Variables and References

Objective

To show you how Unity and C# work together to hold information.

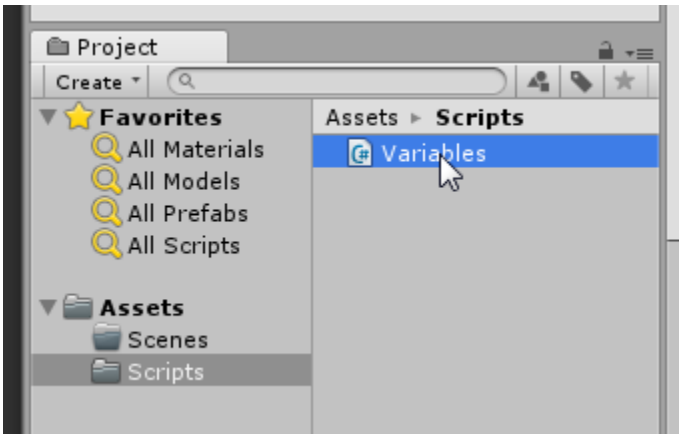
First Scene

Introducing different variables

Setup

Create a new project and scene

Create a new script called '**Variables**'



Create a cube object and drop the '**Variables**' script on it.

Lesson

Open the **Variables** script and add the following code to it.

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

public class Variables : MonoBehaviour
{

    //These numbers are floating point numbers
    public float firstFloat = 0;
    public float secondFloat = 2;
    public float thirdFloat = 5.3f;
    private float forthFloat = 364;

    //whole numbers
    public int firstInt = 1 ;
    public int secondInt = -3;
}
```

Get the class to click on the cube and see the public variables exposed.
Talk about public and private and also setting default values.
Go into the script and change one of the public variable numbers and save the script. Show the class that it does not change the number on the cube and it is already in the scene, but any new instances of the script will have the new changes applied to it.

Get them to add in these other datatypes and see what it does in the inspector.
Get them to make comments in their code with what each type does.

```
public class Variables : MonoBehaviour
{

    //These numbers are floating point numbers
    public float firstFloat = 0;
    public float secondFloat = 2;
    public float thirdFloat = 5.3f;
    private float forthFloat = 364;

    //whole numbers
    public int firstInt = 1 ;
    public int secondInt = -3;

    //Strings
    public string myString = "";
    public string mySecondString = "Hello";

    //Other Variable Types
    public Color myColour = Color.black;
    public bool myBool = true;
    public char myChar = 'd';
    public Vector2 myVector2 = new Vector2(0,3);
    public Vector3 myVecotr3 = new Vector3(0, 3, 2);
    public Vector4 myVector4 = new Vector4(0, 3,0.1f,3);
    public List<float> myFloatList = new List<float>();
    public List<int> myIntList = new List<int>();
    public List<Color> myColourList = new List<Color>();
    public List<Vector3> myVectorList = new List<Vector3>();

}
```

Talk to the class about data types and what each one stores and where it is used.
Get them to make notes on each data type.

Outcome

The students should have a basic understanding or variables/data types and how unity makes them public and private.

Using variables

Show how these variables can be used in code

Setup

Create a new script called **'CubeController'** and place it on a newly created cube in the scene.

Lesson

Get the class to make this script and put it on their cube

Add this line to the script.

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

public class CubeController : MonoBehaviour {

    // Update is called once per frame
    void Update()
    {
        //Make the cube Rotate
        transform.Rotate(Vector3.up * 2);
    }

}
```

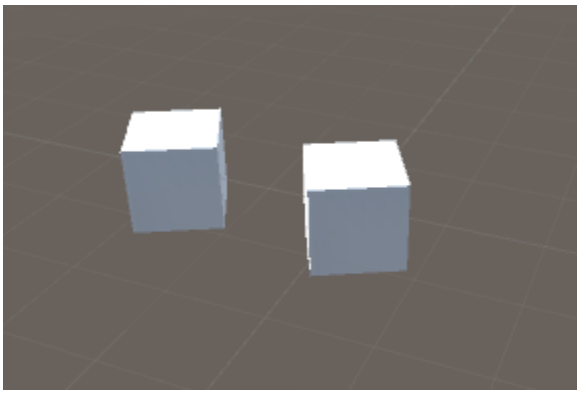
Run the script and they should see the cube rotate

Now, how do we make this cube run faster.

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

public class CubeController : MonoBehaviour {

    // Update is called once per frame
    void Update()
    {
        //Make the cube Rotate
        transform.Rotate(Vector3.up * 5);
    }

}
```

This works!..... But there is a problem.

Select the cube in the scene and press **Ctrl + D** this will duplicate the cube.
Move our new cube alongside our original one and press play now.



Now they will both spin together.
But what if we wanted them to both spin at different speeds? Well we could write a new script, but that would take a lot of work. Also what if we have 5 cubes that all had to rotate at different speeds?

There is a rule in programming called **D.R.Y.** **(**Don't Repeat Yourself)

Basically, if at any time you are repeating the same code you are doing it wrong.

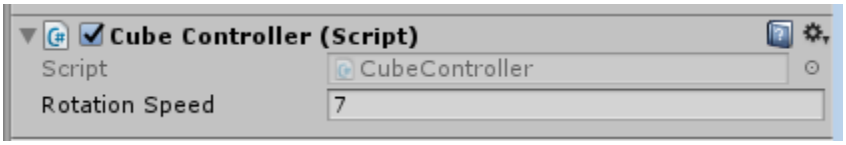
So how do we do it right then?

Well Let's update the script to this:

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

public class CubeController : MonoBehaviour {

    public float rotationSpeed = 7;

    // Update is called once per frame
    void Update()
    {
        //Make the cube Rotate
        transform.Rotate(Vector3.up * rotationSpeed);
    }

}
```

