Introduction to NetLogo

- Notes based on tutorials and parts of manual at http://ccl.northwestern.edu/netlogo/docs/
- Why NetLogo
- NetLogo GUI
- Elements

Why NetLogo

- Easy to get running
  - Web interface available with support for most features
  - Mature (since 1999!), but still active
- Easier to focus on agent-related aspects
- Provides graphical display showing agent behaviors
- Many examples

Issues

- You do need to learn a new syntax
  - Basic programming language is NetLogo, version of Logo adapted for working with agents
- Not all agent-related features supported in basic installation
  - But extensions available for major features

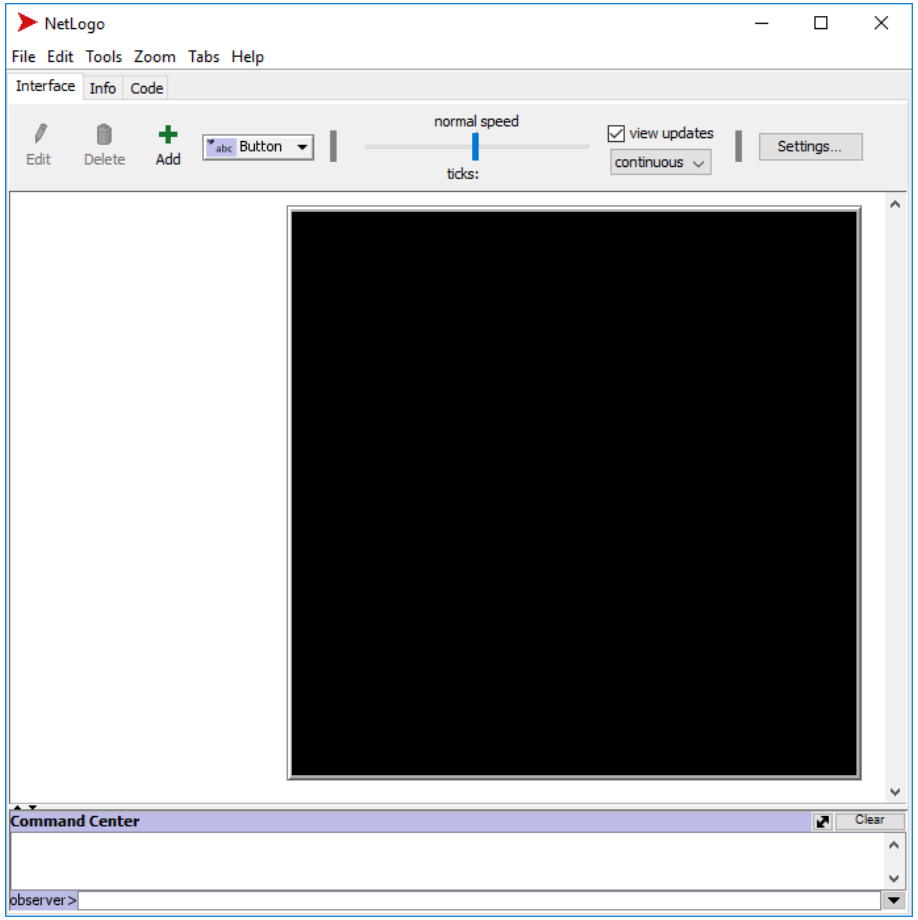
Other general features

- Does not support OOP (cannot define new classes)
  - But does have built-in object-like elements
  - Does support using Java and other languages based on the Java VM
  - In particular, for extensions, manual recommends Scala
- Many commands support abbreviations

NetLogo GUI

- Will focus on the download – much of this is supported in the browser option, too
- When installing, option given to install short-cuts on desktop
  - Otherwise, can run program from NetLogo installation folder
  - For Mac OS Sierra, click on Release Notes on docs URL for updated info
    - NetLogo 6.0.2 has an attempted fix, but there is also a link for a workaround
  - For Linux, update to Java 8 if you have not already done so

NetLogo GUI (run NetLogo to see details)



NetLogo GUI

- File menu includes command to choose from existing model library
- Tabs below main menu line let you toggle between user interface, documentation for current model, and code for model
  - Here, “code” refers to NetLogo code
  - [Supporting Java/Scala code would be developed in a standard IDE, then included in the model as a jar file]
- Below the Tabs, there are controls for the UI that let you modify the settings, add GUI components (buttons, sliders, and more)
- Next is the UI for your model, and finally an area to enter commands while running the model

