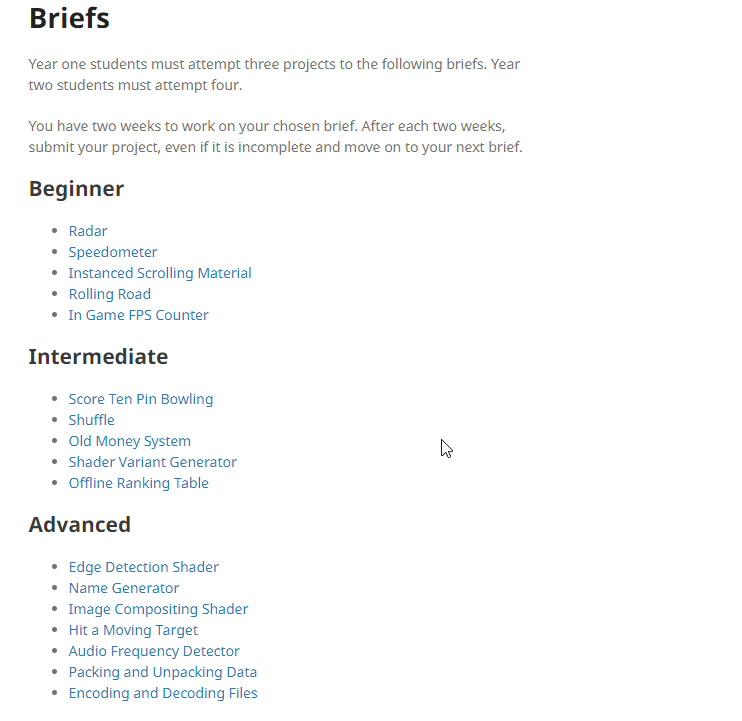
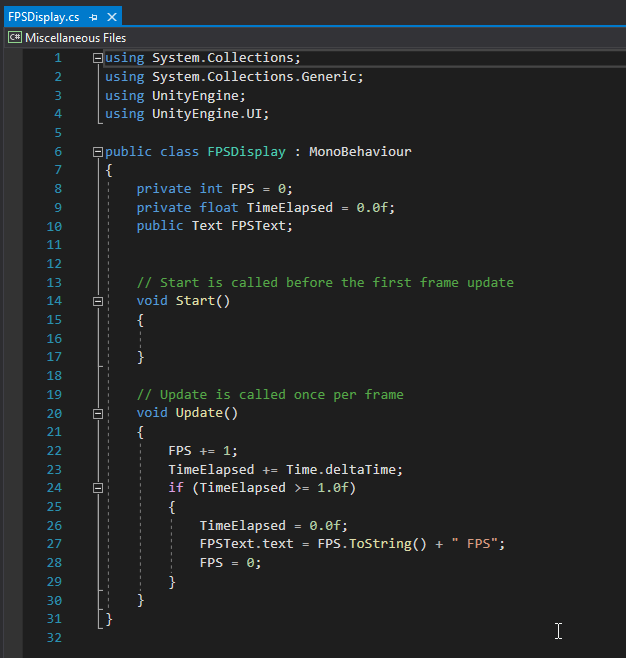
Reflective Statement - Programming

