Introduction to Artificial Intelligence project

Backgammon

Project members
Name
Adan Akariya
Elias Shubeita
Yousef Abu Gosh

Abstract

In this project, we thoroughly explore the game of backgammon, aiming to develop highly effective AI agents. Specifically, we will create agents based on the Expectiminimax and Reinforcement Learning (RL) algorithms. These algorithms will utilize heuristics that we design based on our experience and understanding of the game. By comparing the performance of these agents, we will analyze their effectiveness and draw meaningful conclusions about the application of these methods in stochastic games.

Contents

1. INTRODUCTION:

The Backgammon game presents a fascinating challenge for artificial intelligence due to its unique blend of skill and chance. As a two-player game, it involves strategic decision-making based on board states and dice rolls, creating high uncertainty and a vast state space. This complexity makes Backgammon an excellent candidate for testing and refining AI techniques. Our project seeks to address this by using Temporal Difference (TD) Learning, a reinforcement learning method, to create a TD-Gammon model. This model learns to play through self-play and navigates the numerous move possibilities with the aid of a neural network. We also incorporated an expectiminimax agent to anticipate future moves and optimize decision-making for the best results.

1.1 Rules of the game:

In this project, we used the rules of Backgammon as outlined on BGGM [1], which include moving checkers based on dice rolls, hitting opponents' checkers, and bearing off checkers from the board. Understanding these rules is crucial for developing an effective strategy and modeling the game accurately for artificial intelligence purposes.

1.2 The problem:

The core challenge of this project is to teach an agent to master Backgammon from scratch, with no prior knowledge beyond the game's rules. Additionally, the complexity is increased by the vast number of potential moves, making it difficult to develop a strategy that consistently ensures victory.

1.3 Proposed Approach

To address this problem, we have implemented the TDAgent, which utilizes a neural network model to evaluate game states and make decisions. This model is trained to predict the value of a given state, allowing the agent to make informed decisions based on its learned experience.

For the expectiminimax approach, we employed the lookahead method to evaluate the board. This involves examining possible moves within the immediate future of the current position. Due to the large branching factor, this evaluation is typically restricted to a small number of moves (1-3).

In the next section, we review prior research on backgammon approaches. Section 3 details our model, including the assumptions and success criteria. Section 4 presents the results, followed by Section 5, which revisits the problem, model, and findings, and offers a critique of the work. Additionally, throughout the development process, we utilized ChatGPT, a Large Language Model, to assist in generating and refining code, analyzing performance metrics, and ensuring clarity in documentation. This integration of ChatGPT significantly enhanced the accuracy and efficiency of our work.

2. PREVIOUS WORK:

The reinforcement learning algorithm employed in this project is inspired by Gerald Tesauro's work on TD-Gammon [2], developed in the 1990s. TD-Gammon combined Temporal Difference learning with nonlinear function approximation to excel at Backgammon. Tesauro's approach involved training a neural network using backpropagation, with TD errors guiding the learning process.

We also reviewed the work of students from last year's course [3], whose TD agent did not achieve satisfactory results after training. To address this, we developed a new model for TD-Gammon with the goal of improving performance.

In addition, we investigated the Expectiminimax algorithm, which is commonly used in adversarial settings. We chose to implement this algorithm with enhanced heuristics to better manage the complexities of Backgammon and improve its performance.

As a foundation, we utilized the existing Backgammon code from Awni Hannun[4], which provided a Python implementation of the game's rules. Initially, we addressed some bugs we encountered in the code. Following these corrections, we expanded upon the framework to implement various agents and models for our project.

We will now discuss the algorithms used to solve the problem: TD Learning and the Expectiminimax algorithm.

1.Temporal Difference Learning

TD learning is a reinforcement learning technique where an agent learns to predict the value of a state by updating its predictions based on future states.The algorithm's success was based on the ability to train a neural network that could evaluate board positions and guide the agent's actions without relying on handcrafted rules.

2. Expectiminimax Algorithm

Expectiminimax is a generalization of the minimax algorithm, commonly used in games of chance, where randomness plays a key role (e.g., dice rolls). Expectiminimax incorporates probabilistic outcomes by including chance nodes alongside the traditional decision nodes for the agent and its opponent.