We have made a new variable called **rotationSpeed**. To this we have assigned a default value of **7.**

Now if we select one of our cubes we can see this variable



> Note that we write **'Rotation Speed**' as **rotationSpeed**. Note how the first letter of the first word is lowercase and the first letter of the second work is Uppercase. This is called **Lower Camel Case**. We will explain later why we use this standard. But for all variable names this standard must me kept.

In the Inspector change this number to any other number, and then if you select the second cube you will see that the second cube is still on the default value.

Run the game and you should see the two cubes spin at different speeds.

You can even change these values while the game is running.
But if you do, you will notice that when you stop the game the values that you changed will default back to the values they were before you ran the game. A great way of testing how fast something should move.

Outcome

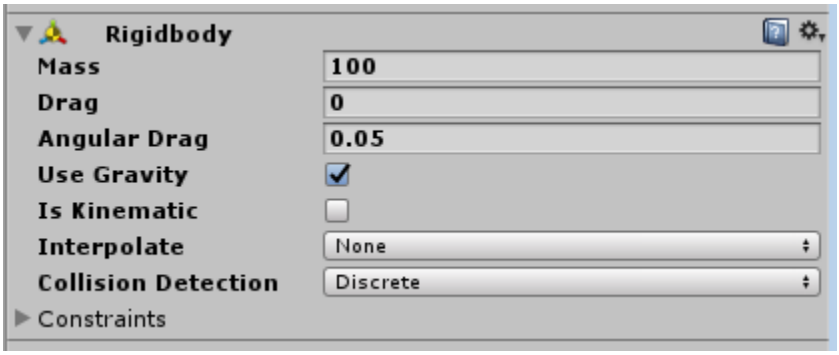
You should see the basic benefits of exposing variables in code and using one script to do different things on different objects.

Make a Tank

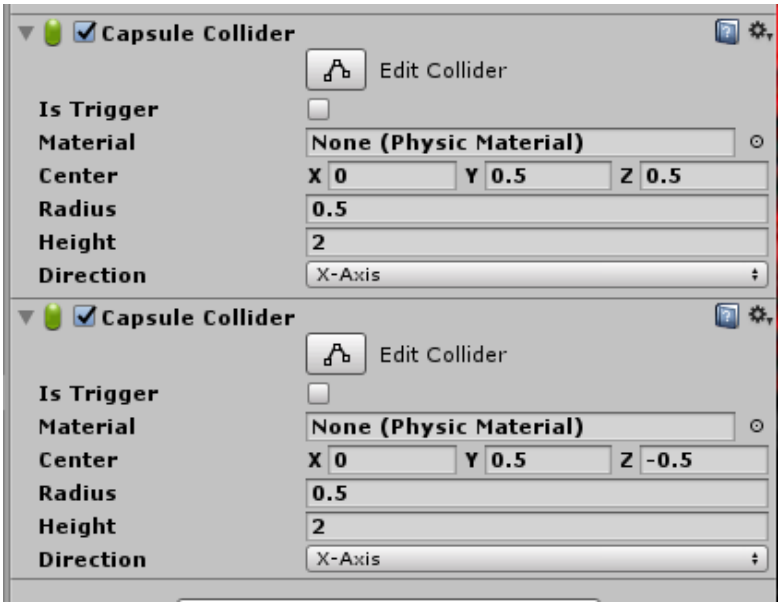
We are going to make a two player tank game to will teach about variables and references. Also split screens and basic inputs

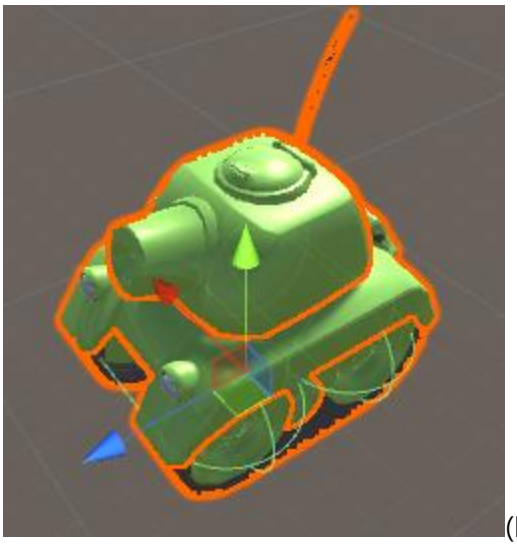
Setup

- Start a new scene
- Import the provided tank package
- Place the tank prefab into the scene (in the Prefabs folder)
- Place a Rigidbody onto the tank
    - Change the Mass to **100**

