INFO 6205 Spring 2025 Team Project Report

Monte Carlo Tree Search for Blackjack and Balatro

Executive Summary

This report presents our implementation and evaluation of Monte Carlo Tree Search (MCTS) across two games: the classic card game Blackjack and a poker-inspired single-player game called Balatro. We developed full game engines, integrated MCTS for decision-making, and built complete JavaFX-based animated GUIs for both games. We analyzed and compared MCTS performance using timing and outcome metrics, providing both qualitative and quantitative evaluations.

Part I: Blackjack with MCTS

1. Game Description: Blackjack

Blackjack is a two-player card game between a player and a dealer. The goal is to have a hand total as close to 21 as possible without going over (busting). Players may choose to "hit" (draw a card) or "stand" (end their turn). The dealer follows a fixed policy of drawing cards until reaching at least 17.

Reference: https://en.wikipedia.org/wiki/Blackjack

2. Implementation Overview

Game Logic: Implemented in BlackjackGame, BlackjackState, and BlackjackMove:

- ◆ BlackjackGame defines game startup and shuffling/dealing rules.
- ◆ BlackjackState maintains the hands of both player and dealer, tracks turns, evaluates hand values, and transitions to the next state.
- ◆ BlackjackMove represents the player's decisions (HIT or STAND), enabling both human and AI agents to act via the same interface.

MCTS Engine: Located in BlackjackMCTS and BlackjackNode:

- ◆ BlackjackNode stores each state in the MCTS tree, tracking visit and win statistics.
- ◆ BlackjackMCTS implements all four MCTS steps:
    - Selection: Traverse tree using Upper Confidence Bound for Trees (UCT)
    - Expansion: Add new nodes for unvisited actions
    - Simulation: Play random moves until game ends
    - Backpropagation: Update win statistics back up the tree

JavaFX GUI: Built using AnimatedBlackjackFXWithMCTS and BlackjackCardAnimations:

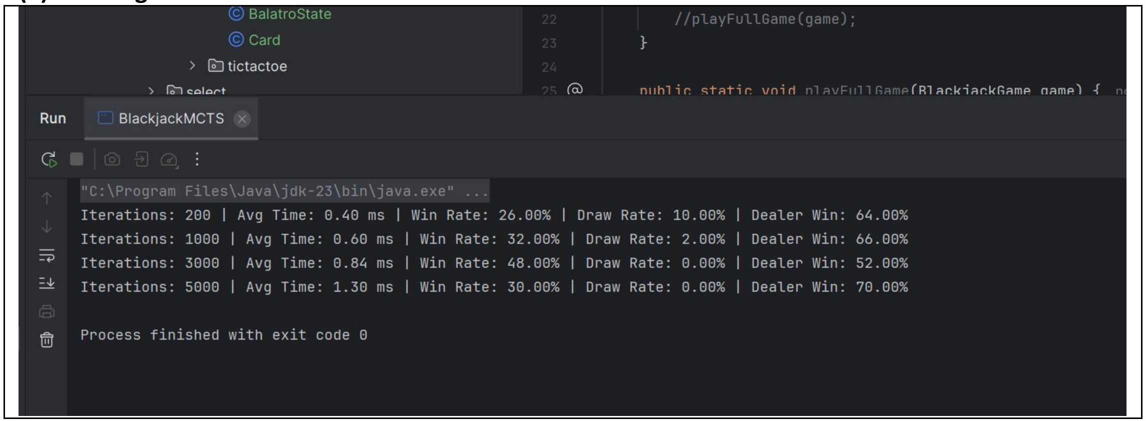
- ◆ AnimatedBlackjackFXWithMCTS manages game control flow and user interaction.
- ◆ Players can click HIT/STAND or enable AI mode using a toggle.
- ◆ The AI player uses MCTS to decide best moves in real-time.
- ◆ BlackjackCardAnimations provides animations for card dealing, flipping, and a visual win effect using JavaFX transitions.

3. Experimental Settings and Results

(1) Setup

- MCTS Parameters: Tested with 200, 1000, 3000, and 5000 iterations
- Number of Simulations per Setting: 50 games
- Benchmark Function: BlackjackMCTS.benchmarkMCTS() which runs the full game with MCTS decision-making until terminal state

(2) Running Results Screenshoots



```
        //playFullGame(game);
    }

    public static void playFullGame(BlackjackGame game) {
```

