CS 30700

Project Design Document

Admiral Radar

Utkarsh Agarwal
Ramsey Ali
Sam Buck
Panagiotis Kostouros
Delun Shi

2 February, 2018

Contents

1 Introduction **2**

1.1 Mission Statement 2

1.2 Project Description 2

2 Game Rules **3**

2.1 Original Rules 3

2.2 Adapted Rules 8

3 Design Outline **8**

3.1 Design Pattern 8

3.2 Structural Model 9

3.3 Component Interactions 9

4 Design Issues **10**

4.1 Functional Issues 10

4.2 Non-Functional Issues 11

5 Design Details **13**

5.1 Class Descriptions 14

5.2 Class Interactions 17

5.3 Sequential Diagrams 18

5.4 Program States 24

5.5 Database Design 24

5.6 Interface Design 25

1 Introduction

1.1 Mission Statement

Board games are a lost art, and part of a treasured history. By adapting the popular board game *Captain Sonar*, we attempt to digitize the game and open it to a wider audience through online play, more intuitive gameplay, and additional functionalities not possible on a physical board. *Admiral Radar: Space Expedition* allows players to enjoy the gameplay mechanics originally found in *Captain Sonar* but without the need for players to be in the same geographic location. Some game features are simply not even possible in physical board games such as team-only communication, persistent player statistics, or single player gameplay, demonstrating the need for board games such as *Captain Sonar* to be brought into the digital space.

1.2 Project Description

Admiral Radar: Space Expedition allows players to enjoy the gameplay mechanics originally found in *Captain Sonar* but without the need for players to be in the same geographic location. *Captain Sonar* along with many other board games don't have as much of a chance to shine in the spotlight due its eclipsing in popularity from video games. A game like this has huge entertainment potential if properly ported to a more complex, convenient and popular platform, specifically on the computer. The enhanced complexity of the computer platform can allow for this game's shortcomings to be eliminated - and that is what this project aims to do.

There has been significant prior work in this area in the form of efforts to computerize various board games. The most familiar example of such work may be the video and computer game versions of the board game Monopoly. The experience provided by such app is generally high-quality, however is limited to that specific game. More complicated, less universally known games are less frequently converted into applications. We intend to develop our game on a more user friendly GUI and implement all the benefits associated with digitization.

Captain Sonar is a board game released in 2016 that provides an engaging mechanic for team building and cooperative play. In the game, players form teams of up to four and assume various roles onboard a submarine. The captain steers the submarine, the radio officer determines the location of an enemy submarine, the first mate ensures that sensors and weapons will be ready to be used, and the engineer fixes broken systems. Each movement of the submarine is accompanied by actions from each team member, which together comprise one game turn. Turns for each team may occur at different rates - in other words, each team's turn is not dependent on the others'. Such a game structure, called a "real time" board game, is uniquely fun and engaging while an exceptional promoter for teamwork. Thus, our project will use the mechanics and rules from *Captain Sonar* while using a space-based backstory and visual theme.

2 Game Rules

2.1 Original Rules

The game of *Admiral Radar* is based on the rules of the board game *Captain Sonar*, which are described here. The game is normally played by two teams of four people, each of which represent the crew on a submarine during wartime. The overall objective of the game is to sink the opposing submarine by removing all four of its health points. Each member of the crew has a unique role: Captain, First Officer, Radio Officer, and Engineer. Each of these roles performs different functions, all of which are necessary for the submarine to successfully navigate, find, and damage the opposing team's submarine. In the physical game, each role has a different board design in order to accomplish these tasks. The teams are separated by a large physical barrier to prevent one team from seeing the other team's boards.

The game is played in turns, with each turn consisting of a series of moves by the Captain, First Officer, and Engineer. Most significantly, the turns of the two submarines are not necessary coordinated. If the game is played in "turn-based" mode, one submarine completes its turn and then the other submarine does the same, trading back and forth until the end of the game. If the game is played in "real time" mode, however, one submarine may begin their next turn as soon as it completes the previous one: the turns of the teams are not synchronized. In both cases, the actions of the Radio Operator are linked to the actions performed by both submarines, although their play will inform the choices made by their team's Captain during their turn.

The Captain's game board has a dot-grid diagram of the game map. This grid has certain points blocked off, representing islands. At the beginning of the game, the Captains will select a starting point for their submarine and, using an erasable marker, note it on their board. Every turn is begun by the Captain commanding a direction: North, South, East, or West, for the submarine to travel. They must vocally announce this for all players to hear. After the First and Engineering Officers have completed their tasks, the Captain will mark the direction taken on their board by drawing a line to the dot in the direction on the map which they selected, indicating that the submarine has moved to this new position on the map. They may not direct the submarine in any direction that would contact an island or the boundaries of the map. Additionally, they may not move the submarine in such a way that it would cross its previous path.

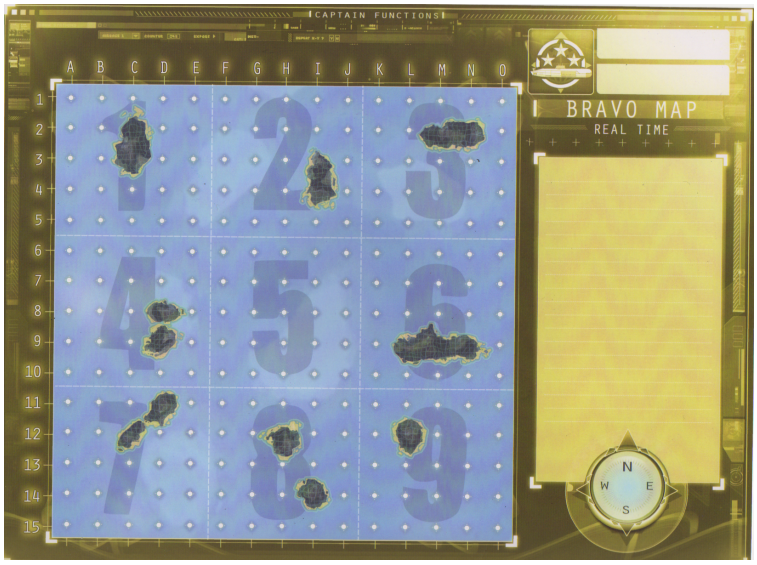

Figure 1: Game Map

After the Captain has ordered a direction, the First and Engineering Officers complete their responsibilities. The board of the First Officer has a series of six gauges representing the ship's six special systems: Torpedoes, Mines, Drones, Sonar, Silence, and Scenario. Every turn, they distribute one unit of power to a system of their choice, and, once complete, announce "done" to the Captain. If they have completely filled a gauge, they must also vocally announce that the system corresponding to that gauge is ready.

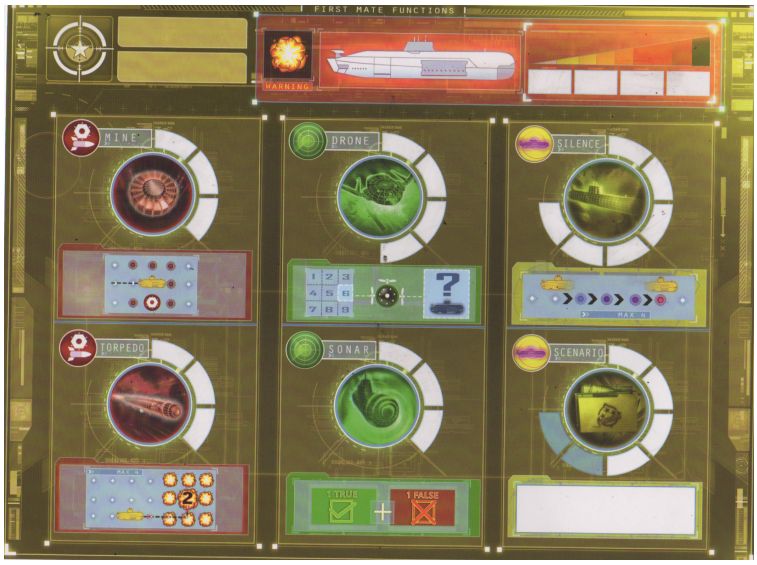

Figure 2: First Officer's Game Board

At the same time, the Engineering Officer is performing their own task. With every move, they must select a certain ship system to disable for maintenance. Which components can be

disabled are dependent on the direction that the Captain has announced. Depending on the Engineer's choice, a component of the sensory, tactical, propulsion, or axillary systems will be disabled, making those systems unavailable for the Captain's use. If, however, the Engineer disables all members of a specific set (called a circuit) of components, those components will be repaired, and the systems they belong to will again be available for the Captain's use. Some components cannot be repaired in this way, as they are not part of one of the circuits on the Engineer's board. Of special significance are the reactor components. While the Engineer choosing to take them offline does not diable a submarine system, if all reactor components are disabled, the submarine looses one point of health and the Engineer re-enables all systems. Because the reactor components are not part of any circuit, they cannot be repaired except by the Surfacing mechanic (discussed later). After marking a component as disabled, or, if a circuit is complete, marking the component enabled again, the Engineer, like the First Officer, announces "done." Once both the Engineer and First Officer have announced that they are done, the turn is complete.

