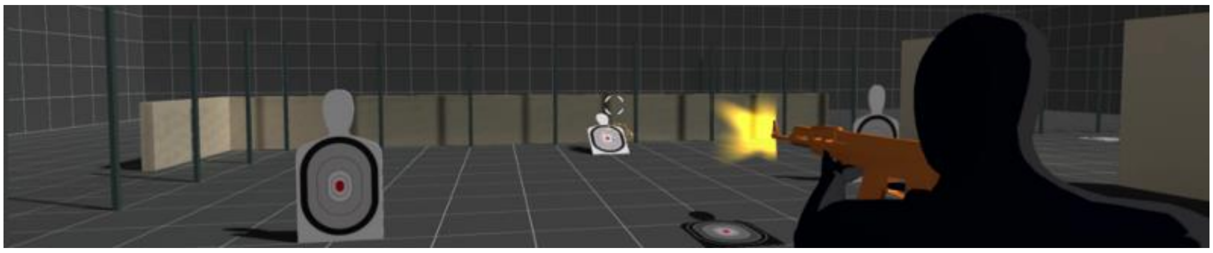
COVER + SHOOTING SYSTEM



Thank you for acquiring the **Cover + Shooting System – Third Person Shooter** asset for the *Unity 3D* engine!

This asset contains scripts for player **covering** and **shooting** behaviours, besides the basic movements like **run, sprint, jump, aim** and a dynamic **third person camera**, with collision detection and custom modes for special player movements.

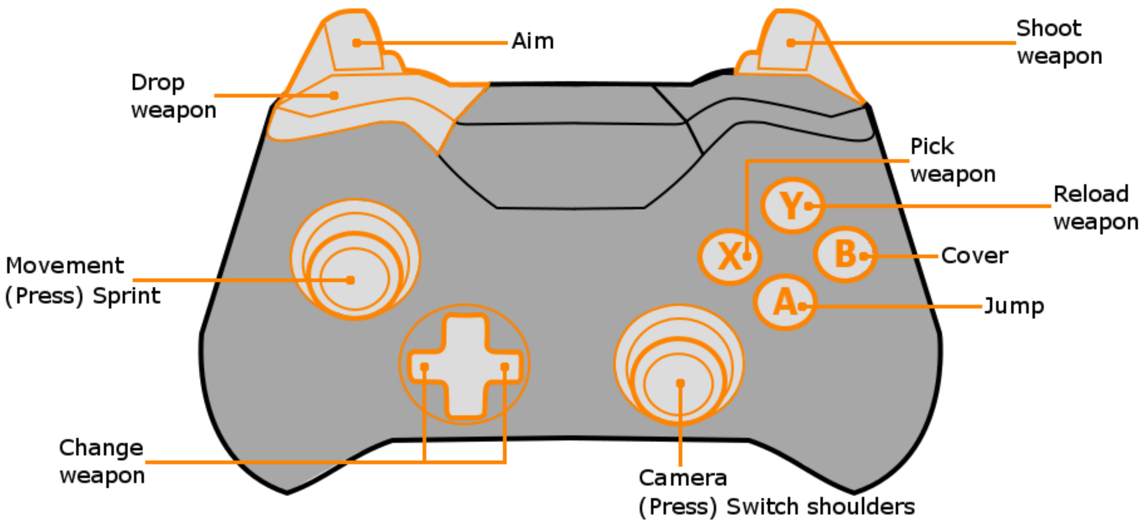
The **cover** system features auto **navigation** to cover on short distances, **peeking** around corners, **turning** on cover corners, quickly **change** to nearby covers, and **jump over** cover movements.

The **shooting** system features full interaction with weapons, like **pick up, aim, shoot, reload, drop**, and **change** weapon actions.

