Raycasts

Objective

To make a basic FPS using projectiles and raycasts.
To show how invisible rays can be used to get information from an object at a distance.

First Scene

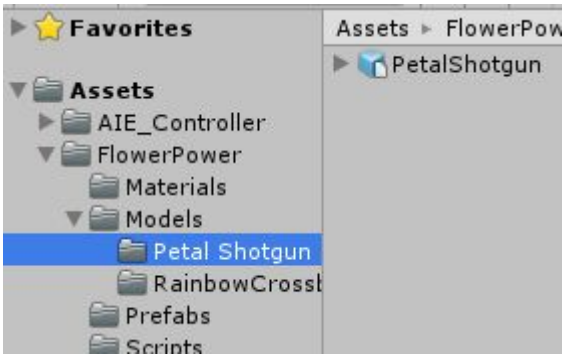
Create a basic FPS projectile based gun with explosive force.

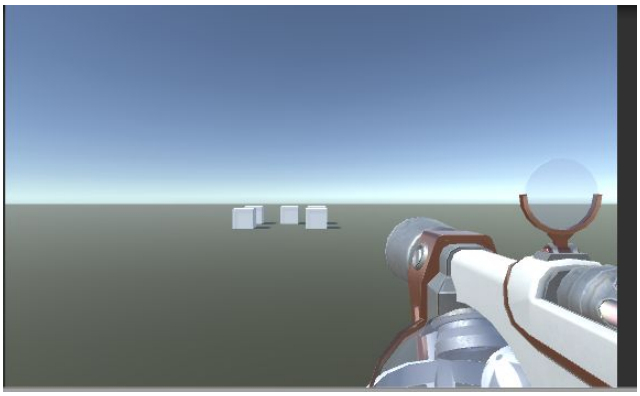
Setup

- Make a large plane as the ground
- Place several random cubes in the scene each with rigid bodies
- Import the FPS player controller and the Flower power gun packages.
- Delete the Main Camera

Lesson

Place the **petalGrenadeLauncher** prefab and parent it to the player's camera. Placing it so it is in the correct position.

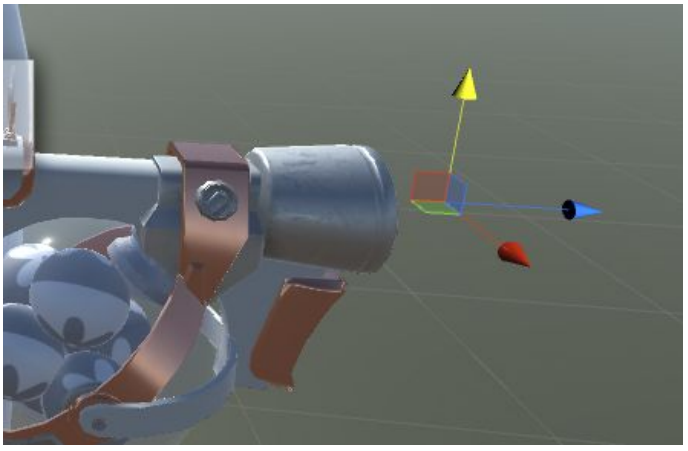




Create a new script called **FireGrenade** and place it on the gun.

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

public class FireGrenade : MonoBehaviour {

    public GameObject grenadeLaunchPoint;
    public GameObject flowerGrenade;
    public float launchPower = 2;

    // Update is called once per frame
    void Update()
    {
        //If the player presses the left mouse button
        if (Input.GetMouseButtonDown(0))
        {
            //Create a grenade
            GameObject GO = Instantiate(flowerGrenade,
                grenadeLaunchPoint.transform.position, Quaternion.identity) as GameObject;

