

Final Project Report – INFO 6205

Program Structures and Algorithms – Spring 2025

Raghavendra Prasath Sridhar – (002312779) – AI logic, GUI design, move validation, MCTS integration

Nikhil Pandey - (002775062) – Game engine logic, difficulty scaling, UI polish

Ayush Patil - (002325566) – Testing, debugging, performance tuning, documentation

GitHub Link: <https://github.com/raghavendrprasath/MCTSPProject/tree/main>

Executive Summary

This project explores the application of Monte Carlo Tree Search (MCTS) in both deterministic and stochastic environments by implementing intelligent agents for Tic Tac Toe and 2048. Using an interface-driven Java framework, we built modular logic for simulation-based decision-making. The GUI for each game was developed using Java Swing with full user interactivity and performance benchmarking. Results show that MCTS can make optimal decisions in Tic Tac Toe and provide competitive high scores in 2048. The project demonstrates strong software design principles, algorithmic understanding, and real-time benchmarking of AI behavior.

Project Overview

This project demonstrates the implementation of the Monte Carlo Tree Search (MCTS) algorithm to solve two distinct games: Tic Tac Toe (a deterministic, perfect-information game) and 2048 (a stochastic single-player puzzle). The objective was to apply MCTS to both domains, design an interactive user interface, benchmark AI performance, and ensure extensibility for future applications.

Project Objectives

- Understand and apply the MCTS algorithm with domain-specific adaptations.
- Build an object-oriented Java framework using abstract interfaces.
- Develop an interactive GUI using Java Swing.
- Integrate performance benchmarking (simulation rollouts, timing, evaluations).
- Support both deterministic and probabilistic game logic.

Game 1: Tic Tac Toe with MCTS

Implementation Highlights

- Full AI logic using MCTS with pluggable simulation depth.
- Modular structure with custom TicTacToe classes.
- Difficulty modes (Easy, Hard), win/draw detection, replay, dark mode.
- Java Swing GUI with sound, animations, and player input.

Performance Logging

- Rollouts per AI move printed to terminal.
- Timing per simulation using `System.nanoTime()`.

Game 2: 2048 with MCTS

Implementation Highlights

- Adapted MCTS for stochastic tile generation.
- Evaluation-based scoring, AI autoplay, score tracking.
- Grid-based Java Swing GUI with animations and dark mode.

Performance Logging

- Terminal log: move count, score, max tile, average time.
- Public `evaluate()` method prints evaluation score.

Unit Testing

We added JUnit test cases for both game 2048 and updated logic in tictactoe to ensure correctness and MCTS compatibility. Coverage includes state transitions, move validity, and simulation outcome propagation.

MCTS Algorithm Summary

1. Selection: Traverse the tree using UCB1.
2. Expansion: Add one or more child nodes.
3. Simulation: Play random rollouts.
4. Backpropagation: Propagate the result up the tree.

How to Build & Run

Prerequisites

- Java 17+
- Maven 3.8+
- Git

Run Instructions

Clone the repo:

```
git clone https://github.com/raghavendrappasath/MCTSPProject.git
```

Screen recording video has been attached in the GitHub repo to clone, run the game and a short gameplay to display features

Game Rules References

Tic Tac Toe: <https://papergames.io/docs/game-guides/tic-tac-toe/basic-guide/>

2048: <https://rosettacode.org/wiki/2048>

Project Structure

```
MCTSPProject/
├── src/
│   └── mcts/
│       ├── core/
│       ├── game2048/
│       └── tictactoe/
├── test/
│   ├── core/
│   ├── game2048/
│   └── tictactoe/
├── pom.xml
└── README.md
```

