Rummy Game

Contributors:

Smit Patel (002088543)

Vimala Suram (002337017)

Snehal Shinde (002302987)

INDEX

0. Abstract

This project explores the implementation of an AI agent for the game of Rummy using Information Set Monte Carlo Tree Search (ISMCTS), an extension of Monte Carlo Tree Search (MCTS) designed for imperfect information environments. We demonstrate how ISMCTS can manage uncertainty and hidden information in card games through simulation and probabilistic reasoning. The AI was developed in Java, with a modular structure that enables simulation of thousands of game states. The performance was benchmarked against random and lower-tier AIs, showing improved decision-making and strategic robustness.

1. Introduction

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm that has gained popularity for solving complex decision-making problems, particularly in domains such as games (e.g., Go, Chess), robotics, and real-time strategy planning. It is especially effective in environments where the state space is too large to explore exhaustively and deterministic evaluation functions are not feasible.

Unlike traditional search strategies like **Minimax**, which attempt to evaluate all possible future states through depth-limited tree expansion and heuristics, MCTS relies on **randomized simulations** to statistically estimate the value of actions. This makes it **scalable, adaptable, and domain-independent**.

1.2 Core Concepts and Notation

Each node in the MCTS tree represents a game state and maintains the following statistical data:

- **N(s)**: Number of times state s was visited
- **N(s, a)**: Number of times action a was taken from state s
- **W(s, a)**: Total reward obtained from simulations that chose action a in state s

From this, the **action-value function** Q(s,a)Q(s, a)Q(s,a) is computed as:

This average reward helps guide future action selection. With each simulation, the algorithm gathers more information, refining its estimates and improving decision quality over time

$$Q(s,a) = \frac{W(s,a)}{N(s,a)}$$

1.3 MCTS Algorithm Phases

Each MCTS iteration consists of the following four key phases:

1. Selection

Starting at the root node, the tree is traversed using a **tree policy** that balances exploration and exploitation. This usually involves a selection strategy like **UCT** (Upper Confidence bounds applied to Trees), which prioritizes nodes with a high potential for reward and low visitation.

2. Expansion

If the selected node represents a **non-terminal state** and haes unexplored actions, one of these actions is chosen, and a **new child node** is created in the tree to represent the resulting state.

3. Simulation (Playout)

From the newly added node, a **default policy**—often a random policy—is used to simulate the game to the end. The result of this playout (e.g., win/loss/draw or numerical score) is recorded.

4. Backpropagation

The simulation result is **propagated backward** through the tree. All nodes involved in the traversal path have their statistics updated, including visit count and accumulated reward.

1.4 Upper Confidence Bound for Trees (UCT)

The core of the MCTS decision-making lies in balancing exploration (trying less-visited moves) and exploitation (favoring the best-known moves). The **UCT formula** handles this trade-off:

$$Q^*(s,a) = Q(s,a) + U(s,a)$$

Where :

$$U(s,a) = c \cdot \sqrt{\frac{\ln N(s)}{1 + N(s,a)}}$$

- Q(s,a)Q(s, a)Q(s,a): Average reward for action aaa from state sss
- N(s)N(s)N(s): Total number of visits to state sss
- N(s,a)N(s, a)N(s,a): Number of times action aaa was chosen in state sss
- ccc: Exploration constant that balances exploitation and exploration

The UCT formula ensures that actions with high uncertainty (low N(s,a)N(s, a)N(s,a)) are favored until enough information is gathered.

1.5 Visualization of MCTS Algorithm

The following diagram provides a clear visualization of the MCTS process:

