

Program Structures and Algorithms MCTS Project Report

Spring 2025

Team members:

Chinnasurya Prasad Vulavala NUID:002056815

Sai Lalith Pulluri NUID:002056829

Github url : https://github.com/chinnasuryaprasad1612/INFO-6205_Project

Abstract

In this project, we implemented Monte Carlo Tree Search (MCTS) to develop AI agents for two games: Tic Tac Toe and Dots and Boxes. The project aimed to explore how MCTS behaves with an increased number of iterations and benchmark its performance across increasing iteration counts. We analyzed how game outcomes evolve as simulations become deeper, ultimately observing optimal and strategic play patterns.

Overview:

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that is used to make game decisions. It creates a search tree dynamically and uses a statistical approach to balance exploration and exploitation. MCTS works exceptionally well for games with vast state spaces.

The algorithm contains four major phases:

- Selection: To balance exploration and exploitation, traverse the tree from the root using the UCT (Upper Confidence Bound for Trees) algorithm.
- Expansion: Create additional child nodes in the tree from unexplored legal movements.
- Simulation: Perform a random or heuristic-guided playout from the new node to the terminal state.
- Backpropagation: Involves updating the win/playout statistics up the tree based on the simulation results.

Implementation:

Heuristics Used(for higher number of iterations)

- In Dots and Boxes, we added heuristics to:
 - Prioritize moves that complete a box.
 - Avoid moves that give away boxes to the opponent (trap detection).

- In Tic Tac Toe, heuristic logic in simulation prioritizes center and corner moves, enhancing strategy quality.

Features Implemented

- Graphical User Interfaces (GUIs) were implemented for both games.
- Users can interactively play against the AI.
- Each game allows players to select the difficulty level, which adjusts the number of iterations and whether heuristics are enabled.
- Support for four difficulty levels: EASY, MEDIUM, HARD, and EXPERT

Rules for the tic-tac-toe game

Tic Tac Toe is a 3×3 grid game with two players: Player 1 (X) and Player 2(O). Players take turns marking empty cells, with X always going first. The goal is to be the first to line up three markers in a horizontal, vertical, or diagonal row. If all cells are filled and neither player wins, the game is a draw. In our simulation, a win is worth one point, a draw is worth half a point, and a defeat is worth zero points.

Dot-and-Box Rules

Dots and Boxes is a turn-based game with a rectangle grid of dots. Players take turns painting horizontal or vertical lines between nearby dots. Completing the fourth edge of a box awards the player one point and another turn. The game ends when no more lines can be drawn. The player who has fulfilled the most boxes wins. A draw happens when both players claim the same amount of boxes.

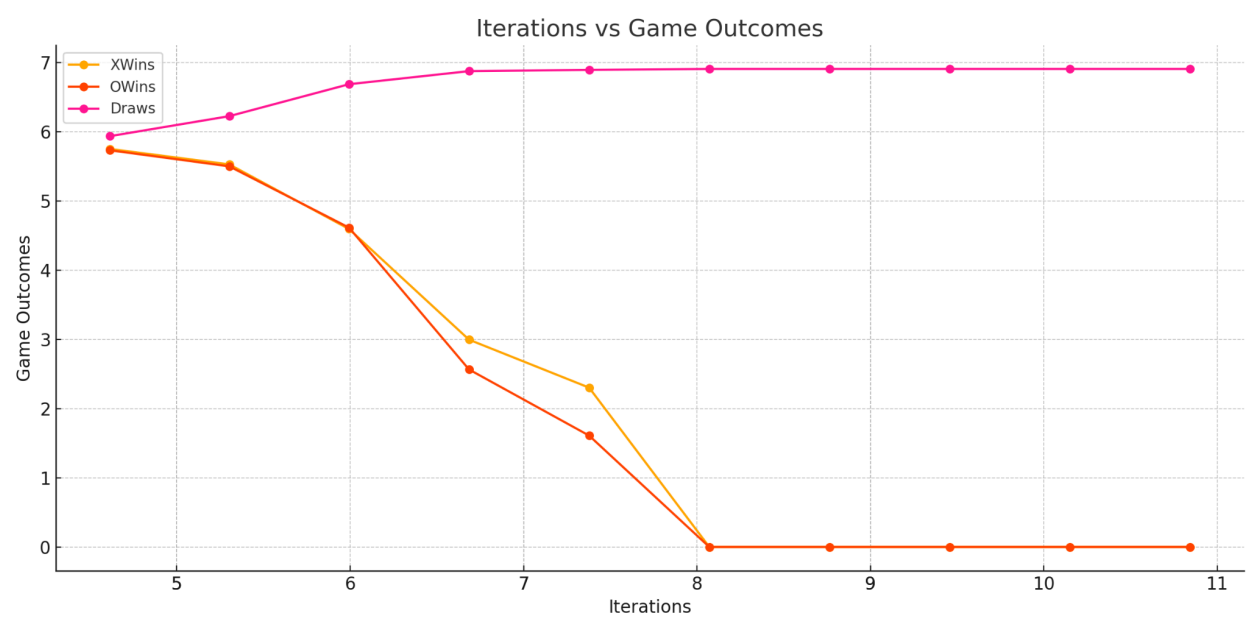
Observations:

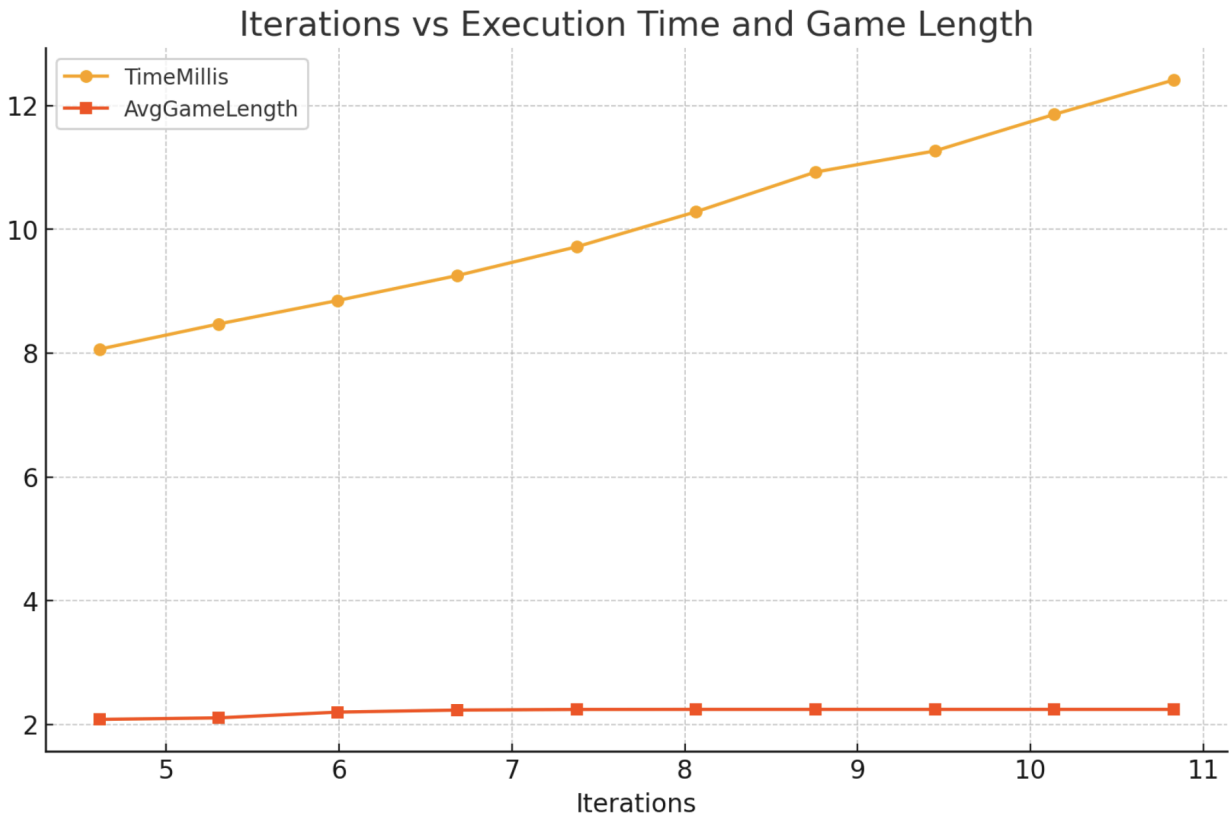
We conducted extensive benchmarking to analyze MCTS behavior across both games. For each configuration, we ran 1000 AI vs AI games using iteration counts from 100 to 51200, doubling each time. Metrics such as player win rates, draw percentages, average game length, and execution time were recorded and log -log graphs were drawn. ($\log(1+x)$ vs $\log(1+y)$ was considered since we have zeroes in our results)

Tic-Tac-Toe Game:

Iterations	Total Games	XWins	OWins	Draws	WinRateX	WinRateO	AvgGameLength	TimeMillis
100	1000	314	308	378	31.4	30.8	6.94	3153
200	1000	251	244	505	25.1	24.4	7.48	4466
400	1000	98	100	802	9.8	10.0	8.5	6691
800	1000	19	12	969	1.9	1.2	8.94	10355
1600	1000	9	4	987	0.9	0.4	8.97	16612
3200	1000	0	0	1000	0.0	0.0	9.0	29151
6400	1000	0	0	1000	0.0	0.0	9.0	47203
12800	1000	0	0	1000	0.0	0.0	9.0	78316
25600	1000	0	0	1000	0.0	0.0	9.0	141457
51200	1000	0	0	1000	0.0	0.0	9.0	245824

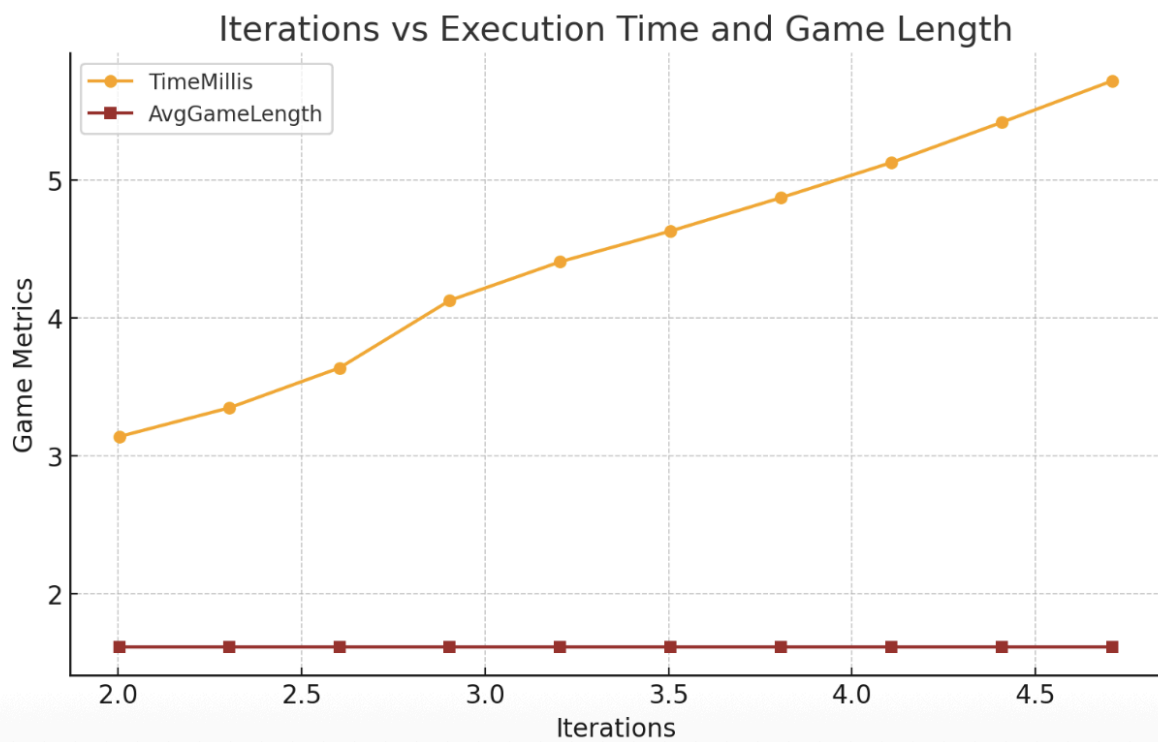
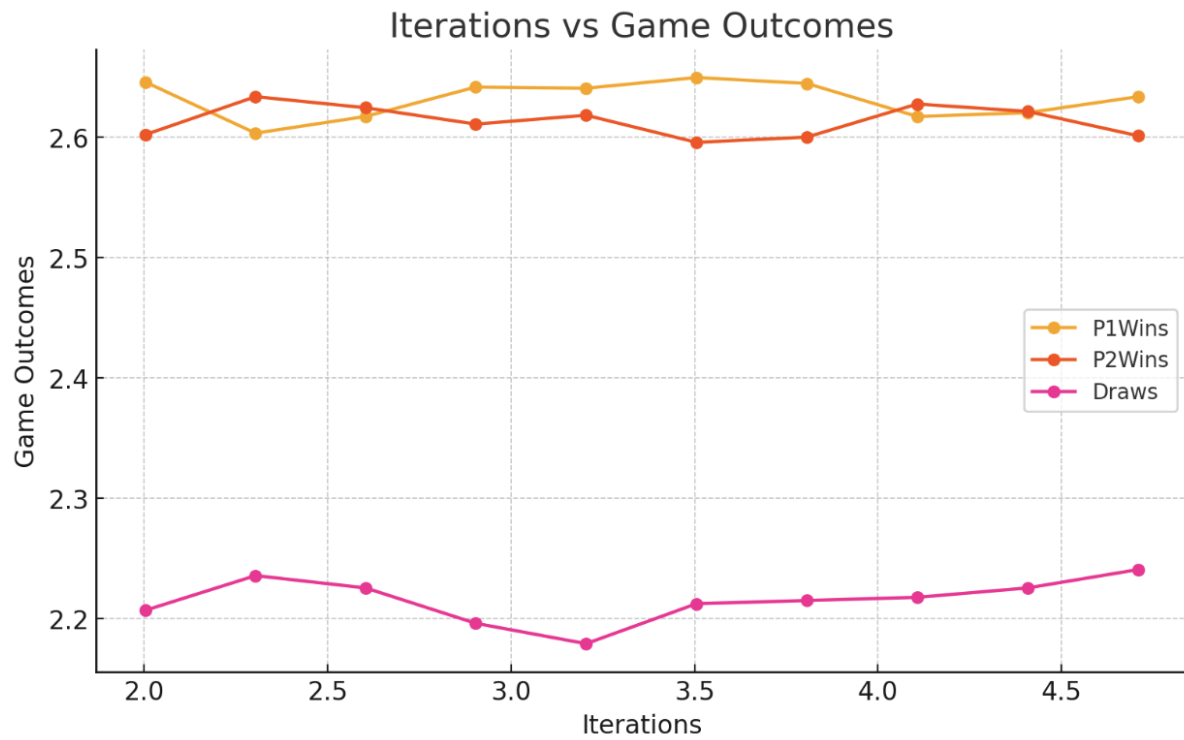
Graphs:





Dots and boxes:

Iterations	TotalGames	P1Wins	P2Wins	Draws	WinRateP1	WinRateP2	AvgGameLength	TimeMillis
100	1000	396	437	167	39.6	43.7	40.0	17180
200	1000	433	417	150	43.3	41.7	40.0	34266
400	1000	434	406	160	43.4	40.6	40.0	77117
800	1000	433	392	175	43.3	39.2	40.0	136892
1600	1000	417	410	173	41.7	41.0	40.0	293294
3200	1000	413	437	150	41.3	43.7	40.0	529142
6400	1000	432	420	148	43.2	42.0	40.0	1048816
12800	1000	426	429	148	42.6	42.9	40.0	1895322
25600	1000	428	427	145	42.8	42.7	40.0	3522315
51200	1000	421	430	145	42.1	43.0	40.0	6932178

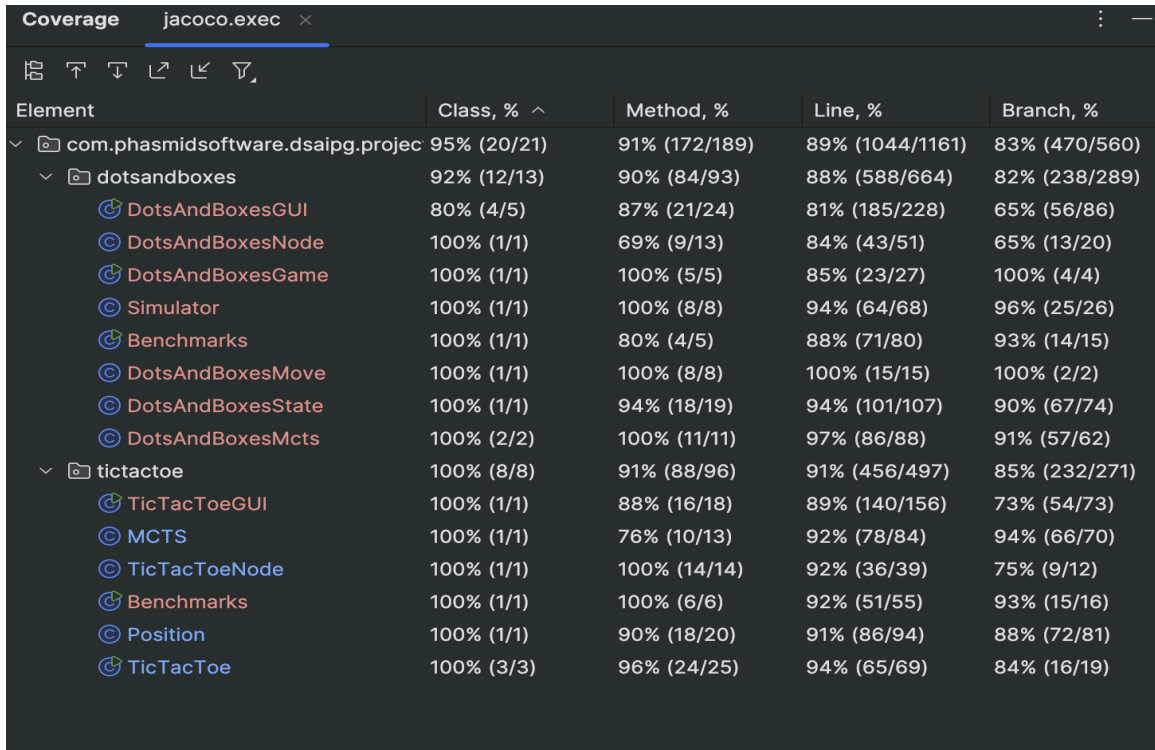


In the case of **Tic-Tac-Toe**, as the number of iterations increased, both players' victory rates fell, while draws increased dramatically. The majority of games ended in draws at 1600 iterations or more, showing convergence toward optimal gameplay. The average game length stabilized at 9 moves, matching the maximum number of moves conceivable in a 3×3 grid. Execution time increased as expected with more simulations, but game duration remained the same, demonstrating that deeper search improves decision quality rather than game structure. These trends were readily seen in the benchmarking graphics. Furthermore, the use of heuristics during simulations at higher iterations, such as prioritizing center and corner plays, preventing urgent threats, and selecting winning moves, helped greatly to achieve optimal play at higher iteration levels.

