CS 251 Intermediate Programming
Lab 2: Gomoku

Brooke Chenoweth

Fall 2017

Please note: This is an individual assignment. It is designed for you to get back even more into programming mode. The idea is that you will have to use all of the skills that you learned in cs152 (or its equivalent) and get further acquainted with Java, while solving some interesting problems.

This assignment will give you some experience implementing an interface, using enums, and interacting with my GUI.

Problem Specification

We are going to implement the game of Gomoku, also known as five-in-a-row.

The game is very similar to tic-tac-toe, only that you are playing on a larger board (usually consisting of a squared paper). The goal is to get five-in-a-row. You are allowed to place your marker anywhere there's an empty square on the board. The two players (X and O) take turns to place their markers on an empty square on the board, and may try to play offensively or defensively. The person with the best strategy and sharpest eyes usually wins the game.

What you're getting

I've prepared the visualization for you in a class called `GomokuGUI`. This class has a static method that is passed in an object that has functionality specified as in the "class" `GomokuModel`. The methods in this class that's really an *interface* (we'll talk about that in class), are the following:

- `int getNumRows()`
  `int getNumCols()`
  `int getNumInLineForWin()`
  These methods specify the size of the game board and the number of squares in a line needed for a win. To play the default game, just have these methods return the default constants defined in `GomokuModel`. When testing your code, you may find it easier to experiment with a smaller board. In that case, you can change the values returned by these methods and the GUI should be able to adapt.

- `Outcome handleHumanPlayAt(int row, int col);`
  This method is called by the GUI whenever a click occurs on the game board. It's supposed to update the state of the game board, and the return value is a value that describes the current state of the game after the newly entered click was applied.

- `void startNewGame();`
  This method is called by the GUI whenever a new game is begun. It should set up whatever bookkeeping is needed to start a fresh game.

- `String getBoardString();`
  This method returns a string representation of the board. The GUI calls this method whenever it needs to know what the board looks like in order to be able to redraw it. The string that is returned must match the specific format described in the interface documentation.

- `void setComputerPlayer(String opponent);`
  This method configures what sort of computer player (if any) will be used.

Please refer to the documentation for `GomokuModel.java` for more information.

What you have to do

To get started, create a file called `Gomoku.java`. It should be a class that *implements* the model interface. I'll show you in class how to make this happen. However, the main method should look something like this:

```
public static void main( String[] args ) {
    Gomoku game =  new Gomoku();
    if(args.length > 0) {
        game.setComputerPlayer(args[0]);
    }
    GomokuGUI.showGUI(game);
}
```

Three things happen in this `main` method:

1. Create a instance of the class you are writing.

2. Possibly configure the Gomoku object based on a command line argument. (Any command line arguments given to a Java program are available in the String array parameter of the main method.)

3. Pass the Gomoku object to a static method in the GomokuGUI class that will actually get the GUI started.

There are three parts to the assignment. Each part needs to be completed before the next part is attempted.

1. **Game play** - Make the game work to the point that when clicking the game board, every other click yields a ring and a cross. It should not be possible to place content on a field that has already been played. Basically, in order to do this, you'll have to write all the methods. However, for this part of the game, you are not required to create logic to figure out if someone wins - i.e., the `handleHumanPlayAt` method can always return `GAME_NOT_OVER` status flag.

2. **Win detection** - Everything from the minimum required part, but added in the logic to determine who won the game. Also, the player who won the last game should get to go first in the following round.

3. **Computer player** - Create a rudimentary computerized player. You'll do this by making the `handleHumanPlayAt` method also fill in a value for the player who didn't start. That will now be the computer player. For full credit, this player's strategy must be better than just random placement on the board, and the player should be able to win a game at least if playing against a somewhat stupid human.

   The computer player should only be enabled if it was configured via the `setComputerPlayer` method. (You may assume that this method will not be called in the middle of a game.) An argument of NONE should disable the computer player, while an argument of COMPUTER should enable it. If you want to explore additional computer player strategies, feel free to recognize additional Strings. If you do this, please be sure to document it so that the grader will know to try them.

Turning in your assignment

Once you are done with your assignment, use UNM Learn to turn in the `Gomoku.java` file that you have created. Note that each source code file should contain a header comment with your name, class, and other related information. Failure to properly comment your files may lead to point deductions.