Samuel Sicklick

Approach

My implementation of the *EquationSolver* utilizes a genetic algorithm where the "chromosomes" were instances of the implemented *SolutionI* object, and the "genes" were their *x* and *y* values. Each generation instantiates a *Population* of a given size of *SolutionI* objects with random *x* and *y* values, ranging from 0 to 100. These *SolutionI* objects are then inspected by fitness to determine if a correct solution has been found or not, whereupon the algorithm will either return a correct *SolutionI* or evolve into the next generation. The evolution process goes through the population to reproduce, crossover, or mutate its "chromosomes" into the next generation. This process is repeated until a "chromosome" is generated with a fitness greater than or equal to the threshold, or until the given maximum number of generations have evolved from the initial *Population*.

Fitness Criterion

Being that the only correct solution was (3, 2), fitness in my algorithm was determined by the closeness of the *x* value to 3 and the *y* value to 2. Fitnesses remained at 0 unless the values of *x* and *y* were somewhere between 0 and their appropriate values.[1]

Threshold

Being that the highest possible fitness was 7.0, as per the system described above (Fitness Criterion, see footnote), I utilized a threshold value of 7.0.

Selection Type

I found that the *Roulette* selection type provided the best results, which is logical since it was proportional to the fitness of each "chromosome"

Mutation

My mutation function altered the *x* and *y* values of a cell to cause them to near towards the correct values via incrementation and decrementation.

Crossover

My crossover function simply swapped the *y* values of two "chromosomes".

[1] For Example: (1,2) would have a fitness of 5, as it was awarded 2 fitness points for the *x* value of 1 and an additional 3 fitness points for the *y* value 2.