DSCI -6612-01 Intro Artificial Intelligence Fall 2023

TIC TAC TOE GAME USING AI STRATEGIES

<u>TEAM MEMBERS</u>

TEJA REDDY
DURGA SYAMALA

Instructor: Dr. Shivanjali Khare

ABSTRACT

This study compares the performance of two AI algorithms, Q-learning and Markov Chain Monte Carlo (MCMC), in playing Tic-Tac-Toe. Through simulations, the effectiveness of these algorithms is evaluated based on factors such as win rate, decision-making speed, adaptability, and handling opponent strategies. The findings contribute to understanding AI strategies in game playing and decision-making, providing insights into the strengths and limitations of Q-learning and MCMC in the context of Tic-Tac-Toe.

Table of Contents

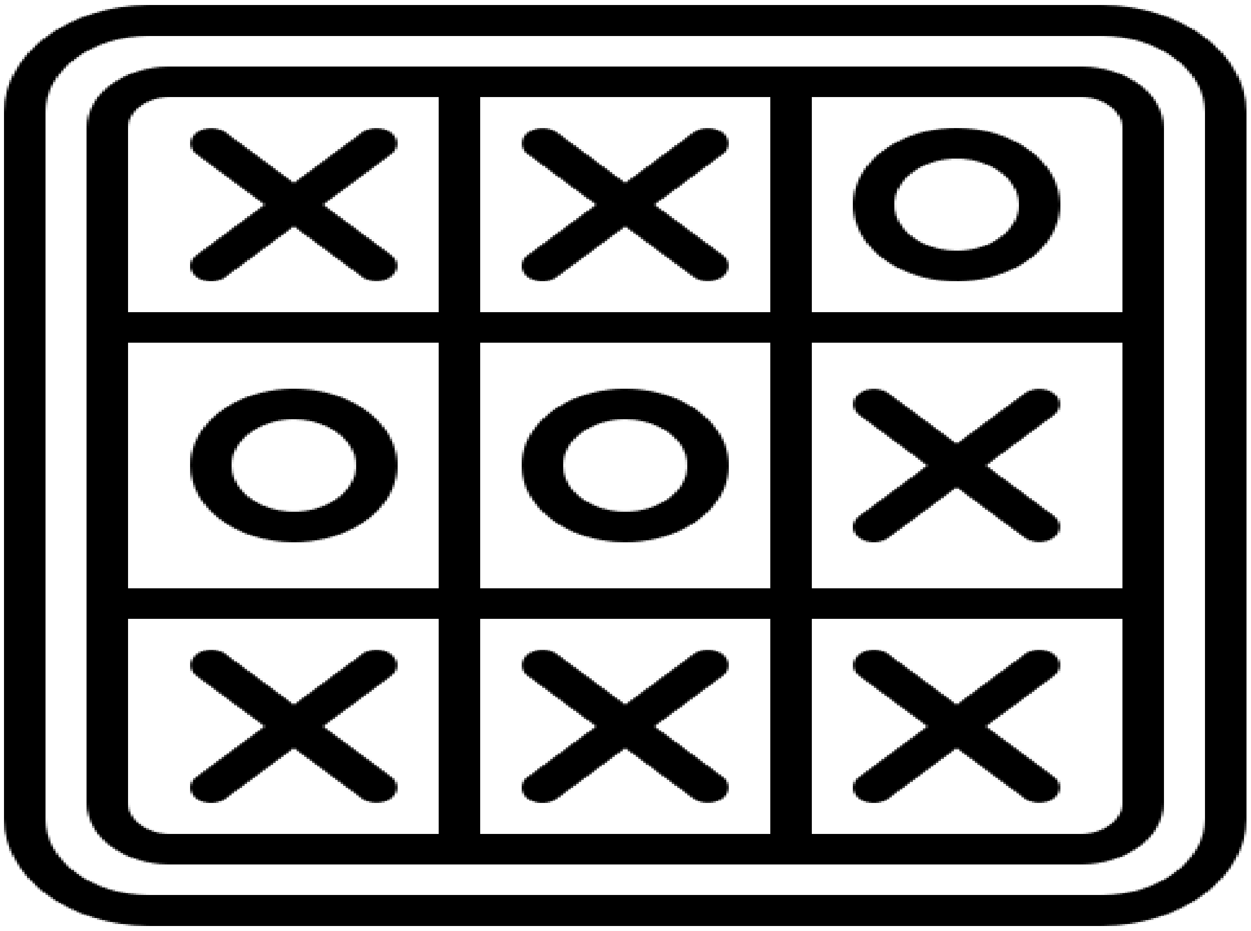
1.Introduction

Tic-Tac-Toe is a classic game played on a 3x3 grid where players take turns marking spaces with "X" or "O". The objective is to create a line of three of their symbols horizontally, vertically, or diagonally to win. It's often used as a simple example in AI and game theory for decision-making and strategic planning. In this study, we explore the application of AI algorithms in playing Tic-Tac-Toe. Specifically, we focus on comparing the performance of two AI algorithms: Q-learning and Markov Chain Monte Carlo (MCMC). By simulating games between Q-learning and MCMC players, we aim to evaluate their effectiveness in playing Tic-Tac-Toe and understand the strengths and limitations of each approach.

2.Overview

2.1. ***Project overview***

Tic-Tac-Toe, a classic game played on a 3x3 grid, has long served as an illustrative example in the realms of AI and game theory, showcasing principles of decision-making and strategic planning. In this study, we delve into the realm of AI algorithms and their application in playing Tic-Tac-Toe. Our primary focus is to compare the performance of two specific AI algorithms: Q-learning and Markov Chain Monte Carlo (MCMC). The objective of the game is simple: players take turns marking spaces on the grid with their respective symbols, "X" or "O," with the ultimate goal of forming a line of three of their symbols either horizontally, vertically, or diagonally to achieve victory. Through simulated games between Q-learning and MCMC players, we aim to rigorously evaluate the effectiveness of these AI algorithms in playing Tic-Tac-Toe. By