Sphere Defense

Nathan Alam and Liam Johansson

COS 426 Final Project Written Report

Abstract

Sphere Defense is a tower defense game that twists the traditional gameplay mechanics by placing the world in a 3D, top down environment and by allowing players to alternate between controlling automatic turrets and a powerful, manually controlled aircraft. The game takes place on an icosahedral board, where a single tower exists to be protected against an incoming swarm of "blobs" that pathfind towards the tower. This game takes advantage of alternating camera perspectives when a player chooses to enter and exit the controlled aircraft, various animations to visually indicate increasing and diminishing health, projectile motion as turrets and aircraft fire at blobs, and semblances of physics to animate movement of blobs and turrets from one tile to another.

Introduction

At its core, the game contains elements of the tower defense game, with strategy similar to games such as the widely popular *Bloons Tower Defense* series[1], in which waves of enemies follow a predetermined track, and the player must place various turrets along the path to slow down their advance. The main twist to distinguish this game is the 3D geometry, which adds a layer of complexity as the player can never have a full view of the board, requiring them to instead interact shift perspectives to discern enemy troop movements. The main objective of this game, then, was to address the ever present challenge of video game development to come up with new ways to challenge players, encourage people to develop new strategies, and find a new way to balance difficulty with entertainment. While the *Bloons Tower Defense* game is widely considered successful, it is no secret that the strategies to win the game are fairly straightforward given the well defined path, and do not encourage experimentation.

The small 3D world concept was inspired by the *Super Mario Galaxy* series[2], where a major recurring theme is the existence of small planets acting as a staging ground for game events. While this series is a mix of a platformer and an adventure game, one thing we noticed was that the use of 3D spherical geometries, instead of more traditional 3D planar geometries, is that an element of surprise and discovery exists when events occur "behind the scenes", on the other side of the planet. Whereas in *Super Mario Galaxy*, this may have meant requiring the user to synchronize, say, buttons on different sides of the planet, in Sphere Defense, this may mean surprise at the discovery of a cluster of advancing enemy troops in an unaddressed area.

After realizing that the tower defense aspect did not provide for the most engaging gameplay, we decided to include a flight aspect. We felt that a more interactive part would benefit the game greatly, and the ability to travel the planet in an orbiting plane seemed like an interesting concept. The flight controls were our own design, but inspiration came from simplistic 2D

[1] https://en.wikipedia.org/wiki/Bloons_Tower_Defense
[2] https://en.wikipedia.org/wiki/Super_Mario_Galaxy

arcade-like flight games like *Altitude*[3]. A player's ability to intuitively control the path of a plane by simply pointing with the mouse, even if the flight path is not completely physically realistic, provides an easy-to-learn, unique method of traversing the game's world. A natural extension of flying the plane is the plane contributing to the defense with the ability to shoot.

We quickly found out that adding this 3D geometry, with multiple possible paths to the tower for the troops, can make this game considerably more difficult than traditional tower defense games, even before adding the complexity of competitors with varied enemy and turret types. In our approach, we decided to offload "game-balancing" issues to the player themselves, by giving them control of the spawn rate of troops, the range and damage of the turrets, and access to a powerful aircraft capable of doing considerably more damage. Furthermore, adding this flight mechanic helps secure players with two different video game preferences. The turret control mode may be more apt for players fond of board games or strategical, top down video games like in *Sid Meier's Civilization* series[4] or the *Age of Empires* series[5], but the fighter plane mode may be more interesting to players who prefer a manual, skill based operation rewarding reaction time, aim, and maneuvering, as is present in, for example, the *Star Fox* series[6]. All together, we believe that his mix of gameplay styles can appeal to a wider audience, and scalable difficulty may make initial steps with determining a good game balance easier after user testing.

Methodology

Site Hosting

Before we could get started with developing the game, we needed to decide how to architect the canvas for displaying the game itself. We wanted to host this on Github Pages to make this easily shareable with others, and so went with an HTML5 Canvas and Three JS approach to implement the project. Because Github Pages serves static sites, we did not develop a server to provide information for the game, which ultimately limits multiplayer functionality and the storage of any high scores on a real time server somewhere. Although we could have used a third party database solution like Firebase[7] to store data about user profiles, we decided that this would not be the main focus of the project starting out.

