Charles Sturt University

Neural Networks on Nintendo Entertainment System

Software Development Project - Life Cycle Objective Milestone (LCOM)

Submitted to:

James Tulip
School of Computing & Mathematics
Charles Sturt University

Prepared By,

Joshua Beemster *(11570593)*
Jasim Schluter *(11539147)*
Loic Nyssen *(11557424)*

Contents

1. Project Vision

The NES-NN (Nintendo Entertainment System - Neural Networks) project is aimed at researching and building upon the work done by Julian Togelius and others in regards to the use of Neuroevolution within Games[1]. In the paper “Neuroevolution in Games: State of the Art and Open Challenges” 7 open challenges were presented for an artificial intelligence (AI) that plays video games. These challenges ranged from the ability to reach record breaking performance to being able to build generalisable neural networks to play any game.

Our vision is to first develop a simple framework that allows us to interact with NES games programmatically and to which we can hook-in several different Neural Network (NN) frameworks. This will yield the ability to test many different types of games with relative ease and to be able to benchmark the performance of many different types of networks against these games. Performance in this context being how long a network takes to be able to solve a particular problem which, for our project, is playing NES games.

From this research we aim to be able to categorise attributes of different networks in relation to how well they progress in playing games. Aspects like how fast a network can learn, how often it becomes stuck on a problem, differences between game play - we want to understand the why and how behind the network and how it applies to this field.

As a stretch goal we would also like to be able to develop the ability to train certain traits and features into any given network. Traits like caution where the network plays the game by avoiding enemies or fearlessly charging into danger.

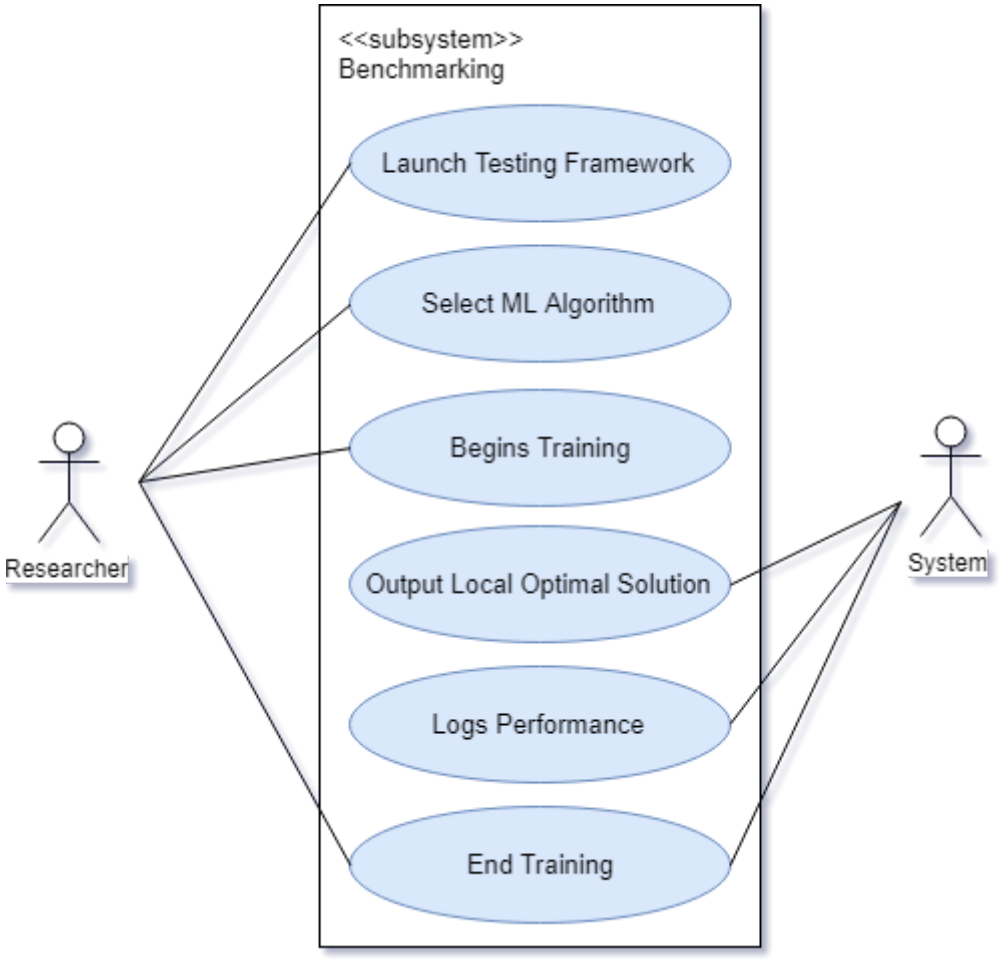
The end result of this project should culminate in an advanced testing framework for Neural Networks to play NES games and the ability to experiment with individual networks and to be able to benchmark these networks against each other. Coming back to the linked paper we hope to be able to achieve or begin to understand the difficulties in achieving the outlined challenges. Especially those relating to “record-beating performance” and “general video game playing”.

