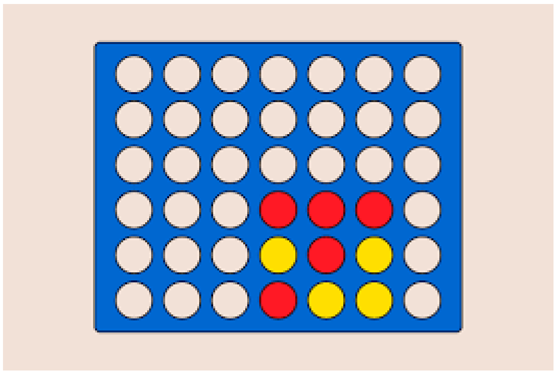
Connect 4 using Reinforcement Learning: Report

Spring 2024

IIT Indore

GAME OVERVIEW

Connect Four is a **two-player game** that takes place on a 6x7 rectangular board placed vertically between them. One player has 21 yellow coins and the other 21 red coins. Each player can drop a coin at the top of the board in one of the seven columns; the coin falls and fills the lower unoccupied square. Of course, a player cannot drop a coin in a particular column if it's already full (i.e. if it already contains six coins).



Even if there's no rule about who begins first, we assume that the red side makes the first move. We also use a notation to represent a square on the board. That is, we number rows from 0 to 6 starting from the bottom and the columns from 5 to 0 starting from the leftmost.

Characteristics of the Game

- Connect Four is a solved game. The first player can always win by playing the right moves.
- There are 4,531,985,219,092 positions.
- Mathematically, the upper bound for the number of state-action pairs is 4,531,985,219,092 x 7 = 31,723,896,533,644 (about 31.7 trillion pairs)

PROBLEM STATEMENT

The rules of the game are as follows:

- Every player tries to connect their coin type in a sequence of 4.

- After every move by any player, the game's status is checked whether it is over. A game is considered over in the following scenarios:
    - Either of the players have 4 coins on the board in a sequence vertically, horizontally or diagonally.
    - The board is complete; that is, after 42 moves, none of the players got a sequence of 4. In this case, the game is a tie.

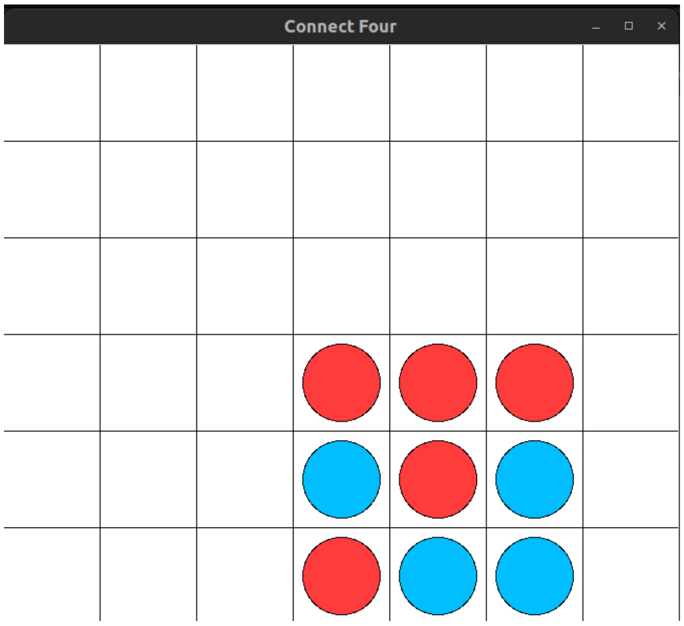
OBJECTIVES

Our goal is to train an agent who plays the game optimally with a fair understanding of the game's rules. The trained agent should be able to play with human players and maximise its number of wins.

METHODOLOGY

Environment

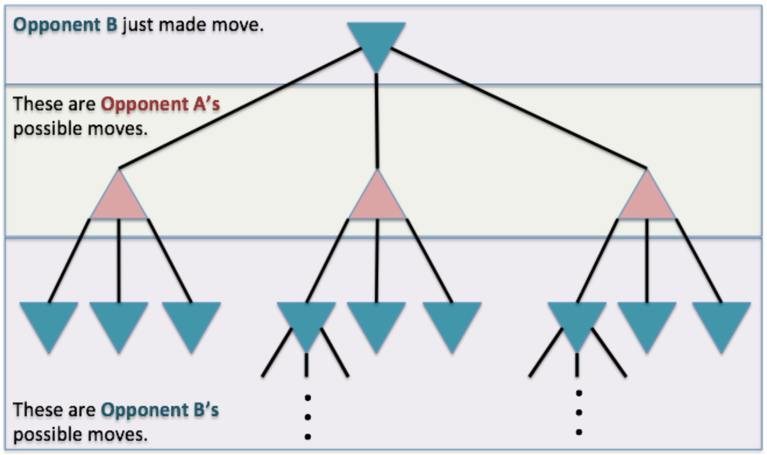
We began by implementing the Connect Four environment, which includes the game board, rules, and player interactions as defined above. Next, we defined the state representations, action spaces, and reward functions tailored to the Connect Four game.



