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. The asset has been designed with a focus on keeping scripts simple and ensuring that motion and effects are smooth and high quality. 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, swim, 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, explosives, enemy fire, and elevators is also included.

Advanced weapon positioning using sine bobs, iron sights, weapon sway, deadzone aiming, customizable 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. An early version of third person mode is also available.

Weapon behavior customization allows switchable fire modes, droppable weapons, grenades, bow and arrows, offhand melee attacks, shotguns, scoped weapons, accuracy and recoil settings, single shell reloading, and melee weapons all by modifying the variables of one script from the editor - no scripting required to make or edit a 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.

NPC AI with navmesh pathfinding, a wave spawning system, and faction awareness. NPCs can recognize friends and enemies and track targets throughout the level. Friendly NPCs can follow the player and be directed to move to specific positions.

Ease of use through inspector tooltips and commented code, simple drag and drop functionality of game objects into scenes, and customizable variables accessible from the inspector.

Are there any more features planned?

Yes, we plan to continue adding features and improving the asset however we can. The addition of multiplayer, mobile touch screen controls, and vehicles are possible in the future.
Follow our progress at http://azulinestudios.blogspot.com

Contents

Version Notes

1.45

-Fixed underwater surface shader.

-Added NpcDamage var to DamageZone.cs to gradually apply damage to NPCs while in damage trigger.

-Fixed origin of bubble particles firing effect.

-Fixed player character rigging for new Unity versions.

-Removed legacy image effects (except sun shafts) and replaced with Unity Post Processing Stack.

-Fixed bullet shell rotations.

-Fixed fired arrow velocity.

1.44

-Fixed missing Project Settings in uploaded versions.

1.43

-Fixed AI initialization error in Unity 2017.2.

-Removed HUD GUITextures and GUITexts and replaced them with UI Canvas.

-Fixed stuck arrows becoming unaligned in Unity 2017.2.

-Added unarmed player character animations.

-Added allowFirstPerson var to CameraControl.cs for third person only games.

1.42

-Removed unused animation events for Soldier@walk animation in Unity 5.2 package version.

-Fixed player model in third person mode not playing shooting animations when strafing, running, jumping, and firing.

1.41

-Fixed spinning of weapon model roll angles when setting up new weapons with certain models.

-Raised third person camera height when zoomed and swimming to better match camera height when not swimming and zoomed.

-The third person camera sphere cast now ignores objects with the "Usable" tag to prevent arrows from blocking camera.

-Added extra check to PlayerWeapons.cs to prevent weapon switching if toggle camera button is being held.

-Added fpTorsoAlwaysAims var to PlayerCharacter.cs to allow player model to always aim in first person mode (shadow will always aim in look direction).

-Fixed incorrect weapon rotation if Remove Prefab Root was unchecked.

-Added new tutorial for changing the player model to documentation.

1.4

-Removed VisibleBody.cs and replaced with PlayerCharacter.cs, which animates player character model in third person and creates and animates first person visible body model.

-Added ThirdPersonWeapons.cs script which holds assets used for third person weapons.

-Updated animations to use Mecanim and removed all legacy animations.

-Removed HorizontalBob.cs and VerticalBob.cs and replaced bobbing effects with keyframe animations.

-Removed Perlin Noise animation of weapon models since the same effect can be achieved with new keyframe animation bobbing.

-Allowed instant kill triggers to kill invulnerable player.

-Fixed fire on release weapons not canceling after hiding weapon when climbing or holding objects.

-Can no longer drop infinite weapons in third person mode.

-Fixed frame rate dependence of pivot pitch angles.

-Fixed muzzle flash not rotating roll angle randomly.

-Added stepHeight and maxStepHeight variables to FPSRigidBodyWalker.cs to allow player controller to automatically detect steps and move up them without needing invisible collider ramps over steps (though they can still be used).

-Forward obstacle detection capsule cast now checks in move direction.

-Fixed forward obstacle detection capsule cast when crouching.

-Fixed offhand grenade not throwing if ammunition was depleted.

1.27

-Fixed missing particle system error in FPSPlayer.cs when blocking attacks.

-Increased tracer particle effect speed.

1.26

-Fixed delay of NPC movement animation when they first move from stationary position.

-Fixed gunsmoke not playing after being interrupted by offhand throw.

-Fixed obsolete navmesh agent calls in Unity 5.6.

-Improved frame rate independence of vertical weapon bobbing animation.

1.25

-All legacy particle emitters replaced with Shuriken particle systems.

-Removed World Collision layer for simpler setup of scenes. Player and weapons just interact with the Default layer now.

-Added On Die event to CharacterDamage.cs for the RFPSP Save System by Pixel Crushers.

-Added ApplyDamageEmerald function to FPSPlayer.cs for compatibility with Emerald AI by Black Horizon Studios.

-Added input dead zone and smoothing to joystick control when sprinting and strafing to prevent excessive zig-zagging.

-Fixed reload animation not playing if hiding weapon/climbing interrupts automatic reload.

-Added audio engine check to allow weapon fire if audio engine is not running.

-Fixed AI hunting behavior.

-Added crouchWalkYposition and weaponWalkYposition to WeaponBehavior.cs to allow customizing of vertical weapon position when walking or crouching.

-Made vertical pivot bob scale properly with joystick input.

-Clamped Joystick and WASD input to prevent exceeding of maximum movement speed.

