Project 1 Connect M Computer Game

Jared Cox and Deesha Rajiv

March 2, 2025

Project Description

For this project, we were tasked with the challenge of creating a Connect M Computer Game, and implementing a sophisticated AI algorithm that allows human players to play our game against an AI. More specifically, we were asked to implement alpha-beta pruning, as well as a min function and a max function to support our AI algorithm.

Agent Model and Agent Environment

The agent model that matches our computer program is a **model-based, goal-based agent**. The reasoning for this is that the agent maintains an internal model of the environment, or the game board, to predict the future states. It does this with minimax search and alpha-beta pruning. That is the model-based part of it. For the goal-based part, the goal is pretty clear. The goal of the agent is to achieve a victory and avoid a loss, aka connect M pieces in a row on the board. It is these two aspects of our agent that led us to the conclusion that our computer program closely matches the model-based, goal-based agent presented in class.

The environment for our agent is **fully observable**, as the agent can see the entire board and take action. It is also **deterministic**, given the fact that the agent can determine its next state, or its next move on the board, based on the current state of the board as well as the last action taken by the human player. The environment is also **sequential**, as the current action of the agent is dependent on the previous actions, this makes sense as the agent can only decide on where to place its piece based on the locations of the previous pieces. The environment can also be described as **static**, the board does not change while the agent waits to make its decision. The environment is **discrete**, as there is a finite number of places on the board to evaluate, the time and states are not continuous. Finally, it is **multi-agent** as two agents are playing the game, the human and the AI. In conclusion, the environment for our agent is fully observable, deterministic, sequential, static, discrete, and multi-agent.

Data Structures

In our program, we used a 2D vector (**vector<vector<char>>**) which represents the game board. Each cell in the grid stores characters like **'X'**, **'O'**, or ' ' (space) to keep track of each player's move on the grid. The vector is initialized based on the size of the board (**N x N)** and is updated whenever a player makes a move. This data structure allows easy indexing **(grid[row][column]**) and makes it efficient to detect wins, check for valid moves, and update the board after every play. Integer variables **N** (board size), **M** (winning condition), and **column** (column chosen for a move) are all used to track game parameters. In addition, a boolean variable **humanTurn** is also used to see who plays first.

For the AI decision-making process, a recursive tree structure in the **minimax** function is used with alpha-beta pruning. To make the decision-making process faster, the AI evaluates different possible moves by producing board states and pruning unnecessary branches. The AI determines its move using a list of column numbers stored in a vector (**vector<int>**). In this strategy, it helps the AI make powerful moves by using the center of the board to its advantage.

Alpha-Beta Pruning

The alpha-beta pruning algorithm in this project speeds up the decision-making process in this two-player game. It works by pruning parts of the game tree that don't need to be explored because they cannot influence the final decision. This algorithm keeps track of the alpha value and beta value. The alpha value is used to keep track of the best value the maximizing player can guarantee, while the beta value keeps track of the best value the minimizing player can guarantee. While the search continues, the algorithm can find a situation where the move is worse than another option, this is when it prunes the branch and stops evaluating it. This helps minimize the number of nodes that are explored, helping the process go by fast and still making the best move possible.

Heuristic Evaluation Function

The heuristic function that we implemented can be described as a terminal-based heuristic function, meaning that it evaluates the game state by only looking at if the game state is **terminal**, which means it is either a win, loss, or a draw. For our function, it first checks to see if the game board is in a winning state for the AI. If it is, the function returns a score of 1000. This of course indicates a highly favorable game state for the AI player. On the other hand, it checks to see if the game board is in a winning state for the human player. If so, the function returns a score of -1000 which indicates a highly favorable game state for the human player. If neither player has reached a winning state, the function returns 0 and the game continues.