A Monte Carlo Tree Search Algorithm for the Game of Connect 4

Dynamic Programming and Reinforcement Learning (DPRL) – Assignment 3

December 8, 2024

Group 57

Sara Abesová (2735679)
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
s.abesova@student.vu.nl

Matúš Halák (2724858)
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
m.halak@student.vu.nl

1 INTRODUCTION

Connect 4 is a classic strategy game played on a 6x7 grid. The game board consists of 6 rows and 7 columns, and the objective is for a player to "connect" four of their tokens either horizontally, vertically, or diagonally. The game concludes when one of the players achieves this alignment or when the board is fully occupied. The unique feature of Connect 4, is that tokens are dropped into the columns and, therefore, can only be placed on the lowest available vertical position in any given column.

In this project, we implement and investigate a model-based Monte Carlo Tree Search (MCTS) reinforcement learning (RL) algorithm enhanced with the Upper Confidence Bound for Trees (UCT) for the game of Connect 4. We test the algorithm under two starting conditions: an empty board and a pre-filled board configuration.

The ultimate goal was to create an adaptive agent capable of beating a random opponent from any board configuration. By balancing exploration and exploitation, the agent successfully navigates the challenges of Connect 4 and adapts to unpredictable gameplay.

2 METHODS

2.1 State Space and Action Space

2.1.1 State Space. The playing board consists of 6 rows and 7 columns, resulting in a total of 42 slots. Each slot can be either empty, occupied by a token from Player 1, or occupied by a token of Player 2. For the empty board, all 42 slots are empty at the start. In our formalization, the state space is $\mathcal{X} = \{x \in \{0, 1, 2\}^{6\times7}\}$ with elements x corresponding to individual board positions. Thus, the size of the state space corresponding to the maximum number of board positions is $|\mathcal{X}_{\text{empty}}| = 3^{42}$. However, this is a crude estimate, since many board configurations are illegal according to the game rules of Connect 4. After subtracting illegal board configurations which can never occur in a Connect 4 game, a more accurate estimate of $|\mathcal{X}_{\text{empty}}|$ is 7.1×10^{13} [1]. In the pre-filled configuration, 5 out of 7 columns are completely filled and unavailable for placing a token. Therefore a crude estimate of $|\mathcal{X}_{\text{pre-filled}}|$ is 3^{12}.

2.1.2 Action Space. In the game of Connect 4, the action space consists of columns in which a player can place their token. In this case, the action space depends on the state space $a \in \mathcal{A}_x$, specifically, the actions available from any given state x (board configuration) are only those, in which there are empty slots available.

2.2 MCTS with UCT[1]

The implemented Monte Carlo Tree Search (MCTS) algorithm consists of many iterations of four main steps: selection, expansion, simulation, and back-propagation. The process begins at the root node, which represents the current board configuration.

2.2.1 Selection. In the selection step, the algorithm traverses the search tree by following a path determined by a balance between exploration and exploitation. This was computed using the Upper Confidence Bound applied to Trees (UCT) formula which determines which action to take in any state x:

$$\text{UCT}(x) = \arg\max_{a\in\mathcal{A}_x} Q(x,a) + c\sqrt{\frac{\ln(N(x))}{n(x,a)}}$$

Where each term is defined as:

- $Q(x,a)$: the incrementally updated average value of simulations that selected action a from state x (exploitation),
- $N(x)$: the total number of visits (simulations) of node x,
- $n(x,a)$: the number of times action a was selected in state x,
- c: the exploration parameter (set to $\sqrt{2}$).

In this formula, $Q(x,a)$ drives exploitation of best moves by reflecting the average reward from all previous simulations that selected action a from state x, while the second term encourages exploration of less visited actions. This is crucial, especially in early stages of the search where Q-values based on few random roll-outs are unreliable and the tree first needs to be sufficiently explored.

Our implementation includes positions where it is Player 2's turn (enemy board positions) in the search tree. The selection stage from enemy board positions can happen via random state transitions, where in each pass through the search tree, the enemy move is chosen randomly from $\mathcal{A}_x$. Alternatively, to model an optimal opponent, enemy action selection can also be set to *minimax* mode, where selection from enemy board positions follows the UCT formula with argmin instead of argmax, forcing the algorithm to find the best responses from the worst-case scenarios every move.

