Agent-Based Modelling using NetLogo

F. Delbressine

May 12, 2017

1. Introduction

2. Agent-Based modelling
   - Agents
   - Environments
   - Interactions
   - Observer/User interface
   - Schedule

3. NetLogo

Introduction

Give a person a hammer, and the whole world looks like a nail [1].

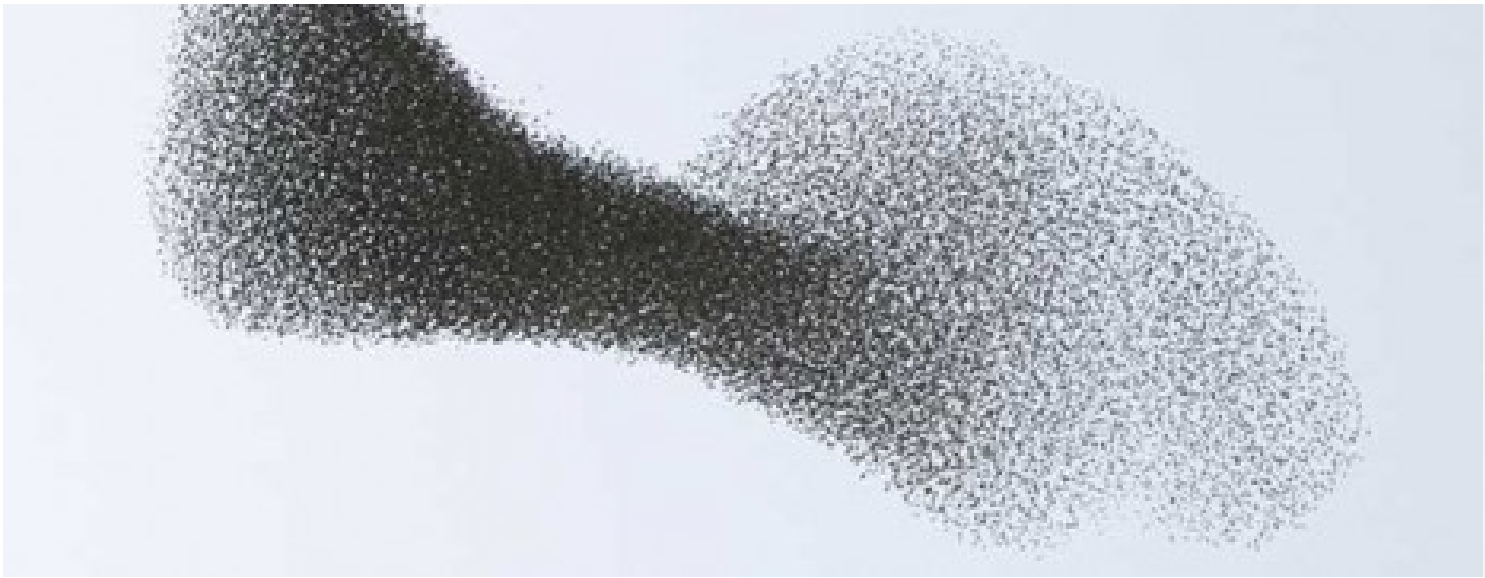
[1] The way we see the world is deeply influenced by the tools we use.

Centralized versus decentralized systems

Examples of decentralized systems:

- nature (flocks, ...)
- financial systems
- social systems
- research
- ...

Example of decentralized control systems



A flock of birds sweeps across the sky. How do birds keep their movements so orderly, so synchronized?
Most people assume that birds play a game of "Follow the Leader".
But that is **not** so. Rather each bird follows a simple set of rules, reacting to movements of the nearby birds. Orderly flocking patterns rise from these simple, local interactions[2].

[2] See the NetLogo Flocking model

Flock NetLogo code

```
to flock  ;; turtle procedure
  find-flockmates
  if any? flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor < minimum-separation
        [ separate ]
        [ align
          cohere ] ]
end

to align  ;; turtle procedure
  turn-towards average-flockmate-heading
end
```

The Components of Agent-Based Modeling [3]

- Agents
- Environments
- Interactions
- Observer/User Interface
- Schedule

[3] U. Wilenski et al., An Introduction to Agent-Based Modelling, 2015

Agents

The two main aspects that define agents are:

- the internal and external states that they have and
- the actions they can perform (behavior).

Issues regarding agent design:

- grain-size: the level of detail and
- agent cognition: how much capability do the agents have to observe the world around them and to make a decision?

Properties

Agent properties describe an agent's current state, the items that you see when inspect an agent. In NetLogo:

- Turtles and links have COLOR as property,
- Patches have PCOLOR as a property.
- Turtles have XCOR and YCOR as properties
- Patches have PXCOR and PYCOR as properties.