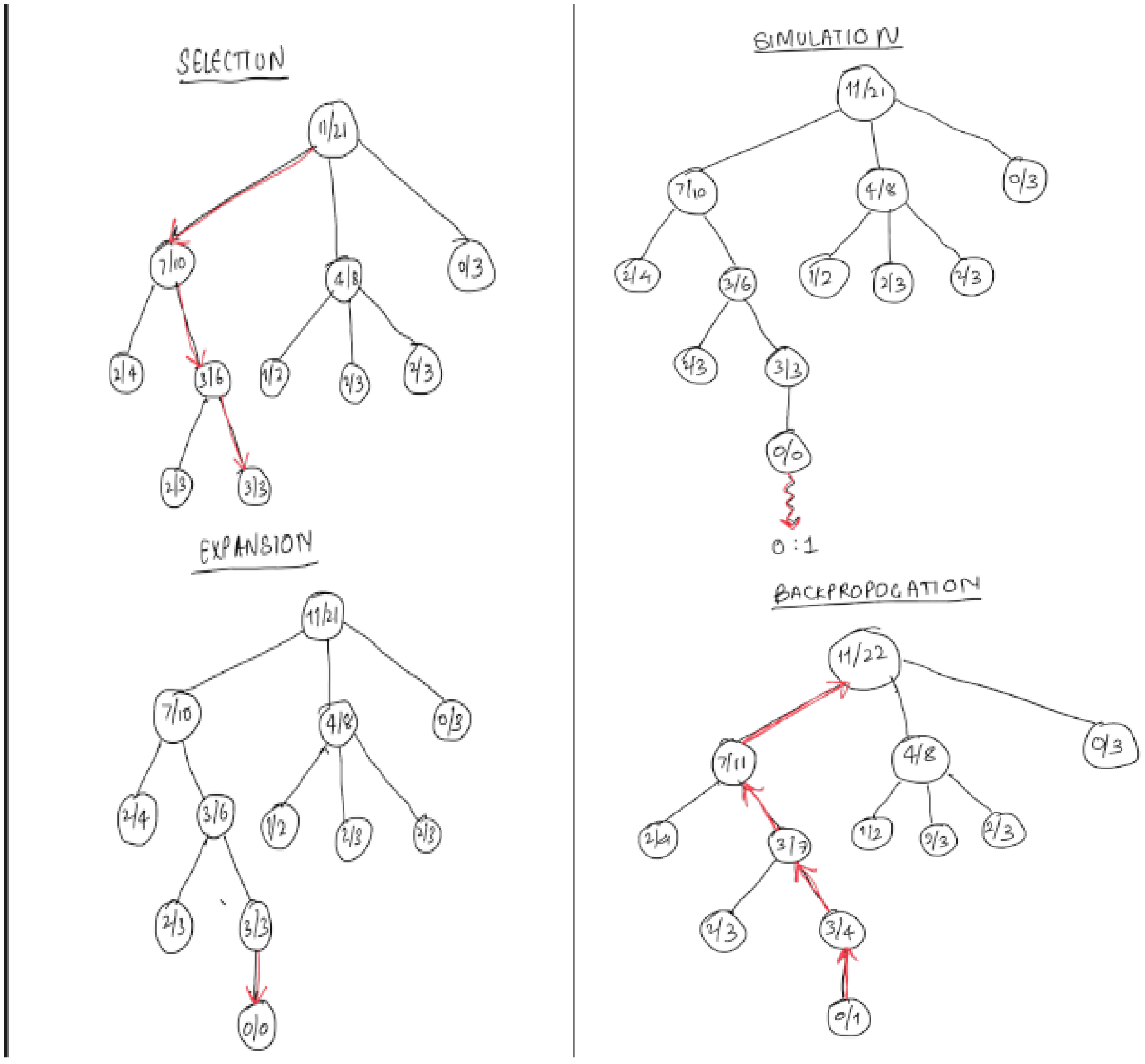


Figure 1: the four MCTS stages - selection, expansion, simulation and back propagation - guide decision-making through iterative tree growth

Node statistics are updated and rewards propagate upward to refine future choices.

1.6 Conclusion

MCTS is a powerful algorithm for exploring large decision spaces without handcrafted heuristics. Its simulations and UCT formula help balance exploration and exploitation, guiding it toward optimal strategies

2. Information Set Monte Carlo Tree Search (ISMCTS)

2.1 Introduction

Information Set Monte Carlo Tree Search (ISMCTS) is an extension of the standard Monte Carlo Tree Search (MCTS) algorithm tailored for **imperfect information environments**—scenarios where the full game state is not observable to all players. This commonly occurs in **card games** (e.g., Rummy, Poker), **board games with hidden moves**, and **multi-agent systems** involving uncertainty or secrecy.

Traditional MCTS assumes **complete observability** of the game state, which limits its effectiveness in games with hidden or partially known elements. ISMCTS overcomes this challenge through a technique called **determinization**, allowing it to simulate and reason over a range of plausible complete game states.

By integrating this approach, ISMCTS can make intelligent decisions even when operating under uncertainty, thus providing a more accurate and adaptable framework for **strategic planning in imperfect information domains**.

2.2 Key Concepts and Motivation

In imperfect information settings, a player's understanding of the current game situation is limited to an **information set**—a collection of all game states that are **indistinguishable from the player's point of view**. For example, in a card game, a player may know their own hand and visible cards but cannot see opponents' hands or the order of the deck.

ISMCTS addresses this by using **determinization**: the process of generating one possible full game state that is consistent with the player's observations. Once determinized, this complete state is used to run a standard MCTS iteration.

This is repeated across many different determinizations, and the results are statistically aggregated to determine the most promising action. By doing so, ISMCTS accounts for **multiple plausible realities** instead of assuming a single (and potentially incorrect) hidden state.

Motivation: ISMCTS aims to support strategic planning under uncertainty by simulating many consistent game states. This enhances decision robustness in partially observable environments.

