ComS 229 Project 2 (part 3)

The Game of Life — Generalization

April 14, 2013

0 Cellular Automata

A *cellular automaton* is a regular grid of cells, in any number of dimensions. At any point in time, each cell in the grid may be in one of a finite number of states. The state of a cell evolves over time according to the states of its neigbors, according to various rules. The dimensionality and shape of the grid, the number of states for each cell, and the rules for changing states define the *type* of the automaton. For example, Conway's Game of Life is one type of cellular automaton.

Cellular automata[1] are examples of models of computation. Each type of automaton can be viewed as a type of computer, while the initial configuration can be viewed as a *program* for the computer. Theoreticians are interested in studying the limits of what may be computed with various types of computers.

For part 3 of this project, you will extend your three programs to handle some other types of cellular automata. You will always deal with two–dimensional, rectangular grids.

0.1 Brian's Brain

In Brian's Brain, each cell may be in one of *three* states:

0. Off,
1. Dying,
2. On.

The rules for changing states are as follows.

State at generation n	State at generation $n+1$
Off	On, if exactly two neighbors are in state "on"; Off, otherwise.
Dying	Off
On	Dying

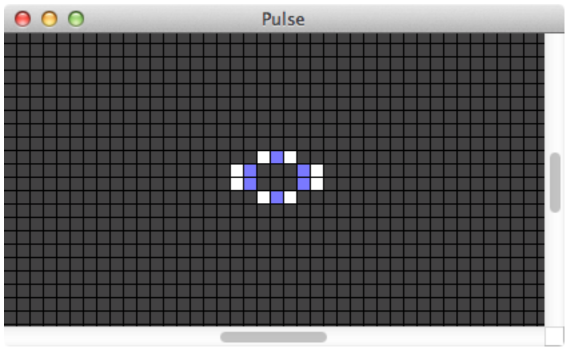
The *neighbors* of a cell are the same as Game of Life — the eight adjacent cells. An example is shown in Figure 1.

0.2 Wire World

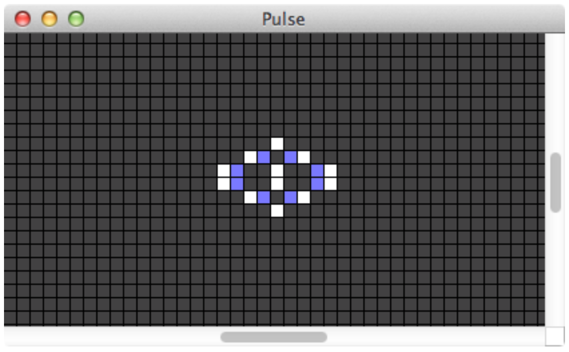
In Wire World, each cell may be in one of *four* states:

0. Empty,
1. Electron head,

[1] "Automata" is the plural form of "automaton".



Generation 3



Generation 4

Figure 1: Example of Brian's Brain (white: on, blue: dying)

2. Electron tail,
3. Wire.

The rules for changing states are as follows.

State at generation n	State at generation $n + 1$
Empty	Empty
Electron head	Electron tail
Electron tail	Wire
Wire	Electron head, if exactly one or two neighbors are electron heads; Wire, otherwise.

The *neighbors* of a cell are the same as Game of Life — the eight adjacent cells.

0.3 Langton's Ant

Each cell is either "black" or "white". An ant moves on the grid and changes the cell colors. Thus, each cell may be in one of the following states:

0. White, no ant
1. Black, no ant
2. White, ant facing north
3. White, ant facing east
4. White, ant facing south
5. White, ant facing west
6. Black, ant facing north
7. Black, ant facing east
8. Black, ant facing south
9. Black, ant facing west

When the ant moves from generation n to generation $n + 1$, it does the following.

1. If its cell is white, turn right (e.g., if facing north, change to east), otherwise turn left.
2. Change the color of its cell.
3. Move forward one step (based on the direction it is facing).

You should be able to handle more than one ant. To make things simpler, we will assume that if more than one ant lands on the same cell at the same generation, they immediately annihilate each other.

1 The programs

There are no new programs for this part. You must update your `showgen`, `sim-tui`, and `sim-gui` programs to handle the new types of automata.

2 Submitting your work

You should turn in a gzipped tarball containing your source code, makefile, and a `README` file that documents your work. The tarball should be uploaded in Blackboard.

Your executables will be tested on `popeye.cs.iastate.edu`. You should **test early, and test often on `popeye`**. We will most likely use some scripts to check your code; this means you should not change the name of the executables or the switches.

3 Grading

The distribution of points corresponds *roughly* to the degree of difficulty and length of time required for each component. Your mileage may vary.

showgen (Life only):	450 points
sim-tui (Life only):	300 points
sim-gui (Life only):	300 points
showgen (part 3):	80 points
sim-tui (part 3):	60 points
sim-gui (part 3):	60 points
Makefile:	100 points

Typing '`make clean`' should remove *all* generated files, and '`make`' should build *all* executables. You may include extra targets if you like.

Usability:	100 points

Style, from the user's perspective. Are the programs easy to use? Are error messages informative?

Documentation & Style:	150 points

Style, from a programmer's perspective. Is the design reasonable? Can another programmer read the comments and understand the implementation? Include a `README` text file to explain the purpose of each source file.

Total:	1600 points

A `.aut` file format

You must handle the following new statements (c.f. Figure 2).

```
Name "Diode";
Xrange -20 20;
Yrange -10 10;
Rules WireWorld;
Chars 32, 35, 43, 46;
Colors (64, 64, 64), (255, 64, 64), (255, 64, 255), (64, 64, 255);
Initial {
  # Set these to state 1
    Y= 3: -9;
    Y=-3: -9;
  State 2;  # The following are set to state 2
    Y= 3: -10;
    Y=-3: -10;
  State 3;  # The following are set to state 3
    Y= 3: -8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
    Y= 2: -1, 0;
    Y= 4: -1, 0;
    Y=-3: -8, -7, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
    Y=-2:  0, 1;
    Y=-4:  0, 1;
};
```

Figure 2: Example `.aut` file for two wireworld diodes.

`Rules` Sets the type of the automaton. This statement must appear *after* the `Xrange` and `Yrange` statements, and *before* the `Chars`, `Colors`, and `Initial` statements. The format is

```
Rules Identifier;
```

where `Identifier` is either "`ConwaysLife`", "`BriansBrain`", "`WireWorld`", or "`LangtonsAnt`". If this statement is missing, the default should be "`ConwaysLife`".

`State` This statement appears inside an `Initial` statement, and sets the state value for all following coordinates. The format is:

```
State <integer>;
```

Before the first `State` statement, the state value is one.

The following existing statements must be updated.

`Chars` The format is now

`Chars state0, state1, ..., state`n`;`

i.e., characters are specified for each of the $n+1$ states.

`Colors` The format is now

`Colors (r0, g0, b0), (r1, g1, b1), ..., (r`n`, g`n`, b`n`);`

i.e., colors are specified for each of the $n+1$ states.