analyzing various performance metrics such as win rates, decision-making speed, adaptability to different board states, and the ability to counter opponent strategies, we can gain a comprehensive understanding of the strengths and limitations of each approach. Q-learning, reinforcement learning algorithm, learns through a process of trial and error. It constructs a table of state-action pairs and continually updates their values based on the rewards and penalties received during gameplay. On the other hand, MCMC, a probabilistic algorithm, explores the game's state space by employing random sampling techniques, making decisions based on probabilities. By conducting these comparisons, we aim to uncover valuable insights into the capabilities of Q-learning and MCMC algorithms in the context of Tic-Tac-Toe. The findings of this study will contribute to the broader understanding of AI strategies in game playing and decision-making, providing valuable knowledge for future advancements in AI-based game playing and strategic planning.



TIC-TAC-TOE GAME IMAGE

2.2. **Project goal overview**

The goal of this project is to explore and compare the performance of two AI algorithms, Q-learning and Markov Chain Monte Carlo (MCMC), in playing the game of Tic-Tac-Toe. By simulating games between players utilizing these algorithms, the project aims to achieve the following objectives:

1. Evaluate Effectiveness: Assess the effectiveness of Q-learning and MCMC algorithms in playing Tic-Tac-Toe by analyzing metrics such as win rates, decision-making speed, adaptability to different board states, and the ability to counter opponent strategies.
2. Understand Strengths and Limitations: Gain insights into the strengths and limitations of Q-learning and MCMC approaches in the context of Tic-Tac-Toe. Identify scenarios in which each algorithm excels and areas where they may struggle.
3. Comparative Analysis: Conduct a comprehensive comparative analysis between Q-learning and MCMC algorithms to determine which approach performs better in terms of overall game-playing performance.
4. Enhance AI Strategies: Contribute to the understanding and advancement of AI strategies in game playing and decision-making. Provide valuable knowledge and insights that can be utilized in future developments of AI-based game playing and strategic planning.

