P6: Evo-Boids (Extra Credit)

Goal

Genetic algorithms provide a mechanism for searching large dimensional spaces for optimal or near-optimal solutions. The process mimics natural selection by mixing “DNA” of highly successful solutions as well as providing instances of random mutation in order to perturb solutions into new areas of the search space. Your task is to complete the implementation of genetic algorithms provided by Kate Compton, driving the behavior and aesthetic expression of flocks of birds.

Overview

This assignment is comprised of three components, each representing a specific facet of genetic algorithms.

**Gene Expression**

While the DNA of the flocks are represented as arrays of numbers, these values need to be connected to behavioral or other phenotypic expression (e.g. color, wing shape). In the current state of the program, the last four genes (behavior0, behavior1, behavior2, and behavior3) are not linked to any behavior. You will tie three of these genes into additional forces that act upon the flocks. Suggestions for these forces are given in the code section later on.

**Crossover**

A strong force behind the evolutionary search of the state space is the combination of successful yet distinct individuals in the population. Crossover provides a mechanism for taking a portion of one individual and mixing it with a portion of another. The basic process is as follows:

1. Select a “crossover point” randomly along the DNA sequence.
2. Create two new arrays, which will hold DNA for the “children” of the two selected individuals. Initially, copy in the parents DNA.
3. From the crossover point to the end of the sequence, swap the gene values within the children.
4. Add the children to the population for the next generation.

You will need to implement this functionality for creating new populations within the simulation.

**Fitness function**

The purpose of a fitness function is to evaluate the success of an individual, providing a set of “winners” that will seed the population for future generations. Within the interface, users can select flocks manually to be added to the winner pool. In this way, the user acts as a fitness function, selecting according to his or her own preferences.

However, we want to automate the evaluation of the population when the “Calculate winners” button is pressed. By default, the application selects three winners maximizing the amount of food collected. You will implement two alternative fitness functions over the flocks’ behavior or aesthetic expressions.

Genome

Each flock is described by an array of twenty values, forming a “DNA strand”. As a reference, we have provided the names of the corresponding variables below:

0. wingLength

1. length
2. wingWidth
3. hue0
4. hue2
5. pastel
6. style0
7. style1
8. style2
9. style3
10. drag
11. power
12. flapRate
13. separation
14. alignment
15. cohesion
16. behavior0
17. behavior1
18. behavior2
19. behavior3

Running the interface

To run the JavaScript program, open index.html in a web browser. The panels can be dragged around with the mouse.

The left panel displays the genomes of the flocks currently displayed. You can adjust the genome values by hand by dragging the corresponding gene. Furthermore, you may select “winners” of a generation by double clicking the picture of the flock you’d like to select.

The right panel controls the evolution process.

* “Calculate winners” - selects three winners according to the fitness function. By default, the fitness function simply selects flocks maximizing food collected.
* Radio buttons - provide options for how the next generation of flocks is generated.
  + Randomize - generates new flocks randomly
  + Mutate all - mutates all existing flocks
    - Mutation chance and magnitude are controlled by the corresponding sliders
  + Mutant offspring from winners - creates mutated versions from winners
    - Mutation chance and magnitude are controlled by the corresponding sliders
  + Crossover pairs of winners - creates offspring by performing crossover between pairs of winners
  + Custom - ignore these
* “Next generation” - uses the selected radio button mechanism to generate the next generation of flocks.

A typical genetic algorithm alternates mutation and crossover, allowing the combination of successful individuals with the random perturbation into new possible solution spaces. Play around with how you evolve flocks!

Code

**boid.js**  contains the behavior and aesthetic expressions of an individual within a flock.

* preUpdate: The boids are moved by a number of forces. Kate has implemented most of them, including alignment, cohesion, and separation as well as basic forces to keep them on screen and moving forward. Kate has further added extra gene values (behavior0-behavior3) which can be set to influence other types of forces. Example behaviors from Kate:
  + Line 114: A "wander randomly" force
  + Line 118: A rudimentary "go after food" force (to show how to query the food map)
  + Line 134: A "mouse following" force, turned off (to show how to interact with the mouse).
  + Try mixing the values for behaviors 0-3 in with these forces, or create your own force.

**evo-panel.js** contains the evolutionary behavior for the flocks. Mutate all and mutate winners are both implemented as examples of how to access values.

* Line 140 (“breedWinners”): Implement crossover, combining the DNA of two winners.
* Line 177 (“#set-winners”): Implement two new fitness functions. By default, the fitness function sorts the flocks by how much food they’ve gathered. Play around with other optimization goals. **You may need to alter the Boid class to store aggregate data at each tick of the main loop. You should update these values in the preUpdate method.** Potential ideas to optimize for:
  + A specific dispersion level of the flock (not too tight or too spread out)
  + Minimizing the distance to the cursor
  + A multi-objective function, combining two or more objectives. For example: evolving successful flocks that are drawn to the cursor ***and***maximize food gathered.

Submission Instructions

In addition to submitting your code, include a document answering the following questions:

* What new forces did you add to affect the behavior of a flock? How did you tie in the genetic values?
* Describe your two new fitness functions. What do they try to optimize? Try evolving flocks for several generations under each. How successfully do the flocks evolve to fit the desired outcomes from these selective pressures? Can you outperform the evolutionary process by adjusting flock genomes by hand?
* Experiment with evolving flocks using only mutation, only crossover, and a mix of both. Describe your findings in how the evolutionary mechanisms differ over time.

[Submission Link](http://www.formpl.us/form/5731599844900864/)