Little Feeders

A Genetic Algorithm that Simulates Biological Evolution

Daniel Knowles

CPCS 5185G

Report for Final Project

Table of Contents

[Tools Used During Development 1](#_Toc260341609)

[Program Description 1](#_Toc260341610)

[Findings 1](#_Toc260341611)

[Program Instructions 1](#_Toc260341612)

[Potential Future Enhancements 1](#_Toc260341613)

# Table of Figures

[Figure 1 Default Appearance 1](#_Toc260341634)

[Figure 2 View Menu 1](#_Toc260341635)

[Figure 3 File Menu 1](#_Toc260341636)

[Figure 4 Paused 1](#_Toc260341637)

[Figure 5 Options Menu 1](#_Toc260341638)

[Figure 6 Feeders without FOV displayed 1](#_Toc260341639)

[Figure 7 Additional Options Dialog 1](#_Toc260341640)

# Tools Used During Development

* Programming Platform: JDK SE version 6 Update 20
* Language: Java version 1.6
* IDE: NetBeans IDE version 6.8
* Operating System: Microsoft Windows 7 Professional

# Program Description

Little Feeders is a program that attempts to simulate a very simple natural environment. In this environment are feeders and food. Each feeder is given a set of traits - eyesight strength, speed, and intelligence level. Each of these traits is given a number value that indicates the strength of that trait. The feeders use these traits to find and eat food (note also that food objects are stationary). At the end of each generation, each feeder is given a fitness score based on the amount of food it has eaten. The feeders that eat the most (and thus have the highest fitness scores) are most likely to mate and pass on their traits to the next generation. It should be noted, however, that every feeder has a chance to mate and pass on their traits even if they ate no food at all.

Each trait is given two values. It is given a raw value and an effective value. The raw value is used to encode the trait while the effective value is used to set the actual value of the traits. It is the effective value that the feeders use to interact with the environment. The maximum raw value of each trait is set to 15. This allows each trait to be encoded as a four bit binary string. When each feeder’s traits are being encoded, they are turned into a 12 bit Boolean array which is used to represent a 12 bit binary string (true = 1, false = 0). This means that each set of traits is used for crossover rather than each individual trait. (A potential enhancement for future development might be to add another crossover point so that the feeders could split their chromosomes at more than one point when mating.) The effective traits are calculated by multiplying each raw trait value by a corresponding modifier. The modifier for eyesight and intelligence is 6.67. The modifier for speed is 0.013. This allows the program to set the maximum effective eyesight and intelligence values to 100.05 (15 \* 6.67) and 0.195 (15 \* 0.013) for the effective speed.

Traits are passed on through the use of a genetic algorithm (GA). The steps the GA takes are as follows:

1. Encode a set of traits.
2. Use the encoded traits to create a population of genotype objects.
3. Each genotype object is used to create a new feeder.
4. The feeders are given the chance to roam the environment for food until the end of the generation.
5. At the end of the generation, the GA evaluates the population of genotypes using the feeder fitness function.
   1. The feeder fitness score returns the raw fitness score for the genotype.
   2. The GA normalizes each genotypes fitness score by dividing the raw score by the sum of all genotype fitness scores.
6. The GA then sorts the genotypes in the population in descending order by fitness score (i.e. highest fitness score first).
7. The GA uses roulette wheel selection to iterate through the population and choose two genotypes at a time from the population to mate.
8. Each genotype pair mates and returns one offspring.
   1. A check is made against the crossover rate to determine if the offspring shares the traits of the two genotypes. If not, the offspring is a copy of the first genotype selected for mating.
   2. A check is made against each bit in the offspring’s “chromosome” to determine if it will be mutated. If so, the bit will be flipped (i.e. true (1) to false (0)).
9. The new population of offspring becomes the main population, and the GA starts again at step 2

The generation length is determined by the number of times the program has entered the “game loop” which updates the environment, feeders, and food. The game loop is a very basic implementation of calling the repaint() method after each iteration. This calls the environment’s paintComponent() method which then calls the update method, starting the process again. The current setting for a generation is 7300 iterations (365 \* 20). The maximum number of generations is set to the maximum value of a Java integer (2,147,483,647 generations).

