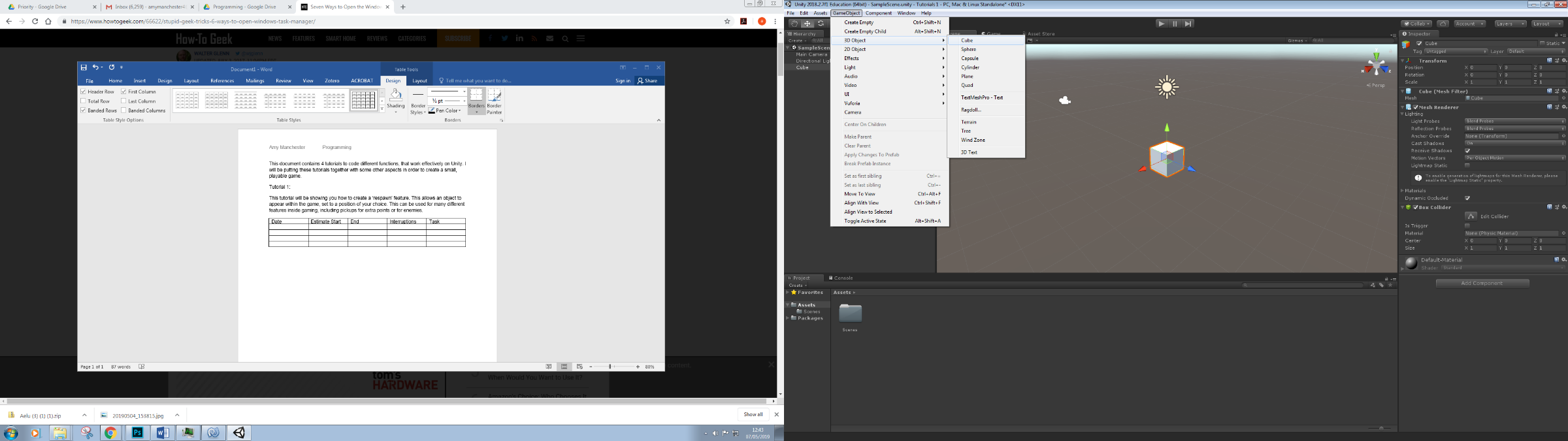
This document contains 4 tutorials to code different functions, that work effectively on Unity. I will be putting these tutorials together with some other aspects in order to create a small, playable game.

Tutorial 1:

This tutorial will be showing you how to create a ‘respawn’ feature once the original object has been destroyed by colliding with the enemy. This allows an object to appear within the game, set to a position of your choice. This can be used for many different features inside gaming, including pickups for extra points or for enemies.

Step 1:

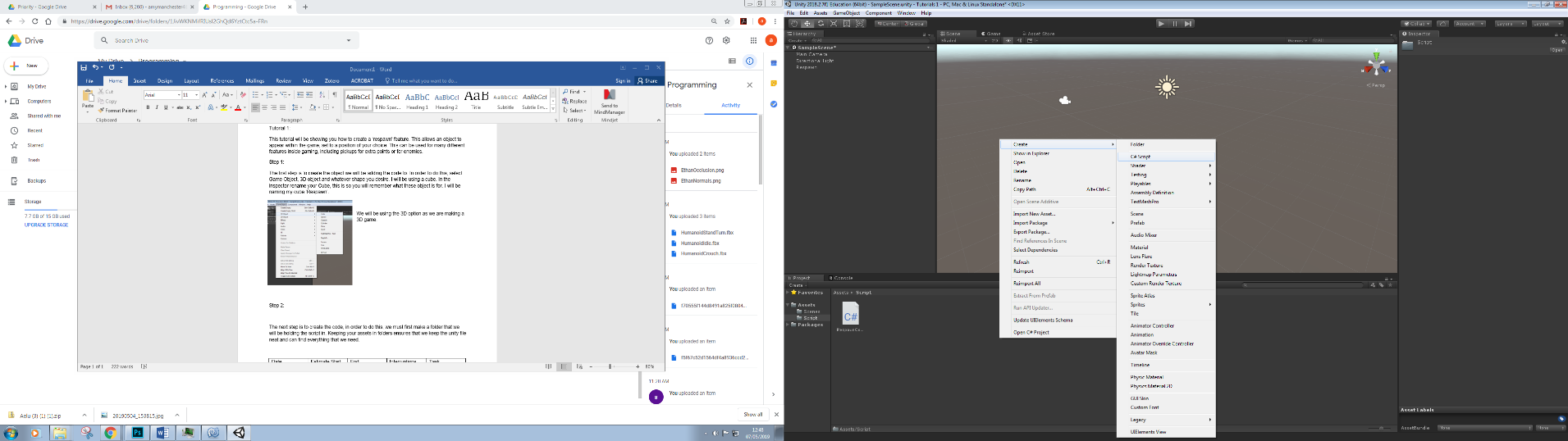
The first step is to create the object we will be adding the code to. In order to do this, select Game Object, 3D object and whatever shape you desire. I will be using a cube. In the inspector rename your Cube, this is so you will remember what these object is for. I will be naming my cube ‘Respawn’.



