Self-Play Reinforcement Learning Without Tree Search in Connect Four

Umer Raja
Department of Computer Science
LUMS
Lahore, Pakistan
26100063@lums.edu.pk

Haris Rathore
Department of Computer Science
LUMS
Lahore, Pakistan
26100064@lums.edu.pk

Abstract

The remarkable success of AlphaZero demonstrated that combining deep neural networks with Monte Carlo Tree Search (MCTS) yields superhuman performance in complex board games. However, the dependence on large-scale search introduces substantial computational overhead, limiting applicability in resource-restricted environments. This paper explores an alternative approach: training a model-free agent via self-play without explicit lookahead search. We focus on Connect Four, modeling it as an Egocentric Markov Decision Process (E-MDP). To overcome the instability inherent in non-stationary self-play, specifically the emergence of reactive strategy bias and cycling, we implement Fictitious Self-Play (FSP) using a history pool of past policies. Our results demonstrate that a Proximal Policy Optimization (PPO) agent, stabilized by FSP and action masking, can achieve robust strategic performance, consistently defeating baseline opponents and showing monotonic improvement against its own historical versions.

1 Introduction

The mastery of combinatorial board games has long been a benchmark for Artificial Intelligence. The landmark success of AlphaZero **?** demonstrated that a generalized reinforcement learning (RL) algorithm could achieve superhuman performance across Chess, Shogi, and Go without domain-specific heuristics. However, AlphaZero and its successors rely heavily on Monte Carlo Tree Search (MCTS), a lookahead planning algorithm, to guide the neural network's training and inference. While effective, MCTS introduces significant computational overhead, rendering such agents unsuitable for real-time applications or resource-constrained environments where decisions must be made instantly based on "intuition" (policy) rather than exhaustive calculation.

This project explores the capabilities of model-free Deep Reinforcement Learning in the absence of explicit tree search. We focus on Connect Four, a zero-sum, perfect-information game with a state-space complexity of approximately 4.5×10^{12} positions. While Connect Four is theoretically solved **?**, it remains a challenging testbed for learning tactical, spatial strategies purely from self-interaction.

Training agents via self-play without a search-based supervisor presents unique challenges. First, the learning environment is non-stationary; as the agent improves, the opponent (itself) also changes, creating a "moving target" for the optimization landscape. Second, naive self-play often suffers from cycling (e.g., Rock-Paper-Scissors dynamics), where an agent learns to exploit specific weaknesses of its current policy but forgets how to defend against earlier, simpler strategies. In our initial experiments, we also observed a reactive strategy bias, where the agent converged to a passive, defensive local optimum, allowing the second player (Player 2) to dominate by simply exploiting the first player's over-extensions.

Preprint. Under review.

To address these instabilities, we propose a robust training pipeline using Proximal Policy Optimization (PPO) combined with Fictitious Self-Play (FSP). Our approach employs an Egocentric Markov Decision Process (E-MDP), enabling a single neural network to generalize across both player positions by transforming the board state relative to the current player's perspective. Furthermore, we mitigate cycling and reactive bias by maintaining a history pool of past policy versions. By training the agent against a mixture of its latest self and historical snapshots, we force the emergence of robust, proactive strategies that generalize across varying skill levels.

2 Literature Review

Recent research has started to directly tackle Connect Four using deep RL algorithms without incorporating tree search, to see how close self-play alone can come to expert play. Several works and experiments have applied popular deep RL methods – such as DQN, policy gradients, and actor-critic algorithms – to train Connect Four agents via self-play. Here we review some notable findings.

2.1 Deep RL vs Search: Experimental Comparisons

A study by Dabas et al. (2022) trained three different agents to play Connect Four, using (1) a minimax algorithm with alpha-beta pruning, (2) Monte Carlo Tree Search, and (3) a Deep Q-learning agent (specifically, a Double DQN) in a self-play regime. In their setup, each agent was given comparable computational resources (time or simulations) for making decisions. The results of head-to-head matches were illuminating: the MCTS agent emerged as the strongest player overall, consistently beating the deep Q-learning agent; the deep Q agent in turn won consistently against the minimax agent under those conditions[10]. In fact, the minimax agent was ranked last in that experiment – a somewhat surprising outcome, since perfect-play minimax should never lose. The caveat was that the minimax agent was intentionally limited in search depth for fairness (it "was not able to explore higher depth in the game tree" given the time limits)[10]. This allowed the learning-based agents to exploit the shallower search. The Double DQN agent improved through self-play and eventually outperformed the constrained minimax player, demonstrating that a neural network can learn effective Connect Four tactics that beat a non-optimized search. Nevertheless, the MCTS agent was strongest, defeating the DQN agent in the majority of games[10][13]. This hierarchy (MCTS > Deep RL > limited-depth Minimax) highlights that with equal computational budgets, a learning agent can overtake a weaker search, but a well-informed search (like MCTS with many simulations) still held an advantage in that study.