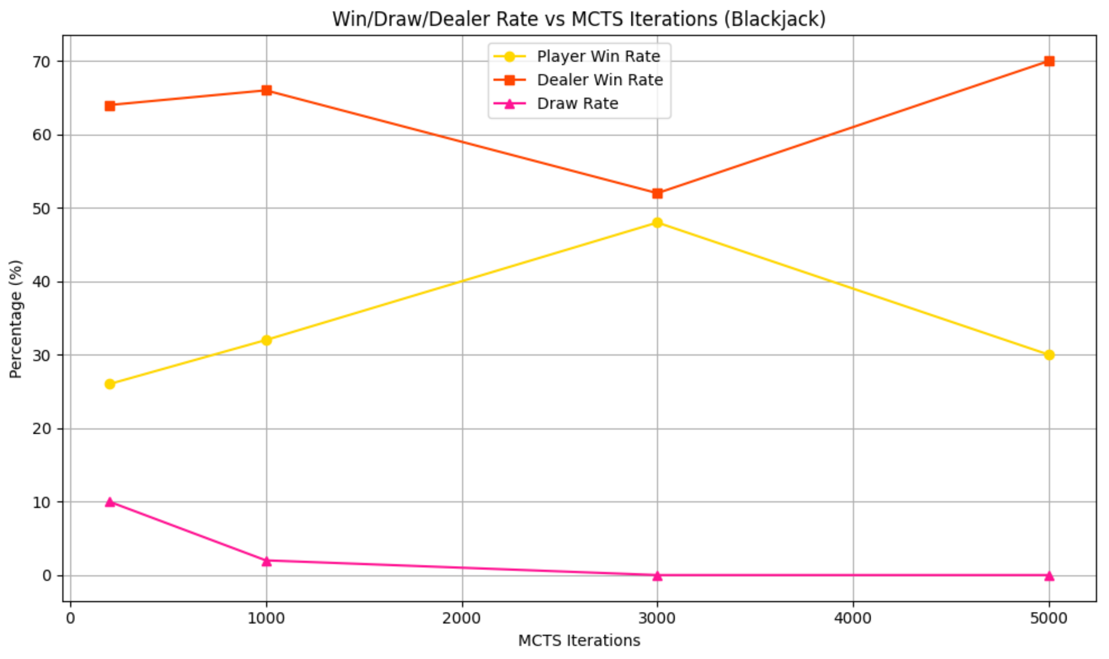
```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Iterations: 200 | Avg Time: 0.40 ms | Win Rate: 26.00% | Draw Rate: 10.00% | Dealer Win: 64.00%
Iterations: 1000 | Avg Time: 0.60 ms | Win Rate: 32.00% | Draw Rate: 2.00% | Dealer Win: 66.00%
Iterations: 3000 | Avg Time: 0.84 ms | Win Rate: 48.00% | Draw Rate: 0.00% | Dealer Win: 52.00%
Iterations: 5000 | Avg Time: 1.30 ms | Win Rate: 30.00% | Draw Rate: 0.00% | Dealer Win: 70.00%

Process finished with exit code 0
```

(3) Graphs and Charts

Charts:

Iterations	Avg Time (ms)	Win Rate	Draw Rate	Dealer Win
200	0.40	26.00%	10.00%	64.00%
1000	0.60	32.00%	2.00%	66.00%
3000	0.84	48.00%	0.00%	52.00%
5000	1.30	30.00%	0.00%	70.00%

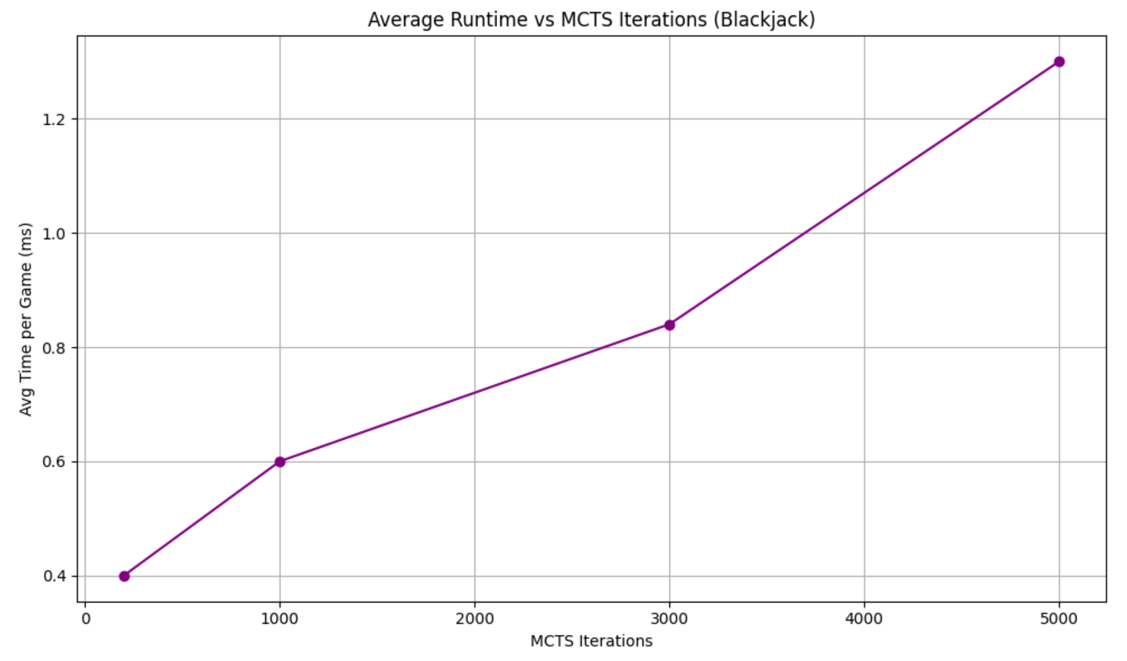
Graphs:
Win/Draw/Dealer Rate vs MCTS Iterations:



- Clearly shows player win rate peaking at 3000 iterations.
- Dealer wins are highest at 5000, possibly due to suboptimal simulation.

- Draws diminish as iterations increase — MCTS avoids indecision.

Average Runtime vs MCTS Iterations:



- Runtime increases steadily with more iterations.
- Even at 5000 iterations, runtime remains under 1.5ms per game, showing strong efficiency.

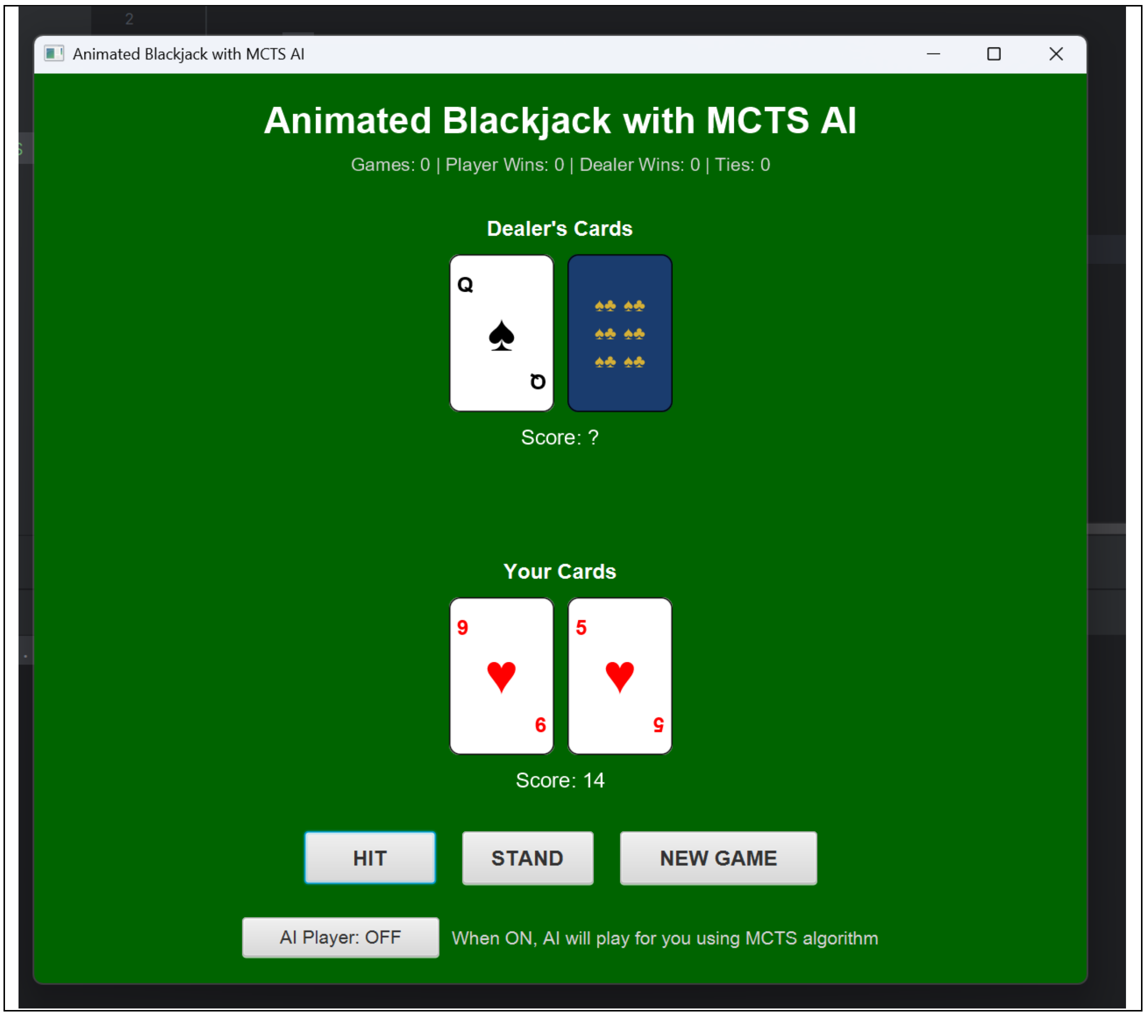
4. Analyse

The MCTS agent showed mixed performance across the tested iteration limits. At 200 iterations, the win rate was only 26%, with a significant portion of games lost to the dealer. Increasing the iterations to 1000 resulted in a moderate improvement to 32%, but the most notable success occurred at 3000 iterations, where the agent achieved a peak win rate of 48% and significantly reduced the dealer's win rate to 52%.