            //Add force to the grenade's rigidbody to push it forward
            GO.GetComponent<Rigidbody>().AddForce(
                grenadeLaunchPoint.transform.forward * launchPower, ForceMode.Impulse);
        }
    }
}
```

Create an empty GameObject on the gun, and name it **GrenadeLaunchPoint**



From the inspector attach the **GrenadeLaunchPoint** and the **PetalGrenade** (found in the prefabs folder)

Result

You should be able to fire grenades around the level that explode and push rigid bodies around.

Looking at raycasts

Basic setup of raycasts using a raycast from the mouse position to the game world.

Setup

- Same as previous Lesson
- Deactivate the **PetalGrenadeLauncher**
- Add the **RainbowCrossbox** (in the prefabs folder) the say we we did the **PetalGrenadeLauncher**
- Name each of the cube in your scene a different name (Cubey 1, Mr Cubey, Sir Cubealot, Cubey Mc Cubeface etc...)
- Create a new script called **FireRay** and place it on the main camera

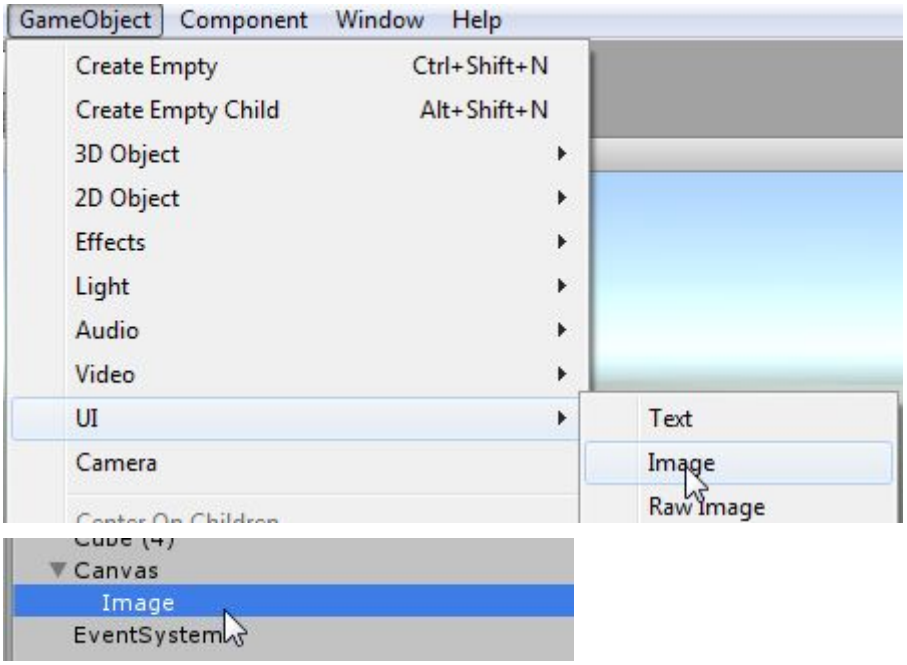
Lesson

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

public class FireRay : MonoBehaviour {

    // Update is called once per frame
    void Update()
    {
        //A Physics Hit object to store info we are going to get about where the ray hit.
        RaycastHit hit;

        //The distance of the ray that we are using
        float distanceOfRay = 100;

        //Cast the ray and check if it hits anything
        if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, distanceOfRay))
        {
            Debug.Log(hit.transform.name);
        }

        //Draw a ray in the editor
        Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * distanceOfRay);
    }

}
```

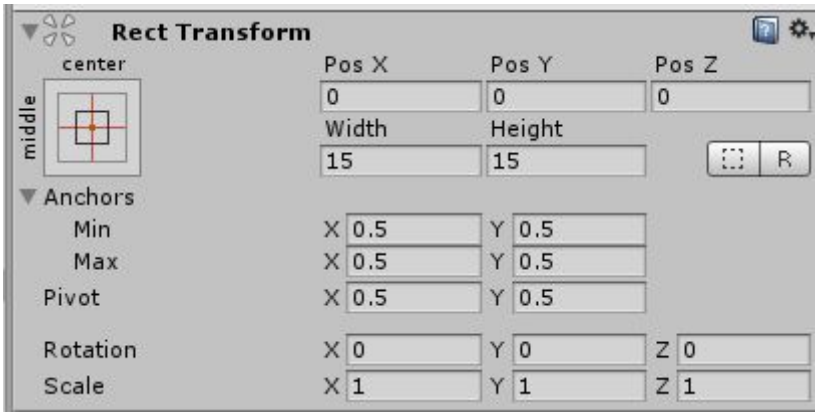
Now, if you run the game the Console should tell you the name of the game you are looking at.

But we need a crosshair so we can see exactly where we are looking.

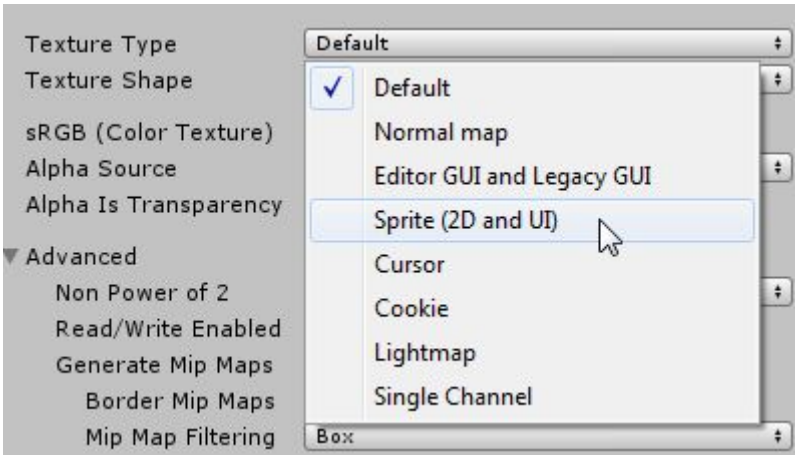
We need to add a **Canvas** and an **Image** to our game, to do that go to **GameObject -> UI -> Image**



