Othello

For More Details

This project was implemented for the needs of the course of AI, in the context of my undergraduate studies. The main goal was to create a program that would allow the user to play the Othello game against his computer. The approach used uses the MiniMax algorithm with a-b pruning.

Rules

A key part of the implementation was ensuring the application of the rules of the game. This is achieved by using the two classes, OthelloPanel and ControlMoves.

ControlMoves

In this class there are eight methods that cover the possible directions - up, down, right, left as well as the four diagonals.

Example - up :

```
public boolean lookUp(int player,int board[][],int i, int j) {
    int opponent = findOpponent(player);
    int up = i - 1;
    if(up > 0 && board[up][j] == opponent) {

            while(up >= 0 && board[up][j] == opponent){
            up--;
         }

            if(up >= 0 && board[up][j] == player) {
                    //this cell is a possible move
                    return true;
            }
    }
    return false;
}
```

Using them, the MovesToPlay() method creates a list of the next valid moves of each player and ChangePoints() by selecting a valid move gives the new status of the game.

The final addition is the findOpponent() method, which based on the current player finds the opponent's id.

The contract that has been made for the needs of the methods coordination is :

- The blanks in the table are denoted by 0,
- the positions of the black birds with 1 - first move make the black,
- the places of the white birds with 2 - second movement make the whites
- and respectively the id of the first player is 1 and of the second 2.

OthelloPanel

In this class there are methods for graphics and part of the user interface, as well as methods that coordinate the functions between the player-user and the player-computer.

The most important is changePlayers() :

```
//Control Players -change between them
public void changePlayers() {

    if(this.totalPieces<64) {
        //check moves for this player
        moves = cm.MovesToPlay(this.board, this.turn);
        if(!moves.isEmpty()) {
            System.out.println("player : "+this.turn);
            highlights(moves,true);
            if(this.turn==pc) {
                //pc's turn
                PC_handlerTimer.start();
            }else {
                //human player's turn
                this.awaitForClick = true;
            }
        }
        //if he has no moves check for the other player
        else {
            //this method takes currents players id and changes it to next players id
            changeTurns();
```

```
                moves = cm.MovesToPlay(this.board, this.turn);
                if(!moves.isEmpty()) {
                    highlights(moves,true);
                    if(this.turn==pc) {
                         //pc's turn
                        PC_handlerTimer.start();
                    }else {
                        //human player's turn
                        this.awaitForClick = true;
                    }
                }
                else {
                    //if you are here the game is over
                    findWinner();
                    endGame();
                }
            }
        }else {
            findWinner();
            endGame();
        }
}
```

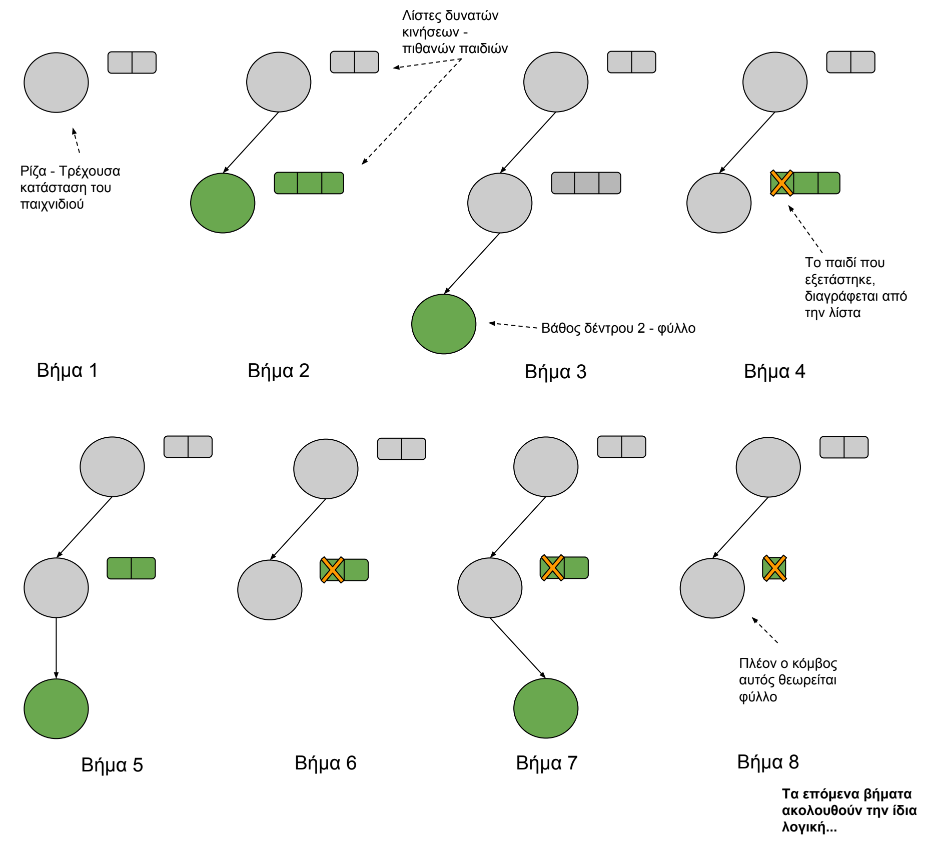
The command line displays messages about who is playing each time and when someone loses their turn.