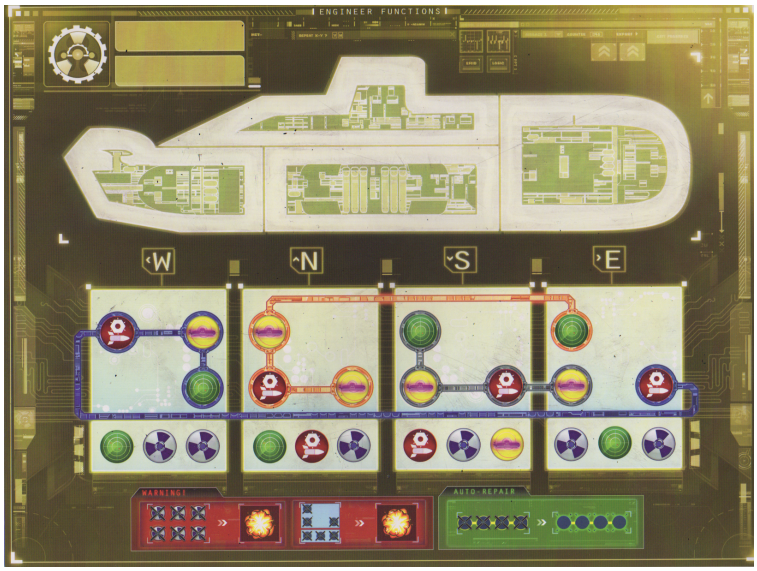

Figure 3: Engineering Officer's Game Board

The final role on the submarine is that of the Radio Officer, whose job is to determine the location of the enemy submarine. Like the Captain, their board has a dot-grid map of the play area; however their job is to draw the path of the opposing submarine, which they learn by listening to the opposing Captain's directional commands. While this means that they know the path of the enemy submarine, they do not know the starting location, which they must determine from available clues. As the game goes on and the submarine's path gets longer, fewer and fewer coordinates become viable guesses for the actual starting location of the enemy submarine. The Radio Officer draws the path of their opponent's vessel on a transparent plastic sheet, which they may move overtop the map to determine its present location. When the Radio Officer has a good guess of where the enemy is, they notify their Captain, who can use that information to attack their opponent.

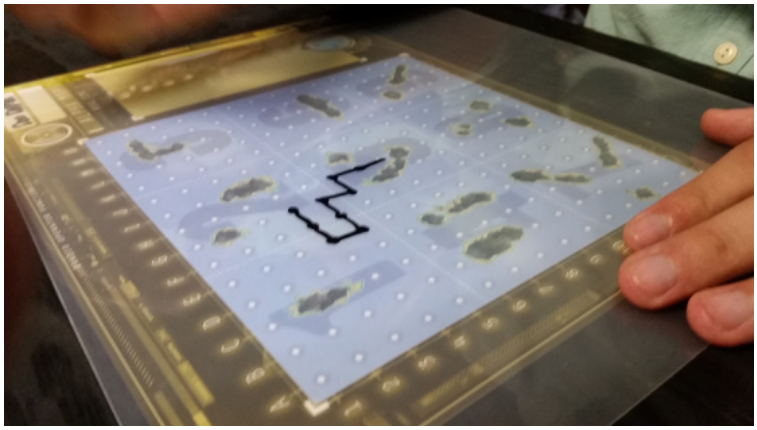

Figure 4: Radio Officer's Station

In addition to movement, the Captain has, at their disposal, seven special actions which they may order the submarine to perform. The first six actions must be charged by the First Officer before being available for the Captain's use, while the seventh, "surfacing," may be ordered at any time. Firing a torpedo is one of the two tactical actions of the game. When the Captain fires a Torpedo, they select a grid location up to four cardinal steps from the location of their submarine and report it to the opposing Captain. The opposing Captain compares the announced location to the location of their submarine: if the announced coordinate is the same as the one which the submarine currently inhabits, the submarine looses two health points. If the selected coordinate is adjacent to the submarine in any direction, it looses one health point. Otherwise, the enemy submarine looses no health. The opposing Captain will announce how much health their submarine has lost from the Torpedo. The other tactical system, Mines, work slightly differently. At any point, the Captain announces that they are laying a Mine, and indicates to a Radio Officer a coordinate one unit away from the current location of their submarine in any direction, which the Radio Officer records on their map (not their overlay). At any point after, the Captain may then remotely detonate that Mine, which will cause damage in the same pattern as if a Torpedo was targeted at the location of the Mine.

Figure 5: Torpedo Firing Diagram

The next two systems are the sensory systems: Drones and Sonar. Both of these provide information about the location of the enemy submarine to the Radio Officer. When the

Drone is used, the Captain selects one of the major grid squares (called sectors) on the map to test for the presence of the enemy submarine, announcing the number of that sector to the opposing Captain. The opposing Captain then answers, truthfully, if their submarine is located in that sector or not. The Radio Officer then records this information on their map, as it allows them to eliminate a number of potential locations at which the enemy submarine could possibly be located. The Sonar action provides a similar general function, although the exact information which the enemy must provide is different. If a ship uses its Sonar, its opponent must announce two pieces of information about their location from three possible choices: the row in which they are located in, the column they occupy, or the number of their current sector. Although two pieces of information are provided, only one of these is truthful: the other is deliberately false. Like when executing the Drone action, the Radio Officer will mark down the information they have gained about the enemy's current position on their board.

The fifth action which the submarine may perform is to Silence itself. If the Captain orders the submarine to Silence, it instantly jumps up to four coordinates away, in a straight line. No information, besides that the Silence action was performed, is reported to the opposing team. Like a standard movement, a Silenced movement may not intersect with an island, map edge, or the submarine's previous path. Unlike the other four discussed actions, performing a Silence does not pause the game (which is done when a Torpedo, Mine, Drone, or Sonar action is used) until it has been completed: both submarines continue taking their turns. The Scenario action, the final one available to the submarine, does nothing in the basic version of the game: it can perform special actions depending on specialized rules for alternate game variants, which are beyond the scope of this description.

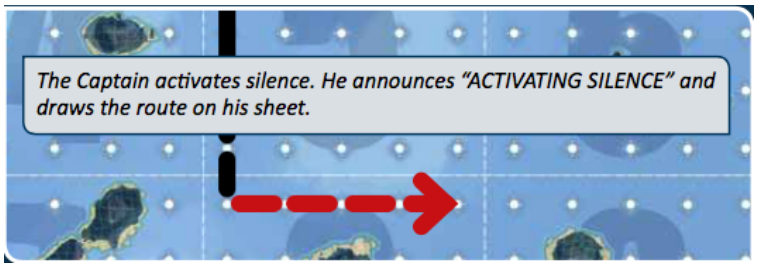

Figure 6: Silenced Movement

Finally, at any time the Captain may order the submarine to Surface. They must also inform the enemy team which sector the submarine occupies. Once a Surface is performed, the Engineer re-enables all submarine systems which have been disabled. The Captain may clear the previous route from their board, allowing the submarine more room to maneuver, as it does not have to avoid the path it followed prior to surfacing. If the game is being played in turn-based mode, the enemy submarine gets three turns. If the game is being played in real-time mode, the enemy continues to perform actions while the surfaced submarine performs a task designed to waste time (tracing the outline of certain shapes on the Engineer's player board). During the time the submarine is surfaced, its Radio Operator may not record the enemy submarine's movements: they must remember and record the movements once the submarine has submerged again. At this time gameplay resumes as normal.

