

Foundations of Artificial Intelligence

Dots-and-Boxes

Group 9

Manish Kumar, Jessica Kunkel

AI 801 (Foundations of Artificial Intelligence)

[Spring 2025]

**Document Control**

Work carried out by:

| Full Name | Email Address | Exhaustive list of Tasks |
| --- | --- | --- |
| Jessica Kunkel | jkk5521@psu.edu | * Dots and boxes gameplay environment * Pygame implementation for visualization * Reinforcement learning agent * Integration of MCTS and human gameplay code, including bug fixes and revisions for compatibility * Maintaining the source codebase * Project report and presentation |
| Manish Kumar | mkk6362@psu.edu | * Monte Carlo Tree Search agent * Human gameplay * Project report and presentation |

Revision Sheet

| Date | Revision Description |
| --- | --- |
| 05/01/2025 | Initial release |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

TABLE OF CONTENTS

[1 Introduction 4](#_Toc1467953574)

[2 Problem Statement 4](#_Toc1144614370)

[3 Challenges 4](#_Toc1591209447)

[3.1 RELATED WORKS 5](#_Toc193748449)

[3.2 IMPORTANCE AND IMPACTS 6](#_Toc1747816409)

[3.2.1 Importance and Impacts 6](#_Toc491771860)

[4 Data Collection and Preprocessing 7](#_Toc1094757196)

[5 Methodology 9](#_Toc128321157)

[6 Results and Interpretation 9](#_Toc842002399)

[Model Performance Overview 9](#_Toc2140509933)

[Quantitative Results 9](#_Toc253701994)

[Interpretations 9](#_Toc76815035)

[Summary 9](#_Toc977081916)

[7 Discussion of Results 9](#_Toc92922956)

[8 Your Feedback 9](#_Toc2047052520)

[9 References 9](#_Toc165985092)

# Introduction

Dots-and-Boxes is a classic pen-and-paper game in which two players take turns drawing lines between dots in a grid, aiming to maximize the number of boxes they create while minimizing the opponent’s boxes. Dots-and-Boxes is a perfect information, deterministic game in which the state of the game board is fully observable by all players and there is no randomness affecting future states. Although its rules are simple, the Dots-and-Boxes game has a high level of complexity due to the size of its search space (a 5x5 grid composed of a total of 60 potential moves, with many potential states), posing a challenge for AI.

This project explores how two types of Artificial Intelligence agents can learn and perform effectively in a Dots-and-Boxes game environment. One agent is based on **Monte Carlo Tree Search (MCTS)** and the other uses **Reinforcement Learning (RL)** to compete against each other and human players in Dots-and-Boxes. The Dots-and-Boxes game provides a contained yet non-trivial setting for testing decision-making under uncertainty, reward-based learning, and adversarial strategies.

This project aims to showcase how traditional AI algorithms can be effectively applied to interactive, real-time game environments by simulating human-like reasoning and long-term planning through game-playing search and reinforcement learning techniques.

# Problem Statement

This project involves developing two AI agents for the classic strategy game Dots and Boxes, focusing on implementing and comparing two artificial intelligence techniques: **Monte Carlo Tree Search (MCTS)** and **Reinforcement Learning (RL)**. The objective is to explore how these methods perform in real-time strategic decision-making, particularly in adversarial game settings. The MCTS agent uses a tree-based simulation approach to evaluate game states, while the RL agent decides which actions to take based on learning through self-play or training against an MCTS player, utilizing its learning from simulated games and formation of a Q-network to determine optimal moves without a structured search tree.

This dual-approach framework allows us to assess the strengths and limitations of each method and demonstrate their effectiveness in a competitive environment against each other.

# Challenges

One challenge of this approach is the process of building two AI agents using different methods. The MCTS agent uses a kind of “look-ahead” method to plan its moves, utilizing a search tree, while the Reinforcement learning agent learns by playing many games and learning from the rewards it gains through gameplay. There is a challenge in programming both agents so that they are compatible with and work well within the Dots-and-Boxes game environment.

