DOCUMENTATION FOR SEMESTER PROJECT:

BATTLESHIP GAME

GROUP: LIGHTNING TORPEDOS

GROUP MEMBERS:

1. ASADUJAMAN NUR
2. MUHAMMAD AMJAD BIN ABDUL MALIK
3. JOHNSON KWABENA

**Table of Contents**

1. Introduction

*1.1 Purpose of Document*

*1.2 Overview of Project*

*1.3 Requirements of Project*

2. Overall Architecture

3. Major Design Decisions

* 1. *Employed Tools and Technology*
  2. *User Interfaces*

4. Important Implementation Details

*4.1 Artificial Intelligence Technology*

5. Workloads and Protocols

6. References

INTRODUCTION

In this semester, we were appointed to gather up a group of 3 students for our project, which is to create a Battleship game in our Programming and Software Engineering course.

The purpose of this project is to explore the basics in the C language, including its data structures and graphical components. The purpose is not to define and develop new and original software or academic knowledge; hence a game of Battleship serves as a case study, for exploring the basics of software design and programming in C Programming. The choice of programming a game of Battleship, allows one to gain experience with both the concepts of the C programming language and the details and methods of data structures. Secondly, it also allows for experimenting with user interface design, without the risk of getting bogged down in user interface details, compared to designing more elaborate software systems.

Next, the purpose of this project is to have the students to design of a fairly complicated project using concepts we have learned and then implement the solution using some of the coding knowledge that we gained from the lectures, along with writing some codes in the previous exercise classes and then implement all those knowledges that we have acquired into our semester project which is creating a Battleship game. We are also required to complete the whole picture of all the processes on how we created the game and how it works which will be explained more details in this documentation.

*1.1 Purpose of Document*

This document applies to the Battleship game that we are working on during this semester. This document describes all of the steps and the processes that we have done during a month of developing and creating our semester project, which is the Battleship Game that is implemented by using C Programming language.

The reader is assumed to have basic knowledge of C Programming language. Knowledge and understanding of UML diagrams is also required.

*1.2 Overview of Project*

Since we were assigned to develop our own Battleship game, here are the brief overview of the Battleship game to make the reader understand more on what this game is actually about and get an idea on how it should be played.

Battleship is a guessing game for two players. It is played on four grids. Two grids (one for each player) are used to mark each players' fleets of ships (including battleships). The locations of the fleet (these first two grids) are concealed from the other player so that they do not know the locations of the opponent’s ships. Players alternate turns by ‘firing torpedoes’ at the other player's ships. The objective of the game is to destroy the opposing player's entire fleet. In our game, ‘firing a torpedo’ will be allowing the player to take a guess at where on the grid their opponent may have placed a ship. In the requirements, we will set forth other simplifying rules to limit the scope of this project.

*1.3 Requirements of Project*

We need to apply whatever that we have learned through out this whole semester in both subjects, which are Software Engineering and Programming (C Programming Language) to realize all of the requirements that will be stated below. We also need to develop everything regarding this project by our own using all the methods and techniques taught in the lectures. Here we include the requirements for this Battleship Game project:

* It must be a Two-person game
* Playing field consist of squares or map of 10 x 10
* The horizontal boxes are identified with numbers 1-10.
* The vertical boxes are marked with letters a – j.

Ships Placement:

* Total ships are 10 ships
* 4 ships over 2 boxes, 3 ships over 3 boxes, 2 ships over 4 boxes and 1 ship over 5 boxes
* The ships can be placed either horizontally or vertically
* The ships must remain in a line and not around a corner (diagonally)
* At least one box must remain between the ships that are being placed

Rules:

* The game should be played in turns
* If a player has hit a piece of a ship, he or she may guess again. As soon as all pieces of a ship have been found, one says "sunk".
* The player names a field or simply enter a coordinate on the game board to shoot or guess. The fields result from the crossing points of the letters and numbers, e.g. G5.
* The game continues until one of the two players has found or "sunk" all of his or her opponent's ships.

Players:

* One player game (playing against the computer with at least two difficulty levels)
* Two player game (multiplayer mode)

OVERALL ARCHITECTURE

In this section, UML diagrams will be used to make the explanations clearer on certain of the essential parts.

