



Agent Based Modeling using Netlogo Desktop (GIS extension)

“NetLogo is a multi-agent programmable modeling environment. It is used by many tens of thousands of students, teachers and researchers worldwide. It also powers HubNet participatory simulations. It is authored by Uri Wilensky and developed at the CCL. You can download it free of charge. You can also try it online through NetLogo Web.”



NetLogo is a procedural type programming software that creates models based in Evolutionary Game Theory (EGT) to aid in prediction from interrelated criterion. Agent-based modeling (ABM) is a methodology that uses input parameters (behaviors) to any number of individual units and tracks their movement in a constructed environment. Before taking this workshop it is suggested that you peruse:

Introduction to NetLogo: <https://subversion.american.edu/aisaac/notes/netlogo-basics.html>

Programming Guide : <https://ccl.northwestern.edu/netlogo/docs/programming.html>

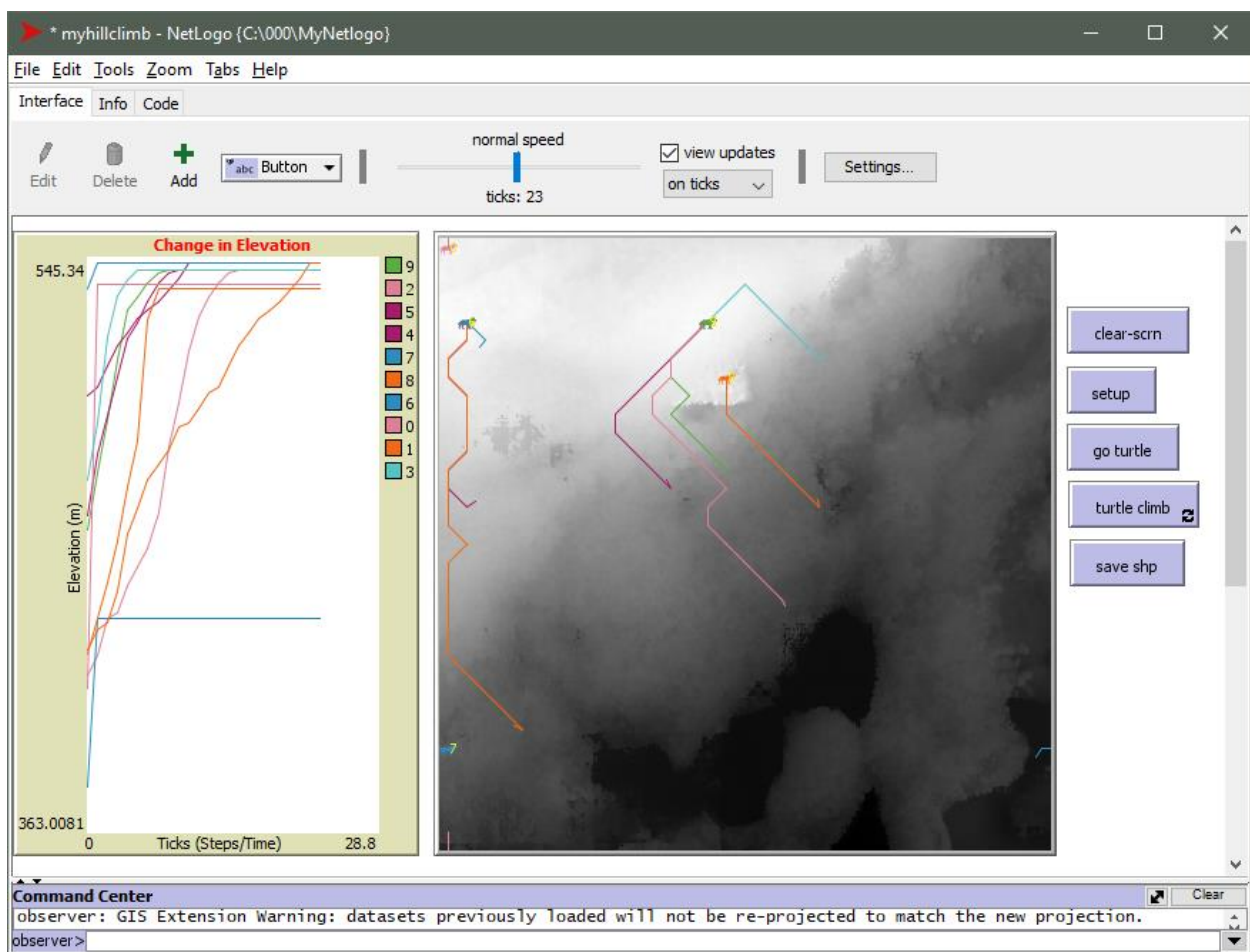
There are many other sites that explain the finer points of NetLogo as it is quite popular for creating many types of prediction models. Also, NetLogo has many samples available within the software under the File>Models Library. Since, the code and layout is fully open one can quickly get an idea of how this procedural code works. Another prerequisite for this workshop is a comfortable understanding of various scripting languages as this will facilitate setting up and trouble shooting NetLogo's procedural coding.

