

```
public ABNegamaxResult abNegamax(ArrayList state, int score, int depth, Boolean isAI, int aplha, int beta) {
    if (isEndGame(state) || depth == 5) { // 5/7 is best and running time controlled in 10s
        return new ABNegamaxResult(score, null);
    }

    int[] bestMove = new int[2];
    int bestScore = -INFINITY;

    for (int row = 0; row < 6; row++) {
        for (int column = 0; column < 6; column++) {
            if (((ArrayList) state.get(row)).get(column) == null) {
                ArrayList<ArrayList> newBorad = new ArrayList<ArrayList>();
                for (int i = 0; i < state.size(); i++) {
                    newBorad.add(new ArrayList<ArrayList>((ArrayList) state.get(i)));
                }
                ((ArrayList) newBorad.get(row)).set(column, "A");

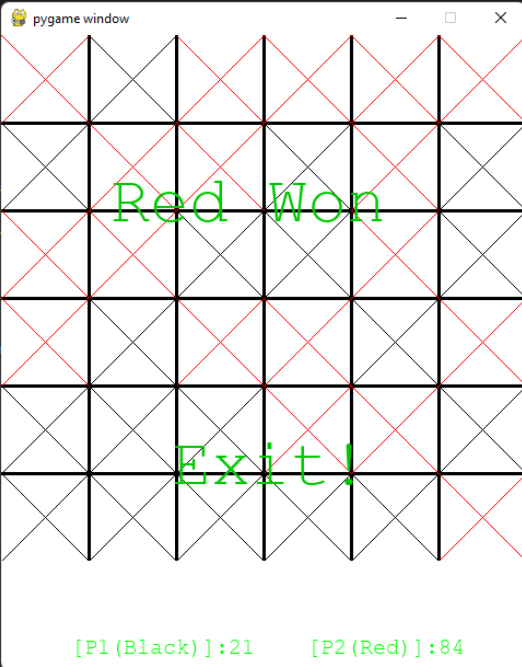
                int[] newMove = new int[] { row, column };
                int newBoradScore = calNewScore(newBorad, !isAI, score);

                //Recurse. if depth is even, !isAI
                ABNegamaxResult recursedResult = abNegamax(newBorad,
                    -beta,
                    -1 * Math.max(aplha, bestScore));
                int currentScore = -recursedResult.bestScore;

                // Update the best score
                if (currentScore > bestScore) {
                    bestScore = currentScore;
                    bestMove = newMove;
                }

                // prune node
                if (bestScore >= beta)
                    break;
            }
        }
    }

    return new ABNegamaxResult(bestScore, bestMove);
}
```



[P1 (Black)] : 21 [P2 (Red)] : 84

CS4386 Assignment 1 (Semester B, 2021-2022)

YIP Yiu Cheung | 55775890

Contents

The Design of Algorithm (MiniMax with Alpha-Beta pruning).....	2
Difference Between Applying AlphaBeta and without AlphaBeta.....	3
AI Running Example.....	4
Declaration	6

The Design of Algorithm (MiniMax with Alpha-Beta pruning)

The concept of designing this algorithm is based on the Lecture 2 (P. 21).

First, the program has an if-condition to check is that board status is full or is arrived the depth that we defined. If true, that is mean the recurrence is finished and then return the board score for the corresponding move for evaluation. If not, continuous to go through each move until is board status is end. Here the depth is set to 5 as it is best for evaluating the best move to getting score and allow the running time controlled within 7s. If the depth is set to larger (i.e. 7), the running time goes up geometrically and the AI may not win the game. Although the AI can search more possibility with high depth but the rule of the game is to get the point as soon as possible for winning the game. That is why the prediction of next 3 to 5 steps is the key rather than predict more steps.

```
if (isEndGame(state) || depth == 5) { // 5/7
    return new ABNegamaxResult(score, null);
}
```

When going through each move of the board, we have to create a new variable to save the new board. To prevent the java continuous to reference the main board, we need to new the arraylist for each arraylist in state.

```
public ArrayList makeNewBoard(ArrayList state, int[] newMove) {
    ArrayList<ArrayList> newBorad = new ArrayList<ArrayList>();

    for (int i = 0; i < state.size(); i++) {
        newBorad.add(new ArrayList<ArrayList>((ArrayList) state.get(i)));
    }

    ((ArrayList) newBorad.get(newMove[0])).set(newMove[1], "A");

    return newBorad;
}
```

After creating the new board, the new board score will be calculated and continuous to simulate the next step until reaching the depth. If the simulated move belongs to opponent, our score will be deducted if opponent's move constructs the 3 or 6 linked cells in same column or row. Then, there are conditions for evaluating the scores of the move. The further simulation may stop if the bestScore is larger, which means the corresponding move is able to get the point as soon as possible.

```
int currentScore = -recursedResult.bestScore;

// Update the best score
if (currentScore > bestScore) {
    bestScore = currentScore;
    bestMove = newMove;
}

// prune node
if (bestScore >= beta)
    break;
```

