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:
    - Activity:
- **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:**
    - Time:
    - Activity:
- **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:** N/A
- **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:** N/A
- **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:** N/A
- **Andrew: N/A**

3. Battleship Game Documentation

Ship Class:

- Contains the parameters for the ship that is to be placed by the user.
- It determines the size of each ship depending on the number of ships the user places on the grid.
- If the user would like to place 5 ships, then it would ask the user to place each ships of the following dimension, according to the mentioned orientation.
    - 1st boat = 1x1, 2nd boat = 1x2, 3rd boat = 1x3, 4th boat = 1x4, 5 th boat = 1x
- The class also makes sure the following:
    - It doesn't allow any of the factors such as dimension or orientation that would cause it to go beyond the bounds.
    - It also makes sure that the ships of the same player don't overlap with each other.
- Member variables
    - Size
    - Position
    - Orientation (horizontal or vertical)
    - Coordinates (initialized using get_coordinates helper function
    - Destroyed boolean (determines if ship is sunk)
- Is_within_bounds(board_size)
    - Checks if the ship is within the board boundaries
- get_coordinates()
    - Helper function which returns a list of coordinates in a tuple
- overlaps_with(other_ship)
    - Helper function that checks is the ship overlaps with another ship

Player Class

- The Player class represents a player in a battleship game. Each player has a board for placing ships and another board for tracking their guesses on the opponent's ships.
- Member Variables
    - Name: name of player
    - Board: each player has its own board

    - Guesses: each player has another board corresponding to its attacks/guesses
- Place_ships(num_ships)
    - 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
- Print_boards()
    - Helper function to print both the player's boards, (guesses and personal board)
- Submit_guess(opponent, position)
    - Handles player attacks. Requires an opponent player class, and a coordinate for the attack (position)
    - Used as a helper function in make_guess and other AI attack methods

- **make_guess(self, opponent)**
    - Allows player to guess opponent's ship positions.
    - Player enters a position, checked for validity.
    - Result of the guess is recorded with:
        - 'X' for hit
        - 'O' for miss.
    - **Loop for valid guess**:
        - Prompts for guess until valid input.
        - Validates input format (LetterNumber format, e.g., B3).
        - Converts input into coordinates:
            - X: Number part (0-based index).
            - Y: Letter part (0-based index).
        - Checks if guess is within bounds.
        - Submits guess to opponent.
- **AI Player Logic**
    - **get_random_position()**
        - Returns a random tuple (position on board).
    - **add_tuples(tuple1, tuple2)**

        - Adds two tuples element-wise.
- **AIDifficulties Enum**
    - EASY: Random firing dummy AI.
    - MEDIUM: Classic human strategy AI.
    - HARD: Cheating strategy AI.
- **AI_factory(difficulty)**
    - Factory function to get AI players with varying difficulties.
- **AIPlayer Class (inherits from Player)**
    - Contains the AI logic.
    - **init()**:
        - Initializes board and guesses board.
    - **place_ships(ship_count)**
        - Places ships on board randomly.
    - **print_boards()**
        - Placeholder function.
    - **make_guess(opponent)**
        - Not implemented in the base class (requires subclass implementation).
    - **AIPlayerEasy(AIPlayer)**
    - **init()**:
        - Initializes easy AI player with name "AI (EASY)".
    - **make_guess(opponent)**
        - AI randomly selects a position.
        - Submits guess until valid.
- **AIPlayerMedium(AIPlayer)**
    - **init()**:
        - Initializes medium AI player with additional hit tracking variables:
            - initial_hit
            - previous_hit
            - hit_direction
    - **clear_strategy_if_sunk(opponent)**
        - Resets strategy if ship associated with initial_hit is sunk.
    - **make_guess(opponent)**
        - Three cases for making a guess:
            1. **No ship targeted**:
                - Randomly guesses until a hit is found.
            2. **Hit but no orthogonal hit**:

        - Probes in all 4 directions to find valid hits.
    3. **Hit with direction strategy**:
        - Continues probing in the direction of the hit.

- **AIPlayerHard(AIPlayer)**
    - **init()**:
        - Initializes hard AI player with name "AI (HARD)".
    - **make_guess(opponent)**
        - Scans the board and automatically hits uncovered ship positions.

Board Class

- Initializes a board with a default size of 10x10 and an empty grid.
- Ships are stored in a list.

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