1. Estimate of Person-Hours

Group Discussion/Planning (Sept 20th, 4:30-5 pm): 30 minutes for all members

- **Rationale**: The initial planning meeting is essential for defining goals, assigning roles, and discussing the requirements such as AI levels and custom additions. We allocated 30 minutes to ensure clarity and direction for the project, focusing on efficiently assigning tasks and addressing key project objectives.

Coding: 12-18 hours per lead coder (Andrew, Ellia)

- **Rationale**: With the addition of AI functionality (three difficulty levels) and handling any incomplete Project 1 functionality, coding will require more time. Each coder estimates 12-18 hours, accounting for the AI's complexity, integration of custom additions, and debugging.

Documentation: 6 hours for the documenter (Victor)

- **Rationale**: Project 2 introduces new features and a UML diagram for the custom addition, requiring additional effort to clearly explain the AI functionality and new game mechanics. Victor will spend around 6 hours ensuring the documentation is detailed and accurate.

Commenting: 3-5 hours for the commenter (Deborah)

- **Rationale**: Proper commenting is crucial for ensuring code readability, especially with complex AI features. Deborah will spend 3-5 hours ensuring the code is well-commented, explaining key logic, particularly the AI and custom addition.

•Demo Preparation: 4-6 hours for the demo lead (Xavier)

- **Rationale**: The demo requires showcasing the different AI difficulty levels and the custom addition, along with ensuring all Project 1 features work. Xavier will need 4-6 hours for careful preparation, including rehearsing the demo to ensure a smooth presentation.

2. **Actual Accounting of Person-Hours (Day-by-Day)**

Friday, September 20th

- **Group:**
  - **Time**: 4:30-5:00 pm (30 minutes)
  - **Activity**: Initial meeting to define goals, assign roles, and discuss requirements such as AI levels and custom additions.

- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:**
  - Time: 30 Minutes
  - Activity: Forked GitHub repository and refactored the project's file structure to fix inconsistencies and spelling errors from the previous team.

Saturday, September 21st

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:**
  - Time: 1.5 Hours
  - Activity: Fix ship sinking logic (previous team had it so that a single hit would sink a ship). Fix ship count logic (previous team had it so that players could have different numbers of ships). Add inheritance-based framework for later implementing the various AI difficulties.

Sunday, September 22nd

- **Group: N/A**

- **Xavier:**
    - Time: 15 Minutes
    - Activity: Briefly spent time to look over the code and see how the project worked.
- **Ellia:**
    - Time: 15 minutes
    - Activity: Spent time reviewing code to understand the structure.
- **Deborah:**
    - Time: 45 minutes
    - Activity: Reorganized the commenting format of the code to match the new files created and to ensure easier readability. Reviewed the old comments and decided on what to keep and what to update.
- **Victor:**
    - Time: 20 minutes
    - Activity: Took some time to review the code and understand how the project functions.
- **Andrew:**
    - Time: 1 Hour
    - Activity: Add a board view into the game (previous team had it so that the board was displayed neither during placement nor between turns). Add UI prompts to allow the user to select an AI adversary.

Monday, September 23rd

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:**
    - Time: 50 minutes
    - Activity: Created UML state diagram for custom project element
- **Deborah:**

    - Time: 30 minutes
    - Activity: Reviewed the code in order to edit and update potential spelling errors and bulky comment blocks to make the code look uniform.
- **Victor: N/A**
- **Andrew:**
    - Time: 1 Hour
    - Activity: Finish the various AI difficulty tiers, specifically the medium tier. Preformed rudimentary testing to ensure functionality.

Tuesday, September 24th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:**
    - Time: 20 minutes
    - Activity: Researched libraries to utilize for sound effects
- **Deborah:**
    - Time: 25 minutes
    - Activity: updated the comments on the new code committed to the git to ensure consistency and to easily understand the updated code.
- **Victor:** N/A
- **Andrew: N/A**

Wednesday, September 25th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:**
    - Time: 40 Minutes
    - Activity: Sourced and edited sound files for use by battleship project.