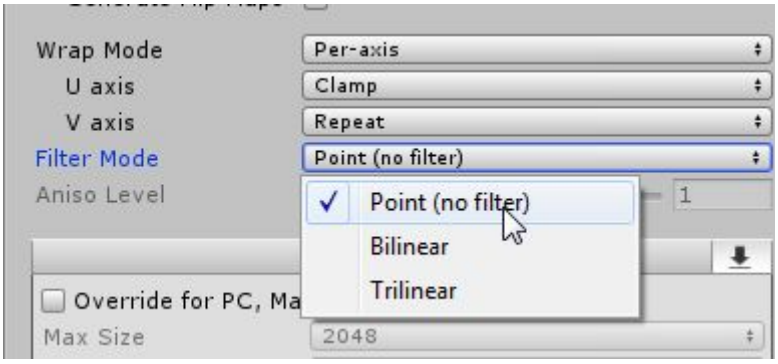
This will create the Canvas and the Image for us in the Hierarchy.
Rename the Image to **Crosshair**
Change the **Rect Transform** of the Crosshair to the following values



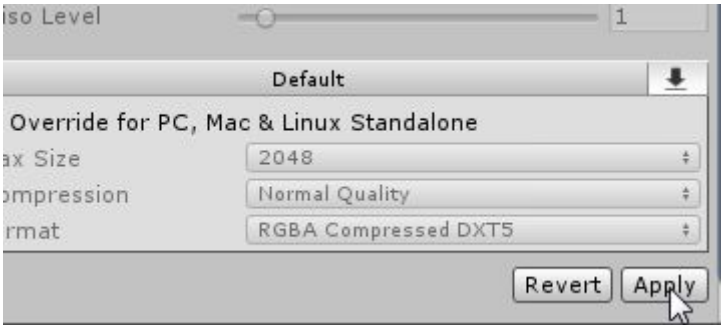
We have provided you with a Crosshair image, Import it into your game
Select the Crosshair Image and change **Texture Type** from Default to **Sprite (2D and UI)**



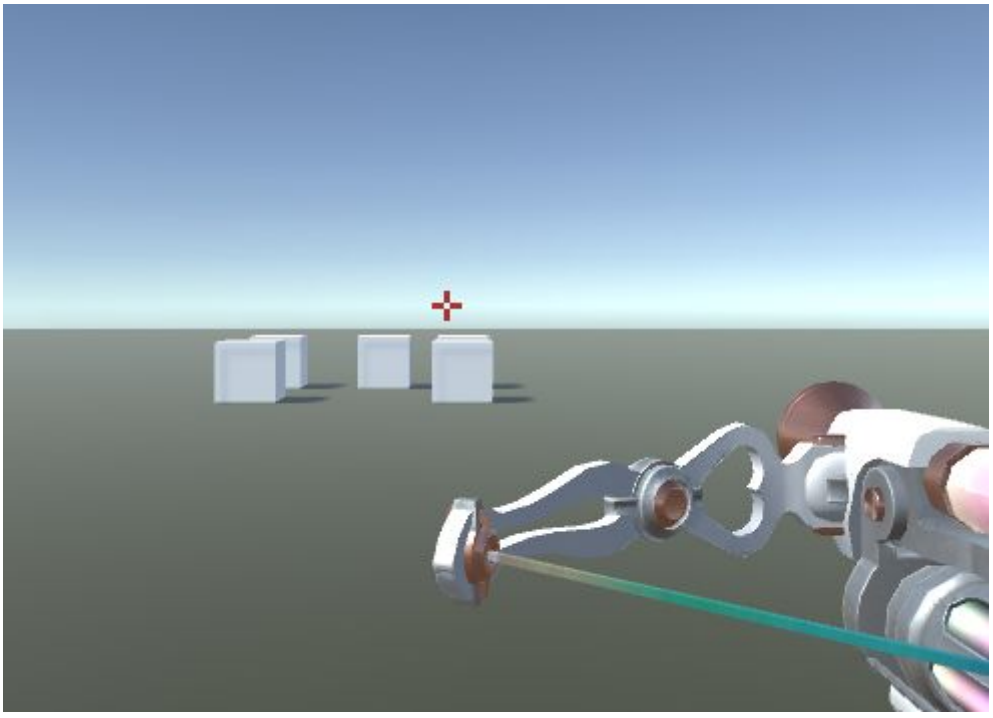
Set the **Filter Mode** to **Point(no filter)**



Click the **Apply** button



You should now have a crosshair in your game



Result

When the game is run you should be able to mouse-over the primitives in the scene and get their name to appear in the console.

