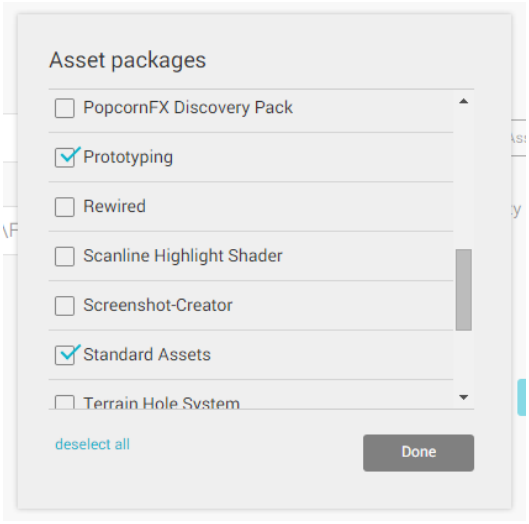
Tutorial – A Simple FPS

In this tutorial we're going to build a simple FPS game. There will be a weapon to find, an enemy to destroy, and a door to unlock.

This tutorial assumes that you are familiar with the basics of using Unity. It was created using Unity 5.5.0f3, but the instructions should be valid for other versions as well.

1. Create our project.

1. Open Unity start a new Project. Make sure that it's set as a 3D project, and include the "Standard Assets" and "Prototyping" packages before creating the project.

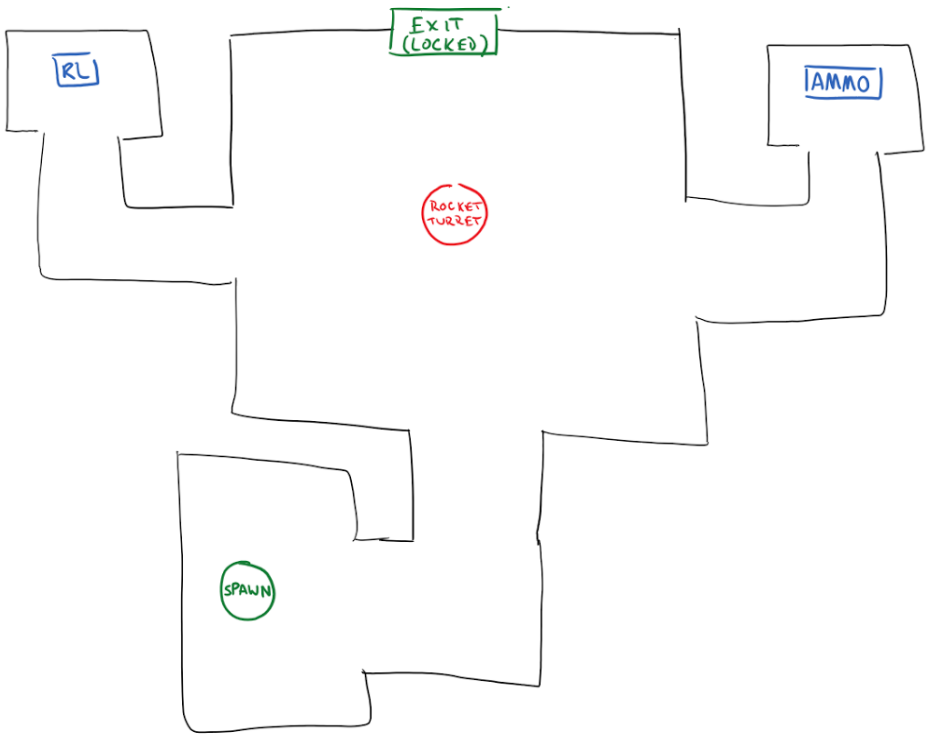


2. "Block-in" our level.

1. The first thing I'm going to do, before I build anything in the Editor, is do a sketch of my level's overall layout. This isn't going to be a work of art, it's just a planning step so that I'm clear on what I'm going to build. This helps me answer a few questions really easily:

    - What is the level's layout?
    - What things are in the level? Pickups, enemies, interactive objects, and so on.
    - What does the player have to do to win?

