Battleship Game - EECS 581 Project (Team 27 - Team 3 Additions)

Introduction

This repository contains the implementation of a classic Battleship game, developed as part of EECS 581 Project 1. The game is a command-line interface (CLI) application written in Python. The objective of the project is to simulate the Battleship game, allowing two players to engage in turn-based gameplay. This document provides an in-depth look at the structure, functionality, and flow of the program, as well as detailed descriptions of the key classes and methods used.

Table of Contents

Overview

The Battleship game is a two-player turn-based strategy game. Each player places a set number of ships on their board. Players alternate turns to attack each other by selecting coordinates on the opponent's board. The game ends when one player successfully sinks all of the opponent's ships.

This implementation features:

- A fully functional game loop.
- Turn-based logic to allow players to attack each other.

- Dynamic placement of ships with checks for valid ship positions and game conditions.

File Structure

The codebase is split into the following key files:

- **`ship.py`**: Contains the `Ship` class.
- **`player.py`**: Contains the `Player` class.
- **`board.py`**: Contains the `Board` class.
- **`battleship.py`**: Acts as the main game engine and handles the game loop.

Class Descriptions

Ship Class

File: `ship.py`

The `Ship` class represents a ship in the game. It manages the following attributes and functionality:

- **Attributes**:
  - `size`: The length of the ship (number of squares it occupies).
  - `hits`: The number of successful attacks the ship has sustained.
  - `orientation`: The alignment of the ship on the board (horizontal or vertical).
  - `location`: The coordinates of the ship's position on the board.
- **Methods**:
  - `is_sunk()`: Checks if the ship has been sunk (i.e., if the number of hits equals the ship's size).

Player Class

File: `player.py`

The `Player` class represents a player in the game and handles player-related functionality.

- **Attributes**:
  - `name`: The player's name.
  - `board`: The player's personal board, which tracks the placement of their ships.
  - `attack_board`: A secondary board used by the player to track their attacks on the opponent.
  - `ships`: A list of ships the player has placed on their board.
- **Methods**:
  - `place_ship()`: Allows the player to place ships on their board, with checks for valid positioning.
  - `attack()`: Allows the player to attack a specific coordinate on the opponent's board.
  - `is_defeated()`: Checks if all of the player's ships have been sunk, which determines if they have lost the game.

Board Class

File: `board.py`

The `Board` class manages both the player's board and their attack board. It tracks all ship placements and interactions during the game.

- **Attributes**:
  - `board`: The main board, used to track the player's ship placements and the opponent's attacks.
  - `attack_board`: A separate board used by the player to track their own attacks on the opponent.
- **Methods**:
  - `place_ship()`: Validates and places a ship on the player's board.
  - `update_boards()`: Updates both boards based on the outcome of an attack.
  - `check_hit_or_sunk()`: Determines whether an attack was a hit, miss, or resulted in a ship being sunk.

- `print_boards()`: Displays the current state of both the player's board and attack board.

Game Flow

The Battleship game consists of the following phases:

1. **Initial Setup**:
    - Each player specifies the number of ships they will use.
    - Players take turns placing their ships on their board, ensuring valid placement and no overlaps.
2. **Game Loop**:
    - Player 1 selects a coordinate to attack Player 2's board.
    - Player 2's board is updated, and the attack result is displayed (hit, miss, or sunk).
    - The game checks if Player 2 has been defeated (i.e., all ships have been sunk). If yes, Player 1 wins and the game ends.
    - Player 2 then repeats the process by attacking Player 1's board.
    - The game continues in a loop until one player is defeated.

Extensions - Team 3

Development Team

This project has been further developed and maintained by the following team members:

- Ethan Doughty
- Jack Piggot
- Aiden Murphy
- Daniel Bobadilla
- Vy Luw

3x3 Shot Functionality

An extension of the functionality is the **3x3 Shot** feature, which will be activated after every three consecutive hits by a player. This will allow the player to make a 3x3

shot by choosing the center square they would like to shoot, adding a greater probability of sinking their ships.

- **3x3 Shot Rules**:
    - After three successful consecutive hits, the player earns a 3x3 Shot.
    - During the player's next turn, they can target a single coordinate, and fire
    - The Triple Shot can only be used once per three-hit streak.
- **Required Modifications**:
    - Update the `attack()` method in the `Player` class to support a large shot
    - Modify the game loop to handle the activation of the 3x3 Shot after a valid streak.

Opponent AI

The game now includes three levels of AI difficulty, which can be selected by players when playing against the computer. Each difficulty level defines the strategy the AI uses for selecting its attacks.

AI Difficulty Levels

1. **Easy**:
    - The AI selects attack coordinates at random for each turn.
    - There is no strategic targeting, and it does not adjust its behavior based on hits or misses.
2. **Medium**:
    - The AI fires randomly until it lands a hit on a player's ship.
    - Once a hit is made, it targets orthogonally adjacent spaces (up, down, left, right) to find and sink the entire ship.
3. **Hard**:
    - The AI is fully aware of the player's ship positions.
    - It lands a hit on a ship every turn, effectively "cheating" by targeting known ship locations.

AI Ship Placement

- Regardless of the difficulty level, the AI places its ships randomly on the board during setup.
- The placement follows the same rules as human players, ensuring all ships are positioned legally (no overlapping or out-of-bounds ships).