Advanced Tic Tac Toe Game

Test Report

1.0 Overview

purpose: validate all game modes (PvP, AI vs player, Mega tic-tac-toe).

2.0 Testing categories

2.1 PvP tests

2.1.1 valid move execution *move form is (row,col)

Test name	Input	Expected output	Actual results
validMove	Player x moves to (1,1)	Move is placed successfully, board updates with 'X'	Passed

2.1.2 Invalid move rejection

Test name	Input	Expected output	Actual results
PreventOverWrite	Player x moves to (0,0) Player O attempts move at (0,0)	Player X move is **accepted** and the board updates. Then player O move is **rejected**, and the cell still holds 'X'.	Passed
outOfBoundsIndex	Player X attempts to move at (3,3) Player O attempts to move at (-1,-1)	Both moves **rejected**, board **unchanged**	Passed

2.1.3 Win condition detection

Test name	Input	Expected output	Actual results

checkWin	Player X moves to (0,0), (0,1), (0,2)	checkWin('X') == true (X wins)
		checkWin('O') == false (O does not win)
			Passed

2.1.4 tie scenario

Test name	Input	Expected output	Actual results
DetectTie	Board is completely filled with **no winner**	**No win detected** for either player Game correctly identifies a tie	Passed

2.2 AI vs player tests

2.2.1 AI move logic validation

Test name	Input	Expected output	Actual results
validMoves	Instantiate three AI instances: easy, normal and hard. For each AI, call ai.moveAi(&move) .	Each instance should return a move value within the board range (0 to 8) for its respective difficulty level.	Passed

2.2.2 AI response to winning opportunities

Test name	Input	Expected output	Actual results
AiWinningMove	Tree initialized with AI ‘X’ starting first. input board {'X', 'O', 'X'}, {'O', 'X', ' '}, {'O', ' ', ' '}	AI's best move should be index 8 (position (2,2)) to win the game	Passed
AiBlock2movesinadvance	Tree initialized with player ‘X’ starting first. input board {'X', 'O', 'X'}, {' ', ' ', ' '}, {' ', ' ', ' '}	AI's best move should be index 8 (position (1,1)) to prevent ‘X’ from winning	Passed

2.2.3 AI decision-making on blocking opponent moves

Test name	Input	Expected output	Actual results
AiBlockPlayerWinning	Player ‘X’ starts first. input board {'X', 'O', ' '}, {'X', ' ', ' '}, {' ', ' ', ' '}	AI's best move should be index 6 (position (2,0)) to block the win for player X.	Passed

2.3 Mega tic-tac-toe tests

2.3.1 Sub grid win detection *move form is (row,col,subgrid)

Test name	Input	Expected output	Actual results
ValidMove	Player X moves to (1,1,0)	Cell (1,1,0) value is X	Passed
PreventOverWrite	Player X moves to (0,0,0) then player O moves to (0,0,0)	Cell (0,0,0) value is X And O's move fail	Passed
OutOfBoundsIndex	Move to (3, 3, 0) with symbol 'X' Move to (-1, -1, 0) with symbol 'O'	Both moves fail	Passed

2.3.2 Correct sub grid targeting after moves

Test name	Input	Expected output	Actual results
CorrectSubgridTargeting	- Player X moves at (1, 1,0) (next allowed Sub grid becomes 4). - Player O attempts to make move (1,1,0), then (0,0,4)	Player O first attempt fails then the correct move succeeds	Passed

2.3.3 Handling winnings/ties within sub grids

Test name	Input	Expected output	Actual results
CheckSubgridWin	Moves in sub grid 0: (0,0) = 'X', (0,1) = 'X', (0,2) = 'X'	Sub grid 0 win for 'X' and check win for 'O' return false	Passed
DetectSubgridTie	Move in sub grid 0: {'X', 'O', 'X'}, {'X', 'X', 'O'}, {'O', 'X', 'O'}	Check win for both 'X' and 'O' return false And sub grid 0 is detected full	Passed