By achieving these project goals, the study aims to enhance our understanding of AI algorithms in the context of Tic-Tac-Toe and contribute to the broader field of AI and game theory.

3.Features & variables

Features and Variables for the Project:

1. Algorithm Type: This refers to the specific AI algorithms being compared in the project: Q-learning and Markov Chain Monte Carlo (MCMC).
2. Win Rate: This indicates the percentage of games won by each algorithm, showing how successful they are in winning Tic-Tac-Toe matches.
3. Decision-making Speed: This measures how quickly each algorithm makes decisions during gameplay, reflecting their efficiency in selecting moves.
4. Adaptability: This assesses the ability of the algorithms to adjust and adapt their strategies to different board states, indicating how well they handle varying game situations.
5. Opponent Strategy Handling: This evaluates how effectively the algorithms respond to and counter different strategies used by opponents, showcasing their ability to anticipate and counteract opponent moves.
6. Game State Exploration: This quantifies the extent to which the algorithms explore and analyze different game states during play, demonstrating the depth of their decision-making process.
7. Strengths and Limitations: These summarize the notable advantages and disadvantages observed for each algorithm, highlighting their specific areas of expertise and potential challenges

These features and variables provide insights into the performance, characteristics, and suitability of the Q-learning and MCMC algorithms in playing Tic-Tac-Toe. By analyzing and comparing these aspects, we can gain a comprehensive understanding of their capabilities and limitations in the game.

4.Environment

The project takes place in a simulated environment that emulates the game of Tic-Tac-Toe. This environment consists of the following elements:

1. Game Board: The game board is represented as a 3x3 grid where players can mark spaces with "X" or "O". It serves as the playing field for the game.
2. Game State: The current state of the game is continuously tracked, reflecting the arrangement of symbols on the game board. It is updated after each move, capturing the progress of the game.
3. Move Validation: The environment ensures that each move made by the AI algorithms is valid, preventing players from selecting already occupied spaces or making illegal moves.
4. Opponent Moves: The environment simulates the moves of the opponent player, either controlled by the other AI algorithm or following a predetermined strategy.
5. Win Condition: After each move, the environment checks if any player has achieved a winning combination of symbols, such as three in a row horizontally, vertically, or diagonally.
6. Reward System: The environment assigns rewards to the AI algorithms based on the game outcomes. Winning results in positive rewards, while losing or making invalid moves leads to negative rewards or penalties.
7. Game Termination: The game ends when a player wins, resulting in a conclusive outcome, or when the game board is completely filled without a winner, leading to a draw.

This simulated environment provides a controlled setting for evaluating the performance and behavior of the Q-learning and MCMC algorithms in playing Tic-Tac-Toe. It ensures fair gameplay and allows the algorithms to interact with the game state and opponent moves, enabling a comprehensive analysis of their strategies and decision-making processes.

