KU LEUVEN

FACULTEIT
INGENIEURSWETENSCHAPPEN

Multi-Agent Learning in Canonical Games and Dots-and-Boxes

Final Report

Dag Malstaf (r0799028)
Pedro Gouveia (r0980130)
Prateek Grover(r0971804)
Academic Year **2023–2024**

Originality Declaration

We declare that we are the original and sole authors of this submitted report, except for feedback and support provided to us by the educational team responsible for the guidance, and except for authorised collaboration agreed upon with the same educational team. With regard to this paper, we also declare that:

1. We have taken note of the text on academic integrity at `https://eng.kuleuven.be/en/study/plagiarism`.
2. No plagiarism has been committed as described at `https://eng.kuleuven.be/en/study/plagiarism#Definitie:%20wat%20is%20plagiaat?`.
3. All experiments, tests, and measurements have been carried out as described in this present text, and no data or measurements have been manipulated.
4. All source references and citations have been included in good faith.

Contents

1 Problem Statement **1**

2 Task 1: Literature Study **1**

3 Task 2: Learning & Dynamics **1**
3.1 Matrix Games . . . 1
3.2 Reinforcement Learning Strategies . . . 2
3.2.1 Epsilon-Greedy Q-Learning . . . 2
3.2.2 Boltzmann Q-Learning . . . 2
3.2.3 Lenient Boltzmann Q-Learning . . . 2
3.2.4 Lenient Frequency Adjusted Q-Learning . . . 3
3.3 Results . . . 3
3.3.1 Subsidy Game . . . 4
3.3.2 Battle of the Sexes . . . 4
3.3.3 Prisoners Dilemma . . . 4
3.3.4 Biased Rock Paper Scissors Game . . . 4
3.4 Hyperparameter sensitivity analysis . . . 5
3.4.1 Epsilon - Greedy Q Learning Agent . . . 5
3.4.2 Boltzmann Q Learning Agent . . . 5
3.4.3 Lenient Boltzmann Q Learning Agent . . . 6

4 Task 3: Minimax for small Dots-and-Boxes **6**
4.1 Minimax Algorithm with Transposition Table . . . 6
4.2 Minimax with Transposition Table and Symmetry . . . 7
4.3 Minimax with Transposition Table, Symmetry and Chains . . . 7
4.4 Solutions Per Board Size . . . 7
4.5 Discussion on Results . . . 7

5 Task 4: Full Dots-and-Boxes **8**
5.1 Monte Carlo Tree Search . . . 8
5.2 AlphaZero . . . 8
5.2.1 Approach . . . 8
5.2.2 Model Architecture . . . 9
5.2.3 Results and Improvements . . . 9
5.3 Graph Neural Net . . . 9
5.3.1 Approach . . . 9
5.3.2 Board State to Graph Conversion . . . 9
5.3.3 Model Architecture . . . 10
5.3.4 Variable Board Size . . . 10
5.4 Discussion of Results . . . 10

6 Code submit folder **11**

7 Hours Worked **11**

A Task 2 **12**
A.1 Nash Equilibrium . . . 12
A.1.1 Pure Nash Equilibrium . . . 12
A.1.2 Mixed Nash Equilibrium - 2 Players . . . 12
A.1.3 Mixed Nash Equilibrium - 3 Players . . . 13
A.2 Payoff Matrices . . . 13
A.2.1 Subsidy Game . . . 13
A.2.2 Battle of the Sexes . . . 13
A.2.3 Prisoner's Dilemma . . . 13
A.2.4 Rock, Paper, Scissors . . . 13
A.3 Replicator Dynamics for Lenient Boltzmann Q-Learning . . . 13
A.4 Parameter Selection . . . 14

B Task 3 **14**

C Task 4 **16**
C.1 Architecture diagrams 16
C.2 GNN Hyperparameters 16
C.3 Center and spokes algorithm 17

1 Problem Statement

This project aims to develop an intelligent agent capable of playing the Dots-and-Boxes game at a competitive level. Our approach involves applying machine learning and game theory techniques to understand and solve the complexities of the game, especially considering its vast number of possible states and game outcomes. The report reviews relevant literature, explores basic reinforcement learning, applies the Minimax algorithm to small board sizes and employs Monte Carlo Tree Search with deep learning for larger board sizes.

2 Task 1: Literature Study

Multiagent Reinforcement Learning (MARL) has rapidly advanced, providing robust solutions for complex decision-making involving multiple interacting agents. This section provides an overview of seminal works that frame our understanding and implementation of MARL, combined with insights from game theory. Key contributions include Busoniu et al.'s categorization of MARL algorithms, highlighting cooperative and competitive dynamics [7], and Zhang et al.'s overview of MARL theories within Markov and stochastic games [35]. Innovations in deep reinforcement learning by Lowe et al. introduce the Multi-Agent Actor-Critic framework, tailored to overcome traditional algorithmic challenges in multi-agent settings [18]. Additional strategic analyses, such as the iterated prisoner's dilemma by Sandholm and Crites, and foundational concepts like Nash equilibrium by Holt and Roth, are crucial for strategic interaction modeling [12].

Mnih et al.'s development of deep Q-networks marks a significant advancement in applying high-dimensional sensory input to AI decision-making, significantly outperforming human capabilities in complex environments [20]. Furthermore, exploration-exploitation trade-offs are refined by Tokic's Value-Difference Based Exploration and Watkins and Dayan's foundational work on Q-learning, essential for optimizing agent actions in controlled domains [27, 29]. In our endeavor to enhance understanding and develop strategies for multi-agent learning within the goal of the Dots-and-Boxes game, we undertook a comprehensive literature review in key theoretical domains. Central to our exploration were groundbreaking works on Evolutionary Dynamics of Multi-Agent Learning, the intricacies of Multi-Agent Systems, and foundational principles of Game Theory and Nash Equilibrium [24, 22].

To address the challenges presented by our benchmark games in task 2, we incorporated three distinct methodologies: ϵ-greedy strategy, Lenient Boltzmann Q-learning, and an innovative adaptation known as Lenient Frequency Adjusted Q-Learning. This latter approach, inspired by an innovative paper in the field, demonstrates robust potential for converging towards Pareto optimal equilibria [5]. Grounded in the principles of evolutionary game theory, these strategies offer a promising avenue for the advancement of our project objectives. By integrating these methodologies, we aim to conduct a comprehensive evaluation and comparison based on their respective vector fields and learning trajectory plots. In tasks 3 and 4, we reviewed key algorithms and frameworks to enhance our approach. The Monte Carlo Tree Search (MCTS) algorithm, as explored by Chaslot et al.[8], is crucial for decision-making in high-dimensional spaces through random sampling. Berlekamp's analysis of Dots and Boxes[4] offers strategic insights into this combinatorial game. The AlphaZero framework, detailed by Zhang et al.[34] and expanded by DeepRL[11], advances game strategy via self-play and neural network optimization. Targ et al.'s work on Residual Networks (ResNet)[26] improves deep learning by addressing vanishing gradient issues. Xu et al.[32] extend deep learning to non-Euclidean structures with Graph Neural Networks (GNN), which were crucial for our agent's adaptability to different board sizes.

3 Task 2: Learning & Dynamics

3.1 Matrix Games

In this section, we investigate the dynamics of four benchmark matrix games: Biased Rock-Paper-Scissors, Subsidy, Battle of the Sexes, and Prisoner's Dilemma. Each of these games exemplifies a unique category: zero-sum, social dilemma, or coordination. Zero-sum games result in a net gain of zero across all players, while social dilemmas introduce a conflict between individual and collective interests. Coordination games, on the other hand, require players to align their strategies to achieve mutually beneficial outcomes. [33]

Our primary objective in analyzing these games is to identify the Nash equilibria and Pareto optimal solutions for each game through mathematical analysis. Nash equilibria represent stable states where no player can benefit by unilaterally changing their strategy, while Pareto optimal solutions ensure that no player can be made better off without making another worse off. Additionally, we aim to identify mixed Nash equilibria, where players randomize over strategies to maximize their expected payoffs. The calculations for determining these mixed Nash equilibria, along with the payoff matrices for all the games analyzed in this study, can be found in the appendix (A.1, A.2). These theoretical findings will later be validated using various game-playing agents developed during this study. For a concise overview of the expected pure Nash equilibria, mixed Nash equilibria, and Pareto optimal points for each of the games, please refer to Table 1.