[1] "Neuroevolution in Games: State of the Art and Open Challenges - arXiv." 3 Nov. 2015, https://arxiv.org/pdf/1410.7326. Accessed 5 Apr. 2018.

2. Initial Requirement Model

Use Case Models

Benchmarking Neural Networks against their ability to play NES games



Description: Record and compare the results of neural networks in their ability to learn to play NES games via key metrics such as speed, fitness and generality.

Actors: Anyone interested in understanding how different neural networks behave when learning to play NES games

Preconditions:

1. The user has a copy of the precompiled NES testing framework binary
2. The user has acquired the necessary ROMs to launch a compatible NES game
3. The user has generated the required state file for use with the training system
4. The user has access to a Windows 10 PC on which to run the framework

Postconditions: The user can visually compare the results of N neural networks against their ability to play the selected NES game of their choice.

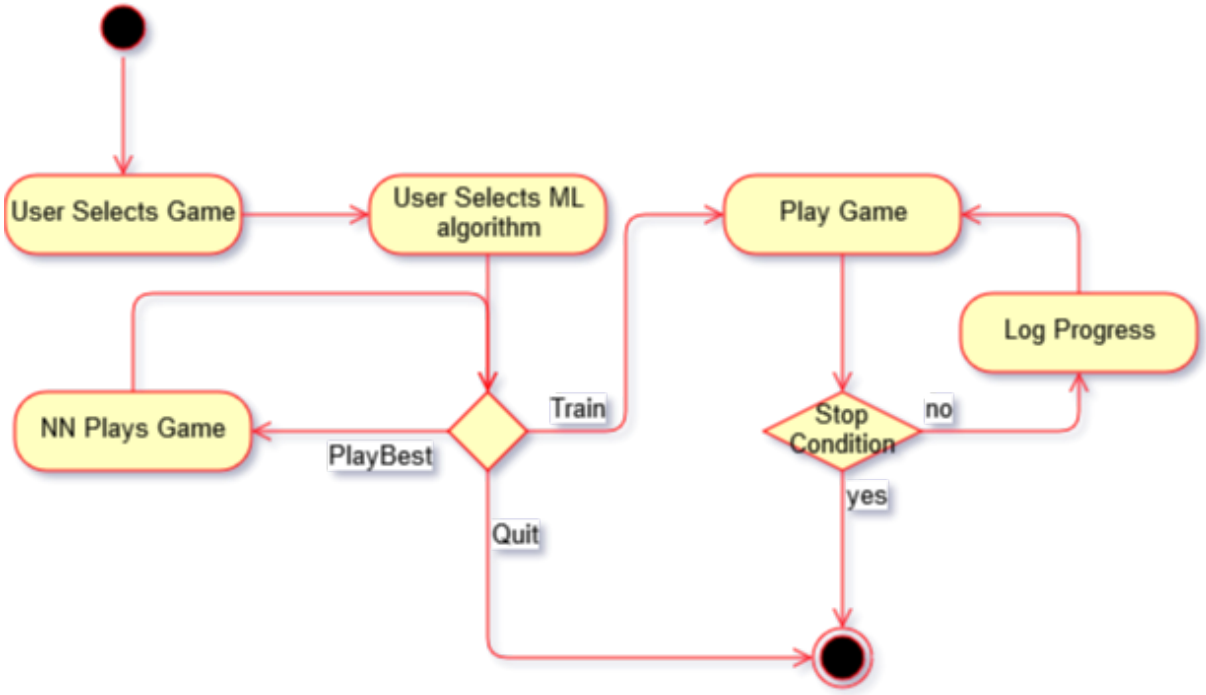
Course of action:

1. The use case begins when a user wants to test different Neural Networks against a selected NES game
2. The user launches the testing framework on their PC and loads the NES state game to train
3. The user selects a machine learning algorithm from the available selection
4. The system begins the training and logs the performance to a CSV file
5. The system, while running, will also output the local optimal solution of the network
6. The user ends the training run once they are satisfied with the resultant network
7. The user uses external tools to graph and analyze the performance of the machine learning algorithm.
8. The use case ends.

Alternative Flows

1. The user select a corrupt state file.
   If at step 2 the user selects an invalid or corrupt state file the system will throw an exception
   a. The use case ends.
2. The user kills the app mid training.
   a. The use case ends.

Activity Diagram for "Benchmarking Neural Networks against their ability to play NES games"



Test Bed System

Name: Train a Neural Network against a defined game
Description: As a researcher I need to be able to train a Neural Network to play NES games so that I can build a non-trivial Neural Network for analysis purposes.

Name: Let a trained Neural Network play a defined game
Description: As a researcher I need to be able to allow a trained Neural Network, built by this tool, to be hooked up to any NES game so that the generality and fitness of the network can be evaluated.

Name: Record training statistics
Description: As a researcher I need to have the statistics on the training process saved so that they can be compared with other networks to allow for an understanding of the differences between neural networks.

Game Playing Subsystem

Name: Emulate the Nintendo Entertainment System
Description: As a Researcher, I would like to be able to emulate the Nintendo Entertainment System, to make integrating with the system a lot easier then physical hardware, and have a lot more control then the hardware provides to pause/play/skip levels and to be able to read RAM directly.

Name: Play NES game ROMS
Description: As a Researcher, I would like to Play NES games, since they are real games with moderate challenges, but not as sophisticated as modern gaming hardware with much more sophisticated control inputs and pixel outputs per frame.

Name: Save state of games
Description: As a Researcher, I want the current state of games to be savable, so that I can manually play through start screens and otherwise setup the game into a desired starting state for training.

Name: Load State of games
Description: As a Researcher, I want to load the state of a game, so that training can start and restart from a desired starting game state easily.

Name: Retrieve live sprite information from game
Description: As a Researcher, I want to retrieve live sprite information out of the game RAM, so that I can optionally train the NN at a level of abstraction above the machine vision level of raw pixels.

Name: Retrieve live game stats
Description: As a Researcher, I want to retrieve live game stats, so that information like score and time can be fed into the NN without having to work through raw pixels to find it.

Name: Programmatically control emulator's game controller
Description: As a Researcher, I want to have programmatic control of the emulators controller input, so that the results of a NN can be mapped into actions and fitness of those actions can be assessed.

NN Training Subsystem

Name: Set NN strategy
Description: As a Researcher, I want to be able to set different Neural Network training strategies, so that I can reuse the testbed for many experiments on Neural Network training.