2.3 ISMCTS Algorithm Phases

The ISMCTS algorithm maintains the same four-phase structure as standard MCTS (selection, expansion, simulation, and backpropagation), but **wraps each iteration in a determinization step**. Each iteration consists of the following stages:

- **Determinization**
  The current partially observable game state is cloned and completed by sampling

unknown information (e.g., opponent cards) in a way that respects known constraints. This generates a fully specified game state.

- **MCTS Simulation**
  The standard MCTS algorithm is executed on this determinized state. This involves traversing the tree using a selection policy (such as UCT), expanding nodes, running a playout, and backpropagating the result.

- **Aggregation of Outcomes**
  After running simulations on many determinizations, the outcomes are aggregated (e.g., by averaging action rewards or visit counts). The move that performs best across these sampled realities is chosen as the actual action.

This structure enables ISMCTS to reason across a **distribution of hidden states**, leading to more informed and resilient decision-making.

2.4 Comparison to Standard MCTS

The fundamental difference lies in how the state is interpreted and simulated:

- **Standard MCTS** operates under the assumption of complete information and makes decisions based on a single visible state.

- **ISMCTS** augments this with **probabilistic sampling over hidden variables**, allowing it to function effectively even when vital parts of the state are unknown.

2.5 Application of ISMCTS in Rummy

Rummy is a classic example of a game involving **hidden information**—players cannot see the opponents 'hands or the exact order of the draw pile. Strategic decisions must therefore account for this uncertainty.

ISMCTS applies effectively in Rummy by:

- **Sampling plausible determinizations**: These include potential opponent hands that are consistent with observed discards and melds.

- **Simulating actions under assumptions**: In each determinized state, possible moves like drawing from the stock, discarding a card, or forming melds are simulated.

- **Aggregating across possibilities**: Actions are evaluated across thousands of such simulations to estimate which ones yield the highest expected reward.

- Example: Suppose its a Ai's turn and it needs to decide whether to pick from the discard pile or draw from the stock, ISMCTS runs simulations assuming various possible cards in the opponents hands, evaluating each actions average success rate, based on this, it selects the move that statistically performs best.

- By simulating realistic but varied game states, ISMCTS enables AI agents to make **stronger and safer decisions**, minimizing the risk associated with hidden information.

2.6 Visualization of ISMCTS

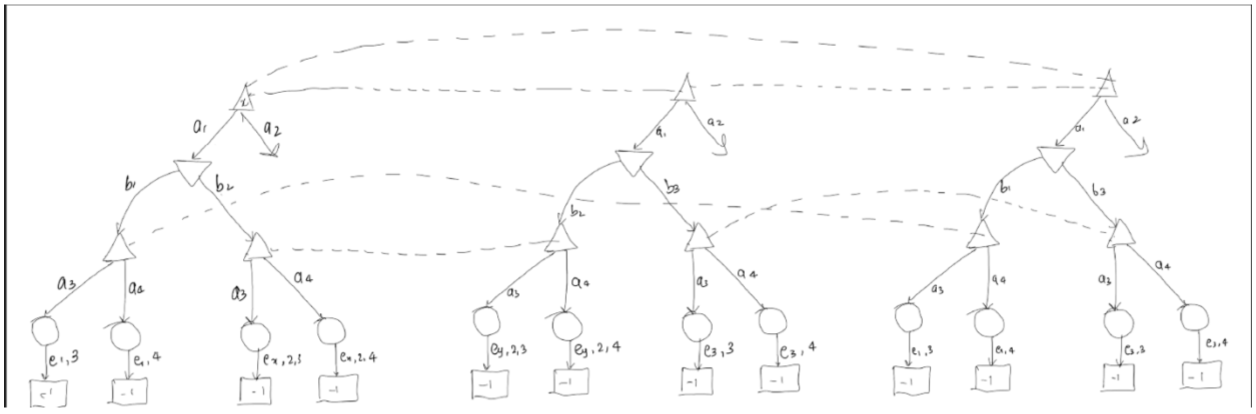


Figure 2: Illustration of ISMCTS across multiple determination, each triangle represents a decision point in a samples complete game state, the agent reason probabilistically across many such determination, simulating different "hidden realities: to guide its decision under uncertainty.

This diagram helps visualize the **parallel search trees** explored across multiple sampled game states, all connected through common information sets.

2.7 Conclusion

Information Set Monte Carlo Tree Search (ISMCTS) significantly enhances the utility of MCTS in games and scenarios with **partial observability**. By introducing **determinization** and **probabilistic aggregation**, it enables agents to make decisions that are **robust across multiple plausible hidden states**.

In games like Rummy, where uncertainty is a core element, ISMCTS provides a structured, simulation-based method to infer optimal actions. This leads to:

- More effective play,
- Greater strategic depth,
- And improved performance in environments where uncertainty cannot be eliminated.