-Added subtle Perlin Noise to weapon and camera positions for more natural variation of bob cycle.

-Replaced unneeded sphere cast for AI line of sight check with ray cast for increased performance.

-Reduced pivot bobbing when moving and deadzone-aiming.

-Added a jumping and landing animation to weapons.

-Reduced complexity of tree shadows in demo scene for performance gain.

-Fixed weapon smoke position when bullet time is active and player starts sprinting.

-Spotlights from weapons no longer shine on weapon models.

-Smoothed zooming transition to prevent jerky movement with rapid zooming/unzooming.

-Improved smoothing of shield blocking transition.

-Improved rotation of camera when dead zone/free-aiming.

-Improved barrel smoke emission code to allow for customization of barrel smoke emission duration.

1.24

-Fixed errors in AI.cs for Unity 5.5.

1.23

-MoveTrigger.cs allows NPCs to lead the player through the scene by detecting when the player enters a sequence of triggers.

-AzuObjectPool.cs pools frequently used game objects like bullet shells, hit marks, and projectiles.

-Cookable grenades added as both selectable and offhand throw weapons.

-Offhand melee/secondary attacks for weapons.

-Bow and arrow weapons with realistic arrows, that stick into surfaces and can be recollected into player inventory.

-Sword and shield weapon to demonstrate blocking and melee possibilities

-Early version of third person mode (incomplete player model animations).

-Deadzone aiming/free aiming for independent weapon angles from camera. Options for unzoomed free aiming (like ARMA) and zoomed free aiming (like perfect dark, goldeneye).

-Weapon pivot bobbing for more weapon bobbing variation.

-Weapon pivot rolling for roll sway and bobbing of weapons.

-Forward and backward bobbing of weapons.

-Npc target detection FOV and backstabbing, with customizable weapon backstabbing angles and positioning.

-Melee weapon blocking, customizable to block only melee, or melee and ranged attacks, option for keeping guard after successful block, or canceling guard after successful block.

-Reduced dependency on layers for object identification, for better compatibility with deferred rendering.

-MovePlayerAndCamera.cs releases main camera from script control to allow other scripts to use it for cinema scenes, or other purposes. Also allows teleporting player after camera switch, or without camera switch.

-Friendly NPCs (faction 1) that can follow player and be commanded to move to a location by pressing the use key over a position.

-NPC taunt and alert vocal sound effects.

-Main menu added for scene selection and toggling of game options. Scalable with screen size, or unscaled screen size.

-AI optimizations, such as preventing inactive NPCs from being added to NPC registry, and caching of data.

-Water zone caustic effects when underwater and detection of camera submersion, instead of only player submersion.

-ReconfigurePrefab.cs allows dynamic switching between single camera and dual camera setups and configuration of both setups from one script. There is only one player prefab now instead of FPS Main 1 Cam and FPS Main 2 Cam as in version 1.22.

-Dual camera setup improved with larger near clip distances to reduce z fighting at long distances from scene origin.

-Weapon smoke and shell effects made independent of weapon model scale, for consistent effects in dual and single camera setup.

-Flashlights added as both a separate, selectable weapon, and weapon attachment.

-NPC locational damage and seamless transition to ragdoll (and back again).

-Options for initiating bullet time/slow motion for a few seconds after killing NPC or killing NPC with a hit to a certain body part.

-Improved player capsule friction and velocity handling so player won't slide down slopes (high friction), but still be able to walks up stairs (low friction) and not launch over obstacles at high speeds (vertical velocity limit).

-Player capsule customization improvements. Capsule radius value can be set in inspector and camera and capsule heights for various player states customizable from FPSRigidBodyWalker.cs in inspector.

-Most variables now have tooltips, visible by hovering mouse over them in the inspector. Variable descriptions removed from documentation for quicker updating of docs.

-Some script variables grouped by subject in inspector.

-Improved weapon impact and water particle effects

-Player capsule and rigidbody no longer get stuck against certain complex, concave mesh colliders.

-Ejected shells no longer use two game objects (rigidbody and lerped mesh) since fixed timestep is scaled with Time.timescale, which prevents stuttering objects in slow motion and allows the fixed timestep to stay as high/infrequent as possible (optimization).

-Capsule casts in FPSRigidBodyWalker.cs now ignore trigger volumes (allows player to walk through and crouch/uncrouch in them).

-Weapon fire framerate independence greatly increased, allowing for more consistent rapid fire rates in varying framerate situations.

-Holding use button over rigidbody now picks up object, and can be thrown by clicking left mouse, instead of having two separate buttons (Z,X) for these actions.

-Shading of albedo color for weapon model materials when raycast from player to sun is obstructed, for simulation of shadowing with dual camera setup. Can be toggled to allow for bright interiors or controlled by triggers.

-Hitmarker crosshair and sound effect for feedback when weapon damages object.

-Crouch Lean Amt and Stand Lean Amt of FPSRigidBodyWalker.cs allows different maximum leaning distances for standing and crouching states.

-Reach Distance variable in FPSPlayer.cs allows for customizing the distance player can pick up and activate objects.

1.22

-Added condition checks to WeaponBehavior.cs to prevent shells from ejecting from weapons by selecting "none" for Shell Prefab RB and Shell Prefab Mesh references in the inspector.

-Added godMode var to FPSPlayer.cs.