This tutorial is a primer for demystifying NetLogo by building a hill climb model using the GIS extension. The model created within is based on one of the sample models and it is extended to include some GIS principals. Essentially this is simplified multiple-criteria decision analysis using agents to traverse upwards a raster in a least cost manner using what is essentially a greedy algorithm.

Setup

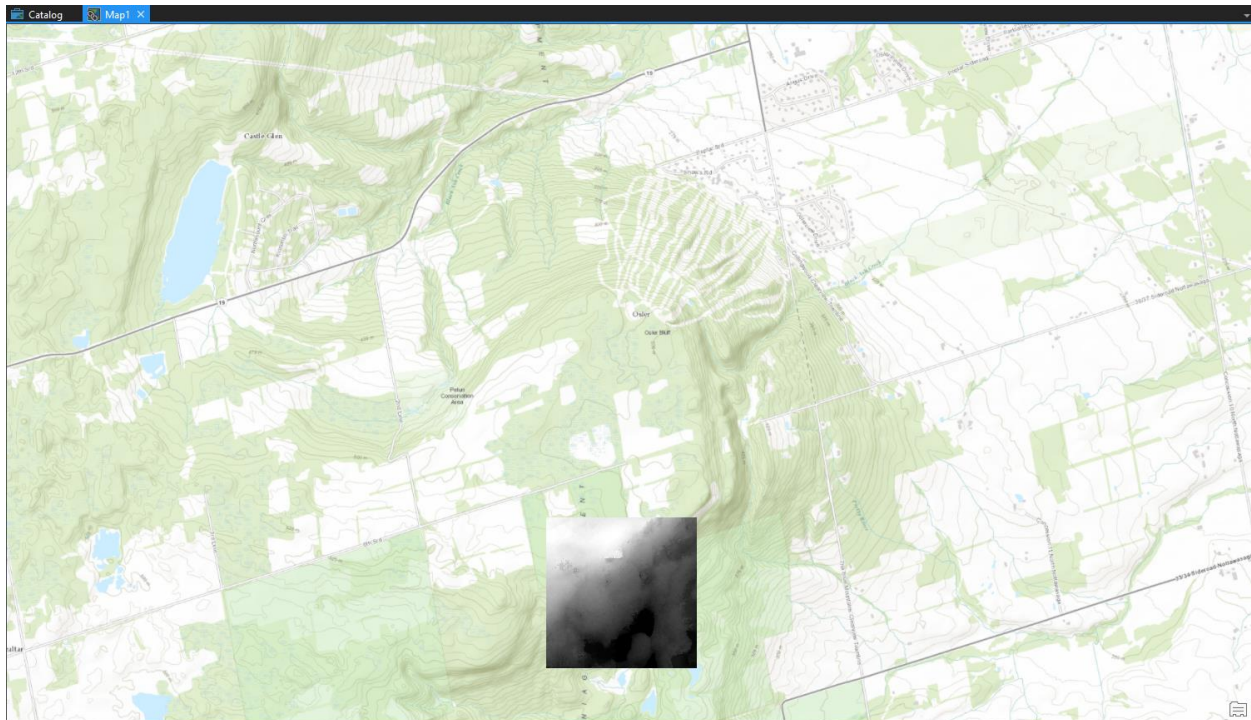
Download and install the latest version of NetLogo specific to your operating system. This software can be freely downloaded by anyone. There is a web version also available but for the purposes of this tutorial only the desktop version will be used.

In this guide, we are going to learn how to implement the GIS extension and some spatial context utilizing the simplified procedural scripting environment in NetLogo. An understanding of GIS file types, projections, file extensions, programming with scripting languages, implementing variables, extensions, and software building environments is a must to get the most out of this tutorial and product.