Opponent Agents

We created different agents that will help the RL agent to learn from and play against. These agents also helped us evaluate the performance of the RL agent being trained. The agents include:

1. **Random Agent:** An agent that plays any random move given the valid moves it can take.
2. **Smart Agent:** An agent that is better than a Random Agent. It plays randomly, but when it encounters a position where it can win, it chooses that action promptly.


Opponent B just made move.
These are Opponent A's possible moves.
These are Opponent B's possible moves.

3. **Minimax Agent:** An agent that foresees K next moves and puts the coin in the column that maximises the reward it can get. The reward is allocated based on some configurations. We also optimised this algorithm using alpha-beta pruning.

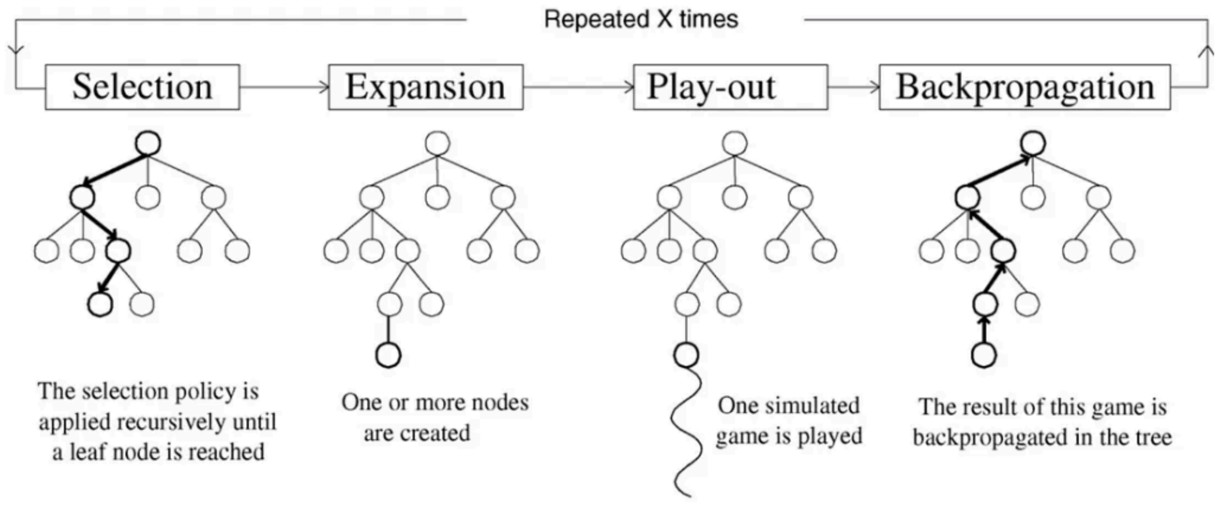
We used these opponent agents to train our RL agent, allowing it to learn optimal strategies for playing Connect Four. We employed two Reinforcement Learning algorithms: **Q-learning** and **Monte Carlo Tree Search (MCTS)**.

Proposed Agents

- **Q-Learning:** A model-free RL algorithm that updates action-value estimates iteratively using observed rewards, enabling the RL agent to learn optimal policies and improve decision-making over time.
- **Monte Carlo Tree Search (MCTS):** A heuristic search algorithm that incrementally builds a search tree by simulating random plays from the current game state. It

balances exploration and exploitation to guide decision-making, maximising the agent's chances of winning.

Initial Setup — Start with a single root (parent) node and assign a large random UCB value to each non visited (child) node.

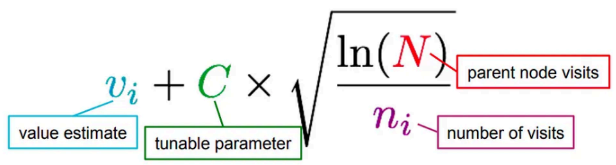

Repeated X times
Selection
Expansion
Play-out
Backpropagation
The selection policy is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

The four main steps of Monte Carlo Tree Search.

Selection — In this phase, the agent starts at the root node, selects the most urgent node, applies the chosen actions, and continues till the terminal state is reached. To select the most urgent node upper confidence bound of the nodes is used. The UCB process helps overcome exploration and exploitation dilemmas.