As soon as one submarine has lost four units of health, either from reactor damage or enemy weapons, the submarine is sunk and looses the game. The surviving submarine crew is declared the winning team.

2.2 Adapted Rules

While *Admiral Radar* will have nearly the same gameplay as *Captain Sonar*, two important sets of changes must be made in order to fully make the rules we've described applicable. First, we'll be retheming the game around spacecraft in orbit, rather than submarines. On account of this change, certain mechanisms have been renamed:

- Torpedoes are now Missiles
- Sonar is now Radar
- Silencing is now Boosting
- Surfacing is now going on a Spacewalk
- Islands are now Asteroids
- The atomic reactors are now solar panels.

In addition, Admiral Radar will add an additional gameplay element: more than two Submarines. To support this, the Radio Officer will keep track of multiple tracks, and the Sonar and Drone mechanisms will report information from all enemy vessels simultaniously.

3 Design Outline

3.1 Design Pattern

We decided to implement a client-server model for our project. Since our project is a game that allows many users to play together and requires coordination between team members, we thought that this design pattern was appropriate. Each instance of the server handles a single game of Admiral Radar and connects to multiple clients, one for each user. Also, when a player logs in to their account the server will access a database for authentication.

1. Desktop Client
   (a) Each player will have their own client, with changes in GUI based on player role.
   (b) Clients send player actions to, and receive information about the states of the game, from the Server.
   (c) Clients update GUI based on game data received from the server.
2. Game Server
   (a) Performs the majority of calculations required for the game.

(b) Accepts requests for information and commands from the clients, and performs the appropriate action.

(c) Queries Login Database for user authentication info.

(d) Stores all information related to game state. (Position and Health of ships, Location of Mines etc.)

3. Login Database

(a) Contain User Data for each account. Data includes Username, Password, Avatar, Past Game Statistics.

(b) Provide authentication for user connections.

3.2 Structural Model

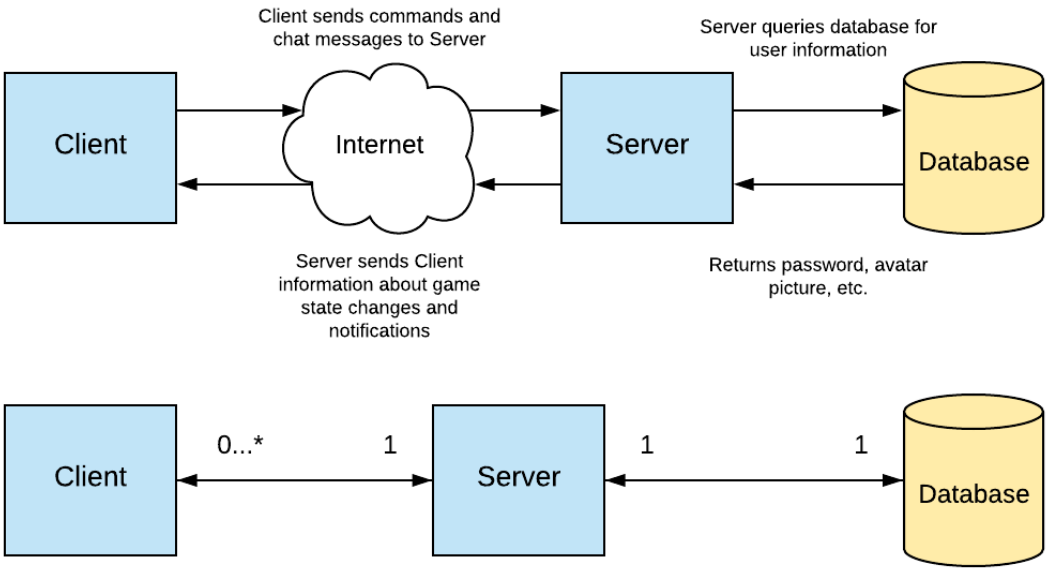

Figure 7: High Level Overview of Components

3.3 Component Interactions

The user client forms a many-to-one relationship to the game server. In our project, the server handles most of the calculations and stores the data for the actual game. The clients are relatively thin, handling graphical displays of game data, playing audio, and sending commands, such as moving a ship or firing a torpedo, and chat messages issued by the player to the server. The server executes the commands it receives and then updates the game variables, sending updated info to the appropriate clients based on game rules. It also relays chat messages to appropriate targets (e.g. team members). In the case of actions that

require multiple team member inputs, such as issuing a move command, the server will wait for all appropriate commands from every necessary client before executing the command. Also, the client will wait for the server to notify them that other team members have finished before allowing further actions.

4 Design Issues

4.1 Functional Issues

Issue: ***How will users communicate with the other users on their team?***

- **They will communicate through a simple IRC based chat room.**
- They will communicate through voice chat.

The main reason for choosing this option is that it'll be simpler to implement as well as be less stressful on the server. However, these options aren't mutually exclusive (in fact both would be optimal as is common in most modern video games), so we may choose to implement the voice chat option at a later time in a game update.

Issue: ***How will we convert a Java Swing based desktop application into a web based application?***

- **We will use an API called SwingWeb.**
- We will use Mia Transformer.

We've collectively had more experience with SwingWeb and have had no issues in using it whatsoever. Taking the time to learn an alternative method would be most unwise as our software would be prone to more bugs.

Issue: ***How will multiple games run simultaneously on the server?***

- Each game will run on a separate pool of threads.
- **Each game will run on a separate thread.**

There wouldn't be enough functionality to warrant running multiple processes for a single server request as the game logic is fairly simple. However, we may consider other options in the future should we decide to greatly improve game's performance and efficiency and/or if we decide to implement new features that would potentially make the game unstable.

Issue: ***How will realtime gameplay be handled?***

- **Orders will be processed by the time they are received by the server.**
- The client will enqueue each order into a type of "OrdersBuffer" queue that would then be sent to the server for processing at every given arbitrary quantum.

The buffered option, while easier to implement, would most likely be more temperamental should the server lag. A great many orders could be lost and would be most irritating to the player.

Issue: ***How will the AI interact with the team members?***

- They will interact through console commands via the chat window.
- **They will interact through a Swing based GUI.**

A GUI will allow players to easily keep track of each submodule and the game as a whole, as well as be more aesthetically pleasing. Using a text based AI wouldn't be very engaging nor easy to monitor.

Issue: ***How will the actions required of the different players be presented graphically?***

- Through game/server messages presented at the side of the screen.
- **Through a visual design consistent with game theme and intuitive usability standards.**

The bold option is not only logical but also engaging and entertaining. Having a flashy visual and audio representation of a player's actions is far more pleasing than having meaningless text pop up on the screen as it follows the old adage, "Show don't tell".

4.2 Non-Functional Issues

Issue: ***What kind of database will we use?***

- MongoDB
- **SQL**

The game is structured in such a way where the stored data would be easily interconnected through relational entities, making SQL (or any SQL option) the simplest to implement.

Issue: ***How will a team play with fewer than four members?***

- An AI would take control of the slots lacking players.
- **Allow a user to occupy multiple spots.**

This is a very simple fix and potentially the most fun for players. The challenge of having fewer people control a ship creates real tension and greater reward should this team pull off a victory against the odds. Of course, we will look into developing an AI at some point in the future which would allow the players to decide either option as they wish.

Issue: ***How will users be delayed in gameplay during a spacewalking action? (equivalent to a "surfacing" in original game)***

- **One at a time, each user will have to trace a shape using their mouse in order to proceed.**
- The users would have to each move at the same time with the arrow keys.

Having to go one at a time brings back a bit of that board game feel where players share the same physical space and creates tension with the entire team watching one player move across the board.

Issue: ***How will the clients handle a server disconnection?***

- **Pause the game for up to around 30 seconds and, if a connection isn't restored, return the user to the lobby.**
- Once disconnection occurs, push any actions along with time stamps held by the clients on a temporary queue to then be sent to the server once connection is restored and the server will sort each action in the order of the time performed and execute it in that order.

Pausing the game and having players wait will be the simplest to implement given the time restraints, and even we were able to implement the other option competently, players might find a way to leverage server timing to get certain actions performed before others, so it also acts as a preventative cheating measure.

Issue: ***How will the server handle a database disconnection?***

- **Send a message to the player letting them know what happened, then allow the game to complete and cache results until connection is re-established. No further login would be allowed during that time.**
- Have the player on a loading screen at startup until connection is reestablished.