We will be using the 3D option as we are making a 3D game.

Step 2:

The next step is to create the code, in order to do this, we must first make a folder that we will be holding the script in. Keeping your assets in folders ensures that we keep the unity file neat and can find everything that we need.

Name the code something clear, I will be naming mine RespawnCode. Do not use any spaces in this name.

Code:

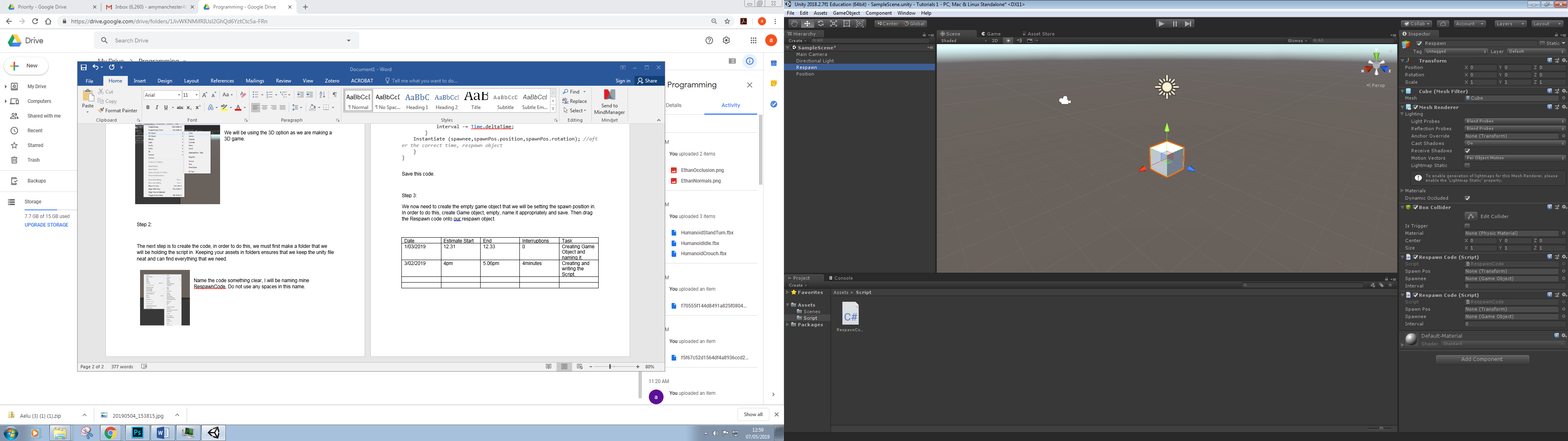
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class LaserSpawner : MonoBehaviour {  
  
    public Transform spawnPos; *//where the object will respawn- set via an empty game object in the inspector*  
    public GameObject spawnee; *// what object we will be respawning*  
    public float interval; *//how often to respawn*  
  
    void OnTriggerEnter(Collider other) {   
        if (other.gameObject.tag== "Enemy") { Destroy (gameObject); }} *//if the enemys collider collides with this gameobjects collider, destroy our object.*  
  
    void Update (){  
        if (interval > 0){ *//if interval is set higher then 0, this part of code will run.*  
            interval -= Time.deltaTime;  
        }  
    Instantiate (spawnee,spawnPos.position,spawnPos.rotation); *//after the correct time, respawn object*  
    }  
}

Save this code.

