AGENT BASED PROGRAMMING, LAB 1

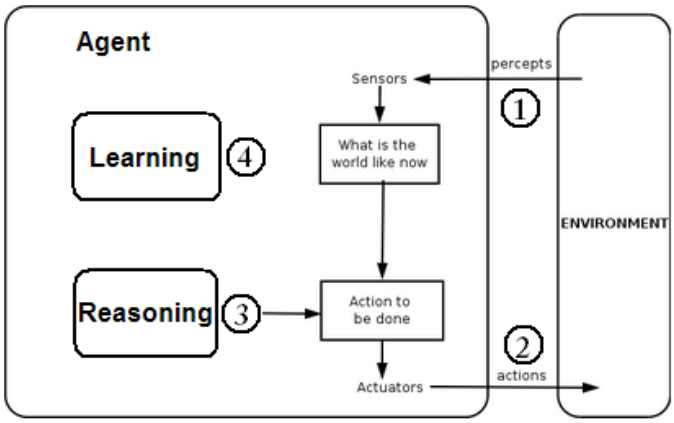
INTRO TO NETLOGO

NAME: _____

INTRODUCTION:

NetLogo is a cross-platform multi-agent programmable modeling environment. NetLogo was authored by Uri Wilensky in 1999 and is under continuous development at the Center for Connected Learning at Northwestern University. NetLogo is free of charge and can be downloaded here: http://ccl.northwestern.edu/netlogo/download.shtml

We are using Netlogo to introduce the concepts behind computer agents and the agent-based programming paradigm.



Using NetLogo will also give you more hands on experience in programming in a language that requires you to type in your own code. Remember, in Netlogo misspelling a word, or adding an extra space in a command will mean **your program won't work**.

The complete programming manual for NetLogo can be found here: http://ccl.northwestern.edu/netlogo/docs/ (alt link)

A shorter easier to user QuickStart Guide can be found here: http://ccl.northwestern.edu/netlogo/resources/NetLogo-4-0-QuickGuide.pdf (alt link)

There are 2 sections to this lab. Collectively these selections introduce the Netlogo environment, simulations, and the basic agents and command sets.

PART 1: WORKING WITH A SIMULATION:

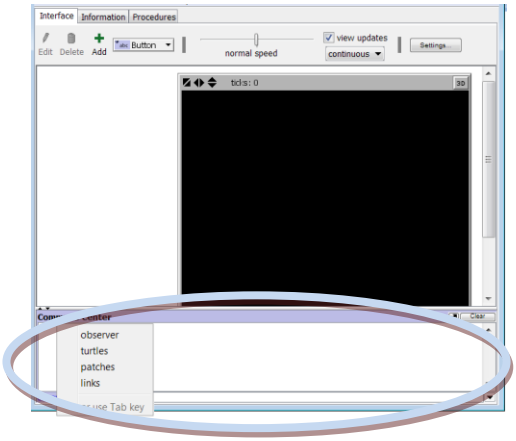
In this section you will open an existing simulation and work with the model to explore the NetLogo user interface. You will complete the standard Netlogo Tutorial #1. You can find this tutorial by going here (don't spend more than 15 minutes on it):

http://ccl.northwestern.edu/netlogo/docs/tutorial1.html (alt link)

PART 2: BASIC COMMANDS:

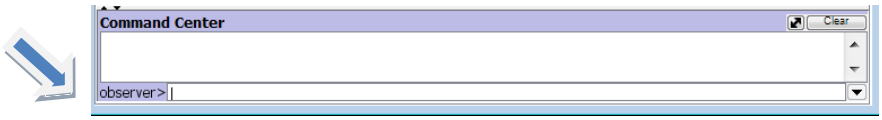
COMMAND CENTER

Start a new NetLogo program by going to File/New.



The command center is found at the bottom of the NetLogo Interface tab window. The command center allows you to send instructions to the "observer" or to other agents or agent-sets.

You can make the Command Center larger so that you can see the previous commands that you have issued. You can also change who the command center issues instructions too, by clicking at the start of the command line. For now we want to issue instructions to the "observer" so the bottom of your command line window should look like this:



The "observer" manages the interface window and can be used to issue instructions to all other agents as well as to monitor events in the simulation.

We are going to issue a series of instructions to the observer which will result in the display "view" on the Interface tab changing. Type each of the following statements (one at a time) into the command center command line (hit enter after each instruction) and see what happens.

clear-all
create-turtles 50 [set color red]
ask turtles [forward 15]
ask turtle 0 [set color green set pcolor blue]

What happened? What does the screen look like?