First Controlling the object

Now that we have access to the object we can do things with it.

Setup

Same as previous

Lesson

All code is the same as above but will go inside this if statement with the added

Here are 5 different things we can do with our script.
Once you have written all 5 of these. Comment out 4 of them and test **one line at a time**

```
    //Cast the ray and check if it hits anything
    if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out hit, d
    {
        //Debug.Log(hit.transform.name);

        //if the left mouse button is clicked
        if (Input.GetMouseButtonDown(0))
        {

            //Move the object
            hit.transform.position = hit.transform.position + Vector3.up;

            //Scale the object
            hit.transform.localScale = hit.transform.localScale * 0.9f;

            //deactivate the object
            hit.transform.gameObject.SetActive(false);

            //Destroy the object
            Destroy(hit.transform.gameObject);

            //Change the colour of the object
            hit.transform.gameObject.GetComponent<Renderer>().material.color =
                new Color(Random.Range(0f, 1f), Random.Range(0f, 1f), Random.Range(0f, 1f));
        }
    }

    //Draw a ray in the editor
    Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * distanceOfRay);
}
```

Once you do you can see how you can take an object that your raycast hits and tell it to do things like

- Change its position
- Change its size
- Deactivate it
- Destroy it
- Change its colour
- And much much more

Results

When you run through each of these, you should be able to see how they can access the gameobject that the raycast hit.

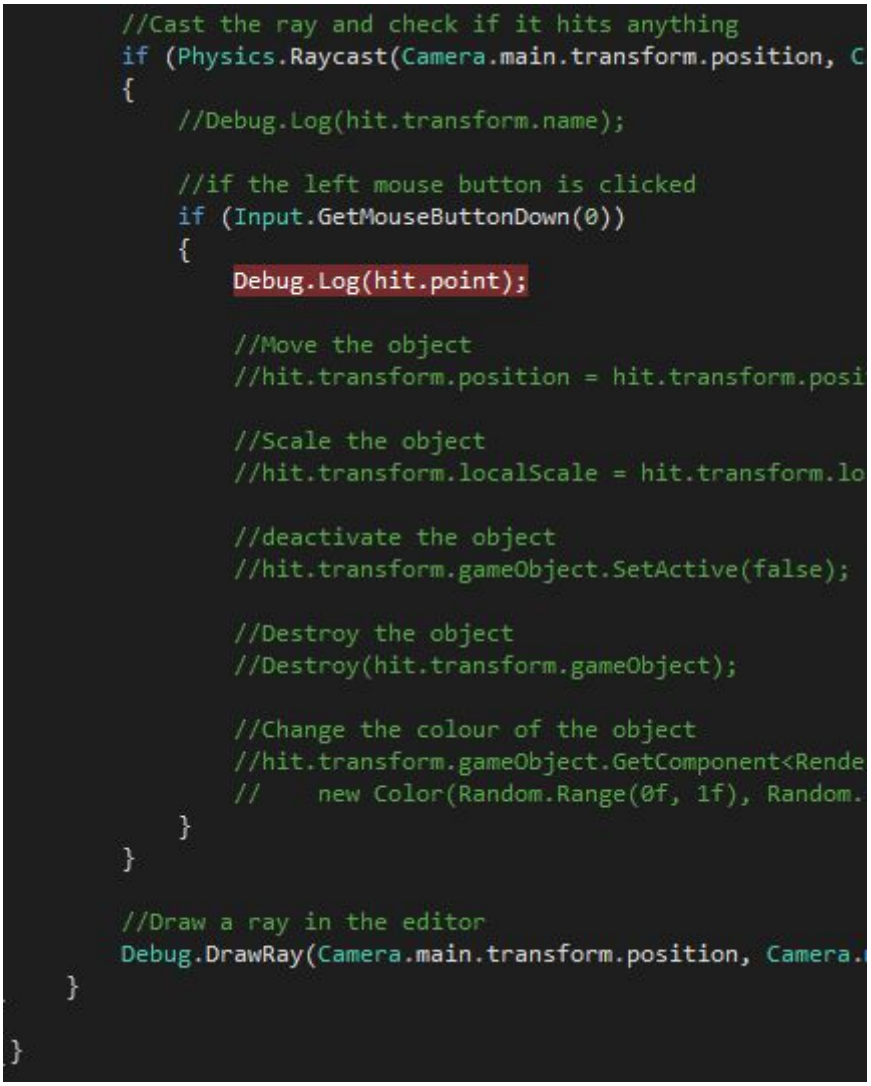
Using raycasts for position

Showing that we can get the position of where the ray hits.

