GUI-based Coding Challenge

(Assessment Task 2) – Project Software Development Process

Jack Mooney - 13230559

# Problem Specification

To design and develop a GUI based traffic simulator that simulates a set of vehicles (car, bus, motorbike), that move in multiple directions on multiple roads and obey the stop and go commands of multiple traffic lights. The user will be able to enter two modes: City editing, that allows the user to build, save, edit and open a city that they created. And the simulation mode in which the simulation will run.

The program will be executed within a SimCity class and utilize a statistics class to keep track of the results of the simulation. Of course these are only the most basic classes and during the implementation of the project some auxiliary classes may appear.

This program will be used during JCU open day in 2020. It will behave like a simplified traffic simulator.

# Problem Decomposition

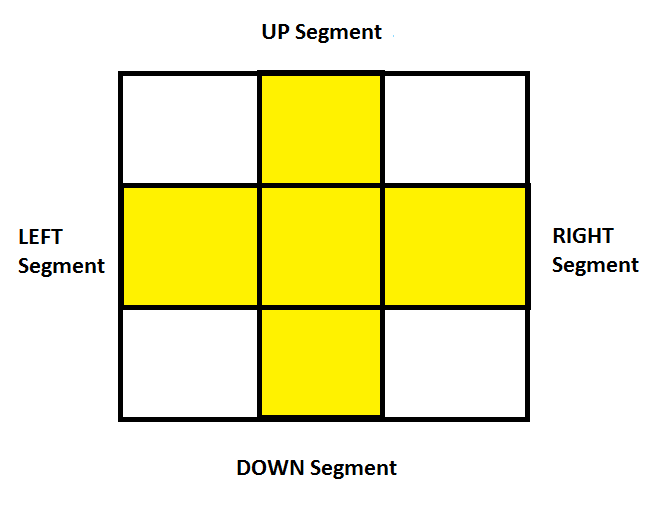
This program will be constructed using ~~fourteen~~ classes, in which comprises of many objects and methods.

## 2D City Road Planning

At the current stage of implementing the coordination for the city, I define a **CityCell** class. The **City** object will contain a fixed grid size of 20 by 20 cells (for simplicity sake). And after much deliberation with a hint of frustration, a new concept for the roads will be implemented to ensure a successful intersection.

Each city cell will contain 0 – 4 road segments, this layout is what will make a working intersection possible.

**Diagram:**



The central square will act as the intersection. When a vehicle reaches this square, the vehicle will choose a direction depending on the presence of the adjacent connected road segments.

A file format is needed in order to save the state of the city. The first line in the file will contain the grid dimensions (I haven’t used any of this yet) and the remaining lines will contain the sections of straight roads.

The coordinates set the beginning and end of the road:

**Example:**

X1, Y1, X2, Y2 …

Since each cell contains 0 – 2 segments of straight road, the layout of the coordinates are as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cell 0 | | Cell 1 | | Cell 2 | | Cell 3 | | Cell … | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | … | … |

Let’s say we have a horizontal road, then the odd coordinates will indicate that the road begins (or ends) with the left segment, and even coordinates mean that the road begins (or ends) with the right segment. This is the same for vertical roads.

So inside the city.txt file (describing the city) the format is:

20, 20

1,1,1,40

1,1,40,1

21,2,21,39

10,10,30,10

This defines a city that consists of **20 by 20** cells in which contain **4 roads.**

The functions for loading a file into the **City** class are also implemented

(loadCity(String) and addRoad(int, int, int, int)).

Since the **CityView** class is the direct descendant of the JPanel, redefining the paintComponent() function to draw the city grid is necessary.

~~As a result, I have this view of the city:~~

## SimCity

This class simply creates a window and is responsible for starting the execution of the program.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## SimCity2D

A JFrame that will contain a city view, a status bar (for displaying statistics and a menu bar.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

City

This class contains all the objects of simulation. It contains three separate lists for the vehicles, traffic lights and roads. This class allows the user to add new objects and implements the method step() in which performs one cycle of the simulation, calling the appropriate methods for all of the objects.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## CityView

This class is a representation of the city and will be implemented as a JPanel in which all the objects of the simulation will be displayed.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## CityCell

This class is used to represent the roads within the city. It is responsible for driving the coordinates that the vehicles will follow in order to move from one road to the next.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Statistics

This class contains a reference to the City object and calculates the total number of vehicles, traffic lights and the average speed. This information will then be displayed in the status bar for every simulation cycle.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

SimItem

This abstract class will be the base class for all the objects (vehicles, traffic lights and roads) in the simulation and implements the *Position* and *Drawable* interfaces.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Road

This class is responsible for constructing the road/boundaries that the vehicle object will travel on. Much the same as in the first past of the assignment but should be redesigned to be able to intersect with other roads (so far this only supports a straight connection, but not an intersection, I’m thinking of how to implement this correctly.)

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Direction

Direction represents a direction within a 2D plane. Each direction stores a relative position.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## TrafficLight

Defines the condition that changes the behavior of the vehicle to go or stop.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Vehicle

This abstract class and it’s inheritors (Car, Bus, Motorbike) define various vehicles. The move() method will determine the logic of the car’s movement. It follows the vehicle if there is one in front, chooses a random direction at intersections and obeys the commands of a traffic light.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Car

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Motorbike

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Bus

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Drawable

This class is responsible for determining the method for drawing an object.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |

## Position

This Class determines the current coordinates of the object.

|  |  |
| --- | --- |
| **Attributes** | **Behaviors** |
|  |  |