**Documentation for Unit Tests**

**Purpose**

The purpose of this documentation is to outline the testing strategies, detailed test cases, and results for the unit and integration tests implemented using Google Test for the Board and MiniMaxAgent classes. This documentation serves to ensure thorough testing coverage, identify any defects, and verify that the system meets its requirements.

**Test Strategy**

Our test strategy encompasses:

* **Unit Tests**: Testing individual components (methods) of the classes Board and MiniMaxAgent to ensure they function correctly in isolation.
* **Integration Tests**: Testing the interaction between different components to ensure they work together as intended.

**Test Coverage**

The following classes and functions are covered by the tests:

**BoardTest**

**Test: boardAccessValid**

This test verifies the correctness of accessing the board after setting player inputs.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs at specific positions.
3. Check that the inputs are correctly set.

**Assertions:**

* EXPECT\_EQ(BoardMarks::X, board.at(1, 1));  
  *Expected Result:* The board position (1, 1) should contain 'X'.
* EXPECT\_EQ(BoardMarks::O, board.at(1, 2));  
  *Expected Result:* The board position (1, 2) should contain 'O'.
* EXPECT\_EQ(BoardMarks::Empty, board.at(2, 2));  
  *Expected Result:* The board position (2, 2) should be empty.

**Test: resetBoard**

This test verifies the reset functionality of the board.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs at specific positions.
3. Reset the board.
4. Ensure all positions are empty.

**Assertions:**

* EXPECT\_EQ(BoardMarks::Empty, board.at(0, 0));  
  *Expected Result:* The board position (0, 0) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(0, 1));  
  *Expected Result:* The board position (0, 1) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(0, 2));  
  *Expected Result:* The board position (0, 2) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(1, 0));  
  *Expected Result:* The board position (1, 0) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(1, 1));  
  *Expected Result:* The board position (1, 1) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(1, 2));  
  *Expected Result:* The board position (1, 2) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(2, 0));  
  *Expected Result:* The board position (2, 0) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(2, 1));  
  *Expected Result:* The board position (2, 1) should be empty.
* EXPECT\_EQ(BoardMarks::Empty, board.at(2, 2));  
  *Expected Result:* The board position (2, 2) should be empty.

**Test: VerticalWins**

This test verifies the detection of vertical wins.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs to create vertical wins for both X and O.
3. Evaluate the board for correct win detection.

**Assertions:**

* EXPECT\_EQ(BoardState::XWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'X' as the winner vertically.
* EXPECT\_EQ(BoardState::OWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'O' as the winner vertically.

**Test: HorizontalWins**

This test verifies the detection of horizontal wins.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs to create horizontal wins for both X and O.
3. Evaluate the board for correct win detection.

**Assertions:**

* EXPECT\_EQ(BoardState::XWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'X' as the winner horizontally.
* EXPECT\_EQ(BoardState::OWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'O' as the winner horizontally.

**Test: DiagonalWins**

This test verifies the detection of diagonal wins.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs to create diagonal wins for both X and O.
3. Evaluate the board for correct win detection.

**Assertions:**

* EXPECT\_EQ(BoardState::XWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'X' as the winner diagonally.
* EXPECT\_EQ(BoardState::OWins, board.evaluateBoard());  
  *Expected Result:* The board should detect 'O' as the winner diagonally.

**Test: Tie3x3**

This test verifies the detection of a tie game.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs to create tie scenarios.
3. Evaluate the board for correct tie detection.

**Assertions:**

* EXPECT\_EQ(BoardState::Tie, board.evaluateBoard());  
  *Expected Result:* The board should detect a tie game.

**MiniMaxAgentTest**

**Test: PlayerStartAIWin**

This test verifies the MiniMaxAgent's ability to win when the player starts first.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs.
3. Let the AI play its turn.
4. Evaluate the board for AI's win.

**Assertions:**

* EXPECT\_EQ(BoardState::OWins, board.evaluateBoard());  
  *Expected Result:* The AI should win ('O' wins).
* EXPECT\_EQ(BoardState::XWins, board.evaluateBoard());  
  *Expected Result:* The player should win ('X' wins).

**Test: AIStartAIWin**

This test verifies the MiniMaxAgent's ability to win when the AI starts first.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs.
3. Let the AI play its turn.
4. Evaluate the board for AI's win.

**Assertions:**

* EXPECT\_EQ(BoardState::XWins, board.evaluateBoard());  
  *Expected Result:* The AI should win ('X' wins).

**Test: PlayerStartAIBlock**

This test verifies the MiniMaxAgent's ability to block the player when the player starts first.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs.
3. Let the AI play its turn.
4. Evaluate the board for AI's block.

**Assertions:**

* EXPECT\_EQ(BoardMarks::O, board.at(2, 2));  
  *Expected Result:* The AI should place 'O' at position (2, 2).
* EXPECT\_EQ(BoardMarks::O, board.at(0, 2));  
  *Expected Result:* The AI should place 'O' at position (0, 2).
* EXPECT\_EQ(BoardMarks::O, board.at(1, 0));  
  *Expected Result:* The AI should place 'O' at position (1, 0).

**Test: AIStartAIBlock**

This test verifies the MiniMaxAgent's ability to block the player when the AI starts first.

**Operations performed:**

1. Initialize a 3x3 board.
2. Set player inputs.
3. Let the AI play its turn.
4. Evaluate the board for AI's block.

**Assertions:**

* EXPECT\_EQ(BoardMarks::X, board.at(1, 1));  
  *Expected Result:* The AI should place 'X' at position (1, 1).
* EXPECT\_EQ(BoardMarks::X, board.at(1, 2));  
  *Expected Result:* The AI should place 'X' at position (1, 2).
* EXPECT\_EQ(BoardMarks::X, board.at(1, 0));  
  *Expected Result:* The AI should place 'X' at position (1, 0).

**Test Results**

* All tests were executed using Google Test framework. The tests passed successfully, indicating that the functionalities of Board, MiniMaxAgent classes, as well as the utility functions, work as expected. The following is a summary of the test results:

| **Test Case** | **Result** |
| --- | --- |
| boardAccessValid | Pass |
| resetBoard | Pass |
| VerticalWins | Pass |
| HorizontalWins | Pass |
| CheckTie | Pass |
| DiagonalWins | Pass |
| PlayerStartAIWin | Pass |
| AIStartAIWin | Pass |
| AIStartAIBlock | Pass |
| PlayerStartAIBlock | Pass |

# References

|  |  |
| --- | --- |
| [1] | IEEE, "829-2008 - IEEE Standard for Software and System Test Documentation," IEEE, 2008. |
| [2] | "ChatGBT," [Online]. Available: https://chat.openai.com/. |