The process feeders use to decide whether or not to eat is outlined below:

1. The food must be in the feeder’s field of vision (FOV). The FOV is determined by the feeder’s eyesight strength. The feeder can see up to the effective eyesight in front of it. The center FOV point creates a line with the center of the feeder that bisects a 120 degree angle. This gives the feeder 60 degrees of peripheral site on each side.
2. Once a piece of food is within sight, the feeder checks its intelligence to determine whether or not it recognizes the object as potential food. This is done by getting a random double between zero and the maximum effective intelligence level a feeder can have. If the feeder’s effective intelligence level is higher than the random number, the feeder moves towards the food.
3. Once the feeder is at the food’s location, the feeder performs another intelligence check as described in step 2. If the intelligence check passes, the feeder eats the food. If it does not pass, the feeder moves on.

# Findings

The program produces some interesting results and usually produces a different population (in terms of traits) each time it is started. I have noticed, however, that each generation does not necessarily produce a more fit population, but it does produce a different population. I have also noticed that usually, each generation will produce either a higher average fitness score and/or a feeder with a higher fitness score that the previous generation’s highest fitness scores. However, this only continues for a few generations before the fitness scores begin to drop again. I would expect that adding more feeders and/or food would produce better results faster. The only problem is that the program is somewhat resource intensive, and adding too many feeders and food to the environment slows the simulation down considerably.

# Program Instructions

The source files for this program can be found in the “DanielKnowlesFinalProject” zip file at “DanielKnowlesProjectJava\src\danielknowlesprojectjava”.

To run the program, double click the DanielKnowlesProjectJava.jar file included in the DanielKnowlesFinalProject.zip file. Instructions on using the application are given below.

