Ass2 Adversarial Search

Team member:

Xiwen Shen xs143

Yifan Zhang yz745

Yiqian Li yl1165

Tin Fung tyf3

General introduction:

Our project has several different classes：

“Piece.java” is used to recording the information of the piece which including the team it belongs to and the type of the pieces (Wumpuses, heroes, mages)

“Board.java” is used to recording the board information, which included the position of each piece and the current state of the board (the metric value of the current condition)

“buildTree. java” will create a game tree we need to record the possible choice. Each node in the tree is the map.

"Minmax.java" is the main part do the AI part, which does the question 3,4,6.

And the terminal will show the BOARD and AI selection. Here is a well playing example of our algorithm.

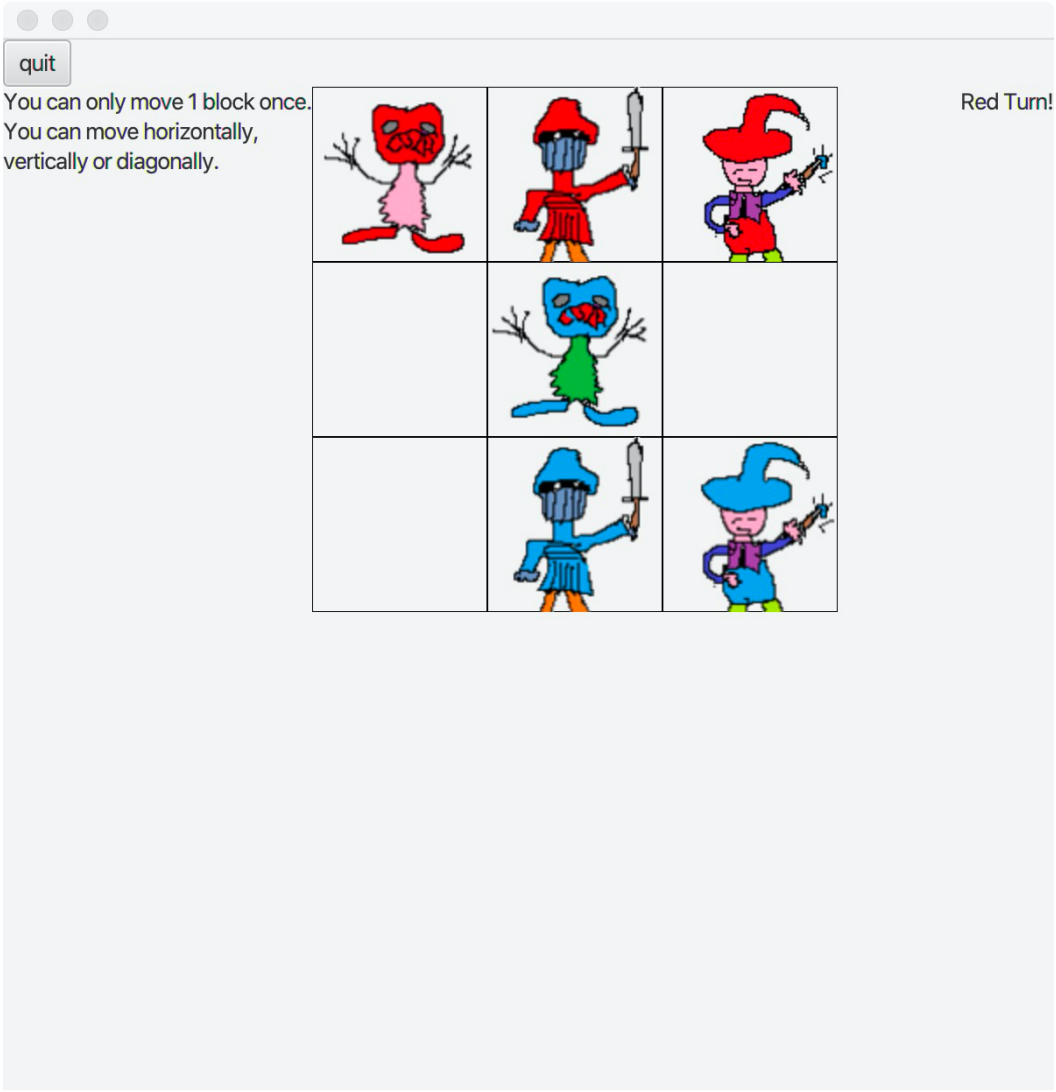
```
build a 3*3board.
        0       1       2
0       rW      rH      rM

1       *       *       *

2       bW      bH      bM

Red trun!
choose your piece:
0 0
move to:
1 1
now:
        0       1       2
0       *       rH      rM

1       *       rW      *

2       bW      bH      bM

Board player: AI(blue)
Board value: 0
...
AI ... calculating ...

        0       1       2
0       *       rH      rM

1       *       bH      *

2       bW      *       bM

Red trun!
choose your piece:
0 1
move to:
1 1
now:
        0       1       2
0       *       *       rM

1       *       *       *

2       bW      *       bM

Board player: AI(blue)
Board value: 79
```

```
AI ... calculating ...

        0       1       2
0       *       *       rM

1       *       *       *

2       *       bW      bM

Red trun!
choose your piece:
0 2
move to:
1 1
now:
        0       1       2
0       *       *       *

1       *       rM      *

2       *       bW      bM

Board player: AI(blue)
Board value: 79
...
AI ... calculating ...

        0       1       2
0       *       *       *

1       *       bW      *

2       *       *       bM

AI wins!
```