Setup

Same as previous

Lesson



```
        //Cast the ray and check if it hits anything
        if (Physics.Raycast(Camera.main.transform.position, C
        {
            //Debug.Log(hit.transform.name);

            //if the left mouse button is clicked
            if (Input.GetMouseButtonDown(0))
            {
                Debug.Log(hit.point);

                //Move the object
                //hit.transform.position = hit.transform.posi

                //Scale the object
                //hit.transform.localScale = hit.transform.lo

                //deactivate the object
                //hit.transform.gameObject.SetActive(false);

                //Destroy the object
                //Destroy(hit.transform.gameObject);

                //Change the colour of the object
                //hit.transform.gameObject.GetComponent<Rende
                //    new Color(Random.Range(0f, 1f), Random.
            }
        }

        //Draw a ray in the editor
        Debug.DrawRay(Camera.main.transform.position, Camera.
    }

}
```

Run the game and shoot at things

Results

Now when you run the game the console will show the vector 3 position of where the raycast hit

Place a marker at this point

Now we know the ray hit point we can do things with it.

Setup

- Create a sphere
    - Change its scale to 0.5,0.5,0.5
    - Give it a brightly coloured material
    - Remove its collider
    - Rename it to **HitMarker**

Lesson

Update the **FireRay** script with these two lines.

```
public class FireRay : MonoBehaviour {

    public GameObject raycastMarker = null;

    // Update is called once per frame
    void Update()
    {
        //A Physics Hit object to store info we are going to get about where the ray
        RaycastHit hit;

        //The distance of the ray that we are using
        float distanceOfRay = 100;

        //Cast the ray and check if it hits anything
        if (Physics.Raycast(Camera.main.transform.position, Camera.main.transform.fo
        {
            //Debug.Log(hit.transform.name);

            //if the left mouse button is clicked
            if (Input.GetMouseButtonDown(0))
            {

                raycastMarker.transform.position = hit.point;

                Debug.Log(hit.point);

                //Move the object
                //hit.transform.position = hit.transform.position + Vector3.up;

                //Scale the object
```

Make sure you update the inspector with a reference to the Hitmarker

Results

When the game is run and you click the left mouse button the hitmarker will appear where you are aiming

Being able to destroy things

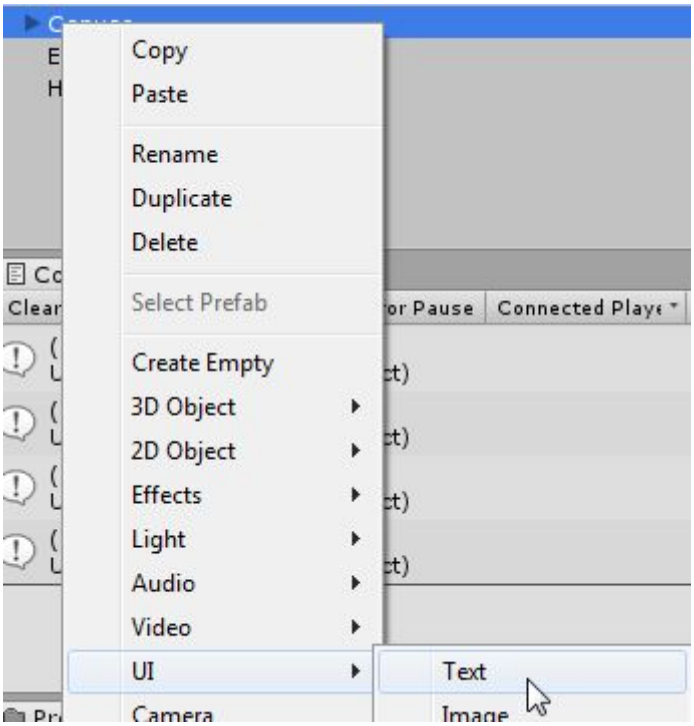
Now let's make it so we can kill things

Setup

- Same as previous lesson

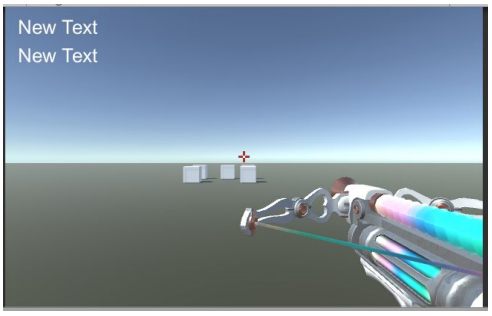
Lesson

First we need some ammo
Add 2 Text objects to the canvas to show the ammo count and the clip count.



