offset from the screen width to position the GUIText
Vertical Offset : the offset from the screen height to position the GUIText

HelpText.cs

This script displays the "Press F1 for controls" message at the start of the game and handles the fading and display of this message and default button list at runtime.

Text Color : the color to render the text of the GUIText component
Horizontal Offset : the offset from the screen width to position the GUIText
Vertical Offset : the offset from the screen height to position the GUIText

Item Pickup Objects

Ammo, health, and weapon pickup objects are instantiated or dragged from the project library into the scene and can be used by the player. The pickup objects should have a box collider component, have the tag “Usable” set, and assigned to layer 9, the Ragdolls and Pickups layer, so the item detection rayCast in FPSPlayer.cs will detect and call the PickUpItem() function in the pickup script.

AmmoPickup.cs

This script is attached to ammo pickup objects and calculates the ammo to add to the player’s inventory, plays item sound effects, and determines if this pickup should be removed after use.

Weapon Object : the number of the weapon object reference in the Weapon Order array of the Player Weapons component that uses this ammo type
Remove On Use : if true, this pickup will be removed on use, if false, pickup will stay after use (for armories or weapon crate style pickups)
Pickup Sound : audioClip to play when this item is picked up
Full Sound : audioClip to play when this item can’t be picked up due to player having max ammo
Ammo To Add : the ammo amount to add when this item is used (usually a multiple of the bulletsPerclip amount, but can be set to any number)
Ammo Pickup Reticle : custom crosshair texture to display when player points at this item

WeaponPickup.cs

This script is attached to weapon pickup objects and plays item sound effects, updates dropWillDupe and haveWeapon vars of WeaponBehavior.cs, and determines if this pickup should be removed after use.

Weapon Object : the number of the weapon object reference in the Weapon Order array of the Player Weapons component that corresponds to this weapon pickup
Remove On Use : if true, this pickup will be removed on use, if false, pickup will stay after use (for armories or weapon crate style pickups)
Pickup Sound : audioClip to play when this item is picked up
Full Sound : audioClip to play when this item can’t be picked up due to player having this weapon already and having max ammo for this weapon
Weapon Pickup Reticle : custom crosshair texture to display when player points at this item

HealthPickup.cs

This script is attached to health pickup objects and plays item sound effects, updates hitPoints var of FPSPlayer.cs, and determines if this pickup should be removed after use.

Health To Add : the amount of health to add when player uses this item
Remove On Use : if true, this pickup will be removed on use, if false, pickup will stay after use (for armories or weapon crate style pickups)
Pickup Sound : audioClip to play when this item is picked up
Full Sound : audioClip to play when this item can’t be picked up due to player having max health
Weapon Pickup Reticle : custom crosshair texture to display when player points at this item

Interactive Objects

These scripts are used for water zones and climbable objects and all apply forces to or affect the player's rigidbody velocity in some way.

Ladder.cs

This script can be attached to a game object with a box collider set to Is Trigger, to create climbable surfaces. This script manages the climbable var in the FPSRigidBodyWalker.cs script and resets other movement vars when player exits the ladder trigger.

Play Climbing Audio : true if climbing sounds should be played for this ladder

EdgeClimbTrigger.cs

A basic ledge climbing script that can be placed on a trigger collider near an edge to have the player climb themselves up when they enter the trigger. The edge climbing trigger is deactivated when the player exits it to prevent them from hovering when walking over the trigger from above.

Upward Pull Force : the upward force applied to the player when they enter the trigger

EdgeClimbTriggerActivate.cs

This script reactivates an edge climbing trigger. This script should be placed on a trigger collider where the player will be in position to climb up the edge climbing trigger stored in the Trigger To Activate reference.

Trigger To Activate : reference to a trigger box collider with an EdgeClimbTrigger.cs component to reactivate when player is in position for climbing and enters this trigger

WaterZone.cs

This script can be attached to a cube trigger collider to treat the volume of the collider as a swimmable liquid. This script manages audioSources, particle emitters, and render settings to create underwater and surface effects.