Common Assumptions with previous work
TD-Gammon assumes that self-play is an effective method for training agents. The idea is that by playing against itself, the agent can develop a robust strategy without

requiring external expert data. Another common assumption, which we adopted, is the way features are represented, similar to how Tesauro approached it. Earlier work also assumes that the agent can extract relevant features from the game board, such as the number of pieces on each point and whose turn it is.

Benchmark Performance

The performance of our Backgammon agents was benchmarked against several baseline agents, including a random agent, a close agent, and an eater agent, which will be described in the next section. After extensive training through self-play, we assessed how frequently our agents won against these baseline opponents. We initially evaluated the performance of the existing agents and then compared these results with the performance of the agents we developed

3. METHODOLOGY:

The first model implemented in this project is a reinforcement learning agent based on Temporal Difference (TD) learning, which uses neural networks to approximate the value function of game states. The network consists of two dense layers: 50 units with sigmoid activation in the first layer and a single output value, also with sigmoid activation, to evaluate board positions.

To input board configurations into the network, we encoded the board using Tesauro's method from TD-Gammon. This feature vector includes information on the number of pieces on each point, whose turn it is, and the pieces off the board or on the bar.

During training, the model plays multiple episodes of Backgammon against itself. Two instances of the TDAgent, both using the same underlying model, face off in each game. At every step, the current player makes a move, transitioning to a new state. After each move, the model updates its value estimate for the current state based on the value of the subsequent state, leveraging the core principle of TD learning. The reward function, typically 1 for a win and 0 for a loss, provides the feedback for learning. The learning process revolves around Temporal Difference (TD) error, which measures the difference between the predicted value of the current state and the predicted value of the next state. By minimizing this error through backpropagation, the model updates its neural network weights, refining its value predictions.

The learning is based on several key assumptions. First, it is assumed that self-play is a valid method for generating diverse training data and enabling continuous improvement. Second, it assumes the extracted game features represent the necessary information for decision-making. The model also assumes that its neural network architecture has enough capacity to effectively learn and approximate the value function without overfitting. Furthermore, TD learning is presumed suitable for updating the model, with learning rate decay helping stabilize the training process. Finally, the model relies on the assumption that the reward function provides sufficient feedback for guiding its learning.

Success in this approach is measured by several key metrics. A reduction in the loss function, calculated as the Mean Squared Error (MSE) between predicted and actual game state values, indicates improved prediction accuracy. The model's winning rate in self-play scenarios is another success indicator; an increasing rate suggests effective learning and strategy enhancement. Additionally, the convergence of TD error reflects successful learning, as a decreasing TD error shows the model is narrowing the gap between predictions and actual outcomes. Finally, stability in performance metrics like loss and accuracy, with minimal fluctuations, signals robust learning without issues such as overfitting.

Through self-play, the model gains experience by linking board positions to winning or losing outcomes. Over time, it reduces prediction errors and refines its strategy, improving decision-making and proficiency in Backgammon.

The second algorithm that we implemented was the expectiminimax. The algorithm evaluates possible moves by iterating through them and selecting the optimal option based on a predetermined search depth. At depth 1, the algorithm assesses all possible moves and identifies the one that maximizes the main player’s score, utilizing an evaluation function defined in a heuristic file. This evaluation is straightforward, focusing solely on the immediate outcomes of the player's actions.

As the depth increases, the algorithm's complexity and foresight grow. At depth 2, it not only evaluates the main player’s moves but also anticipates the opponent's optimal response to each of those moves across all potential dice values. This means that the algorithm returns the move that maximizes the main player's score while considering the best possible counter-move from the opponent. At depth 3, the algorithm further expands its analysis by evaluating the first moves for the main player, factoring in the opponent's best responses, and then examining all potential second moves for the main player. This two-move lookahead allows for a more nuanced selection of moves that can lead to favorable positions.

The algorithm operates under several key assumptions to ensure its effectiveness. First, the game state is represented as a board with player pieces, and all possible moves are deterministically derived from the current state and available dice rolls. Additionally, the game follows a strict turn-based format, with players alternating turns while the algorithm recursively evaluates future states. It assumes that both players are rational and will always play optimally, aiming to maximize their expected outcomes based on the current game state. The only source of randomness is the roll of the dice, influencing the available moves but not altering the deterministic nature of player actions. Finally, a heuristic evaluation function is employed to score the game state based on critical factors such as piece positions, the number of pieces off the board, and overall board control.

