Designing Monte Carlo Tree Search Based Heuristic AI Agents

A dissertation submitted in
partial fulfilment for the degree of

Master of Technology

in

Computer Science

by

Himanshu Yadav
Roll no. - [**CS2214**]

under the supervision of

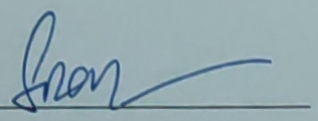
Dr. Swagatam Das
Professor
Electronics and Communication Sciences Unit

Indian Statistical Institute, Kolkata
June, 2024

CERTIFICATE

This is to certify that the dissertation entitled "**Designing Monte Carlo Tree Search Based Heuristic AI Agents**" submitted by **Himanshu Yadav** to the Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of Master of Technology in Computer Science is an authentic and genuine record of the research work conducted by the candidate under my supervision and guidance. I affirm that the dissertation has met all the necessary requirements in accordance with the regulations of this institute.

June,2024



Dr. Swagatam Das
Professor
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108

Declaration

I, **Himanshu Yadav**, with Roll No. **CS2214** hereby declare that the material presented in the dissertation titled **Designing Monte Carlo Tree Search Based Heuristic AI Agents** represents original work carried out by me for the degree of **Master of Technology, Computer Science** at **Indian Statistical Institute, Kolkata**.

Furthermore, I affirm that no sections of this report have been sourced or copied from external references without proper attribution. I am aware that any instances of plagiarism or the utilization of unacknowledged materials from third parties will be treated with utmost seriousness and consequences.

Himanshu Yadav
Roll no.- CS2214

Acknowledgement

I would like to express my utmost gratitude to my esteemed advisor, **Dr. Swagatam Das**, from the Electronics and Communication Sciences Unit at the Indian Statistical Institute, Kolkata. His unwavering guidance, continuous support, and encouragement have been invaluable to me throughout this research journey. Under his mentorship, I have learned the art of conducting thorough and impactful research, and her insightful ideas have constantly motivated and inspired me. I am thankful that he helped me explore such an interesting topic during this time.

I extend my deepest thanks to all the research scholars at the Indian Statistical Institute for their invaluable suggestions and engaging discussions, which have significantly enriched the depth and quality of my research work.

I would also like to express my sincere appreciation to all my friends for their unwavering assistance and support. I am grateful to everyone who has contributed to my growth and success, even if I have inadvertently missed mentioning them in the above list.

List of Figures

Notation and Abbreviations

AI	Artificial intelligence
MCTS	Monte-Carlo tree search
UCB	Uniform confidence bound
i.n.i.d.	independent and non-identically distributed

Abstract

Recent advancements in Monte Carlo Tree Search (MCTS) methods have garnered significant attention in the AI community. Inspired by these developments, we implemented tree-based search algorithms to create game-playing agents for Tic-Tac-Toe and Ludo. Our analysis primarily focuses on the Minmax and MCTS algorithms. Minmax, a well-known strategy in game theory, effectively handles games with smaller state spaces like Tic-Tac-Toe, ensuring optimal play. However, for more complex games like Ludo, MCTS proved to be more efficient by using random sampling and tree-based search to make decisions.

We further explored Bayesian MCTS for Tic-Tac-Toe, which incorporates probabilistic models to enhance decision-making. This approach allowed our agent to handle uncertainty better, leading to improved performance. These implementations demonstrate the potential of combining traditional AI techniques with advanced search algorithms to create robust game-playing agents.

Contents

Chapter 1

Introduction

The gaming industry is expanding at an exponential rate most of which is an outcome of computational advancements made during the last decade. A lot of focus has been observed in using artificial intelligence to simulate different kinds of experiences for the human players.
India's online gaming market has witnessed an unparalleled surge in growth, drawing substantial financial contributions from both local and foreign investors. This rise has been driven by the increasing accessibility of smartphones and technological advancements, which have brought in a large number of industry participants, particularly those engaged in real-money gaming, such as online poker and online rummy, where the stakes and financial hazards are extremely real.
Games like rummy and poker, are recognized as games of skill around the world, but the Indian online gambling industry is beset by a complicated regulatory framework. There are numerous state laws and rules that either forbid or restrict gambling, while games of skill are usually exempt. In the Indian legal framework, a game is classified as a game of skill if it is predominantly determined by skill, whereas a game reliant largely on luck is considered a game of chance.

"Everyone gets lucky once in a while, but no one is consistently lucky." -Doyle Frank Brunson, Poker Hall of Fame Inductee

This trend is also driving the race to build optimal expert automated game agents and can be seen as one of the real world use cases.
It is noteworthy that technical advancements in gaming have been observed for all classes of games. The classification can be done in multiple ways such as, Outdoor

vs Indoor, Chance vs Strategic, Cooperative vs Non-Cooperative and so on. Just a necessary mention that outdoor games like football, and cricket can be played from the ease of the player's room nowadays.

One of the development niches where programmers and academicians spend a lot of time learning what artificial intelligence (AI) can and cannot do is the online game business.

The next actual question that arises is, how to begin with developing competitive AI game playing agents? This question leads us to a class of methods that are based on heuristic tree search algorithms.

MCTS was initially presented as a technique for creating computer players in Go by Kocsis and Szepesvári (2006) and Coulom (2006) [1]. Gelly et al. (2012) declared it to be a significant advancement right away since it made it possible to go from 14 kyu, which is regarded as an advanced level but not yet professional, to 5 dan, which is an average amateur level. Prior to MCTS, a variety of minimax alpha-beta pruning algorithm modifications, like MTD(f), were used by bots for combinatorial games and manually created heuristics (Plaat 2014)[2]. We focus on tree based heuristic search algorithms to build optimal agents mainly inspired by the fact that Monte-Carlo Tree Search (MCTS) methods are drawing great interest after yielding breakthrough results in computer Go. We implement MinMax, MCTS and Bayesian MCTS for tic-tac-toe and Ludo as our test bed. Tic-tac-toe being a simple 2 player game with full information game and Ludo being multiplayer (four) complex, non-deterministic game with partial information collectively help us analyze the agents in different competitive settings.

Chapter 2

Tic-Tac-Toe and Ludo

2.1 Tic-tac-toe Game

Traditionally, tic-tac-toe is a 3×3 tile board game with no variations as such. We will discuss the game environment on a high level in this section. We will discuss the components of the game below.

2.1.1 Tile grid

It is represented by a 2d matrix of dimension 3×3.

2.1.2 players

Players are represented by 'X' and 'O' where 'X' is the symbol reserved for the player who takes the first turn and vice versa. In each turn player chooses one of the vacant tiles to mark by its symbol.

2.1.3 Game termination

If any row/column/diagonal of the board gets occupied by the same symbol after the immediate last turn then the game terminates immediately and the player with that symbol is the winner. Otherwise, after all the tiles get filled with symbols and yet there is no winner then it is a draw.

