# Design Brief

#### NEUROEVOLUTION FRAMEWORK FOR GAMES

**Some things may change in the following brief as I gain a better understanding of genetic algorithms and neural networks.**

## Purpose

The purpose of this library is to be able to create a neural network based artificial intelligence in any game setting without requiring a training dataset. This will be accomplished using an implementation of conventional neuroevolution to train a neural network. In contrast to supervised learning (such as backpropagation), genetic algorithms require only a measure of how well a neural network can complete a task (my implementation will try to maximise a fitness value). In the context of video game development, measuring fitness is significantly simpler than creating a dataset full of correct input-output pairs.

## Maths and Algorithms

**Mathematical operations**

All basic mathematical operators will be used (probably).

A Sigmoid function will be used for the neurons’ activation function. Sigmoid functions are very commonly used for this application because they are continuous, constantly increasing, differentiable everywhere, and are nonlinear. An activation function needs to be continuous and constantly increasing (or decreasing) to guarantee all input values have a unique output value. It also needs to be nonlinear and differentiable for 1) the use of backpropagation, which requires the activation function’s derivative to be dependent on the input values (not used for my implementation) and 2) so that the final output values can have a nonlinear relationship with the input values. Sigmoid functions specifically also ensure all neuron’s output values are between 0 and 1, useful for some use cases (where the final output is a percentage, for example).

**Algorithms**

Note: specific algorithms not yet decided

The neural network will be trained to generates outputs that accomplish a high fitness value for a user specified situation. It will not be using algorithms like backpropagating or other supervised learning techniques, instead using evolutionary algorithms to train the neural network.

My neuroevolution algorithm will be conventional; it will not modify the topology of the neural network, instead just modifying the weights (the topology will instead be defined by the user). I decided to do this in order to lessen the complexity of the project. The system will use direct encoding, meaning all neurons and connections will be directly specified in the genotype.

## Implementation

As required, the system will be modular in its implementation, and will not be bound to a specific application. The neural network will be able to be generated with a variable number of neuron layers, each with a variable number of neurons. The user can also supply the input data, the output data format (aka the number of neurons in the final layer), and a function that calculates the fitness of the network. Given this the program will be able to train a neural network to increase it’s fitness. The user will also be able to read and modify the neural network’s values at will, which can be used to save and load values as needed. After training the neural network the user will easily be able to use it in their own applications.

The library will be developed in c++. It will be built as a static library for use in c++ programs, and a dynamic library specifically so it can be used in unity (utilising platform invoking to access functions). The code will also be publicly available on github, but not in an easy-to-use state (the visual studio project will be publicly available).

## Dependencies

The final library will be written in c++ and will be dependent on glm.