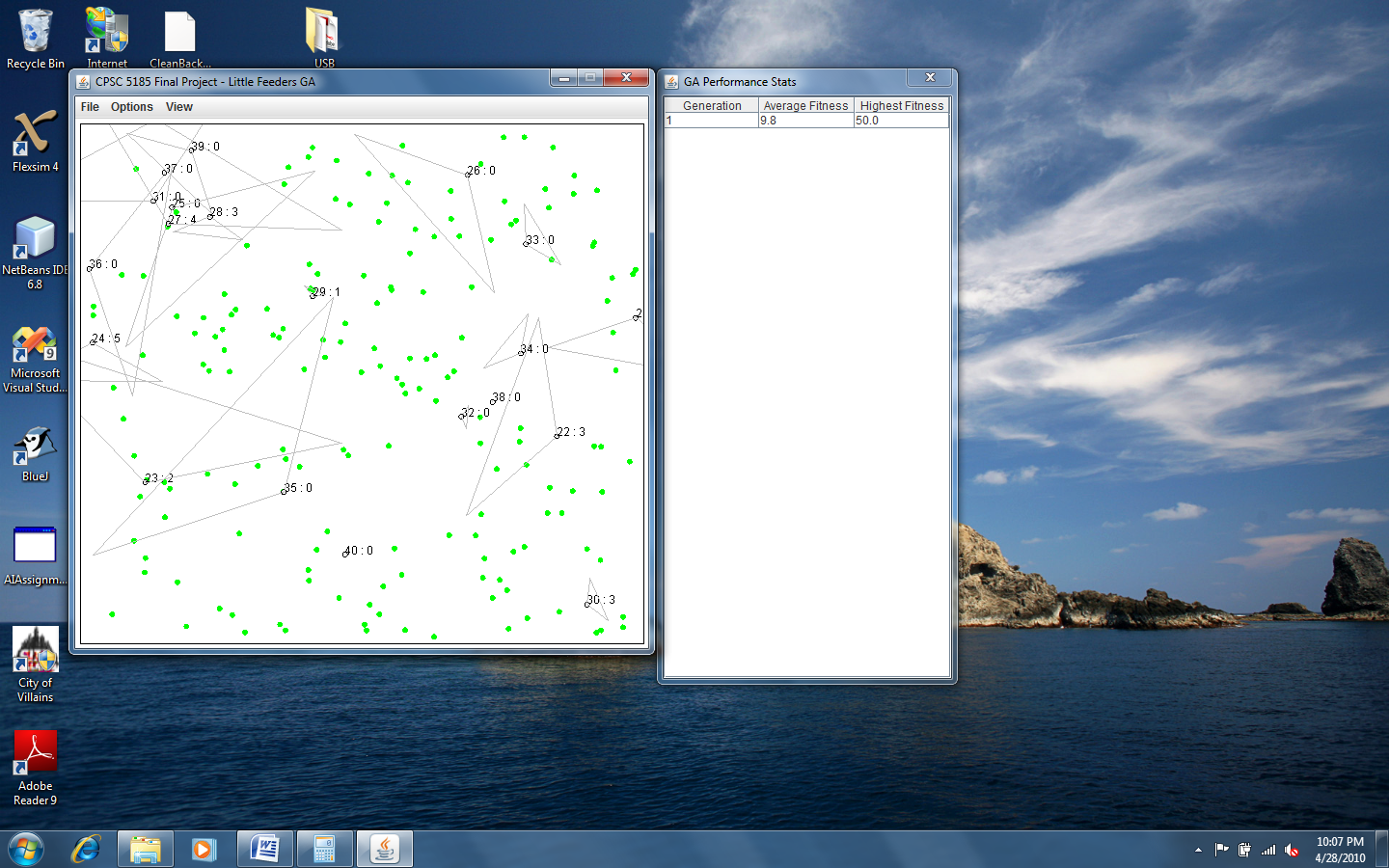


Figure Default Appearance

Figure 1 illustrates the application’s appearance when it is first started. The feeders are represented by black circles (with no fill). The food objects are represented by filled in green circles. The default settings render the feeders’ fields of vision (FOV), each feeder’s unique ID, and the amount of food eaten by each feeder. The ID and food eaten is displayed above each feeder, ID first as [ID] : [Food Eaten]. For example if feeder number one (1) has eaten six (6) food objects, the numbers above the feeder will appear as “1 : 6”. The FOV is displayed as a triangle with the largest side facing the direction the feeder is moving in. Feeders that have a speed of zero (0) will not render a FOV triangle. It is also worth noting that the triangles are not an exact representation of the feeders’ FOVs. The actual FOV is calculated as a cone where the peripheral vision is 60 degrees to the right and the left of the feeder’s center point (creating a 120 degree angle from the center of the feeder). The actual FOV is determined by the effective eyesight strength (described in the Program Description section above). For instance, if the effective eyesight value is 20, the feeder’s FOV will be 20 pixels from the center of the feeder from the left peripheral vision to the right. Plotting these points exactly would result in a cone. However, to reduce computing overhead (and programming time), a triangle is used to show the approximate FOV.

Figure 1 also displays another feature of the program; a table illustrating the genetic algorithm (GA) performance. The screenshot displays the scores for the first generation; however, when the program starts, this table contains no data and is updated at the end of each generation. The table displays the generation number, the generation’s average fitness score (raw fitness), and the raw fitness of the most fit feeder.

