Rotman School of Management
UNIVERSITY OF TORONTO

Rotman

**Group Number:** Team 2

**Assignment Title:** Final Report

**Course Code:** RSM8521

**Instructor Name:** Brian Keng

In submitting this **group** work for grading, we confirm:

- That the work is original, and due credit is given to others where appropriate.
- That all members have contributed substantially and proportionally to each group assignment.
- That all members have sufficient familiarity with the entire contents of the group assignment so as to be able to sign off on them as original work.
- Acceptance and acknowledgement that assignments found to be plagiarized in any way will be subject to sanctions under the University's Code of Behaviour on Academic Matters.

Please **check the box and record your student number** below to indicate that you have read and abide by the statements above:

| | | | |
|---|---|---|---|
| ☒ | 1002419395 | ☒ | 1007554745 |
| ☒ | 1002450297 | ☒ | 1002070108 |
| ☒ | 1007555138 | ☐ | |

Assignments are to be submitted using Student ID Numbers *only*; do not include your name. Assignments that include names or that do not have the box above checked **will not be graded.**

Please pay attention to Course Outline for specific formatting requirements set by instructors.

If submitting this assignment online please use the following "Standard File Naming Convention":

**Full Course Code *(including Section)*–Group Name or Number–Assignment Title**

**Example:** RSM1234HS.2016-0101-Group1-Homework1

# Table of Contents

# 1. Introduction

Have you ever played Flappy Bird? This is a simple game in theory, where the objectives are to allow a bird to 'flap' through pipes and drops by tapping on the bird to flap up and not tapping to move down. Contrary to the simplicity of the game, getting a high score is not necessarily easy. After many hits on the pipes and falls on the drops, we asked ourselves, '*What if we can make the Flappy Bird teach itself how to go through the obstacles?*'. This was possible with a genetic algorithm called NeuroEvolution of Augmenting Topology (NEAT). In simple terms, just as a baby naturally grows to crawl, walk, and talk, this algorithm allows evolving artificial neural networks to teach themselves to play the game instead of being pre-trained like traditional machine learning methods. In this report, we discuss the details of the NEAT algorithm and its applications to the game, Flappy Bird. After reading this report, you will have the secret recipes to the honour of the highest scores among your friends.

# 2. Background

Two of the most frequently used artificial intelligence models to optimize games like Flappy Bird are reinforcement learning and genetics algorithm (GA). Traditional reinforcement learning is an approach derived from the concept of behavioural psychology where there is a set reward objective (the score) and the model outputs the best 'policy' or 'actions' to take to maximize the reward in the given environment (game). This policy is learned by the model ('agent') continuously learning what best set of actions (movements up and down) maximized the reward in given various environments (rounds of games).

The NEAT algorithm is a type of genetic algorithm using unsupervised learning, shown to outperform traditional reinforcement methods on various key applications. It does not output a strategy for the model to take, but it evolutionarily builds a neural network that optimizes the scores. It can also be embedded in reinforcement learning strategies. Details of this algorithm will be discussed further in the next sections.

# 3. Genetic Algorithm

Genetic algorithm (GA) is a search-based technique inspired by Charles Darwin's theory of natural evolution that implements the evolutionary strategy (EA). It reflects the process of natural selection, in which the most fitted individuals are selected and combined to produce offspring. These offspring which are even better individuals compared to their parents will compose the next generation. Through iteration, the best generation for the problem will be found. (Mallawaarachchi, 2017)

There are mainly six steps involved in this algorithm (Mallawaarachchi, 2017):

- **Initial population:** This is the starting point of the process, where we have our initial group of individuals. Each individual has its own chromosome which consists of a different combination of genes.


*Figure 1: Initial Population Demonstration*

- **Fitness function:** Here we calculate a score for each individual to assess its fitness to the problem. It is comparable to the optimization or loss function in the artificial neural network.
- **Selection:** At this phase, the most fitted individuals will be selected to be the parents for the next generation.
- **Crossover:** Each pair of the parents that are selected will reproduce offspring through exchanging and combining genes in their chromosome.

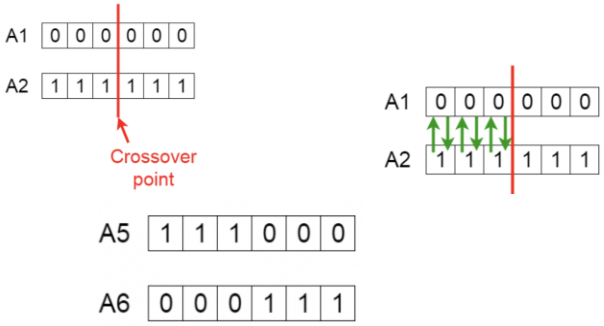



*Figure 2: Crossover Demonstration*

- **Mutation:** For some of the offspring, certain genes in their chromosome can be mutated with a percentage to add new variants to the population.

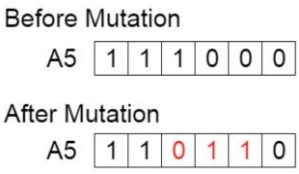



*Figure 3: Mutation Demonstration*

- **Termination:** The algorithm would stop when the population converges, like when the new generation is not significantly different from the prior one.

- The sample **pseudocode** for Genetic Algorithm:
    - START
      Generate the initial population
      Compute fitness
      REPEAT
        Selection
        Crossover
        Mutation
        Compute fitness
      UNTIL population converged or user-defined criteria
      STOP

Some application areas of the genetic algorithm include optimization, economics, image processing, neural network, routing, multimodal optimization, etc. (Tutorialspoint). Some related open-source code examples are listed in **Appendix A**.

# 4. NEAT Algorithm

NEAT algorithm is one type of genetic algorithm. NeuroEvolution is a structure where we apply the evolutionary strategy to the neural network structure. A traditional neural network relies on gradient descent to provide the direction for improvement. Although it is faster in most cases, using gradient descent may find a local optimum solution instead of global solution. However, since the evolutionary strategy starts with multiple points (for example, the initial population in the genetic algorithm), it allows us to overcome the problem of local optimum. Moreover, we could leverage the power of parallel computing to scale up the size of training points, which will generate the result more efficiently.

There are two different types of NeuroEvolution network structure. The first one is having a fixed neural network topology, the optimal solution is found by tuning the weight of each parameter. The second method is adjusting the layers and the weights simultaneously during the training process. This method is more robust and has a better performance. Moreover, similar to the idea of the genetic algorithm, there are three main steps in the NEAT algorithm.

## Step 1: Genetic Encoding

In genetic algorithms, information is encoded in a series of numbers (gene). There are two different types of genes in the NEAT algorithm: node genes and connection genes. Node genes are responsible for forming the structure of the neural network, they may be classified as input, hidden or output genes. Connection genes are responsible for forming the edge between nodes in different layers. Each individual data point is represented by a list of connection genes. They specify the nodes that are connected, the weight of the edge, whether such edge is enabled or disabled etc. An innovation number is a unique

number used to identify the mutation procedure, which is essential information when performing crossover in the NEAT algorithm.

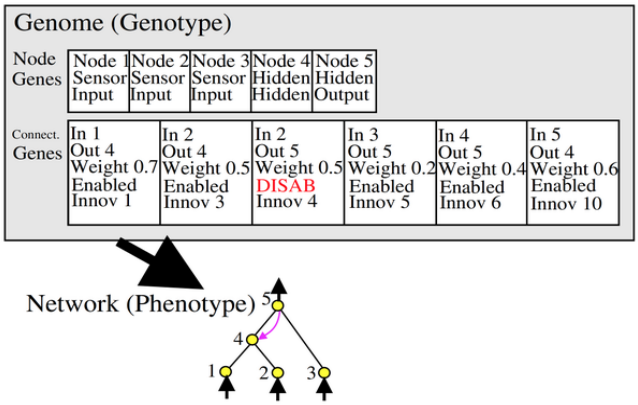



*Figure 4: Genetic Encoding Demonstration in NEAT*

## Step 2: Mutation

As shown in the figure 5, there are two types of mutation performed in the NEAT algorithm. We could mutate the genes by adding more connections. For example, we could add a new edge between Node 3 and Node 4 to mutate the gene. The mutation could also be performed by adding more nodes in the neural network. As shown in the bottom part of the figure, we could add a new interaction between Node 3 and Node 5 to break down the direct connection.