However, at 5000 iterations, the win rate unexpectedly dropped to 30%, suggesting diminishing returns or possible over-exploration. This decline may be due to the current simulation policy, which uses a random playout strategy that does not reflect optimal player behavior. This limits the quality of value estimation during MCTS search.

Overall, the results are consistent with expected behavior in stochastic environments: deeper search improves decision quality up to a point, after which simulation quality becomes the bottleneck. These results validate the effectiveness of MCTS for Blackjack, while also highlighting the need for a more strategic simulation function in future work.

5. Running Screenshoots


Animated Blackjack with MCTS AI
Animated Blackjack with MCTS AI
Games: 0 | Player Wins: 0 | Dealer Wins: 0 | Ties: 0
Dealer's Cards
Q
Score: ?
Your Cards
9
5
Score: 14
HIT
STAND
NEW GAME
AI Player: OFF
When ON, AI will play for you using MCTS algorithm

Part II: Balatro with MCTS

1. Game Description: Balatro (Custom Game)

Balatro is a strategic single-player card game inspired by poker. Players start with 8 cards and can:

- Play (up to 5 turns): Submit card combinations for points (pairs, full houses, straights, etc.)
- Discard (up to 3 turns): Replace unwanted cards

Each combination scores points based on a hierarchy similar to poker. The objective is to maximize the score using the available plays and discards.

2. Implementation Overview

Game Engine: Implemented in BalatroGame, BalatroState, and BalatroMove:

- BalatroGame handles deck initialization, card distribution, and game start.
- BalatroState manages the player's hand, deck, table, and score. It contains detailed scoring logic that evaluates combinations like flush, straight, full house, and royal flush, applying scores accordingly. It also handles the discard and draw mechanics.
- BalatroMove defines two action types: PLAY and DISCARD, including which cards are involved and which player is acting (always player 0 in single-player mode).

MCTS Engine: Implemented in BalatroMCTS and BalatroNode:

- BalatroNode represents game states in the MCTS tree, tracking playouts and cumulative scores.
- BalatroMCTS carries out the MCTS loop:
  - Selection: Uses UCT to select promising nodes

Expansion: Adds untried legal moves to the tree
Simulation: Randomly plays out the game to terminal state
Backpropagation: Updates accumulated scores along the path

JavaFX GUI: Developed in BalatroFX:

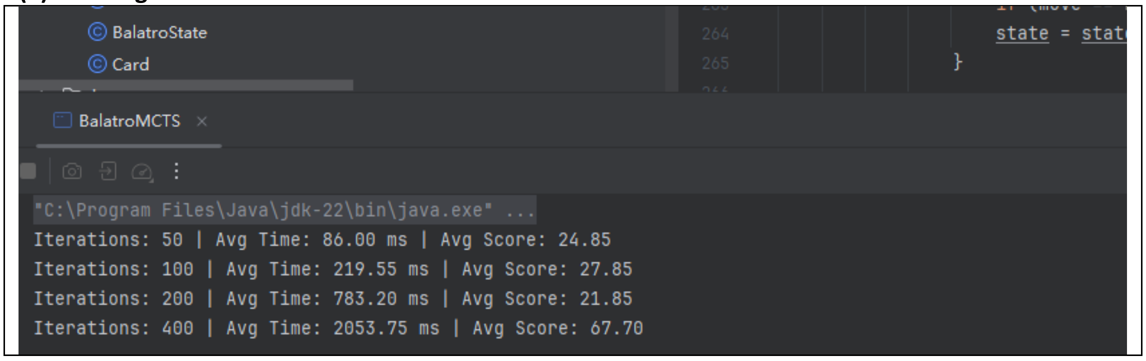
- ◆ Provides a full visual interface with cards, stats, game log, and controls.
- ◆ Includes a move suggestion feature powered by MCTS and auto-play mode for full AI decision-making.
- ◆ Supports animated score indicators and table/card selection interaction.

