CP 476 Final Project Report

Evgeniya Vashkevich 143178980
Nicholas Hallman 150357790

A live version of the game can be found here.

Project description

For this project, we created a Battleships game. Battleships is a two-player turn-based game, where players are trying to sink each other's ships. The player who successfully sinks all opponent's ships wins the game. The game has two steps: setting up the board and placing the ships, which players do simultaneously, and a game itself, where players are taking turns making a shot. There are different variations of the game rules. We chose to have 5 ships, which can be placed vertically or horizontally. They can touch each other but cannot intersect or lay on top of each other. The turn is over once a player clicks on a cell on the opponent's board, making a shot. Then, the mark is placed on both player's board, showing whether it was a hit or a miss, and the other player can shoot.

Website flow and logic

Please note that to test the game from the same machine, players must be using different browsers (ie Firefox for player 1 and Chrome for player 2)

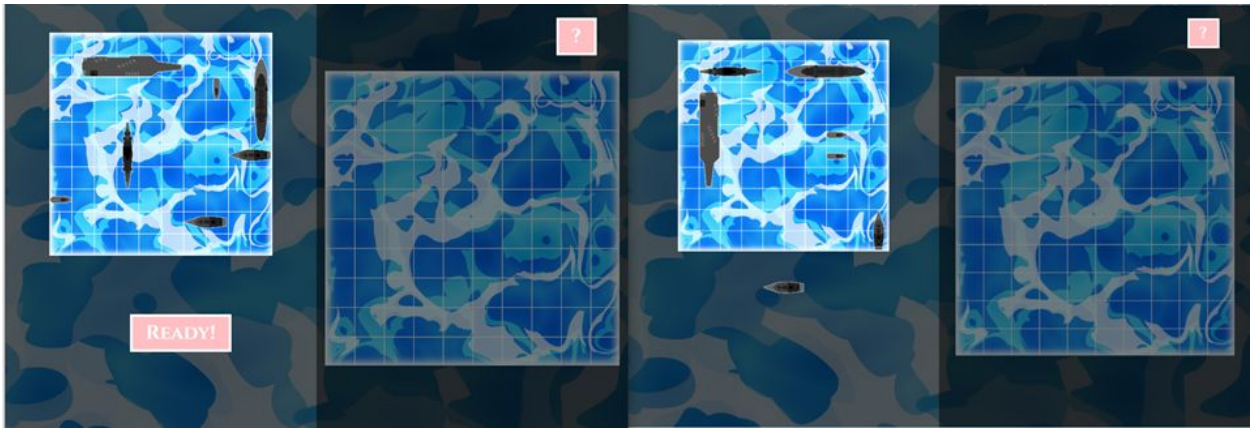
On the front-page, users can start the game by pressing the "Play" button. Users can also login or register to allow the game to keep track of their score, or to view the scoreboard by pressing the corresponding button on the top panel.

The login and registration forms are done as a sliding side-panel, where user can enter their credentials and client will pass this form to the server. Server passes this data to the database module, where validation is performed. Depending on the form, different results are returned. As a rule, we return the success flag and flags indicating the error. The JSON containing the result is then passed to the client through the server, where it is interpreted, and the message is returned notifying user of the success or containing the list of errors that should be corrected.

