ODD for Traffic Intersection Models

Isaiah Hollars, Alana Martinec, and Ilaria Youssef

Introduction

Our model description follows the ODD (Overview, Design concepts, Details) protocol for describing individual and agent-based models (Grimm et al. 2006; 2010; Railsback and Grimm 2018). We start with a complete ODD for our first model of the standard traffic circle (circle.nlogo). Our other 3 traffic models are based on this model. We outline their key differences in the last section. This modeling problem is based on the 2009 MCM Problem A. The NetLogo files and data analysis files can be found on our Github Traffic repo.

Purpose and patterns

Most are familiar with different road intersection designs such as the 4-way stop, traffic light, and roundabout. The purpose of our model(s) is to determine what 4-way intersection design leads to optimal traffic flow.

It is known that roundabouts are able to handle larger volumes of traffic than traffic lights and 4-way stops. We analyze our models against this pattern.

Entities, state variables, and scales

The world is initialized as a 201x201 grid without wrapping. Simulations run for a total of 5000 time steps (we denote these as ticks). An appropriate distance and time scale is yet to be implemented. So ticks only represent an abstract time unit.

Cars are the main entity for the model. Cars are often referred to as turtles since we are working in NetLogo. When cars are created, they are given a spawn number and an exit number. These never change and are used by cars to decide when to exit the roundabout. The spawns are numbered 0-3. The exit number ranges from 1-3. If a car has exit number 3, it will take the “third exit.” Each car also keeps track of its constantly updating coordinates. Lastly, each car tracks the number of blue patches it has run over. Key patches of the model are drawn a certain color, but patches have little role to play besides serving as a location for cars.

Each entrance road has a probability represented by the globals north-spawn-prob, south-spawn-prob, east-spawn-prob, and west-spawn-prob. These four are all set to the slider variable spawn-prob-all in the current version. The model could be adapted to make these four spawn probabilities different. There is also a global variable called stopping-distance, which does what it says. Lastly, the global variable num-complete tracks how many cars have reached their destination.

Process overview and scheduling

Every tick, the model executes a procedure to spawn new cars at road entrances (spawn-turtles) and then a procedure that makes cars move (turtle-go). The turtle-go procedure is run on every car. Lastly, the model increments the tick count. These two procedures are discussed more in the Submodel Section.

The scheduling order of spawn-turtles does not make a difference. The scheduling order for turtle-go is chosen randomly. Thus, the order in which turtles each execute this procedure is randomized each tick (see here).

Design concepts

Emergence: Patterns in the roundabout traffic are emergent (not coded in). For example, even as traffic at a roundabout builds, a large window usually opens up allowing most of the cars on an entering road to enter the circle all at once.

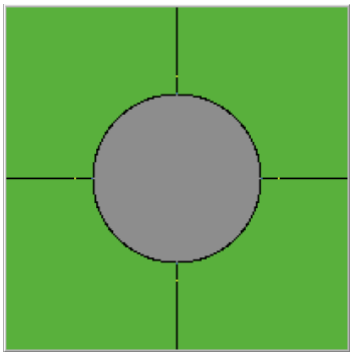
Sensing: Cars are able to sense the cars ahead of them on an entering road and once they're in the circle. This allows the cars to stop within stopping-distance of the car in front. Additionally, cars are able to sense if the roundabout is safe to enter or if they should yield.

Interaction: Car interaction is facilitated through yielding and stopping behind other cars.

Stochasticity: The car spawning procedure relies on randomness. Each tick, there is a fixed probability of spawning a car at one of the four road entrances. Additionally, the procedure that asks all cars to go uses a random scheduling procedure.

Observation: The total number of cars that complete their destination at the end of one simulation (5000 ticks) is observed. Cars that reach the end of their exit road are deleted and added to this total. We seek to maximize the number of destinations completed, because this is a measure of traffic flowing well. We use this measure to compare other intersection designs and determine their relative effectiveness.

