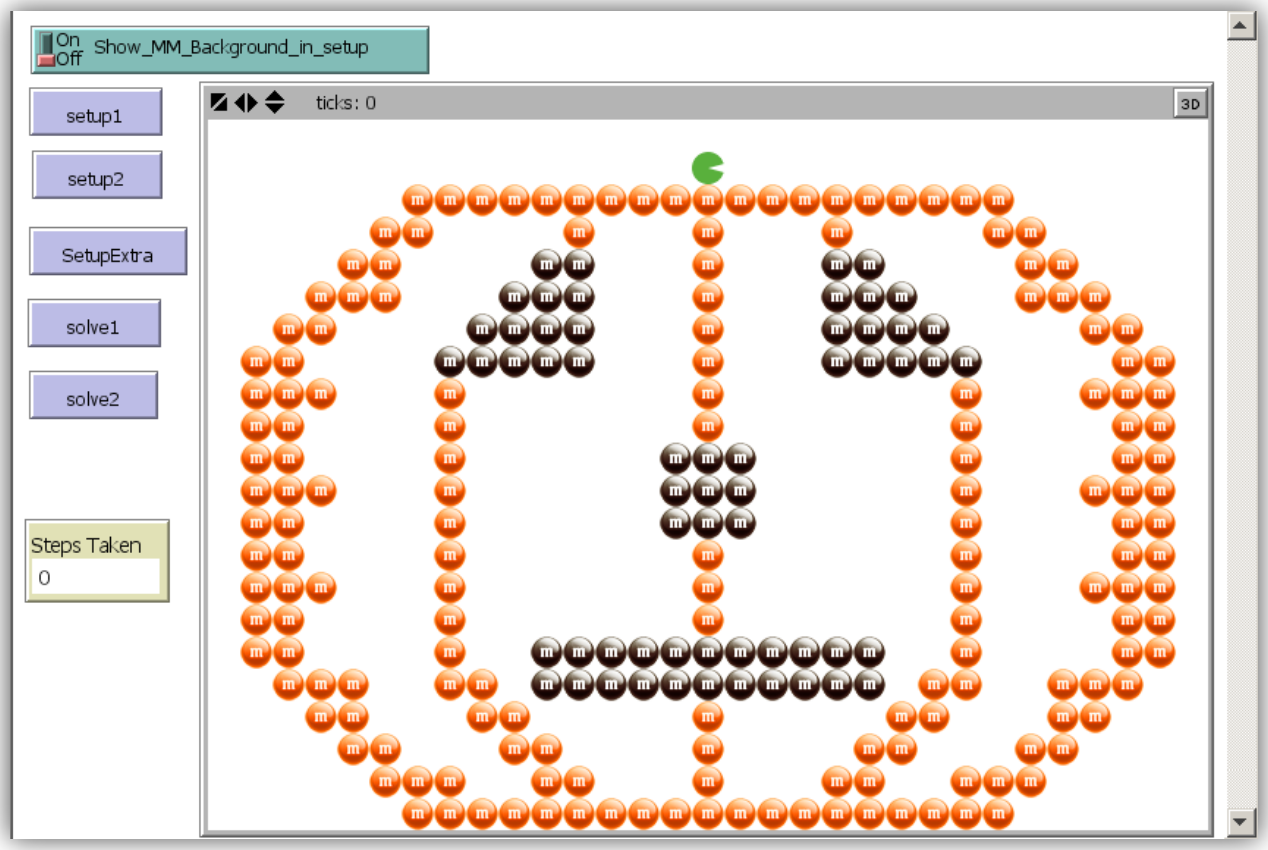
Lab 9: m&m Chomper



Model Overview:

Given the Netlogo ***m&m Chomper*** model, your task is to improve its chomping efficiency. The green chomper dude is on a quest to devour all m&ms. In the version you are given, the chomper dude does this for both setup1 and setup2, but to do so, the dude chomps through 682 patches. Your goal is to teach the chomper not to walk through so many empty patches by rewriting the code in **`solve1`** and **`solve2`**.