Things to notice

1. Turtles must be created; they do not exist by default.
2. Turtles move across the patches in the view
3. We can command the observer to **ask** every turtle to move.
4. Blocks of related code are bracket delimited: [forward 10]
5. A turtle is identified by its id (it's *who* value), not its coordinates (xcor, ycor).
6. Although pcolor is a patch attribute, not a turtle attribute, we can use it to directly ask a turtle to change the color of the patch it is on: [set pcolor green]

Limitations of the Command Center

The command center is limited. It **cannot** be used to

1. Introduce new global variables
2. Introduce new local variables (for use in more than one input line)
3. Define procedures (functions) or reporters
4. Change the interface window (make buttons, graphs, etc.)

Context in the Command Center

You can change who you issue commands too. This is known as changing the context:

- **observer** content - > used to create new turtle and link agents, issue commands to the interface window (like clear-all) and can be used to instruct pass commands to patches or turtles (as in -> *ask turtles [forward 15]*)
- **patches** context -> give instructions directly to patches
- **turtles** context -> give instructions directly to turtles
- **links** context -> give instructions directly to links

Turtles Context

Switch to the ***turtle*** context by clicking on the front of the command line (it should currently say observer) and choosing turtles. Enter the following commands into the command center:

```
fd 5                ;; move forward 5 units
rt 180              ;; turn right 180 degrees
pd fd 5 pu          ;; pen-down, move forward 5, pen-up
set hidden? true    ;; turn invisible
set pcolor white    ;; change patch color to white
```

NOTE*: Comments are ignored by the NetLogo IDE. Comments are delineated by two semicolons (;;). All the remaining commands you will see in this lab include comments. They will be ignored by the NetLogo program and are only there to describe what the commands are meant to do.*

Primitives

In NetLogo "primitives" are variables (attributes) and instructions (commands) that are built into the NetLogo language; part of the "semantics" of the language. The instructions that you have been using (forward, clear-all, set) and the variables that you have been using (color, pcolor) are all referred to as "primitives"

INTRODUCTION TO TURTLES

Variables: Each turtle comes with certain built-in data attribute primitives (variables):

Name	Description
who	A unique number that identifies each turtle (created automatically).
breed	Used like a sub-type, so you can have turtles which are also "sheep" or "wolves".
hidden?	A Boolean value (true or false) indicates if turtle is invisible
xcor ycor	Two variables that define a turtles position in the x and y plane (by default 0, 0)
color	Color of turtle (can use hex numbers). Defaults to random.
heading	Direction turtle is pointing in degrees. 0 = up, 90 = right, (defaults to random)

Name	Description
shape	A bitmap image to use for the turtle. Defaults to triangle
size	Size of image relative to the source. 1 = 100%, 2 = 200%, 0.5 = 50% etc.
label	A description which will appear over the turtle. Default is blank.
label-color	Color of label, default is white.
pen-mode	Current status of pen (up, down, or erase).
pen-size	Size of pen, in pixels.

To see the attributes for a turtle you can right click on the turtle and choose the inspect option for the turtle or type inspect into the command line. Type the following into the command line:

inspect turtle 0

You can close the inspect turtle window by clicking on the x in the top right of the window.

Instructions: Each turtle comes with many built-in procedures (functions). Here are few:

Name	Description/Example
create-turtles (crt)	crt 5 ;; create 5 new turtles
cro	*As above but turtle headings range from 0 to 360, evenly spaced.*
clear-turtles (ct)	ct ;; deletes all turtles
forward (fd) back (bk)	ask turtles [fd 5] ;; go forward 5 steps
setxy	ask turtle 5 [setxy 5 5] ;; move to 5,5
right (rt) left (lt)	ask turtles [rt 5] ;; turn right 5 degrees

Name	Description/Example
jump	ask turtles [jump 5] ;; go forward 5 steps
hatch	ask turtle 0 [hatch 1] ;; ask turtle 0 to create 1 new turtle
die	ask turtle 0 [die] ;; ask turtle 0 to delete itself.
pen-down pen-up	ask turtle 0 [pen-down] ;; ask turtle 0 to start drawing
ht st	ask turtles [ht] ;; hide all turtles
random-xcor random-ycor	ask turtles [setxy random-xcor 0] ;; move turtles to a random xcor

Accessing turtles by ID

Each turtle is given a unique id number at creation, starting at 0. This id number is the turtle's "who" attribute. We can access groups of turtles using the 'who' command which will return a list of turtles that meet our requirements. Enter the following commands into the command line (turtles context should be set) and observe the results:

```
set hidden? false
if who = 0 [fd 10]
if who < 6 [fd -5]
if who >= 6 [set color yellow]
if who > 5 [set pcolor blue]
```

INTRODUCTION TO PATCHES

Variables: Each patch comes with certain built-in data attribute primitives (variables):

Name	Description
pxcor pycor	These numbers used together can be used to select specific patches.
pcolor	The color of the patch

Name	Description
plabel	A description which will appear over the patch. Default is blank.
plabel-color	Color of label, default is white.

