Test Plan

March 22, 2015

1 Introduction

This test plan has been created to check if our system conforms to all functional requirements and also to specify the types of testing which will be used to validate the functional requirements, action required on failed tests and time line to perform testing of the system. Exhaustive testing of system is not possible but we will use a diverse range of tests to find bugs and errors in the system including unit, functional, error and system testing. System is implemented using java therefore cross platform testing will be performed too. Automation testing will not be possible because most of the methods needs objects to be passed as parameters and their behaviour is changed on every tick of simulation.

2 Objectives and Tasks

2.1 Objectives

The objective of testing our system is to provide adequate testing of functional requirements, validation and behaviour of system under simulated condition. Testing will be done by running the simulation, observing the agents' behaviour and comparing it with expected behaviour. Debug methods are written to eliminate bugs in the code and exceptions are handled to avoid the crashes. Simulation is run atleast more than 3 times in order to test the expected behaviour of agents during simulation run.

2.2 Tasks

- Identifying functional requirements and writing the tests cases.
- Executing tests.
- Record the failed test cases and reporting them to team.
- Performing the re-test on failed test cases one bugs are fixed.

3 Scope

In context to our system, which is implemented using agent-based model it will be tested at three different levels of functionality.

3.1 Levels of Functionalities

- **Agent behaviour:**
  Can be checked by changing environment variables around agents. These behaviours of agents also forms the functional requirements of the system as well. We have two agents in our system.

  1. **Vehicles**
     step() method of Vehicle class is the entry point of control into simulation. All methods related to vehicle behaviour are invoked inside this method.

     - **Test Case1:** Vehicle Impasse
       Expected Output: This "if condition" in step() method should not be true i.e. no Vehicle should be stuck in impasse.
       Actual Output: Console output does not print the statement "Vehicle is stuck in impasse. Cannot move...". "if condition" remains false.
       Result: Passed

     - **Test Case 2:**moveTowards()
       Input:Coordinate and displacement
       Expected Output:Every vehicle on the network should move to a new coordinate and exception should not be thrown printing on console "Could not move vehicle for some reason.".
       Actual Output:Vehicle moved. No exception thrown.
       Result:Passed
     - **Test Case 3:**Accelerate() Every vehicle must accelerate in order to move.
       input:Takes ACCELERATION and time t to calculate velocity.
       Expected Output:Vehicles should be moving not more than maximum velocity.
       Actual Output:No vehicle colliding and no vehicle static if car ahead is not stopping.
       Result:Passed

     - **Test Case 4:** slowDown()Every vehicle must slow down in order to stop.
       Input:Takes ACCELERATION and t*2 to calculate velocity.
       Expected Output:If velocity is less than zero then vehicles must stop.

Actual Output:Vehicles stopped when velocity is less than zero.
Result:Passed

2. **Traffic Lights**
   - **Test Case 5:** Traffic Light Green
     Expected Output:Cars should keep moving if traffic light is green.
     Actual Output:Cars did not stop.
     Result:Passed.
   - **Test Case 6:** Traffic Light Red
     Expected Output: Cars should stop if traffic light is red.
     Actual Output: Cars stops when traffic light is red.
     Result: Passed
   - **Test Case 7:** Traffic Lights On More Than 2 Roads Junction
     Traffic lights should only be displayed on junctions that contain more than two roads.
     Expected Output: Only junctions with more than two roads display traffic lights.
     Actual Output: Traffic lights are displayed only on more than two road junctions.
     Result: Passed

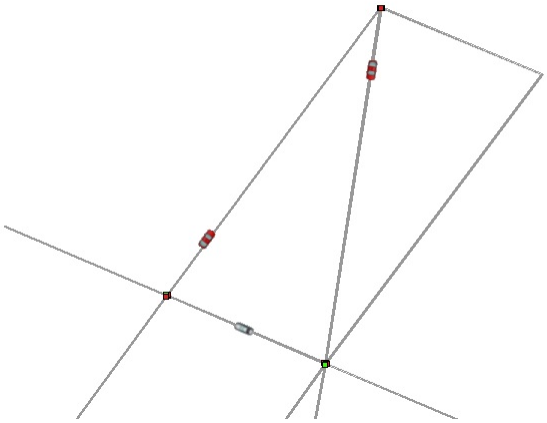

Figure 1: Network with traffic lights.