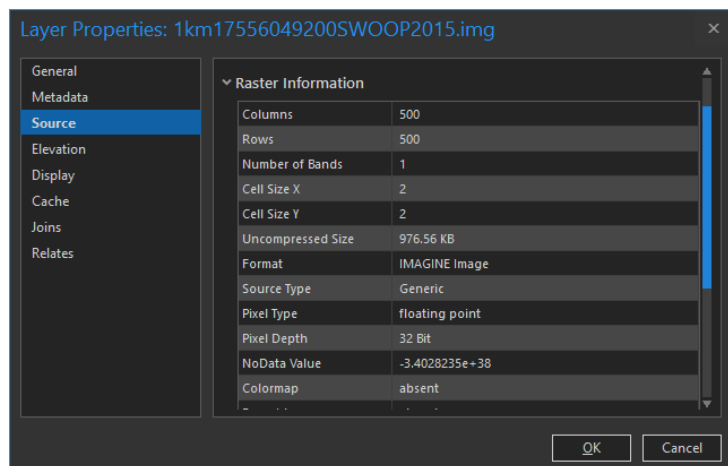


File Preparation

The GIS files that are required for this workshop is an ASCII raster and its projection. NetLogo can also use a GRID raster but I have not had any luck with them. The raster file will provide the base or cost (Patch) the agents (turtles) will traverse. Using GIS Software convert, in this case a DEM, into an ASCII raster. ArcGIS has a tool called 'Raster to ASCII' that will produce the appropriate .asc and associated files.



This is the source data for the original raster make a note of the number of Columns, Rows, Cell Size x, and Cell Size y. When converting to an ASCII raster some of the data may change slightly due to the algorithm used. Since the raster is only being used as a base cost slight variations are negligible.



Once the ASCII raster is prepared it is imperative to change the projection file to Well-Known Text (WKT) format, which is how the projection file is written for shape files. This done because NetLogo's GIS extension only reads this projection format. While it is not calibrated for all projections it will understand most popular types and there is a list on the GIS:Extensions wiki page. Below is an example of a .prj file that is created by the raster to ASCII conversion process (From) and the Well-Known Text (WKT) text file that NetLogo expects.

From:	To:
Projection UTM Zone 17 Spheroid CLARKE1866 Units METERS Zunits NO Parameters	PROJCS["NAD27(76) / UTM zone 17N",GEOGCS["NAD27(76)", DATUM["D_NAD_1927_Definition_1976", SPHEROID["Clarke_1866",6378206.4,294.9786982138982]], PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin",0], PARAMETER["central_meridian",-81], PARAMETER["scale_factor",0.9996], PARAMETER["false_easting",500000], PARAMETER["false_northing",0],UNIT["Meter",1]]

Getting Started

Since everyone grasps programming languages differently this workshop will be structured around an already functioning analysis. As mentioned above there are many models in the Models Library and the Modeling Commons to borrow code snippets in order to assemble your own analysis. The premise is to walk through the code with emphasis on bringing in spatial data, running an analysis on it and exporting that analysis to a csv file (currently there is not the ability you to export as a shapefile). I found that importing a raster into NetLogo and making it the cost base for a hill climb analysis was slightly more difficult to find useful examples.

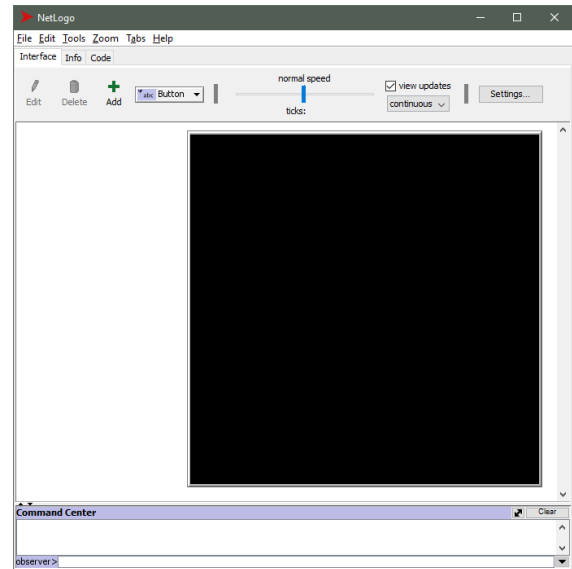
1. Open NetLogo

- This first part will walk you through placing a button on the Interface screen and when it is clicked calling the procedure behind it on the Code screen.