In contrast, **Dots and Boxes** demonstrated a distinctly different pattern. Benchmarks were conducted on a 5×5 board, resulting in 40 total possible line placements. Due to its cumulative, point-based scoring system, draws remained rare even with high iteration counts. Most games ended in a win for one of the players, with win rates hanging around 43-44% for each player and draws continuously ranging between 15-17%, with only small variations as simulations progressed. The average game length remained set at 40 moves, as determined by the board configuration. Execution time increased with deeper simulations due to the growing search tree and larger branching factor. However, the use of domain-specific heuristics at higher number of iterations such as prioritizing box completion and avoiding trap-creating moves proved crucial in enhancing performance and reducing early-game mistakes. Unlike Tic-Tac-Toe, deeper search did not result in increased draw rates; rather, it led to more strategic, competitive play without significantly changing game outcomes. These findings, supported by visual graphs, demonstrate MCTS's capacity to adapt differently to deterministic victory conditions than point-accumulation games.

Test cases coverage:

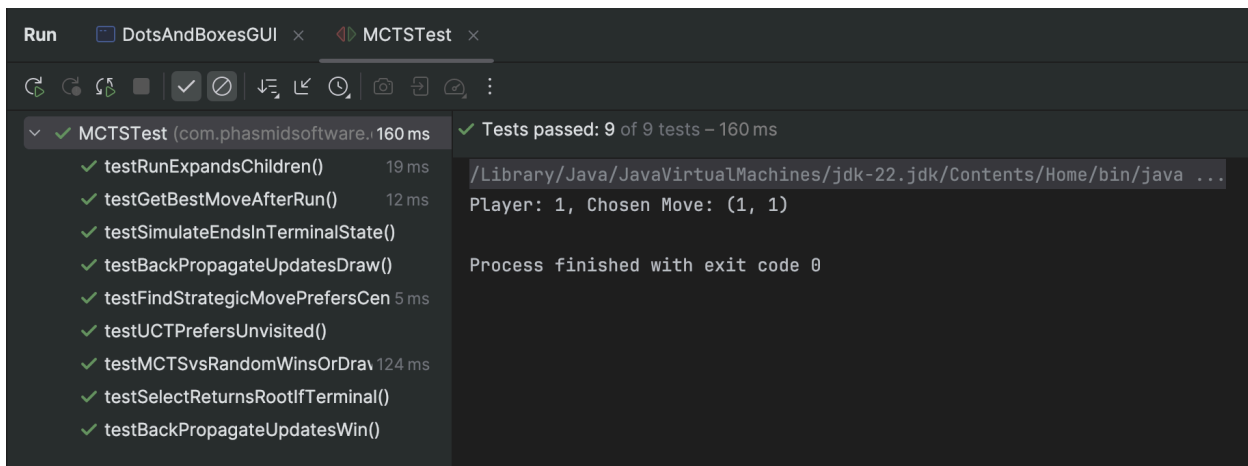
To assure the accuracy and dependability of our implementation, we created rigorous unit tests for both the Tic Tac Toe and Dots and Boxes modules. Our test suite obtained 95% class coverage, 91% method coverage, 89% line coverage, and 83% branch coverage, according to JaCoCo. These high coverage numbers demonstrate the completeness of our testing methodology and the reliability of our MCTS-based game logic, simulation, and GUI components.



