

Program Structures and Algorithms

Spring 2025

FINAL PROJECT REPORT

Team Members

Nihar Sanjaysinh Dabhi – 002341792

Yu Tzu Li – 002376679

Wei Cheng Tu – 002475770

GitHub Link: - [Code Base Link](#)

MONTE CARLO TREE SEARCH



What is Monte Carlo Tree Search

Monte Carlo Tree Search is searching algorithm that uses random simulations (Monte Carlo) to explore a game or problem space and build a search tree, iteratively selecting, expanding, simulating, and backpropagating to find the best course of action.

How Monte Carlo Tree Search Works

Explanation of MCTS core Methods

1.) Selection

The selection of the best child node is core of MCTS. The **UCB** formula is commonly used here, it constantly selects child node with high to low “**UCB**” values.

- **Exploitation:** Focusing on nodes that have shown high win rates in past simulations.

2.) Upper Confidence Bound (UCB)

UCB is a mathematical formula used to evaluate each child node during selection. It considers both the win ratio and the visit count, ensuring a fair balance between good moves and potentially better untested ones.

Formula for “Upper Confidence Bound”

$$\frac{w_i}{s_i} + c \sqrt{\frac{\ln s_p}{s_i}}$$

- w_i : this node's number of simulations that resulted in a win
- s_i : this node's total number of simulations
- s_p : parent node's total number of simulations
- c : exploration parameter

3.) Expansion

Once the selection phase reaches a node that has unexplored legal moves, one of them is chosen and added as a new child node. This process allows the algorithm to grow its search tree selectively, focusing only on relevant parts of the game space.

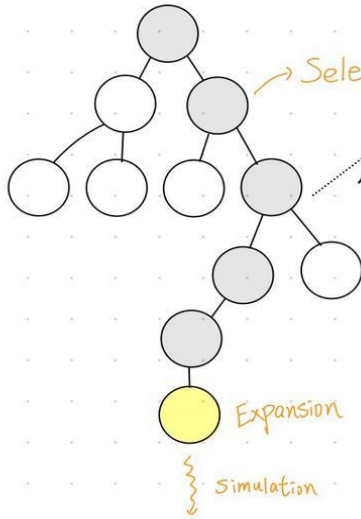
4.) Simulation

During simulation, the algorithm plays out the game from the expanded node using random moves until an outcome is reached.

5.) **Backpropagation**

After the simulation concludes, the result (win/loss/draw) is passed back up to the tree. And from the expanded node we will swim up to the root node and update the cell state.

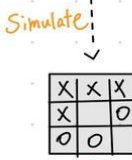
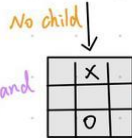
Four Phase of “MCTS” Visualization



Round 1

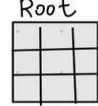


Select



X Win
Win = 2
Payout = 1

Round 2



* U_n = UCT value

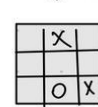


Has child



select

No child

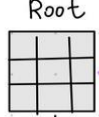


Simulate

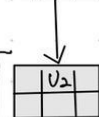


O Win
Win = 0
Payout = 1

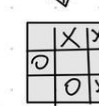
Round 3



Selection



No child

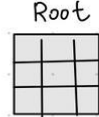


Simulate

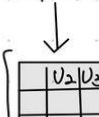


X Win
Win = 2
Payout = 1

Round 4



Selection



No child



X Win
Win = 2
Payout = 1

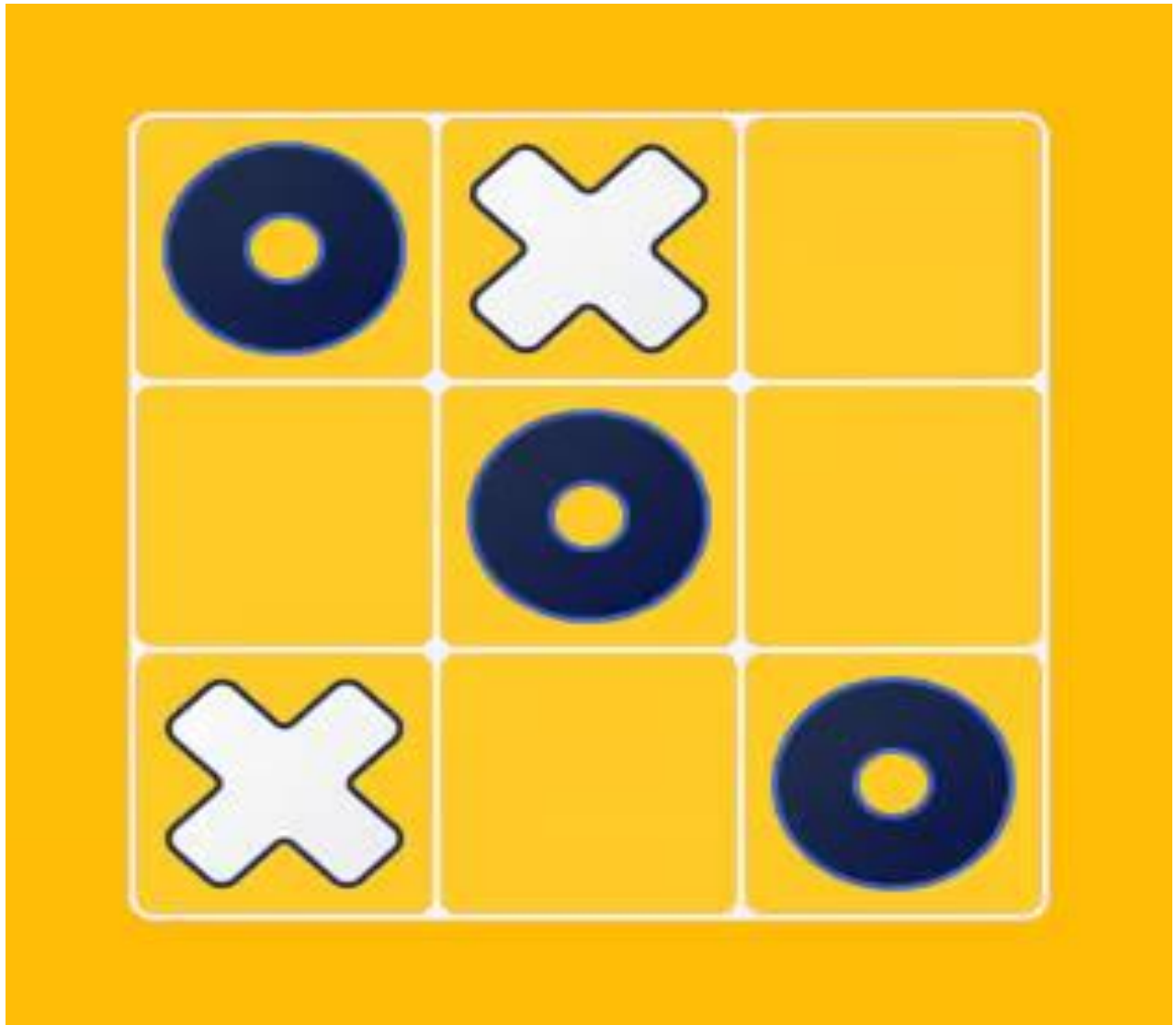
Round 5

Round 6

This demonstration shows the MCTS mechanism within a specific path.

