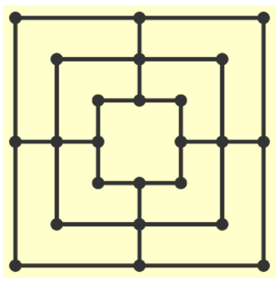
Yunchao Liu **UTD ID: 2021177759**

Project Report

1. Program (Nine Men's Morris) Description:

1.1 Game Rules:



Nine Men's Morris game is a board game between two players: White and Black. Each player has 9 pieces, and the game board is as shown. Pieces can be placed on intersections of lines(There are a total of 24 locations for pieces). The goal is to capture opponents pieces by getting three pieces on a single line (a mill). The winner is the first player to reduce the opponent to only 2 pieces, or block the opponent from any further moves. The game has three distinct phrases: opening, midgame, and endgame.

- Opening: Players take turns placing their 9 pieces – one at a time – on any vacant board intersection spot.
- Midgame: Players take turns moving one piece along a board line to any adjacent vacant spot
- Endgame: A player down to only three pieces may move a piece to any open spot, not just an adjacent one(hopping).
- Mills: At any stage if a player gets three of their pieces on the same straight board line(a mill), then one of the opponent's isolated pieces is removed from the board. An isolated piece is a piece that is not part of a mill.

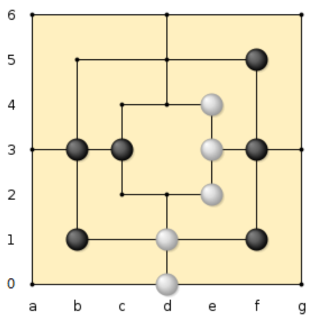
1.2 My computer program that plays Nine Men's Morris:

My program will be the user's opponent and playing White. It will take input by reading a .txt file, which contains the current board state information, and generate another .txt file showing the new board state after it makes a move by following the game rules. It uses evaluation function for deciding which move it should make. The user then can make move by editing the .txt file. However, the .txt file is not in a visual form of the board, but instead a string showing the current state(This will be further explained in the solution part below) so another Draw.java program is used for displaying the current board state in a visual form for user's convenience. The program will pop out message when either the user wins or loses.

2. Proposed Solution:

2.1 Representing board positions:

the board position is represented by an array of length 24, containing the pieces as the letter W, B, x.(The letter x stands for a "non-piece"). The array specifies the pieces starting from bottom-left and continuing left-right bottom up. Here is an illustration:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
a0	d0	g0	b1	d1	f1	c2	d2	e2	a3	b3	c3	e3	f3	g3	c4	d4	e4	b5	d5	f5	a6	d6	g6
x	W	x	B	W	B	x	x	W	x	B	B	W	B	x	x	x	W	x	x	B	x	x	x

2.2 Implementing game rules:

Game rules will be implemented by using a move generator function to generate legal moves. It will take a board position b as input and generate a list of all positions reachable by a white move(which is the color the program plays with). Then by evaluating different moves, it will output the optimal solution in the same form as input.

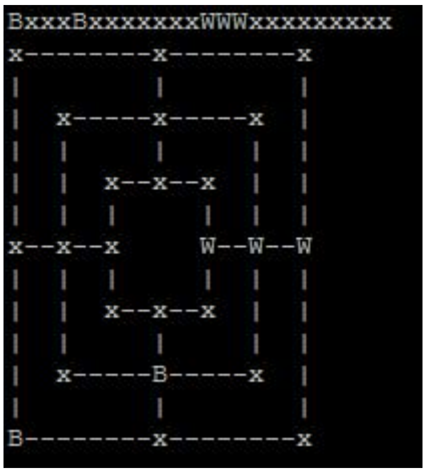
2.3 Evaluation function:

In the program, I am going to use static evaluation for each state. For static evaluation, from the computer's point of view(i.e, from the white player's pointer of view), during mid and end period of game, the value of a state is calculated from the number of white pieces, number of black pieces and the numb of moves that black can make. If the number of black pieces is less than 3, the value is 10000, which means white wins. If the number of white pieces is less than 3, the value is -10000, means white lose. If the number of black moves is 0, the value is 10000, which means black has nowhere to move and white wins. Else from the above condition, the value is 1000*(numWhitePieces – numBlackPieces) – numBlackMoves. During Opening, the value is calculated from the 1000*numWhitePieces – numBlackPieces.

