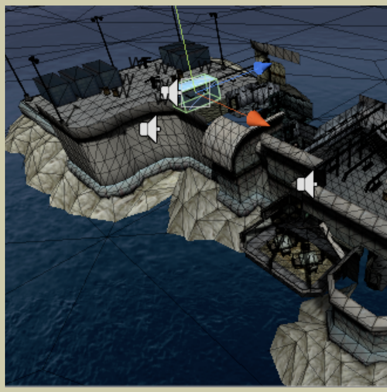


Creating a First Person Shooter (FPS)

Part 3

Author: Graham McAllister
Revised by: Jeff Aydelotte &
Amir Ebrahimi

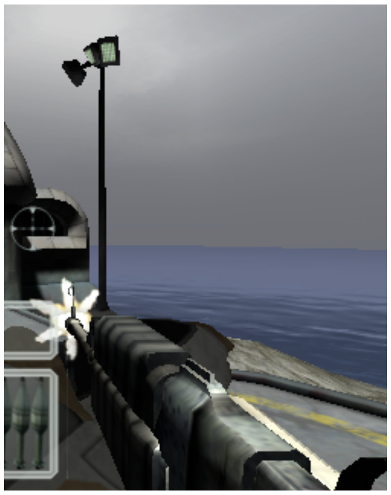
Time to complete: 3-4 hours

Last Revision: 10-July-2009

Contents

Part 3: Advanced FPS

This advanced-level tutorial extends upon the previous FPS tutorials by introducing game elements such as waypoints, enemy AI, ragdolls and animation.



Prerequisites

This tutorial assumes that you are familiar with the Unity interface and basic scripting concepts. Additionally, you should already be familiar with the concepts discussed in Part 1 and 2 of the FPS tutorial series.

Setting up

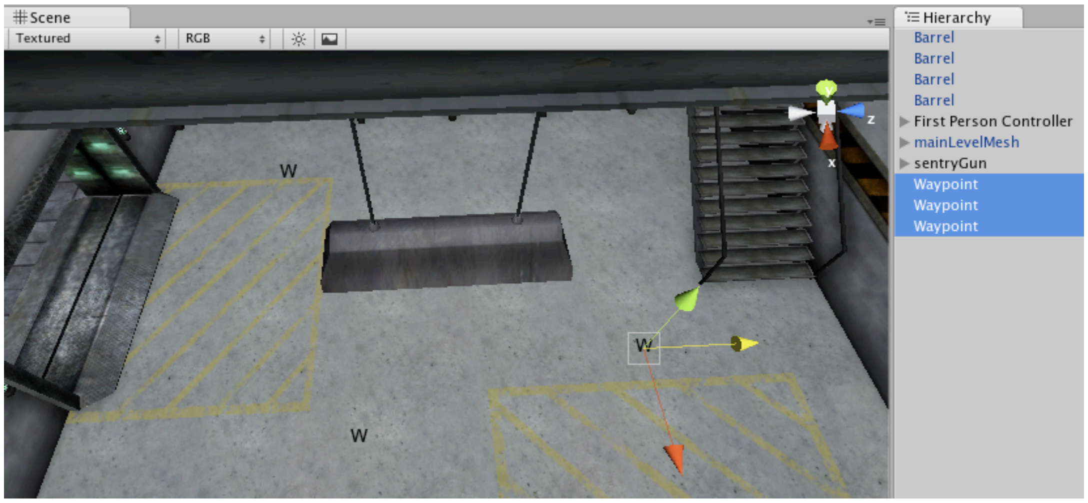
We're going to build upon the previous FPS tutorial, so we'll begin by opening that:

- Open the previous project (FPS Tutorial 2).

Waypoints

This section will introduce waypoints to our game, these are used to inform the robots of the path that they can walk around. Let's add three waypoints to our game:

- Create an empty game object and rename this to Waypoint. Make sure the game object is placed approximately one meter above the ground level.
- Add AutoWayPoint script to Waypoint. Notice how the empty game object now displays a W in the Scene View.
- Duplicate the WayPoint game object twice and arrange the waypoints in a tri-angle shape (position quite far apart).



- Check that the waypoints are visible to each other by firstly clicking on any waypoint then selecting Update Waypoint from the context menu of the AutoWayPoint script (the right-most button on the component in the Inspector View). A green line will be drawn between visible waypoints, a red line for those which don't have a clear line of sight.