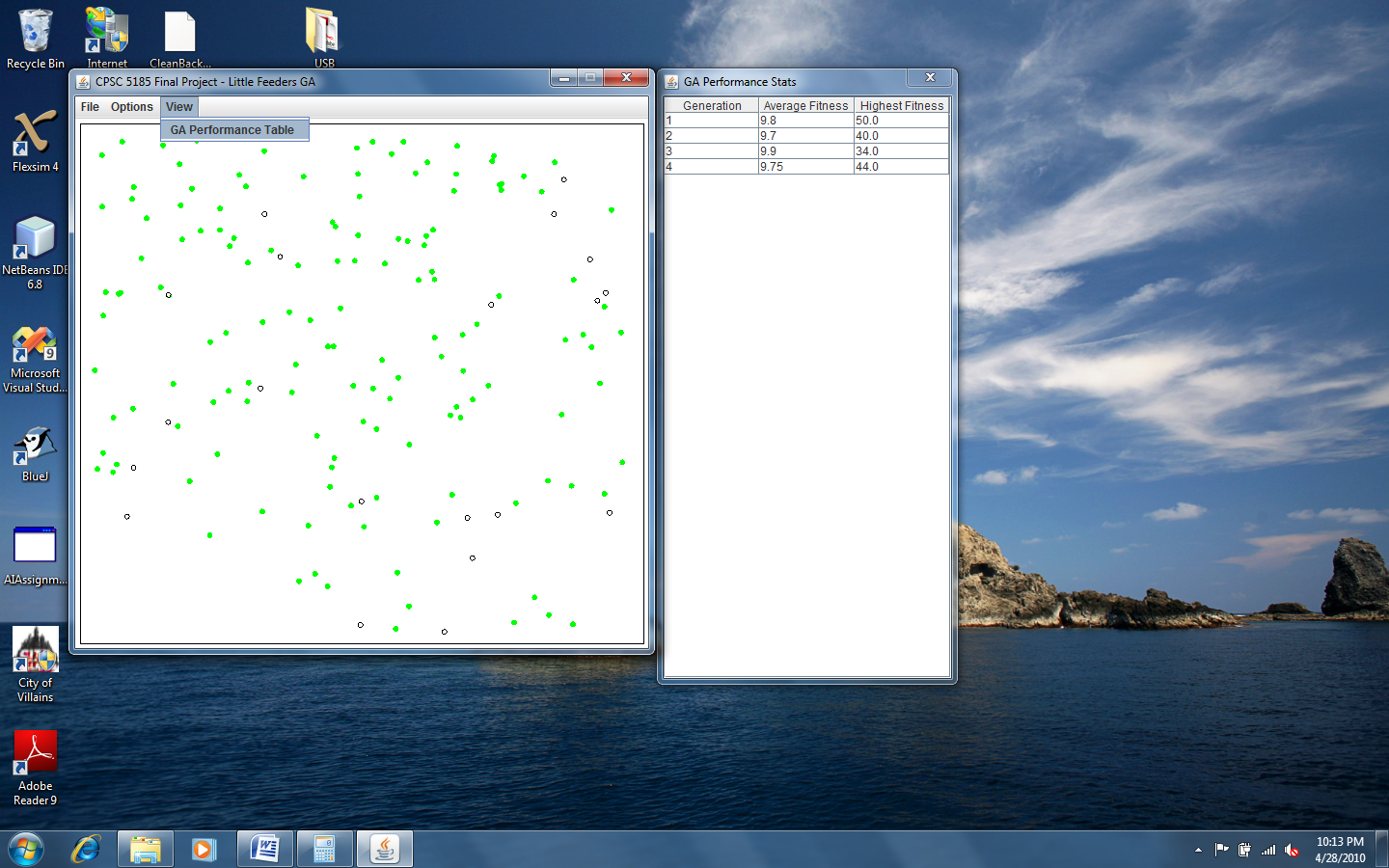


Figure View Menu

As Figure 2 illustrates, the view menu contains an option to show and hide the GA performance table. Selecting this option while the performance table dialog is displayed will hide the dialog. If the dialog is not displayed, however, selecting this option will display the performance dialog.

Figure 3 displays the File menu. This menu contains options to reset the simulation, pause the simulation, and exit the program.

