Bournemouth University

MSc CAVE - Animation Software Engineering - NCCA

ASE: Design Document

Giles Penfold
November 15, 2017

1 Proposed Method

Our proposed method takes from the CGI Tech report [1], using Genetic Algorithms [2] combined with Long Short Term Memory Deep Neural Networks [3] to adapt the output of a basic flocking system such as from Reynolds' paper [4]. The aim is to prove that behavioural techniques can be taught to a flocking system, such as object avoidance, predator avoidance and leadership, without the implementation of any other algorithms. The proposed method can be seen in the pseudocode in Algorithm: 1. The proposed layout for the project can be seen in the UML diagram in Figure: 1.

Initially only a single network will be trained without a genetic method being applied, using a fixed flock size. As the project progresses, we hope to include the genetic method for training multiple networks, as well as being able to handle multiple flocks with different sizes concurrently and in a robust manner. We expect to be able to teach the flocks to avoid predators, eventually being introduced to new predators and reacting in a realistic manner towards them - with specific boids taking stronger roles within the flock as a whole.

Algorithm 1 Hybrid LSTM Flocking

```
1: procedure GENETIC METHOD
2:     L ← Array of LSTM RNNs
3:     generation limit ← Maximum number of generations
4:     max epoch ← Maximum number of epochs to train
5: loop through each RNN in L:
6:     Train network alongside flocking algorithm
7:     if current epoch is greater than max epoch then
8:         goto prune
9:     else
10:        goto loop
11: prune:
12:    if current gen is greater than generation limit then
13:        close;
14:    else
15:        Keep top 5 networks, remove all others
16:        Breed remaining networks until L is full
17:        goto loop.
```

1.1 Boid

This is the core class for the flocking algorithm. It provides all the capabilities of a single boid to function within the rules of flocking: Separation, Alignment and Cohesion. It can track it's own position, as well as the position of neighbouring boids, reacting in response to their own movements. When the Flock function is called, the three separate algorithms are called within it and the weighting of each can be adjusted at run-time to allow for dynamic flock movements. The design of this class is based from the code found within the processing.org website.

1.2 Flock

This class represents a cluster of boids that are allowed to interact with each other as a flock. It will store a vector of boids, as well as having the functionality to add boids to the group. After the initialization of the flock, it will simply update the flock movements as well as accept adjustments to movements from the NetworkLinker class. A Boid cannot exist in the scene without being within a flock.

1.3 ColObject

This class is the base class for any collision objects within the scene. Boids would not fall under this category as they will be avoiding objects using the LSTM. These objects can serve as walls, pillars and any other type of static object within the scene.

1.4 Predator

This class inherits from ColObject and is the main threat to the boids. It will move around the scene attempt to kill as many boids as it possibly can, with the network being updated on which boids were killed from the predators after the movements it has predicted the predators to make.

