Organic NPCs through the use of neural networks

Student name: Oscar Johnson

Student ID number: 16025481

Supervisor name: James Huxtable

Date submitted:

Module: UFCFS4-30-3 Creative Technologies Project

  
  
Summary

This project uses neural networks to drive an NPC agent that behaves realistically in an RPG setting.

Using three neural networks, one for idle behaviour, one for combat behaviour and one final emotionally driven network, this project simulates a realistic bar brawl setting where NPCs can act on and react to in game events.

This project also includes a framework for developing additional behaviours through neural networks and training these networks to behave as expected.

Brief biography

I am a young and aspiring student currently studying Games Technology(Bsc).

I have always been interested in AI and wanted to use this project as a way learn more about the current practices used in the industry and how other methods could be applied to better effect. I chose neural networks as they employ machine learning, a skill I have always been infactuated with and one that many employers are seeking as they belive it is at the forefront of current technology.

How to access the project (not included in word count)

Please put down the URL and/or details of how to access your project, and the ***URL of your final video***. If we need to see any back end / administration interface please provide us with necessary passwords and URLs to access this. Basically, what we cannot access will not be considered for marking. You can change the passwords after you receive your mark.

We also want to be able to see source code, and the best way is to download it from your site or github. Please clearly comment code to show us what is your own and what has been used from frameworks, libraries, OSS or borrowed from elsewhere.

If there is some other method for providing access to your project you will need to provide instructions here. Also if there is anything we need to know about the work that will not be self-explanatory, then also provide brief instructions here.

Introduction 400

Briefly introduce the project in its professional context. Tell us what it was about, what problems it solved, why this is important, interesting or valuable. What were you seeking answers to, how did they arise, why were they worth investigating?

Also list your deliverables / project objectives.

This project set out to solve the issue of developing realistic NPCs that allow for greater player immersion as well as system optimization.

The objectives were as follows:

* AI that act ‘realistically’ or, as the player expects.
* AI that has a low overhead an impact on system performance.
* AI that is easy to develop and can be tweaked to the developer’s needs.

Development of AI systems are often restricted as they are less marketable than other game features such as graphical fidelity and gameplay content, therefore they are often lackluster and overly simplified to decrease their overhead and impact on system performance.

Using neural networks, this project has achieved an AI system that has a minimal impact on performance, is easily integrated into rapid development cycles and delivers an immersive experience for player.

Practice 1500

State the main outcomes of your practice. Insert graphs, screenshots, diagrams, short code snippets if necessary.

Reflection: How has your practice developed *after* the research phase, and in hindsight how successful was this developmental phase? How did your research impact on your practice and vice-versa? How did user/ peer/ tutor feedback impact on it?

Evaluate: How did you deal with problems you encountered along the way? Give us a few specific examples. These could be conceptual, technical, practical, legal (think copyright) or ethical (think user studies). Tell us how you succeeded, or at least how you exhausted every method within your grasp.

## Developing the neural network class

public NeuralNetwork(int[] layout, string n = "Default")

{

\_layers = new int[layout.Length];

for (int i = 0; i < \_layers.Length; i++)

{

\_layers[i] = layout[i];

}

// Initialize the neurons

List<float[]> neurons = new List<float[]>();

for (int i = 0; i < \_layers.Length; i++)

{

neurons.Add(new float[\_layers[i]]);

}

The network class was developed to be modular, easy to modify and have a low overhead.

A constructor was created that takes an array of integers along with a string.

Each element in the array represents a layer with the value representing the depth of the layer, this allows the networks to be fully modifiable in number of inputs, number and depth of each layer and the number of outputs.

The string is used for identification so that the networks can be sorted later.

As an object, this network class can be added to any unity mono behavior allowing for network control in any script.

The activation function tanh was chosen for the network after multiple rounds of testing training effectiveness. The research phase discussed using a linear or sigmoid activation function rather than a step function as it allowed for a greater degree of control over the outputs akin to fuzzy logic.

Upon implementation it was found that after a few rounds of training and mutation the weights became too large, causing the delta between outputs to grow. This made it difficult to train for different behaviors and data sets as small weight adjustments lead to massively differing outputs meaning more rounds of training were required to fix errors.

More research into methods of managing weights such as weight decay to regulate growth (source) and alternative activation methods (Source) was required. This was a short sight of the research phase and could easily have been avoided if this topic was covered with more scrutiny.

A paper studying the performance of different activation functions (Bekir Karlik) found that on average, the tanh activation function gave better accuracy overall than a sigmoid function. As discussed in “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning”, Tanh skews the range of outputs between -1 and 1 which removes the delta between outputs caused by larger weights.

|  |  |
| --- | --- |
| Activation function: | Average fitness after 5 training rounds: |
| Linear | 652 |
| Sigmoid | 899 |
| Tanh | 1355 |

After implementing Tanh, the training results showed less errors caused by differing outputs and helped improve the overall fitness of the networks in fewer training rounds.

Also discussed in the research report was implementing backpropagation for improved training results. For simplicity the network script was setup with a mutation function for evolving weights in-between training sessions, with the plan of removing this and implementing back propagation later.

