Table of Contents

**Operations Guide1**

Software and Hardware Restrictions1

Installation Guide2

Data Files3

Installation Tests4

Modifying the Genome6

Backing up Genomes12

System Recovery13

**Programming Guide**1**4**

Module Chart14

Testing Procedures16

Source Code Modules17

**Operations Guide**

The WANTool is a piece of software that will allows users to run regressions using an algorithm called NEAT (Neuro Evolution of Augmenting Topologies). The operations guide will show you how to install, modify, and test the underlying software of the system.

**Software and Hardware Restrictions**

In order to run this software, the system must have a 32-bit processor or better with any video card (most any desktop or laptop computer will fulfill this requirement). In addition, the system needs to have Java 1.6 or newer installed.

|  |  |
| --- | --- |
|  | Requirements |
| **Windows, Mac, Linux** | |
| **Operating System** | Windows 7 or higher, Mac OS X Lion or higher, Linux |
| **CPU** | Core 2 Quad Q6600 at 2.4 GHz or AMD Phenom 9850 at 2.5 GHz |
| **Memory** | 4 GB RAM |
| **Media** | CD-ROM, 2x or higher |

**Installation Guide**

In order to install the WANTool, the user should place the WANT CD into the appropriate disk drive. Once in the disk drive the user must navigate to the appropriate CD Drive location should they not already be prompted to do so by their operating system. To navigate to the CD Drive location follow the steps below that correspond to your operating system:

**Windows:** Open File Explorer and navigate to “This PC”(Windows 8.1 or higher) or “Computer”(Windows 8 and prior). Double click on the appropriate disk drive to find the WANT software.

**MAC OS:** Open finder and press shift+cmd+c. Double click on the appropriate disk drive to find the WANT software.

Now that you have successfully navigated to the disk drive you should see a file named “**WANT.tar.gz**”. Extract the files to the desired directory by using your preferred file extractor and then selecting the appropriate file path. Navigate to the chosen directory and open the folder. You should now see the jar file (WANT.jar). To start the program double-click the jar file.

**Data Files**

The WANTool CD comes with four data files already included to test the system. The files are listed below and can be found in the saves folder:

NSDUHData-Short.txt

NSDUHData-Long.txt

NSDUHPredictions.txt

params.gapf

Inside these files is the data you will need to run the system. The current data sets provided to the system has 14 columns and varying rows. The NSDUHData-Short contains 100 rows and the NSDUHData-Long contains 1000 rows. The NSDUHData-Long.txt file contains a dataset with more variety which will help the network make better predictions and the NSDUHData-Short.txt file allows the network to train more quickly but sacrifices accuracy by doing so. Each column of the data set represents a different factor that could influence someone to smoke and is illustrated in Figure 1.1 below. The final dataset NSDUHPredictions.txt is the entire data set that should be fed into the network once training is completed. This is the dataset that the network will try and predict and once it has done so it will output how accurate the network was in predicting the dataset.

As stated above the original dataset has 14 rows and each row corresponds to a different factor of an individual. For example, someone with a 1 in the first column makes less than $20,000 and someone with a 1 in the addicted column has addictive personality traits.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <$20,000 | $20,000-$49,999 | $50,000-$74,999 | >$75,000 | College | Addicted | <17 | 18-25 | 26-34 | >35 | Employed | Govt Assitance | M/F | Demand |

Figure .1

Other than the three datasets the user should also be familiar with the data that is saved when a genome is saved to a file. The genome is saved to a special file called epoch.gaif which stands for genetic algorithm input file. The epoch.gaif file contains all the data from a saved genome and does not come with the CD but rather will appear after the user first starts the program and starts saving genomes.

