CMPE 492 Final Report

Yener Mete

Instructor : Tunga Güngör

The Algorithm

I have chosen to tell all the classes one by one in this project.

1. knight.java

Knight is the easiest piece of all the pieces to be implemented. It moves in 'L' shape. No piece can stop it from moving. A knight object's side is checked by looking at its value. If it is positive, it is a computer object, otherwise it is a player object. According to the side of the object, all 8 places that it can move are checked. If an enemy piece exists there, threat array's corresponding indices set to true. If that square is empty, both protect and threat arrays' corresponding indices are set to true. If that square is occupied by a piece of the same side as the considered knight object, then protect array is set to true. canMove array holds all the places that the knight can go. In my algorithm, canMove array is the same as threat array. But since canMove and threat are different for pawns and kings, I chose to make canMove a parameter of all the piece objects. Mobility counter is increased by each true canMove array element. Double attack counter is increased if any double attack condition happens. This is checked by considering if there are two or more threat elements that are true, and the knight is not threatened by any other piece.

2.bishop.java

Bishop is a relatively hard object to implement. This object can move in four directions. For each direction, its move can be stopped if any piece (self or enemy) is on its way of moving direction. So this object has a boolean variable that checks whether a move search has ended in the considered direction. Before starting a search in a direction, this variable is set to false. Threat array's corresponding indices are set to true if the bishop does threat an enemy piece. Protect array's corresponding indices are set to true if the bishop protects its own piece. When a self or enemy piece is met, the search ends in that direction and the move end checking boolean variable is set to true to end the search. If an empty square is also available to move to, then this square's protect and threat functions are set to true. canMove consists of the same values as the threat array, and each canMove element that is true increases the counter. If any two true threat elements have higher values than the bishop's value; then double attack counter catal is incremented.

3.rook.java

The rook has the same logic as the bishop. It can move in four directions, but in an absolutely straight direction rather than cross. It again has a boolean variable to check if it is to end its move search in a direction. All mobility, canMove, threat and protect arrays are completed in the same way as the bishop. The major difference of this object is that it has no double attack counter. This is because there are only two pieces in chess game that are more valuable than it, namely king and queen. However, there can still be game states in which the rook can make a double attack too. In the future work, this approach can be added to the rook object too.

4.queen.java

There is not much to say about this object. It is just the combination of rook and bishop. The only thing worth mentioning is that this piece, in the future work, should have a

different double attack algorithm, because it is invaluable, even grandmasters can not afford to lose it. Other pieces can be lost after they capture the more valuable enemy piece, but the queen must not be lost after capturing an enemy piece, unless it is a queen.

5.king.java

This is the hardest piece to implement in my algorithm. It must not move to any place on the board that is threatened by the enemy. To achieve this objective, two big threat arrays are present, one for the enemy and one for the computer. If any piece of one side threats a square, enemy king can not move to that piece. So far so good, but if this object threats an enemy piece, it does not necessarily mean that the king can move there, because that enemy square may be threatened by another object. So threat and canMove functions of the king are absolutely different. Protect function is also completed like all the other pieces, but again canMove is not the same for all protect elements. Other than that, there is also the castling condition. This is checked according to the side and color of the king. If the king is white, and no piece threats the squares between it and the castle with which it will castle, then canCastle[position of king after castling] is set to true. so the king can move to the true elements of canMove and canCastle. Castling for the black king is checked in the same way. If a king is threated and its canMove array has no true variable, and no piece can stop the check condition, then the variable mat is set to true.

6.pawn.java

Pawn has the longest code of all the objects. This is because its side differs in the way it can move. It can be promoted, but in two different ways and into four different objects. I will tell the white pawn's function briefly, black is done in a symmetric way.

A white pawn can move two squares forward if it is at the b index of the board (2 of y direction in my board algorithm) and the two squares ahead of it are empty. It can move to the next square if it is empty. It can also move to cross positions if an enemy is present. So canMove and threat arrays are again different. Threat array is set to true if the piece can capture an enemy piece, canMove array is set to true if the piece can move to an empty square or its cross squares ahead of it are its own pieces or enemy pieces. Protect array has the same logic as the threat array.

A white pawn can be promoted if it is at the 7^{th} column and it can move to the 8^{th} column. If a pawn moves to the last column, user is asked to choose between four pieces, knight, bishop, rook, queen. However, rook and bishop choices also create a queen. The computer always chooses to create a new queen, as it is the most powerful piece on the board. The pawn is destroyed at the next step, as it no longer exists.

The same logic goes for a black pawn too, by applying symmetry.

7.evaluate.java

This is the most important aspect of my and all chess algorithms. All the aspects of the game are created and calculated here, except for the alpha-beta algorithm.

Threat arrays of both players are calculated here. All the pieces are created here with trivial positions and false constructors. A false constructor means all the variables it has to be used in the evaluation function are set to zero. So creating all the pieces is not a problem.

To check for which piece has been created with a true constructor parameter, there are boolean arrays for each kind of object that are false if a piece has not been constructed properly. Proper construction means all board has been checked and a piece has been constructed if it exists.

All the board is checked, and if a knight or –knight value is met (self or enemy value), a check is made at the knight's Boolean arrays that check for how many knights there are on the board. If there are no knights, then first knight of that side is created. If there is already a knight, second knight of that side is created. All the board is checked in the same manner and new objects are created when necessary. After each creation of those objects, that object's function is run to calculate this piece's parameters' values. (I have forgotten this on the last code, but in the future work I will have completed all these)

After all the board is checked, the return value of the evaluation function is incremented by all pieces' mobility value, double attack counter and piece values. However, computers' parameters are added, and players parameters are subtracted. As will be seen shortly, this does mean that computer is the maximizing player, and the player is the minimizing player of the minimax algorithm.

8.Minimax.java

This was the hardest part of my project, and it is the main reason that the project is not complete. But I will tell the ideas that I have used to construct this class.

The depth of the minimax tree is 4. I make the search depth first. I have three arrays that represent the Board. One array holds the second depth nodes' board states. Other arrays hold the other depths' nodes' board states. Each array is given to the evaluate object's function to calculate which moves can be made. There are three ArrayList objects consisting of strings. They hold which moves can be made. According to the move made, a parent node's state is changed and put into the child node's board state array. Original array (that of parent's) does not change, because the siblings will also need the original array to get their own board arrays. When depth four is reached, the evaluation function is run with the last board array and that node is given this value. All the moves are checked like this, and the bottom nodes are all created. According to who plays max and min, the appropriate value is selected (i.e. if max is to play, highest node value is chosen) and assigned to the parent node. This is done from bottom to top and the best state leading move is chosen as the next move.

Alpha-beta algorithm lacks. It is not much of a problem, as only the same depth nodes will have to be checked with alpha and beta. This is part of the future woek that I am thinking.

9.play.java

This function is not complete, it just will get the input from the user. And give an output to the user as the computer's next move. This is part of the future work, but still easy to be implemented.

This class's new instance will be used in the main function and the game will run finally.

Mutation

This is the adaptive part of the algorithm. As the program does not work, it consists of only ideas. Everything is as follows :

Mobility counter multiplier, double attack counter multiplier, and piece values except king and pawns are given as parameters to vector individuals. Three vectors are initially created manually, by any parameter value that is required. However, trying sensible values is more logical.

Individual 1	Individual 2	Individual 3	??	??	??

These first three individuals are called parents of the first generation. Two parameters are mutated from each vector. Either one of the piece values of bishop, knight, rook or queen is mutated and the mobility counter multiplier or the double attack counter multiplier is mutated. Mutation can be made in terms of an advanced probabilistic function, but I chose to make the mutation in the following way :

If a piece that is known to have a value bigger than another piece has a lower value than that piece after the mutation, then that piece's value is changed to the supposedly lower value having piece's value. Each piece has upper or lower bounds for each mutation, so a mutation does not create a white offspring from a black parent.

From each mutated parent, last 3 individuals are created. And this generation is complete from now on.

Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6

Individual 1 plays with one of the remaining individuals randomly. Whoever wins becomes the next generation's first parent. The remaining individuals play with each other in the same manner, and next generation's parents are formed in this manner. These second generation parents also are mutated to have offsprings, and they again play with each other to form the third generation, and so on. This should continue for around 30-50 generations, and the last generation's parents will have the best values to play against the best players or computer programs in the world.

Future Work

The future work that I am thinking to do consists of the following:

- Developing the algorithm by adding the removed aspects like center control, king safety, pawn development, lonely pawn negative bonus, etc.
- Adding a visual interface and making the game to be fully interactive, making user menus to increase visual quality.
- Improving the tree search by adding the alpha-beta algorithm
- Detailing the double attack algorithm by considering whether the double attacker will be captured after it has captured one of the pieces, etc.