    Here's my layout sketch:



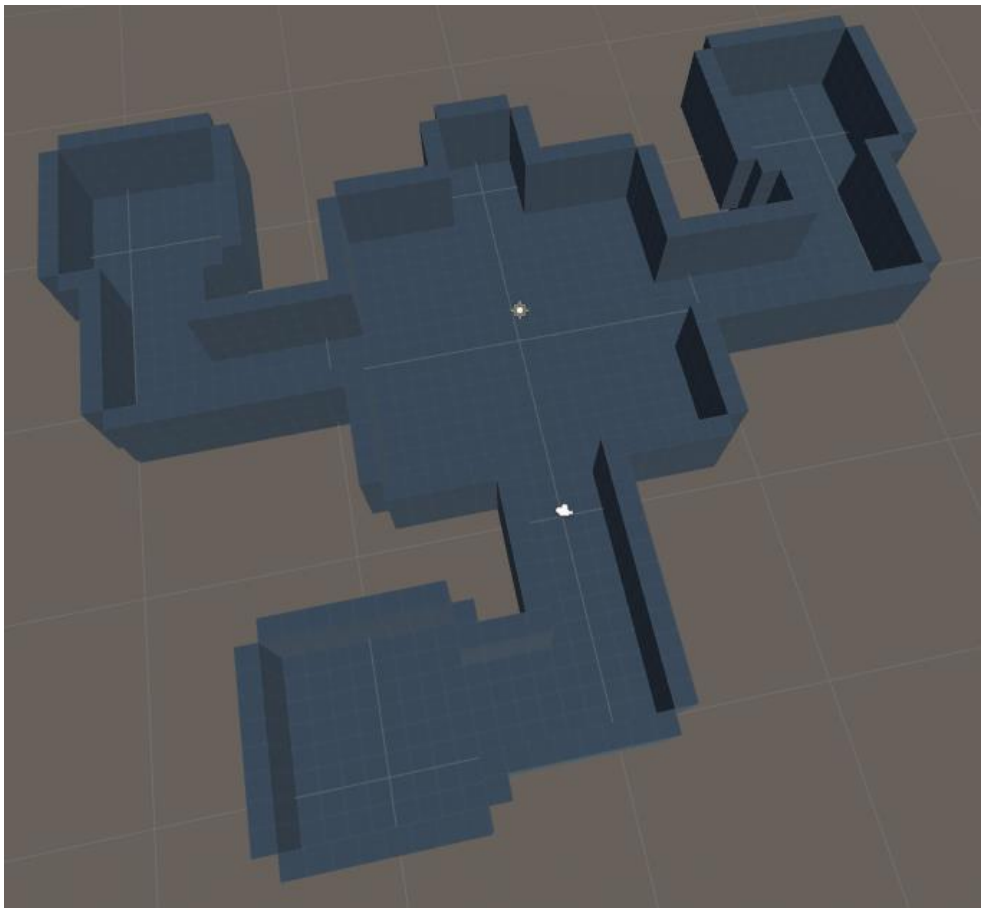
    It's pretty clear that I'm no artist! But, this only took a few moments to sketch, and it's really useful.

    - The green bits are the player spawn and the exit location. The exit is marked as locked, so the player will have to find a key to exit and win the level.
    - There's a red circle labelled "Rocket Turret". This is our enemy. When it is destroyed it will drop the key that opens the exit.
    - Finally, we have two blue things. "RL" is a rocket launcher pickup, and "AMMO" is some rockets.
    - You can also see the general layout of the level. There's a big open area in the middle dominated by the rocket turret. This is a dangerous area, and the player

won't want to stand still there as long as the turret is active. There are three cornered corridors coming off the main area with small rooms. Because the corridors aren't straight the rocket turret can't see into the rooms, making them safe spaces for the player.

2. Now we know what we're building, lets put our level together. I'm going to use Unity's provided "Prototyping" assets (Standard Assets/Prototyping/Prefabs) to assemble my level so that it roughly matches the layout I've sketched.

   Here's my result:



   You'll notice that it doesn't *exactly* match my sketch. That's because I've built it with standard pieces from the prototyping package, and I've stuck to using chunks of their standard sizes. The fundamental design is the same, though.

   The other difference that you may notice is that I've put some floor and walls on the far side of where the exit is going to go, so the player can step through it without falling into a void.

3. Chances are that while we were adding all of the walls and floors to our scene we've ended up with stuff just *everywhere* in our Hierarchy tab. Lets make life a little easier for ourselves.