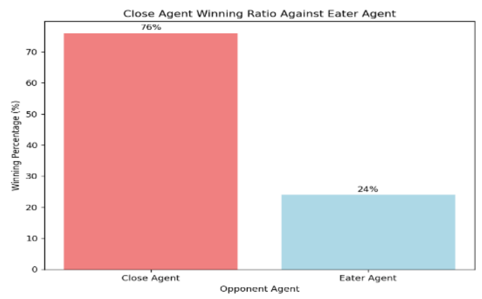
The success of the Expectiminimax algorithm relies on key criteria. The evaluation function is crucial, assigning scores based on advancing pieces toward the home board, rewarding pieces that have exited, and penalizing vulnerable pieces (blots). Its effectiveness is measured by the ability to consistently select optimal moves, quantified by win rates against baseline strategies and average score improvements. Additionally, computational efficiency is essential, as the algorithm must operate within reasonable time constraints during deeper searches to balance thoroughness and speed.

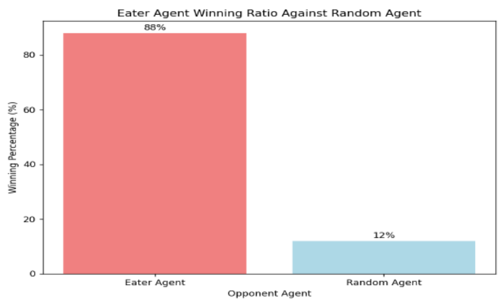
To evaluate our TD agent and Expectiminimax agent, we assessed their generalization to new opponents, specifically various agent types: Random, Close, and Eater agents. The Random Agent selects moves randomly without analyzing the game state, resulting in poor performance due to a lack of strategy. In contrast, the Close Agent focuses on creating points with two or more pieces, effectively blocking the opponent's moves and protecting its own pieces, making it the most effective of the three. Meanwhile, the Eater Agent prioritizes capturing the opponent's pieces but struggles when there are no exposed pieces, leading to inconsistent results. This evaluation highlights the strengths and weaknesses of each agent type, providing insights into how our TD and Expectiminimax agents perform against different strategies.

4. RESULTS:

In the following, we present the results of various experiments. We benchmark the models in a way of getting useful insights is by comparing new models to some other agents that are used as a baseline.

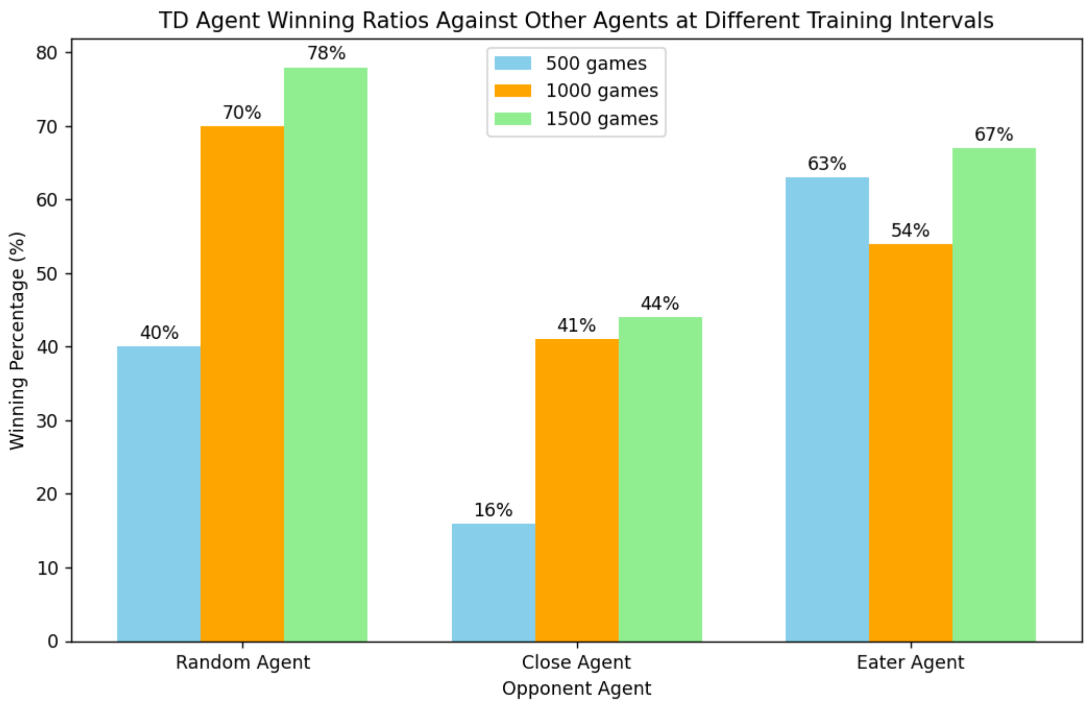
At first, we tested the Random Agent against the Eater Agent for 100 games, and then we tested the Eater Agent against the Closer Agent.





