# Monte Carlo Tree Search On Pacman

TEAM 14


Venkatarao Rebba
*Arizona State University*
Tempe, AZ, USA
vrebba@asu.edu

Sri Surya Bandaru
*Arizona State University*
Tempe, AZ, USA
sbandar9@asu.edu

Nithin Jayakar Padala
*Arizona State University*
Tempe, AZ, USA
npadala@asu.edu

Sai Krishna Vamsi Mahadasa
*Arizona State University*
Tempe, AZ, USA
smahadas@asu.edu



***Abstract*—This study determines a winning policy for the Pacman world using the Monte Carlo Tree Search algorithm and tracks its performance using a T-test on complex scenarios of the Pacman world. Monte Carlo Tree Search algorithm considers both best likely and unlikely paths and performs simulations, making it a better algorithm than the Expectimax algorithm for the Pacman world. It simulates without evaluating all the possible states; consequently, it is much faster and more efficient.**


## I. Introduction

The adaption and response of an agent reflect its cognitive intelligence. Consequently, it is challenging to develop such an intelligent system to perform operations. An AI game system generally requires adeptness and responsiveness; developing one would make the research one step closer to finding an efficient AI system. If the sum of utility values is a constant, then such a game is classified as a zero-sum game. For example, there can be one winner in the chess game, and eventually, another player will be the loser. The Expectimax algorithm [1] has an advantage in zero-sum games as the two players provided alternate levels of the Expectimax tree. Player 1 tries to maximize the utility score of a tree-level; however, its next-level utility score is minimized by player 2. One significant problem in the Expectimax algorithm [1] is that it evaluates all possible states in the game tree, making it vulnerable in complex games like Go as its search space is exponential. Even Alpha-Beta pruning [2] can reduce the number of nodes needed to be expanded, but the search space remains exponential.

Turn-based games like Pacman demand a sequential choice of actions, and it's challenging to form a heuristic function; moreover, it is a multiagent game with multiple ghosts. Accordingly, the Monte Carlo Tree Search algorithm addresses all the above deficiencies and operates on four steps:

- Selection
- Expansion
- Simulation
- Back-propagation

The Monte Carlo Tree Search algorithm is intended to develop a plan while maintaining the balance of exploration and exploitation with or without a heuristic function. Thus the algorithm makes Pacman agent try to maximize the utility score, and the Ghost agents try to take down Pacman by minimizing the utility score. Consequently, In this research, the Monte Carlo Tree Search is practiced to find the optimal strategy for the agent to win in the Pacman world.

The performance of the Monte Carlo Tree Search algorithm [4]on The Monte Carlo Tree Search algorithm is intended to develop a plan while maintaining the balance of exploration and exploitation with or without a heuristic function. Thus the algorithm makes Pacman agent try to maximize the utility score, and the Ghost agents try to take down Pacman by minimizing the utility score. Consequently, In this research, the Monte Carlo Tree Search is practiced to find the optimal strategy for the agent to win in the Pacman world. world is briefly described in the following sections of the technical approach, results, conclusion.

## II. Technical Approach

Monte Carlo Tree Search algorithm forms an asymmetric statistics tree that maps onto the entire game of packman world. The statistics tree with values on each node calculated by simulation guides the agent to look at the most interesting nodes in the game tree, concurrently giving a fair chance to other nodes. Monte Carlo Tree Search algorithm forms such a statistics tree in four steps.

Selection: Initially, Monte Carlo Tree Search will begin making its statistics tree at the Initial state. The algorithm selects a node to expand the statistics tree further. The criteria for selection is to prioritize unexplored nodes over explored. If all nodes are explored at least once, then the Upper confidence bound algorithm is used for the selection process. At a level $l_n$ the node with maximum action value is selected. The Action value $A_n$ at a node is

$$A_n = \text{argmax}_a\left(Q_n(a) + c\sqrt{\frac{\log n}{N_n(a)}}\right)$$

Fig. 1. Upper Confidence Bound Formula

Where $Q_n(a)$ is the determined value of action 'a' at a node. n is the number of times the parent node is explored. $N_n$ (a) is the number of times the action 'a' is chosen at a child node. c is a confidence value.

The rate of exploitation is controlled by Qn(a) in the Upper confidence bound formula, which works on the greedy principle the current best action is chosen based on the action value. The rate of exploration is controlled by hyper-parameter c. The upper confidence bound algorithm uses Qn(a) and c terms to balance exploitation and exploration. As time advances, the value of exploration decreases, whereas the value of exploitation increases, and selection is based on it.

