ReadMe file for MaxConnect4

This file will assist you in understanding the MaxConnect4 program written by:
1) Vignesh Karunanithi (Student ID: 800888187)
2) Mahesh Kurva (Student ID: 800889008)
3) Vijay Kumar Reddy Sama (Student ID: 800867663)

Running the program:

This program has been written using Java - Swing

To Compile:

1) Open command prompt, navigate to the folder containing the AI.java, Cell.java, GameBoard.java, MainMenu.java and MaxConnect4.java files and type "javac MaxConnect4.java"
2) Class files will be generated

To Run:

1) Without closing the command prompt and without changing the directory, type "java MaxConnect4"
2) The steps and the final result is written to a text file named “Report.txt”

Program Design:

The program has five class files:

1) MaxConnect4.java - This class has the main function and it is responsible for initializing the main menu.

2) MainMenu.java - This class is responsible for displaying the main menu.

a) It has **actionPerformed()** method to handle the button click event
a) It has **Start()** method which is responsible for initializing the game board.

3) **GameBoard.java** - This class is responsible for displaying the game board and for handling the core game functionalities. It has the following methods:

a) **isColumnAvailable()** - returns the availability of a column
b) **getFirstAvailableRow()** - returns the first available row in a column
c) **getGameBoard()** - returns the game board
d) **getPieceCount()** - returns the total number of pieces currently placed in the game board
e) **Score()** - Calculates the score
f) **isValidPlay()** - verifies if a move is valid
g) **playPiece()** - places a piece in a column
h) **mark()** - places a piece in a column temporarily
i) **removePiece()** - removes the temporarily placed piece
j) **isFull()** - checks if the game board is full
k) **getTurn()** - returns current turn
l) **getuserColor()** - returns user's color
m) **setTurn()** - sets turn
n) **setUserColor()** - sets user's color
o) **actionPerformed()** - handles button click event
p) **Play()** - Handles the user input
q) **MakeAIPlay()** - Performs the AI move
r) **Report()** - Writes steps and result to the output file

4) **Cell.java** - This class is responsible for defining the structure of each cell in the game board grid. It has the following methods:

a) **getToken()** - returns the token (data of a cell)
b) **setToken()** - sets the token value
c) **Display()** - displays the game board
d) **paintComponent()** - paints the game board

5) **AI.java** - This class represents the AI player. It has the following methods:

a) **findBestPlay()** - finds the best possible move for AI player by implementing the minimax algorithm with alpha beta purning
b) **Max()** - finds the best possible score for the maximizing player
c) **Min()** - finds the best possible score for the minimizing player

Reasons supporting this approach:

The project needs a custom main menu, a game board, an ai player and each of these should be connected to each other. This makes it an ideal candidate for an object-oriented approach.

Also, since the game is a two player, turn-based game where each player attempts to minimize the possible loss for a worst case (maximum loss), we have proceeded with the minimax algorithm.

In order to cut out unnecessary computations, we have also implemented the alpha beta pruning to the mimimax algorithm.

Data Structure design:

The program uses arrays predominantly. There are 5 classes used in the program as explained above. A 2D array made up of Cell class objects acts as the game board. Normal integer arrays are used for score calculations and other purposes. Other than this, normal String, boolean and int variables are used.

Since the program uses Swing, we have also used JFrames, JPanels, JLabels, JRadioButtons and JButtons.

Each move as well as the final result is written to a text file named “Report.txt”.

Assumptions:

The program is written based on the following assumption:

1) While calculating scores, if a stretch of 5 same colored pieces are found (horizontally, vertically or diagonally), then the score is incremented by 2 as it has 2 sets of quadruples within it.
2) A stretch of 6 or 7 same colored pieces are treated in the same way, and hence they will yield a score of 3 and 4 respectively.
3) There is no "Try Again" or "Restart" option and hence the application has to be closed and ran again inorder to play again.
4) The depth of the minimax algorithm which inturn determines the AI diffculty level is set to 5 and can be either increased or decreased by modifying directly in the code.

Complier Used:

javac (jdk 1.7.0_71)

Platform Used:

Windows 8.1 - The code was actually written in Eclipse and can be ran from command prompt as well.

Screenshots:

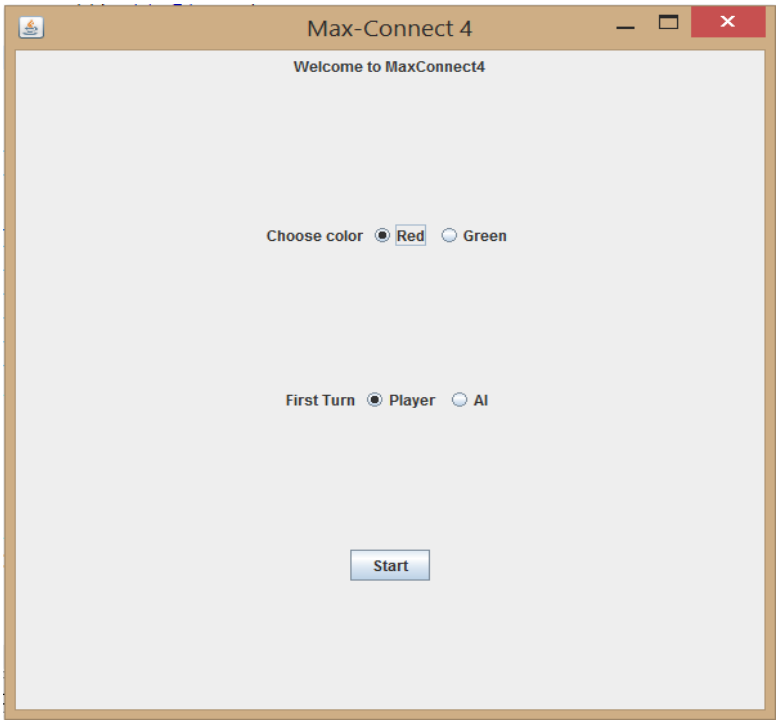

Fig 1: Main Menu

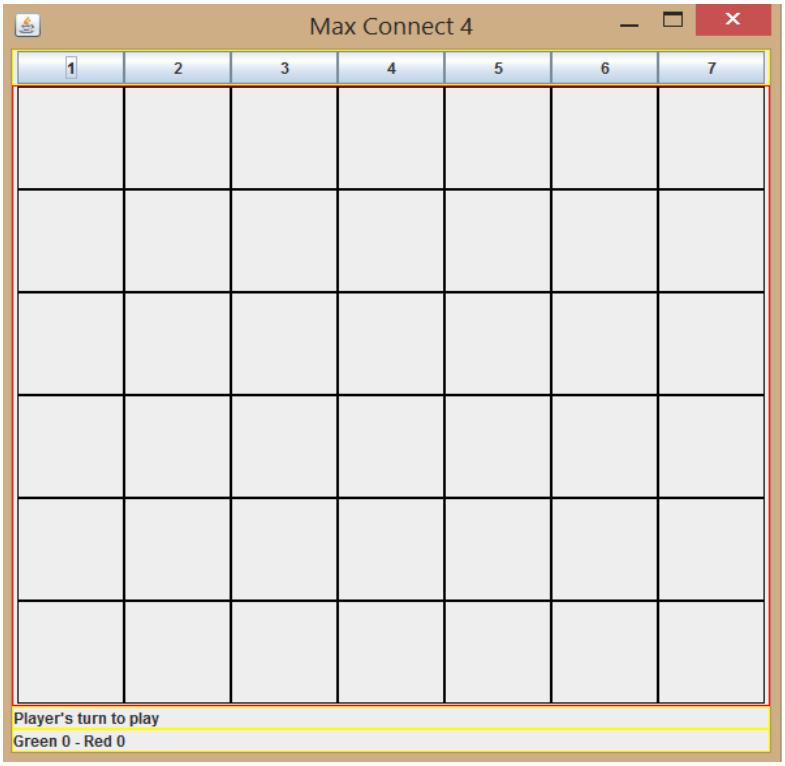

Fig 2: Empty game board, where user is to play first. The status of the game is displayed towards the bottom of the window above the score area.

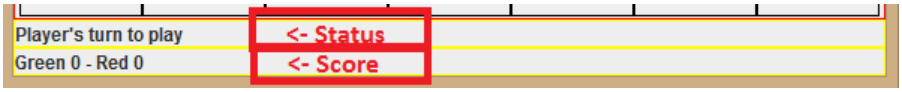

Fig 3: Status of the game and score are displayed towards the bottom of the window.