2.1.4 Symmetry

tic-tac-toe is invariant to all types of symmetries. To capitalize on this, we depict every board that can be changed into another board with just one board representation. It facilitates a game tree's node count reduction. The following are the different transformations:

- **Identity()**
  The input remains unaltered.
- **Rotate90(1)**
  The input is rotated 90 degrees clockwise using this transformation.
- **Rotate90(2)**
  The input is rotated 180 degrees (90 degrees twice) with this transformation.
- **Rotate90(3)**
  The input is rotated 270 degrees clockwise (90 degrees each time) with this transformation.
- **FlipVertical()**
  This transformation turns the input vertically upside down.
- **FlipHorizontal()**
  This transformation performs lateral inversion on the input.
- **Rotate90AndFlipVertical()**
  This transformation first rotates the input by 90 degrees clockwise and then turns it upside down.
- **Rotate90AndFlipHorizontal()**
  This transformation first rotates the input by 90 degrees clockwise and then performs lateral inversion.

Reward

- **RESULT_X_WINS = 1**
  This constant represents the result where player X wins the game.
- **RESULT_O_WINS = -1**
  This constant represents the result where player O wins the game.

- **RESULT_DRAW = 0**
  This constant represents the result where the game ends in a draw.

Board cache

It stores the nodes of a Game tree in the dictionary for faster access.

2.2 Ludo Game

Ludo is a strategy board game for two to four players, in which the players race their four tokens from start to finish according to the rolls of a single die. Derived from the ancient Indian game Pachisi, Ludo is simple and fun, making it a favorite among family and friends worldwide. Here are the basics of how to play Ludo:

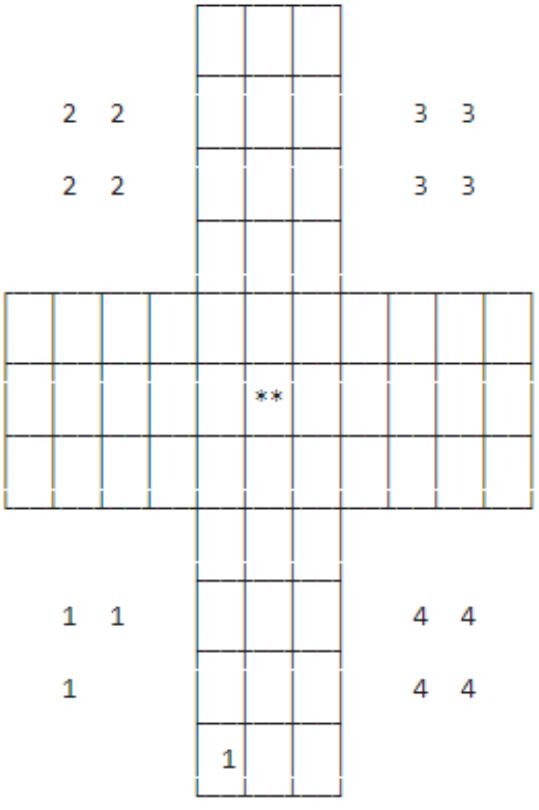


Figure 2.1: Ludo board depiction where one piece of player 1 has just got out

2.2.1 Components

- **Board:** The Ludo board is typically a square with a cross inscribed on it, divided into four colored sections (red, blue, yellow, green), each corresponding to a player.
- **Tokens:** Each player has four tokens of the same color.
- **Die:** A single six-sided die is used to determine movement.

2.2.2 Objective

The goal is to move all four of your tokens from the starting area to your home area (the center of the board).

2.2.3 Setup

Each player chooses a color and places their four tokens in the corresponding starting area (corner of the board). Decide who goes first (often by rolling the die).

2.2.4 Gameplay

- **Starting a Token:** To move a token from the starting area onto the track, a player must roll a 6. If a 6 is rolled, the player gets another turn.
- **Moving Tokens:** Players take turns rolling the die and moving one of their tokens forward by the number of spaces shown on the die. If a player rolls a 6, they get another roll after moving a token.
- **Capturing:** If a token lands on a space occupied by an opponent's token, the opponent's token is sent back to its starting area.
- **Safe Spaces: (Not implemented in our version)** There are certain spaces on the board marked as safe spaces (usually the colored squares), where tokens cannot be captured.
- **Home Column:** Each player has a home column leading to the home area. Tokens must move up this column by exact count to reach the home area.

2.2.5 Winning

The first player to move all four of their tokens to the home area wins the game.

2.2.6 Variations

Ludo has many variations, with some including different rules for rolling the die, moving tokens, and capturing. It's a versatile game that can be adjusted to suit different preferences and ages.

Ludo is a game of both luck and strategy, making it enjoyable for players of all ages.

2.3 Game theoretic properties of Ludo and Tictac toe

2.3.1 Tic-tac-toe

- **Finite game:** It has finite number of possible movements and outcomes thus tic tac toe is a finite game. To be more precise, there are just 255,168 possible games (including duplicate variations).
- **Perfect information:** In tic tac toe, each player always knows exactly how the game is going. This indicates that the entire board and all past moves are visible to both players, and there are neither chance nor hidden components in the game.
- **Zero-sum game:** The overall gains and losses incurred by the players in the tic-tac-toe game add up to zero. Every gain made by one player is compensated by a loss made by the other. In the event of a tie, neither player wins nor loses anything. "If one player wins, the other player loses."
- **Complete analysis:** The best course of action for each player can be determined by evaluating every potential game condition. Because of this, it is feasible to play Tic Tac Toe precisely, guaranteeing the person who makes the best movements a win or a tie.
- **Optimal strategy:** The optimal strategy for Tic-tac-toe is well-defined. If both players play optimally, the game will always end in a draw. This optimal strategy

involves selecting the best move at each turn to either win the game if possible or force a draw if winning is not possible.

- **Branching factor:** The branching factor in Tic-tac-toe refers to the average number of possible moves available to a player at each decision point. In Tic-tac-toe, the branching factor decreases as the game progresses since fewer empty spaces are available.

- **Symmetry:** There are various symmetries present in Tic-tac-toe due to the rotational and reflectional symmetries of the game board. Exploiting these symmetries can reduce the computational complexity of analyzing the game.

2.3.2 Ludo

- **Finite Game:** Ludo is a finite game as it has a limited number of possible moves and outcomes. Each game concludes once a player successfully moves all their tokens to the home area.

- **Stochastic Game:** Ludo is a stochastic game due to the element of chance introduced by the roll of a die. The randomness of the dice roll affects the players' decisions and the game's progression.

- **Strategic Moves:** Players must decide which token to move based on the outcome of the dice roll. Strategic considerations include:

  - **Token Safety:** Moving tokens to safe spots to avoid capture.
  - **Aggression:** Capturing opponents' tokens to hinder their progress.
  - **Advancement:** Prioritizing the movement of tokens that are closer to the home column.

- **Nash Equilibrium:** A Nash equilibrium in Ludo occurs when players adopt strategies where no player can benefit by changing their strategy while the others keep theirs unchanged. Given the stochastic nature, identifying pure strategy Nash equilibria is complex and often involves mixed strategies where players randomize their moves based on probabilities.

