CS 6364 Artificial Intelligence - Final Project Report

Vedang Wartikar
VPW210000

13 July 2023

Contents

1 MiniMax Algorithm

1.a Opening Phase

1.a.1 White's Play

Consider the example, wherein the White player has to place its next piece. The `MinimaxOpening.py` optimally places the W piece, forms a mill, and removes one of the Black's pieces.

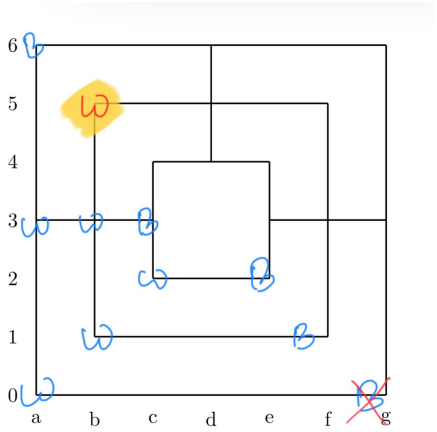

Figure 1: MiniMax Opening Move - White

```
Input position: WBWBWBWWBxxxxxxxxxBxx
Ply: 2

Board Position: WxWBWBWWBxxxxxxWxxBxx.
Positions evaluated by static estimation: 155.
MINIMAX estimate: 1.
```

1.a.2 Black's Play

Consider the example in which the Black has to place its 4th piece. There is a potential mill position for the White player. The `MinimaxOpeningBlack.py` correctly places the B piece so that the White player cannot form a mill.

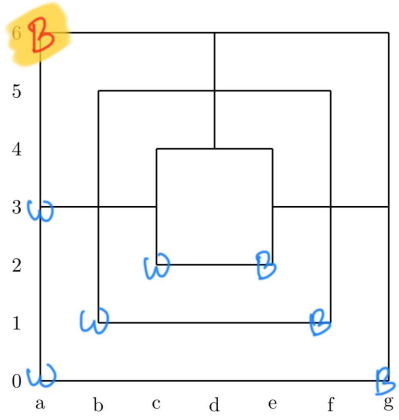

Figure 2: MiniMax Opening Move - Black

```
Input position: WBWBWBWxxxxxxxxxxxxxxx
Ply: 2

Board Position: WBWBWBWxxxxxxxxxxxBxx.
Positions evaluated by static estimation: 221.
MINIMAX estimate: 1.
```

1.b MidGame and Endgame Phase

1.b.1 Move

In this example, the White piece shifts its piece to the right forming a vertical mill, and removes one of the pieces of the Black player. It is played using `MinimaxGame.py`

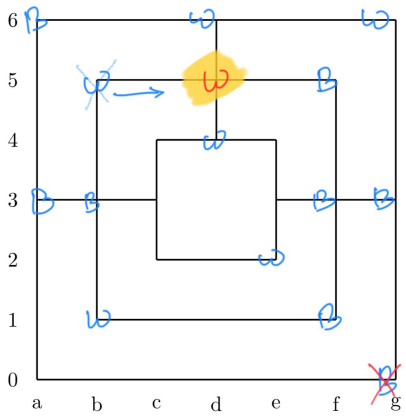

Figure 3: MiniMax Game - Move

```
Input position: xBWBxWBBxxBBxWxWxBBWW
Ply: 2

Board Position: xxWBxWBBxxBBxWxxWBBWW.
Positions evaluated by static estimation: 90.
MINIMAX estimate: -1017.
```

1.b.2 Hop

In this example, the White piece should not close a mill. If it closes a mill and removes a Black piece, there are only 3 'B' pieces and the Black player can hop and win the game. The white player correctly places its piece considering the future possibilities.

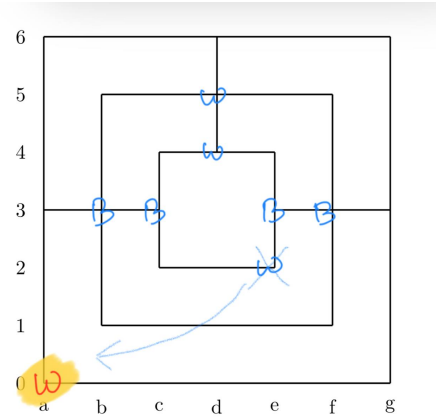

Figure 4: MiniMax Game - Hop

```
Input position: xxxxxWxBBBBxxWxxWxxxx
Ply: 4

Board Position: WxxxxxxBBBBxxWxxWxxxx.
Positions evaluated by static estimation: 542109.
MINIMAX estimate: -45.
```

2 Alpha-Beta Pruning Algorithm

2.a Opening Phase

For the following input board, the MiniMax and Alpha-Beta Pruning algorithms (ply 3) smartly place the White player's piece on the top right corner making it a possibility of at least 1 mill irrespective of the Black player's next move. The positions evaluated by the static estimation function and the Minimax estimate have been compared for both algorithms. We can clearly see that Alpha-Beta Pruning is very efficient in minimizing the unnecessary computations done by the MiniMax algorithm.
After running `MiniMaxOpening.py` and `ABOpening.py` on the following input board position, we get,