Classes

Rook.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package satranc;

/**
 *
 * @author Administrator
 */
public class rook {

    int x;
    int y;
    boolean exists;
    public rook(int coorX, int coorY, boolean var) {
        x = coorX;
        y = coorY;
        exists = var;
    }
    public double catal;
    public boolean threat[][] = new boolean[9][9];
    public boolean protect[][] = new boolean[9][9];
    public boolean canMove[][] = new boolean[9][9];
    public double mobility = 0;
    boolean end = false;
    int coorX;
    int coorY;
    double value = evaluate.valueRook;

    void function(double Board[][]) {
        if (exists = true) {
            if (Board[x][y] < 0) {
                end = false;
                for (int a = 1; a < 9; a++) {
                    for (int b = 1; b < 9; b++) {
                        coorX = x;
                        coorY = y;
                        while ((coorX + 1 < 9) && (end = false)) {
                            if (Board[coorX + 1][coorY] == 0) {
```

```
        mobility++;
        protect[coorX+1][coorY] = true;
        threat[coorX+1][coorY] = true;
        canMove[coorX+1][coorY] = true;
        coorX++;
    } else if (Board[coorX+1][coorY] > 0) {
        mobility++;
        threat[coorX+1][coorY] = true;
        canMove[coorX+1][coorY] = true;
        end = true;
    } else {
        protect[coorX + 1][coorY] = true;
        end = true;
    }
}

end = false;
coorX = x;
coorY = y;
while ((coorX - 1 < 9) && (end = false)) {
    if (Board[coorX - 1][coorY] == 0) {
        mobility++;
        protect[coorX - 1][coorY] = true;
        threat[coorX - 1][coorY] = true;
        canMove[coorX-1][coorY] = true;
        coorX++;
    } else if (Board[coorX - 1][coorY] > 0) {
        mobility++;
        threat[coorX - 1][coorY] = true;
        canMove[coorX-1][coorY] = true;
        end = true;
    } else {
        protect[coorX - 1][coorY] = true;
        end = true;
    }
}

end = false;
coorX = x;
coorY = y;
while ((coorY - 1 < 9) && (end = false)) {
    if (Board[coorX][coorY - 1] == 0) {
        mobility++;
        protect[coorX][coorY - 1] = true;
        threat[coorX][coorY - 1] = true;
        canMove[coorX][coorY - 1] = true;
        coorX++;
    } else if (Board[coorX][coorY - 1] > 0) {
        mobility++;
        threat[coorX][coorY - 1] = true;
        canMove[coorX][coorY - 1] = true;
        end = true;
    } else {
        protect[coorX][coorY - 1] = true;
        end = true;
    }
```

```
            }

            while ((coorY + 1 < 9) && (end = false)) {
               if (Board[coorX][coorY + 1] == 0) {
                  mobility++;
                  protect[coorX][coorY + 1] = true;
                  threat[coorX][coorY + 1] = true;
                  canMove[coorX][coorY + 1] = true;
                  coorX++;
               } else if (Board[coorX][coorY - 1] > 0) {
                  mobility++;
                  threat[coorX][coorY + 1] = true;
                  canMove[coorX][coorY + 1] = true;
                  end = true;
               } else {
                  protect[coorX][coorY + 1] = true;
                  end = true;
               }
            }
         }
      }
   } else {
      end = false;
      for (int a = 1; a < 9; a++) {
         for (int b = 1; b < 9; b++) {
            coorX = x;
            coorY = y;
            while ((coorX + 1 < 9) && (end = false)) {
               if (Board[coorX + 1][coorY] == 0) {
                  mobility++;
                  protect[coorX + 1][coorY] = true;
                  threat[coorX + 1][coorY] = true;
                  canMove[coorX+1][coorY] = true;
                  coorX++;
               } else if (Board[coorX + 1][coorY] < 0) {
                  mobility++;
                  threat[coorX + 1][coorY] = true;
                  canMove[coorX+1][coorY] = true;
                  end = true;
               } else {
                  protect[coorX + 1][coorY] = true;
                  end = true;
               }
            }

            end = false;
            coorX = x;
            coorY = y;
            while ((coorX - 1 < 9) && (end = false)) {
               if (Board[coorX - 1][coorY] == 0) {
                  mobility++;
                  protect[coorX - 1][coorY] = true;
                  threat[coorX - 1][coorY] = true;
                  canMove[coorX-1][coorY] = true;
                  coorX++;
               } else if (Board[coorX - 1][coorY] < 0) {
```

```
                    mobility++;
                    threat[coorX - 1][coorY] = true;
                    canMove[coorX-1][coorY] = true;
                    end = true;
                } else {
                    protect[coorX - 1][coorY] = true;
                    end = true;
                }
            }

            end = false;
            coorX = x;
            coorY = y;
            while ((coorY - 1 < 9) && (end = false)) {
                if (Board[coorX][coorY - 1] == 0) {
                    mobility++;
                    protect[coorX][coorY - 1] = true;
                    threat[coorX][coorY - 1] = true;
                    canMove[coorX][coorY-1] = true;
                    coorX++;
                } else if (Board[coorX][coorY - 1] < 0) {
                    mobility++;
                    threat[coorX][coorY - 1] = true;
                    canMove[coorX][coorY-1] = true;
                    end = true;
                } else {
                    protect[coorX][coorY - 1] = true;
                    end = true;
                }
            }

            while ((coorY + 1 < 9) && (end = false)) {
                if (Board[coorX][coorY + 1] == 0) {
                    mobility++;
                    protect[coorX][coorY + 1] = true;
                    threat[coorX][coorY + 1] = true;
                    canMove[coorX][coorY+1] = true;
                    coorX++;
                } else if (Board[coorX][coorY - 1] < 0) {
                    mobility++;
                    canMove[coorX][coorY+1] = true;
                    threat[coorX][coorY + 1] = true;
                    end = true;
                } else {
                    protect[coorX][coorY + 1] = true;
                    end = true;
                }
            }
        }
    }
}else{
    mobility = 0;
    catal = 0;
    value = 0;
    for(int i=1;i<9;i++){
```

```
            for(int j=1; j<9; j++){
                canMove[i][j] = false;
                protect[i][j] = false;
                threat[i][j] = false;
            }
        }
    }
  }
}
```

Bishop.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package satranc;

/**
 *
 * @author Administrator
 */
public class bishop {

   int x;
   int y;
   boolean exists;
   public bishop(int coorX, int coorY, boolean var) {
      x = coorX;
      y = coorY;
      exists = var;
   }
   public int catal;
   public boolean threat[][] = new boolean[9][9];
   public boolean protect[][] = new boolean[9][9];
   public boolean canMove[][] = new boolean[9][9];
   public double mobility = 0;
   boolean end = false;
   int coorX;
   int coorY;
   double value = evaluate.valueBishop;

   void function(double Board[][]) {
      if(exists = true){
         for (int a = 1; a < 9; a++) {
            for (int b = 1; b < 9; b++) {
               if (Board[x][y] < 0) {
                  coorX = x;
                  coorY = y;
                  while ((coorX + 1 < 9) && (coorY + 1 < 9) && (end = false)) {
                     if (Board[coorX + 1][coorY + 1] == 0) {
                        mobility++;
                        protect[coorX+1][coorY+1] = true;
                        threat[coorX+1][coorY+1] = true;
                        canMove[coorX+1][coorY+1] = true;
                        coorX++;
```

```
        coorY++;
    } else if (Board[coorX + 1][coorY + 1] > 0) {
        mobility++;
        threat[coorX + 1][coorY + 1] = true;
        canMove[coorX+1][coorY+1] = true;
        end = true;
    } else {
        protect[coorX+1][coorY+1] = true;
        end = true;
    }
}
// get original position
coorX = x;
coorY = y;
end = false;
while ((coorX - 1 > 0) && (coorY - 1 > 0) && (end = false)) {
    if (Board[coorX - 1][coorY - 1] == 0) {
        mobility++;
        protect[coorX - 1][coorY - 1] = true;
        threat[coorX - 1][coorY - 1] = true;
        canMove[coorX - 1][coorY - 1] = true;
        coorX--;
        coorY--;
    } else if (Board[coorX - 1][coorY - 1] > 0) {
        canMove[coorX - 1][coorY - 1] = true;
        mobility++;
        threat[coorX - 1][coorY - 1] = true;
        end = true;
    } else {
        protect[coorX - 1][coorY - 1] = true;
        end = true;
    }
}

coorX = x;
coorY = y;
end = false;
while ((coorX + 1 < 9) && (coorY - 1 > 0) && (end = false)) {
    if (Board[coorX + 1][coorY - 1] == 0) {
        mobility++;
        protect[coorX + 1][coorY - 1] = true;
        threat[coorX + 1][coorY - 1] = true;
        canMove[coorX + 1][coorY - 1] = true;
        coorX++;
        coorY--;
    } else if (Board[coorX + 1][coorY - 1] > 0) {
        mobility++;
        canMove[coorX + 1][coorY - 1] = true;
        threat[coorX + 1][coorY - 1] = true;
        end = true;
    } else {
        protect[coorX + 1][coorY - 1] = true;
        end = true;
    }
}
```

```
    coorX = x;
    coorY = y;
    end = false;
    while ((coorX - 1 > 0) && (coorY + 1 < 9) && (end = false)) {
      if (Board[coorX - 1][coorY + 1] == 0) {
        mobility++;
        protect[coorX - 1][coorY + 1] = true;
        threat[coorX - 1][coorY + 1] = true;
        canMove[coorX - 1][coorY + 1] = true;
        coorX--;
        coorY++;
      } else if (Board[coorX - 1][coorY + 1] > 0) {
        canMove[coorX - 1][coorY + 1] = true;
        mobility++;
        threat[coorX - 1][coorY + 1] = true;
        end = true;
      } else {
        protect[coorX - 1][coorY + 1] = true;
        end = true;
      }
    }
  } else {
    coorX = x;
    coorY = y;
    while ((coorX + 1 < 9) && (coorY + 1 < 9) && (end = false)) {
      if (Board[coorX + 1][coorY + 1] == 0) {
        mobility++;
        protect[coorX + 1][coorY + 1] = true;
        threat[coorX + 1][coorY + 1] = true;
        canMove[coorX + 1][coorY + 1] = true;
        coorX++;
        coorY++;
      } else if (Board[coorX + 1][coorY + 1] < 0) {
        canMove[coorX + 1][coorY + 1] = true;
        mobility++;
        threat[coorX + 1][coorY + 1] = true;
        end = true;
      } else {
        protect[coorX][coorY] = true;
        end = true;
      }
    }

    coorX = x;
    coorY = y;
    end = false;
    while ((coorX - 1 > 0) && (coorY - 1 > 0) && (end = false)) {
      if (Board[coorX - 1][coorY - 1] == 0) {
        canMove[coorX - 1][coorY - 1] = true;
        mobility++;
        protect[coorX - 1][coorY - 1] = true;
        threat[coorX - 1][coorY - 1] = true;
        coorX--;
        coorY--;
      } else if (Board[coorX - 1][coorY - 1] < 0) {
        canMove[coorX - 1][coorY - 1] = true;
```

```
                    mobility++;
                    threat[coorX - 1][coorY - 1] = true;
                    end = true;
                } else {
                    protect[coorX - 1][coorY - 1] = true;
                    end = true;
                }
            }
            coorX = x;
            coorY = y;
            end = false;
            while ((coorX + 1 < 9) && (coorY - 1 > 0) && (end = false)) {
                if (Board[coorX + 1][coorY - 1] == 0) {
                    mobility++;
                    canMove[coorX + 1][coorY - 1] = true;
                    protect[coorX + 1][coorY - 1] = true;
                    threat[coorX + 1][coorY - 1] = true;
                    coorX++;
                    coorY--;
                } else if (Board[coorX + 1][coorY - 1] < 0) {
                    mobility++;
                    canMove[coorX + 1][coorY - 1] = true;
                    threat[coorX + 1][coorY - 1] = true;
                    end = true;
                } else {
                    protect[coorX + 1][coorY - 1] = true;
                    end = true;
                }
            }

            coorX = x;
            coorY = y;
            end = false;
            while ((coorX - 1 > 0) && (coorY + 1 < 9) && (end = false)) {
                if (Board[coorX - 1][coorY + 1] == 0) {
                    mobility++;
                    protect[coorX - 1][coorY + 1] = true;
                    threat[coorX - 1][coorY + 1] = true;
                    canMove[coorX - 1][coorY + 1] = true;
                    coorX--;
                    coorY++;
                } else if (Board[coorX - 1][coorY + 1] < 0) {
                    mobility++;
                    canMove[coorX - 1][coorY + 1] = true;
                    threat[coorX - 1][coorY + 1] = true;
                    end = true;
                } else {
                    protect[coorX - 1][coorY + 1] = true;
                    end = true;
                }
            }
        }
    }
}
}else{
    mobility = 0;
```

```
            catal = 0;
            value = 0;
            for(int i=1;i<9;i++){
               for(int j=1; j<9; j++){
               canMove[i][j] = false;
               protect[i][j] = false;
               threat[i][j] = false;
               }
            }
         }
      }
}
```

