Testing Documentation

For

Advanced Tic Tac Toe

Prepared by TicTacTactics

Submitted to: Dr. Omar Nasr

**Table of Contents**

[1. Overview 1](#_Toc170228478)

[2. Test Plan 1](#_Toc170228479)

[2.1 Objectives 1](#_Toc170228480)

[2.2 Scope 1](#_Toc170228481)

[3. Test Environment 1](#_Toc170228482)

[4. Test Cases 2](#_Toc170228483)

[4.1 Initialize Board 2](#_Toc170228484)

[4.2 Make Valid Move 2](#_Toc170228485)

[4.3 Make Invalid Move 3](#_Toc170228486)

[4.4 No Move After Win 3](#_Toc170228487)

[4.5 Check Horizontal Win 4](#_Toc170228488)

[4.6 Check Vertical Win 4](#_Toc170228489)

[4.7 Check Diagonal Win 5](#_Toc170228490)

[4.8 Check Lose 5](#_Toc170228491)

[4.9 Check Draw 6](#_Toc170228492)

[4.10 Easy AI Move 6](#_Toc170228493)

[4.11 Medium AI Move 7](#_Toc170228494)

[4.12 Hard AI Move 8](#_Toc170228495)

[4.13 Medium AI Avoid Loss 9](#_Toc170228496)

[4.14 Hard AI Win 9](#_Toc170228497)

# Overview

*This document outlines the testing strategy, objectives, scope, test cases, and expected outcomes for the Tic-Tac-Toe application. The tests are implemented using the Google Test framework.*

# Test Plan

## Objectives

* *Ensure the Tic-Tac-Toe board initializes correctly.*
* *Verify valid and invalid moves.*
* *Confirm that no moves are allowed after a win.*
* *Validate the detection of horizontal, vertical, and diagonal wins.*
* *Check the handling of lose and draw scenarios.*

## Scope

* *Unit tests for the core functionalities of the Tic-Tac-Toe game.*
* *Focus on board initialization, move validation, and win/lose/draw conditions.*

# Test Environment

* *Google Test framework*
* *Development environment with the necessary C++ compiler and libraries.*

# Test Cases

## Initialize Board

***Description****: Check if the Tic-Tac-Toe board initializes correctly with the default values.*

***Test Steps****: Create an object “game” from class “TicTacToe”.*

***Expected Outcome****: Each cell in the 3x3 board should contain the values '1' to '9'.*

TEST(TicTacToeTest, InitializeBoard) {

TicTacToe game;

int m = 1;

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

EXPECT\_EQ(game.getBoardValue(i, j), '0' + (m++));

}

}

}

## Make Valid Move

***Description****: Verify that a valid move is correctly processed.*

***Test Steps****:* *Make 'O' move at (2, 1).*

***Expected Outcome****: The move should be marked as "Valid move", and the board should reflect the new value.*

TEST(TicTacToeTest, MakeValidMove) {

TicTacToe game;

EXPECT\_EQ(game.makeMove(2, 1, 'O'), "Valid move");

EXPECT\_EQ(game.getBoardValue(2, 1), 'O');

}

## Make Invalid Move

***Description****: Verify that an invalid move is correctly handled.*

***Test Steps****:* *Make 'X' move at (1, 2). Then attempt to make another 'X' move at the same position (1, 2).*

***Expected Outcome****: The move should be marked as "Invalid move".*

TEST(TicTacToeTest, MakeInvalidMove) {

TicTacToe game;

game.makeMove(1, 2, 'X');

EXPECT\_EQ(game.makeMove(1, 2, 'X'), "Invalid move");

}

## No Move After Win

***Description****: Check if the game prevents moves after a win condition is met.*

***Test Steps****: Make 3 'X' moves at (0, 0), (0, 1), (0, 2). Then attempt to make other moves.*

***Expected Outcome****: Any move after a win should be marked as "Invalid move".*

TEST(TicTacToeTest, NoMoveAfterWin) {

TicTacToe game;

game.makeMove(0, 0, 'X');

game.makeMove(0, 1, 'X');

game.makeMove(0, 2, 'X');

EXPECT\_EQ(game.makeMove(1, 1, 'O'),"Invalid move");

EXPECT\_EQ(game.makeMove(1, 2, 'X'),"Invalid move");

}

## Check Horizontal Win

***Description****: Verify detection of a horizontal win.*

***Test Steps****: Make 3 'X' moves at (0, 0), (0, 1), (0, 2).*

***Expected Outcome****: The winning condition should be true.*

TEST(TicTacToeTest, CheckHorizontalWin) {

TicTacToe game;

game.makeMove(0, 0, 'X');

game.makeMove(0, 1, 'X');

game.makeMove(0, 2, 'X');

EXPECT\_TRUE(game.checkWin());

}

## Check Vertical Win

***Description****: Verify detection of a vertical win.*

***Test Steps****: Make 3 'X' moves at (0, 1), (1, 1), (2, 1).*

***Expected Outcome****: The winning condition should be true.*

TEST(TicTacToeTest, CheckVerticalWin) {

TicTacToe game;

game.makeMove(0, 1, 'X');

game.makeMove(1, 1, 'X');

game.makeMove(2, 1, 'X');

EXPECT\_TRUE(game.checkWin());

}

## Check Diagonal Win

***Description****: Verify detection of a diagonal win.*

***Test Steps****: Make 3 'O' moves at (0, 0), (1, 1), (2, 2).*

***Expected Outcome****: The winning condition should be true.*

TEST(TicTacToeTest, CheckDiagonalWin) {

TicTacToe game;

game.makeMove(0, 0, 'O');

game.makeMove(1, 1, 'O');

game.makeMove(2, 2, 'O');

