12 Finding Your Way.

Introduction

The goal of this assignment is to introduce you to the design of larger programs using the Model-View-Controller design pattern. The model for your program is designed to find a path between two nodes of a graph. The view can be display of the path in the Canvas or a description of the path in a text form in the console, or even an animation of the path search process. We give you fairly detailed specifications for the model part of the program. The requirements for the view are minimal. They only describe the basic functionality — you are free to enhance your presentation as you wish — and will be given credit for the work.

One part of the credit for this assignment will be given for a design document that describes the data, the organization of the program, the key program components, and the design of tests. Imagine you want someone to keep improving your program — provide a road map that explains what your program does and how does it do it. This document should complement the Javadoc generated web pages. A separate document will give you a more detailed guidelines for what we expect.

One part of the credit for this assignment is for the model part.

You will also get credit is for the user interactions (view) — grading both the design of the user views and the design of the program that drives it. A small bonus may be earned for exceptionally well designed display or interactions. It is better if the user interaction is done only through the console, but is well designed and documented, than if a fancy GUI display is driven by a code that another programmer cannot understand and maintain.

12.1 Project Presentation

You will present your project (both partners together) during the lab on Tuesday, April 15, or during the regular lecture time on Wednesday, April 16th. Each partner should be able to describe any part of the code in the project, regardless of *who wrote it*, as we expect that both partners work on the project together. More information about the presentations will be provided shortly.

The Model

Graph

Your program needs to represent a graph with nodes that represent capitals of the 48 US states. Each node has a name — the name of the state. For each node, record the information about the capital of that state. Each edge represents a bi-directional connection between two adjacent states. You may consider the *four corner states: Colorado Utah, Arizona and New Mexico* as connected to each other. Each edge has a value that represents the distance between the capitals of the two states. The distances between two cities are based on the geographic distance. (See a separate announcement for a shortcut you can use to compute this distance.)

Algorithms

Your model should implement three graph traversal algorithms:

- Depth-First Search: uses a *Stack* to record the *ToDo* information
- Breadth-First Search: uses a *Queue* to record the *ToDo* information
- Shortest Path Search: uses a *Priority Queue* to record the *ToDo* information

To implement the shortest path you need to represent a priority queue.

The detailed description of the algorithm appears in a separate document. You will encounter a significant penalty for repeating the code - one algorithm implementation should run all three variants, distinguishing between them by selecting the appropriate implementation of a common interface for dealing with the *ToDo* information.

Using Libraries

Furthermore, throughout the project you are encouraged to leverage as much as possible from the existing Java libraries (both the Java Collections Framework, and the JPT libraries). The designer should focus on the design of interfaces between tasks, between components, wrapper and adopter class that allow you to use an existing library class in a customized setting.

The View

The requirements

The view at the minimum should have the following functionality:

- User should be able to see a representation of the graph.

  This can be a graphical display, a text that lists the nodes and the edges (with their weights), a graphical display of the text that lists the nodes and the edges.

- User should be able to select which of the three algorithms is to be used for the subsequent task.

- User should be able to specify the origin and the destination of the desired path.

- The user should be able to see the resulting path.

  Again, this may be a graphical display, or just a text listing the nodes along the path.

The frills

Of course, the view can be much more elaborate. Here is a list of possible enhancements:

- Highlight the path is a different color in the graphics display.

- Display the steps in the search by highlighting in a different color the *v*isited nodes, the *f*ringe nodes (those currently in the queue or the stack), the *o*rigin, the *t*arget, and the *u*nseen nodes. Animate the process using either the timer, or a user advance triggered by a key press.

- Animate the reconstruction of the path by traversing from the found target back to the previous node, all the way up to the origin.

- Select the origin and the target in a graphics display using a mouse.

- Display in a GUI the path length and possibly the nodes along the path.

- Choose the algorithm through a GUI.

- Make the graphics look like a game — e.g. traversing a street map or a maze.

12.2 The Advice

The design part of each project typically takes the greatest amount of time. the more time you spend thinking things through, the easier it is to actually write the code.

Make sure you think the whole framework through before you start programming. Spend some time researching the Java libraries to see what tasks can be done using the existing tools. Write sample adapters to see how the existing class can be used in your setting.

For example, in our sorting assignment the Traversal interface allowed us to supply the data to the algorithm in a number of different ways — and allowed us to the produce the result in a universally readable manner as well.

Then design the key component by specifying their interfaces — the method headers, the interfaces that various classes must implement or use to get information from others.

For now, you have not learned about various tools and techniques to support such design process — other than class diagrams. Any description that you find helpful in clarifying the roles of the different classes and interfaces in your program is acceptable.

The design document you produce should include a brief user's guide, give a general overview of the project organization as well as describe all data definitions and the key methods. The Javadocs supplement this with detailed information about the actual implementation.

Enjoy