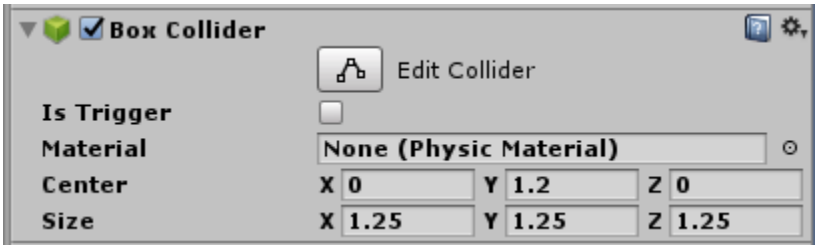


- Add two **Capsule Colliders** and set them up as follows: (note the **Direction is set to -Axis)**

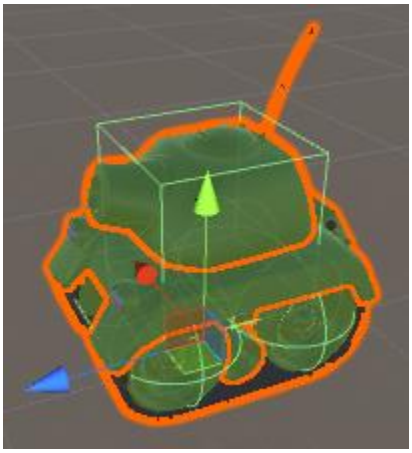


 (It should look like this)

- Attach a Box Collider to the tank and set it's values to these



It should now look like this



- Place a large plane into the scene to act as the ground
- Create a new script called 'TankController'

Lesson

Open the tank controller script and enter the following:

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

public class TankController : MonoBehaviour
{

    //Variables for tank movement
    public float forwardSpeed = 0.1f;
    public float backwardSpeed = 0.08f;
    public float rotationalSpeed = 2f;

    void Update()
    {

        //If the W key is pressed move the tank forwards
        if (Input.GetKey(KeyCode.W))
        {
            transform.position = transform.position + transform.forward * forwardSpeed;
        }

        //If the S key is pressed move the tank forwards
        if (Input.GetKey(KeyCode.S))
        {
            transform.position = transform.position + transform.forward * -backwardSpeed;
        }

        //If the A key is pressed rotate the tank to the left
        if (Input.GetKey(KeyCode.A))
        {
            transform.Rotate(transform.up * -rotationalSpeed);
        }

        //If the D key is pressed rotate the tank to the Right
        if (Input.GetKey(KeyCode.D))
        {
            transform.Rotate(transform.up * rotationalSpeed);
        }
    }
}
```

This should be similar to previous exercises, if you save the script and then run your game you should be able to move your tank around and also adjust the speed with the exposed public variables.

Next we want to control the turret moving.

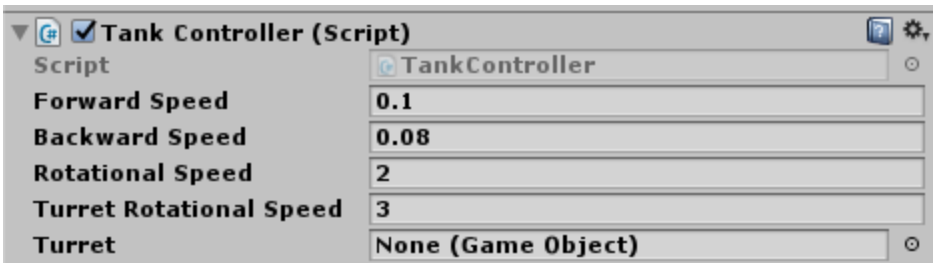
```
public class TankController : MonoBehaviour
{

    //Variables for tank movement
    public float forwardSpeed = 0.1f;
    public float backwardSpeed = 0.08f;
    public float rotationalSpeed = 2f;

    //Variable for turret rotational speed
    public float turretRotationalSpeed = 3f;

    //Reference to the turren game object
    public GameObject turret;


    void Update()
    {
        //If the Q key is pressed rotate the turret to the Left
        if (Input.GetKey(KeyCode.Q))
        {
            turret.transform.Rotate(0, -turretRotationalSpeed, 0, Space.Self);
        }

        //If the E key is pressed rotate the turret to the Right
        if (Input.GetKey(KeyCode.E))
        {
            turret.transform.Rotate(0, turretRotationalSpeed, 0, Space.Self);
        }

        //If the W key is pressed move the tank forwards
        if (Input.GetKey(KeyCode.W))
        {
            transform.position = transform.position + transform.forward * forwardSpeed;
        }
```

Now, this is the first time we are controlling an object that this script isn't on.
So what we have done is created a reference to another game object here