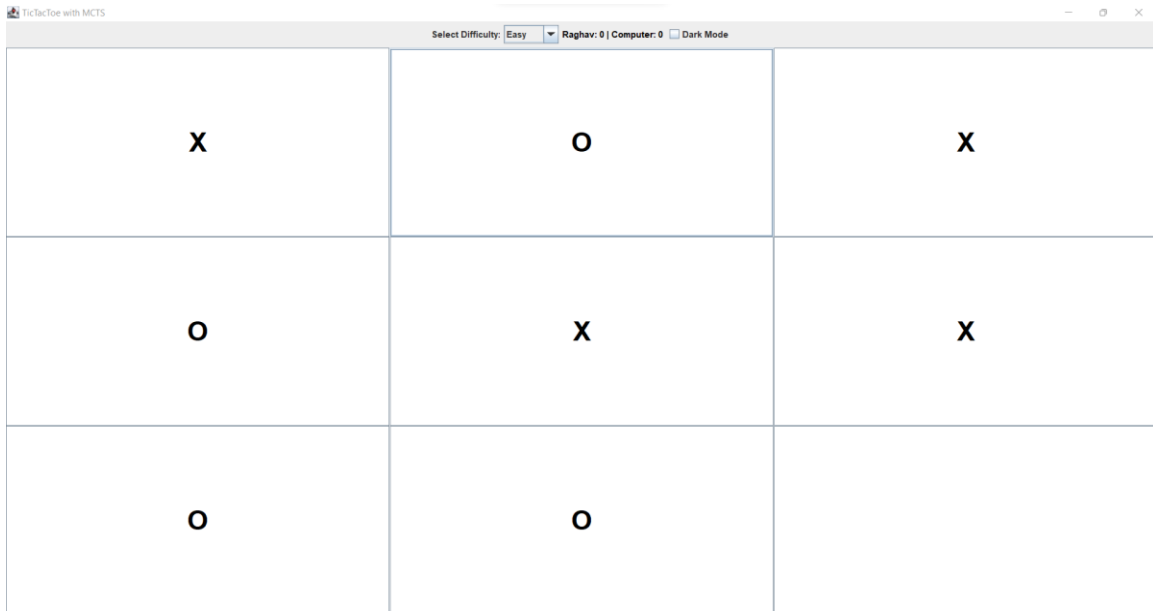
References

- [Monte Carlo Tree Search – Wikipedia](#)
- [Monte Carlo Tree Search: A Review of Recent Modifications and Applications – arXiv:2103.04931](#)
- [Monte Carlo Tree Search: A New Framework for Game AI – Cameron Browne et al., IEEE 2012](#)
- [A Survey of Monte Carlo Tree Search Methods – Robert Coulom](#)

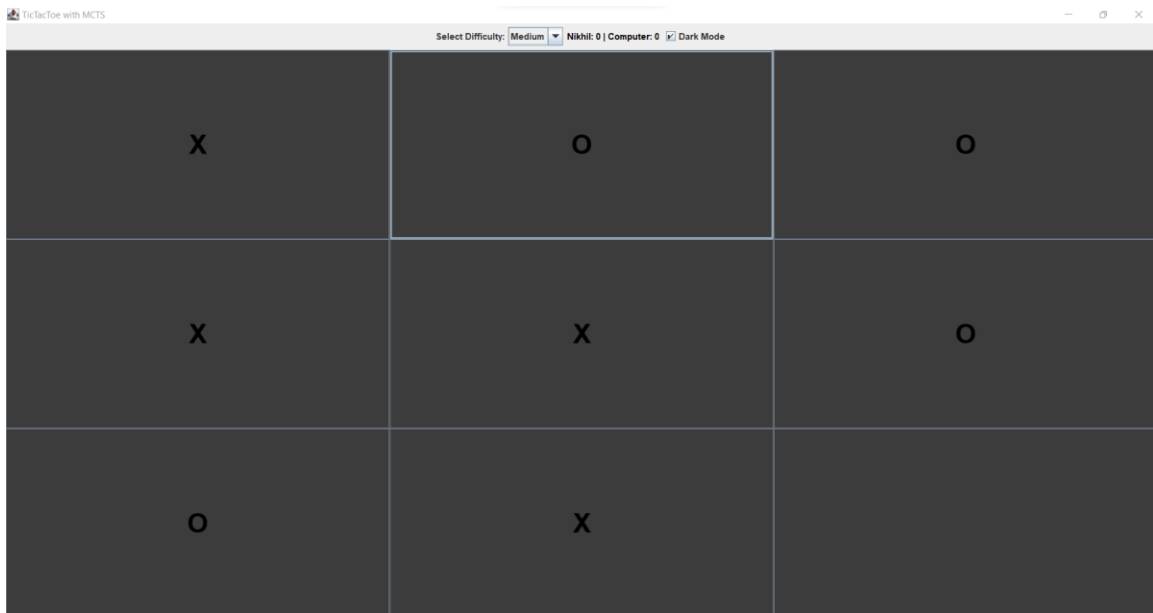
Screenshots

Graphical User Interface of Tic-Tac-Toe (running WelcomeScreen.java):