2.2 Self-Play RL Performance

Another comprehensive comparison is provided by Taylor and Stella (2024), who evaluated a range of algorithms on Connect Four, including a Q-learning agent (model-free RL), a pure MCTS player, and a classical minimax player, along with "advanced" variants of each[14]. Their findings showed that, under consistent conditions, the MCTS-based agent achieved the highest win percentage, followed by the minimax (alpha-beta) agent, and lastly the model-free RL agent[15]. The Q-learning agent (augmented with function approximation) learned to play reasonably well, but it did not reach the level of the search-based methods. One positive note for the RL approach was efficiency: the learned agent made decisions almost instantly (a forward pass in the network), whereas MCTS and minimax consumed more time per move. Indeed, Taylor Stella report that the Q-learning agent was the fastest to move, whereas MCTS and minimax, while stronger, were slower due to simulation and search overhead[16]. This underscores a key trade-off: the RL agent offers computational simplicity at runtime, though its skill fell short of the best search-based play. It's worth noting that Taylor & Stella also cite results from Scheiermann & Konen (2022) for a hybrid AlphaZero-like approach: an agent that uses an RL-trained network plus an MCTS wrapper. In Connect Four and other games, that hybrid approach was found to outperform all individual methods – for example, an AlphaZero-inspired agent (neural network + MCTS) handily beat both a pure RL agent and a pure search agent in head-to-head play[9]. In one case, a basic MCTS with 10k playouts had a 100% win rate against a stand-alone RL agent, while the network-assisted MCTS (AlphaZero style) had a 99% win rate against basic MCTS[9]. This again highlights that combining learned networks with search can yield superior performance, but the pure RL agent on its own is the weakest of the bunch.

2.3 Improving RL Agents: Reward Shaping

A core challenge in pure RL for games like Connect Four is the sparsity of rewards: the agent only gets a clear positive or negative signal at game end (win or loss), and games can involve dozens of moves. To address this, researchers have explored reward shaping, providing intermediate feedback to guide the learning process. For example, one can assign small rewards for intermediate achievements such as getting two in a row or three in a row, or for occupying the center column, etc., which are heuristically known to correlate with winning chances[20]. As long as these shaped rewards are potential-based (derived from a potential function on states), they theoretically should not alter the optimal policy, only accelerate learning. Taylor & Stella (2024) mention a shaped reward function used in literature: e.g. +1 for each own piece in a row, +2 for two in a row, +6 for three in a row, with a large terminal reward for a win (+40) and negative for a loss (30), among other small bonuses[20]. By providing this richer feedback signal, the RL agent can learn important tactical motifs (like “3 in a row is good”) faster than it would from win/loss alone. Indeed, prior work found that such shaping, combined with decaying exploration (epsilon-greedy) and learning rates, was effective in training Q-learning agents for Connect Four[21]. Reward shaping must be designed carefully to avoid misleading the agent – ideally, it should reflect actual progress toward winning. Empirically, shaped rewards and other tricks (like curriculum learning or self-play against progressively stronger opponents) have been key to achieving reasonable training times in these domains.

2.4 Deep Q-Network (DQN)

This value-based method learns a Q-value for each state-action pair, using deep neural networks to generalize across the huge state space. In a Connect Four context, a DQN would take a board position as input and output Q-values for the 7 possible column moves. Training is typically done via self-play games: the agent uses an ϵ-greedy policy (mostly selecting the move with highest Q-value, but occasionally exploring a random move) to play both sides, and updates its Q estimates from the game outcomes. DQN has to contend with a non-stationary environment (since the “opponent” is the agent itself, continually improving). Techniques like experience replay can help stabilize training by learning from a buffer of past self-play experiences, and using separate target networks can mitigate oscillations. In practice, extensions like Double DQN are used to avoid overestimating Q-values. DQN-based agents have shown the ability to learn decent Connect Four strategies[22], but they may converge slowly and can get stuck in suboptimal policies if exploration is insufficient or if the reward is too sparse. Additionally, standard Q-learning does not explicitly account for the opponent’s optimal responses, whereas in a two-player zero-sum game an ideal agent should anticipate best play from the opponent (we return to this point under self-play dynamics below).

