AGENT BASED PROGRAMMING, LAB 3

CREATING A NETLOGO SIMULATION

NAME: ____

INTRODUCTION:

NetLogo is a cross-platform multi-agent programmable modeling environment. 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 to explore how simulations are created and how they can be used. 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)

In Lab 2 we followed the instructions for the standard Netlogo tutorial #3 and then we added to that model. In this lab we will create our own simulation from scratch.

PART 1: UNDERSTANDING THE PROBLEM (CHAIN-REACTION)

Nuclear fission is a nuclear reaction in which an atom splits into multiple smaller parts (smaller atoms). Nuclear fission releases a tremendous amount of energy and often produces free neutrons as a side-effect.

In a nuclear chain-reaction free neutrons, created by a fission event, collide with other fissile materials (other fissionable atoms) which then also undergo fission. In order for a chain-reaction to occur, a fission event must occur within a high concentration of fissionable material, like uranium 235 (see image on right).

A controlled nuclear chain reaction can be used to heat water to drive a power plant. An uncontrolled nuclear chain reaction can be used to destroy cities.

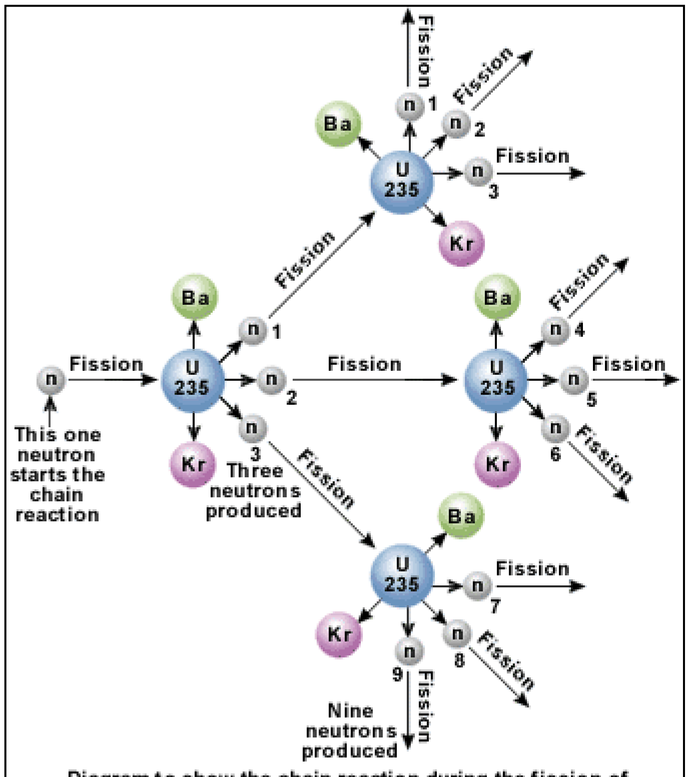
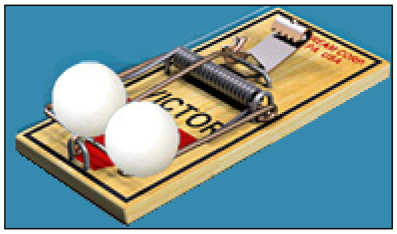


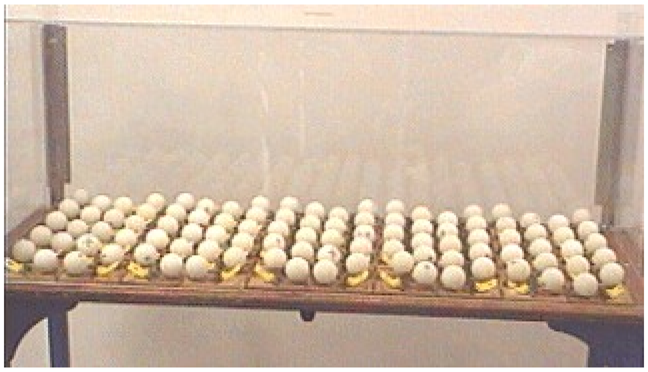
Diagram to show the chain reaction during the fission of uranium-235 with a neutron

ANOTHER TYPE OF CHAIN-REACTION



We can create other types of chain reactions. One famous example of an uncontrolled chain reaction uses mouse-traps and ping-pong balls. In this experiment many mousetraps are set and two ping-pong balls are set on top of each mousetrap.

The set mousetraps are then arranged within a large enclosed space. And then finally a single ping-pong ball is tossed inside. The result is a large, fast and noisy uncontrolled chain reaction.