Having the player know what is wrong with the internals of the game as it happens will dispel any confusion with the issue. The player won't have to troubleshoot the error themselves.

Issue: ***On what hardware will the server be implemented?***

- It will be implemented on a personal Linux desktop.
- **It will be implemented on Linux servers operated by team members.**

Linux servers are the most versatile and are widely used throughout the world because of that fact. We are also most experienced with implementing Linux servers so it would save a lot of development time.

5 Design Details

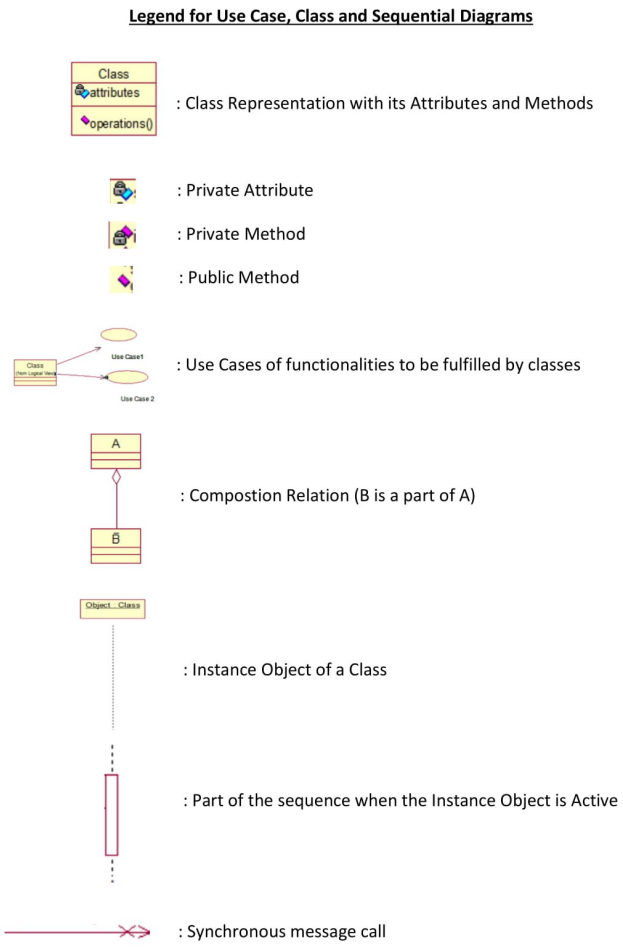

Figure 8: Diagram Legend

5.1 Class Descriptions

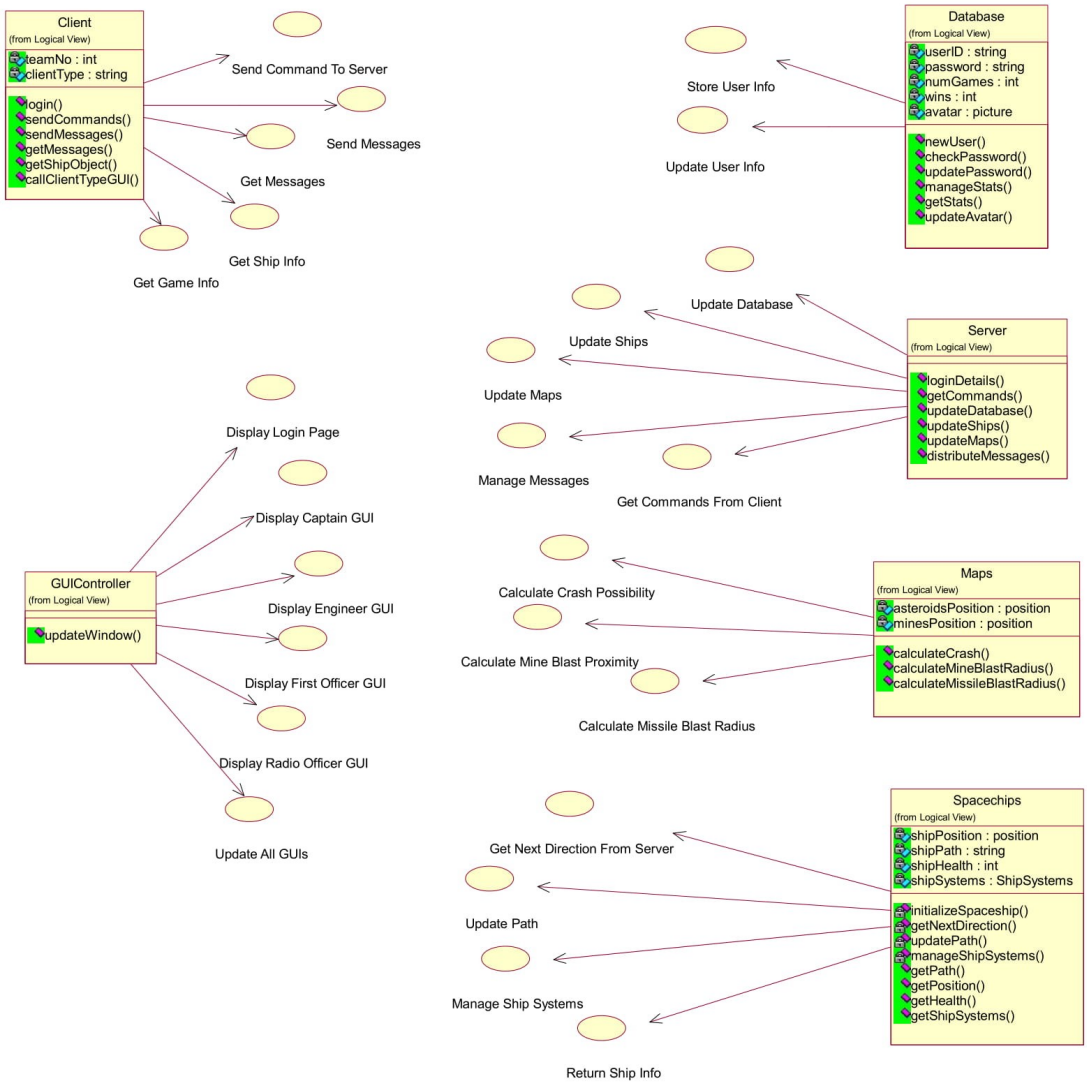

Figure 9: Use Case Diagram

¡¡¡¡¡¡¡ HEAD In Figure 9, the Use Case Diagram, we show what roles each actor is going to play for the successful functioning of *Admiral Radar : Space Expedition*. It is a behavior diagram which depicts a set of actions, services and functions that the system needs to perform. ======= In Figure 9, the Use Case Diagram, we show what roles each actor is going to play for the successful functioning of *Admiral Radar : Space Expedition*. It is a behavior diagram which depicts a set of actions, services and functions that the system needs to perform. ¿¿¿¿¿¿¿ 5f065fc57074bf09d1fdf8142e30c988eacae9c8