Viewport

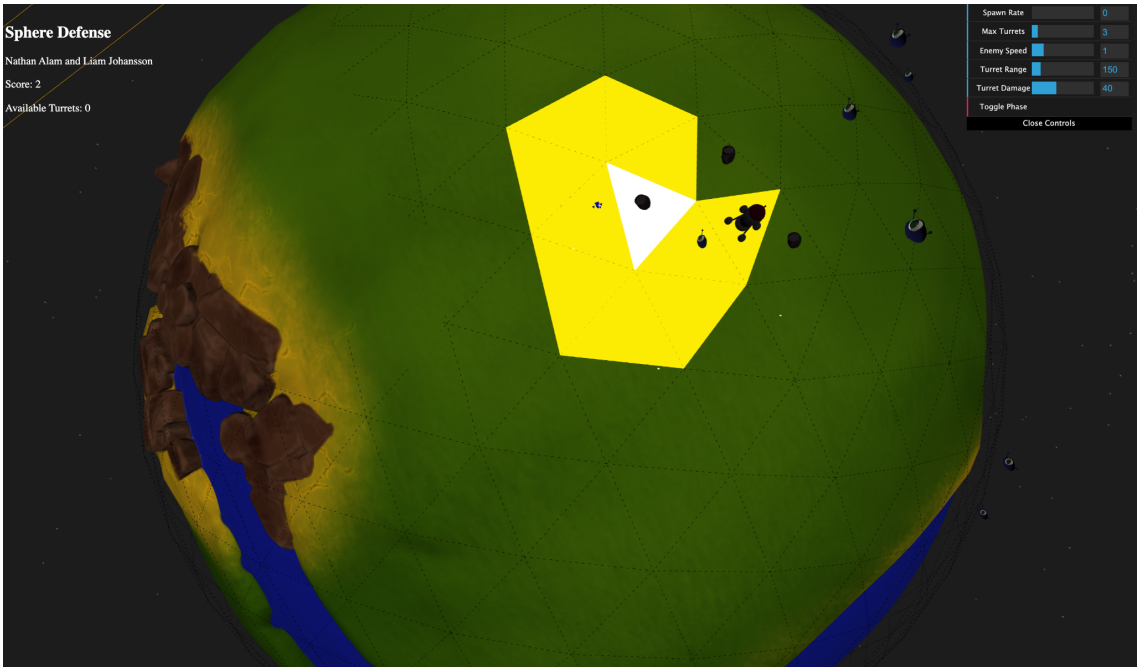
Initially, we started with rendering the canvas in a rectangle of uniform size within a larger webpage, akin to many old-school browser based games, with a fixed resolution of 600 x 400 pixels. However, in practice we saw that this led to a "claustrophobic" gaming experience, especially on larger screens. Additionally, behavior when the mouse pointer reached the edges of the game led to erratic behavior, especially during flight control. As such, we decided to move all the description text and interactions onto elements that hovered above the rendered HTML5 canvas itself. This allowed the game to take advantage of the full browser window, which was

[3] https://en.wikipedia.org/wiki/Altitude_(video_game)
[4] https://en.wikipedia.org/wiki/Civilization_(video_game)
[5] https://en.wikipedia.org/wiki/Age_of_Empires
[6] https://en.wikipedia.org/wiki/Star_Fox
[7] https://firebase.google.com/

particularly helpful as so much of the game involves interacting with the planet and searching for enemy blobs to destroy.

Trackball Camera

Our first game development challenge was determining the camera experience for the users in interacting with the world in the top down view, which would give them the best view of where towers, turrets, and blobs were at any given time. We chose to use Three JS's TrackballControls[8], which allowed fluid and intuitive interaction with the world using the mouse on a computer screen, and even with touch screens on applicable devices. Originally, we had tried the OrbitControls[9] instead, thinking that this physically made sense as the camera would orbit around the planet's core, the origin. However, we saw that in practice the OrbitControls encouraged the developer to enforce some notion of an "up" direction, which was not meaningful in our game where "up" was simply the normal vector for any given tile on the board.