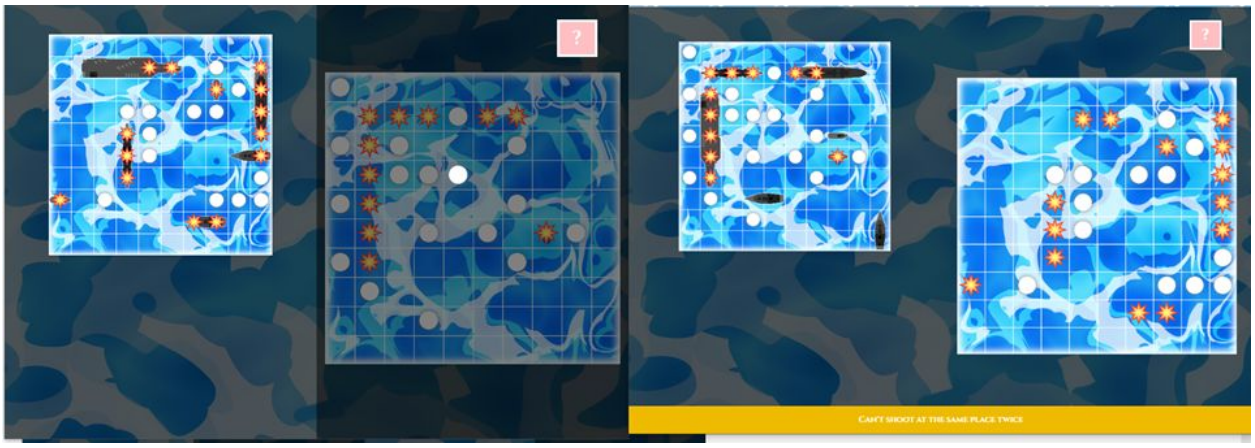
The Scoreboard page contains the list of top 10 registered users sorted by their score. It leads back to the main page.

When player presses the “Play” button on the main page, he or she is forwarded to the Matching page. There, they are waiting for the other player to join the “room” to start the game. Rooms are the standard way to group users provided by Socket.IO. Once there are two players in the room, they are forwarded to the game page.

There, they are presented with a board and ships. Each player needs to drag ships from the bottom of the page and place them on the board. If player tries to put them in illegal spot, ship is placed back to the bottom of the screen. To pick up the ship, user needs to press the left mouse button, and then let it go once it is on correct spot on the board. To rotate the ship, user needs to do the right click. The question mark button on top left corner of the page will display the rules. Once all ships were placed on the board, the “Ready!” button pops up, allowing the user to submit their board. After both players submit their board, the second part of the game begins.
The ship placement rules are enforced on the client side. Once all ships were placed in a way that no ship conflicts with the other ones, pressing the “ready” button sends ships location to the server. Before doing that, client converts ship location from pixel to grid coordinates.



Player1 starts the game. Server sends clients the result of previous turn as well as the sequence id. It is Player1's turn if the sequence id is odd and Player2's turn otherwise. When it is player's turn, the opponent's board becomes fully visible and player can click on it. Client checks that that location wasn't clicked on before. If it was already checked, user gets a notification message and his or her turn continues. Otherwise, client converts the shot coordinates from pixel to grid and send this data to the server. Server checks if the corresponding cell belongs to the ship and returns that information along with the sequence id of the next turn. Client reads the message it got from the server and place a mark on the board corresponding to the player, whose turn it was. White mark indicates a miss and fire mark indicates a hit.
Once one of the players sinks all the opponent's ships, players get the message saying whether they won or lost. After a couple seconds, they are redirected to the scoreboard.



Functionality

Site Functionality

- Log in
- Registration
- View Scores
- Join a match
- Hashed passwords

Game Functionality

- Online matchmaking
- Server side rule engine
- Turn based combat
- Ship collision detection
- Client side move verification

Technologies

For this project the technologies were split over three different areas: server, client, and the database.

Server

The server required that we be able to experiment and iterate quickly as this was the section we had the least experience in. As many companies now transition to a full JS stack this seemed like a great opportunity to learn. As such, we went with Node and it's package manager, even though we were unfamiliar with it, so we could experiment with the projects structure. Node is also light weight, cross platform and can be run without any additional software tools like XAMPP. Express was used to develop the views for our application and served as a routing system for our server. Socket IO allowed the clients to communicate with the game server and Morgan simplified the debugging process by providing us with more descriptive error messages.

Finally, Sessions was used so the server could remember a player‘s username, turn order, and id’s. In the end we chose this mix of technologies:

- Node, npm
- Express
- Socket IO
- Morgan
- Session

Front End

For this end of the stack we wanted to stick with technologies we were familiar with. As we both had experience with HTML 5, CSS and JQuery it made the most sense for this project. JQuery was used to return error messages for the user login and registration as it provides an easy way to update DOM depending on what was received from the server.

