Introduction to NetLogo

Christine Harvey

June 16, 2015

Abstract

Introduction to model development in NetLogo including the GUI, basic coding principles, and terminology. Includes a walkthrough of editing a model to increase complexity and add monitoring capabilities. Intended audience is those new to NetLogo or with a minimum amount of programming experience.

1 Introduction

If you don't already have NetLogo installed, go to `https://ccl.northwestern.edu/netlogo/download.shtml`, to download.

Go to `File` and then `Model Library` which should open up a interface with a list of possible example models to review.

Browse to the `Biology` section and select the `Wolf Sheep Predation` Model and open it. Once opened, the view should look like Figure 1

2 Terminology

Some language used in NetLogo is specific to Agent Based Modeling as well as the programming language itself:

Agents: The individuals or organisms in the model. The behavior of a class of agents is determined by a modeler-defined set of rules. In NetLogo

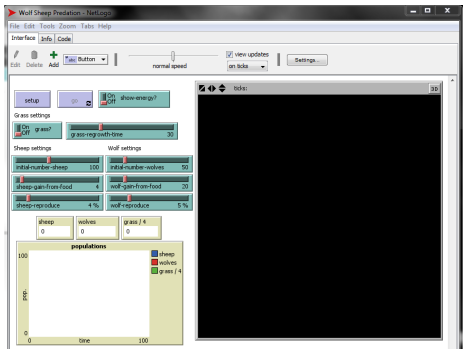

Figure 1: Initial view of the Wolf Sheep Predation Model

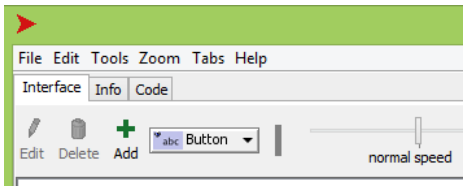

Figure 2: NetLogo tabs view.

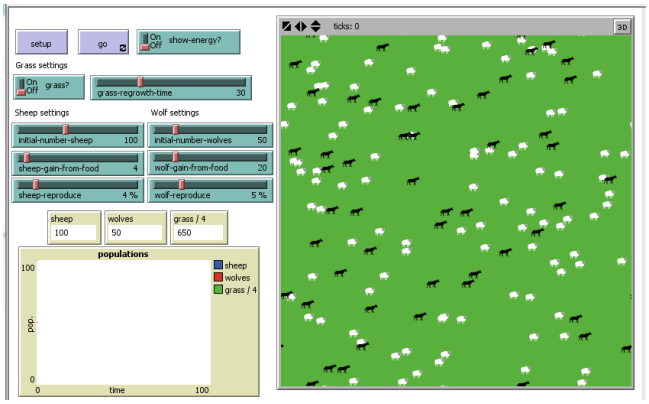

Figure 3: Initialized view of the Wolf Sheep Predation Model

there are two types of agents: the mobile agents called "turtles" and the stationary/background agents known as patches.

NetLogo Tabs: NetLogo has three tabs at the top of its window. An Interface tab, an Information tab, and a Procedures tab.

- **Interface Tab:** The Interface is the main tab where you interact with your model and see your agents behavior and visualize data. It can contain buttons, slider bars, switches, monitor boxes, and plots in addition to the "world" the agents live in (the larger black window).
- **Information Tab:** The Information tab simply contains textual information for the user on how to best understand and use the model.
- **Code Tab:** The Code tab is also a textual environment that contains the rules that determine the behavior and attributes of the agents in the model.

3 Run a Model

Click the `setup` button to initialize the model which should create the display seen in Figure 3. This completely starts up the model, initializes all parameters and agents that will be interacting in the model.

Click the `go` button to launch the model, press this button again to pause. This will cause the model to run continuously until it is pressed again to stop of pause the model.

Stop the model and once it has been stopped, you can right-click on a sheep or a wolf to inspect the agent, there are many options here, you can inspect the element further or choose to follow the particular agent.

Experiment with the sliders and the monitors available on the screen to understand the controls.