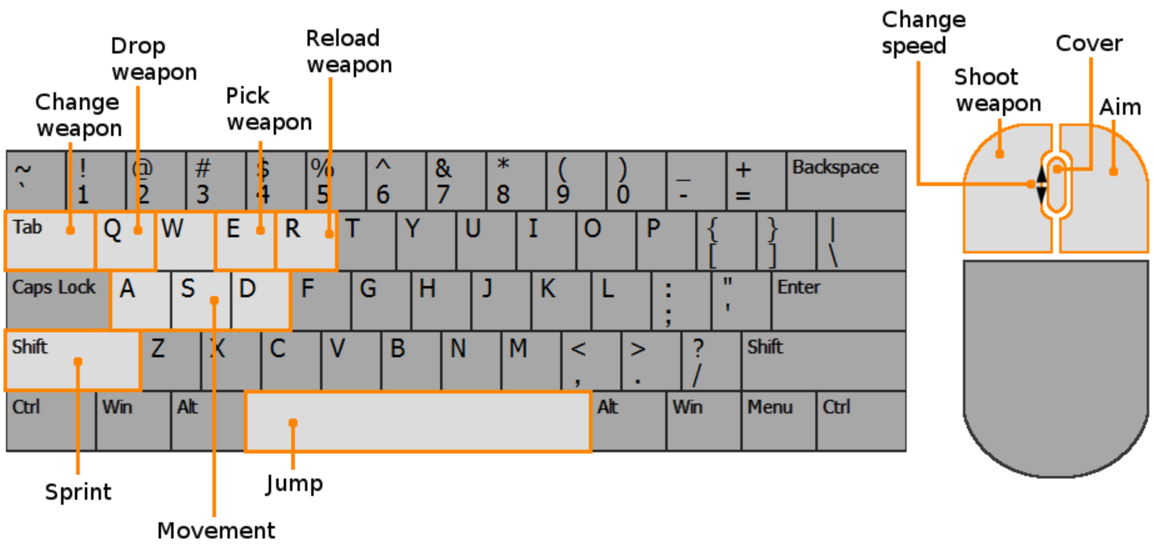
This asset also presents configurable **short** and **long** weapons, like pistols, assault rifles, etc. Features 3 different weapon modes: **semi-automatic**, **automatic** and **burst mode**. Effects for **shot**, **flash**, **tracer**, and **bullet holes** are included.

SETUP NOTES

This asset comes with full support for the **Xbox One** controller. The buttons and axes are configured under the *Project Settings* > *Input* section. The controls are mapped as follows:



For keyboard and mouse, the input is mapped as follows:



If you want to use another game controller, you should configure the correct buttons and axis on *Input* menu to map it accordingly.

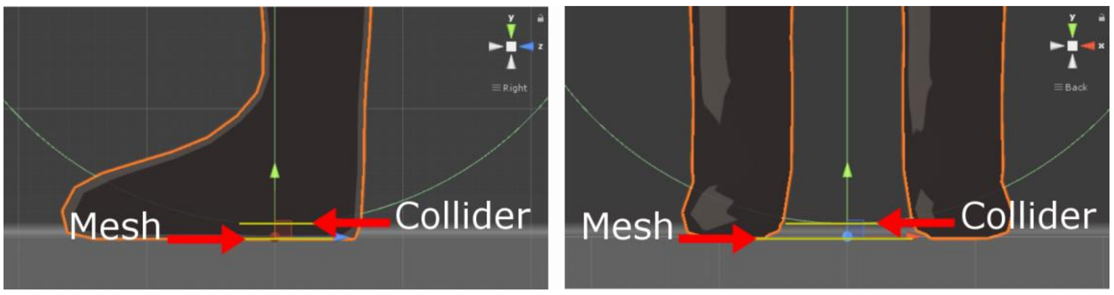
You can press **F2** on the keyboard, or the **Xbox One Controller** ☰ button to see the input commands on the example demo scene.