The last file included with the CD the user should be aware of is the params.gapf which stands for genetic algorithm preferences file. Inside this file is all of the data needed to modify the way the WANTool modifies the code, but this file will be explained in depth in the section below

**Installation Test**

A series of tests can determine whether or not the system has been installed correctly. Starting the application, if any red colored messages are received in the main output, there is a problem with the installation. Make sure all data files, including the system manual documents, are present; the application is located in a directory with read/write privileges within your operating system; and try again. Once there are no errors at startup, start by loading the default parameter file: **params.gapf**.

If more red error text is present within the main output due to the parameter file, or this file has been lost, make sure all of the values within the file are positive integers. For the default test case, the values are as follows:

**Population Size = 1**

**Number of Generations = 100**

**Backup Every Nth Generation = 100**

**Dataset file name = "NSDUHData-Short.txt"**

**Advanced Network Creation = True**

**Number of Inputs = 13**

**Number of Outputs = 1**

**Number of Hidden Layers = 1**

**Number of Neurons in each Hidden Layer = 1**

**Number of Hidden Layers = 1**

**Neural Network = 13, 1, 1**

**Maximum number of Species = 6**

**Species Threshold = 0.15**

**Threshold Perturbation = 0.02**

**Extinction Limit = 20**

**Reward for Youth = 1.2**

**Penalty for Old Age = 0.8**

**Old Age = 35**

**Young Age = 16**

**Add Neuron = True**

**Add Neuron Rate = 0.05**

**Maximum Checks for Neurons = 20**

**Add Link = True**

**Add Link Rate = 0.05**

**Maximum Checks for Links = 20**

**Add Looped Link = False**

**Add Looped Link Rate = 0.10**

**Maximum check for Loop Link = 10**

**Change Type = False**

**Chance of Changing Type = 0.1**

**Chance of Changing to Sigmoid = 0.5**

**Change Bias = True**

**Chance of Changing Bias = 0.2**

**Change Input Link = True**

**Chance of Changing Link Weight = 0.8**

**Change Input Neurons = False**

**Chance of Changing Input Neurons = 0.5**

**Mutate Neuron Weights = True**

**Mutate Link Weights = True**

When these values have been entered and saved, the program will allow you to continue on. More about these values will be explained in the next section. The final step to ensure the installation process has successfully completed is to start a new generation process by clicking the “Start” button on the bottom right of the screen. The application will now successfully processing generations, and if not, these issues may have to do with a problem to find the “NSDUHData-Short.txt” file. Make sure this is located within the same directory as the executable.

**Modifying the Genome**

This section describes how to modify the underlying WANTool, and all of the different parameters that the WANTool provides to the user.

As briefly explained in the section above there is a special file included on the CD called params.gapf inside the saves folder. Inside this file are numerous parameters that the user can turn on and off or change the value of. Below is a list of the parameters that can be changed and what each does to the system and the effect the parameter has on the system. The params.gapf file is broken down into sections that modify different parts of the genome. For example there is a section to modify all the species parameters, a section for the mutation parameters, and many more sections.

The first section found in the file is a section for the epoch parameters. Within this sections is the population size and number of generations. Both of these parameters will determine how long it will take to run the program and it is suggested that for the population size you choose a value between 50 and 150 for the best results. Below is a listing of the parameters in this section and what they do:

*Population Size:* The population size describes how many genomes will exist per generation. As mentioned above the ideal size for a population is between 50 and 150. Lower number will produce faster results and likely converge faster but may suffer from converging to the wrong point.

*Number of Generations:* The number of generations parameter describes the number of generations your genomes will go through before exiting the program. Note that as the generation number increases so too does the time for the program to complete. The program slows down as generations get larger because the overhead for the network becomes larger as well.

*Backup every Nth Generation:* This parameter is one of the most important in the entire file. This parameter describes how often the genome gets saved to an output file. If this value is set to zero, then this feature is essentially disabled and this is not recommended. The value of this parameter can actually vary by a lot depending on the type of analysis being done. Data sets with a low number of input variables will go really fast and thus the value of this param would be higher. The recommended value, is a value that saves the genome every 5-10 minutes thought users can choose something smaller if they wish.

The next section is used to describe the data set parameters or input file parameters. This section allows the user to change the data the system reads in and allows the user to predict different data sets simply by changing the structure of the starting neural network and changing the input file. Note: The network can handle up to about 1000-2000 data inputs before slowing down due to the sheer size of the computation needed to process the data.

There are two ways to change the structure of the network. The first is simpler and is easier to read. The user simply states the number of input neurons the network will use, the number of outputs, the number of hidden layers, and the number of neurons within a hidden layer. Keep in mind to keep these numbers as low as possible as the WANTool also tries to minimize the network and can’t if it starts out to big. To use the advanced creation set the advance network creation to True and then follow the following format below:

Number of Input neurons, Number of neurons in hidden layer 1, Number of neurons in hidden layer 2, …, Number of neurons in hidden layer x, number of output neurons

The advanced creation gives the user more flexibility in the hidden layer by allowing the user to specify the number of neurons in each hidden layer individually. Below is a listing of the parameters the user can change in this section and an explanation of how it affects the genome.

*Dataset file name:* This is the name of the file where the users data is located. This will be explain this later, but all of the data values need to be either 0 or 1 for the network to train properly. The file also has to be in the current working directory or the installation directory in order to be loaded properly.

*Advanced Network Creation:* This parameter takes in a Boolean value (True or False) and determines which set up the user wishes to use. The user should set the value to true if they wish to use the advanced format or false if they wish to use the simpler format.

*Number of Inputs:* This parameter takes in any value greater than 1 as an input. Though it can take in any value greater than 1, it is suggested not to exceed 20 inputs to the network as this will cause the network to slow to a crawl. This parameter just describes the number of inputs the given dataset has, keeping note that all dummy variables are included in this count.

*Number of Outputs:* This parameter is similar to the number of inputs parameters and should only take in values between 1 and 20 though it is possible to take in more. This parameter simply describes the number of outputs in the model and in most cases will be 1.

*Number of Hidden Layers:* This parameter describes the number of hidden layers in the genome. It is recommended that the user stick with the defaults for this value as well as it is beneficial to start the network as small as possible so the WANTool can do its work and create the best network it can and keep it as small as it can. There does exist the possibility that the user wants to create a specific network to start with and that is why this parameter exists.

*Number of Neurons in each Hidden Layer:* This is exactly what the parameter says it is. This describes the number of neurons in each hidden layer. This is the major difference between the simple setup and the advanced setup. The user is more limited with the simple setup than advanced. Like the number of hidden layers, it is recommended that the user keep the defaults as the network should start as small as possible.

*Neural network:* This parameter was described a little bit in the above part but this parameter is just a combination of the 4 parameters above and creates the beginning network topology. An example of a network with 9 input neurons, 2 hidden layers with 4 neurons in the first hidden layer and 3 in the second, and 5 output neurons is shown below:

 9, 4, 3, 5

The first value 9 represents the number of input neurons and the last value 5 represents the number of output neurons. All the number in between represent the number of neurons in that hidden layer. Each value needs to be separated by a comma and each value has to be greater than 1.

The next section, Species Params, describes all the parameters that define how speciation works. This section has the potential to being the most useful and is one with a lot of possibilities and has little experimentation. The user is encouraged to play with the default values and see which ones work best as there are plenty of combinations to be tried.

*Maximum Number of Species:* This parameter describes the maximum number of species that can exist in a generation. This parameter can take on a value between 1 and the population size, though it is suggested that the user select a value that is significantly lower than the population size. It is important that some thought is taken with this variable as setting it too high can lead to a population not converging at all and setting it too low can lead to a fast convergence or one species may dominate the others.

*Species Threshold:* This value is adjusted during runtime to reflect the maximum number of species but can be a good way to encourage or discourage speciation in the beginning by setting it high or low. This variable primarily affects the first 100 generations and starts to converge to a value that reflects the maximum number of species. Note that setting the value extremely high will encourage speciation in the beginning and setting the value extremely low will discourage speciation. As will be explained in the next parameter the species threshold works in tandem with threshold perturbation.