MiniMax

The MiniMax algorithm was implemented as a dynamic tree to offer memory savings.
In each situation where the algorithm is called to decide its movement, the node of the current state is created (root node) and in it a list of points [x, y] concerning the possible options - its candidate children. Each point is examined in turn, the corresponding node is created for it, with a list of its own possible children and depending on the depth possibility given by the user, it ends after n movements in sheet mode. Another way to end up in a sheet state is the absence of permissible moves.
Then its value is calculated, the father of the leaf is called and examines (either as Max or as Min) if he wants to keep it, the leaf - child is destroyed.
The father then removes the child being examined from the list and checks if he has other children to examine. If not, he enters the category of the sheet - with the value he chose to keep from one of his children - and the process continues accordingly.
When her children's list is empty at the root, the move is returned.
Throughout this process, before the creation of a new node, the check for sawing

a-b is made and in cases where it is decided, the point is deleted from the list of children and the node as well as those who would succeed it are never created.

Figure 1: Dynamic MiniMax algorithm. Pruning a-b is not shown.


Λίστες δυνατών κινήσεων - πιθανών παιδιών
Ρίζα - Τρέχουσα κατάσταση του παιχνιδιού
Βάθος δέντρου 2 - φύλλο
Το παιδί που εξετάστηκε, διαγράφεται από την λίστα
Βήμα 1
Βήμα 2
Βήμα 3
Βήμα 4
Πλέον ο κόμβος αυτός θεωρείται φύλλο
Βήμα 5
Βήμα 6
Βήμα 7
Βήμα 8
Τα επόμενα βήματα ακολουθούν την ίδια λογική...

All of the above are implemented in the MinMaxTree and Node classes.

Evaluation

The evaluation of the leaves of the tree is done in the ValueOfState class.
To improve it, some different methods were tried.

In the beginning there was simply the choice based on the maximum number of birds that the computer would take from each possible movement. But it was found that it was not so efficient.

Weights were then added to each move - with the number of inverted birds playing a smaller role, especially at the beginning of the game..

These weights were selected based on criteria, tips from various sources - for ways to better choose movements.

```
int[][] b = {
    {100, -5, 10 , 5 ,5 , 10, -5, 100},
    {-5 ,-5 , -1, -1, -1, -1, -5, -5},
    {10 , -1,20 , 5 , 5 ,20 , -1, 10},
    { 5 , -1, 5 ,10 ,10 , 5 , -1, 5 },
    { 5 , -1, 5 ,10 ,10 , 5 , -1, 5 },
    {10 , -1,20 , 5 , 5 ,20 , -1, 10},
    {-5 , -5, -1, -1, -1, -1, -5, -5},
    {100, -5, 10 , 5 , 5 ,10 , -5, 100}};
```

At the same time it seemed to give better results, adding more value in case of final situation, with computer victory.
This is done in the method valueByVictory().