The screenshot shows the JaCoCo Coverage report for the jacoco.exec file. The report is displayed in a table with columns for Element, Class, %, Method, %, Line, %, and Branch, %. The data is organized into a tree structure, showing coverage for the entire project and its sub-modules.

Element	Class, % ^	Method, %	Line, %	Branch, %
com.phasmidsoftware.dsaipg.projec	95% (20/21)	91% (172/189)	89% (1044/1161)	83% (470/560)
dotsandboxes	92% (12/13)	90% (84/93)	88% (588/664)	82% (238/289)
DotsAndBoxesGUI	80% (4/5)	87% (21/24)	81% (185/228)	65% (56/86)
DotsAndBoxesNode	100% (1/1)	69% (9/13)	84% (43/51)	65% (13/20)
DotsAndBoxesGame	100% (1/1)	100% (5/5)	85% (23/27)	100% (4/4)
Simulator	100% (1/1)	100% (8/8)	94% (64/68)	96% (25/26)
Benchmarks	100% (1/1)	80% (4/5)	88% (71/80)	93% (14/15)
DotsAndBoxesMove	100% (1/1)	100% (8/8)	100% (15/15)	100% (2/2)
DotsAndBoxesState	100% (1/1)	94% (18/19)	94% (101/107)	90% (67/74)
DotsAndBoxesMcts	100% (2/2)	100% (11/11)	97% (86/88)	91% (57/62)
tictactoe	100% (8/8)	91% (88/96)	91% (456/497)	85% (232/271)
TicTacToeGUI	100% (1/1)	88% (16/18)	89% (140/156)	73% (54/73)
MCTS	100% (1/1)	76% (10/13)	92% (78/84)	94% (66/70)
TicTacToeNode	100% (1/1)	100% (14/14)	92% (36/39)	75% (9/12)
Benchmarks	100% (1/1)	100% (6/6)	92% (51/55)	93% (15/16)
Position	100% (1/1)	90% (18/20)	91% (86/94)	88% (72/81)
TicTacToe	100% (3/3)	96% (24/25)	94% (65/69)	84% (16/19)

Tic Tac Toe:



The screenshot shows the Run console output for the MCTSTest class. The output displays the results of 9 tests, all of which passed. The tests are listed with their names and execution times. The output also shows the Java command used to run the tests and the exit code of the process.

Test Name	Execution Time
MCTSTest (com.phasmidsoftware..)	160 ms
testRunExpandsChildren()	19 ms
testGetBestMoveAfterRun()	12 ms
testSimulateEndsInTerminalState()	
testBackPropagateUpdatesDraw()	
testFindStrategicMovePrefersCen	5 ms
testUCTPrefersUnvisited()	
testMCTSvsRandomWinsOrDraw	124 ms
testSelectReturnsRootIfTerminal()	
testBackPropagateUpdatesWin()	

Tests passed: 9 of 9 tests – 160 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Player: 1, Chosen Move: (1, 1)

Process finished with exit code 0

Run DotsAndBoxesGUI **PositionTest**

✓ **PositionTest** (com.phasmidsoftware. 11 ms) **Tests passed: 16 of 16 tests – 11 ms**

- ✓ testReflectVertically() 7 ms
- ✓ testValidMoveUpdatesBoard()
- ✓ testMoveThrowsOnFullBoard() 1 ms
- ✓ testParseCell()
- ✓ testFullBoard()
- ✓ testWinnerDetectionRow()
- ✓ testRenderToString()
- ✓ testWinnerDetectionDiagonal()
- ✓ testThreeInARow()
- ✓ testProjectRowAndCol() 1 ms
- ✓ testMovesListSizeAndOrder()
- ✓ testMoveThrowsOnOccupied() 2 ms
- ✓ testProjectDiagonals()
- ✓ testWinnerEmpty()
- ✓ testWinnerDetectionColumn()
- ✓ testReflectHorizontally()

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0

Run DotsAndBoxesGUI **TicTacToeGUITest**

✓ **TicTacToeGUITest** (com.phas 1 sec 247 ms) **Tests passed: 12 of 12 tests – 1 sec 247 ms**

- ✓ testHighlightWinningCells_Color 762 ms
- ✓ testResetButton() 136 ms
- ✓ testSetDifficultyToMedium() 25 ms
- ✓ testHighlightWinningCells_RowW 49 ms
- ✓ testUpdateDifficulty() 31 ms
- ✓ testMakeAllMoveDoesNotCrash() 18 ms
- ✓ testHighlightWinningCells_DiagonalWin() 28 ms
- ✓ testHandlePlayerMoveAndMakeI 93 ms
- ✓ testHighlightWinningCells_AntiDi 24 ms
- ✓ testConstructor() 17 ms
- ✓ testSetDifficultyToEasy() 17 ms
- ✓ testSetDifficultyToHard() 47 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

