CPSC-335-02 AI — Project #1 — Cella Rule 150

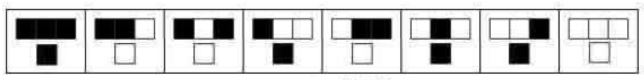
Project #1 – Cella Rule 150
Introduction

This project is to write a program to display the generational progress of Wolfram's Rule-150 cellular automaton. The program will be written in Javascript and P5.JS (for animation) 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; the top row being the initial setup generation. Each generation is a stage in the "life" of the cellular automaton.

Rule-150

Wolfram's Rule-150 (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 – or the two colors of your choice that are distinguishable by us) and the states of its two neighbors; 3 cells in all. With 3 binary cells controlling what happens to the center cell in the next generation, there are 8 possible 3-cell configurations (8 = 2^3). A Rule needs to specify what happens to a cell with each of those 8 neighborly configurations. Rule-150 looks like this, (showing a section for each of the 8 3-cell triples) with the top row of each section being the old generation 3-cell triple, and the bottom cell representing what the "growth-change" was for the center cell of the old 3-cell triple in the creation of new generation (black is filled -- state 1, white is empty – state 0):


rule **150** Wolfram, 2002

This Rule format is interpreted as follows: leftmost is a section top 3-cell triple containing all black (1's, AKA 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 changes to state 0 in the next generation. A similar analysis is used for the other 3-cell triple states. Now, if we treat the next generation 1-cell states as 8 bits of a binary number, then these 8 1-cell configurations generate (from left to right) the binary bits 01011010, which equals a decimal 150. Hence, this set of 8 state transition rules based on 3-cell triple is called Wolfram's Rule-150.

Setup

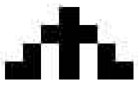
Your program should initialize a 401 by 401 square grid to have all cells full (state 1). Each row will represent a growth generation, with the top row being the initial generation. Then set the top row's 201st (the center) cell in state 0. This represents the initial (0th == "seeded") generation. Make your team name visible on the web page above the grid. (If you wish, you can reverse the colors if that seems more pleasing.)

Running

After setup, your program should show on the next row down the next (#1) generation of Rule-150 operating on every row cell. It does this by analyzing each 3-cell triple, say, from the left – moving a 3-cell "window" to the right by 1 cell, and generating the correct center cell state on the row below the triple-cell. It would be convenient if you placed a cell border (say of yellow or red) just inside the cell of the next generation that you are creating so that it is easy to follow which cells are being generated. You can use the P5.JS draw() function to perform this animation change.

Borders: 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 updating the animation.

NB, generations #0 thru #3 should look like this:



Architecture Analysis

You should prepare a 1-page paper (or shorter) describing your algorithm using the concepts described in class.

Team

Your team may contain up to four members. We would prefer larger teams. Pick a four-to-eight-letter name for your team (e.g., "Bozobus"). You can also include digits after the first letter. [For the next project, teams, and their names, can be changed.]

Project Reports

Standup Status Report, twice weekly: The Standup Status Report is due on or before Friday noon and Sunday midnight, to split up the week's reporting into 2 segments. One report per team – but everybody in your team should be CC'd. It should contain,

o- The **3 Standup Q&A** for each team member

o-The current **Progress Board**, listing WBS tasks and Task phase columns

o- The current **WBS**, tasks & sub-tasks of work on the project

The **3 Standup Q&A**, 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 big tasks into "**Half-Day Rule**" sub-tasks so as to be able to have a task completion (or two) for each status – tasks for which completion is easily **visible** (AKA demonstrable).

WBS: The Work Breakdown Structure is a task hierarchy for the project. You create sub-tasks as you understand what is needed. You don't need to have the details at the start of the project;. (It is always helpful to try to identify a task before you do it.) Example: ID: T24, Name: "Make Tail Follower" Mom: T5 . Here, the kid task T24 is a sub-task of the Mom task T5. For top-level tasks, explicitly indicate T0, the whole project. Build numbers as you go, skipping some if you like. You can change your task hierarchy at any time.

The **Progress Board**, lists each WBS **Task-card** in the **WIP** columns.

WIP: The Work-In-Progress columns: Ready, Working, QA, Done. Ready means the task is clear enough so it can be worked on. Working means a team member has taken ownership and begun work on it. QA means the work has been completed, but now needs a visual check by another team member. Done means the task is finished.

Task-card: A list of the task ID (eg, T24), the team member working it if any (eg, initials "CS" for us), a start date-time if any (eg, "2/7.10"), a QA team member if any (again initials), and an end-date-time if any. A Task-card in the done WIP column would have all these things. 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 a bit of 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 and section, your team name, the word "Standup", and the date in **YYMMDD format**.
E.g., "**335-02-Bozobus-Standup-200204.pdf**".

CPSC-335-02 AI — Project #1 — Cella Rule 150

Readme File in Final Project Submission

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** `XXX-YY-teamname-pZ`. Where XXX-YY is the class number and section (eg, "335-02") and Z is the project number (1, 2, 3 or 4).
E.g., for team Bozobus's third 335-02 project you would use "**335-02-Bozobus-p3**".

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**.

JS files: For each JS file, change its name by adding a ".txt" extension to it, producing "foo.js.txt". This is to workaround a few email systems that refuse to send a zip file containing JS files.

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, Github or other cloud storage, unless you have tried and failed to send a zip file.

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