A detailed tutorial on how to setup the essential files on a custom project is provided on a video that can be accessed through the following link:

Tutorial: How to setup on a custom project

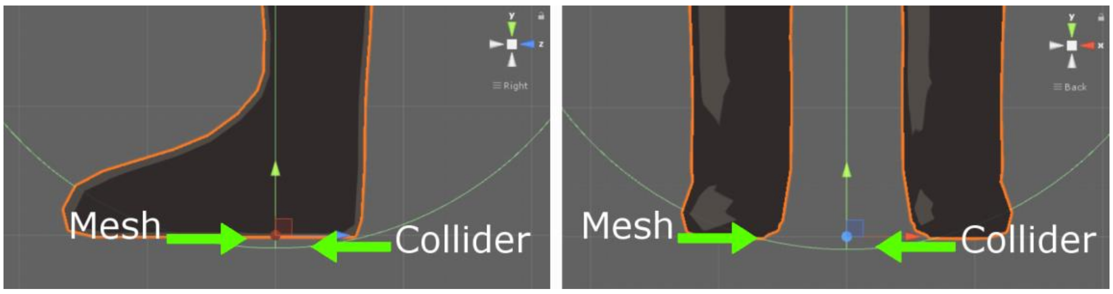
IMPORTANT NOTES

. When configuring the *Capsule Collider* on your own character avatar, you may carefully avoid the following situation:

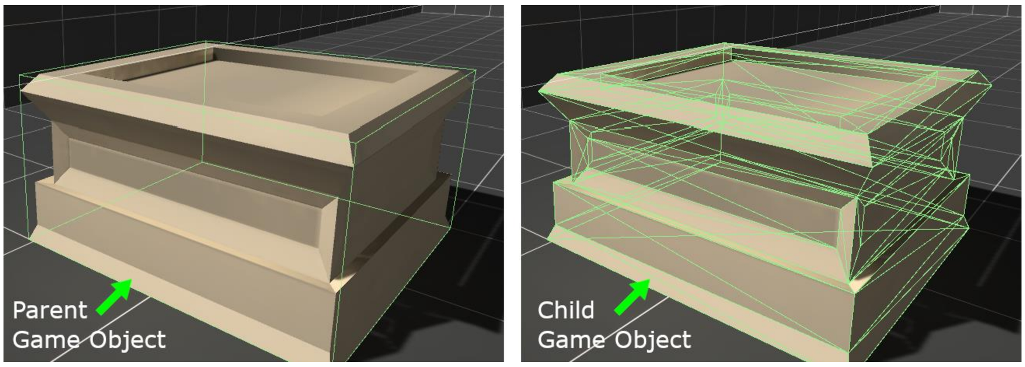


If the collider end is positioned above the mesh end, the character will get stuck when trying to move. Do not let the avatar mesh surpass the collider!

The correct position is set when the bottom of the collider is at the same level or under the mesh end:



. You can use a simpler mesh as a cover container for more complex meshes. For that, you must put the primitive collider on a game object and set its Layer as *Cover Invisible*. Then, put the complex mesh and collider on a game object under this one.



Do not set the child game object with the *Cover Invisible* layer. Using the complex mesh as a collider for the child game object ensures the shots will work properly. For more information upon setup, please refer to the *stand* game object inside the demo scenes.

. Remember to set all cover objects on the scene as *Static*, and mark it as *Cover* or *Cover Invisible* under the *Layer* parameter.

. Finally, remember to re-bake the *NavMesh* after any modifications on the environment. Otherwise the pathfinding may not consider a new obstacle between the player and a cover, and the script may not work properly.

You may find some specific setup notes on configuring player behaviours in the *Debug Notes* subsections of each corresponding behaviour described below.