Table 1: Nash Equilibria and Pareto Optimal Points for Matrix Games

Game	Pure Nash Equilibria	Mixed Nash Equilibria	Pareto Optima
Biased Rock-Paper-Scissors	-	{P(R)=0.625, P(P)=0.3125, P(S)=0.0625}	Every State
Subsidy Game	(S_1,S_1); (S_2,S_2)	{P(S_1)=0.9091, P(S_2)=0.0909}	(S_1,S_1)
Battle of the Sexes	(O,O); (M,M)	{P(preference)=0.6, P(non-preference)=0.4}	(O,O); (M,M)
Prisoner's Dilemma	(D,D)	-	(C,C); (C,D); (D,C)

3.2 Reinforcement Learning Strategies

In this section, we explore and evaluate a series of reinforcement learning strategies designed to solve the benchmark matrix games discussed previously. These strategies represent a progression of methods, each building upon the previous one to address specific limitations and enhance performance in finding optimal solutions. Each of these strategies is assessed for its ability to converge to Nash equilibria and Pareto optimal solutions within the complex dynamics of the matrix games.

3.2.1 Epsilon-Greedy Q-Learning

Epsilon-Greedy Q-Learning [31] is a reinforcement learning strategy that strikes a basic balance between exploration and exploitation. In this method, the agent primarily exploits the current knowledge by selecting the action with the highest Q-value but occasionally explores other actions with a probability of ϵ. Specifically, at each decision point, the agent chooses a random action with probability ϵ and selects the best-known action with probability $1 - \epsilon$. This approach allows the agent to explore the action space to discover potentially better actions, especially in the early stages of learning when the Q-values are not well established. The value of ϵ is often decayed over time, starting with a higher value to encourage exploration and gradually reducing it as the agent becomes more confident in its learned policy. This helps the agent to converge to an optimal solution by focusing more on exploitation as the learning progresses.

3.2.2 Boltzmann Q-Learning

Boltzmann Q-learning [13] [14] is a reinforcement learning strategy that uses a Boltzmann (softmax) action selection mechanism to balance exploration and exploitation. In this method, the probability of selecting an action i at time t is given by the Boltzmann distribution:

$$x_i(t) = \frac{e^{Q_i(t)/T}}{\sum_k e^{Q_k(t)/T}} \tag{1}$$

Boltzmann Q-learning is a foundational method within our analysis. where $Q_i(t)$ represents the Q-value associated with action i at time t, and T is the temperature parameter controlling the exploration-exploitation tradeoff. As $T \to 0$, the policy becomes greedy, favoring actions with the highest Q-values (pure exploitation). Conversely, as $T \to \infty$, the action selection becomes random (pure exploration). It is noteworthy that Boltzmann Q-learning can be viewed as a specific case of Lenient Boltzmann Q-Learning where the leniency parameter κ is set to 1 [6]. This setting effectively removes the leniency effect, making it equivalent to the standard Boltzmann exploration strategy, which balances exploration and exploitation based solely on a temperature parameter.

3.2.3 Lenient Boltzmann Q-Learning

Lenient Boltzmann Q-Learning [23] is an enhancement of the standard Q-learning algorithm specifically designed for cooperative multi-agent environments. The key concept behind leniency is the idea of "forgiving" initial suboptimal actions or mis-coordination between agents, thereby allowing them to explore and learn without being prematurely discouraged by early low rewards. This approach addresses the challenge where agents, particularly in environments with conflicting or fluctuating rewards, might otherwise converge to suboptimal strategies due to early negative reinforcement.

The primary advantage of introducing leniency is that it significantly improves the agents' ability to converge to Pareto optimal equilibria in cooperative games. By softening the impact of initial "wrong" decisions, Lenient Boltzmann Q-Learning ensures that agents remain open to exploring potentially better strategies even if their early attempts result in poor rewards. This leads to more robust learning and better overall performance, particularly in environments where coordination among agents is crucial.

The replicator dynamics for Lenient Boltzmann Q-Learning are an extension of those for Boltzmann Q-learning, incorporating a leniency factor κ, which governs the degree of leniency. The dynamics are given by [6] and can be found in appendix A.3. The hyperparameter $\kappa > 1$ introduces leniency by biasing the learning towards higher rewards encountered during the exploration phase. By setting $\kappa = 1$, the Lenient Boltzmann Q-Learning reduces to the standard Boltzmann Q-learning, where the algorithm follows the classical selection dynamics without additional leniency.

In our experiments, we observed that incorporating leniency significantly improved the agents' ability to handle conflicting rewards during initial exploration phases. The addition of leniency not only enhanced the theoretical model underlying the RL strategy but also resulted in noticeable changes in the vector field plots. With $\kappa > 1$, the dynamics shifted towards favoring the exploration of higher payoff strategies, leading to a more comprehensive search of the solution space and better convergence to optimal strategies.

3.2.4 Lenient Frequency Adjusted Q-Learning

The Lenient Frequency Adjusted Q-learning (LFAQ) [6] algorithm is an advanced reinforcement learning strategy that integrates the benefits of both Frequency Adjusted Q-learning (FAQ) and Lenient Boltzmann Q-Learning. LFAQ addresses some of the discrepancies observed in the standard Q-learning approach by adjusting the frequency with which actions are updated and by incorporating leniency to handle initial mis-coordination effectively.

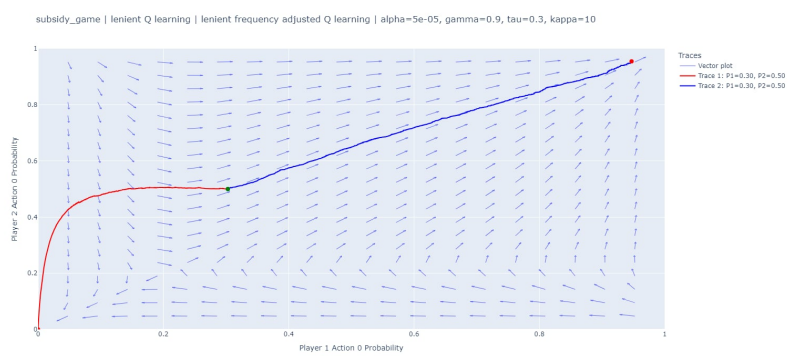
In standard Q-learning, only the selected action is updated asynchronously, leading to potential discrepancies with the theoretical model, especially with large step sizes. The evolutionary model assumes infinite evaluations, which isn't practical. As learning progresses, actions are disproportionately evaluated, causing bias. FAQ addresses this by introducing a weight inversely correlated to the action's selection probability, making updates more robust and better aligned with the model:

$$Q_i(t+1) \leftarrow Q_i(t) + \min\left(\frac{\beta}{x_i}, 1\right) \alpha \left[r(t+1) + \gamma \max_j Q_j(t) - Q_i(t)\right] \tag{2}$$

Here, β is a variable that controls the range of the policy space for which FAQ is valid, and x_i is the probability of selecting action i. When $\beta = 1$, FAQ reduces to standard Boltzmann Q-learning, effectively removing the frequency adjustment, making the learning process identical to the classical Boltzmann exploration strategy.

The introduction of frequency adjustment demonstrates that the learning trajectories are less sensitive to pessimistic starting Q-values, ensuring that the traces will better follow the vector field of the model. While introducing leniency ensures a better model that is more forgiving of initial suboptimal rewards, introducing frequency adjustment does not change the model but makes the learning strategy more robust to initial Q-values. This robustness is particularly evident when the starting Q-values are pessimistic, as the learning traces will still tend to follow the expected paths predicted by the theoretical model.

In this plot, the impact of frequency adjustment is illustrated. For the Subsidy Game, the Pareto optimal point is at (1,1), which corresponds to both players choosing the S1 strategy. Despite both learning strategies starting with the same initial Q-values, Lenient Boltzmann Q-Learning (LQ) tends to converge to the suboptimal equilibrium at (0,0), while Lenient Frequency Adjusted Q-learning (LFAQ) converges toward the Pareto optimal point. This highlights the advantage of combining leniency with frequency adjustment: LFAQ not only helps agents avoid getting stuck in suboptimal equilibria but also ensures that their learning paths remain aligned with the optimal vector field trajectories predicted by the evolutionary model.



3.3 Results

We evaluate the four learning strategies by analyzing game results and learning traces from different initial Q-values, trained for 10,000 iterations to ensure convergence and identify optimal points for each game.

Across all games, distinct convergence patterns emerge. Epsilon-Greedy Q-Learning shows slower and less consistent convergence, particularly in complex games. Boltzmann Q-Learning, with its temperature parameter, supports broader exploration, leading to more reliable convergence to Nash equilibria, though outcomes vary with initial conditions. Lenient Boltzmann Q-Learning enhances exploration by forgiving early suboptimal rewards, increasing the likelihood of reaching Pareto Optimal points like (S1, S1). Lenient Frequency Adjusted Q-Learning (LFAQ) further refines this process by balancing exploration and exploitation, leading to more stable convergence to equilibrium points.

Overall, incorporating Boltzmann temperature, leniency, and frequency adjustments significantly improves the convergence of learning algorithms, especially in games with multiple equilibria and complex payoffs. In Section 3.4, we discuss how we came to optimal values of hyperparameters to use in each of the learning agents above via a parameter search performed on variety of games. The table in the Appendix provides these values A.4.