*Threshold Perturbation:* This parameter represents by how much the species threshold changes during runtime. This parameter can take on any value between 0.00 and 0.99, though it is recommended to keep the value between 0.01 and 0.05. Note: Do not set the value to 0 unless you want to hold the species threshold constant. Holding the species threshold constant means the species is free to move between any number of species and the maximum number of species parameter no longer applies.

*Reward for Youth*: This parameter defines the reward given to those species who fall below the defined limit for young age. The parameter can take on any value but should be between 1.00 and 2.00. Setting this value high encourages new species to flourish and setting it too low could discourage the newer species.

*Penalty for Old Age*: This parameter is similar to the reward for youth parameter but does the opposite and defines the penalty given to older species. Though this parameter can take in any value, it is suggested to give it a value between 0.25 and 1.00.

*Old Age*: This parameter simply defines the age at which a species is defined as old and receives the old age penalty.

*Young Age:* Like above this parameter simply defines that age at which below a species is considered young and receives the reward for being young.

The next section is the mutator params section and is another one of the more important sections to the file as it has many options and defines how the network mutates. There are plenty of variables to change and it is important to recognize that not all of them need to be turned on in order for the network to be effective. In fact, a good combination of 5-6 mutators is typically best and if the user finds that some mutators aren’t working then this section allows them to choose different mutators and reload the genome.

*Add Neuron/Add Link:* These two are the basic mutators and should almost always be enabled. These both take in a Boolean value and by default are set to true. Adding neurons and links is what allows the network to change topologies and is the main principle behind the WANTool. The only time these should be disable is if a specific network is required, but if this is the case then a traditional neural network with backpropagation will work a lot better.

*Add Link/Neuron Rate:* These parameters define the rate at which links and neurons are added to the network. The parameter will accept any value between 0.0 and 1.0 but it is suggested not to pick a number that is really high as the network will grow very large very quickly and could not find the minimum topology.

*Maximum Check for Links/Neurons:* This parameter defines how many times the network will search for a neuron or link to add. This is because even though the network may choose to add a neuron or link based on the add link/neuron rate, the network then has to find a place to add the neuron or link and this parameter defines how many times it will look for a place to add the neuron or link. This parameter can take on any value and it is actually suggested the user choose a large number (20 - 40). This parameter exists so the network doesn’t infinitely loop searching for a place to add a link when none exist.

*Add Looped Links:* This value turns on the mutator that adds links. This parameter is actually ineffective at the current writing of this manual and will be included in the next release of the WANTool.

*Add Looped Link Rate:* This value determines the rate at which looped links are added. As with many of the other parameters like this one, it can take a value between 0.0 and 1.0 but it is suggested to pick a number closer to zero than one because looped links can be added quite frequently and should be used sparingly.

*Maximum Check for Loop:* Similar to the max check for neurons and links, this parameter determines how many times the network looks for a place to add a looped link and it is suggested to choose a larger number.

*Change Type:* This parameter turns on the function that will change the neuron type from either sigmoid to step or vice versa. Be careful using this mutators as unless the user specifies the speciation parameters well, the network will converge very quickly and will likely not change.

*Chance of Changing Neuron Type:* This parameter defines the chance that a neuron will attempt to change its type. As mentioned above, care should be taken when setting this parameter as changing neuron types can cause really quick convergence.

*Chance of Changing to Sigmoid:* This parameter defines the chance that the neuron will be changed to a sigmoid neuron. The reason for this parameter is because a user may want to have more sigmoid neurons in the network than step neurons and this parameter allows the user to specify this. This is also because the step neurons are what are likely to cause the network to converge too quickly but can also give the best network that slight little edge to make it better.

*Change Bias:* This mutator changes the bias weights of the neuron and it is strongly encouraged to utilize this parameter as it strongly affects how neurons activate and can encourage randomness in the genome.