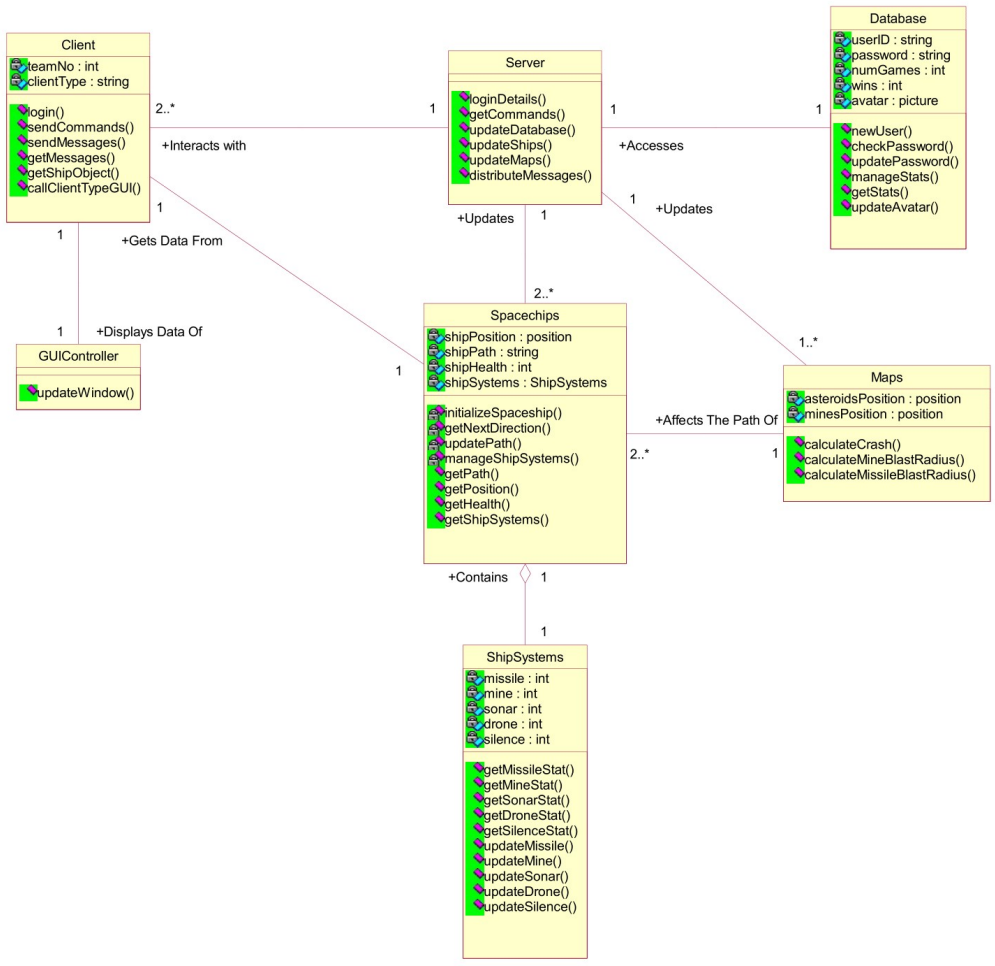

Figure 10: Class Diagram

Figure 10 depicts a representation of how our classes will be structured when we implement our design. This is a mock design and can undergo changes based on requirements or limitations encountered in the process of our implementation. The major classes included in our code would resemble the following descriptions:

1. Client

    - Each player in a game is imagined to be a separate client and interacts with the server independently.
    - Contains data teamNo (which team the player belongs to) and clientType (what role does the player play in the team).

- Can send commands to the server for a change in the position or system state of the spaceship.
- Can send messages to the server to send it to his other team members.
- Can get messages of fellow team members from the server.
- Is accessed by the player through the GUI.

2. GUIController

   - Is the visual representation for the Client class.
   - Consists of four sub-classes: Captain, Engineer, First Officer, Radio Officer.
   - Can display Login window on the start of the game.
   - Can ask to start game.
   - Can ask to select role.
   - Can display recursively updated interface for the selected role.

3. Server

   - Main controller where all the actions would take place.
   - Can get commands from the clients.
   - Can update current Maps object based on the command.
   - Can update current Spaceships object based on the command.
   - Can get messages from a player and send it to all his teammates.
   - Can access and update database of user accounts.

4. Spaceships

   - Represents unique spaceships of each team.
   - Contains data shipPosition (where is the ship on the map), shipPath (string of letters based on the path the ship has taken), shipHealth (what is the health of the ship, i.e. from 1 to 4), shipSystems (object of class ShipSystems).
   - Can initialize object when the game is started.
   - Can get the next direction from the player based on the command.
   - Can update the path of the ship based on recent movement.
   - Can return all the data of the ship to the client.

5. Maps

   - Represents all the types of maps the players can play on.
   - Contains data asteroidPosition (array of positions where all the asteroids are present) and minesPosition (where are the mines been placed).
   - Can calculate crash based on the movement.

- Can calculate all the positions mine blast would affect.
- Can calculate all the positions missile blast would affect.

6. ShipSystems
    - Represents all the systems in a spaceship.
    - Contains data missile (missile system), mine (mine system), sonar (sonar system), drone (drone system), silence (silence system).
    - Can return the status of each system.
    - Can update the status of each system.
7. Database
    - Used to store all the user account data.
    - Contains data userId (which player is playing), password (password to login), numGames (no of games played by the user), wins (no. of games won by the user) and avatar (picture chosen by user).
    - Can initialize a new user.
    - Can check the password entered for login.
    - Can update password after login.
    - Can see statistics of player.
    - Can update new avatar from a list of choices.

5.2 Class Interactions

From the above Class Diagram, we can say that the classes are related to each other in the following ways:

- Client is related to GUIController in a one to one cardinality. This means that each client will have only one GUI representing it and it would be based on the clientType. GUI displays the data present in client in a graphical manner.
- Client is related to Server in a 2..* to 1 cardinality. This means that there is only one sever but the server can have clients from 2 to n. Client interacts with Server by sending commands, messages, getting ship object etc.
- Client is related to Spaceships in a one to one cardinality. This means that each client can have only one ship as its object. The client gets the ship object from Server and can only get the ship statistics but cannot manipulate the ship data directly.
- Server is related to Spaceships in a 1 to 2..* cardinality. This means that there is only one server but the server can have spaceships from to 2 to n based on the number of teams playing. Server can manipulate all the data in the Spaceships class and sends the Spaceship object back to the client after every update.

- Spaceships is related to ShipSystems in a one to one cardinality. This means that each spaceship will have only one set of systems. Spaceships can manipulate all the data in ShipSystems based on commands received.
- Server is related to Maps in a 1 to 2..* cardinality. This means that there can be multiple maps that the user can choose to play on and based on this choice the map will be initialized. Server can manipulate the data in Maps class and send the updated map to players.
- Server is related to Database in a one to one cardinality. This means that there is only one server and one database which we will be dealing with in our game. Server can access and update the data in Database based on the user's choice.

5.3 Sequential Diagrams

Login

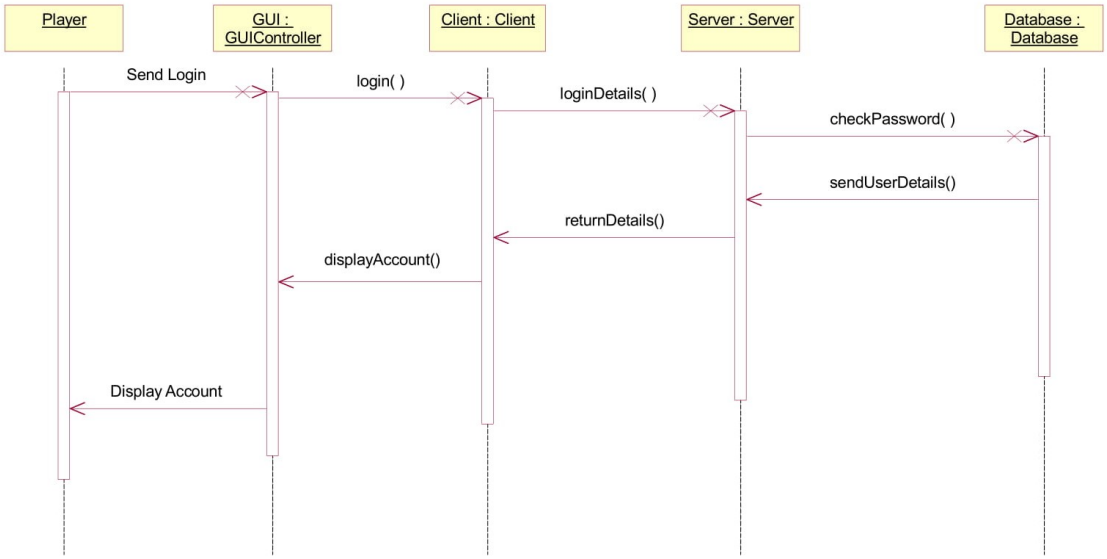

Figure 11: Sequence Diagram for Login

When a player sends his login details like username and password through GUIController, the details are sent to the Client and then to the Server. The Sever sends this data to the Database to check the password. Once the password is confirmed, the Database sends user details like the username, avatar, statistics etc. back to the user in the form of a GUI.