Difference Between Applying AlphaBeta and without AlphaBeta

The program is also work without storing the alpha and beta variable. However, the running time of program will be increased as the function call of “abNegamax()” will search deeply for different move and evaluating the score of new board. It will become not time-efficient (running time for the second move is 8.47s).

```
public ABNegamaxResult abNegamax(ArrayList state, int score, int depth, Boolean isAI) { // int alpha, int beta
    if (isEndGame(state) || depth == 5) { // 5/7 is best and running time controlled in 10s
        return new ABNegamaxResult(score, null);
    }

    int[] bestMove = new int[2];
    int bestScore = -INFINITY;

    for (int row = 0; row < 6; row++) {
        for (int column = 0; column < 6; column++) {
            if (((ArrayList) state.get(row)).get(column) == null) {
                int[] newMove = new int[] { row, column };

                ArrayList<ArrayList> newBoard = makeNewBoard(state, newMove);

                int newBoardScore = calNewScore(newBoard, !isAI, score, newMove); // if depth is odd, !isAI

                // Recurse. if depth is even, !isAI
                ABNegamaxResult recursedResult = abNegamax(newBoard, newBoardScore, depth + 1, !isAI);
                // -beta, -1 * Math.max(alpha, bestScore)
                int currentScore = -recursedResult.bestScore;

                // Update the best score
                if (currentScore > bestScore) {
                    bestScore = currentScore;
                    bestMove = newMove;
                }

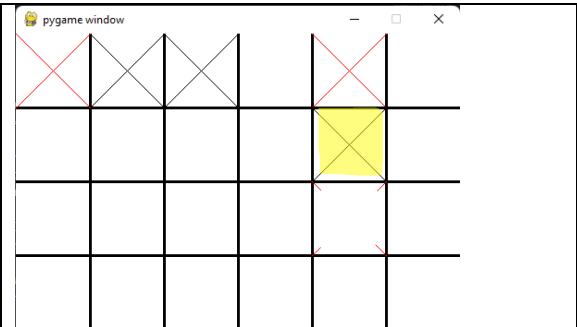
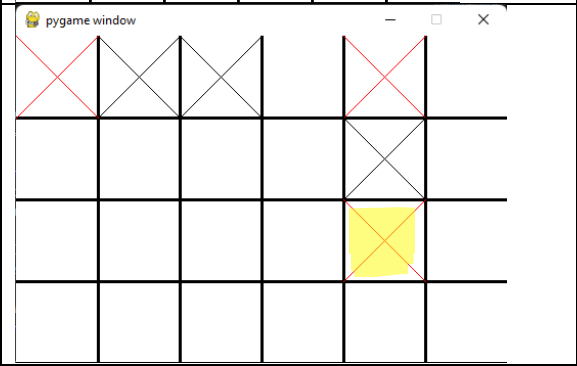
                // prune node
                // if (bestScore >= beta)
                // break;
            }
        }
    }
}
```

```
155 int newBoardScore = calNewScore(newBoard, !isAI, score, newMove);
156
157 // Recurse. if depth is even, !isAI
158 ABNegamaxResult recursedResult = abNegamax(newBoard, newBoardScore, depth + 1, !isAI);
159 // -beta, -1 * Math.max(alpha, bestScore)
160 int currentScore = -recursedResult.bestScore;
161
162 // Update the best score
163 if (currentScore > bestScore) {
164     bestScore = currentScore;
165     bestMove = newMove;
166 }
167
168 // prune node
169 // if (bestScore >= beta)
170 // break;
```

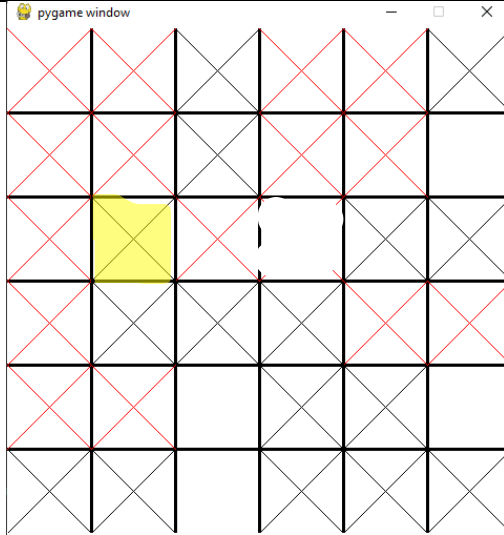
PROBLEMS 61 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\YipYi\Documents\GitHub\2022_CS4386_assignment1>
.\Users\YipYi\.vscode\extensions\ms-python.python-2022.2.19\
'1'
pygame 2.1.2 (SDL 2.0.18, Python 3.9.11)
Hello from the pygame community. https://www.pygame.org/contributing.html
the first player is AI (JAVA)
the second player is AI (JAVA)
player1 (black) plays first
Player1 (Black,JAVA), move is: 5 2
Player1 (Black) Time: 0.0010004043579101562
Player2 (Red,JAVA), move is: 0 0
Current score for player1 (black), player2 (red): 0 0
Player2 (Red) Time: 8.473088026046753
```

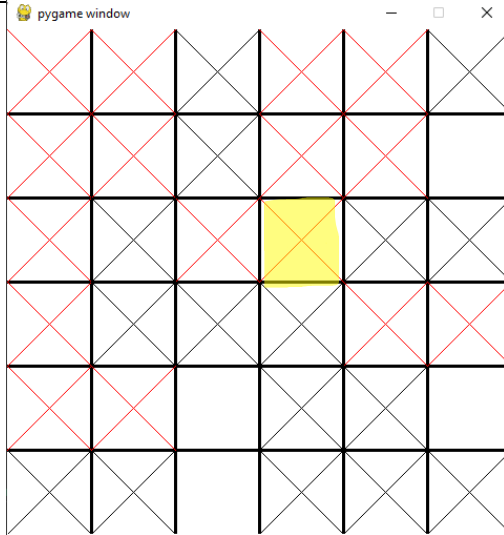
AI Running Example

My Turn	
AI's Turn	

My Turn



AI's Turn



Declaration

The design concepts of the algorithm are reference to Lecture 2 (Pseudo Code –AB Negamax).

The score calculation is reference to game.py with few changes.