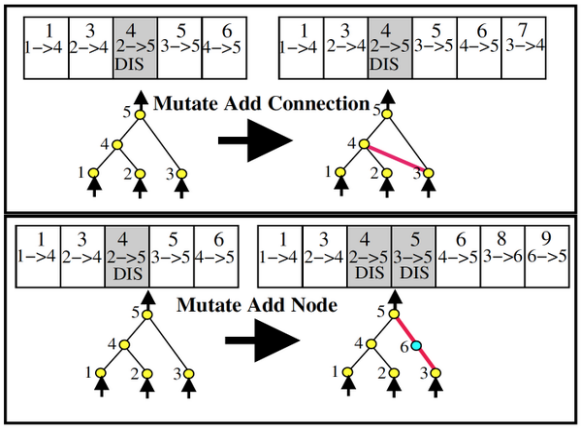



*Figure 5: Mutation Demonstration in NEAT*

## Step 3: Crossover

When performing crossover, parents' genes are matched by their innovation number. When both parents share the same gene, the offspring will select one to generate a new list of connection genes. If there is only one gene with a particular innovation number, the offspring will directly inherit the gene.

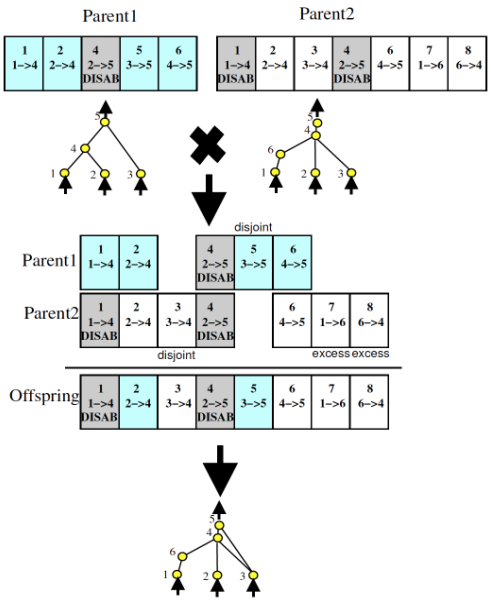



*Figure 6: Crossover Demonstration in NEAT*

There are many applications of the NEAT algorithm, it could be used in general game playing such as Flappy Bird, snake etc. It could also be applied to evolutionary robotics or self-driving vehicles.

# 5. Description of Method

## 5.1 Experiment

Flappy Bird is a simple game, controlling a bird to fly through as many obstacles as possible. Simple rules are easier for us to understand the evolution of the genetic algorithm, and to see the improvement over time.

To set up the NEAT algorithm an initial neural network must be created with a corresponding configuration file containing the hyperparameters (**Appendix B**). There are essentially three input modes: the top pipe distance, the bottom pip distance, and the location of the bird with an output node of whether to jump. The neural network is a feedforward network with all activation functions being tanh to limit the range of the output to [-1,1].

We initiate the experiment pool with 50 neural networks or birds and have them each controlling a bird. Initially, each bird's behaviour is dependent on the default setting of the model they are based on. After several iterations, the surviving birds will start showing different behaviour, some die early, some last longer. Models that make birds last longer (higher score) will remain in the pool, those who vanished in the process of evolution will be replaced by new models. The score in this instance is the fitness function and is the criteria used by the NEAT algorithm to either keep or remove a bird.

After every iteration, surviving models are mutated in terms of activation function used, node size, enabled connections, bias, and weights. After several generations, the neural networks left tend to be the ones that allow the birds to survive the longest.

From this process, we can observe the evolution of models by using the NEAT package from python (**Appendix B**). This package will allow us to see the size of the neural network and the number of enabled connections for each generation as well as the fitness score of the best bird. From Appendix B you can notice that for each generation a species of the population is selected, which in this case are neural networks. Additionally, the size attribute consistently changes from (1,3) to (2,4) and so on for each generation. The brackets (a,b) represent the number of nodes and connections in the neural network, with a being the number of nodes and b being the number of connections. For the sake of demonstration, the mutation rate for adding a new node was set to 100% to show how the model does adjust the neural network to find the best fitness neural network. Using the output seen in Appendix B we can use this experiment as an easy-to-understand demonstration of the power of the NEAT and genetic algorithm.

