Software Design Specification Document

Version: (1.0 Final)
12/15/2106

1. Introduction

The idea behind this project is to design a program which implements the core functionality of the game "Battleship" with unique functions to make it more friendly towards modern consumer base. These new functions include but are not limited to Rewards, Leaderboards, Player to Player messaging, and give the players the ability to choose from an array of boats. In addition, this design document will present a broad final skeleton of the project and how we will be implementing and integrating our ideas.

2. Design Considerations

There are several key issues which will have to be addressed before we migrate our design into development stage. The foremost issue is to make sure our integration of different software languages are compatible in the sense that we can properly handle the data between the database as well as the program itself without corrupting the data.

We omitted our idea of using C++ with SQL to store, view, and extract our Database values. Instead we proceeded with using a simple text file which stores all the Usernames, Passwords, Wins, Loses, Experiences. We will then use the text file and read it on to a multi-dimensional array to display it on to the Leaderboard GUI Frame. Our only use for a Database is to store user information and we strongly believe that using a Text-File would be more efficient rather than extracting it from a Database.

2.1 Assumptions

Our original assumption was that we would use C++, Java, as well SQL for our primary programming languages and Database respectively but we decided to just use Java and not use SQL or any form of Database to store our player information. We went along with using Google Server Implementation to connect to the clients using Sockets. Once we connect the two clients, we simply have the server perform the Game Logic.

2.2 Constraints

Our system is only limited by its operating system and the CPU's level of functionality. Since we are building software rather than hardware, our program should be able to be run for a long period of time without any interrupts if our computer supports it. With that being said, only if a certain system does not understand the language we develop in, will there be some form of problem. Aside from that, the only real uncontrollable constraint will have to be the server itself. Since servers are being rented and not in our possession, we are at the mercy of our hosts, and if their server crashes, then so does most of our implementation since it is heavily dependent on our servers.

2.3 System Environment

Describe the hardware and software that the system must operate in and interact with.

Our system environment will initially be a Windows OS before attempting to migrate to OSX. Though we are not making an immense program, we need to assume that our CPU has enough memory as well as the high level of RAM required to run our design. The system must be able to convert our code into commands and follow the user's inputs. To make sure of this, we need to import proper packages and extend classes in our code itself. Essentially, we will attempt to make our code run on any downloading OS.

We used a vast array of packages and libraries to read the files, create GUI, create threads. The only problem we encounter between the migration between Windows OS and OSX is the formatting of the GUI. The functionality of the program works perfectly.

3. Architectural Design

The architecture provides the high-level design view of a system and provides a basis for more detailed design work. These subsections describe the top-level components of the system you are building and their relationships.

3.1 Overview

The face of the program is the GUI running on the Client side of the application. Here you will be sent to an initial screen where the user/player has the ability to either "Log In" or "Sign Up" where a Username and Password is created for new users. If the player already as a both these then he may continue to the Log In window where you will be prompted to type in your Username and Password.

This will then dispose the old window and recreate a new window which will have 5 buttons: StartGame, LeaderBoard, Chat, Options, Help. All of which are self-explanatory for the user with the Help button providing a more in depth explanation if the User desires.

When we press the StartGame button, new window will be displayed with three panels. The left most panel can either place yours ships or have them be placed randomly. The other two panels holds the grid for both your ships as well as your opponent's ships. Once you have placed your ships you will be prompted to enter coordinates to fire and a new panel will appear so you can chat with your opponent (minor bugs to be handled with the chat).

We would also like to point out the major difference between our initial Design Specification Documentation and the final one is our implementation of Rewards/Level-Ups is not fully functioning when integrated with the core GameEngine.

3.2 Rationale

The rationale behind our Overview is simply sequential. We wanted our game to proceed from one state to another without jumping too often between states. Thus once the user has passed a state we removed the option to go to previous state unless it is absolutely required. This rationale helped us conceptualize the code and develop it faster to get it onto production and testing.

