Bryce Sadelski, Zach Etzkorn, Alec Yaagoub

April 25, 2024

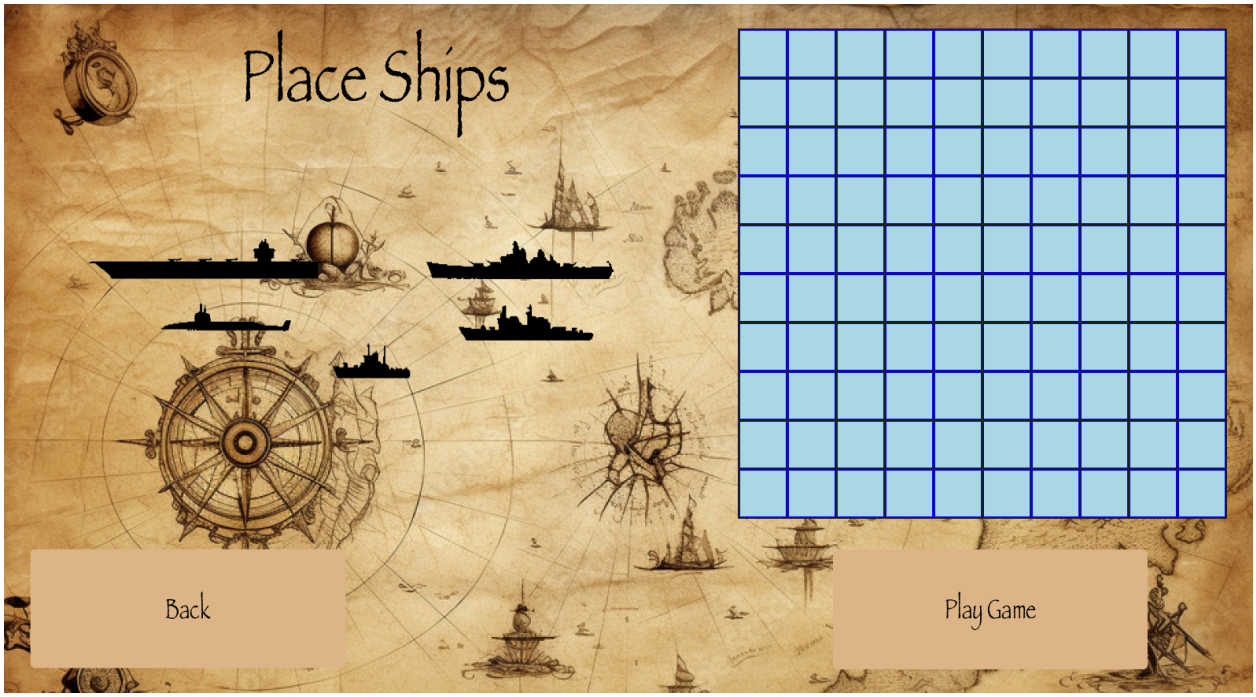
CS-342

Prof. McCarty

Battleship Report

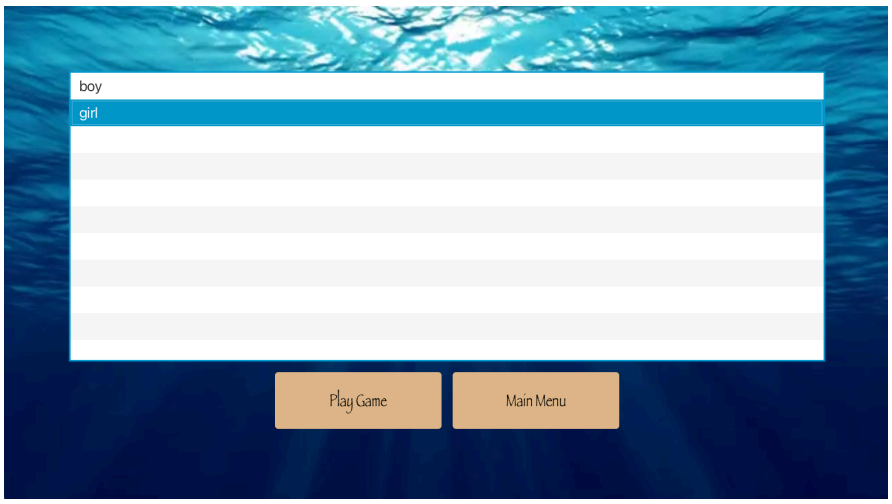
For this battleship project, we attempted to make a pirate themed game that allows the user to play the game with various features involved. We utilized the Media object to include music, namely “A Pirate’s Life” which is the theme song for the *Pirates of the Caribbean* attractions at Disney theme parks. Additionally, we used videos as the background for the player selection and game selection scenes. In order to do this, we had to update the modules we used to include media. We have never used videos/sounds before, so this was something we had to learn to go above and beyond. We based many design aspects on the pirate theme, including the papyrus font and color choices of buttons. We chose to customize our fonts/colors/sizes to give us the best looking product possible, while following our pirate theme. The backgrounds we used include pirate ships and pirate maps to adhere to our theme. Our game scene even has a pirate themed skull and crossbones when the game ends in a loss, and treasure if it is a win. To curate a more user-friendly approach to placing boats, we decided to implement a drag-and-drop feature for the user. This proved to be one of the group’s most challenging aspects, as we found it difficult for the boats to snap onto the cells it was placed on. Other classmates told us how they were able to place the ships, the group said they were using buttons to map the ships to coordinates. Though this approach seemed to make sense, we prevailed in our implementation of the drag-and-drop feature and it makes our game stand out in terms going above and beyond the requirements. The outside resource we referenced to make this was stack overflow. There were