Step 3:

We now need to create the empty game object that we will be setting the spawn position in. In order to do this, create Game object, empty, name it appropriately and save. Then drag the Respawn code onto our respawn object.

We need to fill out the inspector for our code to work:



The spawn pos, is where we drag our empty game object.

The spawnee object is what will be respawning and the interval can be set to how you like.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Estimate Start | End | Interruptions | Task |
| 1/03/2019 | 12.31 | 12.33 | 0 | Step 1- Creating Game Object and naming it. |
| 3/02/2019 | 4pm | 5.06pm | 4minutes | Step 2- Creating and writing the Script. |
| 3/02/2019 | 5.15pm | 5.20pm | 0 | Step 3- creating empty and setting inspector. |

You can now press play and test this code.

Tutorial 2:

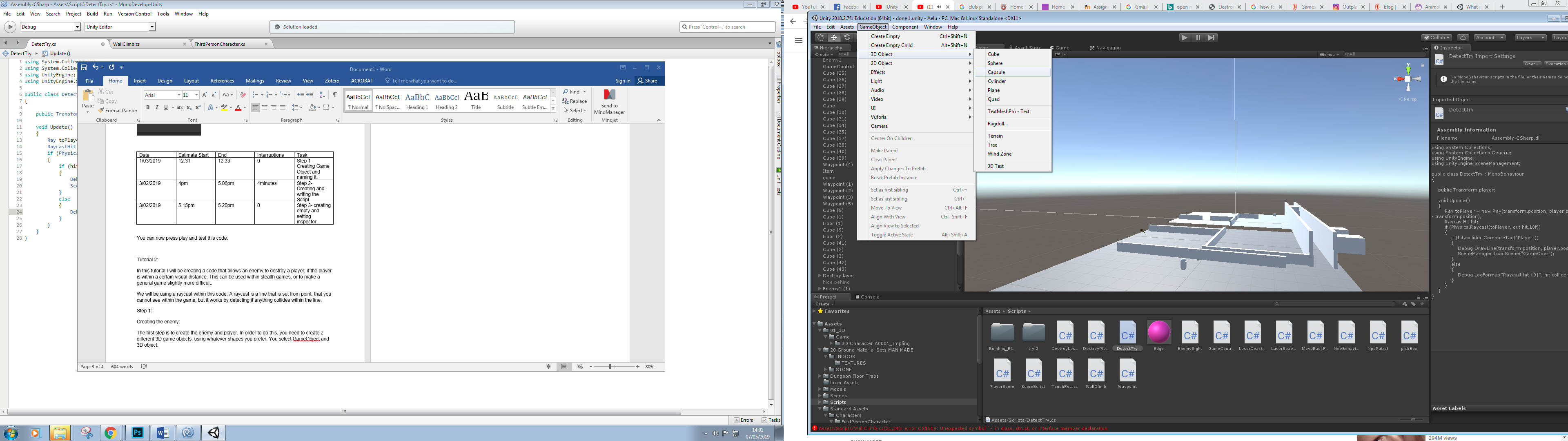
In this tutorial I will be creating a code that allows an enemy to destroy a player, if the player is within a certain visual distance. This can be used within stealth games, or to make a general game slightly more difficult.

We will be using a raycast within this code. A raycast is a line that is set from point, that you cannot see within the game, but it works by detecting if anything collides within the line.