3.3.1 Subsidy Game

There exist two Nash equilibria in pure strategies, they are: $(S1, S1)$ and $(S2, S2)$. There exists one Pareto optimal point at $(S1, S1)$ which dominates the one at $(S2, S2)$, since the payoff obtained at $(S1, S1)$ is greater for both players. Replacing payoff table values in the above equations, we observe a Nash equilibrium in mixed strategies. Player 1 plays the mixed strategy of $(S1, S2) = (0.9091, 0.0909)$, with an expected payout of 10.9092. Similarly, Player 2 also plays the mixed strategy of $(S1, S2) = (0.9091, 0.0909)$, with an expected payout of 10.9092.

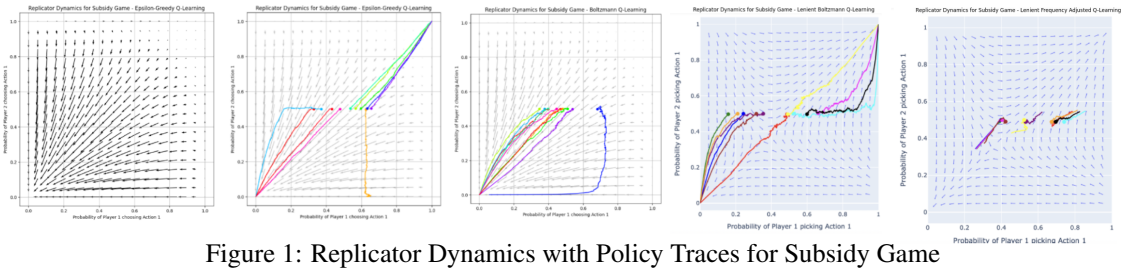


Figure 1: Replicator Dynamics with Policy Traces for Subsidy Game

3.3.2 Battle of the Sexes

There exist two Nash equilibria in pure strategies, they are: $(\text{Opera}, \text{Opera})$ and $(\text{Movies}, \text{Movies})$. Both of these points are Pareto optimality points, since a higher reward cannot be obtained by a player unless by reducing the reward of the complimentary player. There also exists a Nash equilibrium in mixed strategies. We obtain the mixed Nash equilibrium by replacing payoff table values in the above equations. Player 1 plays the mixed strategy of $(\text{Opera}, \text{Movies}) = (0.6, 0.4)$, with an expected payout of 1.2. Similarly, Player 2 also plays the mixed strategy of $(\text{Opera}, \text{Movies}) = (0.6, 0.4)$, with an expected payout of 1.2.

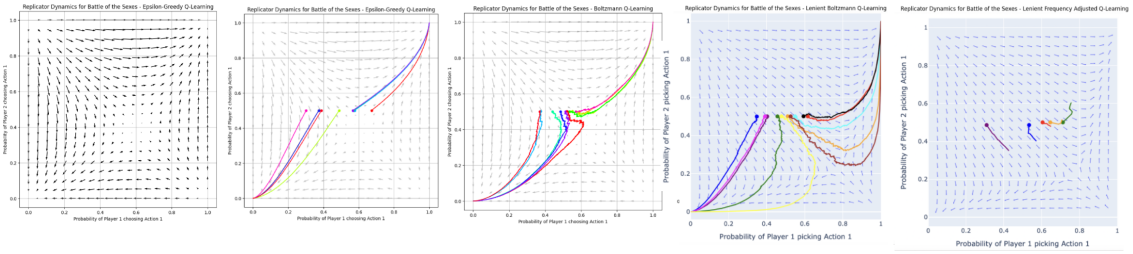


Figure 2: Replicator Dynamics with Policy Traces for Battle of the Sexes

3.3.3 Prisoners Dilemma

There exists one Nash equilibrium in pure strategies, which is: $(\text{Deceive}, \text{Deceive})$. There does not exist a Nash equilibrium in mixed strategies, as evident when we replace payoff table values in the system of equations, we do not get a solution. The Pareto optimality points however, exist for $(\text{Confess}, \text{Deceive})$, $(\text{Deceive}, \text{Confess})$, and $(\text{Confess}, \text{Confess})$.

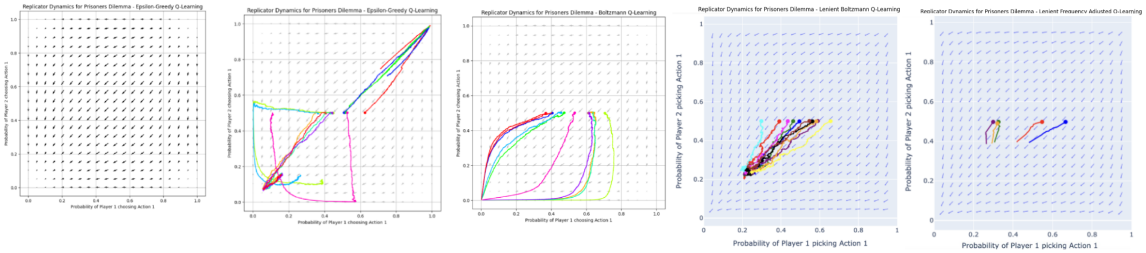


Figure 3: Replicator Dynamics with Policy Traces for Prisoners Dilemma

3.3.4 Biased Rock Paper Scissors Game

This game is a variant of the classic zero-sum game with an agnostic structure, characterized by a trio of actions available to each player. The complexity of the game is augmented by the absence of dominant strategies, which typically facilitate analytical simplifications in traditional game-theoretic models. Notably, there is no pure Nash equilibrium in this game. Unlike traditional scenarios where pure strategies might constitute equilibrium points, in the biased Rock-Paper-Scissors game, no single strategy pair (such as $(\text{Rock}, \text{Rock})$,

(Paper, Paper), or (Scissors, Scissors)) can serve as a best response to the other. Consequently, players are incentivized to continuously adjust their strategies, preventing any pure strategy from being a stable equilibrium. However, there does exist a mixed Nash equilibrium in this game. This stochastic equilibrium is located at $\{P(\text{Rock}) = 0.625, P(\text{Paper}) = 0.3125, P(\text{Scissors}) = 0.0625\}$. At this point, no player can improve their expected payoff by unilaterally changing their strategy, meaning that this mixed strategy combination is stable under the dynamics of the game. Detailed derivations on the calculation of this mixed Nash equilibrium can be found in Appendix A.1.

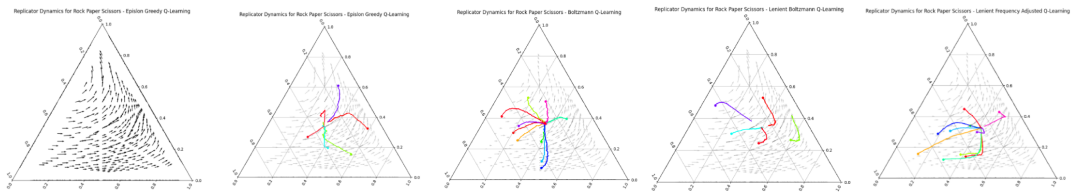


Figure 4: Replicator Dynamics with Policy Traces for Biased Rock Paper Scissors

3.4 Hyperparameter sensitivity analysis

3.4.1 Epsilon - Greedy Q Learning Agent

The figure shows that learning trajectories are highly sensitive to the choice of learning rates in Battle of the Sexes Game. Learning rates greater than 0.01 lead to noisy behavior as the trajectory approaches the endpoint. Conversely, learning rates below 0.001 are too slow, preventing the trajectory from reaching the equilibrium point within 10,000 steps.

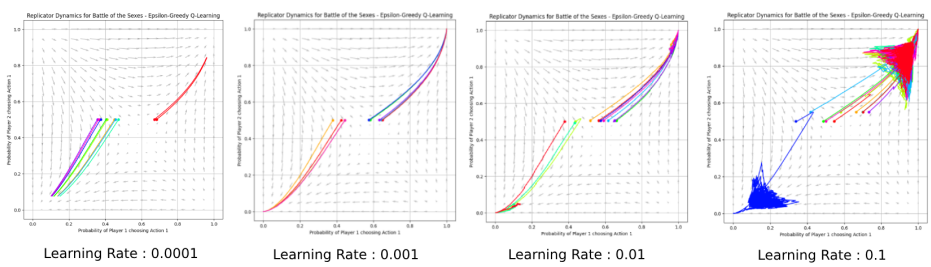


Figure 5: Epsilon Greedy Learning Trajectories with Different Learning Rates

Consequently, we also analyze epsilon, the exploration-exploitation factor, and observe minimal variation across most values. However, extreme values below 0.001 and above 0.5 introduce noticeable noise. Based on this, an epsilon value of 0.1 appears to be appropriate for our study.

