**Neuro Evolution of Augmented Topologies (NEAT)**

*By Bernardo Olivera Santamaría*

Hello, my name is Bernardo Olivera Santamaría, better known as Bernardo Olisan. I have been coding since I was 10 years old, I began with robotics, learning Arduino and building things from scratch, like wireless chargers and drones. I also went to nationals in robotics and won second place. Since then, I’ve coding and learning all by myself, moving into all the areas of computer science, ML, game development, web development, hardware hacking and more…

1. **Why NEAT?**

For my university project I decided to build a neuro evolution from scratch with pure math, and C++. Good computer scientist are the ones that understand deeply how the things work, because of this, I consider that doing things from scratch is the key for learning. On the other hand, is not a good habit to apply this in work, because you need to be efficient and use the best solid tools.

Genetic algorithms are formidable, they find a solution based on a context, looking at this technology in training process, is just stunning. That is why I choose it. I am not going to explain how this technology works but is good to give a little bit of context.

It all starts with a blank neural network, input, and output layers only, no hidden, no connections. At the first lifecycle, the neural networks randomly mutate by just adding a hidden neuron or a random connection, at the end of the lifecycles, the ten well performed structures crossover and output five offspring’s, these ones are multiply later, and the process repeats again until we find the perfect structure for the task.

The task for **NEAT** that I created was a rocket game. The rocket needs to exit at the point mark in the atmosphere. The rocket is going to learn using the **NEAT** algorithm until a rocket is successful.

**2.0 Project Structure**

One thing to notice in the code is that we use a structure to order it, we count with three important files. *Main.cpp* is the main code, where all functions and modules are there, as well as it contains the **evolve process**. *Model.h* is where all the functionality is. The class **RocketNN** is also there, were we defined the rocket graphics, weighted sum, draw\_nn, and rocket\_mutation. It also contains the atmosphere function and the key function. *Modules.h* has all the functions that were called more than once.

**3.0 Game Graphics**

For the UI I used OPENGL, a famous, low level, cross platform API, for rendering graphics, with the help of GLFW for the window context creation and key callbacks.

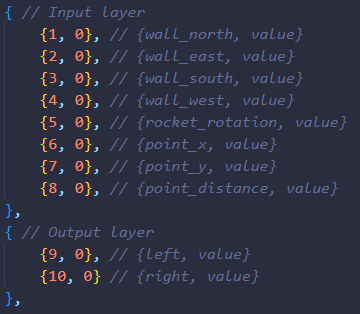
Our graphics are divided into two windows, nn\_window and window. The window is the one that render the rocket game (atmosphere, rockets, and point marks). The nn\_window is the one that render the best neural network structure (synapsis and neurons).

For the **atmosphere** is quite simple math, using trigonometric functions and raw loops for curving the line. Also adding the point mark in the right line position.

For the **rocket** graphics I tried to do it as simple as possible, the important thing to denote here is that we need to get the data from the rocket to pass it to the input layer. Is just a simple rectangle.

The others are just key events to make it more game familiar.

**4.0 Rocket Neural Network (RocketNN)**

In *model.h* we have a class named **RocketNN**, this is the brain for each rocket. For **NEAT** to work well it needs encoding schemes, this are arrays that defines the structure for connections and nodes, in Biology is the same as genotype data.

Each rocket has eight input neurons and two output neurons. The **input layer** receives the position of cardinal directions, point position and rotations. The **output layer** only outputs the decisions for the rocket, left or right.

**4.1 rocket()**

In the class we have a function called rocket(), this manages the movements, collisions, rocket graphics, input values and initial mutations. The initial mutation is the important thing in here, because is going to make the difference in the behavior of the rocket. At run time the rocket is going to take the graphics values and pass the changes to the input layer.

**4.1 rocket\_mutation()**

This function provides us a random mutation for the evolve process, it chooses randomly two mutation options.

1. **Hidden neuron in an old connection.** This will add randomly a hidden neuron in an old connection and disable the old connection. The disable-enable part comes from the encoding scheme functionality, this is how **NEAT** works, as well as we transfer the weight of the old connections.
2. **New connection.** This will add a new connection in the structure, it randomly selects between three types of connections, from input to output, input to hidden and hidden to output, if it has no hidden neurons, it only adds a connection between input to output.

**4.2 draw\_nn ()**

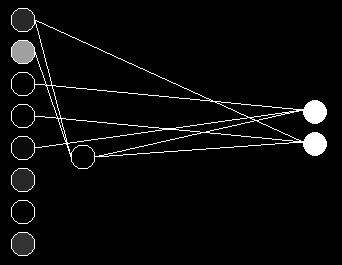
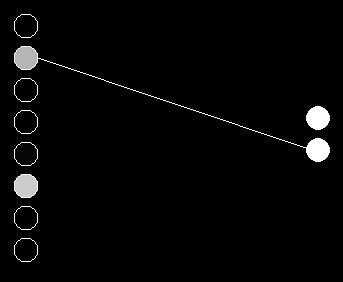
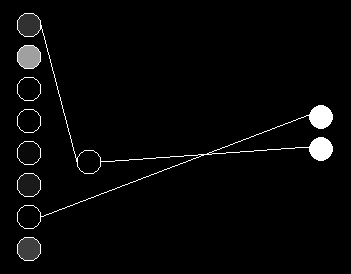
A good practice is to observe the neural network changes at run time, this is the purpose of this function. It draws the neural network by only having the encoding schemes, which is why the schemes are so important, they are the key for controlling everything and observe anything. Draw\_nn manages connections very well, so that at the time of observing, it prevents a mess of connections and nodes.

**4.3 weighted\_sum()**

This function applies the famous weighted sum () to our neural network structure, then it passes to the output neurons the decisions taken and active them.

**5.0 Evolution Process**

The intention of **NEAT** is to use the evolutionary theory of Darwin in artificial neural networks. This is how the neural networks uses genetic algorithms to learn. The evolution process is in *main.cpp*, at the first lifecycle they have an **initial mutation** (explained above), they do not know anything.



**5.1 Half well performed**

At the end of each lifecycle is where the evolve process acts, the first step is to identify the half well performed rockets of the speciation, in this case, we only have twenty rockets, so the half is going to be ten best rockets. We measure them by a score each rocket has, the score is the distance between the rocket and the point mark. After that, we saved them into an array called half well performed.

**5.2 Crossover Phase**

Once we have the half well performed rockets, the crossover phase starts. We crossover ten of them into five children, to apply this, we need two parents, so that they output a child. In this case is like this, rocket 1 with 10, 2 with 9, 3 with 8 and so on.

We will end up with five offsprings genetically enhanced by their parents, then we multiply them by 4 so that they are 20 again and continue.

**5.3 Random Mutation**

Once we have the genetically enhanced offsprings multiplied by four, we randomly mutate them with **rocket\_mutation()** and start again until we find a rocket that has a 9.2 score or higher.

**5.4 Better-Worse System**

In the tryouts I notice that sometimes the structure becomes huge, and it cannot even get a good score, which is why I created this system. Before mutation and evolve process, I saved the old speciation structures (20 rockets) with the sum of the half well performed score and let the new rockets evolve. If the new rockets manage to get a better score than the old rockets, we continue with the new rockets unless if the old rockets have a better score than the new ones, we keep the old ones until the new ones manage to get a better score.

It is worth mentioning that **NEAT** does not use this technique, I use it because it helps to the continued improvement for the rockets.