- Selection: Select the position with highest UCT value.
- Expansion: If the grid has no more UCT value, then expand a new node for simulation.
- Simulation: Randomly move till the game is over.
- Back propagation: Send the outcome from the Expansion grid to the Root.

Implementation for Tic Tac Toe



Game Setup and MCTS Integration

In our Tic Tac Toe game, two players compete on a 3x3 grid. The objective is simple, place three of their symbols (X or O) in a row either horizontally, vertically, or diagonally (both forward and backward) to win the game. The game ends when one of the players achieves this objective or when the board is full, resulting in a draw.

In this project, there are two parts:

- **Game Launcher** – where a human player plays against either the MCTS algorithm or random moves.
- **Simulation for Benchmarking** – where we run the game with different numbers of iterations/simulations to benchmark the performance of the MCTS algorithm.

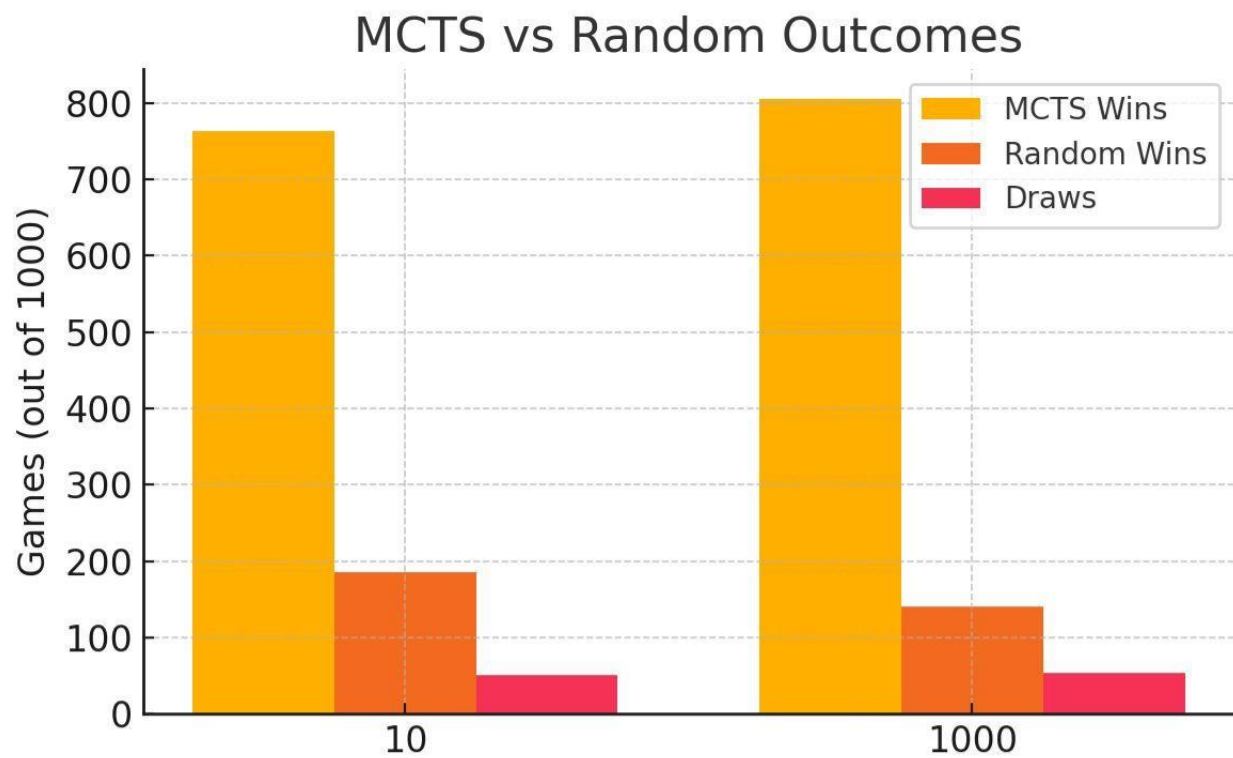
Why Use MCTS for Tic Tac Toe?

While “tic tac toe” game isn’t that hard game to play since it has limited number of cells to choose from, by applying “MCTS” algorithm we can understand the concept of tree search and able to know that how simulation-based tree search algorithms work.

Also “MCTS” does not have to feed specific domain data to train it. It can decide what move is optimal to make in the tic tac toe game on the go.

Explanation of Tic tac toe simulations (Screen shot)

Picture 1



Outcome	Count
MCTS Wins	805
Random Wins	141
Draws	54

Outcome	Count
MCTS Wins	764
Random Wins	185
Draws	51

The graph here shown is Comparison between MCTS vs Random Move for 1000 games that we play.

So, for better comparison we give “**10**” and “**1000**” numbers of simulation for MCTS respectively.

There are 3 different metrics we can see in the graph “**MCTS win**”, “**Random win**” and “**Draws**”. Here “**Random wins**” means it is randomly making a move and win.

As, we can see if there is “**10**” number of simulations; number of wins for MCTS is around “**750**”, Random Wins is around “**180**” and Draw is “51”

For “**1000**” numbers of simulation MCTS wins “**805**”, Random win is decreased to “**141**” and Draws is “**54**”.

So, in conclusion we can say that if we try to increase the number of simulations the more accurate MCTS would be for Tic Tac Toe game.