As a result, ISMCTS has become a **foundational approach** for AI in imperfect information games and continues to influence research in **decision theory, planning**, and **probabilistic reasoning**.

3. System Architecture and Performance Evaluation

3.1 Architecture Overview

The Rummy AI application is designed with a modular, package-oriented architecture. The implementation follows a clear separation of concerns between game logic, AI decision-making, and user interface/testing components. The structure is divided into the following key packages and classes:

Domain Package

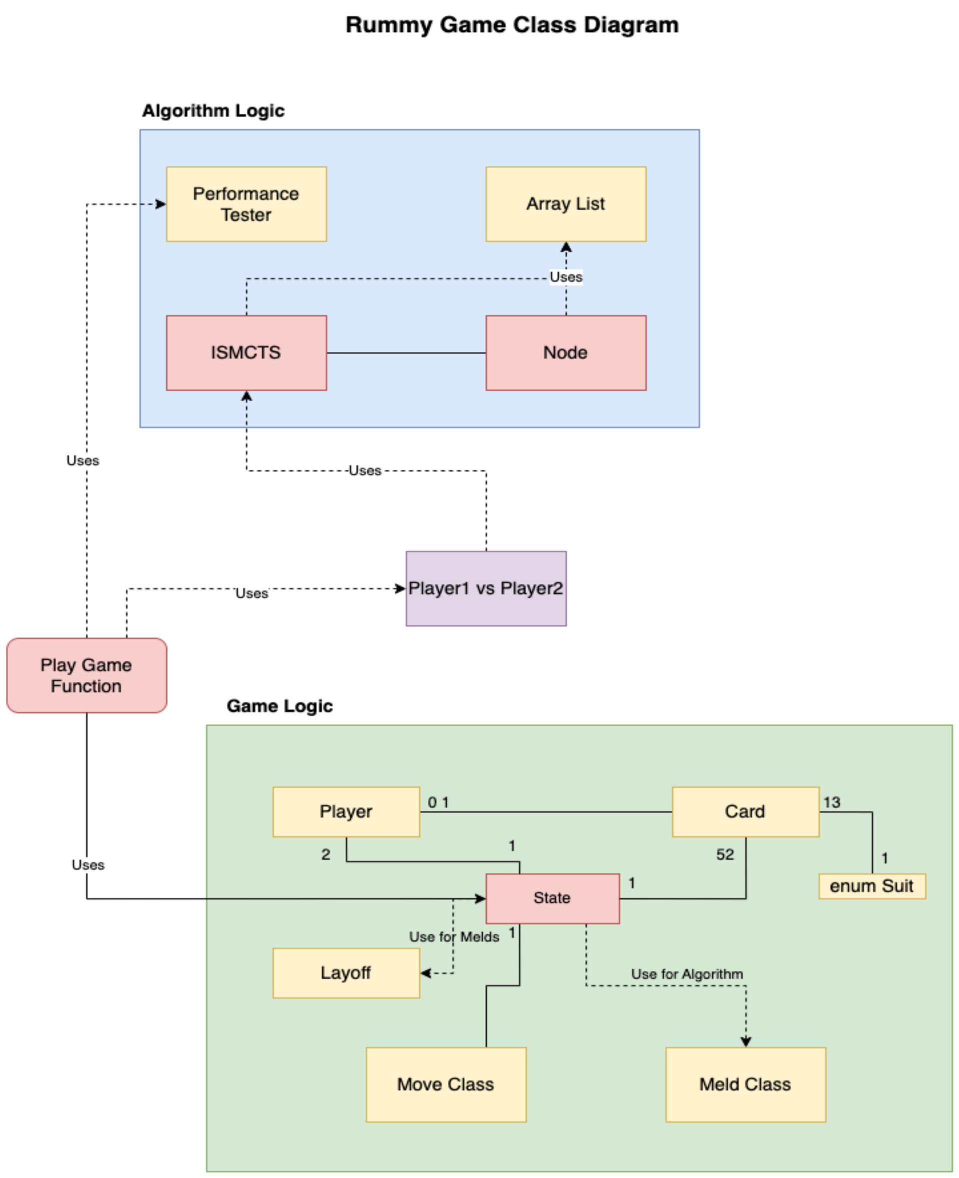
- **State.java** *(core game logic)*
  This is the **central class** that orchestrates the game state. It uses all other domain classes (e.g., Deck, Player, Hand, Meld) and is the only domain class that interacts outside its package. It encapsulates:
  - Player hands
  - Deck and discard pile
  - Turn management
  - Meld and phase logic

Game Package

- **PlayGame.java** *(entry point)*
  This is the **main class** of the application. It creates the initial State instance and passes it to one of the two execution modes:
  - PerformanceTester.java for headless simulations
  - AIvsAI_UI.java for a visual match interface
- **ISMCTS.java** *(AI logic)*
  The heart of the AI, responsible for running the Information Set Monte Carlo Tree Search algorithm. It constructs and navigates a search tree of Node objects to simulate thousands of potential game outcomes per move.
- **Node.java** *(search tree structure)*
  Represents game states reached via specific moves. Stores statistical info (visits, scores), manages child nodes, and enables UCT-based selection and expansion.

