Agent-Based Modelling and Simulation with NetLogo

Day 1: Session 3

NetLogo programming language

Session 3 Outline

- Variables, procedures and reporters
- Basic operators.
- Variable scopes and code contexts.
- Control flow and logic.
- NetLogo dictionary: testing built-in commands.

Variables

- Variables are places to store values such as numbers.
- Variables can be:
  - **Global variables**: there is only one value for the variable, and every agent can access it.
  - **Local variables**: defined and used only in the context of a particular procedure.
  - **Agent variables**: Each turtle has its own value for every turtle variable. The same goes for patches and links. Think of it as agent properties.

Variable values

- Any variable can tipicaly receive any value type at any given time except global variables that come from GUI components (sliders, etc)
- In NetLogo, variable values can be of the following types:
  - **Numbers**: 1, -2, 0.5, -0.1235
  - **Strings:** “xpto”, “a”, “123xpto”
  - **Boolean:** true, false
  - **Agents:** turtle 1, patch 0 0, link 0, one-of turtles
  - **Agentsets:** turtles, patches, n-of turtles, n-of links
  - **Lists**: [1 2 3 4], list 1 2, (list 1 2 3 4 5 6)

Creating variables

- **Global variables:**
  - adding a switch, slider, chooser, or input box**.**
  - using globals[variable-name variable-name2] at the beginning of the code.
- **Local variables:**
  - using the **let** command like: let variable 0
  - if you define a variable at the top of a procedure it exists only inside the procedure.
  - if you define a variable inside a set of **square brackets** for example inside an **ask** command it exists only inside those brackets.
  - This is what we call the scope of a variable.

Creating agent variables

- Using the command turtles-own, patches-own or links-own, for example: **turtles-own [energy speed].**

- **Note:** agents already possess some built-in variables such as color, who (turtle id), xcor, ycor, etc

Setting variables

- You can set variable values by using the **set** command: **set variable-name value**
- You can set **global variables** anywhere in the code as these can be accessed by any agent.
- **Local variables** are only accessible inside the procedures or code blocks where they where defined with let.
- **Agent variables** can be read outside an agent with the **of** command: **[color] of one-of turtles.**
- **Agent variables** can only be set by the agents they belong to with **ask** (inside an agent context this is).

Procedures and Reporters

- **Procedure:** executes a finite set of instructions and exists the procedure. Defined with:

```
to procedure-name

end
```

- **Reporter:** same as a procedure but returns a value to the point where it was called. Defined with:

```
;reporter that reports the value 0
to-report reporter-name
  report 0
end
```

Procedures and Reporters with Parameters

- All the procedures and reporters may receive parameters being defined as follows:

```
;reports the sum of two parameters
to-report report-sum [num1 num2]
  report num1 + num2
end
```

- A reporter or procedure can then be called anywhere in the code using its name and passing the necessary parameters:

```
;show the sum of two numbers
let result report-sum 1 2
show result
```

Basic operators

- **Arithmetic infix operators:**

  +, *, -, /, ^, <, >, =, !=, <=, >=

- They all take **two inputs** except except for the definition of negative numbers for which you have to add parentesis **(- n)**
- For other operations check the NetLogo dictionary: **sqrt**, **abs**, **acos**, **asin**, **atan**, **sin**, **cos**, **exp**.

Variable scopes and code contexts.

- Parameters are passed by value and treated as local variables inside a procedure:

```
;A couple of procedures to explain scopes
to scopes1
  let param 0
  scopes2 param
  show (word "param inside scopes1: " param) |
end

to scopes2 [param]
  ;param is visible only inside this procedure
  set param param + 1

  show (word "param inside scopes2: " param)
end
```

- Result for calling scopes1?

```
observer> scopes1
observer: "param inside scopes2: 1"
observer: "param inside scopes1: 0"
```

More Scopes

```
;A couple of procedures to explore scopes an ask blocks
to scope1-turtles
  let value 0
  ask turtles[
     set value value + 1
  ]
  show value
end

to scope2-turtles
  let value 0
  ask turtles [scope3-turtles]
  show value
end

to scope3-turtles
;     set value value + 1 ;nothing named value is defined in the scope of this procedure
end
```

Control Flow and logic

- Instructions that define the way the program instructions are executed.
- Already seen to define **procedures** and **reporters**: **ask**, **to**, **to-report**, **end**
- **conditional control flow:** if, if-else, ifelse-value
- logic expressions: **and**, **or**, **not**, **xor**

Conditional Expressions

if:

```
if boolean-expression [
    set of instructions
]
```

ifelse:

```
ifelse boolean-expression[
    set of instructions
][
    set of instructions
]
```

Boolean expressions

- An expression that combines boolean values with **logical operators:** and, or, not, xor

- **Example:** p1 **and** p2 **and not** p3

Examples

```
let p1 true
let p2 false

if p1 and p2 [
   ask turtles [show "hello"]
]

ask turtles with [color = red and not energy > 50][
   show "hello"
]

ask turtles with [[pcolor] of patch-here = black and [food] of patch-here > 0][
       show "hello"
]

ask turtles [
       if [pcolor] of patch-here = black and [food] of patch-here > 0 [
              show “hello”
       ]
]
```

Exercise

- Create a simple NetLogo model:
  - A set of n turtles
  - A set of n patches with food
  - Turtles have energy
  - Turtles move in random directions
  - Moving consumes energy
  - Stepping on food patches recharges energy
  - If a turtle runs out of energy, it dies

Next Session

- Re-visiting and building a model of residential segregation from scratch.