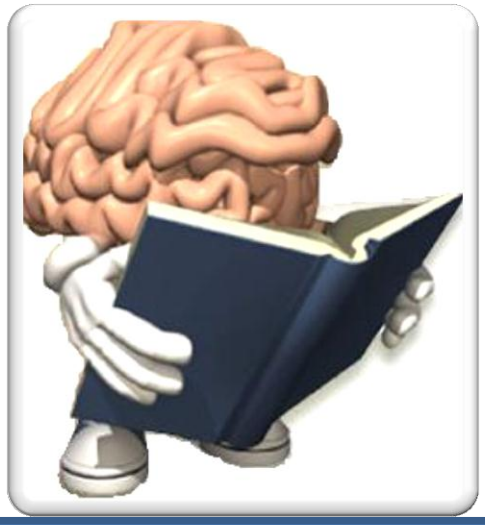

Project Report

Unbeatable Othello version 1.0

Submitted By :

Shubham Jain – y08uc118

Tanuj Sharma – y08uc131

Deepak Joshi – y08uc044

Acknowledgments

Our sincere thanks to our faculty advisor, ***Prof. Subrat K. Dash***, for offering his valuable advice and unconditional support from the very start of this project. Without his ideas, this project would not have exhibited some of the startling features.

Report Overview

This document will explain some of the fundamental principles that are essential for the construction of the application and will go on to the description of the development process comprising conclusions in the closing segment.

Note : For completeness and intrinsic knowledge, please refer to the source code available on http://users.lnmiit.ac.in/~y08uc118/index.html

Introduction

The game Othello is considered to be a game that requires comprehensive experience to be mastered. It is also said that Othello ``takes a minute to learn and a lifetime to master". What makes it particularly interesting is the difficulty in telling which player is in a point of advantage until the late stages of the game. This is, in principle, where lack of skill and experience take their toll. ***Unbeatable Othello*** is an implementation of the game Othello in which a user can play with a computer.

Rules of Unbeatable Othello

1. Othello is played on a conventional 8-by-8 ``checker" board, which initially has two white and two black pieces in the center four squares with the white pieces and black pieces aligned diagonally.
2. There are two players, black and white, and they alternate turns.
3. Black moves first by placing a black piece on the board.
4. If player stones surround the opponent's stone(s) horizontally, vertically, or diagonally, the opponent's surrounded stone(s) shall change into player stone(s).
5. Players can put their stones only on the place where they can surround their opponent's stone(s).
6. The game ends when either of the player do not have a legal move, and the player with the most pieces on the board wins.

Special Features of Unbeatable Othello

1. ***Inbuilt Animation*** when turning the stones color.
2. User can also get the best move as a hint from the help of computer using the "***Hint Move***" button.
3. User can also undo computer's move by using the "***Undo Move***" button.
4. A new game instance can be started at any point of time using the "***New Game***" button.
5. Top-right corner of the game displays the no. of stones of each player on the board.
6. The game screen also displays which player turn it is.
7. When the game ends, the game screen displays the game result "You Lost" or "You Won" in a very dramatical manner.

Source Code

The code of our program has been written in JAVA Applet (jdk1.6.0_20 version). The game can run on any platform (Windows/Unix/Linux/Mac) having Java Runtime Environment.

In our program, we have made two classes :

1. first class is Othello
2. second class is othelloPlayer

In the Othello class, all the graphics functions and "user" turn function are declared. And in othelloPlayer class, the "computer" turn function are declared.

The 'computer' turn functions are the brain part of our code.

Graphics functions:-

- *Paint( )* : this function paint the board after every turn.
- *Drawboard( )* : this function is called by the paint for drawing the board.
- *countStone( )* : this function count the stone present on the board for each player.
- *drawStone( )* : this function draw the no. of stone on the board of each player. This function calls the countStone function.
- *drawTurn( )* : this function shows the which player has the chance presently.
- *showWinner( )* : this function shows the winner of the game. This function actually call the *endgame( )* function.

There are other functions also but above function are the core part of the graphics.