- **Payoff Matrix:** The payoff in Ludo is determined by the game's objective of getting all tokens to the home area. Players' payoffs are usually defined as:

  - Winning the game.
  - Finishing second, third, or last, depending on the order of reaching the home area.

- **Cooperative vs. Non-Cooperative Play:** Ludo is generally a non-cooperative game where each player competes individually. However, players might form temporary alliances to hinder a leading player, making the game partially cooperative.

- **Strategies and Heuristics:**
  - **Aggressive Strategy:** Focus on capturing opponents' tokens to slow down their progress. This strategy involves higher risk but can be rewarding if executed well.
  - **Defensive Strategy:** Prioritize moving tokens to safe spots, minimizing the risk of being captured. This strategy is generally safer but might result in slower progress.
  - **Balanced Strategy:** A combination of aggressive and defensive strategies, adapting to the game state and opponents' moves.

Chapter 3

Theory

3.1 Q-Learning Algorithm

Q-learning is a model-free reinforcement learning algorithm used to find an optimal action-selection policy for any given finite Markov decision process (MDP). It does not require a model of the environment and can handle problems with stochastic transitions and rewards without requiring adaptations.

3.1.1 Definitions

- **State (s)**: The current situation or configuration.
- **Action (a)**: The set of all possible moves the agent can take.
- **Reward (r)**: The immediate return received after transitioning from state s to state s' due to action a.
- **Q-value ($Q(s, a)$)**: The expected future reward of taking action a in state s.

3.1.2 Q-Value Update Rule

The core of Q-learning is the Q-value update rule, which updates the Q-value $Q(s, a)$ based on the observed reward and the maximum expected future reward. The update rule is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{3.1}$$

where:

- α is the learning rate ,$0 < \alpha \leq 1$.
- γ is the discount factor, $0 \leq \gamma$ ¡ 1.
- s' is the next state after taking action a from state s.
- a' is the next action that maximizes the Q-value for state s'.

3.1.3 Q-Learning Algorithm Steps

1. **Initialize Q-values**: Initialize the Q-values table $Q(s, a)$ arbitrarily (often set to 0).
2. **Observe State**: Observe the current state s.
3. **Choose Action**: Choose an action a based on the current state s (e.g., using an ϵ-greedy policy).
4. **Take Action**: Take the action a, observe the reward r and the next state s'.
5. **Update Q-Value**: Update the Q-value using the Q-value update rule.
6. **Update State**: Set the state s to the new state s'.
7. **Repeat**: Repeat steps 3-6 until convergence or for a specified number of iterations.

3.1.4 Decremental Q-Learning

It assumes the learning rate α in (3.1) to be decreasing with time. It helps in achieving outperformance.

3.2 Game Tree

Our work involves the use of heuristic tree search algorithms to build AI agents. Before proceeding further it is essential to define how tree structure is being used for our purpose.
Game tree is the tree representation of the game. In game tree G=(V,E), V is the collection of all the game states possible and E is the collection of directed edges that join a stat to another if there is an action that can lead to that state.

In the case of tic-tac-toe, board states represent different nodes that are joined if any of the nine actions lead to another board state.

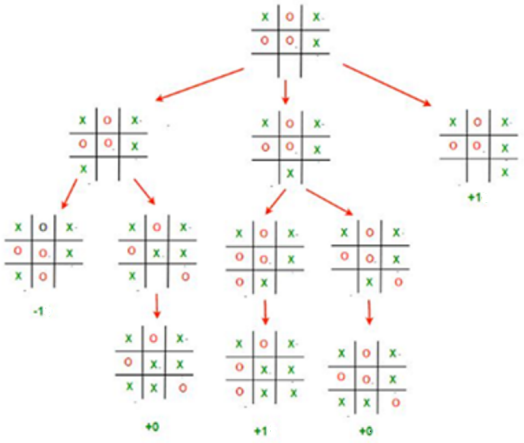
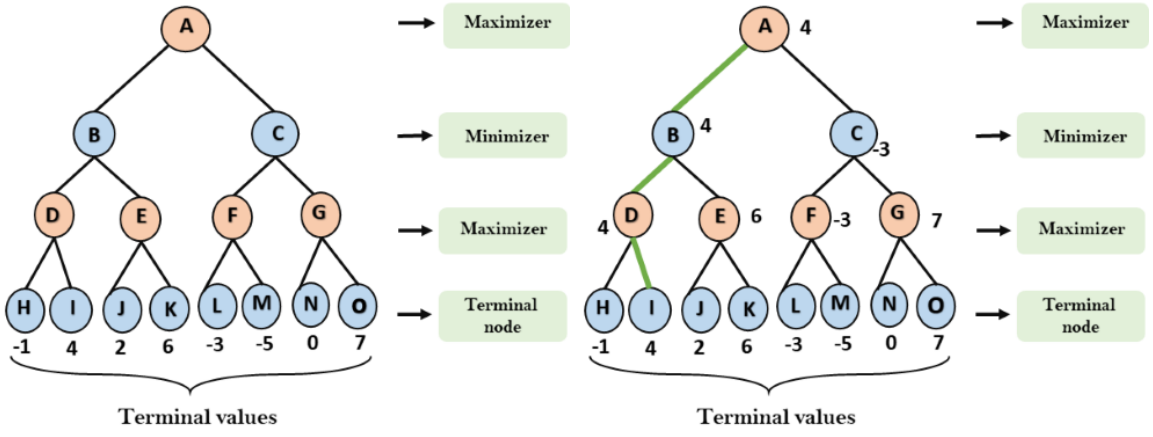


Figure 3.1: Part of a tic-tac-toe game tree

3.3 MinMax Tree



(a) Tree with values only at leaves of a tree (b) Tree with all intermediate values after minmax

Figure 3.2: Illustration of minmax algorithm

The Min Max algorithm is a decision-making algorithm used in the field of game theory and artificial intelligence. We will be using it to build an AI opponent but it

has applications in decision-making, auctions and negotiations. When 2 players say MAX and MIN battle against each other such that they get the maximum benefit and the opponent gets the minimum benefit. Here, MIN and MAX are opponents of each other where MAX wants to maximize its reward and MIN wants to minimize the MAX's reward thus maximizing its own reward. This interplay sets the foundation of the algorithm.

3.3.1 Pseudocode

Game tree maximizer and minimizer alternatively at each level. At the maximizer level, the node value is max of all the child node values as it wants to maximize eventual reward and similarly for minimizer. Here, is the pseudocode:

Algorithm 1 Minimax Algorithm

```
1: function minimax(node, depth, maximizingPlayer)
2: if depth == 0 or node is a terminal node then
3:     return static evaluation of node
4: end if
5: if maximizingPlayer then
6:     maxEva = −∞
7:     for each child of node do
8:         eva = minimax(child, depth - 1, false)
9:         maxEva = max(maxEva, eva)
10:    end for
11:    return maxEva
12: else
13:    minEva = +∞
14:    for each child of node do
15:        eva = minimax(child, depth - 1, true)
16:        minEva = min(minEva, eva)
17:    end for
18:    return minEva
19: end if
```

3.3.2 Properties of MinMax

- Complete: Since it is a complete algorithm, the MinMax algorithm can determine the best move for both players in any two-player zero-sum game. A game

with a zero-sum outcome is one in which the sum of winnings and losses for a player is always equal to zero. That is, if one player wins, the other player must lose.