Our tests revealed that the Closer Agent outperformed the others due to its effective strategy of blocking the opponent and protecting its own pieces. By creating points with two or more pieces, it significantly slowed down the opponent's progress and avoided having its pieces captured, giving it greater flexibility in future moves. In contrast, the Random Agent performed the worst because it lacked any strategic insight, making moves without considering future turns or protecting its pieces, which led to poor positioning. The Eater Agent performed well when opportunities to capture opponent pieces arose, temporarily gaining an advantage by sending pieces to the bar. However, it struggled when no capture opportunities were available, often resorting to random moves, especially when facing the Closer Agent's solid defense.

After testing the Random Agent, Eater Agent, and Closer Agent, we then tested the TD-Gammon agent, which was trained on different numbers of games (500, 1000, and 1500 games), and observed the following results:



The TD-Gammon agent trained on 500 games showed only a basic understanding of the game. While it occasionally protected its pieces, it still made random or ineffective moves and often missed opportunities to create blocking points or capitalize on the opponent’s mistakes.

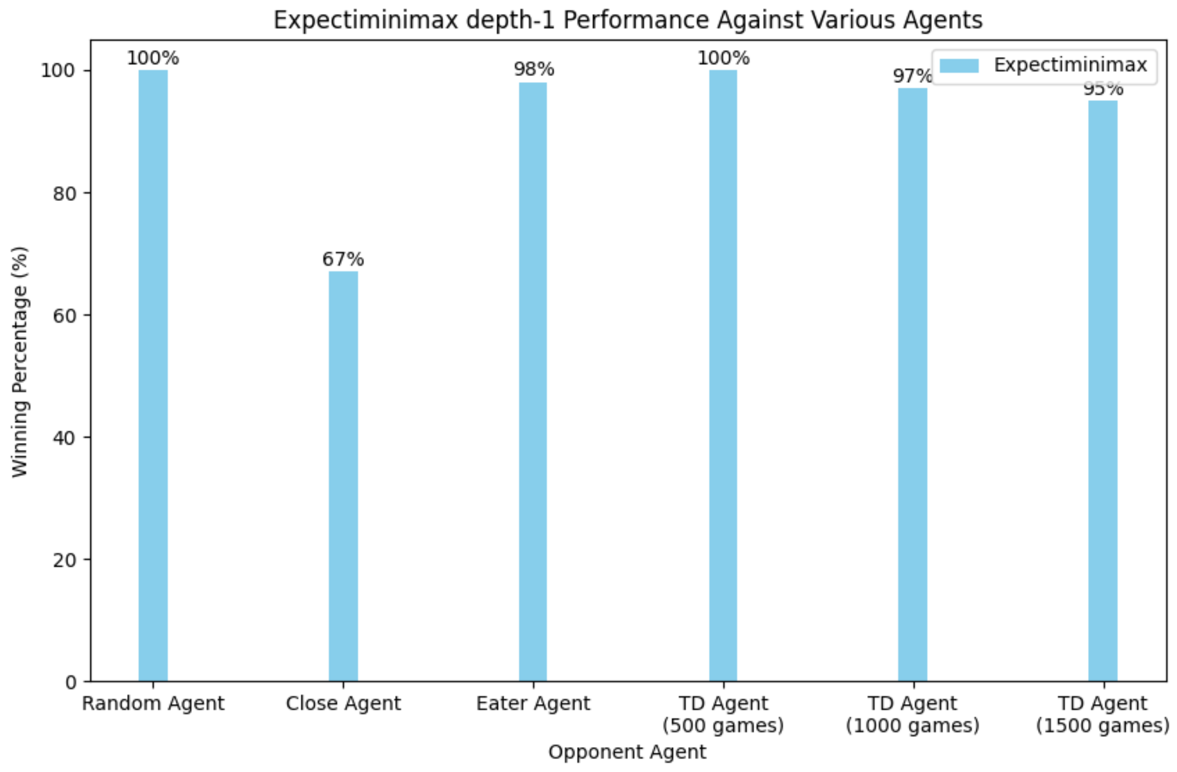
After 1000 games of training, the TD-Gammon agent improved significantly, especially against the Random and Closer agents. It developed a stronger ability to block opponents and avoid having its pieces captured, and it began to adopt more strategic moves. However, it struggled against the Eater Agent. This was because the

