335 — Algorithm Engineering — Cella Larks Ant 34

Project #1 – Cella Larks Ant 34
Introduction

This project is to write a program to display the “computing” progress of a 2D cellular (“cella”) Turing Machine (TM) “Ant”. 2D cella ants started with Langton's Ant (1986) where the TM ant was placed on a grid and whose “brain” programming 1) changed its “nose” direction based on the color of the grid cell it was on (Left for Black and Right for White), 2) flipped the (black or white) color of the cell, and then 3) moved to the neighboring cell. You can read about this at the Wikipedia page for ***Langton's Ant***.

Turk & Propp (1994) (TP) extended Langton's Ant to use more colors, and add a color index code to determine the direction in which to change the ant's “nose”. Read about it in that page's section on ***Extension to Multiple Colors***. The color index was written in binary with 1 indicating a Left turn and 0 for a Right turn. There was one bit per color. For example, let the colors be RGB where Blue is bit index 0, Green is bit index 1, and Red is index 2. Then let the binary color index code for Left/Right turns be 110. So, 110 maps both to RGB and to LLR. It maps to RGB by bit position with high bit being R and middle bit being G and low bit being B. It maps to LLR because the high big is 1 meaning Left, the middle bit being 1 means Left, and the low bit being 0 means Right. So for nose direction changes, a Blue cell means turn Right, and a Red or Green cell means turn Left. Also, when changing cell color, instead of flipping, we increment the color index modulo 3 (eg, from color 0=Blue to 1=Green to 2=Red and wrapping around back to 0=Blue, again). There are a few pictures to illustrate what goes on at the Wiki page.

We extend the TP ant to a “Larks” ant by adding a mode whereby the Larks ant will sometimes ignore the color for nose direction changes but instead continue in a straight line for awhile. (Larks, or LRCS, means Left-Right-Color-Straight.) This works as follows. We extend the TP binary color index code to a trinary color index code. Like the TP ant, each trinary digit indicates a behavior, where 0 still indicates turn Right, and 1 indicates turn Left, but 2 indicates Go Straight for awhile. Here is the Larks ant brain cell processing in detail:

```
1. Pick Procedure based on current Mode (either LR Mode, Set-Count Mode, or Countdown Mode):
If in LR Mode then // Left-Right Mode
  2. Change nose direction based on your cell color and its code (similar to the TP ant)
  3. If cell color is code 2 (indicating straight), then change to Set-Count Mode // in prep for the next cell
Else if in Set-Count Mode then // we'll go straight, no change in direction
  2. Set the counter to your cell's color index (based on the color indexes for the type of Larks ant)
  3. Change to Countdown Mode // in prep for the next cell
Else you're in Countdown Mode then // we'll go straight, no change in direction
  2. Decrement the counter
  3. If counter is 0, then change to LR Mode // in prep for the next cell
4. Increment cell's color // modulo the number of colors of Larks ant
5. Move to neighbor cell
```

So, the Larks ant is a bit more complex than the TP ant. Sometimes it decides to go straight based on the cell color index (for a count indicated by the next cell it steps into), and while it goes straight it doesn't try to turn until it's finished counting down to zero.

Because we'll be running the Larks ant on a finite grid (rectangle) in a browser webpage, we will have the ant wrap around to the opposite cell if it tries to “step off” an edge of the grid.

We will build and run the Larks ant with trinary code of 1021 (base 3) which is a decimal 34. The colors will be black, red, yellow, and blue, in that order, with indexes 3, 2, 1, and 0. So, on a Black or Blue cell the ant would turn Left in LR mode. On Red the ant would turn Right in LR mode. And on Yellow in LR mode the ant would switch to Set-Count Mode and not turn (ie, it would go straight for awhile).

The Larks #34 ant will start in an all black 60 by 40 grid (with each grid cell being 10 by 10 pixels) at cell location (30,20), and the ant's nose will be facing up (north) toward the cell (0,20) at the top-edge of column 20.

The program will be written in **P5.js**+**Javascript** with an **HTML** web page for display (as described in lecture). You do not have to show grid lines or row/column numbers.

Running

After setup, your program should run for 2,000 and moves, showing the grid changes after each move. You do not have to show the ant, itself – merely show each cell color changes.

Team

Your team may contain up to four members. We would prefer 3-4 sized teams. Pick a short-ish (<= 12 letter and/or number) name for your team (e.g., "ABX", or "Groggy", or "we-like-cats", etc.).

Project Development Reporting

Standup Status Report, twice weekly. The Standup Status Report is due **Monday's** and **Friday's** by noon-ish. One report per team, **CC'ing the other team members**. It should contain, team name and the member names. This documents should be delivered **as a PDF file**; and the **filename** should be in the following format: include your course and section number, project number, your team name, the document type (Standup), and the date as YYMMDD:. E.g., "`335-02-p1-Groggy-Standup-210231.pdf`". ("02" or "03" is your section.)

Standup Status Report contents should be, for each team member, a list of the 3 **Standup question short answers**: Q1: what have you **completed** (name sub-tasks) by the time of this Standup report; Q2: what do you **plan to complete** (name sub-tasks) by the time of the next Standup report; and Q3: what obstacles if any (1-line description) are **currently blocking you** (for which you've reasonably tried to find the answers by yourself, including asking your team about them – known as "due diligence"). Note, that you can email the professor, or ask questions during office hours to get answers.

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** like your Standup report files: `335-02-p1-Groggy`.

Project Zip File: Then zip up this folder. Name the .zip file the **same as the folder name**, like `335-02-p1-Groggy.zip`. Turn in by 11pm on the due date (as specified in the bulletin-board post) by **submitted via**

emailed zip file(s) (preferred), or via accessible cloud (eg, Github, Gdrive, Dropbox) with emailing the accessible cloud link/URL. See the Syllabus for the correct email address. The email subject title should include **the folder name**, like `335-02-p1-Groggy`. Note that some email-programs block .ZIP files (but maybe allow you to change .ZIP to .ZAP) and block sending zipped files with .JS files inside (but maybe allow you to change .JS to .JS.TXT).

Email Body: Please include your team members' names (but not your 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, under an "ISSUES" section.

Grading

- 80% for compiling and executing with no errors or warnings
- 10% for clean and well-documented code (Rule #5(Clean))
- 5% for a clean and reasonable documentation files
- 5% for successfully following Submission rules