Underwater Sound : sound effect to player when player is submerged
Above Water Audio Sources : an array for above water audio sources that should be paused when the player is submerged and the underwater sound is playing
Flip Water Plane : true if water plane object should be flipped when player is submerged so they can see the surface from underwater
Water Plane : the water plane object that can be flipped when player dives
Ripple Effect : the particle emitter for ripples around player when treading water
Particles Effect : the particle emitter for underwater ambient bubbles
Water Splash : the particle emitter group for splashes on the water's surface
Splash Trail : the particle emitter object for splashes around player while swimming
Underwater Fog Enabled : true if fog effect should be used underwater
Underwater Fog Mode : the fog mode for underwater fog
Underwater Fog Color : the color of underwater fog
Underwater Light Color : the color of the scene's ambient light when underwater
Underwater Fog Density : the density of the underwater exponential fog
Underwater Linear Fog Start : the start range of the underwater linear fog
Underwater Linear Fog End : the end range of the underwater linear fog

InstantDeathCollider.cs

When this script is attached to a trigger collider, the player, NPC, or game object with collider component will be instantly killed and/or removed from scene. This script can be used to prevent a player or objects from falling indefinitely out of the map or scene.

Platforms and Elevators

Platforms and elevators are rideable game objects that loop from their starting point to their end point.

MovingElevator.cs

This script is attached to a game object with a collider to make it a rideable elevator that cycles vertically from its initial position to the position of the Point B transform reference.

Point B : the upper extent of the elevator's travel path which is defined by the transform of a duplicated dummy object
Speed : the speed the elevator travels from its initial position to the Point B position

MovingPlatform.cs

This script is attached to a game object with a collider to make it a rideable platform that cycles horizontally from the Point A transform position reference to the Point B transform position reference.

Point B : the far horizontal extent of the platform's travel path which is defined by the transform of a duplicated dummy object
Point A : the initial horizontal extent of the platform's travel path which is defined by the transform of a duplicated dummy object
Speed : the speed the platform travels from the Point A position to the Point B position

ElevatorCrushCollider.cs

This script can be attached to a trigger collider underneath an elevator to make the elevator crush the player if they are trapped underneath it.

Squish Snd : the sound effect of the player getting crushed by the elevator

Destructible Objects

These scripts are attached to objects that can take damage and be destroyed by the player's weapons or are triggered when the player enters the object's detection radius.

BreakableObject.cs

When attached to an object with a box collider and mesh particle emitter components, this script plays breaking sound effects using attached audio source component and its current audio clip, emits debris particles from the attached particle emitter component, and removes the object from the scene when its hit points have been depleted.

Hit Points : the amount of damage this object can receive before being destroyed

ExplosiveObject.cs

When attached to an object with a collider, this script plays explosion sound effects, emits explosion particles, applies damage and physics force to nearby objects, and removes the object from the scene when its hit points have been depleted.

Hit Points : the amount of damage this object can receive before detonating
Explosion Damage : the damage to inflict at center of the explosion (falls off from center)
Damage Delay : delay before this object applies damage and force to surrounding objects
Blast Force : physics force to apply to surrounding objects
Explosion Effect : reference to a game object containing explosion effect particle emitters
Explosion Height : height above or below object center to emit explosion effects
Shockwave Height : height above or below object center to emit shockwave effects
Radius : radius from center of object where damage and force is applied to nearby objects
Blast Mask : layers to check for objects to apply explosive damage and force

MineExplosion.cs

This script aligns the parent mine object to the ground normal so they don’t have to be perfectly aligned with the ground in the editor, detects objects entering the mine trigger radius, plays explosion sound effects, emits explosion particles, applies damage and physics force to nearby objects, and removes the object from the scene when it has been damaged or triggered.

Explosion Damage : the damage to inflict at center of the explosion (falls off from center)
Damage Delay : delay before this object applies damage and force to surrounding objects
Blast Force : physics force to apply to surrounding objects
Explosion Effect : reference to a game object containing explosion effect particle emitters
Explosion FX : sound effect of mine explosion
Beep FX : sound effect of mine being triggered by nearby object
Is Radius Collider : true if this is the child object with a sphere collider attached which detects collisions, false if this is the parent mine object, whose collider surrounds the landmine mechanism itself and detects when it is shot or damaged
Blast Mask : layers to check for objects to apply explosive damage and force
Init Pos Mask : layers to check for objects to align the mine object to when scene is loaded

Support
Azuline Studios is dedicated to providing support for our product to help you make great games!

We may be reached here:

- The Realistic FPS Prefab thread at the Unity3D Community Forums
- Send a private message to Azuline Studios through the Unity3D Community
- Or by emailing AzulineStudios at gmail dot com

Credits
Code, modeling, animation, and level design by Azuline Studios© All Rights Reserved

Thanks to: Millenia, Sargent99, Benderexable, Kamiomi, Pbcrazy, Angryfly, druggon, 3D4Ds, syntheno, Unity3D, 3dheaven, Peacemaker, mp6arms, pixelhouse (http://www.pixelhouse.com.ar) for Creative Commons Licensed models

Thank you for your purchase!