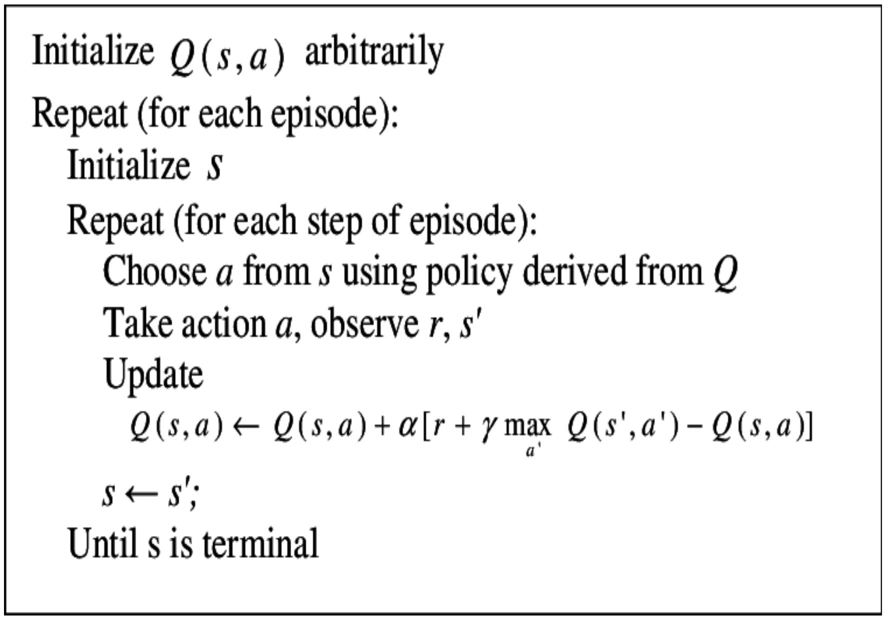
5. Algorithm

5.1. Q-learning Algorithm:

The Q-learning algorithm and A3C (Asynchronous Advantage Actor-Critic) is utilized in this project. This specific algorithm belongs to the group of model-free Reinforcement Learning algorithms, and it employs features to estimate the Q-values for every potential move in a given state. In this context, the Q-value represents the expected long-term reward for performing a particular action in a specific state.

Here is a breakdown of the algorithm's functioning:

1. To begin, assign arbitrary values to the Q-values of each state-action pair
2. Find an epsilon value for the epsilon-greedy strategy that regulates the trade-off between exploration and exploitation during the decision-making phase of the agent.
3. To complete each episode, follow these steps: a. Start with a jumbled arrangement of the chess. b. Determine the next move to make by considering the Q-values and following the epsilon-greedy policy. c. Execute the selected action on the chess, resulting in a new configuration. d. Calculate the reward for the new state. e. Update the Q-value for the previous state-action combination based on the reward and the Q-value of the new state. f. Repeat steps b-e until the chess is solved.
4. For a certain number of episodes, repeat step 3 to let the agent learn and find out the best moves needed to solve the chess.



```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$
        Take action $a$, observe $r$, $s'$
        Update
            $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$;
    Until s is terminal
```

6.Training

During the training phase of this project, we teach the AI algorithms, Q-learning and Markov Chain Monte Carlo (MCMC), how to play Tic-Tac-Toe. Here's a simplified breakdown of the training process:

1. Getting Ready: We set up the initial settings and tools needed for training the algorithms.
2. Playing and Learning: The algorithms play many games against each other or a basic opponent. We keep track of the game states, moves made, and the rewards earned in each game. This data helps train the algorithms.
3. Q-learning Training:

Building Knowledge: We create a table that maps different game situations to expected rewards. This helps the algorithm make smarter decisions.

Learning from Experience: We go through the recorded data and:

- Choose a game situation.
- Decide on an action, either by picking the best option based on previous experience or by trying something new.
- Use a formula to update the expected reward for the chose action in the chosen game situation.
- Repeat this process multiple times, gradually improving the expected rewards for different game situations and actions.

1. MCMC Training:

- Creating Alternatives: We develop a way to generate new game situations based on the current situation and a chosen action.
- Learning from Alternatives: We go through the recorded data and:
- Select a game situation.
- Try out different actions based on their probabilities, favoring actions that seem more promising.
- Generate new game situations by applying the chosen action.
- Evaluate the new situations to see how good they are.
- Adjust the probabilities of choosing different actions based on the evaluations.
- Repeat this process multiple times, gradually improving the action selection

probabilities based on the evaluations.

2. Checking Progress: We assess how well the trained algorithms perform by having them play against each other or a basic opponent. We look at factors like how often they win, how quickly they make decisions, and how adaptable they are.

3. Improving and Trying Again: We analyze the results and make necessary adjustments. This might involve changing settings, improving the way game situations are represented, or tweaking the decision-making strategies. Then, we repeat the training process, incorporating these changes, to make the algorithms better at playing Tic-Tac-Toe.

Through the training phase, we enable the AI algorithms to learn from their experiences and develop strategies for playing Tic-Tac-Toe more effectively.

