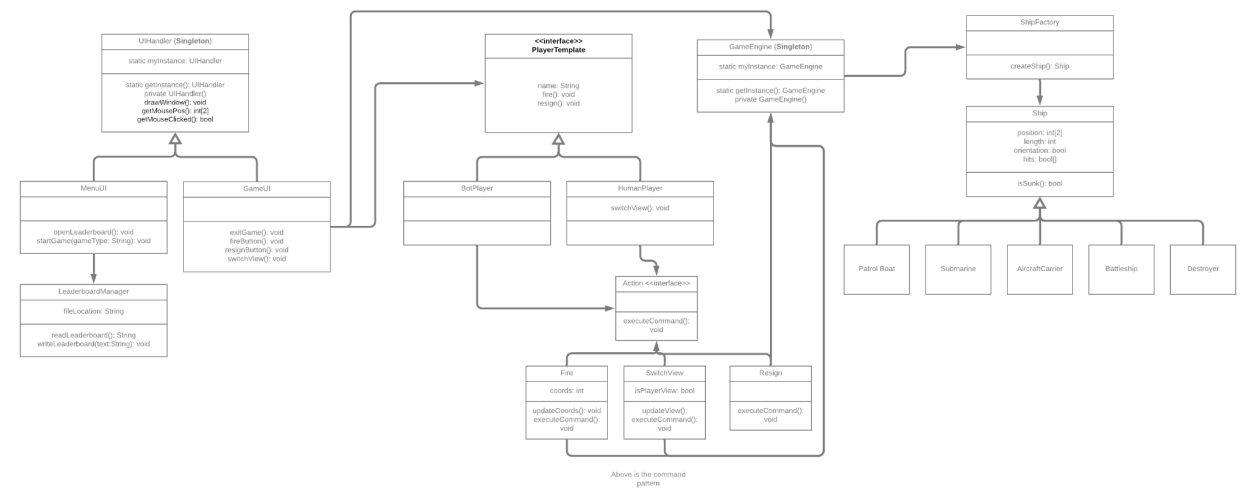
Final Project Report

Project Name: Battleship Game
Team Members: Noah Svensson and Jake Martin

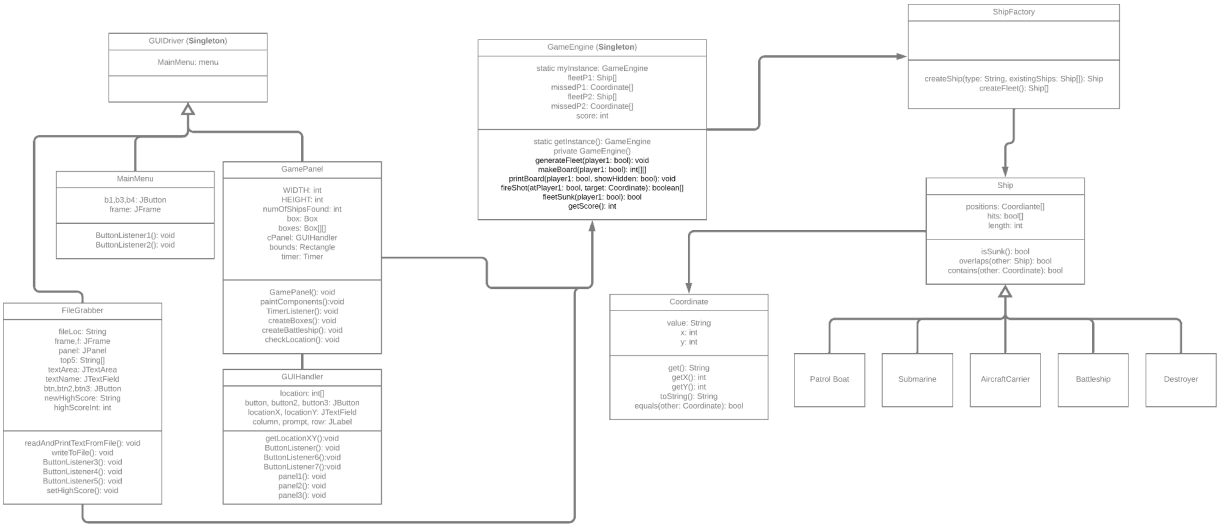
Final State of System: The final version of the project is working very well. The system works by opening the gui alongside the command prompt, but all interaction is done in the gui. The command prompt has certain messages for events/results from game play as they were easier to display there. The game works by submitting a coordinate to try and fire at in the gui, which is then updated in the game engine and then the gui (very short delay between updates) to play the game. We removed the functionality of having local multiplayer, sound effects, and bot difficulty as they became way too difficult to implement with the given time due to issues with other aspects of the game. The game functions as a fully functional battleship game where the player tries to sink the enemy fleet by firing at various coordinates until they win. The scoring system gives points for a hit and subtracts them for a miss. When the game is over and the player wins (which they either win or give up as there is no time limit or move limit) the leaderboard is presented with the ability for them to enter their name to add their new high score to the leaderboard. The leaderboard works by having the leaderboard stored in a text file, that is read from and displayed to show it along with being read into an array for sorting, and then the new name is inserted into the sorted array and the whole array is written to the file and the file is displayed again to show the updated version.

Final Class Diagram and Comparison:

Old UML:



New UML:



Description/Comparison:

One of the design patterns we used was Factory, which was to create the fleet of Ships for a player. We also used Singleton for the GameEngine and GUIDriver, to make it easy to use their functionality across the code base while always using the same instance. The Command pattern is implied in how the GUI calls on the GameEngine to process firing at a set of coordinates. While not explicit, the various button listeners are functionally similar/equivalent to the Observer pattern, in that they wait for an update from a button before then sending that update elsewhere.

Between versions, we removed the Template stuff for handling the player/bot as it turned out to be unnecessary. We added details and implementation changes for both the UI and GameEngine, to remove unneeded elements and add necessary ones. We also added the Coordinate object to help with processing coordinates in the GameEngine. The Command pattern usage for firing was also removed from an explicit object to an implicit reference in the UI for simplicity and usability.

Third-Party code vs. original code: In the gui, there was code borrowed from the following github: https://github.com/aawantika/battleship-gui that was used for the basis of the layout for the game board in the GUI. The code however has been heavily modified to fit our purposes and the only thing that really still comes from the original is the general layout of the game board, which has been increased in size to 10x10, along with the functionality of tiles to change based on whether it was a hit or miss. The Game Engine doesn't use any 3rd party code.

OOAD Process for project: Setting up the GUI portion had a lot of issues as things kept breaking in the process and trying to get everything to look decent was mostly trial and error. When we were working on the merge, getting the coordinates and whether there was hit or miss communicated between the engine and the gui caused many problems. We tried an observer but kept running into issues because of everything being across multiple classes with large amounts of separation. Lastly, there were challenges in converting all of the text-based game features implemented via the GUI merge, and as such some were unfortunately left out.