A Comparative Study of AI Agents for Connect-4

Tamojit Bera, Abhijit Gottumukkala

May 2025

Abstract

In this paper we train multiple algorithmic models to play the classic board game Connect 4 including the simplest greedy algorithm, minimax with alpha beta pruning with different heuristics, and Monte Carlo Tree Search. The agents played against each other to measure their comparative skill levels using their win rates against each other. It was found that all of the algorithms were able to beat a random strategy 100 percent of the time and from playing against each other we found that minimax with alpha beta pruning and the second heuristic we created had the highest win rate against all the other algorithms followed by the same algorithm using the first heuristic and lastly Monte Carlo Tree search and the greedy algorithm with the Monte Carlo Tree search being slightly better.

1 Introduction

Artificial Intelligence (AI) has been increasingly applied to classic board games as a means of exploring strategic decision-making, search algorithms, and heuristic evaluations. One such game is Connect 4, a two-player, zero-sum game played on a vertical 6-row by 7-column grid. The objective is for a player to align four of their colored discs consecutively in any direction: horizontal, vertical, or diagonal before their opponent does. Despite its relatively simple rules, the game exhibits substantial complexity, making it a compelling candidate for algorithmic exploration and optimization.

This project focuses on developing intelligent AI agents to play Connect 4 using the Minimax algorithm, a recursive decision-making strategy used for adversarial games. By evaluating possible game states assuming optimal play from both sides, Minimax enables an AI to determine the most favorable moves. However, due to the exponential growth of the game tree, naive implementations of Minimax quickly become computationally infeasible. To address this, we incorporate Alpha-Beta Pruning, a technique that reduces the number of game states that must be evaluated by eliminating subtrees that cannot influence the final decision.

In addition to Minimax, we implement two alternative strategies to broaden our exploration of AI methodologies. The first is a Monte Carlo Tree Search (MCTS) agent, which

balances exploration and exploitation through randomized simulations, making decisions based on statistically promising outcomes. The second is a greedy algorithm, which employs a fast heuristic evaluation function to make locally optimal decisions based on immediate board conditions, such as maximizing piece alignments or blocking the opponent.

We compare two heuristic evaluation functions to estimate the utility of non-terminal board states. These heuristics allow the AI to make informed decisions without exploring the full game tree, balancing accuracy and efficiency. Our work explores the design of these heuristics and their impact on gameplay performance, particularly in terms of search depth, win rate, and computational time.

We extend an existing open-source implementation of Connect 4 developed by Jake Oeding [1], which provides a basic framework for a two-player game. Building upon this, we introduce both non-intelligent and intelligent AI agents including Minimax, MCTS, and Greedy and evaluate their effectiveness through direct competition. Our objective is to explore the trade-offs between different strategies and to gain insights into how classical and stochastic AI techniques can be used to create competitive game-playing agents.

2 Related Works

2.1 Minimax and Alpha-Beta Pruning

The Minimax algorithm is a classical game tree search method used to choose optimal moves by simulating all possible future game states. In the context of Connect-4, it evaluates board positions recursively by assuming both players play optimally. However, the unoptimized version of Minimax becomes computationally expensive with increasing depth. Nasa et al. [7] noted that exploring the game state space up to a depth of 4 required 2799 iterations and approximately 33 milliseconds of computation time.

A study by Chaudhary et al. [2] presents a practical implementation of Connect-4 using both Minimax and Alpha-Beta pruning, demonstrating how pruning drastically reduces computation time and search iterations compared to the unoptimized Minimax algorithm. This optimization reduces the number of nodes evaluated in the game tree by pruning branches that won't affect the final decision because it assumes that the opponent will never choose unoptimal moves, and because of this, the algorithm can ignore and prune these unoptimal states. Alpha-beta pruning accomplishes this by introducing two bounds: α (the best value the maximizer can guarantee) and β (the best value the minimizer can guarantee). If at any point $\alpha \geq \beta$, further evaluation of that branch is halted, significantly speeding up the search.