Move the text object to the top left hand corner of the screen, Change the font/colour/size to be more readable and rename it to **AmmoCounter**

Do this again and this time call the text object **ClipCounter**

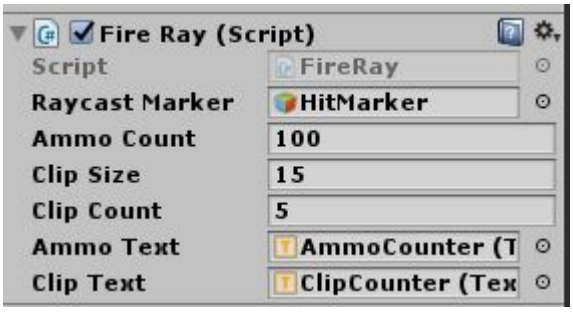


Now update our **FireRay** Script

Add a reference to the UI library at the top and references to the two new text objects

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

public class FireRay : MonoBehaviour {

    public GameObject raycastMarker = nu

    public int ammoCount = 100;
    public int clipSize = 15;
    public int clipCount = 5;

    public Text ammoText;
    public Text clipText;
```

Make sure you update the inspector



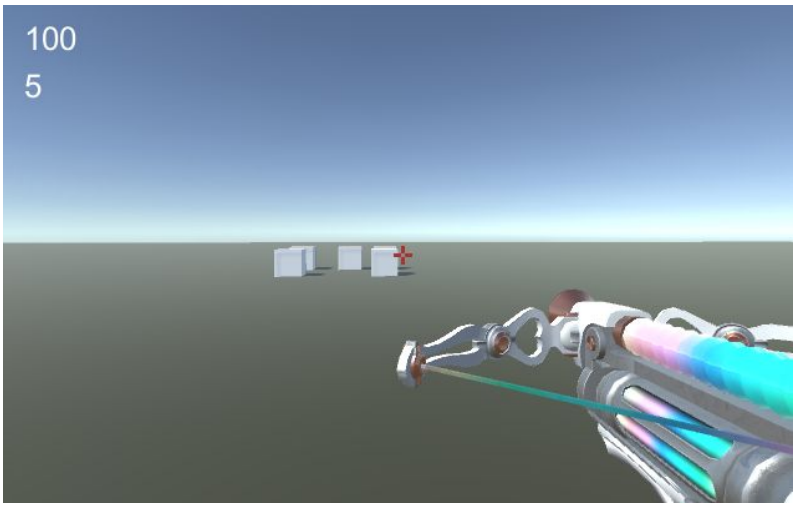
Add a new function to deal with updating the text objects

```
private void UpdateText()
{
    ammoText.text = ammoCount.ToString();
    clipText.text = clipCount.ToString();
}


// Update is called once per frame
void Update()
{
    //A Physics Hit object to store info we are going to get ab
    RaycastHit hit;

    //The distance of the ray that we are using
    float distanceOfRay = 100;

    //Cast the ray and check if it hits anything
    if (Physics.Raycast(Camera.main.transform.position, Camera.
    {
        //Debug.Log(hit.transform.name);
```

Add in the Start function to call the UpdateText Function at startup.

```
public GameObject raycastMarker = null;

public int ammoCount = 100;
public int clipSize = 15;
public int clipCount = 5;

public Text ammoText;
public Text clipText;

void Start()
{
    UpdateText();
}

private void UpdateText()
{
    ammoText.text = ammoCount.ToString();
    clipText.text = clipCount.ToString();
```

Now if you run the game you should see the text objects update with their relevant numbers



Add these few lines in to allow you to shoot and run through your clip ammo

```
void Update()
{
    //A Physics Hit object to store info we are going to ge
    RaycastHit hit;

    //The distance of the ray that we are using
    float distanceOfRay = 100;

    //Cast the ray and check if it hits anything
    if (Physics.Raycast(Camera.main.transform.position, Cam
    {
        //Debug.Log(hit.transform.name);

        //if the left mouse button is clicked
        if (Input.GetMouseButtonDown(0))
        {
            //If Clip is empty dont fire
            if (clipCount <= 0)
            {
                return;
            }

            clipCount--;
            UpdateText();

            raycastMarker.transform.position = hit.point;

            //Debug.Log(hit.point);