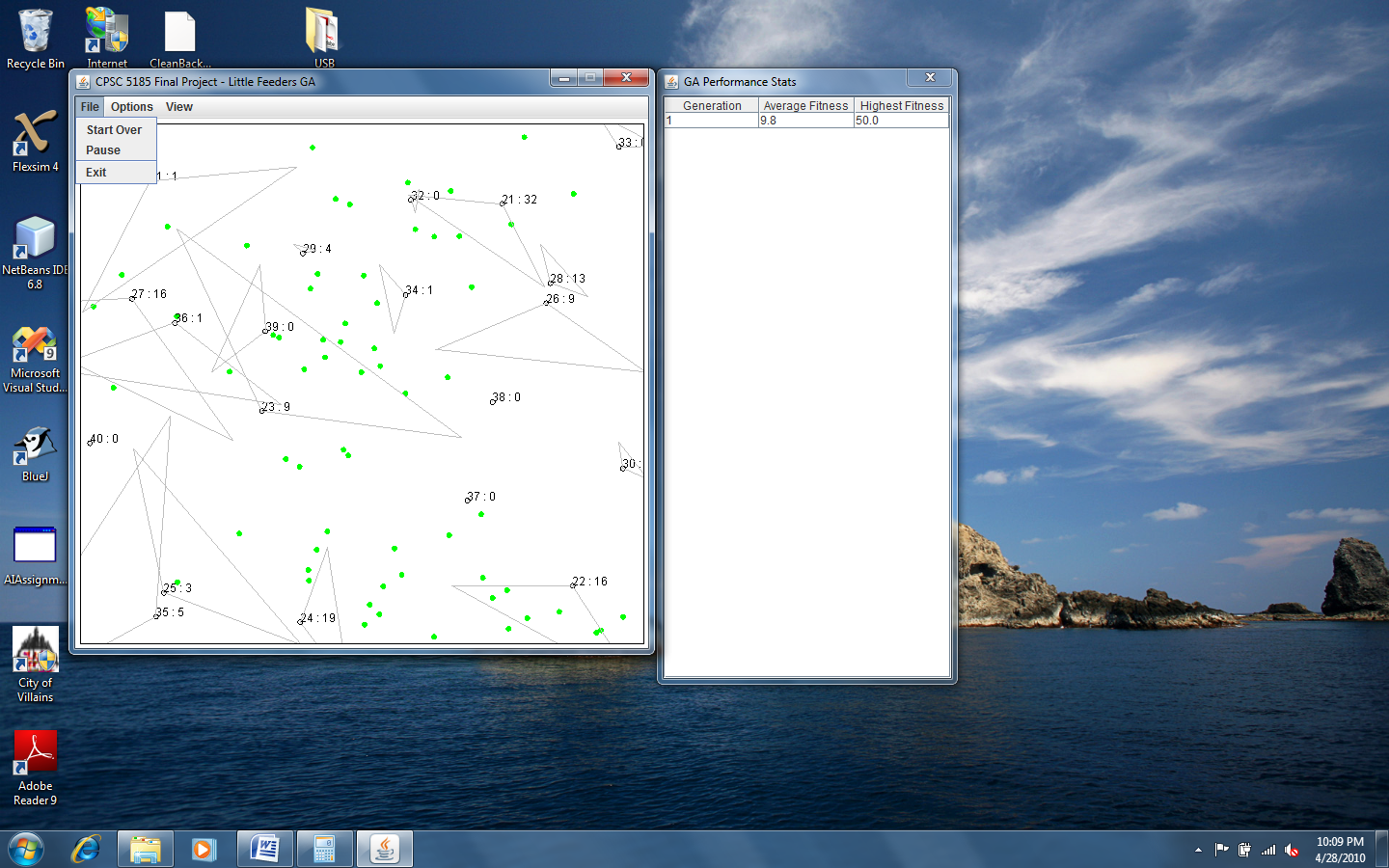


Figure File Menu

The “Start Over” option causes the program to start the simulation over again using the current settings. This and the “Additional Options” option discussed later uncover one of the program’s known issues. This option is supposed to reset the GA performance table so that none of the previous run’s performance data is displayed. The table is reset; however, data for the last generation of the run being cleared will be displayed above the data for the new run’s data.

The “Pause” option will cause the simulation to pause. It also clears the environment and displays the “Paused” dialog as displayed in Figure 4.

