CPSC 565: Emergent Computing, Winter 2020

Assignment 3: Genetic Algorithm

Due Date: Friday, April 3, 2020

Bob's Fountain Repair has been given the contract to repair one hundred identical fountains that have been damaged. Each fountain has had some tiles broken and they need to be replaced. To help automate this job, Bob has purchased Tilly, the tile-laying robot. The problem is that Tilly needs to be programmed to efficiently repair all the different fountains.

The Fountains

Each fountain is essentially a raised platform of 100 tiles arranged in a 10X10 grid. Each fountain has been vandalized by having individual tiles broken. Each tile in each fountain has had a 50% chance of being broken. Note, this does not mean that each fountain has had 50 tiles broken. Statistically it is possible (but very unlikely) that a fountain could have had all 100 tiles broken or no tiles broken at all. Thus each fountain is a similar, but not identical, repair job.

Tilly

Tilly has a predefined set of sensors and actions it can perform. Tilly is able to sense the state of the tile it is on as well as the state of the four tiles in the Von Neumann neighbourhood to a radius of one around it. Each of these five visible tiles is in one of three states: good tile, broken tile, or over the edge of the 10X10 grid. Tilly is able to perform the following actions while on any given tile: do nothing, move north (one tile), move west, move south, move east, move a random direction, or replace a tile. While Tilly has an unlimited supply of tiles its battery can only supply enough power to take 200 actions on a job (even if some of those actions are to do nothing).

Costs and Profits

Tilly's actions result in either profits or costs for Bob's business. For every broken tile that is successfully replaced Bob earns $10, and the state of the tile is changed to good. For every unbroken tile that is unnecessarily replaced it costs Bob $1. And for every time Tilly falls off the edge of the fountain it costs Bob $5. Obviously Bob would like to program Tilly to maximize his profits over the 100 fountains he needs to repair. For each fountain job Tilly starts in a corner of the fountain's grid. For each of Tilly's 200 actions on the job it must check the state of the five visible tiles, select and perform an action based on that neighbourhood, and accrue any profit or cost of the action. If an action results in Tilly falling off the grid it is replaced to its previous position before the next action is taken. A good solution for Tilly should, in theory, net Bob an average profit of $500 per fountain over all the jobs.

Implementation Requirements:

Design and implement a Genetic Algorithm to evolve a successful solution for Tilly to repair any given fountain. You may use Python, Java, or C/C++ for this but your code must be able to be compiled (if needed) and run from the terminal command prompt. You may not use any third party or custom libraries, your code should run with standard libraries only. Python is the preferred language.

Your program should save the genome of the best solution to a text file when it completes a run of a certain number of generations. A filename that indicates an average fitness score and number of generations would be useful.

Your program should include either a text based console output or graphical display of the best fitness score in each generation as the algorithm is running. You will need this for the write up. (see example in class)

You should include a brief program that takes your genome text file as input and animates a job run of Tilly using that solution. This should be easily viewed by a human in order to observe the solution in action. It does not need to be fancy or have beautiful artwork, simply enable you to watch Tilly at work. You will need this for the write up as well. (see example in class using Python Turtle Graphics)

You must provide a brief write up including the following:

- Describe how you have encoded solutions into genomes

- Briefly describe the selection, cross-over, and mutate functions you have created
- How does changing the parameters (mutate rate, population size, selection probabilities, etc) affect convergence rate?
- Watch an animation of a solution with a fitness score around 100, what bad behaviours does Tilly exhibit that could be blamed for this low score?
- Watch an animation of a solution with a fitness score between 200 and 300, what behaviours have improved, what is still bad?
- Watch an animation of your best scoring solution (hopefully around 500), what unexpected behaviours seem responsible for its success?

BONUS – create a solution for Tilly by hand and run it in your animation, how does it compare? Do this after you have a working GA but before you watch and analyze the other required animations for best results.

Submission:

Submit all required source code, a readme file of how to compile and run your code, and your write-up, using the D2L system under the corresponding entry in Assessments/Dropbox.

<u>HINTS!!!!</u>

Streamline your genomes for faster execution – Tilly will never be on a tile that is over the edge of the grid, Tilly will never have over the edge tiles on more than one visible tile except in corners, etc. This is the GA equivalent of "junk DNA".

Watch your output to get a feel for how many generations seem to create a solution with fitness of 100, 250, or 400+ so you can stop your algorithm at these points and save out a genome to animate for your write-up.

Parameters you can play with to improve convergence – number of robots in a generation, mutation rate, mutation radius, number of crossover points, number of runs averaged to calculate fitness, etc.

When designing your selection function give thought to exploration vs exploitation.

Be sure you understand what the fitness function really is in this assignment.

The fountains are a 10X10 grid, but should your algorithm's internal representation be 10X10 or something else?

Consider working in a group. This is not a truly difficult assignment but it can be time consuming. Groups of up to three members will be allowed.

Ask for help or clarification if you are stuck ….

Grading:

Your grade will be a letter grade based on the following rubric.

A+	All requirements for an A plus either: Bonus hand made solution Excellent fitness scores with low generation counts Just really good work as a whole
A	All requirements met as listed Good documentation and analysis
A-	One or two minor errors in implementation, or Documentation or analysis not quite adequate
B+	One or two minor errors in implementation, and documentation or analysis not quite adequate, or A significant implementation error but no other problems
B	A significant implementation error as well as one or two minor issues and docs or analysis lacking

B-	A significant implementation error, one or two minor issues and docs or analysis lacking
C+	A little better than a C
C	Multiple errors of note, documentation or analysis completely inadequate or missing, but an effort was made
C-	A little worse than a C
D	Little or no effort made, solution produces some results
F	Submission missing or non-functional