```
//Reference to the turrent game object
public GameObject turret;
```

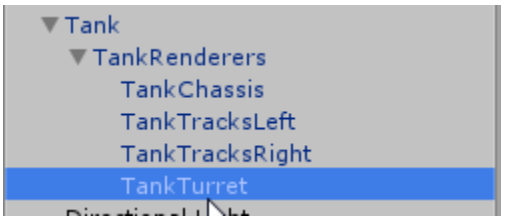
All we need to do is tell Unity where this other object is.

If you save the script you should now see this in the inspector:

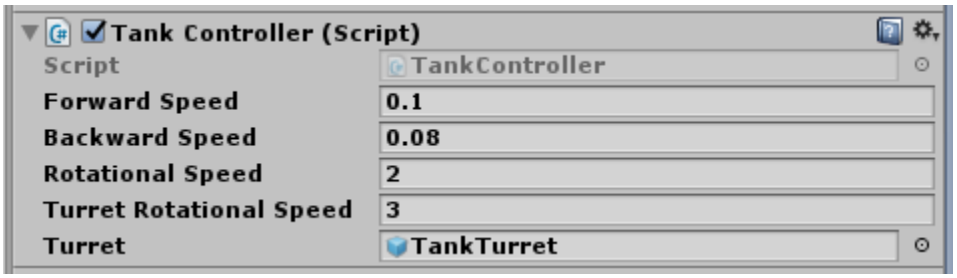


Note that **Turret** is currently set to **None (Game Object)**. To fix this we need to drag the game object we want to be controlled into this box.

The gameobject we want is this one in the prefab called **TankTurret**.



Drag this into the **Turret** box in the Inspector and you should see this



Run the game and you should have more control of the tank.

Outcome

You should have a better idea of how variables work and now understand we can control a different Game Object to the one the script is on.

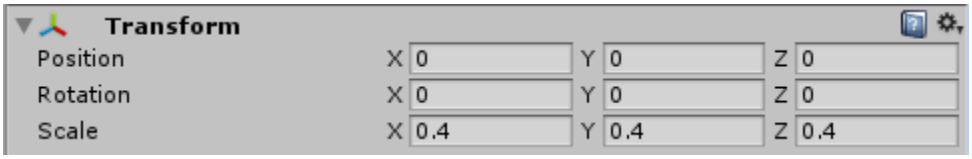
Make the tank fire

We will add more functionality and make the tank fire

Setup

Continued from last lesson

- Create a sphere
- Scale it down to 0.4f scale



- Name it '**Shell**'
- Add a rigid body to the Shell
- Make a script called '**Shell**' and attach it to the sphere
- Make a prefab of the bullet by dragging it from the **Hierarchy** view to the **Prefabs** folder
- Delete the Shell in the scene

Place a few large cube around the scene to drive the tank around

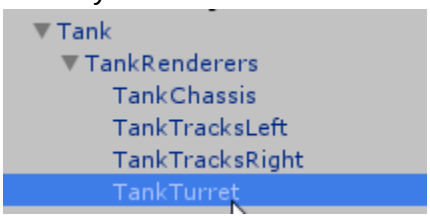
Lesson

Continue editing the **TankController** Script

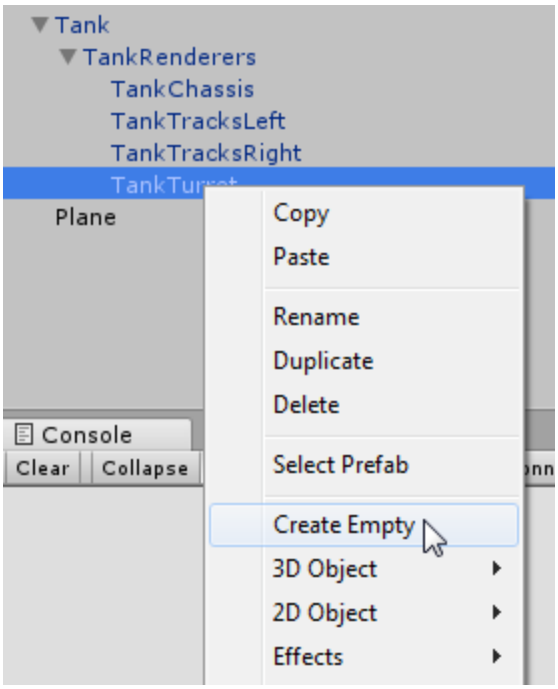
```
public class TankController : MonoBehaviour
{

    //Variables for tank movement
    public float forwardSpeed = 0.1f;
    public float backwardSpeed = 0.08f;
    public float rotationalSpeed = 2f;

    //Variable for turret rotational speed
    public float turretRotationalSpeed = 3f;

    //Reference to the turrent game object
    public GameObject turret;

    //The Speed the shell will travel
    public float shellSpeed = 20;

    //A reference to the shell prefab
    public GameObject shellPrefab;

    // A position where the shell will spawn at
    public Transform shellSpawnPoint;

    void Update()
    {

        //If the Space Key is pressed then a shell is fired
        if (Input.GetKeyDown(KeyCode.Space))
        {
            GameObject GO = Instantiate(shellPrefab, shellSpawnPoint.position, Quaternion.identity) as GameObject;
            GO.GetComponent<Rigidbody>().AddForce(turret.transform.forward * shellSpeed, ForceMode.Impulse);
        }

