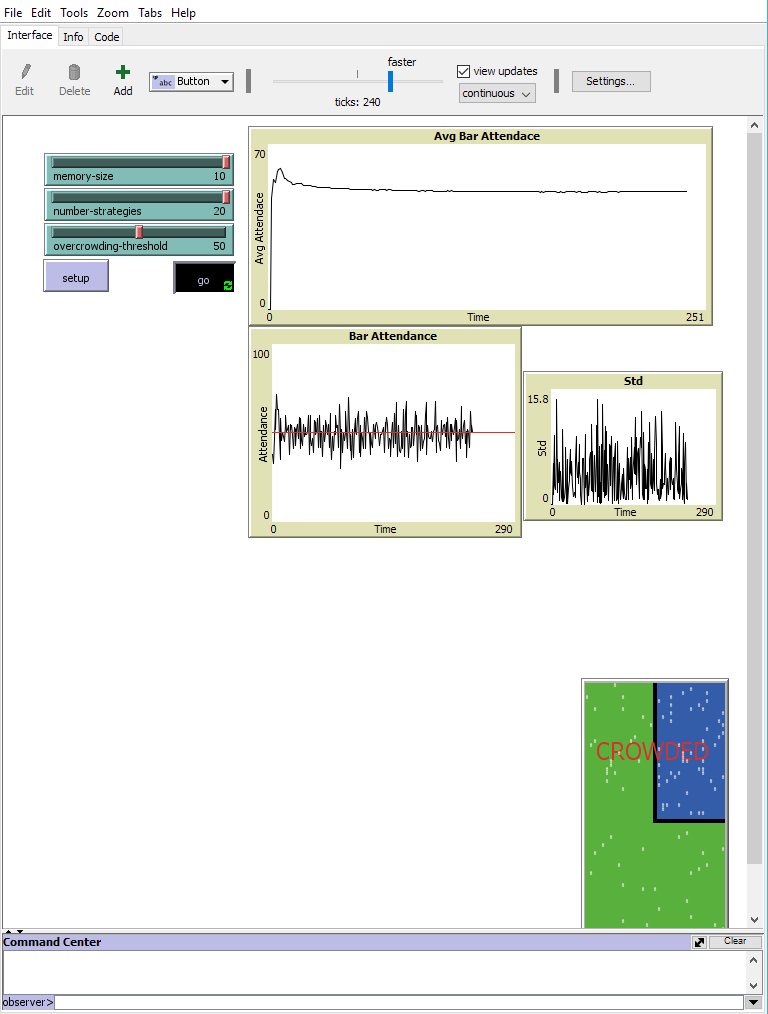
ABM with Netlogo

Netlogo is a multi-agent programmable modelling environment. The language used is very simple and thus allows rapid prototyping. Controlling of important parameters is very intuitive and batch runs of models with multiple permutations of parameters is also possible.



Program Life cycle goes as: -

turtles-own->Setup Function -> Go function -> end function (not compulsory)

turtles-own: -

sets up all the variables associated with each turtle.

Setup function: -

Initialize the environment / variables

Go function: -

Things to do when each tick happens (clock advances by one).

How it Works: -

An agent will go to the bar on Thursday night if they think that there will not be more than a certain number of people there — a number given by the OVERCROWDING-THRESHOLD. To predict the attendance for any given week, each agent has access to a set of prediction strategies and the actual attendance figures of the bar from previous Thursdays. A prediction strategy is represented as a list of weights that determines how the agent believes that each time period of the historical data affects the attendance prediction for the current week. One of these weights (the first one) is a constant term which allows the baseline of the prediction to be modified. This definition of a strategy is based on an implementation of Arthur’s model as revised by David Fogel et al. (1999). The agent decides which one of its strategies to use by determining which one would have done the best had they used it in the preceding weeks.

Interestingly, the optimal strategy from a perfectly rational point-of-view would be to always go to the bar since you are not punished for going when it is crowded, but in Arthur’s model agents are not optimizing attending when not crowded, instead they are optimizing their prediction of the attendance.

