Sensing

EES 4760/5760

Agent-Based and Individual-Based Computational Modeling

Jonathan Gilligan

Class #11: Thursday, September 30 2021

Getting Started

Getting Started

Log in to a computer and download the following:

- Link demo model from the download page or https://ees4760.jgilligan.org/models/class_11/link_demo.nlogo
- Team project templates from the download page https://ees4760.jgilligan.org/downloads/team_project_templates/
- If you are working on a lab computer, be sure to save your work to Box, a thumb drive, or some other place you'll be able to have access after class.

Review of Homework 6.3

Review of Homework 6.3

```
turtles-own [ path ]

to setup
  clear-all
  create-turtles 1
  [
    set color red
    pen-down
    set path (list patch-here)
    repeat 100
    [
      rt (random 91 - 45)
      fd 1
      set path fput patch-here
path
    ]
  ]
end
```

```
to go-back
  ask turtles
  [
    set color blue
    foreach path
    [
      a-patch -> set heading towards a-
patch
      fd 1
    ]
  ]
end
```

- Can you see what’s wrong here?

Review of Homework 6.3

- Original model movement:

```
repeat 100
[
  rt (random 91 - 45)
  fd 1
  set path fput patch-here
path
]
```

- Original go–back movement:

```
foreach path
[
  a-patch -> set heading towards a-
patch
  fd 1
]
```

- Improved go–back movement:

```
foreach but-first path
[
  a-patch -> set heading towards a-
patch
  fd 1
]
```

Better model

- Remember the turtle's heading, not the patch it was on.
- setup movement

```
repeat 100
[
  rt (random 91 - 45)
  fd 1
  set path fput heading path
]
```

- Working go–back

```
foreach path
[
  a-heading -> set heading (a-heading + 180)
  fd 1
]
```

Sensing: Important Points

Variable scope

- Global variables (`globals []`)
  - Same value throughout model
- Agent variables (`turtles-own []`)
  - Each agent has its own value
  - For specialized breeds, `<breeds>-own []`:

```
breed [ butterflies butterfly ]
butterflies-own [
  starting-patch
]

to setup
  clear-all
  create-butterflies 10 [
    move-to one-of patches with [
      not any? butterflies-here
      ]
    set shape "butterfly"
    set starting-patch patch-here
  ]
  reset-ticks
end
```

- Patch variables (`patches-own []`)
  - Each patch has its own value
- Link variables (`links-own []`)
  - Each link has its own value
- Local variables (`let`)
  - Only exists within submodel, reporter, or square brackets `[]`

```
to reproduce
  ; num-offspring only exists inside to
reproduce
  let num-offspring 1
  if random-float 1.0 < prob-twins
  [ set num-offspring 2 ]

  hatch num-offspring [
    ; friends only exists inside [ ... ]
    let friends n-of 3 turtles in-radius 10
    set happiness mean [happiness] of
friends
  ]
end
```

Links and Networking

Links and Networking

- Links allow you to connect turtles
  - Friendships
  - Family relationships
  - Business relationships
  - ...
- Two kinds of links:
  - Undirected:
    - `create-link-with` *turtle*
    - `create-links-with` *turtleset*
  - Directed:
    - `create-link-to` *turtle* and `create-link-from` *turtle*
    - `create-links-to` *turtleset* and `create-links-from` *turtleset*

Directed vs. Undirected Links:

- For any pair of turtles:
  - There can only be one kind of link between them (*directed* or *undirected*)
  - If there is a *directed* link between them, there can be links in both directions:

```
let partner one-of other turtles
create-link-to parther
create-link-from partner
```

  - A turtle can have *directed* links to or from some turtles and *undirected* links with other turtles
  - But **the same pair of turtles can't mix *directed* and *undirected* links**

```
let partner one-of other turtles
create-link-to parther
create-link-with partner ; this causes an error!
```

Working with Links

- **Links:**
  - `my-out-links`: *Directed links pointing away* **and** *undirected links*

    ```
      ask my-out-links [ set color pink ] ; link turns pink
    ```

  - `my-in-links`: *Directed links pointing to* **and** *undirected links*

    ```
      ask my-in-links [ set size 2 ] ; makes link thicker
    ```

  - `my-links`: All links (*directed* **and** *undirected*)

    ```
      ask my-links [ set thickness 2 ] ; size is thickness of
    line
      set link-avg mean [link-length] of my-links
    ```

Working with Link Neighbors

- **Turtles at the other end of links:**
  - `link-neighbors`: Turtles at the other end of *directed links* **or** *undirected* links with myself