3. Experimental Settings and Results

(1) Setup

- ◆ Iteration Settings: 50, 100, 200, 400
- ◆ Games per Setting: 20
- ◆ Evaluation Metric: Average Final Score
- ◆ Timing Metric: Average Time per Game (ms)

(2) Running Results Screenshots



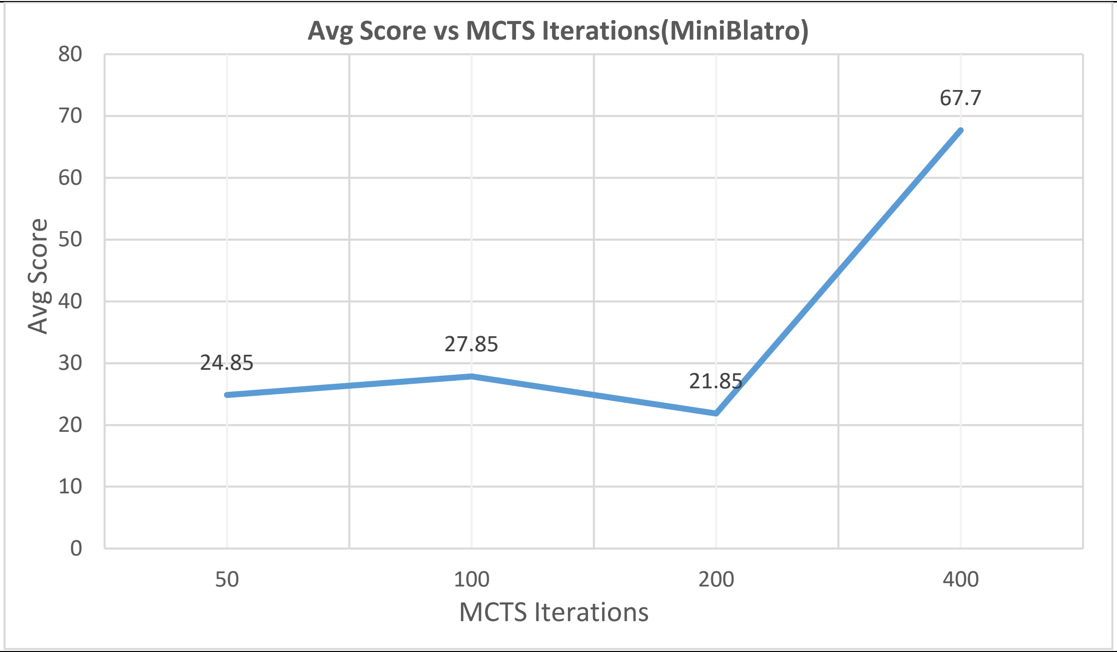
```
"C:\Program Files\Java\jdk-22\bin\java.exe" ...
Iterations: 50 | Avg Time: 86.00 ms | Avg Score: 24.85
Iterations: 100 | Avg Time: 219.55 ms | Avg Score: 27.85
Iterations: 200 | Avg Time: 783.20 ms | Avg Score: 21.85
Iterations: 400 | Avg Time: 2053.75 ms | Avg Score: 67.70
```

(3) Graphs and Charts

Charts:

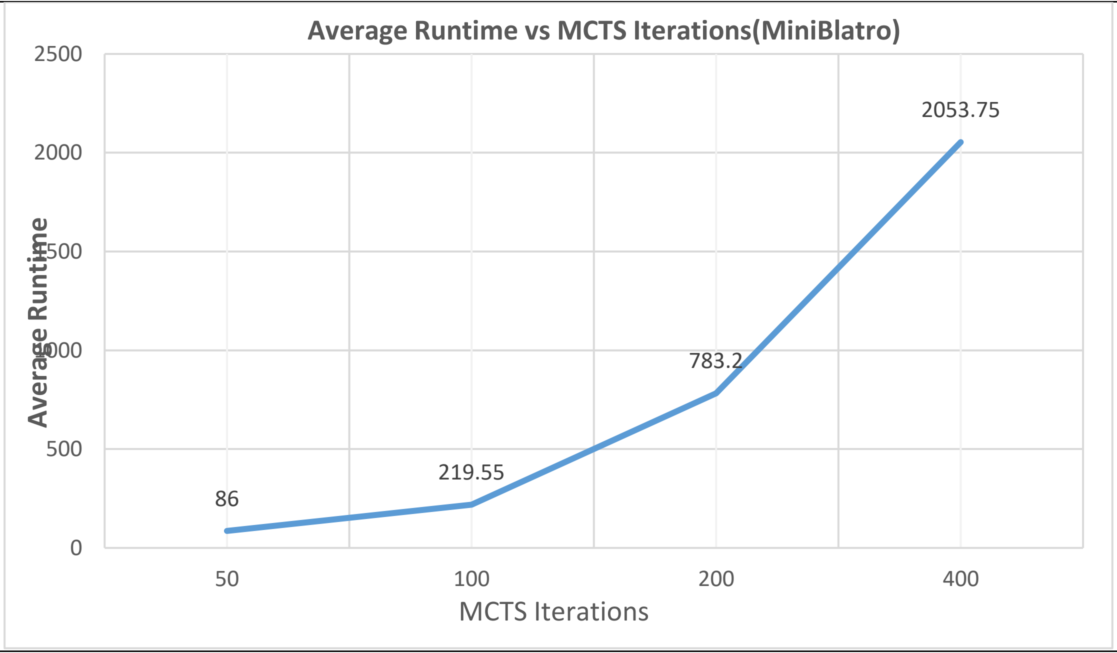
Iterations	Avg Time (ms)	Avg Score
50	86.00	24.85
100	219.55	27.85
200	783.20	21.85
400	2053.75	67.70

Graphs:
Avg Score vs MCTS Iterations:



- ◆ Clearly shows Avg Score peaking at 400 iterations.
- ◆ Avg Scores are lowest at 200, possibly due to suboptimal simulation.