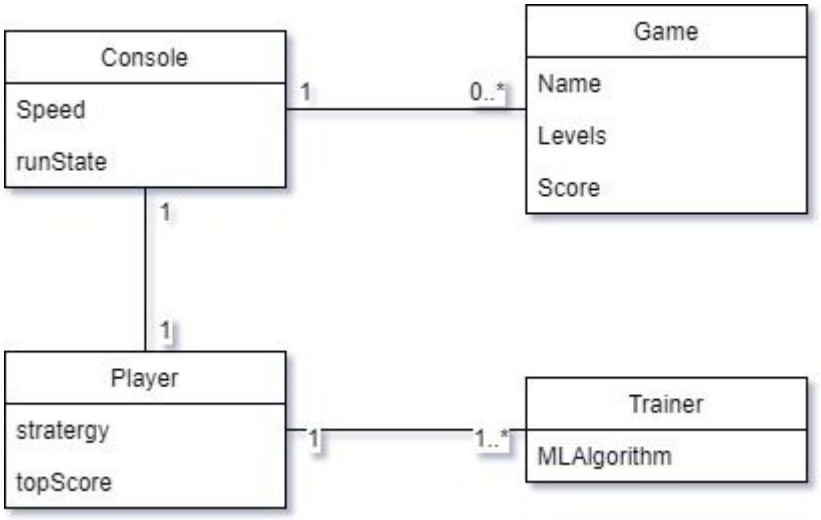
Name: Run experiment
Description: As a Researcher, I want to be able to run Neural Network training, so that I can research the effect of the strategy, and different inputs and fitness calculations on rate of fitness increase and otherwise experiment with Neural Networks.

Name: Calculate fitness
Description: As a Researcher, I want to be able to calculate the fitness of the Neural Network in a configurable way, so that I can experiment with NN that are more cautious, or daring, depending on what they are rewarded for.

Name: Train NN
Description: As a Researcher, I want to be able to train a Neural Network, so that we have an AI that can play NES games in which we can experiment.

Name: Persist NN for reuse
Description: As a Researcher, I want to be able to persist a trained Neural Network, so that the results of a training session can be viewed, studied and compared to other strategies.

Domain Model



Non-Functional Requirements

NFR	Justification	Priority
Extensibility	The system needs to be able to be extended to allow for the adding of other NN training strategies.	1
Configurability	The system has a strong need to be configurable since researching will involve tweaking settings to find optimal solutions.	2
Reliability	The system needs to reproduce results. There is no need to support any failover scenarios. Ability to continue a training run after abrupt termination of the application is desirable.	3
Performance	The system should leverage parallel training and multi-core systems if possible. Splitting out the code for training from the code for replaying to allow faster training will be good.	4
Usability	This research project should have a simple UI and be simple to start test runs. The usability of the software is not a major factor of the project, however the results and documentation should be understandable and reproducible.	5
Supportability	The supportability of the software is not a high priority, however open sourcing the code base and having frequent small commits is desirable for aiding others in understanding the code.	6

3. Proposed Architecture

Our proposed architecture consists of the following System and subsystems: Test Bed, Training Strategy, and Emulator.

This allows us to configure and see the results of training the neural network interactively. The Test Bed is a UI application and the Game Emulator is optionally viewable.

The Training Strategy is a plugin that allows for iterating through different strategies. This makes
the system extensible, and fit for using as a research test bed.

Another way of running the system is this:

The trainer is a console application, and the Game Emulator is not viewable. This is designed to allow for the fastest training possible. Increasing the performance of training is important since the system needs to run this 1000s of times.

Game Emulator is configurable to play different games
The input of the sprites or Raw pixels along with high level stats from the games needs to be configurable so that the system can play many different games. Each game stores these at different memory addresses, so mapping from memory is critical.

Experiments are configurable to receive high-level information of the game (sprites, time) or low level (pixels). Experimenting at different levels of abstraction require being able to read raw pixels, sprites, sprites with extra information on the type of sprite (good, bad, etc).

Fitness calculations are configurable. The NN tries to optimise for the fitness equation, so if we weigh points above speed, we expect to create NNs with different “personalities”. Having those equations be configurable allows us to run these types of experiments.

