Table of Contents

1. Introduction

1.1 Purpose

This document specifies the requirements of the board game “Battleship”. More precisely, the document describes the scope, objectives, and features of this game. It illustrates the high-level design choices that were made such as the incorporation of object-oriented design in the architecture of the game. The document describes functional and non-functional requirements using structured language, tabular specifications, and natural language. There will be several user stories to characterize the proposed features in the document. The document models system usage and architecture with a UML use-case diagram and a UML class diagram. Moreover, the purpose of this document is to guide the design and implementation of Battleship.

1.2 Project Summary

Project Name:	Battleship
Team Name:	Badger
Team Members:	Angela Yung, Zachary Chan, Kunlong He, Curtis Huang, Kyle Mollard

1.3 Intended Audience

Our target audience will range from seasoned veterans to novice children interested in outwitting their AI opponent. Appealing to this wider target audience, our goal is to provide an accessible and rewarding gaming experience that sharpens critical thinking skills while encompassing the essence of Battleship.

1.4 Project Scope

The aim of this project is to develop a digital implementation of the classic Battleship board game. The game will allow two players to engage in a strategic battle by placing

their fleet of ships on a grid and attempting to sink their opponent's vessels by accurately guessing their locations. The game will include the basic gameplay mechanics such as ship placement, firing shots, and tracking hits and misses. The game will provide an intuitive user interface with appealing graphics and sound effects to enhance the gaming experience. Furthermore, this implementation of Battleship will have one game mode only: single-player versus an AI bot. In summary, this project will be a re-creation of the 1967 original game of Battleship, combining the nostalgia of the classic game with the allure of a modern digital adaptation.

1.5 Background

At the beginning of the game, each player is given five different ships to place on their own 10x10 grid of squares. These ships include: an aircraft carrier (five squares), a battleship (four squares), a cruiser (three squares), a submarine (three squares), and a destroyer (two squares). Each ship occupies a consecutive sequence of squares. Ships can only be placed horizontally or vertically on the grid without overlapping ships or crossing the boundaries of the grid. The user can see two grids - an ocean grid containing the user's ships and a target grid which corresponds to the opponent's ocean grid. Each turn a player will guess a square where they suspect an opponent's ship will be on, and a bomb will be shot towards the square. A hit is when a bomb lands on a square occupied by some ship. On the other hand, a miss is when a bomb does not land on a square occupied by a ship. This mechanic introduces strategic deduction as a ship's orientation constraints and lengths within the grid. Once an entire ship is hit across turns, it will be considered sunk. Once all the ships of a player are sunk, the opposing player is the winner and the game ends.

2. Overall Description

2.1 Features

- Provide a menu that users can easily navigate upon launching the game.
- Allow the user to play through a tutorial of the game.
- Allow the user to play against an AI (referred to here as "bot").

- Allow the user to check the rules of the game by opening a rule book at any time during the game.
- Allow the user to create a new game, load a previous game, and save the current game.
- Allow the user to win the game when all of the bot's ships are destroyed.
- Allow the user to win achievements.
- Allow the user to view their score at all times during the game.

2.2 User Stories

Title: Main Story
User Story: As a player, I want to have a fun and engaging experience while playing Battleship so that I can enjoy the game and feel invested in my progress.
Acceptance Criteria: 1. The game has a user-friendly menu that is easy to navigate. 2. The game implements the core mechanics of Battleship. 3. The user can place their fleet of ships on their ocean grid using a visually appealing and practical graphical interface. 4. The game should provide clear instructions and visual cues to help the user make strategic decisions. 5. It should be clear when it is the user's turn to make a move or the bot's turn. 6. On the user's turn, it should be clear and easy to select a target on the target grid to launch an attack. 7. The game should provide feedback on the outcome of an attack, indicating whether it was a hit or miss. 8. The ocean and target grids should always be updated to show the result of the bot's attacks, marking hits and misses accordingly. 9. The game should continue alternating turns until all ships on one player's board have been completely sunk. 10. The game will provide a notification on the screen when an achievement is won. 11. After the game ends, there should be a clear indication of the winner, and the option to start a new game or quit the game. 12. The game should provide an option to save the game progress.

13. The game should have appealing graphics, animations, and sound effects to enhance the overall gaming experience.
14. The game should be responsive and run smoothly on different platforms and devices, ensuring a seamless gameplay experience.

Sub Stories

Title: AI

User Story:
As a player, I want to be able to play against AI with knowledge of how to strategize so that I can challenge myself against a logical opponent.