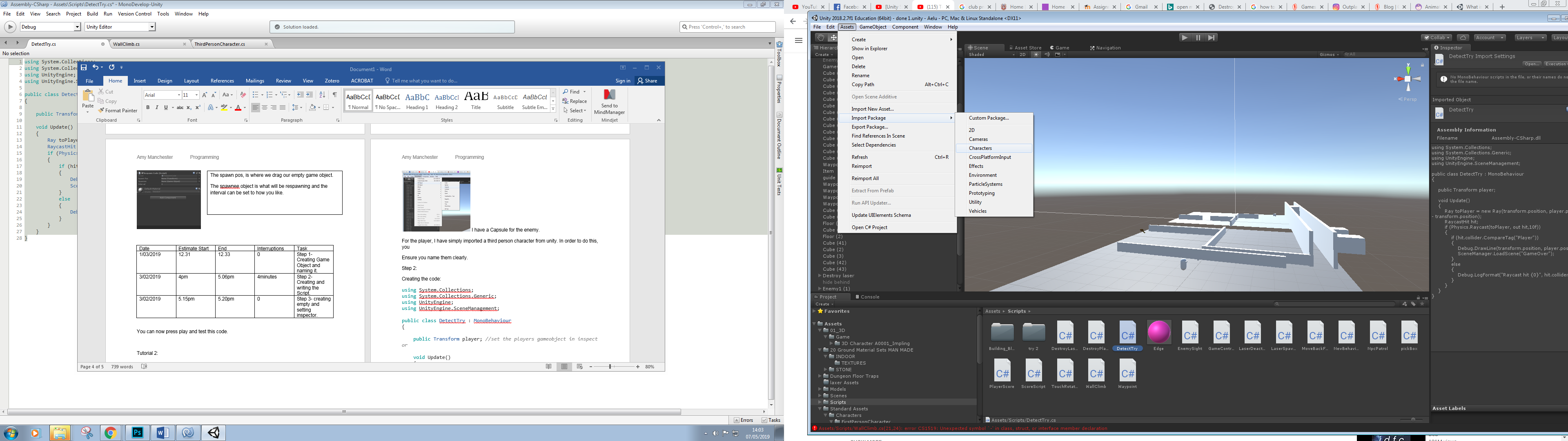
Step 1:

Creating the enemy:

The first step is to create the enemy and player. In order to do this, you need to create an 3D game objects, using whatever shapes you prefer. You select GameObject and 3D object:

 I have a Capsule for the enemy.

For the player, I have simply imported a third person character from unity. In order to do this, you select assets, import package and select characters.

 You can use whatever person you prefer, I selected third person as I find this the easiest to use.

Step 2:

Creating the code:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.SceneManagement; //this function allows us to load or use different scenes in our code.  
  
public class DetectTry : MonoBehaviour  
{  
  
    public Transform player; *//set the players gameobject in inspector*  
  
    void Update()  
    {  
        Ray toPlayer = new Ray(transform.position, player.position - transform.position);  
        RaycastHit hit;  
        if (Physics.Raycast(toPlayer, out hit,10f)) *//if the player is within 10f of the enemies set raycast line,run the code below.*  
        {  
            if (hit.collider.CompareTag("Player")) *//if the player is hit by the raycast,and the objects tag is Player, load the scene GameOver*  
            {  
                Debug.DrawLine(transform.position, player.position);  
                SceneManager.LoadScene("GameOver");  
            }  
            else  
            {  
                Debug.LogFormat("Raycast hit {0}", hit.collider); *//show a message in the console to show code functions correctly*  
            }  
        }  
    }  
}

Step 3:

Setting up the scene manager:

This is the scene that will load when our enemy’s raycast collides with our player.

We do this by saving our scene we are now on, by selecting file, save as and naming our current scene as level 1.

We then select new scene, saving this as GameOver. This scene’s name must match to the one we placed in our code or it will not load. If you want, you can create UI assets to give the scene more life, but as we are testing our code, we do not need to do this.

You then need to add this scene into the build, you do this by selecting File, Build and dragging the scene into the build.

Step 4:

Testing the code.