Pick each node with probability proportional to:

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$



Formulation of Upper Confidence Bound to select the next action

Expansion — When UCB can no longer be applied to find the next node, the game tree is expanded further to include an unexplored child by appending all possible nodes from the leaf node

Simulation — Once expanded the algorithm selects the child node either randomly or with a policy until it reaches the final stage of the game

Backpropagation — When the agent reaches the final state of the game with a winner, all the traversed nodes are updated. The visit and win score for each node is updated.

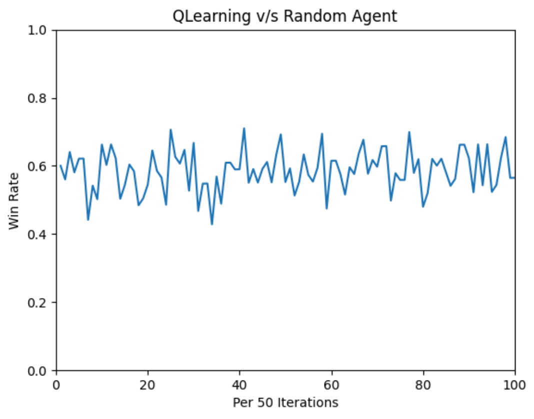
PERFORMANCE METRICS

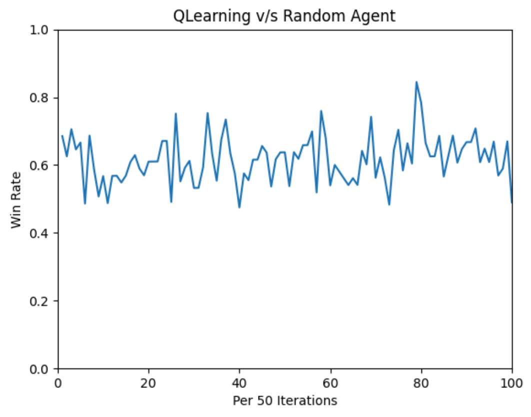
To evaluate the effectiveness of our trained RL agent, we measure its win rate against the Random Agent.