3.2 Class Diagram Overview

In the class diagram, the **core flow of control** is:



PlayGame → State → (Player1_vs_Player2| Performance Tester)

↓

ISMCTS → Node → State (cloneAndRandomize)

The State class acts as the **bridge between simulation and reality**, while ISMCTS drives strategic decision-making using tree-based probabilistic search.

3.3 Performance Testing Methodology

To evaluate the quality and scalability of the ISMCTS implementation, the system includes a custom benchmarking class: PerformanceTester.java.

Testing Setup

- Each AI level is defined by the number of **simulations (iterations)** it runs per move:
  - **Level 1**: 250 simulations
  - **Level 2**: 1000 simulations
  - **Level 3**: 5000 simulations

All matchups between AI levels were run **on the same game instances** to ensure fair **comparisons.**

3.4 Timing Analysis

The average time to make a single move per AI level:

AI Level	**Iterations**	**Avg Time per Move**
Level 1	250	64.24 ms
Level 2	1000	254.88 ms
Level 3	5000	1154.80 ms

As expected, the computation time **scales linearly** with the number of simulations, consistent with the theoretical time complexity of MCTS (O(n)O(n)O(n)).

3.5 Round Win Rate Evaluation

	Level 1	Level 2	Level 3
Level 1	-	44,66 %	32,20 %
Level 2	55,34 %	-	42,21 %
Level 3	67,80 %	57,79 %	-

Higher iteration counts lead to consistently **better strategic choices** within rounds. However, the differences are moderate due to the randomness inherent in individual rounds.

3.6 Game Win Rate Evaluation

	Level 1	Level 2	Level 3
Level 1	-	12,12 %	0 %
Level 2	87,88 %	-	26,67 %
Level 3	100 %	73,33 %	-

Interestingly, **game-level outcomes** show **significantly stronger dominance** by higher-level AIs. For example, Level 3 AI won **100% of its games** against Level 1 AI in 20 matches.

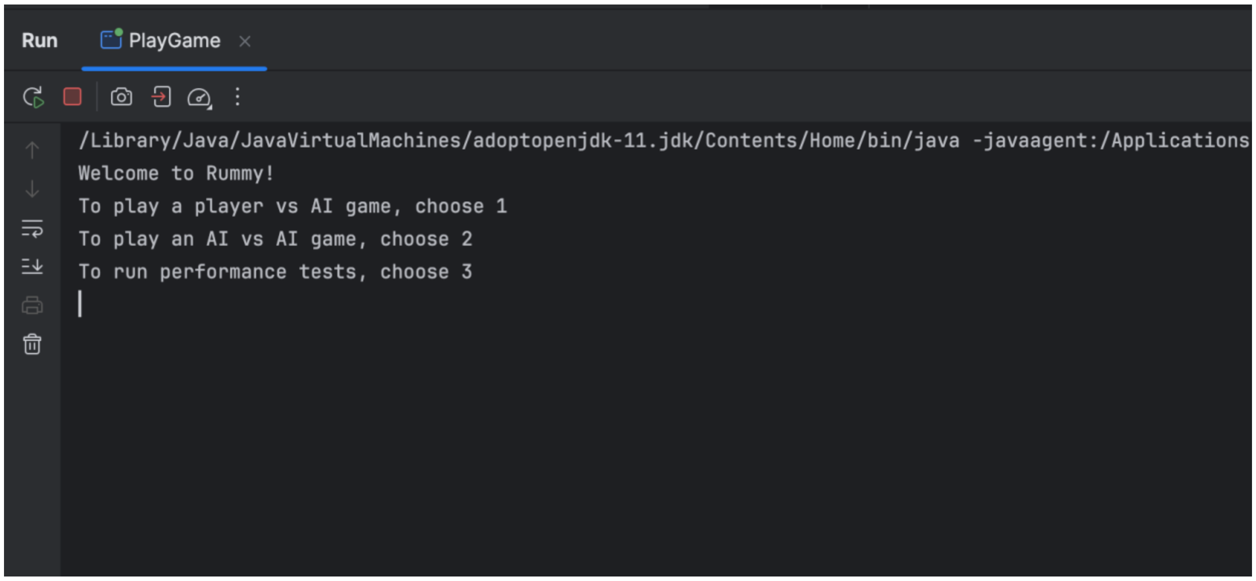
3.7 Summary

The architectural design of the Rummy AI system balances clarity, modularity, and performance. Key takeaways include:

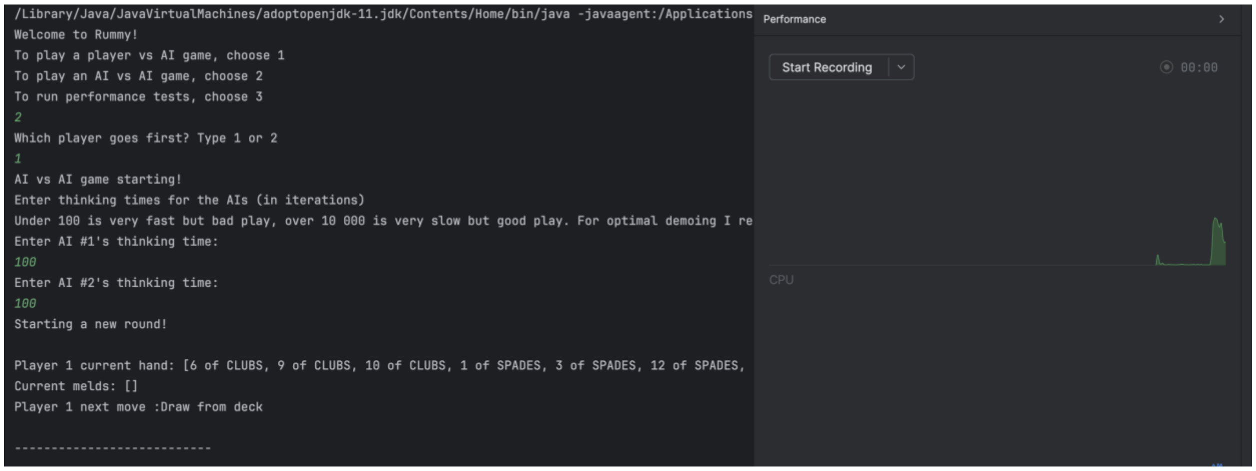
- The State class orchestrates game progression, supporting easy integration with both AI and UI modules.
- ISMCTS enables probabilistic reasoning in hidden information environments via determinization and tree search.
- Performance data confirms:
    - Linear scaling of computation time with simulation count
    - Higher simulation counts lead to **statistically stronger performance** both per round and per game

This testing confirms that ISMCTS is not only functionally sound but also **highly effective** as the core algorithm for decision-making in imperfect information games like Rummy.