Initialization



A traffic circle is drawn about the origin with a radius of 50. The x-axis and y-axis are drawn as incoming roads to the traffic circle. These all serve the purpose of 2-way roads, so cars are able to travel both directions on them. Axis lines inside the circle can then be erased. If all goes right, cars will not be driving inside the circle!

We draw in four yellow patches at the coordinates (0, ± 60) and (± 60, 0). These yellow patches are where incoming cars will stop if they must yield to circling traffic. We also draw four blue patches at the coordinates (0, ± 50) and (± 50, 0) at the edge of the circle. These will help cars determine when to exit. When the model is initialized, all global variables are initialized (e.g. stopping-distance is set to 5 and spawn-prob-all is set to 0.03).

Submodels

spawn-turtles: This runs the same spawn procedure for the four road entrances. These spawn positions are at (0, ±100) and (±100, 0). As an example, turtles spawned at (0,100) will be initialized with spawn-number 0, a random exit-number from 1-3, and a heading that points south towards the circle. We always spawn turtles with their heading towards the circle. The spawn procedure is as follows. Generate a random number from 0-1. If the random number is less than the spawn probability for that location, then we spawn a new turtle as long as there isn't already another turtle close by with the same heading. By close by, we mean within a radius of stopping-distance (a global variable).

turtle-go: This is the most involved procedure. It is broken up into cases based on the number of blue patches a turtle has run over. If the turtle has run over 0 blue patches, it is still entering the circle. If there is a turtle ahead within the stopping-distance, then we must stop no matter what. Otherwise, we check if the turtle is currently on the yellow yield. If the turtle is not on the yield patch it goes forward 1. If the turtle is on the yield patch but it is safe to proceed, the turtle also goes forward 1. Lastly, we check to see if we ended up on a blue patch. If we did, we turn the turtle right by 90 degrees and set the number of blue patches run over to 1.

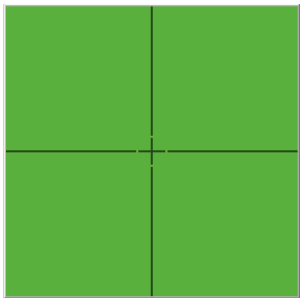
In the case that the number of blue patches run over equals (exit-number + 1), then the turtle will be exiting already. So, we move the turtle forward by 1. If the turtle bumps into the edge of the world, we delete the turtle and increment num-complete by 1. A turtle has reached its destination!

In the case where the number of blue patches run over is anything else, the turtle will still be circling. We have the turtle arc forward by distance 1 around the circle. The turtle also checks if its current patch is blue to update the number of blue patches run over. If the number of blue patches reaches (exit-number + 1), we set the turtle's heading to prepare for exit.

yield?: The turtle checks to see if it should yield to circling traffic. This reports true if the car should yield and false if it's safe to go. To decide, the turtle checks the quarter of the traffic circle to its left. It then narrows in on the roughly 75% of this arc that is closest to them. If there are cars on this arc of the circle, then the car decides to yield to them (report true). Otherwise, it is safe to proceed (report false).

Related Models

Alternative Traffic Circle (bad_cirlce.nlogo)
This model has the same road layout as the base traffic circle model. In this model, we have cars inside the traffic circle yield to cars driving into the circle. This is accomplished by moving around the yield patches and making the needed adjustments to the yield? and turtle-go procedures.



Four Way Stop (4way_naive.nlogo)
The circle is taken away. Roads are drawn in as the x-axis and y-axis. Four yellow patches are drawn near the intersection for cars to stop at. We keep track of the first cars to stop with a queue. The cars proceed through the intersection in the order they stopped at their yellow patch.

Traffic Light (lightv1.nlogo)
Again, roads are drawn as the axes and yellow patches indicate where to stop. Yellow patches are moved further back. Each road (north, east, south, and west) takes turns letting their cars through the intersection. These turns last 150 ticks. In addition, the cars make sure the intersection has cleared before taking their turn.