2.5 Policy Gradient (REINFORCE)

Policy-based methods learn a direct mapping from states to a probability distribution over moves, without explicitly estimating values for each state. A simple approach is REINFORCE, which adjusts the policy in the direction that increases the probability of moves that led to wins (and decreases moves leading to losses), using the game outcome as a reward signal. In Connect Four, a policy network (often a convolutional neural network given the grid input) can output a vector of 7 move probabilities. Self-play training with REINFORCE will gradually favor moves that are part of winning sequences. The advantage of policy gradients is that they can directly optimize for the final objective (winning games) and can naturally handle stochastic policies. However, REINFORCE is known for high variance in its updates – a single game’s result assigns credit or blame to many moves, and it can take a long time to discern which specific moves truly contributed to victory or defeat. In practice, baseline subtraction (using a value function estimate to normalize rewards) is used to reduce variance. Pure policy gradient methods can learn strong strategies but often require a lot of training samples; they might be less sample-efficient than value-based or actor-critic methods for a game like Connect Four.

2.6 Actor-Critic and PPO

Actor-critic methods combine the ideas of policy learning and value learning. An actor-critic agent for Connect Four would have a policy network (actor) proposing moves and a value network (critic) estimating the expected outcome from a given position. During training, the critic’s evaluation is used to compute an advantage for the moves actually taken (how much better was that move than the expected value), and the policy (actor) is updated to favor moves with positive advantage. This approach tends to be more stable and efficient than REINFORCE because the critic provides a dense learning signal at each move. Modern actor-critic algorithms like Proximal Policy Optimization (PPO) have become popular due to their stability. PPO uses a clipped surrogate objective to ensure that policy updates do not destabilize the policy too much in a single step. In self-play settings, PPO has been successfully used for very complex games – notably, OpenAI Five used a multi-agent PPO variant for Dota 2, and achieved superhuman results through self-play. For Connect Four, PPO (or related actor-critic methods) is a promising choice to experiment with, as it can efficiently handle continuous learning in a competitive environment. Its on-policy nature means it learns from the distribution of states it currently encounters; as the agent improves, it focuses on the more relevant parts of the state space. One must be careful with on-policy methods in self-play – if the agent changes too quickly, it might forget how to deal with past strategies (a phenomenon akin to catastrophic forgetting). In practice, techniques like playing occasionally against older versions of the agent or maintaining a population of agents can mitigate this. Overall, actor-critic methods strike a good balance between the unbiased but high-variance policy gradient and the lower-variance value-based approach.

3 Problem Formulation

We model the game of Connect Four as a finite, two-player, zero-sum Markov Game, formally defined by the tuple $\mathcal{G} = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$.

3.1 Game Dynamics

- **Players ($\mathcal{N}$):** The set of players $\mathcal{N} = \{1, 2\}$.
- **State Space ($\mathcal{S}$):** The game is played on a grid with $H = 6$ rows and $W = 7$ columns. A raw state S_{raw} is represented as a tensor $B \in \{-1, 0, 1\}^{H \times W}$, where 1 represents Player 1's pieces, -1 represents Player 2's pieces, and 0 represents empty cells.
- **Action Space ($\mathcal{A}$):** The action space is discrete, corresponding to the column indices $\mathcal{A} = \{0, 1, \ldots, 6\}$. An action $a \in \mathcal{A}$ is valid in state s if and only if the chosen column is not full (height $< H$). We denote the set of legal actions as $\mathcal{A}_{legal}(s)$.
- **Transition ($\mathcal{T}$):** The transition function $\mathcal{T}(s, a) \to s'$ is deterministic. Upon selecting column a, the piece is placed in the lowest available row r in that column. The turn then passes to the opponent.

3.2 Egocentric MDP Formulation

Standard multi-agent RL often trains separate policies for each player. To enable efficient Self-Play using a single set of parameters θ, we formulate the problem as an Egocentric MDP.