Captain

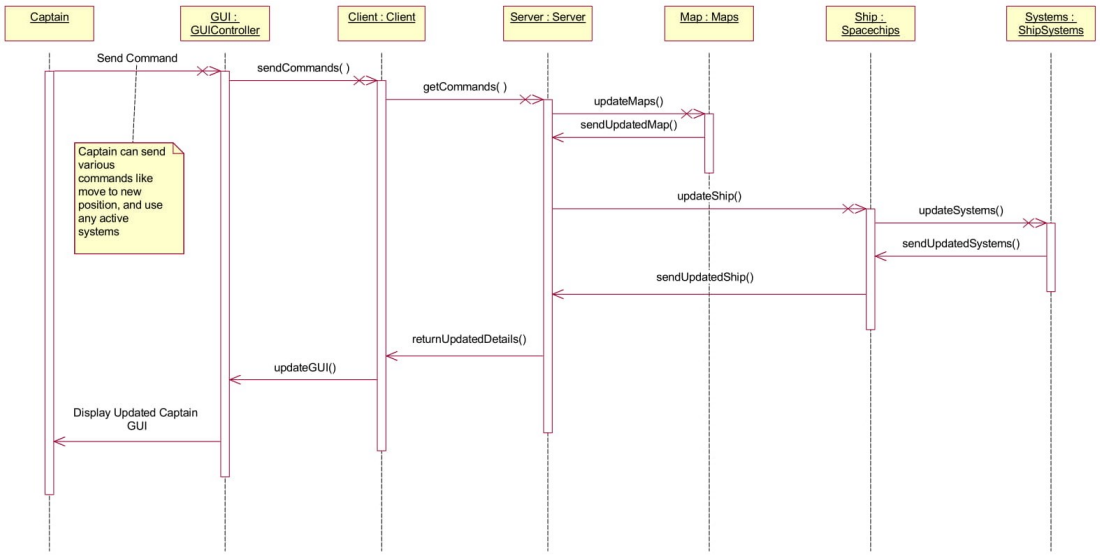

Figure 12: Sequence Diagram for Captain

The Captain is tasked with navigating the spaceship by telling which direction to move. He also has other tasks like commanding when to use the spaceship systems which have been activated. These commands all update the objects of maps and spaceships based on the commands. In case the Captain commands to place a mine, the map stores details of the mine. If the Captain shoots a missile, the Server should calculate the blast effect. After these updates, the updated GUI is displayed to the captain.

First Officer

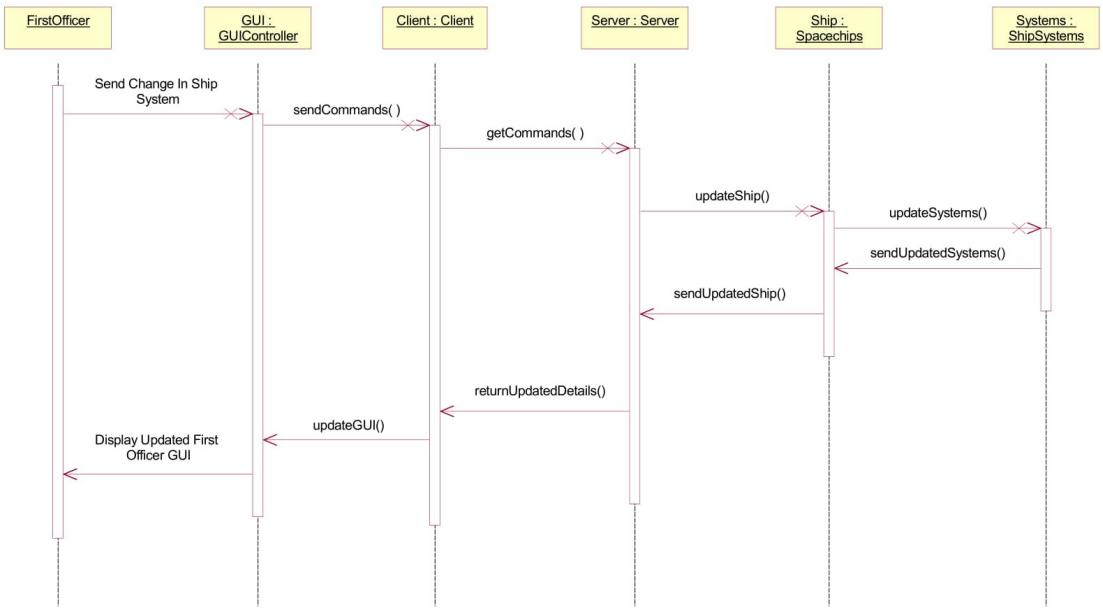

Figure 13: Sequence Diagram for First Officer

First Officer charges the systems with each move. As the systems of the ship are charged, they are activated and the object of ShipSystems is updated. This makes the system available to use by the team. The updated GUI is sent to the First Officer.

Engineer

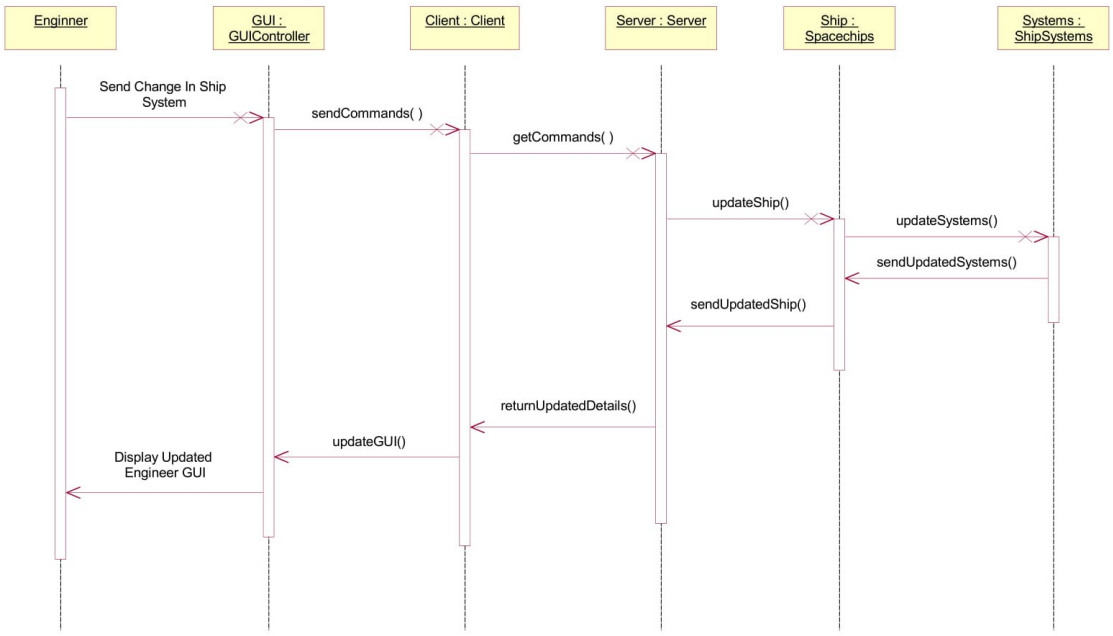

Figure 14: Sequence Diagram for Engineer

Engineer has the responsibility to disable a system of the ship with each move based on the direction moved. As the systems of the ship are disabled, they are deactivated and the object of ShipSystems is updated. This makes the system unavailable to use by the team. The updated GUI is sent to the Engineer.

Radio Officer

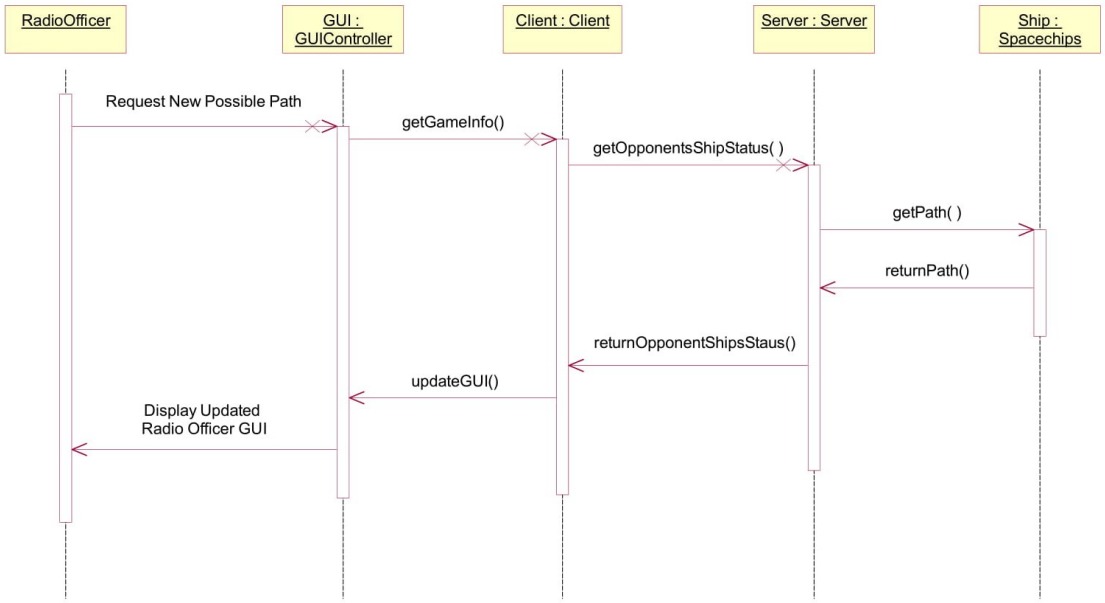

Figure 15: Sequence Diagram for Radio Officer

Radio Officer plays a very important role which involves interaction between the GUI, client and server. When the Radio Officer requests a new possible path the GUI is supposed to plot a path of the opponent ship by getting it through the server. Radio Officer is supposed to keep the team members updated on the possible positions of the opponents.