EXPECT\_TRUE(game.checkWin());

}

## Check Lose

***Description****: Verify that the game correctly identifies a non-winning condition.*

***Test Steps****: Make 3 'O' moves at (0, 0), (0, 1), (2, 2).*

***Expected Outcome****: The winning condition should be false.*

TEST(TicTacToeTest, CheckLose) {

TicTacToe game;

game.makeMove(0, 0, 'O');

game.makeMove(0, 1, 'O');

game.makeMove(2, 2, 'O');

EXPECT\_FALSE(game.checkWin());

}

## Check Draw

***Description****: Verify detection of a draw condition.*

***Test Steps****: Make 9 moves to fill the board without a win.*

***Expected Outcome****: The game should recognize a draw.*

TEST(TicTacToeTest, CheckDraw) {

TicTacToe game;

game.makeMove(0, 0, 'X'); game.makeMove(0, 1, 'X'); game.makeMove(0, 2, 'O');

game.makeMove(1, 0, 'O'); game.makeMove(1, 1, 'O'); game.makeMove(1, 2, 'X');

game.makeMove(2, 0, 'X'); game.makeMove(2, 1, 'O'); game.makeMove(2, 2, 'O');

EXPECT\_TRUE(game.checkDraw());

}

## Easy AI Move

***Description****: Verify that the easy AI makes a move after the player's initial move.*

***Test Steps****: Make initial move by 'X' at (0, 0).Then trigger easy AI ('O') move.*

***Expected Outcome****: The easy AI ('O') makes a move on the board after the player's move ('X'). Specifically, at least one cell on the board contains 'O', indicating the AI's move.*

TEST(TicTacToeTest, EasyAIMove) {

TicTacToe game;

game.makeMove(0, 0, 'X'); // Assuming initial move is by 'X', and AI plays as 'O'

game.computer\_turn\_easy();

bool aiMoved = false; // Verify AI made a move

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

if (game.getBoardValue(i, j) == 'O') {

aiMoved = true;

break;

}

}

if (aiMoved) break;

}

EXPECT\_TRUE(aiMoved);

}

## Medium AI Move

***Description****: Verify that the medium AI makes a move after the player's initial move.*

***Test Steps****: Make initial move by 'X' at (0, 0).Then trigger medium AI ('O') move.*

***Expected Outcome****: The medium AI ('O') makes a move on the board after the player's move ('X'). Specifically, at least one cell on the board contains 'O', indicating the AI's move.*

TEST(TicTacToeTest, MediumAIMove) {

TicTacToe game;

// Assuming initial move is by 'X', and AI plays as 'O'

game.makeMove(0, 0, 'X');

game.computer\_turn\_medium();

bool aiMoved = false;

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

if (game.getBoardValue(i, j) == 'O') {

aiMoved = true;

break;

}

}

if (aiMoved) break;

}

EXPECT\_TRUE(aiMoved);

}

## Hard AI Move

***Description****: Verify that the hard AI makes a move after the player's initial move.*

***Test Steps****: Make initial move by 'X' at (0, 0).Then trigger hard AI ('O') move.*

***Expected Outcome****: The hard AI ('O') makes a move on the board after the player's move ('X'). Specifically, at least one cell on the board contains 'O', indicating the AI's move.*

TEST(TicTacToeTest, HardAIMove) {

TicTacToe game;

// Assuming initial move is by 'X', and AI plays as 'O'

game.makeMove(0, 0, 'X');

game.computer\_turn\_hard();

bool aiMoved = false;

for (int i = 0; i < 3; ++i) {

for (int j = 0; j < 3; ++j) {

if (game.getBoardValue(i, j) == 'O') {

aiMoved = true;

break;

}

}

if (aiMoved) break;

}

EXPECT\_TRUE(aiMoved);

}

## Medium AI Avoid Loss

***Description****: Verify that the medium AI avoids a losing condition by blocking the player's potential winning move.*

***Test Steps****: Make 2 'X' moves at (0, 0) and (0, 1). Then trigger medium AI ('O') move.*

***Expected Outcome****:* *The medium AI ('O') correctly blocks the player's potential winning move* *by occupying the specified board position (0, 2) with an 'O' move.*

TEST(TicTacToeTest, MediumAIAvoidLoss) {

TicTacToe game;

game.makeMove(0, 0, 'X');

game.makeMove(0, 1, 'X');

game.computer\_turn\_medium();

EXPECT\_EQ(game.getBoardValue(0, 2), 'O'); // AI should block the winning move

}

## Hard AI Win

***Description****: Verify that the hard AI will take the chance to win if given the opportunity during its turn.*

***Test Steps****:*

* *Make an ‘X' move at (1, 1) and trigger hard AI to make an 'O' move.*
* *Make another 'X' move at (0, 2) and trigger hard AI again to make an 'O' move.*
* *Make another 'X' move at (1, 2) and trigger hard AI again to make an 'O' move.*

***Expected Outcome****: The hard AI blocks the player's potential winning move by occupying the specified board position (2, 0) with an 'O' move. Then it takes the chance to win by occupying the specified board position (1, 0) with an 'O' move.*

TEST(TicTacToeTest, HardAIWin) {

TicTacToe game;

game.makeMove(1, 1, 'X');

game.computer\_turn\_hard(); // AI plays at (0,0)

game.makeMove(0, 2, 'X');

game.computer\_turn\_hard(); // AI plays at (2,0)

game.makeMove(1, 2, 'X');

game.computer\_turn\_hard(); // AI plays at (1,0)

EXPECT\_TRUE(game.checkWin()); // AI should make the winning move

}