2.2 Monte Carlo Tree Search (MCTS)

Dabas et al. [4] used Monte Carlo Tree Search, which is another AI strategy for playing Connect-4. Unlike Minimax, MCTS does not require a predefined evaluation function. In-

[1] https://github.com/jakeoeding/connect-4?tab=readme-ov-file

stead, it relies on random simulations and statistical sampling to evaluate possible moves. MCTS builds a search tree incrementally using the Upper Confidence Bound (UCB) formula to balance exploration and exploitation:

$$\text{UCB} = \frac{W_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

Here, W_i if the number of wins for child node i, n_i is the number of times node i has been visited, N is the number of visits to the parent node, and c is an exploration constant. In their study, Sheoran et al. [11] found that while MCTS outperformed Minimax and Double DQN under time-constrained settings, its performance suffered when compared to an optimized Minimax, due to its inherent randomness and exploration-heavy strategy.

2.3 Neural Networks and Reinforcement Learning

Schneider and Rosa [10] propose Neural Connect-4, an approach which employs a multilayer perceptron architecture that learns through the supervised backpropagation algorithm for intelligent gameplay in Connect-4. This architecture featured a single hidden layer with five different topologies - differing in the hidden neuron count from 84 to 672 - with the output consisting of seven neurons for the possible column choices. Networks with too few hidden neurons struggled to learn, while the networks with too many hidden neurons were slow and struggled to generalize. Best results were obtained from the mid-sized networks, which outperformed symbolic algorithms like minimax in head-to-head matches. Curran and O'Riordan [3] evolved Connect-4 neural agents using genetic algorithms and cultural learning.

Building on these neural foundations, Deep Q-Networks (DQN) represent a powerful evolution that incorporates reinforcement learning principles into the training. This approach uses two neural networks: a policy network for estimating Q-values (expected future rewards for actions) and another slowly updated target network, which is a copy of the policy network with its parameters updated less frequently, to stabilize learning. Dabas et al. [4] extended this model to use dual replay memory, which stores the experiences along with their mirror image, causing the dataset to double, called Double DQN (DDQN). Rather than learning from static datasets, DDQN agents learn by interacting with the game environment through self-play (300,000 self-play games). Through experience replay, where gameplay transitions are stored and randomly sampled during training, DDQN achieves stable convergence.

DDQN was shown to outperform both Minimax (even with Alpha-Beta pruning) and MCTS in various testing scenarios. It demonstrated superior generalization and adaptability, making it highly effective for automated Connect-4 agents [4].

In a conference paper, Faußer and Schwenker [5] discuss applying neural approximation of Monte-Carlo Policy evaluation, where they combine Monte-Carlo policy evaluation with a multilayer perceptron, resulting in achieving a lower computing time while still having a similar accuracy.

2.4 N-Tuples Features

Runarsson and Lucas [8] discuss using a linear function of n-tuple features as an evaluation function for the different game states, the values of which can then be utilized as the heuristic function of a tree search algorithm like minimax or Monte-Carlo tree search.

Runarsson and Lucas [9] previously applied the same n-tuple features approach to the board game Othello. They created a linear function of n-tuple features to approximate the preferred moves from a thousand tournament games from the French Othello league, and the resulting evaluation function was able to perform well against functions generated from the algorithm playing against itself. Thill, Koch, and Konen [12] also apply the n-tuples method to connect-4 in their conference paper in a similar manner.

2.5 Quantified Boolean Formula Encoding

Gent and Rowley [6] propose encoding Connect-4 as a Quantified Boolean Formula (QBF), framing the game as a logic-based adversarial problem. Moves by the two players are modeled using alternating existential and universal quantifiers. Their encoding for the standard 7×6 board uses over 18,000 variables and 70,000 clauses.