The Code tab gives the collection of commands and procedures that define the rules for behavior and attributes of each agent in relation to the buttons, sliders, graphs, etc. that are found on the Interface tab. This is where the modeling takes place and it will be the focus of building your own predator-prey model in the next section.

4 Building a Simple Model

We want to simulate people in a banking environment.

People in the environment have a wallet, savings account, and loans. When a person is on the same patch as someone else, they will either give the other person zero, two or five dollars based on a randomly generated number. After the transaction, the person sorts out their money with the bank:

- If the person has a positive balance in their wallet, they put this money into their savings.
- If the person has a negative balance, they will use money in their savings to pay off the balance.
- If the savings account is empty and the person has a negative balance, they will take out a loan from the bank if funds are available to borrow. If not, they have to maintain their negative balance.
- If someone has money in savings and money borrowed from the bank, they will pay off as much of the loan as possible with their savings.

4.1 Setting Up the Interface

Open up NetLogo, go to the `File` menu and select `New`. We will start by adding the two button used in almost every NetLogo program, "setup" and "go".

- Creating the "Set Up" Button:
  1. Choose the button option from the pull-down menu and then click the add `Add` button and left click in the white portion of the Interface tab to position the new button in the window. This will pop open a window like the one below:
  2. In the "Commands" window, type the word "setup". This will be the name of the procedure that will define how the world should be initially populated. (No blank spaces).
  3. In the "Display Name" window, type the name of the button as you would like it to appear to the user, for example "Set Up".
  4. Click `OK` when done.

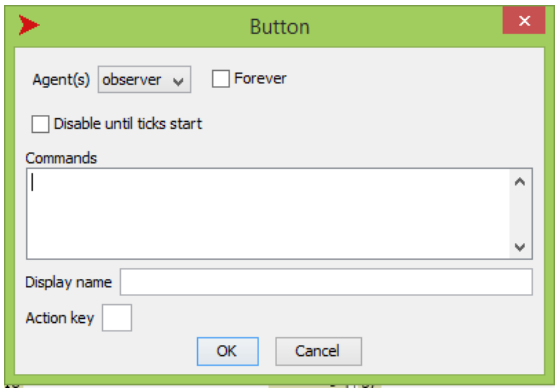

Figure 4: Adding a NetLogo button.

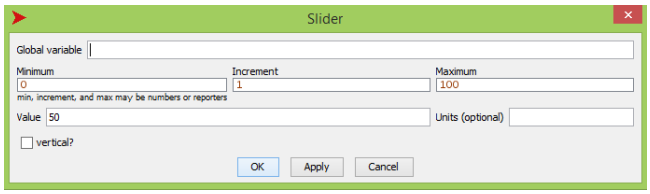

Figure 5: Adding a NetLogo Slider.

- Creating the "Go" Button: In a similar manner, add a button that has associated with it the command: go Note that if you would like hitting the go button to cause the simulation to run continuously (rather than for just one step in time), click the Forever button at the top right portion of the pop-up window.

- We will have two slider bars in the model for global variables. Adding slider bars for the parameters:

    1. Choose the slider option from the pull-down menu and then click the add `Add` button and left click in the white portion of the Interface tab to position the new button in the window. This will pop open a window like the one below:
    2. In the "Global Variable" line, type the single word used to represent the parameter, "People". Note once again that this should not have any blank spaces in the name. This is the name of the parameter representing the number of people in the model.
    3. NetLogo gives default minimum, maximum, and initial values for this slider bar, as well as an increment measurement. For People, we've set the minimum value to 0, the maximum value to 200, the initial value to 50, and the step size at 1.0.
    4. Click `OK` when done.

Repeat this process the "Reserves" global variable, with a minimum of 0, maximum of 100, incrementing by 1, and a default value of 52.

