Starlogo and its Relatives

• **Logo (Papert)**
  - Language for teaching mathematics graphically
  - Tell turtle how to move

• **Starlogo (Resnick) & StarlogoT (Wilensky)**
  - Many turtles
  - Tell turtles how to interact with each other and world
  - Ontology: Agents (turtles) and world (patches)

• **Netlogo**
  - Java version of Starlogo
    • Slightly different syntax (ask patches)
    • Slower
  - Hubnet: each turtle is controlled by an actual human
    • Turtles interact over the internet
Properties of Starlogo

• **Nice**
  - Concise, high-level language
  - Complete (Simulated) Parallelism
  - Makes systems graphically intuitive and concrete
  - Particularly appropriate for spatial systems
  - Useful for world-world, world-agent, and agent-agent interactions
  - Large installed user base
  - Very easy interface building: Sliders, buttons, switches
  - Many useful routines: plot, diffuse, scale color, count neighbors, spawn, etc.

• **Not so Nice**
  - Slow (particularly netlogo)
  - Not suitable for very large simulations
  - Quirks that must be programmed around
  - Some limitations to programming constructs
  - Idiosyncratic syntax (e.g. setheading heading + 1)
Elementary Turtle Commands

- **ca** - clear all
- **crt 10** - Create 10 turtles
- **fd 10** - Move turtles forward 10 places.
- **seth 90** - set heading of turtles to 90 degrees = to right. 0 = upward
- **rt 45** - Turn the turtles 45 degrees to right
- **pd** - Pen down as turtle moves. **PU** - Pen up
- **stamp red** - creates a red patch underneath turtles
- **setcolor green** - sets color of turtle
- **setxcor 0** - sets x coordinate of turtle to left-right middle.
- **setycor screen-edge-y** - sets y coordinate of turtles to top
- **setxy mouse-xcor mouse-cor** - places all turtles at cursor
- **seth (random 360)** - gives each turtle a random heading
- **die**
Elementary Patch Commands

- setpc red - set patches all to red
- setpc pc + 1 - set patch’s color to whatever it was + 1
- setpc (random 256) - set each patch’s color randomly
- repeat 100 [setpc pc + 1] - repeat the color increment command 100 times
- setpc xcor - set patch’s color to its x-coordinate
- setpc xcor + ycor - set patch’s color to sum of its two coordinates
  Ambiguous apparent motion effect
- if xcor < 0 [setpc red] - set patch to red if its x-coordinate < 0
- if (distance 5 5) < 10 [setpc green] - turn patches close to {5,5} green
- if (distance mouse-xcor mouse-ycor) < 4 [setpc blue]
  - if patch is close to cursor, turn it blue. Can embed in repeat
- ifelse ycor < 0 [setpc red] [setpc blue]
  - if patch is on bottom, turn it red, else turn it blue. Note brackets, and spaces
Elementary Interface

- **Buttons**
  - To run a command
  - Forever option makes command continually run

- **Sliders**
  - To dynamically set parameters

- **Monitors**
  - To dynamically observe a variable or quantity

- **Switches**
  - To set a boolean variable

- **Paint brushes for patches and turtles**
Move Around Procedure

• To move ;; make this a forever button
  fd 1
  rt angle ;; make an angle slider to vary
  end
Flocking Around Cursor

- **to setup**
  - ca
crt 100
  - setxcor (random screen-size-x)
  - setycor (random screen-size-y)
end

- **to flock**
  - seth (towards mouse-xcor mouse-ycor) + (random randomness) - (random randomness) ;; randomness is a slider
end
Flocking Around Cursor With Inertia

- `turtles-own [tempangle mousedir]`
- `to flock`
  - `setmousedir (towards mouse-xcor mouse-ycor)`
  - `settempangle heading * inertia + (1 - inertia) * mousedir`
  - `if (mousedir - heading) >180 ;; wrap around 0 degrees`
    - `[settempangle heading * inertia + (1 - inertia) * (mousedir - 360)]`
  - `if (heading - mousedir) > 180`
    - `[settempangle (heading - 360) * inertia + (1 - inertia) * mousedir]`
  - `if (tempangle < 0) [settempangle tempangle + 360]`
  - `seth tempangle + (random randomness) - (random randomness)`
  - `seth tempangle`
  - `fd 1`
  - `end`

- Randomness 0 = circles
- inertia controls radius of circles
Flocking Around Each Other

- turtles-own [neighbor]
- to flock
  setneighbor one-of-turtles ;;randomly choose one turtle
  seth towards-nowrap xcor-of neighbor ycor-of neighbor
  rt (random randomness) - (random randomness)
  fd 1
  end
- Or, we can always move to closest turtle with:
  setneighbor who-min-of-turtles [distance xcor-of myself ycor-of myself]
Conway’s Game of Life

- Grid of cells, each with 8 neighbors
- Cells are either alive (1) or dead (0)
- If a cell is alive, then it dies if it has < 2 alive neighbors (loneliness) or > 3 alive neighbors (overcrowding)
- If a cell is dead, it comes alive if it has exactly 3 neighbors
- A Universal Turing Machine can be implemented in the Game of Life
  - Anything that can be computed can be computed in the Game of Life
  - Rule implement logical structures, gates, states, etc.
  - Multiple realizability
Conway’s Game of Life