-Fixed canZoom var of WeaponBehavior.cs to allow /prevent zooming for a weapon.

-Fixed jump button not moving player up ladders.

-Added legs in first person view and wrote a VisibleBody.cs script.

-silentShots var of WeaponBehavior.cs can be used to make NPCs not detect certain weapons firing.

-Fixed logic error in slope check code

-Created static function to replace old PlayClipAtPoint code. Only one instance of this utility script is needed per scene.

-New player prefab called "FPS Player Main 1 Cam" can be used for 1 camera setups to allow shadows from level geometry to be casted on weapon models and for better compatibility with image effects and shaders.

-New, experimental tree models using Unity 5's LOD system, TODO: fade shaders between LOD states

-Improved a few weapon textures and models.

-Updated game assets to use Unity 5's new standard shader

-Updated WaterZone.cs and WaterZone prefab to use new Unity 5 water shader

-moved zoomSensitivity var to WeaponBehavior.cs to allow for tuning of zoomed mouse sensitivity per weapon

-Improved NPC horizontal tracking of targets. Smoother and more accurate rotation allows them to attack strafing targets much better now.

-NPC registry, spawner, and wave manager scripts created to track, and dynamically add NPCs to scenes.

-Replaced Autowaypoint.cs with WaypointGroup.cs, to optimize NPC patrolling behavior.

-Added Mecanim support for NPC animations.

-Cached most GetComponent calls for Animation and RigidBody components.

-Added removeTime var to WeaponPickup.cs to allow weapon pickups to be destroyed after a time.

-Added reachDistance var to FPSPlayer.cs to allow custom distance for item pickups and activations.

-Fixed landing sound playing more than once sometimes after player was airborne.

-Increased grounded detection height slightly to prevent sliding over uneven terrain at higher speeds.

-Improved 2 camera player prefab by upscaling weapons to prevent spatial jitter on larger terrains. This prefab can be placed in scenes with dimensions up to 12000X12000 before spatial jitter is apparent on weapon models, though shadows from world geometry are not cast on the weapon models while using this prefab.

-Removed WorldRecenter.cs script from player prefabs and added removePrefabRoot var to FPSPlayer.cs, since WorldRecenter is not needed anymore for huge maps.

-Changed shell ejection position handling and added shellEjectPositionZoom var to WeaponBehavior.cs for proper shell ejection using the dual camera player prefab with upscaled weapons.

-Added code to the Update() loop in WeaponBehavior.cs and removed all otherfx.pitch statements in WeaponBehavior.cs to allow reloading sound effect to dynamically scale up or down in pitch to match game speed.

-Increased grounded check capsule cast radius to prevent situations where player became stuck in ungrounded state when confined in a space just narrow enough to fit player collider after becoming ungrounded.

-Added code to prevent null reference errors for several assets if missing and added a console error in AI.cs if navmesh has not been baked for a scene, or NPC can't find navmesh.

-Fixed multiple landing sound effects playing from high drops.

-Optimized AI.cs by bypassing LOS check if target is not within range first.

-Added footSteps, walkStepTime, and runStepTime vars to AI.cs to play footstep sounds for NPCs.

-Added FoliageRustle.cs and the foliageRustles and foliageRustleVol to Footsteps.cs to play rustling sounds when moving through foliage.

-Fixed tracers effects not playing and added useTracers var to WeaponBehavior.cs to make tracer effect optional per weapon.

-Removed gravity var from FPSRigidBodyWalker.cs, which now just uses Physics.gravity.y instead (the gravity amount set in the PhysicsManager).

-Added dropOnPlayerCollision to DragRigidbody.cs. If true, dragged object will be dropped if it contacts player object to prevent pushing or lifting player. This behavior is now optional because the new physics system/player physics material does not allow the player to be pushed as much when dragging rigidbodies.

1.21

-Added checks for player stance in *FPSRigidBodyWalker.cs* to scale forward capsule cast base, height, and cast distance values for smoother jumping over obstacles.

-Stopped the reloading sound if player drops their weapon during a reload.

1.2

-Changed deltaTime check in *WeaponBehavior.cs* to prevent null reference error if starting scene with a weapon selected using the **firstWeapon** var of *PlayerWeapons.cs.*

-Added **showNegativeHP** var to *HealthText.cs* to allow option for preventing display of negative HP.

-*DamageZone.cs* script can now be added to a trigger to damage the player over time when they enter the trigger area.

-Prevented the optional sound effect for catching breath after sprinting from playing multiple times instead of only once.

-Added **limitStrafeSpeed** var to *FPSRigidBodyWalker.cs* to allow fine-tuning of diagonal strafe speed.

-The **burstFire** and **burstAndAuto** vars of *WeaponBehavior.cs* can now allow guns to use a 3 round burst fire mode. If **burstAndAuto** and **fireModeSelectable** is true, all three fire modes will be cycled for that weapon (like an MP5). If **semiAuto**, **fireModeSelectable**, and **burstFire** are true, weapon will cycle between burst and semi auto modes (like the M16A2). If only **semiAuto** and **fireModeSelectable** is checked, weapon will cycle between semiAuto and Auto (like an AK47).

-**burstShots** of *WeaponBehavior.cs* can be used to set the number of shots per burst.

-Added **walkAnimSpeed** and **runAnimSpeed** to *AI.cs* to control playback speed of NPC movement animations.

