**ABM – Week 9 – Seminar – LVL1**

**Purpose**

This task you will introduce you to a means of coding purposeful agent movement, allowing turtles to navigate effectively to a location of interest, while avoiding obstacles in their path. The extension tasks will incorporate ideas of differential knowledge between agents and how this can affect dynamics.

**Model**

Open the model evacuation\_model\_baseline.nlogo. This model represents concertgoers (pink) and staff members (cyan) milling around a concert venue.



In the tasks below, you will simulate the emergency evacuation of agents from the venue via the three exits (green, blue and red).

Look at the ‘move’ procedure in the code, which controls agent movement.

* What are the restrictions on the movement of concertgoers and staff members and what are the effects of these restrictions?
* Why do you think the primitive ‘carefully’ used in this procedure?

Running the procedure ‘show-staffonly’ in the Command Centre in the Interface should help you understand the model more thoroughly.

*[Simply type the name of the procedure into the field marked ‘observer>’ and press enter.]*

**Task**

Simulate an evacuation alarm going off after a certain number of ticks. Change the move procedure so that all agents head to the green exit (Exit 1) when the alarm sounds. Remove agents from the model when they reach the exit. Create a plot to monitor the proportion of concertgoers and staff members who have successfully evacuated the building as time passes.

To allow the agents to navigate to the exit successfully, you will need to give each patch a variable stating its distance from the exit, *accounting for the internal structure of the building*. To set this variable correctly, you will need to write a procedure that starts by considering the patches in Exit 1 and iteratively sets the distance of adjacent patches inside the building until all internal patches have a distance variable assigned.

**Extensions**

* Alter the code so that agents head to the nearest exit – I (green), 2 (blue) or 3 (red) – rather than all heading to the same one. Add a plot to monitor the number of individuals who have been evacuated by each exit.
* Alter the code so that agents may only have knowledge of certain exits and head towards the nearest exit that they know of, even if another exit may be closer.

**Step-By-Step Guide**

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

Setting off the alarm

1. Add global variables to the model to represent:
2. The time at which the alarm sounds;
3. Whether the alarm has sounded (true/false).

Set appropriate values of these variables in the set-globals procedure.

1. In the ‘go’ procedure, add code to change the value of variable (b) at the time indicated in variable (a).

*[Useful primitives: if, set, ticks]*

Your simulation should now run as before, with no visible differences. You will be able to check that the alarm has sounded by letting the model run, then typing the name of variable (b) to see whether it is ‘true’ or ‘false’.

Determining the distance of each patch from Exit 1

In this section, you will create a procedure that runs a (greatly simplified) version of [Dijkstra’s Algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) to determine the distance of each patch from the exit.

The algorithm will select floor patches in turn, starting with those that form part of the exit itself and working away from it throughout the building. When each patch is selected, it will set the distance of any neighbouring floor patches *that do not yet have an assigned distance* to be one greater than its own distance.

1. Create a new procedure called ‘define-evacuation-routes’. Add this procedure to an appropriate point in the ‘setup’ procedure.
2. Add new patch variables to the model to indicate:
3. The distance of a patch from Exit 1;
4. Whether the patch has been selected yet.
5. In the new procedure…
6. … ask the patches that make up Exit 1 to set their distance to 0 and their ‘selected’ variable to ‘false’.
7. … ask all other floor patches to set their distance to a very large number (e.g. 1 million) as a default value.

*[Useful primitives: ask, patches, pcolor, set, with]*

1. (*The trickiest part*) In the same procedure…
2. … create a ‘while’ loop (*see the Netlogo dictionary for the appropriate syntax*) that continues while there are still floor patches with the default distance value used above.

*[Useful primitives: any?, patches, while, with]*

1. Within the loop, create a temporary variable to store the set of all possible patches that could be selected (i.e. those that have not already been selected).

1. Create a second temporary variable called ‘chosen-patch’, selecting one of the patches identified above that has the lowest distance to the exit.

*[Useful primitives: let, one-of, patches, with-min]*

1. Ask the chosen patch to record that it has been selected (by switching the relevant variable to ‘true’.
2. The chosen patch should then ask its four neighbours to set their distance variable to be 1 more than its own distance variable, but only if their distance is currently equal to the default value.

*[Useful primitives: ask, myself, neighbors4, of, set, with*

*Hint: Remember that ‘myself’ refers back to the agent that asked for the current code to be run.]*

Your simulation should now run as before, with no visible differences. However, if you inspect a floor patch, you should see that it has stored the correct distance from Exit 1 (measured by the number of patches in the shortest path).

Altering the turtles movement when the alarm is sounding

In this section, you will make the turtles change their behaviour when the alarm sounds. Rather than milling around aimlessly, they will make their way towards the exit.

1. Rename the ‘move’ procedure as ‘move-normal’ and create a new procedure called ‘move-alarm’.
2. Create a new ‘move’ procedure. In this procedure, if the alarm is sounding, ask the turtles to follow the ‘move-alarm’ procedure; otherwise ask them to follow the ‘move-normal’ procedure.

*[Useful primitives: ifelse]*

Note that turtles have a ‘next-patch’ variable, which stores the next patch that they will visit.

In the ‘move-alarm’ procedure (which is in turtle context)…

1. … set a turtle’s next-patch to be one of the neighbouring 4 patches that:
2. is not a wall;
3. contains no other turtles;
4. has the minimum distance to the exit among all patches that satisfy a) and b).

*[Useful primitives: and, any?, neighbors4, one-of, pcolor, turtles-here, with, with-min]*

1. … copy the code from the ‘move-normal’ procedure that makes the turtle step to its next patch.

Your simulation should now run as before until the alarm sounds. Concertgoers and staff should then all attempt to leave by the green exit. They will then mill around on the street outside, blocking others from exiting the building.

Removing and recording turtles who have successfully evacuated

1. Create a new global variable to count turtles who have left the building, and set this variable to 0 in the ‘set-globals’ procedure.
2. Create a new procedure called ‘remove-evacuees’ and add it to the ‘go’ procedure at an appropriate point.
3. In this new procedure, ask the turtles that have reached the exit to increase the count of turtles who have left by one and then to die.

*[Useful primitives: die, pcolor]*

1. Create a plot in the interface to display the proportions of a) concertgoers, b) staff and c) all turtles that have been successfully evacuated as time passes.

*[Useful primitives: count, plot]*

Your simulation should now run as before, except that turtles will be removed when they reach the exit, rather than spilling out on to the street. The plot will be updated to show how the evacuation is progressing.

Visualising the distance from the exit for all floor patches

This section will not alter the functioning of the model, but it will allow you to see the result of your work following instructions 3-6, above.

1. Create a new procedure called ‘show-evac-routes’.
2. In this new procedure, colour all patches with a positive value for their distance to the exit in shades of yellow, corresponding to the value of this distance.

*[Useful primitives: ask, max, of, patches, pcolor, scale-color, set, with*

*Hint: How did we colour the terrain in shades of green in the butterfly model?]*

You can now run this procedure in the Command Centre in the Interface after setting up the model to visualise the distance of every patch from Exit 1.