Another challenge lies in handling the size of the potential action space of the Dots-and-Boxes board. There are a total of 60 moves in a game using a 5x5 game board, and many more combinations of board states. This makes it more difficult and time-consuming for the MCTS AI agent to search deeply, and for the RL agent to learn which combinations of moves are the most strategic. Making each AI agent performant in such a large search space poses a challenge.

There is a significant challenge in ensuring the reinforcement learning agent learns smart strategies. The RL agent learns from trial and error, and the structure of its thinking and strategy formation is not easily visualized. The most straightforward way to test its performance is through gameplay after training, both of which are time-consuming. It is difficult to determine how many training episodes are sufficient for performance, and to determine which factors within the RL model need to be adjusted for increased performance.

Since MCTS and RL work differently, comparing their performance poses a challenge. To compare the two methods fairly, it is necessary to set equal tests, rules, and board setups to judge which model performs better in the Dots-and-Boxes game environment.

In terms of game environment setup, there are practical challenges such as ensuring the board state is tracked accurately, and neither player can make illegal moves. Maintaining state between each player and tracking which moves are available are essential for gameplay. One player cannot choose to overwrite the move of another, so errors in tracking the board state can lead to invalid gameplay. The Dots-and-Boxes game environment also needs to be able to run with both AI agents taking turns without glitches or delays. If turn tracking is flawed, one AI crashes, or if game rules are not enforced, the match is invalid.

Since part of the project aims to allow humans to view the AI agents’ gameplay, it is important to have a clean and clear visual of the board. To test many games quickly, it is also important for performance to allow the visualization of gameplay to be turned off. Adding the ability to visualize or not visualize the game environment and making sure the game board is easy to follow as the program runs in real time is an additional technical task. The implementation of the game visualization also led to significant debugging efforts to ensure that the game environment was unaffected by the MCTS simulations for each turn.

Allowing the Dots-and-Boxes game environment and the Pygame visualization to allow human gameplay adds an additional challenge. Not only does the environment need to be robust enough to handle mouse button clicks, but it needs to be able to determine which clicks are valid and where the human intended to make each move. In addition, there is a challenge posed against AI, as gameplay against a human player can be less predictable. It is challenging for the AI models to be able to handle many kinds of strategies and still perform well.

## RELATED WORKS

Several AI research studies have explored strategic board games using different algorithms, particularly **Monte Carlo Tree Search (MCTS)** and **Reinforcement Learning (RL)**. These methods have been widely applied in popular games such as Go, Chess, and Tic-Tac-Toe, which provide well-defined environments and large communities of human and AI players for benchmarking.

**Monte Carlo Tree Search (MCTS)** was made famous by its use in AlphaGo, where it played a key role in helping the AI outperform world-class human players. MCTS is known for its effectiveness in situations with many possible moves and incomplete information, which makes it suitable for games like Dots and Boxes. Most existing work using MCTS has focused on games with fixed, well-studied strategies. In contrast, our project applies MCTS to a relatively under-explored game (Dots and Boxes), highlighting its adaptability to a new domain with dynamic game states and different types of strategies (aggressive vs defensive play).

**Reinforcement Learning (RL)** has also seen significant use in training agents for board games. Recent studies have applied deep reinforcement learning in more complex environments, often involving neural networks for learning value functions or policies. However, these methods typically require large datasets or millions of simulated episodes. Our project uses a simpler, more lightweight version of RL, tailored for faster learning and decision-making in real-time, rather than relying on deep learning architectures.

Unlike most previous work, which focuses on improving one method or applying it to a well-known game, our project directly compares two different AI approaches (**MCTS vs. RL**) to the same game. This side-by-side evaluation in Dots and Boxes helps us better understand the strengths and limitations of each approach in real-time competitive settings, especially against human players.

Additionally, there is limited published work specifically targeting Dots and Boxes with AI strategies. While some hobbyists and academic blogs have discussed basic AI players for the game, these often rely on simple rule-based systems or random moves. Our project offers a more structured and comparative analysis using formal AI techniques and aims to contribute original insights into how these methods can be used in adversarial decision-making.

Similar projects include board games like Tic-Tac-Toe, Chess AI, and games that use Reinforcement Learning for strategy.