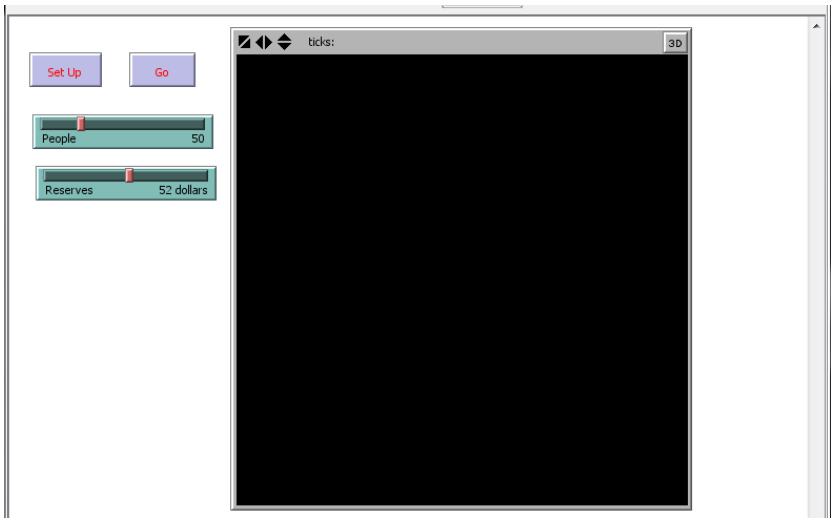

Figure 6: The NetLogo Interface so far.

Your Interface tab should now look like Figure 6. Note that the `Set Up` and `Go` buttons are in red because we have no yet written the code associated with clicking these buttons.

4.2 Writing the Procedures

In order to write the procedures for your model, it is often helpful to open a NetLogo model from the Models Library that shares some of the characteristics of your model and look at it's Procedures tab for ideas. NetLogo also has a good set of tutorials, a user's manual, and a dictionary of NetLogo commands available under the Help menu at the top of your NetLogo window.

4.3 Turtles and Attributes

As mentioned earlier, mobile agents in NetLogo are referred to as turtles.

We can specify attributes associated with our turtles. This is accomplished through the use of the turtles-own declaration, also placed before any procedure definitions. Attributes are helpful when we need to keep track of a quantity associated with or the status of a turtle at any point in time. As each turtle is handled individually, it is possible that two different turtles can have different attribute values.

For instance, we've mentioned that our person needs to have a wallet and a savings account. Thus, we need to know how much money each individual person has, which will depend on the amount of money in their savings account, wallet and loans. We can declare the attributes named wallet, savings, and loans to be associated with each person by the following command: turtles-own [wallet savings loan]. The code needed to start our model could look like this:

```
turtles-own [
  savings
  loans
```

```
  wallet
  wealth
  customer
]
```

4.4 The `setup` Procedure

After defining our turtles and their attributes, we can write the code that will define the actions NetLogo will carry out when the setup button is pushed. As with all user-defined procedures, the setup procedure will have the form:

```
to procedure-name
        code for actions to take
    .
    .
    .
end
```

We want our setup procedure to populate our world with the desired number of people, placed randomly in the coordinate plane. We'd also like to set up any attributes associated with each person when it is created. In addition to the user-defined attributes we noted above with our turtles-own, there are color attributes, size attributes, shape attributes, and location associated with each turtle. We will make use of some of the built in NetLogo commands:

- `clear=all`: Clears the world of any previous turtles and patches
- `set-default-shape turtle ''shape-name''`: Changes the default shape for turtle of breed breed to be portrayed using the shape shape-name found in the Turtle Shapes Editor.
- `crt number`: NetLogo creates number turtles of the type breedname with default color (blue), size (100% normal value), shape (turtle), and location (0,0).
- `setxy random-xcor random-ycor`: Sets the coordinates of a turtle to a randomly generated x and y coordinate
- `set attribute value`: Sets the value of the attribute names attribute to have the value, value
- `random-float value`: provides a randomly generated floating point value between 0 and pos-max-value.

To accomplish what we want from our `setup` procedure, we could use the following code:

```
to setup
  clear-all
  ask patches [set pcolor black]
  set-default-shape turtles "person"
  crt people [setup-turtles]
  reset-ticks
```

```
end

to setup-turtles  ;; turtle procedure
  set color blue
  setxy random-xcor random-ycor
  set wallet (random 10) + 1 ;;limit money to threshold
  set savings 0
  set loans 0
  set wealth 0
  set customer -1
end
```

Note that actions that have more than one line of code are grouped using square brackets and the semicolons used to indicate the start of line of comments. Carefully commented code can be a great help to the user who chooses to adjust some of the rules in your model, and it helps you remember why you may have chosen to include certain commands or procedures!