COVER BEHAVIOUR

The **CoverBehaviour** script has the following external parameters:

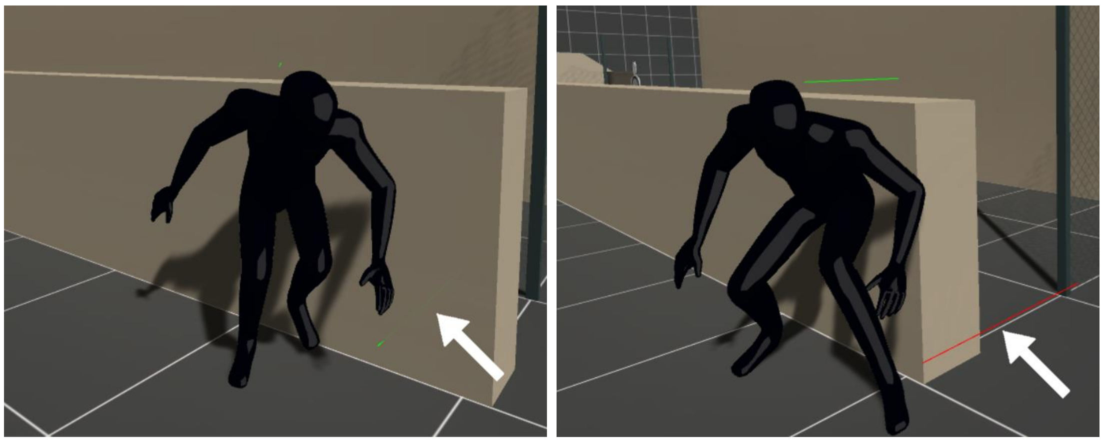
- **Cover Button:** the name of the button on the *Input* settings to activate cover related actions.
- **Cam Corner Offset:** the offset that shift the camera when the player reaches a cover corner.

- **Orientation smooth:** the smoothing factor to re-orientate player when turning corners or rounded covers.
- **Crouch smooth:** the smoothing factor to change from cover standing to crouch cover.
- **Search For Cover Dist:** the maximum direct distance to search for covers.
- **Show Path Delay:** the delay, in seconds, when focusing on a cover, to show the path to navigate to it.
- **Cast Origin Height:** height of the ray cast origin for movement tests (see Debug Notes). If the player is getting stuck when covering and moving up on slanted surfaces, consider raising this value.
- **Path Color:** the color of the line that shows the path to the cover.
  Cover Sign: the canvas image to show on the cover destination when is possible to take cover.
- **Turn Cover Sign:** the canvas image to show after a cover corner, representing a possible turn direction
- **Change Cover Sign:** the canvas image to show after a cover end, when a near cover is next, and is possible to quick change to it.
- **Jump Cover Sign:** the canvas image to show above the player, when is possible to jump the current cover.
- **Cover Mask:** the layer mast to define cover objects.

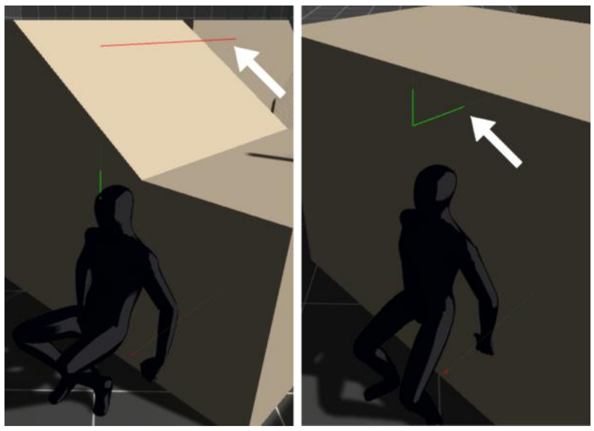
DEBUG NOTES

You will find some debug lines being displayed on the **CoverBehaviour** script. It is important to note that the relative position of this lines should vary with different player avatars, since they follow the size of the player *Capsule Collider*.

