335 — Algorithm Engineering — Larks Ant 0121

Project (Get Acquainted Phase) – Larks Ant #0121
Introduction

This project is to write a program to display the "computing" progress of a 2D cellular Turing Machine (TM) "Ant". 2D ants started with **Langton's Ant** (1986) where the TM ant was placed on a grid and whose "brain" programming 1) read the color of the cell it is on, 2) based on that color the bot performed the color-indicated action (changed its "nose" direction based on the color of the grid cell it was on, Left for Black and Right for White), 3) incremented (flipped) the (black or white) color of the cell (the second part of the "action"), and then 4) moved to the neighboring cell, following its "nose". You can read about this at the Wikipedia page for ***Langton's Ant***.

Let's reiterate what the Langton bot does: 1) a read the cell color, 2) take the color-indicated action, 3) increment/write the cell color (the "second part" of the action), and 4) move (following nose) to the next cell.

Turk & Propp (1994) (TP) extended Langton's Ant to use more colors, and add a color "action" index code to determine the action – the direction in which to change the ant's "nose". (This is why we said "increment" instead of merely "flip", above.) Read about it in that page's section on ***"Extension to Multiple Colors"***. In a TP ant, the direction is changed (based on color), the cell color is "incremented" to the next color (ie, "modified in a cyclic fashion") in the color sequence (for this ant) with wrap-around if needed, and then the ant moves (in its "new" direction) to the next cell. TP ant actions are only change direction Left or Right. Hence these ants are given names like LLRR (for a 4-color sequence of, say Blue, Puce, Lavender, and Pink – where the bot turns Left in a cell of Blue or Puce). Other actions – also directional changes – mentioned on that Wikipedia page are N (no change), U (180), and for a hexagonal grid, R1 (60 clockwise), R2 (120 clockwise), L2 (120 counter-clockwise), L1 (60 counter-clockwise). Langton's ant was given the name "LR" for Left on Black and Right on White.

Color and Action Index

The color of the cell (that the bot is in) determines the action (eg, a directional change, etc.) the bot will take. And before the bot moves, each cell's color is incremented to the next color in sequence (with the last color incremented back to the initial color – wraparound) – the "second part" of the action.

For the two color sequence (B&W) the Langton bot's actions would turn Left for Black and Right for White. The action Left is given an action index of 0 and Right is given an action index of 1, to simplify coding the ant's TM "brain" (a finite state machine, an FSM – but it has no state data). So, the translation from cell color to action index to action is Black → color[0] → action[0] → Turn Left, and White → color[1] = action[1] → Turn Right.

The TP ant called "LLRR", above, would map

Blue → color[0] → action[0] → Left,
Puce → color[1] = action[0] → Left,
Lavender → color[2] = action[1] → Turn Right, etc.

Color in a Browser

Note that in drawing/painting Black or White we use (8-bit) Red-Green-Blue (RGB) luminence levels of 0:0:0 = 0 for Black and FF:FF:FF = FFFFFF for White, as hexadecimal integers. So a more detailed translation from cell color to action was Black = 0:0:0 → 0 → Turn Left, and White = FF:FF:FF → 1 → Turn Right. (And for Browser canvas drawing colors, there is a fourth "A channel" for "transparency", using RGBA values, but we can ignore the (8-bit) A (for "Alpha") transparency channel.

The Larks Ant

So far, the Langton and TP ants need to maintain no internal state – the data needed to decide on an action is in each cell.

335 — Algorithm Engineering — Larks Ant 0121

We extend the ant to a "Larks" ant by adding a mode whereby the Larks ant will sometimes ignore the color changes but instead continue in a straight line for sequence of cell moves; which requires maintaining some state, a countdown. (Larks, or LRCS, means Left-Right-Countdown-Straight.) This works as follows. '

First, as usual our K-color sequence is indexed color[0]..color[K-1].

Second, each color (equivalently, each color index) can map to one of 3 actions: Turn Left, Turn Right, and Straight-Countdown.

Third, the bot FSM is in one of 3 states/modes: Normal or Straight or Countdown mode.

Here is the Larks ant brain FSM processing in detail:

1. Read the cell's color, translate to its color sequence index.
2. Obtain the action given the color index, & store it in the FSM counter // Just in case

If FSM is in Normal mode and an L/R action then

3. Change nose direction accordingly

Else If FSM is in Normal mode and a Countdown-Straight action then

3. Change to Countdown mode & don't change direction

Else if FSM is in Countdown Mode then // we'll go straight, no change in direction

3a. If FSM counter is non-positive, then change FSM to Normal Mode

3b. Decrement the FSM counter & don't change direction

4. Increment cell's color modulo the number of colors // ie, with wraparound
5. Move to neighbor cell // in nose direction

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 previous cell it had been in), 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.

The colors will be black, blue, yellow, and red, in that order, with color indexes 0, 1, 2, and 3.

The actions will be Turn Left, Turn Right, and Straight-Countdown, with action indexes 0, 1, and 2.

The color-action map will be 0 → 0, 1 → 1, 2 → 2, 3 → 1. So the action order is 0121.

The Larks #0121 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.). If you are on your own, pick a 1-person team name. If you want to join or form a team but can't (?), tell me at start of class and I will hold an auction for you.

Naming Docs

Name your documents/files like this: "`335-03-p1-Groggy-<type>-<yymmdd>.<ext>`" where <ext> is "pdf", "zip", etc., and where <type> is "Standup", or missing for the project zip file; and where `<yymmdd>` is for example 210903 for Sept 3rd.

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, **but no CWIDs** (which aught to be private). 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