335 — Algorithm Engineering — Cella Ant 12

Project #1 – Cella Ant 12
Introduction

This project is to write a program to display the generational progress of Turk & Propp's Ant #12, a cellular automaton (Cella). (CF, en.wikipedia.org/wiki/Langton%27s_ant#Extension_to_multiple_colors) The program will be written in Javascript with an HTML web page for display.

The cella will be shown in a 2D grid of black, red, yellow, blue (etc.) cells.

TP Ants

Turk & Propp's Ants (TP Ants) crawl on a 2D grid where each cell has a color (initially all are black). As an Ant crawls, it changes its direction/heading and it also changes the color of the cell under it. In particular, the cell's color is "incremented" to the next color in sequence (wrapping around to the 0th color if needed). And each color is associated with a change of direction: to the Left or to the Right of the Ant's current heading.

Initially, the grid is all black and the Ant is in the center cell, heading West (toward the left).

TP Ant #12

For Ant #12, 12 decimal = 1100 binary ==> LLRR, where a binary 1 means turn Left, and a 0 means turn Right. Each cell has 4 states, one for each binary digit position in the four digits, 1100. Each state is represented on the grid by a color (for easy visuals), with the color/state sequence (from low bit to high bit) being Black=0 → Red=1 → Yellow=2 → Blue=3. Hence, Ant #12 turns Right on Black or Red, and else Left. Also, treat the Ant as a (reasonably visible) white triangle which sits on top of the cell color and points toward the cell edge it is heading toward.

Setup

Your program should initialize a 41 by 41 square grid to have all cells black (state 0). The lone Ant should be in the center cell.

Running

After setup, your program should run at least for 1,000 moves, showing the grid changes after each move.

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 3-4 sized teams. Pick a three-letter name for your team (e.g., "ABX"). [For the next project, teams (and names) can be changed.]

Project Reporting Data

Tasks (AKA "WBS"): Slice the project up into tasks, as best you can. (You can change your task list as you progress.) If a task needs to be split into (kids) sub-tasks, then indicate a sub-task's mom, etc. (It is sometimes handy to not only have a title for a task but to also give it a numeric ID to refer to easier.)

Progress List: A) Ready: What WBS tasks have been thought of be have yet to be begun. B) Begun, by whom? B) Completed (has been demonstrated to a team member) by whom?

Standup Status Report, twice weekly. The Standup Status Report is due Monday's and Friday's by noon. One report per team, CC'ing the other team members. It should contain, team name, the member names, and the current Progress List. This documents should be delivered as **.pdf** file and the filename should include your course and section number, project number, your team name, the document type (Standup), and the date in YYMMDD format. E.g., "`335-02-p1-ABX-Standup-190212.pdf`".

Readme File

When your project is complete, 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