7.Evaluation

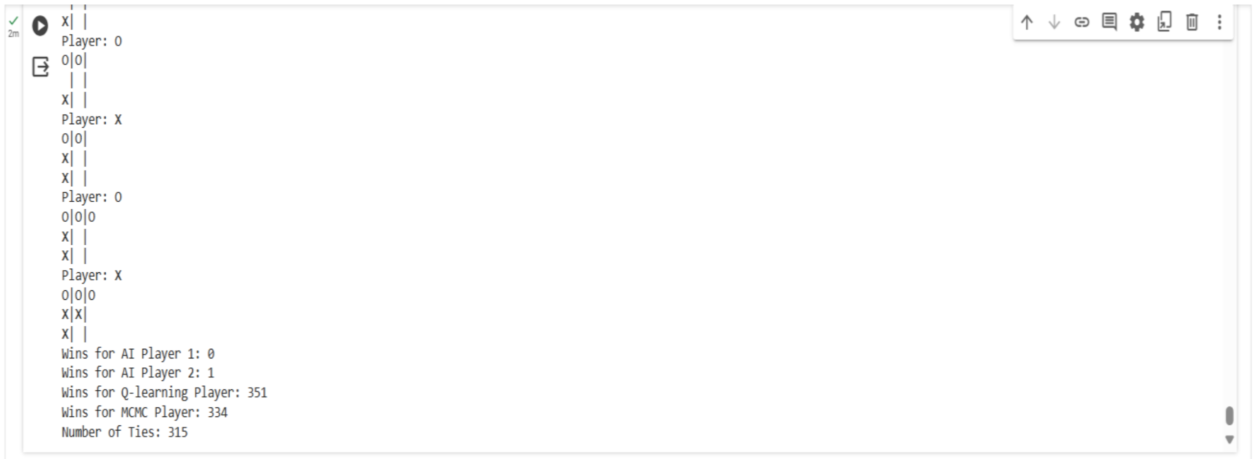
To evaluate the performance of the trained AI algorithms in playing Tic-Tac-Toe, the following aspects can be considered:

1. Win Rate: Measure the percentage of games won by the AI algorithms against opponents, such as other AI players or human players. A higher win rate indicates a more successful and effective strategy.
2. Loss Rate: Similarly, assess the percentage of games lost by the AI algorithms. A lower loss rate indicates a stronger playing ability.
3. Draw Rate: Determine the percentage of games that result in a draw. A balanced draw rate suggests that the AI algorithms are capable of playing competitively and reaching stalemates when facing opponents of similar skill levels.
4. Decision-Making Speed: Measure the time taken by the AI algorithms to make moves and decisions during gameplay. Faster decision-making can be an advantage, indicating efficient and effective strategies.
5. Adaptability: Test the AI algorithms against different opponents or varying difficulty levels. Assess their ability to adjust their strategies and make optimal decisions in response to different playing styles or changing game conditions.
6. Exploration vs. Exploitation: Analyze the balance between exploration and exploitation in the algorithms' decision-making. Evaluate whether they strike a good balance between trying out new moves (exploration) and leveraging their learned strategies (exploitation).
7. Robustness: Assess the algorithms' performance under various scenarios, such as different starting configurations or non-optimal moves by opponents. Evaluate their ability to recover from disadvantageous positions and make informed decisions.
8. Human Comparison: Compare the performance of the AI algorithms against human players. This evaluation can provide insights into the algorithms' strengths and weaknesses and their ability to mimic human-like gameplay.
9. Generalization: Test the algorithms' ability to generalize their learned strategies to unseen or unfamiliar game situations. Evaluate their performance when presented with new board configurations or variations of the game.
10. Iterative Improvement: Monitor the progress of the algorithms over multiple training iterations. Evaluate whether subsequent iterations result in improved performance, indicating successful learning and refinement.