Also note that not all turtle shapes are loaded to the default library of the Turtle Shapes Editor. To see what shapes are available, go to the Tools menu and select Turtle Shapes Editor. If you can't find an image that suits your purpose you can try importing an image from the library or another model or creating your own image.

Once you have this typed into your procedures tab below the turtles-own declarations, you can return to your Interface tab and click on the Set Up button to see if what you intended to happen indeed happens in your world. If there are any errors in code syntax, NetLogo will notify you with an error message at the top of your Procedures tab.

4.5 The `Go` Procedure

The `go` procedure will dictate the behavior of the turtles over time, and will essentially specify the code to accomplish the behaviors outlined. As NetLogo may not have built-in commands or procedures that cause a turtle to "Check to see if you are near another person", you will have to define your own procedures to carry out these actions. When actions get complicated it is common to make subprocedures to help make steps clear. Accepted syntax for user defined procedures and subprocedures is give it a descriptive name using dashes to separate words. For instance, you may want to create a subprocedure that tells a person how to check to see if it is on the same space as another person and perform a transaction. You may name that do=business which is called in a procedure, such as the go procedure, and defined below the go procedure. Here's a snippet of code that illustrates this point:

```
to go:
  .
  .
  do-business
  .
  .
 end
```

```
 to do-business:
   .
   .
  <describe actions involved in doing busniess>
   .
   .
end
```

Here is how we could possibly outline the behavior of the person at the time step:

```
to go
  ask turtles [
    do-business
  ]
  tick
end
```

The above code directs the person to "do business" on every third tick. You'll notice this requires the definition of a subprocess, do-business.

```
to do-business  ;;turtle procedure
  rt random-float 360
  fd 1
end
```

The above code has the person rotate direction and move forward one step.

Now go back to your Interface tab and hit the go button to see what happens. If everything was typed in correctly, you should be able to see movement of the turtles. Because it's hard to notice the change in numbers of our turtles, we use plots and monitor boxes to provide us with a graphical display of this information.

4.6 Presenting Data with Graphs and Monitors

We want to monitor certain attributes of the data such as the number of rich, poor and middle-class people in the environment. To do this, we will add in global variables to the top of the code:

```
globals [
  rich
  poor
  middle-class
]
```

These variables need to be initialized in the `setup` procedure. Alter the setup procedure to run the `initialize-variables` procedure, which is defined below:

```
to initialize-variables
  set rich 0
  set middle-class 0
  set poor 0
end
```

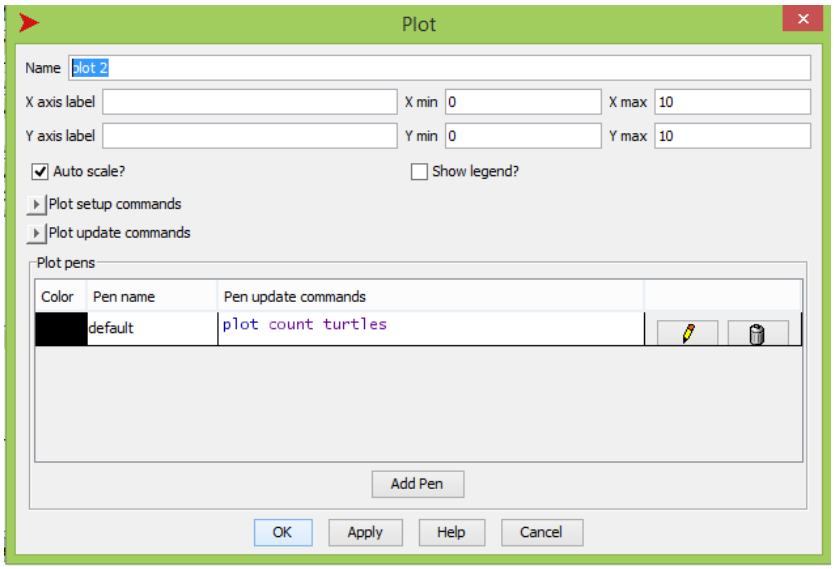

Figure 7: Screen to add a plot to the interface.