## IMPORTANCE AND IMPACTS

This project explores the use of **Monte Carlo Tree Search (MCTS)** and **Reinforcement Learning (RL)** to develop intelligent agents for the strategic game Dots and Boxes. The core objective is to evaluate how these two well-known AI techniques perform in adversarial, turn-based decision-making. While the game itself is simple, it provides a perfect sandbox for studying complex AI behavior, real-time learning, and competitive strategy formulation.

The Dots-and-Boxes AI project gives valuable insights into how AI agents can learn or plan strategies in real time. By comparing MCTS and RL models, we highlight the strengths and weaknesses of planning versus learning in a competitive game environment.

The project also allows bridging of theory and the implementation of artificial intelligence. Concepts often taught in AI courses, such as tree search for gameplay and reinforcement learning, are applied in this project in a real, custom-designed game environment. This hands-on approach helps to bridge the gap between classroom theory and actual AI system implementation.

The Dots-and-Boxes AI project also serves as a strong teaching tool for students and beginners in AI and game theory. The project affords students the opportunity to work with popular modern AI tools such as OpenAI Gym and PyTorch. Students are also able to design, train, and test intelligent agents in a controlled environment.

#### Potential Impacts

The potential impacts of the AI gameplay project include scientific, social, educational, and industry impacts. By applying and comparing AI strategies in a new or underexplored domain such as the Dots-and-Boxes game, this project contributes to the broader field of AI research in strategic games and real-time decision-making.

There is a social and educational impact in that the game is simple and accessible, making it ideal for demonstrations and educational use. The project can be used to teach students how to develop an intelligent agent using modern tools. It can also be used to teach students how AI systems think, adapt, and compete, making AI concepts easier to understand through practical contextualization.

Although this project is based on the Dots-and-Boxes game, there is still significant business and industry relevance. The problem of turn-based decision-making is common in many real-world applications like robotics, supply chain scheduling, autonomous vehicles, and cybersecurity. The principles demonstrated through gameplay in the Dots-and-Boxes environment can be extended to these domains, affording students the opportunity to gain experience in tools that may be applied in the software industry.

Finally, since the AI agents can play against human players, this project also touches on how humans and machines interact in virtual environments. This adds to the potential for future research in explainable AI and adaptive behavior in human-facing systems.

# Data Collection and Preprocessing

### Data Collection

This project does not use an external dataset. Instead, data is generated through self-play by the AI agents during training and evaluation. The two AI agents—one using Monte Carlo Tree Search (MCTS) and the other using Reinforcement Learning (RL)—play multiple games against each other to collect gameplay sequences.

Each game state includes:

* **Board configuration**: A matrix indicating which edges have been drawn and which boxes have been claimed.
* **Player turn**: An indicator of which AI is currently playing.
* **Moves taken**: The selected move (line between dots).
* **Reward signals (for RL)**: Rewards assigned after completing boxes or at the end of the game (win/loss).
  + This generated data is directly relevant to our research question, as it simulates realistic gameplay and provides training material for the RL agent to improve its policy, and for the MCTS to simulate and compare move outcomes.

#### Exploratory Data Analysis

The features generated from the self-play data include:

* **Move index** (categorical)
* **Board state snapshot** (2D matrix, binary values)
* **Box ownership** (categorical: AI1, AI2, none)
* **Game outcome** (categorical: win, lose, draw)
* Reward (continuous, based on score or outcome)

#### Key Data Observations

* The board is always initialized empty and fills gradually.
* Rewards are sparse—only at box completion or endgame.
* The data is balanced in terms of turns, as agents take alternate moves.

### Data Preprocessing

* Missing Values: No missing values exist since all data is generated by rule-based simulation.
* Type Conversion: Game board matrices are converted to flattened arrays for some ML/RL frameworks.
* Standardization: Rewards are normalized between -1 (loss) and +1 (win) for training stability in RL.
* Inconsistencies: Checked that move indices and board updates follow valid transitions (no illegal moves).
* Outliers: Not applicable, as game outcomes and moves are bound and rule-based.

#### Interpretation

The collected gameplay data shows clear trends:

* The RL agent initially explores random moves but improves as it learns from outcomes.
* The MCTS agent tends to avoid risky moves and prefers conservative strategies.
* After enough episodes, both agents converge toward more strategic play styles, seen in increasing win rates and smarter board control.