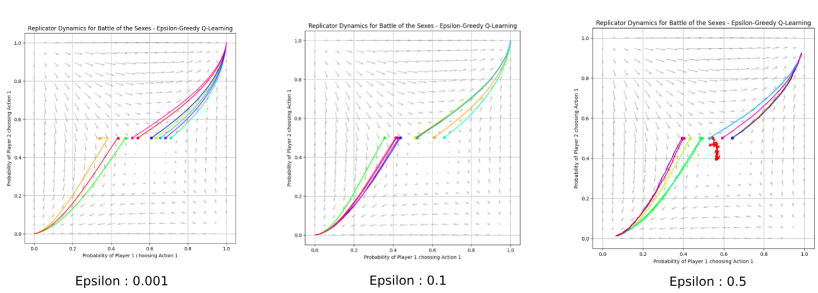


Figure 6: Epsilon Greedy Learning Trajectories with Different Epsilon Values

3.4.2 Boltzmann Q Learning Agent

Next, we examine the impact of temperature on the Boltzmann Q-Learning Agent. Lower temperatures result in a straightforward progression along the gradients toward the equilibrium point. In contrast, temperatures above 1 significantly degrade performance, causing

the agent to struggle in following the gradients within the given timesteps.

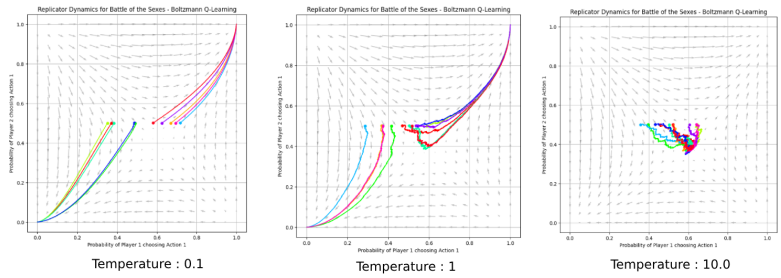


Figure 7: Boltzmann Learning Trajectories with Different Temperature Values

3.4.3 Lenient Boltzmann Q Learning Agent

To understand the impact of leniency, we analyzed trajectories in the Subsidy game. Our observations reveal that increasing leniency significantly influences the replicator dynamics vector field. Specifically, as leniency increases, the vector field becomes more oriented toward the Pareto optimal point. This change is beneficial as it drives more trajectories to converge on this optimal point. Consequently, higher leniency promotes convergence towards the Pareto optimal strategy, enhancing the effectiveness of the learning process.

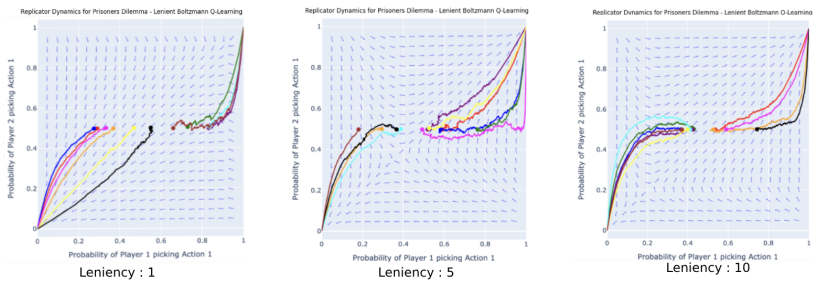


Figure 8: Lenient Boltzmann Learning Trajectories with Different Leniency Values

4 Task 3: Minimax for small Dots-and-Boxes

In this section, we tackled solving small instances ((1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (3,3)) of the Dots-and-Boxes game using the minimax algorithm. Dots-and-Boxes is a classic combinatorial game playable with just paper and pencil. Players take turns drawing horizontal or vertical lines between neighboring dots on a grid, claiming boxes when completed, with the goal of owning the most boxes by game end.

Despite its simple rules, Dots-and-Boxes poses a challenge due to its vast number of possible states and unique games (Table 4). Our objective is to implement and optimize the minimax algorithm to efficiently solve these small game instances. Depending on the grid's size with x rows and y columns, we can compute the number of states and unique games using the following formulas:

$$\text{States} : 2^{x(y+1)+y(x+1)} \qquad \text{Unique Games} : (x(y+1)+y(x+1))! \tag{3}$$

I The generic minimax algorithm [21] is a foundational technique in game theory and artificial intelligence for decision-making in two-player zero-sum games. It recursively evaluates game states to identify the optimal move for the current player. By constructing a game tree from the current state, evaluating terminal nodes using a scoring function, and propagating scores back up the tree, it selects the move maximizing the current player's minimum gain. Benchmarking the base algorithm's results (Table 5) indicates room for improvement, with runtime timeouts occurring for boards larger than 2x2. Hence, further optimizations are required.

4.1 Minimax Algorithm with Transposition Table

The first optimization we did to minimax was the use of a transposition table [4]. In the transposition table optimization, we "cache" the results of previously evaluated states. When we encounter these states again during the search, we can reuse the cached results instead of recomputing them. This speeds up the process of finding the solution when compared with the generic minimax as can be seen in table 5. This happens because we perform fewer unnecessary searches, as repeated states are quickly resolved using the cached results. Each key in the transposition table is composed of the triple (board state, current player, number of captured boxes by player 1 or player 2), as these different aspects are what differentiate different game states in the minimax tree.

4.2 Minimax with Transposition Table and Symmetry

To further optimize the minimax algorithm, we exploited the fact that many game states in Dots-and-Boxes are symmetric [4]. Symmetric states share the same value because the optimal strategies can be applied similarly. By recognizing and handling these symmetries, we can avoid redundant searches and reduce the computational effort.

In our approach, we converted the state string into a grid representation that could be easily transformed. For square grids, we applied horizontal and vertical reflections, as well as 90, 180, and 270-degree rotations, having 8 different kinds of rotations. For non-square boards, we can apply horizontal and vertical reflections on the original state, and then rotate this state 180 degrees and do the same kind of reflections, so we have 4 different types of transformations. By generating these transformations, we identified the canonical form of the state—the lexicographically smallest transformation. This canonical form was then stored in the transposition table. When encountering a state, if its canonical form already existed in the table, we could reuse the stored result, significantly reducing the number of entries in the transposition table, however, the time increased relative to the version with only the transposition table, likely due to the fact of having to generate every canonical form, but still being significantly faster than the generic version.

4.3 Minimax with Transposition Table, Symmetry and Chains

To optimize the minimax algorithm's performance further, we introduced a chain strategy inspired by Berlekamp's work. This strategy aims to identify and exploit chain structures within the game, which can significantly impact its outcome. It adapts based on the current game state, particularly assessing the agent's control. When the agent moves first, control is achieved if both rows and columns have an even number of long chains, or if both have an odd number of long chains. Conversely, when the agent moves second, control is attained under the opposite conditions.

The strategy unfolds in several steps. Firstly, the player aims to form a chain of three or more boxes whenever possible to maximize their potential for capturing multiple boxes in sequence. Secondly, if the player finds themselves in control but is unable to form a long chain, they prioritize creating a short chain of two boxes to maintain dominance on the board. Lastly, if the player is not in control, they initiate the formation of a new longer chain to mislead the opponent and regain control of the game. To implement this strategy, the board is converted into a graph representation, modeling all possible moves as edges between nodes. We then explore the presence of chains in the state and utilize the chain strategy to pick an action if found. Otherwise, we rely on the standard minimax computed action. The code implementation is referred from [17]. With this implementation, we can only see a noticeable reduction in the table size when testing for larger board sizes like 3x3 (7), while on smaller boards there is no notable improvement when compared with the last section's version and comes with a significant decrease in time performance.

4.4 Solutions Per Board Size

In addition to optimizing the minimax algorithm, we also determined the solutions (winners) for each board size, by moving through the entire search space and obtaining the optimal moves. The following table summarizes the winners for each board size. These results are corroborated by [4].

Board Size	1 x 1	1 x 2	1 x 3	1 x 4	2 x 2	2 x 3	3 x 3
Winner	Player 2 wins	Draw	Player 2 wins	Draw	Player 1 wins	Player 2 wins	Player 2 wins

Table 2: Winners for Different Board Sizes in Dots-and-Boxes

4.5 Discussion on Results

Based on performance data, we draw conclusions about the impact of the three enhancements on time and memory usage (more detailed results in tables 7 and 5). Memory usage, illustrated in Figure 9, shows that employing symmetry or symmetry with chains results in significantly smaller transposition tables due to the identification of more potential identical states. However, the chain strategy captures more states as board sizes increase, leading to a larger reduction in key percentage for larger boards, averaging 63%. Despite the memory efficiency, there's an increase in runtime, as shown in Figure 10. The need for symmetry operations and chain calculations introduces additional computational overhead. However, we believe this disadvantage is manageable. For larger board sizes, utilizing chains and symmetry is likely to improve decision-making speed and accuracy, especially under time constraints. The estimated increase in execution time is 77% with symmetry and 380% with the chain strategy. Nevertheless, this trade-off may be justified by potential gains in precision and efficiency for larger and more complex boards.