We have now described the path that an enemy can walk around, now let's add an opponent to the scene.

Robot AI

This section will add an enemy robot to our scene.

- Select Robot Artwork/robot and drag it into the Scene View making sure all of the robot is above the ground.

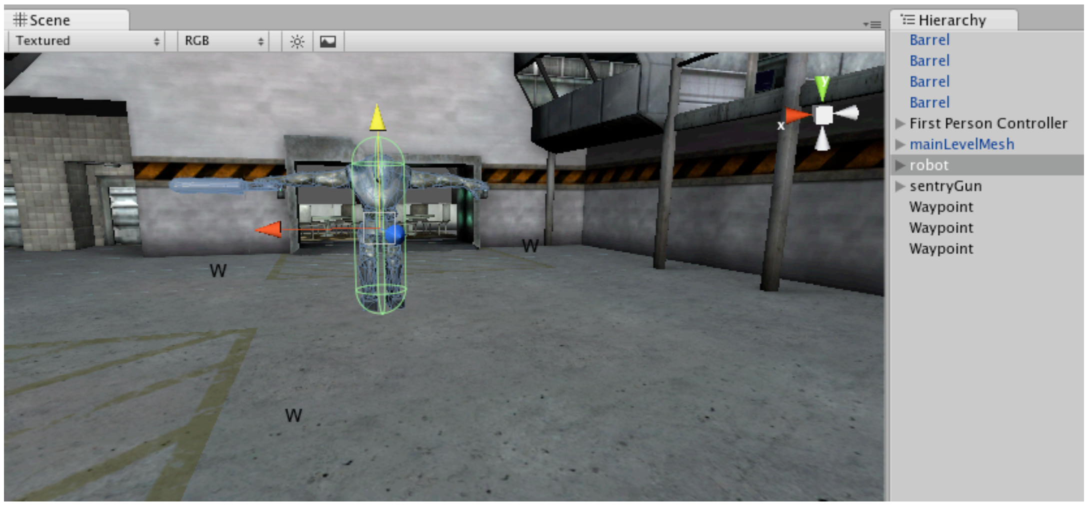
Let's give the robot some behavior.

- Add WeaponsScripts/AI to robot. In the AI script section or robot, assign the FPS controller as the target (so the robot knows who to hunt down).
- Add the AIAnimation script to the robot. This controls the animation of our robot (when to run, when to aim etc). This communicates with the AI script to find out what the AI script is currently doing, e.g. are we running, shooting etc. It then crossfades the animations to provide a smooth transition.

Now we need to make the robot controller object a little larger so that he doesn't intersect with the ground. This is because the character controller which is used to prevent the enemy from moving through walls is using a capsule to represent the AI. We need to make this capsule a little larger to match the actual graphics, this way he will not intersect with the ground anymore.

- Select the robot, then in the Inspector View modify the height and radius values of the Character Controller component so that it encloses the robot. Press play to make sure it works correctly.

- Try making the height and radius values smaller and larger to see the difference.



Now we need to give the robot the ability to fire his gun. The robot is continuously moving and animating. When he shoots his gun we play an animation on the hands and the graphical gun then spawn the rocket from transform relative to the robot. This transform does not animate, it is simply placed approximately at the point where the gun will be when the shoot animation fires the rocket.

- Create an empty game object and make it a child of the robot (use the hierarchy view).
- Rename the game object gun_spawn.
- Now we need to place it properly. In the transform inspector select Reset from the context menu, then move it forward in the z-axis.
- Add the rocket launcher script to the gun_spawn game object. Assign the rocket prefab to the projectile variable.
- Make a prefab of this robot, call it Robot. This will allow you to create more enemies easily.
- Play the game and check that the robot fires at you.

Although you can shoot the robot, it is not configured to take damage, we'll remedy this in the next section.

Robot Damage

- Attach the CharacterDamage script to the Robot prefab.
- Play the game, shoot the robot with the machine gun and he should disappear.

Ragdolls

Ragdolls emulate a natural skeletal bone structure, this will allow our robot to fall naturally when killed. This section will show how to setup a ragdoll.

- Firstly, create a new scene (*File -> New Scene*). We're going to setup our ragdoll in here. Save your current scene if necessary.
- Create a cube (resize if necessary), this will be used as a platform for our robot. Drag in the robot to the Scene View so that it is positioned above the cube.
- Remove the animation component. This is important otherwise the animations will interfere with the physics.
- Now use the ragdoll wizard: *Game object -> Create Other -> Ragdoll*.

