335 — Algorithm Engineering — Cella Rule 90

Project #1 – Cella Rule 90

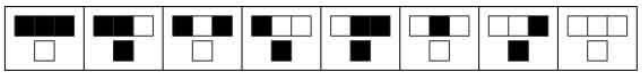
Introduction

This project is to write a program to display the generational progress of Wolfram's Rule-90 cellular automaton. The program will be written in Javascript with an HTML web page for display.

The cella "growth generations" will be shown in a 2D grid of black and white cells. Each row after the top will show the next generation.

Rule-90

Wolfram's Rule-90 (from his 2002 book "A New Kind of Science") is based on a 1D array where each cell is "active". What happens to it depends on its current binary state (1 or 0, white or black) and the states of its two neighbors; 3 cells in all. With 3 binary cells, there are 8 possible configurations. A Rule needs to specify what happens to a cell with each of those 8 neighborly configurations. Rule-90 looks like this:



rule 90 Wolfram, 2002 p53

This Rule format is interpreted as follows: leftmost is a cell containing all black (1's, or filled cells). If the center cell and its two neighbors are in state 1, then in the next generation, the cell will change to state 0 (white, clear). Put differently, if the binary number represented by the 3 cells is 111 = 7; then the middle cell will be changes to state 0 in the next generation. A similar analysis is used for the other 7 triple-cell states. Now, if we treat the next generation states as 8 bits of a binary number, then these 8 configurations generate (from left to right) the bits 01011010, which equals a decimal 90. Hence, this set of state transitions based on 3-cells each is called Rule-90.

Setup

Your program should initialize a 400 by 400 square grid to have all cells empty (state 0). Then set the top row's 200th (just left of center) cell in state 1. This represents the initial (#0 == seeded) generation. Include your team name on the web page above the grid.

Running

After setup, your program should show on the next row down the next (#1) generation of Rule-90 operating on every row cell. You should presume that a "neighboring cell" off the right or left side of the grid is empty. (This limits the right and left border cells to only 4 of the 8 possible configurations, each.) Similarly, continue running until all 400 rows (generations) have been shown. Then stop.

NB, generations #0 thru #3 should look like this (roughly, using empty space (and a hyphen) for state = 0 cells and X for state = 1 cells):

```
----X----
---X-X---
--X---X--
-X-X-X-X-
```

Complexity Order

You should prepare a 1-page paper describing your analysis of the Big-O running time of your algorithm. Address the usual issues such as main operations, input size, etc.

Team

Your team may contain up to four members. We would prefer larger teams. Pick a three-letter name for your team (e.g., “ABX”). [For the next project, teams can be changed.]

Project Reports

Tasks (WBS): Slice the project up into tasks. List each such task. If a task needs to be split into sub-tasks, the list these and indicate which task they are a sub-task of. At the beginning, this does not have to be accurate or final -- it changes as you develop and need to make changes to the list of tasks.

Progress Board: Which WBS tasks have been A) Begun, by whom? B) Completed (and can be demonstrated) by whom, when? C) Verified (QA'd) by whom, when? Include the current Progress Board file at the end of each Standup Status report.

Standup Status, twice weekly. The Standup Status Report is due Sunday by 11pm and on the day before the last class session of the week at 11pm. One report per team. It should contain, for each team member, their Standup answers to the 3 standard Standup questions: Q1. What task(s) have you completed since last status? Q2. What task(s) do you plan to complete by next status? Q3. What obstacles are blocking your progress (on which task(s))? Consider sub-dividing a WBS task into "Half-Day" sub-tasks so as to be able to have a task completion (or more) for each status. Note that if a team member takes on a WBS task and sub-tasks it, then the sub-tasks should be added to the WBS and shown on the Progress Board. Also, note that keeping track of tasks, new/old sub-tasks, and completions takes effort, but can be streamlined. Don't forget that merging your completed task code into the team's mainline code requires integration testing. Tasks that have gone through QA should be demonstrable in class.

These documents should be delivered as .**pdf** files, and each filename should include your course number, your team name, the doc type (WBS, Progbrd, Standup), and the date in YYMMDD format. E.g., "`543-ABX-Standup-190212.pdf`".

Readme File

You should provide a README.txt text file. Be clear in your instruction on how to build and use the project by providing instructions a novice programmer would understand. If there are any external dependencies for building, the README must also list them and how to find and incorporate them. Usage should include an example invocation. A README would cover the following:

- Class number
- Project number and name
- Team name and members
- Intro (including the algorithm used)
- Contents: Files in the .zip submission
- External Requirements (None?)
- Setup and Installation (if any)
- Sample invocation
- Features (both included and missing)
- Bugs (if any)

Academic Rules

Correctly and properly attribute all third party material and references, lest points be taken off.

Submission

All Necessary Files: Your submission must, at a minimum, include a plain ASCII text file called **`README.txt`**, all project documentation files (except those already delivered), all necessary source files to allow the submission to be built and run independently by the instructor. [For this project, no unusual files are expected.] Note, the instructor not use use your IDE or O.S.

Headers: All source code files must include a comment header identifying the author, author's contact info (please, no phone numbers), and a brief description of the file.

No Binaries: Do not include any IDE-specific files, object files, binary **executables**, or other superfluous files.

Project Folder: Place your submission files in a **folder named** `X-pY_teamname`. Where X is the class number and Y is the project number. For example, if your team name is ABC and this is for project #2 in class CS-123, then name the folder "`123-p2_ABC`".

Project Zip File: Then zip up this folder. Name the .zip file the **same as the folder name**.

Turn in by 11pm on the due date (as specified in the bulletin-board post) by **sending me email** (see the Syllabus

for the correct email address) with the zip file attached. The email subject title should include **the folder name**.

ZAP file: If your emailer will not email a .zip file, then change the file extension from .zip to .zap, attach that, and tell me so in the email.

Email Body: Please include your team members' names and campus IDs at the end of the email.

Project Problems: If there is a problem with your project, don't put it in the email body – put it in the README.txt file.

The Cloud: Do not provide a link to Dropbox, Gdrive, or other cloud storage. Note, cloud files (e.g., G-drive) are not accepted.

Grading

- 75% for compiling and executing with no errors or warnings
- 10% for clean and well-documented code (Rule #5(Clean))
- 10% for a clean and reasonable documentation files
- 5% for successfully following Submission rules