Flight Mechanics

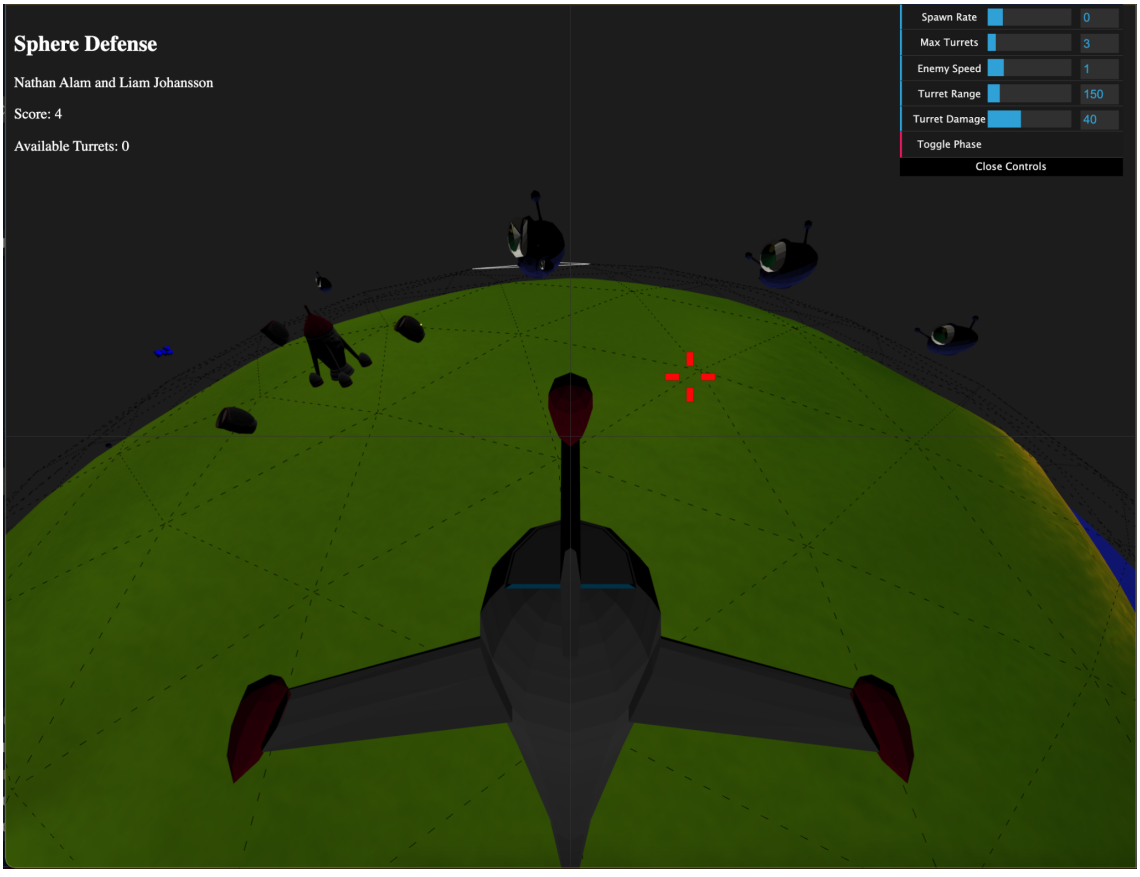
For the flight phase camera, we shifted the perspective camera to be situated above and behind the fighter plane. Mouse position controlled the pitch, yaw, and roll of the plane as it flew forwards. The responsiveness of the plane to the player's mouse was an important and tricky aspect of the project. We wanted the player to only have direct control over the angular velocity and speed of the plane. This means that the player influences the position of the plane indirectly through changing the rotation which changes the velocity, as in an actual plane. Additionally, we wanted to avoid allowing the plane to flip upside down. In other words, the direction describing the bottom of the plane must stay with pi/2 radians of the direction toward the center of the planet. We also wanted the plane to 'lightly orbit' the planet, meaning that the plane will mildly

[8] https://threejs.org/docs/#examples/en/controls/TrackballControls
[9] https://threejs.org/docs/#examples/en/controls/OrbitControls

curve toward the planet and not fly away to infinity with no input. These requirements made the controls a challenging geometry problem.

Our solution involves storing the positional information throughout various nodes in the scene graph. At the global level, a rotation describes the angular position of the plane relative to the center of the sphere. Along with a position vector that gives the distance from the sphere's center, we get the world coordinates of the camera. A local rotation then describes the orientation of the camera at that position. Finally, a rotation of the plane's mesh accounting for roll describes how to display the plane.

