HW#4d Game State

To complete this assignment you will implement the game state for your game and then write unit tests for it.

Specification

- [5%] Create a new Android project that has a GUI that displays a multi-line `EditText` that fills most of the display except for a single button labeled Run Test.
- [10%] Within the same package as the `MainActivity` class, create a new game state class for your game. (Eventually this class will be a subclass of the `GameState` class in the game framework but it doesn't have to be for this assignment.) Your game state class' instance variables should encompass all the information you will need about the current state of the game in order to display it properly for a human user or allow a computer player to make decisions. Things to consider:
  - Information about the resources each player has (e.g., cards, pawns, money)
  - Information about the state of any resources (e.g., card is visible, pawn is yellow)
  - Whose turn is it?
  - Detailed information about shared resources (e.g., the game board, contents of a draw pile)
  - The visibility of certain information from the perspective of each player
  - Current score of each player
  - Current state of a timer
  - Current state of the dice
  - What stage of the game you are at (e.g., setup phase, placement phase, buy phase, main play stage, etc.).
  - You may need to create additional classes to describe specific elements of the game state (e.g., a playing card, a pawn, a tile, etc.)
- [5%] Implement a constructor for your class that initializes all the variables to reflect the start of the game before any actions have been taken.
- [15%] Implement a copy constructor that makes a deep copy of a given instance of your class and adjusts it so it shows only the data visible to a specific player.
  - This must be a *deep copy*. If your game state contains any variables that are of a class type then the objects they refer to must also be copied. If any of these objects class-type variables then those must be copied too, and so on. If you don't understand this, please ask. Shallow copies will be penalized.
- [5%] Add a `toString()` method to the game state class the describes the state of the game as a string. This method should print the values of all the variables in your game state. If the variable is an array (or similar aggregate type) all its values must be printed. Your string should have sufficient formatting so its reasonable for you to tell values go with which variables. Be sure to put the @Override tag on your `toString()` method to verify it has the proper signature.
- [15%] Add a method for each of the actions you defined in actions.txt for the GUI design homework you did previously. Each method should have a `boolean` return value. When called, each method should:

    - verify the move is a legal move for the current game state. If it's not, return `false`.
    - modify the game state to reflect that a given player has taken that action. Then, return `true`.

  Each method should require that the player id of the player who is taking that action be passed in as the first parameter. Other parameters will likely be needed for some actions.
- [25%] Implement an `onClick()` listener method in your MainActivity that's connected to the Run Test button. When the user clicks this button the following things should occur in this order:
    - Any text currently displayed in in the multi-line `EditText` (probably from a previous test run) should be cleared.
    - A new instance of the game state class is created using the default constructor and assigned to a variable named `firstInstance`.
    - Use your other constructor to create a deep copy of firstInstance from the perspective of player one. Assign this copy to a variable named `secondInstance`.
    - Using `firstInstance`, call each method in the game state class at least once. In each case it should be making a legal move in the game. For each method call, a brief description of the action taken should be printed to the multi-line `EditText`. (e.g., "Player 1 has moved his pawn from position 10 to position 14." or "Player 3 has rolled the dice. She rolled a 9." New messages should be appended to previous ones, not overwrite them.
    - Create a new instance of the game state class using the default constructor. Assign this value to a variable named `thirdInstance`.
    - Use your deep copy constructor to make a deep copyof `thirdInstance` from the perspective of player one. Assign this copy to a variable named `fourthInstance`.
    - Call the `toString()` method on `secondInstance` and `fourthInstance`. The two strings should be identical. Your code should verify this. Also, print both strings to the multi-line `EditText` for visual inspection. Again, append these rather than overwrite previous messages.
- [20%] Any code you write for your project should follow the CS301 coding standard. Exception: You have the instructor's permission for the `onClick()` method to be longer than is normally allowed.

Important Note: Keep network play in mind when you create your game state class. Any references to GUI or Game Framework related objects is likely to render network play inoperable. Specifically:

- Do not use references to objects that are subclasses of `GamePlayer` (e.g., `TicTacToeHumanPlayer`). Use a player id instead.
- Avoid references to instances of any class in the `android.graphics` library (e.g., Path, Rect, Canvas, BitMap). Android color values are okay since they are just int values.
- Avoid references to instances of any class in the `android.view` library (e.g., Activity, Button, TextView, SurfaceView, etc.).
- If you use a class that's in the Android or Java libraries, make sure it has implemented the `Serializable` interface. (We'll talk about `Serializable` in lecture at some point in the future.)

Grading and Good Design
The instructor reserves the right to adjust your grade based upon the quality of your design. Think carefully about the overall completeness and efficiency of the choices you make. Obvious oversights will be penalized.

Turning in this Assignment
You are responsible for following these instructions correctly and turning in your homework assignments properly. Failure to follow these instructions will hurt your grade. It also will frustrate those who are grading the assignment.

1. As this is a team project assignment, only one person on each team should turn in the assignment.
2. Include the names of everyone on the team using @author tags at the top of each .java source code file.
3. Please be certain that your project has a unique name so that all the student projects can be loaded into the same workspace.
4. Create a new repository on github.com for your project. Check your project into this repository.
5. Create a text file (.txt extension) that contains the URL of your gibhub repository. Name the file so that it contains the UP userid of every person on your team. For example, if the members of the Beatles were on a team they might turn in a file named: `PennyLane_lennon19_mccart19_harrig20_starr20.txt`.
6. Submit your renamed .txt file via the associated “Turn In Here” link on the course web site.

Do not make any changes to the repository after your due date. If you do, the assignment will be graded as if it had been turned in at the time of the latest change.