Non-Graphics functions:-

- *mouseUp( )* : this function is the main function and this called whenever mouse is clicked on any place on the board.
- *checkAll( )* : this function whether a chance can be taken on any board position or not.
- *checkStone( )* : this function checks the particular position whether a stone can be put or not at that position.
- *turnStone( )* : this function actually turn the stone of the opposite player after the chance of the present player.

othelloPlayer class functions:-

- *computer.decide( )* : this function is the mind of the Othello computer palyer. This function actually find the best move position and then put the stone there as well as turn the corresponding stone of the opposite player. In the *computer.decide( )* function actually there are two options first for beginner player and other for the expert player.

Algorithm Implementation

Heuristic Approach :

We have assigned heuristic values to each of the cell of 8x8 Othello board. The heuristic value denotes the likeliness of the selecting that cell. These heuristic values are evaluated after playing the Othello game a large no. of times.

Heuristic Value Table (assigning heuristic values to each cell) :-

8x8	1	2	3	4	5	6	7	8
1	1.00	-0.25	0.10	0.05	0.05	0.10	-0.25	1.00
2	-0.25	-0.25	0.01	0.01	0.01	0.01	-0.25	-0.25
3	0.10	0.01	0.05	0.02	0.02	0.05	0.01	0.10
4	0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
5	0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
6	0.10	0.01	0.05	0.02	0.02	0.05	0.01	0.10
7	-0.25	-0.25	0.01	0.01	0.01	0.01	-0.25	-0.25
8	1.00	-0.25	0.10	0.05	0.05	0.10	-0.25	1.00

Utility Function :

The utility function give the utility value of the particular cell.

Utility value = flipcount (no. of stones flipped) * heuristic value (of that cell)

On the bases of these utility values, computer chooses the best move available using minimax algorithm. More the utility value, more preferable move it is.

Minimax Algorithm :

(the principles described here and onwards are strictly concerned with games where 2 players are involved and the information is complete)

In a game tree, we can incorporate the results obtained from the utility function. In other words, we can assign to each node of the tree the returned utility value once it has been calculated.

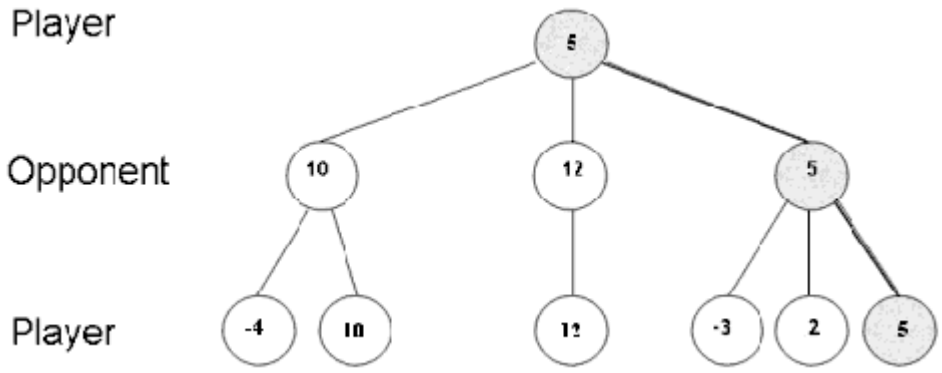

Figure : Minimax algorithm

Knowing that each of the players wishes to maximise his/her pay-off at any given opportunity, we can predict the path that will be followed down the tree. In above figure, it is the case that high values are advantageous to Player 1, whereas Player 2 is after small values. That means that Player 1 will strive to make a maximal choice, whereas Player 2 will settle for the minimal choice. Given these concepts, we finally see why the path picked is sensible. We choose the maximum, minimum, again maximum and so forth at each level of the tree. The name *minimax* has been extracted from this very intriguing behavior.

A substantial step towards our goal has been taken once the minimax algorithm has been realised.

Conclusions

The project implemented and employed algorithms sufficiently strong to perform convincingly well against Human user.

The project may contribute and assist others as it lays out successful approaches for artificially playing this game (or possibly any game) as well as bad approaches. This could push the project even further ahead and result in some slight improvements.