        //If the Q key is pressed rotate the turret to the Left
        if (Input.GetKey(KeyCode.Q))
```

We now need a point where the shell will spawn from. To do this we need to create an empty game object and place it on our tank.

Select your **TankTurret**

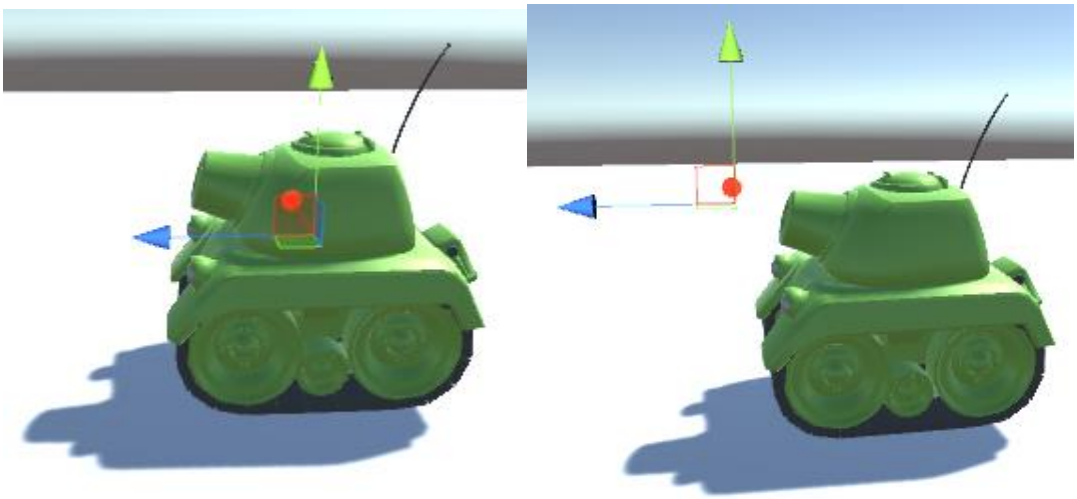


Right Click on it and select **‘Create Empty’**



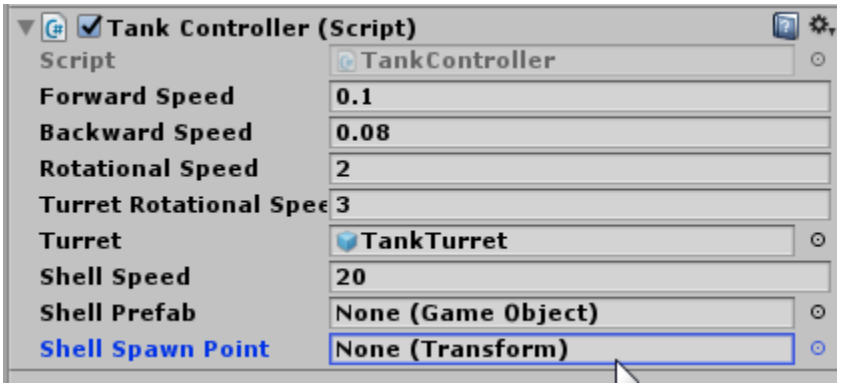
You now should have a new Game Object. Rename it to **'Shell Spawn Point'** by selecting it and pressing **F2**

Now we need to move it to the correct location, With the **Shell Spawn Point** selected, In the scene Move it to just in front of the Tank Turret.

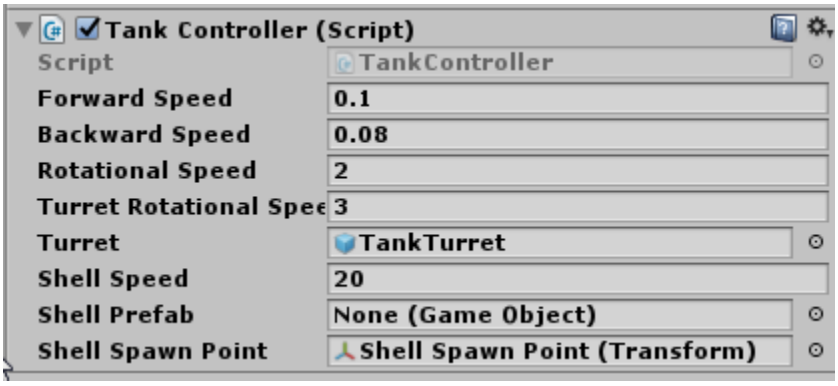


From Here To Here

Now we need to tell our **TankController** script about this new Spawnpoint
Select the tank and you should see the **Shell Spawn Point** is set to **None (Transform)**

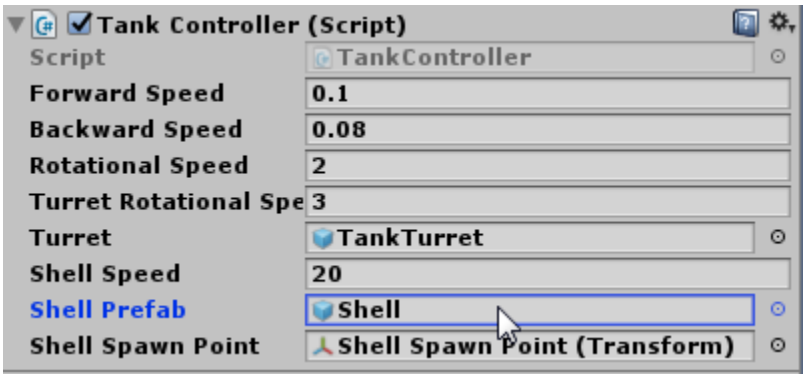


Drag the newly created Shell Spawn Point to this field.



Once these changes have been made, The bullet prefab needs to be referenced.

Currently the **Shell Prefab** says **None (Game Object)**.
Drag our created Shell Prefab into this field



Now if you run the game you should be able to drive the tank around a shoot balls everywhere.

We need to make sure the shell don't go on forever. Each shell takes up a small amount of memory (RAM) so if we play this game for a long period of time the game will start to slow down as the game needs to keep track of all the shells we have shot.

Open the **Shell** script,

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

public class Shell : MonoBehaviour {

    // Use this for initialization
    void Start () {
        Destroy(this.gameObject, 4f);
    }
}
```

This command runs once when the shell is spawned and simply tell it to destroy itself after 4 seconds.