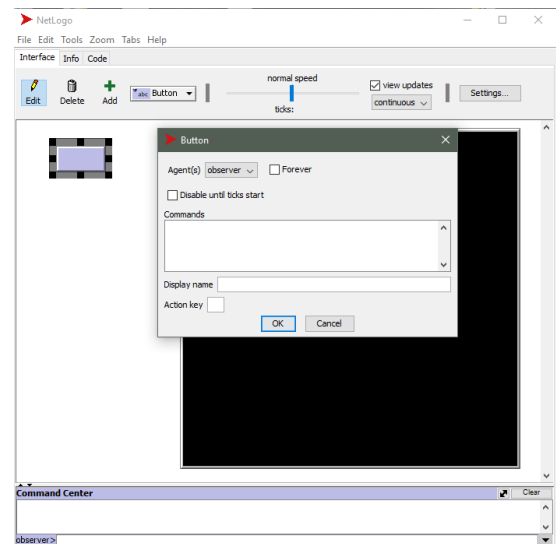
2. NetLogo starts as a New Project.

- a. Tabs
 - i. Interface
 - ii. Information – description
 - iii. Code
- b. Tool Bar
 - i. Add + Dropdown – a selection of interface objects.
- c. Speed Control – controls the speed of the agents.
- d. Ticks – marks the completion of one instruction cycle
- e. Settings – sets options for the Patch
- f. Patch – model window (essentially raster cells with a size dependent on the model window size and a location that is Cartesian with 0,0 in the centre.
- g. Command Centre - enter commands or directions to a model
 - i. observer> - this is where commands are entered for specific objects in the program. Click on observer> and a dropdown appears for the 4 different agents: observer, turtles, patches and links.



3. Placing your first observer agent:

- a. Click the “Button” in the dropdown and place it beside the patch.
- b. Leave the Agent as observer and create a command that calls a clear the patch procedure. This a good and easy procedure to code that allows one to reinitialize anything they write afterwards. E.g. ‘clear-scrn’ this will also be the display name unless renamed in the Display name text box.
- c. Under the Code tab start a command procedure with ‘to’ and finish with ‘end’.



```
to clear-scrn
  clear-all
  reset-ticks
end
```

- d. This is a great procedure to have as it ensures you start with a clean slate (patch, as it were) every time you click it. Also, to comment out code the semicolon is used.

The Sample Code

1. On to the coding – this will be a cursory overview of NetLogo's procedural language.
2. Start with the **extensions** that will be need for the project. In this case we will need a GIS (for bringing in a raster) and csv (for exporting the outcome).
3. Create the **globals** (global variables)
 - a. Elevation-dataset – container for the DEM raster
 - b. Xc and yc – variable for randomizing turtle location
4. Use the **patches-own** command to give the patches agent a local value you supply in the form of a DEM raster from the **globals** global variable.
 - a. Patches agent variable name [elevation]
 - b. If these aren't **set** they begin with a value of 0.
5. Use the **turtles-own** to set a variable (In this case true or false) that is unique to the turtle's agent. In NetLogo **turtles** are agents that explore the modeled environments you give them. Since the ticks (steps) we are asking the turtles to perform are individual ones we need to loop them. This variable is set to 0 and is referred to by a later procedure in a simple 'if' statement. This is from the Hill Climbing model in the Model Library.
6. Here we see the clear screen procedure which is called by the button created in the beginning of this workshop.
7. The next procedure that is common to all models in NetLogo is a setup script that is called by a setup 'Button' placed on the interface. Command: setup - This **to** procedure sets up the world.

```
extensions [gis
csv]
globals [
  elevation-dataset
  xc
  yc
]

patches-own [elevation]

turtles-own [
  peak? ;; indicates whether a turtle has reached a "peak",
        ;; that is, it can no longer go "uphill" from where it stands
]

to clear-scrn
  clear-all
  reset-ticks
end
```