# Introduction

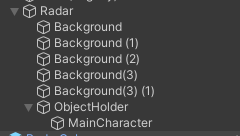
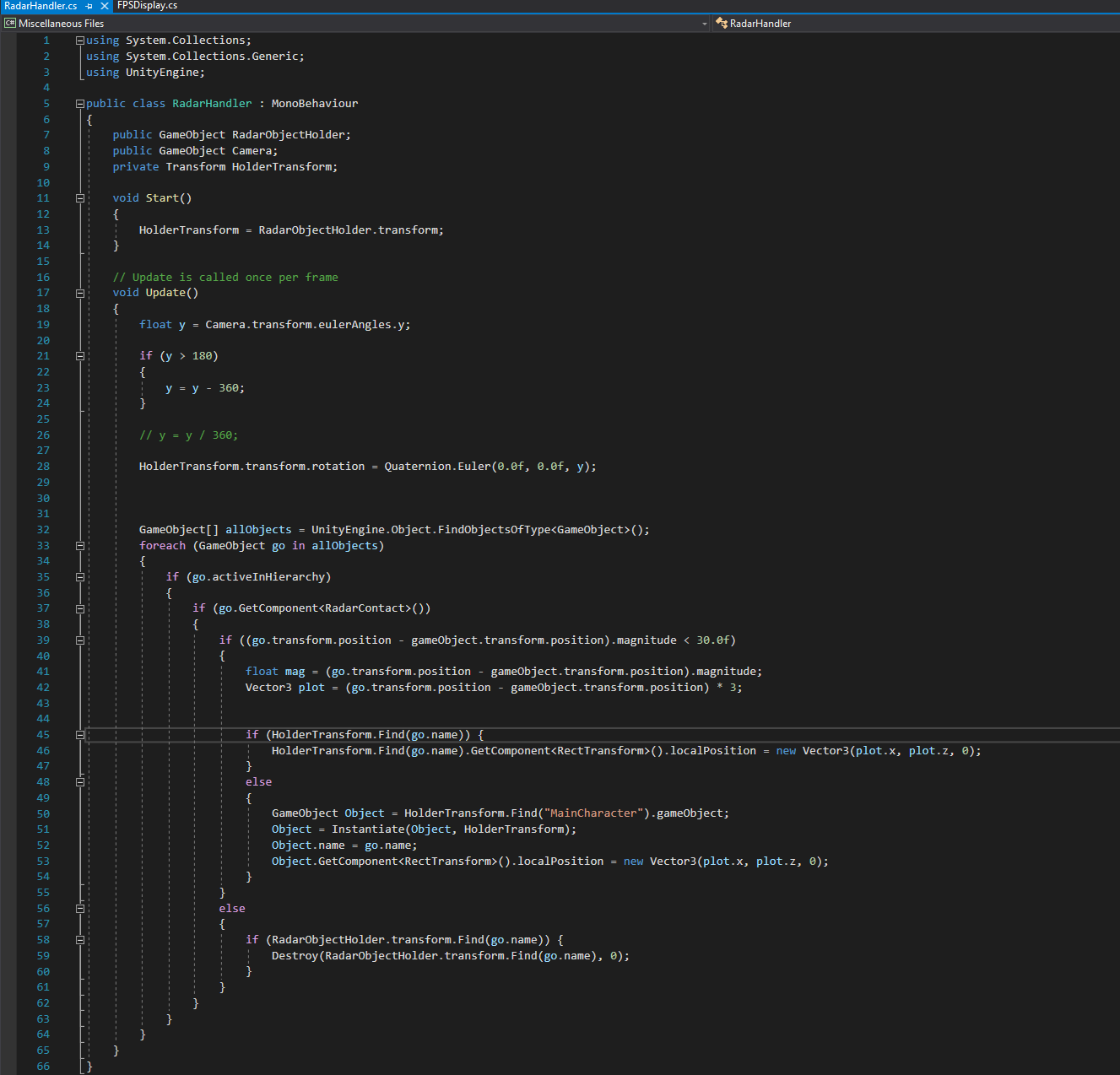
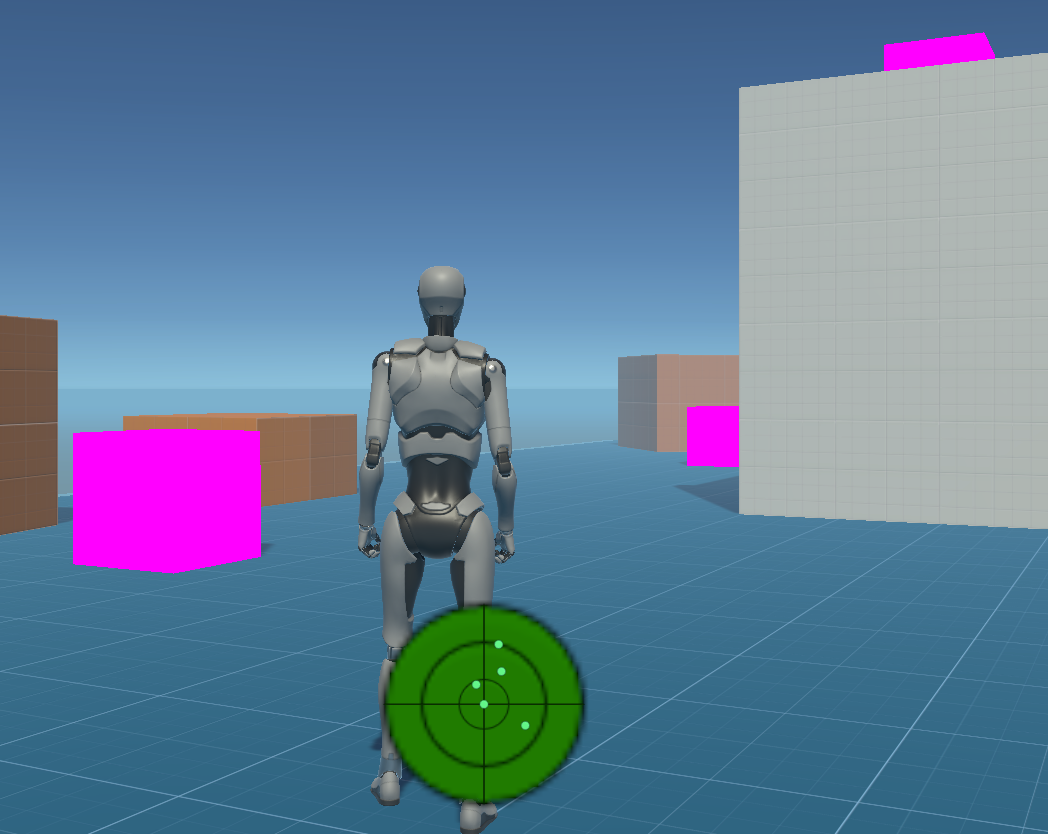
The general goals for the programming game specialism was for me to become comfortable with the basics of C# in Unity. At the start of the semester I had no knowledge of the language and how it operated, so diving into the advanced tasks would’ve caused a lot more problems for me than the basic ones. I had to set myself up with as many of the fundamentals of C# to prepare me for programming at a higher level.