2025-04-20 17:26:35.510 java[37334:15510929] +[IMKClient subclass]: chose IMKClient_Modern

2025-04-20 17:26:35.510 java[37334:15510929] +[IMKInputSession subclass]: chose IMKInputSession_Modern

Process finished with exit code 0

Run DotsAndBoxesGUI x TicTacToeNodeTest x

✓ TicTacToeNodeTest (com.phasmidsoft: 6 ms) ✓ Tests passed: 10 of 10 tests – 6 ms

- ✓ testAddMultipleChildrenUniqueness 3 ms
- ✓ addChild 1 ms
- ✓ testChildrenAfterAddChild 0 ms
- ✓ state 0 ms
- ✓ white 0 ms
- ✓ backPropagate 0 ms
- ✓ testIsLeafWhenNoChildren 0 ms
- ✓ testMultipleBackpropagation 1 ms
- ✓ children 0 ms
- ✓ winsAndPlayouts 1 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0

Run DotsAndBoxesGUI x TicTacToeTest x

✓ TicTacToeTest (com.phasmidsoft: 53 ms) ✓ Tests passed: 13 of 13 tests – 53 ms

- ✓ testSimulateGameOutput() 48 ms
- ✓ testConstructorWithSeed() 1 ms
- ✓ testGameMethod() 1 ms
- ✓ testTicTacToeStateToString() 3 ms
- ✓ testRunGameWithIterations() 3 ms
- ✓ testConstructorWithRandom() 1 ms
- ✓ testStartReturnsValidState() 1 ms
- ✓ testTicTacToeMoveFields() 1 ms
- ✓ testOpener()
- ✓ testTicTacToeStateMethods()
- ✓ testTicTacToeStateConstructorWithStart
- ✓ testRunGameDeterministic()
- ✓ testDefaultConstructor()

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0

Dots and Boxes:

Run DotsAndBoxesGUI x DotsAndBoxesGameTest x

✓ DotsAndBoxesGameTest (com.phasmidsc 50 sec 624 ms) ✓ Tests passed: 2 of 2 tests – 50 sec 624 ms

- ✓ testMain_CustomBoardAi 30 sec 352 ms
- ✓ testMain_DefaultArgs 20 sec 272 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0

Run DotsAndBoxesGUI x DotsAndBoxesMctsTest x

✓ Tests passed: 11 of 11 tests – 11 ms

- ✓ testHardDifficultyConstructor 2 ms
- ✓ testMediumDifficultyConstructor 8 ms
- ✓ testFindSafeMoveFallback 1 ms
- ✓ testSelectFullyExpandedNode 0 ms
- ✓ testFindMoveToStateGetsCovered 0 ms
- ✓ testFallbackToFirstAvailableMove 0 ms
- ✓ testFindBoxCompletingMove 0 ms
- ✓ testExpertDifficultyConstructor 0 ms
- ✓ testCustomConstructor 0 ms
- ✓ testIsFullyExpandedTrue 0 ms
- ✓ testEasyDifficultyConstructor 0 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

Process finished with exit code 0

Run DotsAndBoxesGUI x SimulatorTest x

✓ SimulatorTest (com.phasmidsc 6 sec 77 ms) ✓ Tests passed: 4 of 4 tests – 6 sec 77 ms

