Project – Creating a Game using a Map-based Graph

For the final project, you'll implement your own role-playing computer game, **or another application of your own choosing provided it meets expectations**, 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 be creative so feel free to capitalize on that. Be realistically ambitious!

The project is due by **11:59pm on Monday, December 09**. You will demonstrate your project during class **on Tuesday, December 10**. There will be no extensions.

This is **a project, not a lab, so you're on your own**.

You may discuss general questions of design and implementation with other students, but do not share code or allow other students to see your code. If you are stuck, you may also ask the professor or TA for advice, but you must work out the design and implementation details yourself.

Preliminaries

Working with a partner, 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 initially, 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. For each location, assign values for each property. Include the location label (name) as one of the properties.

***Important note about your design*: Please do not include components that may be deemed offensive to others, such as components that show racism, sexism, or excessive violence.**

Part 0: Proposal due 11:59pm on Tuesday, Nov 19 (partners make separate submissions)

Submit a 1 to 2-page proposal to address the following

1. Names of people working on the project
2. Explain your game in high-level details
3. Explain why your game **requires the use of directed graphs**
4. Explain why **breadth-first** (**or depth-first**) **graph traversal is integral** for the objective of the game

Part 1: File representation of the graph

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 that encapsulates the properties of a location. Include suitable constructors, accessor methods, mutator methods, and *equals* as well as *toString* methods.

Part 2: The GameLayout class

Create a ***GameLayout*** class with the following two instance variables (along with others you deem 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
```

Note that you are using Java classes, not zhstructures classes in this project.

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; this method could return an *Iterator* object that yields location names, one-by-one
3. iterate over all connections for a given location; this method could return an *Iterator* object that yields connected 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/save the current state of the ***GameLayout*** to a specified file so that you may start a game from a saved point

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. 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***)
6. write/save the current state of the game to a specified file so that you may start a game from a saved point

Part 4: Searching for a needle in a haystack

Choose one or more of your location properties 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 (**or depth-first if more applicable to your game**):

```
algorithm breadth-first search (location, desired property)
     create a new empty queue of locations
     enqueue location
     mark start location as seen
     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 the 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, unmark* and *getMark* 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 should be GUI-based using Swing (the *javax* package) or newer versions. Here is a good resource in case you need one https://www.guru99.com/java-swing-gui.html

The player of any interesting computer game typically has a mission to complete, otherwise, the game would be very boring. In this case, it's your job to design and implement the game setup and look-and-feel. 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.
- A player may have to complete more than one mission to win the game or may have a choice of missions or game levels.
- The program may generate missions randomly; for example, the program might select the destination location randomly each time a new game begins, or it might randomly select the property or properties the player needs to find to accomplish the mission.

- You might 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 that drops every time they visit a new location, encounter an adversary, or rescue a kitten from a tree.
- The life/energy reduction value for visiting a location may vary from game to game or even from visit to visit. You could use this approach to simulate a fight or other kind of struggle.
- There might also be randomly occurring life/energy boosts.
- The game ends when all missions are complete (win), no life/energy remains (death), or the player arrives at a dead-end.
- If you want to try a multi-player game, save this feature for last, and don't try to do a network version. Just have the players take turns at the same computer.

Part 6: Demoing your game and submitting your work

You will be asked to demo your finished game on **Tuesday, December 10**.

Demos shouldn't last more than ten minutes during which you are expected to show all the features in your game. Pay special attention to explaining what you did for ***Parts 4*** and ***5*** to make your game more realistic as this will be allocated a significant portion of your grade for this project.

Part 7: Submitting your work --- due 11:59pm on Monday, December 09

Your submission **folder (partners make separate submissions)** should

1. Be self-contained; I should be able to unzip and run your game without the need to download additional libraries and classes
2. Include a reflection report to
    a. Explain why your game requires the use of graphs. Why is breadth-first (or depth-first) graph traversal integral for the objective of the game?
    b. Explain what each member did and allocate percentages accordingly (e.g., you did 40% and your partner did 60%).
    c. Summarize what you did for ***Parts 4*** and ***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 draw attention to what makes your game distinctive from a programming point of view. *Don't expect the instructor to figure this out from your code!*
    d. Describe the limitations of your game as it currently stands.

!!!HAVE LOTS OF FUN!!!