Assignment 2 - Asteroids Arena 3D

Before you begin:

The command to build/run the game can be found in build.sh.

To control the ship:

- Move the mouse around to orient the ship.
- Press '**W**' to move the ship forward.
- Press '**S**' to move the ship backward.
- Press '**R**' to zoom the camera in.
- Press '**F**' to zoom the camera out.
- Press **ESC** to exit the game.

But most importantly…

- Have fun!

Parts undertaken in this assignment:

Arena: Easy/Medium/**Hard**
After going through what felt like a dozen different iterations of the arena, the week eleven lecture content gave a much needed boost into how to achieve the grid/mesh effect that was desired for each of the arena walls.
After that it the next hurdle was trying to figure out where I should store and change the colour values of the walls. My implementation of which is incredibly messy as we are always drawing the white walls regardless, and then we call another draw function for whichever walls the ship comes in warning distance with, albeit this time with the red colour.
Lastly, the skybox was somewhat straightforward once I had a good read through tutorial 11 and its examples, since it was essentially a barebones version of the skybox, this time just drawing the single wall and texturing it with the brick texture. Whilst my implementation of such is probably pretty unsightly, I feel it accomplishes the goal by creating 6 walls and 6 sets of texture coordinates to bind to, each referring to a different image of the cube map. From there it was as simple as placing each one in the space, with a distance of 1 from the camera, ensuring that it would always follow the camera on movement, but not rotation; and lastly just flipping each image by 180 degrees on the y axis to make sure it would all align properly.

Spaceship Model: **Easy**/Medium/Hard
The tinyobjloader-c implementation given to us is probably the single most problematic thing I have had to implement over the last three and a half years at RMIT. Zero thought appears to be put into not only the documentation/examples given by the developer but as well as the choice of this loader for people coding in C. Comparing side-by-side how much code both the C and C++ would need is ridiculous just to get it working. That aside however, once i was ***finally*** able to get it working (after nearly ***400 lines*** of code) the rest became fairly straightforward, since i already had proper ship movement implemented. Unfortunately due to the extremely undocumented and therefore difficult experience I had attempting to implement tinyobjload-c into my game; I was unable to figure out how to generate the texture co-ordinates for the model, as ***zero*** examples were given in the GitHub repository for the object loader.

Asteroid Model: Easy/**Medium**/Hard
Whilst I was initially going to just use the glutSolidSphere to draw my asteroid, I found myself a bit ahead of schedule in terms of the assignment, so using the tutorial 11 content I looked into making a procedurally generated asteroid using the draw_platform code. [TALK ABOUT TEXTURE MAPPING OR COLOURING/SHADING THE ASTEROID DEPENDING ON WHICH YOU DO]

Asteroid Movement: Easy/Medium/**Hard**
After having already implemented camera/ship movement prior to starting the asteroids; the movement for these was fairly straightforward for basic movement, since I just needed to adapt my code for the new asteroid structs. The random rotations for the hard part of the spec was fairly

straightforward too, as each asteroid stores a random x y and z value for which direction to rotate in that is selected between 0 and 1. The bouncing of asteroids off of arena walls and other asteroids was interesting to implement however; since I did not implement these in assignment 1. Since i was already calculating the ship's collision detection with the arena, I just adapted these but reversed the direction of the asteroid when it comes into contact, so if the X-axis comes into contact with the walls first then reverse the x value of the asteroid's directional vector.
The hardest part about this for me would have easily been multiple asteroid collisions, since I had to work out a way to compare each asteroid against one another, but not have it compare against itself.

Lighting: Easy/**Medium**/Hard
By the end of this course, lighting is still one of the more difficult things to wrap ones head around; so to accomplish the easy level I adapted code from tutorial 8 to achieve basic lighting on the asteroid, bullets and ship. I was unsure as to whether the lighting should be applied to the arena walls so I disabled it for them for the sake of clarity and being able to properly see the colours change without having to redo all the colour coding for it. For medium I am unsure as to whether my directional lighting is properly working, however the implementation currently exists, so feel free to check the code and make your judgements based off of that.

Bullets and Shooting: **Easy**/Medium/Hard
Bullets were another new concept to me, as I did not implement these in assignment 1. However it felt pretty straightforward in that we just shoot it out from the centre of the ship upon a mouse press, and travel until it either hits an asteroid or an arena wall. At first I toyed with the idea of having just a single bullet, making it so you need to make sure your shots count, but that felt too slow to play with so i made a maximum number defined by MAX_BULLETS which is easily configurable. Arguably this takes the difficulty away from the game since you could spam all bullets at an asteroid, but you would also need to wait for them to finish shooting before being able to fire it again. This falls in line with the spec since nothing was mentioned about a cool down like assignment 1.

Explosions: **Medium**/Hard
A basic particle system has been implemented for this assignment, so when asteroids are destroyed or a ship collides with the wall/asteroid then an explosion occurs at that point.

Camera and Ship Movement: Easy/**Medium**/Hard
In my implementation, the ship is tied to the camera. This being so when the camera moves, the ship's position is being translated from the camera's current position, remaining static on the screen at all times. The camera movement was the hardest thing to get my head around initially, especially since developing on Mac came with its own set of challenges when it comes to automatically warping the cursor; for which glutWarpCursor does not work on MacOS due to violating Apple's guidelines. However with the help of the discussion boards I was able to find a solution that works. I decided to not have too complex of a movement system, as the spec dictated that it is in fact up to us, so the player can move in most directions for the most part; except there is no roll feature as lecture 11 showed off, and there is also a vertical limit to how far the user can point the mouse/ship, as to save from getting disorientated.
For the medium level of this segment i opted to add in a zoom feature, accessed by pressing the 'R' key to zoom in, and the 'F' key to zoom out. Considering my moveCamera function already had parameters to set the camera behind the ship by a certain value, i was able to give the camera a 'zoom' variable that is adjusted when the user presses either of the zoom keys. I also defined a min/max zoom value in global.h to comfortable numbers so that not only does the camera not go in front of the ship's position, but that it also feels good to play still. I wanted to initially have this feature tied to the scroll wheel on the mouse; however i ran into problems with my scrolls not being detected when testing on a mac, so i opted for a selection of keys that are closer to the directional keys without being too interfering.

Code References:

Camera movement: https://learnopengl.com/Getting-started/Camera
Lighting: Tutorial 8 code
Procedurally-generated Spheres: Tutorial 11 code

Other References:

tinyobjloader-c: https://github.com/syoyo/tinyobjloader-c

spaceship model: https://sketchfab.com/3d-models/low-poly-space-ship-587941c9c11742c6b82dfb99e7b210b9#download

skybox textures: https://wwwtyro.github.io/space-3d/#animationSpeed=1&fov=80&nebulae=true&pointStars=true&resolution=1024&seed=5xoc3imbab28&stars=true&sun=true