-**shotDuration** of *AI.cs* can now be used to control the delay between NPC attacks, with some randomness (**delayShootTime** is used to control time that shot/melee attack is fired after NPC starts attack or raises weapon).

-**shootAnimSpeed** of *AI.cs* now sets the playback speed of the firing animation. A value of 0.5 means animation plays half as fast. This is mostly to allow NPCs to raise their weapons for a longer time when using burst fire weapons like the soldier NPC.

-Fixed timing of weapon sprint animation and position cancel when player is falling.

-Added **fadeTexture** var to *PainFade.cs* to allow textures to be displayed on screen for player damage feedback.

-Added **liquidMask** layer mask to *WeaponBehavior.cs* to allow submerged objects to be hit when firing from above water surface.

-Updated *FPSPlayer.cs* **rayMask** to include layer 19, the interactiveObjs layer, so items can't be picked up from behind glass.

-Added small delay to *WorldRecenter.cs* to prevent camera lerp misalignment during a world recenter.

-Made the forward capsule cast check for player movement take the player's capsule collider height into account so low spaces like vents can be traveled into.

-Created *InputControl.cs* script which handles all input calls for the asset and reads bindings from Unity's Input Manager.

-Cleaned up jumping code to detect button press instead of button hold, which allows better detection and jumping over frontal obstacles player might be walking or running into.

-Moved most GetComponent calls into initialization code (for optimization).

-Updated GUIText scripts to only update parameters if information to display has changed (for optimization).

-Shortened length of folder names to reduce chance of access errors if path to assets was too long.

-Made the slope check in *FPSRigidBodyWalker.cs* better detect uphill or downhill slopes.

-Added a check in *DragRigidBody.cs* to raise weapon if dragged object is destroyed while dragging.

-Added **inputSmoothSpeed** var to *FPSRigidBodyWalker.cs* to allow customization of input smoothing speed.

-Updated landing and footstep sound effect code to prevent null refs if no sounds are defined.

-Added **sprintStrafeRoll** and **walkStrafeRoll** vars to *Ironsights.cs* to control camera roll when strafing.

-Modified *FPSRigidBodyWalker.cs* to allow player to better change crouch states with positive and negative **playerHeightMod** values.

-Added **fallDamageMultiplier** var to *FPSRigidBodyWalker.cs* to control amount of falling damage taken as a product of units fallen times this value.

-Added **healthToRestore** vars to *DrinkPickup.cs* and *FoodPickup.cs* to allow food and drink pickups to also restore player health in addition to thirst and hunger.

-Improved the capsule collision check in **FPSRigidBodyWalker.cs** to prevent player from getting stuck in small, concave spaces.

-Added a CameraJump animation to the Main Camera object which the *FPSRigidBodyWalker.cs* script plays at the beginning of a jump for a better sense of player weight.

-Added **silentShots** var to *WeaponBehavior.cs* and **listenRange** var to *AI.cs* to allow silent weapons and NPC detection of weapon fire.

-Prevented ejected shells from inheriting weapon angles and velocity if player is prone and moving to avoid unrealistic movements of shells.

-**fireSndRandPitch** var of *NPCAttack.cs* can now be used to define lower range of random pitch for attack sounds (pitch will be between this value and 1).

-**allowProne** and **allowLeaning** vars of *FPSRigidBodyWalker.cs* will define if the player can go prone or lean.

-**proneCamHeight** var of *FPSRigidBodyWalker.cs* will define the height of the camera when the player is prone.

-Soldier NPC added which drops an M4 weapon pickup when killed.

-Knife and M4 weapons and pickup items added.

-Fixed barrel smoke particle effect which was not playing for all weapons.

-Improved Physics.CheckCapsule and Physics.CapsuleCast position arguments in *FPSRigidBodyWalker.cs* to use less approximation and better compatibility with a wide range of **playerHeightMod** values.

-Moved ammo and weapon type variables to the top of *WeaponBehavior.cs* for easier/faster accessibility from the inspector.

-The bullet-time feature can now be toggled on and off by clicking the Allow Bullet Time box of the *FPSPlayer.cs* script component.

-Help text display can now be toggled on and off by clicking the Show Help Text box of the *FPSPlayer.cs* script component.

-Added **fontScale** variables to GUIText scripts to allow automatic scaling of text to screen height.

1.19

-Fixed AI search range not increasing after being attacked by player.

-Updated coroutine return and break statements to prevent errors when building for Windows 8 Store.

-Applied layer changes to the Glass prefab to allow it to take damage after being placed in scene from the Project Library.

-Added **cameraIdleBob** and **cameraSwimBob** vars to *Ironsights.cs* to allow customization of vertical bobbing of view while idle or swimming.

-Added **staminaDepleted** var to *FPSRigidBodyWalker.cs* to disable sprinting if stamina is fully depleted and **sprintRegenWait** is true. This is done to prevent the player from sprinting for short times if stamina is low.

1.18

-Removed animation check var in *CameraKick.cs* to prevent rare occurrences of camera being unaligned.

-Replaced GetCurrentProcess() call for quit button with Application.Quit since it has been fixed.

-Modified smoke texture so its alpha mask will be properly read in newer versions of Unity.

-Added smoothing to sprinting animation playback for better motion transition.

-Fixed **playerHeightMod** values between 1.0 and 2.0 not allowing player to move.