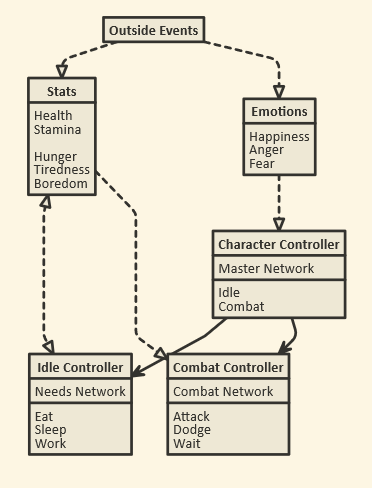
This implementation of mutation allowed for rapid training of test behaviors at a low cost. This sped up the development time as the turn around on prototype builds was short

With feedback from the supervisor of this project, it was found that through mutation the agents were reaching realistic enough behavior that the use of back propagation seemed like a waste of time which could be used to develop additional behaviors.

## Creating behavioral classes

As each behavior required its own network and resulting functions for initializing, storing and running through them, it made sense to create a base class for all agent behavior scripts to inherit from. Not only did this class help streamline the process of creating new behaviors but it also allowed for the polymorphism used in the training script discussed later in this report.

The final project contains three network driven scripts, they can be broken down as follows:

The character controller script acts as a top-level class containing a master network. This network chooses what behaviors to run based off results from the two other networks as well as the agents list of emotions. This integration of the emotion system allows the network to drive the agent’s actions off their emotions much like humans tend too, in an attempt to create more realistic actions. Currently the emotional system contains three emotions: happiness, anger and fear which all play an important role in how the agent makes decisions. The research report details the use of a more complex emotional system of the likes of Robert Plutchiks emotional wheel however, this was cut back as the amount of behaviors currently implemented removed the need for such a complex system. Further development of more behaviors would require more emotions and therefore with more development time a more complex emotional system would be more successful.

Black arrows show flow of control, dotted lines show effect

The needs network drives the agent’s Idle behavior, as chosen by the master network. The network evaluates their current needs (such as hunger, boredom, etc.) and a prioritized list of actions is created.

Initially, the network would only choose one action to complete and would then wait until that action was completed to choose another. As highlighted by the supervisors during the mid-progress demo, this caused issues with expandability as if the agents couldn’t complete this action, say there was nowhere free to eat, they couldn’t complete the task and would stand idle until they could. This was a major hurdle in the development of this project as the system needed to be flexible if it ever were to be implemented into an actual game.

To resolve this issue, instead of evaluating the best result from the network, each output was given a priority based off the value of each node. A simple function was made to check if the most prioritized action was available, If it was it would be added to the agents queue of actions, if not then the next most important action would be evaluated. This queue could be interrupted by manually inserting actions of a higher priority at the front to allow flexibility for other behaviors, much like how system interruptions work on modern computers. This queue also allows for greater optimization as unlike other networks, this network isn’t constantly feeding through inputs if the queue isn’t empty, saving system resources.

The final network controls the agents combat. This network is the simplest due to the need for rapid reactions in a combat scenario, it takes a list of agent’s stats including health and stamina and decides whether they need to attack, dodge or wait for stamina to recover.

* Training the networks
  + Script developed to allow users to select the script to train, how long to train for etc.
  + Best network results are then evaluated and saved, loaded on next entry
  + Ran into major issues training the needs network as expected behavior was more abstract and there were lots of variables to consider. This was a massive downside to the project and cost a big heap of development time.
    - Problem was mitigated with the training script allowing for rapid training sessions and the network visualizer allowing a clear understanding of what the bot is ‘thinking’.
    - Different methods of adding fitness were explored such as on job completion, average of needs, weighted happiness function.
* Developing additional behaviors
  + Event system for crowd behavior
  + Relationship system for interpersonal behavior
* Evaluation user feedback

Discussion of outcomes 900

Reflection: What significance does your completed project have in a wider professional context? What can other professionals draw from your work? What does it improve on? Critically analyse what you have achieved and if necessary suggest different future approaches.

To what extent do the outcomes of your practice satisfy your original objectives, stated in your proposal?

What makes your project *original* in a professional sense; e.g. have you discovered any new methods as you went along? New models that might help others understand processes better? Developed a good protocol for implementing things? Streamlined some processes?

* Project proves neural networks can be used to create functional video game NPCs
* System allows for a streamlined creation process for neural network based behaviors
* Player immersion isn’t quantifiable based off one system, it is the collection of systems working in tandem that allow for player immersion.
* When isolated, the feedback proves that the NPC system created provides an immersive experience.
* Training the networks is still a major issue

Conclusion and recommendations 200

This section should not contain any new information. You should draw conclusions from the research and practical work you completed. Does your work have a longer-term future beyond UWE and if so what might it look like? Think about the impact you could achieve with it. How might it benefit users, professionals, society? Perhaps make some recommendations for further work.

* Future work should focus more heavily on effective ways to train neural networks and effectively communicate what the developer wants the network to achieve.

References (not included in word count)

Bibliography (not included in word count)

Appendixes: (not included in word count)

A Log sheets

B Project timeline

C If necessary, insert a list of interview questions, any larger tables, evidence of design development, longer code snippets or other relevant materials in here. This does not come under the wordcount. Nevertheless only insert useful materials here, please don’t bulk it up. Your report should be able to stand on its own, without relying on appendixes.