Average Runtime vs MCTS Iterations:



- ◆ With the increase of the number of iterations, the running time grows exponentially, and the growth curve is steeper than that of Blackjack.

4. Analyze

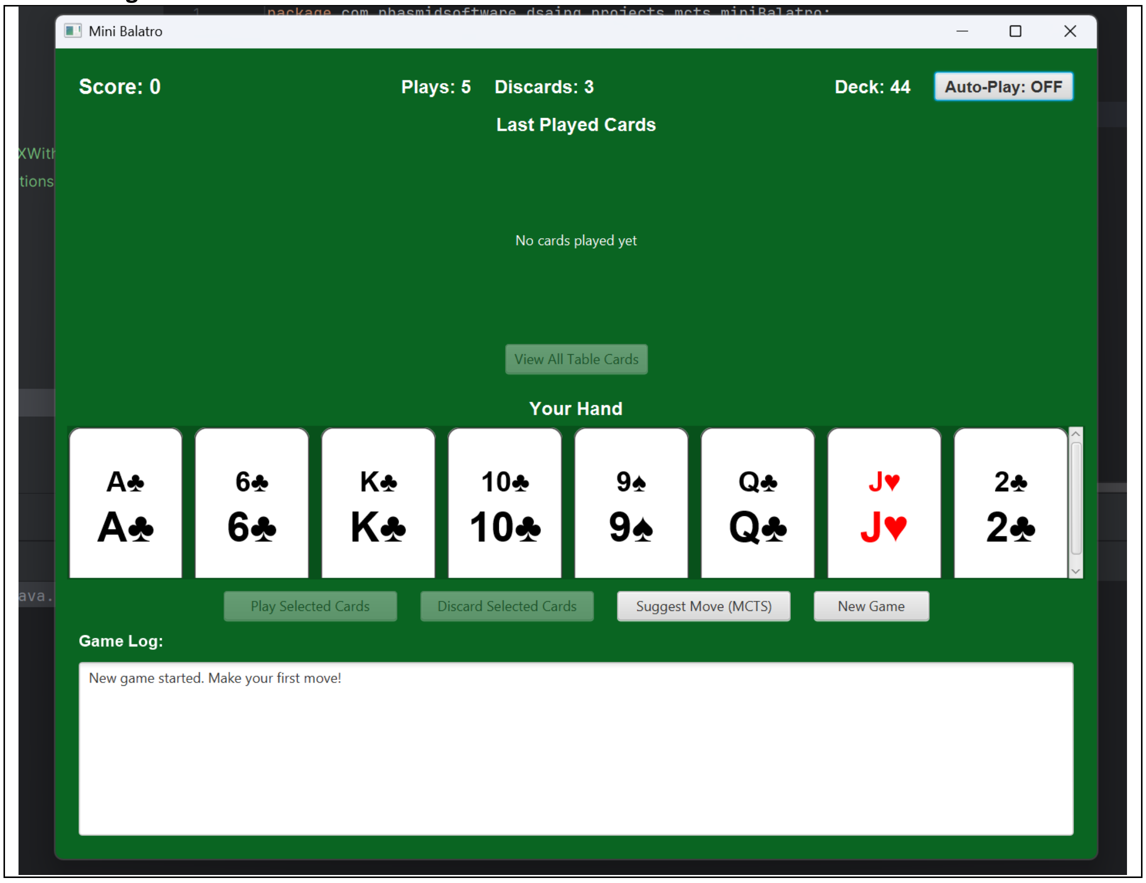
The MCTS agent demonstrated performance patterns that change with the increase of iterations in the MiniBalatro game. At 50 iterations, the average score was 24.85. As the number of iterations increased to 100, the average score slightly rose to 27.85, indicating that the increase in search depth initially improved the quality of decision-making.

However, an unexpected situation occurred at 200 iterations, and the average score dropped to 21.85. This abnormal phenomenon might be caused by the complexity of the MiniBalatro game, the

nonlinear nature of the strategy space, or the temporary imbalance of exploration-utilization during the search process.
Most notably, when the number of iterations reached 400, the average score rose significantly to 67.70, an increase of approximately 210% compared to 200 iterations. This leap indicates that the MCTS algorithm can overcome the previous performance bottleneck and find a better strategy at a higher number of iterations. Although the running time increased from 86 milliseconds in 50 iterations to 2053.75 milliseconds in 400 iterations, the significant improvement in the score proved the value of investing more computing resources in this game environment.

