Project:
Game of Life

Conway's Game of Life

- The Game of Life
  - Is a cellular automation which imposes fixed rules on the cells.
  - The game is "played" on an infinite two-dimensional discrete grid.
  - Successive generations are dependent *only* on the previous generation: there is no input of any kind once the process starts.
  - Because of the two-dimensional nature of the game, each cell has exactly eight neighbors.

The Rules

- The Rules:
  - Each “live” cell will die if it has fewer than two or more than three neighbors
  - Each “dead” cell will come alive if it has exactly three neighbors.
  - For details, please see:
    - Conway's Game of Life
    - Life Wiki

Solution

- In our Repo there is a package called “life” which contains all of the code you need for going from one generation to the next;
- You may *not* start with any fixed patterns;
- Your starting patterns *must* be generated via some random process such as genetic algorithms, or neural nets.

Genetic Algorithms

- If you choose to use GA (and I think you should), then I suggest the following:
- Jenetics
- Or you can build your own;
- Or you can choose something else.

Genetic code (chromosomes)

- For GA, you must choose a genetic code.
- Here's what I suggest (but you can use anything you like):
  - 3 bits: direction
  - 2 bits: set on/off; flip; as is;
  - N bits: length:
    - 0 (telomere) stop
    - 10: move by one
    - 110: move by two, etc.

Mutation

- For GA, you must decide how to do mutation.
- Here's what I suggest (but you can use anything you like):
  - Choose a random number r from 0..M
  - If r=0, then flip bit;
  - If r=1, then repeat bit;
  - If r=2, then repeat two bits;
  - If r=3, then copy the whole chromosome;
  - Otherwise, leave as is.

Genetic Algorithms

- Genetic algorithm (GA):
  - GAs use the same techniques as does nature in order to find a good (not necessarily the best) solution for a problem.
  - GAs are non-deterministic: you do not expect them to give you the same solution each time you run.
  - GAs are useful for problems that have *no analytical solution*, especially problems with are *NP*.
  - GAs are well-suited to problems with huge solution spaces.

GAs Continued

- Remember the Newton-Raphson Approximation from the first week? These are its properties:
  A. Solves a one-dimensional problem;
  B. Looks for a good solution (but not necessarily the best);
  C. Each generation has exactly one candidate.
  D. Is not guaranteed to converge on a solution;
  E. Is susceptible to bad initial guess;
  F. Iterative steps (successive generations) are based on (known) slope of function for current estimate;

GAs Continued

- GAs have the following corresponding properties:
  A. Can solve an *N*-dimensional problem where *N* is large;
  B. Looks for a good solution (but not necessarily the best) [Same];
  C. Each generation has *M* candidates where *M* can easily be in the millions;
  D. Similarly, may not converge—although non-convergence is less likely;
  E. Is much less susceptible to bad initial guess because of §*C*;
  F. Iterative steps (successive generations) are based on random mutations and/or crossover.

Natural Selection

- Natural selection was first proposed as the mechanism for the evolution of life forms in *On the Origin of Species* by Charles Darwin (1809-1882) and also by Alfred Russel Wallace (1823-1913).
- The idea was based on two major observations:
  - Domestic animals and plants change their phenotype (traits) by selective breeding;
  - Wild animals and plants appear to be particularly well-suited to the environment in which they are actually found (Darwin's finches in the Galapagos Islands).
- Could it be that the same mechanism was at work in both cases, albeit rather slower in the wild case?

Genetics

- The mechanisms of genes were totally unknown in Darwin's day (and, of course, the discovery of DNA was a long way in the future);

- Gregor Mendel (1822-1884) is recognized as the discoverer of genetics, although he was equally clueless about the actual mechanism.

Genetics (Nature)

- There are five important concepts you need to understand:
  - Genotype: the set of replicable and heritable information contained within the cells of an organism [especially the genes in which we are particularly interested]; Sometimes referred to as the Genome.
  - Expression: the “mapping” of genotype to phenotype;
  - Phenotype: the set of properties of an organism which in some way interact with the environment (i.e. traits);
  - Environment: the natural eco-system in which an organism thrives;
  - Fitness: how well the organism is suited to the environment—how likely is the organism to survive long enough to reproduce.

Genetics (GA)

- These are the corresponding five concepts in GA:
  - Genotype: the set of replicable and heritable information which is a property of a candidate solution (“organism”)—the genes;
  - Expression: the “mapping” of genotype to phenotype—the mapping between gene and trait does not have to be 1:1;
  - Phenotype: the traits of the candidate which affect how good a solution it is;
  - Problem: what we are trying to solve—corresponds to environment in nature;
  - Fitness: how well the candidate solution solves the problem.

Other concepts

- Mutation
  - In practice, genotypes cannot be copied from parent to child 100% perfectly. Perhaps fidelity is 99.999999%
  - In asexual organisms (e.g. bacteria), mutation is the *only* source of genetic diversity, but is generally detrimental.
- Crossover (I don't think you will need this for Life project)
  - In sexual reproduction, different parts of the genotype are inherited from different parents.
  - In sexual organisms, crossover is the major source of genetic diversity (but mutation is still possible, though relatively rare).
  - Crossover essentially speeds up evolution.
  - In sexual organisms, fitness is not the only factor determining reproductive success: sexual selection (attraction) is equally important. You can ignore attraction in this project: just mate randomly.