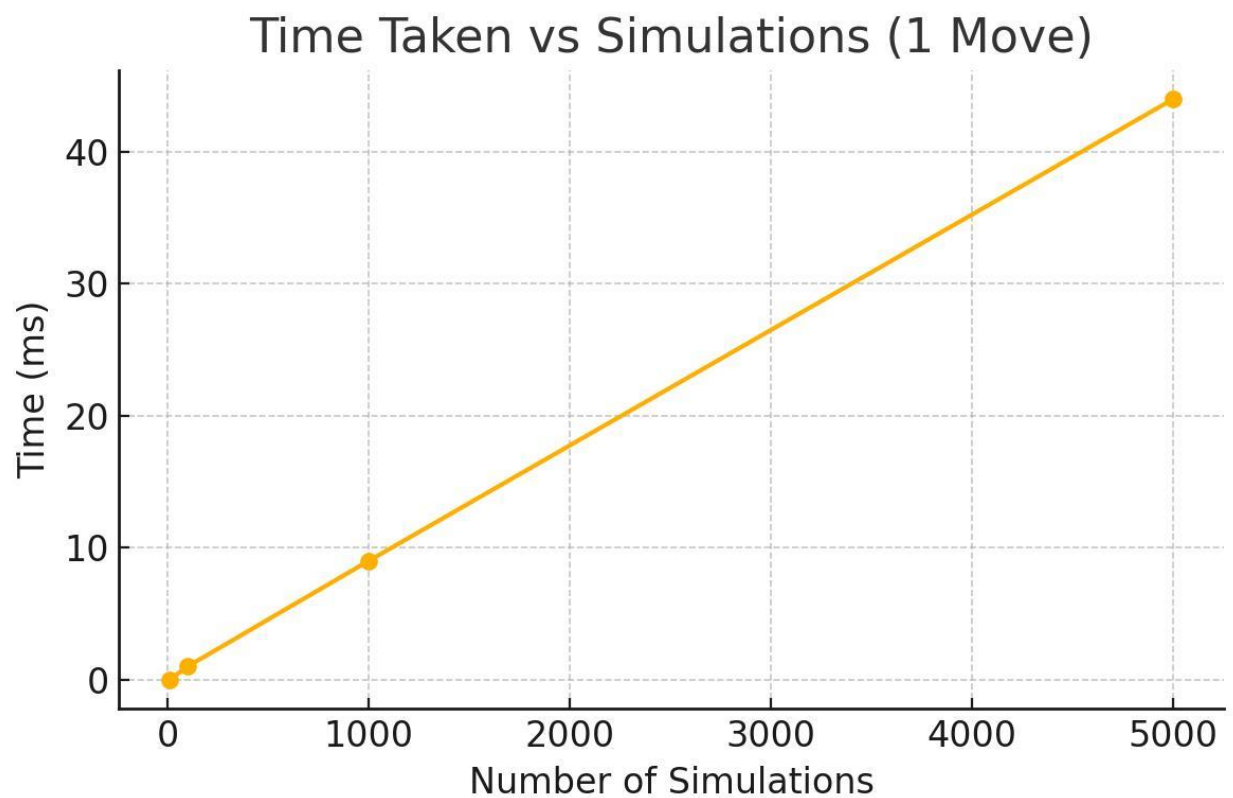
Picture 2

Move	Player	Time (ms)
1	1 (X)	5
2	0 (O)	3
3	1 (X)	2
4	0 (O)	2
5	1 (X)	1
6	0 (O)	1

The image above is based on 1000 simulations of a single round of Tic-Tac-Toe. It shows how the MCTS players take time to make their moves. The game starts with Player 1, who takes approximately 5 milliseconds to think and move. Then Player 2 takes around 3 milliseconds. As the game progresses, both players take less and less time to make their moves. By the 6th move, either one player has won, or the game ends in a draw.

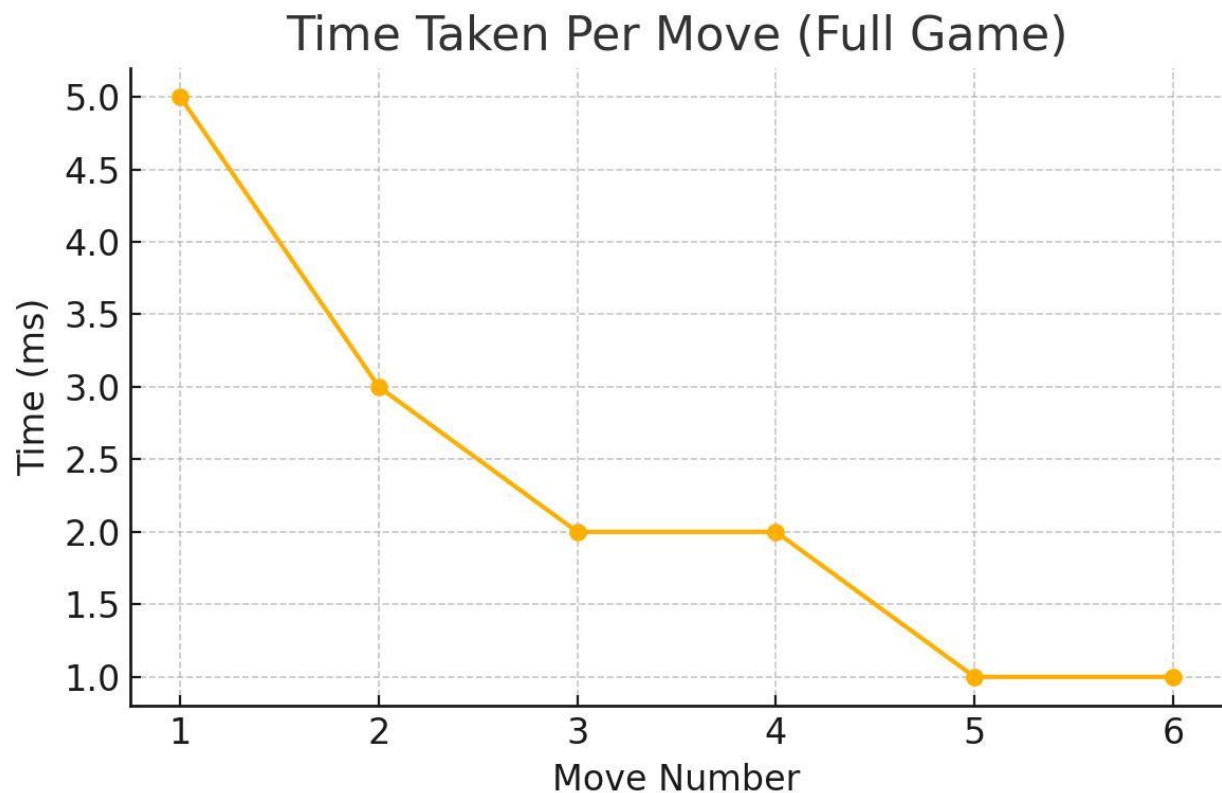
Picture 3

Simulations	Time (ms)
10	0
100	1
1000	9
5000	44



From both given images above, it is evidence of time taken by MCTS algorithm to make a move in tic tac toe game, as we can see if we increase the number of simulations; time taken by MCTS algorithm is significantly increased after some number of simulations. Give more accuracy to MCTS algorithm to choose their move.