Later we will also make the shells destroy themselves when they hit something.

Run the game and test the shooting

Outcome

You should now know how to move a gameobject around with a script that is placed on it. And be able to move another game object through a reference to that object.

Be able to create (instantiate) a gameobject at will, and also destroy it after a set amount of time.

Make the game two player

We will make this a two player tank game.

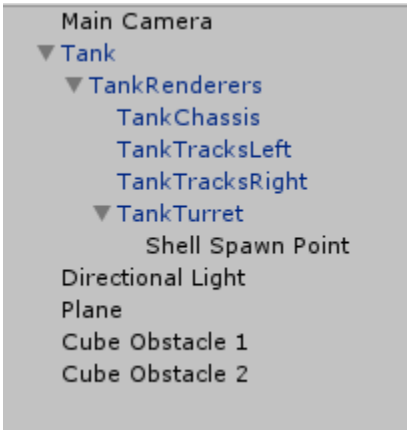
Setup

Same as previous lesson

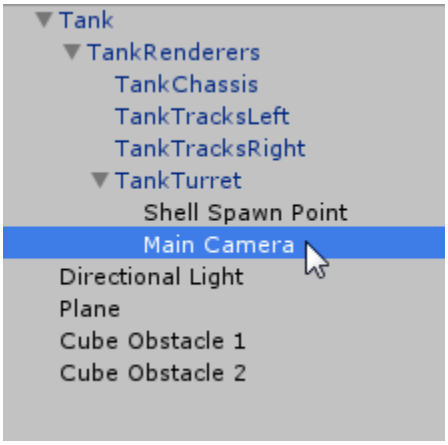
Lesson

First we need to attach the camera to the tank.

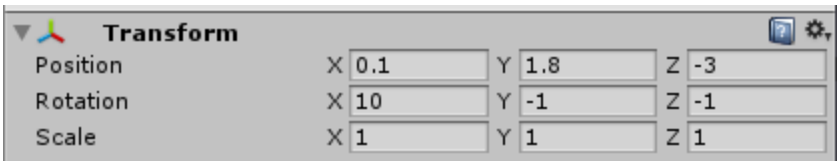
Currently your scene Hierarchy should look something like this



Drag the **Main Camera** to the **TankTurret**



This will now allow the camera to move with the tank.
We now need to alight it to look at the tank.
Select the **Main Camera** and change the **Transform** settings to these



Now if you play the game you should have a 3rd person view.

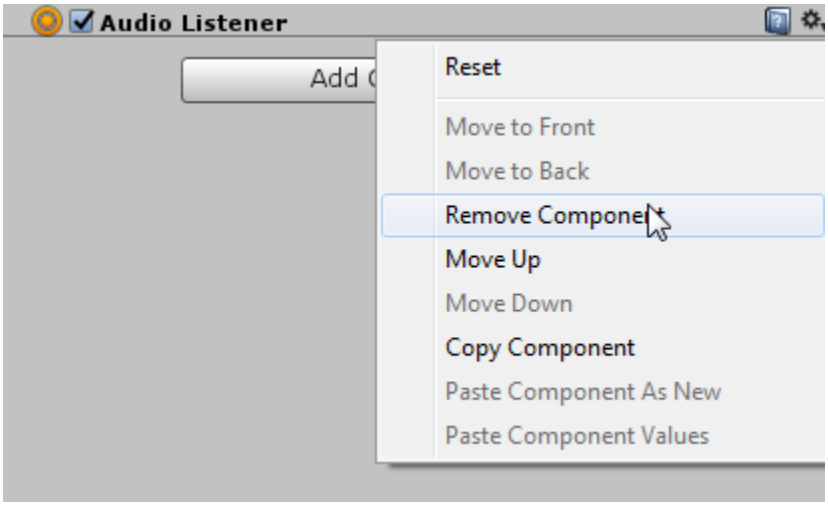
Now if we want to make this a two tank game all we need to do is duplicate the tank.
Highlight the tank and press **Ctrl + D**

Move the newly created tank out from inside of the other.

Now play the game.

You should notice a few things

- You get an error message explaining there is two audio listeners in the scene.
    - This is a component that listens for audio in a scene. Unity only allows one and it lives on the Main Camera by default and we just duplicated it.
    - Select one of the cameras and delete one of these components



- When you move one tank, both tanks will move (pretty pointless two player game) we will fix this later

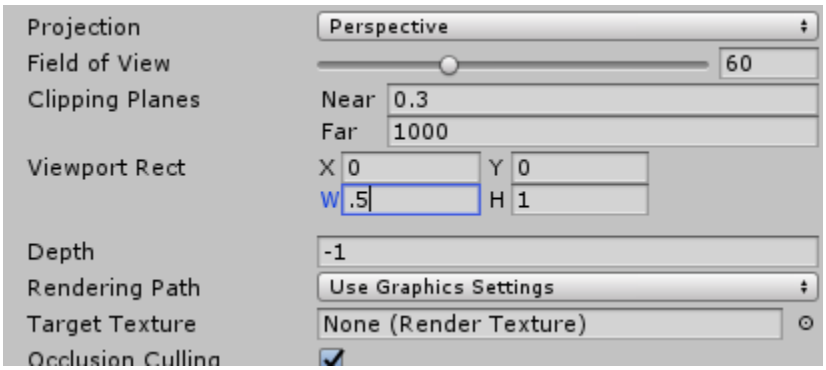
- And that you can only see out of one of the camera views (we will fix this next)