A dialog box appears, now we have to assign all the bones to the correct slots in the wizard. We assign the bones by dragging the values from the Robot in the Hierarchy View onto the correct place in the dialog box. Expand the Robot gameobject in the Hierarchy View if necessary to reveal this.

- Firstly, rootHandle (Hierarchy View) maps onto root (in the dialog box). Drag rootHandle onto the placeholder beside root in the dialog box.
- Set **Total Mass** to 4

Legs

- Assign upleg_L to Left Hip.
- Assign lowleg_L goes to Left Knee.
- Assign heel_L goes to Left Foot.
- Repeat for the right leg.

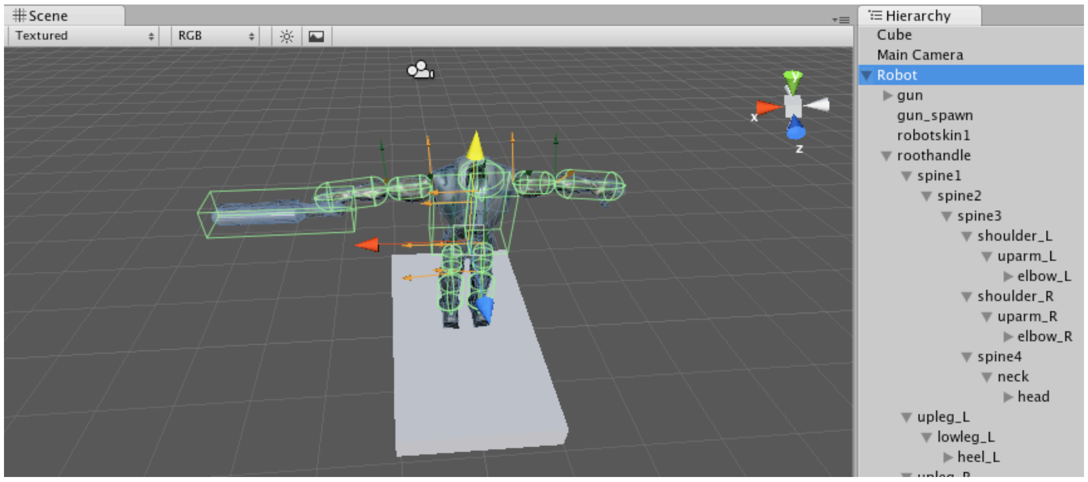
Upper body / Arms

- Assign upArm_L goes to Left Arm.
- Assign elbow_L goes to Left Elbow.
- Repeat for the right arm.
- Assign spine 3 goes to Middle spine.
- Finally, head goes to Head.
- Hit create then press play, the robot should fall, however the gun won't.

To make the gun fall:

- Select the gun in the Hierarchy View.
- Add a rigidbody component.

- Add a box collider. Adjust the size of the box collider to fit the gun, you'll need to adjust the center position also.



Now we put the fully rigged robot into a prefab. To create the prefab:

- *Assets -> Create -> Prefab.*
- Rename it to Robot-Ragdoll.
- Drag the robot (root level) from the hierarchy view into the prefab. This will make sure the gun is attached also.

Now when the robot is killed, we delete the old robot and instantiate the new ragdoll robot, this allows our robot to fall to the ground naturally.

Using the Ragdoll

- Open the original scene (no need to save the current scene), select the robot from the Hierarchy View and in the Character Damage inspector, drag in the Robot-Ragdoll prefab to the **Dead Replacement**.
- Play the game, shoot the robot and it should now fall down when shot.

You may want to tweak the mass of the robot to improve the look & feel of the ragdoll. You can do this in the ragdoll wizard, by changing the mass property. This is especially important in explosion forces where the robot is thrown into the air.

Sound

This section will add sound effects to our game.

Machine gun

- Drag *machineGunSingleShotLoopable* onto the MachineGun gameobject.
- Turn off **Play on Awake**.

Rocket Launcher

We want to attach the sound source of the rocket to the rocket so that the audio levels reduces as it flies away.

- Select the Rocket prefab.
- Add an Audio Source component to the prefab.
- Drag the *RocketLauncherFire* audio source to the **Audio Clip** property of the Audio Source.
- Adjust the **Rolloff Factor** so that the sound fades out faster (0.5 might be ok).