Picture 4



The given image shows the time taken by MCTS algorithm to make their move as we process in the game; time took by MCTS is getting less and lesser. One of the reasons behind the situation is that as we move forward in this game number of cell for making a next move will be decreased so MCTS algorithm will take less time to

expand all possible scenario for the next move and hence will see decrease in the time.

Screen Shot of Test cases (Tic tac toe)

✓	TicTacToeNodeTest (com.phasmidsoftware.dsaipg.projects.mcts.tictac)	4 ms
✓	addChild	1 ms
✓	state	2 ms
✓	white	0 ms
✓	backPropagate	0 ms
✓	children	0 ms
✓	winsAndPlayouts	1 ms

Run		
✓	TicTacToeTest (com.phasmidsoftware.dsaipg.projects.mcts.tictac)	3 ms
✓	runGame	3 ms

UI Screen Shot for Tic Tac Toe

Screen Shot of Game Playing with MCTS Logic in it

Order of screen shot

- 1.) Human wins**
- 2.) Draw Match**
- 3.) MCTS wins**

```

Choose a role you want to play: 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 1
MCTS version!!!!!! Hard mode!!!!
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . .
. X .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move: 1
Current grid:
. . .
. 0 .
. X .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. 0 .
X X .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move: 2
Current grid:
. . .
. 0 .
X X 0
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. 0 X
X X 0
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move: 0
Final result:
0 . .
. 0 X
X X 0
Congratulation you win!

Process finished with exit code 0

```

```

Choose a role you want to play. 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 1
MCTS version!!!!!! Hard mode!!!!
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . X
. . .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move:1
Current grid:
. . .
. . X
. 0 .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . X
X 0 .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move:2
Current grid:
. . .
. . X
X 0 0
Next player: 1 (X)
Computer (1) moved
Current grid:
. X .
. . X
X 0 0
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move:0
Current grid:
. X .
0 . X
X 0 0
Next player: 1 (X)
Computer (1) moved
Current grid:
X X .
0 . X
X 0 0
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move:2
Current grid:
X X 0
0 . X
X 0 0
Next player: 1 (X)
Computer (1) moved
Final result:
X X 0
0 X X
X 0 0
Draw!

```

```
Choose a role you want to play: 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 1
MCTS version!!!!!! Hard mode!!!!
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . .
. . X
Next player: 0 (O)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move:0
Current grid:
. . .
0 . .
. . X
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
0 . X
. . X
Next player: 0 (O)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move:0
Current grid:
. . .
0 . X
0 . X
Next player: 1 (X)
Computer (1) moved
Final result:
. . X
0 . X
0 . X
Computer Win!

Process finished with exit code 0
```

Screen Shot of Game Playing without MCTS Logic in it

Order of screen shot

- 1.) Computer wins
- 2.) Human wins
- 3.) Draw Match

```
Choose a role you want to play: 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 0
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . .
. . X
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move: 2
Current grid:
. . .
. . 0
. . X
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . 0
X . X
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move: 2
Current grid:
. . 0
. . 0
X . X
Next player: 1 (X)
Computer (1) moved
Current grid:
. . 0
X . 0
X . X
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move: 1
Current grid:
. . 0
X 0 0
X . X
Next player: 1 (X)
Computer (1) moved
Current grid:
. X 0
X 0 0
X . X
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move: 2
Invalid move: Position is occupied: 1, 2, do it again!
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move: 0
Current grid:
0 X 0
X 0 0
X . X
Next player: 1 (X)
Computer (1) moved
Final result:
0 X 0
X 0 0
X X X
Computer Win!
```

```

Choose a role you want to play: 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 0
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. X .
. . .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move:0
Current grid:
. . .
0 X .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
0 X X
. . .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move:0
Current grid:
0 . .
0 X X
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
0 . X
0 X X
. . .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move:0
Final result:
0 . X
0 X X
0 . .
Congratulation you win!

Process finished with exit code 0

```



```

Choose a role you want to play - 0=0 or 1=X: 0
Choose a version: 0 for original competitor; 1 for MCTS version competitor: 0
Current grid:
. . .
. . .
. . .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. . .
. X .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move:1
Current grid:
. . .
. 0 .
. X .
Next player: 1 (X)
Computer (1) moved
Current grid:
. . .
. 0 .
X X .
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 2
Enter the column (0-2) where you want to make your move:2
Current grid:
. . .
. 0 .
X X 0
Next player: 1 (X)
Computer (1) moved
Current grid:
. . X
. 0 .
X X 0
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 1
Enter the column (0-2) where you want to make your move:0
Current grid:
. . X
0 0 .
X X 0
Next player: 1 (X)
Computer (1) moved
Current grid:
. . X
0 0 X
X X 0
Next player: 0 (0)
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move:2
Invalid move: Position is occupied: 0, 2, do it again!
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move:2
Invalid move: Position is occupied: 0, 2, do it again!
Enter the row (0-2) where you want to make your move: 0
Enter the column (0-2) where you want to make your move:1
Current grid:
. 0 X
0 0 X
X X 0
Next player: 1 (X)
Computer (1) moved
Final result:
X 0 X
0 0 X
X X 0
Draw!

Process finished with exit code 0

```

Game 2048

1. Introduction

We apply **Monte Carlo Tree Search (MCTS)** to play 2048. Our implementation treats each board state as a node in the search tree and uses random simulations to evaluate possible moves. We compare the results of running **different numbers of simulations** (e.g., 1, 100) to analyze how simulation depth affects gameplay.

This study aims to answer the following questions:

- How well can MCTS play 2048 using purely random simulations?
- How does performance vary with simulation count?
- What trade-offs exist between decision quality and runtime?

Through experimentation and benchmarking, we demonstrate that MCTS can produce good results with sufficient simulations and analyze the effects of simulation budget on final score, maximum tile achieved, and survival length.

2. How to Play

- The game is played on a 4×4 grid.
- Use the W, A, S, and D keys, where W represents UP, S represents DOWN, A represents LEFT, and D represents RIGHT, to slide all tiles in one direction.
- When two tiles with the same number collide, they merge into one (e.g., $2 + 2 = 4$). After each move, a new tile (2 or 4) appears in a random empty spot. In our implementation, there is a 90% probability of spawning a 2 and a 10% probability of spawning a 4.
- Your score increases by the value of the merged tile.
- You can only move tiles if at least one tile changes or merges.
- The game ends when no moves are possible (i.e., no empty spaces and no adjacent tiles of the same value).
- You win by creating a 2048 tile.