Videos of/about this type of experiment:

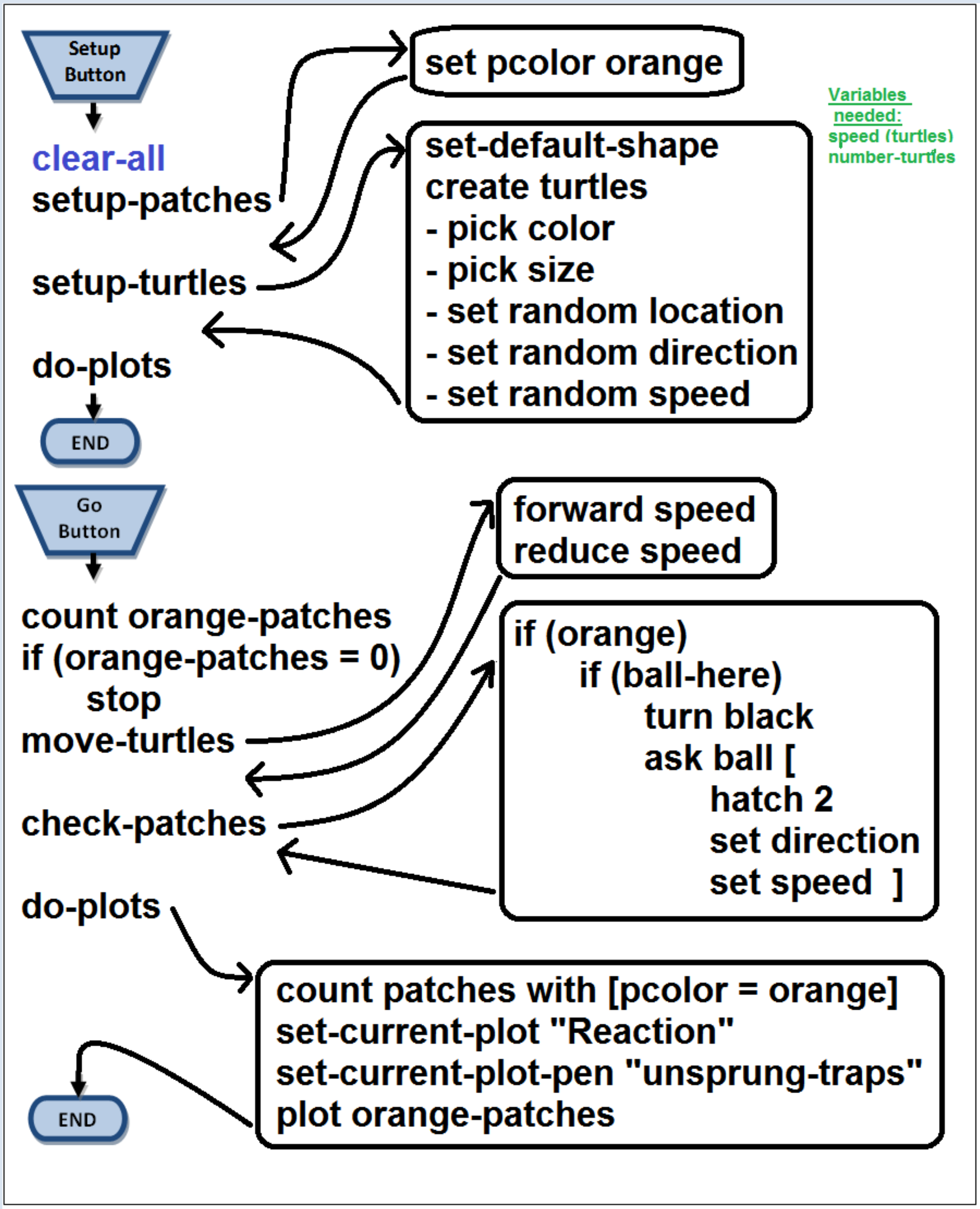
- http://youtu.be/0v8i4v1mieU
- http://youtu.be/hgO1-IqzrZY
- http://youtu.be/vjqIJW_Qr3c
- http://youtu.be/ESpRFkXon7g

PART 2: PLANNING THE MODEL

Never start a NetLogo model by sitting down and coding. **Every** large programming project needs to be planned out, in as much detail as possible, BEFORE you start coding. The more time, effort and detail you put into your planning stage, the faster the coding will actually go.