2.3.4 Overall board win detection

Test name	Input	Expected output	Actual results
MegaBoardWinCondition	Sub grid 0: Moves at (0,0), (0,1), (0,2) set to 'X' Sub grid 1: Moves at (1,0), (1,1), (1,2) set to 'X' Sub grid 2: Moves at (2,0), (2,1), (2,2) set to 'X'	Check Sub grid Win (subgrids, 'X') returns true for subgrids 0, 1, and 2 mega Check Win('X') returns true	Passed
MegaBoardTieCondition	Fill the board with moves to make a tie	Either megaIsFull() or iswinningGridsFull() returns true (confirming the board/subgrids are completely filled/resolved	Passed

2.4 database tests

Test name	Input	Expected output	Actual results

Register And Login User	username = "testuser"
	password = "123456"
		User is registered, getUserId returns valid ID, user can log in	User ID != -1, Auth returns true

Insert Game History And Retrieve	player1 = "Alice", player2 = "Bob", winner = "Draw", board = "XOXOXOXOX"	Game history inserted successfully, returns valid game_id	Game ID > 0
Insert And Load Game Moves	Game_Id from inserted game, moves = { {0,0,'X'}, {0,1,'O'}, {0,2,'X'} }	Moves inserted and retrieved correctly, all values match	Matches original moves

Authenticate With Wrong Password	string username = "user1";
	string correct Password = "correct123";
	string wrong Password = "wrong456";
		Authentication fails with wrong password	Auth returns false

Register Duplicate Username	Username: duplicate, Password: pass (registered twice)	First succeeds, second fails	First = true, Second = false
Authenticate Non Existent User	Username: ghost, Password: nopass	Authentication fails	Auth returns false
Load Moves Empty	Game ID created, no moves inserted	Empty move list	Moves list is empty
Get User_Id Not Found	Username: non_existent_user	ID = -1 (not found)	ID = -1

2.5 Performance tests

2.5.1 Performance tests for AI:

Test name	Input	Expected output	Actual results
initAI_Average	Initialize 100 AI instances (Easy, Medium, Hard, Expert)	Avg execution time < 50 ms, memory < 60 KB per instance	Avg Execution Time: 45.3824 ms, Avg Memory Used: 51436.4 KB
Delete Ai	Create and delete 50 AI instances	Execution time < 40 ms, memory released < 10 KB residual	Execution Time: 35.246 ms, Memory before: 56212 KB, Memory after: 4788 KB, Memory Used: 18446744073709500192 KB (actual release ~51,424 KB)
playMoveAiHard	Simulate 100 moves with Hard AI	Execution time < 1 ms per move, no memory increase	Execution Time: 0 usec, Memory before: 56216 KB, Memory after: 56216 KB, Memory Used: 0 KB
playMoveAiNormal	Simulate 100 moves with Normal AI	Execution Time: < 1 ms per move, no memory increase	Execution Time: 1 usec, Memory before: 56220 KB, Memory after: 56220 KB, Memory Used: 0 KB

playMoveAiEasy	Simulate 100 moves with easy AI	Execution Time: < 1 ms per move, no memory increase	Execution Time: 1 usec, Memory before: 56220 KB, Memory after: 56220 KB, Memory Used: 0 KB
AIcompleteGame	Run 50 complete games with AI (all levels)	Total execution time < 200 ms, memory stable	Execution Time: 1 usec, Memory before: 107852 KB, Memory after: 107852 KB, Memory Used: 0 KB (actual ~166 ms)
StressTestHardMode	Run 10,000 moves with Hard difficulty	Avg time per move < 0.01 ms, memory increase < 10 KB	Execution Time: 47 usec, Avg Time per run: 0.0047 usec, Memory before: 107900 KB, Memory after: 107904 KB, Memory Used: 4 KB
StressTestNormalMode	Run 10,000 moves with Normaldifficulty	Avg time per move < 0.03 ms, memory increase < 10 KB	Execution Time: 237 usec, Avg Time per run: 0.0237 usec, Memory before: 107924 KB, Memory after: 107924 KB, Memory Used: 0 KB
StressTestEasyMode	Run 10,000 moves with Easy difficulty	Avg time per move < 0.03 ms, memory increase < 10 KB	Execution Time: 258 usec, Avg Time per run: 0.0258 usec, Memory before: 107948 KB, Memory after: 107948 KB, Memory Used: 0 KB