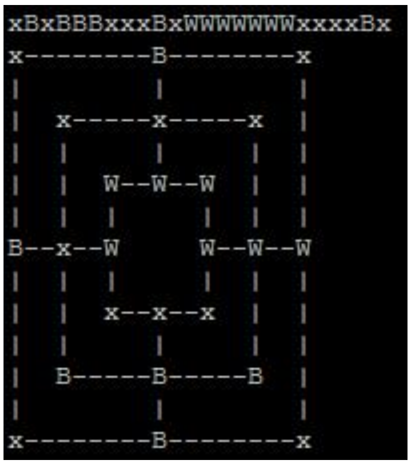
3. Full implementation details

3.1 Representing the board positions

One way of representing a board position is by an array of length 23, containing the pieces as the letters W, B, x. (The letter x stands for a "non-piece"). The array specifies the pieces starting from bottom-left and continuing left-right bottom up. Here are two examples:



```
BxxxBxxxxxxxWWWxxxxxxxx
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--W--W
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----B-----x  |
|        |        |
B--------x--------x
```



```
xBxBBBxxxBxWWWWWWWxxxxBx
x--------B--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  W--W--W  |  |
|  |  |     |  |  |
B--x--W     W--W--W
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
x--------B--------x
```

3.2 Move generator

A move generator gets as input a board position and returns as output a list of board positions that ca be reached from the input position. In the next section we describe a pseudo-code that can be used as a move generator for White.
Input: a board position b.
Output: a list L of all positions reachable by a white move.

In order for evaluate the board, the program also need to know the possible black moves. A move generator for Black is similar to that for White, with replacement of "B" to "W" and "W" to "B".

3.3 A move generator for White

A pseudo-code is given for the following move generators: **GenerateAdd**, generates moves created by adding a white piece(to be used in the opening). **GenerateMove**, generates moves created by moving a white piece to an adjacent location (to be used in the midgame). **GenerateHopping**, generates moves created by white pieces hopping(to be used in the endgame). These routines get as an input a board and generate as output a list L containing the generated positions. They require a method of generating moves created by removing a black piece from the board. We name it **GenerateRemove.**

GenerateMovesOpening

Input: a board position

Output: a list L of board positions

```
    return the list produced by GenerateAdd applied to the board.
```

GenerateMovesMidgameEndgame

Input: a board position

Output: a list L of board positions

```
    if the board has 3 white pieces Return the list produced by GenerateHopping
        applied to the board. Otherwise return the list produced by GenerateMove applied to the board.
```

GenerateAdd

Input: a board position

Output: a list L of board positions

```
    L = empty list
    for each location in board:
        if board[location] == empty{
                b = copy of board;
                b[location = W
                if closeMill( location, b )
                    generateRemove( b, L )
                else add b to L
        }
return L
```

GenerateHopping

Input: a board position

Output: a list L of board positions

```
    L = empty list
    for each location A in board
    if board[A] == W{
        for each location B in board
        if board[B] == empty{
            b = copy of board
            b[A] = empty
            b[B] = W
            if closeMill( B, b ) generateRemove( b, L )
            else add b to L
        }
    }
    return L
```

GenerateMove

Input: a board position

Output: a list L of board positions

```
L = empty list
for each location in board
if board[location] == W{
    n = list of neighbors of location
    for each j in n
    if board[j] == empty{
        b = copy of board
        b[location] = empty
        b[j] = W
        if closeMill(j,b) GenerateRemove( b, L )
        else add b to L
    }
}
return L
```

GenerateRemove

Input: a board position and a list L

Output: positions are added to L by removing black pieces

```
for each location in board:
    if board[location] == B{
        if not closeMill( location, board){
            b = copy of board
            b[location] = empty
            add b to L
        }
    }
}
if no positions were added( all black pieces are in mills) add b to L
```

neighbors and closeMIll

The proposed coding of the methods neighbors and closeMIll is by “brute force”. The idea is as follows.

