1. Estimate of Person-Hours

- **Group Discussion/Planning (Sept 6th, 3-5 pm):** 2 hours for all members
    - **Breakdown of Roles:** Andrew and Xavier (Lead Coders), Deborah (Documenter), Victor (Commenter), Ellia (Demo Lead)
- **Coding:** Each lead coder (Andrew, Xavier) might work around 10-15 hours during the project.
- **Documentation:** Deborah may spend around 5 hours.
- **Commenting:** Victor may spend 3-5 hours reviewing and adding comments to the code.
- **Demo Preparation:** Ellia might spend around 4-6 hours preparing for the demo.

Details on How We Estimated Person-Hours:

- **Group Discussion/Planning (2 hours for all members):**
    - The beginning planning of a project is crucial, and getting everyone on the same page usually takes a bit of time. We set aside two hours for our first meeting to make sure we could thoroughly discuss our goals, decide on the coding language, and assign roles without rushing. It's just the right amount of time to ensure we cover all bases but keep things moving.

- **Coding (10-15 hours per lead coder):**
    - Coding requires thinking through problems, experimenting with solutions, and fixing unexpected issues. From projects we each did in the past; we've found that a couple of days per coder is a realistic timeframe. It gives us enough wiggle room to handle challenges that pop up while coding complex features.

- **Documentation (5 hours for the documenter):**
    - Creating effective documentation is key to ensuring that everyone can understand and use the project long after we're done. Allocating about five hours lets us produce clear, thorough documents that not only explain how everything works but also why we made certain choices. This helps future

users and maintainers navigate and adapt our project without any guesswork.

- **Commenting (3-5 hours for the commenter):**
    - Comments in the code make it easier for someone else (or ourselves in the future) to understand what's going on. We estimate 3-5 hours based on the complexity of the code and how much explanation it needs. This ensures our code is not just functional but also accessible.

- **Demo Preparation (4-6 hours for the demo lead):**
    - A smooth demo takes careful planning, a few run-throughs, and tweaks to make sure everything looks right on the day. Allocating 4-6 hours gives our demo lead enough time to prepare thoroughly, ensuring we present our project in the best possible light.

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

Friday, September 6th

- **Group:**
    - **Time:** 3-5 pm (2 hours)
    - **Activity:** Initial meeting to discuss coding language, division of roles, and planning next steps.
- **Xavier:**
    - Time: 30 Minutes
    - Activity: Working with team to schedule.
- **Ellia:**
    - Time: 40 Minutes
    - Activity: Began working on code outline
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew: N/A**

Saturday, September 7th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:** N/A

Sunday, September 8th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew: N/A**

Monday, September 9th

- **Group:**
    - **Time:** 4:35PM-4:55 pm (**20 minutes)**
    - **Activity:** Second meeting to discuss coding progress, review the rubric together, and work on the technical details such as our GitHub.
- **Xavier:**
    - **Time**: 1 Hour
    - **Activity**: coding basic game loop and logic, proof of concept in terminal
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** N/A
- **Andrew:**
    - **Time:** 30 Mins

- **Activity**: Setup the GitHub repository for the group, inviting all members.

Tuesday, September 10th

- **Group:**
  - **Time:** 12:15PM-12:25 pm (**10 minutes)**
  - **Activity:** Met with the GA to discuss progress on project
- **Xavier:**
  - **Time**: 3 hours
  - **Activity**: Added raylib/graphics, menu and ui to the codebase
- **Ellia:** N/A
- **Deborah:**
  - **Time Spent**: 2 hours
  - **Activity**: Worked on the 'Documentation' document, organizing its format, detailing the document structure, and providing team members with access instructions.

- **Victor:**
  - Time Spent: 1 hour
  - Activity: Updating the comments of the code to match the updates from the lead coders to ensure readability and consistency for better understanding.
- **Andrew:**
  - **Time Spent:** 2 hours
  - **Activity:** Wrote code for more object-oriented/'Pythonic' game core with support for ship placement changes and reorientations. This was later scrapped as the work required to merge into the evolving code base would not be an efficient use of development time.

