Lab12 – Creating a Game based on a Map-based Graph

THIS IS A PROJECT NOT A LAB SO YOU'RE MOSTLY ON YOUR OWN. PLEASE WORK INDIVIDUALLY BUT FEEL FREE TO DISCUSS THINGS IN PAIRS. DO NOT EXPECT MUCH HELP FROM THE PROFESSOR OR TA

For this final lab/project, you'll implement your own role-playing computer game based on a map-based graph. You get to design the game layout yourself, but always remember to keep things simple especially early on. Part of this project is to learn how to build a graph structure using maps. You'll find the instructions presented in this write-up to be minimal giving you the freedom to envision and create your own game so feel free to capitalize on that … be creative and ambitious!

Preliminaries

Start by sketching on paper a design for the layout of a role-playing game with at least ten locations. Give each location a ***unique*** name/label and decide how the locations are connected as a directed graph. For this exercise, the starting location should be unreachable once the player leaves it; in graph terms, the starting location has edges going out but none coming in. Include at least one dead-end location that the player can get to, but can't leave, and maybe a pair of locations that only have edges out leading to each other. Otherwise, it should be possible to get from any location to any other location in some sequence of steps.

Decide on the properties that each location should have: a character, a color, a weapon, an enemy, a tool, etc. To keep things simple, you may limit each location to one instance of each property; i.e., a location can have one character/ color/ weapon/enemy/tool/etc. or none, but not several. If you want to implement multiple instances of a property at a location, you'll need to use a collection for that property. For each location, assign values for each property. Include the location label (name) as one of the properties, but, for now, don't include connections to other locations.

Part 1: File representation of the graph

Now, using the *sampleLibrary.BookCollection* data file from our previous lab as a model, create a text file for the locations and their properties. Each entry should have one line for each property, starting with the location label. If a location doesn't have a particular property, choose a special symbol to put on that line to indicate it doesn't have the property. In this way, each entry will have the same number of lines, so reading the file will be straightforward.

Create a second text file, using a similar format, for the connections among locations. Each entry should start with the location label on the first line, the number of (outgoing) connections on the second line and the labels of locations it's connected to on subsequent lines. Alternatively, you may include a pair of locations on each line to indicate a connection going from the first location to the second location.

Finally, create a *LocationDescription* java class, modeled on the *BookDescription* class that encapsulates the properties of a location. Include suitable constructors, accessor methods, mutator methods, and a *toString* method.

Part 2: The GameLayout class

Create a *GameLayout* class modeled on the *BookCollection* class. It should have the following two instance variables (along with others deemed necessary):

```
//associates a given location label with a set of connected locations
private HashMap <String, Set<String>> connections;

//associates a given location with its description object
private HashMap <String, LocationDescription> descriptions;
```

Your *GameLayout* class should have methods that enable you to:

1. create a new *GameLayout* by reading connections and information from specified files formatted as described above
2. iterate over all locations; the method should return an *Iterator* object that yields location names, one-by-one
3. iterate over all connections for a given location; the method should return an *Iterator* object that yields location names, one-by-one, for the given location
4. get a particular location description for a given location name/label

5. update the description of a location by adding/removing/updating a property
6. write the current state of the *GameLayout* to a specified file

Test that your *GameLayout* class contains and preserves correct information by reading it from files, writing it to different files and comparing the results.

Part 3: The driver class

Create a class that simulates the play of the game. It should begin with the user at the starting location. At any time the user ***should*** be able to:

1. list the names of all possible locations without their connections or descriptions
2. list the properties at the current location (including its label/name)
3. list all locations connected to the current location
4. move to one of the adjacent locations through a connection
5. search (described below in ***Part 4***)

Part 4: Searching for a needle in a haystack

Choose one or more of your location properties (such as weapon or enemy) and implement a search method in the *GameLayout* class for a specified value of that property. Search should proceed from the current location and follow connections, using the following breadth-first search algorithm discussed in class:

```
algorithm breadth-first search (location, desired property)
  create a new empty queue
  enqueue location
  mark location
  while the queue is not empty do
    dequeue current location
    if desired property is at current location then
      unmark all locations      // to allow for future searches
      return current location   //i.e. return first match
    end if
    for each unmarked neighbor of current location do
      mark neighbor
      enqueue neighbor
    end for
  end while
  unmark all locations   // to allow for future searches
  return null       //i.e. no matches found
end breadth-first search
```

To mark locations, add a *boolean mark* instance variable to your *LocationDescription* class and make sure it is initialized to *false*. Then include *mark* and *unmark* methods for the class.

Add a search command to your driver class that gets a property value from the user and then searches for that property value starting from the current location. Report the location where the property value was found, if it was found; otherwise report that it was not found. Test your search method on several property values starting from several locations.

Part 5: Make the game more realistic

Create a new interactive driver program that simulates the play of the game. The game interface can be text-based OR GUI-based (preferred). **Developing your game as an Android App (and demoing it by the due date) will earn you 10 points on the final exam; however, first you need to demo the android exercise and get my preapproval.**

The player of any interesting computer game typically has a mission to complete otherwise the game would be very boring. In your case, the specifics are left for you to design and implement as an exercise. Please be creative and ambitious … this is YOUR game! Here are some suggestions for making your game more realistic:

- A mission maybe something like arriving at a destination location, finding a location with a certain property value, or finding ALL locations with a certain property value.
- You may have more than one mission.
- Missions may be randomly generated; for example, the destination location may be randomly generated each time the game is played. A variant of this might be to randomly generate the property value and/or number of locations with the desired property to be found.
- Incorporate the concept of a player's life/energy as well as cost associated with visiting a location; for example a player may have an original life/energy value which gets reduced every time s/he visits a new location.
- Reduction value for visiting a location may be randomly generated. This might simulate a fight in real games.
- Adding randomly popping "life/energy boosts" at various locations would be another idea. The game ends when no life/energy remains, all missions are complete or one arrives at a dead-end.

Part 6: Demoing your game

You will be asked to demo your finished game during lab. Demos should last for about 10 minutes during which you are expected to show all the features in your game. Pay special attention to explaining what you did for part 5 to make your game more realistic as this will be allocated a significant portion of your grade for this lab. In addition, **you will submit to your lab instructor a one-page printed report which includes your name and summarizes the features that you added for part 5**. You don't need to write an essay; a list of bullet points should do but please remember that it is your responsibility to bring to my attention what you did to make your game more realistic in order for me to give you a fair grade. PLEASE DO NOT EXPECT ME TO FIND THEM ON MY OWN!

!!!HAVE LOTS OF FUN!!!