Also, we develop a nice UI interface. The player can choose two mode:

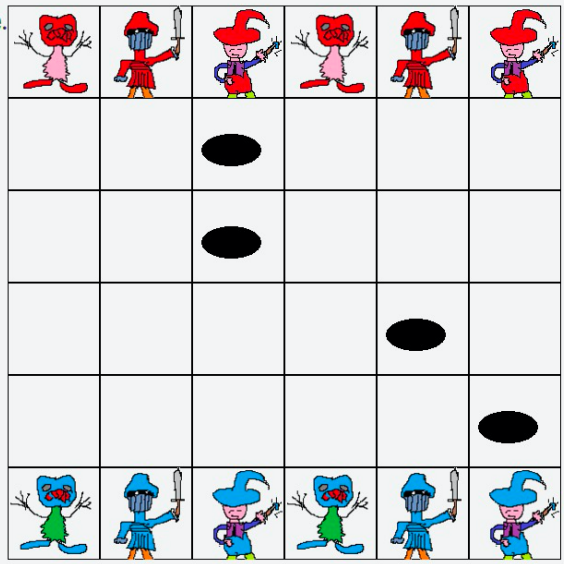
1. Human VS. Human

2. Human VS. AI



quit

You can only move 1 block once.
You can move horizontally,
vertically or diagonally.



Blue Turn!

Q1

When you run WumpusGame.java, the game is starting. It will first print the whole board. Then it will ask you the position for the piece you want to move in the format of x_coordinate and y_coordinate. E.g. for the piece at the right upper corner, you input is "0 0", use space to separate the two input.

This file support both Player VS AI and Player VS Player.

For Player VS Player you need to uncomment “b.setIsRed(!b.getIsRed()” and comment the minmax part.

Q2

We use buildtree.java to create the search. We set the depth to 3, which will check 3 steps. Because each piece of 8 possible direction to go, so every time will have 8*number of pieces choose which is the max number of the children of each node. (if the new position is out of board the possible choice number will be decreased.) E.g at 9*9 board, each time will have a max of 8*9 = 72 children. Therefore, we only set the depth to 3 in order to make sure the problem can run successfully. (Prevent “StackOverflow” and also control the runtime)

Some possible improvements. Currently, AI likes to choose the same type to battle and destroy that same time. This is limited by the tree depth. Because it only considers 3 more steps that might do not have a successful battle happen. It will choose to make both lose a piece.

For the metric value part, we do the calculation based on the number of the piece on the board. Each piece on the board is worth 40 points. Each piece will add extra 30 points * the number of pieces it can beat, if not it will get 10 points. (e.g. if on the board, the blue side have Wumpus and the red side have 2 magic, blue will get 2*30 extra points; if on the board, the blue side have Wumpus and the red side have 0 magic, blue will get 10 extra points)) Also, we will consider the possibility of the same type of piece of battle. The one own more numbers of the type will have 5 extra points * for each more number. (e.g. blue have three magic and red only have one, blue will get 5*2 extra points) We also consider the one piece need to kill more piece. Then the one with less piece will lose 2 points * for each extra piece it needs to kill. (Red have one Wumpus and blue have two magic, therefore red need to kill two-piece use one Wumpus, which is difficult compare to red have two Wumpus to kill two blue magic. The red will lose 2 points.)

Some possible improvements. Now we only calculate the metric value based on the number of the piece. However, in the real world, we should also consider the distance between different pieces. If the red Wumpus moves one step will beat the blue magic, this should have high points than other possible choices. (this increases the accuracy of alpha-beta pruning)

Q3, 4 and 6

Minimax.java file do the minimax algorithm with heuristics described in section 2.

Q5.

5 potential heuristics is based on the order for check alpha and beta numbers.

The basic way is to check all the possible children in the tree. This way does not have any optimization. However, this way is straightforward and does not need to consider sort the children.

The second way is to use the binary search to find the alpha and beta numbers. This way will increase the speed of the program. As we know the O of binary search is log n. (this is the best way to work for the general condition. This will save run time compared to the first one, however, this still do not the best one compare to this special condition)

The third way is to sort all the children, then start from the middle then goes to two different directions. This way is good when alpha and beta numbers and close to the average numbers. (this does not fit the condition in this project)

The fourth one is during max levels you can then explore the children in descending order (highest heuristic value to lowest) while during min levels

you can explore the children in ascending order. This is the best way. Because at the max level we know we are looking for the highest heuristic value and at the min level, we are looking for the lowest value. This only applies to this project. This algorithm is used based on the property of this program.

The fifth one is the opposite of the fourth one which is during min levels you can then explore the children in descending order (highest heuristic value to lowest) while during max levels you can explore the children in ascending order. This way will get a similar result as the basic one maybe even worse because it still needs time to sort. This is the common error in design when the designer confuses with the relationship between heuristic value and the max-min level.