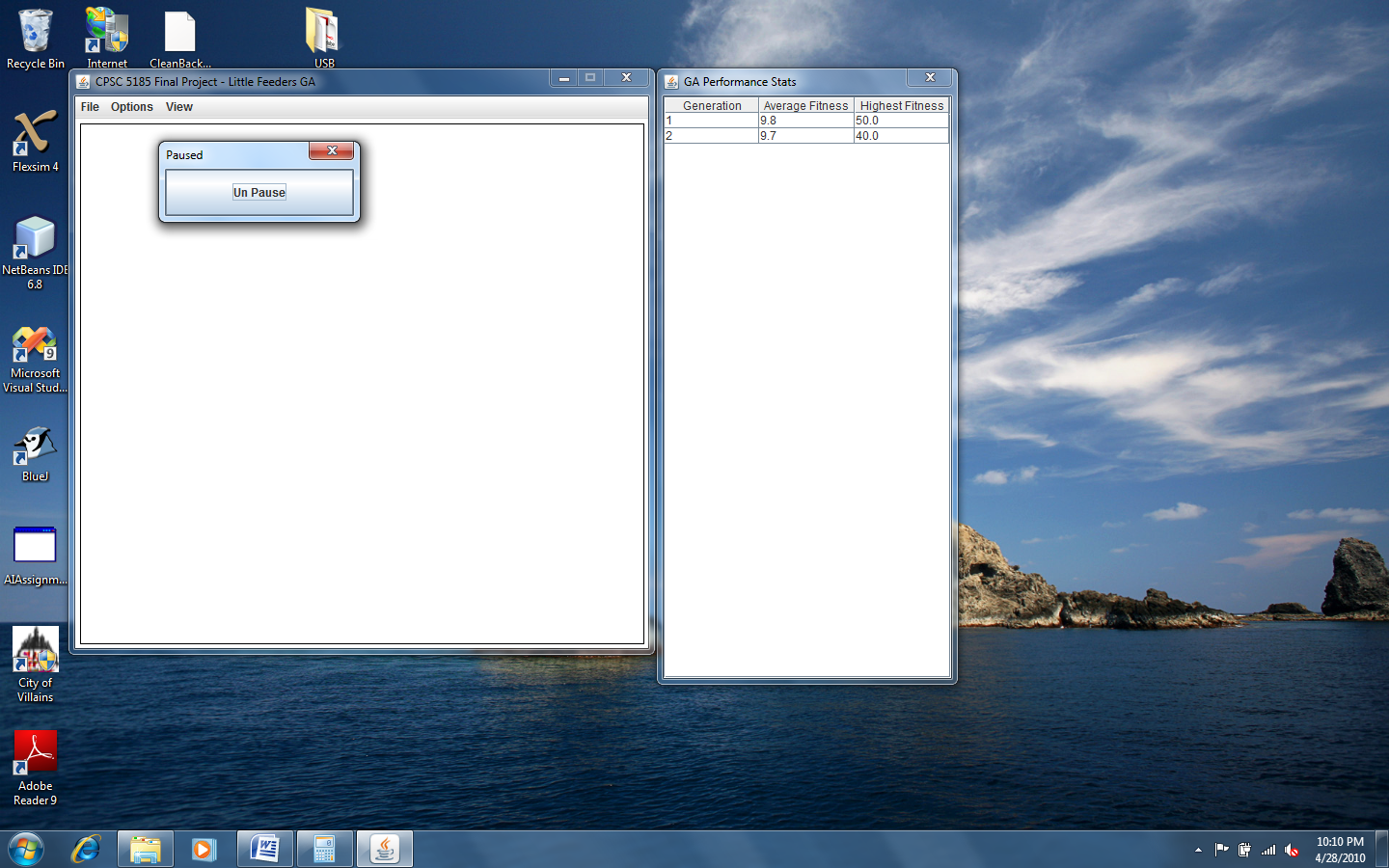


Figure Paused

To resume running the simulation, click the “Un Pause” button. The simulation will continue running from the point when the pause option was chosen.