# Methodology

Our project implements and compares two AI approaches to solve the strategic board game Dots and Boxes: Monte Carlo Tree Search (MCTS) and Reinforcement Learning (RL). Both methods were selected for their success in decision-making tasks and their ability to function within game-playing spaces, without human-designed rules.

## Monte Carlo Tree Search (MCTS)

The Monte Carlo Tree Search agent builds a search tree by simulating random gameplay (rollouts) from the current state, selecting the best moves based on statistical outcomes and a heuristic policy to choose more conservative moves (those that do not give away points to the opponent). This technique was chosen because MCTS is effective in games with large state spaces. MCTS has been proven as a valuable technique in systems like AlphaGo, which utilizes MCTS with deep neural networks to evaluate potential game positions. MCTS does not require prior learning and adapts in real time.

The MCTS agent performs multiple simulations from the current game state to estimate the best action. Each simulation involves:

1. **Selection**: Traverses the tree using UCB1 (Upper Confidence Bound 1) to select promising nodes.
2. **Expansion**: Adds a new child node if unexplored valid actions exist.
3. **Simulation**: Plays out the game using a heuristic policy to estimate the outcome.
4. **Backpropagation**: Propagates the result back up the tree to update node statistics.

The action with the highest visit count after all simulations is selected.

Assumptions of using MCTS include that the game state can be fully represented and that simulation outcomes are reliable indicators of move quality. The performance depends on the number of simulations and the allowed depth of the search for each simulation. More simulations lead to better decisions but greatly increase the time cost. Allowing each simulation to search until the end of the game is detrimental to performance, and a depth limitation was implemented to improve MCTS time performance. To keep performance consistent, the MCTS model used 10 simulations with a depth limitation of 10 to determine its optimal move for each turn.

## Reinforcement Learning (RL)

The reinforcement learning agent was developed utilizing active RL principles, including epsilon-greedy action selection. During training, the RL agent interacts with the Dots-and-Boxes environment and learns from experiences by storing them in replay memory. The neural network predicts Q-values for optimal action selection. Epsilon-greedy action selection aims to balance exploration and exploitation strategies, as the epsilon value starts at 1 and decays by a factor of 0.995 to a minimum epsilon value of 1x10-4. The RL model utilizes deep Q-learning, supplemented by a convolutional neural network (CNN) to represent the Dots-and-Boxes board as a type of image. The deep Q-network (DQN) approach uses a neural network to approximate Q-values and uses replay memory to store and learn from past experiences.

The convolutional neural network is utilized to better learn spatial features of the Dots-and-Boxes board state, rather than using a fully connected DQN that represents the board as a flat vector. The CNN also helps in predicting moves rather than requiring memorization of Q-values for all states, aiming to make the RL agent more efficient in its training. Utilizing a convolutional neural network also required the use of tensors and the PyTorch library to represent the Dots-and-Boxes board.

The reinforcement learning agent was chosen for this project because RL mimics human learning through experience. It allows the agent to improve with training over many games, receiving positive rewards for making boxes, and updating its strategy over time to maximize long-term rewards.

Three flavors of training were implemented for comparison between methods: the RL agent interacts with the Dots-and-Boxes environment through self-play, play against an easier MCTS opponent that uses fewer simulations, or play against the easier MCTS opponent with a more complex reward structure. The updated reward structure included negative reinforcement for moves that would allow the opponent to make a box, positive reinforcement for moves that would lead to an extra turn, and larger positive and negative rewards for winning or losing the game. Each of these RL agents was compared to the MCTS opponent in normal gameplay.

Utilizing a reinforcement learning agent in the Dots-and-Boxes game assumes that a reward signal can guide the agent toward effective strategies without the need for a search tree. One constraint of reinforcement learning is that training requires many episodes and may be slower to learn compared to MCTS in early stages. For this reason, multiple RL agent representations were generated, using different types of training and different numbers of simulations. The initial RL training model was used to train reinforcement learning agents in 10, 100, 500, and 1,000 episodes. As seen in Table 1, the epsilon value of the RL agent plateaued after 100 episodes, so 100 was the maximum episode count used moving forward. The training model utilizing an MCTS opponent was used to train reinforcement learning agents for 10 and 100 episodes (RL Agent 2.0). The training model that utilizes both the MCTS opponent and an updated reward system was used to train RL agents for 10 and 100 episodes (RL Agent rewards). The RL agents and their respective epsilon values are detailed in Table 1 below.