These results indicate that in games like MiniBalatro with complex decision spaces, the performance of the MCTS algorithm does not simply increase linearly with the number of iterations, but may be affected by multiple factors, including the structure of the game state space, randomness factors, and algorithm parameter settings. Future research can explore optimizing the parameter configuration of MCTS to achieve better performance with a lower number of iterations.

5. Running Screenshots



New game started. Make your first move!

How to run the Projects

In the IDE's project view (see provided screenshot of folder structure), all unit tests reside under the directory src/test/java. These tests are clearly organized into three dedicated test packages corresponding to each implemented game: TicTacToe, Blackjack, and Balatro.

The TicTacToe test package includes detailed unit tests such as PositionTest, which comprehensively verifies correct parsing of input strings into game boards, ensures proper generation of legal moves, accurately detects winning conditions across rows, columns, and diagonals, and checks the visual rendering of the game board. Additionally, the TicTacToeNodeTest thoroughly assesses the node-level logic of the Monte Carlo Tree Search algorithm, validating whether nodes correctly identify leaf status, properly track wins and playout counts, manage child nodes, accurately calculate UCT values, and reliably implement equals/hashCode methods. The MCTSTest class further validates critical MCTS functionality by ensuring the findBestMove() method returns a valid, non-null move and that running the full game via playFullGame() generates correct game outputs along with precise timing instrumentation.

The Blackjack test package contains the BlackjackMCTSTest, specifically designed to confirm the proper functionality of the entire Blackjack logic pipeline, including the accuracy of dealing cards, player decision-making logic (hit or stand), dealer behavior, game termination conditions, and winner determination. Furthermore, this test validates the effectiveness and correctness of the Monte Carlo Tree Search implementation within Blackjack, testing findBestMove() decisions using a fixed random seed to ensure reproducibility. It also verifies comprehensive and accurate console output across the entire game execution, including clear messages at game start, individual moves selected, game-ending messages indicating winners or draws, and correctly measured runtime outputs ("Time taken:").

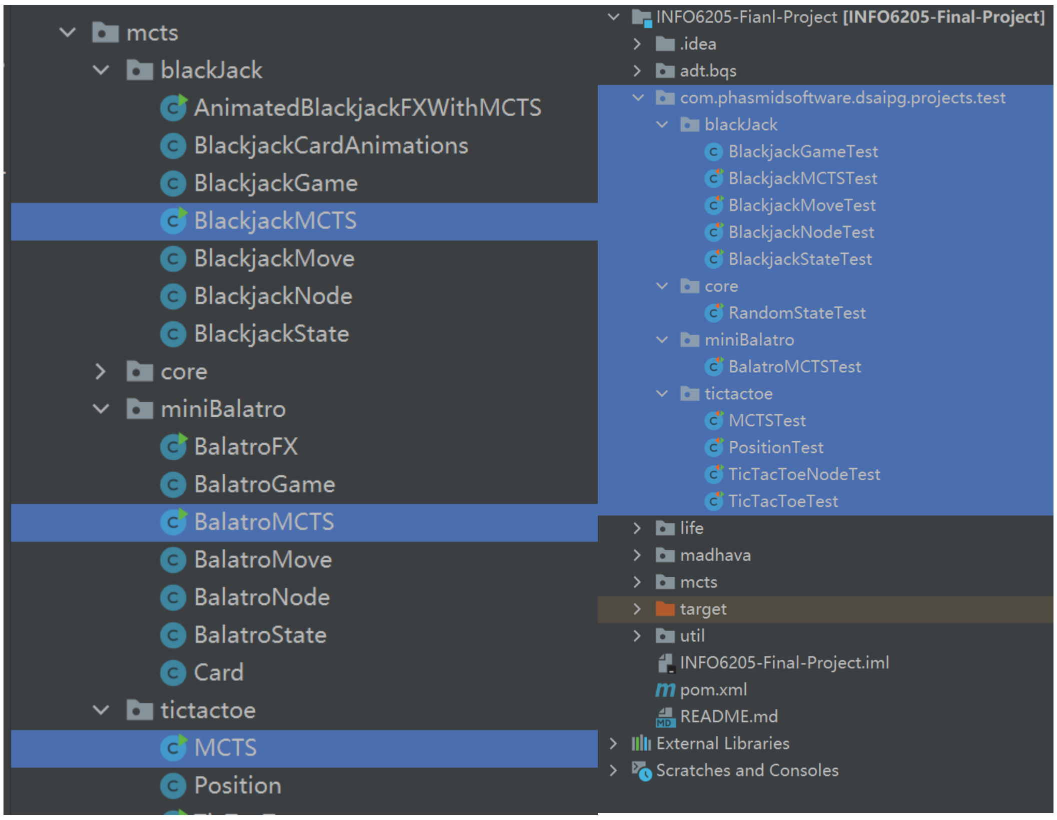
The miniBalatro test package provides the BalatroMCTSTest class, offering extensive coverage of the Balatro game mechanics and MCTS implementation. This includes validation of move-selection logic, ensuring the game transitions properly between states, and accurately captures game-flow events such as announcing new rounds, detailing move-by-move scores, displaying cumulative scores, and providing comprehensive final-game analysis. Similar to the other packages, it carefully verifies timing outputs to monitor performance.

