Touring Machines

Turing Route 66
GitHub page
Trello board

Stan Meyberg, Thijs van den Berg, Casper Smet
TICT-AI-V2A-19

HU University of Applied Sciences Utrecht

2019
December

Contents

1 Our experiment

In our experiment we want to model a 1-lane road based on the Nagel-Schreckenberg model. This model will eventually be expanded with an on ramp and a traffic light. Using this simulation, we will try to answer the following research question:

> How does a traffic light and its timing affect throughput on a highway?

2 Method of Approach

2.1 Data collection

While the simulation is running, it will generate a lot of data. Eventually we want to evaluate our experiment with this generated data. To be able to that we need to collect and store some data from the simulation. We're most interested in the following data:

- The velocity of every agent per time step
- The amount of agents (cars) on the road per time step
- The positions of every agent per time step
- The average velocity of all agents

With this information we ought to be able to formulate an answer to our research question.

2.2 Environment setup

In the simulation, agents will travel and can only travel by road. The road is represented by a grid of cells. This grid of cells defines the Nagel-Schreckenberg model as a cellular automaton. Each cell contains zero or one agent.

The road initially consists of one lane. According to the Nagel-Schreckenberg model, there is no end of the road. The visual end of the road will be connected to the beginning of the road. This will form a continuous highway with no ending.

Time in the environment will develop in discrete time steps. Every time step, each agent will perceive the environment and possibly change their position on the grid. The behaviour of the cars will be further explained at the *Car behaviour*-section.

If time permits, an on ramp will be added to the road. This ramp shall contain a traffic light. The traffic light will regulate the throughput of cars joining the main road. The behaviour of the traffic light will be further explained at the *Basic traffic light behaviour*-section.

2.3 Car behaviour

The basic behaviour of a car in the Nagel-Schreckenberg model is defined as such:

- A car perceives its current velocity. When a car is not at its maximum velocity, it accelerates. Each time step, velocity increases by one unit.
- A car perceives the distance between it and the nearest car in front of it. It is always able to perceive the nearest car. If the distance is smaller than its velocity, it brakes. The car's new velocity then equals the distance.
- Each time step, a car may randomly reduce its velocity by one with a probability of p[1]. Velocity may not be reduced to 0.
- A car then moves forward in cells equal to its velocity.

These four steps are conducted in order from first to last. If time permits, the following behaviours and alterations to behaviour will be added to this model.

2.3.1 Car to car behaviour

While a car travels on the road a car can choose to overtake the car in front of it. To overtake a car in front, there has to be a significant difference in velocity between the two agents in favour of the tailing car.

For overtaking the standard rules for driving on a road apply. This means that a car can't overtake a car in front of it on the right side. The agent also may not overtake when there exists a car in the lane on his left hand side directly beside him. The agent perceives where it is and where other cars in its vicinity are at each time step.

Meanwhile a car may also be overtaken by the car behind him.

2.3.2 Lane merging behaviour

While a car tries to merge on the main road the following set of rules apply:

- The agent on the on ramp can only switch to the lane on his left hand side
- The agent on the on ramp can't switch lanes if there is another car on the lane the agent wants to switch to
- There is a limited space on the on ramp for an agent to merge, when the agent comes to the end of this lane it will halt completely and will try to merge the next time step.

An optional addition will be to let the agents that are already on the road perceive that some cars want to merge, and that these agents will attempt to give them room.

[1] In the original model, $p = 0.5$.

2.4 Traffic light behaviour

The traffic light placed on the on ramp will either be on or off.

If the traffic light is off, the agent on the on ramp will directly try to join the highway. The agent on the on ramp will only succeed in joining the highway *if* any of the cells connected to the on ramp are empty. If the agent on the on ramp is not able to join the highway, the agent will come to a full stop on the on ramp. Any agents directly behind the stationary car will act the same as on the highway and will also come to a full stop.

While the traffic light is on, agents on the on ramp will come to a halt when the light is red. Agents will decrease their velocity by one every time step on the on ramp while the traffic light is orange. While the light is green, agents on the on ramp will increase their velocity by one every time step. The lane merging of agents is described in the *Lane merging behaviour*-section.

The timings of the traffic light will be easy to change. During the batch run, multiple traffic light timings will be tried and analysed.

2.5 Batch run

In order to properly analyse a simulation with a variety of settings, the simulation must be run a multitude of times.

Ideally, you do not have to reset and rerun a simulation for each new set of settings.

This requires a 'Batch run' functionality. For this simulation, the setting in need of testing is the timing of the stoplight.

2.6 GUI

The GUI will not only serve as a pleasure to the eyes but will also serve as a handy tool to easily change the parameters of the simulation. The GUI will also serve as a visual way to verify that the model and underlying scripts are operating as intended.

3 Plan of Action

Eventually we will have to model the environment and the behaviour of the agents that exists in that environment. In our method of approach we described what each specific module contains. In our Plan of Action we will state the specific functions/classes we eventually want to deliver/build. In this plan we have marked the optional alterations with an asterisk in front of it.

3.1 Data collection

- A function to collect the chosen data and store it

3.2 Environment setup

- A single road represented by a grid of cells
- * A 2-lane road represented by a grid of cells

3.3 Car behaviour

- Car class with basic behaviour according to the Nagel-Schrekenberg model
- Car merging behaviour
- * Car overtaking behaviour

3.4 Traffic light behaviour

- A function that regulates the amount of agents that join the highway

3.5 Batch run

- Function to run the simulation with different settings

3.6 GUI

- A visual representation of the road with the agents on it
- * Sliders to control the amount of cars on the road
- * Slider to control the amount of agents joining via the on ramp
- * Slider to control the interval of the traffic light

4 Tool selection

For this particular model, defining agent behaviour is not a point of contention. The two most important aspects of this simulation are:

1. Data collection
2. Environment

The behaviour an agent exhibits is fairly simple, and can be implemented easily. The data collection functionality is necessary to be able to come to any conclusions based on the simulation's results. Without a proper grid-based space or discrete time functionality, this model would be nearly impossible to implement.

