William Herr 998103103
Bryce Stovall 912179113

# Project 2

Part 1) Evaluation Function:

A. Our evaluation function has an offensive and defensive strategy. In the offensive strategy we consider only MAX's tokens where there are either 2 in a line, 3 in a line, or 3 in a line and a 4th empty space. We split the cases into a possible win that is vertical(up case), horizontal(left or right cases), or diagonal(up left or up right cases). For each case we increment a variable "possibleWins" when there are matches. We increment it depending on the value in a matrix and use a multiplier depending on how many matches in a line. The matrices are "winDiagonalMatrix", "winUpMatrix", and "winLRMatrix". The Diagonal matrix is 2 dimensional; its entries represent the number of ways to win in either up left or up right. The Up and LR matrices are 1 dimensional; their entries represent the number of ways to win either going vertically or horizontally. The defensive strategy considers only MIN's tokens when their are 2 inline and 1 side is empty and the other is filled with MAX's token. This effectively favors blocking a setup that would give MIN two ways to win on the next round. We increment "possibleWins" by 50 in an attempt to dominate other moves.

B)

Let Utility be the summation of all the evaluation cases
Let C and R be the column and row we are evaluating respectively.

```
Let winRDiagonalMatrix = {
                {0, 0, 0, 1, 1, 1, 1},
                {0, 0, 1, 2, 2, 2, 1},
                {0, 1, 2, 3, 3, 2, 1},
                {1, 2, 3, 3, 2, 1, 0},
                {1, 2, 2, 2, 1, 0, 0},
                {1, 1, 1, 1, 0, 0, 0}
        }
Let winLDiagonalMatrix = {
                {1, 1, 1, 1, 0, 0, 0},
                {1, 2, 2, 2, 1, 9, 0},
                {1, 2, 3, 3, 3, 1, 0},
                {0, 1, 2, 3, 3, 2, 1},
                {0, 0, 1, 2, 2, 2, 1},
                {0, 0, 0, 1, 1, 1, 1}
        }
Let winUpMatrix = {1, 2, 3, 3, 2, 1}
Let winLRMatrix = {1, 2, 3, 4, 3, 2, 1}
```

Let UpAttack(C,R) be a function that returns the sum of winUpMatrix[R] (if 2 tokens in a line), 2 * winUpMatrix[R] (if 3 tokens in a line), 4 * winUpMatrix[R] (if 3 tokens in a line and a 4th empty token)

Let LeftAttack(C,R) and RightAttack(C,R) be functions that return the sum of winLRMatrix[R] (if 2 tokens in a line), 2 * winLRMatrix[R] (if 3 tokens in a line), 4 * winLRMatrix[R] (if 3 tokens in a line and a 4th empty token) in their respective directions.

Let DLAttack(C,R) be a function that return the sum of winLDiagonalMatrix[R][C] (if 2 tokens in a line), 2 * winLDiagonalMatrix[R][C] (if 3 tokens in a line), 4 * winLDiagonalMatrix[R][C] (if 3 tokens in a line and a 4th empty token) in their respective directions.

Let DRAttack(C,R) be a function that return the sum of winRDiagonalMatrix[R][C] (if 2 tokens in a line), 2 * winRDiagonalMatrix[R][C] (if 3 tokens in a line), 4 * winRDiagonalMatrix[R][C] (if 3 tokens in a line and a 4th empty token) in their respective directions.

Let UpBlock(C,R), LeftBlock(C,R), RightBlock(C,R), DLBlock(C,R), and DRBlock(C,R) be functions that return 50 if there are 2 MIN tokens in a line and 1 MAX token on the left (logical) or right side.

Then our numerical expression is:

```
For all tiles C, R:
    If tile at (C,R) is MAX:
        Utility += UpAttack(C,R) + LeftAttack(C,R) + RightAttack(C,R) +
        DLAttack(C,R) + DRAttack(C,R)
    Else if tile at (C,R) is MIN:
        Utility += UpBlock(C,R) + LeftBlock(C,R) + RightBlock(C,R) + DLBlock(C,R)
        + DRBlock(C,R)
```

C) In the following example MAX is 'X' and MIN is 'O'. In the following board state MIN has just played.

```
.......
.......
...x...
...x...
..oo...
.oox...
```

MAX then places a token at column 3 and row 4 (zero indexing)

```
.......
...x...
...x...
...x...
..oo…
.oox...
```

The first thing that our evaluation function would do for this move is check to see if the game is over; if it is not we continue to check the offensive and defensive utility. We start at the bottom left of the board and move right checking to see how many of our tokens are in a line. The first token we see is the ‘X’ in the middle bottom row. Because there are no tokens to the right of this ‘X’ for 3 spaces we add (4 + 4*2 + 4*4) to our utility--note that there is an enemy token to the left so we don’t pursue any utility in that direction. These numbers come from the fact we value empty spaces next to our own tokens because they can lead to a win. Also because there is nothing in the upper right direction we add (1 + 2*1 + 4*1) to our utility.

Next we look at the token in column 3, row 2. We see that there is one token above it so we add a score of 3. Next we see that there is another one above that (so 3 in a row) and we add a score of 2*3 to the utility. Then, because there is an empty space above our 3 tokens we add a score of 4*3 to the utility. However, from this token we can also win in the upper left and upper right directions. Because of this we also add 2*(3 + 2*3 + 4*3) to our utility. Furthermore, it is also possible to win in the left and right directions because of this we add a score of 2*(4 + 2*4 + 4*4) to our utility.

