Project Name: Stellar Wings

Viktor Fries, Lukas Hoffman, Brandon Huzil, Jared Thompson
CS 458

Progress

Our group has reached what we consider to be the "vertical slice" for our project. Our milestones present the MVP being reached before the vertical slice, but the way we developed each task changed our timeline. The way we presented our milestones did not take into account each member's tasks being carried out concurrently.

Milestone 1	Interactive cockpit designed (switches can be pressed, levers can be pulled, etc)
Milestone 2	Populated and moving environment (asteroids, other ships, etc)
Milestone 3	Barebones combat implementation (Weapons that fire, take damage) **– MVP**
Milestone 4	Polishing, additional ships, variety, adding visual effects to better user's immersion –**Target & Vertical Slice**
Milestone 5/6	Additional game modes, objectives, or a multiplayer aspect – **Stretch**
Milestone 5/6	Additional game modes, objectives or a multiplayer aspect – **Stretch**

Viktor was modeling basic cockpit designs, Lukas worked out the AI/pathing for enemy ships, Brandon worked on procedurally generating asteroid models, Jared created weapons and projectiles. With our milestones, it did not make much sense to wait or for all members to design the cockpit; we started our own individual jobs that we decided on previously. Results thus far include:

- Basic interior designed with working buttons on the console and "steering wheel" that controls the ships movement.
- Basic enemy movement. Enemies will move toward the player target position. The enemy will try to "intercept" the player by moving toward the front of the player's position.
- Procedurally generated asteroids of different shapes and sizes that when destroyed, spawn smaller asteroids.
- Game objects that spawn/fire out other objects via mouse clicks. This system will act as how our weapons will be implemented with button presses in the ships console.

Obviously these need to be refined, as well as integrating with each other's components to create an actual MVP. The cockpit still needs more buttons with scripts to do what they're supposed to do, as well as a polished steering wheel. Right now, there is just a game object that when grabbed and moved, can change the direction the ship is going. Exterior models still need to be modeled for player ship and enemy ships. The enemy AI flies toward an empty game object that is a child of the player ship, and a set distance in front of it. We would like to implement some sort of prediction of where the player ship

will be instead of homing in at one target location in front of the player ship at all times. The asteroids are having issues spawning smaller asteroids when destroyed. The small asteroids are spawning before the bigger asteroid is actually destroyed, so the smaller asteroids are moving unintendedly at times. Our weapons right now are just basic game objects like cubes, that fire out smaller spheres forward when the mouse is clicked. We still need to actual weapon and projectile models as well as how ships and asteroids will handle projectile collisions.

Success

One of the best achievements that was achieved thus far was the asteroids being procedurally generated. Brandon spent a lot of time learning about icospheres and mesh models to generate asteroids. The way these asteroids were achieved was by building a mesh in the shape of a morphed icosphere. Brandon wanted to modify the vertices for a mesh sphere but realized that most types of spheres not not have equal distance between all sets of adjacent vertices. This means that randomly altering the vertices coordinates would result in the asteroids shape having more noise in some spots then others, such as near the poles. The solution is an icosahedron, a shape of 20 equal triangle faces where there is full equality in vertex distances from each other and from the shapes center. We can take each edge on this shape and produce a new vertex in the edges center and project those vertices out from the shapes center and draw the mesh in a new vertex to vertex to vertex order, subdividing each triangle into 4. This refines the icosahedron into an icosphere with 80 faces. Icospheres are then morphed by taking random amounts of sets of 3 vertices corresponding with a single triangle and projecting them out or into the spheres center. We also changed the icosahedron shape equations with minor randomness the produce natural unevenness and roughness. The end result is quite impressive and fits our low poly aesthetic.

```
149     void RefineIcosphere()
150     {
151         // now we take each triangle and break it into 4 smaller triangles
152         // though we cant do this with all of the triangles or we will have
153         // double the nessicary vertices and wont be able to properly deform
154         // the icosphere
155         int verticesIndex = 12;
156         int oldVertexPosition;
157         int a, b, c;
158 
159         Vector3 potentialNewVertex1 = new Vector3();
160         Vector3 potentialNewVertex2 = new Vector3();
161         Vector3 potentialNewVertex3 = new Vector3();
162 
163         for(int x = 0; x < 20; x++)
164         {
165             // calculate the points halfway on the edges of a triangle and returns them in potentialNewVertex's
166             CreateSubTriangle(vertices[baseIcosehedronTriangleIndexOrder[x*3]], vertices[baseIcosehedronTriangleIndexOrder[x*3 + 1]], verti
167 
168             // tests if a vertex already exists
169             oldVertexPosition = ExistsYet(potentialNewVertex1, verticesIndex);
170             if (oldVertexPosition == -1) // if the vertex doesnt exist we create the vertex
171             {
172                 vertices[verticesIndex] = new Vector3(potentialNewVertex1.x, potentialNewVertex1.y, potentialNewVertex1.z);
173                 a = verticesIndex++;
174             }
175             else // else it already exists so we store is so we can add it to vertices[] in the proper order to draw triangles
176                 a = oldVertexPosition;
177 
178             oldVertexPosition = ExistsYet(potentialNewVertex2, verticesIndex);
179             if (oldVertexPosition == -1)
```