neighbors

Input: a location j in the array representing the board

Output: a list of location in the array corresponding to j’s neighbors

```
switch(j){
    case j == 0 (a0): return[1,3,8]. ( These are d0, b1, a3)
    case j == 1 (d0): return [0, 4, 2]. (These are a0, d1, g0)
    etc.
}
```

closeMill

Input: a location j in the array representing the board and the board b
Output: true if the move to j closes a mill

```
C = b[j]; C must be either W or B. Cannot be x.
switch(j){
    case j == 0(a0) :
        if ( b[1] == C and b[2] ==C) or ( b[3] == C and b[6] == C ) or (b[8] == C and b[20] == C ) return true
        else return false
    case j == 1(d3):
        if(b[0] ==C and b[2] == C ) return true
        else return false
    etc.
}
```

3.4 Static estimation

The following static estimation functions are proposed. Given a board position b compute:
numWhitePieces = the number of white pieces in b.
numBlackPieces = the number of black pieces in b.
L = the MidgameEndgame positions generated from b by a black move.
numBlackMoves = the number of board positions in L.

A static estimation for MidgameEndgame:

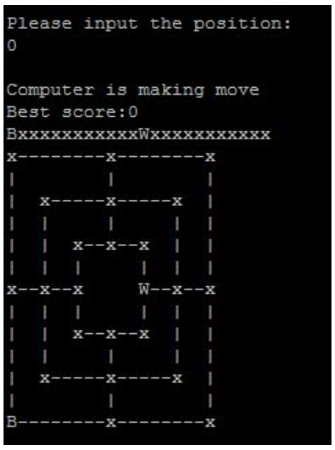
```
if( numBlackPieces<=2) return( 10000)
else if( numWhitePieces <=2) return( -10000)
else if ( numBlackMoves == 0 ) return( 10000)
else return( 1000*( numWhitePieces – numBlackPieces ) – numBlackMoves)
```

A static estimation for Opening:

```
return( 1000*( numWhitePieces – numBlackPieces ) )
```

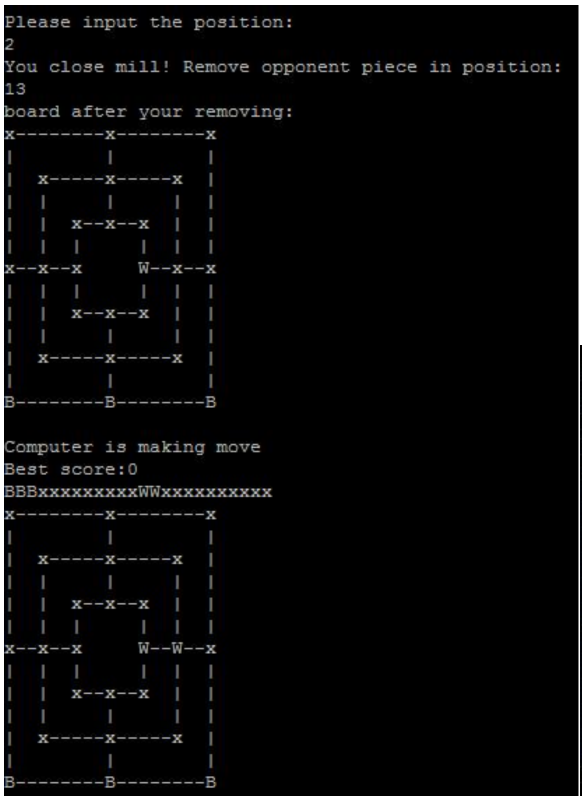
4. Examples

The program will begin with user input a position for B and the computer will play with W. The program will pop out ”Invalid Input” if the input is invalid “. Then user and computer will play in turn. Some situation will be illustrated as follows:
Opening:

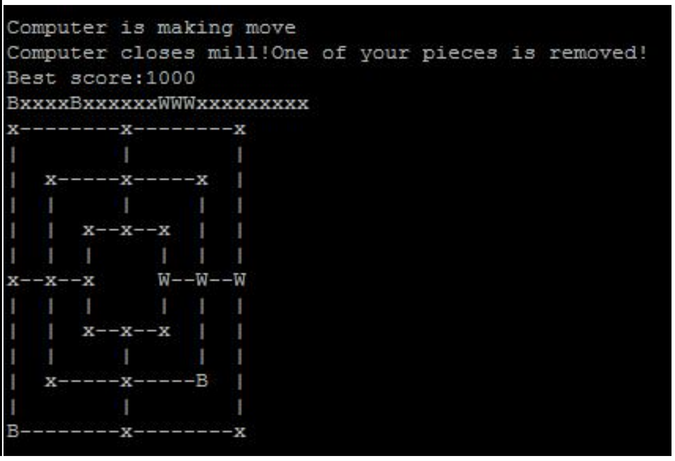


```
Please input the position:
0

Computer is making move
Best score:0
BxxxxxxxxxxxWxxxxxxxxxxx
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--x--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----x-----x  |
|        |        |
B--------x--------x
```

(1)User and computer each makes a move



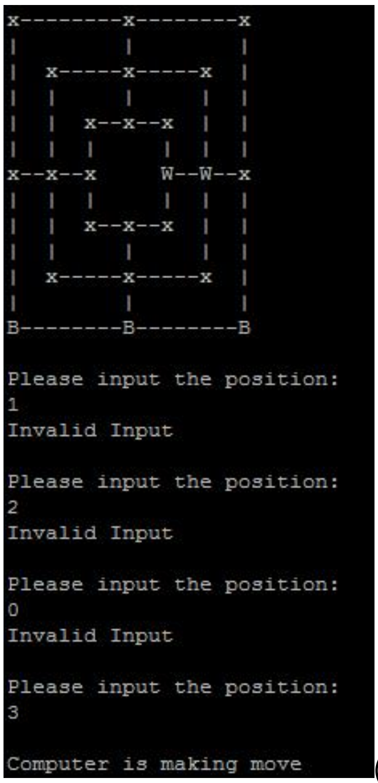
```
Please input the position:
2
You close mill! Remove opponent piece in position:
13
board after your removing:
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--x--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----x-----x  |
|        |        |
B--------B--------B

Computer is making move
Best score:0
BBBxxxxxxxxxWWxxxxxxxxxx
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----x-----x  |
|        |        |
B--------B--------B
```



```
Computer is making move
Computer closes mill!One of your pieces is removed!
Best score:1000
BxxxxBxxxxxxWWWxxxxxxxxx
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--W--W
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----x-----B  |
|        |        |
B--------x--------x
```

(2) User closes mill

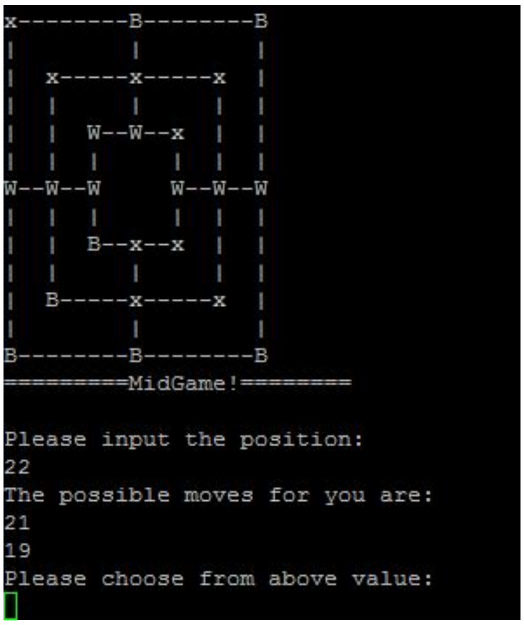
(3)Computer closes mill



```
x--------x--------x
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--x     W--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  x-----x-----x  |
|        |        |
B--------B--------B

Please input the position:
1
Invalid Input

Please input the position:
2
Invalid Input

Please input the position:
0
Invalid Input

Please input the position:
3

Computer is making move
```