Expansion: In this step, the selected node is expanded if it is unexplored and a leaf node. All the possible states from that leaf node were appended to the tree. All possible states were obtained and appended to the game tree from a state for each available legal action in the Pacman domain. All the possible scenarios and moves of the Pacman game were generated in this phase for exploration and exploitation.

Simulation: From the selected node, random actions were chosen to reach the terminal state of the Pacman game. During this rollout, few features were used to obtain the optimal simulation path.

To attain optimal Pacman's behavior, Q-learning is used to enhance the quality of rollout policy. To offer practical simulation, each state value is weighted with random weights, and with the help of Q- learning, weights are adjusted and help the agent to choose the best possible action at a state. On top of it, environmental knowledge such as the proximity of ghosts and food pallets was also introduced. To evaluate the terminal state of the simulation is win or loss, a heuristic function is developed. The action value of the parent node is improved or penalized according to the result of the simulation. Accordingly, ghost agents have developed with randomized and directional behavior; the simulation has an equal proportion of the two ghosts to obtain the optimal path. Finally, After attaining optimal weights and hyper-parameters, the action with maximum value is elected.

Backpropagation: The value for the exploration and the action value increases after the simulation results depending on win or loss. The leaf node action value gets updated, and the simulation result is propagated back to the previous nodes. Till the initial stare is reached, all these operations were looped.

The Monte Carlo Tree Search algorithm traverses the most likely move for Pacman and ghosts. The backpropagation of values is concurrently performed for both the ghosts and Pacman, with opposite values. Thus the computation decrease as there is no need to explore all possible ghost moves. The hyperparameters such as steps, simulation Depth, early stop were tuned to improve the performance of the Monte Carlo Tree Search algorithm. The step parameter limits the number of steps that Pacman can take while performing the Monte Carlo Tree Search, and the simulation Depth restricts the number of levels of the statistics tree formed by the Monte Carlo Tree Search algorithm. The values till that level were backpropagated till the root node, which helps the heretic function to avoid a few computations. The early stop parameter stops the algorithm if the action values don't differ much over time; the Monte Carlo Tree Search algorithm stops at that moment, saving the computational power and improving the algorithm's performance. Apart from these parameters, a few more parameters such as panic percents and simulation weights were introduced to check the previous measure do not result in low performance of the Monte Carlo Tree Search algorithm. Conclusively, the Monte Carlo Tree Search algorithm determines the action for the Pacman for the state with maximum action value, which is determined from average win rates over simulations.

T-test distinguishes the distributions based on the means by generating two T and P scores that quantify the difference of distributions. Therefore the scores distributions of Expectimax, Alpha-Beta pruning, and Monte Carlo Tree Search were compared using T-test.

## III. Results

We used various parameters like win rate, average number of nodes expanded, average time taken per move, average score, average tree depth, average time and average score, in order to evaluate the obtained output. A different set of layouts have been used for twenty games and obtained the above parameter metrics.

Performance is classified mainly based on:

***Explore algorithm(exploreAlg):*** We need to select the node using epsilon greedy or upper confidence bound technique to run the Pacman. With reference to the Table-I, we can find that UCB algorithm performs well for 3 times the average score and 1.8 times the win rate in case of reuse tree. While in case of no reuse tree, it performed well for a 2.57 times the average score and 1.4 times the win rate.

TABLE I
COMPARISON OF PERFORMANCE BASED ON EXPLORE ALGORITHMS AND TREE REUSE PARAMETERS

| | Epsilon Greedy | UCB | |
|---|---|---|---|
| | | Reuse | No Reuse |
| Epsilon / C Value | 0.6 | 150 | 150 |
| Average Score | 503.2 | 1454 | 1295 |
| Win rate | 50% | 90% | 70% |
| Average time | 32 | 60.24 | 90 |
| Average tree depth | 12 | 8 | 10 |

***Tree Reuse(reuse):*** This is used for identifying whether the previous search tree is reused in the next search or not. From the data obtained from Table-I, in this case we have an improved win rate and average score with a lesser tree depth.

***Simulation Depth(simDepth):*** This is set to a positive integer which represents the number for how many turns an agent takes in a simulation while the heuristic still does not generate

a result. In general, the success rate and the simDepth are directly proportional, as the exploration goes deeper. Whereas in our case, unlike stated above the agent performed well for a certain depth and its performance decremented with the increase in the depth from the previous depth. But again, there is an increment in performance with the further increase in depth. This is mainly due to, with the increase in the total number of games, our agent starts to overcome its boundaries and tries to terminate the game, so that it can still survive. This all can be inferred from the data obtained from Table-II.