4. Technical Competency Demonstrator

https://github.com/NES-NN/NES-NN-Testbed/tree/TechnicalCompetencyDemonstrator

The Demonstrator uses SharpNEAT[2], a Neuroevolution of augmenting topologies C# framework, along with the Xyene/Emulator.NES[3], a C# emulator for Nintendo Entertainment System (NES) hardware, to play Super Mario Bros.

As a team we have successfully used machine learning to train a neural network to play the first level of Super Mario Bros with some intelligence.

Building and running the application (from source)

1. Download and install the Visual Studio 2017 IDE for Windows
2. Git clone the following project and branch to your local system:
    a. https://github.com/NES-NN/NES-NN-Testbed/tree/TechnicalCompetencyDemonstrator
3. Launch the `dotNES.sln` file which will load up the project
4. Select the “Start” button from within the IDE
5. Manually copy the “emu.bin” state file from the Interact2 TCD Bundle into the “...\Emulator.NES\dotNES\bin\Debug” directory
    a. **Note**: This state file path and name is currently hard-coded to the same directory as the running binary
    b. This file contains the Super Mario Bros ROM saved at the start of Level 1
6. Select “Start Training” from the “SMBNeat” UI

Running the application (pre-packaged)

1. Download the following ZIP file which includes all binaries, configs and state files required for testing
    a. https://interact2.csu.edu.au/courses/1/S-ITC303_201830_B_D/groups/_36295_1//_534193_1/TCD%20Bundle.zip
2. Double click the `dotNES.exe` file from your Windows desktop
3. Select “Start Training” from the “SMBNeat” UI

[2] "GitHub - colgreen/sharpneat: SharpNEAT - Evolution of Neural ...." https://github.com/colgreen/sharpneat. Accessed 7 Apr. 2018.
[3] "GitHub - Xyene/Emulator.NES: Nintendo Entertainment System ...." https://github.com/Xyene/Emulator.NES. Accessed 7 Apr. 2018.

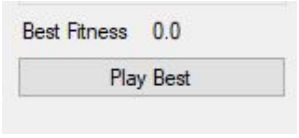
Interpreting training results

Once training commences you will start to see the training output as denoted by the following log output lines:

...
[Generation 0 Genome 38 Display 0] Starting Evaluation
[Generation 0 Genome 38 Display 0] Finished Evaluation - fitness : 0.103359173126615
[Generation 0 Genome 39 Display 0] Starting Evaluation
[Generation 0 Genome 39 Display 0] Finished Evaluation - fitness : 0.103359173126615
…

You will also see the statistics and inputs panels begin to represent the internal game state of Super Mario Bros.

Once you have completed training a generation the system will output the “Best” fitness network. This will allow you to then launch the best network that has been generated so far via the “Play Best” button.



To halt training you will need to close the application window.

5. Risk List

Risk	Mitigation	Iteration*
Inability to assemble a Neural Network training system/framework written in C# with an emulator written in C# without major changes to one or the other.	Prioritised the development and investigation of such a system as this is the core blocker to further development; this has formed our TCD.	1
Inability to read game stats used as inputs for the NN from games RAM.	Developed system for extracting all game statistics from RAM.	1
Inability to create a performant system that could train a Neural Network using our local systems.	Investigating several avenues of attack including parallelism and the removal of non-essential UI components for training cycles.	1
Cannot compare results from different neural networks	Need to ensure that the output from all neural networks follows a standard to allow for simple comparison.	2
Difficulty in creating a pluggable interface for many neural networks within one codebase	We are building a general interface for passing training input from emulator to network and from network to controllers.	2 & 3

**Work has been assigned to iterations in priority order. This is in keeping with Unified Process of putting risky work first.*

6. Master Test Plan

As this is a research project, we do not do acceptance tests against a customer provided list of features. Instead we perform the following types of tests:

Smoke Tests:

Smoke tests are done manually by the developer on their local machine, they check that the NN appears to be learning and that the logs show expected fitness levels.