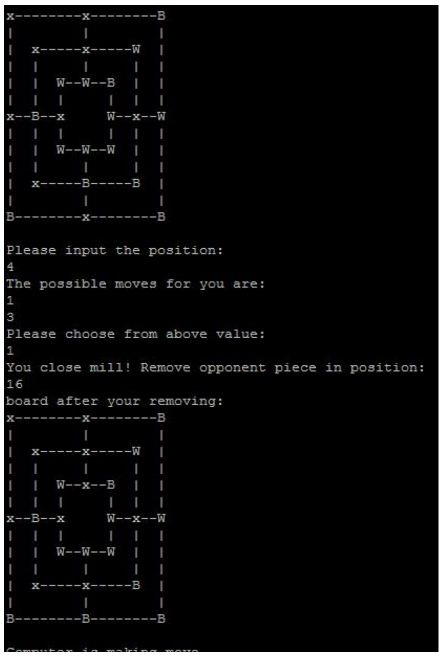
(4) Invalid Input

Midgame:



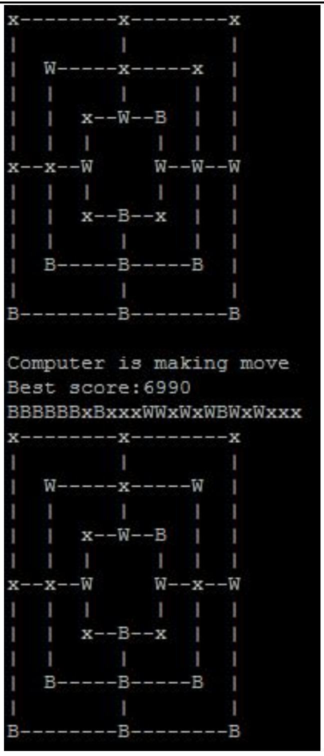
```
x--------B--------B
|        |        |
|  x-----x-----x  |
|  |     |     |  |
|  |  W--W--x  |  |
|  |  |     |  |  |
W--W--W     W--W--W
|  |  |     |  |  |
|  |  B--x--x  |  |
|  |     |     |  |
|  B-----x-----x  |
|        |        |
B--------B--------B
=========MidGame!========

Please input the position:
22
The possible moves for you are:
21
19
Please choose from above value:
```

(5) User makes a move



```
x--------x--------B
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  W--W--B  |  |
|  |  |     |  |  |
x--B--x     W--x--W
|  |  |     |  |  |
|  |  W--W--W  |  |
|  |     |     |  |
|  x-----B-----B  |
|        |        |
B--------x--------B

Please input the position:
4
The possible moves for you are:
1
3
Please choose from above value:
1
You close mill! Remove opponent piece in position:
16
board after your removing:
x--------x--------B
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  W--x--B  |  |
|  |  |     |  |  |
x--B--x     W--x--W
|  |  |     |  |  |
|  |  W--W--W  |  |
|  |     |     |  |
|  x-----x-----B  |
|        |        |
B--------B--------B

Computer is making move
```

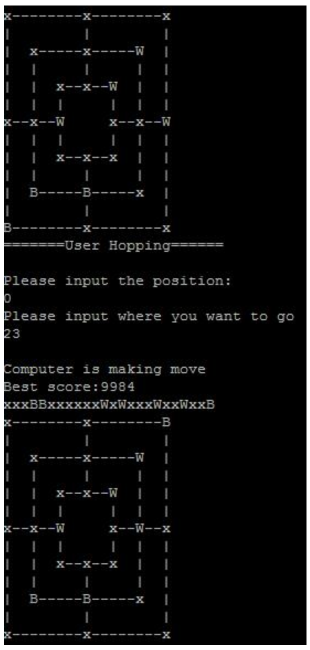
(6) User closes mill



```
x--------x--------x
|        |        |
|  W-----x-----x  |
|  |     |     |  |
|  |  x--W--B  |  |
|  |  |     |  |  |
x--x--W     W--W--W
|  |  |     |  |  |
|  |  x--B--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
B--------B--------B

Computer is making move
Best score:6990
BBBBBBxBxxxWWxWxWBWxWxxx
x--------x--------x
|        |        |
|  W-----x-----W  |
|  |     |     |  |
|  |  x--W--B  |  |
|  |  |     |  |  |
x--x--W     W--x--W
|  |  |     |  |  |
|  |  x--B--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
B--------B--------B
```