To prevent rule violations (e.g., placing multiple pieces), they introduce *cheat variables* and *gameover indicators* that enforce game rules within the logic. Though this QBF formulation provides a theoretical foundation for automated reasoning over Connect-4, practical solving remains challenging at full scale. Other approaches, such as strategic rule-based programs like VICTOR [1], combine knowledge-driven reasoning with search, achieving a full solution for Connect-4 with less brute-force computation.

3 Methodology

3.1 Greedy AI

Our first approach to create a computer player for Connect 4 is to implement the greedy algorithm where we would evaluate each of the immediate possible game states and apply an evaluation algorithm to select the most favorable game state. We calculated the favorability of a state using a score function where:

- If the agent gets 4 discs in a row: 1000 points
- If the agent gets 3 discs in a row: 50 point
- If the agent gets 2 discs in a row: 5 point
- If the opponent gets 4 discs in a row: -1000 points.
- If the opponent gets 3 discs in a row: -80 points

3.2 Minimax and alpha beta pruning

Our next approach was to use the minimax algorithm with the alpha-beta pruning optimization and test several different heuristics with the minimax algorithm. Minimax is a decision-making algorithm used in adversarial turn-based games. The way it works is that all of the possible game states are explored in a tree where each level of the tree represents the next turn in the game and each of the ending states is given a score. For example, 1 if player 1 wins, -1 if player 2 wins, and 0 if it is a draw. The algorithm then propagates these scores up the tree where if it is player 1's turn the state with the maximum score is chosen and if it is player 2's turn the state with the minimum score is chosen. In the end, this creates a path where the players make the best possible decisions to attempt to reach their win states.

One optimization for the minimax algorithm is alpha-beta pruning, which works by discarding branches that do not affect the final decision because, assuming the players are playing optimally, they would never choose a game state with a worse score than one they have already found, and so the algorithm can ignore the game states down this branch because they should never be entered.

Because connect-4 has a large number of possible game states, it is infeasible for us to use minimax to explore the entire state space. So instead we must use heuristics to provide estimates for the scores of the furthest game states we choose to explore and then apply minimax from there.

Our first heuristic was modeled after the description from "Solving Connect 4 using Artificial Intelligence" by Dabas [4]. We use their conditions:

- If the agent gets 4 discs in a row: 1000000 points
- If the agent gets 3 discs in a row: 1 point
- If the opponent gets 3 discs in a row: -100 points
- If the opponent gets 4 discs in a row: -10000 points.

to assign a score to the game states at the given max depth and then apply minimax with alpha-beta pruning to calculate the previous game states. With these conditions, we give the most weight to the agent getting the win condition, a very small weight to getting close to a winning state, and then a penalty for the opponent getting close to a winning state and a greater penalty for them actually winning.

Our second heuristic is similar to the first, except with slightly more tuned values:

- If the agent gets 4 discs in a row: 10000000 points
- If the agent gets 3 discs in a row: 100 point
- If the agent gets 2 discs in a row: 10 point
- If the opponent gets 4 discs in a row: -1000000 points.

- If the opponent gets 3 discs in a row: -150 points
- If the opponent gets 2 discs in a row: -20 points

Here we gave slightly more weight to the agent getting three in a row and added score for getting two in a row, and similarity increased the penalty to states with the opponent having two disks in a row and added a small penalty for anywhere the opponent has two disks in a row.

For the third heuristic we ended up trying the same score function as from our initial greedy algorithm; we decided to try rescaling the score values so that the score for getting 2 and three disks in a row was slightly more in relation to getting 4 in a row compared to heuristics 1 and 2, and we also made it so that the amount of score lost for the opponent winning was equal to the amount of score given for the agent winning and slightly increased the amount of score lost for the opponent getting 3 in a row in an attempt to prompt a somewhat more defensive play style.