Behaviors(Actions)

An agent's actions or behaviors are the ways in which the agent can change the state of the environment, other agents, or itself.
Examples:

- FORWARD, LEFT, HATCH, DIE and MOVE-TO

Example:

```
set speed 0.1 + random-float .9
```

For a complete list of predefined behavior:
https://ccl.northwestern.edu/netlogo/docs/dictionary.html

Define new behaviors for agents

It is also possible to define new behavior:

```
to speed-up-car
    set speed speed + acceleration
end
```

Collection of Agents

Agents are typially divided into three main types:

1. Mobile agents (NetLogo: turtles)
2. Stationary agents (NetLogo: patches)
3. Connecting agents (NetLogo: links)

Links are often used to represent:

- the relationships between turtles.
- the environment (e.g. define transportation routes that agents can move along)
- friendship/communication channels

They share the ability to take actions and perform operations on themselves.

Breeds of Agents

NetLogo modelers can also create their own types of agents called **breeds**.

```
breed [ sheep a-sheep ]
breed [ wolves wolf ]
```

Modellers can also define new properties of breeds:

```
sheep-own [ wooliness ]
```

These properties are in addition to the properties that all agents have.

The Granularity of an Agent

At what level should the agent function?
Some prominent hierarchical levels are:

- atoms, molecules, cells, humans, organisations, and governments.

Guideline:

- Choose the agents such that they represent the fundamental level of interaction that pertains to your questions about the phenomenon.

Agent Cognition

How do agents examine their properties and the world around them to decide what actions to take?
Types of Agent Cognition:

- Reflexive agents (built around very simple if-then rules)
- Utility-based agents (maximize/minimize a utility function)
- Goal-based agents (having goals that dictate their actions)
- Adaptive agents (can change their strategy based on prior experience)

Reflexive agents example code

```
let car-ahead one-of turtles-on patch-ahead 1
ifelse car-ahead != nobody
   [ slow-down-car car-ahead ]
   [  speed-up car ] ;; otherwise speedup
   ...
forward speed
```

Utility-based agents example code

```
let car-ahead one-of turtles-on patch-ahead 1
ifelse car-ahead != nobody
  [ slow-down-car car-ahead ]
  ;; otherwise , adjust speed to
  ;; find ideal fuel efficiency
  [ adjust-speed-for-fuel-efficiency ]
  ...
forward speed
```

Goal-based agents example code

```
if goal = house and
    (member? patch-here [ neighbors4 ] of house)
      [ set goal work ]
if goal = work and
    (member? patch-here [ neighbors4 ] of work)
      [ set goal house ]
```

Adaptive agents example code

```
to adaptive-go
  ;; check to see if our new speed of turtles is better
  ifelse mean [ speed ] of turtles > speed-to-beat
     [ set best-acceleration-so-far acceleration
     set speed-to-beat mean [ speed ] of turtles ]
     [;; give us a chance to learn a better acceleration.
     set speed-to-beat 0.1 * mean [ speed ] of turtles
         + 0.9 * speed-to-beat
     set acceleration best-acceleration-so-far ]
  ;; Try an acceleration to find a better one.
  set acceleration acceleration + random-float 0.002 - 0.001
  go
end
```

Environments

How to design the environment of the agent-based model?
The environment consists of the conditions and habitats surrounding the agents as they act and interact within the model.
Types of environment:

- Spatial Environments (2D & 3D, Geographic Information Systems)
- Network-Based Environments

Interactions

Five basic classes of interactions in Agent-Based models:

- Agent-self (Rabbits Grass Weeds: to-reproduce)
- Environment-self (Rabbits Grass Weeds: grow-grass-and-weeds)
- Agent-agent (Wolf Sheep predation: catch-sheep)
- Environment-environment (Ants: to-go, ask-patches [set chemical ...])
- Agent-environment (Ants: look-for-food)

Agent-self (from: Rabbits Grass Weeds)

```
to reproduce      ;; rabbit procedure
  ;; give birth to a new rabbit,
  ;; but it takes lots of energy
  if energy > birth-threshold
    [ set energy energy / 2
      hatch 1 [ forward 1 ] ]
end
```

Environment-self (from: Rabbits Grass Weeds)

```
to grow-grass-and-weeds
  ask patches [
    if pcolor = black [
      if random-float 1000 < weeds-grow-rate
        [ set pcolor violet ;; grow weed]
      if random-float 1000 < grass-grow-rate
        [ set pcolor green ;; grow grass]
  ] ]
end
```

