STEVENS INSTITUTE OF TECHNOLOGY

Computer Science (Master's)

CS 501 - Introduction To Java Programming

Connect 4 Game Simulation

Submission to:
Professor Peter Jurkat

Submission By:
Chetan Goyal
(Group name: Chetan - CS 501)
CWID: 20005334

Date: December 13, 2021

Playing The Game

- The player is informed about the game and the board is displayed, and the option of exiting the ongoing game is provided(by pressing -1 in ongoing game0
- Always the red player makes the first move
- The red player enters the column number (1-7) to drop the disk in the column of their choice
- Then the Yellow player is asked to make the next move and so on alternate turns
- The game continues similarly until one of the players brings four of his disks together either vertically, horizontally or diagonally
- When the player does the above, they win
- If all spaces for the disk are filled and nobody wins, the game is tied
- At the end of the game, the players are asked if they wish to play again or not(by pressing 1 or zero respectively)

Project Analysis

Chapter 8 - Exercise 8.20

About the game: *(Game: connect four)* Connect four is a two-player board game in which the players alternately drop colored disks into a seven-column, six-row vertically suspended grid, as shown below.

Objective: To connect four same-colored disks in a row, a column, or a diagonal before your apartment can do likewise. The program prompts two players to drop a red or yellow disk alternately. Whenever a disk is dropped, the program redisplays the board on the console and determines the status of the game (win, draw, or continue).

Files: ***BackendForC4Connect.java*** file contains all the methods for the game, and the ***C8E20Connect4Main.java*** is the main file that runs the game, ***Input.txt*** contains inputs for game to test different scenarios in the game

Constructor: *BackendForC4Connect* - It initializes the scanner inp, board as a new hashMap, totalMoves to 0, redTurn to true as red will always play first, totalMovesInEachColumn as another hashMap, and gameWon to false

Methods used: createBoard, displayBoard, playGame, checkIfGameWon, checkIfTied, checkDiagonal2, checkDiagonal1, checkHorizontal, checkVertical, initializeTotalMovesInEachColumn

Boundary conditions:

- **I**f the column is filled then it was made sure that the same is informed, and the user is asked to provide a different column
- If the board is completely filled, the game is tied and same is handled
- Check of horizontal, vertical and diagonals are having many boundary conditions, all of which are handled

Implementation: The code has been written in Java and no external libraries have been used.

A method **createBoard** has been used to create our Connect Four Board for the game. A board of seven columns and six rows is created using a **HashMap**. The column whose values are integers, act as keys which range from 1 to 7, inclusively.

Against these keys(columns), six rows have been put as an **ArrayList** for each column, which denotes the row for each column. All these values have been initialized to 0 denoting that the board is empty.

To note:- The value of 0 has been used to recognize empty places in the board, 1 to recognize Red and -1 to recognize Yellow.

A method **displayBoard** has been used, which at every turn, as well as before the start of the game, displays the updated board on the console by checking the updated values for board. If there are 1’s and -1’s in the board, it accordingly displays the Red as R and the Yellow as Y.

As per the required objective, to win the game the player (red or yellow) needs to bring together four of their respective colors together - which can be done in three ways -

vertically, horizontally, or diagonally.

Now it is needed to check at every turn, if the game is won or not, which is done by four checks after a player makes their move.

Vertical check - Using method **checkVertical,** we take in the updated status of the board, and capture the column that was last entered and the player that entered it, and then we go up and down that column using a for loop and perform the required check. If the four of the same color are found, our method returns true and the game is declared as won by that player, else it returns false.

Horizontal check - We use method **checkHorizontal** to implement the above, and here we capture the column that was last entered, and capture the row that was last used and then alongside that row, we check columns in the left and the right for finding consecutive color of that player. If found that the total count is 4, we return true else we return false, and the game continues

Diagonal check - Since there are two diagonals, we perform a check for each one using two methods - **checkDiagonal1** and **checkDiagonal2**.

The checkDiagonal1 checks from the upper left to the lower right. It captures the column and row that was entered by the last player and then performs a check by increasing the row but decreasing the column value for upper left and increasing the column value but decreasing the row value for lower right, to check for four consecutive last player moves.

The checkDiagonal2 checks from the lower left to the upper right. It captures the column and row that was entered by the last player and then performs a check by increasing the row the column value for upper right but decreasing the column value and the row value for lower left, to check for four consecutive last player moves.
If the count for either is 4 it returns true, else it returns false false and the game continues.

An option to exit the ongoing game is provided in the start of the program wherein if a player enters -1, the current game is exited.

After the player makes a move, it is also needed to check if the board is completely filled, that is all the 42 elements are filled. If that is the case, the method **checkIfTied is run.** An integer count is maintained through totalMoves. If the board is full and nobody wins:- checkIfTied returns false and a Tie is declared.

A HashMap called **totalMovesInEachColumn** is also maintained which keeps mapping the columns with the number of elements filled in it. If the column is filled, this mapping helps to check that and ask the player to re enter a different column. We use the initializeTotalMovesInEachColumn method to initialize the number of elements in each column to 0.

After the game is done, the players are asked if they wish to start a new game, and as per their input (1 for yes, 0 for no), the game is either re-started or the program is finished.