Selection continues down the tree, choosing actions at each node based on the UCT formula (random transitions or minimax at enemy nodes), until it reaches either a terminal state (where the game ends) or a leaf node (board position with unexplored actions).

2.2.2 Expansion. During expansion, a new child node is added to the tree when the selected node is not terminal and has unexplored actions. Each child of any parent node represents the board resulting from taking one of the possible actions from the parent node.

2.2.3 Simulation. After adding a new child node, we estimate the reward of the new board position by playing out a game until conclusion with random moves for both enemy and the agent. The win, draw or loss (+1, 0, -1) result of the random roll-out provides a crude estimate of the child node's board configuration quality, since advantageous board positions produce more wins under random moves than poor ones. If the new child node is a terminal state, it's reward corresponds to the game result.

2.2.4 Back-propagation. Once the reward estimate is obtained either directly from the selected terminal node or via random roll-out from a new child node, the algorithm traces back the state-action pairs that were visited on the path to the terminal / new child node and increments their, as well as, each visited node's visit count. The Q-values for each state-action pair are updated with the reward estimate using the incremental average formula:

$$Q(x,a) \leftarrow Q(x,a) + \frac{R - Q(x,a)}{n(x,a)}$$

Here, R is the reward estimate from the random roll-out or the direct reward from a terminal state reached by a from x, $n(x,a)$ is the updated visit count for the action a taken from state x. This incremental average ensures that the algorithm's estimates of action values in each state become more accurate over time.

[1] In addition to the random opponent, our algorithm is designed to also able to play against a live (human /AI) opponent.

2.2.5 Per-move tree search. We chose to perform a MCTS for each move by shifting the root node to the current board configuration resulting from each enemy move. This approach allowed our algorithm to be fast, while allocating resources to essential decisions, namely finding the best immediately upcoming move from the current board. We reused each search tree, by storing all previously encountered node objects in a dictionary, with hashable string representations of the flattened board positions as keys, enabling us to perform $O(1)$ retrieval of previously encountered board positions during each MCTS and keep updating the Q-values only for state-action pairs relevant for each upcoming move.

2.2.6 MCTS Iterations per Move. A crucial aspect of our MCTS algorithm was determining the constraints on computation allocated for each move. We chose to use *iterations per move* instead of *search depth* or *time limit*, which are also commonly used. We explored up to 5,000 MCTS iterations per move.

2.2.7 Live-game MCTS-based Move Choice. During a live game, our RL agent's each move from a board position (root node) is determined by the root-action pair with the most visits after the 5,000 MCTS iterations, which thanks to the UCT rule, corresponds to the action with the highest Q-value & winning probability.

3 RESULTS & DISCUSSION

3.1 Example game from pre-filled board

In Figure 1 we show a game that our algorithm played against a random opponent from the pre-filled board. The absolute values of Q-value edge labels can be seen as winning probabilities if $Q(x, a) \geq 0$ and losing probabilities if $Q(x, a) \leq 0$ associated with the state-action pairs. We check the winning probabilities and see that MCTS produced correct estimates. Starting with action 4 (column e, 1_4) is certain to win (0.98), because if the enemy plays action 6 (column g, 2_6), we can keep placing our tokens in g until the enemy is forced to place their token in f (not shown). As soon as an enemy token is placed in f (2_4), we can then immediately win (0.99) by also placing a token in f (3_4). Importantly, if we instead choose to follow-up enemy's 2_4 by column g (3_6), we are likely to lose (-0.99), because we give the enemy a directly winning move (column g).

3.2 Convergence Analysis

We analyze the convergence of Q-values for actions from the root node in the first MCTS from the starting position for two board configurations: the pre-filled assignment board (AB) and the empty board (EB). For each board, we compare two selection strategies for enemy moves during the MCTS (random transition and minimax).

