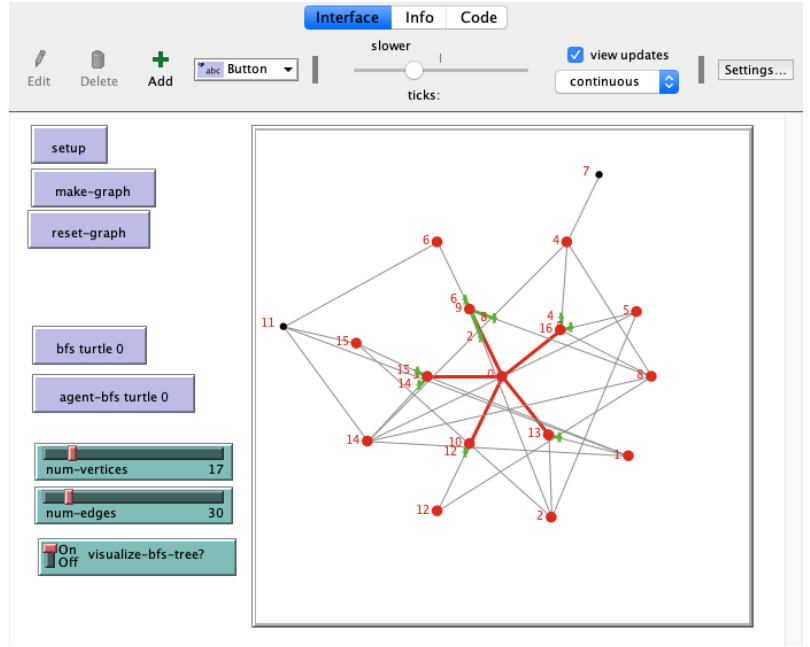
CSCI 390. Day 4

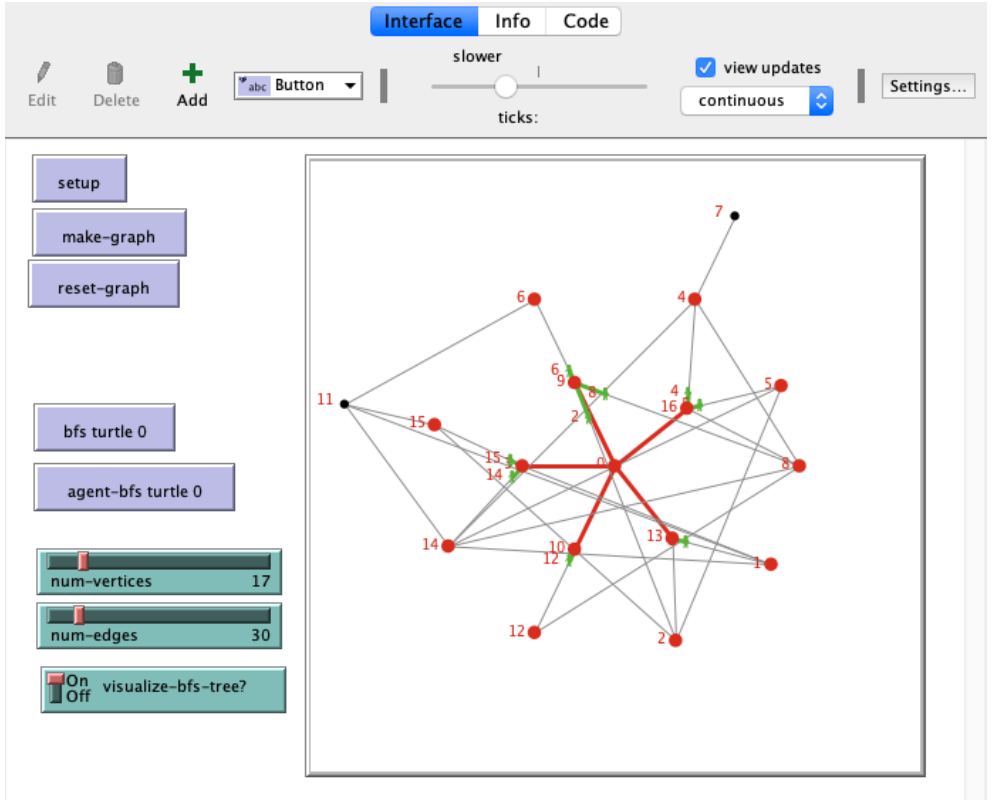
SO FAR…

- We have introduced several aspects of the NetLogo language and world, and practiced programming to get used to syntax, and learn common built-in procedures and reporters including:
  - Basic agent movement
  - Creating breeds and adding instance variables
  - Using links
  - Lists
- **REMINDER**: You should be reading up on these. See the syllabus. For example, when we introduce something like links, go through the procedures and reporters for links.
- We have also started learning how to implement some 201-level data structures in NetLogo
- And have also reviewed some good principles for program design (in any language)!