Knight.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package satranc;

/**
 *
 * @author Administrator
 */
public class knight {

   int x;
   int y;
   boolean exists;
   public knight(int knightx, int knighty, boolean var) {
      x = knightx;
      y = knighty;
      exists = var;
   }
   public double catal;
   public boolean threat[][] = new boolean[9][9];
   public boolean protect[][] = new boolean[9][9];
   public boolean canMove[][] = new boolean[9][9];
   public double mobility = 0;
   double value = evaluate.valueKnight;

   void function(double Board[][]) {
      if (exists = true) {
         if (Board[x][y] < 0) {
            if (((x + 1) < 9) && ((y + 2) < 9) && (Board[x + 1][y + 2] >= 0)) {
               mobility++;
               if (Board[x + 1][y + 2] == 0) {
                  protect[x + 1][y + 2] = true;
                  threat[x + 1][y + 2] = true;
                  canMove[x + 1][y + 2] = true;
               } else {
                  threat[x + 1][y + 2] = true;
                  canMove[x + 1][y + 2] = true;
               }
            } else if(((x + 1) < 9) && ((y + 2) < 9) && (Board[x + 1][y + 2] < 0)){
```

```
    protect[x+1][y+2] = true;
}
if (((x + 2) < 9) && ((y + 1) < 9) && (Board[x + 2][y + 1] >= 0)) {
    mobility++;
    if (Board[x + 2][y + 1] == 0) {
        protect[x + 2][y + 1] = true;
        threat[x + 2][y + 1] = true;
        canMove[x + 2][y + 1] = true;
    } else {
        threat[x + 2][y + 1] = true;
        canMove[x + 2][y + 1] = true;
    }
} else if(((x + 2) < 9) && ((y + 1) < 9) && (Board[x + 2][y + 1] < 0)){
    protect[x+2][y+1] = true;
}
if (((x + 2) < 9) && ((y - 1) > 0) && (Board[x + 2][y - 1] >= 0)) {
    mobility++;
    if (Board[x + 2][y - 1] == 0) {
        canMove[x + 2][y - 1] = true;
        protect[x + 2][y - 1] = true;
        threat[x + 2][y - 1] = true;
    } else {
        threat[x + 2][y - 1] = true;
        canMove[x + 2][y - 1] = true;
    }
} else if(((x + 2) < 9) && ((y - 1) > 0) && (Board[x + 2][y - 1] < 0)){
    protect[x+2][y-1] = true;
}
if (((x + 1) < 9) && ((y - 2) > 0) && (Board[x + 1][y - 2] >= 0)) {
    mobility++;
    if (Board[x + 1][y - 2] == 0) {
        protect[x + 1][y - 2] = true;
        threat[x + 1][y - 2] = true;
        canMove[x + 1][y - 2] = true;
    } else {
        threat[x + 1][y - 2] = true;
        canMove[x + 1][y - 2] = true;
    }
} else if(((x + 1) < 9) && ((y - 2) >0) && (Board[x + 1][y - 2] < 0)){
    protect[x+1][y-2] = true;
}
if (((x - 1) > 0) && ((y - 2) > 0) && (Board[x - 1][y - 2] >= 0)) {
    mobility++;
    if (Board[x - 1][y - 2] == 0) {
        protect[x - 1][y - 2] = true;
        threat[x - 1][y - 2] = true;
        canMove[x - 1][y - 2] = true;
    } else {
        threat[x - 1][y - 2] = true;
        canMove[x - 1][y - 2] = true;
    }
} else if(((x - 1) > 0) && ((y - 2) > 0) && (Board[x - 1][y - 2] < 0)){
    protect[x-1][y-2] = true;
}
if (((x - 2) > 0) && ((y - 1) > 0) && (Board[x - 2][y - 1] >= 0)) {
    mobility++;
```

```
            if (Board[x - 2][y - 1] == 0) {
                protect[x - 2][y - 1] = true;
                threat[x - 2][y - 1] = true;
                canMove[x - 2][y - 1] = true;
            } else {
                threat[x - 2][y - 1] = true;
                canMove[x - 2][y - 1] = true;
            }
        } else if(((x - 2) > 0) && ((y - 1) > 0) && (Board[x - 2][y - 1] < 0)){
            protect[x-2][y-1] = true;
        }
        if (((x - 2) > 0) && ((y + 1) < 9) && (Board[x - 2][y + 1] >= 0)) {
            mobility++;
            if (Board[x - 2][y + 1] == 0) {
                protect[x - 2][y + 1] = true;
                threat[x - 2][y + 1] = true;
                canMove[x - 2][y + 1] = true;
            } else {
                threat[x - 2][y + 1] = true;
                canMove[x - 2][y + 1] = true;
            }
        } else if(((x - 2) > 0) && ((y + 1) < 9) && (Board[x - 2][y + 1] < 0)){
            protect[x-2][y+1] = true;
        }
        if (((x - 1) > 0) && ((y + 2) < 9) && (Board[x - 1][y + 2] >= 0)) {
            mobility++;
            if (Board[x - 1][y + 2] == 0) {
                protect[x - 1][y + 2] = true;
                threat[x - 1][y + 2] = true;
                canMove[x - 1][y + 2] = true;
            } else {
                threat[x - 1][y + 2] = true;
                canMove[x - 1][y + 2] = true;
            }
        } else if(((x - 1) > 0) && ((y + 2) < 9) && (Board[x - 1][y + 2] < 0)){
            protect[x-1][y+2] = true;
        }
    } else {
        if (((x + 1) < 9) && ((y + 2) < 9) && (Board[x + 1][y + 2] <= 0)) {
            mobility++;
            if (Board[x + 1][y + 2] == 0) {
                protect[x + 1][y + 2] = true;
                threat[x + 1][y + 2] = true;
                canMove[x + 1][y + 2] = true;
            } else {
                threat[x + 1][y + 2] = true;
                canMove[x + 1][y + 2] = true;
            }
        } else if(((x + 1) < 9) && ((y + 2) < 9) && (Board[x + 1][y + 2] > 0)){
            protect[x+1][y+2] = true;
        }
        if (((x + 2) < 9) && ((y + 1) < 9) && (Board[x + 2][y + 1] <= 0)) {
            mobility++;
            if (Board[x + 2][y + 1] == 0) {
                protect[x + 2][y + 1] = true;
                threat[x + 2][y + 1] = true;
```

```
            canMove[x + 2][y + 1] = true;
        } else {
            threat[x + 2][y + 1] = true;
            canMove[x + 2][y + 1] = true;
        }
    } else if(((x + 2) < 9) && ((y + 1) < 9) && (Board[x + 2][y + 1] > 0)){
        protect[x+2][y+1] = true;
    }
    if (((x + 2) < 9) && ((y - 1) > 0) && (Board[x + 2][y - 1] <= 0)) {
        mobility++;
        if (Board[x + 2][y - 1] == 0) {
            canMove[x + 2][y - 1] = true;
            protect[x + 2][y - 1] = true;
            threat[x + 2][y - 1] = true;
        } else {
            threat[x + 2][y - 1] = true;
            canMove[x + 2][y - 1] = true;
        }
    } else if(((x + 2) < 9) && ((y - 1) > 0) && (Board[x + 2][y - 1] > 0)){
        protect[x+2][y-1] = true;
    }
    if (((x + 1) < 9) && ((y - 2) > 0) && (Board[x + 1][y - 2] <= 0)) {
        mobility++;
        if (Board[x + 1][y - 2] == 0) {
            protect[x + 1][y - 2] = true;
            threat[x + 1][y - 2] = true;
            canMove[x + 1][y - 2] = true;
        } else {
            threat[x + 1][y - 2] = true;
            canMove[x + 1][y - 2] = true;
        }
    } else if(((x + 1) < 9) && ((y - 2) >0) && (Board[x + 1][y - 2] > 0)){
        protect[x+1][y-2] = true;
    }
    if (((x - 1) > 0) && ((y - 2) > 0) && (Board[x - 1][y - 2] <= 0)) {
        mobility++;
        if (Board[x - 1][y - 2] == 0) {
            protect[x - 1][y - 2] = true;
            threat[x - 1][y - 2] = true;
            canMove[x - 1][y - 2] = true;
        } else {
            threat[x - 1][y - 2] = true;
            canMove[x - 1][y - 2] = true;
        }
    } else if(((x - 1) > 0) && ((y - 2) > 0) && (Board[x - 1][y - 2] > 0)){
        protect[x-1][y-2] = true;
    }
    if (((x - 2) > 0) && ((y - 1) > 0) && (Board[x - 2][y - 1] <= 0)) {
        mobility++;
        if (Board[x - 2][y - 1] == 0) {
            protect[x - 2][y - 1] = true;
            threat[x - 2][y - 1] = true;
            canMove[x - 2][y - 1] = true;
        } else {
            threat[x - 2][y - 1] = true;
            canMove[x - 2][y - 1] = true;
```