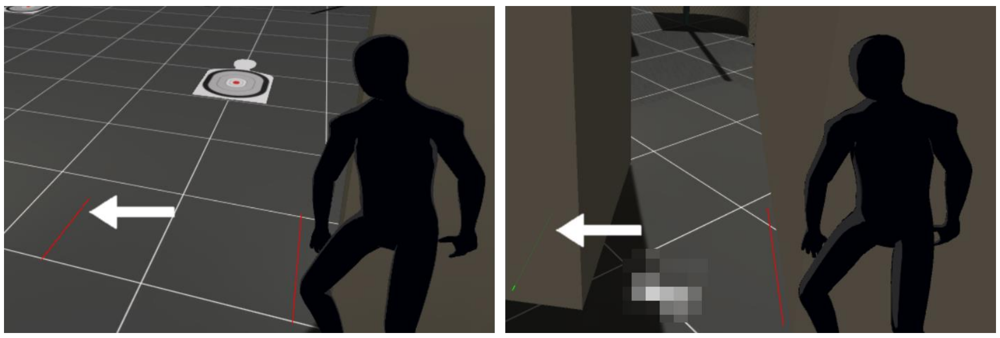
One of the lines defines the collider end of the character. It is related to the test **cast** that looks for the **cover end**, responsible for stopping the player when the end is reached:



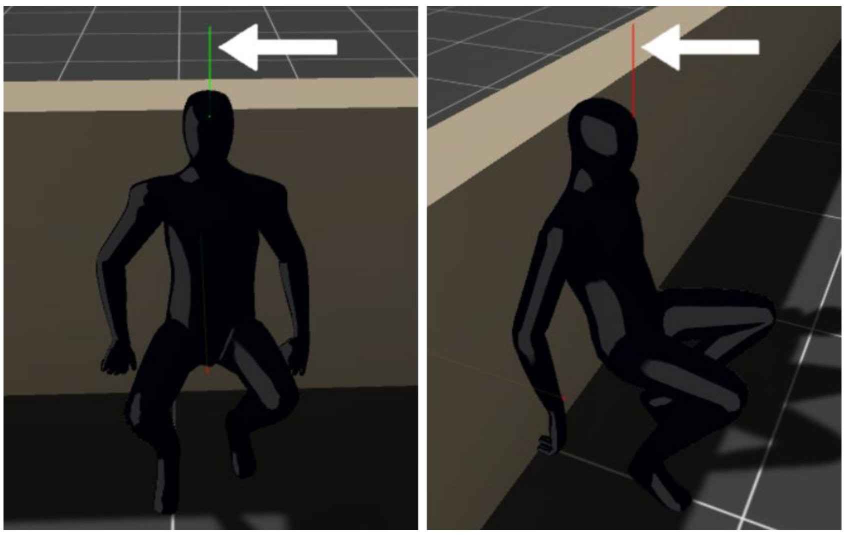
The line above the player is related to the test **cast** for the **cover size**. If possible, the player will **stand up**. If not, the player will move to the **crouching** state:



Another line represents a projection of the player position, and refers to the **reach** of the player **change cover** movement. This test **cast** checks if the player is capable of using a **quick-change** movement to pass to the next cover:



The line on the player's head refers to the player field of view (**FOV**). Following the maximum allowed **angle**, set by an internal parameter, this test **cast** compares the **camera direction** on the horizontal plane and the **player backward direction** when covering. If the angle is within the maximum allowed, the **aim** when crouched, **peek aim** when on corners, and **jump over cover** movements will be allowed:



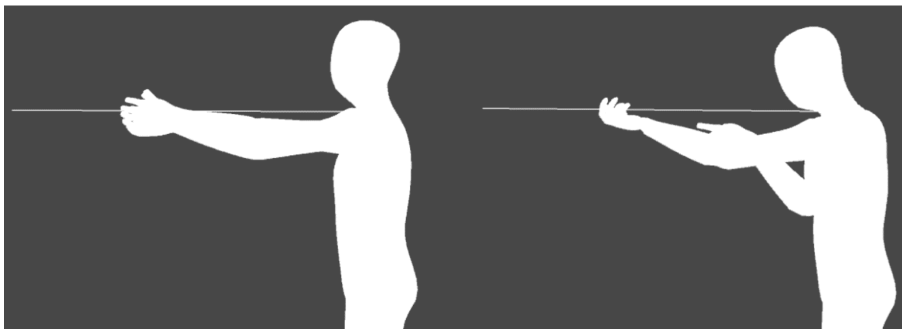
SHOOT BEHAVIOUR

The **ShootBehaviour** script has the following external parameters:

- **Shoot Button:** the name of the button on the *Input* settings to fire a weapon.
- **Pick button:** the name of the button on the *Input* settings to pick up a weapon.
- **Change Button:** the name of the button on the *Input* settings to change weapons.
- **Reload Button:** the name of the button on the *Input* settings to reload a weapon.
- **Drop button:** the name of the button on the *Input* settings to drop a weapon on the scenario.
- **Aim Crosshair:** the crosshair texture to display when aiming with a weapon.
- **Shoot Crosshair:** the crosshair texture to instant display when shooting with a weapon.
- **Muzzle Flash:** the effect which will be shown on the gun muzzle immediately when a shot is fired.
- **Shot:** the particle effect that is emitted from the gun muzzle to the shot destination when shooting.
- **Sparks:** the particle effect that is emitted on the shot destination.
- **Bullet Hole:** the effect that is displayed on the shot destination and is remains there for a time.
- **Max Bullet Holes:** the maximum amount of bullet holes displayed on the current scene.
- **Shot Error Rate:** the error margin of the aim when shooting. Consider the value of 0.01 as a 1% error rate.
- **Shot Rate Factor:** the weapon rate of fire, when in automatic or burst mode.
- **Arms Rotation:** the arms rotation of the player avatar when aiming. Use to align the aim to the horizon.
- **Shot Mask:** the layer mask to cast the player shots.
- **Organic Mask:** the layer mask to define organic matter. Bullet holes and sparks are not placed in organic objects.
- **Left Arm Short Aim:** rotation offset for the left arm when aiming with a short gun.
- **Right Hand Cover:** rotation offset for the right hand when covering with a long weapon.
- **Left Arm Long Guard:** rotation offset for the left arm when standing with a long gun.

DEBUG NOTES

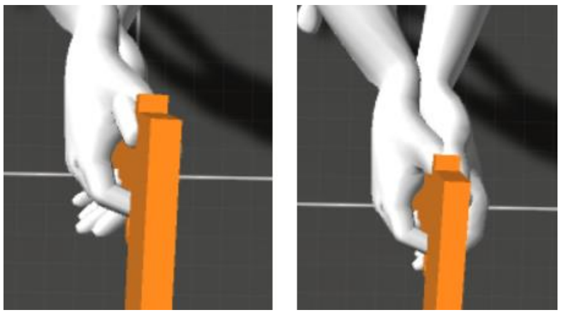
1. There is a unique debug line displayed by the **ShootBehaviour** script. This line is a guide to rotate the player arms correctly, for both types of weapon:



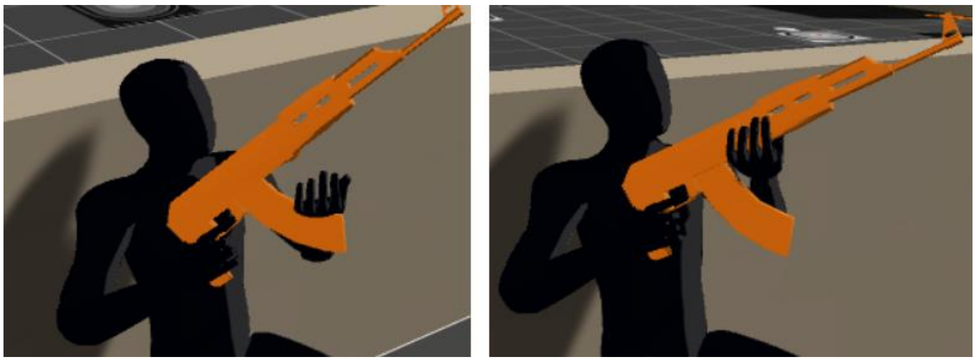
Please refer to the video tutorial linked at the end of the section *Setup Notes* for a detailed explanation on how to align the aim correctly.

