1. How My Program Works and Motivation for My Approach

I decided to implement an Iteratively Deepening Alpha-Beta Search Algorithm enhanced with a Transposition Table. I will go into more detail of how my program works by explaining how its design came to be.

1.a Choosing the Base Algorithm for My Agent

The first question I asked myself was if the main algorithm of my AI should be MCTS or Minimax Search. While MCTS seemed like it would theoretically perform better because of the high branch factor of the Pentago Game, I wanted to be sure.
To compare both potential solutions, I implemented two basic agents, one was using MCTS while the other was using basic Minimax Search. What I found was that surprisingly, even if the Minimax Search agent was using a random very simple evaluation function, it still won most games. This can probably be explained by the fact that minimax, tries its hardest to avoid moves that would lead to the opponent winning while MCTS just chooses its moves based on win rate probability. Nonetheless, I started building my own agent around minimax Search because of this result and the fact I found it simpler and more intuitive than MCTS.

1.b Implementing Alpha-Beta Search

My first step to improving my basic Minimax Search Agent was to implement alpha-beta pruning. As we saw in class, the Alpha-Beta Algorithm is an improvement on Minimax. The algorithm finds the maximum value in the leaves and prunes the rest of the sub-trees when it is known that they will not change the maximum. Whereas Minimax tests all the leaves in the entire tree for the maximum value.

1.c Choosing A Better Evaluation Function

The next step was to write a better evaluation function. With a better evaluation function, each board state value is closer to what it truly is. This leads to a better ordering of the tree nodes in the Alpha-Beta tree when we are eventually forced to cut-off the search because of the time limit.
I played around with a lot of different evaluation function, but they were always built around the following idea:

1. 5 marbles in a row is best
2. 4 consecutive marbles in a row is next
3. followed by 3 consecutive marbles
4. If a certain number of consecutive marbles cannot lead to a win because it is blocked, it is worth less points
5. Pieces at the center of a quadrant are better then pieces on the edges of the quadrant.

I will go into more detail in a later part.

1.d Implementing Iterative Deeping

Next, I decided to implement Iterative Deepening. This is because regular Alpha-Beta set at a fixed depth is not suitable when the game is time-constraints. If the depth is set to high, the algorithm might not finish running and will most likely not return the best move. If the depth is set too low, you are missing out on potential refinement of the best move with following iterations. With iterative deepening however, depth is gradually increased meaning we return the best move even while being time constrained; the best move can be adjusted at each level.

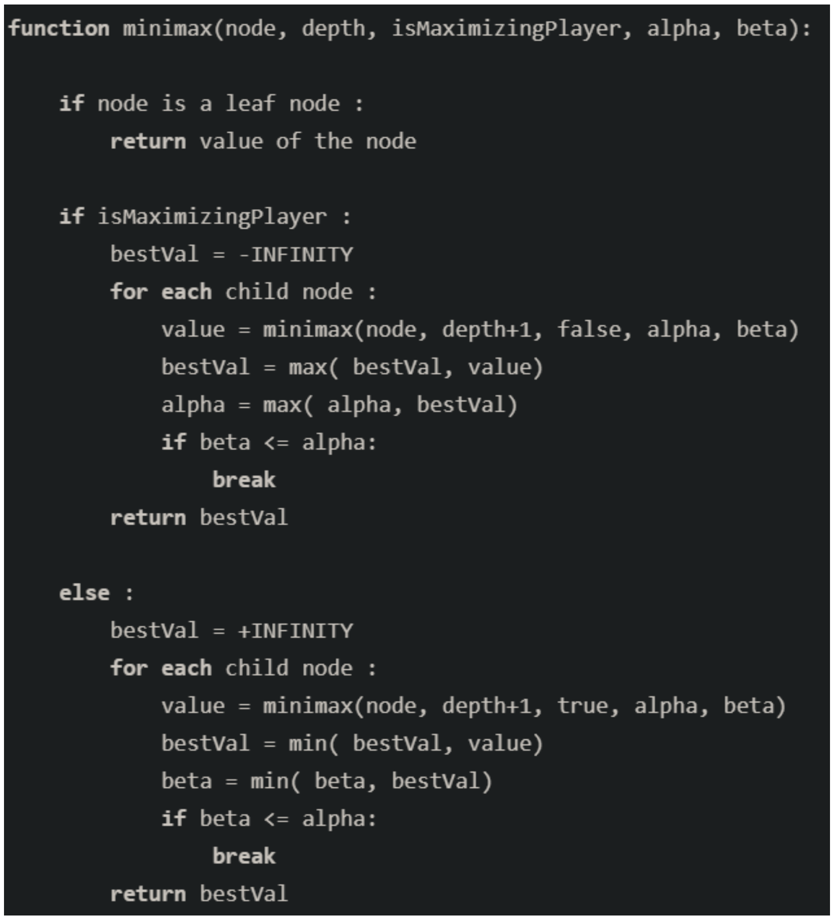
1.f Implementing a Transposition Table

Especially with iterative deepening, a board position can be evaluated more than once. The number of revaluated board states is increased as the game goes along. This creates a lot of redundant computing. We might compute minimax for a state we have just seen in the previous iteration or in bast searches. With a transposition table, we store results of running minimax on a state meaning we can potentially avoid having to re-compute the result if that state ever comes up again. I chose to store all minimax results in a Hash Map. Hash Maps are fast and very efficient for storing elements that do not rely on partial ordering. Thus, before running each iteration of Alpha-Beta Search, search the Transposition Table to see if that state was computed beforehand. If so, there is no need to calculate the resulting evaluation at that state.

2. **Theoretical Basis of The Approach and Algorithms**

2.a Alpha-Beta

We showed in class that Alpha Beta is much more cost effective than Minimax. In fact, with a good move ordering, the search depth can be halfed. Here is the algorithm I implemented



```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

2.b Evaluation Function

To allow myself to play around with multiple potential evaluation function, I decided to get each diagonal, row and column from the board into strings. After that, I could then split them based on the pieces to get the number of consecutive white pieces. I could also find out if a group of consecutive pieces was blocked meaning it couldn't lead to a win in that row/column/diagonal.
I found the algorithm to get diagonals from a square matrix at https://stackoverflow.com/questions/20420065/loop-diagonally-through-two-dimensional-array. I had to adapt it to our situation.
With this system in place, I could play around with different evaluation function for different patterns. I could lower the score for blocked patterns, only account for consecutive pieces up to 4 in a row or down to 2.

3. Advantages and Disadvantages of My Approach

3.a Advantages

My implementation of Alpha-Beta Search can reach depth 5 in the 1st move whereas the regular version of Minimax could only reach depth 2. This proves how much more efficient my implementation is compared to the basic algorithm.

3.b Disadvantages

There is no way to know if my evaluation function was the best. My algorithm might work best with another function. I wish I had had more time to test different evaluation functions against each other. Additionally, with a smaller time constraint, I think my agent would perform worse than an improved version of MCTS.

4. Future Improvements

Increase the accuracy of my heuristic function. Had move ordering before running alpha-beta to reduce runtime even more. Implement the board position into a bitboard similar to how it is said here https://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2011/Buescher_Niklas.pdf.