**Specialism 1 Reflective Journal**

Selecting briefs

In choosing which 3 of the briefs I would tackle, I decided that I wanted 1 to be of an intermediate level so that I could gain some exposure to more intricate concepts of coding. For this intermediate brief, I decided to go with the ‘shuffle’ brief because the function of shuffling items is a must have to many modern titles in both the triple-A and indie circles. For the other two briefs I chose to make the radar and speedometer as I figure both of these were commonplace in some of my favourite types of games: first-person shooters and racing games.

Radar

I chose to create a mini-map radar for a 2D space. I wanted to distinguish between moving enemies and static collectables, and so I first had to code the enemies’ movements. I used a grass tile for the background and a UFO asset for the enemies and creating empty game objects that they would use as location points to alternate between. To make the scene feel organic, I set each UFO to move at a unique speed and to spend a unique amount of time at each location point. Using colliders, I made it so that these UFOs would force the player out of the way should he position himself in their line of movement. For the collectables, I used assets for random items: keys, moneybags, and med kits. I also used a wooden texture to mark the borders of the area and added an image of space for the out-of-bounds background. Finally, I used another UFO asset, this time with a little alien visible, for the player.

I created a circular mini-map and made it so that the enemies would appear as red blips, and the collectable items as green blips, every time the radar’s arm pointed in their direction. The blips then disappear in the same amount of time that it takes for the arm to complete another 360-degree rotation. I also set the enemies and collectables onto a specific layer so that other objects with colliders – that being the player and the map’s borders – won’t show up on the radar at all.

I did not succeed in the bonus task of making it so that the radar blips existed at the edge of the radar when objects were out of range. This is because the tutorials I found were mostly showing how to do this when the blips were constant, rather than flashing and disappearing like mine were.

The brief states that the radar must always point in the direction that the player is facing. But since the game is 2D, rotating the radar to achieve this would look awkward, and rotating the player’s icon on the mini-map would have been even worse, so my only other option was to rotate the entire game and keep the player static.

Speedometer

I opened a 3D Unity project and created a basic humanoid shaped player using a capsule as the body and a rectangle through its top half for the arms. I then added a large, flattened cube for the floor, added physics and began working on the speedometer. Once I converted the speed of the player’s movement onto the speedometer’s needle, I changed its size and position to fit the screen appropriately. For a while I had an issue with the player constantly falling over and the speedometer not slowing down along with the player, but by locking the player’s vertical axis and adjusting the code, these were quickly rectified.

Shuffle

Shuffling the actual list was relatively straightforward and didn’t require much code. Assuring that each shuffled list began with a different song to what the previous one ended with took a little extra thinking to get working. However, once the list was working as it should be, it was required that I create attest code to assure that the script always did as it was supposed to. This was a much more difficult challenge than writing the initial shuffle script.

An NUnit documentation website contained all the code that was needed to test my shuffle script. What was required for the task was code that would enable Unity to rebuild the list, shuffle it, and then “assert to check”. Once the test was working and could assure that the shuffle would always keep the first song of a playlist different to the previous shuffle of said list, I then created a second test that on a broken shuffle to make sure it was capable of catching the repeated song if it were to happen. With this working fine I was finished with the brief.

What did I learn

I learned the importance and relevance of layers and masks – making certain objects interact with one another without affecting others. I learned that information regarding one object can be taken and converted into something else, as with the positions of the items being represented on a radar and the speed of an object being represented by the rotation of a speedometer’s arm.

The testing code I had no idea even existed. It didn’t occur to me before, but when a piece of code is being relied on to make sure something never happens, it makes sense that there ought to be a way of running that code thousands of times in an instant to see if it’s true, rather than checking it manually. For example, even shuffling the music playlist 100 times and visually checking that the first song on one shuffle doesn’t match the last song on the previous shuffle would be an unnecessary waste of time, much less something like, say, checking that a series of items placed into slots of a particular order will always activate a switch whereas any other order would not. To test this without the testing code would require collecting those items and trying every combination of them dozens of times over which would take hours for a single QA tester, casting the company unnecessary amounts of money, when this way a coder could do it himself in a matter of minutes.

On top of these specifics, I also just generally familiarised with Unity’s interface and the conceptual workings of code.