Now we need to compute these numbers during the `go` procedure. Add the following code to the top of the `go` procedure, before the "ask turtles" command:

```
;; tabulates each distinct class population
set rich (count turtles with [wallet > 8])
set poor (count turtles with [wallet < 2])
set middle-class (count turtles - (rich + poor))
```

This will classify everyone with more than $8 in their wallet as rich, everyone with less than $2 as poor, and everyone else as middle-class.

In order to create a plot, follow the steps below:

1. First go to the Interface tab and under the pull-down tab, select the Plot option and click on a blank location in the Interface. This will pop open a plot editor box. See Figure 7 for an example.

2. You will need to specify the plot names and labels for your axes, name the plot, "Income Dist" and label the x-axis Time, and the y-axis people.

3. Add your "Plot Pens", each plot pen corresponds to a different curve/line on the plot, following the format shown in Figure 8

4. Once you've defined all of your Plot Pens, click `OK`.

5. Press the `setup` button to initialize the model and then the `go` button to walk through several steps. You can see that the plot updates at each time step, but does not change. This is because the agents aren't yet interacting with each other to share and give money.

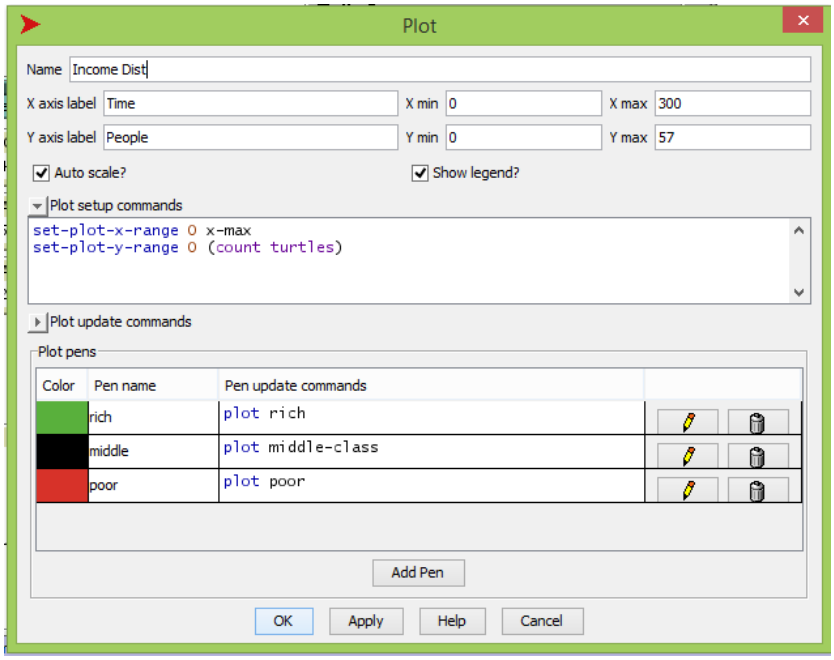

Figure 8: Plot the income distribution between rich, middle-class and the poor.

4.7 Adding Banks

The agents in the model need to interact with a bank, which they can have a savings account with, as well as save and borrow money. To create these banks, update the globals section to look like below:

```
globals [
  rich
  poor
  middle-class
  bank-loans
  bank-reserves
  bank-deposits
  bank-to-loan
]
```

We need to set up these variables, so add a `setup-bank` command to the setup procedure, following the turtle creation. This method belongs after the setup-turtles command.

```
to setup-bank ;; initialize bank
  set bank-loans 0
  set bank-reserves 0
  set bank-deposits 0
  set bank-to-loan 0
end
```