- Optimal: The MinMax algorithm always finds the optimal move for a player, assuming that the other player is also making optimal moves. This means that if the opponent also uses the MinMax algorithm, the game will always end in a draw. The MinMax algorithm will attempt to reduce the maximum possible loss if the adversary is not employing the algorithm.
- Time complexity is $O(b^d)$, where b is the branching factor and d is the depth of the game tree.
- Space complexity is O(bd),

3.3.3 Limitations of MinMax

The following are some of the drawbacks of the Min-Max algorithm:

- It presumes that the adversary likewise makes the best movements, which may not be the case in actual play. Players might make blunders or employ suboptimal tactics in real-world games. The MinMax algorithm may take a longer time to determine the player's best move in such circumstances.
- The games with a wide search space should not use the MinMax algorithm. Evaluating every move in such games could take a while, and the memory needs might become excessively large.
- The MinMax algorithm does not consider the probability of certain events occurring. The outcome of some games, like poker, is determined by the likelihood that specific things will happen, such as the allocation of cards. The MinMax algorithm might not be appropriate in these games.

3.4 MCTS

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying principle of Monte Carlo methods is using randomness to solve problems that might be

deterministic in principle. These belong to the class of heuristic methods, we are not guaranteed precisely the correct or the best answer, but we can get an approximation that can often be good enough.

In MCTS random sampling/simulations are called playouts. A playout is a simulation of a single game, often from beginning to end. Based on the result that we get after the game ends we update the statistics for each state visited during that game where each state is represented as a node of a game tree. Similar to other Monte Carlo methods, larger the number of playouts, better the result statistically.

Algorithm 2 Monte Carlo Tree Search- Pseudocode

```
Function MCTS(root, iterations):
    for i ← 1 to iterations do
        node ← root  while node is fully expanded and non-terminal do
            node ← select(node)
        end
        if node is non-terminal then
            node ← expand(node)
        end
        reward ← simulate(node) backpropagate(node, reward)
    end
    return best_child(root)
Function select(node):
    return child of node with highest UCT value

Function expand(node):
    add a new child to node for one of the possible moves  return the new child node

Function simulate(node):
    while node is non-terminal do
        node ← a random child of node
    end
    return reward for the terminal state
Function backpropagate(node, reward):
    while node is not null do
        update node's statistics with reward  node ← parent of node
    end

Function best_child(node):
    return child of node with highest visit count
```

Remark 1. *Instead of exploring the complete game tree like MinMax, MCTS doesn't start with the complete tree but builds up the tree based on the UCB*

heuristic. i.e. MCTS approximates MinMax as the number of simulations increases.

We compare our MCTS and MinMax agent against other reinforcement learning methods like Q-learning [3].

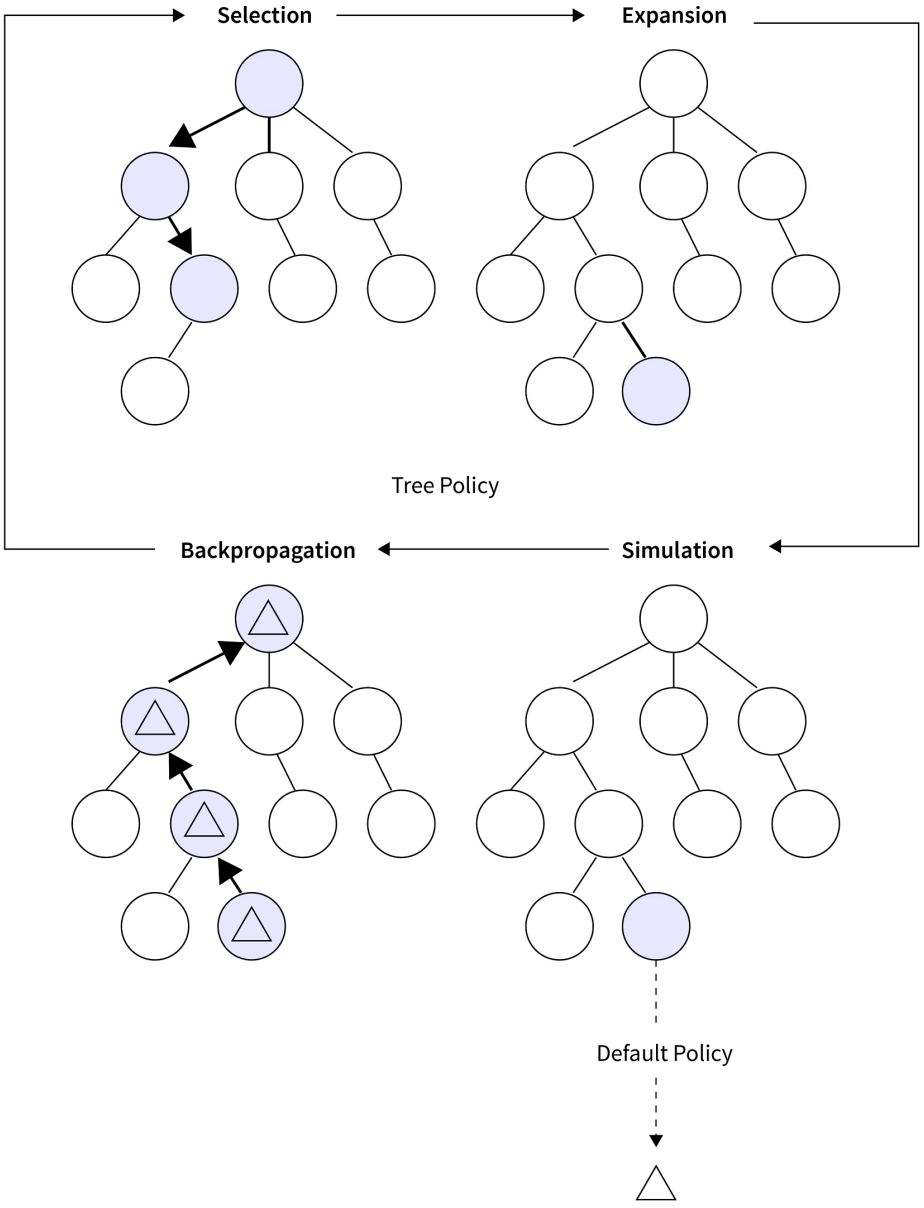


Figure 3.3: Phases of MCTS algorithm

3.5 Various tree policies for MCTS

3.5.1 UCB

$$a_t = \arg \max_{i \in \{1,\ldots,K\}} \left(\hat{\mu}_i + \sqrt{\frac{2 \ln t}{n_i}} \right) \tag{3.2}$$

where:

- a_t is the action chosen at time t.
- K is the number of actions (arms).
- $\hat{\mu}_i$ is the estimated mean reward of action i.
- t is the current time step.
- n_i is the number of times action i has been selected.