Make the cameras Split screen

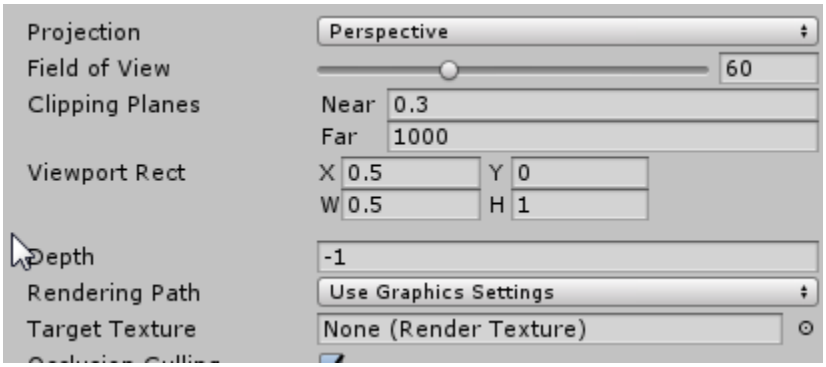
Currently there are two cameras in the scene and both are drawing their view to the screen. But one is being drawn over the top of the other.

To fix this we need to select each camera and change its **Viewport Rect**

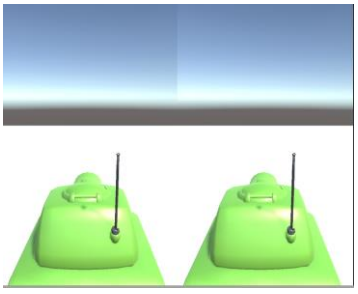
Camera 1



Camera 2



You should have have a split screen view



Now to fix the controls controlling both tanks

In the **TankController** few lines that allow us to change the controls.

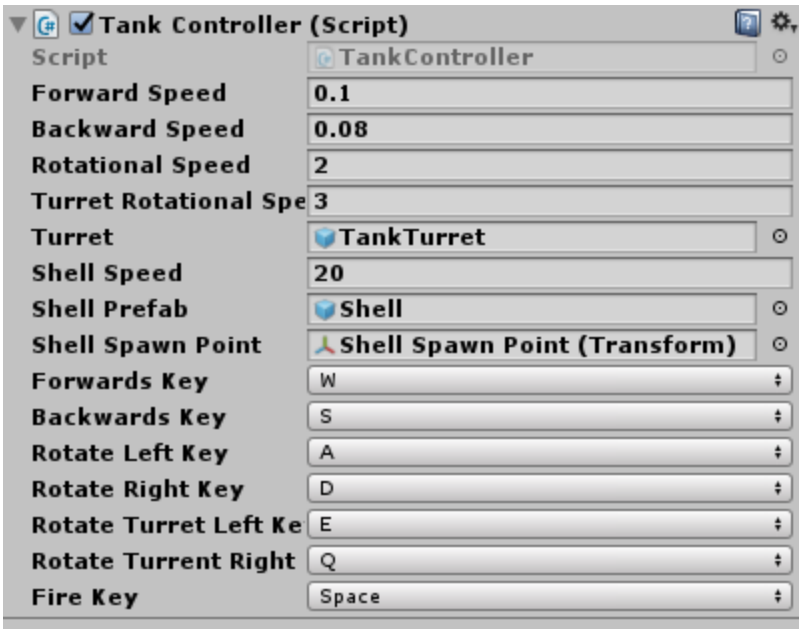
Add these references to the top of the script.

```
//The Speed the shell will travel
public float shellSpeed = 20;

//A reference to the shell prefab
public GameObject shellPrefab;

// A position where the shell will spawn at
public Transform shellSpawnPoint;

//References to the keys required to control the tank
public KeyCode forwardsKey = KeyCode.W;
public KeyCode backwardsKey = KeyCode.S;
public KeyCode rotateLeftKey = KeyCode.A;
public KeyCode rotateRightKey = KeyCode.D;
public KeyCode rotateTurretLeftKey = KeyCode.Q;
public KeyCode rotateTurrentRightKey = KeyCode.E;
public KeyCode fireKey = KeyCode.Space;


void Update()
{

    //If the Space Key is pressed then a shell is fired
```

Now we need to replace some of the code to use these new references

```
void Update()
{

    //If the Space Key is pressed then a shell is fired
    if (Input.GetKeyDown(fireKey))
    {
        GameObject GO = Instantiate(shellPrefab, shellSpawnPoint.position, Quaternion.identity) as GameObject;
        GO.GetComponent<Rigidbody>().AddForce(turret.transform.forward * shellSpeed, ForceMode.Impulse);
    }

    //If the Q key is pressed rotate the turret to the Left
    if (Input.GetKey(rotateTurretLeftKey))
    {
        turret.transform.Rotate(0, -turretRotationalSpeed, 0, Space.Self);
    }

    //If the E key is pressed rotate the turret to the Right
    if (Input.GetKey(rotateTurrentRightKey))
    {
        turret.transform.Rotate(0, turretRotationalSpeed, 0, Space.Self);
    }

    //If the W key is pressed move the tank forwards
    if (Input.GetKey(forwardsKey))
    {
        transform.position = transform.position + transform.forward * forwardSpeed;
    }

    //If the S key is pressed move the tank forwards
    if (Input.GetKey(backwardsKey))
    {
        transform.position = transform.position + transform.forward * -backwardSpeed;
    }

    //If the A key is pressed rotate the tank to the left
    if (Input.GetKey(rotateLeftKey))
    {
        transform.Rotate(transform.up * -rotationalSpeed);
    }

    //If the D key is pressed rotate the tank to the Right
    if (Input.GetKey(rotateRightKey))
    {
        transform.Rotate(transform.up * rotationalSpeed);
    }
}
```