## 5.2 Hyperparameters

When it comes to the NEAT algorithm there are many hyperparameters available in the NEAT package. This makes the tuning process very granular as many aspects of the neural network can be adjusted and can impact performance. For example, the mutation rate, which is the probability that mutation will alter an attribute of the neural network can be adjusted for the bias, weights, connections, and even nodes of the network. To gain a higher-level understanding of the algorithm, the most important hyperparameters are listed and explained in the table below.

*Table 1: Hyperparameters for NEAT*

| Hyperparameters | Influence |
|---|---|
| *Population Size* | Influences starting fitness score |
| *Node Add/Remove Probability* | Adjusts topology of neural network |
| *Mutation Rate* | Influences probability of species changing |
| *Number of Hidden Layer Nodes* | Initial number of nodes in hidden layer |
| *Maximum Stagnation* | Removes species that have not improved after n generations |
| *Change Activation Function** | Activation functions for each layer |

From our analysis, it was found that population size was the most influential hyperparameter as it directly influenced how fast the bird could reach a state of an almost infinite score. This is because having a strong initial population increases the chances of

the algorithm to find a high-scoring species to select and mutate on. Due to this, for the other hyperparameters, we decided to set population size to 3 and random seed to 42, so that we could truly measure the influence of the hyperparameters versus population size being the main reason for success. Additionally, we set a fitness score threshold of 100 where the program stops if the bird dies after reaching that fitness score.

To demonstrate the influence of hyperparameters on fitness score we decided to focus on adjusting the hidden layers from zero to three. As mentioned earlier the initial neural network had 3 input nodes and 1 output node and this change would make the network more complex. The game was run for 50 generations and the fitness score was measured in the **Appendix C** below.

As you can see from the Hidden Layer Node 3 graph, there tends to be a lot of randomness to the fitness score as it does not seem to continuously increase fitness score. This concept of randomness is also better shown in the next two graphs in Appendix C.

The Sigmoid graphs demonstrate two trials with the exact same hyperparameters utilized for 50 generations. As you can see for trial 1 when the activation function was sigmoid instead of tanh the algorithm found the optimal network at generation 46 where the score kept increasing passed the threshold. For trial 2, the model unfortunately did not reach past the threshold and demonstrated completely different behavior across generations.

What all the experiments demonstrate is that, at least when it comes to a NEAT or genetic algorithm, there is a degree of randomness where mutations and crossovers can lead better or worse off neural networks. The mutation rate in this case acts as a learning rate, where having it too high can lead to overshooting the optimal solution and the generation count act as epochs that allows for more exploration.

### 5.3 Comparison to Reinforcement Learning

As Flappy Bird is a game with a fairly simple objective, it is very easy to compare how the NEAT algorithm approach compares to the reinforcement learning approach. To begin with the differences, NEAT is an evolutionary approach that takes advantage of genetic algorithms to find the optimal neural network structure. Reinforcement learning on the other hand is about creating agents that learn an optimal policy that maximizes a reward function. When it comes to more complicated tasks, reinforcement learning, currently, are more appropriate. However, when it comes to simple tasks, such as Flappy Bird, the ability for NEAT or genetic algorithms to be massively distributed by training a population size of n at the same time allows for shorter training time. Theoretically, NEAT can be accomplished using reinforcement learning as well by having the state be a neural network and have the algorithm modify it over time to get higher performance.

# 6. Conclusion

Neural networks were inspired by how the synapses of the brain worked, through this paper we have discovered yet another data science technique inspired by a biological phenomenon: genetic algorithms. By continuously taking advantage of the tenants of natural selection, genetic algorithms can be a valid alternative to finding the local/global optimal value. Through mutating and selecting only the best species, the genetic algorithm can effectively search for optimal solutions. In terms of the NEAT algorithm, which takes advantage of the genetic algorithm, the optimal solution or output is a neural network that maximizes or minimizes a fitness criterion. It helps us to release the constraint of fixed network structure and allows us to find a more effective model. By using Flappy Bird as an experiment, we can effectively observe the evolutionary process of genetic algorithms and how it eventually finds the optimal neural network to maximize the fitness function or total score in the game. Finally, through our experiment, we can gain a better understanding of the benefits and limitations of NEAT and the influence of different hyperparameters on performance. Through our data science journey, we have learned many techniques to tune hyperparameters including GridSearchCV or Keras Tuner, however, these techniques required manual input of the different hyperparameter values to explore. Through the use of NEAT, we can provide a technique where the machine explores by itself in an interactive process without the risk of human error.