3.2.1 Assignment Board (AB). For AB, we see in Figure 2 that using random transitions from enemy nodes, our algorithm converges on the optimal opening move (4) after 1,000 iterations and also identifies the other opening move as risky against a random opponent. In contrast, using the minimax strategy (Figure 4, the algorithm identifies move 4 as optimal quicker (in 500 iterations), but eventually finds the other opening move almost equally good against an optimal opponent. This is because an optimal opponent is predictable and because of the opening move advantage can be beaten with either starting move. However, with suboptimal opponent (such as the random opponent or even a human opponent), focusing the tree search on the optimal game-play can be a disadvantage and lead to under-exploration of random moves, leading to unexpected losses / draws. Therefore, using random transitions to model the opponent from the AB leads to potentially more reliable results.

3.2.2 Empty Board (EB). As expected, for EB the Q-values converge later because a bigger tree with more possible moves needs to be explored to accurately determine the best starting move. Interestingly, we see that both random-transition and minimax strategies converge to the same optimal starting move 3 in the central column (which maximizes the number of possible alignments) after around 2,500 iterations (Figures 3 & 5). However, only minimax converges around 90% winning probability for the opening move within 5,000 iterations. The random-transition strategy is stuck around 30% winning probability even after 5000 iterations, because it is forced to randomly keep exploring branches from each enemy node, unable to rely on the UCT rule for enemy moves to relax exploration in later stages of the search. Therefore, especially with fewer iterations per move, minimax could be more reliable from the EB since it shows more certainty in the optimal moves.

3.2.3 Full-game win rates. When investigating convergence on the level of win-rate for full games against a random opponent, we notice that perfect Q-value convergence on per-move MCTS-level is not essential (Figure 6). After 1000 MCTS iterations per move, our agent achieves 100% win rate across boards and strategies. With as little as 100 MCTS iterations per move (long before Q-value convergence in any of the cases discussed), the agent achieves 98-100% win-rate, with significantly lower computational costs. This is thanks to our per-move MCTS approach, which allows suboptimal start moves to be compensated in future moves, where each successive MCTS needs lower search depth and can re-use more of the previously constructed tree in each iteration.

3.3 Possible improvements

In a benchmark against an optimal Connect4 solver by Pascal Pons [2], our MCTS algorithm could only win against the easy-medium difficulty, but not against the perfect solver. We identified several possible improvements to our MCTS algorithm in the solver's documentation. Our algorithm weighs all wins and losses equally, whether they arise the next turn or in many turns. Instead, rewards from moves that can lead to immediate terminal states should outweigh rewards from moves that lead to distant future terminal states. Perfect Connect 4 solvers assign direct rewards to all board positions, based on the number of moves until a terminal state, assuming both players play optimally. For MCTS, this could perhaps be approximated by scaling average reward estimates by average roll-out depth from a given node. Additionally, the tree search can be greatly optimized using e.g. alpha-beta pruning.

4 CONCLUSION

In this project, we implemented and investigated a model-based RL agent using MCTS for the game of Connect 4. Our agent was able to consistently outperform a random opponent across board configurations and achieved good results against both human opponents, and Connect 4 solvers. These results affirm the utility of model-based reinforcement learning for situations with a perfect generative model of the environment. We suggest improvements to our MCTS algorithm that could potentially lead to results competitive with completely deterministic perfect Connect 4 solvers.