In UCB $\hat{\mu}_i$ is the mean reward for the ith node of the game tree.

$$\hat{\mu}_i = w_i / n_i \tag{3.3}$$

where w_i is the number of wins.
UCB was derived using Chernoff-Hoeffding's inequality as shown in [4]. There is lot of research on (3.2) and several new tree policies [5] have been found that are biased and problem-specific. They perform better in one case and worse in another.

Analysis of UCB

The UCB has 2 terms: winrate and exploitation term. Winrate prefers the action which has provided success quite often while the exploitation term is not so trivial. When a node receives fewer visits than its parent node, in relation to the total number of visits, this term gets longer for that node. The natural log of the total number of visits to the parent node is what we have in the numerator. The quantity of visits to the current child node serves as the denominator. The amount of visits to the parent node will eventually raise the exploration term if we don't visit a certain child node. The exploration term will eventually rise to the point where that child node is chosen.
The total exploration period above will progressively grow if we increase the

number of visits to the parent node without stopping at any particular child node. Its growth is, however, sluggish in comparison to the number of visits since it is based on the natural log.The denominator increases each time we visit a child node, resulting in a reduction in the exploration term.
The value of investigating that option will quickly decline if choosing the child node doesn't raise the winrate because the denominator, in contrast to the numerator, is not scaled down. Generally, if a node has not shown promise in the past, it can take a while for it to be chosen again, but if we perform enough playouts, it will happen eventually.

Chapter 4

AI Game Playing agents for Tic-Tac-Toe and Ludo

4.1 Random Agent

It is a most trivial agent to begin with. It makes random moves out of the available valid moves from any game state. It has been implemented for both games.

4.2 Q-Learner for Tic-tac-toe

It is based on the incremental q-learner as described in 3.1. Here we define state space, actions and reward for the game to build the agent.
State Space: Collection of board states.
Actions: 9 tiles on the board correspond to 9 actions. It is to be noted that all actions are not feasible for each state.
Reward: +1 for win, -1 for loss, 0 for a draw and any move that does not lead to termination.

4.3 RuleBased Agent for Ludo

The `RuleBasedPlayer` class is designed to play Ludo using a set of predefined evaluation rules to determine the best moves. The key components of the class

Game State	Top Left	Top Middle	Top Right	Middle Left	Middle Middle	Middle Right	Bottom Left	Bottom Middle	Bottom Right
	N/A	0.5	N/A	N/A	N/A	N/A	0	0	N/A
	N/A	N/A	N/A	N/A	N/A	N/A	0.5	N/A	N/A
	0.3	0.5	0.3	0.5	0.7	0.3	0.3	0.5	0.3
...	...	...	...	...	...	...	...	...	...

Figure 4.1: Q-table for tic-tac-toe

include evaluating pieces, evaluating threats, calculating the overall evaluation of the board, and selecting the best move. Here's a detailed explanation of each part:

4.3.1 Evaluating Pieces

The `eval_piece` method assigns a value to a piece based on its distance from the exit. The closer a piece is to the exit, the higher its value. This is calculated as:

$$\text{value} = (39 - \text{distance}) \times \text{count}$$

where distance is the number of squares away from the exit and count is the number of pieces on that square.

4.3.2 Evaluating Threats

The `eval_threats` method assesses the threats to a piece from opposing pieces within capturing range (1 to 6 squares away). It evaluates both forward and backward threats:

- For pieces ahead of the current piece, if they belong to an opponent, their value (based on their distance to their own exit) is added to the threat score.
- For pieces behind the current piece, if they belong to an opponent, the value of the current piece is subtracted from the threat score.

The total threat score is then divided by 6 to normalize it.

4.3.3 Overall Board Evaluation

The `eval` method calculates the overall score of the current board for the player. It does this by:

- Iterating through all the board positions and evaluating each piece.
- Adding the value of the player's own pieces to the score and subtracting the value of the opponent's pieces.
- Including the threat evaluations for the player's pieces.
- Adding points for pieces in the exit state (50 points per piece).
- Adding points for pieces that have exited the board (60 points per piece).

4.3.4 Playing the Best Move

The `play` method selects the move that results in the highest evaluation score. It evaluates each possible move by temporarily applying the move, calculating the board score, and then reverting the move. The move with the highest score is chosen.

4.3.5 Summary

The `RuleBasedPlayer` class uses a rule-based approach to evaluate and play moves in Ludo. It prioritizes advancing pieces towards the exit, taking opponent pieces, and protecting its own pieces from being captured. The decision-making process involves evaluating all possible moves and selecting the one with the highest score based on these criteria. This method ensures a strategic approach to moving pieces effectively and safely while maximizing the player's chances of winning.

4.4 MinMax and MCTS Agents

4.4.1 For Tic-tac-toe

The complete game tree for tic-tac-toe is too large for representation purposes. So, we depict the subgraph of it. We use symmetry to reduce the number of nodes drastically to 754. State space, actions and reward is the same as q-learner but, it is to be noted that rewards are returned as node value by leaves of the game tree. MinMax does an exhaustive search on a game tree while MCTS

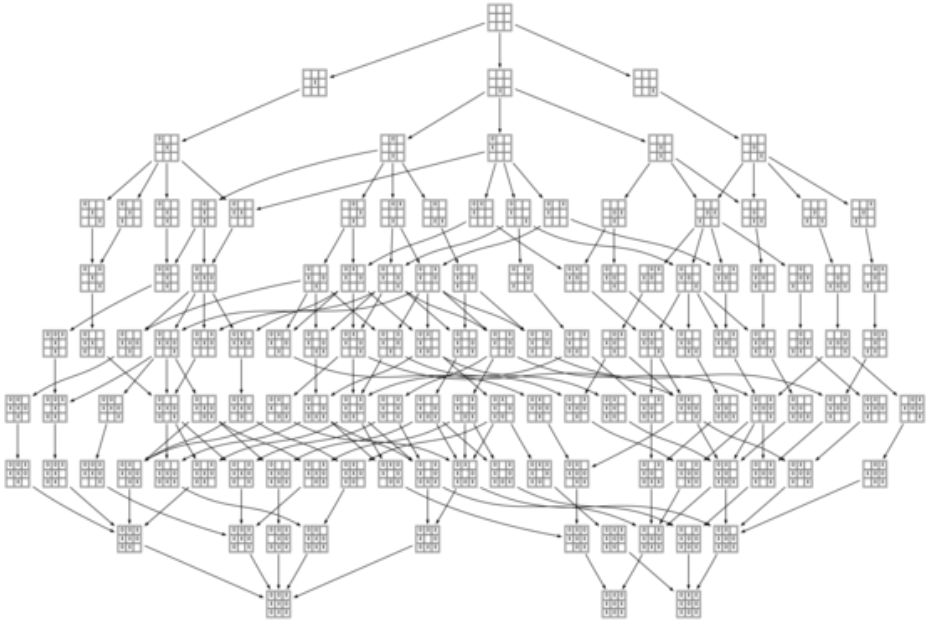


Figure 4.2: Subgraph of tic-tac-toe game tree where both players optimally

explores a game tree with simulations.