    ```
    ask link-neighbors [ set color blue ] ; turtle turns blue
    ```

  - `in-link-neighbors`: Turtles at the other end of *directed* links pointing *to* myself **or** *undirected* links *with* myself

    ```
    ; receive payment from neighbors on in-bound links
    set wealth wealth + 5 * count in-link-neighbors with [wealth >= 5]
    ask in-link-neighbors with [wealth >= 5]
    [ set wealth wealth - 5 ]
    ```

  - `out-link-neighbors`: Turtles at the other end of *directed* links pointing *from* myself **or** *undirected* links *with* myself

Tying Turtles Together

- Tying causes turtles to mirror each other’s actions
  - Tying undirected links:

    ```
    ask one-of links with [is-undirected-link?] [ tie ]
    ```

    If either turtle turns or moves, the other will do the same turn or move.
  - Tying directed links:

    ```
    ask one-of links with [is-out-link?] [ tie ]
    ```

    If this turtle turns or moves, the one at the other end of the out link will do the same turn or move.

    If the other turtle turns or moves, it does not affect this turtle.

Getting Fancy with Links

- Find the best patch next to any turtle within two links on the network

```
let connections out-link-neighbors ; neighbors
set connections (turtle-set connections
                                ([out-link-neighbors] of connections)
                ) ; neighbors of neighbors
let target max-one-of (patch-set [neighbors] of connections)
      [ quality ]
```

- Break down:
  - Set `connections` to out link neighbors
  - Next:
    - Find `[out-link-neighbors] of connections` and combine this in a `turtle-set` with the original `connections`
    - Neighbors and neighbors of neighbors
  - Finally look at all the patches next to any of the `connections` turtles and select the one with the greatest `quality`

Model with Links

Model with Links

```
patches-own [ quality ]

to setup
  ca
  initialize-patches
  initialize-turtles
  initialize-links
  reset-ticks
end

to initialize-patches
  ask patches [
    set quality random-float 100
    set pcolor scale-color green quality 0 300
  ]
end

to initialize-turtles
  create-turtles 50 [
    move-to one-of patches with
      [not any? turtles-here]
    set color red
    set size 0.75
  ]
  ask turtle 0 [
    set color pink
    set size 1.5
    ]
end
```

```
to initialize-links
  ask turtle 0 [
    create-links-to n-of 3 other turtles [
      set thickness 0.2
      set color orange
    ]
    ask out-link-neighbors [
      create-links-to n-of 3 turtles with
          [not any? my-links]
      [
        set thickness 0.1
        set color (orange + 3)
      ]
    ]
  ]
end

to-report best-patch
  let subjects out-link-neighbors
  set subjects other
    (turtle-set subjects
      ([out-link-neighbors] of subjects))
  report max-one-of
      (patch-set [neighbors] of subjects) [ quality
]
end
```

Team Projects

Business Investor Model

Business Investor Model

- **Entities:**
  - Investors (turtles)
    - Each investor invests in one patch
    - Only one investor per patch
  - Businesses (patches)
- **State Variables:**
  - **Global:**
    - *T* = time horizon for investments (5 ticks)
  - **Investors**:
    - *W* = wealth
  - **Businesses**:
    - *P* = profit per tick
    - *F* = probability of failure (investor loses all wealth)
- **Objective**: Maximize wealth over time
- **Adaptation**: Move to best vacant patch they can see
- **Sensing**:
  - Submodel for calculating value of patch
  - Limited range of vision for sensing patches
- Value submodel
  - *U* = expected value (utility) of patch

$$U = (W + TP) \times (1 - F)^T$$

Telemarketer Model

Telemarketer Model

- **Entities:**
  - Telemarketing companies (turtles)
  - Consumer households (patches)
- **State Variables:**
  - **Telemarketers:**
    - Size (# employees, telephones, etc.)
    - Bank balance
  - **Households:**
    - Have they been called already this tick?

- **Process Overview:**
  1. Patches reset "have I been called?"
  2. Telemarketers make sales calls
     - Call customers within some radius of self
     - Bigger firms have larger radius
     - Customer buys a product from first telemarketer that calls them, then rejects subsequent calls.
  3. Telemarketers do weekly accounting:
     - Income from successful sales
     - Cost of payroll, phone bills, etc.
     - If bank balance < 0, go bankrupt
     - If bank balance is large enough, spend money to grow
- Later we will explore large firms acquiring smaller ones.

Start Team Project

Start Team Project

- Download project template if you haven’t already.
- Work with your partner to start writing code from the ODD
- If you’re working on a lab computer,
  Remember to save your work to Box or take it with you at the end of class.