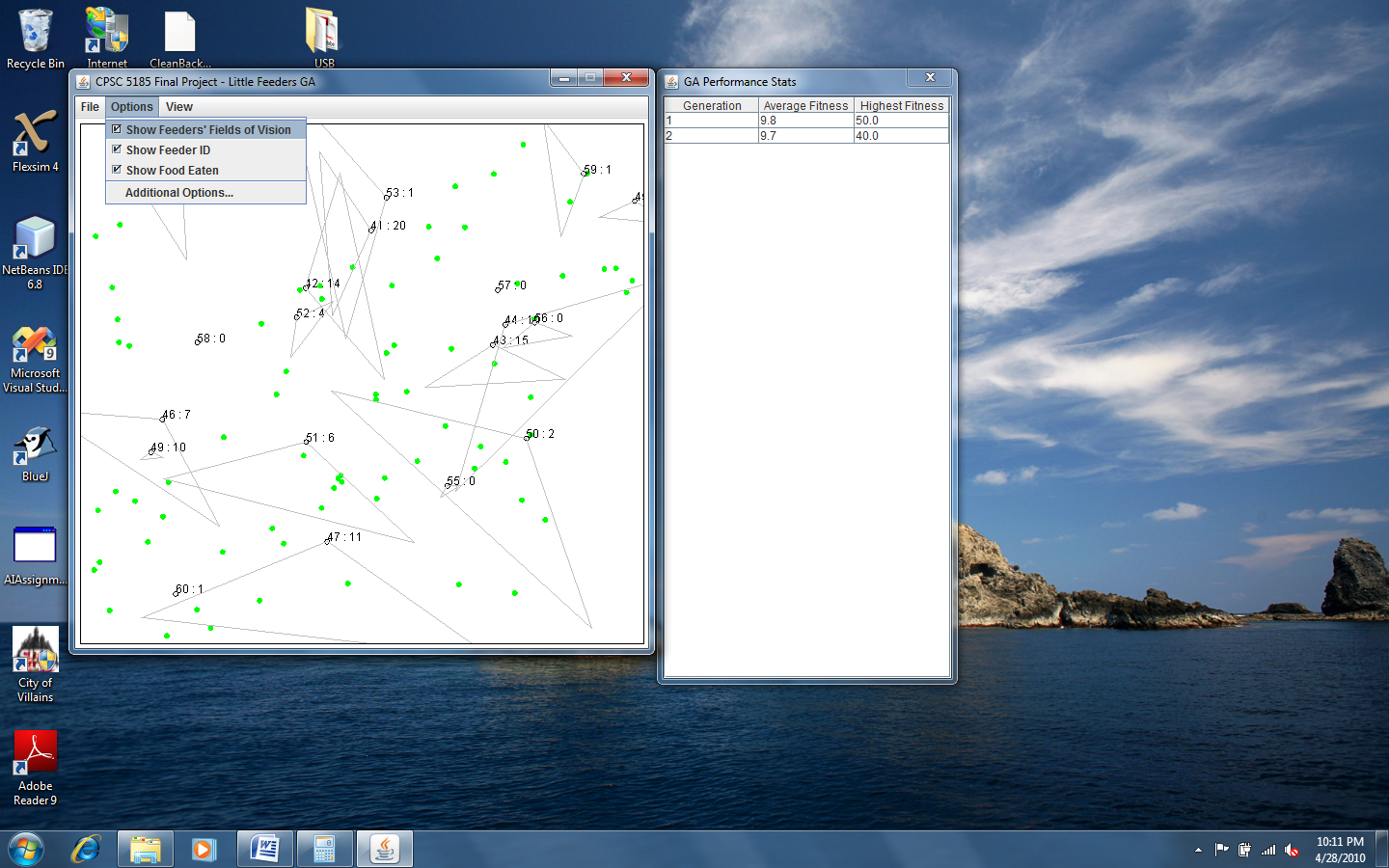


Figure Options Menu

Figure 5 displays the Options menu. From here, the user can choose to show or hide the feeders’ FOV, unique ID, and food eaten. This menu also contains an option to set additional options.

