Model-Based Monte Carlo Tree Search for Pacman

Aarin Shah*, Asish Addanki*, Rushir Bhavsar*
*SCAI, Arizona State University, Tempe, United States

***Abstract*—Monte-Carlo Tree Search (MCTS) is an innovative search method that has shown impressive results in two-player board games by enhancing simulation capabilities and reducing the need for detailed domain knowledge. This project applies a simulation-based version of the MCTS algorithm to the strategic gameplay of Pacman and Ghost. Through the use of T-tests, we demonstrate that MCTS performs better than traditional search algorithms such as MiniMax and ExpectiMax, even with very little domain knowledge. Our findings, supported by T-test statistical analysis, prove the effectiveness of MCTS across various game layouts. Additionally, we explore how MCTS improves decision-making in the game, offering insights into its potential benefits for complex strategy games.**

***Index Terms*—Monte-Carlo Tree-Search, Simulation, Pacman-Game, Search Techniques, T-test Statistics, Game-Agents**

I. Introduction

Search algorithms are pivotal in artificial intelligence, enabling systems to make decisions by exploring possible outcomes to find the best action. These algorithms are fundamental to game-playing AI and are employed to navigate the vast landscape of possible moves and counter-moves. Among these, tree search algorithms stand out for their ability to systematically explore game trees to identify winning strategies in two-player zero-sum games.

Minimax algorithms stand out for their effectiveness in two-player zero-sum games, seamlessly analyzing game trees to identify winning moves. This recursive algorithm, by design, minimizes potential loss in worst-case scenarios. Its efficiency is significantly enhanced when integrated with Alpha-Beta pruning, an optimization technique that streamlines the search process by cutting down the number of nodes Minimax evaluates. However, despite these optimizations, Minimax requires extensive computational resources to explore large portions of the game tree. It assumes a perfectly rational opponent, planning for the worst-case scenario, and heavily relies on a well-designed state evaluation function. Implementing such a function is challenging, and its effectiveness diminishes beyond a certain depth ([1]).

Although tree search algorithms have achieved notable successes, such as IBM's DeepBlue defeating Garry Kasparov in chess, they falter in games with higher branching factors, like Go, which has exponentially more possible moves than chess [2]. This limitation underscored the need for innovative approaches in AI game strategies, highlighting the complexity difference between chess and Go.

The increased incorporation of Monte Carlo Tree Search (MCTS) into intelligent and informed agents has significantly influenced game strategic decision-making. MCTS diverges from traditional tree search algorithms by using random simulations to explore and evaluate the game tree. Instead of attempting to analyze the entire tree or assuming perfect play from the opponent, MCTS focuses on building a more manageable tree based on random sampling of moves. This approach allows it to navigate extensive game trees efficiently, which is particularly suited for games like Go with a high branching factor. The historic victory of DeepMind's AlphaGo over Go world champion Lee Sedol not only showcased MCTS's potential in mastering complex games but also heralded its application across various domains, demonstrating an ability to efficiently navigate extensive game trees with a fraction of the computational demand required by algorithms like Minimax [3]. Following this, the development of AlphaZero expanded upon AlphaGo's achievements, demonstrating the adaptability of MCTS in mastering multiple board games from a blank slate, thus underscoring the algorithm's profound strategic depth and computational efficiency ([4]).

In the context of this project, we applied a model-based variant of MCTS to Pacman, a game with a manageable action space, allowing for effective learning with fewer simulations. Beyond victories in board games, the application of MCTS to diverse areas highlights its adaptability and potential for strategic decision-making across various fields. This broadening of scope illustrates the algorithm's capacity to handle complex scenarios, ranging from digital games with intricate decision spaces to real-world problems that mimic the strategic depth of board games. The continual enhancements and innovative adaptations of MCTS ([5]) showcase its significance in artificial intelligence, marking it as a versatile tool for theoretical exploration and practical problem-solving in complex environments.

Building upon the foundation laid by MCTS, this project aims to explore its applicability and efficiency in a broader range of strategic games beyond Pacman. By extending the algorithm to games with varying complexities and branching factors, we seek to understand the versatility of MCTS and its potential to adapt to different strategic environments. This exploration is critical in evaluating the scalability of MCTS and its ability to maintain performance efficiency across different game domains. Through this investigation, we hope to contribute to the ongoing discussion on optimizing AI strategies in gaming, pushing the boundaries of current technology's achievable goals.

