Jack Stanley and Jack Kenkel
COS 426
Sharks and Minnows

Abstract

We created Sharks and Minnows, a multiplayer game where two players, a shark and a fish, compete against each other through an online server. The game's world is rendered with ThreeJS and utilizes features such as GLSL shaders, texture mapping, and audio to create an immersive and realistic world. We implemented our own collision detection to allow the shark to eat the fish and the fish to eat food and created a heads-up display to give information about the current score, condition of the online connection, and the win and loss messages. While there are many extensions that could be made to the gameplay, the final product is functional and fun to play.

Gameplay and basic features

Sharks and Minnows is a two player game where one player plays as a fish and the other as a shark. It is set in a 3D world that contains procedurally generated schools of fish, which are identical to the main fish and fish food (gold items) throughout. The player in control of the fish wins by collecting enough food, while the player in control of the shark wins by eating the fish. The fish can hide in the schools of fish to avoid detection by the shark.

The approach we took (as suggested in the specifications) to developing our game was to first work toward an mvp. We utilized the provided starter code and then worked toward a simple game that consisted of a cube representing the shark and fish which could swim around. Additionally the food and schools were just cubes at random locations. For this part of the process we worked relatively together, so we could bounce ideas and get a basic product down. Once we had a basic game working we started to divide and conquer in order to maximize efficiency and time. Stanley focused on vertex shaders and texture mapping, implementing networked multiplayer capabilities, and collisions. Kenkel focused on the onscreen control panel, sound, and models. While, we both continued to focus on general gameplay and features.

The Fish: The fish is a 3D model imported in gltf format using THREE.js support for gltf. One of the challenges we faced early on was finding a model that was easily cloneable. The first model we used was animated and made out of bones, which gave us difficulties. A long term extension of the project would likely be adding a model with skinning/animation.

The Shark: The shark is also a 3D model imported in gltf format. Adding movement to the shark, such as a swish of the tail would be another extension that would be cool.

Controls: The controls for the fish and shark are similar. The camera is set up to be positioned behind the shark/fish and visually follows the movement maintaining its constant separation. The fish/shark both move forward at a steady velocity at each timestamp and their direction is controlled by user input through the mouse. The shark moves quicker than the fish in general, which is to make it possible for the shark to catch the fish. The shark however has worse rotational abilities than the fish, which is to make it more fair. This allows the fish to have

an easier time navigating to food and to the other school so it can hide. A potential extension that we were thinking about, but ultimately decided against was making it an option for the players to adjust the rotational sensitivity of their character through the on screen control panel (hud). We decided not to because it made it easier for us to modify the shark and the fish to make the game more even. While the shark is faster than the fish the fish has two special powers that we implemented on the space bar and the shift key. When the space bar is pressed the fish enters a boosted state which when in boosted sets increases the forward velocity of the fish to just below the speed of the shark. This was something we decided on in the process of game development as it made the game more fun and challenging since the fish could now dart out to grab food/catch up to schools of fish. The left shift activates a second orthographic camera positioned to be pointing down from higher on the y axis. This allows the fish to see where the shark and other schools of fish are in relation to it. Another potential extension that we were thinking about here was making this to be a multiple view, such as the one described in the features that is simultaneously displayed while the fish travels but from a gameplay perspective it seemed as though it would make it too easy for the fish if it could constantly see the sharks location.

Advanced Features

The first advanced features that we decided to implement were custom texture mapping and GLSL shaders. We chose to start with these because they relied on much of the same principles and made a huge improvement to the scenery on our ocean floor. We approached the implementation using the ThreeJS fundamentals article on using ShaderToy shaders with ThreeJS. This involved writing a fragment shader with two main functions: first, it takes texture data as a uniform and maps it to the given fragCoords. It then uses Perlin noise based on the fragCoord and simulation time, adding or subtracting brightness to the pixel based on the noise value to model the diffraction of light from the ocean surface. Finally, we needed to create a vertex shader to map from screen coordinates to coordinates on the plane that represented the ocean floor.

The next feature we tackled was the online component of the game. In order for the whole premise of our game to be viable, we needed two people on separate devices to be able to play together so that they could effectively hide from each other. To accomplish this, we wrote a NodeJS server using the *ws* package to interact with the native JavaScript `Websocket` object on the client side. The server sends a signal for anyone trying to join a game a wait signal until another person joins, at which point the server creates a party for the two clients and signals for the a game to start. We wanted to keep the workload of the server as light as possible, namely avoiding sending the positions of hundreds of randomly moving fish on every frame. We worked around this by procedurally generating the schools of fish based on a random seed decided by the server on game start. This seed is sent to both clients, who each use it to generate a series of random values to decide the starting positions of the food and schools of fish. This procedural random number generator continues to be used to decide which way schools of fish bounce off the walls as well. Finally, the server is in charge of passing messages about the current score and when either player wins the game, causing win messages to appear when appropriate.

