**Morris Game, Variant**

**Problem Statement**

The assigned task is to write programs for 4 parts of the project using a programming language stated in the project writeup. Each program gets input for two file names and the depth of the tree needed to be searched. Each program outputs the next optimal move to perform, the positions evaluated by static estimation, and the MINIMAX estimate of the position. Part 1 consists of two programs, the first being an implementation of Min Max for the opening phase of the Morris Variant game and the other being an implementation of Min Max for the mid/end game phase. Part 2 follows instructions of part 1 with alpha beta pruning implemented instead of minimax. Part 3 requires the same programs as part 1 implemented to calculate the move for the opponent. Part 4 is an improved version of part 1 where the static estimation calculation is updated.

**Approach**

I implemented the Morris Variant program in C++ using visual studio code as my IDE. I am using the g++ compiler to run my code. To construct the logic of my program I referred to the Morris-Variant handout which details pseudocode for the move generator functions advised by the professor. My program makes use of lists and arrays to hold positions. I use a char array to store the initial board position and implement a list of all possible board positions for each node in the tree.

To implement the function of Minimax, I referred to the MiniMax-Recursive handout that details a simple recursive function of MaxMin and MinMax, as well as the alpha-beta pruning version. The only difference between minimax and alpha-beta pruning is that the former implements a cut in the tree based on logical contradictions. This results in a faster execution time and a lower number of positions evaluated by static estimation.

**Examples**

Text

Description automatically generated

Figure 1

Figure 1 displays the output of MiniMaxOpening on the board position WxWxxxxxxxxxxBBxxxxxx. The program makes a move to create a closed mill and removes a black piece.

Text

Description automatically generated

Figure 2

Figure 2 displays the output of MiniMaxGame on the board position WBWBWxxBBWWWWBBxBBxWx. The program makes a move to block a black closed mill.

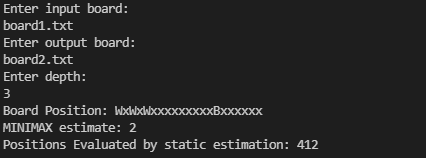


Figure 3

Figure 3 displays the output of ABOpening on the board position WxWxxxxxxxxxxBBxxxxxx. The program makes a move to create a closed mill and removes a black piece. The board position is the same as figure 1, however the positions evaluated by static estimation are less.

Text

Description automatically generated

Figure 4

Figure 4 displays the output of ABOGame on the board position WBWBWxxBBWWWWBBxBBxWx. The program makes a move to create a closed mill and removes a black piece. The board position is the same as figure 2, however the positions evaluated by static estimation are less.

**Improved Static Estimation Function**

My improved static estimation function for the MiniMax opening phase performs similar to the regular static estimation function provided for part 1, however, it also considers the number of neighboring pieces of the same color. This means that my MiniMaxOpeningImproved program will determine a better move to setup closed mills in future turns.

Text

Description automatically generated

Figure 5

Figure 5 displays the output of MiniMaxOpeningImproved on the board position BxxxWxxxxWxxxxxBxxxxx. Here we see that the program determines that a move to board location 6 is the most optimal. The output of MiniMaxOpening from part 1 is BWxxWxxxxWxxxxxBxxxxx on the same board position. I consider the output of the improved static estimation function as an improvement because a closed mill could be made in the next turn.

My improved static estimation function for the MiniMax mid/end game phase performs similar to the regular static estimation function provided in part 1, however, it also considers the number of neighboring pieces of the same color. This leads to more closed mills being setup for later turns.

Text

Description automatically generated

Figure 6

In figure 6 we see that the MiniMaxGameImproved program determines that the best move to play is at location 1 on the board position xBBBBWWWxxxWWxxxBxWWx. This location is different from the MiniMaxGame program. The MiniMaxGame program determines that a move from board location 8 to 16 is the optimal move. This move creates a closed mill, however, the move determined by the MiniMaxGameImproved program’s move determined sets the board up for two closed mill options in the next turn. This does not occur in the MiniMaxGame program.