4.4.2 Random MinMax Agent

Since, MinMAx performs a deterministic exhaustive search it chooses a fixed action everytime out of the actions with the same reward. Random MinMax does this uniformly at random.

4.4.3 For Ludo

Unlike tic-tac-toe, the game tree of Ludo is not perfect. It doesn't have a fixed number of nodes. But, it is guaranteed to that any game will end and from there reward is backpropagated to the intermediate nodes.

Chapter 5

Bayesian Inference in MCTS

Bayesian inference [6] is a statistical method used to update the probability of a hypothesis as more evidence or information becomes available. It relies on Bayes' theorem, which describes the relationship between the probability of a hypothesis before and after accounting for new evidence.

5.1 Bayesian Inference

5.1.1 Key Concepts

a. **Bayes' Theorem:**

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Where:

- $P(H|E)$ is the posterior probability: the probability of the hypothesis H given the evidence E.
- $P(E|H)$ is the likelihood: the probability of observing the evidence E given that hypothesis H is true.
- $P(H)$ is the prior probability: the initial probability of the hypothesis H before seeing the evidence.
- $P(E)$ is the marginal likelihood: the total probability of observing the evidence E under all possible hypotheses.

b. **Prior Probability ($P(H)$)**: This represents what is known or believed about the hypothesis before new evidence is taken into account. It can be subjective and based on prior knowledge or assumptions.
c. **Likelihood ($P(E|H)$)**: This measures how likely the observed evidence is, assuming that the hypothesis is true.
d. **Posterior Probability ($P(H|E)$)**: This is the updated probability of the hypothesis after considering the new evidence.
e. **Marginal Likelihood ($P(E)$)**: This is a normalizing constant ensuring that the posterior probabilities sum to one. It is calculated as the sum of the likelihoods across all possible hypotheses, weighted by their prior probabilities.

5.1.2 Steps in Bayesian Inference

1. **Specify the Prior:** Determine the prior distribution that represents the initial beliefs about the parameters.
2. **Collect Data:** Gather new evidence or data.
3. **Compute the Likelihood:** Calculate the likelihood of the observed data under different hypotheses.
4. **Apply Bayes' Theorem:** Use Bayes' theorem to update the prior distribution with the likelihood to obtain the posterior distribution.
5. **Interpret the Posterior:** The posterior distribution provides updated beliefs about the parameters after considering the evidence.

5.1.3 Advantages

- **Incorporates Prior Knowledge:** Allows the inclusion of prior beliefs and expert knowledge.
- **Flexibility:** Can handle complex models and update probabilities as new data becomes available.
- **Quantifies Uncertainty:** Provides a probabilistic framework for dealing with uncertainty in model parameters.

5.1.4 Challenges

- **Computational Complexity:** Bayesian methods can be computationally intensive, especially for large datasets or complex models.
- **Choice of Prior:** Selecting an appropriate prior can be subjective and impact the results.

Bayesian inference provides a powerful framework for updating beliefs in the light of new evidence, making it a cornerstone of modern statistical analysis and decision-making.

5.1.5 Bayesian Inference for the Beta Distribution

Beta Distribution

The Beta distribution is parameterized by two positive shape parameters α and β. Its probability density function (PDF) is given by:

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \tag{5.1}$$

where $B(\alpha, \beta)$ is the Beta function, defined as:

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\, dt = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{5.2}$$

Prior Distribution

In Bayesian inference, we start with a prior distribution for the parameter of interest. If the prior distribution is a Beta distribution with parameters α_0 and β_0, then the prior is:

$$\textsf{Prior:} \quad \theta \sim \textsf{Beta}(\alpha_0, \beta_0) \tag{5.3}$$

Likelihood

Suppose we have observed data consisting of n Bernoulli trials with x successes and $n - x$ failures. The likelihood function for the data, given the parameter θ, is:

$$\textsf{Likelihood:} \quad P(X = x \mid \theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x} \tag{5.4}$$

Posterior Distribution

Using Bayes' theorem, the posterior distribution is proportional to the product of the likelihood and the prior:

$$\text{Posterior:} \quad P(\theta \mid X = x) \propto P(X = x \mid \theta) \cdot P(\theta) \tag{5.5}$$

Given the Beta prior and the Binomial likelihood, the posterior distribution is also a Beta distribution. Specifically, if the prior is $\mathsf{Beta}(\alpha_0, \beta_0)$, then the posterior is:

$$\theta \mid X = x \sim \mathsf{Beta}(\alpha_0 + x, \beta_0 + n - x) \tag{5.6}$$

We will prove the claim (5.6) now.

Since,

$$P(\theta) = \frac{\theta^{\alpha_0 - 1}(1-\theta)^{\beta_0 - 1}}{B(\alpha_0, \beta_0)} \tag{5.7}$$

where $B(\alpha_0, \beta_0) = \frac{\Gamma(\alpha_0)\Gamma(\beta_0)}{\Gamma(\alpha_0 + \beta_0)}$ is the Beta function.

Likelihood Function

$$P(X = x \mid \theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x} \tag{5.8}$$

Apply Bayes' Theorem

$$P(\theta \mid X = x) \propto P(X = x \mid \theta) \cdot P(\theta) \tag{5.9}$$

Substitute the likelihood and prior into Bayes' theorem:

$$P(\theta \mid X = x) \propto \left[\binom{n}{x} \theta^x (1-\theta)^{n-x}\right] \cdot \left[\frac{\theta^{\alpha_0 - 1}(1-\theta)^{\beta_0 - 1}}{B(\alpha_0, \beta_0)}\right] \tag{5.10}$$

Since $\binom{n}{x}$ and $B(\alpha_0, \beta_0)$ are constants with respect to θ, they can be combined into a normalizing constant C:

$$P(\theta \mid X = x) \propto \theta^x (1-\theta)^{n-x} \cdot \theta^{\alpha_0 - 1}(1-\theta)^{\beta_0 - 1} \tag{5.11}$$

Combine the exponents:

$$P(\theta \mid X = x) \propto \theta^{x+\alpha_0-1}(1-\theta)^{n-x+\beta_0-1} \tag{5.12}$$

Now,Identify the Posterior Distribution

The expression $\theta^{x+\alpha_0-1}(1-\theta)^{n-x+\beta_0-1}$ is the kernel of a Beta distribution. Therefore, the posterior distribution is:

$$\theta \mid X = x \sim \mathsf{Beta}(x + \alpha_0, n - x + \beta_0) \tag{5.13}$$

5.2 Extremum Distribution for Independent and Non-Identically Distributed Sample

The extremum distribution for a sample of independent and non-identically distributed (i.n.i.d.) random variables can be complex, as the lack of identical distribution adds layers of variability. However, the general principles can be outlined and understood with the aid of extreme value theory and order statistics.