2. Due to some variations in the armature (skeleton) structure of different player avatars, inconsistencies on the orientations of some bones may occur. This can cause anomalies such as a hand inside another. Although this is unusual, it can appear in some animations for your custom avatar. To address this, the last three parameters, which are under the **Advanced Rotation Adjustments** section, can be used to adjust the bone rotation for some cases in particular. The following figures show examples for these situations.

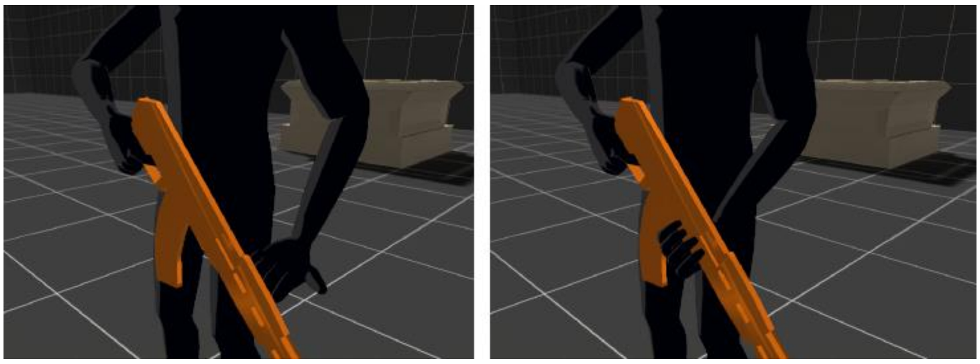
   First, the inaccurate and the adjusted left (support) arm rotation, given by the **Left Arm Short Aim** parameter, resulting in a correct support hand orientation when aiming with a *short* gun:



   Second, the inaccurate and the adjusted right hand rotation, given by the **Right Hand Cover** parameter, resulting in a correct weapon placement over the support hand, when covering with a *long* gun:



   And finally, the inaccurate and the adjusted right arm rotation, given by the **Left Arm Long Guard** parameter, resulting in a correct weapon placement over the support hand, when standing in a guard position with a *long* gun:



INTERACTIVE WEAPON

The **InteractiveWeapon** script is responsible for a weapon interaction with the player. The following external parameters are present:

- **Label:** the name of the weapon displayed on the scene.
- **Shot Sound:** the sound that will be played when a shot is fired.
- **Reload Sound:** the sound that will be played when reloading the weapon.
- **Pick Sound:** the sound that will be played when the player picks the weapon.
- **Drop Sound:** the sound that will be played when the player drops the weapon.
- **Sprite:** the image to display on the screen HUD when the weapon is selected by the player.
- **Right Hand Position:** position of the weapon when active, relative to the player's right hand.
- **Relative Rotation:** rotation of the weapon when active, relative to the player's right hand.
- **Bullet Damage:** the amount of damage a bullet causes when a target is hit.
- **Recoil Angle:** the angle that the camera and aim bounces when shooting the weapon.
- **Type:** the weapon type, choose between **SHORT** and **LONG.**
- **Mode:** the weapon firing mode, choose between **SEMI**, **BURST** or **AUTO.**
- **Burst Size:** the amount of shots sequentially fired when **BURST** mode is selected.
- **Mag:** the weapon mag capacity for bullets.
- **Total Bullets:** the maximum number of bullets for this weapon type that the player can carry.