-Added option for regenerating health.

-Backwards sprinting is now prevented.

-Increased item pickup raycast length slightly to make activating scaled-down weapon pickups easier.

-Added ApplyDamage function parameters for more realistic NPC ragdoll behavior.

-Optimized shotgun impact particle effects played on NPCs.

-Improved timing of ejected shell rotation.

-Added options for hiding weapon when player is underwater or holding an object.

-Added **inaccuracy** var to *NPCAttack.cs* to allow enemies to miss on their attacks.

-Made automatic weapon's recoil spread recover when the fire button is released instead of waiting for semi auto shot timer.

-Added optional player hunger and thirst attributes which are reduced by apple and canteen pickups

-Updated uncrouching obstacle check to prevent player from uncrouching into overhead geometry in some situations.

-Hard-coded the raycast slope limit angle in *FPSRigidBodyWalker.cs* to 60 to prevent player from getting stuck in small indents or tight spaces (the capsule check slope limit can still be customized).

-Scripts now prevent dragging objects while player is zoomed or dead.

-Added **kick back amount** variables to *WeaponBehavior.cs* to allow slight backward motion of weapon during sustained fire.

-Weapon z position (forward & back) for zoomed, unzoomed, and sprinting states are now customizable from the inspector.

-Added **removePrefabRoot** bool to *WorldRecenter.cs* script and deleted RemovePrefabRoot.cs script to allow option for keeping the FPS Main prefab root object or deleting it on scene load to allow the world recenter script to keep the FPS Player object's local and world coordinates synchronized to prevent spatial jitter and allow larger scenes.

-Converted AI scripts from javascript to C# to allow easier compilation for android and windows projects and better communication between scripts.

-Added **walkBobAmountX** and **walkBobAmountY** vars to *WeaponBehavior.cs* for customization of walking bob amounts per weapon.

-Added **walkBobYawAmount**, **CrouchBobYawAmount**, **ZoomBobYawAmount**, and **SprintBobYawAmount** vars to *Ironsights.cs* to allow camera yaw angle bobbing.

-Changed SendMessage call for ApplyDamage() functions to direct function calls identified by game object layer.

-The **sprintRegenWait** and **sprintRegenTime** variables of *FPSRigidbodyWalker.cs* can now be used to disable the sprinting button until sprintRegenTime has elapsed and sprinting stamina is fully regenerated.

-Added **drownDamage** var to *FPSRigidBodyWalker.cs* to set the rate that player takes drowning damage.

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 things to take into consideration when importing the FPS Prefab into projects. First of all, in version 1.23 a script called ReconfigurePrefab.cs has been added, which is attached to the FPS Camera object and allows you to switch between single and dual camera setups.

The single camera setup uses smaller scaled weapon models which are lit according to scene lights and shadows. This requires a closer near clip plane, which reduces the maximum scene size to about 1200x1200 units with the middle of the playable area/terrain being centered over the origin (-600, 600 units on each side).

The dual camera setup uses one camera to render the world, and another to render weapons over it. The weapon models in this setup can have a larger scale and extend past the player collider, allowing for a farther near clip plane, which supports much larger 12000x12000 scenes before floating point error in models becomes apparent.

WorldRecenter.cs can be used with the single camera setup to support larger scenes by recentering all scene objects back to the scene origin if the player has moved past the recentering distance. Currently object batching, occlusion areas, and navmeshes do not move along with the other objects, though this may change in a future version.

Another thing to keep in mind is that the NPC AI requires a navmesh in a scene so they can find their way through a level. If a baked navmesh isn't found, or the NPC spawns too far from one, an error will be displayed in the editor console window and NPCs won't move.

And finally, the Realistic FPS Prefab uses layers and tags to identify certain game objects and automatically configures collision layers in the Start() function of the FPSPlayer script. If the layers and tags in your project are not named like this, please copy the included TagManager.asset file included in RFPSPProjectSettings.rar into your project settings folder. If more layers are needed for other behaviors while using this asset, please verify your layers are named as shown below, and add any additional layers below layer 21.

Tutorials

Adding the Realistic FPS Prefab to a new scene

Adding the asset to your scenes is easy. After importing the downloaded Realistic FPS Prefab package into your project, drag the *FPS Main* object in the RFPSP 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.

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.

Once you have a weapon model, please expand the *FPS Main* object in the Hierarchy window and then expand the *FPS Weapons* objects until you see the individual weapon objects with weapon names. These are the main weapon object parents which have a WeaponBehavior.cs component and the weapon model object and muzzle flash positions beneath them in the hierarchy.