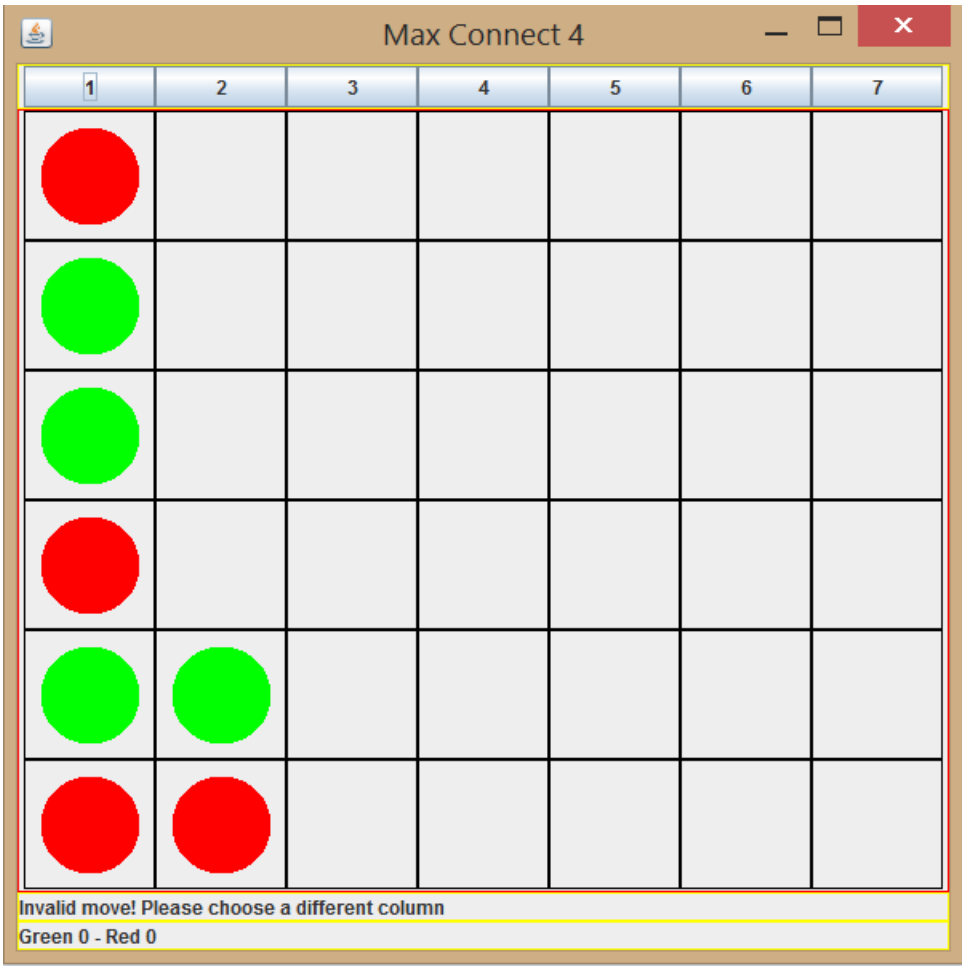

Fig 4: Invalid move by the user is displayed in the status area

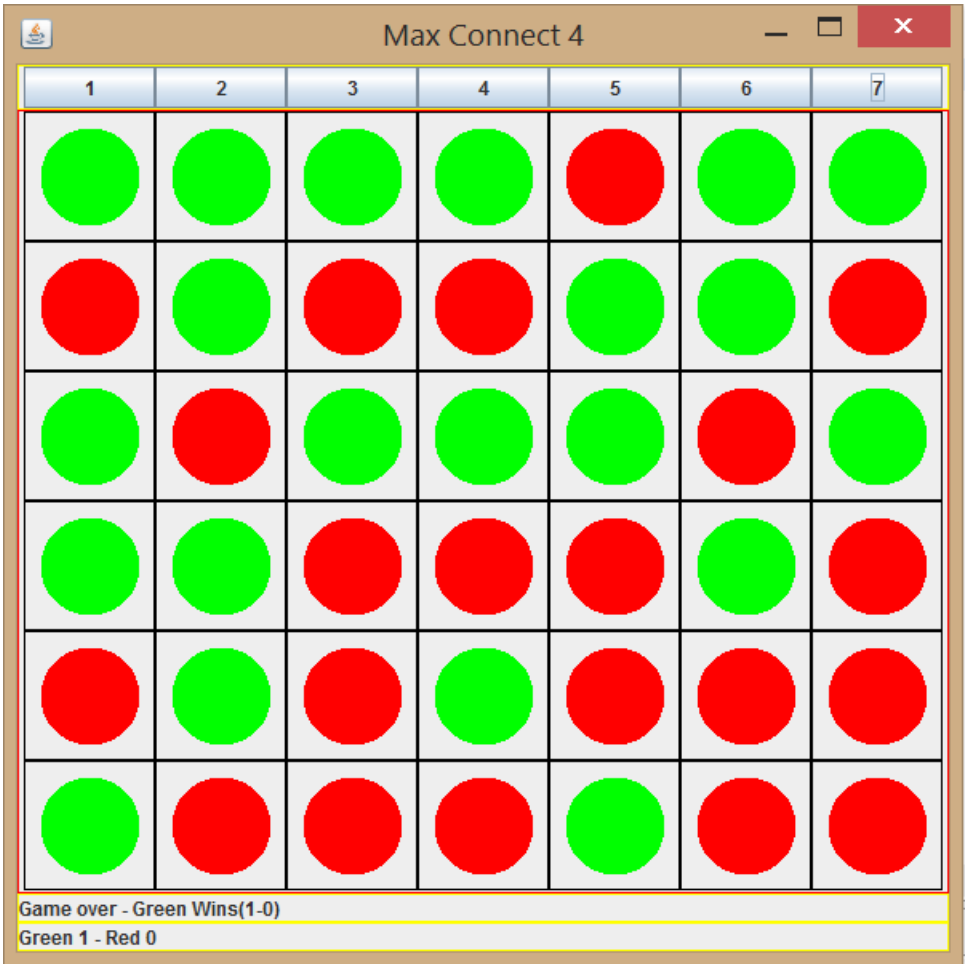

Fig 5: Result is displayed in the status area

References:

We have referred to the following resources for implementing the minimax algorithm with alpha-beta pruning logic

1) https://github.com/mpatt3970/Connect_4/tree/master/src/mines/mipatter
2) http://www.brian-borowski.com/software/connectfour/