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, and jumping movement states with seamless transitions, easily configurable speeds, and air manipulation. Player health management with damage from falling, 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 auto and semi auto modes, shotguns, scoped weapons, accuracy handling, single shell reloading, and melee weapons by modifying the variables of one script instance from the editor - no additional scripting required to make a new weapon.

Item pickups to place in your scene for ammo, health, and weapons that can either be static or moveable (and throwable) rigidbodies.

Weapon effects like bullet impacts, 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 explosives, bows and arrows, and improved platform riding mechanics are possible in the future. Follow our progress at http://azulinestudios.blogspot.com

Version Notes

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 without any errors.
-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,

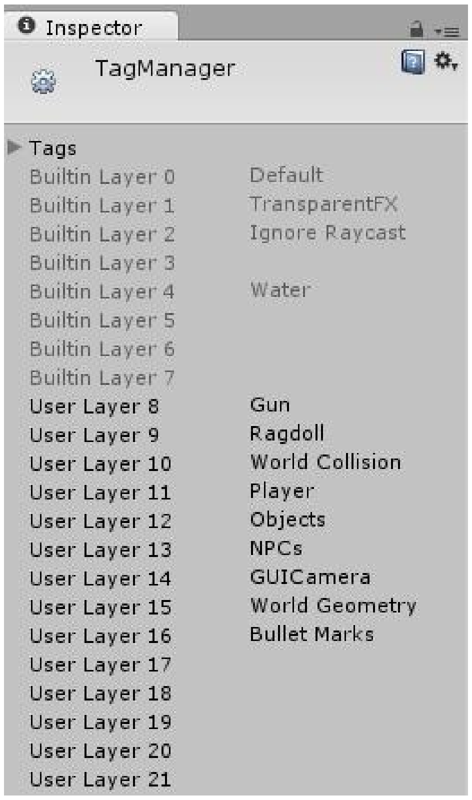
There are a few important pieces of information to take into consideration when importing the FPS Prefab into other projects.

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, the asset has been developed using a Fixed Time Step of 0.01, which is set automatically, along with other physics settings, in the Start() function of the FPSPlayer script. Larger Fixed Time Step values are compatible with this asset, but certain movements of rigidbodies like that of bullet casings and walking on platforms can appear to be choppy. This might be fixed in a future update, but currently movement is smooth with a 0.01 Fixed Time Step, which is a good compromise between the default setting of 0.02.

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 16.



Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	Gun
User Layer 9	Ragdoll
User Layer 10	World Collision
User Layer 11	Player
User Layer 12	Objects
User Layer 13	NPCs
User Layer 14	GUICamera
User Layer 15	World Geometry
User Layer 16	Bullet Marks
User Layer 17	
User Layer 18	
User Layer 19	
User Layer 20	
User Layer 21	

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 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

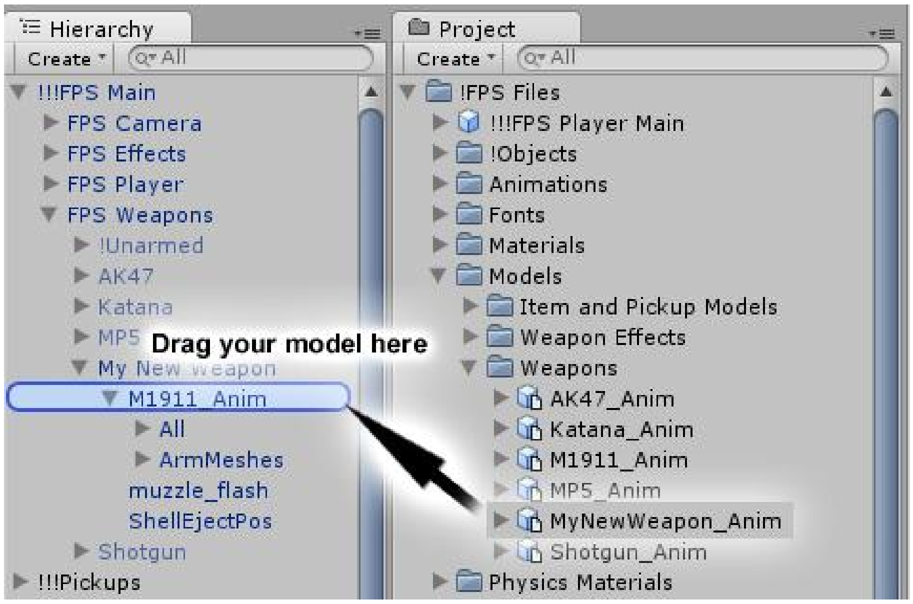
Adding a new weapon to the Realistic FPS Prefab takes only a few steps and requires that you have a weapon model asset imported to your Project Library. The weapon model should have animations to make it move during firing and reloading, though it's possible to add simple weapon animations to a non-animated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