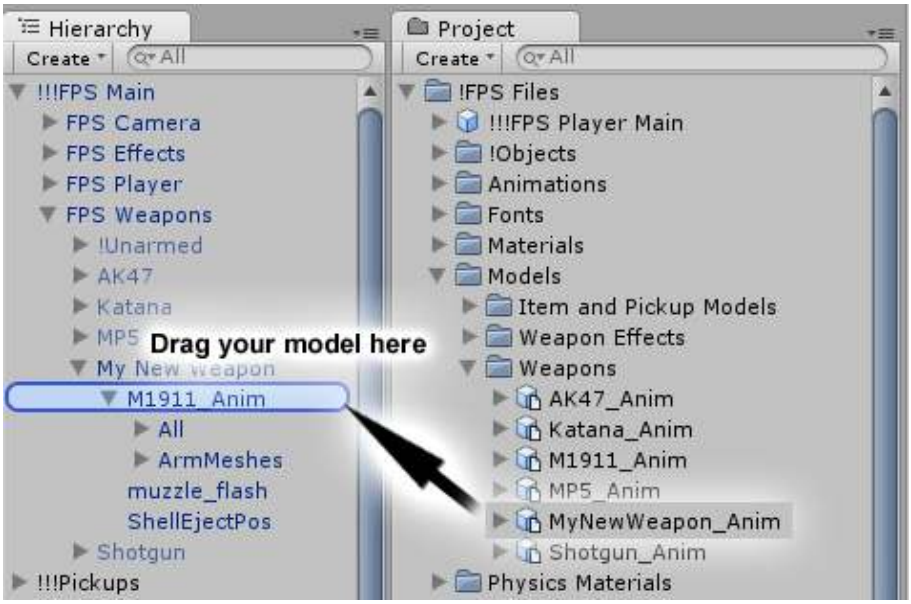
To create a new weapon, find the object that most resembles the type of weapon you want to add, right click that weapon object, and select duplicate. Then rename the duplicated weapon object to your weapon name and select this object with the left mouse. In the inspector window, you should see a script called WeaponBehavior with many editable variables. This script controls nearly all weapon actions.

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, muzzle_light, and a ShellEjectPos object, unless you duplicated the Katana, which only has a weapon model and a null muzzle_flash stand-in object.

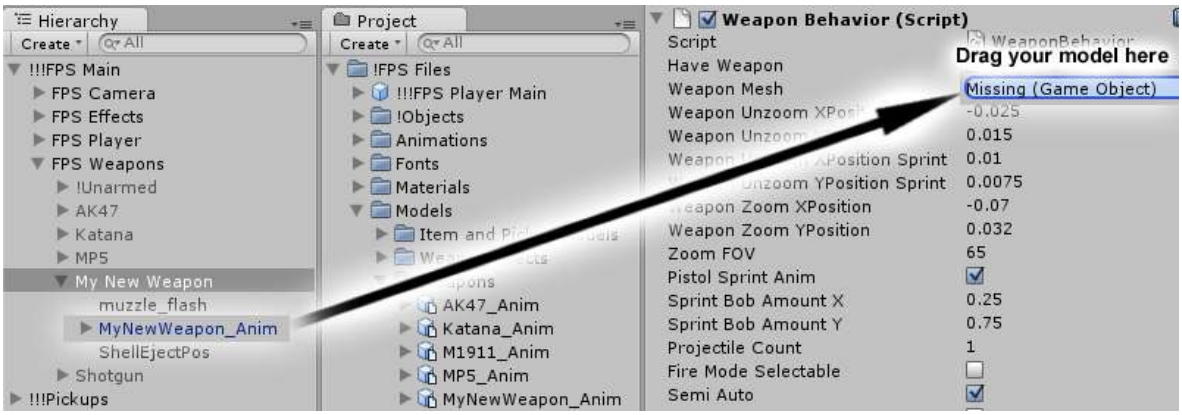
The next step is to add your weapon mesh to the FPS Prefab under your duplicated weapon object's hierarchy. To do this, find your weapon model in the Project Library window, and drag it over the weapon mesh with the _anim suffix under your duplicated weapon object in the Hierarchy window. This will import your mesh as a child of the existing mesh, inheriting its scale, rotation, and position. You can also try dragging your new weapon model over the weapon object parent you duplicated earlier to see if the rotation and scale are closer to that of the existing _anim mesh you will be using as a reference for weapon positioning. At this point, it is important to set the layer of your new weapon model object to layer 8, the gun layer, so the scripts will position the model in front of the camera correctly.

If your new weapon model is a child of the original weapon mesh with the _anim suffix, 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 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.