# 7. Reference

Ashmeet13/Flappy-Bird-Genetic-Algorithm-. (2018). Retrieved 4 May 2021, from https://github.com/ashmeet13/Flappy-Bird-Genetic-Algorithm-?fbclid=IwAR3u172AQV3dsDoVglQzxLv-fwC6z4Zy0v_LyBVygM9YL8G3tMkw4trShso

Genetic Algorithms - Application Areas - Tutorialspoint. (n.d.). Retrieved 2 May 2021, from https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_application_areas.htm

JAIN, S. (2017). Genetic Algorithm | Application Of Genetic Algorithm. Retrieved 2 May 2021, from https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/

Mallawaarachchi, V. (2017). Introduction to Genetic Algorithms — Including Example Code. Retrieved 2 May 2021, from https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Stanley, K., & Miikkulainen, R. (n.d.). Efficient evolution of neural network topologies. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. doi:10.1109/cec.2002.1004508

Ssusnic/Machine-Learning-Flappy-Bird. (2017). Retrieved 4 May 2021, from https://github.com/ssusnic/Machine-Learning-Flappy-Bird?fbclid=IwAR2YFOTfNVA2ANWqtlCF0ol5N7L9BH7INtlXfyex2urU9u876i38tauklO8

Techwithtim/NEAT-Flappy-Bird. (2020). Retrieved 2 May 2021, from https://github.com/techwithtim/NEAT-Flappy-Bird

S, V. (2020). Using Genetic Algorithms to Train Neural Networks. Retrieved 4 May 2021, from https://towardsdatascience.com/using-genetic-algorithms-to-train-neural-networks-b5ffe0d51321

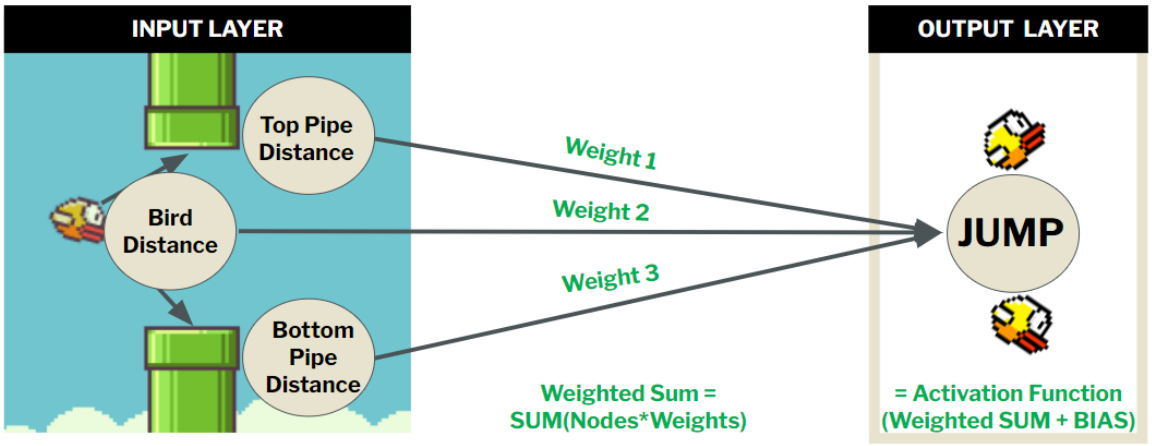
Welcome to NEAT-Python's documentation! — NEAT-Python 0.92 documentation. (n.d.). Retrieved 4 May 2021, from https://neat-python.readthedocs.io/en/latest/index.html