Once you have a weapon model, expand the *!!!FPS 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.

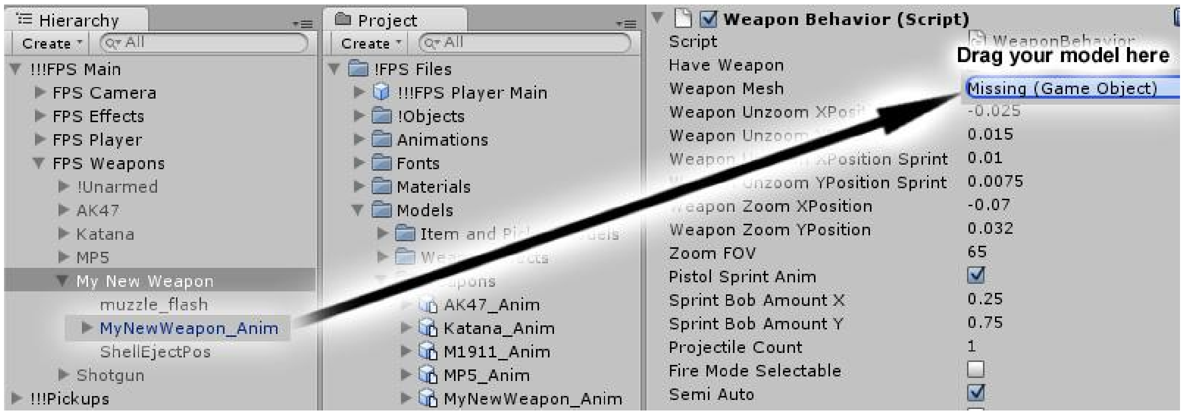
Then, if the other weapon objects are active and their view models are visible and overlapping in the 3D Scene window, hold control and click on all weapon objects except the one you duplicated, disable them by clicking the active check box in the upper left of the inspector window, and select the "Deactivate 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.