3.3 Monte Carlo Tree Search (MCTS)

We also decided to implement Monte Carlo Tree Search, which works by randomly searching game states for a set amount of time and then picking the best option out of the states that were evaluated using the UCT (Upper Confidence Bound for Trees) formula.

$$\mathrm{UCT}(i) = \frac{w_i}{n_i} + c \cdot \sqrt{\frac{\ln N}{n_i}}$$

4 Evaluation

To evaluate the relative performance of various AI agents for Connect 4, we designed a head-to-head simulation framework in Python. All AI agents were initially tested against a random agent across 100 games to observe their individual performance. Each AI agent was then pitted against others across a series of 100 games per matchup, and their win rates and average move times were recorded. This setup enabled us to draw statistically meaningful conclusions about strategic strength and computational efficiency.

4.1 Experimental Setup

The experiments involved 4 AI agents:

- GreedyAI
- Monte Carlo Tree Search AI (MCTS)
- Minimax AI with two different Heuristics

Each AI pair played 100 games, alternating roles every game - AI 1 started first in even-numbered games, and AI 2 started first in odd-numbered ones. This mitigated first-move bias and allowed for a fairer evaluation of the AI players. For each game, we recorded the outcome (win/loss/draw) and the time taken per move by each agent. The average move time was computed as the total time spent making moves divided by the number of moves executed by the AI. All experiments were run on the same machine to ensure timing consistency.

4.2 Avoiding Repetitive Game Trajectories

A key challenge encountered was that some of the algorithms, especially those based on deterministic strategies like Minimax and GreedyAI, consistently produced the same sequence of moves when placed in identical configurations. As a result, repeated games yielded identical outcomes, leading to poor coverage of the state space, little diversity in the game trajectories, and skewed statistical analysis. To address this, two randomization strategies were introduced:

1. Randomized Opening Moves: We forced each AI to make a random legal move on their first turn. Since the board is 7 columns wide and the first two moves are guaranteed to be valid in any column, this introduced 49 distinct possible opening configurations.

2. Stochastic Tie-Breaking: In states where multiple moves were tied for the best score, we introduced a randomized selection among these best moves, adding an additional layer of non-determinism without compromizing strategic integrity.

These adjustments ensured that the games played across the 100-match simulations were sufficiently diverse, leading to a more robust state space exploration.

4.3 Results

Before conducting pairwise matchups, each agent was tested against a RandomAI across 100 games to assess basic competency:

- GreedyAI achieved a 98% win rate, occasionally losing due to lack of long term foresight.

- MCTSAI, MinimaxAI-H1, and MinimaxAI-H2 recorded 100% win rate, demonstrating significant strategic competence in the long term.

These results showed that all these AI players are significantly stronger than a random strategy and are suitable for competitive benchmarking.

Following initial testing, all AI players were matched against one another in sets of 100 games. The results of the matchups are shown in Table 1.

AI 1	AI 2	Win Percentage (%)			Avg. Move Time (ms)	
		AI 1	AI 2	Draws	AI 1	AI 2
GreedyAI	MCTSAI	49	51	0	0.0	218.8
GreedyAI	MinimaxAI-H1	27	68	5	0.5	556.4
GreedyAI	MinimaxAI-H2	9	90	1	0.4	522.7
MCTSAI	MinimaxAI-H1	17	71	12	232.9	486.6
MCTSAI	MinimaxAI-H2	6	94	0	228.5	952.9
MinimaxAI-H1	MinimaxAI-H2	8	71	21	525.3	765.4

Table 1: Head-to-head matchup results

5 Analysis

Table 1 shows the results of the pairwise matchups between the AI agents. These results highlight clear strategic and computational differences between the four AI agents, driven by the underlying algorithms and heuristics used.

5.1 Greedy AI

GreedyAI, while highly efficient with near-zero move times, was strategically inferior in all matchups except its near-even result with MCTSAI. It has a lightweight algorithm that scores immediate board positions, rewarding configurations like setups and blocking immediate threats, without simulating opponent responses. Its 49% win rate against MCTS is likely due to the inherent randomness of the MCTS algorithm. Its fast performance is due to a lack of deeper planning as it cannot anticipate long-term positioning, making it susceptible to more tactical plays, as seen in its low win rates against other smarter agents.

