Team 6

Battleship Game

Table of Contents

EECS 581 - PROJECT 2

1. Introduction

1.1 Purpose

Battleship game is a two-player game with manual ship placement. Players take turns firing shots at each other's board, and the first player to sink all the opponent's ships wins. This project purpose is enhancing the traditional game with addition of AI mode with different levels of complexity and bombing effects so that user can have more challenges and dynamic gameplay experience while playing the game

1.2 Scope

- Modifying the program of traditional Battleship game from Team 35 code and documentations
- Adding AI mode with different levels (easy, medium, hard)
- Implementing a new bomb feature
- Testing and balancing new feature

1.3 References

- EECS 581: Software Engineering II LEC Project Instruction
- Team 35 Project 1 repository

1.4 Overview

- The document is organized into 5 sections, with possible subsections included. The sections include:
    - Introduction - Purpose, Scope, References, Overview
    - Specific Requirements - Functionality and Interface
    - Appendices
    - Getting Started
    - Testing

2. Specific Requirements

2.1 Functionality

2.1.1 **class Ship** : create a ship with properties, including name, size, coordinates and hits.

- **is_sunk(self)**: check if a ship has been sunk, in case a ship is fully sunk, return True.

2.1.2 **class Board**: display the game board for both players, and control the state of the board (placing ships and processing shots fired).

- **display(self, show_ships=False)**: provide for the player a visual representation of the current state of the game board and the ability to see the ship positions.
- **place_ship(self, ship, start_row, start_col, horizontal=True)**: place a ship of a certain size on the board either horizontally or vertically, also, make sure that there is no overlap or out-of-bounds error, and keep track of the ship's coordinates, if the ship placement is valid, add the ship to list.
- **fire(self, row, col)**: processes a shot fired at a certain location on the board, determines if the shot is a hit (prints "Hit!" and change the board cell to X), miss (print "Miss!" and change the cell to O), or the ship is sunk, and updates the board state accordingly.

2.1.3 **class AI**: represents an AI player which ships placing and determining where to shoot opponent's ships. Given several difficulty levels (easy, medium and hard) for the human player by using strategy. decision-making algorithms to modify the AI's behavior from random shot for easy level to highly strategic targeting as hard.

- **__init__(self, board, difficulty)**: Initializes the AI by allocating a difficulty level and a pointer to the game board. Also, initializes last_hit to hold potential next shots based on prior hits and last_hitto record the most recent successful hits.
- **place_ships(self, ships)**: assigns ships to the board at random, selecting the ships' locations and directions for the AI player. AI chooses starting locations and directions (N, S, E, and W) at random.
- **fire(self, opponent_board)**: fires a shot at the opponent's board utilizing a strategy based on the AI's level of difficulty.
    - <u>Easy level</u>: the AI selects randomly a firing point to avoid choosing the same spot twice.

- Medium level: the AI expands possible_targets to append adjacent cells if it scores a hit. It will keep shooting at these nearby cells until it sinks the ship or runs out all possibilities, at which point it resumes firing at random.
- Hard level: the AI attempts to fire at every coordinate of that ship in a systematic fire until an opponent ship is hit, because it is aware of ship placements, the AI becomes nearly unbeatable.

2.1.4 **initialize_game()**: ensures the game is ready to play and has everything the player has everything they need for a smooth gameplay, including creating game boards for player and AI and letting players place their ships manually and AI place its ships based on the level of difficulty.

2.1.5 **place_ship_manually(board, ship)**: allows the player to manually place a ship to the board game. It asks the user enters a valid beginning point and direction for locating a ship, displaying the board for reference, also verifies that the player's inputs are valid and no ships are placed outside the allowed space or overlap one another.

2.1.6 **game_loop(player_board, ai_board, ai)**: manages the game's progression, ensuring both AI and the player alternately fire at each other's board, keep track of the current turn number, show relevant game data and check win conditions. This function also handles an activity of using a bomb, it tracks if a special bomb is used and continues updating the information accordingly.

2.1.7 **fire_bomb(opponent_board, row, col)**: implements a special attack function that processes multiple shots fired at once. The function gives the player a "bomb" that strikes every location in a given row and column, adding a strategic and powerful element to the game. This feature was created in order to raise the chances of hitting and maybe sinking several ships in a single turn.

2.1.8 **player_turn(opponent_board, player_name, bomb_used)**: manages player's action during their turn, providing them a choice between potent bomb attack (if available) or a normal shot. Handles all input, verification and feedback to help the player make a valid move in the game.