# FPS Counter

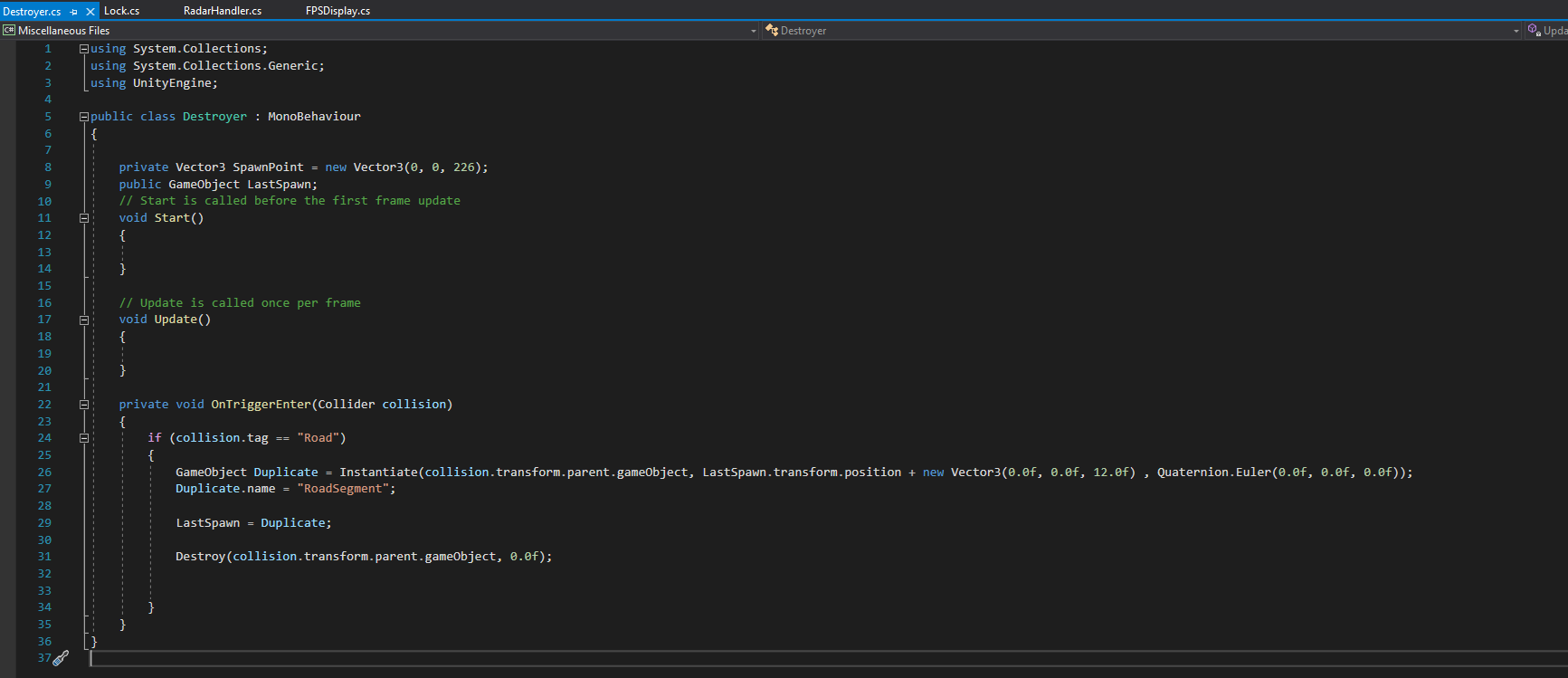
Undeniably the easiest one of all the briefs, it required learning how the User Interface works for Unity which consists of Canvases and Panels with various elements on them. The FPS Counter required a Canvas and a Text label connected to a C# script. The Script added 1 to a private integer variable called FPS every single time the Update() function ran (every frame) and would also count up the time elapsed from when the script started. If the time elapsed was over 1.0 seconds then the script would update the text label with the new FPS number and reset it to 0, as well as resetting the time elapsed back to 0 seconds.