Regression Tests:

With a small team, we don't assign anyone specific testing duty, instead we expect that each developer regression test the system after each feature is added.

Unit Tests:

We discuss where unit tests would be useful in our retrospective meetings, and if decided add them to that area of the project.

Benchmarking:

An existing implementation of NEAT NN playing Super Mario is used as our benchmark for what to expect the evolution of our NEAT NN to look like in our Testbed. This allows us a way to prove that our Testbed works as expected before we try other strategies.

We also compare rates of learning against rates published in related papers, so that we can see if it is functioning as expected.

Logging:

Our Testbed records stats on each training run, so issues with our implementation can be detected. This will also be used in research results.

Architecting for reproducibility:

We are investigating making test runs reproducible for testing purposes.

Procedure for testing:

Developers are responsible for testing their own code. When issues are found in code, a message on Discord is sent for all the developers to know the issue. This is independent of if the issue is in the developers code, or someone else's.

These issues are triaged on the GitHub issue tracker, and assigned to developers during the sprint planning.

Test results from experiments are shared with the team over discord, so that questions can be raised on what the results mean.

7. Initial Project Plan

Iteration 1:

Build Testbed and running NEAT NN training against the “Super Mario Bros” NES game to validate testbed core features of the Game Playing and NN training subsystems working together.

Iteration 2:

Split out training code from UI to speed up training to improve performance.
Investigate Current state of research in this area and open questions.
Document vision, architecture and project plan (LCOM1).

Iteration 3:

Add HyperNEAT training strategy as a way to build out the test bed configurability.
Research getting NN to complete Mario Game. Implement resuming training sessions to increase the projects reliability.

Iteration 4:

Research getting NN to play mario with Raw data only

Iteration 5:

Run NN against PacMan or other game
Research getting a single NN to play both games

Iteration 6:

Research NN “personality” traits for gaming AI.

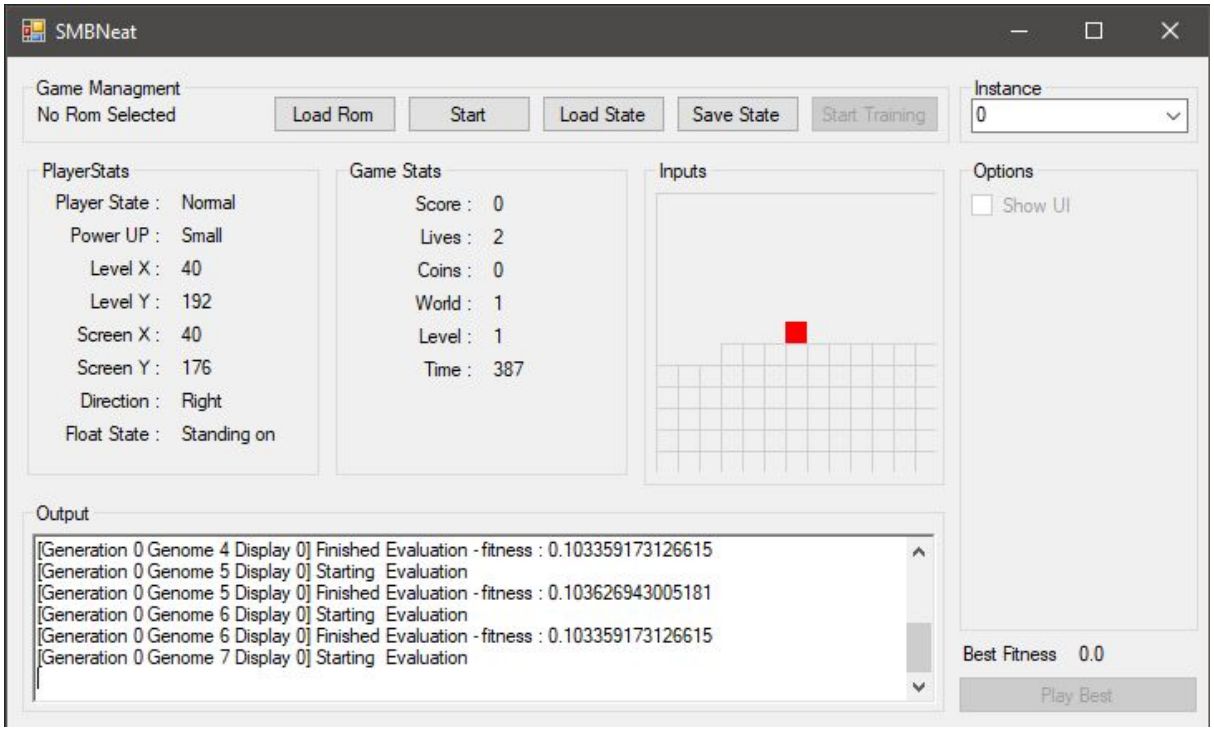
8. Inception Phase Project Status Assessment