EVALUATION

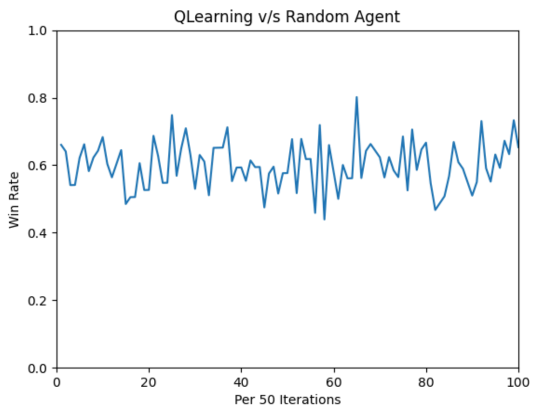
Q-Learning Agent

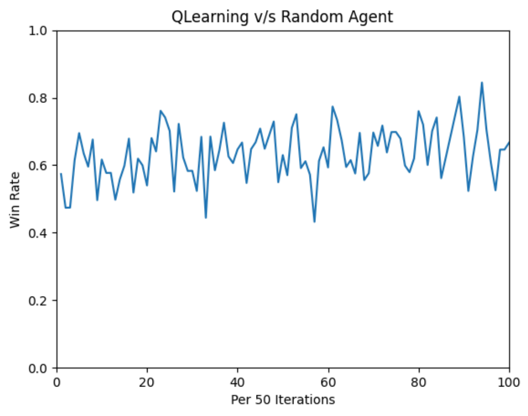
Training with Random Agent



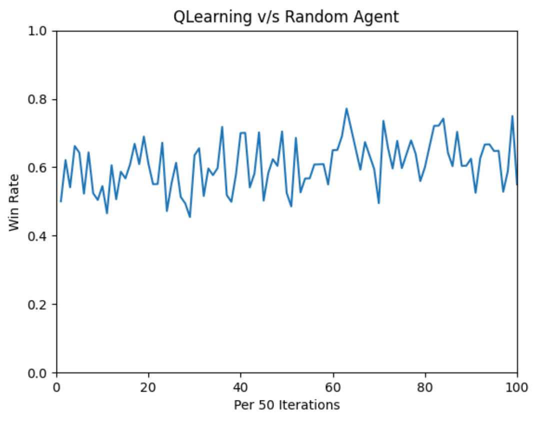


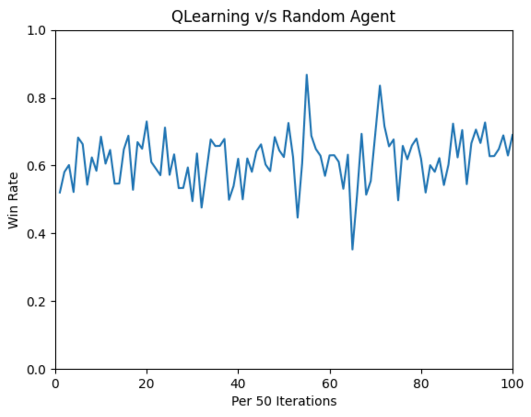
Training with Smart Agent



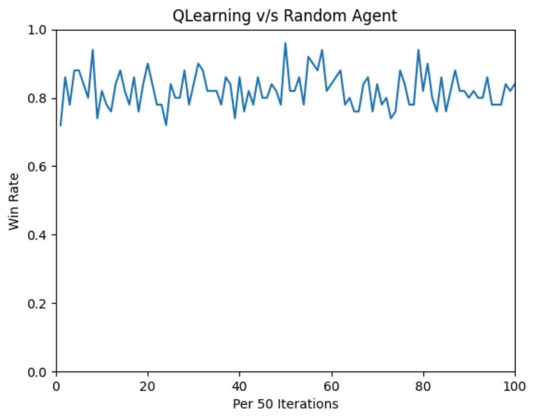


Training with MiniMax Agent

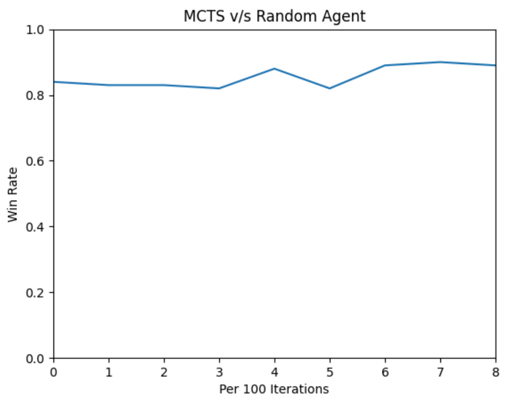


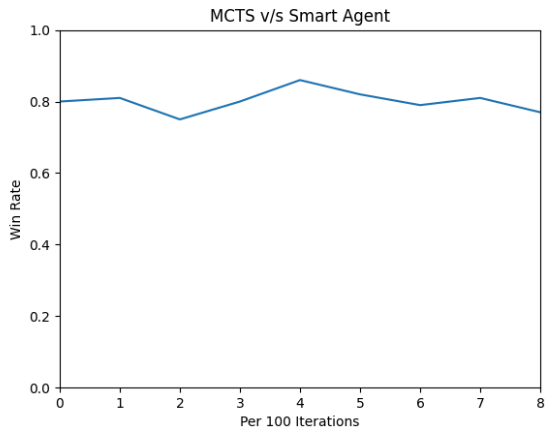


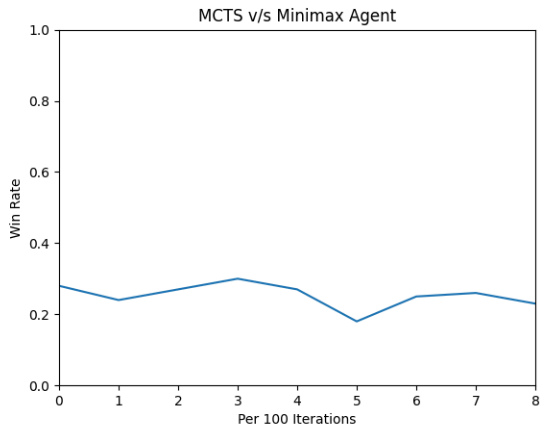
Smart Q-Learning Agent



MCTS Agent







CONCLUSION

Following are the conclusions that we reached:

- The game consists of a vast number of state-action pairs that are beyond our capabilities to compute; thus, there are still many states that the agent does not visit. Therefore, we will need more computing resources to train a flawless RL agent.
- The agent is improving steadily and has not reached its saturation state. This means that the more time we spend on training, the better it will become.
- Using a smarter opponent to train an RL agent is much more beneficial. Though not visible clearly, upon closer inspection, we can observe that the RL agent does better when trained with a MiniMax Agent, followed by SmartAgent and, finally, the Random Agent. This means that you learn better when you play against a more challenging enemy.

- The win rate of the Q-Learning Agent can be easily improved by incorporating model information in the agent. For example, if the Q-Learning agent is designed to choose the best move if it can win in 1 move. The win rate of the agent increases by at least 20%.
- MCTS has been performing better than q learning because of its tree based algorithm using upper confidence bound for balancing exploration and exploitation.

TEAM MEMBERS

1. Amit Kumar Makkad (200001003)
2. Mukul Jain (200001050)
3. Nilay Ganvit (200001053)

Under the guidance of Dr. Surya Prakash, Professor of Computer Science and Engineering IIT Indore.

REFERENCES

1. Connect Four. (2024, April 14). In *Wikipedia*. https://en.wikipedia.org/wiki/Connect_Four