ABM – Week 6 – Seminar – LVL2

Purpose

This task will allow you to explore the concepts of agent interaction and agent memory. You will write code to detect agent interactions, use lists to represent agents' memory of these interactions, and allow agents to change their strategy based on their memory.

Model

Open the model Ringroad.nlogo. This model represents cars driving around a closed circuit of road in both directions.

Look at the code and think about the following things:

- How does the model differentiate road patches and non-road patches?

- The choose-direction procedure allows turtles to follow the road.

    a) Why is "neighbors4" used rather than "neighbors"?
    b) What is the purpose of the code "in-cone 1 300"?

Task

Give the agents a variable defining whether they drive on the left or the right (their 'driving side'. If two agents who have chosen different driving sides meet, while travelling in opposite directions, they will 'crash' (but in this simulation, they should continue their journey); e.g. a "left" driver meeting a "right" driver will 'crash'.

Adapt the code to detect such 'crashes'. Give the agents a memory of the driving sides of all agents they have encountered. Allow agents to adapt their own driving side based on this memory. Plot a graph of the proportion of agents using each driving side over time.

Extensions

- Add road sections leading to the outside world and allow new vehicles to enter and leave the world via these roads.

- Include multiple different possible adaptation procedures:
    - Adaptation using only the *n* most recent interactions;
    - Adaptation only with a certain probability.

Step-By-Step Guide

Giving all turtles a "left" or "right" driving side

Create a turtle variable to store this information and assign values to the turtles.

Identifying crashes

Create a turtle procedure that checks whether a turtle has encountered any turtles travelling in the opposite direction. Of those, check whether any have the opposite driving side. If so, there is a crash; colour all turtles involved in a crash in yellow. Recolour turtles red when they next move. Add this new procedure to the 'go' procedure.

Useful primitives: turtles-here, heading, any?

Giving turtles memory of encounters

Create a turtle variable to represent turtle memory. This variable should start as an empty list (created just like in Python). Whenever a turtle encounters another turtle heading in the opposite direction (you checked for these turtles at the previous step), it should add the driving side of the encountered turtle to its memory.

Note: It is quite tricky to get this stage to work properly so that all turtles in an encounter add to their memory, rather than just one, particularly if there are more than two turtles involved. Perhaps start by writing code that gets one turtle to add to its memory, then continue to the next instruction to complete the task. You can then return at the to get this part working perfectly.

Useful primitives: lput

Making turtles adapt their driving side based on their memory

Create a new turtle procedure that tells a turtle to switch its driving side to the most common driving side of those that it has encountered. The procedure should only operate if the turtle has had at least one encounter. Add this new procedure to the 'go' procedure.

You will need to use the 'modes' primitive, which finds the mode of a list. However, this produces a list (because sometimes there is more than one mode, e.g. if the turtle has encountered an equal number of "left" drivers and "right" drivers). This means you need to set the driving side to a randomly chosen one of these modes.

Useful primitives: if, not, empty?, modes

Creating a graph to monitor the proportion of turtles driving on each side

You can either do this by writing a formula into the plot editing window, or by creating a reporter ('to report') that returns the proportion of turtles driving on a particular side.