Documentation of Testing Strategies and Results for Tic-Tac-Toe Game

In this documentation, we provide a comprehensive overview of the testing strategies employed for the Tic-Tac-Toe game using Google Test framework. The tests are categorized into unit tests and integration tests, covering various functionalities and scenarios of the game.

A. Unit Tests

Unit tests focus on testing individual units (functions and components) of the Tic-Tac-Toe game in isolation. Each unit test is independent and targets specific functionalities.

Initialization Tests

1. **InitArray Test**
   This Tests the initialization of the game board (initArray function). It Verifies that the board is correctly initialized with empty spaces.

Game Logic Tests

2. **DrawBoard Test**
   This tests the drawing of the board (drawBoard function). It Compares the actual output of drawBoard with expected formatted output strings.
3. **ChoosePlayMode Tests**
   This tests the play mode selection (choose_play_mode function). It Simulates user input and verifies that the correct mode (0 for AI vs. AI, 1 for Player vs. AI) is returned. It handles invalid input scenarios to ensure robustness.
4. **GetTwoPlayerSymbols Test**
   This tests the assignment of player symbols (get_two_player_symbols function). It Simulates user input for symbol selection and verifies the assignment of symbols to players.
5. **CheckWin Tests**
   This tests the win condition checking (checkWin function). This has many Tests various winning combinations (rows, columns, diagonals) and tie scenarios.
6. **ExitGame Test**
   This tests the game exit message (exitGame function). It Verifies the correct message based on game outcome (win, lose, tie).
7. **CheckWin_Two_Players Test**
   This tests win condition checking for two players (checkWin_Two_Players function). It is Similar to checkWin but considers symbols for both players.

AI Strategy Tests

8. **MiniMax Test**
   This tests the AI move calculation using MiniMax algorithm (miniMax function).It validates the AI's decision-making against expected outcomes on different board states.
9. **GetRandomMove Test**
   This tests the randomness and functionality of generating a random move (getRandomMove function). It verifies the change in empty spaces count after making a random move.

B. Integration Tests

Integration tests focus on testing interactions between various units or components of the Tic-Tac-Toe game. These tests ensure that different parts of the game work together as expected.

Game Initialization and Play Tests

1. **TestGameInitialization**
   This tests the overall game initialization (initArray function). It verifies that the board is correctly initialized before any moves.

2. **TestTwoPlayerMode**
   This tests the two-player mode functionality. It Simulates a sequence of moves between two players and verifies win conditions. (We will simulate a full game that will lead to win of them)
3. **TestPlayer2Wins**
   This tests scenarios where Player 2 wins. It Validates the game state after Player 2 makes a winning move.
4. **TestDrawScenario**
   This tests the scenario where the game ends in a draw. It simulates a full game leading to a draw and verifies the game outcome.
5. **TestAIMode Tests**
   This tests different AI difficulty levels (Easy, Intermediate, Hard). It verifies AI behavior and ensures the AI makes appropriate moves based on the selected difficulty level.

Testing Strategy Summary

The tests cover various aspects of the Tic-Tac-Toe game, including board initialization, player input handling, game logic (win conditions and tie scenarios), AI decision-making, and overall game flow.

Each test case uses assertions (EXPECT_EQ) to validate expected outcomes against actual results.

Input Simulation: For functions requiring user input (choose_play_mode, get_two_player_symbols), std::istringstream is used to simulate user input via cin.rdbuf() manipulation.

Output Capture: Output from functions like drawBoard and exitGame is captured using testing::internal::CaptureStdout() for validation against expected formatted strings.

Robustness: Tests handle edge cases and invalid inputs to ensure robustness and prevent unexpected behavior.

Conclusion

The Unit and integration tests ensures that the Tic-Tac-Toe game functions correctly across different scenarios and user interactions. The use of Google Test framework facilitates automated testing, providing confidence in the stability and correctness of the game implementation. By following these testing strategies, developers can maintain and extend the game with easily, knowing that new changes are verified using many of tests.

From these testing strategies and results for the Tic-Tac-Toe game, The are providing transparency into the testing process and by this way we ensure that we have high-quality software.