Table 1 Reinforcement Agents

|  |  |  |  |
| --- | --- | --- | --- |
| Agent | Training | Number of Simulations | Epsilon value |
| RL Agent-10 | Self-play | 10 | 4.700E-02 |
| RL Agent-100 | Self-play | 100 | 9.973E-05 |
| RL Agent-500 | Self-play | 500 | 9.973E-05 |
| RL Agent-1000 | Self-play | 1000 | 9.973E-05 |
| RL Agent 2.0-10 | MCTS-play | 10 | 2.201E-01 |
| RL Agent 2.0-100 | MCTS-play | 100 | 9.973E-05 |
| RL Agent rewards-10 | MCTS-play with updated reward structure | 10 | 1.981E-01 |
| RL Agent rewards-100 | MCTS-play with updated reward structure | 100 | 9.973E-05 |

#### Model Comparison

* We compare both agents through head-to-head matches and performance metrics such as win rate, average score, and learning efficiency over time.
* By analyzing game outcomes and behavior, we evaluate which method is more suitable for different gameplay situations (e.g., short-term tactics vs long-term learning).

#### Tools and Software Used

* **Programming Language:** Python
* **Libraries:** NumPy, Pygame for gameplay visualization, PyTorch for CNN and DQN development, and OpenAI Gym for development of a custom-built simulation environment
* **Development Environment:** Jupyter Notebook and Visual Studio Code for training, evaluation, and visualization
* **Data:** No external databases were used; all game data was generated via automated gameplay.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1 Gameplay Example 1

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2 Gameplay Example 2

# Results and Interpretation

This section presents the performance outcomes of the two AI agents, Monte Carlo Tree Search (MCTS) and Reinforcement Learning (RL), and analyzes their behavior and effectiveness in the game Dots and Boxes.

The MCTS was tested against each RL agent, with varying numbers of training episodes and training methodologies as discussed in Section 5.2. Each game was repeated for 50 iterations (Table 2), and the MCTS and RL agents were tested as both the first player (making the first move, Table 3) and the second player (Table 4).

Table 2 Overall MCTS vs RL Results

| Agent | Games Played | Games Won | % Won |
| --- | --- | --- | --- |
| MCTS | 800 | 369 | 46.13 |
| RL Agent-10 | 100 | 54 | 54.00 |
| RL Agent-100 | 100 | 76 | 76.00 |
| RL Agent-500 | 100 | 55 | 55.00 |
| RL Agent-1000 | 100 | 47 | 47.00 |
| RL Agent 2.0-10 | 100 | 40 | 40.00 |
| RL Agent 2.0-100 | 100 | 58 | 58.00 |
| RL Agent rewards-10 | 100 | 47 | 47.00 |
| RL Agent rewards-100 | 100 | 54 | 54.00 |

Table 3 Results, Agent as Player 1

| Agent | Games Played | Games Won | % Won |
| --- | --- | --- | --- |
| MCTS | 400 | 177 | 44.25 |
| RL Agent-10 | 50 | 28 | 56.00 |
| RL Agent-100 | 50 | 39 | 78.00 |
| RL Agent-500 | 50 | 30 | 60.00 |
| RL Agent-1000 | 50 | 22 | 44.00 |
| RL Agent 2.0-10 | 50 | 20 | 40.00 |
| RL Agent 2.0-100 | 50 | 26 | 52.00 |
| RL Agent rewards-10 | 50 | 18 | 36.00 |
| RL Agent rewards-100 | 50 | 25 | 50.00 |