Since the game can be played either as single player or as multiplayer, the program interacts with the players and computer. However, that depends on what mode is selected. If player select “Single player mode”, then the game will be played by the player and computer, if the player Select “Multiplayer mode” then the game will interact among two players instead of one player and computer.

Player Game Computer

Figure 1: Game Environment

The game is played on four grids, two for each player. The grids are typically square of 10×10 and the individual squares in the grid are identified by letter and number. On one grid the player arranges ships and records the shots by the opponent. On the other grid, the player records their own shots. The game basically has four main phases or parts, which are the Introduction part, Preparation part, Shooting part and lastly the End game part.

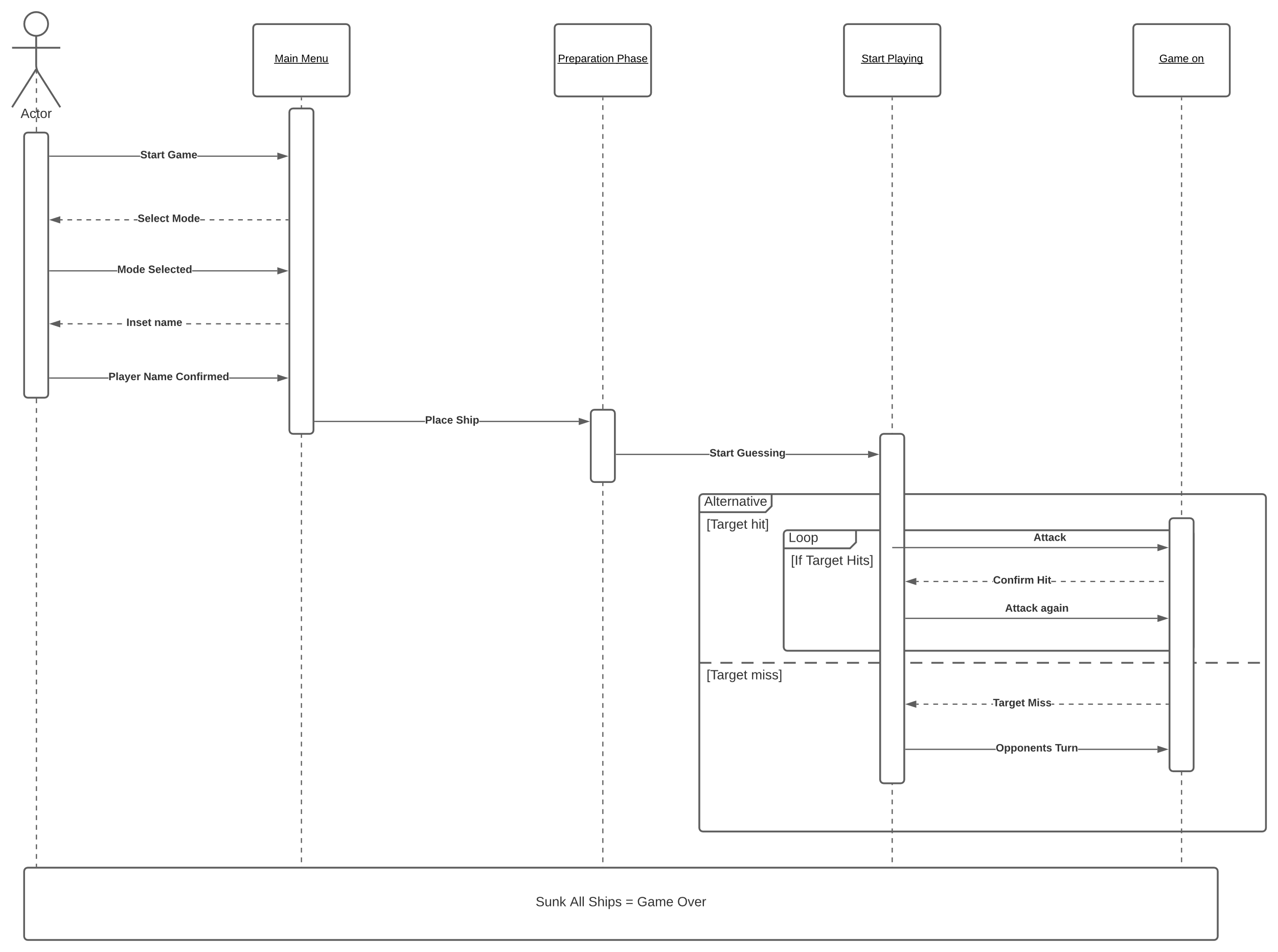


Figure 2: Sequence Diagram of Battleship Game

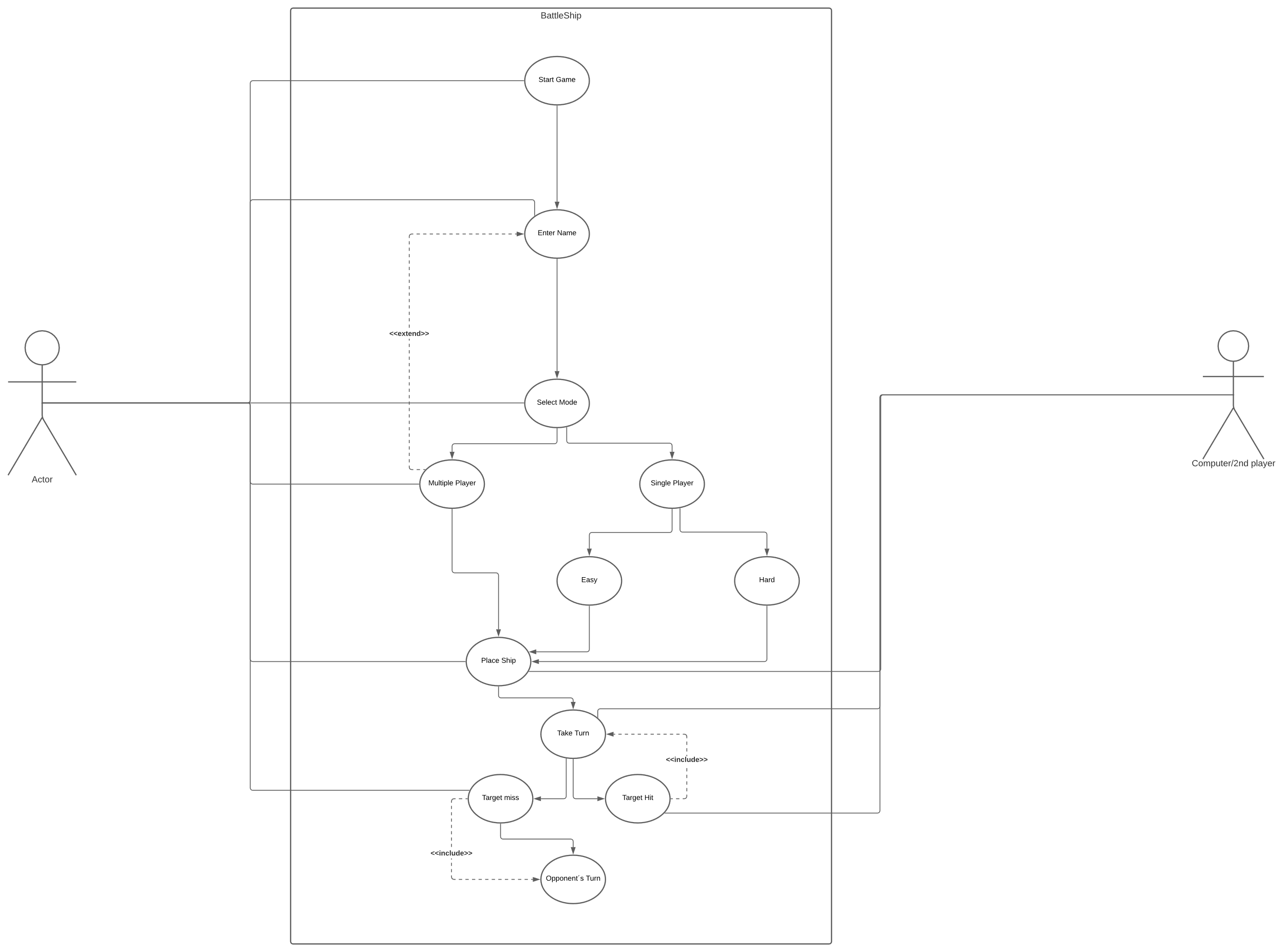


Figure 3: Use Case Diagram of Battleship Game

As the program is being open by the user, the user will be greeted by a welcome interface and the player can choose either to directly jump to ‘Start Game’ or to read and learn the instructions of the game beforehand in the ‘Rule Book’ section. The ‘Rule Book’ generally explains everything about the gameplay.