```
                }
            } else if(((x - 2) > 0) && ((y - 1) > 0) && (Board[x - 2][y - 1] > 0)){
                protect[x-2][y-1] = true;
            }
            if (((x - 2) > 0) && ((y + 1) < 9) && (Board[x - 2][y + 1] <= 0)) {
                mobility++;
                if (Board[x - 2][y + 1] == 0) {
                    protect[x - 2][y + 1] = true;
                    threat[x - 2][y + 1] = true;
                    canMove[x - 2][y + 1] = true;
                } else {
                    threat[x - 2][y + 1] = true;
                    canMove[x - 2][y + 1] = true;
                }
            } else if(((x - 2) > 0) && ((y + 1) < 9) && (Board[x - 2][y + 1] > 0)){
                protect[x-2][y+1] = true;
            }
            if (((x - 1) > 0) && ((y + 2) < 9) && (Board[x - 1][y + 2] <= 0)) {
                mobility++;
                if (Board[x - 1][y + 2] == 0) {
                    protect[x - 1][y + 2] = true;
                    threat[x - 1][y + 2] = true;
                    canMove[x - 1][y + 2] = true;
                } else {
                    threat[x - 1][y + 2] = true;
                    canMove[x - 1][y + 2] = true;
                }
            } else if(((x - 1) > 0) && ((y + 2) < 9) && (Board[x - 1][y + 2] > 0)){
                protect[x-1][y+2] = true;
            }
        }
    }else{
        mobility = 0;
        catal = 0;
        value = 0;
        for(int i=1;i<9;i++){
            for(int j=1; j<9; j++){
                canMove[i][j] = false;
                protect[i][j] = false;
                threat[i][j] = false;
            }
        }
    }
  }
}
```

Queen.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package satranc;

/**
 *
 * @author Administrator
```

```
 */
public class queen {

    int x;
    int y;
    boolean exists;
    public queen(int coorX, int coorY, boolean var) {
        x = coorX;
        y = coorY;
        exists = var;
    }
    public double catal;
    public boolean threat[][] = new boolean[9][9];
    public boolean protect[][] = new boolean[9][9];
    public boolean canMove[][] = new boolean[9][9];
    public double mobility = 0;
    boolean end = false;
    int coorX;
    int coorY;
    double value = evaluate.valueQueen;

    void function(double Board[][]) {
        if (exists = true) {
            if (Board[x][y] < 0) {
                end = false;
                for (int a = 1; a < 9; a++) {
                    for (int b = 1; b < 9; b++) {
                        coorX = x;
                        coorY = y;
                        while ((coorX + 1 < 9) && (end = false)) {
                            if (Board[coorX + 1][coorY] == 0) {
                                mobility++;
                                protect[coorX + 1][coorY] = true;
                                threat[coorX + 1][coorY] = true;
                                canMove[coorX+1][coorY] = true;
                                coorX++;
                            } else if (Board[coorX + 1][coorY + 1] > 0) {
                                mobility++;
                                canMove[coorX+1][coorY] = true;
                                threat[coorX + 1][coorY] = true;
                                end = true;
                            } else {
                                protect[coorX + 1][coorY] = true;
                                end = true;
                            }
                        }

                        end = false;
                        coorX = x;
                        coorY = y;
                        while ((coorX - 1 < 9) && (end = false)) {
                            if (Board[coorX - 1][coorY] == 0) {
                                canMove[coorX-1][coorY] = true;
                                mobility++;
                                protect[coorX - 1][coorY] = true;
                                threat[coorX - 1][coorY] = true;
```

```
            coorX++;
        } else if (Board[coorX - 1][coorY] > 0) {
            mobility++;
            canMove[coorX-1][coorY] = true;
            threat[coorX - 1][coorY] = true;
            end = true;
        } else {
            protect[coorX - 1][coorY] = true;
            end = true;
        }
    }

    end = false;
    coorX = x;
    coorY = y;
    while ((coorY - 1 < 9) && (end = false)) {
        if (Board[coorX][coorY - 1] == 0) {
            mobility++;
            protect[coorX][coorY - 1] = true;
            threat[coorX][coorY - 1] = true;
            canMove[coorX][coorY-1] = true;
            coorX++;
        } else if (Board[coorX][coorY - 1] > 0) {
            mobility++;
            canMove[coorX][coorY-1] = true;
            threat[coorX][coorY - 1] = true;
            end = true;
        } else {
            protect[coorX][coorY - 1] = true;
            end = true;
        }
    }

    while ((coorY + 1 < 9) && (end = false)) {
        if (Board[coorX][coorY + 1] == 0) {
            mobility++;
            protect[coorX][coorY + 1] = true;
            threat[coorX][coorY + 1] = true;
            canMove[coorX][coorY+1] = true;
            coorX++;
        } else if (Board[coorX][coorY - 1] > 0) {
            mobility++;
            threat[coorX][coorY + 1] = true;
            canMove[coorX][coorY+1] = true;
            end = true;
        } else {
            protect[coorX][coorY + 1] = true;
            end = true;
        }
    }
}

if (Board[x][y] < 0) {
    coorX = x;
    coorY = y;
    while ((coorX + 1 < 9) && (coorY + 1 < 9) && (end = false)) {
```

```
   if (Board[coorX + 1][coorY + 1] == 0) {
      mobility++;
      protect[coorX + 1][coorY + 1] = true;
      threat[coorX + 1][coorY + 1] = true;
      canMove[coorX+1][coorY+1] = true;
      coorX++;
      coorY++;
   } else if (Board[coorX + 1][coorY + 1] > 0) {
      mobility++;
      threat[coorX + 1][coorY + 1] = true;
      canMove[coorX+1][coorY+1] = true;
      end = true;
   } else {
      protect[coorX][coorY] = true;
      end = true;
   }
}

coorX = x;
coorY = y;
end = false;
while ((coorX - 1 > 0) && (coorY - 1 > 0) && (end = false)) {
   if (Board[coorX - 1][coorY - 1] == 0) {
      mobility++;
      protect[coorX - 1][coorY - 1] = true;
      threat[coorX - 1][coorY - 1] = true;
      canMove[coorX-1][coorY-1] = true;
      coorX--;
      coorY--;
   } else if (Board[coorX - 1][coorY - 1] > 0) {
      mobility++;
      threat[coorX - 1][coorY - 1] = true;
      canMove[coorX-1][coorY-1] = true;
      end = true;
   } else {
      protect[coorX - 1][coorY - 1] = true;
      end = true;
   }
}

coorX = x;
coorY = y;
end = false;
while ((coorX + 1 < 9) && (coorY - 1 > 0) && (end = false)) {
   if (Board[coorX + 1][coorY - 1] == 0) {
      mobility++;
      protect[coorX + 1][coorY - 1] = true;
      threat[coorX + 1][coorY - 1] = true;
      canMove[coorX+1][coorY-1] = true;
      coorX++;
      coorY--;
   } else if (Board[coorX + 1][coorY - 1] > 0) {
      mobility++;
      threat[coorX + 1][coorY - 1] = true;
      canMove[coorX+1][coorY-1] = true;
      end = true;
```

```
            } else {
                protect[coorX + 1][coorY - 1] = true;
                end = true;
            }
        }

        coorX = x;
        coorY = y;
        end = false;
        while ((coorX - 1 > 0) && (coorY + 1 < 9) && (end = false)) {
            if (Board[coorX - 1][coorY + 1] == 0) {
                mobility++;
                protect[coorX - 1][coorY + 1] = true;
                threat[coorX - 1][coorY + 1] = true;
                canMove[coorX-1][coorY+1] = true;
                coorX--;
                coorY++;
            } else if (Board[coorX - 1][coorY + 1] > 0) {
                mobility++;
                canMove[coorX-1][coorY+1] = true;
                threat[coorX - 1][coorY + 1] = true;
                end = true;
            } else {
                protect[coorX - 1][coorY + 1] = true;
                end = true;
            }
        }
    }
  }
} else {
    for (int a = 1; a < 9; a++) {
        for (int b = 1; b < 9; b++) {
            coorX = x;
            coorY = y;
            while ((coorX + 1 < 9) && (end = false)) {
                if (Board[coorX + 1][coorY] == 0) {
                    mobility++;
                    canMove[coorX+1][coorY] = true;
                    protect[coorX + 1][coorY + 1] = true;
                    threat[coorX + 1][coorY + 1] = true;
                    coorX++;
                } else if (Board[coorX + 1][coorY + 1] < 0) {
                    mobility++;
                    canMove[coorX+1][coorY] = true;
                    threat[coorX + 1][coorY] = true;
                    end = true;
                } else {
                    protect[coorX + 1][coorY] = true;
                    end = true;
                }
            }

            end = false;
            coorX = x;
            coorY = y;
            while ((coorX - 1 < 9) && (end = false)) {
```

```
    if (Board[coorX - 1][coorY] == 0) {
        mobility++;
        protect[coorX - 1][coorY] = true;
        threat[coorX - 1][coorY] = true;
        canMove[coorX-1][coorY] = true;
        coorX++;
    } else if (Board[coorX - 1][coorY] < 0) {
        mobility++;
        threat[coorX - 1][coorY] = true;
        canMove[coorX-1][coorY] = true;
        end = true;
    } else {
        protect[coorX - 1][coorY] = true;
        end = true;
    }
}

end = false;
coorX = x;
coorY = y;
while ((coorY - 1 < 9) && (end = false)) {
    if (Board[coorX][coorY - 1] == 0) {
        mobility++;
        canMove[coorX][coorY-1] = true;
        protect[coorX][coorY - 1] = true;
        threat[coorX][coorY - 1] = true;
        coorX++;
    } else if (Board[coorX][coorY - 1] < 0) {
        mobility++;
        canMove[coorX][coorY-1] = true;
        threat[coorX][coorY - 1] = true;
        end = true;
    } else {
        protect[coorX][coorY - 1] = true;
        end = true;
    }
}

while ((coorY + 1 < 9) && (end = false)) {
    if (Board[coorX][coorY + 1] == 0) {
        mobility++;
        canMove[coorX-1][coorY+1] = true;
        protect[coorX][coorY + 1] = true;
        threat[coorX][coorY + 1] = true;
        canMove[coorX][coorY+1] = true;
        coorX++;
    } else if (Board[coorX][coorY - 1] < 0) {
        mobility++;
        canMove[coorX][coorY+1] = true;
        threat[coorX][coorY + 1] = true;
        end = true;
    } else {
        protect[coorX][coorY + 1] = true;
        end = true;
    }
}
```

```
}

if (Board[x][y] < 0) {
   coorX = x;
   coorY = y;
   while ((coorX + 1 < 9) && (coorY + 1 < 9) && (end = false)) {
      if (Board[coorX + 1][coorY + 1] == 0) {
         mobility++;
         canMove[coorX+1][coorY+1] = true;
         protect[coorX + 1][coorY + 1] = true;
         threat[coorX + 1][coorY + 1] = true;
         coorX++;
         coorY++;
      } else if (Board[coorX + 1][coorY + 1] < 0) {
         mobility++;
         canMove[coorX+1][coorY+1] = true;
         threat[coorX + 1][coorY + 1] = true;
         end = true;
      } else {
         protect[coorX][coorY] = true;
         end = true;
      }
   }

   coorX = x;
   coorY = y;
   end = false;
   while ((coorX - 1 > 0) && (coorY - 1 > 0) && (end = false)) {
      if (Board[coorX - 1][coorY - 1] == 0) {
         mobility++;
         protect[coorX - 1][coorY - 1] = true;
         threat[coorX - 1][coorY - 1] = true;
         canMove[coorX-1][coorY-1] = true;
         coorX--;
         coorY--;
      } else if (Board[coorX - 1][coorY - 1] < 0) {
         mobility++;
         threat[coorX - 1][coorY - 1] = true;
         canMove[coorX-1][coorY-1] = true;
         end = true;
      } else {
         protect[coorX - 1][coorY - 1] = true;
         end = true;
      }
   }

   coorX = x;
   coorY = y;
   end = false;
   while ((coorX + 1 < 9) && (coorY - 1 > 0) && (end = false)) {
      if (Board[coorX + 1][coorY - 1] == 0) {
         mobility++;
         protect[coorX + 1][coorY - 1] = true;
         threat[coorX + 1][coorY - 1] = true;
         canMove[coorX+1][coorY-1] = true;
         coorX++;
```

```
                    coorY--;
                } else if (Board[coorX + 1][coorY - 1] < 0) {
                    mobility++;
                    canMove[coorX+1][coorY-1] = true;
                    threat[coorX + 1][coorY - 1] = true;
                    end = true;
                } else {
                    protect[coorX + 1][coorY - 1] = true;
                    end = true;
                }
            }

            coorX = x;
            coorY = y;
            end = false;
            while ((coorX - 1 > 0) && (coorY + 1 < 9) && (end = false)) {
                if (Board[coorX - 1][coorY + 1] == 0) {
                    mobility++;
                    canMove[coorX-1][coorY+1] = true;
                    protect[coorX - 1][coorY + 1] = true;
                    threat[coorX - 1][coorY + 1] = true;
                    coorX--;
                    coorY++;
                } else if (Board[coorX - 1][coorY + 1] < 0) {
                    mobility++;
                    canMove[coorX-1][coorY+1] = true;
                    threat[coorX - 1][coorY + 1] = true;
                    end = true;
                } else {
                    protect[coorX - 1][coorY + 1] = true;
                    end = true;
                }
            }
        }
    }
  }else{
    mobility = 0;
    catal = 0;
    value = 0;
    for(int i=1;i<9;i++){
      for(int j=1; j<9; j++){
        canMove[i][j] = false;
        protect[i][j] = false;
        threat[i][j] = false;
      }
    }
  }
 }
}
```