The number of potential strategies an agent has is given by NUMBER-STRATEGIES, and these potential strategies are distributed randomly to the agents during SETUP. As the model runs, at any one tick each agent will only utilize one strategy, based on its previous ability to predict the attendance at the bar. In this version of the El Farol model, agents are given strategies and do not change them once they have them, however since they can change their strategies at any time based on performance, the ecology of strategies being used by the whole population changes over time. The length of the attendance history the agents can use for a prediction or evaluation of a strategy is given by MEMORY-SIZE. This evaluation of performance is carried out in UPDATE-STRATEGIES, which does not change the strategies, but rather updates the performance of each strategy by testing it, and then selecting the strategy that has the best performance given the current data. In order to test each strategy its performance on MEMORY-SIZE past days is computed. To make this work, the model actually records twice the MEMORY-SIZE historical data so that a strategy can be tested MEMORY-SIZE days into the past still using the full MEMORY-SIZE data to make its prediction.

In El- farole: -

1)

Each agent has the following properties: -

Strategies => basically list of previous attendances of that agent of previous ticks

best-strategy => weighted sum of previous attendances used for predicting attendance

prediction => Agent’s choice of going or not going in the current tick

code: -

turtles-own [

strategies

best-strategy

attend?

prediction

]

2)

In the setting up function: -

Clear all previous data in the model and start the clock.

Initialize all agents with an initial value of attendance(randomly) and create a memory list which will store their attendance values for memory-size ticks.

The best strategy for tick 0 will be given the value of the attendance at Tick 0

Code: -

to setup

clear-all

set history n-values (memory-size \* 2) [random 100]

set attendance first history

create-turtles 100 [

set strategies n-values number-strategies [random-strategy]

set best-strategy first strategies

update-strategies

]

reset-ticks

end

3)

Going function: -

Make prediction of attendance of each agent

Then update the attendance of every agent by removing the oldest history in the memory and add prediction for the next tick.

Advance the clock by one

Code: -

to go

ask crowded-patch [ set plabel "" ]

ask turtles [

set prediction predict-attendance best-strategy sublist history 0 memory-size

set attend? (prediction <= overcrowding-threshold) ;; true or false

]

ask turtles [

ifelse attend?

[ move-to-empty-one-of bar-patches

set attendance attendance + 1 ]

[ move-to-empty-one-of home-patches ]

]

set attendance count turtles-on bar-patches

if attendance > overcrowding-threshold [

ask crowded-patch [ set plabel "CROWDED" ]

]

set history fput attendance but-last history

ask turtles [ update-strategies ]

tick

end

Complete code: -

globals [

attendance ;; the current attendance at the bar

history ;; list of past values of attendance

home-patches ;; agentset of green patches representing the residential area

bar-patches ;; agentset of blue patches representing the bar area

crowded-patch ;; patch where we show the "CROWDED" label

avg-attendance

sum-attendance

std

temp-attend

]

turtles-own [

strategies ;; list of strategies

best-strategy ;; index of the current best strategy

attend? ;; true if the agent currently plans to attend the bar

prediction ;; current prediction of the bar attendance

]

to setup

clear-all

set-default-shape turtles "person"

;; create the 'homes'

set home-patches patches with [pycor < 0 or (pxcor < 0 and pycor >= 0)]

ask home-patches [ set pcolor green ]

;; create the 'bar'

set bar-patches patches with [pxcor > 0 and pycor > 0]

ask bar-patches [ set pcolor blue ]

;; initialize the previous attendance randomly so the agents have a history

;; to work with from the start

set history n-values (memory-size \* 2) [random 100]

;; the history is twice the memory, because we need at least a memory worth of history

;; for each point in memory to test how well the strategies would have worked

set attendance first history

;; use one of the patch labels to visually indicate whether or not the

;; bar is "crowded"

ask patch (0.75 \* max-pxcor) (0.5 \* max-pycor) [

set crowded-patch self

set plabel-color red

]