Next, here comes the Introduction Part which consist of the player inserting their names and get to choose either to play as a ‘Single Player Mode’ or as a ‘Multiplayer Mode’. As what has been mentioned before, if the user chooses ‘Single Player Mode’, the user or player will be playing against the computer and if the player chooses a ‘Multiplayer Mode’, the player will be playing against the other player which is also using the same console.

If the player chooses to play as a ‘Single Player Mode’, the program will give the option to the player to select the difficulty level that the player wants to play. The player will have two options, which are whether to play the game in easy or hard mode.

After that, we will take a closer look on the next part which is the Preparation Parts. In this phase, we will explain on how the process of placing the ships will take place for both single player mode and also multiplayer mode. Not to forget, some of the constraints or limitations that the players need to take precautions of and rules to follow while placing the ships.

Before the game begins, each player secretly arranges their ships on their primary grid. Each ship occupies a number of consecutive squares on the grid, arranged either horizontally or vertically. The number of squares for each ship is determined by the type of the ship. The ships cannot overlap (only one ship can occupy any given square in the grid and the ships that being placed have at least one square grid space of distance between one and another). The types and numbers of ships allowed are the same for each player. The ship type, the number of square grids occupied by each of them are the number of each type of ships that need to be placed are stated as follows:

* 1. Aircraft Carrier 5 square grids
  2. Battleship 4 square grids
  3. Submarine 3 square grids
  4. Patrol boat 2 square grids

Number of ships that need to be placed:

1. Aircraft Carrier only 1 ship
2. Battleships only 2 ships
3. Submarines only 3 ships
4. Patrol Boats only 4 ships

In order to start the game, both players need to place all of the stated ships correctly and with the correct order or sequence by inserting the coordinates. The order of the placement should begin with placing one Aircraft Carrier, which occupied 5 square grids, followed by two Battleships, three Submarines and four Patrol Boats. But before that, the player will have an option to choose either they want to place their ship manually by themselves or automatically by the program. We included this feature because we know that placing the ships manually would take some time to finish, so we came out with this initiative to create choices for the players.

If the players intended to place the ships manually by themselves, the program will ask the player whether they want to place their ships horizontally or vertically at first, then they can just place their ships on the board or map wherever they desired the ships to be located by entering a coordinate each time they want to place the ships. So, since the total number of ships are 10 ships, it means that the players need to enter 10 coordinates.

The coordinates that will need to be prompted for the user to input their position of their ships is only allowed in the form of a capital letter of (A through J) or with small letter (a through j), followed by a number (0 – 9) indicating where they are targeting for their torpedo. The program then should check the input whether it has some errors or the input prompted by the player has already been inserted before.

After the ships have been positioned, the game proceeds in a series of rounds. In each round, each player takes a turn to shoot or fire the torpedoes on the target squares in the opponent's grid by entering a coordinate.

Just the same like when the players want to place their ship manually, we also implement the same principle here in the Shooting Phase or Shooting Parts where the players enter a desired coordinate to fire a shot. The coordinates that will need to be prompted and allowed for the user to input their next guess is only in the form of a capital letter of (A through J) or with small letter (a through j) and a number (0 – 9) indicating where they are targeting for their torpedo. The program then should check the input whether it has some errors or the input prompted by the player has already been inserted before.

The program will announce whether the shot fired hits the opponent’s ship or not. If it hits one of the opponent’s ships, the program will print out ‘You hit the target! It’s your turn again!’ on the screen of the console and an ‘X’ mark on the board or map to indicate that the shot hits the target. The player will also get another chance to fire another shot. It will ask the current player, who just successfully hits the opponent’s ships to enter another coordinate that he or she intended to fire a shot.

