Project 1 Connect M Computer Game

Chase Lamkin

20 Feb 2022

Project Description

This project aims to create a connectM board game, with a resizable board, changeable win condition, and changeable starting player, which can be played by a player against an AI. The computer will make moves against the player to potentially win, while the player will make moves against the computer to potentially win.

Agent Model & Agent Environment

The agent model used for this project is a utility-based agent system. This is due to each board state being evaluated and interpreted as a utility value. The utility value gives the agent a sense of whether a board state is preferable and is used in parallel with a minimax algorithm that allows the agent to determine which move leads to the best utility value. This method of evaluating different moves is utilized by the agent until a winning move is made by the player or computer.

The agent environment for this project is fully observable, deterministic, sequential, static, discrete, and multi-agent. The environment is fully observable because the board data structure is available to the computer and is the only relevant information to the system. The environment is deterministic because there is no uncertainty in the board game, and moves made are deliberate, not random. The environment is sequential because moves made at any point during the game affect the eventual outcome. The environment is static because while the player or computer is deliberating what their move should be the board state does not change. The environment is discrete because there are a limited number of possible winning board states. Finally, the environment is multi-agent because both a player agent and a computer agent are playing and affecting the board.

Data Structures

The data structures used in this program are lists, trees, and meshgrids. List data structures are used for both the board state, evaluation functions, and minimax tree nodes. The board state is a list of lists, with each subsequent list representing a row in the board. This data structure is used both to visually represent the board and to analyze the board using utility functions. The utility functions used are broken down into three categories: column, row, and diagonal. Each category has small differences but utilizes lists in the same manner. The respective areas of the board relevant to the utility function are read by the utility function and as they are read a list for both the disk analysis and spatial analysis is created and appended with the relevant information. The reason a single list is used for this is that it allows this information to be condensed into an easily traversable data structure, which can be compared to its counterparts for accurate and time-efficient analysis. The list data structure is used for the tree nodes to allow each node to keep track of what moves resulted in the code, without having to create a new board. This saves program time and space at large depths which can cause node expansion at rates as high as 10^L.

Tree data structures are used for the minimax algorithm which utilizes the utility values of each board state. The reason a tree data structure is used is that it allows for easy traversal, by the minimax algorithm, to determine different possible outcomes of different moves.

The numpy meshgrid data structure is used to assist with spatial analysis. Once the game has started a numpy meshgrid of equal size is created, and each position on the meshgrid has a corresponding value. The reason behind this is that the numpy library and data structure allow for the easy creation of a standard deviation table, which can be used to analyze the importance of certain positions on the board.

Alpha-Beta Pruning

The alpha-beta pruning algorithm is used in parallel with my minimax algorithm to prevent the traversal of unnecessary branches. The minimax algorithm without alpha-beta pruning gets the utility values of board states at the bottom of the tree. For each parent node, the children nodes are compared and depending on whose move it is at the parent node, either the maximum value (computer) or minimax value (player) will be assigned to the node. This continues up the tree until the children nodes of the current board state each have a utility value, representing the best possible move that can be made from the current state. This is done recursively, and as the tree expands exponentially at each subsequent depth, the algorithm can begin to take drastically more time. The alpha-beta pruning algorithm helps to alleviate this by assigning two variables, alpha, and beta, the worst and best possible utilities for explored branches. When subsequent branches are explored if branches previously explored contain a much better move for the parent node, the minimax algorithm stops evaluating the branch because it assumes the player and computer will always select the best possible move. This can drastically decrease the time and space costs of the minimax search.

Heuristic Evaluation Function

The evaluation function for the connectM board game is made up of 4 key components: Spacial analysis, disk analysis, threat analysis, and weight analysis. It is important to note that these methods are only used to obtain the static evaluations of each board state and that the minimax algorithm also keeps track of how many moves are made to get to that state. The utility of that board state is then divided by the number of moves made to get there, which emphasizes obtaining preferable board states in fewer moves.

1. **Spacial analysis** - Once the board game has been created with valid parameters, a class variable named gaussian_board is created, which is a numpy meshgrid, equal in size to the board game. This meshgrid is a gaussian distribution board, that is normalized and contains a standard deviation of 1. The reason behind this is that there are more opportunities to win the closer a piece is to the center of the board. When analyzing disks in the disk analysis component, disks are weighted using their position on the gaussian distribution board, which incentivizes placing disks in favorable positions.
2. **Disk analysis** - The base weight of one dimension of a disk is 1 for the computer and -1 for players, when not blocked by other pieces. The total base weight of a disk is 3 for the computer and -3 for the player, which corresponds to the three phases of disk analysis: row analysis, column analysis, and diagonal analysis.
    1. **Row analysis**: Each consecutive touching piece's weight is incremented or decremented by one. For example, 3 consecutive computer pieces contain a total base row weight of 6. However, if player pieces block a potential win with those pieces or empty spaces do not contain pieces below them, preventing a potential win within the next few moves, their weight becomes zero. If the spaces next to the pieces are valid placements and can form potential wins the total row weight for the pieces becomes (row weight + (.2 times the number of adjacent and valid empty spaces)) * (1 + gaussian weights). This formula gives empty valid spaces a 20% weight, incentivizing placing pieces in rows with many potential moves, and then the total calculated value is multiplied by the total gaussian values of the pieces incentivizing placing pieces in favorable positions. After all values for a row have been calculated the total is summed, and once all rows have been calculated their values are summed. This gives provides the static evaluator the total row advantage of the player and computer.
    2. **Diagonal analysis**: Each consecutive touching piece's weight is incremented or decremented by one, similar to the row analysis. Summing weights also only occurs if a potential win can be formed with the pieces, however, where the diagonal analysis differs from the row analysis is that empty spaces are not included in the final evaluation formula. Instead, the diagonal formula is: (diagonal weight * (1 + gaussian weights). The reasoning behind this is that empty spaces are more easily blocked in diagonals, than in rows.
    3. **Column analysis**: Column analysis increments and decrements pieces by one similar to diagonal and row analysis, but only does so for the pieces of the player, which is at the top of the column. The reasoning behind this is that only players at the top of the column can form wins in that column. This results in the following analysis formula for columns: (column weight * (1 + gaussian weight)).

3. **Threat analysis** - The threat analysis is the most heavily weighted component of the evaluation function, but is only calculated in the final weight of the board. While the rows, diagonals, and columns are being analyzed if an empty space is found, which is a valid move and provides a potential win for the player or computer in the next win, the player or computer threat total, respectively, is incremented by one. Once all advantages for rows, diagonals, and columns have been found and weighted against each other, the evaluator function includes the potential threats. The logic for this is as follows: if the player or computer's move is next return the second highest possible advantage in their favor, if two threats exist for the player or computer return the second highest possible advantage in their favor, and if neither is true increase the advantage of the player or computer by 30%. The reasoning behind this is that if a player or computer has a threat on the board and makes the next move, they have essentially won, and the same holds for two threats. However, if they hold lockable threats then they force the player or computer to make an optimal move, making their overall advantage nearly half a move better.
4. **Weight analysis** - This component of my evaluator function is not heavily weighted but the most important if a player or computer is not about to win, because it completes the disk analysis, evaluating how important player and computer advantages in their respective positions truly are to historical iterations of the game. To do this I analyzed a dataset of over 44,000 connect 4 games and found that roughly 50% of all wins came from columns, 30% of all wins came from rows, and 20% of all wins came from diagonals. In the evaluation function, the overall row, diagonal, and column disk advantages are weighted with these values and then normalized before being weighted by potential threats.