CPSC 1045: Final Project Proposal

Kyle Sankey

Topic

My final project will be a simple simulator of evolution using artificial selection. It will be too simple to simulate the evolution of actual animals, but will instead use lines and shapes. Using JavaScript, HTML Canvas, and a set number of "genes" that influence drawing rules such as curveature and branching of lines, I can start with a single line or dot and attempt to evolve it into a more complex shape.

How will this work and what does it have to do with artificial selection? The program will start at "Generation Zero" with a very simple shape. It will then spawn a set of children, each with a different "mutation" in their shape – a particular alteration to the drawing rule. The player can choose one child by clicking on it, and it will become the main shape for the next generation. This continues indefinitely.

The project will be partway between a game and a science project, as it can be interacted with, but will not have any sense in which the player can "win". It will continue evolving until the window is closed or reloaded.

Technical Approach

The majority of the code will be contained inside the Shape class. The class will contain an instance variable storing its genes, which will be numbers between -9 and 9. The class will contain the following methods:

- constructor(…genes)

This will take an array of genes received from the parent (or loaded from the STARTING_GENE_VALUES constant in the case of the Generation Zero shape). It will throw an error if an incorrect number of genes is provided, otherwise it will save them to an instance variable (this.genes).

- draw(ctx)

This will be called when displaying the current generation shape on the screen. It takes as an argument the canvas context object, and, starting with a branching tree pattern, uses this.genes to influence the drawing of this tree. Each gene has a different effect on the tree – for example, this.genes[0] could affect the angle of the branching, this.genes[1] could affect the number of recursive branches, et cetera.

- generateMutationMatrix()

This is the first of two steps to generate the shape's children. Using this.genes, it returns a matrix (2-dimensional array) of all possible +1 or -1 changes to the parent's genes. For each child, only one gene can mutate at once, and can only mutate by +1 or -1. Thus, the number of children is always this.genes.length*2.

- spawnChildren(matrix, ctx)

This is called after generateMutationMatrix(). It takes the returned matrix and canvas context object and uses HTML DOM functions to create a series of smaller canvas tags underneath the main shape's canvas. Using this.draw(ctx) recursively it can fill each small canvas with one of the current shape's children. Each child will have an onclick function that allows the player to choose it as the parent of the next generation of shapes.

Milestones

Week One: Processing genes. During this week I'll ignore the canvas completely and focus on implementing basic gene mutation. Testing and debugging will simply be done through the inspect console.

Week Two: Drawing rules. This will be spent implementing the method for drawing shapes on the main canvas using the genes. I'll need to figure out all the places I can use constants in different drawing rules and pick a number of genes corresponding to that amount.

Week Three: Choosing children. Here I'll focus on the reproduction part, making the method for spawning children with all possible mutational variations and letting the player choose one by clicking on its corresponding canvas.

Week Four: Debugging. In a complicated program like this, a week is a pretty good amount of time to make sure there ae no issues before the presentation and handing in.