*Assignment 3 Problem 3: Hexagon Game*

*Abstract*—The objective of the program was to adversarial search and possibly alpha beta pruning to create an artificial intelligence that could beat a user in a hexagon game.

*A diagram of a diagram of a triangle

Description automatically generated*

*The attempt represented the hexagon playing field using an adjacency matrix which each value in matrix would either 1 for the computer and 2 for the player however this design was only able to run based on the order of the players.*

# Methodology

To represent the hexagon graph we use an Adjacency Matrix. For AdjacencyMatrix[I][J] Given I and J vertices the Adjacency Matrix would hold the selected edge which is a int saying 0 for none, 1 for CPU, 2 for player. The program primarily runs on a True loop which alternates between player 1 and player 2 after then complete their tuns. Player 2 asks for user inputs in the format of “number-number” to define the edge

For player 1 or CPU, the program would run a minimax to find an open and best edge within the Initially the program finds an open space within the matrix to add an edge. Given the coordinates within the matrix, it would calculate the score. The AI should be attempting all possible edges in the matrix and calculating the score for each one. Then it would find the max and alter add the corresponding max edge within the Matrix.

For calculating a score there is a “terminal” which checks if the graph would reach a win, lose, or tie conditions. In this it would return 1, -1, or 0; however, in a situation in which the matrix is unfinished, the program would run a minimax function until it reaches a terminal state then recurse the value until it reaches the original function.

During minimax a program would return a player, depth, and alpha and beta parameters. Player identifies if the minimax is happening to player 1 or player 2 because the alpha beta pruning and minimax switch between the two Alpha and beta parameters to enable alpha and beta pruning. During max, the function would find the biggest value to set to alpha, and during min, the function would find the smallest value and place into Beta. As they transverse into the functions and return the score values. Alpha would be greater than Beta. At this point the program would return the original score it found, and break and not continue search.

The program should run, should alternate between player 1 and plyer 2 until a triangle was found. For player 1 we used a minimax to the best moves using while simulating the worst moves for the player 2. Alpha beta pruning was implemented within the alpha beta pruning limit searching for moves by break and returning the precious score

# Result

## Player 1 goes first

When Player 1 is called to go first, the program runs the minimax function and should’ve calculated a possible move. Because it moved first the expectation was that it would call 1-2 because it was the first edge to have 1; However, the algorithm lasted for 3 minutes, so the attempt was dsicarded

A screenshot of a computer screen

Description automatically generated

## Player 2 goes first

Identify applicable funding agency here. If none, delete this text box.

This program was able to alternate between player 1 and player 2. The program was able to identify any repeated edges from the user by checking the table. Additionally the program was able to print all moves made by both player 1 and player 2. In the end Player 1 was able to beat player 2.When player 2 made a triangle 1-2-3

A screenshot of a computer screen

Description automatically generated

# Discussion

Running player 1 first was able to run, but not able to finish finding the best move. Hexagon map had 15 moves into total, then each subsection had 14, then each had 13 … it came to the point that searching primarily through minimax was not possible. Originally the program was supposed to use C++, but I finalized it to python to easily access the class and methods better. Additionally, because of this huge tree, the alpha beta pruning was necessary to break the function, so it doesn’t search through the entire tree.

Additionally, Minimax adjust based on the moves of player 2, because player 1 was first, it had to search through a larger tree, compared to having player 2 go first, with a smaller search tree. There were multiple methods, to possibly limit the search within the adjacency matrix. Instead of separating the best method function with the minimax. We immediately call minimax from the play function to limit the number searches. Additionally, when we search for possible moves within the adjacency matrix, this method only required only search half of the entire matrix, by restricting added vertices I and J so I <J. This concept came into mind because the adjacency matrix accounts for the edges twice based on the order of the vertices specified. We only needed to account for the edge once, so we made it a priority that index I should never be bigger than or equal to J during search or assignment.

Overall, I think implementing an Adjacency Matrix was a poor choice in my end, because it required implementing a loop to determine the Terminal state. If you also include having to recurse through multiple states to find the “best move” using the same two functions, it makes sense why the program was unable to finish. There was a program that could run successfully, but instead of using an Adjacency list, they directly stored all edges within a set or frozen set. Because there are only 15 edges, it’s a small search within the set. This implementation would’ve been easier, but due to my poor knowledge in frozen set and splicing and couldn’t pursue this given the time frame.

# Conclusion

The program was made with the intent to implement a minimax and alpha-beta pruning within the Adjacency Matrix. This was unsuccessful as the setup for identifying Terminal failed and the data structure required loops to locate the best possible move. With the recursive it created too much time. This attempt includes the accumulation of change the programming language and implementing the alpha beta pruning. In future attempts, even though I have a strong understanding of the algorithm, it’s best to have a strong data structure to ensure searches and assignments takes even shorter time, especially when these functions are part of the recursive function.

.

costs would be equivalent to 17 however after adding their initial number cost g(n) for f(n) the total cost for moving 3 was 20, while moving 5 was 22. To find a solution that could find the cost of 30, the code must follow order of 6,2,5,6,2,3,4,2, but wouldn’t be able to do so because of it had a higher cost. Even if I added a queue, the program would’ve selected the wrong move. This opens the question of whether I should change my heuristic function, once again. I needed to adjust my heuristic to ensure that it was admissible and heuristic, but given the time frame, I was not able to do so.**.**