Next, we observe the token in column 3 row 4, because there is one token above it we add 3 the utility and because there is a blank spot above that we add a 2*3 to the utility. From this token we can also win in the left and right directions so therefore we add 2*(4+2*4+4*4). There is also no token for 2 spaces in the upper left and upper right directions so we add 2*(3+2*3) to the utility.

Finally observe the token in column 3 row 5 can also win in both the right and left directions so we add a score of 2*(4+2*4+4*4). It also has a blank space above it so we add a score of 2. There is also a blank space space towards the upper left and upper right so we add 2*(2).

Next we look at the defensive score of our last move.

First we look at the ‘O’ at column 1 row 0. We see that the right of this ‘O’ there is another ‘O’ and to the right of that there is an ‘X’. Because we value this as a “block” we add 50 points to the utility. In addition, we can see that there is another ‘O’ at column 2 row 1 (in the diagonal right direction) and a ‘X’ in column 3 row 2 (one more in the diagonal right direction) we count this as a block and add 50 points to utility.

Next we look at the ‘O’ at column 2 row 0 we first see if there is anything to the left of this and see that our opponent has a ‘O’ in that spot. We then check to see if there is one of our ‘X’ to the right of this ‘O’ because there is we add another score of 50 to the utility. In this way we are double counting the blocking because we value a board state where the opponent has less possible wins.

For the other ‘O’s there is no blocking done by our ‘X’s so they do not contribute to the defensive score.

Part 3)

```
Player 1: StupidAI
Player 2: minimax_valhalla
x......
x......
o......
x......
x......
x..oooo
Player 2 won
```

```
Player 1: StupidAI
Player 2: minimax_valhalla
x......
x......
o......
x......
x......
x..oooo
Player 2 won
```

```
Player 1: StupidAI
Player 2: minimax_valhalla
x......
x......
o......
x......
x......
x..oooo
Player 2 won
Player 1: minimax_valhalla
Player 2: StupidAI
.......
.......
.......
o......
o......
o..xxxx
Player 1 won

Player 1: minimax_valhalla
Player 2: StupidAI
.......
.......
.......
o......
o......
o..xxxx
Player 1 won
```

---

```
Player 1: RandomAI
Player 2: minimax_valhalla
Seed 1
...o...
...o...
...o...
...o...
x..xx..
x.xoxo.
Player 2 won
Player 1: RandomAI
Player 2: minimax_valhalla
Seed 2
.......
.......
...o...
...o...
...o...
xx.ox.x
Player 2 won

Player 1: RandomAI
Player 2: minimax_valhalla
Seed 3
.......
.......
...o...
...o...
xx.o...
xxoo.x.
Player 2 won

Player 1: RandomAI
Player 2: minimax_valhalla
Seed 4
.......
.......
...x...
..oooo.
x.ooxx.
xxxoox.
Player 2 won

Player 1: RandomAI
Player 2: minimax_valhalla
Seed 5
.......
.......
...o...
...o...
...o.x.
x..oxx.
Player 2 won
```

---

For the following games we did not know if -seed was working in the terminal so we changed the seed in Main.java, MonteCarloAI.java and on the command land with -seed

Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 1

```
.......
..oo...
..xox..
..xxo..
..xoxo.
.ooxxox
```

Player 2 won

Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 2

```
o.xoox.
o.xxoo.
oooxxxo
xxoxoox
xoxooxo
xoxxoxx
```

Player 2 won

Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 3

```
o.xoox.
o.xxoo.
oooxxxo
xxoxoox
xoxooxx
xoxxoxo
```

Player 2 won

```
Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 4
.......
..oo...
..xox..
..xxo..
..xoxo.
.ooxxox
Player 2 won
```

```
Player 1: minimax_valhalla
Player 2: MonteCarloAI
Seed 5
.......
..oo...
..xox..
..xxo..
..xoxo.
.ooxxox
Player 2 won
```

```
Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 6
.......
..oo...
..xox..
..xxo..
..xoxo.
.ooxxox
Player 2 won
Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 7
o.xoo.o
x.xxo.x
oooxo.o
xxoxx.x
xoxoo.o
xoxxo.x
Player 2 won


Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 8
.......
.......
..xoo..
x.oxo..
x.xoo..
xoxxo..
Player 2 won


Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 9
.......
..oo...
..xox..
..xxo..
..xoxo.
.ooxxox
Player 2 won

Player 1: MonteCarloAI
Player 2: minimax_valhalla
Seed 10
.o.ooxx
.xxxxoo
.oooxxo
.xxxoox
oxxoxox
```

```
oooxxox
Player 1 won
```

We won 18/20 games.

Part 4) In order to allow our alpha beta pruning to be most effective we first check the columns that will most likely return the best score first. Because of the board's dimensions, it's always advantageous to place tokens towards the middle of the board instead of the sides. For example, we should always put a token in the middle column as the first move because it can win in both the right and left directions. Therefore, the minValue function (which prunes more when it finds larger values first) first checks the middle columns which have a higher chance of returning a better score. Then, the maxValue function (which prunes more when it finds smaller values first) first checks the columns on the sides. With this strategy we eliminate more branches of the tree by checking moves that are better for each respective function.