Wednesday, September 11th

- **Group: N/A**

- **Xavier:**
    - **Time: 1.5 hours**
    - **Activity:** Coded mouse position on board info, and col-row denotations, merged Andrew's code. Added remaining ships to be placed visualization. Added different color when in place mode and hovering over an invalid cell.
- **Ellia:** N/A
- **Deborah: N/A**
- **Victor:**
    - Time Spent: 1 hour
    - Activity: Formatting the comment structure for every file within the battleship game file to ensure each line code is properly explained to ensure each team member understood the logic of the code and to ensure better understanding.
- **Andrew:**
    - **Time Spent**: 3 hours
    - **Activity**: Adding finishing touches to the codebase, including adding row letters & column numbers, adding the ability to place ships both horizontally and vertically, adding new font face for all the text, and re arranging some of the user interface elements.

Thursday, September 12th

- **Group: N/A**
- **Xavier:**
    - Time: 1 Hour
    - Activity: Added ship sunk cell visuals and message when player sinks a ship. Fixed bug where player could attack a cell they previously sunk/hit/missed.
- **Ellia:**
    - Time: 1 Hour

    - Activity: Spent time performing QA to ensure that game ran smoothly and there were no ways to break the performance. Ran through an example demo.
- **Deborah:** (Pending information)
- **Victor:**
    - Time: 1 hour
    - Activity: Updated the comments of the code to match every bug fix or code change that was performed by the lead coders to ensure consistency throughout the coding process.
- **Andrew: N/A**

Friday, September 13th

- **Group: N/A**
- **Xavier:**
    - Time: 10 minutes
    - Activity: Added documentation to the repository
- **Ellia: N/A**
- **Deborah: N/A**
- **Victor:** N/A
- **Andrew:** N/A

Saturday, September 14th

- **Group: N/A**
- **Xavier:** N/A
- **Ellia:** N/A
- **Deborah:** N/A
- **Victor:** (N/A
- **Andrew:** N/A

Sunday, September 15th

- **Group: N/A**
- **Xavier:**
    - Time: N/A
    - Activity: Update Documentation
- **Ellia:**
    - Time: 20 minutes
    - Activity: Ran through program to check for bugs, created outline for demo
- **Deborah:**
    - Time: 30 minutes
    - Activity: Made final changes to the document and edited everyone's input.
- **Victor:** N/A
- **Andrew:** N/A

3. Battleship Game Documentation

Overview

This documentation details the implementation of a Battleship game developed by Xavier and Andrew. The game is built using Python and incorporates multiple modules to manage game mechanics such as board setup, gameplay logic, and rendering.

Module Descriptions

1. game.py - Game Module

This module defines the Game class, which acts as the central controller for the game. It manages the game phases, player turns, and transitions between different stages of the game (menu, ship placement, and attack phase).

Key Components:

- **Initialization:** Sets up the game, initializing players and setting the initial game phase to the menu.

- **Game Phases:** Controls the transitions between the menu, placing ships, attacking, and the game-end phase.
- **Turn Management:** Handles the logic for alternating turns between two players.
- **Event Handling:** Processes mouse inputs for placing ships and attacking.
- **Message Display:** Updates and displays messages based on game state changes.

2. constants.py - Constants Module

Defines various constants used throughout the game such as ASCII values for key inputs, dimensions for the game window, and color settings for different game elements.

Key Components:

- **ASCII Constants:** Key codes for handling keyboard input.
- **Cell and Board Constants:** Definitions for cell sizes and types (hit, miss, empty).
- **Color Information:** Legends and info texts for rendering the game board.

3. board.py - Board Module

Contains the Board class responsible for managing the game board. This class provides functions to place ships, validate cell states, and check for ship existence at a particular location.

Key Components:

- **Board Initialization:** Sets up a grid representing the game board.
- **Ship Placement Validation:** Checks whether a ship can be legally placed in a desired location.
- **Cell Status Checking:** Determines the status of a cell (e.g., if it contains part of a ship).

4. player.py - Player Module

Defines the Player class that manages each player's state, including their board and the ships they have placed. This module is crucial for tracking individual player actions and their outcomes.

Key Components:

- **Player Initialization:** Each player is initialized with their own board.
- **Ship Management:** Players can place ships and record attacks on their board.

5. renderer.py - Renderer Module

Handles all rendering operations for the game, drawing the board, ships, and text messages based on the game state.

Key Components:

- **Board Rendering:** Draws the game board and updates it based on the changes in game state.
- **Text Rendering:** Displays various messages and game information on the screen.

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 game.py.
- **Data Definitions:** Constants and configurations in constants.py.
- **State Management:** Handled by board.py and player.py.
- **Rendering:** Visual output through renderer.py.

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 and calling its main loop method.

python

```
from battleship import Game

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

Conclusion

This documentation provides a clear 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 or improvements to the game mechanics.