GUI

The GUI (Graphical User Interface) is responsible for giving feedback to the player on the current game status. Typical GUI elements include; number of bullets left, health, mission objectives etc.

In this game we have three main GUI elements.

Overlay

The overlay sets up the different zones of the GUI (health, ammo etc).

- Select the GUI Overlay texture in *GUI/GUIOverlay*.
- Select *GameObject -> Create Other -> GUITexture*, this will place the texture in the centre of the Game View.

We now need to position the overlay in the correct place:

- Modify the transform **Position** to 0, 0, 0.
- Modify the **Pixel Inset** to 0, 0, 256, 128 (in order **X**, **Y**, **Width**, **Height**).

Health

- Select *GUI/healthBar* in the Project View.
- Select *GameObject -> Create Other -> GUI Texture*.
- Set the transform **Position** to 0, 0, 0.
- The pixel inset values are 37, 83, 111, 29.

The healthBar texture is scaled in real-time to cover the correct amount of the rippled area beside the health icon.

Machine Gun

- Select *Game Object -> Create Other -> Text*.
- Set transform **Position** to 0, 0, 0.

- Set **Pixel Offset** to 188,109.

You can change the text if wanted (for now), a script will set text when the game is run.

Rockets

- Select *GameObject- > Create Other -> GUI Texture*.
- Set **Texture** to None.
- Drag *Misc Scripts/DrawRockets.js* onto the GUI Texture.
- Set **Rocket Texture** to *GUIRocket21*.
- Set **Pos Y** to 68.

- Finally, add the *FPSPlayer* script to the FPS controller, assign the GUIs to the properties in the FPSPlayer script section.

Finally...

We've nearly finished our tutorial, but let's add some additional sounds first.

Player Damage Sounds

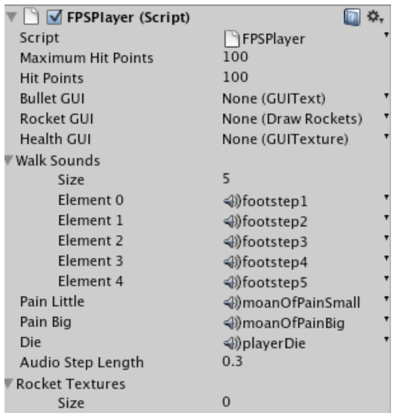
To the FPS Player script of the FPS controller game object assign the following sounds:

- *moanOfPainSmall*
- *moanOfPainBig*
- *playerDie*

Walk sounds

We can assign any number of walk sounds to our FPS controller (to avoid repetition), these are chosen randomly from a variable-sized list.

- Select FPS player. Expand the triangle for **Walk Sounds** and set **Size** to 5. Drag *footstep1-5* from *Sounds/* into each of the five elements.
- Add an Audio Source component to the FPS player (no need to do anything else).



FPSPlayer (Script)

Script	FPSPlayer
Maximum Hit Points	100
Hit Points	100
Bullet GUI	None (GUIText)
Rocket GUI	None (Draw Rockets)
Health GUI	None (GUITexture)
Walk Sounds	
Size	5
Element 0	footstep1
Element 1	footstep2
Element 2	footstep3
Element 3	footstep4
Element 4	footstep5
Pain Little	moanOfPainSmall
Pain Big	moanOfPainBig
Die	playerDie
Audio Step Length	0.3
Rocket Textures	
Size	0

Lights for moving objects

The scene in our game is lightmapped, however the characters and rigidbodies can not use this lightmap since they are moving. Thus, we need to setup some lights which only affect those objects and look similar to the lighting of the lightmap. For this purpose we use the lights culling mask.

- Create a directional light.

We'll alter the light's culling mask, this determines what the directional light is lighting.

- First goto *Edit* -> *Project Settings* -> *Tags*, click on User Layer 8 and type Lightmapped.
- Now select the directional light again and turn off Lightmapped.
- Select mainLevelMesh, change the Layer property to Lightmapped. Apply this change to all child objects when prompted.

The end of the beginning...

This completes our FPS tutorial series. Many key lessons for game development have been presented and will apply to other styles of games: not only FPS-style games. We hope that you have learned the fundamentals of game creation and are now motivated to design your own novel games with Unity.

Acknowledgments

Special thanks go to Joachim Ante (code) and Ethan Vosburgh (graphics) for their help in the making of this tutorial.

Download the complete FPS project from the Resources section to see a polished version of this tutorial.