So while the increased runtime is a notable drawback, it is important to consider the overall benefits in terms of improved decision-making capabilities. For larger and more complex board sizes, the ability to quickly identify optimal strategies becomes crucial, especially in scenarios where time is a limiting factor. The reduced transposition table size, coupled with the strategic advantages provided by symmetry and chains, supports that these optimizations do work and are beneficial. However, minimax is still inefficient for large board sizes.

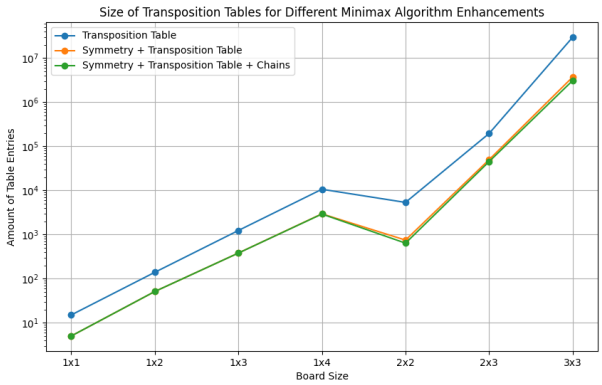


Figure 9: Amount of table entries of the transposition table on different board sizes and optimizations

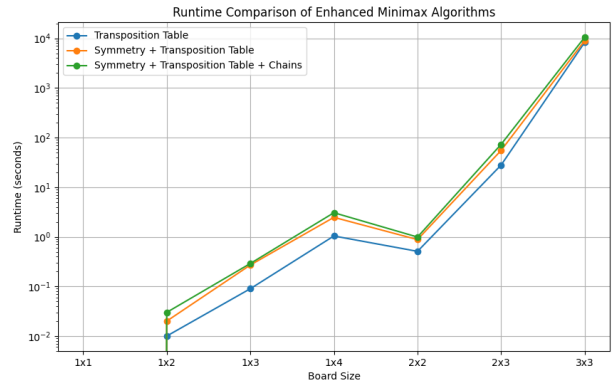


Figure 10: Runtime of minimax on different board sizes and optimizations

5 Task 4: Full Dots-and-Boxes

This section addresses the challenge of solving the Dots-and-Boxes game for larger and variable board sizes. The computational complexity of the Minimax algorithm, as discussed previously, renders it impractical for larger boards. To overcome this limitation, we developed three agents: the MCTS agent and the GNN agent, both optimized for any board size, and the AlphaZero agent, trained specifically for 7x7 boards. We ran multiple benchmarks for these agents and will display them at the end of this section. All neural networks were trained using cloud computing resources, leveraging an Nvidia RTX 4090 GPU. We ran multiple benchmark experiments to evaluate the performance of these agents on a 7x7 board size.

5.1 Monte Carlo Tree Search

In our approach to solving the Dots-and-Boxes game, we first implemented a standard Monte Carlo Tree Search (MCTS) agent [8] [25]. MCTS is a heuristic search algorithm used for decision-making in games, leveraging random sampling of the search space. It is a key component in AlphaZero [34], which utilizes MCTS to evaluate potential moves during self-play, guided by a policy network. The core of MCTS consists of four stages: selection, expansion, simulation, and backpropagation, as can be seen in Figure 15. During the selection phase, the Upper Confidence Bound (UCB) formula is applied to balance exploration and exploitation, ensuring a comprehensive search of the move space. The UCB1 formula, introduced by Kocsis and Szepesvári [9] [15], is defined as:

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

where: w_i is the number of wins for the node considered after the i-th move, n_i is the number of simulations for the node considered after the i-th move, N_i is the total number of simulations after the i-th move run by the parent node of the one considered and c is the exploration parameter, theoretically equal to $\sqrt{2}$; in practice, it is usually chosen empirically. The first component of the formula corresponds to exploitation, being high for moves with a high average win ratio. The second component corresponds to exploration, being high for moves with few simulations.

To enhance MCTS efficiency, we incorporated game-specific strategies such as chain creation, spoke formation, and central block control [2] [4]. Chains allow a player to capture multiple boxes sequentially, spokes establish early control over multiple edges, and central block control secures critical areas early, preventing the opponent from forming long chains. Alternating between spoke and central moves at the game's start helps our agent establish early control and disrupt the opponent's strategies. We developed an algorithm to determine center and spoke moves for any $n \times n$ board size, detailed in the appendix as Algorithm 2. This algorithm ensures our agent can dynamically apply these strategies across various board sizes, maintaining adaptability and robust performance.

5.2 AlphaZero

5.2.1 Approach

The MCTS algorithm is computationally intensive, leading to prohibitively long evaluation times during gameplay, especially for large board sizes where evaluation time increases exponentially. To address this, we adopted the AlphaZero framework, integrating a neural network (NN) within MCTS to approximate the policy and value functions. In this approach, MCTS leverages the NN value estimate instead of the roll-out phase, and the UCT formula is adjusted using prior probabilities as specified in the AlphaZero framework [34].

Training the neural networks followed the AlphaZero framework [11]. This framework employs self-play, enabling the agent to continuously learn by playing games against itself. After each self-play session, the resulting games are used for training. The target value and policy, derived from self-play, are compared with the predicted value and policy. The value loss is computed using mean squared error, and the policy loss is calculated using cross-entropy loss. These losses are summed to produce a total loss, minimized through backpropagation [16]. This framework can be seen in appendix C.1. After training, the new model is compared with the old model. If the new model wins 55% of the games, it replaces the old model, and the loop continues with this updated model. To enhance our dataset, we included benchmark data in the training data, resulting in a total of 850 MB of self-play data generated during our training loops. The complete workings of the framework are illustrated in Figure 13 (see Appendix).

5.2.2 Model Architecture

We utilized a residual network (ResNet) [26], specifically designed to capture spatial patterns from the game board. Convolutional layers within the ResNet are adept at learning hierarchical features from local neighborhoods, thereby facilitating the recognition of crucial board configurations and moves. The incorporation of residual connections allows the model to effectively capture both short-range and long-range dependencies, mitigating the vanishing gradient problem and enabling the representation of complex patterns. Additionally, these connections contribute to the model's ability to generalize across variable board sizes. The complete model architecture, inspired by the AlphaGo implementation [34], is illustrated in Figure 16. The board is represented as an image using the OpenSpiel observation tensor, reshaped to $(3, (\text{num_rows_input} + 1) \times (\text{num_cols_input} + 1), 3)$.

5.2.3 Results and Improvements

Despite the theoretical promise shown by the AlphaGo implementation [34], we encountered practical limitations, including insufficient computational power, large model size, and long inference times, rendering it unsuitable for the tournament. Moreover, the model's input layer was fixed to specific board sizes, conflicting with the need to handle any board size. Thus, we required an alternative architecture. Benchmarking results for this model can be seen in Figure 12. Future enhancements may involve extending training duration, thorough hyperparameter tuning, and developing a more concise board representation. Additionally, incorporating insights from Task 3, such as leveraging symmetries to expand the dataset and investigating the impact of starting as player 1 or player 2, could further improve performance.

5.3 Graph Neural Net

5.3.1 Approach

Given the constraints of the CNN model, we explored a new architecture. The Dots-and-Boxes game is naturally represented as a graph, with dots as nodes and edges as potential line segments. Graph Neural Networks (GNNs) emerged as a promising solution [32] [30]. They are lightweight, accept any input size, and train faster. GNNs capture spatial relationships by aggregating information from neighboring nodes and can use attention mechanisms to focus on critical nodes and edges, identifying important moves and board configurations. GNNs are adaptable to different graph sizes, making them suitable for variable board sizes. A significant advantage of the GNN model is its compact size. The CNN model's weights occupied 305.6 MB, whereas the GNN model's weights were only 280 KB. This substantial reduction in size enhances efficiency, enabling faster training and inference times without compromising performance.

This architecture meets the project's requirements. After implementation, we integrated the model into the AlphaZero framework for self-play and benchmarking. We incorporated hyperparameter tuning using the Optuna framework [1], leveraging its Define-by-Run API and optimization algorithms like TPE (Tree-structured Parzen Estimator) [28] and CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [10]. Optuna maximized the hyperparameters for the win rate against an MCTS agent, while the AlphaZero loop minimized loss for both policy and value predictions.

Once a win rate above 95% against the MCTS agent was achieved, the Optuna feedback loop was discontinued, and the AlphaZero training loop continued improving the network using self-play techniques. This training loop generated approximately 1150 MB of self-play data. The final optimal hyperparameters, listed in Table 8, resulted in a win rate of over 95% when playing 100 games against the OpenSpiel MCTS agent set at 100 simulations.