King.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package satranc;

/**
 *
 * @author Administrator
 */
public class king {
    int x;
    int y;
    boolean exists;
    public king(int coorX, int coorY, boolean var) {
        x = coorX;
        y = coorY;
        exists = var;
    }
    boolean side = play.side;
    public boolean mat = false;
    public boolean etehdit[][] = new boolean[9][9];
    public boolean threat[][] = new boolean[9][9];
    public boolean protect[][] = new boolean[9][9];
    public boolean canCastle[][] = new boolean[9][9];
    public boolean canMove[][] = new boolean[9][9];
    boolean end = false;
    int coorX;
    int coorY;
    double value = evaluate.valueKing;
    void function(double Board[][]){
        if (exists = true){
            if (Board[x][y]<0){
                if((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]>0)&&!evaluate.etehdit[x-1][y-1]){
                    threat[x-1][y-1] = true;
                    canMove[x-1][y-1] = true;
                } else if ((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]==0)&&!evaluate.etehdit[x-1][y-1]){
                    protect[x-1][y-1] = true;
                    threat[x-1][y-1] = true;
                    canMove[x-1][y-1] = true;
                } else if((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]<0)&&!evaluate.etehdit[x-1][y-1]){
                    protect[x-1][y-1] = true;
                    canMove[x-1][y-1] = true;
                }
                if((x>0)&&(y-1>0)&&(Board[x][y-1]>0)&&!evaluate.etehdit[x][y-1]){
                    threat[x][y-1] = true;
                    canMove[x][y-1] = true;
                } else if ((x>0)&&(y-1>0)&&(Board[x][y-1]==0)&&!evaluate.etehdit[x][y-1]){
                    protect[x][y-1] = true;
                    threat[x][y-1] = true;
                    canMove[x][y-1] = true;
                } else if((x>0)&&(y-1>0)&&(Board[x][y-1]<0)&&!evaluate.etehdit[x][y-1]){
                    protect[x][y-1] = true;
                    canMove[x][y-1] = true;
                }
                if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]>0)&&!evaluate.etehdit[x+1][y-1]){
                    threat[x+1][y-1] = true;
                    canMove[x+1][y-1] = true;
                } else if ((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]==0)&&!evaluate.etehdit[x+1][y-1]){
```

```
    protect[x+1][y-1] = true;
    threat[x+1][y-1] = true;
    canMove[x+1][y-1] = true;
} else if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]<0)&&!evaluate.etehdit[x+1][y-1]){
    protect[x+1][y-1] = true;
    canMove[x+1][y-1] = true;

}
if((x+1>0)&&(y>0)&&(Board[x+1][y]>0)&&!evaluate.etehdit[x+1][y]){
    threat[x+1][y] = true;
    canMove[x+1][y] = true;
} else if ((x+1>0)&&(y>0)&&(Board[x+1][y]==0)&&!evaluate.etehdit[x+1][y]){
    protect[x+1][y] = true;
    threat[x+1][y] = true;
    canMove[x+1][y] = true;
} else if((x+1>0)&&(y>0)&&(Board[x+1][y]<0)&&!evaluate.etehdit[x+1][y]){
    protect[x+1][y] = true;
    canMove[x+1][y] = true;
}
if((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]>0)&&!evaluate.etehdit[x+1][y+1]){
    threat[x+1][y+1] = true;
    canMove[x+1][y+1] = true;
} else if ((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]==0)&&!evaluate.etehdit[x+1][y+1]){
    protect[x+1][y+1] = true;
    threat[x+1][y+1] = true;
    canMove[x+1][y+1] = true;
} else if((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]<0)&&!evaluate.etehdit[x+1][y+1]){
    protect[x+1][y+1] = true;
    canMove[x+1][y+1] = true;
}
if((x+1>0)&&(y>0)&&(Board[x+1][y]>0)&&!evaluate.etehdit[x+1][y]){
    threat[x+1][y] = true;
    canMove[x+1][y] = true;
} else if ((x+1>0)&&(y>0)&&(Board[x+1][y]==0)&&!evaluate.etehdit[x+1][y]){
    protect[x+1][y] = true;
    threat[x+1][y] = true;
    canMove[x+1][y] = true;
} else if((x+1>0)&&(y>0)&&(Board[x+1][y]<0)&&!evaluate.etehdit[x+1][y]){
    protect[x+1][y] = true;
    canMove[x+1][y] = true;
}
if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]>0)&&!evaluate.etehdit[x+1][y-1]){
    threat[x+1][y-1] = true;
    canMove[x+1][y-1] = true;
} else if ((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]==0)&&!evaluate.etehdit[x+1][y-1]){
    protect[x+1][y-1] = true;
    threat[x+1][y-1] = true;
    canMove[x+1][y-1] = true;
} else if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]<0)&&!evaluate.etehdit[x+1][y-1]){
    protect[x+1][y-1] = true;
    canMove[x+1][y-1] = true;
}
if((x>0)&&(y-1>0)&&(Board[x][y-1]>0)&&!evaluate.etehdit[x][y-1]){
    threat[x][y-1] = true;
    canMove[x][y-1] = true;
} else if ((x>0)&&(y-1>0)&&(Board[x][y-1]==0)&&!evaluate.etehdit[x][y-1]){
```