Table 4 Results, Agent as Player 2

| Agent | Games Played | Games Won | % Won |
| --- | --- | --- | --- |
| MCTS | 400 | 192 | 48.00 |
| RL Agent-10 | 50 | 26 | 52.00 |
| RL Agent-100 | 50 | 37 | 74.00 |
| RL Agent-500 | 50 | 25 | 50.00 |
| RL Agent-1000 | 50 | 25 | 50.00 |
| RL Agent 2.0-10 | 50 | 20 | 40.00 |
| RL Agent 2.0-100 | 50 | 32 | 64.00 |
| RL Agent rewards-10 | 50 | 29 | 58.00 |
| RL Agent rewards-100 | 50 | 29 | 58.00 |

Overall, the performance of the MCTS model and the RL models was very similar. A two-proportion z-test with Holm-Bonferroni correction was used to assess statistical differences in win rates of the MCTS agent and each RL agent. Before correction, only the RL Agent trained through self-play for 100 episodes (p = 2.10 E-6), showed a significant difference (p > 0.05). After applying the Holm-Bonferroni correction for 8 comparisons, it held that there was a statistically significant difference between the MCTS win rate and the RL Agent-100 win rate (adjusted threshold = 0.00625). Differences for all other agents were non-significant (p > adjusted thresholds).

In addition, a two-proportion z-test was also used to compare the performance between playing as Player 1 versus Player 2 for each agent. Before correction, only the RL Agent trained against an easier MCTS with the updated reward structure for 10 episodes showed a significant difference (p = 0.027); there was no statistically significant difference between the performance of the other AI agents as player 1 and player 2 (p > 0.05). After applying the Holm-Bonferroni correction for 9 tests, it was found that there was no statistically significant difference between the Player 1 and Player 2 performances of RL Agent Rewards-10 (adjusted threshold = 0.00556). In summary, no differences between Player 1 and Player 2 performances were statistically significant.

# Discussion of Results

The goal of our project was to develop a Dots-and-Boxes game environment, develop AI players using Monte Carlo Tree Search and Reinforcement Learning, and compare the performance of the two techniques in a competitive, turn-based game setting. The results demonstrate that while the two types of AI agents had largely similar performance, each approach has its strengths depending on the nature of the problem and the goals of the agent.

#### Usefulness in Solving the Problem

Both agents successfully learned to play Dots and Boxes competitively. MCTS showed strong performance immediately due to its ability to simulate many possible moves, while the RL agent improved steadily through repeated gameplay. In general, it can be seen in Table 2 the nominal performance of the RL agents improved with an increased number of training episodes. This is more obvious when comparing RL Agent 2.0-10 to RL Agent 2.0-100 and comparing RL Agent Rewards-10 to RL Agent Rewards-100.

While there was a statistically significant difference in the performance of RL Agent-100 compared to the performance of the MCTS agent after applying Holm-Bonferroni correction, we believe this is likely an outlier in performance, as it is inconsistent with the results of the RL agents that utilized the same training platform with an increased number of training episodes.

It should also be noted that when a human plays against the MCTS model or the RL Agents (a dummy comparison playing against RL Agent-1000 and RL Agent Rewards-100 was performed briefly), it is extremely easy to beat the AI agents at the Dots-and-Boxes game. While each agent is a valid proof of concept, there is definite room for improvement to meet the level of the state of the art in AI gameplaying agents.

#### Practical Implications

The project highlights how different AI methods can be applied to the same problem and produce effective, though differently optimized, strategies. The overall effectiveness of each AI agent in their gameplay leaves room for improvement, as other RL agents developed for Dots-and-Boxes gameplay are likely more performant. However, there is educational value in the project, as it acts as a tool for teaching AI concepts like decision trees, searching, learning from rewards, and turn-based planning.

There is also scientific insight in directly comparing two AI models. This project could help researchers to understand how different algorithms behave in competitive environments and highlight which strategies work well or need improvement.

The methods used in this project can also be applied, with limitations, to real-life decision-making scenarios such as robotics, logistics, or cybersecurity, where systems must make intelligent decisions against adversaries or under uncertainty. The AI models themselves would need significant revision, but the application of MCTS and reinforcement learning using convolutional networks to those sectors could also highlight areas for potential improvement of the models.