Messages

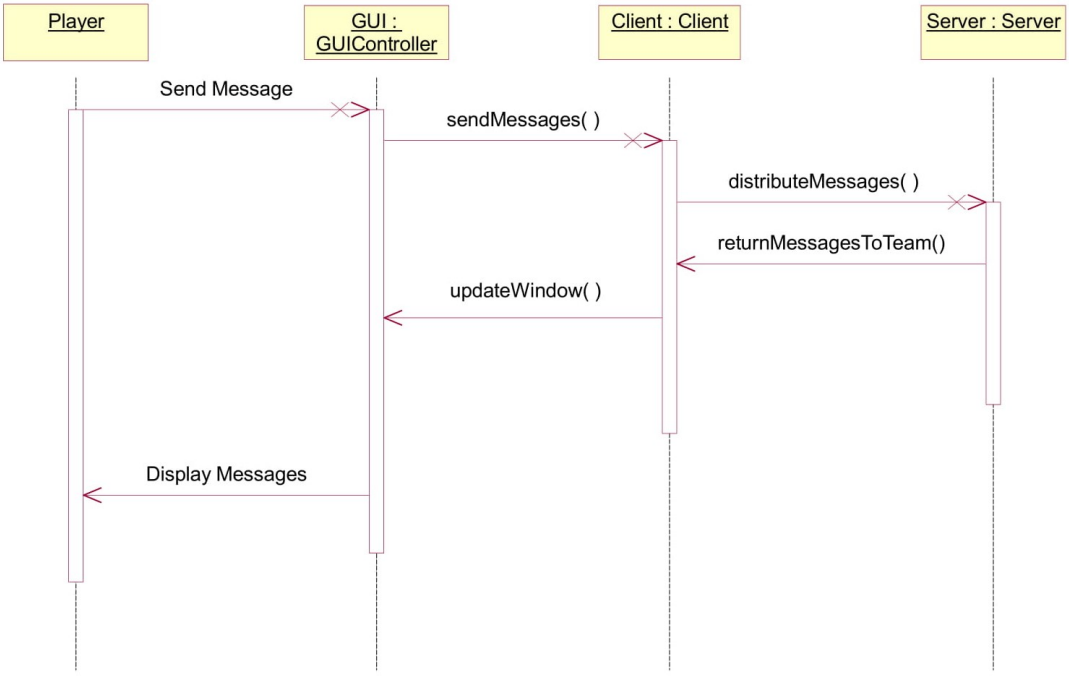

Figure 16: Sequence Diagram for Messages

Each player can message his/her fellow team mates at any time in the game. This messaging system is highly essential for the team members to play the game efficiently. The message is sent to the server through client and it is distributed to the rest of the team mates. This is a simple implementation of IRC server

5.4 Program States

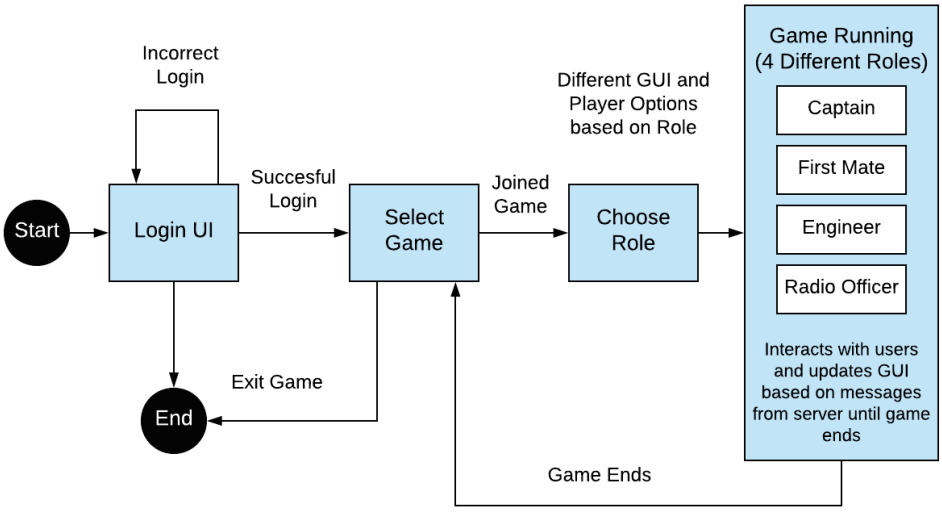

Figure 17: State Diagram

5.5 Database Design

The server connects to a database with the following structure.