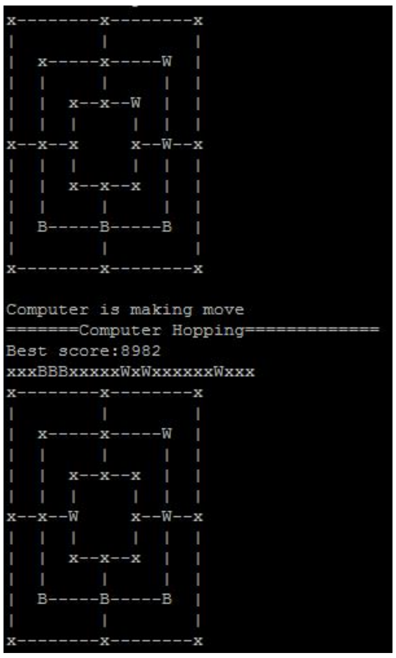
(7) Computer makes a move

EndGame:



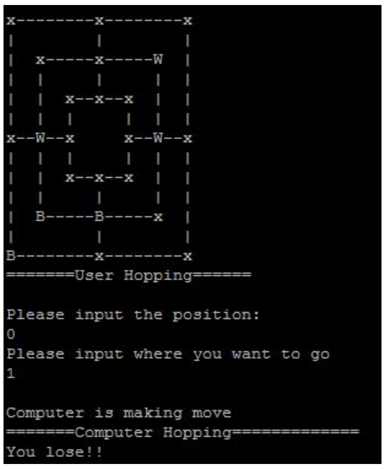
```
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--W  |  |
|  |  |     |  |  |
x--x--W     x--x--W
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----x  |
|        |        |
B--------x--------x
=======User Hopping======

Please input the position:
0
Please input where you want to go
23

Computer is making move
Best score:9984
xxxBBxxxxxxWxWxxxWxxWxxB
x--------x--------B
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--W  |  |
|  |  |     |  |  |
x--x--W     x--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----x  |
|        |        |
x--------x--------x
```

(8) User hopping



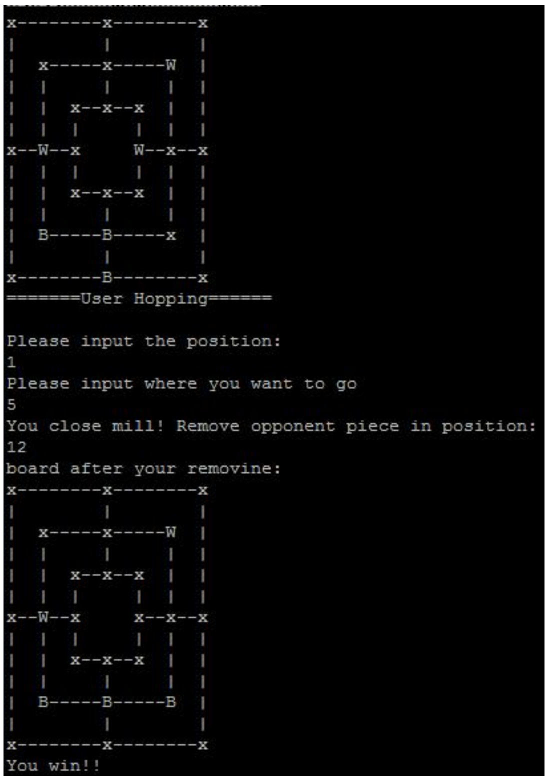
```
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--W  |  |
|  |  |     |  |  |
x--x--x     x--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
x--------x--------x

Computer is making move
=======Computer Hopping=============
Best score:8982
xxxBBBxxxxxWxWxxxxxxWxxx
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--x--W     x--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
x--------x--------x
```