TABLE II
ALGORITHM PERFORMANCE FOR DIFFERENT SIMULATION DEPTH VALUES

| | Depth = 3 | Depth = 5 | Depth = 10 |
|---|---|---|---|
| Average Score | 1145.85 | 1020.2 | 1255.7 |
| Win rate | 100% | 90% | 95% |
| Average time | 10.40 | 13.46 | 40 |
| Average tree depth | 9 | 9 | 10 |

***Optimism:*** In a simulation, this decides the percentage of moves that can be random for the ghosts. This is mainly useful in developing the score in case of random ghosts. Since the Pacman won't be expecting a confrontational move all the time from a ghost.

***Score:*** A score is nothing but a performance report of the Pacman in opposition to the ghosts. In Table-III, we have noted different set of values for the performance of an agent under different search algorithms in both small and medium layouts with random and directional ghosts. In case of random ghosts, our MCTS matches with the Expectimax and while it outperforms the Expectimax with the directional ghosts.

TABLE III
COMPARISON OF DIFFERENT AGENTS

| Maze/Agent | Expectimax | Alpha-Beta | MCTS |
|---|---|---|---|
| Small/Rdm | 1368.4 ± 312.7 | 1254.6 ± 139.89 | 1292.8 ± 410.06 |
| Small/Dct | 403.5 ± 864.7 | 490.7 ± 892.76 | 1230.65 ± 530.76 |
| Medium/Rdm | 1760.3 ± 246.7 | 1525.46 ± 547.59 | 1702.18 ± 165.06 |
| Medium/Dct | 987.26 ± 904.89 | 843.45 ± 814.20 | 1475.8 ± 505.06 |

We have plotted the scores of the agent in opposition to directional ghosts on a box-whisker plot displayed in the Figure-2. While in the Figure-3, we have plotted on behalf of scores of the agent against directional ghosts. The difference between the 2 plots is the layout size or class used. Here, we can compare the range and median values of the three search algorithms. With inference to the data provided, MCTS has a comprehensive superiority over others interms of performance. So far as we observed, the scores of MCTS are distinctively higher when compared with the other two, in both the layouts.

***Win Percentage:*** Sometimes in spite of getting a top score, some of the goals of the game are missed out. In this case,

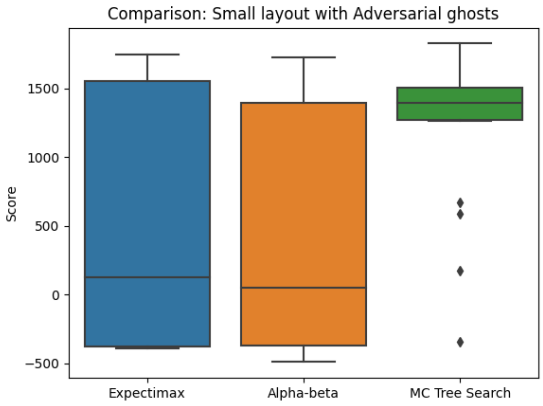


Fig. 2. Distribution of Scores for a size of small layout

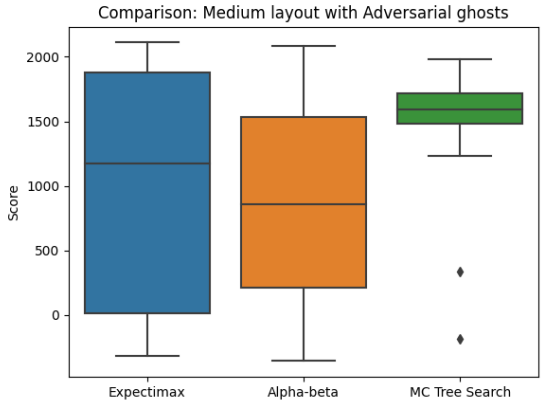


Fig. 3. Distribution of Scores for a size of medium layout

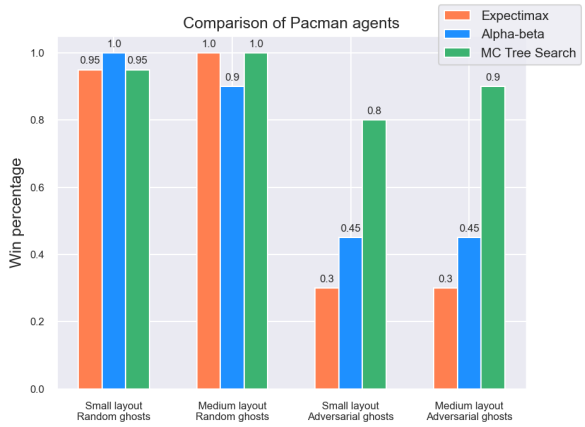


Fig. 4. Win rate comparison

the performance of the agent is a bit varied. With inference to the data generated from the Figure-4, we have compared the performance of the agent W.R.T to different search algorithms as all of them have a similar win rate in case of random ghosts. While the win rate is higher for MCTS when compared to others in case of directional ghosts.