THIRD PERSON ORBIT CAM

The **ThirdPersonOrbitCam** class manages all the third person camera features, such as orbiting around the player, FOV changes, and zooming in/out to avoid collision. This version also features lock on direction (for situations like turning on cover corners), dynamic horizontal rotation limit (used when peeking on wall corners), and camera bounce (for situations like shot recoil). This script has the following external parameters:

- **Player:** a reference to the player's game object.
- **Pivot Offset:** the default offset to point the camera when orbiting the player.
- **Cam Offset:** the default offset to relocate the camera when orbiting the player.
- **Smooth:** the default camera movement responsiveness factor.
- **Horizontal Aiming Speed:** the speed of camera orbit on the horizontal plane.
- **Vertical Aiming Speed:** the speed of camera orbit on the vertical plane.
- **Max Vertical Angle:** the maximum angle to limit camera orbit on the vertical plane.
- **Min Vertical Angle:** the minimum angle to limit camera orbit on the vertical plane.
- **X Axis:** the default horizontal axis input name (for joysticks).
- **Y Axis:** the default vertical axis input name (for joysticks).

This script uses the Unity's default *Mouse X* and *Mouse Y* input axes to control the camera orbit by mouse, alongside with the **X Axis** and **Y Axis** custom inputs for joysticks. Remember to adjust the joystick axis *sensitivity* and *dead zone* to match your needs.

BASIC BEHAVIOUR

The **BasicBehaviour** class manages which player behaviour is currently active or overriding the active one, and call its local functions. This behaviour also contains a basic setup and common functions used by all the player behaviours. This script has the following external parameters:

- **Player Camera:** a reference to the player camera's game object.
- **Turn Smoothing:** the turn speed when moving to match camera facing.
- **Sprint FOV:** the camera *Field of View* to change when player is sprinting.
- **Sprint Button:** the default sprint action input name.

This behaviour uses the Unity's default *Horizontal* and *Vertical* input axes to control the player movement.

MOVE BEHAVIOUR

The **MoveBehaviour** class corresponds to walk, run and jump behaviour, it is generally the default behaviour. This script has the following external parameters:

- **Walk Speed:** the player's default walking speed (controlled by a Blend Tree).
- **Run Speed:** the player's default running speed (controlled by a Blend Tree).
- **Sprint Speed:** the player's default sprinting speed (controlled by a Blend Tree).
- **Speed Damp Time:** the delay to change between the locomotion animations when changing speed.
- **Jump Height:** the default jump maximum height reached.
- **Jump Inertial Force:** the default inertial horizontal force applied to the player when on air, between jumping and landing. The higher it is, the longer the jump distance will be.

This behaviour uses the Unity's default *Jump* input axis to handle the jump action. The speed of locomotion can be controlled by the *Mouse ScrollWheel* axis.

AIM BEHAVIOUR

The **AimBehaviour** class handles the aim and strafe movements. This version also features in place aim and wall peeking (used for covering situations). The following external parameters are present:

- **Aim Button:** the default aim action input name.
- **Shoulder Button:** the default switch shoulders action input name.
- **Crosshair:** the default aiming crosshair. If no parameter is passed, no crosshair is shown.
- **Aim Turn Smoothing:** the speed of turn response when aiming to match the player's orientation with camera facing.
- **Aim Pivot Offset:** the offset to repoint the camera when aiming.
- **Aim Cam Offset:** the offset to relocate the camera when aiming.

ACKNOWLEGMENT

The author acknowledges some third-party assistance that facilitated the production of the demo examples within this asset. This includes the *Carnegie-Mellon University* mocap library, the *SoundBible* and *Freesound.org* websites for the free sound samples, and the fellows that contributes to the *Unity Answers* website.

DISCLAIMER

This document is part of the **Cover + Shooting System – Third Person Shooter** asset, that can be acquired through official channels – such as the *Unity Asset Store*.

If you obtained this asset trough unofficial ways, I kindly ask you to consider giving support for the great assets to come!

Did you enjoy the asset? Please consider leaving a review on the *Unity Asset Store*, it is really important and will be very appreciated!

Support contact

Author's page