Report for Project 1

Author: Yihong Zhou & Mengwen Li

Describe an experiment you did. That could be comparing different evaluation functions. Your grade will depend on the quality of your explanation not on whether your idea panned out.

Originally, the plan decideed only using three parts to calculate heuristic:connect-N in horizontal line, vertical line and diagnal line. But when actually implementing the algorithm, details were added and a lot changes happened to the implementation.

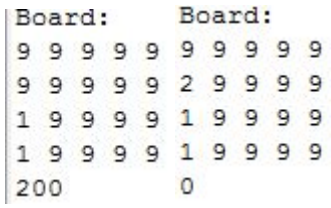
The designed heuristic function checked continuous horizontal connect-N, connect-(N-2),connect-(N-3)....to connect-2. How many pieces aligned together decides the importance of the connection. However, we found out, only giving values to aligned piece we found was not enough. Some aligned pieces are dead since they are restricted by the piece of opponents or the boundary of the board. The existance of those pieces are less important than the pieces with open space beside them. Thus, we improve the calculation by including the empty space around the aligned piece so that with more free space, the align piece would have a higher score.

```
Board:      Board:
9 9 9 9 9   9 9 9 9 9
9 9 9 9 9   9 9 9 9 9
9 9 9 9 9   9 9 9 9 9
9 1 1 9 9   2 1 1 2 9
200         0
```

The comparison of free aligned pieces(connect-3)

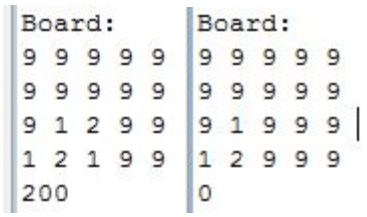
Discreet implementation was not considered first, but we figured out it has to be made. Because a big winning chance would escape if one pieces can complete a line though it is in a middle position and no algorithm can detect such a flaw before the piece compacts the line. The last step of filling a middle position and chekcing the solid pile below the poistion was exetremly critical so we distributed a great mount of score for it.

Vertical checking also considered the free top with higher value comparing to ordinary scanning and count approach. If a opponent piece lies on the top of a stack of player piece, we will discard the pile since we can’t add our piece to it anymore.



The comparison of free hat score and discard pile score(connect-3)

Diagonal checking includes positive diagonal checking and negative diagonal checking. They are very similar but was implemented in an opposite way. The freedom of piece were consider in diagonal checking. The algorithm checks the top and bottom of aligned piece in diagonal and make sure the open space was provided by the pieces underneath. A solid support under the a development of diagonal line would be given higher evaluation.



Diagnally line has a solid ground support and which doesn't(connect-3)

Opponents' behavior was measured with same techique. The score will be greatly effected if opponent is strong. Player could get negative score when the decision is not reasonablly balanced.

Duplicate pieces are ignored such as the piece was been calculated in connet-2 but may be calculated again in connect-3. Because connect-(n-1) or connect-(n-2) or more that is less than connect-n has less connectivity and was already shape in a line if counts multiple times. The score got would be inaccurate.

The final version of heuristic function including the following part: checking continuous horizontal connect-N, checking discreet horizontal connect-N, checking vertical connect-N, checking positive diagonal slope connect-N, checking negative connect-N, and targeted weights for each connect-N in the above 5 sections, which counts both player's score and opponent's score and substract in the subfunctions. The combination fo the method are relative efficient and comprehensive on solving connect-N problem.