Now that we have a bank set up, we can begin modeling how people interact with one another. Begin by programming the `do-business` procedure, edit the procedure to look like the following:

```
to do-business  ;;turtle procedure
  rt random-float 360
  fd 1

  if ((savings > 0) or (wallet > 0) or (bank-to-loan > 0))
    [set customer one-of other turtles-here
     if customer != nobody
     [if (random 2) = 0     ;; 50% chance of trading
           [ifelse (random 2) = 0
              [ask customer [set wallet wallet + 5] ;;give 5
               set wallet (wallet - 5) ] ;;take 5 from wallet
              [ask customer [set wallet wallet + 2] ;;give 2
               set wallet (wallet - 2) ] ;;take 2 from wallet
           ]
       ]
     ]
end
```

The above code declares that if the person's savings or wallet is positive, or if the bank can give loans, then the person looks for someone on their location. If there is someone else on their patch, they either do nothing, or give the other person $5 or $2.

If a transaction is made, the wallet is decremented by the proper amount and the customer's wallet increases by the same amount.

If we go to the `Interface` tab now and click `setup` and `go` several times, we can see the rates for rich, poor, and middle-class are all mixing.

4.8 Balance the Books

We want to add two more procedures to our model to balance the books and bring together the entire financial system. We want a method to balance-the-books and one to color the people according to their wealth.

First, edit the `go` procedure to look like this:

```
to go
  ;;tabulates each distinct class population
  set rich (count turtles with [savings > rich-threshold])
  set poor (count turtles with [loans > 10])
  set middle-class (count turtles - (rich + poor))
  ask turtles [
    ifelse ticks mod 3 = 0
      [do-business] ;;first cycle, "do business"
      [ifelse ticks mod 3 = 1  ;;second cycle
         [balance-books
          get-shape]
         [] ;;third cycl
      ]
```

```
  ]
  tick
end
```

Now we need to add in the code for balance-books and get-shape. The balance-books procedure first checks the balance of the turtle's wallet and either puts a positve balance in the savings or tries to get a loan to cover a negative balance. If it can't get a loan, then it maintains the negative balance until the next round.

```
to balance-books
  ifelse (wallet < 0)
    [ifelse (savings >= (- wallet))
       [withdraw-from-savings (- wallet)]
       [if (savings > 0)
          [withdraw-from-savings savings]

        set temp-loan bank-to-loan  ;;temp-loan = amount available
        ifelse (temp-loan >= (- wallet))
          [take-out-loan (- wallet)]
          [take-out-loan temp-loan]
       ]
     ]
    [deposit-to-savings wallet]

  if (loans > 0 and savings > 0)     ;; enough in savings
    [ifelse (savings >= loans)
       [withdraw-from-savings loans
        repay-a-loan loans]
       [withdraw-from-savings savings
        repay-a-loan wallet]
    ]
end
```

Add the attribute `temp-loan` to the turtle attributes.

Several procedures are listed here which need to be added to the model, including withdraw-from-savings, take-out-loan, and repay-a-loan.

The code for these procedures is simple and can be added to the end of the model:

```
to deposit-to-savings [amount] ;;fundamental procedures
  set wallet wallet - amount
  set savings savings + amount
end


to withdraw-from-savings [amount] ;;fundamental procedures
  set wallet (wallet + amount)
  set savings (savings - amount)
end
```

```
to repay-a-loan [amount] ;;fundamental procedures
  set loans (loans - amount)
  set wallet (wallet - amount)
  set bank-to-loan (bank-to-loan + amount)
end


to take-out-loan [amount] ;;fundamental procedures
  set loans (loans + amount)
  set wallet (wallet + amount)
  set bank-to-loan (bank-to-loan - amount)
end
```

Now we need to set up the to `get-shape` routine, add the following code to the end as well:

```
to get-shape  ;;turtle procedure
  if (savings > 10)  [set color blue]
  if (loans > 10) [set color red]
  set wealth (savings - loans)
end
```

Finally, now that we have the savings and loans implemented, change the lines in the `go` procedure to the following:

```
  set rich (count turtles with [savings > 10])
  set poor (count turtles with [loans > 10])
```