some basic functions for mouse/drag handlers. These functions were good starts, but we needed to redesign them to fit our needs. We needed to include an additional class to handle the draggable images of the ships. Though stack overflow provided us with a good starting point, we found it very difficult to have the ships be dropped correctly. We ran into issues regarding ships snapping to overlapping cells, and also being dropped in the wrong place. Then, from there we had to take the positions that the boats were dropped on and add this to a char[][]. Each char has a different letter for different boats. Then, we used a method in our GameInterface class to display the ships and water at first. Additionally this handles updating the GridPane based on the character array for missiles and attempted shots. This took a great deal of debugging, but we all worked together to get this crucial feature of our game complete. Here is a screenshot of the scene where the ships are able to be dragged and dropped onto a GridPane.



We included a feature that allows the user to rotate the boats by pressing the “r” on their keyboard. Our scenes for choosing a user to play against or to continue a game that a user

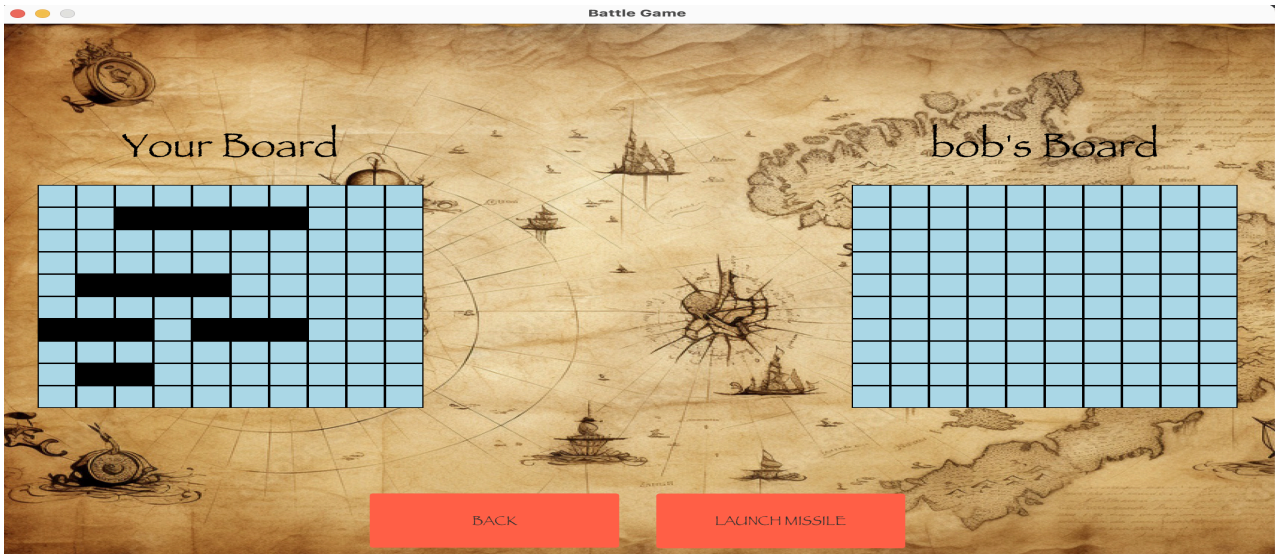
already started implemented a ListView object to allow the user to click on the game or user they would like to choose. The background is a video of waves moving.