These will be the deciding factor in choosing an ABMS tool.

For each module, except for the behaviour modules, the tool will be given a rating out of five based on their suitability. Each tool will also be given a rating out of five based on their feasibility.

4.1 Mesa

4.1.1 Suitability

Data collection **** The data collection requirements are easy to implement in Mesa. Mesa offers the 'Data Collection' module. This module allows for easy model- and agent-level variable saving. The only required parameter being a dictionary containing the names of the variable one wishes to save.

It stores these in Pandas data frames. Pandas being a famously fast module for transforming and analysing large amounts of data.

Environment ****

Environment; Space **** The environment would be easy to implement in Mesa. Mesa offers the 'Space' module. This contains classes which perfectly fit the spacial requirements. See *space.Grid* and *space.SingleGrid*.

Environment; Time **** Mesa also offers the 'Time' module, which again, offers classes which perfectly fit the chronal requirements. See *time.SimultaneousActivation* and *time.StagedActivation*.

Batch run ***** In order to run a simulation many times with differing settings, Mesa offers the 'batchrunner'. Batchrunners allow batch runs. 'Batchrunner' also works well with the 'Data Collection' module. This can even be done using multiprocessing, making the process significantly faster.

GUI *** In order to dynamically alter settings and render simulations, Mesa also offers the 'Visualisation' module. Using this module, one can generate simple web-based visualisations.

This web interface also allows for sliders for any set of settings. It also updates in real time based on the script it is executing on, eliminating the tedium of restarting your interface every time a script is altered.

Perhaps the greatest pro for this implementation of a GUI is how easy it is. A GUI with three sliders, a grid and a graph with live animations took slightly less than 70 lines of code.

4.1.2 Feasibility *******

Due to the prior experience of this team, Mesa should be an easy to use tool. Mesa is used in Python, and all three members of this team have considerable experience in this programming language.

4.2 Unity

4.2.1 Suitability

Data collection *** There is not a standard or build-in solution to data collection in Unity. C# is also not a common language for data collection or data farming. It is possible, however, to read the values of specific agents every frame of the simulation. These variables can then be saved to e.g. a .CSV file.

Environment ***

Environment; Space *** Most projects in Unity are 3D. For the simulation however, we would like to use a 2D grid. Unity offers a way to make 2D projects. Most 2D Unity projects are side-scrollers. For a cellular automaton model it would be more common to use a top-down view. To make the Nagel-Schreckenberg model as a side-scroller would however be very original. With some workarounds it will be possible to implement a grid in Unity.

Environment; Time **** The functions Perceive, Update and Action will be executed every frame of the simulation. This makes time in Unity discrete. These functions shall be executed parallel for multiple agents in one frame.

Batch run *** Just like Data Collection, there is no default solution for batch running the simulation in Unity. Unity offers the ability to change the values of the agents and the environment while the simulation is running. For the batch run to be useful, data during the different configurations needs to be collected and saved. As discussed at Data Collection, this will be very difficult.

GUI ***** The GUI is one of Unity' strengths. Unity offers incredible graphics with little to no effort. It is possible to make sliders to control values as the amount of cars on the road or the maximum velocity of the agents. The underlying code however will need to be written manually and could be rather difficult.

4.2.2 Feasibility *****

Unity is a very powerful game engine which offers a good basis for graphical stunning simulations. A lot of things specific to making simulations however are missing and need to be written manually. The learning curve of Unity is very steep and all the team members have little experience working with Unity and coding in C#.

4.3 NetLogo

4.3.1 Suitability

Data collection *** There isn't really a build-in method for gathering info about a process or the agents. However, the user can make use of global variables which by updating them every tick can eventually be used to plot the data. Furthermore the user can make use of the $file-close$ and $file-open$ commands to write away the values the variables or agents hold.

Environment

Environment; Space *** The needed environment would be relatively easy to implement in Netlogo, because Netlogo already works with a grid of cells called patches.

Environment; Time *** The functions that define an agent will be executed every tick. However these agents will not operate parallel to each other. Netlogo is at its core a synchronous system.

Batch run **** In order to perform experiments with models and to run a model many times with different settings Netlogo has BehaviorSpace. BehaviorSpace is a tool integrated with NetLogo that allows you to run a model many times, systematically varying the settings of the model and recording the results of each model. It lets you explore all the possible behaviours of a model.

GUI ****** The GUI of Netlogo isn't that extensive, but it all works kinda intuitive. Netlogo takes a lot of hassle of displaying the simulation away. The user can literally type $create - turtles20$ and Netlogo will display 20 newly created agents on the screen.

Furthermore with Netlogo the user can create all kinds of inputs, sliders, buttons, etc on the screen to let the end-user choose some parameters for itself.

The style of the GUI of Netlogo may be a little old-fashioned, nonetheless it does its job well and correct.

4.3.2 Feasibility ******

Netlogo was designed for both research and education purposes. Therefore it has a nice environment for building multi-agent simulations and is user friendly. The learning curve is much lower than say for Unity. That said there is no prior experience in the team with Netlogo and its language. Therefore the development of the simulation can perhaps take more time.

4.4 Chosen tool

The chosen tool is Mesa. It scored the highest of all three tools.