```
        protect[x][y-1] = true;
        threat[x][y-1] = true;
        canMove[x][y-1] = true;
      } else if((x>0)&&(y-1>0)&&(Board[x][y-1]<0)&&!evaluate.etehdit[x][y-1]){
        protect[x][y-1] = true;
        canMove[x][y-1] = true;
      }
      if((x==5)&&(y==1)&& (Board[8][1] == evaluate.valueRook)&&
          !evaluate.etehdit[6][1]&&!evaluate.etehdit[7][1]){
        canCastle[7][1] = true;
      }
      if (side = true){   // white
        if(x==5 && y==1&& Board[8][1] == -evaluate.valueRook&&!evaluate.etehdit[6][1]
            &&!evaluate.etehdit[7][1]){
          canCastle[7][1] = true;
        }
        if(x==5 && y==1&& Board[1][1] == -evaluate.valueRook&&!evaluate.etehdit[2][1]
            &&!evaluate.etehdit[3][1]&&!evaluate.etehdit[4][1]){
          canCastle[3][1] = true;
        }
      } else{
        if(x==5 && y==8&& Board[8][8] == -evaluate.valueRook&&!evaluate.etehdit[6][8]
            &&!evaluate.etehdit[7][8]){
          canCastle[7][8] = true;
        }
        if(x==5 && y==8&& Board[1][8] == -evaluate.valueRook&&!evaluate.etehdit[2][8]
            &&!evaluate.etehdit[3][8]&&!evaluate.etehdit[4][8]){
          canCastle[3][8] = true;
        }
      }
    } else{
      if((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]<0)&&!evaluate.tehdit[x-1][y-1]){
        threat[x-1][y-1] = true;
        canMove[x-1][y-1] = true;
      } else if ((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]==0)&&!evaluate.tehdit[x-1][y-1]){
        protect[x-1][y-1] = true;
        threat[x-1][y-1] = true;
        canMove[x-1][y-1] = true;
      } else if((x-1>0)&&(y-1>0)&&(Board[x-1][y-1]>0)&&!evaluate.tehdit[x-1][y-1]){
        protect[x-1][y-1] = true;
        canMove[x-1][y-1] = true;
      }
      if((x>0)&&(y-1>0)&&(Board[x][y-1]<0)&&!evaluate.tehdit[x][y-1]){
        threat[x][y-1] = true;
        canMove[x][y-1] = true;
      } else if ((x>0)&&(y-1>0)&&(Board[x][y-1]==0)&&!evaluate.tehdit[x][y-1]){
        protect[x][y-1] = true;
        threat[x][y-1] = true;
        canMove[x][y-1] = true;
      } else if((x>0)&&(y-1>0)&&(Board[x][y-1]>0)&&!evaluate.tehdit[x][y-1]){
        protect[x][y-1] = true;
        canMove[x][y-1] = true;
      }
      if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]<0)&&!evaluate.tehdit[x+1][y-1]){
        threat[x+1][y-1] = true;
        canMove[x+1][y-1] = true;
```

```
} else if ((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]==0)&&!evaluate.tehdit[x+1][y-1]){
   protect[x+1][y-1] = true;
   threat[x+1][y-1] = true;
   canMove[x+1][y-1] = true;
} else if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]>0)&&!evaluate.tehdit[x+1][y-1]){
   protect[x+1][y-1] = true;
   canMove[x+1][y-1] = true;
}
if((x+1>0)&&(y>0)&&(Board[x+1][y]<0)&&!evaluate.tehdit[x+1][y]){
   threat[x+1][y] = true;
   canMove[x+1][y] = true;
} else if ((x+1>0)&&(y>0)&&(Board[x+1][y]==0)&&!evaluate.tehdit[x+1][y]){
   protect[x+1][y] = true;
   threat[x+1][y] = true;
   canMove[x+1][y] = true;
} else if((x+1>0)&&(y>0)&&(Board[x+1][y]>0)&&!evaluate.tehdit[x+1][y]){
   protect[x+1][y] = true;
   canMove[x+1][y] = true;
}
if((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]<0)&&!evaluate.tehdit[x+1][y+1]){
   threat[x+1][y+1] = true;
   canMove[x+1][y+1] = true;
} else if ((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]==0)&&!evaluate.tehdit[x+1][y+1]){
   protect[x+1][y+1] = true;
   threat[x+1][y+1] = true;
   canMove[x+1][y+1] = true;
} else if((x+1>0)&&(y+1>0)&&(Board[x+1][y+1]>0)&&!evaluate.tehdit[x+1][y+1]){
   protect[x+1][y+1] = true;
   canMove[x+1][y+1] = true;
}
if((x+1>0)&&(y>0)&&(Board[x+1][y]<0)&&!evaluate.tehdit[x+1][y]){
   threat[x+1][y] = true;
   canMove[x+1][y] = true;
   canMove[x+1][y] = true;
} else if ((x+1>0)&&(y>0)&&(Board[x+1][y]==0)&&!evaluate.tehdit[x+1][y]){
   protect[x+1][y] = true;
   threat[x+1][y] = true;
   canMove[x+1][y] = true;
   canMove[x+1][y] = true;
} else if((x+1>0)&&(y>0)&&(Board[x+1][y]>0)&&!evaluate.tehdit[x+1][y]){
   protect[x+1][y] = true;
   canMove[x+1][y] = true;
   canMove[x+1][y] = true;
}
if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]<0)&&!evaluate.tehdit[x+1][y-1]){
   threat[x+1][y-1] = true;
   canMove[x+1][y-1] = true;
} else if ((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]==0)&&!evaluate.tehdit[x+1][y-1]){
   protect[x+1][y-1] = true;
   threat[x+1][y-1] = true;
   canMove[x+1][y-1] = true;
} else if((x+1>0)&&(y-1>0)&&(Board[x+1][y-1]>0)&&!evaluate.tehdit[x+1][y-1]){
   protect[x+1][y-1] = true;
   canMove[x+1][y-1] = true;
}
if((x>0)&&(y-1>0)&&(Board[x][y-1]<0)&&!evaluate.tehdit[x][y-1]){
```

```
                threat[x][y-1] = true;
                canMove[x][y-1] = true;
            } else if ((x>0)&&(y-1>0)&&(Board[x][y-1]==0)&&!evaluate.tehdit[x][y-1]){
                protect[x][y-1] = true;
                threat[x][y-1] = true;
                canMove[x][y-1] = true;
            } else if((x>0)&&(y-1>0)&&(Board[x][y-1]>0)&&!evaluate.tehdit[x][y-1]){
                protect[x][y-1] = true;
                canMove[x][y-1] = true;
            }
            if (side = true){   // white
                if(x==5 && y==1&& Board[8][1] == evaluate.valueRook&&!evaluate.tehdit[6][1]
                        &&!evaluate.tehdit[7][1]){
                    canCastle[7][1] = true;
                }
                if(x==5 && y==1&& Board[1][1] == evaluate.valueRook&&!evaluate.tehdit[2][1]
                        &&!evaluate.tehdit[3][1]&&!evaluate.tehdit[4][1]){
                    canCastle[3][1] = true;
                }
            } else{
                if(x==5 && y==8&& Board[8][8] == evaluate.valueRook&&!evaluate.tehdit[6][8]
                        &&!evaluate.tehdit[7][8]){
                    canCastle[7][8] = true;
                }
                if(x==5 && y==8&& Board[1][8] == evaluate.valueRook&&!evaluate.tehdit[2][8]
                        &&!evaluate.tehdit[3][8]&&!evaluate.tehdit[4][8]){
                    canCastle[3][8] = true;
                }
            }
        }
    }
    else{
        mat = true;
        value = 0;
    }