Implementation

- Here, I am going to *require* a strategy for implementing the five concepts:
  - Genotype:
    - Your genotype **must be different from** your phenotype—it represents the heritable aspect of each organism (candidate);
    - your “genes” (individual elements of the genotype) can be coded any way you like—you don’t have to use the four DNA bases (although you can);
    - the simplest mechanism for implementing crossover is to divide the genotype into two or more *chromosomes*, each containing one or more “genes.”
  - Expression:
    - this is arbitrary but is obviously dictated by your choice of genotype and phenotype.

Implementation (continued)

- Implementation continued:
  - Phenotype:
    - this represents a candidate solution to your problem, in this case, the Game of Life problem, then the phenotype of a candidate would be the starting pattern.
  - Problem:
    - The game of life: you are to find a starting pattern that is proven to grow over time.
  - Fitness:
    - I would suggest the number of generations that your pattern lasts for, the more the better.

Evolution

- As in nature, those that survive a given time period (usually one generation) are the fittest individuals.
    - You need a selection function (best would be a Priority Queue) but any sort function followed by “take” is OK; those organisms that don’t make the cut are “culled”;
    - You need to decide on the fraction of the population that will survive until the next time period—you will probably need to experiment with this—don’t use a small number or you will lose diversity from your gene pool.

Configuration Parameters*

- Initial (“seed”) population: 1000
- Proportion of organisms that survive and breed: 0.5
- Generations to reproductive maturity: 1 (or 2)
- Maximum number of generations: 10,000

* If you can use a configuration file for these, it would be best but not essential

Choosing a problem to solve and running the GA

- Don’t agonize over choosing a problem at the beginning. There’s plenty to do in terms of building your GA framework while you think about problems.

- GAs can be very tricky to get right. Once you have your framework and problem and are running evolutions, you will find that you will need to make adjustments (and/or fix bugs). Try to do this in a logical way: change *one thing at a time!*

Reporting/submission

- Step 1: Send team member names (and section) to your TA via Slack (direct channel)—he will assign you a team number.
- Step 2: Name (or rename) your repo as INFO6205_nn where nn is your team number
- Step 3: Create a short README which describes what problem you want to solve
- Step 4: send the URL of the repo (github) which you will use (to TA).
- Step 5: before the deadline, summarize your problem, findings, results, conclusions, in a document* at the top level. In particular, you must clearly describe items #1, #2, and #3 from the *Implementation Details* slide. **And, of course, you need to show evidence of your program running and passing the unit tests.**

* you choose the format: word, powerpoint, whatever.

Teams

- Form into two- or three-person teams; Three-person teams must provide a user-interface.
- You *may* have a fourth person but then you must provide evidence of parallelization.

Grading

- On-time: **very** important
- Running code with a solution: important
- Unit tests: important (particularly the basics like mutation, gene expression, selection)
- Good clean code: nice to have
- Parallel processing: bonus points

Your repository

- Github offers a template “ignore” file: pick the Java template. Don’t worry about the license. Create a README file.

- Do NOT include class files or other *derived* files (such as jars, IntelliJ IDEA files, etc.).

- If you employ any other jars beyond the SDK, then you should define your dependencies using maven.

Additional Thoughts

- The reason I'm insisting on having all your randomization take place on the genotypes (genes) of your organisms is because that's the best way to ensure maximum coverage of your solution space. If you randomize at the phenotype (trait) level, you will typically bias the results which may result in not finding the best solution.
- I want you to unit-test the most fundamental aspects of your method: evaluating the fitness function, for example. There should be somewhere between 6 and 10 unit tests. Warning: many of the students whose GA's didn't at first work previous semesters were because they had not unit-tested the basics.
- If you want to parallelize: I would recommend that each generation's population be split into *n* "buckets". Perform the gene expression, fitness evaluation, and selection for all the organisms of one bucket in parallel with all other buckets. Then when all buckets are done, merge back into one sorted population. Repeat.

Implementation Details

1. The genetic code and a random generator/mutator of such codes; You can have a free-form chromosome like in the natural world: or, much easier, just have a list of genes];
2. Gene expression: how do individual genes code for particular traits—ain the case of the game of life problem, you will need to translate your bits into a set of points.
3. The fitness function--this is essentially a measure of how good a candidate (organism) *solution* is for the problem you have chosen to solve; The survival function—to select the survivors according to their fitness;
4. The evolution driver--this takes care of the seeding of generation 0, and the births and deaths between generation *N* and *N+1*;
5. A logging function to keep track of the progress of the evolution, including the best candidate from the final generation; use *log4j* for example.
6. A set of unit tests which ensure that the various components are operating properly (especially the low-level functions: fitness, expression, etc.).