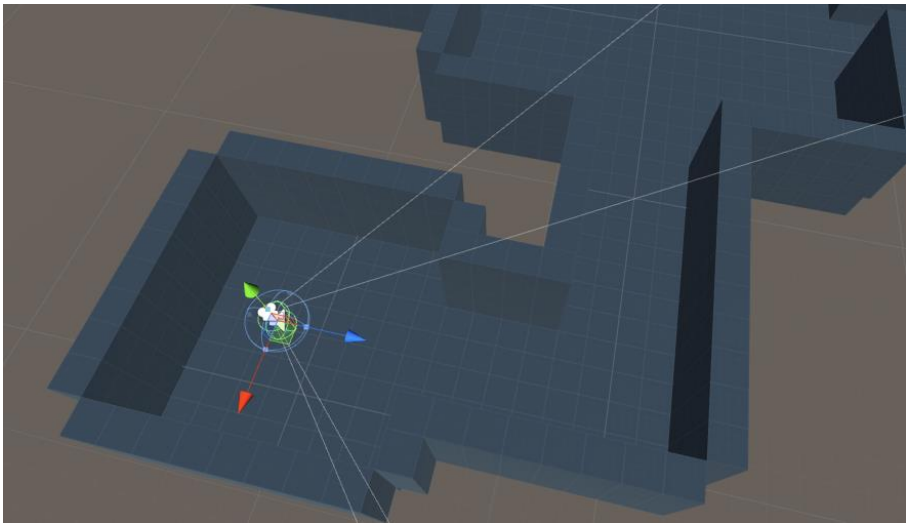
   Create a new empty GameObject, reset it's Transform component by righting click it's label in the Inspector and selecting "Reset", and rename it to "Level Gemoetry". Then, select all of your walls and floors and make them children of that GameObject. Now you can collapse it in the Hierarchy, making it much easier to work with the scene moving forwards.

4. If you haven't already, make sure you save the scene! Make sure you save regularly – this is the only time the tutorial will tell you to save!

3. Make it playable!

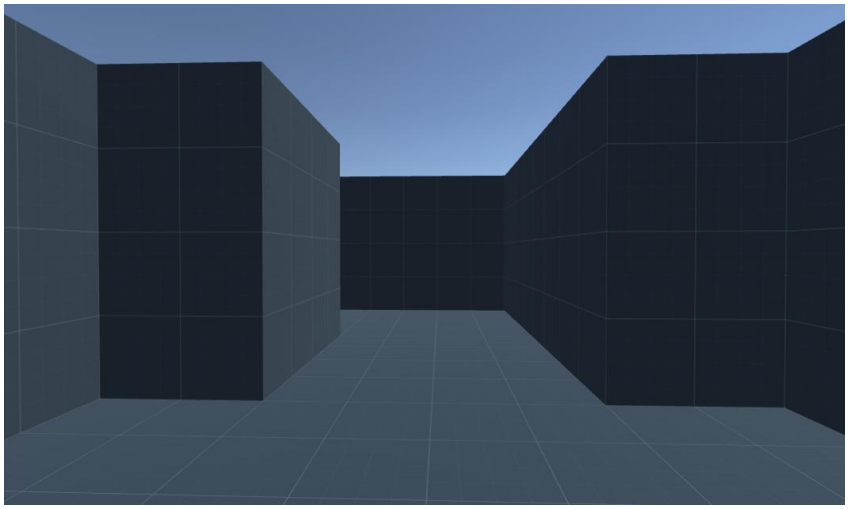
1. We're going to use the Standard Assets FPSController for our game's basic controls. Find the prefab (Standard Assets/Characters/FirstPersonCharacter/Prefabs/FPSController) and add it to your scene in the place indicated on the layout sketch.

2. Rotate your FPSController GameObject so that it's looking towards the corridor, instead of at the wall.



3. We only want one Camera and AudioListener in the scene, so delete the "Main Camera" GameObject that was created by default.

4. Press the "Play" button and make sure everything works as you expect.



4. Adding a weapon

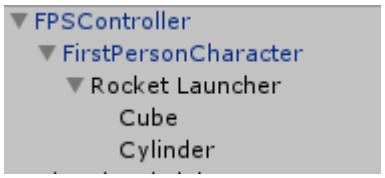
1. Being an FPS, one of the most important things we need to have is a gun. We're going to make our gun as a child of the FPSController GameObject, so start by selecting your FPSController/FirstPersonCharacter GameObject in the scene and then going to GameObject -> Create Empty Child.

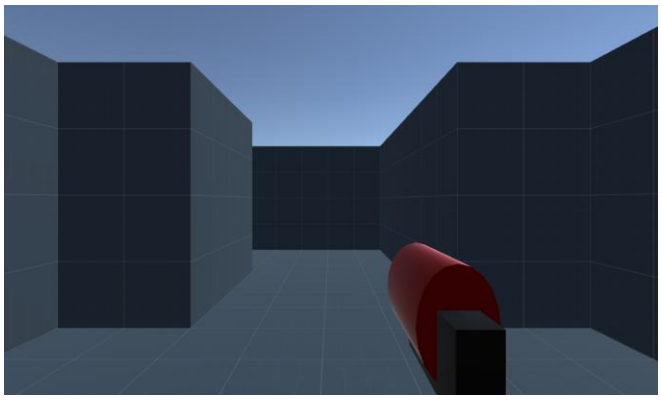
   Note: *It's important that we create our gun as a child of the FirstPersonCharacter GameObject rather than of FPSController itself. That is because FPSController turns, but it does not tilt.*