  }
}
```

Pawn.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package satranc;

/**
 *
 * @author Administrator
 */
public class pawn {
   int x;
   int y;
   boolean exists;
   public pawn(int coorX, int coorY, boolean var) {
```

```
    x = coorX;
    y = coorY;
    exists = var;
}
public boolean threat[][] = new boolean[9][9];
public boolean protect[][] = new boolean[9][9];
public boolean canMove[][] = new boolean[9][9];
boolean end = false;
boolean side;
String input= null;
int coorX;
int coorY;
double valueKnight = evaluate.valueKnight;
double valueQueen = evaluate.valueQueen;
double value = evaluate.valuePawn;

void function(double Board[][]){
    try{
        if(exists){
            if(side = true){ // white
                if(Board[x][y] < 0){
                    if(y == 2) {
                        if(Board[x][y+1] == 0 && Board[x][y+2] == 0){
                            canMove[x][y+2] = true;
                        }
                        if(Board[x][y+1] == 0){
                            canMove[x][y+1] = true;
                        }
                        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]>0){
                            canMove[x-1][y+1] = true;
                            threat[x-1][y+1] = true;
                        }
                        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]<0){
                            protect[x-1][y+1] = true;
                        }
                        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]>0){
                            canMove[x+1][y+1] = true;
                            threat[x+1][y+1] = true;
                        }
                        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]<0){
                            protect[x+1][y+1] = true;
                        }
                    } else if (y>2 && y<8){
                        if(Board[x][y+1] == 0){
                            canMove[x][y+1] = true;
                        }
                        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]>0){
                            canMove[x-1][y+1] = true;
                            threat[x-1][y+1] = true;
                        }
                        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]<0){
                            protect[x-1][y+1] = true;
                        }
                        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]>0){
                            canMove[x+1][y+1] = true;
                            threat[x+1][y+1] = true;
```

```
        }
        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]<0){
            protect[x+1][y+1] = true;
        }
    } else if(y==8){
        boolean test = false;
        while (test = false){
            System.out.println("Enter the name of the piece that you want\n");
            input = String.valueOf(System.in.read());
            if (input.equalsIgnoreCase("knight")){
                test = true;
                Board[x][y] = -valueKnight;
                knight promoted = new knight(x, y, true);
            } else if (input.equalsIgnoreCase("bishop")){
                test = true;
                Board[x][y] = -valueQueen;
                queen promoted = new queen(x, y, true);
            } else if (input.equalsIgnoreCase("rook")){
                test = true;
                Board[x][y] = -valueQueen;
                queen promoted = new queen(x, y, true);
            } else if (input.equalsIgnoreCase("queen")){
                test = true;
                Board[x][y] = -valueQueen;
                queen promoted = new queen(x, y, true);
            } else{
                System.out.println("Try again\n");
            }
        }
    }

} else {
    if(y == 2) {
        if(Board[x][y+1] == 0 && Board[x][y+2] == 0){
            canMove[x][y+2] = true;
        }
        if(Board[x][y+1] == 0){
            canMove[x][y+1] = true;
        }
        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]<0){
            canMove[x-1][y+1] = true;
            threat[x-1][y+1] = true;
        }
        if((x-1>0)&&(y+1<9) && Board[x-1][y+1]>0){
            protect[x-1][y+1] = true;
        }
        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]<0){
            canMove[x+1][y+1] = true;
            threat[x+1][y+1] = true;
        }
        if((x+1<9)&&(y+1<9) && Board[x+1][y+1]>0){
            protect[x+1][y+1] = true;
        }
    } else if (y>2 && y<7){
        if(Board[x][y+1] == 0){
            canMove[x][y+1] = true;
```

```
                }
                if((x-1>0)&&(y+1<9) && Board[x-1][y+1]<0){
                   canMove[x-1][y+1] = true;
                   threat[x-1][y+1] = true;
                }
                if((x-1>0)&&(y+1<9) && Board[x-1][y+1]>0){
                   protect[x-1][y+1] = true;
                }
                if((x+1<9)&&(y+1<9) && Board[x+1][y+1]>0){
                   protect[x+1][y+1] = true;
                }
                if((x+1<9)&&(y+1<9) && Board[x+1][y+1]<0){
                   canMove[x+1][y+1] = true;
                   threat[x+1][y+1] = true;
                }
             } else if(y==8){
                boolean test = false;
                while(test = false) {
                   System.out.println("Enter thename of the piece that you want?\n");
                   input = String.valueOf(System.in.read());
                   if (input.equalsIgnoreCase("knight")){
                      test = true;
                      Board[x][y] = valueKnight;
                      knight promoted2 = new knight(x, y, true);
                   } else if (input.equalsIgnoreCase("bishop")){
                      test = true;
                      Board[x][y] = valueQueen;
                      queen promoted2 = new queen(x, y, true);
                   } else if (input.equalsIgnoreCase("rook")){
                      test = true;
                      Board[x][y] = valueQueen;
                      queen promoted2 = new queen(x, y, true);
                   } else if (input.equalsIgnoreCase("queen")){
                      test = true;
                      Board[x][y] = valueQueen;
                      queen promoted2 = new queen(x, y, true);
                   } else{
                      System.out.println("Try again\n");

                   }
                }
             }
          }

       }
    }else{
       value = 0;
       for(int i=1;i<9;i++){
          for(int j=1; j<9; j++){
             canMove[i][j] = false;
             protect[i][j] = false;
             threat[i][j] = false;
          }
       }
    }
 }catch(Exception e){
```

```
        }
    }
}
```

Evaluate.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package satranc;

/**
 *
 * @author Administrator
 */
public class evaluate {
    public boolean exists;
    public evaluate(){
    }
    public static boolean etehdit[][]= new boolean[9][9];
    public static boolean tehdit[][]= new boolean[9][9];
    public static boolean turn;
    public static double valueKnight;
    public static double valueQueen;
    public static double valueBishop;
    public static double valueRook;
    public static double valuePawn;
    public static double valueKing;
    public boolean at[]= new boolean[2];
    public boolean eat[] = new boolean[2];
    public boolean fil[] = new boolean[2];
    public boolean efil[] = new boolean[2];
    public boolean kale[] = new boolean[2];
    public boolean ekale[] = new boolean[2];
    public boolean piyon[]= new boolean[8];
    public boolean epiyon[] = new boolean[8];
    public boolean promotion[] = new boolean[2];
    public boolean vezir, evezir, sah, esah;
    public static knight at1 = new knight(1,1, false);
    public static knight at2 = new knight(1,1, false);
    public static knight eat1 = new knight(1,1, false);
    public static knight eat2 = new knight(1,1, false);
    public static bishop fil1 = new bishop(1,1, false);
    public static bishop fil2 = new bishop(1,1, false);
    public static bishop efil1 = new bishop(1,1, false);
    public static bishop efil2 = new bishop(1,1, false);
    public static rook kale1 = new rook(1,1, false);
    public static rook kale2 = new rook(1,1, false);
    public static rook ekale1 = new rook(1,1, false);
    public static rook ekale2 = new rook(1,1, false);
    public static queen vezir1 = new queen(1,1, false);
    public static queen evezir1 = new queen(1,1, false);
    public static king sah1 = new king(1,1, false);
    public static king esah1 = new king(1, 1, false);
    public static pawn piyon1 = new pawn(1,1, false);
```

```
public static pawn epiyon1 = new pawn(1,1, false);
public static pawn piyon2 = new pawn(1,1, false);
public static pawn epiyon2 = new pawn(1,1, false);
public static pawn piyon3 = new pawn(1,1, false);
public static pawn epiyon3 = new pawn(1,1, false);
public static pawn piyon4 = new pawn(1,1, false);
public static pawn epiyon4 = new pawn(1,1, false);
public static pawn piyon5 = new pawn(1,1, false);
public static pawn epiyon5 = new pawn(1,1, false);
public static pawn piyon6 = new pawn(1,1, false);
public static pawn epiyon6 = new pawn(1,1, false);
public static pawn piyon7 = new pawn(1,1, false);
public static pawn epiyon7 = new pawn(1,1, false);
public static pawn piyon8 = new pawn(1,1, false);
public static pawn epiyon8 = new pawn(1,1, false);
private double mob_mul, catal_mul;
private double value;
int i,j,k;

public double function(double Board[][]){
   for(i=0;i<2;i++){
      at[i] = false;
      eat[i] = false;
      fil[i] = false;
      efil[i] = false;
      kale[i] = false;
      ekale[i] = false;
      piyon[i] = false;
      epiyon[i] = false;
      promotion[i] = false;
   }
   for(i=2;i<8;i++){
      piyon[i] = false;
      epiyon[i] = false;
   }
   vezir = false;
   evezir = false;
   sah = false;
   esah = false;

   for(i=1;i<9;i++){
      for(j=1;j<9;j++){
         if(Board[i][j] == -valueKnight){
            if(!at[0]){
               at[0] = true;
               knight at1 = new knight(i , j, true);
            } else if(at[0]&&!at[1]){
               at[1] = true;
               knight at2 = new knight(i, j, true);
            }
         } else if(Board[i][j] == valueKnight){
            if(!eat[0]){
               eat[0] = true;
               knight eat1 = new knight(i , j, true);
            } else if(eat[0]&&!eat[1]){
               eat[1] = true;
```

```
        knight eat2 = new knight(i, j, true);
    }
} else if(Board[i][j] == -valueBishop){
    if(!fil[0]){
        fil[0] = true;
        bishop fil1 = new bishop(i , j, true);
    } else if(fil[0]&&!fil[1]){
        at[1] = true;
        bishop fil2 = new bishop(i, j, true);
    }
}else if(Board[i][j] == valueBishop){
    if(!efil[0]){
        efil[0] = true;
        bishop efil1 = new bishop(i , j, true);
    } else if(efil[0]&&!efil[1]){
        eat[1] = true;
        bishop efil2 = new bishop(i, j, true);
    }
} else if(Board[i][j] == -valueRook){
    if(!kale[0]){
        kale[0] = true;
        rook kale1 = new rook(i , j, true);
    } else if(kale[0]&&!kale[1]){
        kale[1] = true;
        rook kale2 = new rook(i, j, true);
    }
}else if(Board[i][j] == valueRook){
    if(!ekale[0]){
        ekale[0] = true;
        rook ekale1 = new rook(i , j, true);
    } else if(ekale[0]&&!ekale[1]){
        ekale[1] = true;
        rook ekale2 = new rook(i, j, true);
    }
} else if(Board[i][j] == -valueQueen){
    if(!vezir){
        vezir = true;
        queen vezir1 = new queen(i, j, true);
    }
}else if(Board[i][j] == valueQueen){
    if(!evezir){
        evezir = true;
        queen evezir1 = new queen(i, j, true);
    }
}else if(Board[i][j] == -valueKing){
    if(!sah){
        sah = true;
        king sah1 = new king(i, j, true);
    }
}else if(Board[i][j] == valueKing){
    if(!esah){
        esah = true;
        king esah1 = new king(i , j, true);
    }
} else if(Board[i][j] == -valuePawn){
    if(!piyon[0]){
```