• **Behavior**
  - Surprising complexity given simple rules
  - Rules produce higher-level structures
    - Stable structures: 4-square
    - Moving structures: gliders
    - Higher-level structures interact

• **Changing the Rules of Life**
  - Birth if odd number of neighbors are on
  - Death if even number of neighbors are on
  - Bridging explanations
The Game of Life as Turing Machine

http://www.rendell.uk.co/gol/tm.htm
One small part: The Outgate
Conway’s Game of Life

- Patches-own [state count]
- To setup
  ifelse (random 100) < density
    [setpc yellow] [setpc black]
  ifelse pc = yellow [setstate 1] [setstate 0]
- To go
  if else pc = yellow [setstate 1] [setstate 0]
  nsum state count ;; find sum of variable “state” for a
  ;; 8 neighbors, and put result in “count”
  ifelse state = 1
    [if ((count < 2) or (count > 3)) [setstate 0]]
    [if count = 3 [setstate 1]]
  ifelse state = 1 [setpc yellow] [setpc black]
  end
Turing Machines

• Turing machines
  – abstract computational framework
  – Infinite tape with 0s and 1s
  – read/write head that reads state at one tape location, and can write a state out to the same location
  – Finite set of states
  – State transitions: if in state W and head reads X then write out symbol Y, move to the left/right one space, and move to state Z
• Universal Turing machine
  – Every Turing machine is just a string
  – Can construct a universal Turing machine that take strings describing other Turing machines as input, and performs the computation the inputted Turing machine would have computed
  – Universal Turing machines can simulate any other Turing machine
• Church-Turing thesis
  – problems solvable by a universal Turing machine are exactly those problems solvable by an algorithm
  – Universal Turing Machines can compute any recursive function, decide any recursive language, and accept any recursively enumerable language
# Turing Machines

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>-&gt;</td>
<td>0</td>
<td>R</td>
<td>s2</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>-&gt;</td>
<td>1</td>
<td>R</td>
<td>s2</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>-&gt;</td>
<td>0</td>
<td>R</td>
<td>s3</td>
</tr>
<tr>
<td>s3</td>
<td>0</td>
<td>-&gt;</td>
<td>1</td>
<td>L</td>
<td>s4</td>
</tr>
<tr>
<td>s3</td>
<td>1</td>
<td>-&gt;</td>
<td>1</td>
<td>R</td>
<td>s3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s4</td>
<td>1</td>
<td>-&gt;</td>
<td>1</td>
<td>L</td>
<td>s4</td>
</tr>
<tr>
<td>s4</td>
<td>0</td>
<td>-&gt;</td>
<td>0</td>
<td>L</td>
<td>s5</td>
</tr>
<tr>
<td>s5</td>
<td>1</td>
<td>-&gt;</td>
<td>1</td>
<td>L</td>
<td>s5</td>
</tr>
<tr>
<td>s5</td>
<td>0</td>
<td>-&gt;</td>
<td>1</td>
<td>R</td>
<td>s1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>State</th>
<th>Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>s2</td>
<td>010</td>
</tr>
<tr>
<td>4</td>
<td>s3</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>s4</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>s5</td>
<td>0101</td>
</tr>
<tr>
<td>7</td>
<td>s5</td>
<td>0101</td>
</tr>
<tr>
<td>8</td>
<td>s1</td>
<td>1101</td>
</tr>
</tbody>
</table>

Step State Tape
--- --- --- ---
1   s1   11   9   s2   1001
2   s2   01   10  s3   1001
3   s2   010  11  s3   10010
4   s3   0100 12  s4   10011
5   s4   0101 13  s4   10011
6   s5   0101 14  s5   10011
7   s5   0101 15  s1   11011
8   s1   1101  -- halt --
Changing the rules of Life

• If a cell is alive, then if it has an odd number of neighbors, it dies.
• If a cell is dead, then if it has an odd number of neighbors, it lives.
• So, a cell changes its state if it has an odd number of neighbors.

• to go
• ifelse pc = yellow [setstate 1] [setstate 0]
• nsum state count
• ifelse state = 1
  • [ifelse (count mod 2) = 1 [setstate 0] [setstate 1]]
  • [ifelse (count mod 2) = 1 [setstate 1] [setstate 0]]
• ifelse state = 1 [setpc yellow] [setpc black]
• end
Other Important StarlogoT commands

- **diffuse chemical 0.8:** Each patch shares 80% of its value on the variable chemical with its 8 neighbors
- **breeds:** creates different kinds of turtles
- **Count-turtles-with [color = red]:** reports count of turtles meeting condition
- **downhill chemical:** returns direction with minimum value of chemical
- **plot:** to create plots of any number of variables simultaneously
- **scale-pc green grass 0 20:** turns patch a shade of green based on variable grass with a range between 0 and 20
- **every 2 [fd 1 wait .1]:** every 2 seconds, go forward 1 and wait .1 seconds
Interactions Between Patches and Agents

- Random walk
- patches-own [steps] globals [minny maxxy]
- to setup
  ca crt 1 setsteps 0 end
- to random-walk
  seth random 360
  fd 1
  tsetsteps steps + 1
  setminny min-of-patches [steps]
  setmaxxy max-of-patches [steps]
  scale-pc red steps minny maxxy
  end
- if pc != black [pstamp pc] ;; to have patch stamp turtle