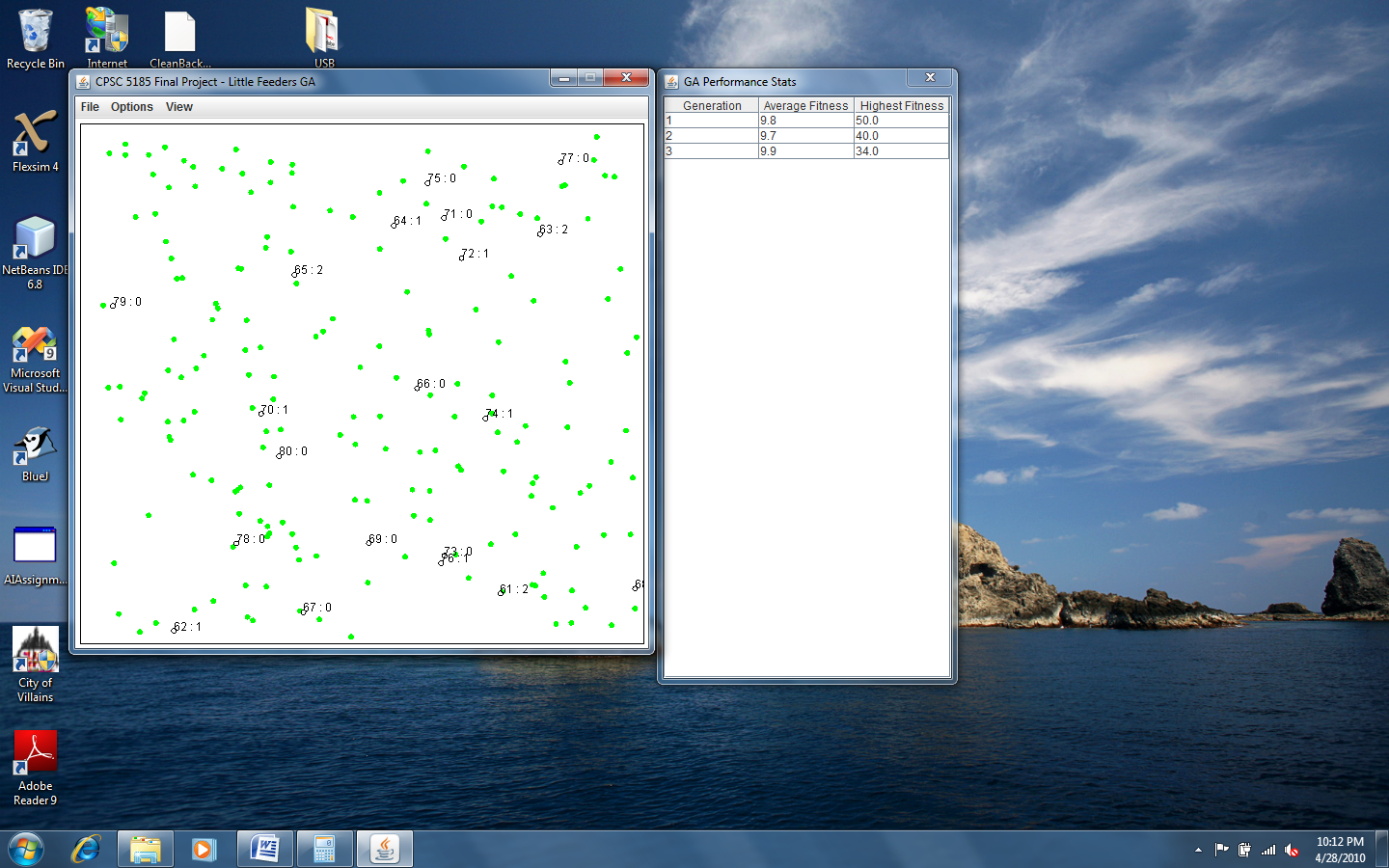


Figure Feeders without FOV displayed

If the feeders’ FOVs are displayed, selecting the “Show Feeders’ Fields of Vision” option will cause the program to stop rendering the FOV triangles. With all other options checked, the environment would resemble Figure 6.

Choosing to hide either the ID or food eaten would result in only the chosen number being displayed without the colon. If none of the options were selected, the feeders would appear as black circles with no decoration or label.

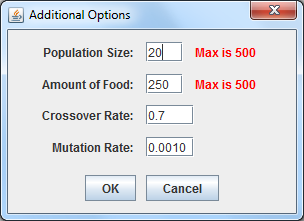


Figure Additional Options Dialog

Selecting the “Additional Options…” option displays the “Additional Options” dialog frame as illustrated by Figure 7. While this dialog is displayed, the simulation will also be paused. Clicking the “OK” button causes the simulation to be reset with the chosen configuration. Clicking “Cancel” causes no change to the simulation’s configuration. Instead, the “Cancel” button simply causes the “Additional Options” dialog to close and un-pauses the simulation.

The available options are population size, amount of food, crossover rate, and mutation rate. Each textbox is initially populated with the current values. The values shown in Figure 7 are the default values that are set when the program first starts. The settings are defined as follows:

1. Population Size: The number of feeders that are interacting with the environment. The default value is 20 feeders.
2. Amount of Food: The number of food objects to place into the environment. The default value is 250 food objects.
3. Crossover Rate: The probability that two mating feeders will split chromosomes and share their traits when creating an offspring. The default value is 0.7 (70%).
4. Mutation Rate: The probability that a feeder will be mutated. More accurately, the probability that any one bit in a feeder’s chromosome will be flipped. The default value is 0.001 (0.1%).

# Potential Future Enhancements

There are several enhancements that would make this program better. The biggest enhancement I would add is other objects for the feeders to interact with in the environment. The feeders only look for and eat food in this application. If the feeder sees the food and recognizes it as a potential meal, it moves to the food’s location. If it is fast enough (i.e. it gets there before another feeder), it will check its intelligence again to see if it still recognizes the object as food. If it does, it eats. If it does not, it moves on. This is a decent beginning, but, if I had more time, I would also add objects that aren’t food. Some would be toxic while others would just be objects that aren’t food. If the feeder eats the nontoxic object, it just eats something that doesn’t count towards its fitness score and wastes time. If the feeder eats the toxic object, it would actually lose health. At the end of the generation, those feeders that are still alive are evaluated on the amount of food eaten as well as the amount of health left.

It would also be interesting to experiment with how the simulation changes when one or two more crossover points are introduced during the mating process. It would also be an interesting exercise to add other selection methods (e.g. elitism) to see how it affects the evolution of the feeders.

I also regrettably admit that this program lacks a good set of user interfaces for evaluating the GA. Currently; the application contains a table that shows each generation’s average fitness score and the highest fitness score. It would be nice to have another interface that displayed the average trait values and the highest scorer’s trait values. I’m sure there are other methods as well, but it took me too long to implement the core functionality to add these enhancements.

Also missing is a feature to change the maximum number of generations and the number of iterations in each generation. Changing these values may also have an impact on the behavior of the simulation as well as the evolution of the population.