- a. `set-current-directory` – points NetLogo at a chosen folder. In this code it is only used to save the .csv file but it is good practice to have a single working folder.
- b. `gis:load-coordinate-system` – is an object in the GIS Extension that can read .prj file. This is required to setup the space for the raster being used.
- c. `set elevation-dataset gis:load-dataset` - This fills the global variable 'elevation-dataset' with the chosen raster.
- d. `gis:set-world-envelope-ds gis:envelope-of elevation-dataset` – this the envelope of the NetLogo world be the raster space given it.
- e. `gis:paint elevation-dataset 0` - Paints the given raster data to the NetLogo drawing layer. If this is not set the raster will be summarized into the patches (super low resolution).
- f. `gis:apply-raster elevation-dataset elevation` – Copies elevation values from the given raster to the patch variable 'elevation'.
- g. `let min-elevation gis:minimum-of elevation-dataset and let max-elevation gis:maximum-of elevation-dataset` – gets the min and max of the raster elevation values.
- h. `ask patches`

```

to setup
  set-current-directory "C:\\MyNetLogo"
  gis:load-coordinate-system "C:\\ testasc.prj"
  set elevation-dataset gis:load-dataset "C:\\ testasc.asc"
  gis:set-world-envelope-ds gis:envelope-of elevation-dataset
  gis:paint elevation-dataset 0
  gis:apply-raster elevation-dataset elevation
  let min-elevation gis:minimum-of elevation-dataset
  let max-elevation gis:maximum-of elevation-dataset
  ask patches
  [ if (elevation <= 0) or (elevation >= 0)
    [set pcolor scale-color black elevation min-elevation max-elevation]]
end

```

This is a specific setting of the 'elevation' variable to scale the colour of the patches based on values of the raster cells from low to high.

8. Now that the world is our cost raster we need some turtles (Agents) setup to roam across it. This **to** procedure is called by a 'Button' (in this case go turtle) placed on the interface. Command: goturtle - This creates turtles and randomly places them in the world.

- create-turtles 10 – This routine creates 10 turtles (Agents)
- Within the square brackets '[]' is where things about are changed (Arguments) like shape, location, weather they leave tracks, etc. for the turtles here.
- set shape – there are many shapes in the Shape Library and Shape Editor.
- xcor and ycor – is the x and y location of the Agents in NetLogo. xcor is for turtles and pxcor is for the location of patches for example. This sets the global xc and yc variables to random and uses those variables to place the 10 turtles in the raster world. If the set xcor -800 and set ycor -800 are used then all 10 turtles will sit in the bottom left of the patch.
- set peak? false – removes any possible value to the peak? global variable.
- set color color – sets each turtle to a random color. Coincidentally this colour matches the corresponding line in the plot.
- pen-down – draw a trail from tick to tick movement.
- reset-ticks – ensures the model starts from 0.

```
to goturtle
  create-turtles 10 [
    set shape "wolf"

    ;Turtle location randomizer
    setxy random-ycor random-ycor
    set xc xcor
    set yc ycor

    ;set xcor -800
    ;set ycor -800

    set peak? false
    set color color
    pen-down
  ]
  reset-ticks
end
```

9. The world is built and the turtles are in their places. Now we want them to climb. This is done by using the Uphill built in routine. What this does, is, it has each turtle look at the 8 patches (patches are larger than the raster cells, see previous) and determines which is lighter in colour (setup by min max in the raster) and moves there (1 tick). This procedure is also called by a button on the interface. This button is setup the same as the others except it has the 'Forever' checkbox checked. This is a loop that runs until the Agent cannot see a lighter colour. Much like a kernel does in interpolation.

- if all? turtles – is a reporter that sets the peak? variable to true if the turtles cannot find a lighter colour patch to move to.
- ask turtles – this uses the 'uphill' routine setting the 'old-patch' and 'patch-here' arguments for the Agent's previous and current moved to location.
- If – queries if the current location and the moved to location is the same. If yes set peak? to true
- tick – mark a step

```
to turtleuphill
  ; stop when all turtles are on peak
  if all? turtles [peak?]
  [ stop ]
  ask turtles [
    ; remember where we started
    let old-patch patch-here
    ; to use UPHILL, the turtles specify a patch variable
    uphill pcolor
    ; are we still where we started? if so, we didn't
    ; move, so we must be on a peak
    if old-patch = patch-here [ set peak? true ]
  ]
  tick
  ask turtles [ set label who ]
  ; all the turtles now are labeled with their who numbers
  ask turtles [ set label-color yellow ]
end
```


- e. ask turtles [set label who] – ‘who’ is an in built flag that gives the number of the Agent.
- f. ask turtles [set label-color yellow] – sets the ‘who’ label to yellow.

10. The final button calls a save routine that uses the csv [extension](#).

- a. csv:to-file – this is a ‘primitive’ that exports into a .csv file specific traits of the Agent. In this case the list, heading, patch x and y coordinates and the raster elevation under the turtle’s location.