Before you begin any model (including your project model) you should do the following:

1) Form hypothesis:
   a) What is the question the model is designed to explore?
   b) What do you think the model will demonstrate?
2) Identify model requirements:
   a) What details from the system are essential; what details can be ignored?
   b) What is the environment?
      i) What will the "view" (the world) look like?
      ii) What assumptions are built into the world? (Example: no friction, air resistance)
3) Who are the agents?
      i) What variables will each agent require?
      ii) What characteristics (behaviors) will each agent display?
      iii) What are the limitations (rules) that that the agents must obey?
      iv) What are the "inter" and "intra" agent relationships?
4) Create Flowchart (pseudo-code model) of your program (next page):


Setup Button
clear-all
setup-patches
setup-turtles
do-plots
END
set pcolor orange
set-default-shape
create turtles
- pick color
- pick size
- set random location
- set random direction
- set random speed
Variables needed:
speed (turtles)
number-turtles
Go Button
count orange-patches
if (orange-patches = 0)
stop
move-turtles
check-patches
do-plots
END
forward speed
reduce speed
if (orange)
if (ball-here)
turn black
ask ball [
hatch 2
set direction
set speed]
count patches with [pcolor = orange]
set-current-plot "Reaction"
set-current-plot-pen "unsprung-traps"
plot orange-patches

PART 2: PROGRAMMING YOUR MODEL

Once you have detailed what your program will do, answered all the questions in the previous section and created a flow chart program for your model, you are ready to start coding.

1. Get a copy of the Project Template (here)
2. Label your project (using comments) in the code at the top of the procedures tab.
   a. You may also wish to fill in the fields on the information tab (but not for this lab).
3. Add any extensions you are going to need to the top of your code in the procedures tab:
   a. Example: *extensions [sound] ;; Add the ability to use sounds*

   No extra extensions are needed for this lab!

4. Create breeds (if needed).
   a. Breeds are turtle (or patch) sub-groups that have unique names.

   No extra breeds are needed for this lab!

5. Adjust world (screen) settings.
   a. How many patches do you need for your simulation (size of world)?
   b. What size should your patches be?
   c. Where will location (0, 0) be?
   d. Torus or rectangle (does the world "wrap")?

   You don't need to change the "view" for this lab, but you CAN do so by right-clicking on the view window and choosing edit.

6. Create **setup** and **go** buttons.
   a. All standard NetLogo simulations will have these two buttons.
   b. The setup and go buttons will call the setup and go functions (respectively).

   The setup and go buttons already exist and you don't need to modify them. You can examine them by right-clicking on them and choosing edit. Notice how the go button has the "forever" option selected. This means it will call the go function repeatedly.

7. Create interface controlled variables (sliders, etc..)
   a. Some variables you will want the users to be able to change, these variables should be implemented using sliders, or input forms.

   We would like a variable called "starting-balls" which will allow the user to pick how many ping pong balls will be thrown in at the start of the simulation. Create a slider by right clicking in the interface window and choosing slider. Change the options so that the user can select between 1 and 100 ping pong

8. Create global and local variables in the procedures tab (if needed).

Create a turtle's only variable (in the variables section) called speed, which will represent how fast an individual turtle is moving: *turtles-own [speed]*

9. Create and fill **setup** procedure with sub-setup procedures.

Find the setup function in the procedures tab. Make it look like the following:

```
to setup
        clear-all           ;; Resets everything, makes screen blank.
        setup-patches       ;; Calls the setup-patches function.
        setup-turtles       ;; Calls the setup-turtles function.
               ;; The do-plots function is at the bottom of the page.
        do-plots            ;; Calls the do-plots function.
end
```

10. Create and fill sub-setup procedures.

1. Go back and look at what we need "setup-patches" to do in the flow-chart we created. Create the setup-patches function (put it right below the setup function) and put inside it the code needed to make all patches orange.
2. Add the following code to the setup-turtles function:

```
set-default-shape turtles "circle"
create-turtles starting-balls
[
        set color white
        set size 0.5
        setxy random-xcor random-ycor
        set speed random 100
        right random-float 360
]
```

3. **Go back to the setup-turtles function and put a comment after each line describing what the line does! REFERENCE LIBRARY LINK**
4. **Test your program by going to the interface window and clicking on the setup button. What happens?**

11. Create and fill **go** procedure with sub-go procedures.

Go back and look at what we need the go procedure to do. What sub-procedures does "go" need to call?

Make an attempt to fill in the go procedure, using the information from the flow-chart, BEFORE moving on to the next page.

1. Make the go procedure look like this:

```
to go
        let orange-patches count patches with [pcolor = orange]
        if orange-patches <= 0 [ stop ]
        move-turtles
        check-patches
        do-plots
        tick
end
```

2. **Go back into the go procedure and put a comment after each line describing what the line does! REFERENCE LIBRARY LINK**

12. Create and fill go sub-procedures.

1. **Create the move-turtles procedure:**
   - Look at the flow-chart and decide what the turtles need to do.
   - Remember you will need to **ask** the turtles
   - Remember they should move **speed** distance (speed is a variable)
   - Remember that each time they move speed should get slower