II. Technical Approach

Adopting the Monte Carlo Tree Search (MCTS) strategy in the Pacman game requires a nuanced approach, blending game simulations with strategic evaluations to effectively prioritize the exploration of promising states. Unlike traditional MCTS implementations, which may rely solely on simulations to their

conclusion without the need for an evaluation function, this project introduces the use of a specialized evaluation function to assess the quality of game states within the simulation process. This adjustment is crucial for navigating the inherent challenges presented by the Pacman game environment. Initially, there was an attempt to apply MCTS in its purest form, following the foundational principles outlined in seminal research. However, it quickly became apparent that such an approach was not entirely feasible for our development objectives. The computational burden of conducting exhaustive simulations across the expansive game environments of Pacman exceeded the capabilities of our available computing resources. To address this, we integrated an evaluation function within the MCTS framework, allowing for a more computationally efficient exploration of the game space by assessing states partway through simulations. This method enriches the strategic depth of the MCTS algorithm for Pacman, making it more adaptable and feasible for implementation on less powerful systems, thereby bridging the gap imposed by hardware limitations.

A. Approach Enumeration

The Monte Carlo Tree Search (MCTS) algorithm for a Pac-Man game can be described in a series of specific steps, aimed at determining the best move for Pac-Man at any given state of the game. The algorithm iteratively builds a search tree by exploring various possible future sequences of moves and evaluating their outcomes.

1) ***Initialization:*** Start by creating a root node for the search tree that represents the current state of the Pac-Man game.
2) ***Iteration Loop:*** Repeat the following steps several times, each time simulating a potential game outcome from the current state:
   (a) ***Selection:*** Starting at the root, nodes are selected based on the highest Upper Confidence bound applied to Trees (UCT) value, which optimizes the balance between exploring new paths and utilizing known rewarding strategies, until reaching a leaf node that suggests unexplored potential.
   (b) ***Expansion:***When a non-terminal leaf node with unexplored options is encountered, a new child node is generated for these possibilities, thus expanding the tree's exploration breadth.
   (c) ***Simulation:*** The simulation conducts a blend of random and strategically evaluated play-outs from the new node's state, continuing until the game concludes either with Pac-Man's capture or the collection of all pellets.
   (d) ***Backpropagation:*** Post-simulation, the outcome informs updates back through the tree, adjusting each node's visit and win metrics according to the results obtained, thereby refining future selection decisions.
3) ***Decision:*** After completing the iterations, choose the move associated with the child of the root node that has the highest win rate (wins divided by visits).

B. Computational Efficiency Enhancements

Key functionalities implemented to improve computational efficiency in the MCTS algorithm for Pacman include:

- ***Feature-Based State Evaluation:*** The evaluation function scores the game state by considering Pacman's position relative to ghosts, food, and walls, directing Pacman towards accessible food while avoiding ghosts and navigating around obstacles. This method streamlines the simulation by focusing on crucial game dynamics, reducing the need for exhaustive exploration.
- ***Dynamic Simulation Depth:*** The simulation adjusts its depth dynamically, factoring in the proximity of food, walls, and ghosts. Near food reduces depth for quick pellet consumption, while distant food and nearby ghosts extend the depth, guiding Pacman through more complex routes to distant objectives, optimizing computational effort for key strategic scenarios.
- ***Strategic Exploration:*** The exploration strategy within simulations incorporates an epsilon-greedy approach, allowing for a balance between exploring new actions and exploiting known advantageous moves. This method is refined further by the game's current state, such as the proximity of food and the presence of ghosts, ensuring a contextually relevant exploration that enhances the MCTS's decision-making capabilities.
- ***Adaptive Action Selection:*** The simulation phase includes a decision-making process that prioritizes actions based on an evaluation function, accounting for immediate threats and strategic opportunities. This ensures that each simulation step contributes meaningfully to discovering efficient game strategies, making the overall MCTS approach more effective.
- ***Efficient State Transition:*** By selectively removing non-beneficial actions (like 'Stop' when better options exist) and adjusting exploration based on game state characteristics, the algorithm ensures that computational efforts are directed toward exploring game states with the highest potential impact.

These enhancements, deeply integrated into the MCTS framework, are pivotal in navigating the computational and strategic complexities of the Pacman game, ensuring both efficient computation and strategic depth in gameplay.

C. Pseudo-Algorithm

To illustrate the enhancement in the Monte Carlo Tree Search (MCTS) optimization for the Pacman game, we focus on the simulation phase—central to understanding the efficiency gains. The other phases are briefly introduced to provide a complete picture of the MCTS process without delving into extensive detail.

1) ***Initialization and Tree Exploration****:* The MCTS begins with initializing the tree's root node to represent the game's current state and proceeds with a loop that explores potential outcomes through selection, expansion, and ultimately, simulation.

Algorithm 1 High-Level MCTS Process

function INITIALIZE($GameState$)
Create $rootNode$ with $GameState$
return $rootNode$

function EXPLORE_TREE($node$)
Select promising $node$ based on UCT
Expand $node$ if not fully explored
Simulate game outcome from $node$
Backpropagate results
return Best action from $rootNode$

*2) **Simulation: The Core of Strategic Exploration**:* The simulation phase unfolds through the lens of the chosen node, blending strategy and randomness to predict outcomes and guide the tree's growth. This phase focuses on evaluating terminal states and enforcing depth limits to maintain computational efficiency, ensuring that each simulation contributes meaningfully to the decision-making process.