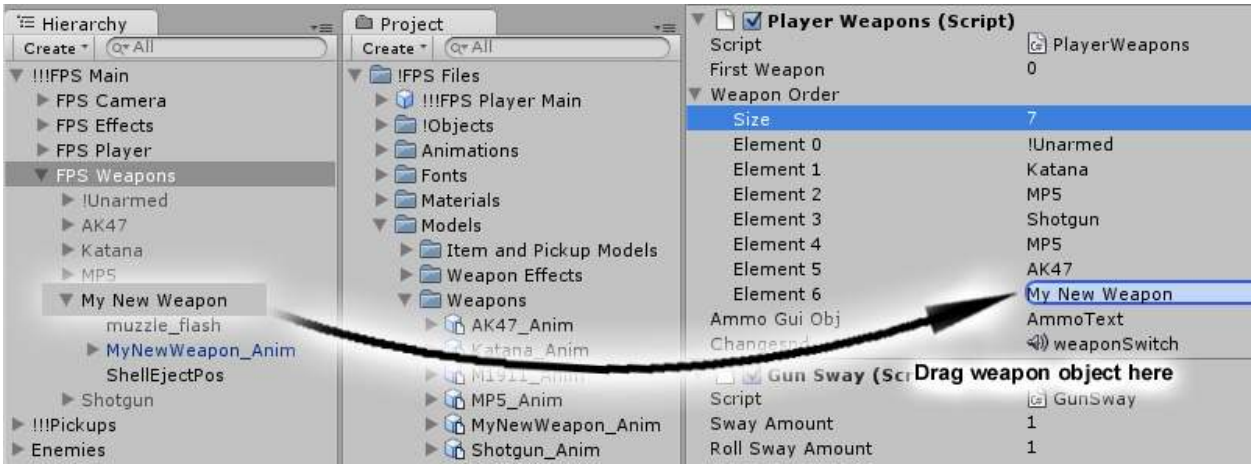


Now we need to tell the weapon script which model to use for your weapon. Near the top of the WeaponBehavior script's variable list, there is a variable named Weapon Mesh. With your weapon object selected in the Hierarchy window and the WeaponBehavior variables visible in the Inspector window, Find your 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. This video http://youtu.be/oh3GzWoceY4 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 *RFPSP/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|RFPSP|NPCs|ZombieNPC* for the zombie NPC or from *Assets| RFPSP|NPCs|RobotNPC|* 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 the original won't accidentally be overwritten.

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.cs 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.cs component. New character models will need to have their character joints set up and each collider needs a LocationDamage.cs component. Unity's ragdoll wizard is a good place to start. 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.

Note: There is also a Mecanim animated NPC model which you can use.

Changing The Player Character Model

Here are the steps to replace the default player character model for the Realistic FPS Prefab version 1.4:

- Place your new character model in the scene and scale it to the same size as the original player character model.

- Copy the PlayerCharacter.cs component from the original Player Character Model object to your new one, and set the Controller of the new model's Animator component to the Player Model animator controller.

- Set the Alternate Body Material of the new PlayerCharacter.cs component to none (if not using one).

- For the third person weapons, you can copy and paste or duplicate the Third Person Weapons object from the original Player Character Model object and make it a child object of your new character model. PlayerCharacter.cs will then activate/deactivate and position the weapon models in the new character's hands.

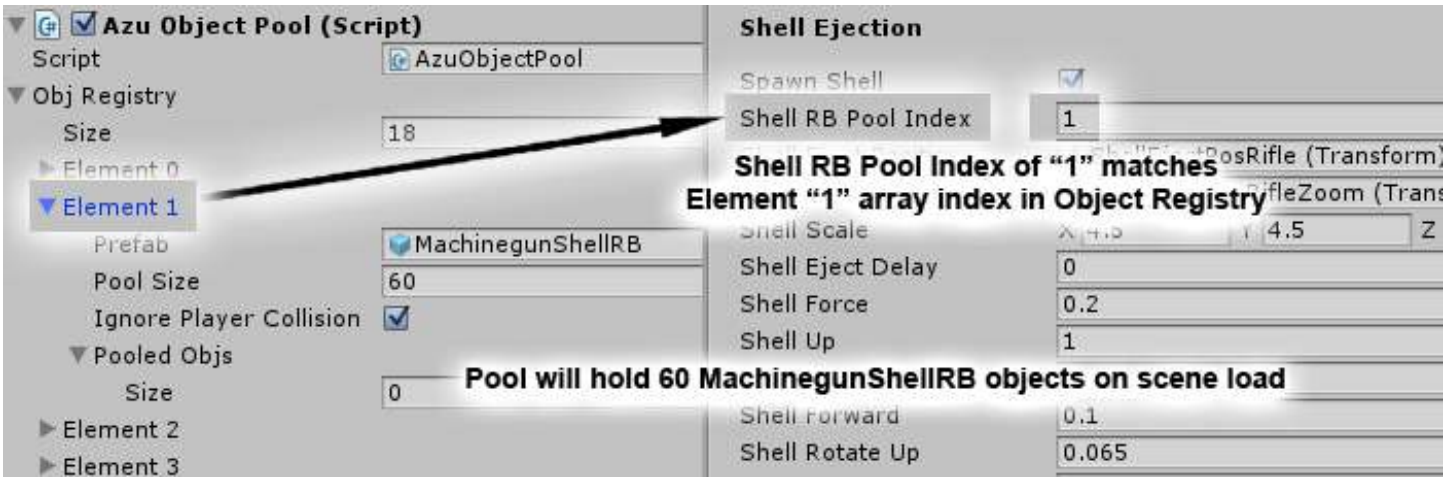
- You can adjust the position of the Weapon Positioning object, which is a child of the Third Person Weapons object by rotating it to fit in the character's hands when running the scene, then copy the Weapon Positioning transform component, stop playback, and paste the transform of Weapon Positioning to save the new weapon position and angles.

- Adjust the avatar definition of your new model to match the alignment of the original player character model to fix any bent limbs (the new character can also be reoriented in a 3D App, or you can use new animations that go with your character model).

- Finally, to get your character model to look up and down, the M Look Ang variables of the PlayerCharacter.cs component can be edited in the inspector. The M Look Ang amounts control the yaw, pitch, and roll amounts of the look input to apply to the rotation of the humanoid spine bone (M Look Ang) and the chest bone (M Look Ang 2). This can take some trial and error, depending on how your new model's bones are oriented in the bind pose. If the look rotation is too twisted, a good method is to zero-out all the M Look Ang values and add to them in small increments until your character looks up and down properly.

The Object Pooling System

Version 1.23 introduced an object pooling system which handles frequently used objects such as bullet shells, hit marks, and weapon projectiles. Object pooling is a way to minimize the frequent creation and destruction of game objects, which can burden the game engine and garbage collector unnecessarily. The main change introduced by the pooling system, is the way that some of the objects are spawned in the scene. In previous versions, this was done by instantiating the object from the project library. Objects can now be spawned from the object pool and recycled when the object is no longer needed in the scene.



