HW1, due Tuesday January 26 in the course DropBox by 8am EST

Submit HW1 individually but indicate your lab partner on the comments. Be sure to use the indicated name (with dashes, and not underscores),

Instructions; Implement two versions of the Dijkstra SSSP algorithm on a connected weighted graph. The edge weights are positive integers, not related to Euclidean distance. The user should be allowed a random graph. Allow a random root or user selected root for the search.

One version should be a standard algorithm animated only by highlighting the edges as the algorithm runs, and showing the resulting SSSP tree.

The second version should use agents to perform the search. Note that the speed of the agent should be set so that length of travel time along an edge is proportional to the weight (and not Euclidean length) of an edge. Both algorithms should correctly compute the SSSP tree from the given root.

BONUS: Let the user choose between a random graph, a graph from a file, or graph from user input.

SUBMIT: Name your code as follows: **`Lab2-LastnameFirstinitial.nlogo`**. Submit the final code to the drop-box before the next class period. HW will be graded on:

- Correctness and functionality.
- Style and structure. (Good use of top-down design and procedural abstraction.n)
- Use of Netlogo primitives.
- Commenting.

Dijkstra’s SSSP Algorithm

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

```
; Initialize the data structure for the vertices (nodes).  r is the root note.
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 by the unvisited node minimizing d(v)
    mark v as visited.
    for all edges (v,w) {
        if d(v)+W(v,w)<d(w) {
            set d(w)=d(v)+W(v,w)
            set p(w)=v
        }
    }
}
```