In-class exercises (lecture 11)

Exercise 1: Review the Cookie Counter example

In lecture-code > Evening > Lecture 9 > cookie counter:

- Is it modular? Spoiler alert: no.
- Does it allow for easy extension e.g. more types of cookie?
- Is there functionality that would be useful in other situations?
  - E.g. broader nutrition calculators?

How might you make it more modular?

Exercise 2: MV* tic-tac-toe game

Design an implementation of tic-tac-toe that follows MV* architecture. If you're not familiar with tic-tac-toe, you can **play it online (https://www.google.com/search?q=tic+tac+toe)** and/or **read the rules (https://en.wikipedia.org/wiki/Tic-tac-toe)**.

Your implementation should be for two players interacting with the game in the IntelliJ terminal. Use Scanner to prompt the players to take their turn and get their moves.

Code for drawing the board in the IntelliJ terminal is available in the lecture-code repo on GitHub > Evening_section > Lecture 11 > tictactoe package > view package. The Board class has a single static method, createBoard, which takes a 2D String array representing the pieces on the board, and returns a String representation of the board state that can be printed to the command line. You can also find sample code for populating an empty board array and adding pieces in main(). You'll probably want to move this logic elsewhere!

The rest of the implementation is up to you. Among other things, you'll need a way to keep track of the board state (which player's pieces are in which cells, whose turn it is, whether or not there's a winner etc.) and a way to allow players to take turns. Feel free to tweak the details—this is an exercise not an assignment :) Key questions to consider:

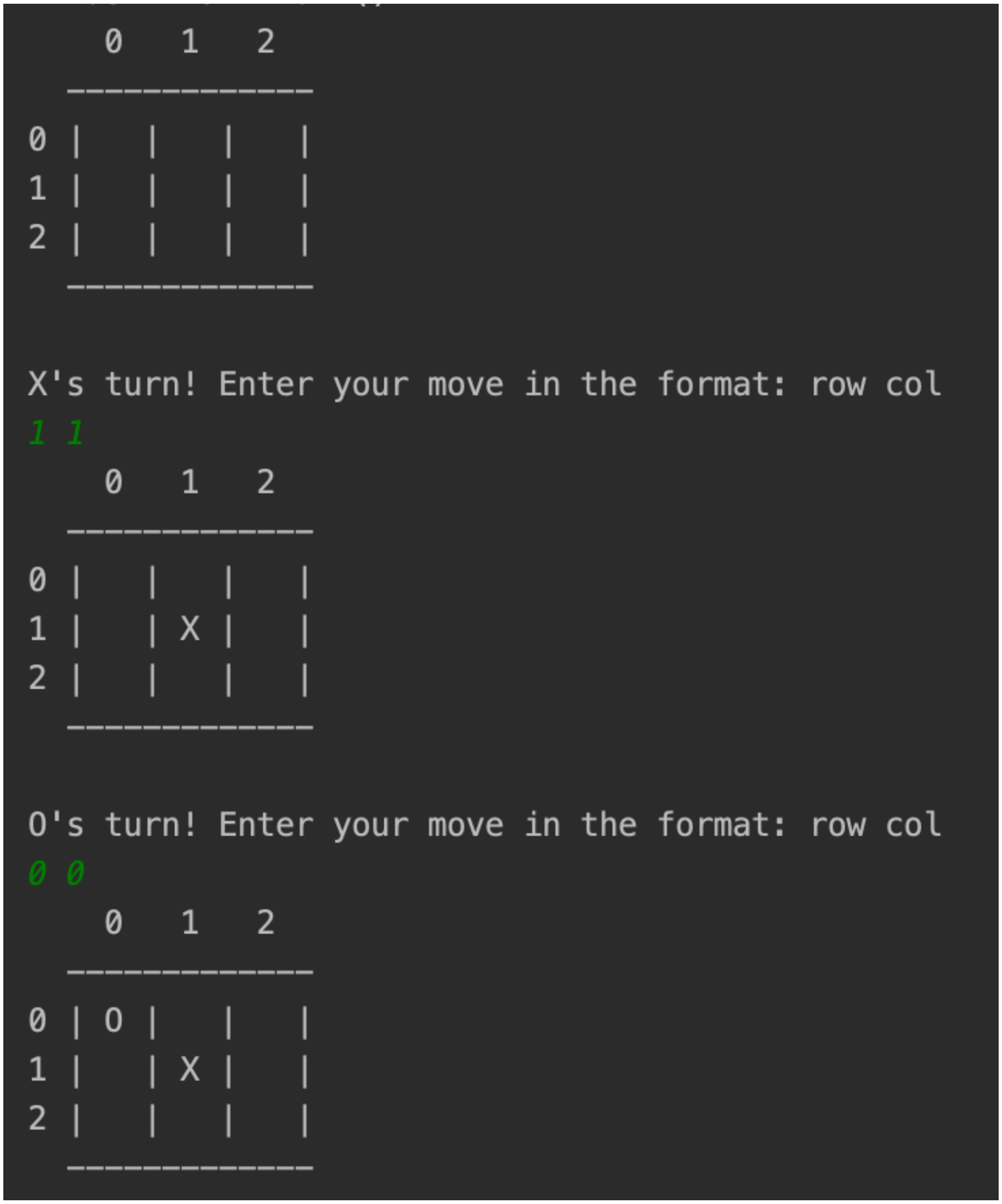
- How will you divide the functionality between model, view, and controller?
- Will the controller be separate from the view or integrated with it?
  - Hint: With command line input, at least some of the control will be stuck in main()

You will not be able to finish an actual implementation in the allotted time but you should be able to come up with a comprehensive UML.

I'll go over one possible implementation after you've had ~20 mins to work on this. A sample solution will also be provided after class.

An example interaction

- To start the game, an empty board is displayed.
- X goes first, choosing to place a piece in the center square [1][1].
- O goes next, choosing to place a piece in the top left square, [0][0].
- The updated board is displayed after each player's turn.



```
    0   1   2
  -------------
0 |   |   |   |
1 |   |   |   |
2 |   |   |   |
  -------------

X's turn! Enter your move in the format: row col
1 1
    0   1   2
  -------------
0 |   |   |   |
1 |   | X |   |
2 |   |   |   |
  -------------

O's turn! Enter your move in the format: row col
0 0
    0   1   2
  -------------
0 | O |   |   |
1 |   | X |   |
2 |   |   |   |
  -------------
```