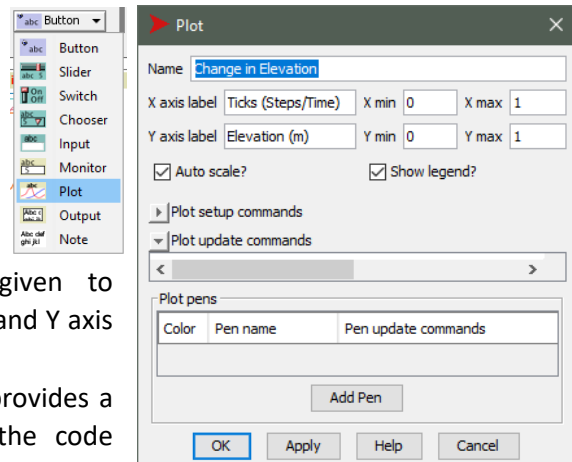
```
to saveLine
; we use the 'of' primitive to make a list of lists and then
; use the csv extension to write that list of lists to a file.
;csv:to-file "C:\\turtles.csv" [ (list heading pxcor pycor elevation) ] of turtles

export-all-plots "C:\\ turtles.csv"
end
```

- b. export-all-plots – this saves the chart/plot data in a .csv file. Described below.

11. Unfortunately NetLogo cannot currently export .shp files of the completed Agent models. In an effort to visualize the change in elevation of the turtles a plot was created.

- a. A line plot was chosen to represent the change in elevation for every tick. This creates a cross-section of every turtle’s journey up the raster.
- b. In the Plot dialogue box inset a Name and X and Y Axis Label.
- c. Auto scale? – check this box. This scales the Plot’s X and Y within the window given to accommodate ‘on the fly’ updates in the X and Y axis as the models are running.
- d. Show legend? – Also check this box. This provides a legend for your Plot that, if asked in the code automatically connects the turtles to their own graph lines (more on this below)



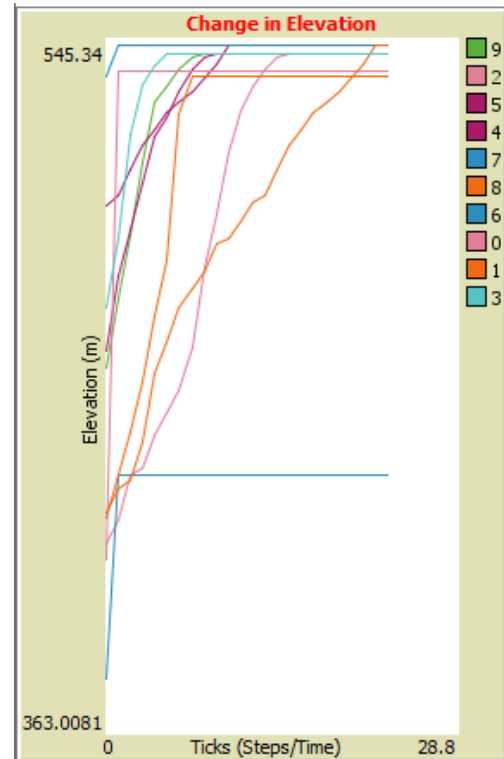
12. To tie the Plot ot the Patch a little bit of code that uses the values in the vairables created on the code tab. This code snipet is placed under the ‘Plot update commands’ dropdown.

- a. The first line sets the graph’s Y axis range linked to the plots minimum and maximum of the elevation-dataset (raster).
 - i. `set-plot-y-range gis:minimum-of elevation-dataset gis:maximum-of elevation-dataset`
- b. The second set ties the plot lines to the Agents (turtles). It “ask’s” the turtles what number they are, colour and their elevation value (vairable from the the elevation-dataset raster) at every tick and plots it. The code below is written the same as in the Code tab.

```
i. ask turtles [
  create-temporary-plot-pen (word who)
  set-plot-pen-color color
  plotxy ticks elevation
]
```

This code creates a temporary pen sets the colour and plots the requested variables on the x and y axis'.

A nice feature in NetLogos' integrated development environment (IDE) is that the Plot is already connected to the Agents in the Patch. This is shown when only 'color' (random) for the turtles in the patch code and the corresponding colour is plot in the line graph and set in the legend. Also, the 'who' number is used without writing code to assure the right Agent is plotted from the Patch.



Wrapping Up

This is a quick look at NetLogo's agent based modeling abilities. This exercise is focused on being more quick-start guide to the difficult to find GIS examples. How to bring in a cost raster, how to run Agent's on it, how to use and retrieve variable values, and how to export model data in the most useful way.

Created 2019 by Markus Wieland
 UNIVERSITY OF WATERLOO
 GEOSPATIAL CENTER
<https://uwaterloo.ca/library/geospatial/>