*Chance of Changing Bias:* This parameter like the others simply describes the chance or rate at which the bias weight will be changed. This parameter can take on a value between 0.00 and 1.00 but users should stay away from the two extremes.

*Change Input Link:* This mutator changes input links/neurons and helps a lot with regression data as it helps assign weights to inputs and makes a huge difference in allowing the network to evolve.

*Chance of Changing Input Link/Neuron:* Just like all the parameters above, this parameter can take the value of 0.00 to 1.00 and describes the rate/chance at which input links/neurons are modified.

*Mutate Neuron Weights/Mutate Link Weights:* These are another mutator that is essential to the WANTool and should be left on as this is another mutator that helps modify the network as a whole.

**Backing up Genomes**

Backing up the genomes is made incredibly easy by the WANTool. The WANTool already backs up genomes every few generations, which is a parameter that can be set by the user in the params.gapf file. Once the genome has been outputted to the file, it is suggested that the user backup the genome by copying the epoch.gaif to an online storage medium or a zip drive. Though very unlikely, a corrupted genome file can be prevented by simply keeping a backup of the genome file in a separate location.

**System Recovery**

In the event of some kind of disaster it is nice to have procedures in place to recover from these disasters. The WANTool makes recovery easy as long as the user has not disabled the recovery feature which is explained in depth in the above section about how to modify the params.gapf. Located inside the params.gapf file is a parameter called Backup Every Nth generation. To disable this feature set the parameter to 0, but to enable it set the parameter between 50 and any value. What this feature does is allow the user to specify how often the genome is saved so that should something happen with the system while it is running, not all data will have been lost.

Power failure: In the event of a power failure, not all is lost. If the user has specified in the params.gapf a value for the backup every Nth generation, then to recover from a power failure all the user has to do is restart the console, and load the last saved genome which should be located in the epoch.gaif. There exists the potential that some data could still have been lost between generations but reloading the last saved genome will prevent the user from losing too much data and should provide a good backup point to recover from. The user should keep this in mind when setting the backup parameter and determine how many generations they can afford to lose. If they are running regression data that take hours to get through 50 generations then it may be useful to set this parameter at a lower number but if the WANTool is creating a new generation every second then the user may decide to set the backup to a larger number. It is recommended to set the parameter to save every 5-10 minutes.