We define an observation transformation function $\phi(S_{raw}, p)$ that orients the board relative to the current player p:

$$\phi(S_{raw}, p) = \begin{cases} S_{raw} & \text{if } p = 1 \\ -S_{raw} & \text{if } p = 2 \end{cases}$$

Under this transformation, the neural network input is always normalized such that the "Agent's Pieces" are represented by positive values (Channel 0) and "Opponent's Pieces" by negative values (Channel 1).

3.3 Reward Structure

We utilize a hybrid reward structure designed to accelerate convergence by combining sparse terminal signals with dense tactical incentives. The base reward signal r_{base} is defined as:

$$r_{base} = \begin{cases} +1 & \text{if the current agent wins} \\ 0 & \text{if the agent loses, draws, or the state is non-terminal} \end{cases}$$

To address the credit assignment problem inherent in sparse-reward settings, we incorporate heuristic reward shaping (PPO+Dense). We define auxiliary rewards r_{dense} for specific tactical maneuvers:

- **Blocking Reward** $(+0.1)$**:** Granted when the agent places a piece that directly blocks an opponent's winning move (preventing an immediate loss).
- **Threat Creation** $(+0.05)$**:** Granted when the agent forms a "threat" configuration (timmediate possible win), incentivizing proactive play.

The total reward at timestep t is the sum $r_t = r_{base} + r_{dense}$. While the zero-sum nature is relaxed in the immediate reward function, the adversarial training loop ensures agents implicitly minimize the opponent's return.

3.4 Optimization Objective

Our goal is to learn a stochastic policy $\pi_\theta(a|s)$ that approximates the Nash Equilibrium strategy. The objective function is to maximize the expected cumulative discounted reward $J(\pi_\theta)$:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

where trajectory τ is generated by playing against an opponent policy π_{opp} drawn from the agent's history.

4 Methodology

We employ a shared Convolutional Neural Network (CNN) architecture optimized via PPO and stabilized by Fictitious Self-Play.

4.1 Neural Network Architecture

- **Input:** $(3, 6, 7)$ tensor. Channel 0: Player pieces; Channel 1: Opponent pieces; Channel 2: Broadcasted legal move mask.
- **Backbone:** Three convolutional layers (32, 64, 64 filters) with 3×3 kernels and ReLU activations.
- **Heads:** A shared 512-unit dense layer splits into:
  - **Actor** (π_θ)**:** Softmax over 7 columns.
  - **Critic** (V_ϕ)**:** Linear output estimating expected return (win probability).

4.2 Proximal Policy Optimization (PPO)

We utilize PPO with two domain-specific modifications:

1. **Action Masking:** Logits of illegal columns are set to $-\infty$ before Softmax, ensuring $\pi_\theta(a_{illegal}|s) = 0$.
2. **Zero-Sum GAE:** We modify Generalized Advantage Estimation (GAE) to account for perspective flips:
$$\delta_t = r_t + \gamma(-V(s_{t+1})) - V(s_t)$$

4.3 Fictitious Self-Play (FSP)

To mitigate reactive bias, we maintain a history pool $\mathcal{P}$ of size $N = 20$. During training, the opponent is sampled via:

- 80%: Latest Policy ($\pi_{current}$).
- 20%: Historical Policy ($\pi_{old} \sim$ Uniform($\mathcal{P}$)).

Crucially, we apply a gradient mask during optimization. The agent updates its policy only based on its own actions in response to historical opponents, treating the historical moves as part of the environment dynamics.

5 Experiments

5.1 Experimental Setup

We trained the agent for 2 million timesteps using 16 parallel environments. Key hyperparameters included a learning rate of 2.5×10^{-4}, entropy coefficient of 0.03, and batch size of 8192 (512 steps $\times$ 16 envs).

5.2 Convergence Behavior

Initial experiments with pure self-play (Latest vs Latest) with a low entropy coefficient of 0.01 resulted in a strong bias towards Player 2, with P2 achieving win rates $\approx 80\%$. We hypothesize this occurred because reactive blocking strategies are easier to learn than proactive trap construction. Upon increasing the entropy coefficient to 0.03, we observed stabilized learning. The Critic loss converged to ≈ 0.1, indicating accurate value estimation, while the Actor loss exhibited a U-shaped curve, reflecting the transition from learning basic rules to optimizing against a skilled adversary.

6 Results

We evaluated three variations of PPO-based agents trained via self-play on the Connect Four environment. These include: (1) Vanilla PPO (baseline), (2) PPO+FSP, which incorporates a policy history for training stability, and (3) PPO+Dense, which uses an augmented reward signal to encourage strategic positioning.