The player's mouse position in NDC informs the local angular velocity in a natural way (x-coordinate gives yaw and roll, y-coordinate gives pitch). At each time step, the angular velocity is used to update the local rotation, which gives the current velocity vector. The velocity vector is used to compute the new world position, and the difference between the old and new world position is used to update the global rotation and the height from the center of the planet. By storing position information in rotations rather than 3D vectors, we get the property that local rotation relative to the planet does not change when there is no angular velocity. For example, if the bottom of the plane is pointed directly at the center of the planet and the player flies the plane directly straight, then the plane will orbit the planet, since the bottom of the plane must remain pointed at the planet. This gives us both the 'light orbiting' idea as well as making it easy to prevent flying upside-down, since we simply clamp the local rotation corresponding to pitch between -PI/2 and PI/2.



Tiling

The board is represented by a tile centric data structure, with the troops, turrets, and tower all interacting with the board's array of tiles, instead of the board mesh itself. This tile centric approach provided a key advantage in flexibility over an alternative approach that would have had each of the aforementioned units interacting with the board mesh faces directly. This flexibility in allowing us to choose what information to encode in a "tile" allowed us to easily implement what would have been fairly complex mechanics. For example, we could store a list of adjacent tiles within the tiles themselves, instead of having to calculate this manually every time a troop or turret must consider its available movements. Another example lies with the troop pathfinding, when generating the array of tiles, we could simply calculate and store the distance of each tile from the tower, and so when a troop pathfinds, it simply moves to the next adjacent tile with the minimum distance from the tower. While currently, the tiles correspond to a face on the mesh, future work could also incorporate more complex 3D board geometries, and abstract several mesh faces to be a member of a single tile.

Hop Animation

To generate the animation curve for an enemy troop or a player controlled turret to hop from one tile to another, we used the a catmull-rom spline[10]. We chose such a curve since it produces a smooth curve interpolating four points, and it has a built-in method in Three JS. The middle control points lie above the line connecting the two endpoints, and they were chosen empirically. When traversing the curve, we chose a parameterization such that the entity moves faster toward the beginning and end of the animation in order to better represent a hopping motion. Specifically, ds/dt scales linearly with absolute difference from the midpoint time.

Unit Health

At this stage in development, we chose to simply scale the size of object meshes with the health of each unit. However, we did not want the units to approach infinitesimal size as they lost health, as that would also make them impossible targets to hit. Instead, we scaled it such that the minimum size of a troop or tower was 25% of its original size. We also added a mechanic such that the troops increase in size with every hop. This added a major incentive for players to move some turrets away from the tower, because if a troop were to spawn far away from the tower, by the time it reached the tower it would have grown in size and overwhelmed any defensive efforts late in the troop trajectory.

Bullet Animation

To display a visual indication that a turret was attacking a blob, and to specify which blob it was attacking, we employed a bullet animation that fired from a turret to an enemy blob. We also included this visual when the plane was firing. The visual involves drawing small, illuminated boxes traveling fast in a straight line. The turrets can never miss a blob. For the player controlled firing, we decided to check hits using a raycaster rather than attempting to find bullet intersections with a blob. This is because hitting moving objects is significantly easier with hitscan as opposed to finite velocity bullets, and flying the plane already adds significant

[10] https://threejs.org/docs/#api/en/extras/curves/CatmullRomCurve3

difficulty in hitting the enemies. Additionally, hitscan should be less intensive from computational and coding perspectives. Since the plane is displayed in third person view, the bullets do not spawn directly along the camera outward axis. To make it appear like the bullets actually cause the damage to the blobs, the paths of the bullets are adjusted so that they pass close to the point where the aiming intersects the blob.

Object Loading

To model the plane, tower, and turrets, we used predefined custom meshes stored as .gltf files to incorporate interesting geometries. These 3D models were obtained from the default Windows 10 3D library and altered through Microsoft's 3D builder[11] and copied into an *models* directory which is used to load our models into the game. Sounds were obtained from Zapsplat[12], and stored in a *sounds* directory. In the future, though, we would want to develop models with a common design language with the blob troops so as to ensure good fidelity throughout the game and improve aesthetic appeal.

Results

Our main metric in evaluating our game during development, in lieu of active user testing and feedback (which will be collected after the submission of this report), was the time the game lasted before it ended, when enemy troops overran the tower. Our target duration was approximately 3 minutes, which would make for a good short mini game. The reason we did not want a game that lasted significantly longer was because we had yet to implement progression mechanics, such as troops spawning more frequently as time passed or upgrades to turrets and planes. If these were added, though, the game might be fun enough to warrant taking up more of a player's time. To balance this limit, we had to scale default troop spawn frequencies and turret damage to account for changes to game balance. The plane, for example, tilted the game in the player's favor, but the growing troop mechanic tilted it again in the favor of the enemy. In the end, though, we will probably want to gauge interest among actual players and assess difficulty from there once feedback gets sourced.