We will now have to test this code. In order to do this, drag the code onto the enemy capsule we created in step 1. As you can see, when our player comes within a certain distance, the GameOver scene now loads.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Estimate Start | End | Interruptions | Task |
| 3/3/2019 | 12.31pm | 1.oopm | 10 minutes | Step 1: creating enemy and importing character |
| 3/3/2019 | 1.05pm | 3.30pm | 8miuntes | Step 2: creating code |
| 3/3/2019 | 3.35pm | 4.00pm | 0 | Setting Up scene manager |
| 3/3/2019 | 4.15pm | 4.20pm | 0 | Step 4: Testing code |

Tutorial 3:

In this tutorial we will be making a score system. We will be using UI aspects to create score test and will be collecting a certain amount of coins, then making a YouWin text aspect within our unity project.

Step 1:

Creating the player and floor:

Firstly, we need to import a character asset, like as shown in the previous Tutorial. We also need to create an area for our player to move around to collect the coins. We will do this by creating a ‘plane’ as the floor.

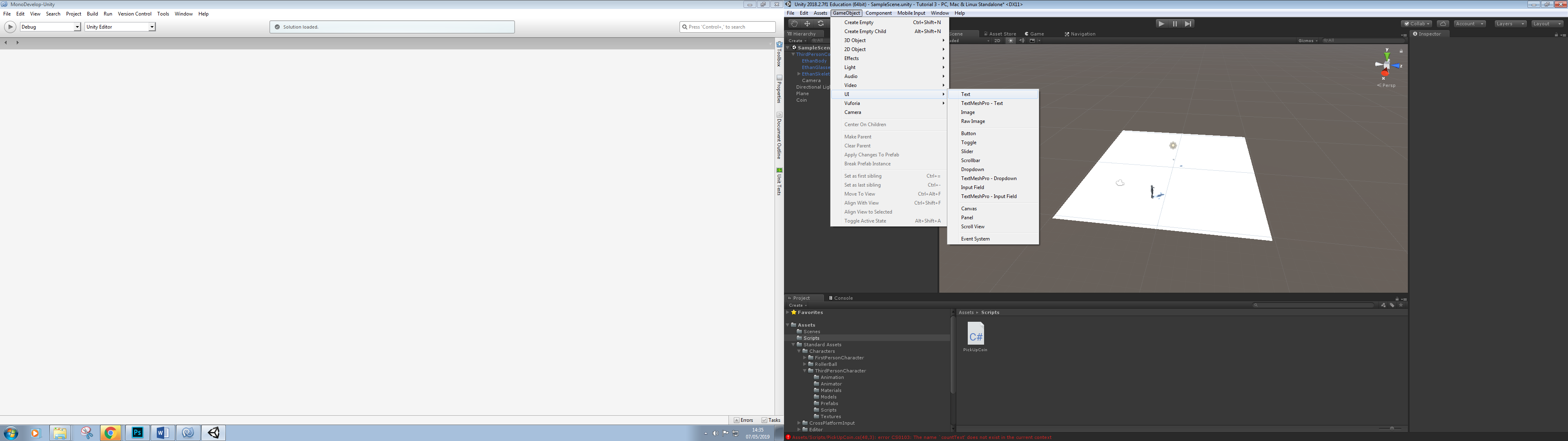
We now need to create a 3D GameObject to be our pickups. I will be using a cube.

Now we are ready to create the UI aspects.

Step 2:

Creating the Score and You Win text:

To view the text, you have created, select the Game view option, so you can place them where you feel suitable. You can also change the colour or the size of the text as you want.