Algorithm 2 Simulation and Evaluation

Require: $node$, $depthLimit$
Ensure: $simulationResult$
function SIMULATE($node$, $depthLimit$)
if $node.state$ is Terminal $depthLimit$ is reached **then**
$result \leftarrow$ Evaluate $node.state$
else
Select a random action a from $node.state$
$newState \leftarrow$ Apply a to $node.state$
$result \leftarrow$ SIMULATE($newState$, $depthLimit - 1$)
end ifreturn $result$

*3) **Depth Estimation for Simulation**:* Depth estimation plays a crucial role in controlling the simulation's granularity, ensuring that simulations are both meaningful and computationally feasible.

Algorithm 3 Depth Estimation

Require: $currentNode$, $vicinityData$
Ensure: $depthLimit$
function ESTIMATE_DEPTH($currentNode$, $vicinityData$)
if $vicinityData.closestFood$ is near **then**
$depthLimit \leftarrow$ Low $vicinityData.foodInVicinity$ AND $currentNode.state$ has strategic advantage
$depthLimit \leftarrow$ Medium
else
$depthLimit \leftarrow$ High
end if
return $depthLimit$

This algorithm dynamically adjusts the depth of simulation based on the proximity of food and the strategic positioning within the game, optimizing the depth of search for balanced exploration.

*4) **Evaluation Function**:* The evaluation function intelligently scores game states to steer the simulation toward favorable scenarios. It calculates a score by weighing key game factors, including proximity to food and distance from threats, crucial for Pacman's strategic advancement. This function not only prioritizes positions with nearby food but also adjusts scores based on the density of food pellets in the vicinity, rewarding Pacman for navigating towards areas with optimal food availability while maintaining a safe distance from ghosts.

Algorithm 4 Evaluation Function

Require: $gameState$
Ensure: $score$
function EVALUATE($gameState$)
$score \leftarrow 0$
$score \leftarrow score+$ Value of remaining pellets
$score \leftarrow score-$ Distance to nearest ghost
$score \leftarrow score+$ Distance to nearest food **return** $score$

III. Results

A. Analysis

1) Grid Size and Complexity Distribution Analysis:: The sample environment set of grids for Pacman comprises wall-food-ghost networks with varied grid sizes, featuring a notable prevalence of around 400 nodes. Additionally, the distribution of network complexity exhibits a distinct organization, with a peak density at around 0.4, indicating prevalent levels of interconnection. These findings provide valuable insights into network characteristics and suggest potential areas for further study and analysis.

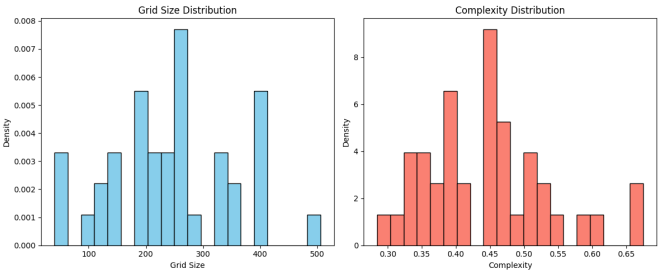

Fig. 1: Visualisation of the Sample Set Distribution

Utilizing the supplied grid layouts, we expanded the sample set to 35, following a normal distribution that incorporates grid size and complexity as attributes to dictate the distribution, which is visualized in Fig 1. This approach enhances the robustness of our analysis, offering a comprehensive overview of the environmental dynamics encountered in Pacman simulations.

2) T-Tests Results: The statistical analysis presented in Table I underscores the comparative performance of the MonteCarloTreeSearchAgent (MCTSA) against a spectrum of game-playing agents. Notably, the MCTSA demonstrates varied efficacy across different agents. For instance, against the ReflexAgent (RA), MCTSA exhibits a statistically significant lower performance, as indicated by a negative T-statistic (-3.31) and a P-value of 0.0020. Conversely, when compared

TABLE I: Performance Comparison of MCTSA with Various Agents

Comparison	T-stat	P-value
MCTSA vs. RA	-3.31	0.0020
MCTSA vs. MA	2.88	0.0052
MCTSA vs. ABA	1.14	0.2568
MCTSA vs. EA	2.84	0.0058
MCTSA vs. MCTRA	4.62	0.0000

with the MinimaxAgent (MA) and the ExpectimaxAgent (EA), MCTSA shows a significantly higher performance, evidenced by positive T-statistics (2.88 and 2.84, respectively) and low P-values (0.0052 and 0.0058, respectively), highlighting its competitive edge in these matchups. The comparison with the AlphaBetaAgent (ABA), however, reveals no statistically significant difference in performance, with a P-value of 0.2568 suggesting parity between the two. Furthermore, the MCTSA significantly outperforms the MonteCarloTreeRandomSearchAgent (MCTRA), as the highest positive T-statistic (4.62) coupled with a P-value of 0.0000 strongly attests to the superiority of MCTSA's strategic approach. This analysis, grounded in statistical rigor, illuminates the nuanced performance landscape of MCTSA in the context of AI-driven game strategies.

