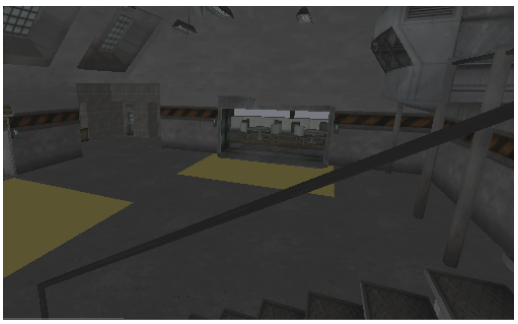
An Advanced FPS in Unity



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

Time to complete: 2 - 3 hours.

Author: Graham McAllister

Contents

1. Aims of this tutorial

This tutorial details the conclusion to our FPS game. On completion of this tutorial, you will have a fully-functioning FPS game with physics, enemy AI, ragdoll damage and character animation.

2. Prerequisites

You should already be familiar with the concepts discussed in both of the previous Unity FPS tutorials. This tutorial builds on the previous FPS tutorials,

This tutorial will not discuss the scripts as much as before, the reader is encouraged to read these in detail and experiment with them.

Note: any text that requires the user to take an action begins with a '-'.

3. Setting Up

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

- Open the previous project (FPS intermediate).

4. 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:

- Select Gameobject->Create empty, 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 triangle 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 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.

---

5. 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 behaviour.

- 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.

---

6. Robot Damage

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

NB need to use explosion-advanced script (not explosion) in the Explosion prefab.

---

7. 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), 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 Hierarchy View onto the correct place in the dialog box.

- Firstly, rootHandle (Hierarchy View) maps onto root (in the dialog box). Drag rootHandle onto the placeholder beside root in the dialog box.

Legs

- upleg_L goes to Left Hip. Expand the robot gameobject in the Hierarchy View if necessary to reveal this.

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

Upper body / Arms

- upArm_L goes to Left Arm
- elbow_L goes to Left Elbow
- upArm_R goes to Right Arm
- elbow_R goes to Right Elbow
- 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.

Let's adjust the mass of the robot, open the dialog again (GameObject -> Create other -> ragdoll, and adjust the mass to 4.

- 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.

8. 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. drag the RocketLauncherFire audio source anywhere in the inspector view, this will add it to the prefab.
- Adjust the rolloff value so that the sound fades out faster (0.5 might be ok).
- You can modify the explosion power also to make the missile weak or strong. A value of 50 creates a good visual effect.

9. 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 Game Object -> 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 to 0, 0, 0.
- Modify the pixel inset to 0, 0, 256, 128 (in order XMin, YMin, XMax, YMax).

Health

- Select GUI -> healthBar in the Project View.

- Select game object -> Create other -> GUI Texture.
- Set the transform position to 0, 0, 0.
- The pixel inset values are 37, 83, 148, 112.

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 to 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 GUIRocket from the Project panel.
- Select game object- > Create other -> GUI Texture.

- Add FPS player script to the FPS controller, assign the GUIs in the FPS player 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, set Size to 5. Drag footstep1 - 5 from Sounds/ into each of the five elements.
- Add an audio source to the FPS player (no need to do anything else).

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. It will ask if you want to make all child objects be in the Lightmapped layer. Say yes to that.

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 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.