;; create the agents and give them random strategies

;; these are the only strategies these agents will ever have though they

;; can change which of this "bag of strategies" they use every tick

create-turtles 100 [

set color white

move-to-empty-one-of home-patches

set strategies n-values number-strategies [random-strategy]

set best-strategy first strategies

update-strategies

]

;; start the clock

reset-ticks

end

to go

;; update the global variables

ask crowded-patch [ set plabel "" ]

;; each agent predicts attendance at the bar and decides whether or not to go

ask turtles [

set prediction predict-attendance best-strategy sublist history 0 memory-size

set attend? (prediction <= overcrowding-threshold) ;; true or false

]

;; depending on their decision, the agents go to the bar or stay at home

ask turtles [

ifelse attend?

[ move-to-empty-one-of bar-patches

set attendance attendance + 1 ]

[ move-to-empty-one-of home-patches ]

]

;; if the bar is crowded indicate that in the view

set attendance count turtles-on bar-patches

if attendance > overcrowding-threshold [

ask crowded-patch [ set plabel "CROWDED" ]

]

;; update the attendance history

;; remove oldest attendance and prepend latest attendance

set history fput attendance but-last history

;; the agents decide what the new best strategy is

ask turtles [ update-strategies ]

set temp-attend avg-attendance

calculate-average-attendance

get-std

;; advance the clock

tick

end

;; determines which strategy would have predicted the best results had it been used this round.

;; the best strategy is the one that has the sum of smallest differences between the

;; current attendance and the predicted attendance for each of the preceding

;; weeks (going back MEMORY-SIZE weeks)

;; this does not change the strategies at all, but it does (potentially) change the one

;; currently being used and updates the performance of all strategies

to update-strategies

;; initialize best-score to a maximum, which is the lowest possible score

let best-score memory-size \* 100 + 1

foreach strategies [ the-strategy ->

let score 0

let week 1

repeat memory-size [

set prediction predict-attendance the-strategy sublist history week (week + memory-size)

set score score + abs (item (week - 1) history - prediction)

set week week + 1

]

if (score <= best-score) [

set best-score score

set best-strategy the-strategy

]

]

end

;; this reports a random strategy. a strategy is just a set of weights from -1.0 to 1.0 which

;; determines how much emphasis is put on each previous time period when making

;; an attendance prediction for the next time period

to-report random-strategy

report n-values (memory-size + 1) [1.0 - random-float 2.0]

end

;; This reports an agent's prediction of the current attendance

;; using a particular strategy and portion of the attendance history.

;; More specifically, the strategy is then described by the formula

;; p(t) = x(t - 1) \* a(t - 1) + x(t - 2) \* a(t -2) +..

;; ... + x(t - MEMORY-SIZE) \* a(t - MEMORY-SIZE) + c \* 100,

;; where p(t) is the prediction at time t, x(t) is the attendance of the bar at time t,

;; a(t) is the weight for time t, c is a constant, and MEMORY-SIZE is an external parameter.

to-report predict-attendance [strategy subhistory]

;; the first element of the strategy is the constant, c, in the prediction formula.

;; one can think of it as the the agent's prediction of the bar's attendance

;; in the absence of any other data

;; then we multiply each week in the history by its respective weight

report 100 \* first strategy + sum (map [ [weight week] -> weight \* week ] butfirst strategy subhistory)

end

;; In this model it doesn't really matter exactly which patch

;; a turtle is on, only whether the turtle is in the home area

;; or the bar area. Nonetheless, to make a nice visualization

;; this procedure is used to ensure that we only have one

;; turtle per patch.

to move-to-empty-one-of [locations] ;; turtle procedure

move-to one-of locations

while [any? other turtles-here] [

move-to one-of locations

]

end

to calculate-average-attendance

if (ticks != 0) [

set sum-attendance (sum-attendance + attendance)

set avg-attendance (sum-attendance / ticks)]

end

to get-std

if (ticks > 1)[

set std standard-deviation (list attendance avg-attendance)]

end