ECS 170 Program 2 Write up

Members: Haozhe Gu 999200555, Xie Zhou 912143385

Part I

a) In my evaluation function, there are two main components.

- Position Score
- Connection Score

Each tile in the 7*6 game board has a specific position score for its position. Tiles in the bottom middle have the highest score because it's easy to connect to other point than other tiles. It's quite intuitive that the tiles at the corner are less likely to help player win the game than the tiles in the middle. The specific scores also follow normal distributions.

The connection score is the score for consecutive (In all eight directions) tiles that might lead to four (the goal state of connect 4). There are three weights for different number of current consecutive tiles because the more consecutive tiles player currently have, the more chance this player will win the game.

b) The expression of Evaluation function is then:

$$f(State = S|Player = x)$$

$$= Position(x, S) - Position(y, S) + Connection(x, S) - Connection(y, S)$$

$$where(x, y)\ \ are\ \ players$$

$$Position(x, S) = for\ \ all\ \ tiles\ \ in\ \ S$$

$$Positon += EvaluationTable(X's\ \ tile)$$

$$Positon -= EvaluationTable(Y's\ \ tile)$$

$$The\ \ Evaluation\ \ Table\ \ is:$$

```
private static int[][] evaluationTable =
        {{3, 4, 5, 7, 5, 4, 3},
        {4, 6, 8, 10, 8, 6, 4},
        {5, 8, 11, 13, 11, 8, 5},
        {5, 8, 11, 13, 11, 8, 5},
        {4, 6, 8, 10, 8, 6, 4},
        {3, 4, 5, 7, 5, 4, 3}};
```

$$Connection(x, S)$$

$$= offset(plus\ \ or\ \ minus) * Weight(three\ \ weights)$$

$$* Possible\ \ point\ \ to\ \ extend\ \ to\ \ 4\ \ points\ \ in\ \ all\ \ 8\ \ directions(With\ \ current\ \ length\ \ 1,2,3)$$

Example:

```
. . . . . . .
. . . . . . .
. . . . . . .
. . . x . . .
. . . o . . .
. . ox . . .
```

$$f(State = S|Player = O) = Position(O, S) - Position(X, S)$$

$$+Connection(O, S) - Connection(X, S)$$

$$= (7 + 8) - (5 + 11) + \text{weight}_2 * 1 - 0$$

$$= \text{Weight}_2 - 1$$

Part II

File: minimax_<ash_bo>.java

Part III

We win all 20 games against three different AI

-With Random AI,We play as player 1.

```
.......
.......
.......
...x...
...o..o
xxxx.oo
Player 1 won
```

```
.......
.......
.......
.......
..o..o.
xxxx.o.
Player 1 won
```

```
.xx....
.ox....
.ox....
.ox.o..
.xoxo..
xoxxoo.
Player 1 won
```

```
.......
.......
...o...
...x...
.o.o...
xxxx.o.
Player 1 won
```

```
.......
...o...
...o...
...x...
..oo...
xxxx...
Player 1 won
```

-With Stupid AI, we play as player 1

```
x......
o......
x......
o......
o......
ooxxxx.
Player 1 won
```

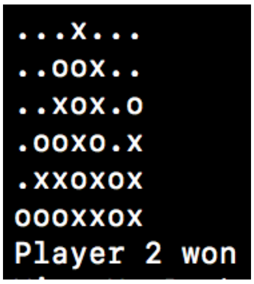
```
x......
o......
x......
o......
o......
ooxxxx.
Player 1 won
```

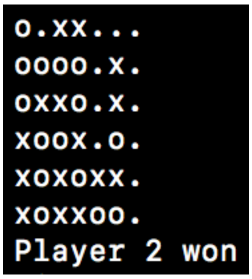
```
x......
o......
x......
o......
o......
ooxxxx.
Player 1 won
```

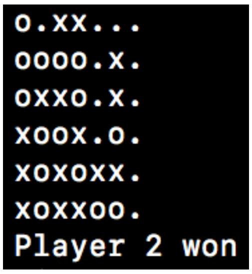
```
x......
o......
x......
o......
o......
ooxxxx.
Player 1 won
```

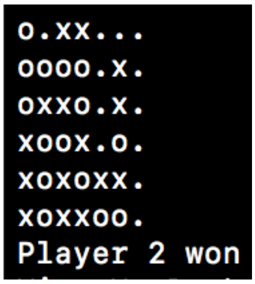
```
x......
o......
x......
o......
o......
ooxxxx.
Player 1 won
```

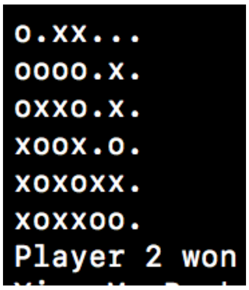
-With MonteCarlo AI, we play as player 2. Seed from 1-10

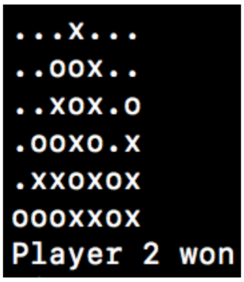


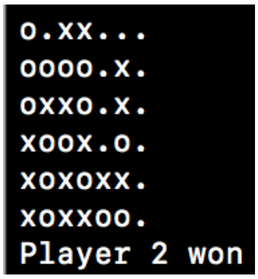


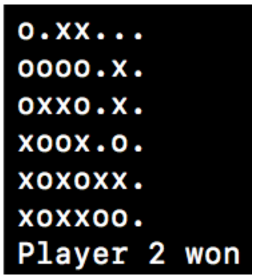


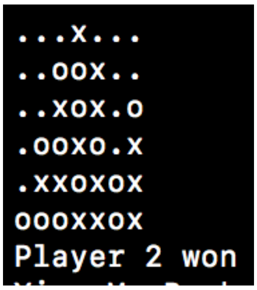


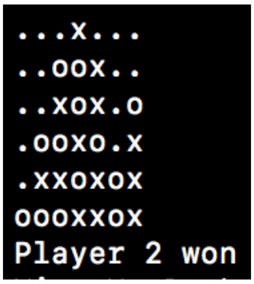












Part IV

Because of the complexity of the game tree and the time limit. We initially can only explore with depth 5. The more state we can explore in the time limit, the better choice we might make to get to the goal state. This is the motivation of our move ordering for the alpha-beta pruning ---- To prune maximum number of nodes from the game tree.

We have two successor functions that is called in MAX function and MIN function.

generateMoves_max
generateMoves_min

This two function along with helper function helped to generate and order moves as successor function. E.g.:
sort_moves,
reverse_moves,
moveComparator implements Comparator<move_value>…

We ordered the moves in Max function descending and ordered the moves in Min function ascending. The criterion of the ordering is the score of evaluation function of that state. The intuitive behind this is that if a state is good at current depth, it is more likely that its children states are good at their depth as well.