Eater Agent’s aggressive strategy of constantly capturing the opponent's pieces disrupted TD-Gammon’s strategic planning. While TD-Gammon performed well against agents that didn’t prioritize capturing, it found it difficult to defend against the chaotic situations created by the Eater Agent’s unpredictable attacks.

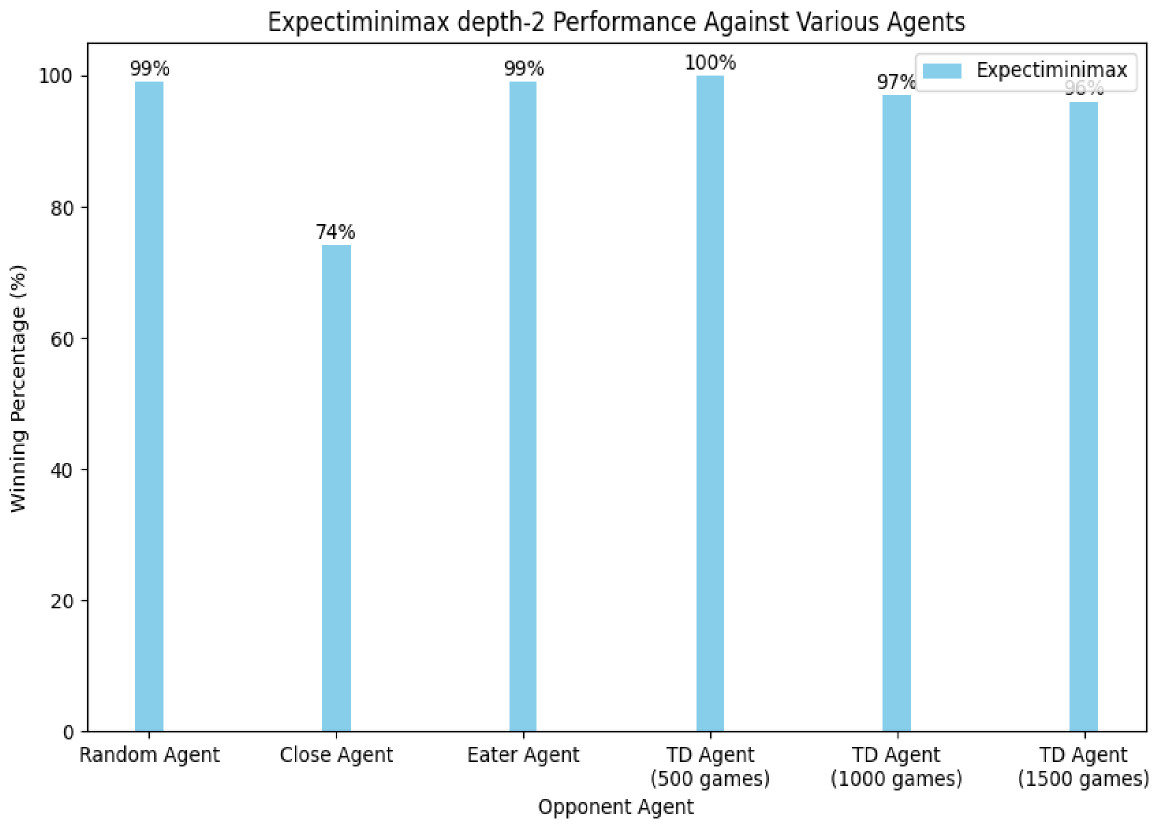
After 1500 rounds of training, TD-Gammon showed marked improvements across all opponents, including the Eater Agent. The additional training allowed it to balance defense and offense better, anticipating capture attempts and adapting to chaotic situations. However, it's essential to note that Backgammon involves a significant element of chance due to dice rolls, which introduce a wide range of probabilities. This randomness can lead to unexpected outcomes, even for highly trained agents. Strong strategies can be undermined by unlucky rolls or fortunate sequences for opponents, making the game both strategically deep and unpredictable. As a result, even well-trained agents like TD-Gammon may experience setbacks due to the inherent variability of the game.