2. Select the newly created GameObject and rename it to "Rocket Launcher", and also Reset its Transform component.

   Your FPSController GameObject under the Hierarchy tab should now look like this:

3. Go ahead and give your Rocket Launcher some visual components, with shapes and Materials taste. Add these as children of the "Rocket Launcher" GameObject, and position the "Rocket Launcher" GameObject so that it can be seen just in front of the camera.

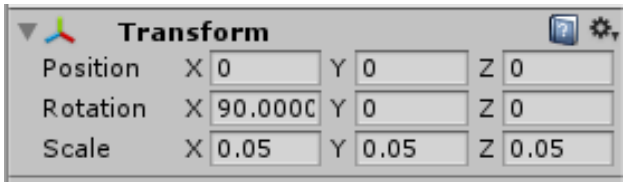




4. Remove any Collider components from your Rocket Launcher's visual components. They aren't needed, and may cause undesired behaviour.

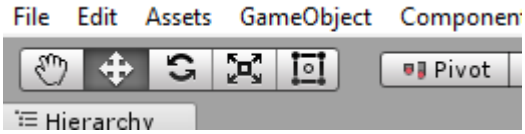
5. Now we need a rocket! Go to GameObject -> Create Empty, and rename the new GameObject to "Rocket".

6. Next create a Cylinder (GameObject -> 3D Object -> Cylinder). Make this a child of the "Rocket" GameObject, then set its Transform matching the following:

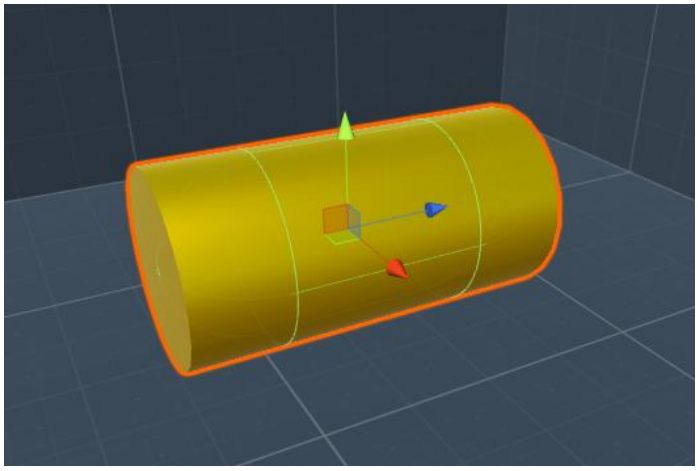


7. Add a Material as desired.

8. Make sure that the gizmo is in Move mode…



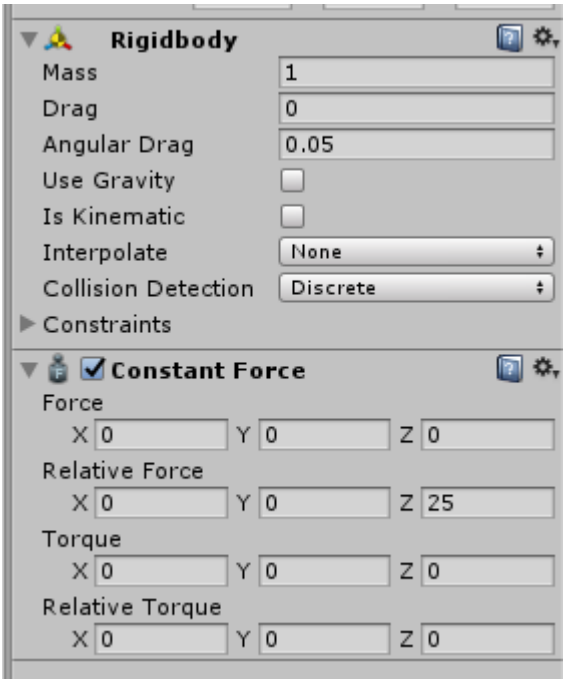
… then select the "Rocket" GameObject and make sure that the blue arrow, which represents the Z axis of the object, is pointed along the cylinder.



Along with step 6, this ensures that the visible part of the rocket object is aligned with the rocket's forward movement direction, because in Unity the Z axis is considered to be forwards.

9. With the "Rocket" GameObject still selected, add a Rigidbody Component and a ConstantForce Component (both through the Component -> Physics menu). There are how we will make the Rocket move.