- **Deborah:** N/A
- **Victor:**
    - Time: 2 hours
    - Activity: Drafted the documentation document, refined the format for consistency, reviewed the code to monitor progress, and consulted the rubric before starting the documentation process.
- **Andrew: N/A**

Thursday, September 26th

- **Group: N/A**
- **Xavier:**
    - Time: 30 Minutes
    - Activity: Took a brief overview of the codebase and tested a basic game loop. Gave feedback to Andrew on some potential bugs and issues.
- **Ellia:**
    - Time: 6 hours
    - Activity: Implemented sound effects with playsound library. Learned about and set-up virtual environment to dependency requirements
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:**
    - Time: 30 minutes
    - Activity: Updated README file, cleaned superfluous files out of the repository, fixed bug that was found by Xavier.

Friday, September 27th

- **Group: N/A**
- **Xavier:**
  - Time: 2 Hours
  - Activity: In-depth review of code. Documented findings in documentation file. Tested playsound implementations. Started test cases list. Added shorter sounds. Fixed dependency requirements.txt not working on windows.
- **Ellia:**
  - Time: 20 minutes
  - Activity: Reviewed documentation and performed QA
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:**
  - Time: 10 minutes
  - Activity: Reviewed & merged changes on the development branch into the main branch.

Saturday, September 28th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:**
  - Time: 15 minutes
  - Activity: Reviewed the code progress and the documentation to confirm that everyone had updated their activities

- **Andrew: N/A**

Sunday, September 29th

- **Group: N/A**
- **Group: N/A**
- **Xavier:**
  - Time: 15 minutes
  - Activity: Tested new changes after comments were added. Deleted sound_test.py. Changed pip requirements which downgrades to older version of playsound.
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:**
  - Time: 2 hours
  - Activity: Revised everyone's contributions in the document for clarity, completed the code documentation, and added final touches.
- **Andrew: N/A**

3. Battleship Game Documentation

Overview

This documentation details the implementation of a Battleship game developed by Deborah and her team. The game is built using Python and incorporates multiple modules that handle various aspects of the game, including board setup, gameplay logic, player interactions, and ship management. The objective of the game is to strategically sink all of the opponent's ships before they sink yours.

Module Descriptions

1. board.py - Board Module

This module defines the Board class, responsible for managing the game board. It provides essential functionality for setting up the board, placing ships, processing player attacks, and checking the game state.

Key Components:

- **Board Initialization:**
    - Sets up a grid that represents the game board.
    - Initializes a 10x10 grid (or customizable size) to represent the playable area.
- **Ship Placement:**
    - Provides methods for placing ships on the board based on user input.
    - Validates whether a ship can be legally placed in a desired location, taking into account existing ships and board boundaries.
- **Attack Handling:**
    - Processes attacks made by players on the opponent’s board.
    - Checks if the attacked coordinates hit or miss a ship and updates the board status accordingly.
- **Win Condition Checking:**
    - Monitors the game state to determine if one player has successfully sunk all the opponent's ships.

2. player.py - Player Module

This module defines the Player class, which manages the state and actions of each player in the game. Each player has a board for placing ships and another board for tracking their guesses on the opponent's ships.

Key Components:

- **Player Initialization:**
    - Each player is initialized with their own instance of the board and a set of ships.
    - Tracks the player's name, score, and number of remaining ships.
- **Member Variables:**

        - **Name:** Name of the player.
        - **Board:** Each player has its own board for placing ships.
        - **Guesses:** Each player has another board corresponding to their attacks/guesses.
- **Ship Management:**
    - Handles the process of placing ships on the player's board.
    - The player is prompted to input the number of ships (between 1 and 5), and for each ship, the player is asked to provide the starting position and orientation (horizontal or vertical). The function checks if the position is valid and places the ship on the board.
- **Guess Handling:**
    - **Submit Guess(opponent, position):**
        - Handles player attacks. Requires an opponent player class and a coordinate for the attack.
    - **Make Guess(opponent):**
        - Allows player to guess opponent's ship positions.
        - Records the result of the guess with 'X' for hit and 'O' for miss.
        - Prompts for a valid guess until input format is correct (LetterNumber format, e.g., B3).

3. ship.py - Ship Module

This module contains the Ship class, which manages the parameters for ships that players can place on the board. It ensures that ships are placed correctly, checking for overlaps and ensuring compliance with the game's rules.