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, their related scripts and child game objects, and other scripts used by the prefab. For more detail on how these scripts operate, please refer to the script tooltips in the inspector or 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 can optionally be deleted if the FPSPlayer.cs script's Remove Prefab Root var is true when loading a scene. This is done to allow the FPS Player's local and world position coordinates to be synchronized and reduce the effects of floating point imprecision.

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.

CameraControl.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. The third person camera's behavior is calculated here as well.

CamAndWeapAnims.cs

This is a simple script that contains four Vector3s whose values are animated by other scripts for bobbing of camera and weapon angles and position.

PainFade.cs

This script is used to display pain texture display on the screen.

MovePlayerAndCamera.cs

Contains methods to teleport the player or release main camera for cinema scenes or other uses.

ReconfigurePrefab.cs

Allows user to toggle between single and dual camera setups.

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:

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.

InputControl.cs

This script contains all input calls for the prefab and makes them available for the rest of the scripts as public bool and float variables. InputControl.cs also contains code to interpret game pad button and axis input.

FPSPlayer.cs

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

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.

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.

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.

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.

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.

GunSway.cs

Handles weapon sway and weapon angle bobbing.

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).

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.

WeaponPivot.cs

Rotates weapon models around pivot points for deadzone aiming.

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.

ArrowObject.cs

Detects hits for flying arrows, sets up collision, and arrow retrieval.

GrenadeObject.cs

Detonates grenade after fuse timer has elapsed.

Player Character Model

This object contains all the other objects needed to animate the player character in third person mode and the visible body in first person mode.

PlayerCharacter.cs

The main script which handles the animation of the player character models, creates a first person visible body object, and contains variables that can be modified in the inspector to fine tune the position of the player model.

ThirdPersonWeapons.cs

This script is placed on the Third Person Weapons object which is a child of the Player Character Model object and contains all the third person weapon models. This script is includes a list that holds all the needed objects and positions for the weapons in third person.

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.

NPCRegistry.cs

This script tracks all NPCs in the scene and gives them targets, taking into account NPC faction alignments and player stance.

NPCSpawner.cs

This script spawns NPCs, using several parameters to control spawn timing and amounts.

WaveManager.cs

This script spawns NPCs from NPC Spawners for successive waves using several parameters to control spawn timing and amounts.

WaypointGroup.cs

This script creates a waypoint group out of its child waypoint objects for an NPC to patrol and optionally draws the waypoint positions and path lines in the editor.

AI.cs

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

CharacterDamage.cs

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.

LocationDamage.cs

Applies damage to specific NPC body part colliders, allowing for headshots and enabling of bullet time/slow motion after killing NPC with hit to specific body part.

NPCAttack.cs

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

RemoveBody.cs

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.

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.

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.

MoveTrigger.cs

Commands an NPC to move to a position when player enters trigger, and sets up next move trigger in sequence (for NPCs that lead the player through the scene).

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.

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. Also restarts a map after player dies.

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.

HUD and GUI objects

HUD and GUI elements are rendered on the with either a GUI script for the Main Menu, or a UI Canvas component for the HUD elements.

MainMenu.cs

Displays main menu GUI and manages button states. Allows player to activate cheats, free aim zooming, exiting and resuming game, and loading different game modes.

AmmoText.cs

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

HealthText.cs

This script displays the player health amount on the screen.

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.

HungerText.cs

This script displays the player hunger amount on the screen.

ThirstText.cs

This script displays the player thirst amount on the screen.

WarmupText.cs

This script displays the wave warmup countdown on the screen.

WaveText.cs

This script displays the wave number an NPC amount on the screen.

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 collider component and their tag set to "Usable", 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.

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.

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.

FoodPickup.cs

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

DrinkPickup.cs

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

AppleFall.cs

This script is attached to suspended apple pickups and causes them to fall when damaged or "picked" when the player grabs them.

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.

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.

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.

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.

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.

DamageZone.cs

When this script is attached to a trigger collider, the player or NPC that enters the trigger area will be damaged over time. This script can be used for fire, poison, or radiation zones in a scene.

FoliageRustle.cs

Plays foliage rustling sounds from trigger object when player or NPC enters the foliage trigger.

FoliageRustle.cs

Plays sound effect when player or NPC enters the object's trigger. Used for playing sounds of running into chain link fence in demo scene, but can be used for any object.

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.

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.

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.

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.

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.

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.

Utility

These scripts perform a variety of useful functions.

PlayAudioAtPos.cs

Plays audio at a point using the PlayClipAt() method, with several parameters for customization of playback, such as volume, pitch, spatial blend, and attenuation distances.

TempAudioTimer.cs

Deactivates a temporary audio object after it finishes playing.

AzuObjectPool.cs

Creates object pools and contains functions for instantiating and recycling pooled objects.

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

Thanks to: Millenia, Sargent99, Benderexable, Kamiomi, Pbcrazy, Angryfly, druggon, 3D4Ds, syntheno, Unity3D, 3dheaven, Peacemaker, jaraxe, mrpokephile, astradamus, mp6arms, Kalamona, dagon19, and pixelhouse for Creative Commons assets

Thank you for your purchase!