APPENDIX

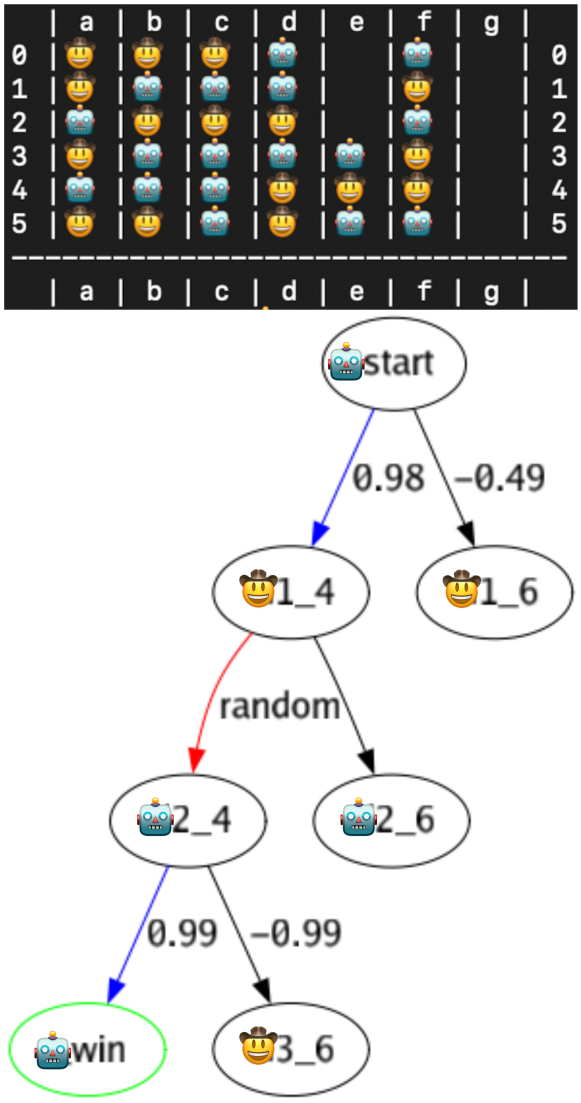

Figure 1: Example Game and MCTS from a Pre-filled Board Configuration

Top: Example Connect 4 game visualization in the terminal, the MCTS agent's tokens are represented by the robot emoji and the random opponent's tokens are represented by the cowboy emoji. One example game is shown. *Bottom*: Depiction of MCTSs for the example game, including edges labeled with average Q-values for state-action pairs from root nodes of each MCTS applied per game move. Q-values for opponent's state-action pairs are not shown. Emoji in a given state indicates which player's turn it is from that board configuration (except terminal states, where it shows the winning player), the first number indicates the depth of this node within the search tree / the number of moves played thus far, the second number after the underscore indicates the parent-node's action (0-indexed) that led to that state. Tree visualization was made using GraphViz: https://graphs.grevian.org.

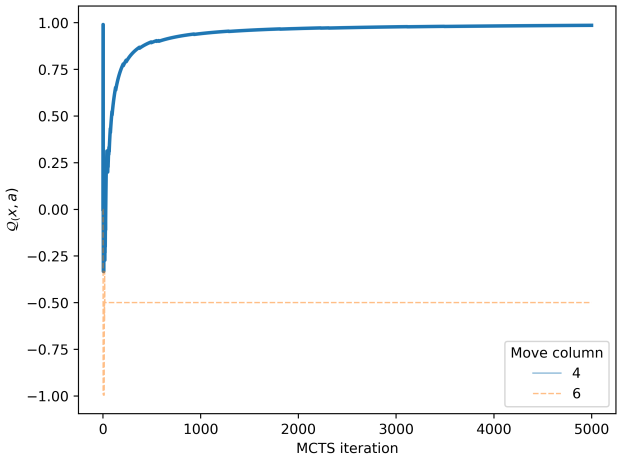

Figure 2: Starting move Q-value convergence on Assignment Board with random state transitions from opponent nodes

Individual lines represent Q-values for individual state-action pairs for all available actions from the starting state (pre-filled Assignment board). The bold line represents the chosen starting move with the highest Q-value, winning probability and number of visits.

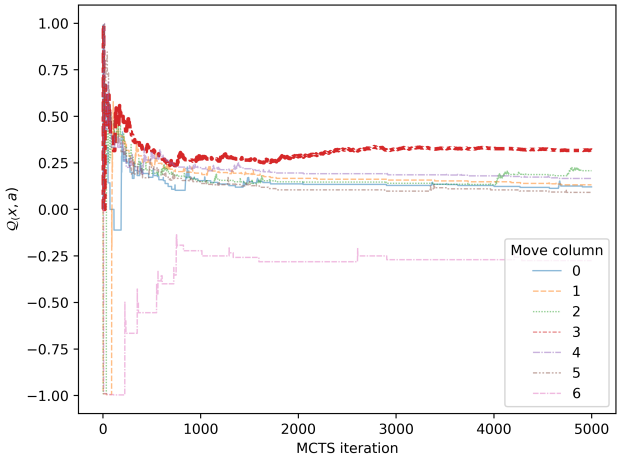

Figure 3: Starting move Q-value convergence on Empty Board with random state transitions from opponent nodes

Individual lines represent Q-values for individual state-action pairs for all available actions from the starting state (Empty board). The bold line represents the chosen starting move with the highest Q-value, winning probability and number of visits.