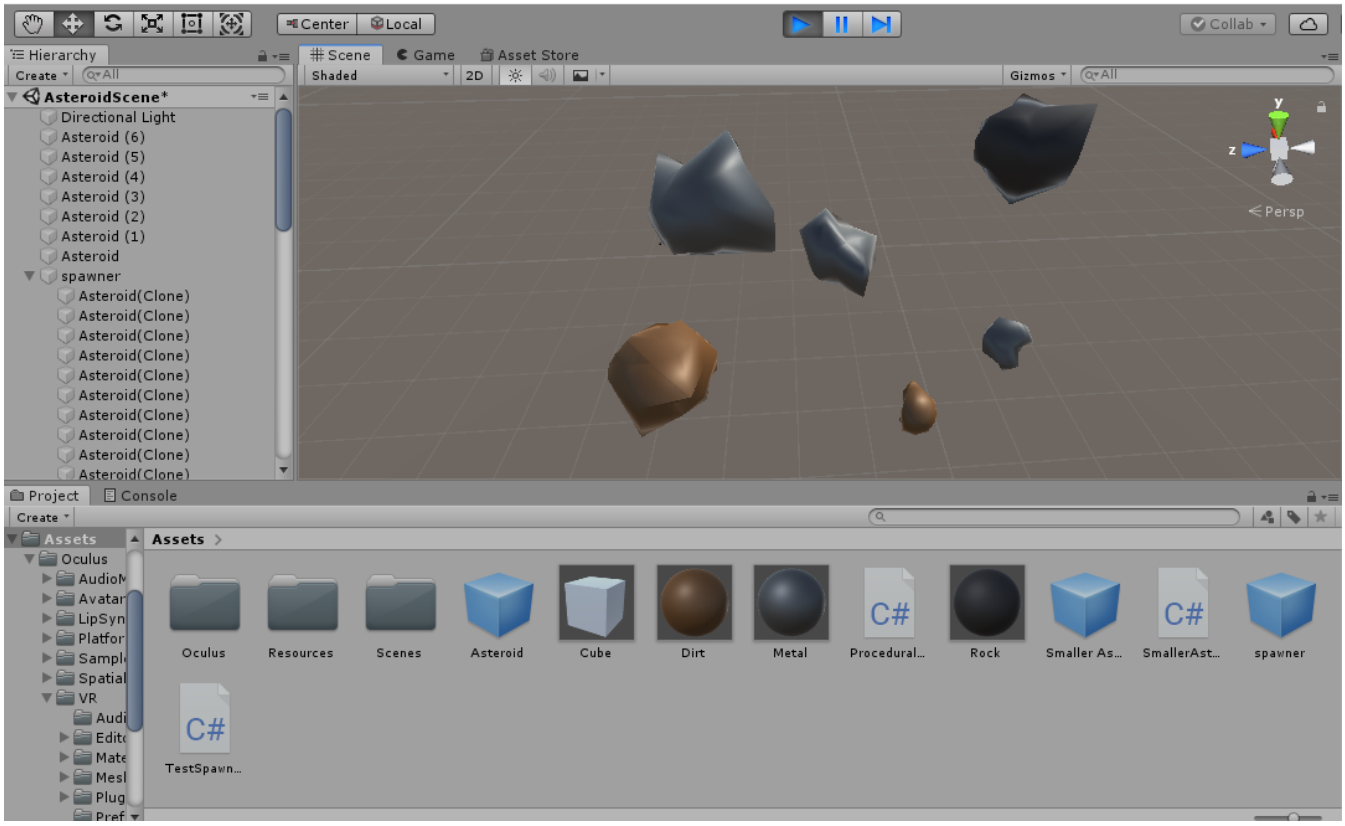
```
            oldVertexPosition = ExistsYet(potentialNewVertex2, verticesIndex);
            if (oldVertexPosition == -1)
            {
                vertices[verticesIndex] = new Vector3(potentialNewVertex2.x, potentialNewVertex2.y, potentialNewVertex2.z);
                b = verticesIndex++;
            }
            else
                b = oldVertexPosition;

            oldVertexPosition = ExistsYet(potentialNewVertex3, verticesIndex);
            if (oldVertexPosition == -1)
            {
                vertices[verticesIndex] = new Vector3(potentialNewVertex3.x, potentialNewVertex3.y, potentialNewVertex3.z);
                c = verticesIndex++;
            }
            else
                c = oldVertexPosition;

            triangles.Add(baseIcosehedronTriangleIndexOrder[x*3]); triangles.Add(a); triangles.Add(c);
            triangles.Add(baseIcosehedronTriangleIndexOrder[x*3 + 1]); triangles.Add(b); triangles.Add(a);
            triangles.Add(baseIcosehedronTriangleIndexOrder[x*3 + 2]); triangles.Add(c); triangles.Add(b);
            triangles.Add(a); triangles.Add(b); triangles.Add(c);

        }

    }
```



Complication

Implementing interact-able rigid body buttons proved to be much more difficult than expected. The initial implementation worked fine, creating a button that could be pressed with a physics object such as the player's hand, but it was quickly discovered that the buttons movement was locked in the world axis, and not in the buttons local axis. This is consequence of how Unity's freeze option works, and was not a simple fix. The problem was circumvented using a script that forces the button to only move in it's local y coordinate and bounds the movement based on a configurable range, which allows the button to be oriented in the world in any direction. A new problem arose though, because attaching the button to a moving object created unpredictable button movement. If the ship accelerates or turns, the rigid body component of the button does not follow it properly, and slowly moves and turns out of position. A simple solution to this problem was not found, and currently the best alternative is using a button on the Oculus controller while hovering over the virtual button to trigger a button press. This solution may not be quite as immersive, but it is functional and consistent, and will allow us to focus on the development of more interesting components for the game. If time permits, we may develop a better solution, more inline with our original vision.