NOTE: when this model runs, it requires the two m&m image files: **`MandM black.png`** and **`MandM orange.png`**, to be in the same folder as the model AND for the file names to be exactly as they are (not even capitalization can be changed).

These image files can be downloaded from the class website:
http://www.cs.unm.edu/~joel/cs151/

THE UNIVERSITY of NEW MEXICO

The Rules:

#1) The model you are given has 9 procedures:

```
to setup1
to setup2
to setupExtra
to setupAll
to-report getPixelX [ x ]
to-report getPixelY [ y ]
to moveMonster [ direction ]
to solve1
to solve2
```

You MUST NOT MAKE ANY CHANGES to the first 7 of these.

You will need to change both the solve1 and solve2 procedures. Additionally, you may add any other procedures that you find useful.

#2) The chomper may ONLY be MOVED by calling the **moveMonster** procedure. You MAY NOT MOVE CHOMPER in any way except by calling **moveMonster**. However, you may still use **ask turtle** to look at or change chomper's patch color, look at or change the patch color of neighbors, etc. The **moveMonster** procedure is observer-only, so you must end any **ask turtles** block before calling **moveMonster**.

The **moveMonster** procedure takes one parameter: a direction which can be any of the 4 global variables: **NORTH**, **EAST**, **SOUTH**, or **WEST**. The **moveMonster** procedure will then move the chomper one patch in the specified direction – devouring any m&m that might be in the patch were it moves.

Getting Points (for your grade):

You get points for teaching chomper to devour all m&ms in each setup in less than 682 steps. The less steps your chomper needs to do its devouring, the higher your score.

Hints:

1) In setup 1, it is not possible to eat all 120 m&ms in just 120 steps. This is because the shape of the pattern will require either some backtracking or some moving off the pattern to traverse.

2) Since you are given the pattern of m&ms, the simplest approach is to program chomper with each turn needed for it to follow what you think is the shortest path. If you want chomper to move 5 patches to the east, you can code:

```
moveMonster EAST
moveMonster EAST
moveMonster EAST
moveMonster EAST
moveMonster EAST
```

or

```
repeat 5
[ moveMonster(EAST)
]
```

whichever you fancy.

3) A random walk will NOT be an improvement (it is likely to take 1000s of steps).

4) The m&ms in setup2 form a single, continuous, twisting line of 307 m&ms. Since this pattern has a start, a finish and no dead ends in between, it can be fully chomped in 307 steps. However, even if your algorithm takes more than 307 steps, you will still earn some points as long as you beat 682 steps. See the grading rubric for details.

5) You can solve the pumpkin setup in 307 steps by coding each turn. This is simple to understand, but rather tedious. A solution that is a bit less simple to understand, but has significantly fewer lines of code is to teach the chomper to follow any single, twisting string of m&ms. This can be done by starting at one end of the chain, always moving to the next uneaten m&m, and changing the patch color in the location of each m&m you eat. NOTE: there is no Netlogo command "moveToNextUnEatenM&M". If there was such a command, you would NOT be allowed to use it. THE ONLY WAY THE CHOMPER (turtle) CAN BE MOVED is by calling the `moveMonster` procedure.

6) The "`Show_MM_Background_in_setup`" switch is provided to help the student who wants to write a chain following algorithm (see hint #5). This is because the m&m ***sprites*** are not actually patch colors. The sprites are drawn on a graphics layer above the white patches. However, if the "`Show_MM_Background_in_setup`" switch is selected before setup is clicked, then in addition to drawing the m&ms sprites, setup1 and setup2 will set the patch color to orange for every patch containing an m&m. All other patches will