To see the attributes for a patch you can right click on the patch and choose the inspect option or type inspect into the command line. Type the following into the command line:

```
inspect patch 0 0
```

Instructions: Each turtle comes with many built-in procedures (functions). Here are few:

Name	Description/Example
clear-patches (cp)	cp ;; reset all patches, defaults to black
sprout	*ask patches [sprout 1]* *;; each patch creates one new turtle*

Name	Description/Example
watch	watch patch 3 3 ;; highlights patch 3 3 ;; rp will reset
diffuse	*Tells each patch to give equal shares of (number * 100) percent of the value of patch-variable to its eight neighboring patches. 'number' should be between 0 and 1. See example in next section.*

Working with Patches

Change the command context from **turtles** to **observer**. Then enter the following commands, and observe what happens:

```
clear-all
ask patches [ set pcolor yellow ]
ask patches with [ pxcor < -5 ]   [ set pcolor black ]
ask patch 0 0 [ set pcolor red ]
watch patch 0 0
reset-perspective
ask patch 0 0   [ask neighbors [set pcolor [pcolor] of myself]  ]
clear-patches
ask patches [set pcolor random 140]
ca                                                    ;; short for clear-all
ask patches [set pcolor white ]
ask patch 0 0 [set pcolor black]
diffuse pcolor 0.5
```

Things to notice

Patches do not have to be created; they exist by default. Patches:

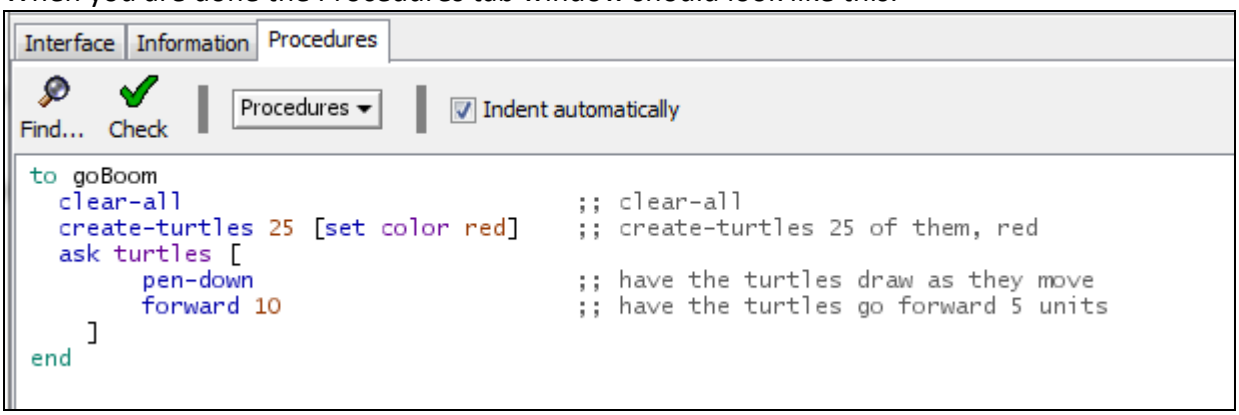
1. can execute commands: [set pcolor white]; code blocks are bracket delimited
2. **do not move**
3. have properties that are often used to represent the state of the "external world", including resources and environmental obstacles (e.g., food supply, rivers).
4. can **sprout** turtles and diffuse (spread to other patches) attribute values.
5. are uniquely identified by coordinates (pxcor, pycor) and we can use these coordinates to determine which patches receive a command.

PART 3: PROCEDURES

We can group sets of instructions under a name into what is called a procedure (function). Click on the "Procedures" tab and enter the following lines of code there:

```
to goBoom
        clear-all                          ;; clear-all
        create-turtles 25 [set color red]  ;; create-ordered-turtles 25 of them, red
        ask turtles [
                pen-down                   ;; have the turtles draw as they move
                forward 10                 ;; have the turtles go forward 5 units
         ]
end
```