But, if the shot misses the target or the opponent’s ships, the program will print out ‘You missed! It’s the opponent’s turn!’ on the screen console and the opponents will have their turn to fire a shot. Then, a symbol of water ‘~’ will appear on the map grids which indicates that the shot just missed the target or the opponent’s ships. An activity diagram is included to help understand and see the flow of the Shooting Phase in this Battleship Game clearer.

Diagram

Description automatically generated

Figure 3: Activity Diagram on Shooting or Hitting Opponent’s ships

As shown in the activity diagram above, whenever each of the player successfully hit the target of their opponent’s ships, the Hit Counts increases. When either one of the players successfully reach the total amounts of 30 hits, then the player will automatically win the game. The total amount of hits is 30 because there are 10 ships allocated for each player in total and the total square grids occupied by all of the 10 ships are 30 square grid spaces all in all. So, when the amount of total ships hits are equal to 30, it means that all of the ships have been sunk by the player. The program then will print out the winner on the screen console of the game.

From this point of view, the reader may think that there is no significant differences between the game playing modes of single player and multiplayer. So, we also include two separate diagrams as an aid to differentiate between these two game playing modes.

Diagram

Description automatically generated

Figure 4: Overall Game Diagram for Multiplayer Mode

Diagram

Description automatically generated

Figure 5: Overall Game Diagram for Single Player Mode

As stated earlier in this section, for our game, we will also create a one-person version of the game where ‘the computer’ will play for the second player. In the single player mode (playing against the computer), we worked on the artificial intelligence (AI) to both place the ships and also to shoot the opponent’s ships on behalf of the computer. The implementations of this AI will be touched more in the next part later on, on how it works and what algorithms it will use to guess the coordinates of opponent’s ships and also the algorithms that is being implemented in the coding parts to make sure it satisfies and follows the rules on how to place the ships correctly without making any errors or fault.

The AI will then need to randomly position the computer’s ships on the grid taking into consideration the same factors as for the player’s input.

After the computer takes its turn, the AI must employ the computer to randomly select a shot that they have not previously taken. Then, the screen console must display to the user what the computer guessed, whether it hit any of the player’s ships, whether a ship was sunk, and then display the player’s placement grid showing where ships are located and what has been hit.

At the end of the game, the program will indicate the game is over and who the winner is after either one of the players ships are all been sunk.

Please note that all of the game’s interfaces will be explained more with the aid of pictures that we have screenshot from our actual program to make things clearer to the reader in the next part which is the Implemented User Interfaces in the Major Design Decision part.

3. MAJOR DESIGN DECISIONS

In this section, we will explain more on the tools that we are using throughout this semester projects and how we design our Battleship Game program with some interfaces that we included to make the game looks more attractive, iterative and more user-friendly.

* 1. *Employed Tools and Technology*

We all know that amidst the pandemic Covid-19, everything is being held online and most people only work remotely or simply work from home. So, we have some troubles in communicating with each other and we did not know how to distribute the task effectively until we found a good idea of solutions.

To ease our way of communication with each other during the project, all of us use Github to work on the task together and use the Webex meeting platform to arrange our weekly meetings. For your information, GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project.

* 1. *User Interfaces*

Next, we would like to explain more on our interfaces that we included in our Battleship Game to ease the user to understand the surrounding to play our game. This will help the reader to have a deeper understanding and get an overview on how the real game would look like. Please note that we only include some of the interfaces from the result of our codes.

We made it as the way that the interface should be simple to build, yet contains enough detail to see what is taking place and what each component is. The interface should also be detailed enough to allow full functionality of the game, providing the user with proper control structures and feedback.

The game will show the welcome interface as the user or player opens the program. The welcome interface is shown in figure 5.

Graphical user interface, text

Description automatically generated with medium confidence

Figure 5: Welcome Interface

As shown in Figure 5, on the welcome interface, the player can choose either to start the game straight away or to go through the Rule Book which consist of everything about the gameplay, rules and further instructions.

Text

Description automatically generated

Figure 5: Playing Mode Selection

Right after the player choose to start the game, this Playing Mode Selection interface will pop out so that the player can choose either to play the game as single player or multiplayer.