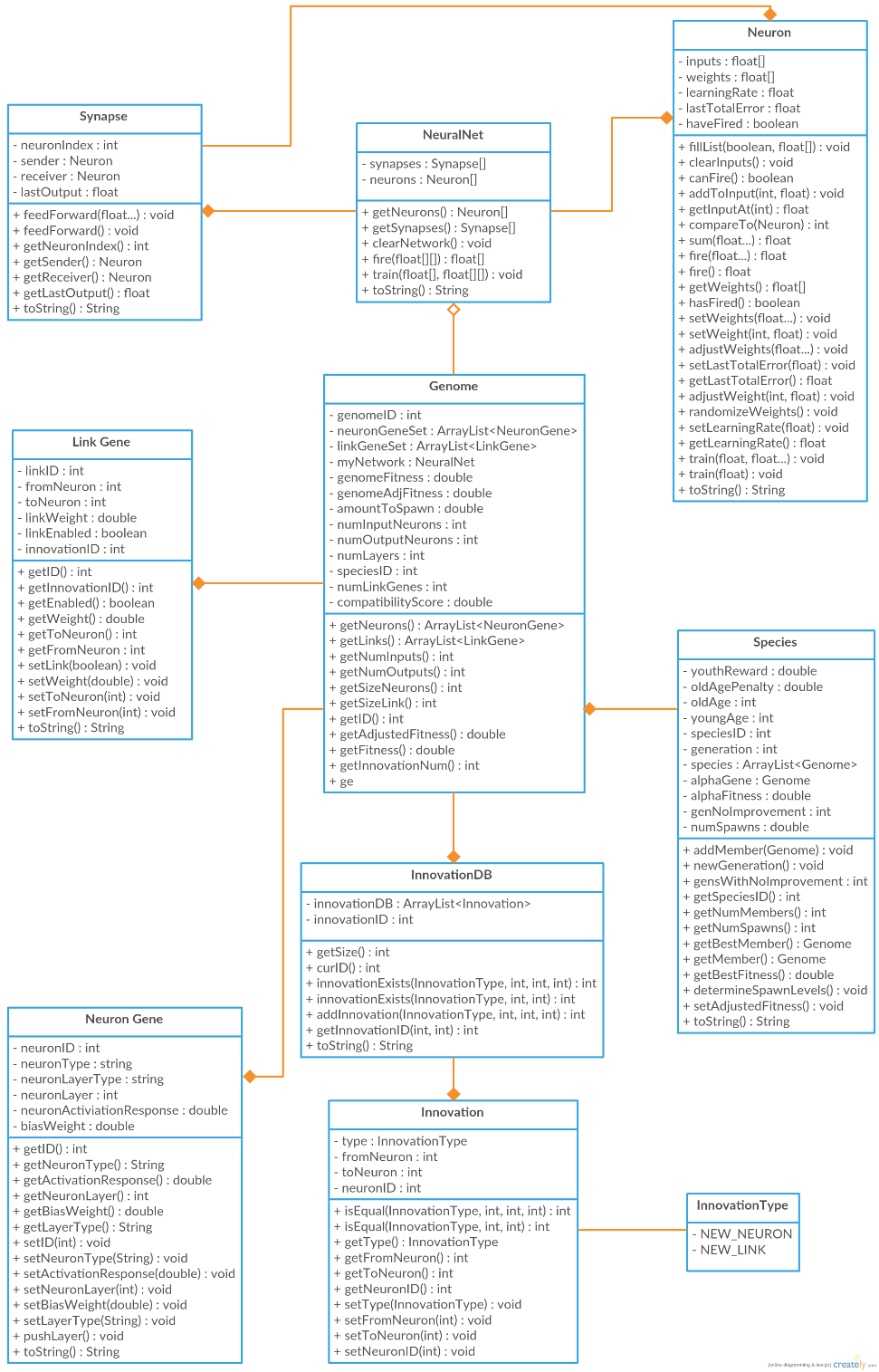
Software failure: There are many things that can constitute a software failure and the WANTool tries its best to mitigate any unnecessary loss of data. The worst case scenario is similar to the situation encountered in a power failure. Simply restart the console and load the last saved genome. The amount of data lost is dependent on the backup parameter explained in the opening paragraph of the recovery section.

File Corruption: The only way to properly recover from a corrupt data file is to follow the proper backup procedures listed above. It is important to maintain copies of the genome object files in case something catastrophic were to occur with the system. To load from backups, simply navigate to the folder containing the back .gaif files and load the backup you wish to start from.

**Programming Guide**

The programming guide describes in depth the underlying code of the system, provides documentation on how to maintain the code, and explains the different modules that are associated with the WANTool.

**Module Chart**

Below is a UML Class diagram of the WANTool: ****

The module chart describes the various classes of the WANTool and how they interact with each other. The highest level of the diagram is the species class which is composed of multiple genomes. Inside genome, there is a lot going on. Genome is composed of 3 different things, and is has a neural net inside it as well. The genome is composed of link genes, neuron genes, and the innovation database. Within the innovation database, is innovations which implement their own type called InnovationType which consists of NEW\_NEURON and NEW\_LINK. As mentioned earlier, Genome also has a NeuralNet class inside it. The NeuralNet is composed of Synapses and Neurons and Synpases is composed of two Neurons.

The module chart also describes the various methods and variables of each class. The (+) sign indicates a public function or variable and the (-) sign means the variable or function is private or protected. The solid diamonds represents composition and the open diamonds represent aggregations.

