There are two Python files in the folder, one of which is the minimax file. This file is used for the first part of the assignment, as well as both bonus sections. The user needs to input the row count (n) and column count (m), and then select the game mode, which is part of the second bonus.

In the next file which is related to the second part of the assignment, MCTS is implemented. MCST (Monte Carlo Tree Search) is a tree search algorithm that is commonly used in game artificial intelligence, particularly for games with a large search space and imperfect information. It's used to estimate the value of a move in a game by simulating many random playouts from the current game state to the end of the game. The algorithm balances exploration of untried moves and exploitation of moves that have already shown to be successful in previous simulations. I implement the following steps:

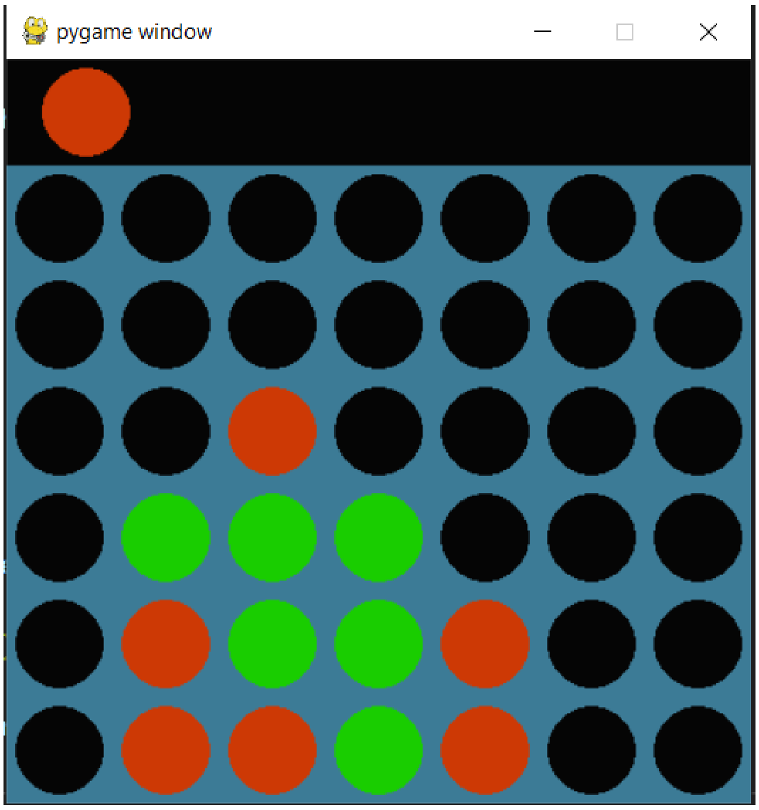
1. Represent the state of the game as a tree: Each node in the tree represents a possible game state, and the edges between nodes represent the moves that can be made to transition between states.
2. Perform simulations: Starting from the root node of the tree, randomly simulate playouts by selecting moves and following them to their end. The playouts can either be deterministic or random, depending on your needs.
3. Update the tree: After each simulation, update the statistics of the nodes that were visited, such as the number of wins, losses, and draws, as well as other information that is relevant to your particular application.
4. Select the best move: After a sufficient number of simulations, use the information stored in the tree to select the best move to make. This can be done by selecting the node with the highest win rate, or by using other metrics such as the upper confidence bound (UCB) to balance exploration and exploitation.
5. Repeat the process: Repeat the simulation and update process until a stopping condition is reached, such as a time limit, a desired number of simulations, or a sufficient level of confidence in the selected move.

There are several advantages to using Monte Carlo Tree Search (MCST) to solve Four Connect:

1. Effective exploration: MCST is able to explore a large state space effectively by using random simulations to guide the search. This is especially useful in games like Four Connect, where there are many possible moves at each step and it can be difficult to know in advance which moves are best.
2. Handling uncertainty: Unlike traditional search algorithms like Minimax and Alpha-Beta pruning, MCST is able to handle uncertainty in the game by simulating many random playouts. This makes it well-suited for games with imperfect information, such as Four Connect.
3. Avoiding over-fitting: MCST balances exploration and exploitation by using metrics like the upper confidence bound (UCB) to decide which moves to simulate next. This

helps to avoid over-fitting to a particular set of moves and ensures that the algorithm is able to find the best move even in complex or unpredictable situations.
4. Flexibility: MCST is a general-purpose algorithm that can be adapted to a wide range of games and search spaces. This makes it a useful tool for solving games like Four Connect, where traditional search algorithms may not be well-suited.
5. Fast convergence: MCST has been shown to converge quickly to an accurate solution in many cases, making it a practical choice for real-time games like Four Connect.

Overall, the use of MCST to solve Four Connect can lead to improved game performance, better handling of uncertainty, and more flexible and effective search algorithms.



A class named "Board" is defined, which will store the state of the game board and provide methods to check if a player has won, if a move is valid, and if the board is full. The class has several methods:

- **init**() method that initializes the board.
- isWinner() method that checks if the player's move results in a win.
- isValidMove() method that checks if the move is within the range of the board and the space is empty.
- isBoardFull() method that checks if the board is completely filled.
- makeMove() method that allows a player to make a move on the board.
- drawBoard() method that draws the board on the console.

The "Board_init" function returns an empty board and the "get_move_MCST" function uses the Monte Carlo Search Tree (MCST) algorithm to find the best move for the player. This function takes in the player, board, and number of iterations as arguments and returns the best move for the player.

The "main" function plays the game, allowing two players to take turns making moves and checks if someone has won or if the board is full. The game continues until either a player wins or the board is full.