team_we_got_this

March 24, 2015

Implementation

In this section, we describe how we approached the implementation stage of our project. We started this project using the waterfall approach but after the intermediate report we realised that this approach was not the best model for our team. Instead we switched to the agile (scrum) methodology. This methodology was more appropriate for our group because the sprint allowed us to focus on a few attributes in one week before moving on to the next. The project will be implemented with the help of designs such as Architectural Patterns and UML diagrams from the previous sections.

From the MVC model, the View is responsible for rendering the Model information to the user. The view also provides the interface required by the user such that their actions can be recorded and sent to the controller. The Model maintains the state information of the engine. The View then queries the Model to fetch the current State Information. The Model is also responsible for maintaining and updating the functionality of the application based on the user requests, i.e pausing the simulation. The Controller receives the user actions from the view and requests the Model to act as per the request.

In our simulation the View is responsible for displaying the collection of cells which form a grid. All the vehicles are then moved on these cells. We have used JFrame class of the Java Swing API to show the grid. The frame is set on a border layout with a certain size as appropriate. There is a Menu Bar that has been added with some basic functionalities that allow the user to control the simulation. One of the actions the user can perform in the view, is to start the simulation based on a map they have already created. However if the user does not have a map they wish to use, they can load the default map. This map will display all the attributes that can represented in our simulation as it contains both a crossroad and a roundabout. Currently a user can start and pause the simulation. This allows them to capture screen-shots of the simulation for a report they may be writing. The user also has the option to pause the data for the simulation and this could be extended to allow the user to pause the traffic lights of the system and collect data. Then reintroduce traffic lights and collect more data. Finally they can use the two sets of data to draw comparisons. This functionality has been implemented using a MenuActionListener object which has the functions required to have a control on the users actions. All the user actions are taken from the view and passed on to the controller for respective

actions that have to be taken. Effort has been made in making sure any change in the status of the model is recorded by the application and is befit to be used wherever necessary in future developments.

An important aspect of our simulation is how the controller reads the user's map. When a user creates and saves their own map, this map is stored as a textfile which is a grid of numbers.

0	0	0	3!	4∗	0	0	0	0
0	0	0	3	4	0	0	0	0
0	0	0	3	4	0	0	0	0
0	0	0	3	4	0	0	0	0
1∗	1	1	5	5	1	1	1	1!
2!	2	2	5	5	2	2	2	2∗
0	0	0	3	4	0	0	0	0
0	0	0	3	4	0	0	0	0
0	0	0	3∗	4!	0	0	0	0