5.3.2 Board State to Graph Conversion

In order to effectively leverage Graph Neural Networks (GNNs) for the Dots-and-Boxes game, we employed a graph representation that reflects the underlying structure and strategies of the game. The game state is converted into a graph structure using an enhanced version of the coin-and-string representation described in [2] [3]. In this representation, **nodes** represent the boxes on the board, with enriched features capturing strategic aspects such as: ownership (0 if empty, 1 if captured by player 1, and 2 if captured by player 2), the number of unfilled sides (indicating potential for future box captures), and additional features for chain-based strategies including a binary indicator for chain membership, chain length, and the potential for a box to join a chain. **Edges** connect adjacent nodes on the board, with attributes

expanded to include: the state of the edge, its potential to form a chain (indicating strategic importance), and a critical edge indicator reflecting whether filling this edge could complete a box that is part of a chain. By incorporating these features, the graph representation is not merely a structural abstraction of the game board but is informed by specific game strategies. This allows the GNN to focus on important aspects of gameplay, such as chain formation and critical moves.

5.3.3 Model Architecture

Graph Neural Networks (GNNs) are well-suited for tasks with graph-structured data. In the Dots-and-Boxes game, the board is converted to a graph, where nodes are dots and edges are potential lines. GNNs perform convolutions on these graphs, efficiently aggregating information from neighboring nodes and edges.

The architecture used in this model is based on the Graph Attention Network (GAT) convolutional layers, which are designed to operate on both node and edge features simultaneously. Specifically, the model consists of three GATConv layers, each with 8 attention heads. These layers compute the output features for each node by attending to its neighbors, allowing the network to focus on the most relevant parts of the graph. The GATConv layers can incorporate edge features into the convolution process through the ‘edge_dim‘ parameter, enabling the model to consider both the state of the nodes and the edges when updating node embeddings. This ability to convolute over edge and node features at the same time is particularly advantageous in the Dots-and-Boxes game, where the relationships between boxes (nodes) and the connecting lines (edges) are crucial for strategy. We follow the same architecture as seen in appendix C.1, where we have replaced the GIN layers with the GATConv layers. The hyperparameters of the GNN model can be found in appendix C.2.

5.3.4 Variable Board Size

An important project requirement was enabling Dots-and-Boxes on any board size. Our GNN architecture supports this flexibility. Although primarily trained on 7x7 boards, we evaluated the model’s generalization across different sizes by competing against a random bot and an MCTS bot in a 20-round tournament. As shown in Figure 11, the model achieves an acceptable 70% win rate on unseen board sizes, demonstrating adaptability. It excels on 7x7 boards with a 95% win rate and performs well on 8x8 boards, indicating robustness. While the model can theoretically play on any (n×m) board, we focused on (n×n) boards up to n=12 due to computational limits. Future work should train on multiple board sizes, but resource constraints prevented this within the project scope.

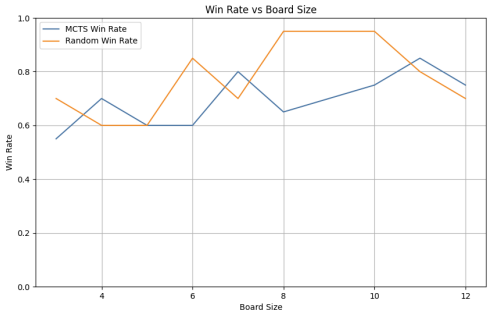


Figure 11: GNN agent vs Random & MCTS agent for increasing board sizes.

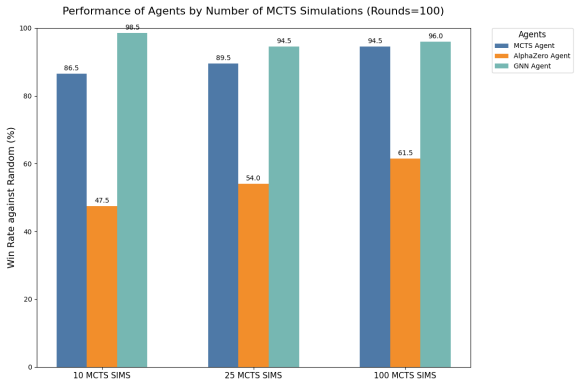


Figure 12: Benchmarking for the three agents against random.

5.4 Discussion of Results

Figure 12 shows a benchmarking overview of our three agents against the OpenSpiel random bot. The MCTS agent with game-specific strategies performs well, and the addition of a trained GNN evaluator significantly enhances its performance. The GNN performs best with 10 MCTS simulations, likely due to training bias. Further verification is needed. Our AlphaZero agent with a ResNet architecture performs suboptimally, likely due to insufficient training and data for such a complex model.

We tested the GNN against the random agent, with some runs achieving a 100% win rate, but we report the worst-case performance to provide a conservative estimate. To validate our agent, we tested it against the OpenSpiel MCTS agent with both agents having 20 MCTS simulations, resulting in a 92.5% win rate for our agent. Additionally, we challenged our agent by allowing it only 10 MCTS simulations against the opponent’s 200 MCTS simulations. Remarkably, our agent achieved a win rate of 82.5% in this scenario. This experiment highlights the strength of our model, demonstrating its ability to win the majority of games despite the opponent having 20 times more MCTS simulations per move.

6 Code submit folder

The code is available on the departmental computers at `/cw/lvs/NoCsBack/vakken/H0T25A/ml-project/r0799028/` and can also be accessed on a private GitHub repository upon request [19]. For Task 4, the final agents are located in the `submit_files` folder, and the output from the final experiments is in the `benchmark` folder. The hours worked on the project can be found in section 7.

7 Hours Worked

Group Member	Task 1	Task 2	Task 3	Task 4
Dag Malstaf	15	20	10	175
Pedro Gouveia	15	10	50	30
Prateek Grover	15	35	35	60

Appendix A Task 2

A.1 Nash Equilibrium

A.1.1 Pure Nash Equilibrium

A pure Nash equilibrium occurs when each player chooses a single strategy with certainty, and no player can benefit by unilaterally changing their strategy. The identification of pure Nash equilibria involves evaluating the payoff matrices for each possible combination of pure strategies and identifying the strategy profiles where no player has an incentive to deviate.

A.1.2 Mixed Nash Equilibrium - 2 Players

The general form of a two-strategy game is represented below:

	α	β
α	a, b	c, d
β	e, f	g, h

1. **Define Probabilities:**
   - Let Player 1's probability of choosing strategy α be $p = P1_\alpha$.
   - Let Player 2's probability of choosing strategy α be $q = P2_\alpha$.
2. **Calculate Player 1's Payoffs:**
   - Payoff to Player 1 when choosing strategy α:
   $$\text{Payoff}_\alpha = b \cdot q + f \cdot (1 - q)$$
   - Payoff to Player 1 when choosing strategy β:
   $$\text{Payoff}_\beta = d \cdot q + h \cdot (1 - q)$$
3. **Find Indifference Condition for Player 1:**
   - For Player 1 to be indifferent between choosing α and β, equate the payoffs:
   $$b \cdot q + f \cdot (1 - q) = d \cdot q + h \cdot (1 - q)$$
   - Solving for q:
   $$q = \frac{h - f}{b - d + h - f}$$
4. **Repeat for Player 2:**
   - Similarly, calculate the payoffs for Player 2 and find the probability p that makes Player 2 indifferent between their strategies:
   $$p = \frac{e - g}{a - c + e - g}$$
5. **Expected Payoffs:**
   - The expected payoff for Player 1 is given by:
   $$\text{Expected Payoff}_{\text{Player 1}} = p \cdot q \cdot a + p \cdot (1 - q) \cdot c + (1 - p) \cdot q \cdot e + (1 - p) \cdot (1 - q) \cdot g$$
   - The expected payoff for Player 2 is given by:
   $$\text{Expected Payoff}_{\text{Player 2}} = p \cdot q \cdot b + p \cdot (1 - q) \cdot d + (1 - p) \cdot q \cdot f + (1 - p) \cdot (1 - q) \cdot h$$

A.1.3 Mixed Nash Equilibrium - 3 Players

For a three-player game, the mixed Nash equilibrium is determined similarly by ensuring each player is indifferent among their strategies:

- Player 1 plays Rock, Paper, Scissors with probabilities p_1, p_2, and $1 - p_1 - p_2$, respectively.
- Player 2 plays Rock, Paper, Scissors with probabilities q_1, q_2, and $1 - q_1 - q_2$, respectively.