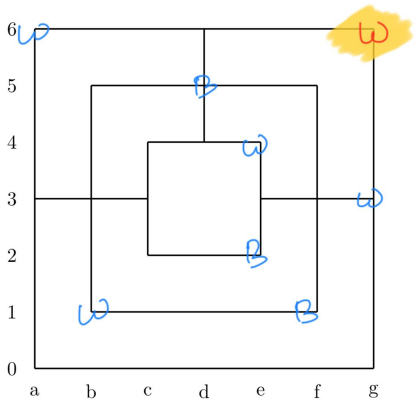

Figure 5: MiniMax VS Alpha-Beta Pruning - Opening

```
Input position: xxWBxBxxxxxWxxWxBxWxx
Ply: 3

-------- MiniMax Algorithm --------
Board Position: xxWBxBxxxxxWxxWxBxWxW.
Positions evaluated by static estimation: 2616.
MINIMAX estimate: 3.

-------- Alpha-Beta Pruning --------
Board Position: xxWBxBxxxxxWxxWxBxWxW.
Positions evaluated by static estimation: 875.
MINIMAX estimate: 3.
```

2.b MidGame and Endgame Phase

For the following input board, using MiniMax and Alpha-Beta Pruning algorithms (ply 5) the White player's piece at position 3 is moved to the top. The move is fairly simple, but as seen below the positions evaluated by Alpha-Beta Pruning are **360** and that of MiniMax is **4643**. Therefore, **Alpha-Beta is 13x more efficient that the MiniMax** algorithm in terms of the positions being evaluated.
After running `MiniMaxGame.py` and `ABGame.py` on the following input board position, we get,

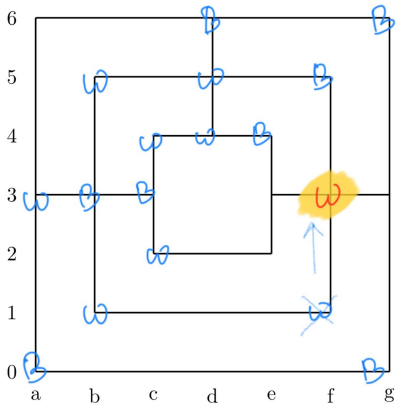

Figure 6: MiniMax VS Alpha-Beta Pruning - Game

```
Input position: BBWWWxWBBxxxWWBWWBxBB
Ply: 5

-------- MiniMax Algorithm --------
Board Position: BBWxWxWBBxWxWWBWWBxBB.
Positions evaluated by static estimation: 4643.
MINIMAX estimate: -1010.

-------- Alpha-Beta Pruning --------
Board Position: BBWxWxWBBxWxWWBWWBxBB.
Positions evaluated by static estimation: 360.
MINIMAX estimate: -1010.
```

3 Improved MiniMax Algorithm

3.a Opening Phase

After simulating 3 moves for each White and Black player, the White places its piece such that Black cannot avoid the formation of at least one White's mill in the next moves. The basic MiniMax algorithm for Black player does not consider such scenarios at the intersection and places its pieces in position which is next available.

• The static estimation function is tweaked to form a **potential double mill at an intersection**.

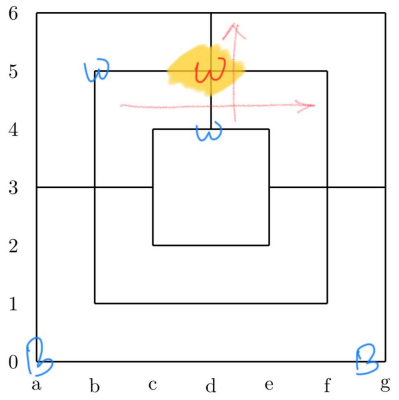

Figure 7: Improved MiniMax Opening Move

```
Input position: BBxxxxxxxxxxxWxWxxxxx
Ply: 2

Board Position: BBxxxxxxxxxxxWxWWxxxx.
Positions evaluated by static estimation: 272.
MINIMAX estimate: 3.
```

3.b MidGame and Endgame Phase

In the example below, after running the `MiniMaxGameImproved.py`, the White player plays a better move in terms of removing the Black player's piece. Optimally, it forms a mill (same as the traditional MiniMax algorithm) but when removing a piece it gives importance to the formation of a potential opponent's mill (highlighted by a red cross). As shown in the diagram below, the traditional MiniMax Algorithm removes the piece highlighted by the color yellow which certainly does not eliminate the future possibility of the Black piece forming a mill. • The static estimation function is updated to have a **penalizing factor for the opponent's potential mill**.

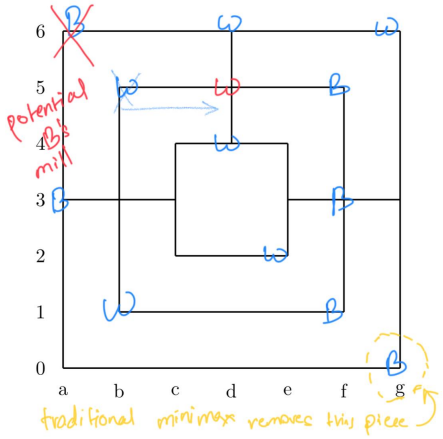

Figure 8: Improved MiniMax Game Move

```
Input position: xBWBxWBxxxBxxWxWxBBWW
Ply: 3

Board Position: xBWBxWBxxxBxxWxxWBxWW.
Positions evaluated by static estimation: 1022.
MINIMAX estimate: -610.
```