            //Move the object
```

Run the game and you can now shoot 5 times before running out of ammo in the clip

Finally we can add a function in to reload our gun from the ammo pool

```
private void UpdateText()
{
    ammoText.text = ammoCount.ToString();
    clipText.text = clipCount.ToString();
}

private void Reload()
{
    ammoCount += clipCount;

    if (ammoCount > clipSize)
    {
        clipCount = clipSize;
        ammoCount -= clipSize;
    }
    else
    {
        clipCount = ammoCount;
        ammoCount = 0;
    }
    UpdateText();
}


// Update is called once per frame
void Update()
{
    //A Physics Hit object to store info
    RaycastHit hit;

    //The distance of the ray that we are
    float distanceOfRay = 100;

    //Cast the ray and check if it hits a
    if (Physics.Raycast(Camera.main.trans
    {
        //Debug.Log(hit.transform.name);

        if (Input.GetKeyDown(KeyCode.R))
        {
            Reload();
        }
```

Results

Now you can reload and shoot until you run out of ammo!

Making the weapon automatic

Currently we can only fire a gun every time we click. We should make it fire at a faster rate

Setup

- Same as previous lesson

Lesson

Let's make the gun an automatic

We add a variable to hold the time between shots

```
public class FireRay : MonoBehaviour {

    public GameObject raycastMarker = null;

    public int ammoCount = 100;
    public int clipSize = 15;
    public int clipCount = 5;

    public Text ammoText;
    public Text clipText;

    public float timeBetweenBullets = 0.2f;
    private bool canShoot = true;


    void Start()
    {
        UpdateText();
    }

    private void ResetShooting()
    {
        canShoot = true;
    }
```

And here

```
//Debug.Log(hit.transform.name);

if (Input.GetKeyDown(KeyCode.R))
{
    Reload();
}

//if the left mouse button is clicked
if (Input.GetMouseButtonDown(0))
{

    //If Clip is empty dont fire
    if (clipCount <= 0)
    {
        return;
    }

    if (canShoot == false)
    {
        return;
    }

    canShoot = false;
    Invoke("ResetShooting", timeBetweenBullets);

    clipCount--;
    UpdateText();

    raycastMarker.transform.position = hit.point;

    //Debug.Log(hit.point);

    //Move the object
    //hit.transform.position = hit.transform.position + Vector3.up;

    //Scale the object
```

And finally we change GetMouseButtonDown to GetMouseButton

```
//Cast the ray and check if it hits anything
if (Physics.Raycast(Camera.main.transform.po
{
    //Debug.Log(hit.transform.name);