5.2.1 Order Statistics for i.n.i.d. Sample

Given a sample $X_1, X_2, \ldots, X_n$ of independent but not necessarily identically distributed random variables with CDFs $F_1(x), F_2(x), \ldots, F_n(x)$ and PDFs $f_1(x), f_2(x), \ldots, f_n(x)$, the k-th order statistic $X_{(k)}$ is the k-th smallest value in the sample.

5.2.2 Distribution of the Maximum (n-th Order Statistic)

The maximum of the sample $X_{(n)} = \max(X_1, X_2, \ldots, X_n)$ can be found by considering the CDF of $X_{(n)}$.

The CDF of the maximum $X_{(n)}$ is given by:

$$\begin{aligned} F_{X_{(n)}}(x) &= P(X_{(n)} \leq x) \\ &= P(X_1 \leq x, X_2 \leq x, \ldots, X_n \leq x) \\ &= \prod_{i=1}^{n} F_i(x) \end{aligned} \tag{5.14}$$

Since the variables are independent, the joint probability is the product of the individual probabilities.

The PDF of the maximum $X_{(n)}$ is the derivative of the CDF:

$$f_{X_{(n)}}(x) = \frac{d}{dx} F_{X_{(n)}}(x) = \frac{d}{dx} \left(\prod_{i=1}^{n} F_i(x) \right) \tag{5.15}$$

5.2.3 Distribution of the Minimum (1st Order Statistic)

The minimum of the sample $X_{(1)} = \min(X_1, X_2, \ldots, X_n)$ can be found by considering the CDF of $X_{(1)}$.

The CDF of the minimum $X_{(1)}$ is given by:

$$\begin{aligned} F_{X_{(1)}}(x) &= P(X_{(1)} \leq x) \\ &= 1 - P(X_{(1)} > x) \\ &= 1 - P(X_1 > x, X_2 > x, \ldots, X_n > x) \\ &= 1 - \prod_{i=1}^{n} (1 - F_i(x)) \end{aligned} \tag{5.16}$$

Again, using the independence of the variables, the joint probability is the product of the individual complementary probabilities.

The PDF of the minimum $X_{(1)}$ is the derivative of the CDF:

$$f_{X_{(1)}}(x) = \frac{d}{dx} F_{X_{(1)}}(x) = \frac{d}{dx} \left(1 - \prod_{i=1}^{n} (1 - F_i(x)) \right) \tag{5.17}$$

5.3 Bayesian inference in MCTS

In [7] an approach inspired by bayesian inference is used to modify the estimates involved in (3.2). It proposes the following:
1. Node value/ reward stored in the intermediate nodes of the game tree to have a prior distribution. In case of situations like tic-tac-toe game where there is only 0/1 payout beta distribution as a prior can be used. $\alpha = 1, \beta = 1$ is chosen for prior distribution as it leads to an intuitive uniform distribution.
2. The main idea of the method lies in better estimation of $\hat{\mu}$ in equation (3.2). In this approach estimator has been redefined as the expectation of the maximum of child node values. child node values are independently distributed with different priors.

$$\begin{aligned} cdf_{\max}(x) &= \prod_{i=1}^{K} cdf_i(x), \\ pdf_{\max}(x) &= \frac{d}{dx} cdf_{\max}(x), \\ \hat{\mu} &= \int_{-\infty}^{\infty} x \, pdf_{\max}(x) \, dx. \end{aligned} \tag{5.18}$$

5.3.1 Approximations and Technical Challenges in the Approach

Since, K in (5.18) is equal to the branching factor of the node therefore in order to find the exact pdf is a challenging and time expensive task using symbolic computation which involves the product of functions and derivatives thereafter. Calculation of the expectation as a last step may not be possible using symbolic computation.

Solution and Approximations

To make the algorithm faster, the paper proposes Gaussian priors although it can lead to accumulation of errors with iterations and also because Gaussian prior is not closed for max extremum distribution.
With Gaussian approximation of prior comes one more thing handy that is analytic form of mean of max of independently distributed sample is available which is used as a bypass to integration and differentiation. Lastly, the analytic

form also performs some floating point operations, to avoid them paper suggests the use of a lookup table to make the computation even faster.

Chapter 6

Result and Discussion

6.1 Tic-tac-toe

6.1.1 Payoff Matrix

We use 1000 simulations for each pair of opponents, Each agent gets a chance to make a first move in half of the total number of games. Payoff matrix, P has entries P(i,j)=(win ratio of agent i, win ratio of agent j). It can be noted that win ratios do not add up to 1 as it is compensated by the draw ratio.

	Random	Q Learn	Minimax	Random Minimax	MCTS
Random		$(0.0, 85.25)$	$(0.0, 91.2)$	$(0.0, 87.45)$	$(0.0, 71.1)$
Q Learn	$(85.25, 0.0)$		$(0.0, 0.0)$	$(0.0, 0.55)$	$(0.0, 0.0)$
Minimax	$(91.2, 0.0)$	$(0.0, 0.0)$		$(0.0, 0.0)$	$(0.0, 0.0)$
Random MinMax	$(87.45, 0.0)$	$(0.55, 0.0)$	$(0.0, 0.0)$		$(0.0, 0.0)$
MCTS	$(71.1, 0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	

Minmax agent > Random MinMax agent >Q-Learn agent > MCTS.
It is noteworthy that MCTS performs poorly despite being sophisticated. Training for more simulations might improve its performance. It indicates that MCTS did not explore optimum arms with limited simulations and has not converged to a better policy.

6.1.2 First move advantage

	Random	Q Learn	Minimax	Random Minimax	MCTS
Random	$(55.7, 29.0)$	$(0.4, 74.6)$	$(0.0, 82.7)$	$(0.0, 78.6)$	$(0.0, 61.0)$
Q Learn	$(95.9, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$
Minimax	$(99.7, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$
Random MinMax	$(96.3, 0.0)$	$(1.1, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$
MCTS	$(81.2, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$	$(0.0, 0.0)$

Unlike the previous payoff matrix, this now is not a symmetric matrix as P(i,j)=(win ratio of agent i, win ratio of agent j) when i made the first move.
It is absolutely evident that the first move gives an advantage to the playing agent. Even for weak random agent, first move advantage is quite significant.

6.2 Ludo: 4 player

6.2.1 Effect of spatiality of players

We make a hypothesis that the position of players in Ludo must impact the chances of winning. Four players can sit in $(4-1)! = 6$ ways. We check the winning percentage of each player when they compete against each other in different position.