2. Create the check-patches procedure so that it looks like this:

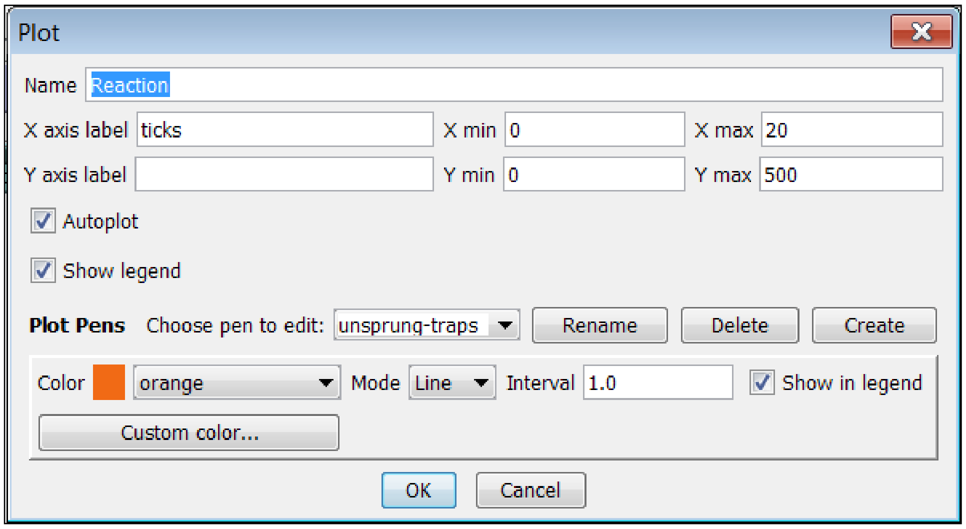
```
to check-patches
    ask patches [
        if pcolor = orange [
            let ball one-of turtles-here
            if ball != nobody [
                ask ball [
                    set speed random 100
                    right random-float 360
                    hatch 2 [
                        set speed random 100
                        right random-float 360
                    ]

                ]
                set pcolor black
            ]
        ]
    ]
end
```

3. **Go back into the check-patches procedure and put a comment after each line describing what the line does! REFERENCE LIBRARY LINK**

13. Create plot and plot pens (in the interface).

1. Go to the Interface window and create a new Plot (right-click) name "Reaction".
2. **We want to track <u>two variables</u> using Plot Pens**
    - **unsprung-traps (make it an orange pen)**
    - **balls-released (make it a black pen)**



Plot

Name: Reaction
X axis label: ticks
X min: 0
X max: 20
Y axis label:
Y min: 0
Y max: 500
[✓] Autoplot
[✓] Show legend
Plot Pens — Choose pen to edit: unsprung-traps
Color: 🟧 orange
Mode: Line
Interval: 1.0
[✓] Show in legend

14. Create and fill do-plots procedure.

1. Make the do-plots procedure look like this:

```
to do-plots
        let orange-patches count patches with [pcolor = orange]

        set-current-plot "Reaction"
        set-current-plot-pen "unsprung-traps"
        plot orange-patches

end
```

2. **Go back into the do-pots procedure and put a comment after each line describing what the line does! <u>REFERENCE LIBRARY LINK</u>**
3. **Add the two lines of code necessary to have the do-plots procedure also plot the number of ping pong balls that have been created and released. (Hint plot the count of turtles after switching pens).**

PART 3: TESTING, EVALUATING, EXPANDING (CHALLENGE EXERCISES)

After you have completed your model (one possible solution) answer all of the questions below, then move on and try to improve your model by completing the challenge exercises.

Once you have completed your model you should test it:

1. Do the agents act as expected?
2. Do the rules behave as expected?
3. Are plots and monitors reporting what you expected?

Evaluate your hypothesis using your model:

1. How does the model change with different setups?
2. What is the effect of randomness on your model?
3. How many times do you think you need to run the model to test it properly?
4. Does the model you created validate or invalidate your hypothesis?

Expand your model (Challenge Exercises):

1. Add an appropriate sound for when the model is setup (and maybe for each time a "trap" is sprung.
2. Have patches occasionally NOT react (don't change color or hatch new ping pong balls) even though a ping pong ball is in the square. Make this percentage chance to not react adjustable (use a slider).
3. Add buckets to your simulation, that is, have certain squares catch the flying ping-pong ping pong balls and destroy them.
4. Keep track of the number of ping-pong balls that were caught in the buckets.
5. Change your ping-pong balls so that they never slow-down.
6. Have your patches able to spawn multiple ping pong balls before they finally turn black. Make the number of times they can spawn ping-pong balls a slider.
7. Change your buckets so that they won't ALWAYS catch a ping pong ball that hits them. Make this percentage chance to not capture the ping-ping ball adjustable (use a slider).
8. Change the simulation so that the user can control the location of the buckets. (Use the event handler mouse-down with the mouse-xcor mouse-ycor variable)
9. Set up upper limit for the number of ping-pong balls that can be on the screen. If that upper limit is broken, stop the simulation (or else risk a meltdown).

NOTE: Each of the challenge exercises listed above is moving the simulation closer to a model of a nuclear reactor (which uses control rods to absorb free neutrons).