Test Plan for Battleship

Description of Approach to Testing

Our approach to testing the Battleship system is comprehensive, covering both unit testing with OUnit and manual testing where necessary. The testing suite aims to ensure the correctness and robustness of the core functionalities of the Battleship game.

What We Tested

Automatically Tested by OUnit

1. string_of_cell Function:
    - Tested with all possible cell types to ensure accurate string representation.

2. coordinates Function:
    - Verified correct conversion from string representation of coordinates to (row, column) tuples.

3. create_board Function:
    - Checked for proper creation of boards with different sizes, ensuring all cells are initialized to Water.

4. get_ships Function:
    - Validated the ship distribution for different board sizes.

5. change_state Function:
    - Ensured that the state of the specified cell changes correctly, from Water to Miss.

6. change_to_ship Function:
    - Verified correct placement of ships on the board.

7. hit_ships Function:
    - Tested the identification of hit ships, ensuring that the function returns the correct coordinates of all hit cells of a ship.

8. is_sunk Function:
    - Checked the status of ships to determine if they are completely hit and thus sunk.

9. validate_ship Function:
    - Tested the validity of ship placements, ensuring they are within bounds, not overlapping, and correctly oriented.

Manually Tested

1. print_grid Function:
    - Manually verified the correct visual representation of the board, including ship placements and hits.

2. print_their_board Function:
    - Checked the opponent's view of the board, ensuring hidden cells and hits are correctly displayed.

3. num_ships_sunk Function:
    - Verified the correct count of sunk ships after various hit scenarios.

4. set_ships Function:
    - Ensured that ships are correctly placed according to game rules, without manual overlap or out-of-bound issues.

Modules Tested by OUnit and Development of Test Cases

Grid Module:

- string_of_cell: Developed using black-box testing to cover all possible cell states.
- coordinates: Black-box testing to ensure all valid input strings map correctly to tuples.
- create_board: Glass-box testing to confirm that the board creation logic initializes cells correctly.
- get_ships: Black-box testing with a variety of board sizes to ensure correct ship distributions.

- change_state: Glass-box testing to verify state transitions in different scenarios.
- change_to_ship: Black-box testing to ensure ships are placed correctly based on input parameters.
- hit_ships: Black-box testing to validate the correct identification of hit ships.
- is_sunk: Black-box testing to confirm the accurate determination of ship status.
- validate_ship: Black-box testing to ensure valid ship placements and identify invalid ones.

Argument for Demonstrating Correctness of the System

The testing approach taken demonstrates the correctness of the system for the following reasons:

1. Comprehensive Coverage:

The test suite covers all essential functions within the Grid module, ensuring that individual components operate correctly.

2. Combination of Black-box and Glass-box Testing:

Using both black-box and glass-box testing strategies ensures that the system is tested from both an external and internal perspective, covering expected and edge case scenarios.

3. Automatic and Manual Testing:

By combining OUnit tests for automated verification and manual tests for visual and complex scenario validation, we ensure the system works correctly in both isolated functions and integrated gameplay.

4. Diverse Test Scenarios:

The test cases cover a range of scenarios including normal, boundary, and invalid inputs, ensuring the robustness and fault tolerance of the system.