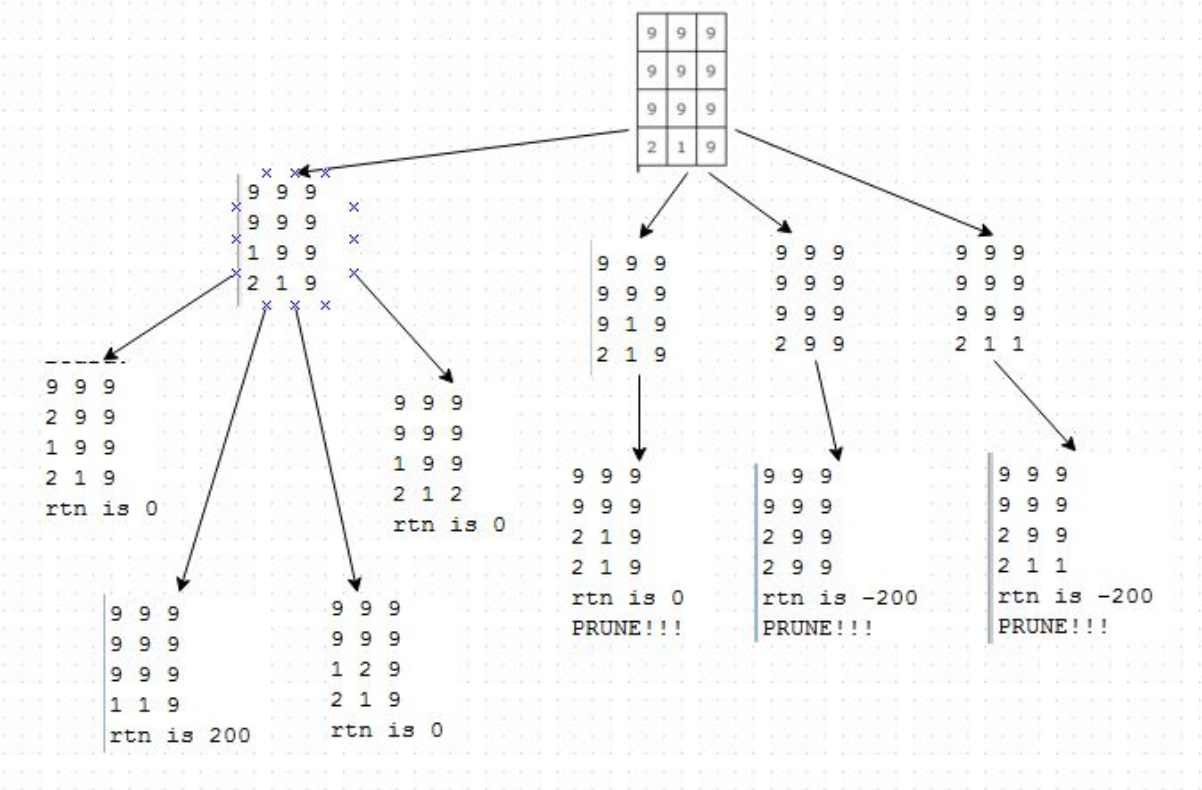
For the minimax algorithm with alpha-beta pruning, we decided to use iterative-deepening minimax approach. We call minimax algorithm in a for loop to increase the depth by 1 at a time. Every time we finish the algorithm, we will update our move to get a better result. we will stop the algorthm 50ms before the time limit so we won't exceed the limit.

Example depth-2 tree generated by our player in a game played by us in out test mode.

Game configuration and first two moves:

```
yzhou8mli2
ENTER YOUR MOVE!!!
p yzhou8mli2 p b
ENTER YOUR MOVE!!!
4 3 3 1 2
1 1
ENTER YOUR MOVE!!!
0 1
```

The tree is as following:



As we can tell, the heuristic function returns the corresponding value to the board state, also, the alpha-beta pruning pruns out the subtrees that definatly won't be the answer. The returned move from the algorithm and the resultent board are shown as the following:

```
0 1
Current board:
Board:
9 9 9
9 9 9
1 9 9
2 1 9
```

Referee Interaction and Documentation

The name of our player is "yzhou8mli2"

Our player is written in Java. The runnable jar file is exported from eclipse by first right clicking on the "Player.java" file, then, click "Export", then, select "Runnable JAR file" under "Java", then, click "Next>", then, select "Player" in the "Launch configuration" part, select "Package required libraries into generated JAR" and click "Finish".

To let our player play against others', use the following command in command line:
java -jar Referee.jar "java -jar path-to-our-player" "java -jar path-to-other-player"

Our player interacts with referee following the protocal given in the assignment.
First, our player sends its name to the referee, then it waits for the player names to be sent be the referee. After it gets the player names, it waits for the game configuration sent by the referee. After it get the game configuration, it detemines who goes first. If it goes first, it will make the first move, if the other player goes first, it will wait until the other player gives its move. After the player reads in the opponent's move, it will calculate its move using iterative deepening minimax with alpha-beta pruning within the time limitation and pass the move to referee.