Side 1	Side 2	Side 3	Side 4
Random	MinMax	MCTS	RuleBased
8.0	28.8	33.4	29.8

Side 1	Side 2	Side 3	Side 4
Random	MinMax	RuleBased	MCTS
6.0	33.0	25.2	35.8

Side 1	Side 2	Side 3	Side 4
Random	MCTS	MinMax	RuleBased
3.8	40.2	33.6	35.8

Side 1	Side 2	Side 3	Side 4
Random	MCTS	RuleBased	MinMax
6.2	34.0	27.0	32.8

Side 1	Side 2	Side 3	Side 4
Random	RuleBased	MinMax	MCTS
6.8	26.6	32.4	34.2

Side 1	Side 2	Side 3	Side 4
Random	RuleBased	MCTS	MinMax
5.4	23.2	40.2	31.2

The cumulative performance of agents over all relative positions is:

Random	MinMax	MCTS	RuleBased
6.034	31.967	36.3	25.7

There are 2 observations that can be made based on the empirical evidence which are as follows:

1. MCTS $>$ MinMax $\gtrsim$ RuleBased $>$ Random. where ordering is on the skill of AI agents.
2. Two patterns can be observed:

   – The winning ratio of a player in a particular game setting depends primarily on the skill of the adjacent player with the next turn. Random, rulebased and minmax agent **perform worst** in the respective settings where the next player is MCTS.

   – MCTS performs best when the next adjacent player is MinMax.

Other patterns can also be observed but they are not true for each game setting above.

6.2.2 Payoff Matrix for 2 Player Ludo

We use 500 simulations for each pair of opponents, Each agent gets a chance to make a first move in half of the total number of games. The payoff matrix has entries P(i,j)=(win ratio of agent i, win ratio of agent j). It can be noted that win ratios do not add up to 1 as it is compensated by the draw ratio

	Random	RuleBased	Minimax	MCTS
Random		$(13.4, 86.6)$	$(18, 82)$	$(12.2, 87.8)$
RuleBased	$(86.6, 13.4)$		$(50.6, 49.4)$	$(41.8, 58.2)$
Minimax	$(82, 18)$	$(49.4, 50.6)$		$(40.2, 59.8)$
MCTS	$(87.8, 12.2)$	$(58.2, 41.8)$	$(59.8, 40.2)$	

MCTS agent> Rulebased agent >MinMAx agent > Random agent.
This was expected as MCTS Agent is best suited for imperfect information games due to its adaptive and probabilistic nature. Rule-based agent performs well with good rules but is less flexible and harder to adapt to new situations. Minimax agent is more suited to perfect information games and struggles with hidden information. Random agent is least effective, lacks strategy and planning.

Chapter 7

Future Work

Games in general are of combinatorial complexity with large state and action spaces. Alphago [8] by Google in 2016 for the game 'Go' was groundbreaking. Since then there has been a rise in research area where MCTS is combined with neural networks to build stronger than before gaming agents.

The AlphaGo-inspired training methodology was also applied to the game of Hex. Gao et al. (2017) introduced a player named MoHex-CNN, [9] which outperformed the previous leading agent, MoHex 2.0 (Huang et al., 2013). MoHex-CNN incorporated a move-prediction mechanism based on a deep convolutional network acting as a policy network, which was integrated with MCTS. Later, Gao et al. (2018) developed a novel three-head neural network architecture to significantly enhance MoHex-CNN, resulting in a new player called MoHex-3HNN [10]. This network produces outputs for policy, state values, and action values.

Takada et al. (2019) proposed a different approach named DeepEzo [11], which achieved a 79.3% win rate against MoHex 2.0, the then world champion. The authors trained value and policy functions using reinforcement learning and self-play, similar to AlphaGo. However, the unique aspect of DeepEzo is that the policy function was trained directly from the game outcomes of agents not utilizing the policy function, meaning the policy function training did not involve continuous self-improvement.

After inspiration from a lot of the above work, work in the direction of building excellent AI agents for tictac-toe and especially for ludo can be done by harnessing and combining the power of neural networks and MCTS.

The main objective of the research would to reduce the depth and breadth of the

MCTS. As discussed in earlier papers, depth can be decreased by estimating the value of the state $v(s)$ and breadth can be decreased by a better policy heuristic. Various neural network architectures can be tried with MCTS in order to achieve exceptional skill in game playing.

There is a lot to be explored regarding MCTS and its applications. Next, we can try to combine evolutionary algorithms with MCTS. [12],[13] and [14] provide a good direction to begin with.

Applications of MCTS are not limited to gaming agents. It has applications in planning, security from attackers, chemical synthesis and scheduling.[15] provides how it can help in Bayesian optimization by variable selection thus reducing the complexity of Bayesian optimization, this is quite an enriching application. Such applications beyond games can be explored in a long time horizon from now.

Bibliography

[1] 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, pp. 106–113, 03 2012.

[2] A. Plaat, "Mtd(f), a minimax algorithm faster than negascout," 04 2014.

[3] J. Johnson, J. Li, and Z. Chen, "Reinforcement learning: An introduction: R.s. sutton, a.g. barto," *Neurocomputing - IJON*, vol. 35, pp. 205–206, 11 2000.

[4] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs.," *Journal of Machine Learning Research*, vol. 3, pp. 397–422, 01 2002.

[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, 07 2022.

[6] G. Casella and R. Berger, *Statistical Inference*. 04 2024.

[7] G. Tesauro, V. Rajan, and R. Segal, "Bayesian inference in monte-carlo tree search," *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, UAI 2010*, 03 2012.

[8] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[9] C. Gao, R. Hayward, and M. Müller, "Move prediction using deep convolutional neural networks in hex," *IEEE Transactions on Games*, vol. PP, pp. 1–1, 12 2017.

[10] C. Gao, K. Takada, and R. Hayward, "Hex 2018: Mohex3hnn over deepezo," *ICGA Journal*, vol. 41, pp. 1–4, 03 2019.

[11] C. Gao, K. Takada, and R. Hayward, "Hex 2018: Mohex3hnn over deepezo," *ICGA Journal*, vol. 41, pp. 1–4, 03 2019.

[12] A. Benbassat and M. Sipper, "Evomcts: Enhancing mcts-based players through genetic programming," pp. 1–8, 08 2013.

[13] A. Alhejali and S. Lucas, "Using genetic programming to evolve heuristics for a monte carlo tree search ms pac-man agent," pp. 1–8, 08 2013.

[14] D. Perez Liebana, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary mcts for general video game playing," 08 2014.

[15] M. U. Chaudhry and J.-H. Lee, "Motifs: Monte carlo tree search based feature selection," *Entropy*, vol. 20, 05 2018.

[16] H. Baier and M. Winands, "Monte-carlo tree search and minimax hybrids," pp. 1–8, 08 2013.

[17] H. Baier and M. Winands, "Mcts-minimax hybrids," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, pp. 167–179, 06 2015.

[18] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius, "Evolving game-specific ucb alternatives for general video game playing," pp. 393–406, 03 2017.

[19] T. Cazenave, "Multi-player go.," vol. 5131, pp. 50–59, 09 2008.

[20] S. Permana, "Implementation of min max algorithm as intelligent agent on card battle game," *IJISTECH (International Journal Of Information System Technology)*, vol. 2, p. 53, 05 2019.

[21] X. Kang, Y. Wang, and Y. Hu, "Research on different heuristics for minimax algorithm insight from connect-4 game," *Journal of Intelligent Learning Systems and Applications*, vol. 11, pp. 15–31, 01 2019.