6.1 Win Rate Evaluation

Each agent was evaluated against the baseline, an agent which selects a legal move uniformly at random.

Across 1000 evaluation games per agent:

- Vanilla PPO achieved a 93.5% win rate against Random.
- PPO+FSP achieved 88.5% vs Random.
- PPO+Dense achieved 95% vs Random.

Player 1 had a measurable advantage in all evaluations, winning approximately 58–64% of games on average, with the PPO+FSP variant showing the smallest gap between players, suggesting improved parity and resilience when playing second.

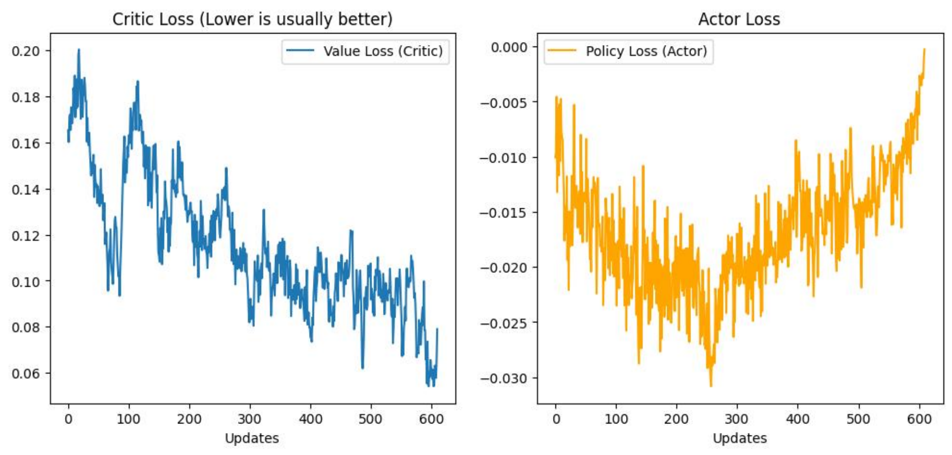
6.2 Performance Summary

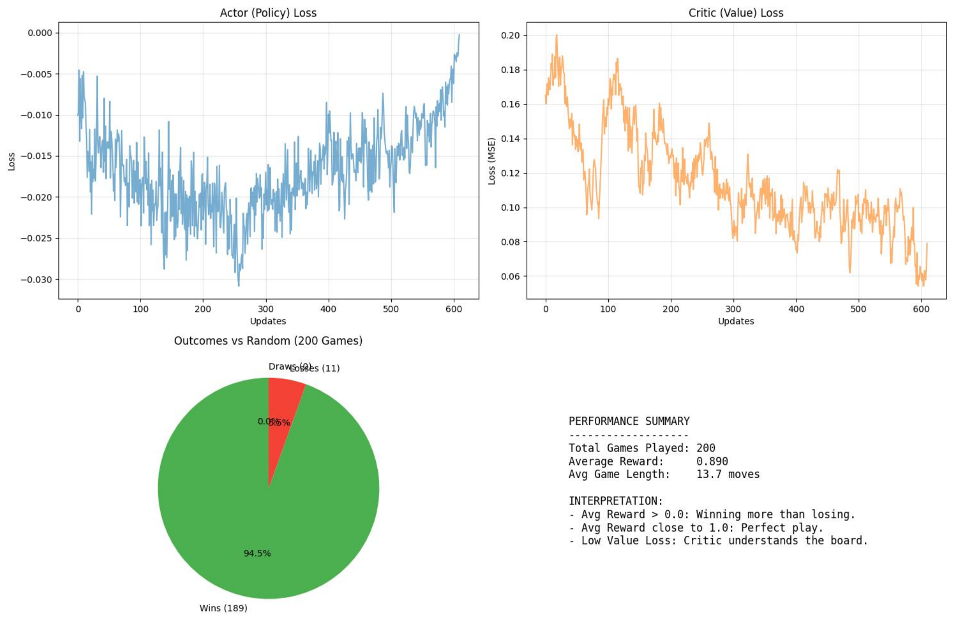
Table 1: Performance of PPO variants across key benchmarks (win % over 1000 games)