    if (Input.GetKeyDown(KeyCode.R))
    {
        Reload();
    }

    //if the left mouse button is clicked
    if (Input.GetMouseButton(0))
    {

        //If Clip is empty dont fire
        if (clipCount <= 0)
        {
```

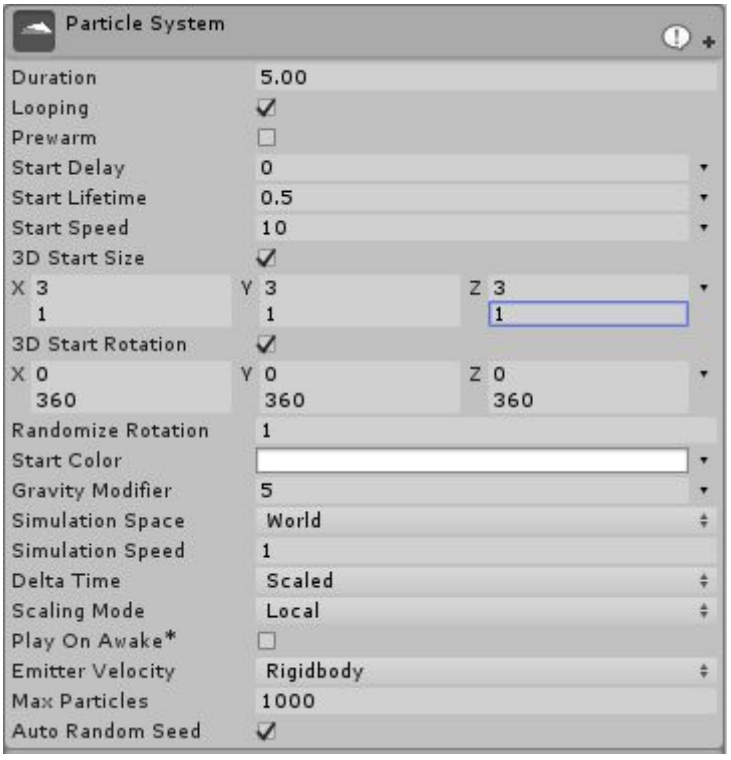
Now the gun should work just by holding down the mouse button.
But you cannot see the effect very well now. So let's add some particles.

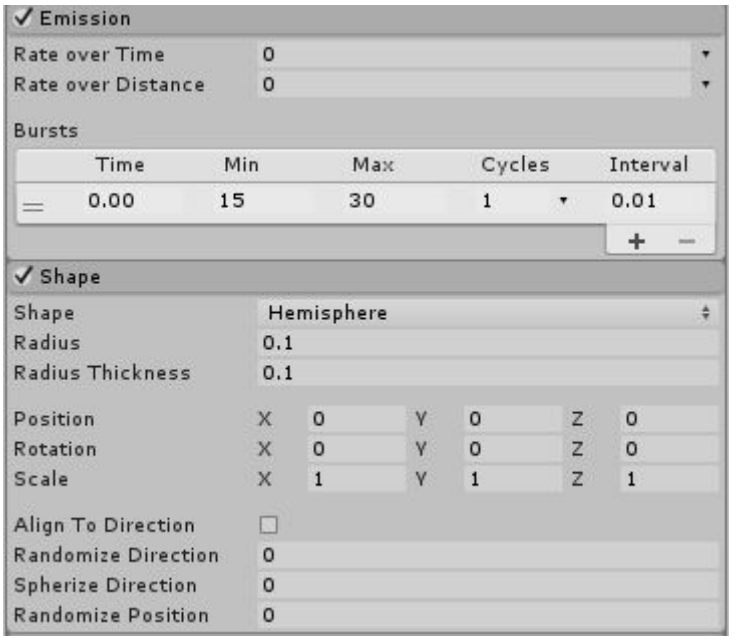
We need to make a particle that shows where the bullets are hitting (sparks or debris)

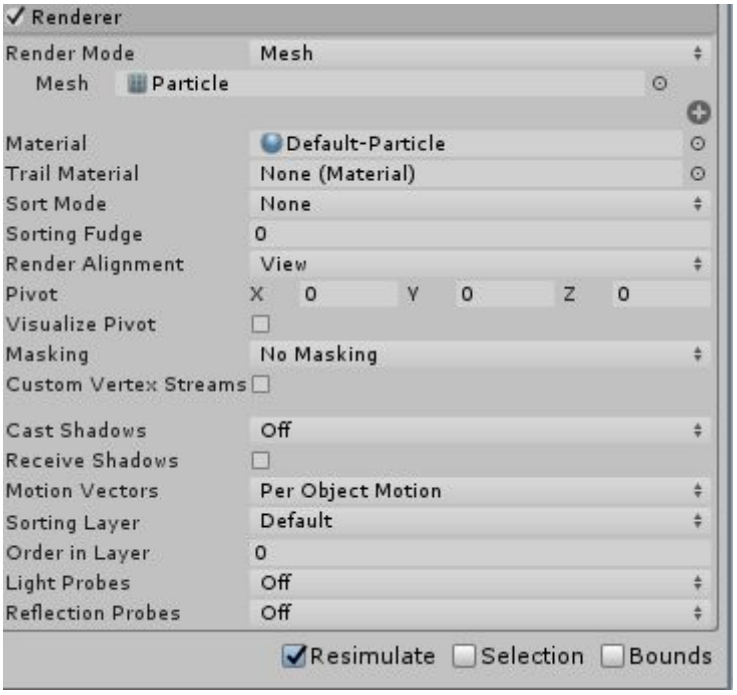
Make a particle effect and attach it to the **HitMarker**
Also delete the Sphere mesh of the **HitMarker**

Here is the settings for the particle emitter, but feel free to make your own
We have provided you an FBX piece of debris you can use

Here is an example one I set up







And update one line of code to make it work

```
//if the left mouse button is clicked
if (Input.GetMouseButton(0))
{

    //If Clip is empty dont fire
    if (clipCount <= 0)
    {
        return;
    }

    if (canShoot == false)
    {
        return;
    }

    canShoot = false;
    Invoke("ResetShooting", timeBetweenBullets);

    clipCount--;
    UpdateText();

    raycastMarker.transform.position = hit.point;
    raycastMarker.GetComponentInChildren<ParticleSystem>().Play();

    //Debug.Log(hit.point);

    //Move the object
    //hit.transform.position = hit.transform.position + Vector3.up;
```

Results

The player should now be able to shoot and reload their gun with visual feedback of shooting.

Further tasks for students to work through

- Add audio to the scene (audio clips provided)
- Make the gun kill targets
- Make the gun add force to targets (tin cans)
- Use a nav mesh to make zombies move towards you