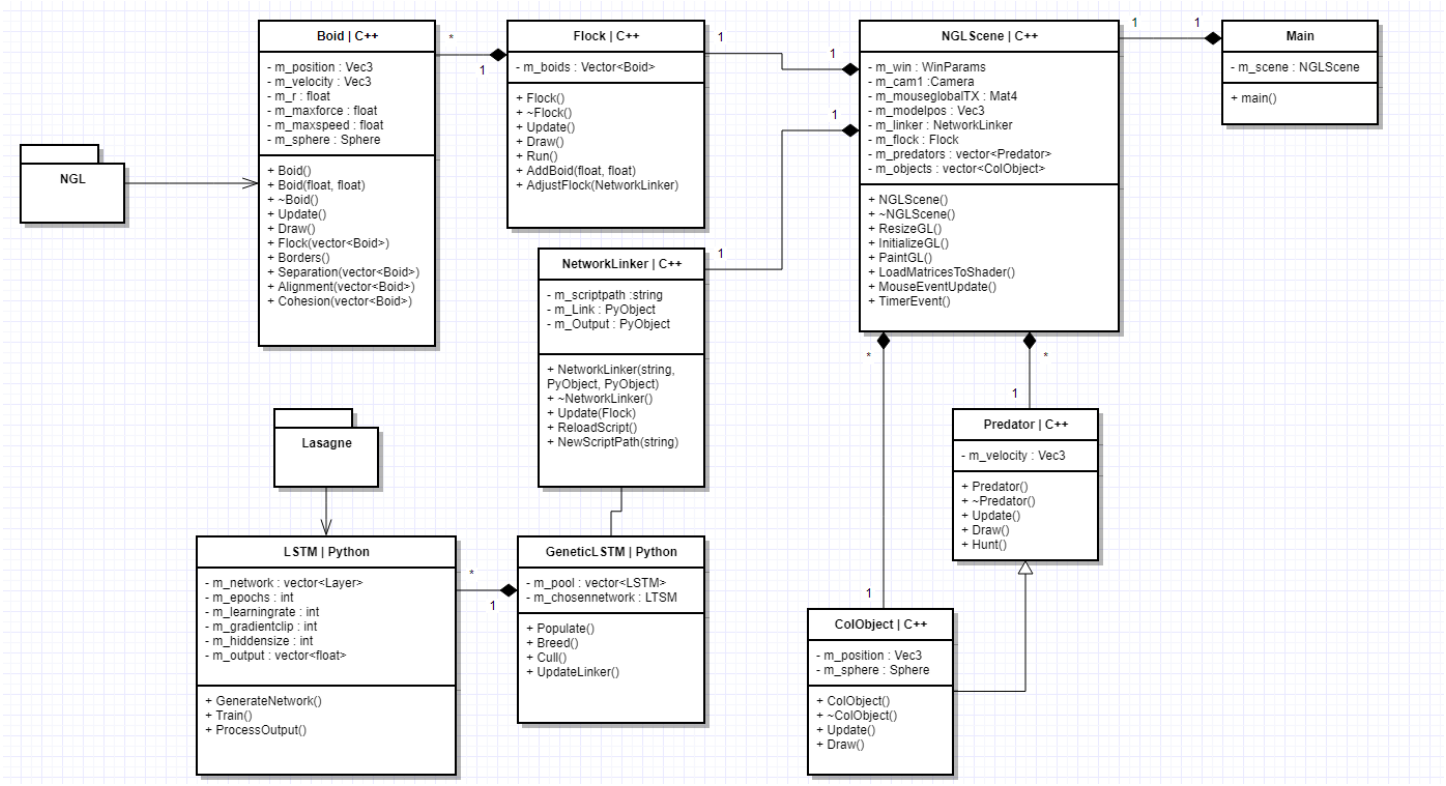

Figure 1: A structured UML layout for the proposed project.

1.5 NetworkLinker

The function of this class is to provide a stable link to the LSTM DNN from Python. It expects to receive the output of a single network from the Python side, a vector of floats, which it can then pass to NGLScene to allow for adjustments to any of the flocks in the scene. Initially this will be dealing with a single network of fixed size output to send to a single flock.

1.6 NGLScene

This class is based off of the NGLScene in the Python demo found on the NCCA Github site. It allows for the drawing of OpenGL objects within the scene, such as the boids of the flock, as well as the input of PyObjects within the scene. This class acts as a container to hold and draw all objects within out scene, also serving as a framework for the communication of the NetworkLinker and any Flocks within the scene. This will be using the NGL library.

1.7 LSTM

This is the core class for a single LSTM DNN, using the Lasagne library as a core framework to build from. It will be able to train itself using the data passed into it from Genetic LSTM, building up a behavioural network for a specific flock. To begin with, there will be a single network with a fixed size handling a single flock. This class will pass its output to the Genetic LSTM class where it can be given to the NetworkLinker.

1.8 Genetic LSTM

This is the Python side class for handling the sending of data to the C++ side, taking the output from the best trained network and handing it to the NetworkLinker. This class also will handle the genetic breeding and production of new networks within the training phase of the project, eventually selecting a single network to represent the best chances of survival for the flock.

1.9 Main

This is the main class within C++ and handles the NGLScene object, as well as the termination of the overall program.

Bibliography

[1] G. Penfold *An Investigation Into Genetic Flocking Simulations: An Opportunity Review and a State of the Art Hybrid Design*. BU. 2017.

[2] P. Koehn. *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*. 1994.

[3] S. Hochreiter, J. Schmidhuber. *Long short-term memory*. Neural Computation. 9 (8): 17351780, 1997.

[4] C. W Reynolds. *Flocks, Herds, and Schools: A Distributed Behavioural Model, in Computer Graphics.*. 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34, 1987.