The game itself was implemented using WebGL and Pixi to simplify the game development process. Pixi provides a good rendering engine and allows the developer to quickly add and draw sprites, animations and sounds to make the game more appealing. It also has a great event library that can use the keyboard and mouse to make the game more interactive. This was especially useful for implementing drag and drop for initial ships placement and turn execution, where player needs to click at some point on the opponent’s board. In the end we used:

- HTML5, CSS3
- JQuery
- WebGL with Pixi.JS

Database

For our project we decided to use SQLite database to store our data. It is very portable and is easily embedded into NodeJS server, which made it a perfect choice for our project. Since we only need one table, we did not need enterprise-level functionalities and decided to go for a more simple option. SQLite is easy to maintain without losing in performance.

Extended Feature Set

As of today Project Coral Sea is a reasonably good minimal viable product. However, to expand it into a commercial grade piece of software we could add the following,

- Sunk ship prompt
- Competitive ladder
- Player profiles
- Save states

A sunk ship prompt would alert the shooting player when they sunk the opponents battleship. The competitive ladder would make playing the game balanced as players of similar skill would be matched against each other instead of random players of various skill. The ladder can also be used to show a player how skilled they are at the game relative to other players and is a good way to keep people playing as a driving motivator. Player profiles would allow user to express themselves with an icon and would create the potential for unlockable content to keep players engaged. Save states would allow users to rejoin a game and continue where they left off instead of having to restart if the server goes down or both players disconnect.

Challenges

Asynchronous Programming

This topic was initially a major challenge. The project required that two clients and a server take turns communicating and sending messages while pulling assets and data from the server. As such, a lot of planning was needed to make sure that the clients and server were ready when they received information and that messages were passed in the right order to ensure proper and effective communication.

One example of this is the room system that groups clients together. The server had to run games and create them at the same time. As each player joined the server it needed to keep track of how many players it had seen, how many rooms it had created and which room and players it was working with now. These messages were not always synchronous and so had to be prepared in a way the server could digest.

Another part where asynchronous programming created a challenge was passing information between the database and the client. The client does not communicate to the database directly, so all requests had to be sent to the server first. Since both requests to the database and communication between client and server are asynchronous, making sure that data is defined before it is sent for further processing required a lot of learning on our side.

This section was the most time consuming as async was integral to multiple sections of the project.

Software Team Collaboration

This project introduced another opportunity to work on a software project in a team environment. Of course, being students we have little experience in this field and we learned a lot from this project. To start, the structure of the project's initial requirements was detrimental to our project as a whole. The first draft required the front end of the project be completed with a style guide and web page layouts. Unfortunately, the vast majority of our project was the server implementation and this forced us to work in a way that de prioritized task splitting and instead forced us to work on complementary tasks. As such much of our code review was spent fighting git collisions.

Secondly, The team collaboration forced us to plan out and comment our code. Any given function may be recycled in the future so comments were necessary to make sure that any given team mate could reference the comment to know what kinds of data the function takes and what they can expect.

Thirdly, This project allowed us to develop better practices with version control software such as GIT. To simplify collisions and commits some members chose to use Visual Studio Code and the Git Desktop Client while others used the basic git tools like git for command line. Even with these different software solutions there were no problems using git. However, branching and committing to a single branch was a mistake frequently made that cause many problems.

Existing Analogies

Since Battleships is a classic game, there are many games of this format already available on the market. One of the examples will be http://en.battleship-game.org/, which is a single-page website, where you can play Battleships with some random opponent. One of the key differences between this version and our project is that it doesn't have a functionality to keep the score of each player. It also has different game rules in terms of the number of ships and their possible positioning on the board.

Overall, the game mechanics are very similar between different online game versions with some variations in the number and size of ships and game functionalities. Some games allow you to play against your friend or against AI, while the most common version is matching two random opponents. This is similar to how our player queue is implemented.

The other key difference is that our game has multiple pages, between which user is navigated. Most of the online Battleship game has only one page, where you hit the play button and game begins. We decided that using multiple pages will provide a bit different experience because it gives player a more intuitive understanding of the current and next steps. Additionally, we provided descriptive notification messages that alert player if they are trying to do some illegal actions or lost connection to the server.