OTHELLO Masters

TEAM MEMBERS

1. Eashan Gupta - 160050045
2. Ankit Goyal - 160050046
3. Ritesh Goenka - 160050047

Introduction

The aim of this project was to explore new things besides the course curriculum and learn to implement abstractions in large programs. We have tried our best to do so keeping the problem simple while implementing some new features.

Description of the Problem

- What you can expect?
  Our project is the DrRacket implementation of the game Othello. It has two modes- single player mode (Vs Computer) and a double player mode. Besides this it also displays the rules of the game and we have also included a testing function to test our single player algorithm vs the greedy algorithm for Othello.

- Basic Rules of the games:
  The two players take alternate turns. If one player cannot make a valid move, the play passes back to the other player. When neither player can move, the game ends. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining squares.
  At the end of the game, the person with more number of pieces wins. So the match may also end in a draw.
  Detailed rules have been covered in our implementation.

Basic Design

- Board and Discs
  The board has been represented as a 2d-vector of 8*8 size. All the squares on the board are represented in the form of structures which store the color of disc placed on them along with the dynamic weights (covered later) assigned to them. Black disc is represented by -1 state while white by +1 state and empty square by 0.

- Graphics
  We have used the Racket GUI library already provided by latest versions of DrRacket for displaying the board, discs, images, buttons etc.

- External Layout

  There are a total of three files along with 6 photos:
  - main.rkt – Contains the project
  - high-score.txt – It stores the latest high-score scored by the player
  - Create new file.rkt – It refreshes the above text file
  - 6 images required in the program

- Internal structure
  - *Abstractions:*

    Various abstractions have been used to minimize the repetition of code.
    Many global variables and states have been defined to reduce the execution time.

    Classes and structures:
    - *Structure square* - stores the data of each square
    - Other than this many classes provided by the Racket GUI library have been used.

    Macros- Mainly 2 macros, namely lc and while, have been used in our program to minimize the code.

    Valid-blocks-dir function – It is function which can determine the validity of a square w.r.t a direction specified by its arguments.

    Higher order functions:
    - *Id function*- A function which determines the number of disc flips along a direction specified by its argument function.
    - *Traverse function*- A function which applies its argument, a function to all the elements of the board.
    - We have also used inbuilt higher order functions like foldr and map.

- Other Features
  - *Multiple modes:*

    Our game has 2 modes viz. Single Player and Double Player.
  - *Undo:*

    It is used to undo or get back to any previous position.
  - *High-score display:*

    It is a feature where we store and display the high-score for the single player mode. Since it is stored in a file, it will display the same high-score even after running the program again. For this input and output ports have been used.
  - *Testing function:*

    This function is used to test our Single player algorithm against a function which implements the greedy algorithm. Since some moves are chosen randomly, the game is played n number of times where n can be chosen by the user to suit him to check the accuracy. This function is merely made to collect the data as to how our program fares against an amateur player (who is expected to play based on the greedy algorithm).

Algorithm for Single Player

We have used the method of probability in Othello along with the concept of initial weights and dynamic weights assigned to all the squares on the board. Initial weights are set according to the chances of winning if a disc is placed there, for example, corner position is favored over edge position which in turn is favored over any other piece. The total weight is calculated by adding dynamic weights to initial weights. Dynamic weights are generated on the basis of total number of moves and number of discs it flips as initially number of discs flipped are not important but towards the end this becomes a critical factor.
Benefits of using this algorithm over other algorithms:

- It executes almost immediately
- It succeeds in approximately 880 games out of 1000 games against amateur players

The testing function was used to check success rate of our algorithms which helped us develop the implemented algorithm. It may also be used to check some other algorithms against ours.

Sample Input/ Output

Majority of the inputs are taken by mouse clicks. The next possible moves are highlighted when mouse hovers above those squares. On clicking, the board is updated and the turn is switched. The menus contain buttons, checkboxes and tabs which can be accessed using mouse clicks. The menus also contain textboxes which can be given input by keyboard. The output is shown on the GUI itself by menus and frames. The testing can be run in the interaction space.

Limitations and Bugs

- The single player is not very effective against good players because it doesn't take into account various other factors like mobility and stability of discs.
- There might be some other limitations that we are not aware of.

Acknowledgement

- Our Professor: Prof Amitabha A. Sanyal
- http://www.ultraboardgames.com/othello/index.php for rules
- Racket Documentation