3. Analysis and Discussion

a. Simulation Budget and Decision Quality

To evaluate the impact of the number of simulations on MCTS performance in the 2048 game, we compare the outcomes of running 1 simulation versus 100 simulations across 10 games each. Results show a clear and significant difference in gameplay.

Benchmark Summary

Benchmark Summary Comparison (Sim 1 vs Sim 100)

Metric	Simulations = 100	Simulations = 1
Total Games	10.0	10.0
Number of Simulations	100.0	1.0
Average Score	10368.0	855.0
Average Moves	648.0	101.0
Max Tile Achieved	1024.0	128.0
Avg Time per Game (ms)	13948.9	63.0
Avg Time per Move (ms)	21.5	0.62

Game Results (Simulation = 1)

Game	Score	Max Tile	Moves
1	1088	128	115
2	1292	128	133
3	288	32	55
4	1364	128	143
5	488	64	70
6	868	64	104
7	748	64	97
8	1124	128	119
9	640	64	84
10	656	64	90

Game Results (Simulation = 100)

Game	Score	Max Tile	Moves
1	14528	1024	845
2	7344	512	509
3	16456	1024	989
4	3084	256	253
5	7172	512	499
6	16180	1024	951
7	6908	512	466
8	12360	1024	733
9	12488	1024	753
10	7164	512	491

As seen in the table above, with only 1 simulation, MCTS behaves nearly randomly, never surpassing a max tile of 128 and achieving low scores. In contrast, with 100 simulations, the agent reached

1024 tiles and generated high scores, indicating more strategic and deliberate play.

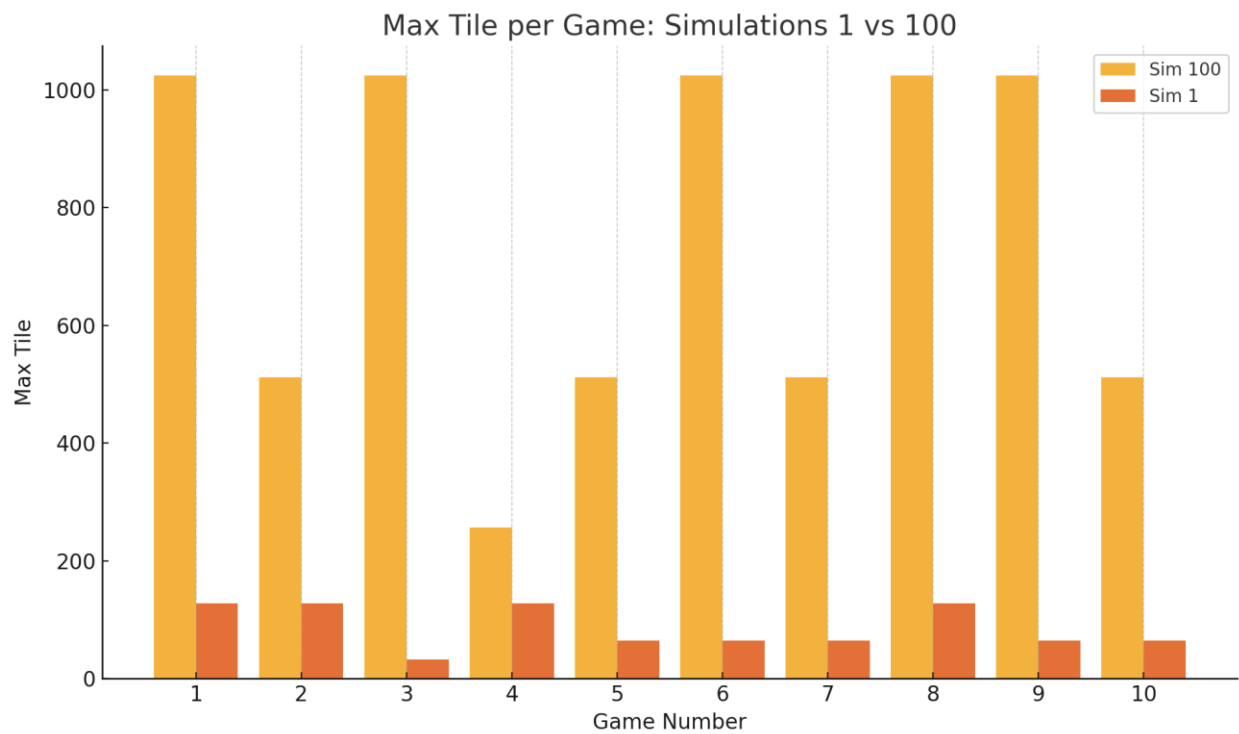
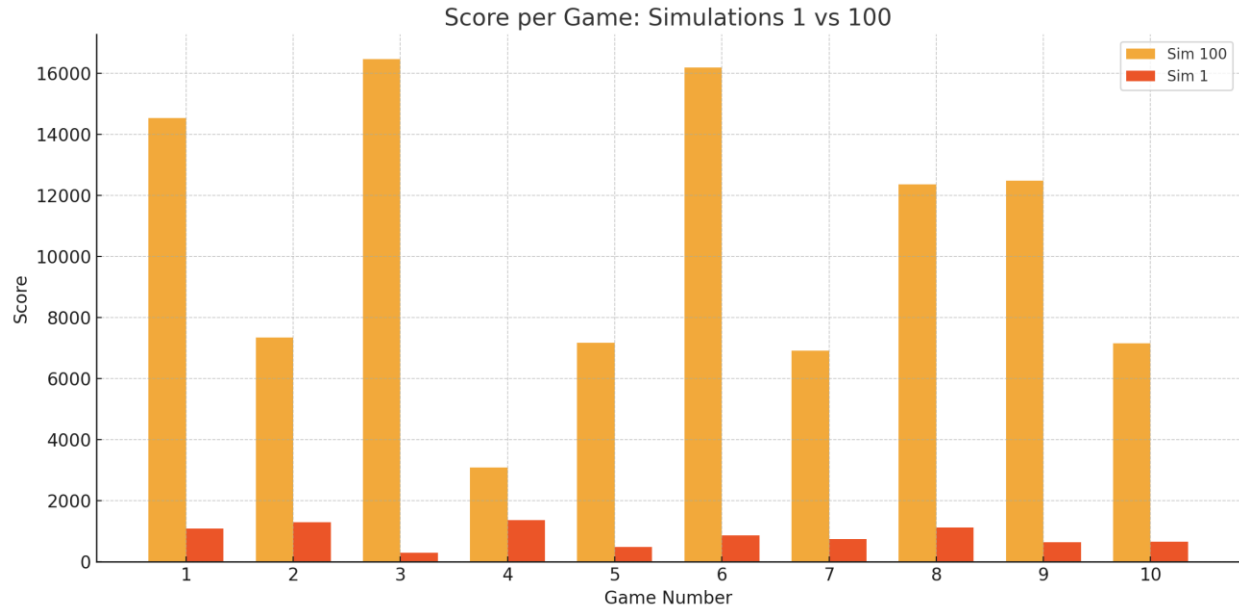
b. Trade-off Between Time and Performance:

The time cost per move is another factor to consider Simulation = 1: Extremely fast ~0.62 ms/move, but performance is poor. Simulation = 100: Much better decision quality but requires ~21.5 ms/move. This highlights a key trade-off: higher simulation counts improve strategic accuracy but come at the cost of slower execution.

c. Game-by-Game Observations:

The score chart below shows that with Simulation = 1, overall performance was low, with the lowest score recorded at 288. In contrast, Simulation = 100 resulted in several high-scoring games, some surpassing 12,000 points.