With these changes the tanks will still work the same but now if you look at the inspector you should see this



Now you can change the controls on one of the tanks to whatever you want.

Outcome

You should have two tanks controlled separately that can shoot at each other.
You also know how to use references to setup custom keyboard inputs.

Make the tanks shoot each other.

Lets allow the tanks to kill each other.

Setup

Same as previous

- Tag both the tanks as ‘Player’

Lesson

Open the Tank controller script and add a few lines

```
public KeyCode rotateTurretLeftKey = KeyCode.Q;
public KeyCode rotateTurrentRightKey = KeyCode.E;
public KeyCode fireKey = KeyCode.Space;

//The health of the tank


public int health = 100;

//Custom function that gets called when a shell hits the tank
public void TakeDamage(int damageToTake)
{
    health = health - damageToTake;
}


void Update()
{

    if (health <= 0)
    {
        //this exits the function and does not run anything after it.
        return;
    }

    //If the Space Key is pressed then a shell is fired
    if (Input.GetKeyDown(fireKey))
    {
        GameObject GO = Instantiate(shellPrefab, shellSpawnPoint.posit
        GO.GetComponent<Rigidbody>().AddForce(turret.transform.forward
    }
```

We have now made a new function (explained more in a future exercise) and that we have made it public (so it can be accessed from the shell script).

Now open up the **Shell** script

We will add a new variable to control how much damage the shell will give when it hits a tank
And a special function that is called when the rigidbody of the shell hits something.

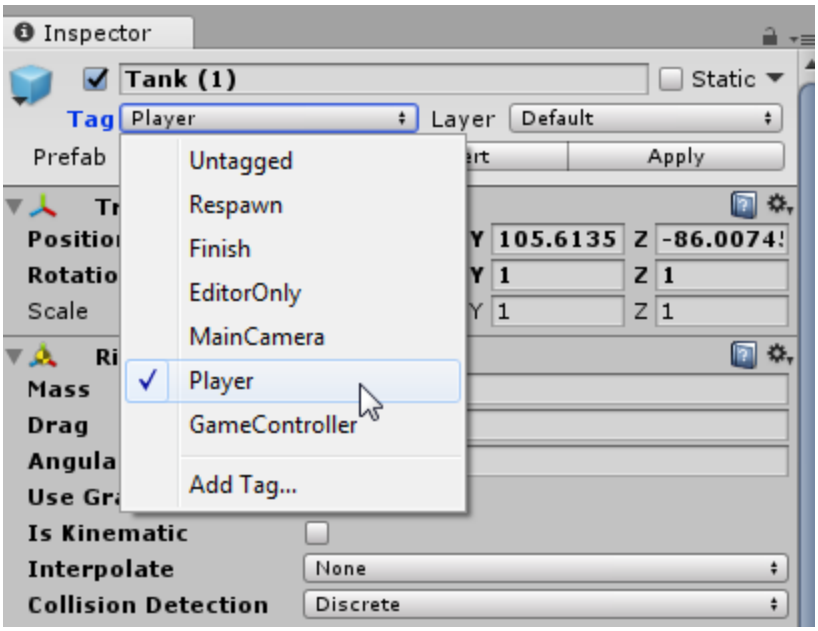
```
public class Shell : MonoBehaviour {

    // The amount of damage the shell will do when it hits a tank
    public int damage = 20;

    // Use this for initialization
    void Start () {
        Destroy(this.gameObject, 4f);
    }

    //Called when the RigidBody hits something
    void OnCollisionEnter (Collision other)
    {
        if (other.gameObject.tag == "Player")
        {
            other.gameObject.GetComponent<TankController>().TakeDamage(damage);
        }
        Destroy(this.gameObject);
    }

}
```

If you read this code, what is happening is:

OnCollisionEnter Gets Called when the shell hits something.

- The code will then check to see the tag of the game object of the object it hits
- If the object is tagged as “Player” then it finds the **TankController** script on it and runs the function **TakeDamage** and passes it the value **Damage**
- It then destroys the shell

Note that we have not tagged the tanks yet so this script sill not work
To do this select each tank and click on **Tag** and select **Player**



Outcome

Now when you hit a tank you should see that other tanks life go down, and when it hits 0 or less it stops working.

Further tasks

- Make a full arena for the tanks to drive around
- Change the game to a One camera looking down at the two tanks shooting each other. Which way is more fun?
- Look up Explosive force and make the bullets throw rigid bodies around
    - https://docs.unity3d.com/ScriptReference/Rigidbody.AddExplosionForce.html
- Make particles that activate at correct times
    - Dust when they drive
    - Explosions when the bullets hit
    - Large Explosion when the tank dies.