   Configure the two components to match the below:



   Note that we haven't had to add a collider, because the Cylinder child object comes with a Capsule collider by default.

10. We want to use lots of Rocket GameObjects, so we want to turn this into a Prefab.

    In your Project panel, create a folder called "Prefabs". Then drag your "Rocket" GameObject from your Hierarchy panel into the Prefabs folder.

    Then, delete the "Rocket" GameObject from your scene.

11. Create a "Scripts" folder in your Project panel, and then create a C# script called "RocketLauncher".

    Open the script file to edit it.

12. Complete the script as follows:

```
using UnityEngine;

public class RocketLauncher : MonoBehaviour {

    // Is this RocketLauncher controlled by the player?
    public bool playerControlled = true;

    // How many seconds between shots for this Rocket Launcher?
    public float fireInterval = 3.0f;

    // A timer to tell if we can fire again.
    private float fireTimer = 0;

    // A reference to a Rocket prefab so that we can spawn it.
    public GameObject rocketPrefab;

    // A position offset for where the rocket is spawned. This is to
    // make sure we can spawn rockets outside of the gun.
    public Vector3 spawnOffset;

       void Start () {
        // We should start ready to shoot
        fireTimer = fireInterval;
       }

       void Update () {
        // Advance our timer
        fireTimer += Time.deltaTime;

        // Have we waited long enough to fire again?
        if (fireTimer >= fireInterval)
        {
            // If we are player controlled, has the fire button been pressed?
            if (playerControlled && Input.GetButtonDown("Fire1"))
            {
                Fire();
            }
        }

       }

    public void Fire()
    {
        // Spawn a new rocket from our prefab.
        GameObject rocketInstance =
                               GameObject.Instantiate<GameObject>(rocketPrefab);

        // This applies the spawn offset relative to the gun's position.
        rocketInstance.transform.position = transform.position +
            transform.right * spawnOffset.x +
            transform.up * spawnOffset.y +
            transform.forward * spawnOffset.z;

        // Rotate them to match the gun.
        rocketInstance.transform.rotation = this.transform.rotation;

        // Reset the shoot timer
        fireTimer = 0;
    }
}
```

13. Save the script, check for any compile errors, and correct them.

14. Select the "Rocket Launcher" GameObject in your scene, and attach the RocketLauncher script to it. In the "Rocket Prefab" field, add a reference to your Rocket prefab.

15. Press the Play button and make sure everything works as you expect. You should now be able to fire a small rocket once every 3 seconds by clicking the left mouse button.

5. Handling ammo

1. Currently our Rocket Launcher has unlimited ammo. That's not how an FPS usually works – we want to make our players find extra ammo in the levels.

   Return to your RocketLauncher.cs script file, and at the top with the other variables add an ammo variable as follows:

```
    // How many rockets does the player have?
    public int ammo = 5;
```

   Then update the Fire() method as follows:

```
    public void Fire()
    {
        // Only shoot if we have ammo.
        if (ammo <= 0)
        {
            return;
        }

        // Spawn a new rocket from our prefab.
        GameObject rocketInstance =
                                GameObject.Instantiate<GameObject>(rocketPrefab);

        // This applies the spawn offset relative to the gun's position.
        rocketInstance.transform.position = transform.position +
            transform.right * spawnOffset.x +
            transform.up * spawnOffset.y +
            transform.forward * spawnOffset.z;

        // Rotate them to match the gun.
        rocketInstance.transform.rotation = this.transform.rotation;

        // Reset the shoot timer.
        fireTimer = 0;

        // Subtract from our ammo.
        ammo -= 1;
    }
```

