Battleship Game Code Documentation

Project Overview

Battleship is a classic two-player game where each player places ships on a grid and takes turns firing at the opponent's grid, trying to sink their ships. The goal of this project is to create a digital version of this game, adhering to the requirements provided.

Team Members

1. Abir Haque
2. Nikhil Raja
3. Shad Ahmed
4. Rahul Amatapu
5. Gregory Markose

Quick Start and Usage

You may play Battleship by typing ***python owo.py*** in a command line interface. This assumes you are utilizing Python version 3.12 on a Windows 11 machine.

Code Documentation

1. Owo.py

   owo.py is the main file for the entire program. **If one wants to start playing Battleship, they must call owo.py.** The file creates a game object and then initiates the game based on that class.

2. Board.py

   The **board.py** file is critical for managing the Battleship game, which includes board setup, ship placement, hit processing, and visualization. The **Board** class creates two 10x10 grids for the player and opponent, each populated with **Tile** objects that track ship presence and hit status. It keeps an empty list, **self.ships**, of the player's ships. The **validate_and_add_ship** method places ships correctly, preventing overlap and boundary issues, and updates the board with ship IDs. The **perform_hit** method handles hits by marking tiles and checking for ship destruction, whereas the **record_opponent_hit** method updates the opponent's board based on hit results. The'show_board' method displays the board depending on the game phase, using **_show_single_board** to show the player's board during ship placement and **_show_double_board** to show both boards during the game.

3. Game.py

   The **game.py** file controls the Battleship game, including player interactions, game setup, and turn-based gameplay. The **Game** class starts with two player instances, each given a number (1 or 2). The **_game_setup** method prompts players to select the number of ships, validates the input, and then calls **place_ships** to set up both players' boards. The **_get_input_tile** method asks the current player for an attack tile and validates it with is **_valid_tile**. The run method initiates the game loop, alternating turns and processing hits until one player's ships are completely destroyed. It concludes by announcing the current player as the winner. The **game_stats** method is a placeholder for future features such as tracking game statistics.

4. Player.py

   The **player.py** file defines the Player class, which controls the player's board, ship placement, and attack actions in Battleship. The Player class is initialized with a board instance and a player number. It has a **_validate_placement_input** method that ensures whether the ship placement is valid and is within the board boundaries. The **has_lost** method determines whether all the ships have been destroyed, indicating a loss. Players can place ships on their board using the **place_ships** method, which validates input and calls **validate_and_add_ship** for each ship. The **perform_hit** method executes an attack on the provided tile and returns whether a ship was hit, whereas **record_opponent_hit** updates the board with the outcome of an opponent's attack. The **show_board** method displays the board's current status, with different views for game setup and active play.

5. Ship.py

   The **ship.py** file specifies the Ship class, which initializes a list of Tile objects and an ID, which represents the ship's size or type. During initialization, each tile associated with the ship is labeled with its ID. The **get_id** method returns the ship's ID and its length. The **is_destroyed** method checks to see if all of the ship's tiles have been hit, determining whether the ship is completely destroyed. It returns True if all the tiles are hit, and False if not. When a ship is sunk, the **print_destroyed_message** method will print a message informing the user of its size.

6. Tile.py

   The **Tile.py** class allows for the creation of tile objects. For our purposes, we arrange them in a 10x10 grid. There are two public member variables: 1) ship_id (initially None) and 2) is_hit (initially False). There are public getters and setters for both ship_id and is_hit.

7. Util.py

   The **util.py** file contains utility functions for the Battleship game, primarily for board coordinate conversions and input validation. The **get_board_pos** function converts the tile reference (eg. B5, A10 etc.) to board coordinates, returning a tuple containing the 0-based row and column indexes for easier manipulation. The **is_valid_tile** function will validate the tile string to ensure whether it is properly formatted and within board boundaries. It returns True if the tile is valid and False if not, after verifying that the row ranges from A to J and the column number is between 1 and 10.