For a portion of the server-client logic, we were able to repurpose code from project three that made connecting our networking seamless. Our server stores two arrayLists. One for clients, and one for games. When the client sends “REQUEST_USER_LIST” or “REQUEST_GAME_LIST” the server sends the requested information to be displayed in our GuiClient class in a ListView as discussed previously. The client list is also used to check if the user’s entered name is already taken or not. We found that debugging our code together was the best way to streamline our development process. Division of labor included each team member starting on different aspects of the code and then working together to put it all into a finished product. Zach initially created the client GUI and worked with scene transition and button handling. Alec worked on the game logic and a lot of the backend in the Game class. Bryce implemented the server-client connections and made the updates to deal with proper username handling and sending the arrayLists to the client. In terms of the drag-and-drop, we needed a lot of attention to resolve our issues, so we all worked together to get a working product. In order to make the pieces of the code we created individually work together, we all had to make the

connections between what goes where, how the classes work together, and how to update the GUI. After that, we needed to implement the actual gameplay. Our gameplay scene consists of two GridPanes, one for your board and one for your opponent. To appeal to user simplicity, we created a feature that allows the user to click on the opponent's board, and when clicked the cell turns green. From there, if you hit a boat it turns red and if you miss it turns white. The default color being blue for the water.

The code also includes the multiple game functionality. The server holds all of these games and is updated every time something changes. Even when two games are played between the same users, that game would have the same title. In our code we handle that by checking all the titles and adding a counter to differentiate between those same-named games. So, when the user wants to continue playing a game, the server sends back all the game titles for the listView. Then upon clicking a title, then clicking the button, the user will be sent to the current stage of the game they were playing. This was a tedious task to fulfill, taking a long time to perfect. This functionality definitely goes above and beyond the basic battleship game. Now a user knows every game with differentiability and they can return to any game at any time.



It’s fascinating that a user can log in with the server and then disconnect from the server, and come back and finish the game later. It’s very innovative that the player can play multiple games at the same time. In terms of how we used A.I resources, we used ChatGPT to help us format our code, and to attempt to do things we had no knowledge of doing. ChatGPT seems to be decently proficient in completing elementary coding tasks such as making classes, basic functions, and general syntax but apart from that it gives us a lot of garbage. We would try to troubleshoot using ChatGPT and it was sometimes effective, but most of the time just gave us repetitive answers or answers that were simply incorrect that it insisted on being correct. However, it can be useful in detecting errors and looking for ideas.

Overall, the functionality of our battleship is very user-friendly and having open access to outside resources allowed us to expand our knowledge in areas we may not have gotten the chance to learn about. We tried to implement features we were unfamiliar with, but in the end we learned that it is important to focus on understanding the code you are writing so you do not run into errors later involving code that you are unfamiliar with. It is always much easier to troubleshoot when you are familiar with the code you are working with.