Acceptance Criteria:
1. The AI should never fire at the same square twice.
2. The AI should notice when one of its bombs hits my ship and aim for adjacent squares near my ship in future moves as a basic strategy.

Tasks:
1. Write an algorithm that the bot will use as its strategy.
2. Conduct thorough testing to ensure the AI is in fact placing bombs logically throughout the game as opposed to randomly.

Title: Game Mechanism

User Story:
As player, I want to be able to place my ships on the ocean grid before the battle begins, and once the battle begins, I want to be able to select a cell on the opponent's grid to launch an attack, and know if the opponent's ship was successfully hit so that I can have a gaming experience true to the mechanics of Battleship.

Acceptance Criteria:
1. The game provides clear instructions and guidelines for ship placement.
2. Ships can be placed either manually or randomly.
3. The game provides visual feedback on whether the attack was a hit, miss, or a sinking of an entire ship.
4. The game enforces the rules of Battleship and detects illegal moves.

Tasks:

1. Design and implement the ship placement interface.
2. Implement grid cells to represent the game grid.
3. Develop algorithms for validating ship placement and enforcing grid limitations.
4. Integrate options for manual and random ship placement.
5. Design and implement the attack interface.
6. Create interactive grid cells for targeting opponent's ships.
7. Implement logic for determining hit, miss, and ship sinking outcomes.
8. Develop algorithms for validating attack moves and enforcing game rules.

Title: Game Progress

User Story:
As a player, I want to be able to save my progress and continue my game at any time so that I can take breaks.

Acceptance Criteria:

1. The game can be saved at any point during gameplay.
2. Upon saving the game, all relevant game data including ship positions and hit/miss records should be accurately stored.
3. The game can load a previously saved game from the main menu.
4. Loading a saved game should resume gameplay from the exact point where the game was saved.

Tasks:

1. Implement a mechanism for detecting game completion and declaring a winner.
2. Design and implement the game completion screen.
3. Integrate buttons or options for playing again or returning to the main menu.
4. Implement a mechanism for saving progress (i.e. storing the current game)

Title: Achievements

User Story:
As a player, I want to be informed of when I win an achievement so that I can feel

motivated to play more of Battleship.

Acceptance Criteria:

1. The game accurately notifies the user upon winning an achievement.

Tasks:

1. Determine the design and placement of achievement notifications in the user interface.
2. Develop a notification system that can track and detect when an achievement is earned.
3. Implement the logic to determine which achievements have been earned based on user progress.
4. Conduct testing to verify that achievement notifications are displayed correctly and at appropriate times.

Title: User Interface & Sound

User Story:

As a player, I want to be able to navigate the game easily with an intuitive user interface, so that I do not get frustrated in the process of setting up the game.

Acceptance Criteria:

1. The game provides a menu that users can easily navigate upon launching the game.
2. The game allows the user to play through a tutorial of the game.
3. Allows the user to check the rules of the game by opening a rule book at any time during the game.
4. Allows the user to view their score at all times during the game.

Tasks:

1. Design and implement a user-friendly interface with clear and responsive controls.
2. Integrate a rulebook that can be referred to at any point in time
3. Test and optimize the user interface for a smooth and enjoyable gameplay experience.
4. Design a tutorial of the game to teach users how to play

3. System Features

3.1 Functional Requirements

Grid Update

Function	Add new markers on the ocean grid and target grid at the end of a turn.
Description	Adds new markers on the ocean grid and target grid at the end of a turn so that a new hit or miss will appear on the user's grid.
Inputs	Grid coordinates and a boolean value (isHit) corresponding to whether a ship was bombed at the end of a turn.
Source	The coordinates are provided by either the user or the AI.
Outputs	None.
Destination	Grid Manager class, GUI Manager class.
Action	A user's turn ends when they enter a valid coordinate for bomb placement. The input scanning and validation is handled by an Input Scanner class. The value of "isHit" is set by a method that handles checking if a bomb landed on a ship or not. At the end of the user's turn, a hit marker is added to the target grid if isHit is true, otherwise a miss marker is added. At the end of the bot's turn, a hit marker is added to the user's ocean grid if isHit is true, otherwise a miss marker is added. The GUI Manager class uses this information and updates the grids on the user's browser. The Grid Manager class also updates its information to track the changes on the grids logically.
Requires	Input Scanner class, a grid coordinate for bomb placement, and a method that checks if a bomb landed on a location occupied by a ship.
Precondition	None.
Postcondition	The correct location of hit/miss markers appear on the user's ocean grid and target grid at the end of a turn.
Side Effects	None.