Collision detection was another important part of our games functionality. In order for the fish to collect food or the shark to eat the fish, we needed a way to detect when the objects were close enough and trigger an event accordingly. This was achieved by making a `SphereCollider` class whose constructor took the two objects to detect collisions between, a radius of each collider's sphere, an offset from each object's center, a callback function, and finally a boolean that decides whether the collision should trigger more than once. When the collider's `checkCollision()` was called during the main scene's `update()` method, the collider takes the positions of the objects, applies the given offset, and checks if the distance between the points, calling the callback function if the distance is less than the sum of the two radii. Since this function was being called many times per frame, we optimized it by pre-calculating many of the comparison values in the constructor and checking squared distances to avoid expensive square root calculations. A collider was created between the shark and fish as well as one for each food and the fish.

The onscreen control panel or HUD was another feature we chose to implement to provide players with relevant information about the game. The HUD is used to dynamically update the score (for the fish player) by incrementing the score by one each time a collision between the fish and food is detected. The control panel is also used to control sounds with the user being able to update the status of a checkbox contained in the panel that allows the music to play when it is checked and stop when it is not checked. It also creates the headers that pop up when the fish wins or when the shark wins. If the shark collides with the fish it displays that the shark has one. If the score for the fish is greater than or equal to the number of food pieces the fish needs to collect to win then it displays that the fish won.

Another advanced feature we chose to implement was sound to enhance the user experience. The option to have sounds and music is controlled through the HUD with a checkbox to select music or not. There is general background music with an uptempo feel. On the collection of food a coin collecting sound plays to reward the fish and to let the shark know another piece of food has been collected. When the shark collides with fish and wins the game a shark bite sound plays to the delight of the shark and agony of the fish player. The audio files are imported using audio() to create an HTMLelement that can be played in association with actions or in the case of the background music played on a loop if the user selects music.

Extensions

While we accomplished many of the functions we intended for Sharks and Minnows, there are still many ways that we could improve and expand upon our game. There are more features that could improve the variety and replayability of rounds as well as some smoothing of the current features. For example, while the procedural moving of the schools of fish worked great on paper, in practice, slight differences in latencies between the two clients sometimes leaves the fish out in the open from the shark's point of view even though the fish sees itself hidden in a school. This could potentially be improved by either adding an element of synchronization to the server or by eliminating the procedural generation of the fish in favor of simply relaying the position of the schools every frame. Taking the latter approach would also allow for the schools' movements to be more advanced, for example using a version of the boids algorithm to let the fish move more freely.

Another improvement we could make to our existing game is some animation of the fish and shark models. We originally used a fish model built from bones and skin, however ran into issues with instancing where we couldn't get ThreeJS to share geometry amongst the hundreds of fish models, leading to slow loading and a lot of framerate issues during gameplay. With more time, we could have worked through this problem to instance the animated model or updated the geometry of our new model every frame to animate its tail and create a swimming effect.

In addition to some of the improvements we could make to our existing features, we also see the opportunity to expand our game to be a more battle-royale style game by letting multiple fish join a party with one shark and have the fish compete to be the last one standing. This extension would only require changing the server's policy on creating parties and support to send the coordinates of all the clients rather than just the one other client and would add potential for multiple game modes.

Another idea we thought of was adding more places that the fish could hide and potentially that the shark could not enter at all. During development, we played around with a large sunken ship model that the fish could use to hide, but ultimately decided against using it because it would have required much more advanced, non-spherical colliders and an element of physics to make sure the characters couldn't pass through the obstacles.

Conclusion

Overall, we were very satisfied with our final result, especially given the time constraint and limited manpower of a two-person group. We were somewhat skeptical that we could get the online aspect of the project working since neither of us had experience with an application that updated in real-time, but were able to achieve a final product with an acceptable amount of latency even with the server running in a third-party host. Our visuals also came out better than expected since we were able to use relatively detailed models even with the large number of models in the level, and the ocean floor ended up looking fairly realistic.

Another takeaway from the project was realizing the importance of the actual gameplay versus just cramming as many features into the game as possible. We spent a lot of time debating various implementation options and many of them came down to trying to get the best user experience possible. For example, there was a lot of thought put into how to design the controls for the fish and the shark in a way that would make the game suitable and challenging for both players.

Works cited

Shark model:

This work is based on "Finn The Shark" (https://sketchfab.com/3d-models/finn-the-shark-c60933a54ffd4a21a5853592daf19d55) by XelaDoesArt (https://sketchfab.com/XelaDoesArt) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

Fish model:

This work is based on "Fish" (https://sketchfab.com/3d-models/fish-83036df81d05423dab94d4d12b3166e5) by ErickJohnP (https://sketchfab.com/ErickJohnP) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

Sounds:

“Success 1”, Leszek Szary. https://pixabay.com/sound-effects/success-1-6297/ license (https://pixabay.com/service/terms/#license)

“Monster bite”, Soundmast123. https://pixabay.com/sound-effects/monster-bite-44538/ license (https://pixabay.com/service/terms/#license)

“Game music”, Magntron. https://pixabay.com/sound-effects/gamemusic-6082/ license (https://pixabay.com/service/terms/#license)