For Player 1:

$$E(R) = -0.05 \cdot q_2 + 0.25 \cdot (1 - q_1 - q_2) = -0.25q_1 - 0.30q_2 + 0.25$$
$$E(P) = 0.05 \cdot q_1 \cdot q_2 - 0.50 \cdot (1 - q_1 - q_2) = 0.55q_1 + 0.50q_2 - 0.50$$
$$E(S) = -0.25 \cdot q_1 + 0.50 \cdot q_2 \cdot (1 - q_1 - q_2) = -0.25q_1 + 0.50q_2$$

To make Player 1 indifferent among the moves, solve $E(R) = E(P) = E(S)$ to obtain:

$$q_1 = 0.625, \quad q_2 = 0.3125, \quad q_3 = 0.0625$$

Similarly, for Player 2, we solve for:

$$p_1 = 0.625, \quad p_2 = 0.3125, \quad p_3 = 0.0625$$

At these probabilities, the expected payoff for both players is 0.0.

A.2 Payoff Matrices

A.2.1 Subsidy Game

	$S1$	$S2$
$S1$	$12, 12$	$0, 11$
$S2$	$11, 0$	$10, 10$

A.2.2 Battle of the Sexes

	O	M
O	$3, 2$	$0, 0$
M	$0, 0$	$2, 3$

A.2.3 Prisoner's Dilemma

	C	D
C	$-1, -1$	$-4, 0$
D	$0, -4$	$-3, -3$

A.2.4 Rock, Paper, Scissors

	R	P	S
R	0, 0	-0.05, 0.05	0.25, -0.25
P	0.05, -0.05	0, 0	-0.5, 0.5
S	-0.25, 0.25	0.5, -0.5	0, 0

A.3 Replicator Dynamics for Lenient Boltzmann Q-Learning

$$u_i = \sum_j \frac{A_{ij} y_j \left[\left(\sum_{k:A_{ik} \leq A_{ij}} y_k \right)^\kappa - \left(\sum_{k:A_{ik} < A_{ij}} y_k \right)^\kappa \right]}{\sum_{k:A_{ik} = A_{ij}} y_k} \tag{4}$$

$$w_i = \sum_j \frac{B_{ji} x_j \left[\left(\sum_{k:B_{ki} \leq B_{ji}} x_k \right)^\kappa - \left(\sum_{k:B_{ki} < B_{ji}} x_k \right)^\kappa \right]}{\sum_{k:B_{ki} = B_{ji}} x_k} \tag{5}$$

$$\frac{dx_i}{dt} = \frac{\alpha x_i}{\tau} \left(u_i - x^T u \right) + x_i \alpha \sum_j x_j \ln \left(\frac{x_j}{x_i} \right) \tag{6}$$

$$\frac{dy_i}{dt} = \frac{\alpha y_i}{\tau} \left(w_i - y^T w \right) + y_i \alpha \sum_j y_j \ln \left(\frac{y_j}{y_i} \right) \tag{7}$$

A.4 Parameter Selection

Table 3: Hyperparameters of Different Agents

Hyperparameter	ϵ-greedy	Boltzmann Q-Learning	Lenient Boltzmann Q-Learning	LFAQ-Learning
epsilon	0.1	-	-	-
temperature	-	0.1	0.1	0.1
leniency	-	-	3	3
alpha	0.001	0.001	0.001	0.001
gamma	0.9	0.9	0.9	0.9
beta	-	-	-	0.001

Appendix B Task 3

Algorithm 1 Chains Strategy

```
Input: rows r, columns c, number of long chains n
Output: agent's move
if agent moves first then
    if (r and c are even and n is odd) or (not (r and c are even) and n is even) then
        agent gains control
    end if
else
    if (r and c are even and n is even) or (not (r and c are even) and n is odd) then
        agent gains control
    end if
end if
if agent is in control then
    if there is a chain of three or more boxes then
        agent takes the chain
    else
        agent aims for a short chain of two boxes
    end if
else
    if there is a chain of three or more boxes then
        agent takes the chain
    else
        agent initiates a new longer chain to deceive opponent
    end if
end if
```

Board Size (x, y)	**Number of States**	**Number of Unique Games**
1 x 1	$2^4 = 16$	$4! = 24$
1 x 2	$2^7 = 128$	$7! = 5040$
2 x 1	$2^7 = 128$	$7! = 5040$
2 x 2	$2^{12} = 4096$	$12! \approx 4.79 \times 10^8$
2 x 3	$2^{17} = 131072$	$17! \approx 3.55 \times 10^{14}$
3 x 2	$2^{17} = 131072$	$17! \approx 3.55 \times 10^{14}$
3 x 3	$2^{24} \approx 1.67 \times 10^7$	$24! \approx 6.2 \times 10^{23}$

Table 4: Number of states and unique games for different grid sizes

Board Size	**Generic Minimax**	**Transposition Table**	**Symmetry + Transposition Table**	**Symmetry + Transposition Table + Chains**
1 x 1	0.00	0.00	0.00	0.00
1 x 2	0.06	0.01	0.02	0.03
1 x 3	36.42	0.09	0.27	0.29
1 x 4	Timed out at 360 minutes	1.04	2.47	3.06
2 x 2	5851.03	0.51	0.88	0.99
2 x 3	Timed out at 360 minutes	27.95	55.15	73.11
3 x 3	Timed out at 360 minutes	8303.74	9004.38	10544.34

Table 5: Runtime Comparison of Generic Minimax and Enhanced Minimax Algorithms for Different Board Sizes

Board Size	**Transposition Table**	**Symmetry + Transposition Table**	**Symmetry + Transposition Table + Chains**
1 x 1	15	5	5
1 x 2	139	51	51
1 x 3	1232	380	379
1 x 4	10604	2943	2913
2 x 2	5341	748	638
2 x 3	194742	49755	44989
3 x 3	29598505	3712128	3087188

Table 6: Size of Transposition Tables for Different Minimax Algorithm Enhancements

Board Size	**Transposition Table**	**Symmetry + Transposition Table**	**Symmetry + Transposition Table + Chains**
1 x 1	0.0006	0.0002	0.0002
1 x 2	0.005	0.002	0.002
1 x 3	0.04	0.018	0.018
1 x 4	0.28	0.14	0.14
2 x 2	0.14	0.04	0.02
2 x 3	10	2.5	2.5
3 x 3	1280	160	160

Table 7: Memory allocation of Transposition Tables for Different Minimax Algorithm Enhancements in MBs

Appendix C Task 4

C.1 Architecture diagrams

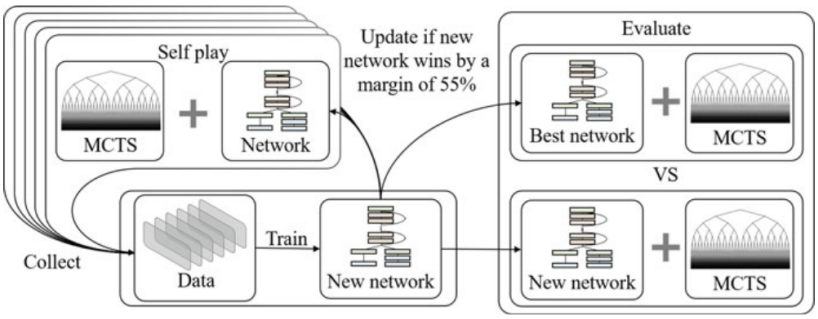


Figure 13: AlphaZero Training Framework.

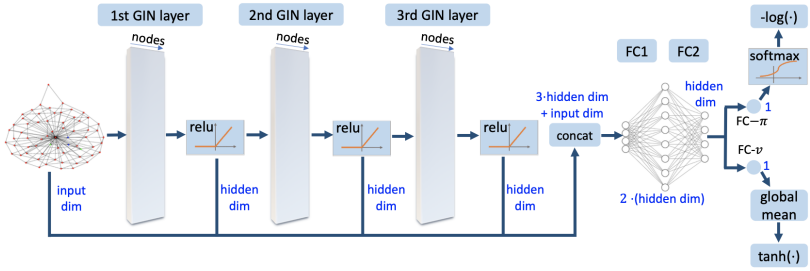


Figure 14: GNN Model Architecture.

C.2 GNN Hyperparameters

Node Features	5
Edge Features	3
Number of Channels	104
Learning Rate	4.1225e-05
L2 Coefficient	0.009083
GAT Attention Heads	8
Dropout Rate	0.5
Epochs	23

Table 8: GNN Hyperparameters.