After finishing the training for TD-Gammon, we then tested an Expectiminimax agent with a depth of one across 100 games against all the previously evaluated agents, yielding the following results:

For ExpectiMiniMax with depth

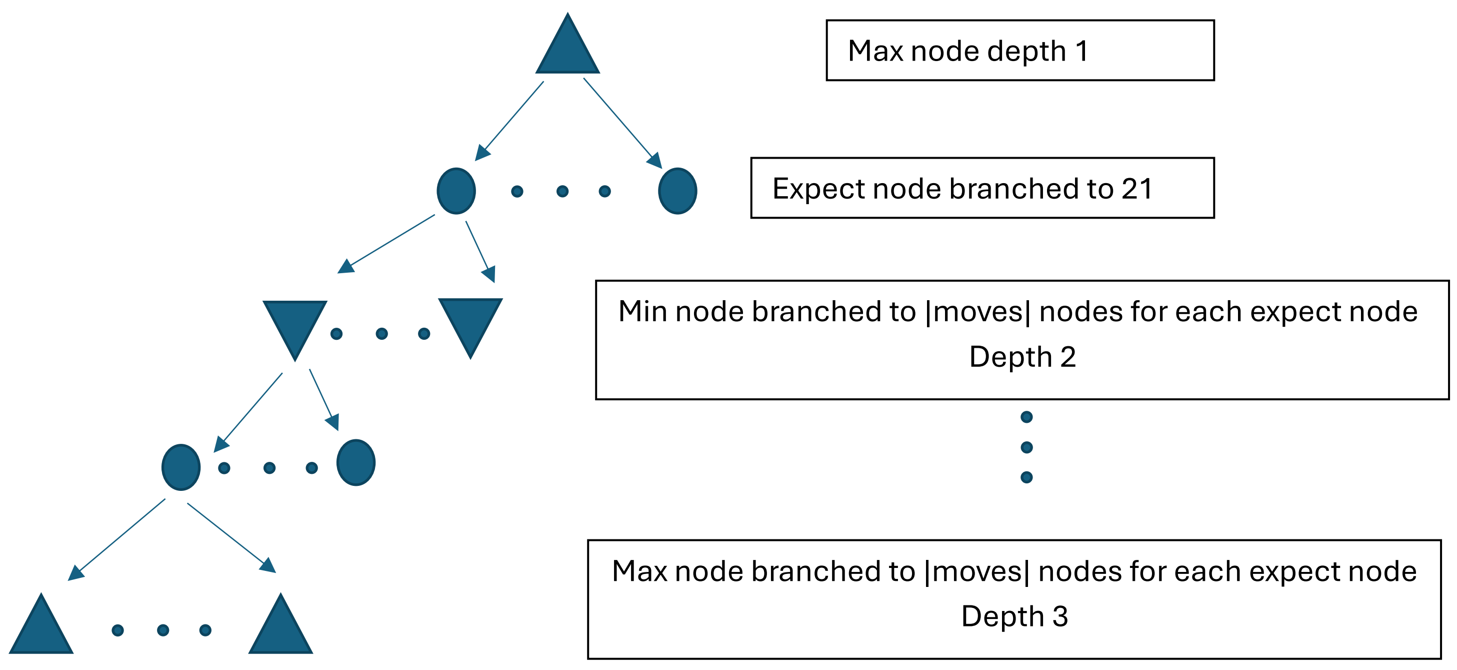


For ExpectiMiniMax depth-2



Although the small difference in performance between depth 1 and depth 2, the Expectiminimax agent proved to be highly effective, winning the majority of its matches, even at this shallow depth. Its success can be attributed to its ability to incorporate both strategic decision-making and the probabilistic nature of the game. By evaluating potential moves based on expected outcomes rather than just the best-case scenarios, the agent effectively navigated the complexities introduced by the randomness of dice rolls in Backgammon. This allowed it to make informed choices that optimized its chances of success while anticipating the moves of its opponents. Consequently, the Expectiminimax agent consistently outperformed the other agents, demonstrating a robust approach to managing both strategy and chance.

We chose to run the Expectiminimax agent at depth one and two only due to the significant increase in computational complexity. The branching factor of the game tree is quite large, with 21 possible outcomes for each dice roll combined with the number of moves available.