- **Runtime Parameters:**
  Elements and data which loads at run time will form part of runtime testing. It will

be tested, giving all valid and invalid inputs and check results. Following elements will be tested at run time:

- **Test Case 8:** SignalGreenBuilder Implements ContextBuilder
  This class contains methods to build the context and initialise all variables required during the simulation execution.
  Input: Run signalGreen Model, to load user GUI which contains GIS map, Vehicle agents and TrafficLight Agents.
  Expected Output: Map should be loaded and displayed along with three different types of vehicles and traffic lights on all junctions that have more than two edges.
  Actual Output: Map loaded with 100 vehicles which is default value and traffic lights on junctions with more than two edges only.
  Result: Passed
- **Test Case 9:** Number of vehicles
  Input: Enter a number in "Number of vehicles" field on parameters tab.
  4, 0 , -1 and a character given as input.
  Expected Output: Only entered number of vehicles should appear on road network. No vehicles should appear for negative numbers and characters.
  Actual Output: Vehicles are equal to number entered. No vehicles for negative and character input.
  Result: Passed

- **Test Case 10:** No Traffic Lights
  "Traffic Lights" on parameters tab, if unchecked traffic lights should not appear on display and cars should not consider traffic lights.
  Expected Output: Cars should travel on road without stopping at traffic lights as there are none.
  Actual Output: No traffic lights are displayed and cars carry on moving on roads.
  Result:Passed.

- **Overall behaviour of system:**
  Checks that all agents of the system produce expected results in a given scenario. This will include testing of agents interaction with other components of environment (roads, traffic lights) when programme will be executed at run time.

  - **Test Case 11:** Every junction holds a queue of vehicles running on a particular outward road.
    Expected Output of next.printVehiclesQueue(origin):[signalGreen.Vehicle@54008645, signalGreen.Vehicle@1a43b86b] Peek vehicle: signalGreen.Vehicle@54008645
    Actual Output: Using probe tool, selecting the area which we want to inspect: There are indeed two vehicles running on W 34th St, and their object ID match.
    Result: Passed

Figure 2: Probe Selection

4 Conclusion

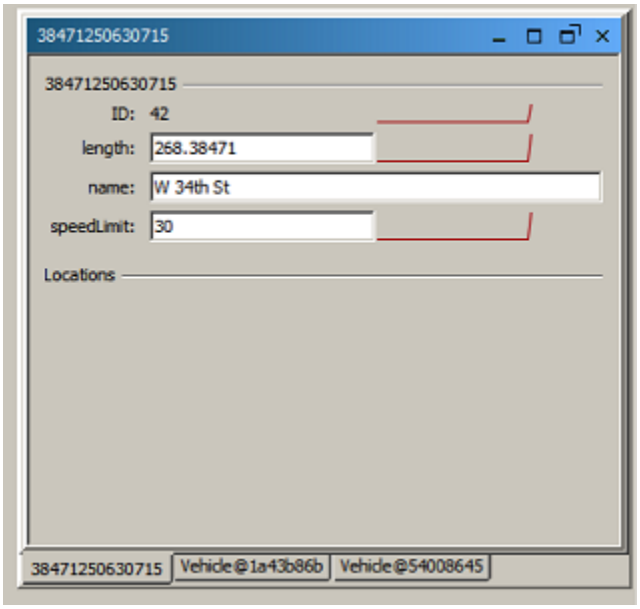

Figure 3: Vehicle IDs