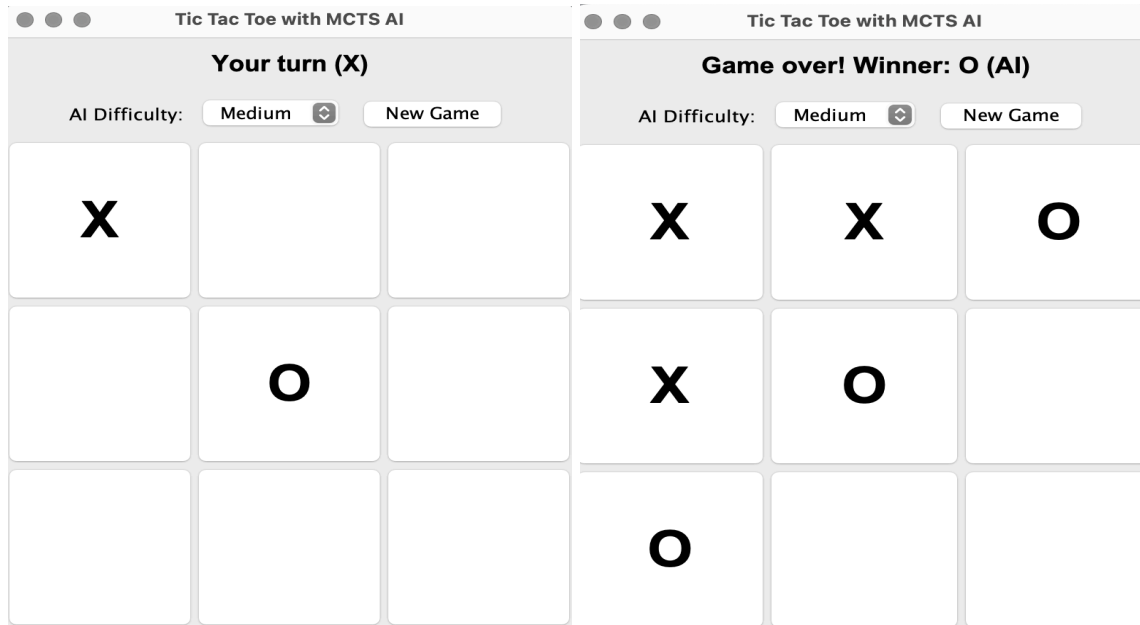
- ✓ testGetCurrentState 4 ms
- ✓ testSimulationRunComplete 6 sec 71 ms
- ✓ testIncrementMoveCount 1 ms
- ✓ testDisplayAndWinnerOutput 1 ms

/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home/bin/java ...

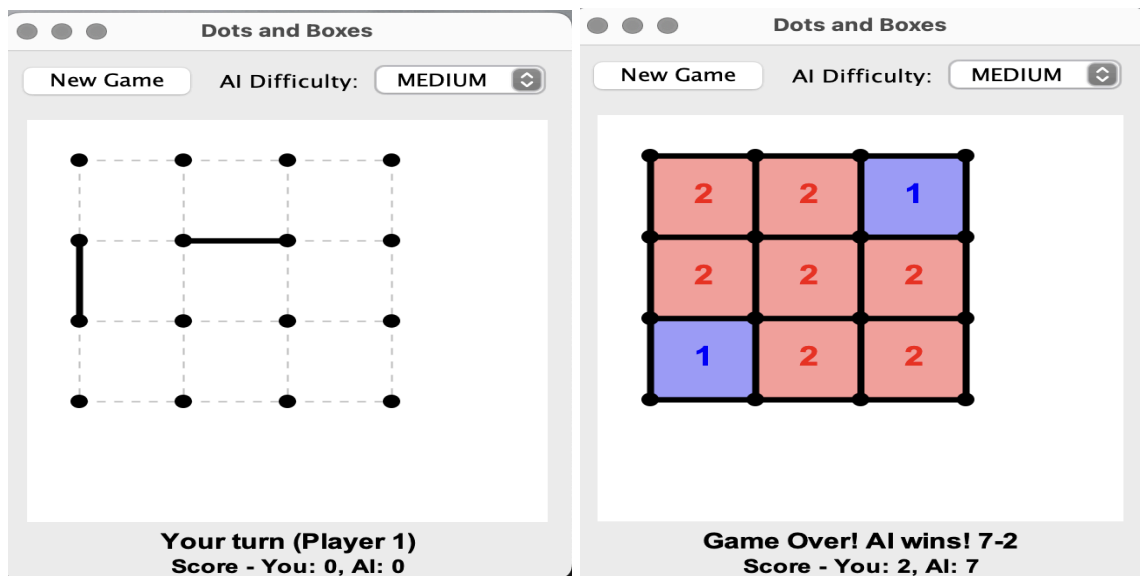
Process finished with exit code 0

GUI:

Tic Tac Toe Game



Dots & Boxes Game



Conclusion

- Tic Tac Toe converged to optimal play with increasing iterations, resulting in nearly 100% draws.
- Dots and Boxes, which are more complicated and strategic, did not exhibit dominance of a single player when employing MCTS with heuristics.
- Heuristics were important in Dots and Boxes, particularly at low iteration counts, as they helped to avoid traps and ensure box completions.
- The difficulty level mechanism helped demonstrate how increasing iteration depth improves AI performance.
- The combination of UCT-based exploration and domain-specific heuristics enabled MCTS to generalize across both simple and moderately complex games.

To run this project:

Watch this short demo video to help you set up the project and start playing the game

[Dots and Boxes Game Demo](#)

References:

[Monte Carlo Tree Search](#) (Wikipedia)

<https://martin-ueding.de/posts/tic-tac-toe-with-monte-carlo-tree-search/>

<https://playgama.com/blog/general/how-can-i-implement-an-ai-opponent-for-the-dots-and-boxes-game-in-my-upcoming-puzzle-game/>