Similarly, Max Tile per Game chart shows that Simulation = 100 achieved higher tile levels, with 50% of games reaching 1024, while Simulation = 1 never surpass 128 tile.



d. Limitation:

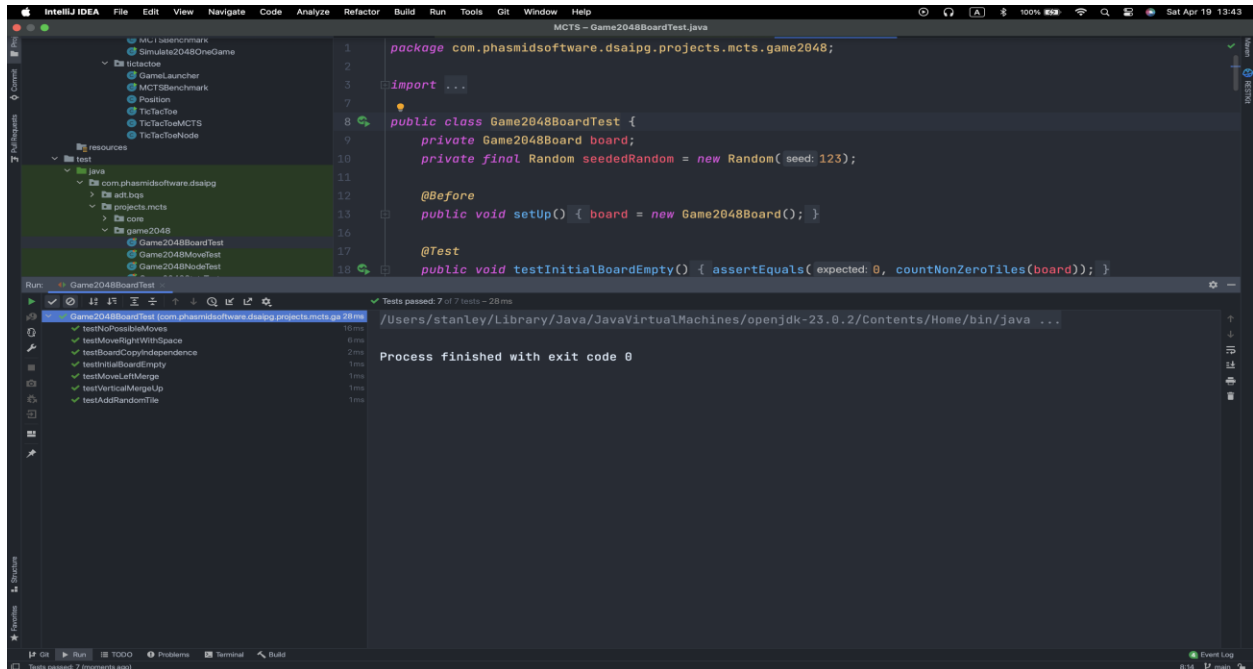
Despite improved results with 100 simulations, none of the games reached the 2048 tile. This suggests that MCTS, in its current form, may be inefficient due to its reliance on random simulations and may lack the long-term planning necessary to reach late-game tiles.

e. Demo of MCTS playing 2048 with 1000 random simulations:

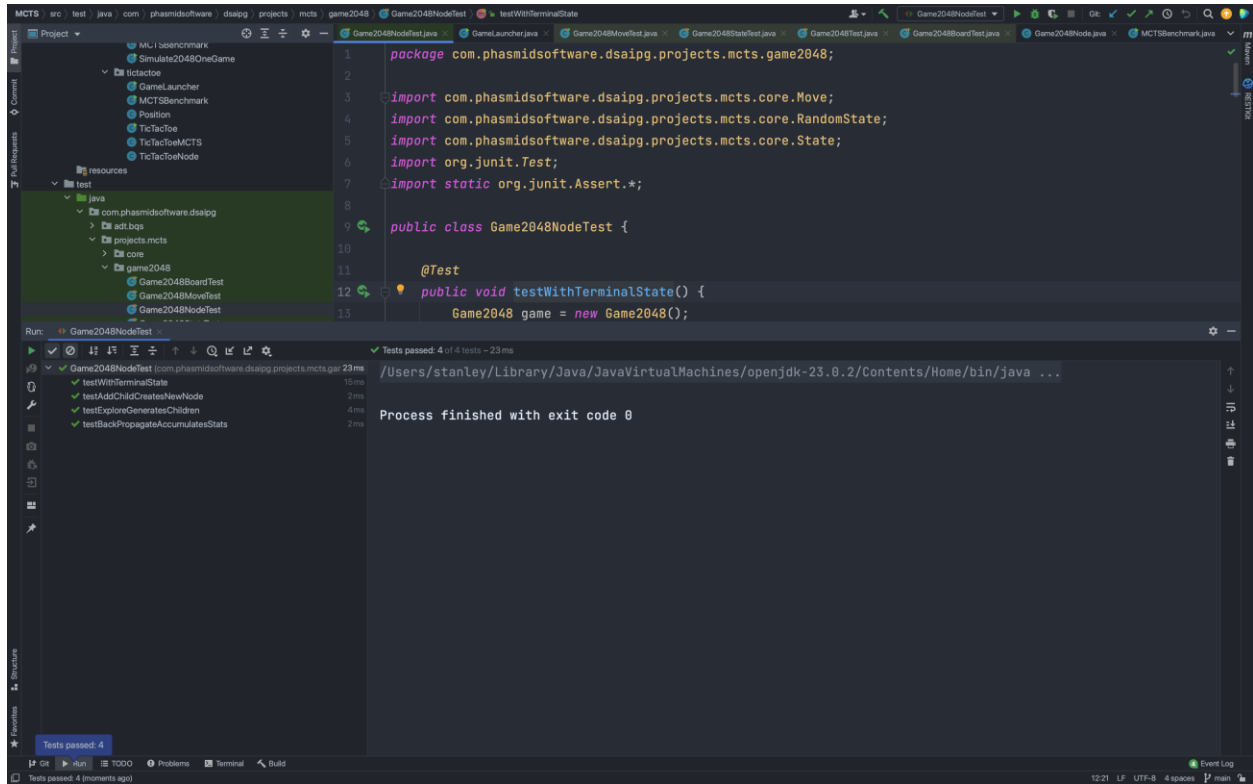
[video demo on you tube](#)