have a color of white. This does not look as pretty, but it makes an easy way for your algorithm to "smell" m&ms from a distance. For example, one way of doing this, from with an **ask turtles** block, is:

```
let target one-of neighbors4 with [pcolor = orange]
if (target != nobody)
[ ask target
  [ ;;Your code for doing something with the target goes here.
    ;; In particular, you will want to do something that lets you decide which
    ;; direction to pass to moveMonster. The rub is that you will not be
    ;; able to call the moveMonster procedure from within this block.
    ;;  This is because one-of neighbors4 must be within a turtle
    ;;  context whereas moveMonster is observer-only.
    ;; What can be done to get around this is to save the direction you
    ;; decide to move in a local variable. This will need to be a local variable
    ;; that is defined and initialized before you enter the turtle context so
    ;; that it has a scope defined outside the turtle context.
    ;; If you do not understand what is meant by a variable's "scope" then
    ;; rewatch the video:
    ;;   "Variables and Scope"  by Maureen Psaila-Dombrowski
  ]
]
```

nobody is Netlogo's keyword for an empty agent set. In the case of the pumpkin pattern, if the turtle starts at one end of the m&m chain, and never makes a misstep, then the only time **one-of neighbors4 with [pcolor = orange]** would report **nobody** is at the other end of the m&m chain.

For more information on Netlogo's **one-of**, **neighbors4**, and with, see the NetLogo Programming Guide:
http://ccl.northwestern.edu/netlogo/docs/programming.html

7) If you find it helpful, it is okay to change the patch color. For example, you could change the patch color of every patch you enter or to one of 4 different colors used to keep track of which direction you entered the patch from.

8) The **setup1** and **setup2** procedures each set the global value **mmCountAtStart**. If you want, you may count down from this so as to know when your chomper has nothing left to chomp.

THE UNIVERSITY of NEW MEXICO

Grading Rubric: [20 points total]:

[A: -20 points]: Your program moves the chomper (turtle) in some way other than calling **`moveMonster`**.

[B: 1 points]: Submit one documents to your instructor: Netlogo source code named: `W10.`*firstname*`.`*lastname*`.nlogo`.

[C: 1 points]: The first few lines of your code tab are comments including your name, the date, your school, and the assignment name.

[D: 2 points]: The code in the code tab of your program is appropriately documented with "inline comments".

[E: up to 8 points]: Your "solve 1" teaches the chomper eat all m&ms.

0 points: Not all m&ms are eaten.
0 points: All m&ms eaten in more than 528 steps.
1 point: All m&ms eaten within 528, but more than 402 steps.
3 points: All m&ms eaten within 402, but more than 350 steps.
4 points: All m&ms eaten within 350, but more than 234 steps.
6 points: All m&ms eaten within 234, but more than 173 steps.
8 points: All m&ms eaten in 173 steps.

[F: up to 8 points]: Your "solve 2" teaches the chomper eat all m&ms.

0 points: Not all m&ms are eaten.
0 points: All m&ms eaten in more than 609 steps.
2 point: All m&ms eaten within 609, but more than 500 steps.
3 points: All m&ms eaten within 500, but more than 400 steps.
4 points: All m&ms eaten within 400, but more than 350 steps.
6 points: All m&ms eaten within 350, but more than 307 steps.
8 points: All m&ms eaten in 307 steps.

THE UNIVERSITY of NEW MEXICO

Extra Credit: [+10 points]:

For extra credit, add a new button called "SolveAll". This must teach chomper to devour the m&ms in for all three setups (setup1, setup1 and setupExtra) while following some additional rules:

1) Chomper may never move into a space that does (or did) not contain an m&m.
2) Unlike the regular credit solutions for setup1 and setup2, the extra credit solution cannot be a list of moves that traces a particular pattern. The extra credit solution must use the same algorithm to solve all three setups.
3) The extra credit algorithm must be ***reasonably efficient***. That is, it is okay if the algorithm causes chomper to backtrack somewhat more than necessary, but it is not okay for the chomper to wander around and around the m&m path taking 1000s of steps to find all the m&ms.

I will post a short video showing some working general solutions in action.