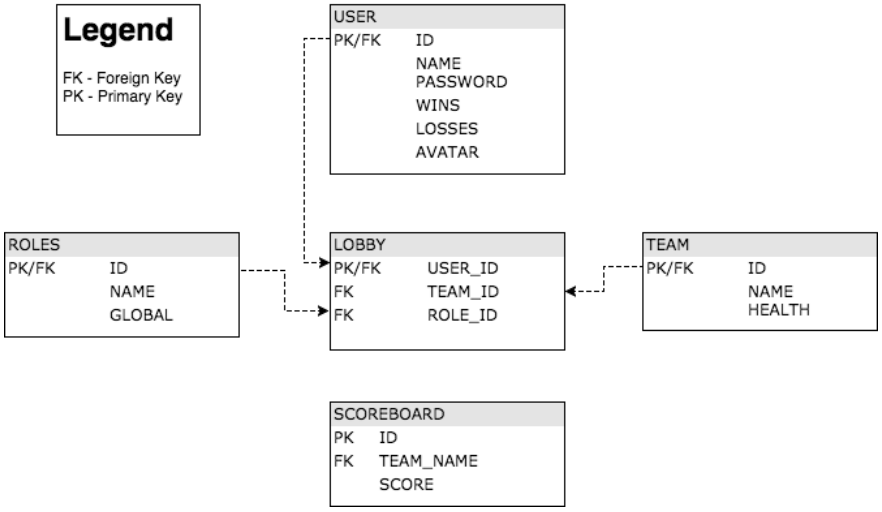

Figure 18: Database Entity-Relationship Diagram

5.6 Interface Design

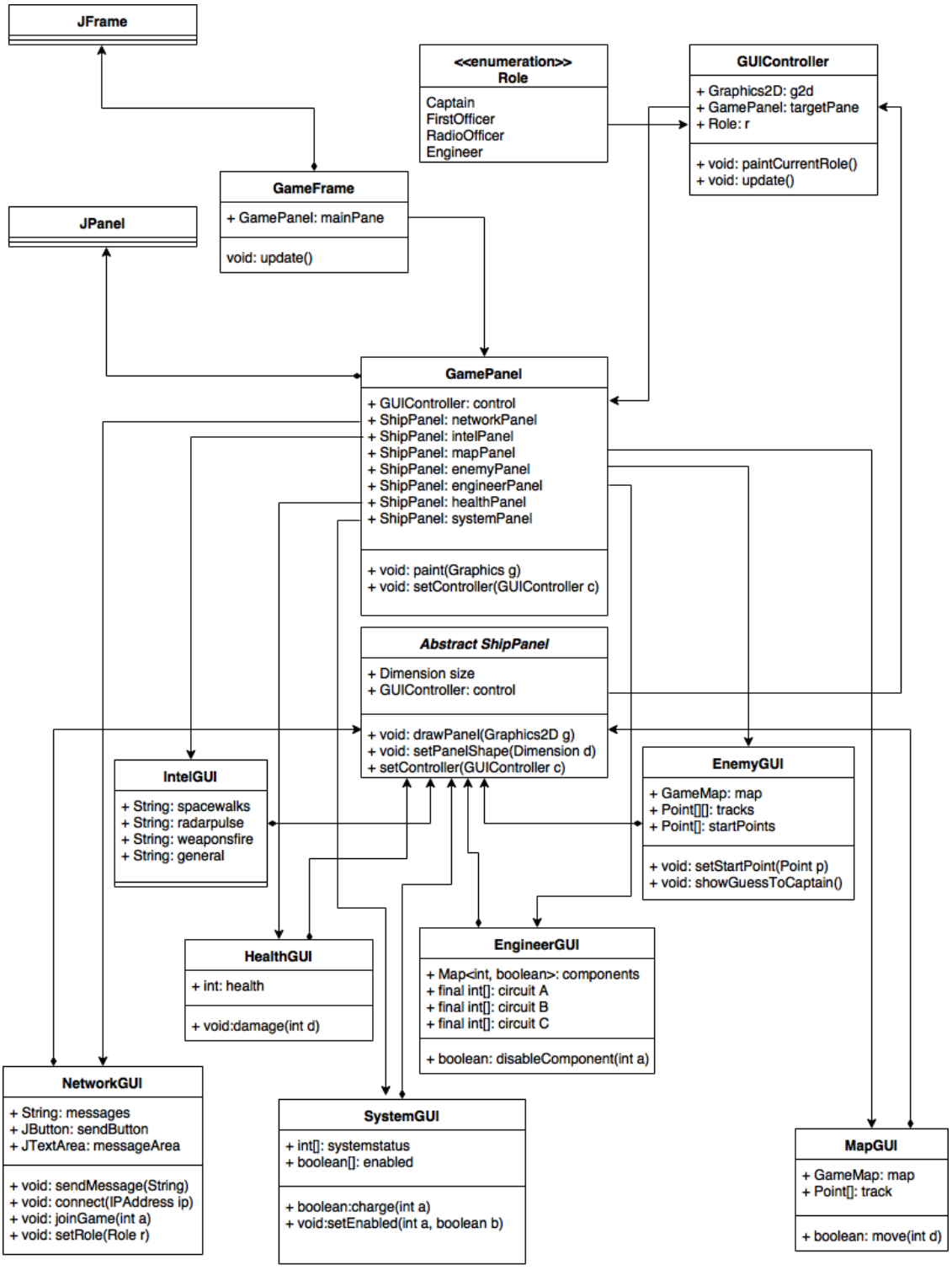

Figure 19: GUI Model

Admiral Radar's desktop client software will provide the user with a GUI for interactive gameplay. The GUI will connect to the backend classes of the client software using a Model-View-Controller design paradigm, allowing the graphical view and interface to be developed in a modular fashion. The unique challenge in developing the interface will be that there are multiple separate interfaces which must be included just during the game: one each for each game role. Additionally, a separate layout must be developed for the program in order to allow the user to connect to the server and begin a game. Finally, several special interfaces are required for two infrequent-yet-important game situations: responding to a radar pulse and going on a spacewalk.

The interface is planned to be implemented in Java's Swing library, as shown in figure 19. The reason for this choice was primarily due to its familiarity to the members of the team and its simple polymorphic design. The application will display the user one central window capable of taking on alternate arrangements of a core set of display windows. The five primary arrangements, which we refer to as interface modes, correspond to the Captain, First Officer, Engineer, and Radio Officer roles, in addition to a connection settings screen. Contained within these windows should be, in different configurations, sub-panels for the map with the vessel path, map with the speculated enemy path, game announcements, dials representing system status, submarine health, network connection and chat, engineering component status. Each of these panels can be displayed in multiple sizes and in multiple locations on the screen, depending on the interface mode required.

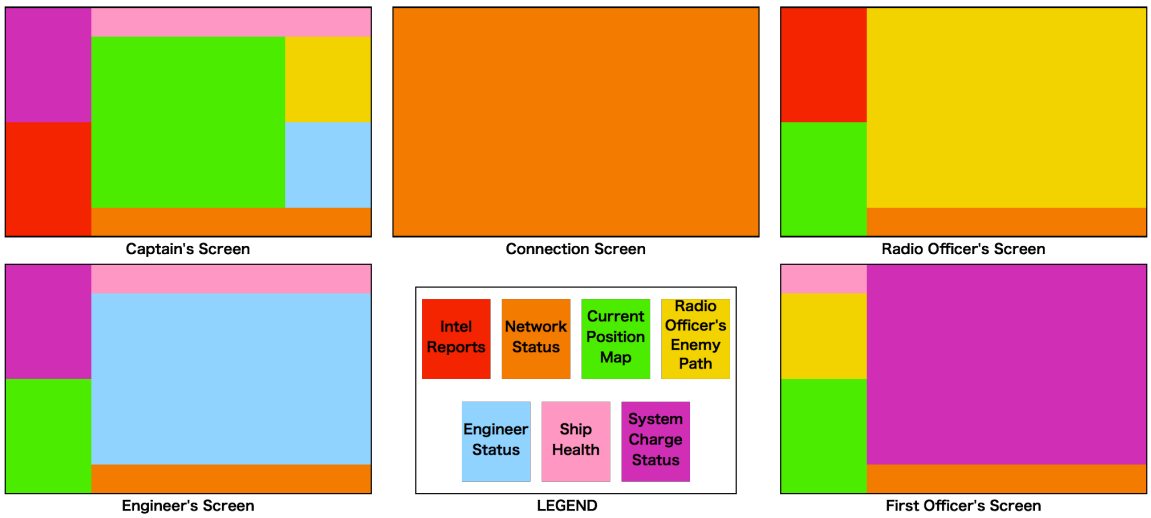

Figure 20: Sample Component Panels

Additional overlay windows will provide the functionality required for Radar and Spacewalk interactions. The radar response window will appear overtop the rest of the GUI of the Captain's client interface when triggered. Other functions of the Captain are disabled, as are the functions of the other members of their team.

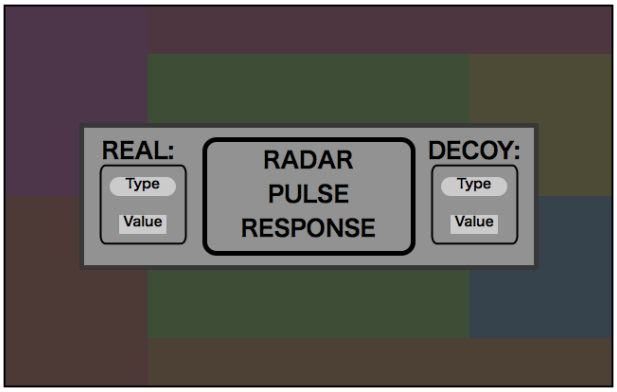

Figure 21: Radar Response Overlay

When the Captain initiates a spacewalk, each player's standard activities are paused. In a random order, each player must then trace a shape displayed in a pop-up window with their mouse and then click the "next" button, which enables the tracing interface for the next player. After four successful traces around the shapes, the Spacewalk concludes, and the players resume their mission with systems repaired.

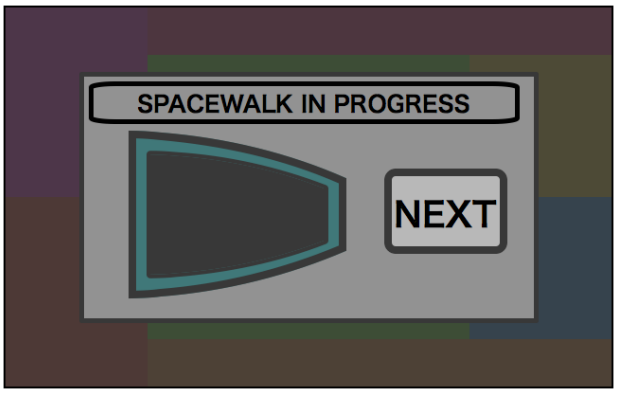

Figure 22: Spacewalk Activity