f. Screenshot of Unit Test cases for Game 2048

1 unit test case for Game2048 Board test



2 unit test case for Game2048 Node test

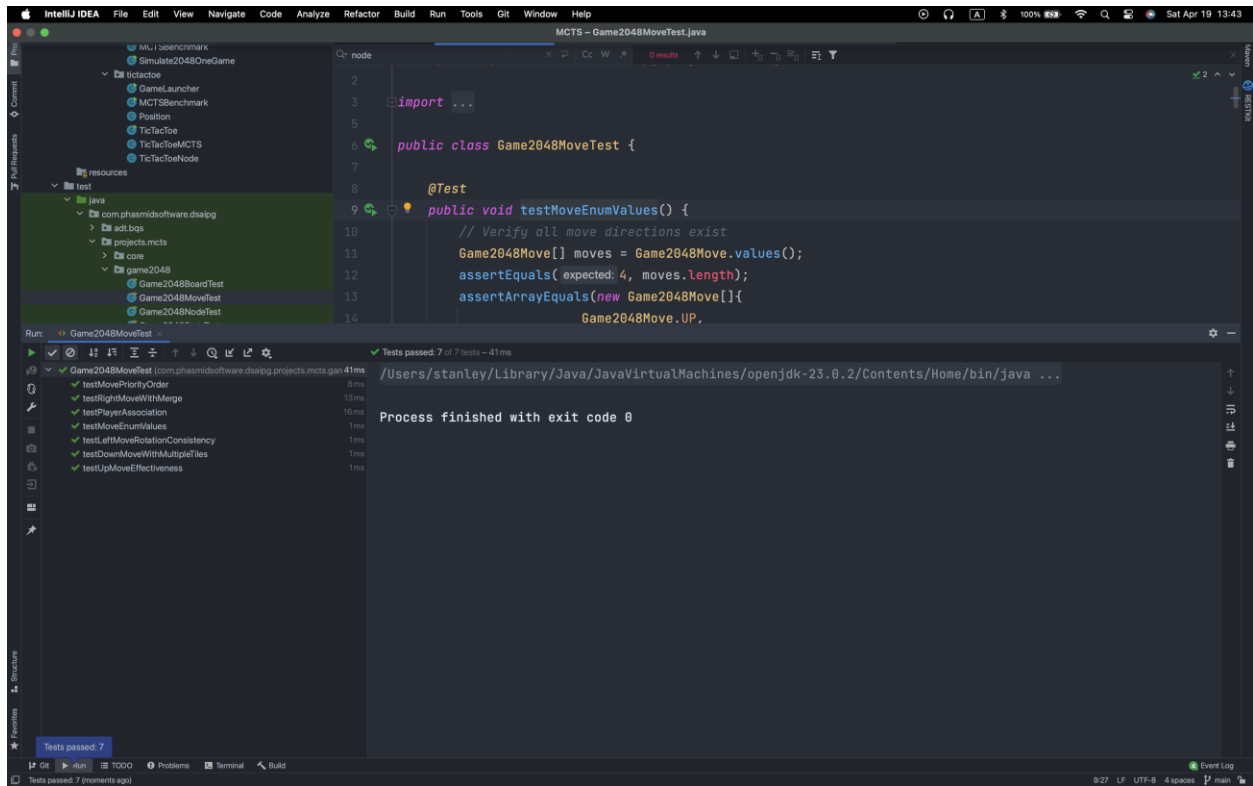


The screenshot displays an IDE with the following components:

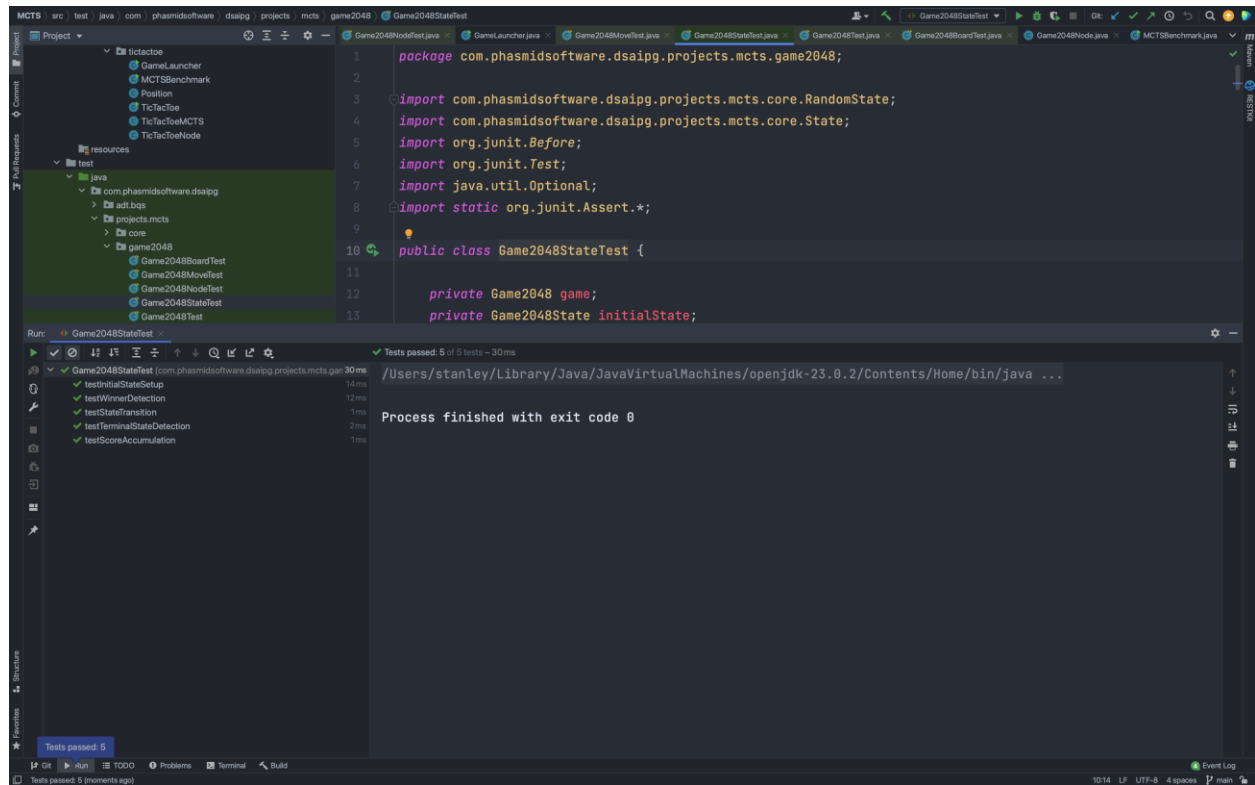
- Project Explorer:** Shows the project structure with folders for `resources` and `test`. The `test` folder contains a `java` sub-folder with the package `com.phasmidsoftware.dsaipg.projects.mcts.game2048`. The `Game2048NodeTest` class is highlighted.
- Code Editor:** Contains the following Java code:

```
1 package com.phasmidsoftware.dsaipg.projects.mcts.game2048;
2
3 import com.phasmidsoftware.dsaipg.projects.mcts.core.Move;
4 import com.phasmidsoftware.dsaipg.projects.mcts.core.RandomState;
5 import com.phasmidsoftware.dsaipg.projects.mcts.core.State;
6 import org.junit.Test;
7 import static org.junit.Assert.*;
8
9 public class Game2048NodeTest {
10
11     @Test
12     public void testWithTerminalState() {
13         Game2048 game = new Game2048();
```
- Run Console:** Shows the execution of the test. The output indicates that 4 tests passed in 23 ms. The tests listed are:
 - Game2048NodeTest (com.phasmidsoftware.dsaipg.projects.mcts.game2048) 23 ms
 - testWithTerminalState 15 ms
 - testAddChildCreatesNewNode 2 ms
 - testExploreGeneratesChildren 4 ms
 - testBackPropagateAccumulatesStats 2 ms
- Terminal:** Displays the message "Process finished with exit code 0".