```
                piyon[0] = true;
                pawn piyon1 = new pawn(i, j, true);
              }else if(piyon[0]&&!piyon[1]){
                piyon[1] = true;
                pawn piyon2 = new pawn(i, j, true);
              }else if(piyon[0]&&piyon[1]&&!piyon[2]){
                piyon[2] = true;
                pawn piyon3 = new pawn(i, j, true);
              }else if(piyon[0]&&piyon[1]&&piyon[2]&&!piyon[3]){
                piyon[3] = true;
                pawn piyon4 = new pawn(i, j, true);
              }else if(piyon[0]&&piyon[1]&&piyon[2]&&piyon[3]&&!piyon[4]){
                piyon[4] = true;
                pawn piyon5 = new pawn(i, j, true);
              }else if(piyon[0]&&piyon[1]&&piyon[2]&&piyon[3]&&piyon[4]&&!piyon[5]){
                piyon[5] = true;
                pawn piyon6 = new pawn(i, j, true);
              }else
if(piyon[0]&&piyon[1]&&piyon[2]&&piyon[3]&&piyon[4]&&piyon[5]&&!piyon[6]){
                piyon[6] = true;
                pawn piyon7 = new pawn(i, j, true);
              }else
if(piyon[0]&&piyon[1]&&piyon[2]&&piyon[3]&&piyon[4]&&piyon[5]&&piyon[6]&&!piyon[7]){
                piyon[7] = true;
                pawn piyon7 = new pawn(i, j, true);
              }
            }else if(Board[i][j] == valuePawn){
              if(!epiyon[0]){
                epiyon[0] = true;
                pawn epiyon1 = new pawn(i, j, true);
              }else if(epiyon[0]&&!epiyon[1]){
                epiyon[1] = true;
                pawn epiyon2 = new pawn(i, j, true);
              }else if(epiyon[0]&&epiyon[1]&&!epiyon[2]){
                epiyon[2] = true;
                pawn epiyon3 = new pawn(i, j, true);
              }else if(epiyon[0]&&epiyon[1]&&epiyon[2]&&!epiyon[3]){
                epiyon[3] = true;
                pawn epiyon4 = new pawn(i, j, true);
              }else if(epiyon[0]&&epiyon[1]&&epiyon[2]&&epiyon[3]&&!epiyon[4]){
                epiyon[4] = true;
                pawn epiyon5 = new pawn(i, j, true);
              }else if(epiyon[0]&&epiyon[1]&&epiyon[2]&&epiyon[3]&&epiyon[4]&&!epiyon[5]){
                epiyon[5] = true;
                pawn epiyon6 = new pawn(i, j, true);
              }else
if(epiyon[0]&&epiyon[1]&&epiyon[2]&&epiyon[3]&&epiyon[4]&&epiyon[5]&&!epiyon[6]){
                epiyon[6] = true;
                pawn epiyon7 = new pawn(i, j, true);
              }else
if(epiyon[0]&&epiyon[1]&&epiyon[2]&&epiyon[3]&&epiyon[4]&&epiyon[5]&&epiyon[6]&&!epiy
on[7]){
                epiyon[7] = true;
                pawn epiyon7 = new pawn(i, j, true);
              }
            }
```

```
        }
      }
      value +=  (at1.mobility + at2.mobility - eat1.mobility - eat2.mobility + fil1.mobility +
fil2.mobility
            - efil1.mobility - efil2.mobility + vezir1.mobility - evezir1.mobility)*mob_mul;
      value += (at1.catal + at2.catal + fil1.catal + fil2.catal + kale1.catal + kale2.catal - eat1.catal
            - eat2.catal - efil1.catal - efil2.catal - ekale1.catal - ekale2.catal) * catal_mul;
      value += at1.value + at2.value - eat1.value - eat2.value + fil1.value + fil2.value - efil1.value -
efil2.value
            + kale1.value + kale2.value - ekale1.value - ekale2.value + vezir1.value - evezir1.value
            + sah1.value - esah1.value + piyon1.value + piyon2.value + piyon3.value + piyon4.value +
piyon5.value
            + piyon6.value + piyon7.value + piyon8.value - epiyon1.value - epiyon2.value -
epiyon3.value
            - epiyon4.value - epiyon5.value - epiyon6.value - epiyon7.value - epiyon8.value;
      return value;
   }
}
```

Minimax.java

```
/*
 * Minimax.java
 *
 * Created on June 5, 2008, 12:07 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package satranc;

import java.util.ArrayList;
import javax.swing.tree.DefaultMutableTreeNode;

/**
 *
 * @author Administrator
 */
public class Minimax {
   public ArrayList<String> canMove1 = new ArrayList();
   public ArrayList<String> canMove2 = new ArrayList();
   public double Board1[][] = new double[9][9];
   public double Board2[][] = new double[9][9];
   public evaluate sonuc = new evaluate();
   public evaluate sonuc3 = new evaluate();
    public evaluate sonuc2 = new evaluate();
   /** Creates a new instance of Minimax */
   public Minimax() {
   }
   public ArrayList maxCanMove(double Board[][]) {
      sonuc.function(Board);
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.at1.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.at1.x) + Integer.toString(sonuc.at1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
```

```
            }
        }
        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.at2.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.at2.x) + Integer.toString(sonuc.at2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.fil1.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.fil1.x) + Integer.toString(sonuc.fil1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.fil2.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.fil2.x) + Integer.toString(sonuc.fil2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.kale1.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.kale1.x) + Integer.toString(sonuc.kale2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }
        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.kale2.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.kale2.x) + Integer.toString(sonuc.kale2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.vezir1.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.vezir1.x) + Integer.toString(sonuc.vezir1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.sah1.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.sah1.x) + Integer.toString(sonuc.sah1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }
```

```
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon1.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon1.x) + Integer.toString(sonuc.piyon1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon2.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon2.x) + Integer.toString(sonuc.piyon2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon3.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon3.x) + Integer.toString(sonuc.piyon3.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon4.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon4.x) + Integer.toString(sonuc.piyon5.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon5.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon5.x) + Integer.toString(sonuc.piyon5.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon6.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon6.x) + Integer.toString(sonuc.piyon6.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon7.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon7.x) + Integer.toString(sonuc.piyon7.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }
```

```
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.piyon8.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.piyon8.x) + Integer.toString(sonuc.piyon8.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }
      return canMove1;
   }

   public ArrayList minCanMove(double Board[][]) {
      sonuc.function(Board);
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.eat1.canMove[i][j] = true){
            canMove1.add(Integer.toString(sonuc.eat1.x) + Integer.toString(sonuc.eat1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.eat2.canMove[i][j] = true){
            canMove2.add(Integer.toString(sonuc.eat2.x) + Integer.toString(sonuc.eat2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.fil1.canMove[i][j] = true){
            canMove2.add(Integer.toString(sonuc.efil1.x) + Integer.toString(sonuc.efil1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.efil2.canMove[i][j] = true){
            canMove2.add(Integer.toString(sonuc.efil2.x) + Integer.toString(sonuc.efil2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }

      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.ekale1.canMove[i][j] = true){
            canMove2.add(Integer.toString(sonuc.ekale1.x) + Integer.toString(sonuc.ekale2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
            }
      }
      for(int i=1; i<9; i++){
         for(int j =1; j<9; j++)
            if(sonuc.ekale2.canMove[i][j] = true){
            canMove2.add(Integer.toString(sonuc.ekale2.x) + Integer.toString(sonuc.ekale2.y) +  " " +
Integer.toString(i) +Integer.toString(j));
```

```
            }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.evezir1.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.evezir1.x) + Integer.toString(sonuc.evezir1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.esah1.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.esah1.x) + Integer.toString(sonuc.esah1.y) +  " " +
Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon1.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon1.x) + Integer.toString(sonuc.epiyon1.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon2.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon2.x) + Integer.toString(sonuc.epiyon2.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon3.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon3.x) + Integer.toString(sonuc.epiyon3.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon4.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon4.x) + Integer.toString(sonuc.epiyon5.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon5.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon5.x) + Integer.toString(sonuc.epiyon5.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
```

```
            }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon6.canMove[i][j] = true){
                canMove2.add(Integer.toString(sonuc.epiyon6.x) + Integer.toString(sonuc.epiyon6.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon7.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.epiyon7.x) + Integer.toString(sonuc.epiyon7.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }

        for(int i=1; i<9; i++){
            for(int j =1; j<9; j++)
                if(sonuc.epiyon8.canMove[i][j] = true){
                canMove1.add(Integer.toString(sonuc.epiyon8.x) + Integer.toString(sonuc.epiyon8.y) +  " "
+ Integer.toString(i) +Integer.toString(j));
                }
        }
        return canMove2;
    }


    public String Tree(double Board[][]){
        String temp;
        String temp2;
        String temp3;
        int ilkx;
        int ilky;
        int sonx;
        int sony;
        char initial1;
        char initial2;
        char final1;
        char final2;
        ArrayList first = new ArrayList();
        ArrayList second = new ArrayList();
        ArrayList third = new ArrayList();
        double Board1[][] = new double[9][9];
        double Board2[][] = new double[9][9];
        double Board3[][] = new double[9][9];
        DefaultMutableTreeNode agac = new DefaultMutableTreeNode();
        first = maxCanMove(Board);
        for(int i =0; i<first.size(); i++){
            Board1 = Board;
            temp = (String)first.get(i);
            initial1 = temp.charAt(0);
            initial2 = temp.charAt(1);
            final1 = temp.charAt(3);
```

```
            final2 = temp.charAt(4);
            ilkx = Integer.valueOf(initial1);
            ilky = Integer.valueOf(initial2);
            sonx = Integer.valueOf(final1);
            sony = Integer.valueOf(final2);
            Board1[sonx][sony] = Board1[ilkx][ilky];
            Board1[ilkx][ilky] = 0;
            second = minCanMove(Board1);
            agac.add(new DefaultMutableTreeNode());
            for(int j =0; j<second.size(); j++){
               Board2 = Board1;
               temp = (String)second.get(j);
               initial1 = temp.charAt(0);
               initial2 = temp.charAt(1);
               final1 = temp.charAt(3);
               final2 = temp.charAt(4);
               ilkx = Integer.valueOf(initial1);
               ilky = Integer.valueOf(initial2);
               sonx = Integer.valueOf(final1);
               sony = Integer.valueOf(final2);
               Board2[sonx][sony] = Board2[ilkx][ilky];
               Board2[ilkx][ilky] = 0;
               third = maxCanMove(Board2);
               ((DefaultMutableTreeNode) agac.getChildAt(i)).add(new DefaultMutableTreeNode());
               for(int k = 0; k<third.size(); k++){
                  Board3 = Board2;
                  temp = (String)second.get(j);
                  initial1 = temp.charAt(0);
                  initial2 = temp.charAt(1);
                  final1 = temp.charAt(3);
                  final2 = temp.charAt(4);
                  ilkx = Integer.valueOf(initial1);
                  ilky = Integer.valueOf(initial2);
                  sonx = Integer.valueOf(final1);
                  sony = Integer.valueOf(final2);
                  Board3[sonx][sony] = Board2[ilkx][ilky];
                  Board3[ilkx][ilky] = 0;
                  ((DefaultMutableTreeNode) agac.getChildAt(i).getChildAt(j)).add(new
DefaultMutableTreeNode(sonuc2.function(Board3)));

               }
            }
         }
      }

}
```