2.5.2 Performance tests for Data Base:

Test name	Input	Expected output	Actual results
Register And Authenticate Performance	Register and authenticate 100 users	Avg execution time < 150 ms, memory increase < 50 KB	Avg Execution Time: 45.3824 ms, Avg Memory Used: 51436.4 KB

Insert Game History and Moves Performance	Insert game history and moves for 50 games	Execution time < 1 ms per operation, memory increase < 50 KB	Execution Time: 0 ms, Memory before: 5872 KB, Memory after: 5912 KB, Memory Used: 40 KB
Bulk User Registration	Register 1,000 users with game data	Total execution time < 7,500 ms, memory increase < 100 KB	Execution Time: 7191 ms, Memory before: 5912 KB, Memory after: 5924 KB, Memory Used: 12 KB

2.5.3 Performance tests for Game:

Test name	Input	Expected output	Actual results
Board Operations Profile	1,000 cycles of board init, moves, win checks, display	Avg time per operation < 0.002 ms, memory stable	Avg Init: 0.000033ms, move: 0.000031ms, Win Check: 0.000033ms, Display: 0.001478ms, Total Test Time: 8 ms
Mega Board Operations Profile	500 cycles of mega board init, moves, subgrid/mega win checks	Avg time per operation < 0.001 ms, memory stable	Avg Init: 0.000048ms, Move: 0.000028ms, Subgrid Win: 0.000031ms, Mega Win: 0.000034ms, Total Test Time: 5 ms
Game Simulation	Simulate 100 complete games	Avg time per game < 0.001 ms, games/sec > 1,000,000	Avg Game: 0.000139ms, Games/sec: 7194244.604317, Total Test Time: 0 ms
Stress Test	200 stress cycles with continuous operations	Real time < 0.5 ms, memory < 200 MB, cycles/sec > 500,000	Real: 0.228600ms, CPU: 0.000000ms, Memory: 155.199219MB, CPU%: 0.000000%, Cycles: 200, Cycles/sec: 874890.638670, Avg/cycle: 0.001143ms, Total Test Time: 0 ms

3.0) Summary & Recommendations

3.1. Test Coverage: We have made 41 tests

- The test_output.log indicates that 41 tests were run across 7 test suites (AiTests, PerformanceTestAi, PerformanceTestDatabase, GameStructurePerformanceTest, DatabaseTest, megaBoardTest, BoardTest), all of which passed. This represents the total number of tests executed, covering AI behavior, performance metrics, database operations, and game logic for both standard and mega boards.

3.2. Areas for Improvement: (Any uncovered scenarios?)

The 41 tests focus heavily on functional and performance aspects, such as AI move validation, database transactions, and board operations. However, there are gaps in coverage:

No tests specifically address GUI interactions (e.g., button clicks or window initialization).

User profile management (e.g., stats updates, win streaks) is not tested.

Integration flows (e.g., end-to-end game sessions) are missing.

Authentication edge cases (e.g., special characters, empty inputs) are not included. These uncovered scenarios suggest a need for additional tests to ensure comprehensive validation of the application.

3.3. Next Steps: (Further testing needed before final validation?)

Conduct tests for GUI elements, user profile features, and integration workflows to address the identified gaps.

Expand performance testing to include stress scenarios for CPU, memory, and database under high load.

Perform user acceptance testing with real users to validate usability and edge cases.

Schedule a final review of all test results by early July 2025 to ensure readiness for validation.