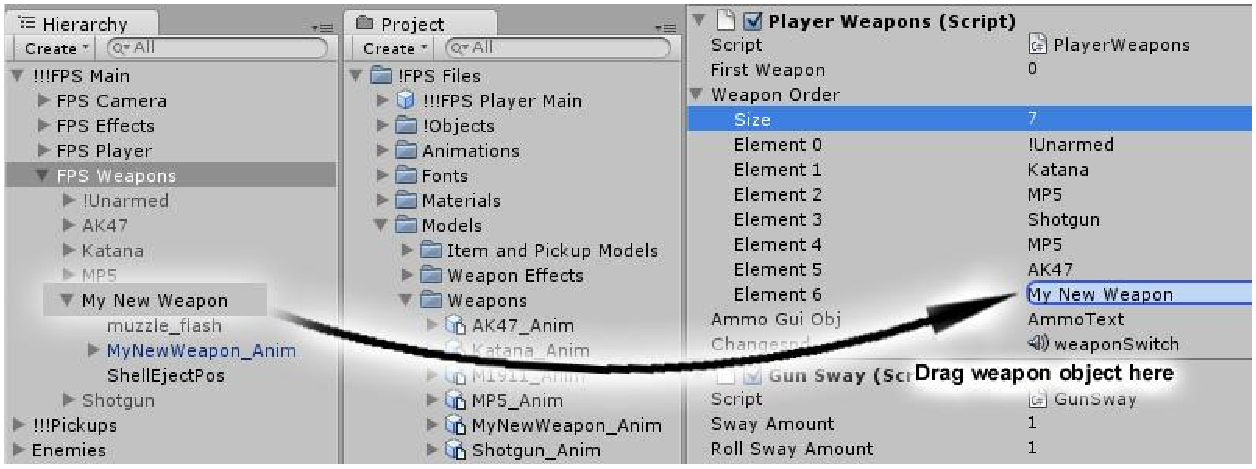


Weapon Behavior (Script)

Script	WeaponBehavior
Have Weapon	
Weapon Mesh	Missing (Game Object)
Weapon Unzoom XPos	-0.025
Weapon Unzoom	0.015
Weapon XPosition Sprint	0.01
Unzoom YPosition Sprint	0.0075
eapon Zoom XPosition	-0.07
Weapon Zoom YPosition	0.032
Zoom FOV	65
Pistol Sprint Anim	✓
Sprint Bob Amount X	0.25
Sprint Bob Amount Y	0.75
Projectile Count	1
Fire Mode Selectable	☐
Semi Auto	✓

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.



Player Weapons (Script)	
Script	PlayerWeapons
First Weapon	0
Weapon Order	
Size	7
Element 0	!Unarmed
Element 1	Katana
Element 2	MP5
Element 3	Shotgun
Element 4	MP5
Element 5	AK47
Element 6	My New Weapon
Ammo Gui Obj	AmmoText
Changesnd	weaponSwitch
Gun Sway (Script)	
Script	GunSway
Sway Amount	1
Roll Sway Amount	1

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.

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). 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.

Overview

The *!!!FPS 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 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 takes samples of mouse input over a defined period of time and finds the average to achieve 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 are 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.

FPS Effects

This group is parent to all particle effects used by the prefab. There are no scripts or components needed for this object as it is only for organization.

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 most of 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, physics initialization, and weapon pickup mechanics.

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
Crosshair Enabled : to enable or disable the aiming reticle
Reticle : the texture used for the aiming crosshair
Hand : the texture used for the pick up crosshair
Pain Little : audio clip of player taking light damage
Pain Big : audio clip of player taking damage when low on health
Die : audio clip of player death

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 **Walk Sounds** and **Climb Sounds** arrays.

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.

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, climbing, and crouching states, movement speeds, collisions with platforms and elevators, vertical movement angle limits, falling damage, air manipulation, and grounded calculations.

Run Speed : movement speed while running
Walk Speed : movement speed while walking
Jump Speed : movement speed while jumping
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
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
Climb Speed : speed that player moves upward when climbing
Anti Bunny Hop Factor : to limit the time between player jumps
Landfx : audio clip of player landing from height
Jumpfx : audio clip of player jumping
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.

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, 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
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

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 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
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
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
Shot Spread : defines accuracy cone of fired bullets
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
Kick Up : amount to kick view up when firing
Kick Side : amount to kick view sideways when firing
Shell Prefab : game object to use as empty shell casing and eject from shellEjectPosition
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
Spark Particles
Hit Spark
Tracer Particles
Slow Smoke Particles
Muzzle Smoke Particles : smoke that rises from muzzle after firing
Fast Smoke Particles
Debris Particles
Bullet Mark Obj : Bullet marks that are instantiated at bullet hit locations

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
Hit Sounds : sound of bullets hitting surfaces

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, for Creative Commons Licensed models

Thank you for your purchase!