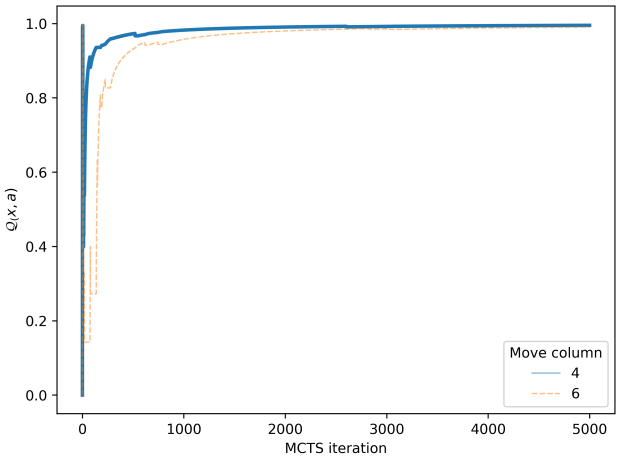

Figure 4: Starting move Q-value convergence on Assignment Board with minimax MCTS strategy

Individual lines represent Q-values for individual state-action pairs for all available actions from the starting state (pre-filled Assignment board). The bold line represents the chosen starting move with the highest Q-value, winning probability and number of visits.

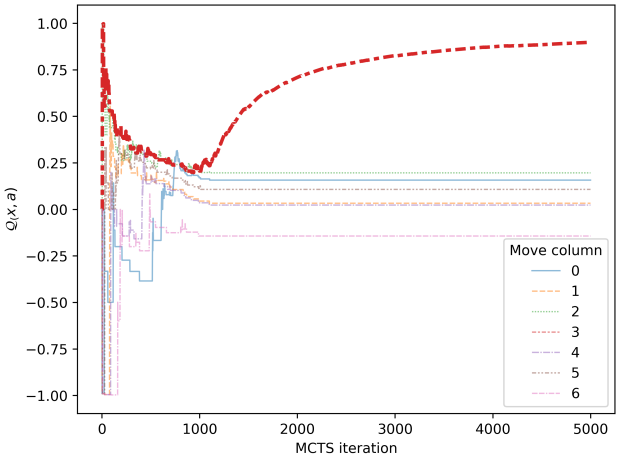

Figure 5: Starting move Q-value convergence on Empty Board with minimax MCTS strategy

Individual lines represent Q-values for individual state-action pairs for all available actions from the starting state (Empty board). The bold line represents the chosen starting move with the highest Q-value, winning probability and number of visits.

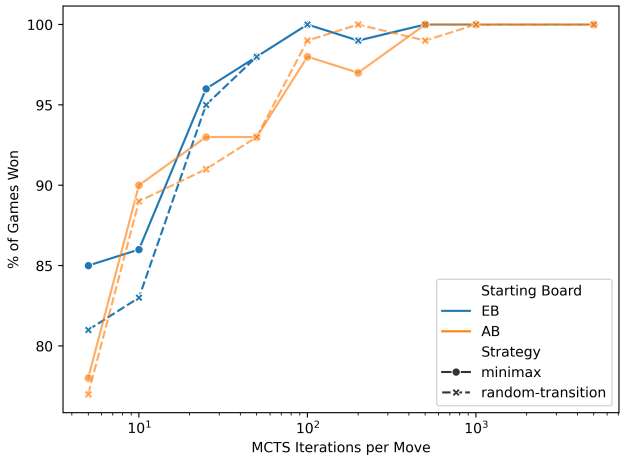

Figure 6: Win rate comparison across starting boards and strategies

Win rate (% of Games Won) was obtained by playing out a 100 full games against a random opponent from a given starting board (EB / AB), with a given number of MCTS iterations / move (shown on a logarithmic scale - 5/10/25/50/100/200/500/1000/5000), using a given selection strategy for enemy moves during MCTS (random-transition / minimax).

REFERENCES

[1] Victor Allis. 1988. A knowledge-based approach of Connect-Four. *ICGA Journal* 11, 4 (Dec 1988), 165–165. https://doi.org/10.3233/icg-1988-11410

[2] Pascal Pons. 2019. Solving connect 4: How to build a perfect AI. http://blog.gamesolver.org/