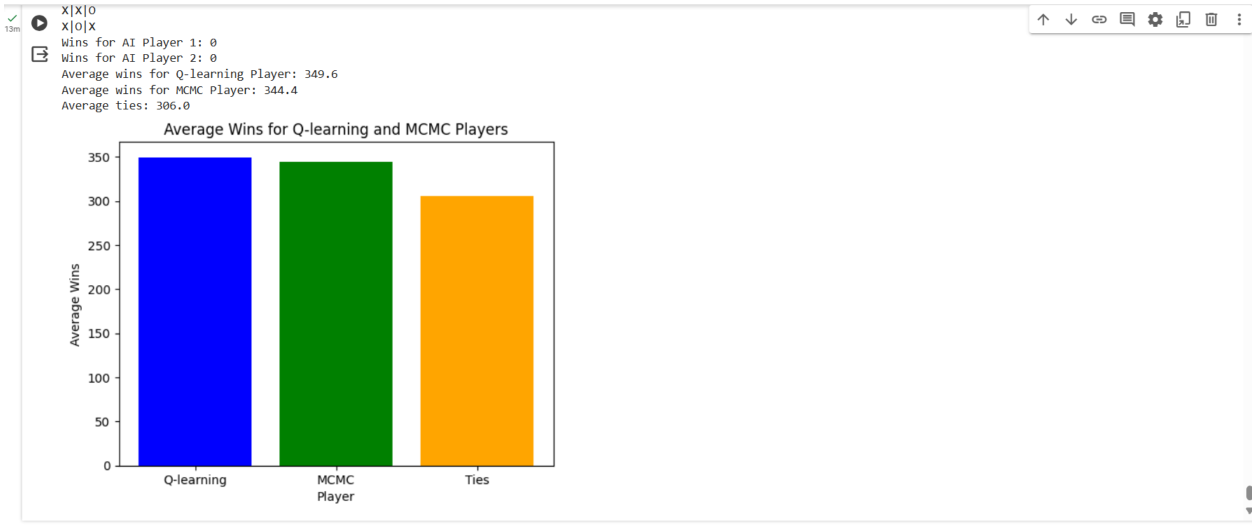
By evaluating the AI algorithms based on these criteria, we can gain a comprehensive understanding of their playing abilities, strengths, and areas for improvement in playing Tic-

8.Results

The pictures below show the results of our project.
The 1st picture shows the output for thousand (1000) simulations.
The 2nd picture shows the output for 5 runs. Each runs have 1000 simulations. The bar shows the number of wins for q-learning player, for MCMC player and the number of ties between them,



```
X| |
Player: O
O|O|
 | |
X| |
Player: X
O|O|
X| |
X| |
Player: O
O|O|O
X| |
X| |
Player: X
O|O|O
X|X|
X| |
Wins for AI Player 1: 0
Wins for AI Player 2: 1
Wins for Q-learning Player: 351
Wins for MCMC Player: 334
Number of Ties: 315
```



```
X|X|O
X|O|X
Wins for AI Player 1: 0
Wins for AI Player 2: 0
Average wins for Q-learning Player: 349.6
Average wins for MCMC Player: 344.4
Average ties: 306.0
```

9. CONCLUSION

1. Decision-Making Process:
   - Q-Learning: Based on learned Q-values.
   - MCMC: Estimates move probabilities through simulations.

2. Exploration vs. Exploitation:
   - Q-Learning: Balances exploration and exploitation.
   - MCMC: Focuses on exploitation without explicit exploration.

3. Performance Robustness:
   - Q-Learning: Adapts strategies based on learned experiences.
   - MCMC: Estimates move probabilities for robust performance.

4. Adaptability to Opponents:
   - Q-Learning: Effective against predictable opponents.
   - MCMC: Adaptable across different playing styles.

Ultimately, the choice between Q-Learning and MCMC should be based on the specific characteristics of the problem at hand, considering factors such as problem complexity, computational resources, and the balance between exploration and exploitation**.**

11.REFERENCES

1.https://github.com/arramos84/reinforcement-learning/blob/master/reinforcement/qlearningAgents.py

2.https://github.com/sgalal/cs61a/blob/master/Labs/lab08/classes.py

3. https://cs10.org/bjc-r/cur/programming/python/object-oriented-programming-joshhug/ttt.html?topic=berkeley_bjc%2Fpython%2Fbesides-blocks-oop-joshhug-edition.topic&course&novideo&noreading&noassignment