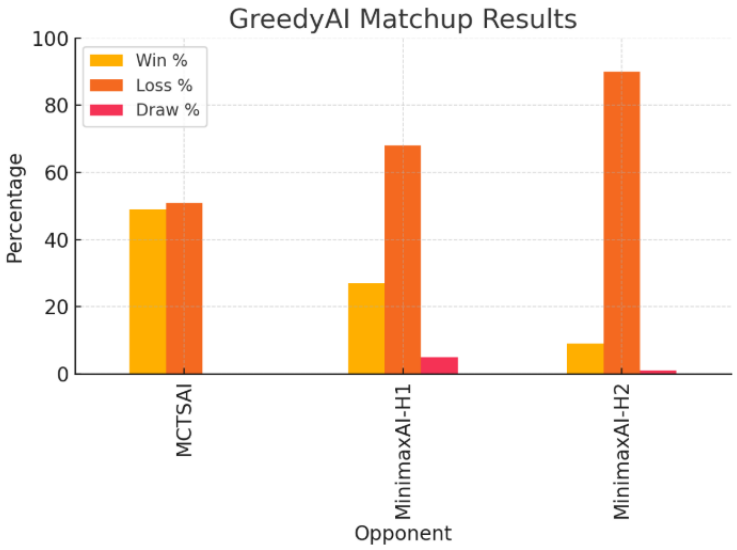

Figure 1: GreedyAI Matchup Results

5.2 Monte Carlo Tree Search (MCTS) AI

The MCTS AI conducts simulations for a fixed time (0.25 seconds) per move, using random playouts to estimate move quality. While it lacks a crafted evaluation function, it excels at handling stochastic or unknown environments. MCTSAI has a slightly superior performance compared to GreedyAI and it offers a computationally cheaper alternative to the Minimax variants, with performance suffering only slightly for reduced time budgets. However, its inability to reason about threats or guarantees an make tactical plays in adversarial settings results in significant losses againt both the minimax agents.

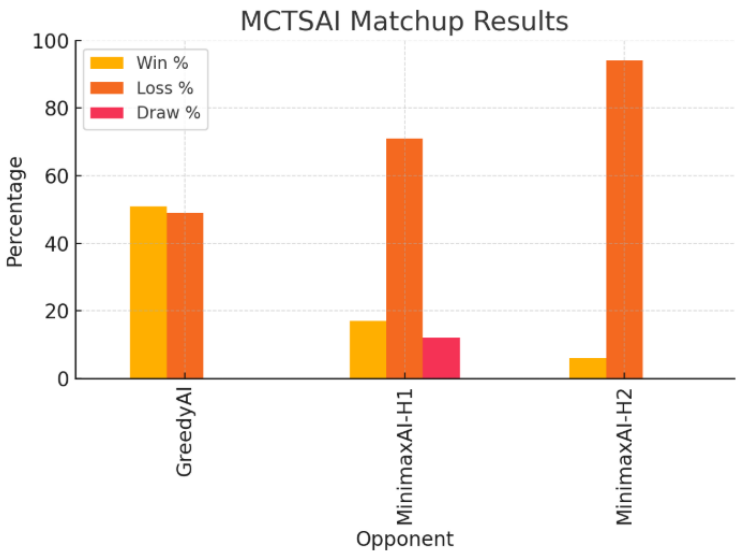

Figure 2: MCTSAI Matchup Results

5.3 Minimax AI with Heuristic 1

MinimaxAI-H1 uses a depth-limited minimax algorithm with alpha-beta pruning. It performs deeper searches and reasons through sequence of moves, giving it a strong advantage over short sighted agents like GreedyAI (68% wins). Despite a lack of nuance in the heuristic, which prioritizes terminal win conditions and lightly scores intermediate opportunities and threats, its logical structure and depth-based foresight allowed it to strongly outperform MCTSAI (71% wins), demonstrating how even a basic implementation of minimax can outperform probabilistic methods when determinism and foresight are advantageous. Its moderate average move time ($\approx$ 525 ms) suggests a balance between quality and efficiency, making it a lightweight alternative to H2, while still being relatively competitive.