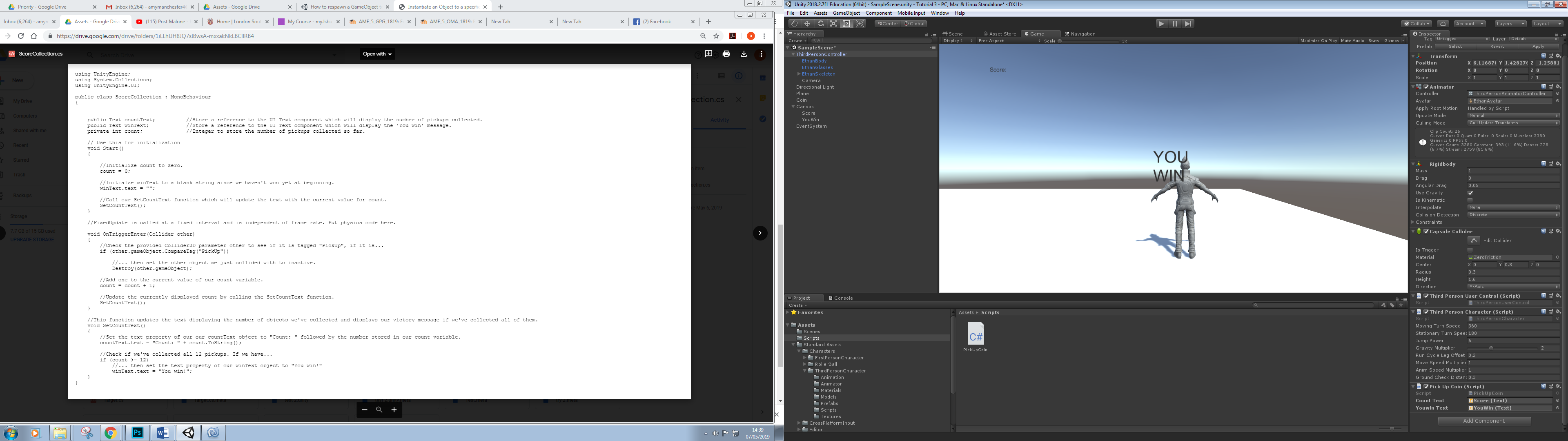


Step 3:

Creating the code:

﻿   
using UnityEngine;  
using System.Collections;  
using UnityEngine.UI; *//as you are using the unity UI assets, you need to include this*  
  
public class PickUpCoin : MonoBehaviour  
{  
  
  
    public Text countText;      *//how you will show the text*  
    public Text YouwinText;        *//set the text you want to appear when you win*  
    private int count;        *//integer to keep count of how many coins collected*  
  
    *// Use this for initialization*  
    void Start()  
    {  
  
        *//start with 0 coins collected*  
        count = 0;  
  
        *//no win text to show when the game starts as the player has yet to win*  
        YouwinText.text = "";  
  
  
        SetCountText(); *//set the count amount*  
    }  
  
  
  
    void OnTriggerEnter(Collider other)  
    {  
        *//check we have collided with the coin*  
        if (other.gameObject.CompareTag("PickUpCoin"))  
  
        *//if we have then destroy it*  
            Destroy(other.gameObject);  
  
        *// add one to the count of coins we have*  
        count = count + 1;  
  
        *//Show that we have collected this coin*  
        SetCountText();  
    }  
  
  
    void SetCountText()  
    {  
          
        countText.text = "Count: " + count.ToString();  
  
        *//if we collect 5 coins then our player wins*  
        if (count >= 5)  
            *//show the you win text*  
        YouwinText.text = "You win!";  
    }  
}

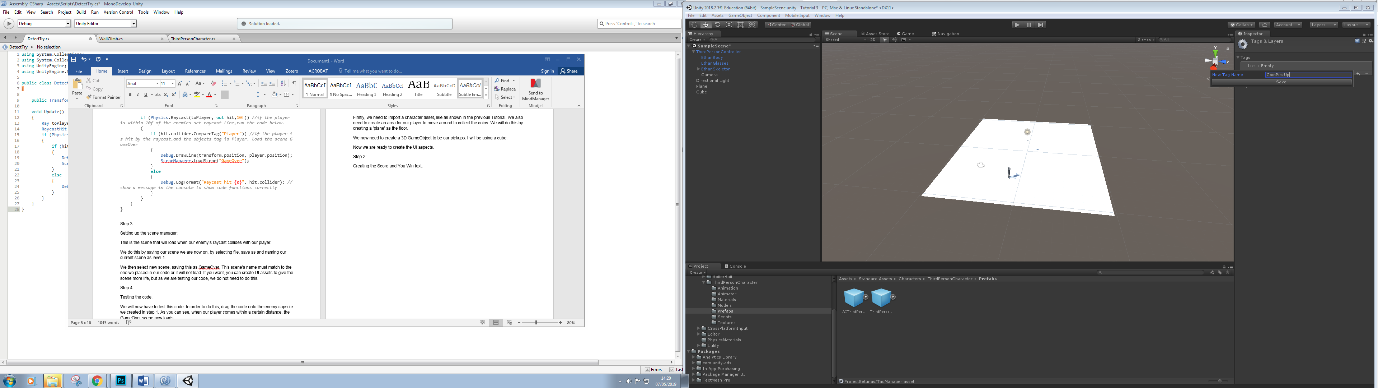
You now need to drag the text UI elements into the inspector below once you have dragged the script onto the player:



Step 4:

Now the only thing we need to do is to change the coins tag to suit the one we wrote in our code. As we have named the tag “CoinPickUp” we will rename our tag as this. To do this select the tag option in the inspector, select name, rename it and add it to our coin.

As we have a maximum of 5 coins before our YouWin text appears, duplicate this coin 5 times and move them around the scene.



Step 5:

You can now test this code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Estimate Start | End | Interruptions | Task |
| 8/3/2019 | 2.01pm | 2.16pm | 0 | Creating testing area |
| 8/3/2019 | 2.20pm | 2.34pm | 5 minutes | Creating UI assets |
| 8/03/2019 | 2.42pm | 3.58 | 0 minutes | Creating the Code |
| 8/03/2019 | 4.2pm | 4.12pm | 0 Minutes | Tag Chances |

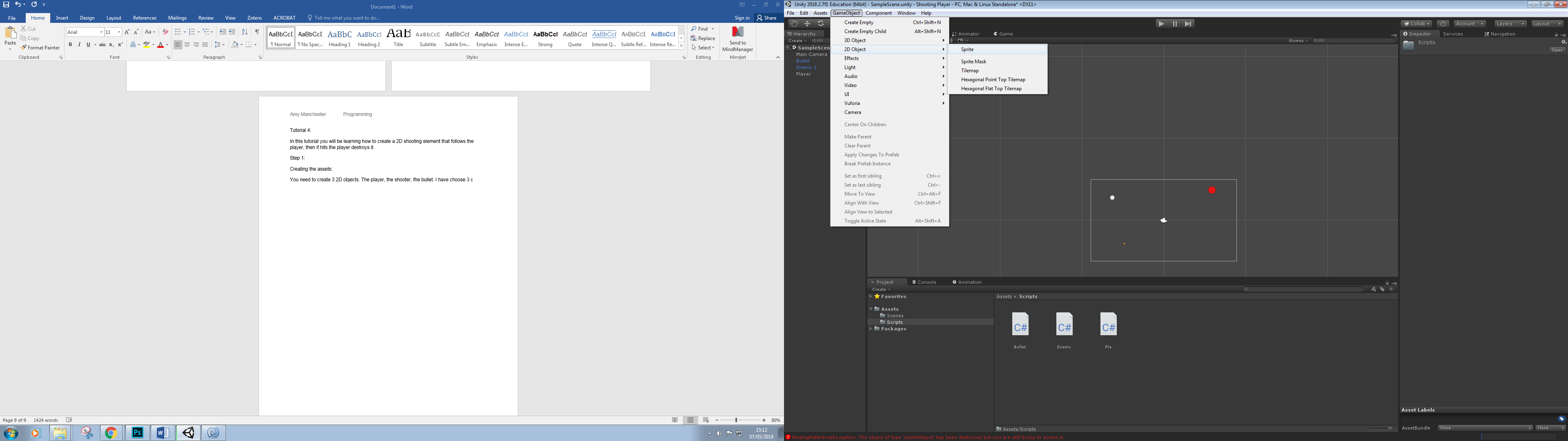
Tutorial 4:

In this tutorial you will be learning how to create a 2D shooting element that follows the player, then if hits the player destroys it.

Step 1:

Creating the assets:

You need to create 3 2D objects. The player, the shooter, the bullet.