Wolf-sheep predation model (Tutorial 1)

- File menu > Models Library > Biology > Wolf Sheep Predation
  - Click on the Open button at the bottom right
- Info tab discusses the main model options
  - Sheep-wolves
    - Wolves feed on sheep
      - Wolves expend energy during each time period (“tick”), gain energy from eating sheep
      - If energy reaches 0, wolf dies
    - Model assumes sheep have an unlimited amount of grass
      - Note that models will make assumptions to keep things manageable
      - Here, the author indicates this is practically equivalent to “plentiful” resources
    - Reproduction rates, energy gained per sheep eaten, initial populations, all configurable

Wolf-sheep predation model

- Sheep-wolves-grass
  - Now sheep have to eat grass to stay alive as well
- To run
  - Click on “setup” button (shows initial locations of wolves and sheep),
  - Click on “go” button (starts simulation)

Command Center

- Console for text input/output
- Enter commands through “observer >” prompt
- Ex: setup sheep-wolf, then enter
  - `setup`
  - `ask patches [set pcolor red]`
  - `Go`
  - `GO ; case-insensitive`

Command Center

- Can interact with individual agents
- Ex: setup sheep-wolf, then right-click near a wolf
  - Pop-up menu gives you information on patches and wolves near where you clicked
  - Find the index for the wolf (ex: 649), then at observer prompt enter:
    - `ask wolf 649 [set color gray]`

Code tab

- Click on Code tab to see code for model
- Support for searching text of code, syntax checking, jumping to definition of a specific procedure, autoindent for editing
- ; for comments – like //
- NetLogo keywords, primitives are colored green, blue, purple
  - No packages/namespaces, so avoid using these identifiers
- Constant values are brown
- NetLogo is case-insensitive

More NetLogo general characteristics

- Distinguishes between void functions (“commands” or “procedures”) and functions with a return value (“reporters”)
- Uses [and] for statement blocks, lists
  - **`end`** used to mark end of functions, procedures
- White space to separate – for not only lists, but function calls as well
  - () not used

Agent types

- Turtles – can move in 2-D world
  - xcor, ycor : coordinates (can have > 1 turtle at same location)
  - Lower values of xcor to the left, of ycor to the bottom
    - Can customize where origin is located
  - min-pxcor, min-pycor, max-pxcor, max-pycor: boundaries of world
  - 3-D world is also supported
- Patches – stationary, but active agents
  - pxcor, pycor: coordinates
- Links – to connect related turtles
- Observer
  - Can create other agents, give instructions to them
  - No location

NetLogo and environment types

- Fully observable (for at least Observer)
- Can support deterministic or nondeterministic environment
- Can be episodic or sequential
- Static environment
- Discrete (does support “fractional ticks” for time)
- Multi-agent
- Known (for Observer)
- Simulated

Contexts

- Observer provides a “global” context
  - Code is associated with observer by default
- Turtles, patches, links behave a little like classes
  - Can message all instances of any type or a single instance
- Turtles can access the patch of their location
  - Turtle is a bit like a friend class of patch

Commands, reporters

- Commands to initiate an action
- Reporters to compute and return some value
- NetLogo provides many built-in commands and reporters
- User can define new commands and reporters as procedures (functions)

```
ask turtles [ right random 360 forward ]
```

NetLogo procedure definition

- Syntax:

```
to proc_name
   Statements
end
```

- To call a procedure
  - For global procedure, just use its name + space-separated parameters
  - For procedures of turtles/patches/links, use **`ask`**
    - Ex:

```
ask turtles [
  right random 360 ; set direction of turtle
  forward 1 ; move forward 1 patch
]
```

Specifying individual agents

- Turtles are created with *who* numbers 0, 1, 2, … in order
  - `ask turtle 2 […]` messages only 3rd turtle
- Patches identified by `pxcor pycor`
- Links identified by who numbers of linked turtles

Agentsets

- Set of agents of one type. NetLogo provides the types:
  - `turtles` returns agentset of all turtles
  - `patches` returns agentset of all patches
  - `links` returns agentset of all links
  - Can specify subsets of any of these types
- Can do operations with all members of any agentset
- Can select specific agents
  - Ex: `turtle 0` produces first turtle
  - Ex: `patch 0 0` produces patch at origin