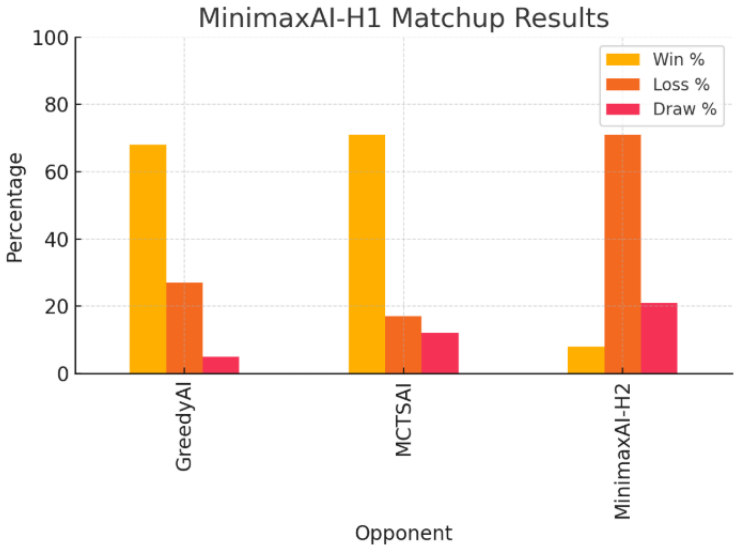

Figure 3: MinimaxAI-H1 Matchup Results

5.4 Minimax AI with Heuristic 2

MinimaxAI-H2 came out at the top as the strongest agent, with a win rate of over 90% against GreedyAI and MCTSAI, and 71% against MinimaxAI-H1 with about 20% draw. It uses a more refined and aggressive heuristic that highly rewards offensive configurations and strongly penalizes opponent threats. This balance of offense and defense gives it a strategic edge, as seen in its consistent dominance among the four agents. It also had the highest computation time ($\approx$ 950 ms), a direct result of the increased heuristic complexity. Its consistent win rate over H1, however, confirms that better heuristics can significantly improve the performance of minimax, even at the same depth.

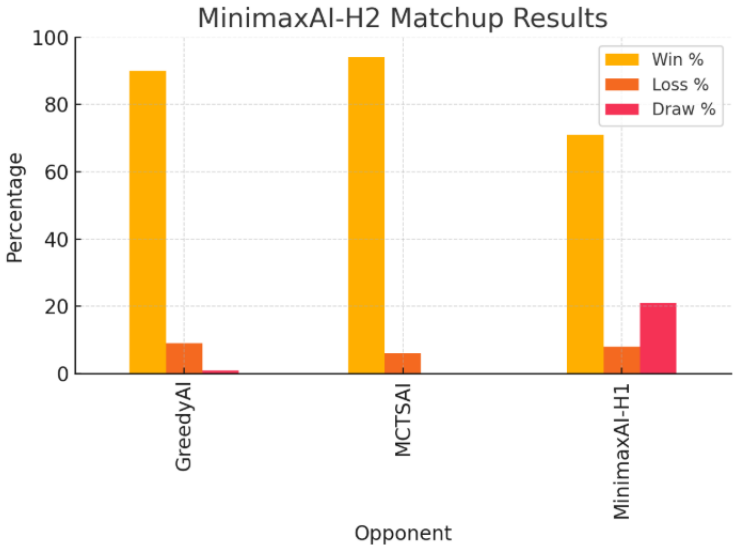

Figure 4: MinimaxAI-H2 Matchup Results

6 Conclusion

Our project focused on developing intelligent AI agents to play Connect 4 using a range of algorithmic strategies: a Greedy heuristic-based approach, Monte Carlo Tree Search (MCTS), and Minimax with different heuristics. The goal was to investigate how variations in evaluation heuristics, search algorithms, and computational costs influence gameplay performance.