Grid Set Up

Function	Place all of the user and bot's ships on their ocean grids.
Description	At the start of the game, the bot and user will both place their ships on their grids following these constraints: no two ships overlap, every part of the ship must be on a tile, and ships must be placed horizontally or vertically, but not diagonally.
Inputs	A set of coordinates provided by the user for ship placement.
Source	UI.
Outputs	A list of coordinates of all the ships for the user, another list for the bot.
Destination	Bot class, Player class, GUI Manager class.
Action	In order to set up the game, the user must select a set of coordinates on which to place their ships. An Input Scanner class will scan and validate the user's selection of coordinates based on the constraints listed in the description. If the input is invalid, the user must reselect the set of coordinates. Once the user input is valid, the user's board is properly set up. The bot will randomly generate coordinates for its ship placement such that it follows the grid constraints. Both the user and bot's choice of coordinates are then stored in the Bot class and Player class in order to reference the location of the ships for later use in the game. The GUI Manager class uses these coordinates and updates the user's ocean grid to show the recently placed ships.
Requires	Two sets of grid coordinates (one for user's ships and one for bot's ships), and Input Scanner Class.
Precondition	None.
Postcondition	The user's recently placed ships will appear on the ocean grid.
Side Effects	None.

Win Condition Tracking

Function	Track if the user or bot has won the game.
Description	Tracks if the user or bot has won the game at the end of each turn.
Inputs	List of ships for bot and user.
Source	Bot class, Player class
Outputs	isGameOver - a boolean value that is set to true if either the user or bot meets win conditions, false otherwise.
Destination	Game object.
Action	At the end of each turn, the game object needs to know if it should keep running (to allow the user to continue playing). It continues running the game if isGameOver is false. The variable isGameOver is true only if the user's number of ships reaches 0 or the bot's number of ships reaches 0. Otherwise, isGameOver is set to false and the game continues. The number of ships for the user and bot are tracked in the Player class and Bot class respectively. The user will be notified if they won after identifying who the list of ships destroyed belonged to. If the list of ships destroyed belonged to the bot, then the user wins. If it belonged to the user, the bot wins.
Requires	List of ships for bot and user.
Precondition	None.
Postcondition	The correct game status is returned to the game that is running.
Side Effects	None.

Win Condition (Tabular Specifications)

Condition	**Action**
If Bot.numShips == 0	isGameOver = true
If Player.numShips == 0	isGameOver = true
If Bot.numShips > 0	isGameOver = false
If Player.numShips > 0	isGameOver = false

Game Statistics

Function	Track the current game statistics and all-time game statistics.
Description	Tracks the current game statistics such as number of ships remaining for the user to view in-game and tracks the user's all time game statistics in order to award the user achievements.
Inputs	List of ships for user and bot, isTutorialComplete - a boolean variable which is true if the user completes the tutorial for the first time, isTutorialPreviouslyCompleted - a boolean variable which is true if the user already completed the tutorial at least once, and numWins - the number of wins by the user.
Source	Bot class, Player class, Game class, all_time_game_stats.txt
Outputs	numShipsUser - the number of ships remaining on the user's grid, numShipsBot - the number of ships remaining on the bot's grid, and any achievement messages printed to the user's screen.
Destination	User's screen.
Action	For each achievement condition that the user meets, the user's screen will display a message corresponding to the achievement. The user's screen will also display the current number of ships remaining for both the user and bot. The Game Statistics class will observe for any changes in the number of ships remaining and notify the GUI Manager class to update the user's screen. All-time game statistics will be recorded in the document all_time_game_stats.txt and will be updated in the Game class as the user plays. The text document will contain the data for variables isTutorialPreviouslyCompleted and numWins. There will be a separate class which parses data from all_time_game_stats.txt.
Requires	List of ships, isTutorialComplete, all_time_game_stats.txt, GUI Manager class.
Precondition	all_time_game_stats.txt must not be empty and must have game stat fields initialized.
Postcondition	The result or state change that occurs after the achievement is unlocked or progress is made.
Side Effects	None.