C.3 Center and spokes algorithm

Algorithm 2 Calculate Center and Spokes Moves

```
function GET_CENTER_AND_SPOKES_MOVES(n)
    center_moves = []
    total_moves = 2 * n * (n + 1)
    i_divs = []
    j_divs = []
    if n % 2 == 0 then
        lines = (n - 1) // 2
        for i in range(0, lines) do
            i_divs.append([n // 2])
        end for
        for i in range(0, lines) do
            j_divs.append([lines - i, n - lines - i)])
        end for
    else
        boxes = n // 2
        for times in range(0, boxes) do
            i_divs.append([boxes - times, n - boxes + times])
        end for
        for times in range(0, boxes) do
            current_div = []
            for index in range(boxes - times, boxes + times + 1) do
                current_div.append(index)
            end for
            j_divs.append(current_div)
        end for
    end if
    for z in range(0, len(i_divs)) do
        for i in i_divs[z] do
            for j in j_divs[z] do
                center_moves.append(n * i + j)
            end for
            for i in i_divs[z] do
                for j in j_divs[z] do
                    center_moves.append(tot // 2 + (n + 1) * j + i)
                end for
            end for
        end for
    end for
    spokes_moves = [n, total_moves // 2 + 1, 2 * n - 1, total_moves // 2 + n - 1, total_moves // 2 - 2 * n, total_moves - n, total_moves // 2 -
n - 1, total_moves - n + (n - 2)]
    return center_moves, spokes_moves
end function
```

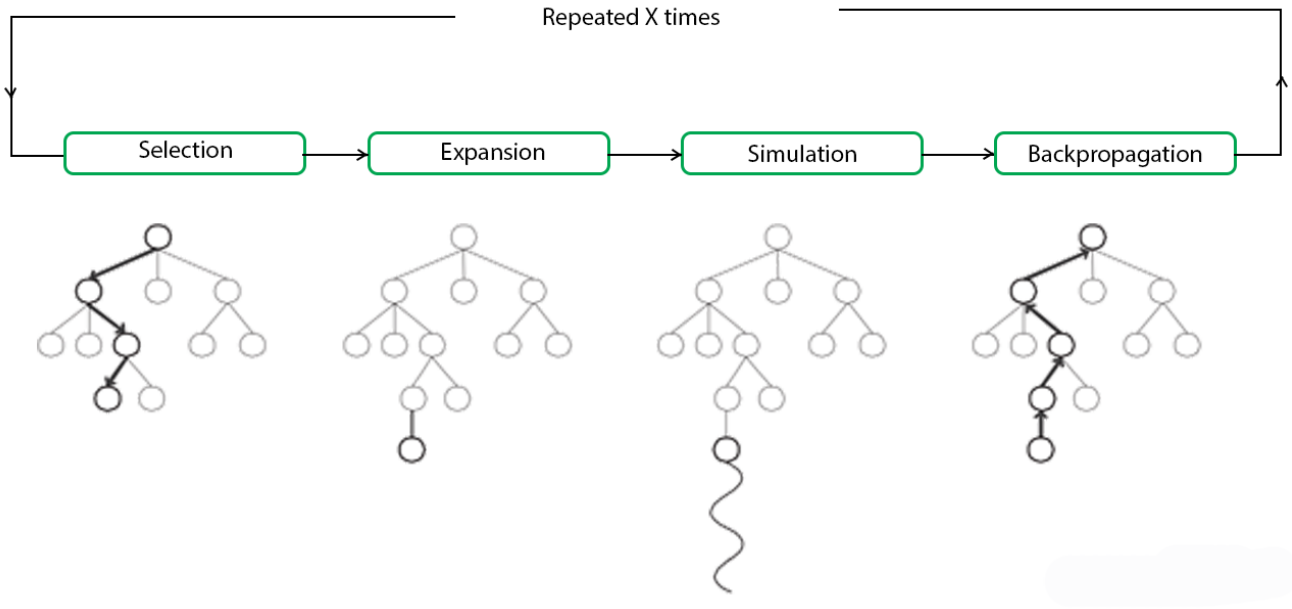


Figure 15: Four phases of MCTS (GeeksForGeeks).

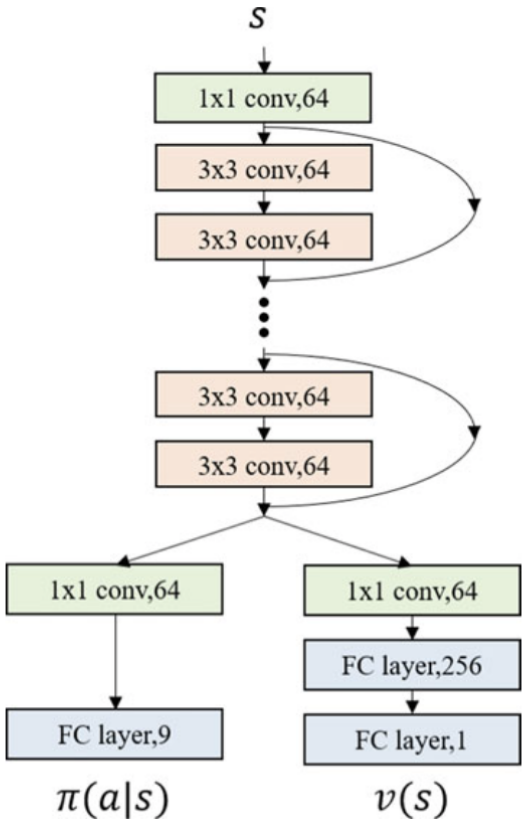


Figure 16: CNN Model Architecture.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *ACM*.

[2] Joseph Barker and Richard Korf. Solving dots-and-boxes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 414–419, 2012.

[3] Shai Ben-Assayag and Ran El-Yaniv. Train on small, play the large: Scaling up board games with alphazero and gnn. *arXiv preprint arXiv:2107.08387*, 2021.

[4] Elwyn R. Berlekamp. *The Dots-and-Boxes Game: Sophisticated Child's Play*. AK Peters/CRC Press, 2000.

[5] Daan Bloembergen, Michael Kaisers, and Karl Tuyls. Lenient frequency adjusted q-learning, 2010.

[6] Daan Bloembergen, Michael Kaisers, and Karl Tuyls. Lenient frequency adjusted q-learning. `http://www.flowermountains.nl/pub/Bloembergen2010lfaq.pdf`, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands.

[7] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38:156 – 172, 04 2008.

[8] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 4(1):216–217, Sep. 2021.

[9] András György, Levente Kocsis, Ivett Szabó, and Csaba Szepesvári. Continuous time associative bandit problems. In *IJCAI*, pages 830–835, 2007.

[10] Nikolaus Hansen. The cma evolution strategy: A tutorial, 2016.

[11] Shanghang Zhang Hang Yuan Hongming Zhang Jingqing Zhang Yanhua Huang Tianyang Yu Huaqing Zhang Ruitong Huang Hao Dong, Zihan Ding. *Deep Reinforcement Learning: Fundamentals, Research, and Applications*. Springer Nature, 2020. `http://www.deepreinforcementlearningbook.org`.

[12] Charles A. Holt and Alvin E. Roth. The nash equilibrium: A perspective. *Proceedings of the National Academy of Sciences*, 101(12):3999–4002, 2004.

[13] Michael Kaisers and Karl Tuyls. Replicator dynamics for multi-agent learning: An orthogonal approach. In Matthew E. Taylor and Karl Tuyls, editors, *Adaptive and Learning Agents*, pages 49–59, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[14] Ardeshir Kianercy and Aram Galstyan. Dynamics of boltzmann q learning in two-player two-action games. *Phys. Rev. E*, 85:041145, Apr 2012.

[15] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1:1–22, 2006.

[16] S Anupama Kumar, Andhe Dharani, and Chandrani Chakravorty. Efficacy of autoencoders on image dataset. *Sentiment Analysis and Deep Learning*, page 1001.

[17] Thomas Lorincz. Arduino dots-and-boxes. `https://github.com/thomaslorincz/arduino-dots-and-boxes`, 2017.

[18] Ryan Lowe et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[19] Dag Malstaf, Pedro Gouveia, and Prateek Grover. Multi-agent learning in canonical games and dots-and-boxes. `https://github.com/DagMalstaf/ml-project-kul`, 2024. Project Team: Dag Malstaf (r0799028), Pedro Gouveia (r0980130), Prateek Grover (r0971804).

[20] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[21] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[22] Alvaro Ovalle Castañeda. Deep reinforcement learning variants of multi-agent learning algorithms, 2016.

[23] Liviu Panait and Karl Tuyls. Theoretical advantages of lenient q-learners: an evolutionary game theoretic perspective. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery.

[24] Yoav Shoham and Kevin Leyton-Brown. Multiagent systems: Algorithmic, game-theoretic, and logical foundations, 2009.

[25] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

[26] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[27] Michel Tokic. Adaptive ϵ-greedy exploration in reinforcement learning based on value differences. *Annual Conference on Artificial Intelligence*, pages 203–210, 2010.

[28] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[29] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[31] Michael Wunder, Michael Littman, and Monica Babes. Classes of multiagent q-learning dynamics with ϵ-greedy exploration. `https://icml.cc/Conferences/2010/papers/191.pdf`, 2010. RL3 Laboratory, Department of Computer Science, Rutgers University, Piscataway, NJ USA 08854.

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[33] Muhamet Yildiz. Game theory lecture notes. *Massachusetts Institute of Technology*, 2018.

[34] Hongming Zhang and Tianyang Yu. Alphazero. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 391–415, 2020.

[35] Kaiqing Zhang et al. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.