Requirement Specification Document

team_we_got_this

February 11, 2015

Requirement Engineering

Requirements Engineering is an important part of software development life-cycle which includes such activities as requirements elicitation (collecting requirements from current users by talking to them and collecting information about workflows in an organisation), identification of new requirements, analysis (checking requirements to be realistic and also that they truly reflect customers needs and wants) and creating a requirements specification: a document which gathers all the requirements and is usually signed by the client so that in the end the system can be checked against the document to make sure everything that has to be there is there. Some other processes are requirements validation to make sure they stay up-to-date and continue meeting shareholders needs as the time of development progresses, and requirements management i.e. techniques of how to introduce changes in requirements. As we use the waterfall model, we will perform listed activities and the outcome of this will be the requirements document.

Requirements Elicitation

The following initial requirements come from the Introductory Lecture Slides:

Meta-requirements:

- Development must be coordinated through github repository.
- The source code can be written in any language.
- Documentation must be provided in Latex format and exported as PDF.
- Unit testing of high quality and coverage should be performed.
- Code documentation and comments for methods and variables should be provided.
- Work should be critically evaluated to highlight encountered problems and also parts of the project that worked well.

System Requirements:

1. The system must simulate individual vehicles such as cars, coaches and buses.
2. The road network must have different parts including roundabouts and multi-lane junctions.
3. The network must have place where vehicles enter and where they leave.
4. The model should be able to simulate individual behaviour of drivers, e.g. reckless, cautious and normal.
5. The system should be able to time each cars journey to present efficiency statistics according to purpose and patterns of use.
6. The simulation might support emergency services such as ambulance to give them priority at traffic lights.
7. The model should be able to make use of different policies and test their effectiveness and report on their success or failure rates using a particular measure (e.g. average speed, congestion rate).
8. The engine should have a particular state which depends on how long the simulation has run for and a time granularity constant (macroscopic or microscopic) must be chosen which indicates on how often the state is updated, with vehicles changing their position and new cars being created, old cars being removed.
9. Users should have the ability to configure their own maps of an arbitrary scale.
10. A GUI or command line should be used to visualise maps and simulations. After considering the available requirements, our team decided to go ahead and implement a number of ideas of how such requirements can be modelled. We came up with 3 main ideas:
    (a) state automaton or discrete [screenshot] [brief description]
    (b) continuous or formula-based [screenshot] [brief description]
    (c) network-flow model [screenshot] [brief description]

We have chosen to go on and implement the state automaton model, therefore new requirements which are more precise than elicited ones has to be drawn from requirements identification based on the prototypes developed. Also, non-functional requirements are presented there.

Non-functional Requirements

Reporting: the system should report on results of a run simulation according to a number of factors, which are: how conjectured the roads were (percentages of road cells occupied by cars to empty road cells) and how long it took cars to get from A to B (in respect to the minimum time it would have taken them without any other cars).

Performance/ response time: the simulation should run without lag on a regular University computer (e.g. in MSc lab 534), i.e. 64-bit Inter Core2 Quad CPU @ 2.5GHz for a medium-size simulation. However, when the complexity increases to more than 500x500 cells and 250 cars, the system is allowed to show a reasonable decrease in performance. With 1000x1000 cells and 500 cars, system is not guaranteed to continue responding to user input or show feedback on its state.

Testability: the code for the project must be written by programmers with a fact in mind that it will have to be tested using a unit test framework. For example, in case of Java, the unit-testing framework would be JUnit. Test might be written beforehand to promote test-driven development, but it is not compulsory. On the other hand, it is required that every method has a corresponding test code written for it to make sure it works correctly in a variety of situations, such as boundary and corner conditions, therefore edge-cases must be tested.