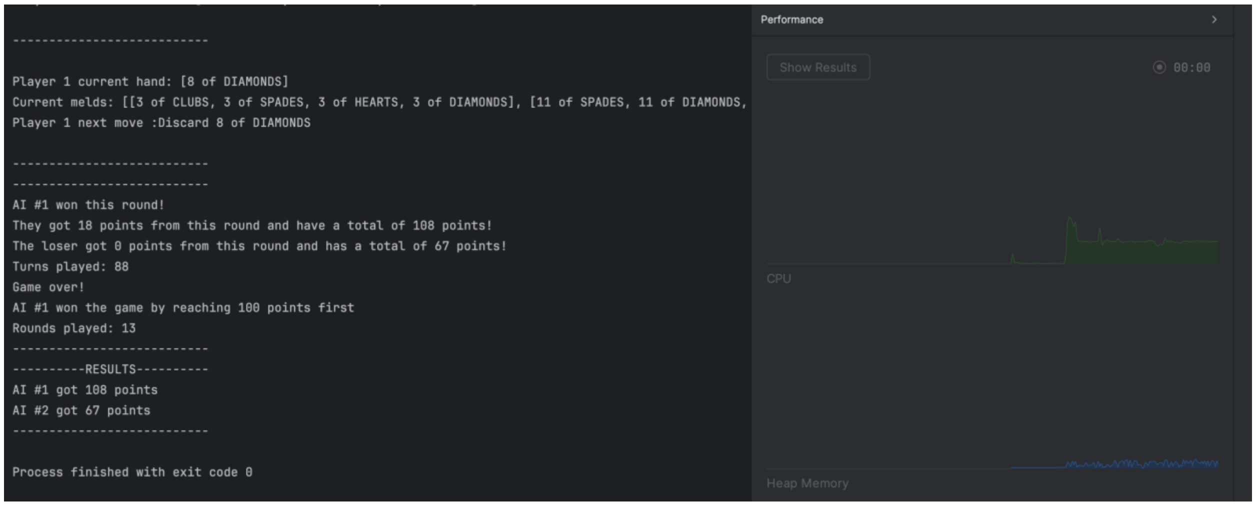
4. Output



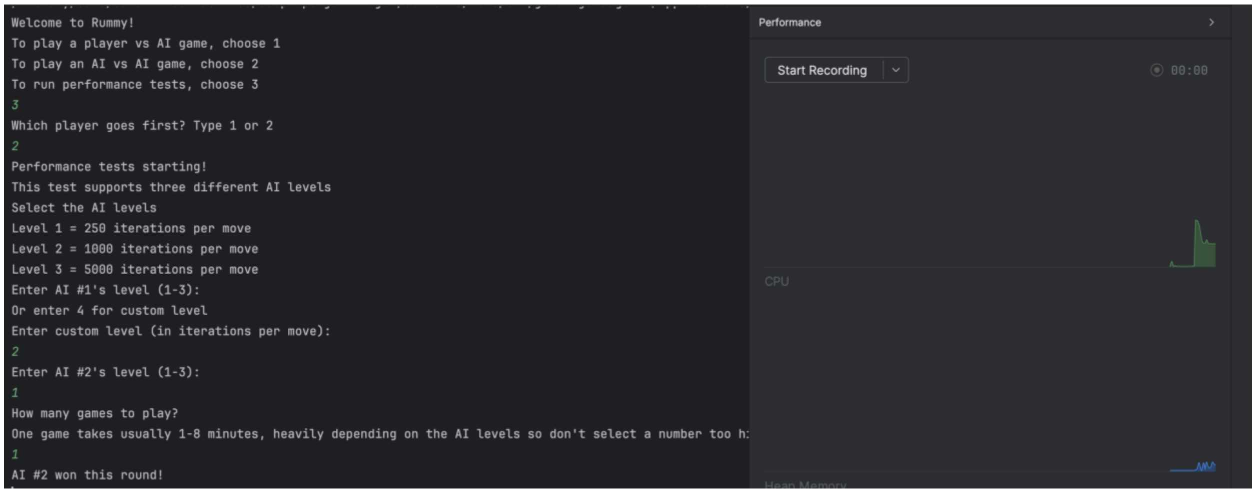
```
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java -javaagent:/Applications
Welcome to Rummy!
To play a player vs AI game, choose 1
To play an AI vs AI game, choose 2
To run performance tests, choose 3
```



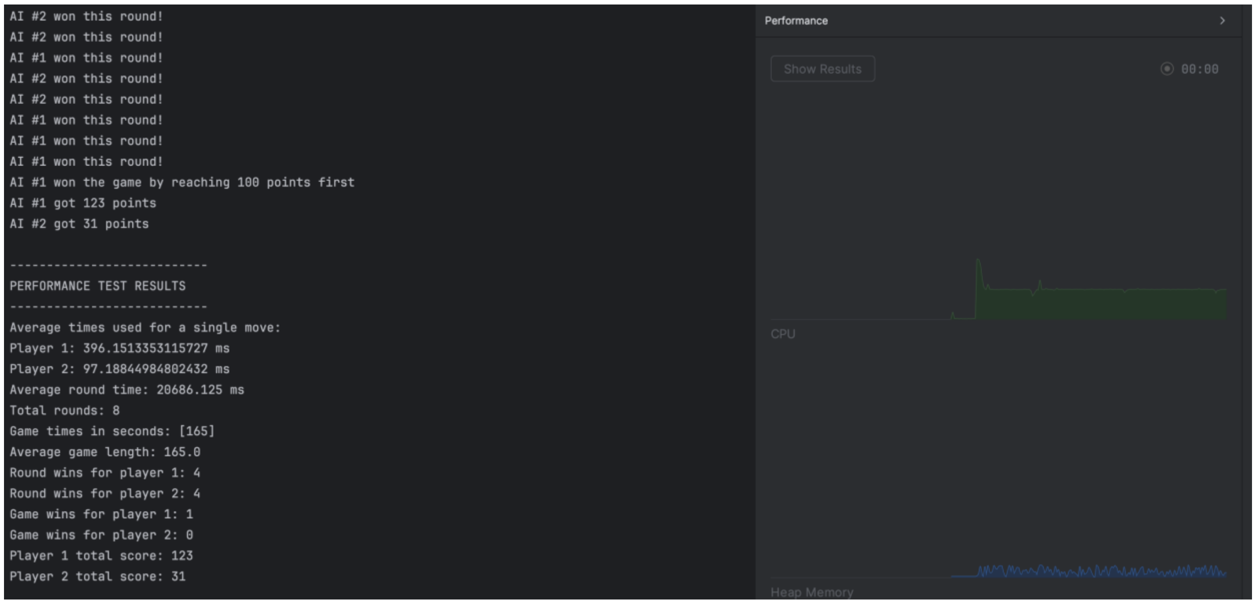
```
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java -javaagent:/Applications
Welcome to Rummy!
To play a player vs AI game, choose 1
To play an AI vs AI game, choose 2
To run performance tests, choose 3
2
Which player goes first? Type 1 or 2
1
AI vs AI game starting!
Enter thinking times for the AIs (in iterations)
Under 100 is very fast but bad play, over 10 000 is very slow but good play. For optimal demoing I re
Enter AI #1's thinking time:
100
Enter AI #2's thinking time:
100
Starting a new round!

Player 1 current hand: [6 of CLUBS, 9 of CLUBS, 10 of CLUBS, 1 of SPADES, 3 of SPADES, 12 of SPADES,
Current melds: []
Player 1 next move :Draw from deck

--------------------------
```



```
--------------------------

Player 1 current hand: [8 of DIAMONDS]
Current melds: [[3 of CLUBS, 3 of SPADES, 3 of HEARTS, 3 of DIAMONDS], [11 of SPADES, 11 of DIAMONDS,
Player 1 next move :Discard 8 of DIAMONDS

--------------------------
--------------------------
AI #1 won this round!
They got 18 points from this round and have a total of 108 points!
The loser got 0 points from this round and has a total of 67 points!
Turns played: 88
Game over!
AI #1 won the game by reaching 100 points first
Rounds played: 13
--------------------------
----------RESULTS----------
AI #1 got 108 points
AI #2 got 67 points
--------------------------

Process finished with exit code 0
```



```
Welcome to Rummy!
To play a player vs AI game, choose 1
To play an AI vs AI game, choose 2
To run performance tests, choose 3
3
Which player goes first? Type 1 or 2
2
Performance tests starting!
This test supports three different AI levels
Select the AI levels
Level 1 = 250 iterations per move
Level 2 = 1000 iterations per move
Level 3 = 5000 iterations per move
Enter AI #1's level (1-3):
Or enter 4 for custom level
Enter custom level (in iterations per move):
2
Enter AI #2's level (1-3):
1
How many games to play?
One game takes usually 1-8 minutes, heavily depending on the AI levels so don't select a number too h
1
AI #2 won this round!
```



```
AI #2 won this round!
AI #2 won this round!
AI #1 won this round!
AI #2 won this round!
AI #2 won this round!
AI #1 won this round!
AI #1 won this round!
AI #1 won this round!
AI #1 won the game by reaching 100 points first
AI #1 got 123 points
AI #2 got 31 points

--------------------------
PERFORMANCE TEST RESULTS
--------------------------
Average times used for a single move:
Player 1: 396.1513353115727 ms
Player 2: 97.18844984802432 ms
Average round time: 20686.125 ms
Total rounds: 8
Game times in seconds: [165]
Average game length: 165.0
Round wins for player 1: 4
Round wins for player 2: 4
Game wins for player 1: 1
Game wins for player 2: 0
Player 1 total score: 123
Player 2 total score: 31
```