TODAY

- Get a little bit more into the language, and how it functions as an agent-based modeling language.
- Look at BFS and a simple NetLogo list-based implementation of a queue.
- Look at a BFS implemented with agents instead of a queue.
- Extend the BFS concept to a weighted graph with the Dijkstra algorithm.



What Really Happens with an Ask?

Consider the statement: `ask agentset [commands]`

For example:

```
ask patches [set pcolor red]
ask turtles [fd 1]
```

Conceptionally, we imagine all the agents in the agentset performing the given commands in parallel (simultaneously). But the CPU running the instruction only executes sequentially!

The goal is to simulate parallel behavior.

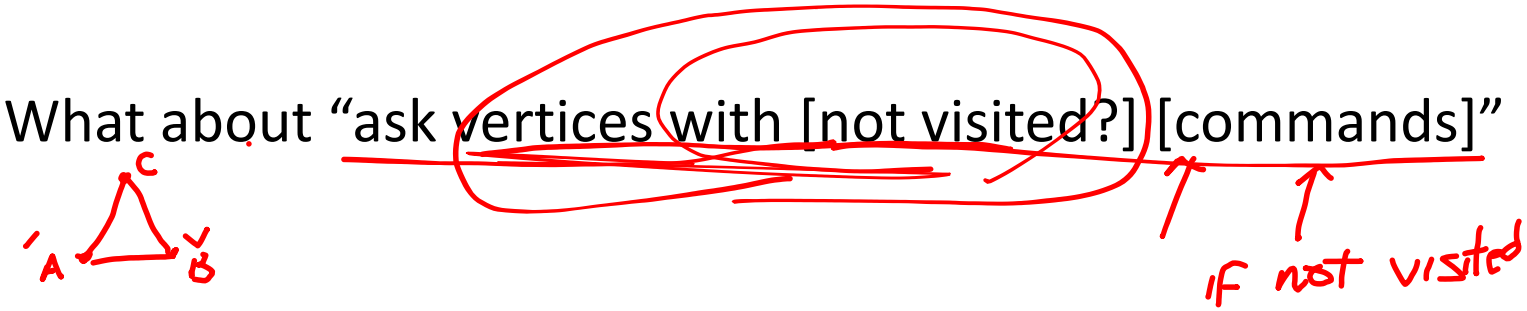
How? The agents in the *agentset* perform the commands sequentially but in random order. Each agent completes the entire command, and then the next agent (in random order) performs the command.

Demonstrate with the examples above, with the slider set to slow.

What Really Happens with an Ask?

Consider the statement: `ask agentset [commands]`

- Broader principle: agentsets are unordered.
- The agents in the agentset run the given commands. Because agentset members are always read in a random order, when **ask** is used with an agentset each agent will take its turn in a random order.
- An agent can **ask** an agentset to perform an action. However only the observer can ask all turtles or all patches.
- The agents that are in the agentset *at the time the ask begins* run the commands.
  - **Example**: What about "ask vertices with [not visited?] [commands]"



Efficiency?

What is the result of the following?

```
ask turtles [ ask other turtles with [distance myself <= 5]
              [create-link-with myself]]
```

What is the result of the following?

```
ask turtles [ ask other turtles in-radius 5 [create-link-with myself]]
```

Which is more efficient?

How can you have all patches containing a turtle to turn blue? Ideas?

```
ask patches [ if any? turtles-here [set pcolor blue]]
ask turtles [ask patch-here [set pcolor red]]
```

Which is better?

BFS (Breadth First Search)

A BFS of a graph…

- …computes the shortest path from every other vertex to the foot.
- The resulting BFS-tree contains all those shortest paths.
- The BFS visits all other nodes a distances of 1 from the root, and then all those a distance of 2, etc. It visits all nodes a distance of i from the root before visiting any a distance of i+1

Work backwards today.

- Instead of live coding, we will look at some working BFS code and see what it does, so we have a mental picture of the process, and then look at the completed code. We will demonstrate and then show the code for:

1. Standard graph BFS using a queue (FIFO list)
2. Agent-based BFS where agents search a graph in (simulated) parallel, taking the place of the queue.

Dijkstra's SSSP Algorithm

Given a connected non-negatively weighted graph G=(V,E) and a root node r in V.

```
; Initialize
for all nodes v {
    mark v unvisited
    initialize distance d(v)=infinity.
    set parent p(v)=v.
}
d(r)=0.

; Primary loop, will iterate n times
; Each time adds a new vertex to SSSP tree
While any nodes remain unvisited {
    let v be the unvisited node minimizing d(v)
    mark v as visited.
    for all edges (v,w) {
        if d(v)+Weight(v,w)<d(w) {
            set d(w)=d(v)+Weight(v,w)
            set p(w)=v
        }
    }
}
```