# 8. Appendix

## Appendix A: Open sourcing code examples for genetic algorithm

- Image drawing: https://github.com/anopara/genetic-drawing
- Optimization: https://github.com/ameya98/GeneticAlgorithmsRepo
- Stock trading: https://github.com/charlescao2019/capstone_stock_trading_policy_optimization-master

## Appendix B: Model




## Initial Hyperparameters

```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 100
pop_size              = 50
reset_on_extinction   = False

[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
```

```
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

# connection enable options
enabled_default         = True
enabled_mutate_rate     = 0.01

feed_forward            = True
initial_connection      = full

# node add/remove rates
node_add_prob           = 0.2
node_delete_prob        = 0.2

# network parameters
num_hidden              = 0
num_inputs              = 3
num_outputs             = 1

# node response options
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_max_value      = 30.0
response_min_value      = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_max_value        = 30
weight_min_value        = -30
weight_mutate_power     = 0.5
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism      = 2

[DefaultReproduction]
elitism            = 2
survival_threshold = 0.2
```

## Appendix C: Model Output

```
****** Running generation 0 ******

Population's average fitness: 3.81000 stdev: 1.82617
Best fitness: 7.30000 - size: (1, 3) - species 1 - id 10
Average adjusted fitness: 0.288
Mean genetic distance 1.945, standard deviation 0.357
Population of 10 members in 1 species:
  ID   age  size  fitness  adj fit  stag
 ====  ===  ====  =======  =======  ====
    1    0    10      7.3    0.288     0
Total extinctions: 0
Generation time: 2.794 sec

****** Running generation 1 ******

Population's average fitness: 5.02000 stdev: 2.19718
Best fitness: 7.30000 - size: (2, 4) - species 1 - id 12
Average adjusted fitness: 0.535
Mean genetic distance 1.416, standard deviation 0.410
Population of 10 members in 1 species:
  ID   age  size  fitness  adj fit  stag
 ====  ===  ====  =======  =======  ====
    1    1    10      7.3    0.535     1
Total extinctions: 0
Generation time: 2.786 sec (2.790 average)

****** Running generation 2 ******

Population's average fitness: 3.60000 stdev: 0.58310
Best fitness: 4.10000 - size: (3, 6) - species 1 - id 26
Average adjusted fitness: 0.706
Mean genetic distance 1.440, standard deviation 0.341
Population of 10 members in 1 species:
  ID   age  size  fitness  adj fit  stag
 ====  ===  ====  =======  =======  ====
    1    2    10      4.1    0.706     2
Total extinctions: 0
Generation time: 1.378 sec (2.319 average)

****** Running generation 3 ******

Population's average fitness: 3.29000 stdev: 0.70915
Best fitness: 4.00000 - size: (4, 7) - species 1 - id 29
Average adjusted fitness: 0.556
Mean genetic distance 1.374, standard deviation 0.410
Population of 10 members in 1 species:
  ID   age  size  fitness  adj fit  stag
 ====  ===  ====  =======  =======  ====
    1    3    10      4.0    0.556     3
Total extinctions: 0
Generation time: 1.346 sec (2.076 average)

****** Running generation 4 ******
```

Population's average fitness: 3.43000 stdev: 0.65582
Best fitness: 4.00000 - size: (5, 9) - species 1 - id 38
Average adjusted fitness: 0.644
Mean genetic distance 1.015, standard deviation 0.320
Population of 10 members in 1 species:

| ID | age | size | fitness | adj fit | stag |
|---|---|---|---|---|---|
| 1 | 4 | 10 | 4.0 | 0.644 | 4 |

Total extinctions: 0
Generation time: 1.347 sec (1.930 average)

****** Running generation 5 ******

Population's average fitness: 3.59000 stdev: 0.60241
Best fitness: 4.10000 - size: (6, 9) - species 1 - id 50
Average adjusted fitness: 0.700
Mean genetic distance 1.361, standard deviation 0.226
Population of 10 members in 1 species:

| ID | age | size | fitness | adj fit | stag |
|---|---|---|---|---|---|
| 1 | 5 | 10 | 4.1 | 0.700 | 5 |

Total extinctions: 0
Generation time: 1.381 sec (1.838 average)

## Appendix D: Model Output