Agent	**vs Random**
Vanilla PPO	93.5%
PPO+FSP	88.5%
PPO+Dense	95.0%

6.3 Figures

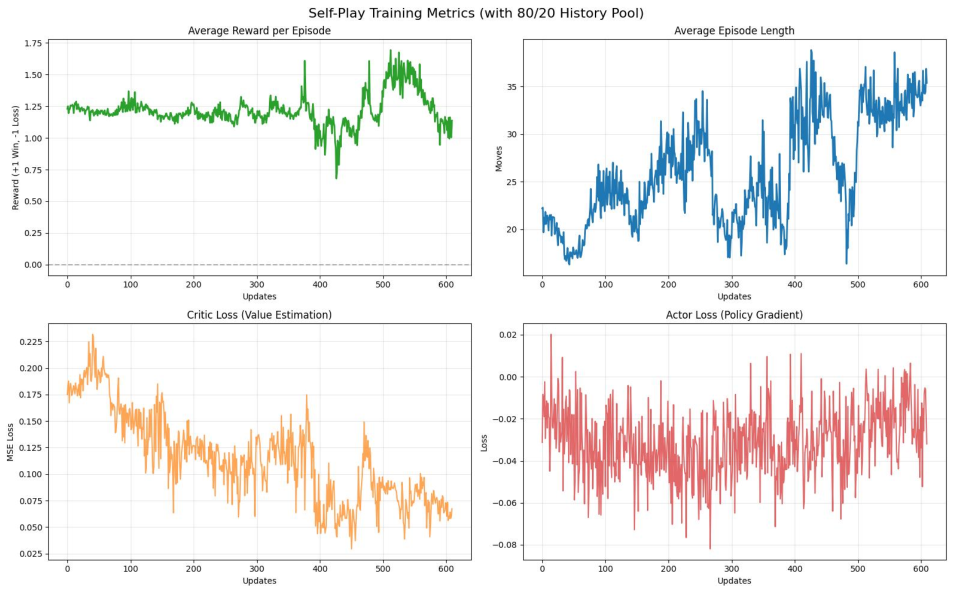
To support these findings, we include below the following figures:





```
PERFORMANCE SUMMARY
-------------------
Total Games Played: 200
Average Reward:     0.890
Avg Game Length:    13.7 moves

INTERPRETATION:
- Avg Reward > 0.0: Winning more than losing.
- Avg Reward close to 1.0: Perfect play.
- Low Value Loss: Critic understands the board.
```



7 Conclusion

This work demonstrates that model-free Reinforcement Learning, when stabilized with Fictitious Self-Play and Egocentric state formulation, can master Connect Four without MCTS. Future work will extend this approach to more complex variants, such as 7×8 grids or Connect-5, to test the scalability of search-free policy learning.

Code and Resources

The complete source code for all experiments, environment setup, model implementations, training pipelines, and evaluation scripts is available on GitHub:

`https://github.com/UmerSR/Connect-Four-RL`

The repository includes setup instructions, pretrained models, and logs to facilitate reproducibility of the results reported in this paper.

Acknowledgments and Disclosure of Funding

We acknowledge the use of the Gymnasium library for environment simulation and PyTorch for neural network implementation. This project was conducted as part of the CS 6314 course requirements.

References

[1] Silver, D., et al. (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 1140-1144.

[2] Schulman, J., et al. (2017) Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

[3] Heinrich, J., & Lanctot, M. (2016) Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. *arXiv preprint arXiv:1603.01121*.

[4] Edelkamp, S., & Kissmann, P. (2008) Symbolic classification of general two-player games. *KI 2008: Advances in Artificial Intelligence*, 185-192.

[5] Allen, J. D. (1988). *Solving Connect Four using Knowledge-Based Methods*. MSc thesis.

[6] Allis, L. V. (1988). *A Knowledge-Based Approach to Connect Four: The Game Is Solved*. M.Sc. Thesis, Vrije Universiteit.

[7] Silver, D. et al. (2018). *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. Science, 362(6419), 1140–1144.

[8] Tesauro, G. (1995). *Temporal difference learning and TD-Gammon*. Communications of the ACM, 38(3), 58–68.

[9] Mnih, V. et al. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529–533.

[10] Schmid, U. et al. (2019). *Benchmarking Deep Reinforcement Learning for Connect Four*. arXiv:1906.00210.

[11] Heinrich, J., & Silver, D. (2016). *Deep Reinforcement Learning from Self-Play in Imperfect-Information Games*. arXiv:1603.01121.