Game Statistics (Tabular Specifications)

Condition	**Action**
If tutorial is completed by user (isTutorialComplete == true) for the first time (isTutorialPreviouslyCompleted == false)	Print “You’ve unlocked: Complete tutorial”
If numWins in all_time_game_stats.txt is 1 (numWins == 1)	Print “You’ve unlocked: Rookie”
If numWins in all_time_game_stats.txt is 5 (numWins == 5)	Print “You’ve unlocked: Talented”
If numWins in all_time_game_stats.txt is 20 (numWins == 20)	Print “You’ve unlocked: Expert”

Interactable Menu

Function	An interface that allows users to browse and select various menu options.
Description	Displays a menu to the user’s screen and executes the menu option selected by the user.
Inputs	User’s menu option selection.
Source	Scanner class, GUI Manager class, Game class.
Outputs	menuOption - a runnable object that is displayed to the screen and can be selected by the user.
Destination	User’s screen.
Action	The Menu object will consist of a list of entries that are runnable - that is, the menu option selected by the user will be executed. These entries will be methods from the Game class. Examples of some menu options are start game, load game, save game, tutorial, rulebook, and quit game. Using the GUI Manager class, these entries will be displayed to the user’s screen. By using a graphical interface, the user’s menu selection will never be invalid since the user can only select the options available to them on the screen. The user can also

	enter the menu any time during the game by selecting the menu icon on the screen handled by the GUI Manager class.
Requires	User's menu option selection and menu options from Game class to display to the screen.
Precondition	None.
Postcondition	The menu option selected by the user is run.
Side Effects	None.

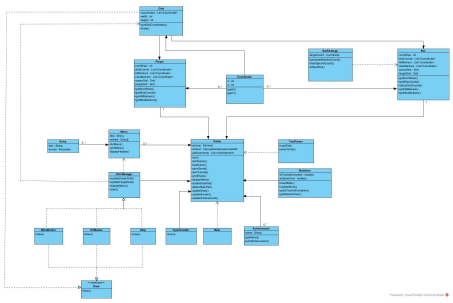
AI Strategy

Function	Compute the bot's next shot.
Description	Computes the bot's next shot based on a set of criteria.
Inputs	List of hit markers, list of miss markers
Source	Bot class.
Outputs	targetCoord - the coordinate of the next shot sent to the user's ocean grid.
Destination	Game class.
Action	When the hit marker list is empty, the bot will pick a random valid coordinate as the target coordinate since it has no hits to make decisions off of. When the hit marker list has at least one entry, the algorithm will prioritize the entries that already have adjacent tiles in the hit list (which means the bot has likely destroyed multiple parts of the ship already) and pick another adjacent tile as the next target. This output will be used in the Game class to update the logic of the game.
Requires	List of hit markers, list of miss markers
Precondition	None.
Postcondition	A target coordinate is chosen as the bot's next move and game logic is updated.
Side Effects	None.

AI Strategy (Tabular Specifications)

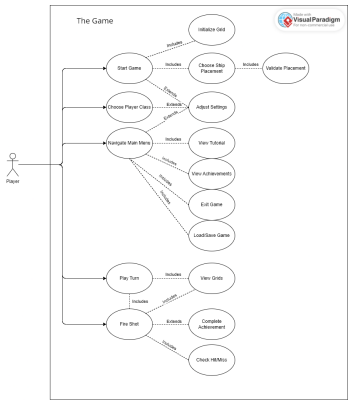
Condition	**Action**
If hit marker list is empty	Generate a random number between 0 and 9 for both the x and y coordinates, and if that coordinate does not exist in the miss marker list, then set targetCoord to the randomly generated coordinate.
If hit marker list is not empty	For each hit in the hit list, check if any entry already has an adjacent coordinate in the hit list. These will be prioritized first. If the entry has at least one adjacent coordinate, then randomly pick another coordinate adjacent to any one of the entry's adjacent tiles. Verify that the randomly picked coordinate is within bounds and does not exist in the miss list or hit list. Then set targetCoord to the randomly picked one. In the other case, if no entries in the hit list have adjacent coordinates, then randomly pick the adjacent coordinates of one of the entries in the hit list as the target.

3.2. UML Class Diagram



(Full version is found in the file named "Battleship_UML_Class_Diagram" in the git repository)

3.3. UML Use-Case Diagram



(Full version is found in the file named “Battleship_UML_Use-Case_Diagram” in the git repository)

4. Non-functional Requirements

4.1 Performance

- The display should update the user’s ocean grid and target grids within 2 seconds of the user ending their turn.
- The user should not have to wait longer than 3 seconds for the bot to place a bomb.
- The user should not have to wait longer than 3 seconds for the bot to place all of its ships down during the Board Set-Up Phase.
- The site should be loaded in less than 3 seconds for someone with an average hardware and network.

4.2 Usability

- The user should be able to understand the rules of the game after either playing the tutorial or reading the rulebook.
- The user should be able to easily navigate the menu and understand what each menu option does.
- The user should be able to place a bomb or ship down easily by using their cursor.