As you can see, they do not appear with the shapes as the option of 3D gameObjects do. In the inspector, select which shape you’d like to have. Ensure you name them correctly.

Step 2:

Adding materials:

In order to have a clear view of what object is what, add 3 different colours to them. To do this, select Assets, Material, select which colour you prefer and drag this onto the object.

Step 3:

Creating the scripts. We have to create 3 scripts, so let’s go through them one at a time, starting with the player:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class Pla : MonoBehaviour {  
      
        [SerializeField]  
        float moveSpeed = 5f; *//what speed to move at*  
  
        float dirX , dirY; *//which direction*  
        Rigidbody2D rb; *//refering to the rigidbody attached to the object*  
  
        *// Use this for initialization*  
        void Start () {  
            rb = GetComponent<Rigidbody2D> ();   
        }  
  
        *// Update is called once per frame*  
        void Update () {  
            dirX = Input.GetAxis ("Horizontal") \* moveSpeed; *//as unitys go to controls are the key arrows, we do not have to set these, but this part of code shows which direction to travel in and when.*  
            dirY = Input.GetAxis ("Vertical") \* moveSpeed;  
        }  
  
        void FixedUpdate()  
        {  
            rb.velocity = new Vector2 (dirX, dirY);  
        }  
}

We also need to add a tag to the player, add a tag named “Pla”, you can name the tag whatever you prefer but ensure you change this in the next script.

Bullet:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class Bullet : MonoBehaviour {  
  
    float moveSpeed = 7f; *//what speed the bullet moves at*  
  
    Rigidbody2D rb; *//refering to the rigidbody attatched to the player*  
  
    Pla target;  
    Vector2 moveDirection; *//what direction the bullet will travel in*  
  
    *// Use this for initialization*  
    void Start () {  
        rb = GetComponent<Rigidbody2D> ();  
        target = GameObject.FindObjectOfType<Pla>(); *//what object are you aiming the bullets towards*  
        moveDirection = (target.transform.position - transform.position).normalized \* moveSpeed; *// move towards the targets location at the given speed*  
        rb.velocity = new Vector2 (moveDirection.x, moveDirection.y); *//the new direction in which the bullet will move*  
  
    }  
  
    void OnTriggerEnter2D (Collider2D other){  
  
  
        if(other.gameObject.tag.Equals ("Pla"))  
            {  
                  
                Destroy(other.gameObject); *//destroy player*  
            }  
        }  
    }

Enemy:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class Enemy : MonoBehaviour {  
  
    [SerializeField]  
    GameObject Bullet; *// what object will be the bullet, this iwll be se tin the inspector.*  
  
    float fireRate; *//how often it fires*  
    float nextFire; *// how much time between the fires*  
  
    *// Use this for initialization*  
    void Start () {  
        fireRate = 1f;  
        nextFire = Time.time;  
    }  
  
    *// Update is called once per frame*  
    void Update () {  
        CheckIfTimeToFire ();   
    }  
  
    void CheckIfTimeToFire()  
    {  
        if (Time.time > nextFire) {  
            Instantiate (Bullet, transform.position, Quaternion.identity);  
            nextFire = Time.time + fireRate; *//check if its enough time to fire*  
        }  
           
    }  
  
}

Step 4:

We now need to set up the inspectors to ensure our code will work. In the player’s inspector, change the Move speed to your desired speed.

We need to make the bullet a Prefab, as it will not start on screen, but appear once the game is running. In order to do this, simply go onto assets, create prefab and drag the Bullet object we have been previously using onto the prefab. It is now a prefab. Ensure it is renamed Bullet.

For the enemy inspector, ensure you have dragged the bullet prefab into the Bullet option.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Estimate Start | End | Interruptions | Task |
| 21/03/2019 | 2.17pm | 2.30pm | 2 minutes | Creaitng assets |
| 21/03/2019 | 2.32pm | 2.38pm | 0 | Adding Materials |
| 21/03/2019 | 3.00pm | 4.32pm | 5 minutes | Writing the code |
| 21/03/2019 | 4.44pm | 4.50pm | 0 | Setting up inspector |