#### Limitations

There are a few limitations in the Dots-and-Boxes AI gameplay project. First, Dots-and-Boxes is a relatively simple game, so results would likely not directly translate into more complex environments, such as more complex games or real-world decision-making. In addition, the RL agent required many gameplay episodes to learn effectively, which may not be feasible in time-sensitive applications. The MCTS agent would also likely not translate into time-sensitive applications, as the execution of 10 simulations per turn before the limitation of the depth of its search was prohibitive to gameplay.

#### Future Work

Future versions of this project could include more thorough testing of the AI agents against human players to evaluate adaptability and real-world performance. Human vs AI testing would also highlight any issues in the AI models, potentially requiring changes to the MCTS and RL players to improve their performance against real players.

In addition, hybrid models that combine MCTS and RL could lead to stronger agents that benefit from both planning and learning. This type of agent would more closely reflect the state of the art for AI game-playing models, such as the QDab model, which utilizes an artificial neural network and Monte Carlo Tree Search to learn the Dots-and-Boxes game effectively [5].

Lastly, applying MCTS and RL techniques to more complex games or real-world problems would test their generalizability and potentially highlight areas for improvement that were not immediately obvious in the Dots-and-Boxes environment.

#### Model Limitations and Improvements

Each AI model generated for this project has limitations and room for improvement. The Reinforcement Learning model’s learning was based on basic reward signals. Using more advanced strategies, such as policy gradients, could improve learning speed and strategy. The Monte Carlo Tree Search agent was limited by the allowed number of simulations and depth of search per move. Future versions could include more effective heuristics to guide simulations and reduce computation time for each move.

# Your Feedback

Working on this project provided a valuable opportunity to apply AI concepts, such as Monte Carlo Tree Search and Reinforcement Learning, in a practical, hands-on environment. It helped solidify our understanding of strategic decision-making, simulation-based planning, and learning from rewards.

We especially appreciated the flexibility to choose and compare different AI techniques. This encouraged creative problem-solving and made the project more engaging. It also offered great insight into the strengths and limitations of various models in a competitive setting.

#### Positive Aspects

Positive aspects of the course project include its promotion of independent research and experimentation in the AI space. The project also effectively combined AI theory and the application of relevant concepts. Lastly, comparing two distinct AI methods added depth to the learning experience afforded by the project.

#### Suggestions for Improvement

First, providing a basic AI game engine or starter code could help students focus more on implementing strategies rather than the setup of the OpenAI Gym environment. There was a significant amount of time and learning dedicated to setting up the gameplay environment, which limited the time available to develop the Monte Carlo Tree Search and Reinforcement Learning models.

Second, providing more practical instruction on creating a reinforcement learning model using modern tools like PyTorch would also have helped students understand the methodology of building a Reinforcement Learning model. The act of developing the RL model was time-consuming and difficult to understand, especially without clear guidance from the course materials. A cohesive introduction to AI tools like PyTorch would have made the project progress more smoothly.

Lastly, a built-in performance evaluation template (e.g., win-rate tracker, logging system) would make it easier to analyze results. As it stands, the project was extremely open-ended in its expectations for the AI models. While this afforded students the freedom to pursue whichever methods they felt fit, some guidance on recommended metrics would have made the project expectations clearer.

Overall, the project was intellectually rewarding and an excellent way to learn how AI techniques can be applied in games and real-world decision-making.

# References

1. *Dots and Boxes* – [Wikipedia](https://en.wikipedia.org/wiki/Dots_and_boxes)
2. *Game Rules* – [UCLA Math Department](https://www.math.ucla.edu/~tom/Games/dots&boxes.html)
3. AlphaGo & MCTS Research:
   1. Silver D, Huang A, Maddison CJ, et al. “Mastering the game of Go with deep neural networks and tree search.” Nature 529: 484-489 (2016). <https://doi.org/10.1038/nature16961>
4. RL Tutorials & OpenAI Gym documentation – <https://gymnasium.farama.org>
5. Zhuang Y, Li S, Peters TV, Zhuang C. “Improving Monte-Carlo Tree Search for Dots-and-Boxes with a novel board representation and artificial neural networks.” *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, Tainan, Taiwan: 314-321 (2015). <https://doi.org/10.1109/CIG.2015.7317912>.

[GitHub - jkunkel299/ai801-project: Dots and Boxes game with AI players](https://github.com/jkunkel299/ai801-project)