B. Discussions

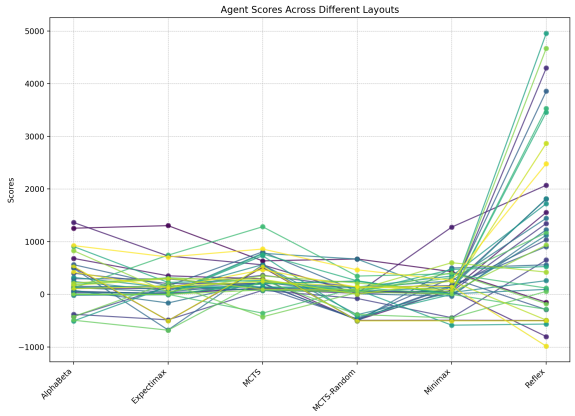

Fig. 2: Agent Score Comparison

Our project has advanced an algorithm to adeptly navigate the myriad of strategic choices within games by incorporating simplified state representation, dynamic simulation enhancements, and refined decision-making processes between exploration and exploitation. The crux of the algorithm's success hinges on its streamlined decision-making capabilities, facilitated by predictive analyses of future game states and strategic evaluations. This optimization significantly bolsters the algorithm's efficacy in game scenarios, underscoring the role of computational intelligence in enhancing gaming strategies.

Through comparative analysis of various game-playing agents, we observed that their performance largely adhered to a normal distribution, with notable exceptions where some

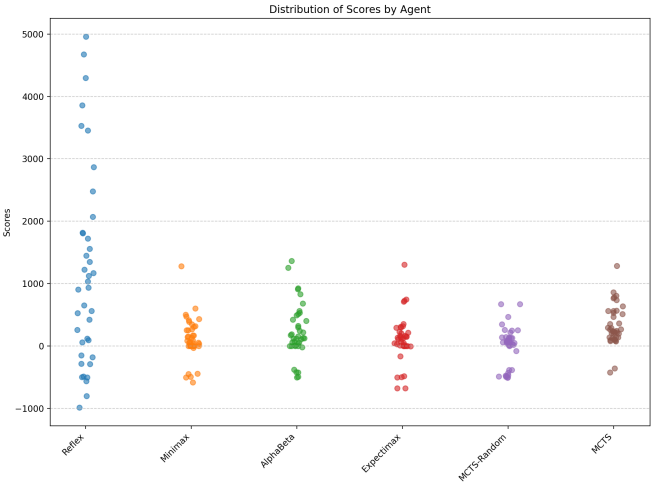

Fig. 3: Scatter plot for score

agents surpassed the performance of the traditional Monte Carlo Tree Search (MCTS) algorithm. These agents employed a diverse array of strategies, ranging from reflex actions and Minimax strategy to Alpha-Beta pruning and distinct iterations of MCTS incorporating random search tactics. Our findings illuminate the strategy effectiveness's dependency on the game's specific environment, reinforcing the value in exploring alternative strategies that may offer superior outcomes in certain contexts. Further investigation into these strategic variances promises to aid in selecting the optimal agent strategy for diverse gaming environments. The performance comparison is visually represented in Figures [scatterplot-placeholder] and [lineplot-placeholder], which illustrate the variance and trends in agent strategies across different game scenarios.

IV. Conclusions

The enhanced Monte-Carlo Tree Search (MCTS) algorithm emerged as a formidable contender in our comparative analysis, demonstrating notable success across varied game layouts and achieving positive outcomes more consistently than its counterparts. While the MCTS occasionally trailed in speed compared to agents like Minimax and Expectimax, its strategic depth and the calculated decision-making process—especially following a random move—proved critical in securing higher scores. Notwithstanding, the MCTS was outperformed by both the Alpha Beta and Reflex agents in certain scenarios, suggesting areas for further refinement, such as deepening the search or increasing simulation counts for improved move evaluations.

Contrary to the Reflex Agent, which exhibited significant performance variability, the MCTS showcased consistent results with minimal instances of negative scores, indicating a reliable baseline for strategic gameplay. Moving forward, we aim to enhance the MCTS by deepening the evaluation depth and augmenting the simulation frequency, which is anticipated to sharpen its strategic acumen further.

REFERENCES

[1] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[2] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte carlo tree search and extensions," *Communications of the ACM*, vol. 55, no. 3, pp. 106–113, 2012.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[5] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.