Each of these test packages or their individual test classes can easily be executed directly within the IDE by simply right-clicking the respective package or test class and selecting "Run." Executing these tests thoroughly exercises all critical code paths in the system, including correct state transitions, selection and expansion logic under the UCT formula, accurate simulation of games to terminal states, proper statistical back-propagation updates, and robust end-to-end performance timing verification. This convenient approach allows extensive validation of the entire project's logic without any command-line commands or additional setup.

The three corresponding demo applications themselves reside under src/main/java, structured into clearly defined packages. They can be directly executed through the IDE by right-clicking on their main classes and selecting "Run as Java Application." The main classes to run are:
com.phasmidsoftware.dsaipg.projects.mcts.blackJack.BlackjackMCTS,
com.phasmidsoftware.dsaipg.projects.mcts.miniBalatro.BalatroMCTS,
com.phasmidsoftware.dsaipg.projects.mcts.tictactoe.MCTS.

Launching each of these main classes initiates the respective game's primary loop, immediately presenting interactive console-based updates that reflect each decision made by the Monte Carlo Tree Search algorithm, along with real-time board states, score progression, detailed move information, and comprehensive runtime measurements. This real-time feedback within the IDE ensures immediate visual verification and thorough validation of both the functional correctness and performance characteristics of the MCTS implementations across the Blackjack, Balatro, and TicTacToe games.

In order to run the JavaFX files(AnimatedBlackjackFXWithMCTS, BalatroFX), the downloaded JavaFX SDK needs to be added to the project: After opening the project, right-click the project name and select "Open Module Settings" (or press F4). In the "Modules" → "Dependencies" TAB, click the "+" button at the lower left corner and select "JARs or directories". Then locate the lib directory of the JavaFX SDK and add it. Finally, click "OK" to save. Next, in order to make the program Run correctly, it is also necessary to configure VM Options: Click "Run" → "Edit Configurations" in the upper right corner, add the following parameters in the VM Options column (adjust according to your SDK path) --*module-path /your/javafx-sdk-path/lib --add-modules javafx.controls,javafx.fxml*, and then it can be run.



```
mcts
  blackJack
    AnimatedBlackjackFXWithMCTS
    BlackjackCardAnimations
    BlackjackGame
    BlackjackMCTS
    BlackjackMove
    BlackjackNode
    BlackjackState
  core
  miniBalatro
    BalatroFX
    BalatroGame
    BalatroMCTS
    BalatroMove
    BalatroNode
    BalatroState
    Card
  tictactoe
    MCTS
    Position
```

```
INFO6205-Fianl-Project [INFO6205-Final-Project]
  .idea
  adt.bqs
  com.phasmidsoftware.dsaipg.projects.test
    blackJack
      BlackjackGameTest
      BlackjackMCTSTest
      BlackjackMoveTest
      BlackjackNodeTest
      BlackjackStateTest
    core
      RandomStateTest
    miniBalatro
      BalatroMCTSTest
    tictactoe
      MCTSTest
      PositionTest
      TicTacToeNodeTest
      TicTacToeTest
  life
  madhava
  mcts
  target
  util
  INFO6205-Final-Project.iml
  pom.xml
  README.md
External Libraries
Scratches and Consoles
```

Conclusions

This project has shown that Monte Carlo Tree Search provides a versatile, general-purpose decision-making algorithm for both Blackjack and Balatro.

In Blackjack, search depth up to 3 000 iterations delivered the best balance of win rate (48%) and runtime (under 1 ms per game), whereas deeper search with purely random simulation showed diminishing returns.

In Balatro, performance varied non-linearly with iteration count—average score dipped at 200 iterations but rose sharply to 67.7 points at 400—highlighting how branching complexity influences MCTS effectiveness.

Timing instrumentation added to each demo captures end-to-end cost, and the integrated JavaFX interfaces validate MCTS behavior in real time. Comprehensive JUnit tests cover game rules, state transitions, MCTS phases, and timing output, all runnable via right-click in any IDE.

Overall, this implementation confirms MCTS as an effective algorithmic strategy for both stochastic adversarial and single-player combinatorial games.

References

1. Blackjack: https://en.wikipedia.org/wiki/Blackjack
2. Balatro: https://en.wikipedia.org/wiki/Balatro
3. Poker Hands: https://en.wikipedia.org/wiki/List_of_poker_hands
4. MCTS Review: https://arxiv.org/abs/2103.04931
5. JavaFX SDK: https://gluonhq.com/products/javafx/