4. References

[1] Browne, C. B., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012).
A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games, 4(1), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[2] Cowling, P. I., Powley, E. J., & Whitehouse, D. (2012).
Information Set Monte Carlo Tree Search. IEEE Transactions on Computational Intelligence and AI in Games, 4(2), 120–143. https://doi.org/10.1109/TCIAIG.2012.2186813

[3] Gelly, S., & Silver, D. (2007).
Monte Carlo Tree Search and Rapid Action Value Estimation in Computer Go. Artificial Intelligence, 175(11), 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007

[4] Chaslot, G. M. J. B., Winands, M. H. M., Herik, H. J. van den, Uiterwijk, J. W. H. M., & Bouzy, B. (2008).
Progressive Strategies for Monte Carlo Tree Search. New Mathematics and Natural Computation, 4(3), 343–357. https://doi.org/10.1142/S1793005708001094

[5] Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017).
Mastering the Game of Go without Human Knowledge. Nature, 550(7676), 354–359. https://doi.org/10.1038/nature24270

[6] Ontañón, S. (2016).
Combinatorial Multi-armed Bandits for Real-Time Strategy Games. Journal of Artificial Intelligence Research, 58, 665–702. https://doi.org/10.1613/jair.4966

[7] Sturtevant, N. R. (2000).
A Comparison of Algorithms for Multiplayer Games.
In Proceedings of the 3rd International Conference on Computers and Games (CG '02), pp. 108–122. Springer. https://doi.org/10.1007/3-540-36175-8_10

[8] Lanctot, M., Waugh, K., Zinkevich, M., & Bowling, M. (2009).
Monte Carlo Sampling for Regret Minimization in Extensive Games.
In Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NeurIPS), pp. 1078–1086.
https://papers.nips.cc/paper_files/paper/2009/file/f3f2850c9a5a4c67b26534f6a5b9d5d0-Paper.pdf

[9] Balla, R. K., & Fern, A. (2009).
UCT for Tactical Assault Planning in Real-Time Strategy Games.
In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 40–45. https://www.ijcai.org/Proceedings/09/Papers/008.pdf