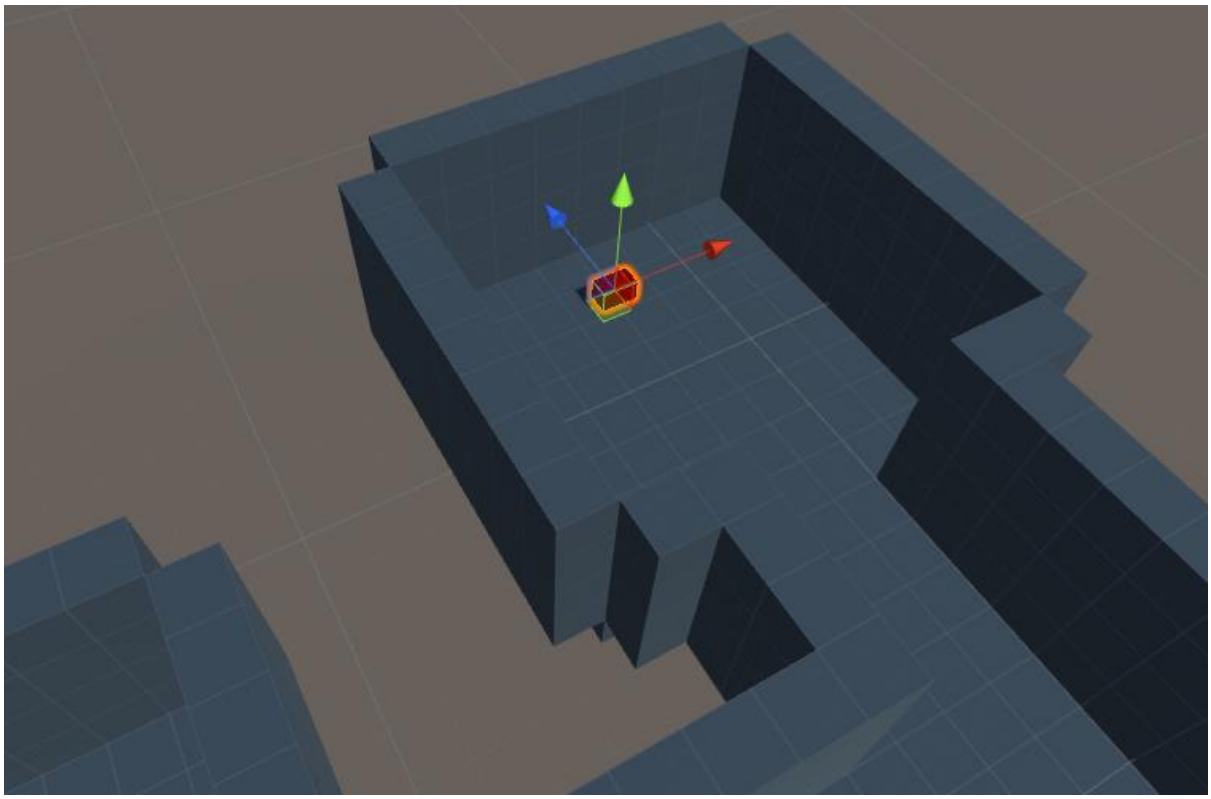
2. Save the script and fix any errors. Then press the Play button again and make sure you can only fire 5 shots.

   Note: *When testing things like this, save yourself time by changing public values in the Inspector. Instead of waiting 3 seconds per shot, adjust the fireInterval down to half a second or less.*

3. The next step is adding an Ammo pickup to our game.

   Start by creating a cube (GameObject -> 3D Object -> Cube). Name it "Rocket Ammo Pickup", give it your desired shape and material, and add it to the room where "Ammo" is indicated in our level sketch.

   Here's mine:



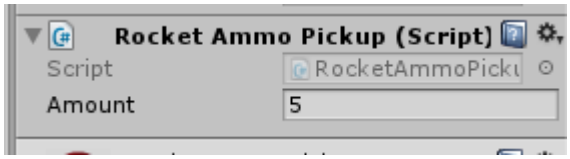
4. With the "Rocket Ammo Pickup" GameObject still selected, look in the Inspector and tick the "Is Trigger" checkbox.

5. In your Scripts folder in the Project panel, create a new C# script called "RocketAmmoPickup".

6. Open and complete the script as follows:

```
using UnityEngine;

public class RocketAmmoPickup : MonoBehaviour {

    // How many rockets is this pickup worth?
    public int amount = 5;

       void OnTriggerEnter(Collider other)
    {
        // See if the GameObject that collided with us has a RocketLauncher
        // component attached to any of its children.
        RocketLauncher rocketLauncher =
                        other.gameObject.GetComponentInChildren<RocketLauncher>();

        // If it does, then...
        if (rocketLauncher != null)
        {
            // ... add some ammo to that RocketLauncher ...
            rocketLauncher.ammo += amount;

            // ... and deactivate ourselves.
            gameObject.SetActive(false);
        }
    }
}
```

7. Save the script and fix any errors, and then add the RocketAmmoPickup script to your GameObject.



8. Play the game again to test it. Fire all 5 of your shots, then pick up the ammo and check that you can fire another 5.

   Stop the game and play it again, this time with the "Rocket Launcher" GameObject selected and visible in the Inspector. Make sure that the "Ammo" variable behaves as expected when you pick up the ammo.

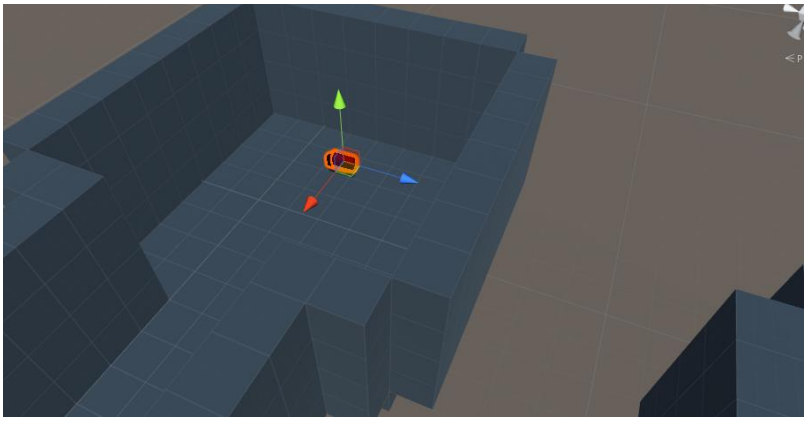
9. Turn your "Rocket Ammo Pickup" GameObject into a Prefab by dragging it from your Hierarchy panel into the Prefabs folder in your Project panel. Now you can easily add more to your game as needed.

6. Picking up guns

1. A major part of FPSs is picking up new guns. We'll make our Rocket Launcher collectable now.

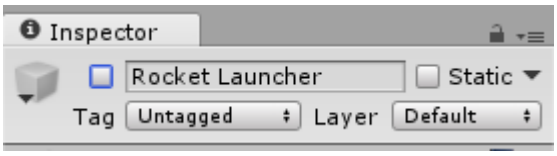
   Start by making a GameObject in the scene that looks similar to your existing RocketLauncher, calling it "Rocket Launcher Pickup", and putting it in the room marked with "RL" on our sketch.

   Here's mine:



   I made my "Rocket Launcher Pickup" GameObject by copying my "Rocket Launcher" GameObject, removing the RocketLauncher script, and scaling it up a bit.

2. Turn off the player's existing "Rocket Launcher" GameObject. It won't be re-activated until they collect this pickup! Select the "Rocket Launcher" GameObject, and deactivate it by unticking the checkbox at the top of its Inspector.



3. Now, we need a script that keeps track of which weapons the player has collected, and which one they are currently using.

   Create a new script called "WeaponSelector", and complete it as follows:

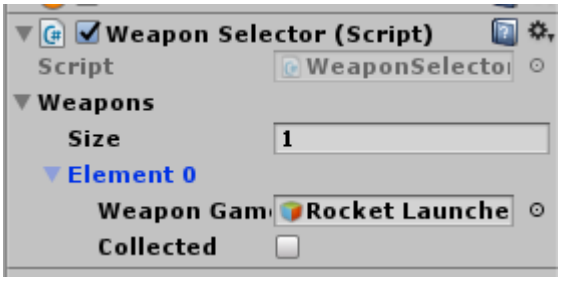
```
using System.Collections.Generic;
using UnityEngine;

public class WeaponSelector : MonoBehaviour {
    // This defines a custom class that we're using to associate a reference
    // to a weapon along with whether or not it has been collected.
    [System.Serializable]
    public class SelectableWeapon
    {
        public GameObject weaponGameObject;
        public bool collected = false;
    }

    // This is a list of that class. This is where the data is actually
    // stored.
    public List<SelectableWeapon> weapons = new List<SelectableWeapon>();

       void Update () {
        // Here we have one button per weapon, and select the weapon if the
        // associated button is pressed. Alternatively, we could have next/
        // previous weapon buttons.
              if (Input.GetButtonDown("SelectWeapon1"))
        {
            SelectWeapon(0);
        }
        else if (Input.GetButtonDown("SelectWeapon2"))
        {
            SelectWeapon(1);
        }
        else if (Input.GetButtonDown("SelectWeapon3"))
        {
            SelectWeapon(2);
        }
        else if (Input.GetButtonDown("SelectWeapon4"))
        {
            SelectWeapon(3);
        }
    }

    public void CollectWeapon(string collectedWeaponName)
    {
        // Look at each weapon in our list of selectable weapons...
        for (int i = 0; i < weapons.Count; i++)
        {
            // ... and if we find a match, set it's "collected" variable to
true...
            if (weapons[i] != null & weapons[i].weaponGameObject.name ==
                                                                collectedWeaponName)
            {
                weapons[i].collected = true;

                // ... and also select it.
                SelectWeapon(i);
            }
        }
    }
```

```
    // Select the weapon at a specified index of the collection.
    public void SelectWeapon(int selectedIndex)
    {
        // Make sure that the selection is valid - it must be in range and
        // must refer to a weapon that has been collected.
        if (selectedIndex < 0 ||
            selectedIndex >= weapons.Count ||
            weapons[selectedIndex] == null ||
            weapons[selectedIndex].collected == false)
        {
            return;
        }

        // Activate the selected weapon, deactivate all others.
        for (int index = 0; index < weapons.Count; index++)
        {
            if (weapons[index] != null)
            {
                weapons[index].weaponGameObject.SetActive(index ==
                                                          selectedIndex);
            }
        }
    }
}
```