3 unit test case for Game2048 Move Test



4 unit test case for Game2048 Move test

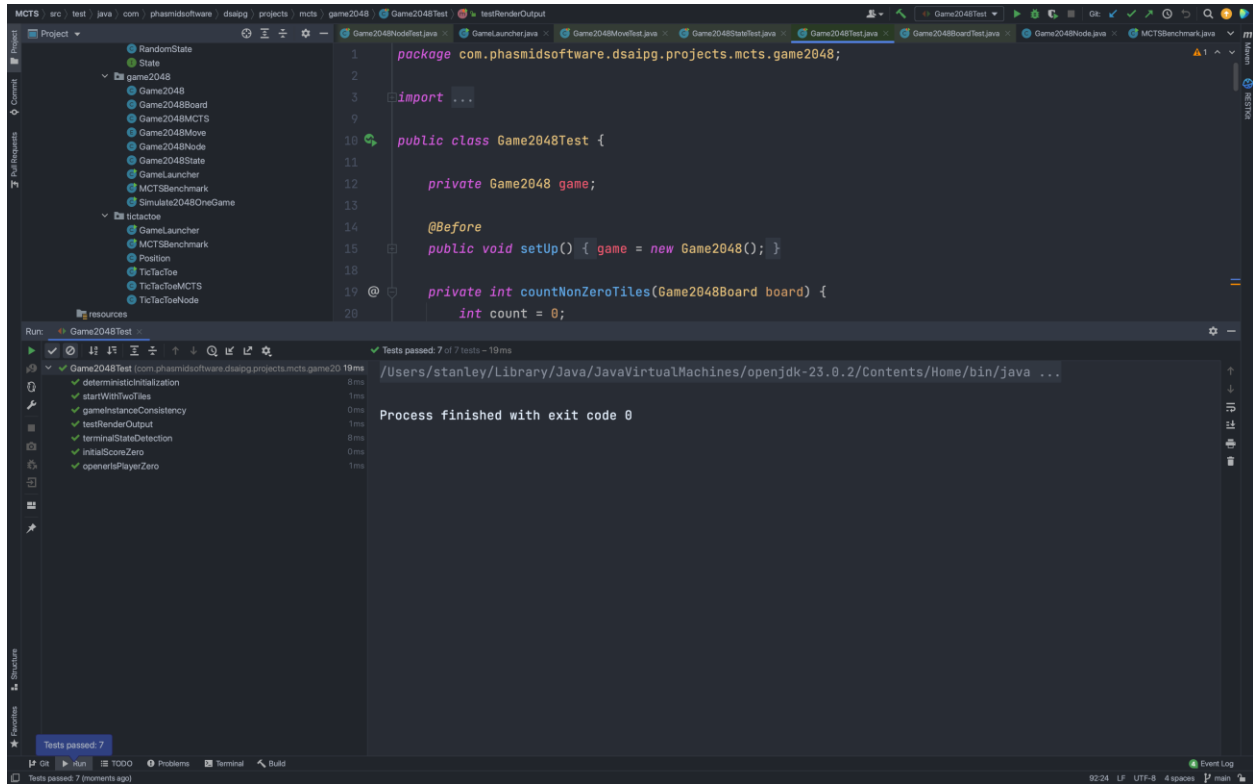


The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with folders for `src`, `test`, `java`, `com.phasmidsoftware.dsaipg.projects.mcts`, and `game2048`. The `Game2048StateTest` class is highlighted under `game2048`.
- Code Editor:** Displays the `Game2048StateTest` class with the following code:

```
1 package com.phasmidsoftware.dsaipg.projects.mcts.game2048;
2
3 import com.phasmidsoftware.dsaipg.projects.mcts.core.RandomState;
4 import com.phasmidsoftware.dsaipg.projects.mcts.core.State;
5 import org.junit.Before;
6 import org.junit.Test;
7 import java.util.Optional;
8 import static org.junit.Assert.*;
9
10 public class Game2048StateTest {
11
12     private Game2048 game;
13     private Game2048State initialState;
```
- Run Console:** Shows the test results for `Game2048StateTest`. The tests passed are:
 - `testInitialSetup` (14 ms)
 - `testWinnerDetection` (12 ms)
 - `testStateTransition` (1 ms)
 - `testTerminalStateDetection` (2 ms)
 - `testScoreAccumulation` (1 ms)The total time for all tests is 30 ms. The console output shows: `Process finished with exit code 0`.

5 unit test case for Game2048 test



References

1 [what is MCTS](#)

2 [what is game 2048](#)

3 [how to use MCTS in game 2048](#)

4 [about tic tac toe game](#)