(9)Computer Hopping



```
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--W--x     x--W--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----x  |
|        |        |
B--------x--------x
=======User Hopping======

Please input the position:
0
Please input where you want to go
1

Computer is making move
=======Computer Hopping=============
You lose!!
```

(10) Computerwins



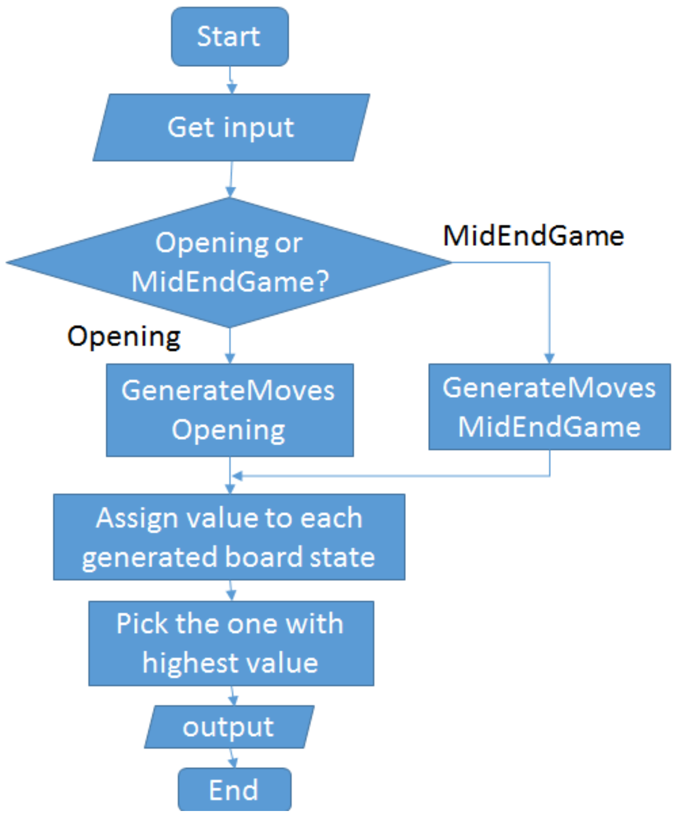
```
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--W--x     W--x--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----x  |
|        |        |
x--------B--------x
=======User Hopping======

Please input the position:
1
Please input where you want to go
5
You close mill! Remove opponent piece in position:
12
board after your removine:
x--------x--------x
|        |        |
|  x-----x-----W  |
|  |     |     |  |
|  |  x--x--x  |  |
|  |  |     |  |  |
x--W--x     x--x--x
|  |  |     |  |  |
|  |  x--x--x  |  |
|  |     |     |  |
|  B-----B-----B  |
|        |        |
x--------x--------x
You win!!
```

(11)User wins

5. Programming tools (including third party software tools used)

This project runs in the cs1 server of ECS, which is a Linux environment and was written in Java. No other third party software was used.

6. Architectural Diagram



7. Results

The program implements game rules. And the computer will play with user and tries to stop user from winning!

8. A summary of the problems encountered during the project and how these issues were resolved

At first, I was using a file to read in input board. But later I found it was too bothering to edit file every time user wants to input. So I create a while loop and in that, user and computer will take turns playing, which gives a much better user experience. Another problem was that for static estimation, sometimes even closing mill will not get higher score, which leads the computer not to choose to close mill. I solved this problem by giving closing mill a much higher score thus computer will have a higher chance of closing mill.

9. Pending Issues

The static evaluation function can only look ahead one move ahead, hence it will not be intelligent enough to prevent user from closing a mill, or detect some tricks by human which makes more than one move to close a mill. Also, because of this foreseeing limit, it cannot develop complicated, or tricky strategy in order to close a mill.

10. Potential Improvements

The above mentioned issues are mainly caused by the fact that static evaluation function only exam the benefit one step ahead, which makes it impossible to think further. So the improvement should address this problem. One way to improve it is to use MiniMax search method, which examines all possible solution systematically and finds the best path towards winning. To reduce the search time, we can also use Alpha-Beta pruning algorithm to avoid evaluation those unneeded.