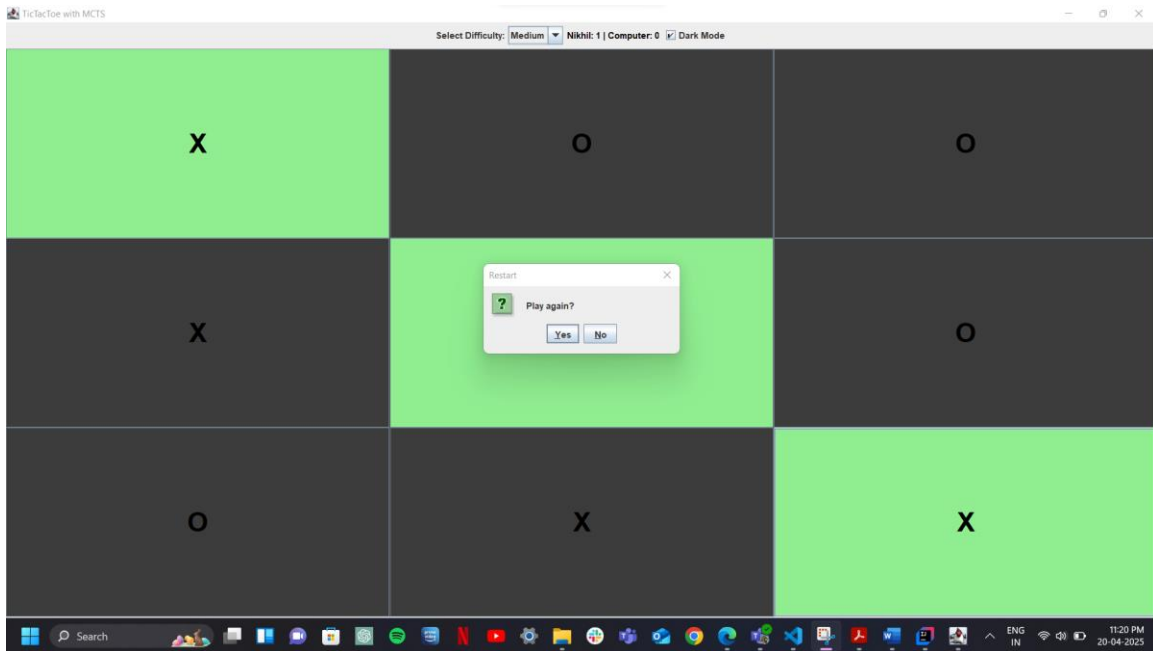
Player vs Computer in Easy Difficulty in Light Mode:



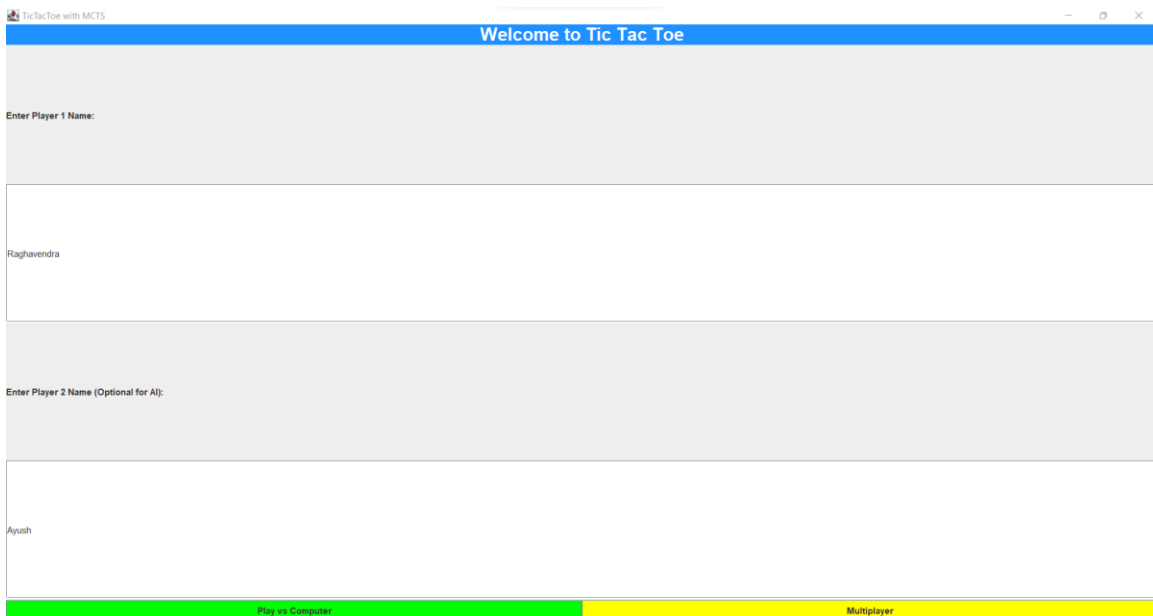
Player vs Computer in Medium Difficulty in Dark Mode:



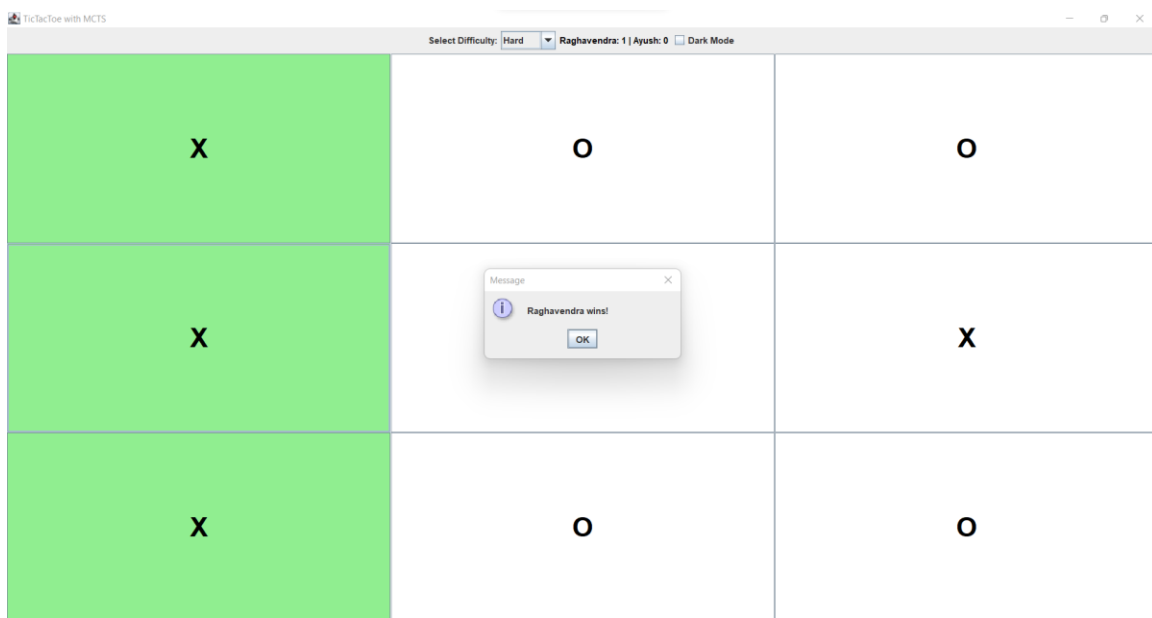
Restarting a game:



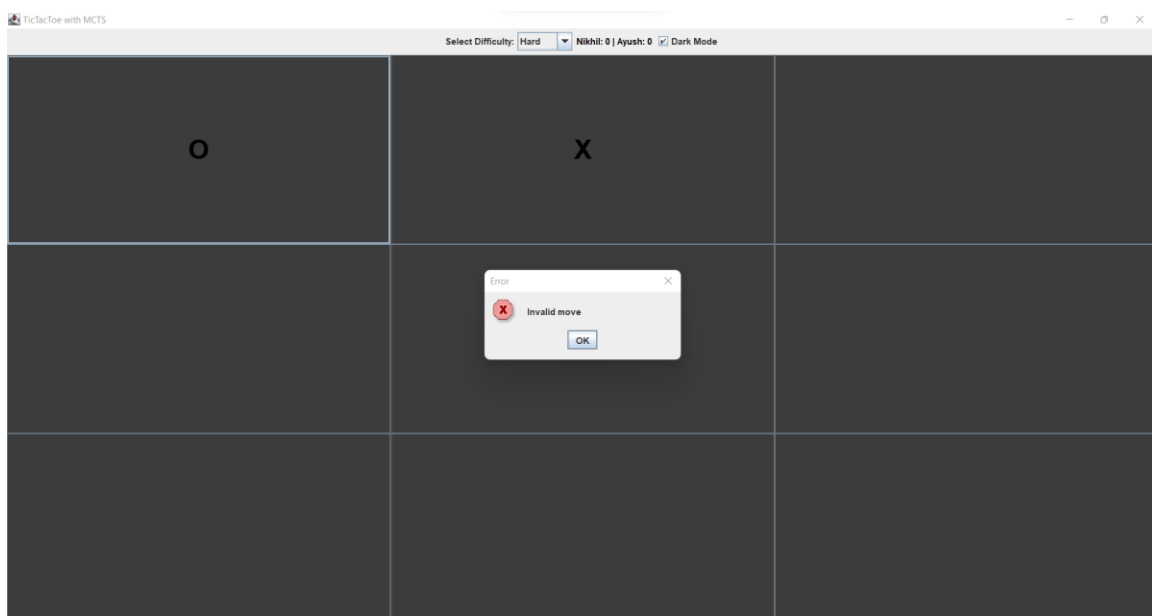
Input player names in Welcome Screen:



Outcome of Result in Multiplayer Mode:

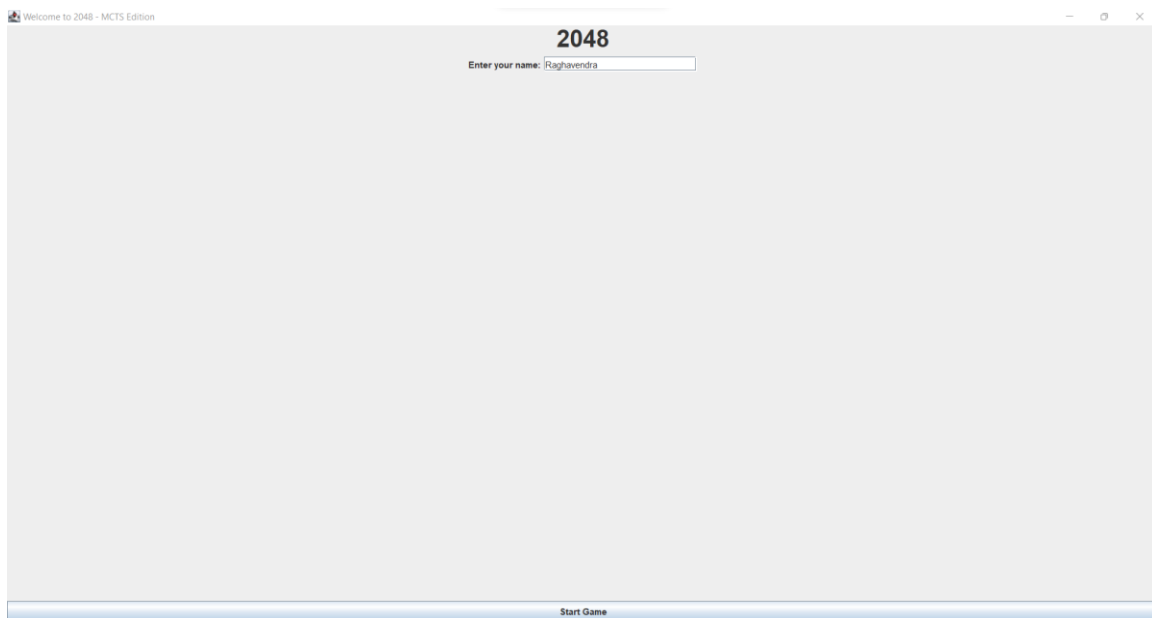


Making an invalid move:

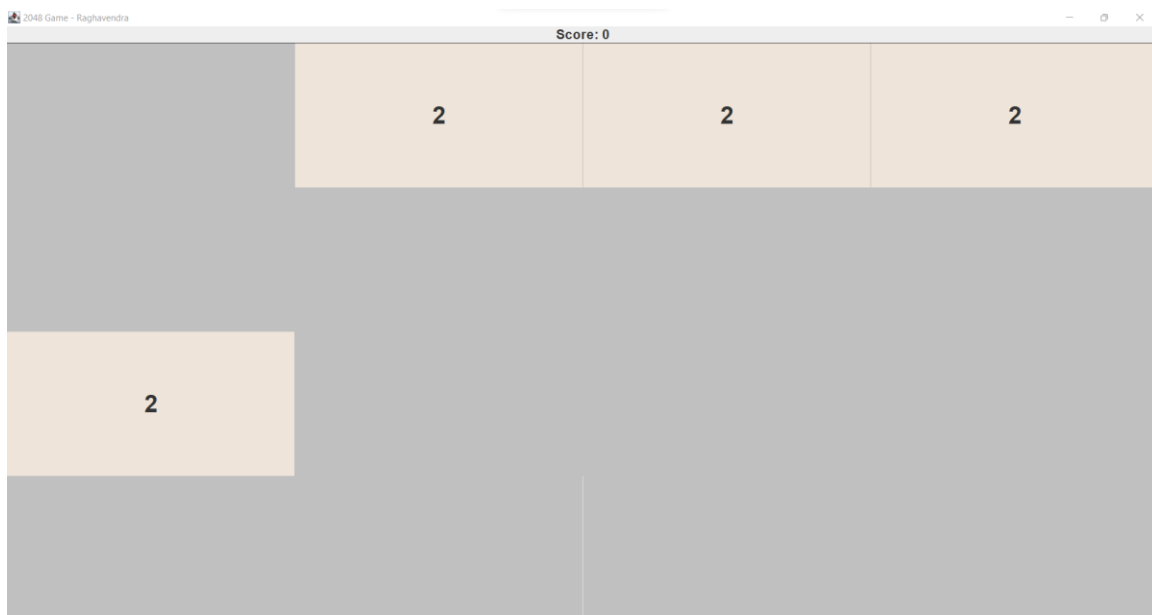


Graphical User Interface of 2048 Game (running Game2048Runner.java):

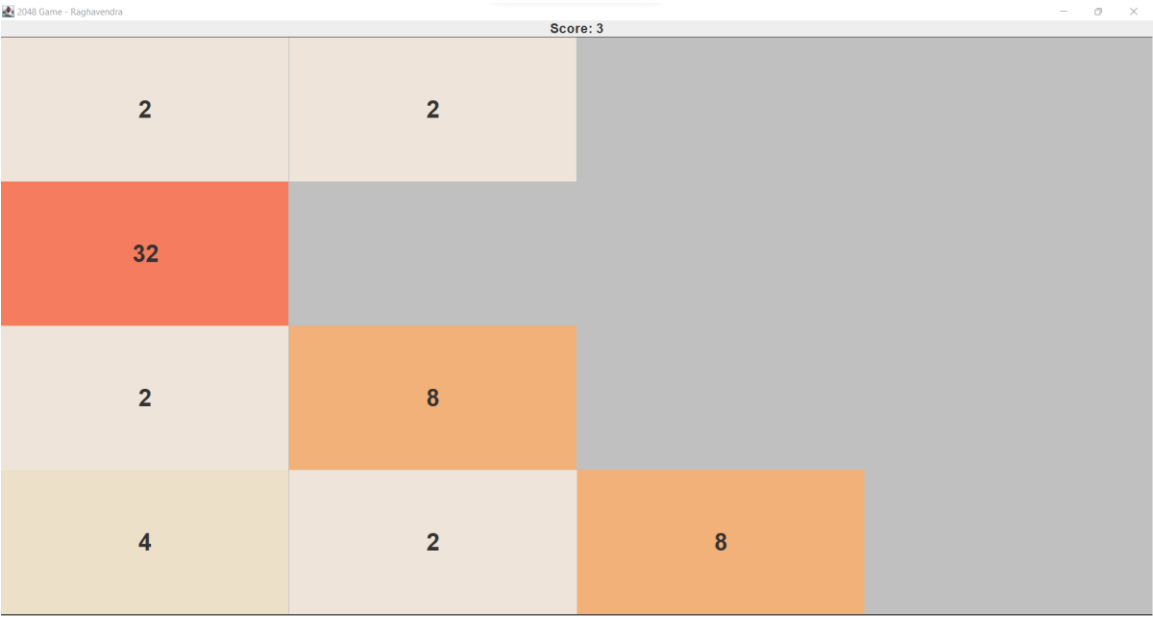
Welcome Screen:



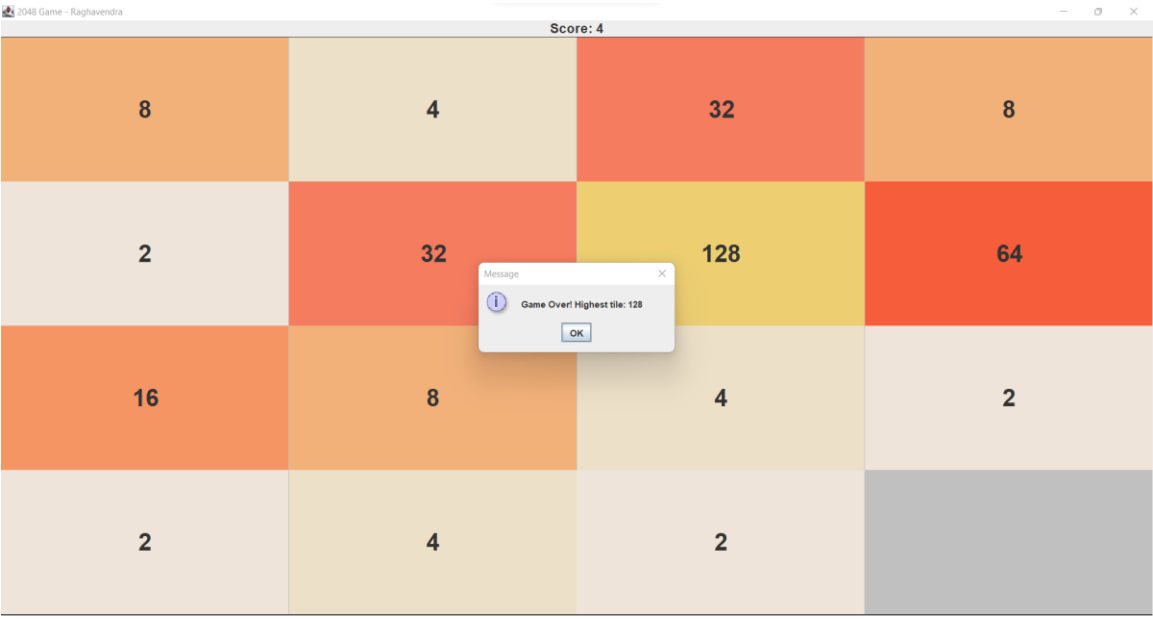
Started the game:



As player continues playing:



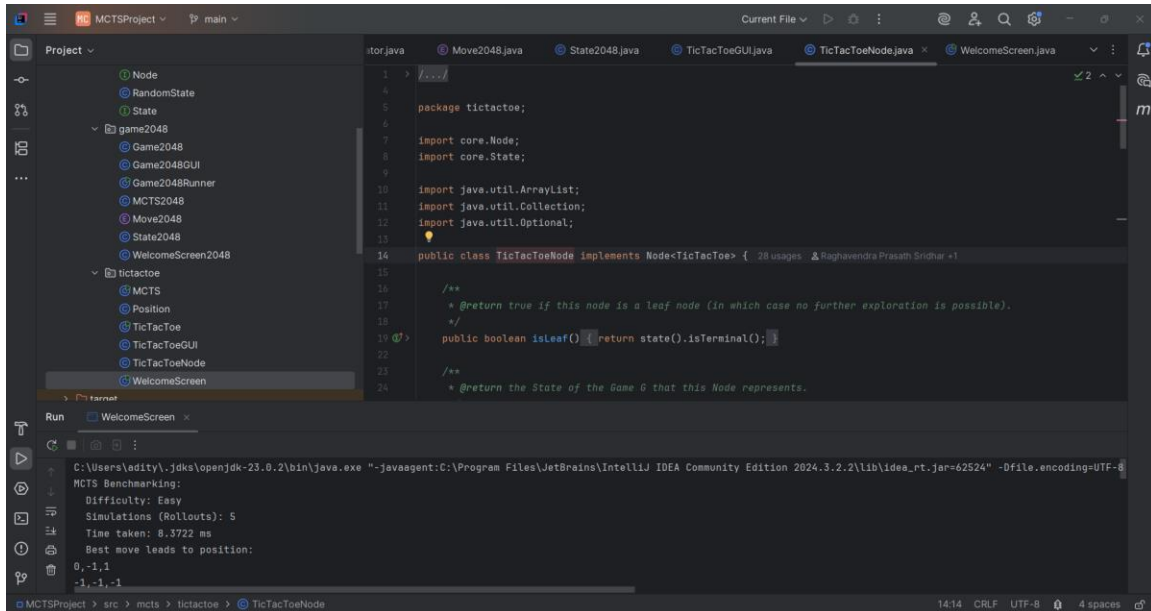
Outcome of the game displaying the highest score achieved by the player:



Performance Logging:

Tic-Tac-Toe:

Displays in terminal after each move by player:

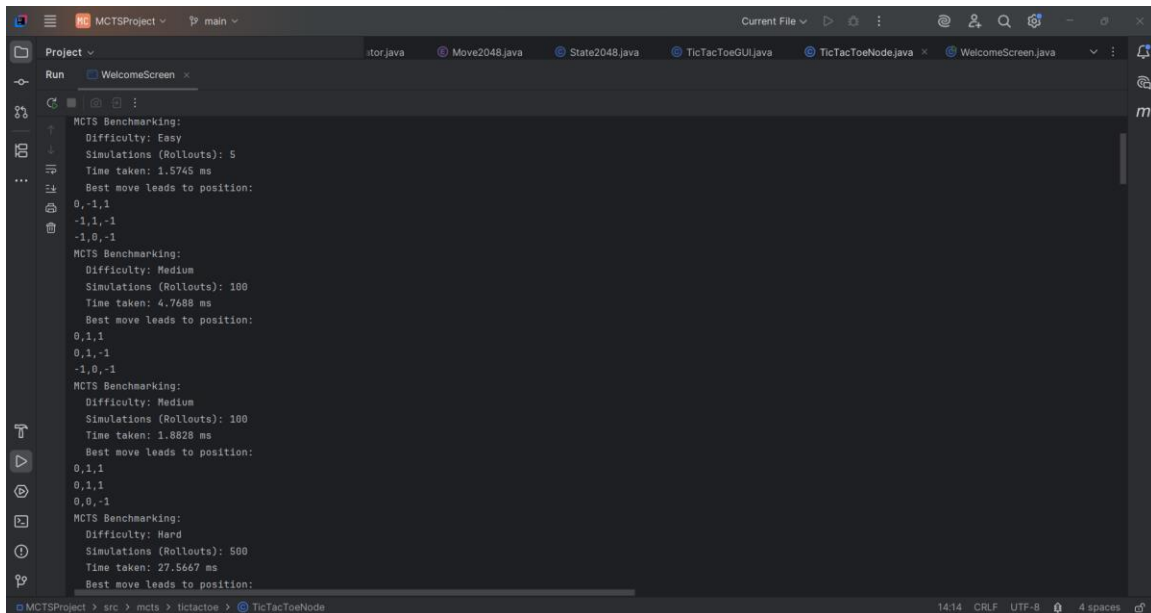


The screenshot shows an IDE with the following components:

- Project View:** A tree structure showing the project hierarchy, including folders like 'game2048', 'tictactoe', and 'WelcomeScreen2048'.
- Editor:** The 'TicTacToeNode.java' file is open, showing Java code for the 'TicTacToeNode' class. The code includes imports for 'core.Node', 'core.State', 'java.util.ArrayList', 'java.util.Collection', and 'java.util.Optional'. It defines a 'public class TicTacToeNode implements Node<TicTacToe>' with methods like 'isLeaf()' and 'state()'.
- Run Window:** A terminal window at the bottom displays the output of a Java application. It shows the command used to run the application and the resulting performance logs for the Tic-Tac-Toe game.

The terminal output shows the following performance logs for the Tic-Tac-Toe game:

```
C:\Users\adity\jdk\openjdk-23.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.2.2\lib\idea_rt.jar=62524" -Dfile.encoding=UTF-8
MCTS Benchmarking:
Difficulty: Easy
Simulations (Rollouts): 5
Time taken: 8.3722 ms
Best move leads to position:
0,-1,1
-1,-1,-1
```



The screenshot shows an IDE with the following components:

- Project View:** A tree structure showing the project hierarchy, including folders like 'game2048', 'tictactoe', and 'WelcomeScreen2048'.
- Editor:** The 'TicTacToeNode.java' file is open, showing Java code for the 'TicTacToeNode' class. The code includes imports for 'core.Node', 'core.State', 'java.util.ArrayList', 'java.util.Collection', and 'java.util.Optional'. It defines a 'public class TicTacToeNode implements Node<TicTacToe>' with methods like 'isLeaf()' and 'state()'.
- Run Window:** A terminal window at the bottom displays the output of a Java application. It shows the command used to run the application and the resulting performance logs for the Tic-Tac-Toe game.

The terminal output shows the following performance logs for the Tic-Tac-Toe game:

```
MCTS Benchmarking:
Difficulty: Easy
Simulations (Rollouts): 5
Time taken: 1.5745 ms
Best move leads to position:
0,-1,1
-1,1,-1
-1,0,-1
MCTS Benchmarking:
Difficulty: Medium
Simulations (Rollouts): 100
Time taken: 4.7688 ms
Best move leads to position:
0,1,1
0,1,-1
-1,0,-1
MCTS Benchmarking:
Difficulty: Medium
Simulations (Rollouts): 100
Time taken: 1.8828 ms
Best move leads to position:
0,1,1
0,1,1
0,0,-1
MCTS Benchmarking:
Difficulty: Hard
Simulations (Rollouts): 500
Time taken: 27.5667 ms
Best move leads to position:
```

2048:

Displays in terminal after each game:

