The collectivities simulation, coded in NetLogo, can be downloaded from <https://study.sagepub.com/researchmethods/qass/gilbert-agent-based-models-2e>. To run this program, you will first need to install NetLogo, available from <http://ccl.northwestern.edu/netlogo/download.shtml>

## Commentary on the Program

The following explains each line of the simulation program (which has been written using NetLogo 6.0 – but it should be compatible with later versions also).

breed [ agnts agnt ]

This line names the agent class that will be used in the model, in both its plural and singular forms (the name is not a mis-spelling: ‘agent’ is a reserved word in NetLogo and cannot be used as the name of a breed).

The agent class has two attributes, around and visible. Every agent has its own values for these variables, which store the number of other agents surrounding the agent and the set of agents that it can see, respectively.

to setup

clear-all

ask patches [set pcolor white ]

The setup procedure is executed when the user presses the Setup button on the interface. The procedure first deletes any agents left over from a previous run and then colors all the patches on the grid white. The code ask patches [...] tells all the patches (recall that a patch is a cell on the grid) to execute the code inside the square brackets. The term pcolor is a NetLogo variable that sets the color of a patch.

create-custom-agents 500 [

set color green

set size 2

Then 500 agents are created, each facing in a random direction. The commands inside the square brackets are executed by each of the new agents independently. The image of each agent on the view is colored green, and its size is set to twice the minimum.

setxy random-pxcor random-pycor

Then each agent is moved to a random spot on the grid (random-pxcor yields a number corresponding to a random patch coordinate between the left and right edges of the grid, and random-pycor does the same for a patch coordinate between the top and bottom edges).

We do not want more than one agent on a patch, and the procedure be-alone will test whether the agent is alone and if not, make it move forward until it lands on an empty patch. This procedure is defined at the bottom of the code and we will see how it works in a moment.

]

reset-ticks

end

reset-ticks starts the tick counter. There will be one tick of the clock for each step of the model run. This completes the commands that are executed by each agent immediately after it is created, and also brings the setup procedure to an end.

to go

ask agents [ count-those-around ]

ask [ move ]

end

The go procedure gets executed repeatedly when the Go button on the interface is pressed. Each time through the procedure, the simulation moves on one time step (one tick). At each step, all the agents are asked to count the number of agents that are in their local area on the grid, and then all the agents are asked to move. It is important that all agents complete counting the agents around them before any agents move; otherwise, the counts may be confused by agents moving before they have been counted. NetLogo guarantees that all agents complete the commands in a construction such as ask agnts[:::] before the next command (in this case, the second ask agnts[:::]) is started. This would not have been the case if the program had been written ask agnts [count-those-around move] because then each agent would have started to move as soon as it had finished counting, and for some agents, this may have occurred before other agents had added them to their counts. This is an example of the kind of timing problem that one needs to guard against when programming agents that appear to act simultaneously.

The count-those-around and move procedures are both methods of the agnt class, that is, procedures that are defined in the program code to show what it means for agents to count and move. The next lines specify what count-those-around consists of.

to count-those-around

set around count agents with [self !=myself] in-radius local-radius

set visible agents with [self !=myself] in-radius visible-radius

end

In the first line of this procedure, the variable around, owned by each agent, is set to the number of agents that are located within a radius of local-radius. The user sets the value of local-radius before the simulation starts by moving a slider on the interface. It is easiest to see what this line of code does by working backwards from the end. The code in-radius local-radius yields those agents that are located at or within a distance of local-radius away from the agent executing this procedure. This set of agents includes the agent itself (it is zero units away from itself, and therefore closer than local-radius). The code with [self != myself] excludes that agent, and count returns the number of agents remaining. The agent’s attribute around is set to this number. One can think of the agent looking around, counting the number of agents it sees within the local-radius and remembering this in the around variable.

The next line of code is very similar. It finds which other agents can be seen by the agent, that is, which agents are within the visible-radius, a parameter whose value is set by another slider on the interface. The second line is slightly different from the first because the variable, visible, stores not a count of the visible agents but the names of the agents themselves. This is necessary so that they can later be identified (in procedures face-towards and face-away).

The move procedure both decides the direction in which the agent is to move and performs the action.

to move

if any? visible [

The procedure begins by checking whether there are any agents visible to the agent executing this method. If there are, the agent needs to decide whether it is ``lonely’’ or ``crowded’’.

ifelse around>threshold

[

face-away

set color red ]

The condition consists of checking whether the number of agents around this agent is greater than the threshold parameter set with a slider on the user interface. If it is, it is ``crowded’’ and the agent must turn away from the center of the crowd. The procedure face-away does this. In addition, to show on the view that it is “crowded”, its color is changed to red.

[

face-towards

set color green ]

]

The alternative is that the agent is ``lonely,’’ and it must turn toward the crowd and become green.

Now that the agent is facing the right way, it should move forward, at a rate determined by the value of the speed parameter (random speed returns a random number less than the value of speed).

forward random speed

be-alone

end

As before, if the agent lands on a patch that is already occupied by another agent, it continues to move forward in the same direction until it finds an empty patch.

That concludes the move procedure. We still have to define face-towards and face-away.

to face-towards

face max-one-of visible [around]

end

The face-towards procedure retrieves the set of agents that this agent can see (the agents that have been remembered in the visible variable) and finds how many agents surround each of those agents. One of the agents with the most agents around it (max-one-of visible [around]) is considered to be at the center of the crowd toward which this agent wants to move. The command face sets the heading of the agent to the direction of that central agent.

to face-away

set heading towards max-one-of visible [around] - 180

end

The face-away procedure is almost the same, but the direction is 180 degrees opposite to the direction of the most central agent.

Finally, the be-alone procedure is defined.

while [any? Other agnts-here] [ forward 1 ]

It is possible that two or more agents end up on the same patch, one standing on the head of another, so this procedure gets the agent to check whether there are any other agents on the patch. If there are, the agent advances by one unit (forward 1) in the direction of its current heading and checks again, continuing to advance until it has found an unoccupied patch. Because the grid is a toroid, if the agent moves off an edge, it will reappear on the opposite side.