There is a GenomeGUI class that ties together all of the classes to create a working program but this class was left out of the class diagram because it contains all of the classes in the above diagram.

**Testing Procedures**

The WANTool comes with a separate tester file for the underlying mutators of the code. The tester file is named Tester.java and is located in the \src\ directory. The Tester.java file contains the source code for running the test modules. If the user wishes they can go into this file and turn on some test cases and see how everything works. Once the proper changes to the file have been made, to run and compile the code follow the steps outlined below in a command prompt:

To complile and run

Compile: javac –cp lib\\* src\Tester.java

Go to src directory

Run: java -cp ..\lib\\*; Tester

The Tester.java allows the user to change 4 different test functions. The functions are outlined below:

*Test AddLink:* This function tests the addition of links to the genome. The test function first prints out the old genome and then attempts to add links to the genome. It loops through 10 times to try and add a link as it may not always find neurons to add a link to. Once the network has added some links it will prompt the user to output the new genome which should have some extra links now.

*Test AddNeuron:* This function tests the addition of neurons to the genome. The test function first prints out the old genome and then attempts to add neurons to the genome. It loops through 5 times to attempt to add neurons to the genome. Once it has finished adding the neurons, the test will prompt the user to print the new genome which should have some extra neurons now.

*Test Fire:* The test fire function is a simple function that takes the genome and fires its neural network. The test will have passed if 4 outputs are seen. The value of the outputs is irrelevant and will tend to be close to 1.

*Test Mutators:* This test function is a comprehensive test of all of the current mutators. The test will prompt the user to print the old genome, it will then do all its mutations, and print out whether the mutations were successful or not. The test will then prompt the user to output the new genome which should have updated weights.

**Source Code Modules**

**Main.java**

**Written By:** Coved Oswald

**Date Last Modified:** 28 Apr 16

**Description:** The main class is the class that brings all the classes together and runs the entire program. The main class calls the GUI class which is located in the Utils directory. The GUI then makes the appropriate calls to start the epoch, load param files, load genomes, and save genomes.

***Genetic Algorithm***

**Epoch.java**

**Written By:** Kaleb Luse, Coved Oswald

**Date Last Modified:** 29 Apr 16

**Description:**

**Genome.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 29 Apr 16

**Description:** This class describes a collection of LinkGenes and NeuronGenes to form a NeuralNet. This class will form the phenotype in order for an individual to fire, and trians a neural network based on the N. E. A. T. algorithm. Of course, this class relies on methods defined in GenomeHelper.java to do this.

**GenomeHelper.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 24 Apr 16

**Description:** GenomeHelper.java defines a set of functions that help a Genome operate correctly. These functions are final and should not be overwritten or modified: their functionality is defined well and cannot be extended.

**Innovation.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 24 Apr 16

**Description:** This class represents a single Innovation to be entered into the InnovationDB. An Innovation is any addition to the network that can be described in a database setting to be used for NEAT. A single Innovation consists of an InnovationType, and different values depending on whether or not this type is adding link or a neuron. If a neuron is to be added, then the neuron is put between two neurons on a link, making the "sending" neuron the left neuron, and the "receiving" neuron the right neuron, while the neuron's id is set to the added neurons ID. However, if a link is to be added, the sending and receiving neurons are what you think they would be for a link, but the neuron ID is set to -1.

**InnovationDB.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 29 Apr 16

**Description:** An InnovationDB is a data structure that allows for the storing of Innovations that describe change in a population of Genomes. These Innovations could later be used by the Genome to test past changes in order to make better changes in the future, ensuring the same changes aren't being made.

**InnovationType.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 22 Apr 16

**Description:** This enumeration describes the different types of innovations that can be entered into a database. This includes, for now, NEW\_LINK as well as NEW\_NEURON.

**LinkGene.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 19 Apr 16

**Description:** This class contains the metadata used to construct a Synapse. This class also contains data used to help form and maintain generations within a population.