I had to also create a public Text variable called FPSText, which linked to the “Text (Legacy)” game object seen in the canvas. This was so I could easily update the text every second to the FPS by just dragging the text object into the field inside of the script. I also had to concatenate the string with “ FPS” so that it would display “165 FPS” which makes it look more appealing than just a number. String concatenation is also a massive part of game development at the higher levels so it was best for me to learn it as early as possible.

# Radar

The radar caused many problems due to requiring it to always face the forward direction of the player object. I started by creating the radar using just blank UI elements such as a panel and numerous images. I created a blank panel called ObjectHolder which was going to hold all the plots of the objects that the radar was scanning for. I then created a pink cube called “RadarCube” and attached a blank script called “RadarContact” to it, allowing the radar to be able to specifically search for the cubes. Plotting the points was very straightforward. All that was needed was the difference of the 2 positions (target - origin) to get the difference of the vectors which shows the direction the target is compared to the origin. I simply turned that difference into X and Y coordinates to create new plots for the radar cubes and then they would be placed accurately away from the character. On the right you can see all the code used for the radar. I used Instantiate as a way to create the plots and it copied the MainCharacter point from before. This script runs every frame so the tracking is in real-time, like the radars used in call of duty and fifa. If the object is out of bounds then it destroys it because it is not within 30 units of the player. Rotating the radar was definitely the hardest part because Quaternions and rotation is very different to other game engines. After around 3 hours of testing different ways to rotate the radar based on the camera’s rotation I discovered that the best way to do it is to use Quaternion.Euler alongside a small piece of code that spreads the range of degrees from -180 to 180 instead of 0 to 360, making it seamless as it goes around.

# Rolling Road

I had actually done a rolling road type of game for my A Level final exam so all the logic for this brief was in my head. The main thing for this was Instantiate which I had already learned from creating the radar so this did not take long. I created a prefab for part of a road called RoadSegment which included a box collider and a rigidbody. I also created an object called Destroyer which holds all the logic for the game which destroys roads when touched and generates new ones immediately after. The Destroyer script was a script that had an OnTriggerEnter function which checked for the tag “Road”. It would then create a duplicate of the road and spawn it 12 units behind where the last one was spawned. Afterwards I created a second script called Lock which locks the rotation and velocity of the roads to ensure that it is seamless throughout the entire play. I originally attempted to use a Constant Force to keep all of the roads moving in the same direction but that had a dampening effect on it which caused it to accelerate at the start instead of immediately being full speed. That issue was fixed with Rigidbody.velocity. Another problem I was faced with for this whole project was the use of IsTrigger on the roads because they were composed of multiple different parts. I had the white lines in the middle use box colliders as well and it caused the Destroyer script to generate 4 roads every time one was destroyed since it was counting the white lines as roads themselves. To fix this I had to disable the box colliders on everything except for the main gray mass part of the road I named Asphalt.

# Summary

Overall, I have learned almost all of the fundamentals of Unity C#. I have learned how to concatenate strings, create user interfaces, generate objects, apply a constant force to an object, create a timer using Time.deltaTime and use the difference of vectors to understand where 2 objects are compared to each other. These fundamentals and more have vastly increased my knowledge and understanding of C# allowing me to improve my programming for the Game Production group task as well as for my own benefit. Next year I hope to complete the advanced tasks as I will be practicing C# over the summer regularly.