Background pattern

Description automatically generated

Figure 6: Grid map

The grid map that will be used in this Battleship Game is shown in Figure 6. Notice that the columns are marked with the number of (0 - 9) and the rows are marked with (A – J) to ease the players to read choose the coordinates in the purpose of placing the ship manually and shooting or hitting the opponent’s ships.

Graphical user interface

Description automatically generated

Figure 7: Ship Placement Options

In figure 7, there is an interface that we use to clearly distinguish the options on either placing the ships manually or automatically. With this approach, the player can see clearly their choice and can decide easily by prompting ‘1’ to place their ships manually and prompting ‘2’ to place their ships automatically, in which the AI will just randomly place the ship for the player.

Text

Description automatically generated with medium confidence

Figure 8: Ship Placement Order or Sequence

Before placing the ships (if the player chooses to place the ships manually), the program will show again the correct order to place the ships correctly as a reminder.

Diagram

Description automatically generated

Figure 9: Ship Placement Position

After that, the program will ask the player whether to place the ships vertically or horizontally before asking the players to enter an input of coordinates.

4. IMPORTANT IMPLEMENTATION DETAILS

In this section, we will talk about some of the details on the important implementations that we used and applied in our projects which is the details on the Artificial Intelligence that we implemented in our program.

*4.1 Artificial Intelligence Technology*

One of the priorities in having this AI technology is for the computer to be able to place ships on the board, so that the human player can start shooting and sinking ships. This functionality requirement, along with the possibility of extending AI functionally in future versions of the game, leads us to the design of an AI Player.

One could of course consider, to allow the human player to use parts of the AI functionality. If the game really should be expanded, one could imagine a ‘quick game’ menu option, where the human player is then able to utilize the automatic placing of ships on the game board, simply to get started playing right away, not spending any time placing ships manually on the board.

The properties of any ship are in this version of the game fairly simple. Any given ship can have a size (length of the ship) and an orientation (it can lie either horizontally or vertically on the board). The size of the ship can then be represented as a set of coordinates, the number of which determines the length. It does not require a lot of programming effort to develop some routines that are able to generate a random starting coordinate for placing the first part of a ship, and then, based on a random orientation, creating a set of coordinates needed for creating and placing a ship on the board. And this procedure would then have to be followed for each ship that is to be placed on the game board. The constraints in this situation will (of course) be that no ship can have any of its parts on a game board field that is already occupied by another ship. Moreover, all the parts of a ship must lie within the borders of the game board.

But the placing of ships is only one aspect of the AI functionality that a game of Battleship could have. Like a human player, the machine player would also have to fire shots at the human player’s game board. The firing of shots is quite straight-forward if not considering tactics or some sort of intelligence added to the action. A random set of coordinates within the borders of the game board could easily be generated, and then the shot could be placed on the board. This would be a very basic implementation of shooting capabilities, but it would probably not feel very authentic to play against such a machine.

A few other relevant observations regarding AI features can be made. One possibility would be to extend the capability of the ship placing mechanism and add some tactical sense to the procedure. For example, measures that we have taken is ensuring that each ship has a minimum distance to other ships, so that ships will not end up right next to each other, which would make it easier for the human player to sink the ships, once the group of ships was found on the board. Possibly there would be some advantage to the machine player (it would make the game harder for the human player) if thought was given to the distribution of ships on the game board. Without going into detail here, one could calculate the probability of hitting ships when these are placed in a certain way and apply this calculation to the placing of ships. Through the results and analysis, we are glad that the game appears to work as intended alongside with the implementation of the AI.

5. WORKLOADS AND PROTOCOLS

Contributions of each team members on this documentation and project:

|  |  |
| --- | --- |
| Name | Contributions in percentage (%) |
| Asadujaman Nur |  |
| Johnshon Kwabena |  |
| Muhammad Amjad Bin Abdul Malik |  |

6. REFERENCES

I. http://www.wikipedia.org

II. http://neweranet.com

We would also attach a Github link of our group regarding the codes that we have made for the whole program. This link will redirect straight to the codes in C.

[We-tried/Battleship\_Team\_lightning\_Torpedos.c at main · AsadujamanNur/We-tried (github.com)](https://github.com/AsadujamanNur/We-tried/blob/main/Code%20Implementation/Final%20Version/Team_Lightning_Torpedos/Battleship_Team_lightning_Torpedos.c)