Max node depth 1
Expect node branched to 21
Min node branched to |moves| nodes for each expect node Depth 2
Max node branched to |moves| nodes for each expect node Depth 3

For depth one, we calculated |moves|; for depth two, it would be |moves| * 21 * |moves|, and for depth three, |moves|^3 * 21^2. This exponential growth made depth increase effect the game time exponentially for example an agent that runs a depth-2 would take approximately 200 seconds per game, compared to just 0.6 seconds at depth one. Running depth three would have been even more prohibitive, requiring around 6000 seconds per game. To address this issue, we implemented a pruning strategy that cut off the depth of a state that have too much moves, (the number of moves varies between 0-150 possible moves). We did that using hyper-parameter "pruning factor" that is for every state produce bigger number of possible moves than the pruning factor we cut at that depth and return the evaluation,

We tried deferent pruning factors (consider the pruning factor as a hyper-parameter) then we saw that a small pruning factor of 10 can enhance the time drastically, effectively cutting the time to ten times faster, with acceptable results.

From these results, we can deduce that strategic depth and adaptability are crucial for success in Backgammon. While aggressive strategies like those of the Eater Agent can disrupt opponents, a balanced approach that considers both offense and defense, as demonstrated by the Closer and Expectiminimax agents, tends to yield better results overall. The TD-Gammon agent's progressive improvement illustrates the importance of training and experience in developing effective strategies within the game's probabilistic framework.

5. SUMMARY:

In this project, we developed AI agents capable of playing Backgammon, a game that combines both strategy and chance, by implementing two advanced models: one based on Temporal Difference (TD) learning and another on the Expectiminimax algorithm. The TD agent learns by playing against itself, refining its strategy using a neural network that adapts over time. Meanwhile, the Expectiminimax agent predicts future moves by simulating potential game states, including dice roll outcomes, and optimizing its decision-making accordingly.

To evaluate the performance of these agents, we tested them against three baseline agents: the Random Agent, which plays without any strategic focus; the Closer Agent, which prioritizes blocking and defensive play; and the Eater Agent, which aggressively captures opponent pieces. Our results demonstrated that, as the TD agent was trained on more games, its performance significantly improved. It outperformed both the Random and Eater Agents and began to approach the level of the Closer Agent. The Expectiminimax agent, even at shallow search depths of 1 and 2, performed very well, consistently winning against the Random and Eater Agents, and winning the majority of games against the Closer Agent.

Throughout this work, several assumptions were made. First, we assumed that self-play would provide the diversity needed for the TD agent to develop its strategies. While this did lead to marked improvements, relying solely on self-play might limit the agent's ability to encounter and learn from advanced strategies that human players could employ. This could constrain the agent’s ability to adapt to more complex real-world playstyles.

Additionally, we assumed that the reward function used to guide the agent’s learning, based largely on win/loss outcomes, was sufficient. However, the learning process might have benefited from a more detailed reward system that offered feedback on in-game milestones, such as successfully blocking an opponent or avoiding a capture. This could have accelerated the agent’s learning by encouraging more sophisticated strategic play.

There are also alternative methods that could be explored for this problem. Monte Carlo Tree Search (MCTS) is one approach that could further optimize decision-making in the face of the game’s inherent uncertainty. Given that Backgammon’s dice rolls introduce a stochastic element, MCTS could enhance decision-making by simulating numerous potential outcomes and selecting moves based on their expected value. Implementing MCTS, either during gameplay or as part of the training process, could yield stronger agents, though this would increase computational complexity due to the large number of possible game states. This might require the use of parallel computation to efficiently handle the expanded decision tree.

It would also be worth considering hybrid models that integrate both TD learning and Expectiminimax during training. Combining these algorithms could allow the agent to leverage both self-play and deeper probabilistic search methods, potentially leading to a more well-rounded and competitive agent. Overall, while the results of this project were promising, there is ample room for further exploration, particularly through more sophisticated learning mechanisms and algorithmic combinations.

6. References:

[1]. Rules of Backgammon https://www.bkgm.com/rules.html

[2]. Temporal Difference Learning and TD-Gammon By Gerald Tesauro https://www.csd.uwo.ca/~xling/cs346a/extra/tdgammon.pdf

[3]. backgammon-ai https://github.com/amitbar05/backgammon-ai/tree/main

[4]. https://github.com/awni/backgammon