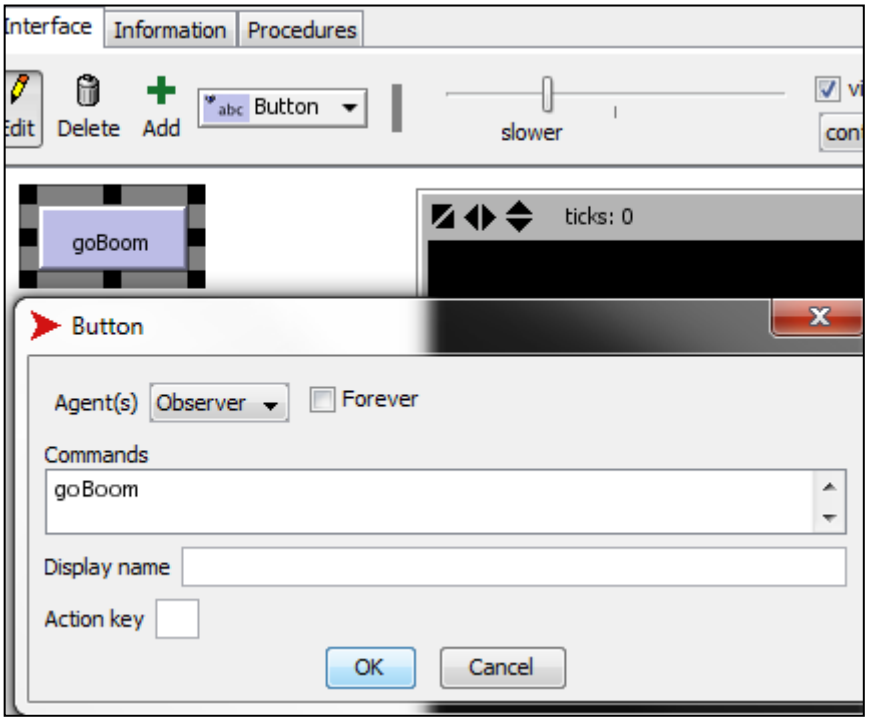
When you are done the Procedures tab window should look like this:



We now have a procedure named "goBoom" which encompasses all of the instructions from the word goBoom to the word end. We can call goBoom using its name from anywhere (***including from inside other procedures***). But for now let's create a button to call goBoom.

PART 4: CHANGING THE INTERFACE

Go the Interface tab and right click in the upper corner of the screen near the view window. You should get a pop-up box, with several options in it, choose **button.**

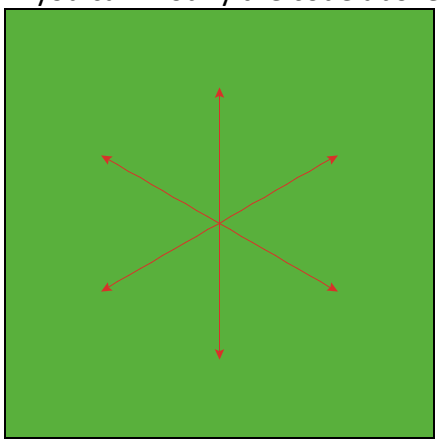


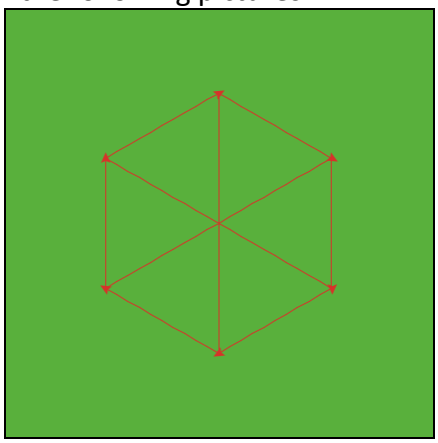
We are going to use this button to call our goBoom procedure. Type goBoom into the commands box. Your screen should look something like the picture to the right. Then hit OK.

Click on the button you just created. What happened?

PART 4: CHALLENGE EXERCISES

See if you can modify the code above to create each of the following pictures:

1. 

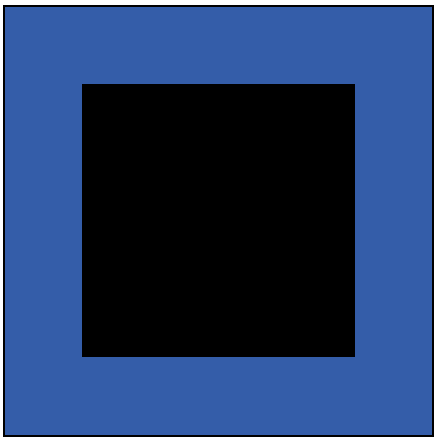
2. 

Hints (for 3 and 4):

1. You only need to modify the patches.
2. You can use parenthesis. Example ((pxcor / 2) = 1)
3. ask patches with
4. ask patches [if

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

abs acos asin atan ceiling cos e exp floor int is-number? ln log max mean median min mod modes new-seed pi precision random random-exponential random-float random-gamma random-normal random-poisson random-seed remainder round sin sqrt standard-deviation subtract-headings sum tan variance

3. 

4.