2.1.9 **if __name__ == "__main__"**: begins the main game loop and initializes important components for setting up the game environment. Determines whether to enable debugging mode (**DBG**) in response to command line arguments. This block is essential since it ensures that every element is initialized correctly before the game starts.

2.2 Interface

- Text-based and operates through the command line or terminal.

3. Appendices

- Online.Visual-Paradigms (Not Required)

4. Getting Started

1. Preparation
   - Have Python 3.x installed.
   - Save the `battleship.py` file to your working directory.
2. Running program
   - Open a terminal and navigate to the directory containing `battleship.py` using the command `cd`.
   - Run the program by typing `python battleship.py`.
3. Setup game
   - Select the difficulty for the AI, type "*easy*", "*medium*" or "*hard*" and press Enter.
   - Enter the number of ships (from 1 to 5) you wish to have in this game.
   - Place ships by specifying a starting location and orientation. The input can be in lowercase or uppercase (e.g., a1 or A1).
   - If you have more than one ship, enter the direction for longer ships (N, S, E, W).
4. Human vs AI gameplay
   - Player and the AI take turns in this turn-based game.
   - Player's turn:

+ Player will be prompted to fire a shot or use a potent bomb (only once per game).
+ To fire a shot, input "*shot*" and then enter the desired attack coordinates (e.g., A5).
+ To use a bomb (if available), select it by typing bomb and the coordinates (e.g., A5). The entire row and column will be attacked by the bomb.

- AI's turn:
  + The AI will automatically take a turn based on the difficulty level that the player chose from the game setup then display the outcome (e.g., hit!, missed, or AI wins).

5. Exiting

- When the player or AI's ships are all sunk, the game will end. And the system will announce the winner.
- In case user want to exit early, use Ctrl+Q.

5. Testing

Test cases are used in the Battleship game to verify that key features such as ship placement, shot accuracy, win conditions, and AI decision-making are working as intended. Through extensive testing of user input processing, board state updates, and special actions like using bombs, they assist in finding flaws and guarantee a seamless and enjoyable gaming experience for gamers.

Test Case ID	TC-01
Description	Testing easy mode with valid input
Test Data	number of ships: 2, choose using shot for the entire game
Expected output	user and AI take turns playing the game, AI fires a shot randomly until one side has all ships sunk
Actual output	AI miss its shot several time which mean it fires shots randomly, player won
Status	Pass

Test Case ID	TC-02
Description	Testing medium mode with valid input
Test Data	number of ships: 2, choose using shot for the entire game
Expected output	user and AI take turns playing the game, after AI score a hit, it fire nearby cells until sink all ships from player
Actual output	AI randomly fire a shot until score its first hit, AI continue firing nearby cells, AI won
Status	Pass

Test Case ID	TC-03

Description	Testing hard mode with valid input
Test Data	number of ships: 2, choose using shot for the entire game
Expected output	user and AI take turns playing the game, AI hit exactly the location of player’s ships, AI win
Actual output	AI aware player’s ships placement and sink all of them, AI won
Status	Pass

Test Case ID	TC-04
Description	Testing the attack bomb
Test Data	AI mode: easy, number of ships: 2, choose using bomb for first turn
Expected output	print out the AI’s board with attacked row and column including O(if miss) and X(if hit)
Actual output	used a bomb but miss all targets, print AI board with attacked row and column
Status	Pass

Test Case ID	TC-05
Description	Testing invalid amount of ships
Test Data	number of ships: 6
Expected output	Please enter a number between 1 and 5 Then, ask the player again for the amount of ships
Actual output	Please enter a number between 1 and 5. Enter the number of ships (1 to 5):
Status	Pass

Test Case ID	TC-06
Description	Testing out of bounds ship placement
Test Data	number of ships: 1, starting point: A11
Expected output	Position out of bounds! Please try again.

	Then, ask the player again for the starting point
Actual output	Position out of bounds! Please try again. Enter starting position (e.g., A5):
Status	Pass

Test Case ID	TC-07
Description	Testing overlap ships placement
Test Data	number of ships: 2, ship 1 location: A3, ship 2 location: A3
Expected output	Ship overlaps with another ship! Please try again. Then ask the player to enter a different coordinate
Actual output	Continue asking user to enter a different location for ship 2 until valid
Status	Pass