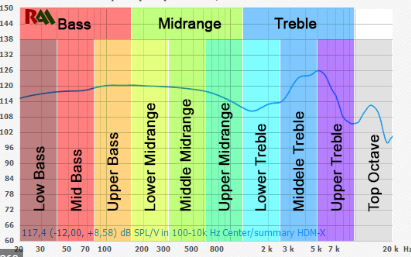
**Reflective Report**

**Audio Frequency Detector**

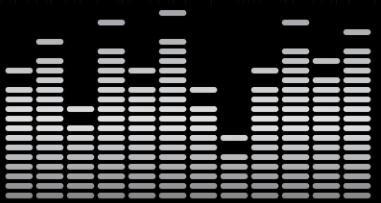
For my first brief I chose to take on one of the advanced briefs being the Audio Frequency Detector brief. The goal here was to separate audio at runtime into different frequency bands and trigger events based on select frequencies surpassing variable thresholds. To achieve this, I needed to read audio spectrum data at runtime from either a audio source or listener and to test the sample data I created an array that read all the sample data and displayed it within the inspector although this was changing too fast at runtime so I mapped this data against a bunch of primitive cubes using the data to scale their size on the Y axis, this however proved to show very small values and so I multiplied each value by some big number set within the inspector which resulted in something like this.



This was not ideal though as the bass bands showed high amplitude and their values ranged quite far. It was harder to read any of the other bands being the lower midrange, upper midrange and treble bands as they were of significantly higher frequency. To combat this I found data which displays what frequency ear band contains which is as follows.



Using the frequency data along the X axis of this graph I was able to multiply the values within my measured frequency sample array against the corresponding frequency band which normalized the values within the arrays and now displays a more traditional audio equalizer in which I can now read frequency data from.



Now I needed to group a few of the bars together to make a band and if any one of them passed a certain value I would need to trigger an event. When that was done, I could now shift my focus onto making the brief more accessible by generating all of the required display primitives at runtime during scene load and organizing them into separate parents also generated at runtime and finally adding all required components to an object once the script is attached.

**Instanced Scrolling Material**

For my second brief I decided to take on the Instanced Scrolling Material brief as I believed it seemed handy and would prove to be practically useful at some point. The required part of this brief was simple enough as all that is needed was to get a reference to the mesh renderer of the object with the material and then the reference to the material from there and then scroll their X and Y offsets using a Vector2 variable for scrolling speed as a function of time. After that I wanted to add more options to the script to make each material more unique such as variable material tint and shader support by allowing the user to choose what texture to scroll. Finally to add more ease to customisation, I added the ability for the material to scroll during runtime, which eliminates the need to constantly switch between scene view and playmode to ensure the material is scrolling at the user’s desired speed.

**Rolling Road**

Progressing to my next brief, the more I thought about the Rolling Road brief and how it would clean up after itself or not completely break. So instead of constant speculation I just started on the brief by making 3 prefabs: A straight road, a left bend and a right bend. These pieces had 3 main parts which were the start point, end point and mesh that connected the two. Initially the Road Manager would create a road of length 100 spawning each new piece at the end point of the last. However, these pieces did not connect together properly and always rotated towards the positive Z axis, to fix this each piece would inherit the rotation of the end point also which then had to face forward relative to the direction of the road. Now I had to fix the issue of the road colliding with itself. My initial solution was to only generate 10 pieces of road at a time which was not a complete fix as in few cases the road would still loop in on itself. Secondly, I then tried deleting the last segment then generating a different one to try and avoid the road, but this proved to be both inefficient and ineffective, so finally I opted to creating a check to the end of each piece which would raycast a set distance forward which was long enough to fit a new road segment, if a segment could not generate, the road would wait until that segment was deleted to generate a new one which would occur once the player runs over a segment.

To simulate the player running on the road I made a camera that would constantly run forwards and rotate on bends using a rigid body and trigger colliders for points where the dummy runner would rotate. There would also be a marker at the end of each road which once passed would delete the current segment and trigger to see if a road obstruction was cleared. To view the road generated at runtime, I also created a simple minimap that centered over the player that drew to a render texture asset that would display to a raw image on the canvas which was masked to a circular shape.

**Radar**

Finally, for my last brief I chose the Radar brief as it is another brief which seems practical and I have not attempted a code based radar before. First I set up the canvas and the rendering camera using the Render Texture method again along with another mask. Once this was operational, I made two scripts called Radar Manager and Radar Contact. When looking into making points appear on the radar I wanted to know how I could create world space sprites that didn’t rely on a canvas, however, I discovered I could create an empty game object that had its own canvas component (rendered in world space) and also a image component on that as well essentially creating world space images. Once I had a prefab template for my radar pings, I could then customize their settings from the Radar Contact script which attaches per object that appears on the radar, from this script I could control the sprite, color and size of each ping. As for the Radar Manager script, this would create all the radar pings in world space using the settings provided by each contact script and they would also use the same position of each object. I then made the resolution of the radar variable to allow the user to suit the radar to their needs and then proceeded to move each radar ping should they go out of the radar’s range by getting a vector direction between the player and each object then multiplying that by the current resolution of the radar. I also added a state where if the distance of a pinged object was twice that of the resolution the ping would disappear off of the radar. Finally for extra complexity, I added a raycasting system to delete objects and their ping off the map along with the object.