4.9 Balance the Banks

The final part of the model to be added is to set aside the required amount from liabilities into reserves, regardless of outstanding loans. This may resutl in a negative bank-to-loan amount, which means that the bank will be unable to loan money until it can set enough aside to account for reserves.

```
to bank-balance-sheet ;;update monitors
  set bank-deposits sum [savings] of turtles
  set bank-loans sum [loans] of turtles
  set bank-reserves (reserves / 100) * bank-deposits
  set bank-to-loan bank-deposits - (bank-reserves + bank-loans)
end
```

Finally, update the `go` procedure to use this method.

```
to go
  ;;tabulates each distinct class population
  set rich (count turtles with [savings > rich-threshold])
  set poor (count turtles with [loans > 10])
  set middle-class (count turtles - (rich + poor))
  ask turtles [
    ifelse ticks mod 3 = 0
      [do-business] ;;first cycle, "do business"
      [ifelse ticks mod 3 = 1  ;;second cycle
```

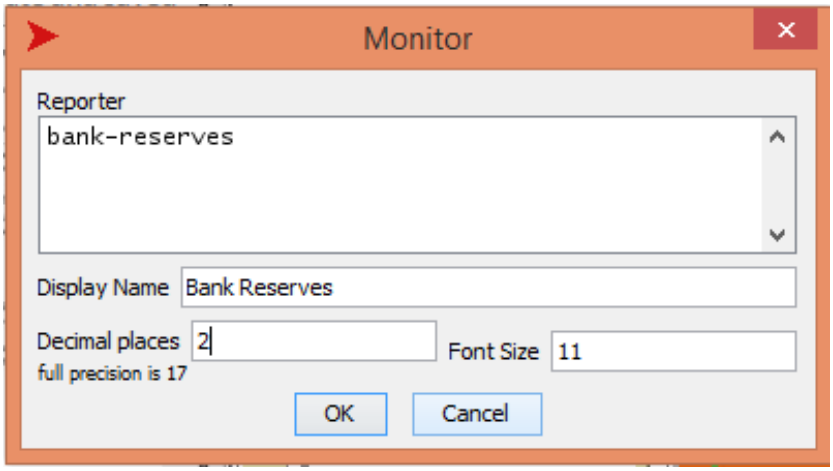

Figure 9: Screen to add a monitor to the interface.

```
            [balance-books
             get-shape]
            [bank-balance-sheet] ;;third cycle, "bank balance sheet"
        ]
    ]
    tick
end
```

5 Additional Monitoring

This final section walks through how to add in additional monitoring tools for the model. For this simulation, we also want to track the data in the model.

5.1 Reporters

Reporters are small displays of what's happening in the data, for this model, we want to display the total money, total savings, total loans, total in wallets, bank reserves and bank-to-loan.

To add a reporter, find the Monitor option from the drop-down menu, click the `Add` button and click on the white space of the screen. This will open the Monitor window, as seen in Figure 9.

Fill in the proper display and variable name that is being references, and this should successfully create a monitor for the variable.

Repeat for the `bank-to-loan` variable.

For the other items we want to monitor, we need to add code to perform the computations. At the bottom of the Code Tab, add the following code:

```
to-report savings-total
  report sum [savings] of turtles
end
to-report loans-total
```