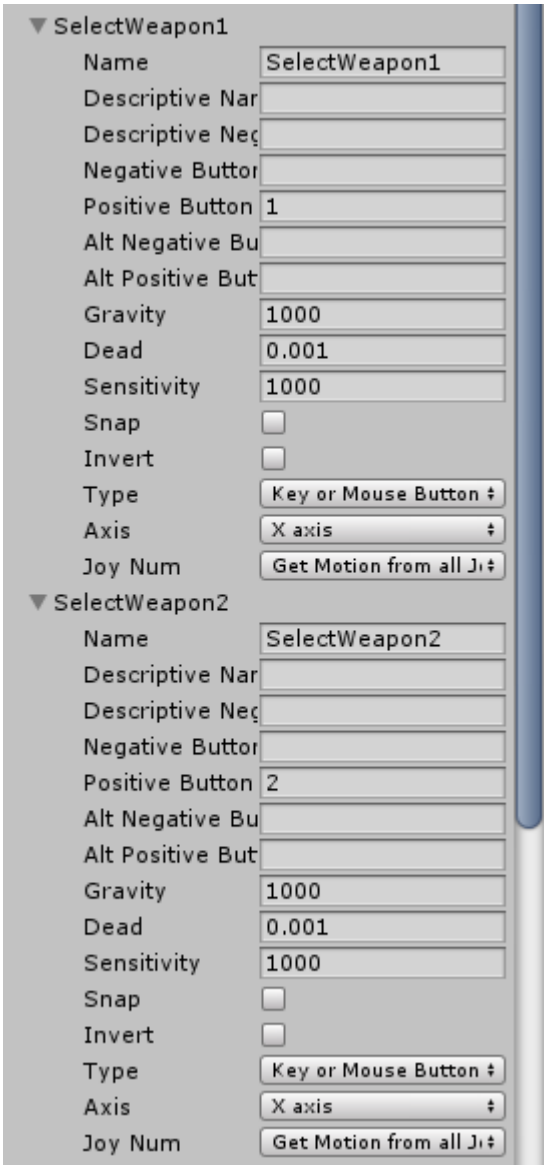
4. Save the script and address any compiler errors.

5. Add the WeaponSelector Component to your "FPSController/FirstPersonCharacter" GameObject.

   Add your "Rocket Launcher" GameObject to the Weapons list, leaving "Collected" un-ticked.



6. In the Unity Editor, go to Edit -> Project Settings -> Input. Add four new inputs, called "SelectWeapon1" through to "SelectWeapon4". Clear the other settings, and set the "Positive Button" fields to the corresponding number keys as follow:



These are the weapon selection buttons for the WeaponSelector. We are not going to use them because we only have a single weapon, but because the script uses them they must exist.

7. Now it's time to make a script that lets us collect the Rocket Launcher. In your Scripts folder create a new C# script called "WeaponPickup", and complete it as follows:

```
using UnityEngine;

public class WeaponPickup : MonoBehaviour {

    // The name of the weapon this should enable upon collection.
    // It defaults to an empty string.
    public string weaponName = string.Empty;

    void OnTriggerEnter(Collider other)
    {
        // Does the GameObject that collided with us have a WeaponSelector?
        WeaponSelector weaponSelector =
                    other.gameObject.GetComponentInChildren<WeaponSelector>();
        // If there is a WeaponSelector, tell it to mark the matching weapon
        // as collected.
        if (weaponSelector != null)
        {
            weaponSelector.CollectWeapon(weaponName);
            gameObject.SetActive(false);
        }

    }
}
```

8. Save the script and handle any compiler errors, then attach it to your "Rocket Launcher Pickup" GameObject.

   Set the "Weapon Name" field to "Rocket Launcher". Note that this needs to exactly match the name of the weapon GameObject referenced by the WeaponSelector script!

   Also add a Box Collider (Component -> Physics -> Box Collider), size it appropriately, and set "Is Trigger" to true. This is so that the OnTriggerEnter(…) method gets called.

9. Play the game again to test it. You should start without a gun, then get the weapon by walking over its pickup.