ABM – Week 8 – Seminar – LVL1

Purpose

This task will allow you to use links and to model agent interactions using game theory. Agents will interact in pairs and will grow or shrink based on their success in those interactions. The code uses the 'matrix' extension.

Model

Open the model international_relations_baseline.nlogo. You will develop this model to represent nations interacting periodically in pairs. The interactions will take the form of the following game, where nations may choose to be friendly, neutral or hostile (payoffs are to the row player):

	Friendly	Neutral	Hostile
Friendly	2	1	0
Neutral	3	1	0.3
Hostile	4	1	0.6

Take a couple of minutes to look at the code and take particular note of the following things:

- What are the global variables?
- What variables do the turtles have?
- How does the model define the payoff matrix?
- How has the shape of the turtles been changed?
- How have the turtles been arranged across the world?

Tasks

1. Create directed links between all the turtles. At each tick, select a link and make the turtles at either end play the game shown above. Since the nations all have the “random” strategy, they should choose their row/column at random. Based on the outcome of the game, make each turtle gain the appropriate amount of utility. Complete the set-sizes procedure so that a nation’s size is proportional to its utility.

2. Introduce a second strategy option for my-strategy, called “hostile”. Make sure that some turtles use this strategy. Turtles with this strategy should always be hostile in their interactions. Colour nations differently according to their strategy.

Step-By-Step Guide

After each change you make, check that the code runs before you move on to the next step…

EXERCISE ONE

<u>Creating directed links between all turtles</u>

1. Add a line to the crt block inside the setup-turtles procedure to create directed links to all other turtles.
   [Useful primitives: create-links-to, other, turtles.]

Your simulation should now run as before, but all turtles will be connected by arrows in both directions.

<u>Selecting a link at random and identifying the turtles at each end</u>

2. In the 'interaction' procedure:

   a. Create a temporary variable called interaction-link, set as one of the links, chosen at random.
   [Useful primitives: let, one-of, links.]

   b. Create temporary variables called player-1 and player-2, to store the turtles at each end of the selected link.
   [Useful primitives: let, end1, end2, of]

   c. Set the thickness of interaction-link to 1.
   [Useful primitives: ask, set, thickness]

   d. Set the thickness of the interaction link to 0.1.

[Useful primitives: ask, set, thickness]

Your simulation should now run as before, but (if you run the simulation slowly enough) each link will flare up briefly whenever there is an interaction along it.

<u>Coding the random strategy</u>

3. Create an empty reporter called 'random-strat'.
   [Useful primitives: to-report, end]

4. Complete this reporter to return a random number from 0 to 2.
   [Useful primitives: report, random]

This reporter will choose a row at random for a turtle who is playing the game.

<u>Playing the game</u>

5. Create an empty procedure called 'play-game' that takes two arguments called first-player and second-player.
   [These arguments represent turtles who are interacting.]
   [Useful primitives: to, end]

6. In this procedure, create temporary variables called player-1-strat and player-2-strat and set each equal to the ouput of the reporter 'random-strat'.
   [Useful primitives: let]

7. Call the 'play-game' procedure in the 'interaction' procedure, before the lines that set the thickness of the interaction link to 0.1. The arguments should be 'player-1' and 'player-2'.

<u>Updating turtle utility</u>

8. Create an empty procedure called 'update-utility' that takes four arguments, called 'first-player', 'second-player', 'first-strat', 'second-strat'
   [These arguments represent turtles who are interacting and their chosen row/column of payoff matrix.]
   [Useful primitives: to, end]

9. Complete the 'update-utility' procedure by asking 'first-player' to increase their 'utility' (i.e. set 'utility' equal to 'utility' + something) by the appropriate value selected from the payoff matrix.
   [Useful primitives: ask]
   [Necessary procedure from matrix extension: matrix:get]

10. Call the 'update-utility' procedure <u>twice</u> just before the end of the 'play-game' procedure. The first time, the arguments should be 'first-player', 'second-player', 'first-strat', 'second-strat'; the second time the arguments should be 'second-player', 'first-player', 'second-strat', 'first-strat'.

[This will make sure that BOTH turtles update their utility based on the game.]

Your simulation should now run as before, but if you inspect a turtle (on a Mac: CTRL click, then click on the name of the turtle), you should be able to watch its utility increasing as the simulation runs.

<u>Making turtle size reflect utility</u>

In the 'set-sizes' procedure…

11. Create a temporary variable called 'average-utility'. Let this variable be equal to the mean utility of all turtles.
    [Useful primitives: let, mean, of, turtles]

12. Ask the turtles to set their size equal to the global variable 'average-turtle-size', multiplied by their 'utility', divided by 'average-turtle-size'.
    [This will make a turtle's size proportional to its 'utility', but will keep the average size of the turtles constant.]
    [Useful primitives: ask, set, size]

Your simulation should now run as before, but turtles will change in size as they gain or lose utility relative to others. (If you run the simulation too fast, you will not see this effect.)

EXERCISE TWO

<u>Coding the hostile strategy</u>

13. Create an empty reporter called 'hostile-strat'.
    [Useful primitives: to-report, end]

14. Complete this reporter to return the value 2.
    [Useful primitives: report]

This reporter will always choose Row 2 (hostile) for a turtle who is playing the game.

<u>Setting up turtles with either the random or the hostile strategy</u>

15. Edit the 'set-strategy' procedure so that, rather than setting 'my-strategy' to "random", it insteady sets 'my-strategy' randomly to either "random" or "hostile".
    [Useful primitives: one-of]

16. In the same procedure, set the colour of the current turtle to white if it has the "random" strategy and red if it has the "hostile" strategy.
    [Useful primitives: if, set, color]

Your simulation should now run as before, but turtles will be either white or red. If you inspect a turtle, you will see that 'my-strategy' corresponds to their colour.

However, they all still choose their strategy at random at the moment. We have not yet told the "hostile" turtles how to behave in interactions.

<u>Making the hostile turtles use the hostile strategy</u>

17. Create an empty reporter called 'choose-strategy'.
    [Like the 'set-strategy' procedure, this reporter will be in turtle context.]
    [Useful primitives: to-report, end]

18. In this new reporter, if the current turtle has "random" as the value of 'my-strategy', report the output of the reporter 'random-strat'.
    [Useful primitives: if, report]

19. Also, in this same reporter, if the current turtle has "hostile" as the value of 'my-strategy', report the output of the reporter 'hostile-strat'.
    [Useful primitives: if, report]

20. Finally, edit the 'play-game' procedure. Rather than just setting 'player-1-strat' and 'player-1-strat' equal to the output of 'random-strat', instead set them equal to the 'choose-strategy' reporter, evaluated by 'first-player' and 'second-player' respectively.
    [Useful primitives: of]

Your simulation should now run as before, but "random" turtles will play the game randomly while "hostile" turtles will always be hostile. This should be visible, since the red and white turtles will settle down to two different sizes, in the long term (speed the simulation up to observe this).