CSC120 2025S Project - Writing a Connect-Four Program

1 An Overview

The project aims to develop a program for the game Connect Four. Connect Four is a game like Tic-Tac-Toe with two players, with the following features:

- The game is played on a board with seven columns and six rows.
- The players take turns and insert their pieces in a column. A column can accept a piece if it has less than six pieces.
- In each column, a piece is inserted flush to the bottom.
- A player must form a line with four pieces to win the game. A line can be horizontal, vertical, diagonal, and anti-diagonal.
- The pieces have two colors, and each player has her assigned color.

Here is an example of a board configuration. `'X'` represents Player 1, and `'O'` represents Player 2.

row positions								
	5							O
	4							O
	3				O			X
row positions	2	O		O	X			X
	1	X	O	O	X			O
	0	O	X	X	X	X	O	X
		0	1	2	3	4	5	6

column positions

The following array elements show a situation in which Player 1 has won.

We will call the class to be developed `Connect`. You are given a template `StaticBoard` for the class in which some key methods are fully developed.

2 Class `Connect`

You must write the program `Connect` using the aforementioned static methods (not required to use the static constants). *Deviating from the instruction will result in point reduction, since you can use a chat-AI to generate a code.* You can rename the template class `StaticBoard` to `Connect` and the required methods.

The general requirement for the code you add is as follows:

(I) The program announces the start of the game and initializes the board.

(II) Then, as long as there is no winner, the program does the following:

  (a) The program presents the board.
  (b) The program obtains the column for Player 1 to place her piece.
  (c) The program checks whether or not Player 1 has won. If Player 1 has not won, the program advances to the next step.
  (d) The program presents the board.
  (e) The program obtains the column for Player 2 to place her piece.
  (f) The program checks whether or not Player 2 has won. If Player 2 has not won and the board is open, the program returns to the start of the loop.

(III) The program presents the final configuration of the board and announces the outcome.

To accomplish the functions above, the following methods will be needed:

(i) `static void present( String[] board )`: **This is a method for presenting the board. You must write the code for this method.**

(ii) `static boolean isWinner( String[] board, int player )`: **This is a method for testing if `player` has won the game. You must write the code for this method.**

(iii) `public static void main( String[] args )`: **This is the main method. You must write the code.**

3 The Template: `StaticBoard`

In the template class, all the constants and methods are static. You can copy them into your code or rename it as `Connect`, add new methods, and rewrite its main method. This template's main method is for testing the functionality. By running it, you can understand how the methods work.

In `StaticBoad`, a board is represented by a `String` array with seven elements. The elements of the array are the columns, and in each `String` element, its `char` comments represent the individual pieces. The length of each `String` is the number of pieces the corresponding column has. We use `'1'` and `'2'` to denote the pieces for Players 1 and 2. The game configuration in the previous section is thus represented as a seven-element array

{ `"212", "12", "122", "1112", "1", "2", "121122"` }.

Here, we read the characters from bottom to top, column-wise. We also use the `int` value of 0 to represent an open spot on the board. In the spring representation, the 0s representing open slots are omitted. This omission makes it easy to update the board by inserting a piece.

The template `StaticBoard` has the following constants:

- `public static final int COL_SIZE`:

  This is a constant for the column size, i.e., 7. The template uses this constant instead of using the literal 7.

- `public static final int ROW_SIZE`:

  This is a constant for the row size, i.e., 6. The template uses this constant instead of using the literal 6.

- `public static final String[] PIECE`:

  This is a three-element array, whose elements are `" "`, `"X"`, and `"O"` in this order. In other words, `PIECE[0]` gives a single space, `PIECE[1]` gives `"X"`, and `PIECE[2]` gives `"O"`.

- `public static final String[] COLOR_PIECE`:

  This is a color-added version of the array PIECE.

The template also has the following methods for inquiring about the validity of column and row indices and status of the board:

- `public static boolean isValidPosition( int col )`:

  This method receives an `int` parameter `col`. Then, the method returns `true` if the value of `col` is between 0 and 6, i.e., it is a valid column index; the method returns `false` otherwise.

- `public static boolean isValidPosition( int col, int row )`:

  This method is a two-parameter version of the previous method. It receives two `int` parameters `col` and `row`. Then, the method returns `true` if the values of `col` and `row` are between 0 and 6 and 0 and 5, respectively, i.e., the combination is a valid column-row position; the method returns `false` otherwise.

  Note that by *method overloading* this method has the same name as the previous one.

- `static int nextRow( String[] board, int col )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and an `int` parameter `col` and returns the row position the next piece in the column will settle in. If the column index is invalid or the column is full, the method returns `-1`.

- `static boolean isOpen( String[] board, int col )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and an `int` parameter `col`, returns a `boolean` indicating if the column `col` can accept a piece.

- `static boolean isOpen( String[] board )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and returns a `boolean` indicating if some column can accept a piece.

  Note that by *method overloading* this method has the same name as the previous one.

In addition, the template has the following methods for accessing an individual piece, adding a new piece, and removing a piece:

- `public static int getPiece( String[] board, int col, int row )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and two `int` values, `col` and `row`, which are supposed to represent a column-row position, and returns the player number of the piece at the position (1 or 2). The method returns `0` if the position is open. The method returns `-1` if the position is invalid.

- `public static void add( String[] board, int col, int player )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and two `int` values, `col` and `player`, which are supposed to represent a column index and a player number, respectively. Then, the method places the player's piece in the column.

- `public static void remove( String[] board, int col )`:

  This method receives a `String[]` parameter `board`, which is supposed to represent a board, and an `int` value, `col`, which is supposed to represent a column index and a player number, respectively. Then, the method removes the top piece in the column.

  This method is not used in this version. You may use it for the advanced task described later.

In addition, the template has the following methods for determining a valid column number through interactions with the user and initializing the board:

- `public static int inputCol( String[] board )`: This method interacts with the user to receive the index to an open column. The interaction is repeated until the user enters a valid open column. Then, the column index is returned. Like the previous methods, `board` represents the board.

  The method in itself instantiates `new Scanner( System.in )`, stores it in a local variable `keyboard`, and uses it to receive input from the user.

- `public static String[] initialize()`: The method returns the board in the initial form. The main method can declare and initialize the board by:

```
String[] board = initialize();
```

4 Part1: Writing a Simple Version (12 points)

In this part, you must complete `Connect` so the program receives column selections for both players.

- [**3 points**] Correctly write the method `present`.
- [**4 points**] Correctly write the method `isWinner`.
- [**5 points**] Correctly write the method `main`.

5 Part2: Writing an Advanced Version (9 points)

You must write an advanced version of `Connect`, called `ConnectNew`. In the advanced version, the program acts as Player 2.

- [**5 points**] The program selects a column for Player 2. The points are determined based on the sophistication of the selection mechanism.
  - [**3 out of 5**] The selected column is not concerned with winning or losing. For example, the program selects an arbitrary available column (for example, a random one).
  - [**5 out of 5**] The program selects a winning column if there is one. A winning column is one such that if Player 2 selects the column, Player 2 immediately wins the game.
  - [**6 out of 5**] **That is, one extra point.** If there is no winning column and there is a column Player 2 should play/avoid to prevent an immediate loss, select a column accordingly.
- [**2 point**] Provide detailed comments in your code.
- [**2 point**] Submit a video recording demonstrating that the program runs correctly. of the program.