# Retrospective

For the radar brief, I initially had a lot of trouble translating world positions to points on the screen. I decided to convert the world space to radar space using conversion of metres-to-pixels, since unity UI uses pixels to position objects. This worked for the most part, but the positions did not update when rotating the player. I tried to fix this using quaternions, but it caused many more problems than it solved, so I decided to instead rotate the radar blip overlay by the same amount as the player’s current rotation. This gives it the same effect and was much easier to implement.

For the edge arrows I reused the same idea resourcefully, I already had a check in place to find out if contacts were outside range, so I made an arrow graphic and offset its centre so that the arrow would appear at the radar’s edge if the image was placed at the centre of the radar, and used this as a pivot to rotate the images towards their designated contact by calculating the angle to the player.

If I had more time I would have liked to add different colours for contacts, assigned within the contact script per object, as well as a height indicator similar to what is used in the popular game series Halo, where enemies above you become triangles, and those below you become darker circles.

The scrolling material was much easier than it seemed at first, I thought I would have to instantiate the materials myself before scrolling them to get multiple materials to work independently, but it turns out unity does this automatically. I decided instead to challenge myself to compress the script as much as I could and still have it fully functional, just for fun. I managed to get it down to 173 bytes.

If I had more time at the end I would’ve gone back to this to improve its optimization. Materials that are automatically instantiated are not cleaned up and stay in memory, which could have adverse effects on performance if left too long.

The predictive aim brief gave me no end of grief. I was having a lot of trouble with the maths calculations even with the calculation of local vs relative spaces, eventually after adding the character’s velocity vector to its position and using that as a target coordinate, it somewhat worked. This is when I figured out that it only worked with a bullet speed of 1, because this was actually a multiplier and the bullet was speeding up and slowing down based on the player’s distance.

None of my maths was salvageable, so I went back to the drawing board and began investigating the law of cosines, essentially the Pythagorean theorem for triangles with no right-angles. It took me hours of learning from various websites only to find that my resulting code didn’t work. I usually try to code without tutorials as it helps me to internalise what I learn much better, but since this was mostly about my weakest area, maths, I decided to cave and watch a tutorial on implementing the law of cosines in unity. Although the tutorial was written for 2D, it converted to 3D perfectly.

I was overall very happy with how this turned out, but if I had more time I would’ve taken time to learn the underlying maths properly and figure out the equations by myself.

I had a lot of fun with the name generator brief, as I was always curious about games like No Man’s Sky where they use a lot of similar ideas to generate random names for planets and systems. I wanted to make a mix of scientific, alien and sci-fi names, so I decided on writing a few lists to draw words from. I decided for pre-made words I would use the Greek letters, gods, elements and general space-themed words like “cluster” or “sector”.

These would be mixed in with some alien words which I generated by combining multiple phonetic sounds from an alien sounds array I made, creating names like “aktha” or “sa’al” by randomly mixing and matching, as well as occasionally placing an apostrophe. I also generated three letter initials for more scientific-sounding systems, using the English alphabet, and some randomly placed hyphens. For this I had to look up unity’s UTF-8 table, so I could generate an int between the min and max letters of the alphabet and convert them to chars and then to a string. I included options to disable initials and alien words too if more standard names were desired.

I made a lot of use of switch statements for much neater conditional name generation than if I had used if statements. The script generates two words using the functions “beginning” and “end” to combine together, but the value generated for “beginning” is global so it can be checked in “end” to prevent, or in some cases encourage repetition. For example, if the first word generated was alien, the switch statement in “end” would allow the second word to also be alien only if the first was, which allows occasional generation of entirely alien names.

I had initially wanted to write some code to calculate the number of possible unique names, but I had a lot of trouble figuring out the correct algorithm and implementing factorials in c#. if I had more time I would definitely like to implement this properly and have it update live as certain settings are tweaked and paths are enabled and disabled. I would also like to add more complex word generation, and improve the pronounceability of generated names, such as by adding a hyphens or apostrophes between stark sounds, or by somehow detecting if multiple sounds can be combined seamlessly.