Discussion

While the game has been fairly fun to play in personal testing so far, a major area of further work remains gathering user feedback for the parts of the game that truly are fun. A major challenge exists in distinguishing whether certain game mechanics, like the 3D board and real time mechanics, are fun or just stressful, and because that element is ultimately suggestive and varies from individual to individual, user testing will be the main focus of future work. However, we believe that because of the many control points that we have set to be easily adjustable, such as the turret power and troop spawn frequency and speed, we have many different ways to adjust this game and balance it to be more enjoyable. Taking this approach, we believe we have derisked one of the main anticipated challenges of a game that is too difficult in comparison to traditional tower defense games.

---

[11] https://www.microsoft.com/en-us/p/3d-builder/9wzdncrfj3t6#activetab=pivot:overviewtab
[12] https://www.zapsplat.com/sound-effect-category/alarms-and-sirens/

An alternative approach we had thought of towards the beginning would have been a more symmetrical game, with two towers by two players and a turn based game taking place instead. This, too, would have been a tower defense game, but instead of something like *Bloons Tower Defense*, it would have felt much more like traditional board games. It is difficult to say whether this game would necessarily be better, as it could also have led to interesting strategies employing the 3D space that 2D board games cannot but may also lose the hectic nature of the real time game we implemented here.

Further work exists to increase the visual appeal of the game, by using better models for individual units, texturing the planet with color and normal mapping, and introducing time progression in sky color somehow. Additionally, another major game mechanic that could add to complexity and visual appeal would be the introduction of randomly generated obstacles such as foliage, that could alter troop pathfinding and create potential strategic choke points for turret placement.

One major lesson from the project was the tricky balance in game development in managing difficulty, and how in early stages, parameterization of game parameters can be very useful in introducing some flexibility. We were also surprised at the relatively low computational load that the game has, despite the real time nature of the game with many animated units. Although Javascript is inherently single threaded, it was fairly surprising that the game still runs quite smoothly on standard modern hardware, given the hit scan and animation calculations being performed constantly. Finally, another lesson we learned was that user interaction workflows can be quite complex and challenging to encode. For example, representing the workflow behind selecting tiles to spawn turrets and move them results in a fairly complex deterministic finite automaton (DFA) that is encoded in a lengthy series of “if-else” statements.

Conclusion

Although the game has diverged from the original aim of a two player, two tower turn based game, we think it is more interesting and takes far better advantage of the 3D planetary geometry than the original concept would have allowed. The various control points in game difficulty have led to a game with a highly tunable sense of game balance, and efficient use of scene groups has enabled the game to remain performant and run smoothly. User testing remains one of the major outstanding steps in evaluating the enjoyability of the game, where we will figure out how to best tune game parameters, and added visual polish should improve the aesthetics of the game to make it an attractive and presentable project that is fun to play.

Contributions

Because the Icosahedron geometry was not an indexed BufferedGeometry, we had to employ the use of a third party adaptation[13] that could convert non-indexed geometries to indeed ones. To load objects, we used another adaptation[14] that appended an ObjLoader class to Three JS.

[13] https://cdn.rawgit.com/mrdoob/three.js/master/examples/js/loaders/GLTFLoader.js
[14] mrdoob / http://mrdoob.com/

Works Cited

1. https://en.wikipedia.org/wiki/Bloons_Tower_Defense
2. https://en.wikipedia.org/wiki/Super_Mario_Galaxy
3. https://en.wikipedia.org/wiki/Civilization_(video_game)
4. https://en.wikipedia.org/wiki/Age_of_Empires
5. https://en.wikipedia.org/wiki/Star_Fox
6. https://firebase.google.com/
7. https://threejs.org/docs/#examples/en/controls/TrackballControls
8. https://threejs.org/docs/#examples/en/controls/OrbitControls
9. https://threejs.org/docs/#api/en/extras/curves/CatmullRomCurve3
10. https://www.microsoft.com/en-us/p/3d-builder/9wzdncrfj3t6#activetab=pivot:overviewtab
11. https://www.zapsplat.com/sound-effect-category/alarms-and-sirens/
12. https://cdn.rawgit.com/mrdoob/three.js/master/examples/js/loaders/GLTFLoader.js
13. mrdoob / http://mrdoob.com/