Agent-agent (from: Wolf Sheep predation)

```
to catch-sheep  ;; wolf procedure
  let prey one-of sheep-here  ;; grab a random sheep
  if prey != nobody   ;; did we get one?  if so,
    [ ask prey [ die ]   ;; kill it
      ;; get energy from eating
      set energy energy + wolf-gain-from-food ]
end
```

environment-environment (from: Ants, ask patches)

```
to go  ;; forever button
  ask turtles
  [ if who >= ticks [ stop ] ;; delay initial departure
    ifelse color = red
    [ look-for-food  ]  ;; not carrying food? look for it
    [ return-to-nest ]  ;; carrying food? take it to the nest
    wiggle
    forward 1 ]
  diffuse chemical (diffusion-rate / 100)
  ask patches
  [  ;; slowly evaporate chemical
    set chemical chemical * (100 - evaporation-rate) / 100
    recolor-patch ]
  tick
end
```

agent-environment (from: Ants)

```
to look-for-food  ;; turtle procedure
  if food > 0
  [ set color orange + 1      ;; pick up food
    set food food - 1         ;; and reduce the food source
    rt 180                    ;; and turn around
    stop ]
  ;; go in the direction where
  ;; the chemical smell is strongest
  if (chemical >= 0.05) and (chemical < 2)
  [ uphill-chemical ]
end
```

Observer/User interface

The observer is high-level agent that is responsible for ensuring that the models runs and proceeds according to the steps developed by the model author.

- User Input and Model Output

Schedule

The schedule is the description of the order in which the model operates.

- SETUP and GO idiom of NetLogo.
- Asynchronous vs. Synchronous Updates
- Sequential vs Parallel Actions

Asynchronous vs. Synchronous Updates

Asynchronous Update: When agents change their state, that state is **immediatly** seen by other agents.

Synchronous Update: Changes made to an agent are **not** seen by other agents until the next clock tick- that is all agents update simultaneously.

Asynchronous updates are more like the real world where agents act and update independently of each other rather than waiting for each other.
Synchronous updates can be easier to manage and debug.
Examples:

Asynchronous Update: Wolf Sheep Predation, Ants, Segregation and Virus

Synchronous Update: Fire, Ethnocentrism and the Cellular Automata

Sequential vs Parallel Actions

Within the realm of asynchronous updating, agents can act sequentially or in parallel.

Sequential Actions: Involve only one agent acting at a time

Parallel Actions: Are those actions in which all agents act independently.

In NetLogo Sequential Actions is the standard behaviour for agents.
NetLogo support: **Simulated concurrency** using the "turtle forever" buttons. Each agent acts completely independent of the others.
Example: the Termites model, see the go procedure.

Simulated Concurrency, from: Termites

All of the turtles are executing this procedure:

```
to go ;; turtle procedure
   search-for-chip
   find-new-pile
   put-down-chip
end
```

NetLogo in short[4]

- **NetLogo** is an extension of **Logo**, a programming language that is generally used for teaching children programming (Papert, 1980).
- Using **Logo** children create geometric patterns by giving commands to a graphical turtle on the computer screen.
- The turtle can be used to represent any type of object in the world.
- To support decentralized modelling **NetLogo** has many turtles
- All turtles can perform their actions in "parallel"
- **NetLogo** turtles can detect things (like other turtles) in their local environment
- The turtles' world is divide into small square sections called patches.

[4] V. S. Colella et al., Adventures in Modelling, 2001.

Syntax

Keywords: **globals, breed, turtles-own, patches-own, to, to-report, end, extensions**

Scope: NetLogo is lexically scoped. Local variables (including inputs to procedures) are accessible within the block of commands in which they are declared.

Comments: **;** (Lasts until the end of the line)

Structure: A program consists of optional declarations (**globals, breed, turtles-own, patches-own, breedname-own, extensions**) in any order, followed by zero or more procedure definitions.

Procedure: Every procedure definition begins with **to** or **to-report**, the procedure name, and an optional bracketed list of input names. Every procedure definition ends with **end**.

Syntax cont.

Commands:

- All commands are prefix => **maximum 45 78**
- Command take zero or more inputs
- No punctuation separates or terminates commands
- No punctuation separates inputs
- Identifiers must be separated by whitespace or by parentheses or square brackets

Operator precedence:

1. **with, at-points, in-radius, in-cone**
2. (all other primitives/user-defined procedures)
3. $\wedge$
4. $*$, $/$, **mod**
5. $+$, -
6. $<$, $>$, $<=$, $>=$
7. $=$, $!=$
8. **and, or, xor**

NetLogo 6.0.

- NetLogo 6.0 Quick Guide link

NetLogo models skeleton

```
globals [ ... ]      ;; global variables
breed [ ... ... ]  ;; define your data type
turtles-own [ ... ] ;; user-defined turtle variables
patches-own [ ... ] ;; user-defined patch variables
links-own [ ... ]   ;; user-defined link variables
<breed>-own [ ...] ;; user-defined <breed> variables

to setup
  clear-all
  setup-patches
  setup-turtles
  setup-links
  reset-ticks
end
```

NetLogo models skeleton cont.

```
to go
  conduct-observer-procedure
  ask turtles [ conduct-turtle-procedure ]
  ask patches [ conduct-patch-procedure ]
  ask links [conduct-link-procedure ]
  tick ;; this will update every plot
end

to-report a-particular-statistic
  report the-result-of-some-formula
end
```