Maze Generation: Comp 8280

Justin Sybrandt

October 24, 2016

1 Data Structures

The following subsections detail specific implementation choices in the Maze Generator, with a focus on the data structures used.

1.1 Class Hierarchy

The overarching data structure governing the Maze Generator is the MVC programming paradigm. Additionally, the Maze Generator defines a framework for creating and showing mazes.

1.2 MVC

All classes in the Maze Generator project are located in either the model, view, or controller package. By splitting classes into these packages, it is easier to ensure that there are no direct connections between the model and the view classes. The controllers each are given a reference to a specific view, in this case a JavaFX pane, and instantiate the necessary model objects needed to populate that view. Views are very lightweight, and contain only logic needed to do their specific task. For example, the MazeCanvas, despite its name, contains only logic needed to scale and display LineData objects and Polygons. This canvas is given a list of LineData and two optional Polygons from the MazeController with no knowledge of what these shapes represent. This puts the onus on the MazeController to obtain a list of LineData objects which properly represent the maze.

1.3 Maze Framework

The maze model defines and implements a framework which makes adding new algorithms and layouts incredibly straightforward. Three abstract classes define the majority of this framework: Room, Maze, and MazeGenerator. At a high level, a Maze is comprised of many Rooms, and a MazeGenerator can run a construction algorithm on a Maze.

1.3.1 Rooms

A Room is defined by a number of walls, a rotation, and a location in maze-space, a two dimensional space which is later scaled by the MazeController and MazeCanvas to eventually arrive at pixel-space. Rooms each contain a collection of Wall objects. It is up to the Maze to inform each Room about its neighbors, which can be done by calling setAdjacentRoom. This method searches through both Room's Walls until finding two Walls which overlap. Upon discovering the shared wall, this method updates both Room's Wall collection to contain a reference to the same object. This allows for fast access to the shared wall from both rooms with no additional indirection.

Walls are defined by a Pair of locations and a Pair of Rooms. The Pair data structure is a custom creation which stores two Optional values of the same type. The main inspiration for this structure is to have a symmetric adjacency relationship between rooms. To accomplish this, the Pair defines a way to get its second value provided you have the first. For example, if Room x is adjacent to Room y, then a Wall exists with a Pair(x,y). When either x or y wants to find its neighbor, it could just call Pair.getOther. The Pair also defines equality such that pair a and b are equal if either $a.left = b.left$ and $a.right = b.right$ or $a.left = b.right$ and $a.right = b.left$. A Wall is considered to be externally facing if it only has one Room in its Pair. Walls are considered to be visible in the maze if Wall.isOpen is set to false.

1.3.2 Mazes

The Maze abstract class defines a common interface for all Mazes to be used by the MazeGenerators. Mostly, this means that a Maze is responsible for creating and positioning a collection of Rooms, setting adjacent Rooms, and returning a row of Rooms if queried, as needed by Ellers Algorithm. Each Maze implementation creates its rooms in its constructor, given its own positioning algorithm.

1.3.3 Generators

The MazeGenerator abstract class defines a common interface for all Generators as well as some helper functions needed by all Generators, such as selecting a start and end room. Each generator takes a Maze object in the constructor, and runs its algorithm in the Generator.generate method. Each algorithm removes walls from a fully disconnected maze by moving within a collection of Room objects by traversing between neighbor rooms. In that manner, generators typically use the Room collection like a graph.