Usability: it must be easy for users to understand how to use the simulation capabilities. Jakob Nielsen's 10 general principles for interaction design [ref: http://www.nngroup.com/articles/ten-usability-heuristics/] should be considered and applied for implementation of user interface, for example there should be a match between the system and real world so users understand the analogy, that means that cars in the simulation should look like cars and the designs for road and roundabouts must be as realistic as possible. Also, help should be provided for users with explanations of how to use the simulation engine. A heuristic testing for all the 10 principles must be performed in the end of the UIs development life-cycle by an expert.

Portability: the system should be able to run on a computer with JVM installed (thus ensuring that the program can be started on any device which supports Java), or as an applet on a web-page.

Maintainability: the code should make use of interfaces and appropriate design patterns to make sure that all programmers can understand it and make required changes (refactor it) easily.

Extensibility: it should be possible to introduce new features into the system without having to restructure the engines core if additional requirements emerge. For example an appropriate level of abstraction should be used to en-

sure that a cyclist lane occupied by cyclists, taxis, buses and motorbikes can be incorporated into the system and the later stages of development.

Documentation: every public method must be documented in the source code in Javadoc-style, including the purpose of the function, description of each parameter and return value, as well as clarification of exceptions, using @param, @return, @throws tags. @author and @version tags must be specified for each class. Documentation of a private method is encouraged when it can help other people working on the project to understand its purpose. Finally, UML diagrams (use-case, class and sequence at least) must be created in the design stage to make sure the principles of overall model are clearly conveyed to shareholders and also team of developers.

Functional Requirements: map designer

1. Users should be able to create a new blank map by specifying its width and height in cells.
2. Users should be capable to click on each cell and choose whether it is empty or a road.
   (a) If a cell is a road, users should be able to specify in which direction it is going (NORTH, EAST, SOUTH or WEST).
   (b) If a cell is a road, users should have an option to make it an entry node or an exit node.
   (c) If a cell is a road, users can place a traffic light on it, or remove existing traffic light from it.
3. A map can be saved to a file with .map extension, which contains binary data of serialised grid object.
4. Users can load a map saved earlier by choosing a file
5. Users should be able to erase all objects from the map (i.e. to start over).
6. Map builder should be programmed in ActionScript 3. [Screenshot of a blank map, and map with road, cars and traffic lights on it]

Functional Requirements: simulator engine

1. The system must be able to load a map file and initialise the appropriate state of the simulation
2. The cars must appear from entry nodes according to the flow specified as a parameter.
3. The cars must disappear from exit nodes every tick of a clock.

4. The simulation controller must move cars according to their speed and direction, making sure that collision do not occur, e.g. if a car needs to be moved to the next cell which is already occupied by another car, it will not be moved. In future, a collision can be allowed to occur and cause a traffic jam, but this is subject to changes in requirements.

5. The cars must be allowed to go ahead in their current direction, or change direction when they reach junctions.

6. The appropriate methods should exist to make cars enter, follow and exit roundabouts.

7. Traffic lights colour must change according to a certain delay associated with it by the global controller. reconsider this: The possible states are: RED, RED-AMBER, AMBER, GREEN. The exact behaviour of cars in these situations must be specified by a policy.

8. Each car should have a timer which starts when it enters the map, and stops when it exits to collect statistics of time taken for it to travel through the road network.

9. The simulation might be paused by the user and its state can be saved by him/her to load later. do we need/have time for this?

10. The vehicles may be able to overtake each other.

11. The behaviour may be added to each vehicle which determines speed they are driving with and decisions they make at traffic lights (e.g. always stop on AMBER for cautious drivers).

12. Special vehicles may be included in the simulation such as ambulance, fire brigade and police services which will get priority at traffic lights and other vehicles will be required to change lanes to give way.

13. Time-granularity constant should be included as a global static variable so it can be changed easily to test the behaviour of the system and give user freedom of control.

14. The model should have a GUI user interface implemented in Java Swing with menu items at the top, grid display on the left and buttons for control on the right.

15. The users must be able to specify policy and receive reports when the simulation ends.