TABLE IV
EXPANDED NODES AND COMPUTATION TIME

| Maze/Agent | Expectimax | Alpha-Beta | MCTS |
|---|---|---|---|
| Small/Rdm | 5010 / 50.5 | 1590 / 21.5 | 222 / 31.50 |
| Small/Dct | 4150 /23.67 | 1230 / 14.15 | 245 / 42.05 |
| Medium/Rdm | 2415 / 66.16 | 2299 / 21.09 | 210 / 45.02 |
| Medium/Dct | 3402 / 66.89 | 1173 / 27.67 | 212 / 32.98 |

Data Format: Nodes expanded / Time

***Expansion of Nodes:*** Some of the parameters we have used so far like randSim, reuse, simRelevance, tillBored and earlyStop are a bit responsible in acting against the expansion of nodes. With the reference to the data inferred from Table-IV, our MCTS is comprised of the top win percentage and score W.R.T other two search algorithms while taking into consideration of a little expansion of nodes.

***Time of computation:*** With increase in the size of layout and also increasing the ghost number, the average time taken by MCTS keep on increasing respectively.

## IV. TESTING

We have used T-test for comparing scores of MCTS against Expectimax and Alpha-Beta pruning algorithms. Here, we have used the means of two groups of samples for our testing. In this way, we can determine how the average score of MCTS significantly varies from the other two. Typically, it facilitates to draw the comparison between the two distributions. We performed two separate tests for Expectimax and Alpha-Beta pruning and combined the result into Table-V. Additionally, various environments were considered to derive notable conclusions.

TABLE V
P-VALUES FROM T-TEST

| Maze/Agent | Expectimax | Alpha-Beta |
|---|---|---|
| Small/Rdm | 0.120 | 0.132 |
| Small/Dct | 0.0035 | 0.0035 |
| Medium/Rdm | 0.02 | 0.09 |
| Medium/Dct | 0.046 | 0.0065 |

From Table-V, we can conclude that for smallClassic and MediumClass layouts with RandomGhost, the p-value indicates strong evidence for null-hypothesis. Whereas, p-value for the environment with Directional Agent is less than 0.05 which indicates strong evidence against null-hypothesis. Therefore, we can reject this null hypothesis and opt for an alternative.

## V. CONCLUSION

Monte Carlo Tree Search algorithm has become the leading approach for a lot of games. It also has a wider range of applications for different set of domains. In this paper, we have demonstrated how MCTS is effective in case of the Pacman domain. Exploration of different algorithms and consideration of Upper Confidence Bound(UCB) for the sake of selection has been done. Using the set of defined parameters improved the efficiency of our algorithm.

We have compared the win percentages of MCTS, Expectimax, Alpha-Beta Pruning algorithms W.R.T to small and medium class layouts and concluded that MCTS has the upper hand interms of performance over the other two. And also comparison of node expansion and time of computation is provided as tabular data.

Performed T-testing of Our MCTS against Expectimax and Alpha-Beta pruning algorithms and computed the data into Table-V. Clearly displayed the test results W.R.T. Expectimax and Alpha-Beta pruning in terms of small and medium class layouts with both random and directional ghosts.

Ultimately, for this Pacman domain, with inference to the overall data obtained and the performance of our MCTS implementaion, we can clearly come to a conclusion that MCTS is comprehensively superior to both the Expectimax and Alpha-Beta Pruning search algorithms.

## REFERENCES


[1] Gunawan and I. Kurniawan, " Expectimax Algorithm to solve Multiplayer Stochastic Game" in Proceedings of the 5th International Conference on Information Communication Technology and Systems, NY, USA, Dec. 2009.

[2] M. Sravya, B Tejashwini, V Sanjana, " Expectimax Algorithm to solve Multiplayer Stochastic Game" in Proceedings of the International Journal of Innovations in Engineering Research and Technology, Jan. 2020.

[3] B. Abramson and R. E. Korf, "A model of two-player evaluation functions." in AAAI, 1987, pp. 90–94.

[4] Kien Quang Nguyen and Ruck Thawonmas, "Monte Carlo Tree Search for Collaboration Control of Ghosts" in Proceedings of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES,, March. 2013.

[5] Yngvi Bj¨ornsson and Hilmar Finnsson, "A Simulation-Based Genral Game Player" in Proceedings of the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 1, NO. 1, MARCH 2009