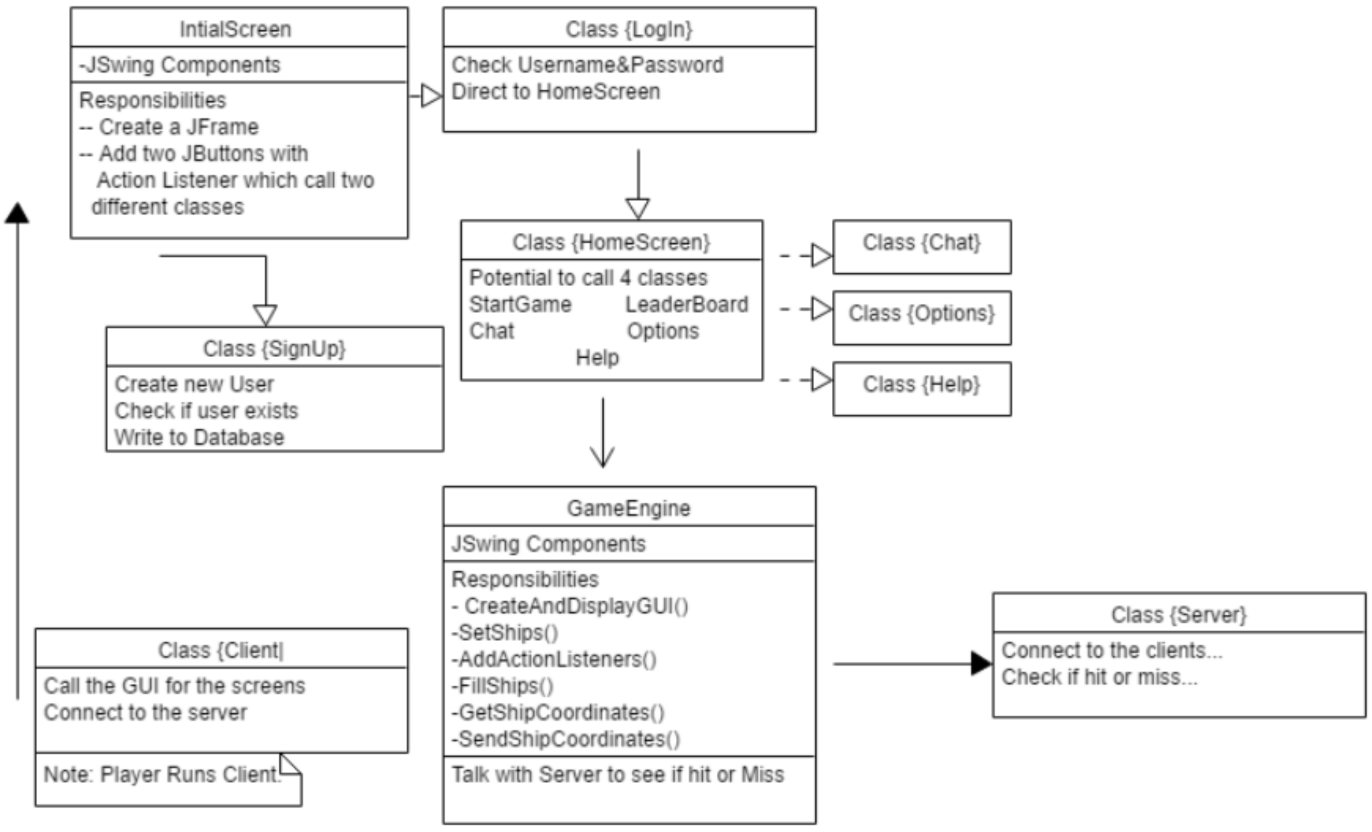
3.3 Conceptual (Logical View)

To get a better understanding of our code and to list all the classes and what each individual class will be performing. We have a created a table with class and their descriptions. In the table below, we also made sure to list the major methods and attributes of the classes.

No.	*Class*	*Responsibility*
1	*InitialScreen*	*Creates a GUI Window which as two buttons (LogIn or SignUP)*
2	*LogIn*	*Creates a GUI with Textfields for UserName and PassWords*
3	*HomeScreen*	*Creates a GUI to navigate (StartGame, LeaderBoard, Chat, Options, Help)*
4	*GameEngine*	*Creates a GUI for the board (Has the majority of GameLogic)*
5	*LeaderBoard*	*Creates a GUI based on the Database File. Updates the Database for Wins,Loses,Exp*
5	*Server/Client*	*These call upon the rest of the classes to start the game, and have players make moves*

No	*Action*
4	*Create Board/Set Ships/Get Ship Coordinates/Show Ship Locations on GUI/Hit or Miss*
5	*Read DataBase/Put Values in MultiD-Array/Check who won/Update Wins,Loss,Exp*

4. Low Level Design



4.2 Sequence Diagram

In our conceptual diagram you can see the states from different JWindows. Each Window calls upon another window depending on the action we are aiming for. It then disposes the old window.

5. User Interface Design

Initially we want to create a gui interface in which it will greet the user. The interface will include different options for the user to select initially from; from playing a game to leaderboards to a store-reward system, as well as options for adjustment, game help, and development information.

Play: Where the gameplay occurs. When clicked a player can choose his game against a person next to or online. During which we will try to implement a guy interface showing ships, hitmarkers, points, as well as possibly a pvp live chat system.

Leaderboards: User interface which will show leaderboards, players with the highest scores, as well as the highest averages. Compare one another to their friends.

Store: Interface which will feature a point and reward system. The player can upgrade ships, get difference bomb upgrades, and change personal colors.

Options: Options menu for turning off the music or sound effects.

Help: Help menu when clicked will show the fundamental rules of playing the game of battlehship, as well as explains upgrades and equipment

Information: Information lists copyright, developer info, resources, and other development information regarding the game.

We haven't changed our ideas behind User Interface Design too drastically, we simply omitted the Store and Information and added on the Chat feature.