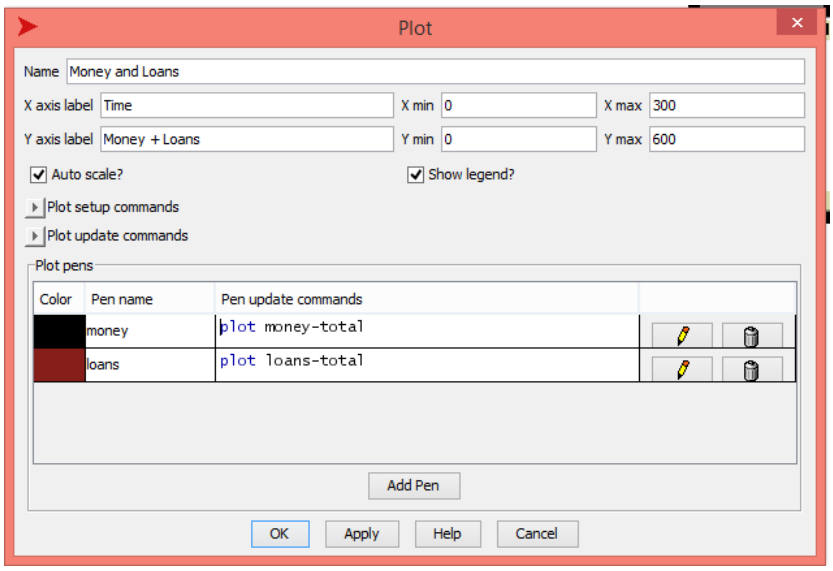

Figure 10: Screen to add a plot monitoring money and loans to the interface.

```
  report sum [loans] of turtles
end
to-report wallets-total
  report sum [wallet] of turtles
end
to-report money-total
  report sum [savings + wallet] of turtles
end
```

Now these reporters have been made, add them as monitors to the interface in the same manner as before.

5.2 Graphs

For additional graphs, we would like tor track money alongside loans, savings alongside wallets, and a historgram of the wealth distribution.

Add a plot in the same fashion as before, naming this one "Money and Loans". Set up the plot as described in Figure 10.

Finally we want to add in a histogram, the code is slightly more complicated to add something like this. Follow the same steps to add a plot to the system, naming the plot "Wealth Distribution Histogram". Label the axis as demonstrated in Figure 11, and fill out the plot setup commands.

Click the pencil next to the hist pen selected to enter the advanced edition screen, seen in Figure **??**. This screen allows for the larger input screen to include the complex function that we need for the histogram. Enter the following code into the "Pen update commands section":

```
if (ticks mod 10 = 1) [
let max-wealth max [wealth] of turtles
let min-wealth min [wealth] of turtles
```

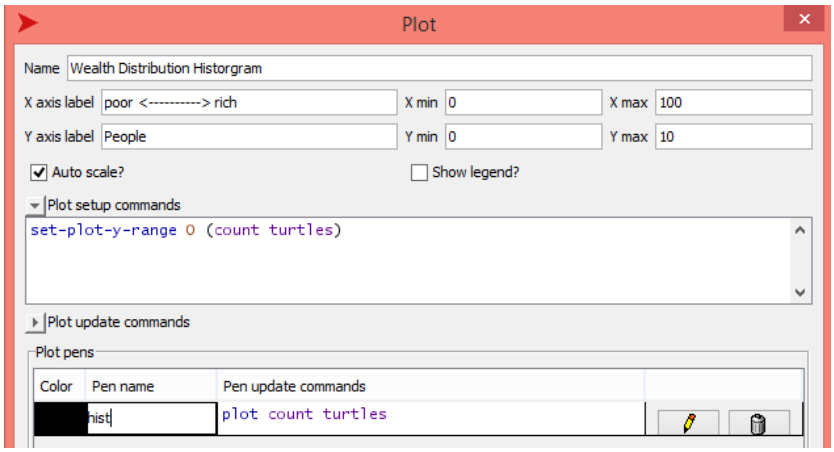

Figure 11: Screen to add a Histogram monitoring wealth distribution.

```
let one-fifth-wealth 0.2 * (max-wealth - min-wealth)
let num-bins 10
let index 1
let interval round ((plot-x-max - plot-x-min) / num-bins)
plot-pen-reset
repeat num-bins [
plotxy ((index - 1) * interval + 0.002)
  (count turtles with [
    wealth < (min-wealth + index * one-fifth-wealth) and
    wealth >= (min-wealth + (index - 1) * one-fifth-wealth)
    ]
    )
plotxy (index * interval)
  (count turtles with [wealth < (min-wealth + index * one-fifth-wealth) and
  wealth >= (min-wealth + (index - 1) * one-fifth-wealth)
  ]
)
plotxy (index * interval + 0.001) 0
set index index + 1
]
]
```

6 References

This tutorial is heavily borrowed from the Shodor tutorial on Individual Based Modeling with NetLogo `http://shodor.org/tutorials/NetLogo/Introduction` and used the model built in: Wilensky, U. (1998). NetLogo Bank Reserves model. http://ccl.northwestern.edu/netlogo/models/BankReserves. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.