Our experiments showed a clear hierarchy among the agents. MinimaxAI-H2, which used a finely tuned heuristic emphasizing both offensive and defensive play, consistently outperformed all others. MinimaxAI-H1, with a simpler heuristic, performed respectably but fell short of H2. MCTSAI offered moderate results with acceptable move times but struggled against deterministic agents. GreedyAI, while extremely fast, lacked strategic foresight and underperformed in most matchups. We also experimented with a third heuristic (H3), which adapted the Greedy evaluation function for use in Minimax. Preliminary results showed it performed significantly better than H1 but was not as good as H2.

For future exploration, we aim to benchmark H3 more thoroughly and investigate how varying the search depth in Minimax affects performance and runtime. This will help determine whether deeper searches lead to linear or diminishing improvements in decision quality.

The code for all the AI agents and experiments presented in this paper is available at: https://github.com/phantom660/connect-4

References

[1] Louis Victor Allis. "A knowledge-based approach of connect-four". In: *J. Int. Comput. Games Assoc.* 11.4 (1988), p. 165.

[2] Shivam Chaudhary, Siddhant Pandey, and Ms Heena Khera. "Alpha-Beta Pruning in Mini-Max Algorithm–An Optimized Approach for a Connect-4 Game". In: ().

[3] Dara Curran and Colm O'RIORDAN. *Evolving Connect-Four Playing Neural Networks Using Cultural Learning*. Tech. rep. Technical Report NUIG-IT-081204, National University of Ireland, Galway, Ireland, 2004.

[4] Mayank Dabas, Nishthavan Dahiya, and Pratish Pushparaj. "Solving connect 4 using artificial intelligence". In: *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2021, Volume 1*. Springer. 2022, pp. 727–735.

[5] Stefan Faußer and Friedhelm Schwenker. "Neural Approximation of Monte Carlo Policy Evaluation Deployed in Connect Four". In: *Artificial Neural Networks in Pattern Recognition*. Ed. by Lionel Prevost, Simone Marinai, and Friedhelm Schwenker. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 90–100. ISBN: 978-3-540-69939-2.

[6] Ian P Gent and Andrew G Rowley. "Encoding Connect-4 using quantified Boolean formulae". In: *2nd Intl. Work. Modelling and Reform. CSP* (2003), pp. 78–93.

[7] Rijul Nasa et al. "Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game". In: *Int. Res. J. Eng. Technol* 5.4 (2018), pp. 1637–1641.

[8] Thomas Philip Runarsson and Simon M Lucas. "On imitating Connect-4 game trajectories using an approximate n-tuple evaluation function". In: *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2015, pp. 208–213.

[9] Thomas Philip Runarsson and Simon M Lucas. "Preference learning for move prediction and evaluation function approximation in Othello". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.3 (2014), pp. 300–313.

[10] Marvin Oliver Schneider and JL Garcia Rosa. "Neural connect 4-A connectionist approach to the game". In: *VII Brazilian Symposium on Neural Networks, 2002. SBRN 2002. Proceedings.* IEEE. 2002, pp. 236–241.

[11] Kavita Sheoran et al. "Solving connect 4 using optimized minimax and monte carlo tree search". In: *Mili Publications. Advances and Applications in Mathematical Sciences* 21 (2022), pp. 3303–3313.

[12] Markus Thill, Patrick Koch, and Wolfgang Konen. "Reinforcement learning with n-tuples on the game Connect-4". In: *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12*. Springer. 2012, pp. 184–194.