Key Components:

- **Ship Initialization:**
    - Sets up a ship with a specified size, position, and orientation (horizontal or vertical).
    - Initializes a destroyed state to track if the ship has been sunk.
- **Member Variables:**
    - **Size:** Tracks the size of the ship.
    - **Position:** Tracks the position on the grid (starting point).
    - **Orientation:** Tracks the orientation ('H' for horizontal, 'V' for vertical).
    - **Coordinates:** Initialized using the get_coordinates helper function.
    - **Destroyed:** A boolean flag to determine if the ship is sunk.

- **Coordinate Generation:**
    - **Get Coordinates():** Helper function that returns a list of coordinates in a tuple based on ship orientation and size.
- **Boundary Checking:**
    - **Is Within Bounds(board_size):** Checks if the ship is within the boundaries of the board.
    - Prevents ships from being placed outside the grid or extending beyond the limits.
- **Overlap Checking:**
    - **Overlaps With(other_ship):** Checks if this ship overlaps with another ship by comparing their coordinates.

4. AI Logic

This section describes the AI logic used to enhance the gameplay experience for single-player modes.

- **AIPlayer Class (inherits from Player):**
    - Contains the AI logic, initializing with its own board and guesses board.
    - **Make Guess(opponent):** Not implemented in the base class (requires subclass implementation).
- **AIPlayerEasy(AIPlayer):**
    - Initializes easy AI player with a predefined name.
    - Randomly selects positions for guesses and submits until a valid one is found.
- **AIPlayerMedium(AIPlayer):**
    - Initializes medium AI player with additional hit tracking variables.
    - Employs strategies based on previous hits to make educated guesses.
- **AIPlayerHard(AIPlayer):**
    - Initializes hard AI player that scans the board and automatically hits uncovered ship positions.
- **AIDifficulties Enum:**
    - Defines difficulty levels for the AI: EASY (Random firing), MEDIUM (Classic human strategy), HARD (Cheating strategy).

5. main.py - Game Logic Module

This module implements the main game logic for the Battleship game, handling the overall flow of the game and player interactions. It integrates the functionalities of other modules to create a cohesive gameplay experience.

Key Components:

- **Game Initialization:**
  - Sets up the game environment and initializes players.
  - Configures the game phase to start with ship placement.
- **Turn Management:**
  - Alternates turns between players, allowing each to place ships and attack.
  - Implements logic to handle player actions and ensures turn order is followed.
- **Event Handling:**
  - Processes user inputs for ship placement and attacks, typically through a console or graphical interface.
  - Updates the game state based on player actions and communicates outcomes (hit/miss).
- **Win Condition Check:**
  - Monitors game progress to determine if a player has won by sinking all of the opponent's ships.
  - Triggers end-of-game scenarios and displays appropriate messages to the players.

Architecture

The game follows a modular architecture with clear separation of concerns. Each module is responsible for a specific aspect of the game:

- **Game Control:** Managed by main.py, which orchestrates the flow of the game.
- **Data Definitions:** Handled in ship.py and board.py, defining ships and board operations.
- **State Management:** Managed by player.py, which tracks player actions and statuses.

- **Rendering:** Although not included in the provided snippets, the game can be extended with a rendering module to visually display the game state and player interactions.

Setup and Execution

To set up and run the game, ensure all modules are placed within the same package directory. The game is initialized and run by creating an instance of the Game class (in main.py) and calling its main loop method.

```
from battleship import Game

if __name__ == "__main__":
   game_instance = Game()
   game_instance.game_loop()
```

Conclusion

This documentation provides a comprehensive overview of the Battleship game's structure and functionality. Each module is designed to handle specific tasks, ensuring that the codebase is easy to manage and extend. Future development can build upon this foundation with additional features, improvements to the game mechanics, or enhancements to the user interface.

4. Testing

No	**Requirement**	**Test Case Passed**
1	Invalid input for ship placement check	Yes
2	Invalid input for ship attack check	Yes
3	Correct sound being played on move	Yes
4	Correct easy enemy AI	Yes
5	Correct medium enemy AI	Yes
6	Correct hard enemy AI	Yes

7	Player cannot attack previous attack position	Yes