**NeuronGene.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 27 Apr 16

**Description:** This class represents a Neuron as it appears inside of a genetic algorithm context. A NeuronGene consists of the metadata used to form many different kinds of Neurons. This will allow another class to take advantage of this metadata in order to form an actual Neuron.

**Predictions.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 28 Apr 16

**Description:** This class takes in a trained neural network and then uses the network to make predictions of a dataset.

**Species.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 25 Apr 16

**Description:** Determines a cluster of similar Genomes, effectively creating new Species within a given population. This class will be used by the main method of this application to assert new species in order to save the newest outliers within a population. This is because new code Genomes may have good topologies, but contain bad weights initially.

**Tester.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 29 Apr 16

**Description:** The tester class tests the functionality of the genome class and all of the genomes mutator functions.

**Species.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 25 Apr 16

**Description:** Determines a cluster of similar Genomes, effectively creating new Species within a given population. This class will be used by the main method of this application to assert new species in order to save the newest outliers within a population. This is because new code Genomes may have good topologies, but contain bad weights initially.

***Neural Network***

**NeuralNet.java**

**Written By:** Coved Oswald

**Date Last Modified:** 8 Apr 16

**Description:** A NeuralNet is a set of Neurons interconnected by Synapses to compute a set of outputs. This is done by feeding the NeuralNet a set of inputs into a set of Neurons, and calling the interconnected Synapses.

**Neuron.java**

**Written By:** Coved Oswald

**Date Last Modified:** 6 Apr 16

**Description:** This class represents a single artificial cell in a network of said cells that receives input, processes those inputs, and generates an output. This model was described by Warren S. McCulloch and Walter Pitts in 1943, and is now prevalent in the form of a Perceptron. It is this model that is implemented in this class.

**Synapse.java**

**Written By:** Coved Oswald

**Date Last Modified:** 2 Mar 16

**Description:** Describes a connection between two different Neurons. This connection is not weighted; it is up for the Neuron to decide the weights for every input. The Synapse does keep track of the receiving and sending Neurons, and the Synapses method should be called every time a network is to feed a value from one layer to the next.

**XORTest.java**

**Written By:** Coved Oswald

**Date Last Modified:** 8 Apr 16

**Description:** This class is a tester class that tests the functionality of the NeuralNet class and the classes that NeuralNet is composed of.

**SigmoidNeuron.java**

**Written By:** Coved Oswald

**Date Last Modified:** 04 Apr 16

**Description:** A Neuron that uses the sigmoid function as its activation function.

**StepNeuron.java**

**Written By:** Coved Oswald

**Date Last Modified:** 04 Apr 16

**Description:** A Neuron that uses a step function as its activation function. This step is determined by a StepNeuron threshold and can be set by the programmer.

***Util***

**Debug.java**

**Written By:** Kaleb Luse

**Date Last Modified:** 29 Apr 16

**Description:** A class that allows for the output of information to a debug terminal. This is determined by the com.riskybusiness.util.Debug#DEBUGGING switch that allows a programmer to set debugging to either true or false. However, once the debug window has closed, it can no longer be reopened until a new instance of the application has been started again.

**GenomeGUI.java**

**Written By:** Coved Oswald

**Date Last Modified:** 29 Apr 16

**Description:** The GenomeGUI class defines the Graphical User Interface that is used to interact with the Epoch method and params file.

**GenomeGUIOutputStream.java**

**Written By:** Coved Oswald

**Date Last Modified:** 28 Apr 16

**Description:** A class that sends output from an output stream to the console screen of the application. The console screen is part of GenomeGUI, but the output stream can be from a file, standard output, or any valid print stream.

**PrinterHelper.java**

**Written By:** Coved Oswald

**Date Last Modified:** 27 Apr 16

**Description:** This class provides methods helpful for printing using the javax.swing.text libraries. This class is final can cannot be overridden; the methods declared here should not be changed whatsoever.