The project in its current form has managed to achieve our first major milestone which is the creation of an initial Testbed and proof-of concept. This represents the creation of a framework than can map the inputs of a NES game into a Neural Network and which can then evaluate these inputs to provide outputs that relate to button pushes. In short we can pass game data to a Neural Network which represents an artificial game "player".

This proof-of-concept work has let us resolve several of the early risks in this project around our potential inability to extract useable inputs from a NES game and subsequently being able to map these into a Neural Network system. With these risks being met we now have a foundation to build upon.

As part of this initial development we have created two systems of training a NEAT Neural Network. The first being a UI based system that forms the basis for our Technical Competency Demonstrator (TCD). This system illustrates the specific abilities of the system to:

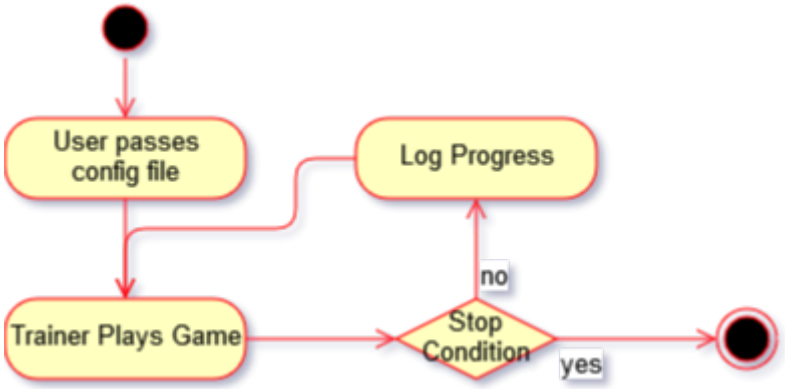
1. Map the games memory into a useable format for training, along with a visualiser of that data for the Researcher to view.
2. Illustrates the ability of the Testbed to load and reload specific game state which is imperative to training cycles.
3. Illustrates and shows the progress of the network as it learns to play the game.
4. Replay a local optimal NN against the game to demonstrate the NNs abilities.



Note: In the testbed the red square represents Mario, and Green squares are enemies.

Once we had successfully started using this format for training the system we developed a more performant CLI based system. The UI, while useful, does not provide the best performance for what is a very heavy compute process. What this system provides is the ability to train a network much faster than is possible with additional UI overhead and with the ability to use much higher levels of parallelism.

The CLI based system works like this:



Fast Trainer Activity Diagram

What the above deliverables have gotten us is the ability to prove that, conceptually, the system works. However there are several issues present in both implementations that require work in upcoming iterations. There are several performance and usability bugs present that need to be resolved still before we can progress with creating a truly pluggable system for multiple algorithms to be used within this system.

The UI implementation has several “features” that currently do not work as expected. We cannot yet achieve parallelism with this system without crashing the training process. The logic behind several of the interaction buttons of the UI are either non-functional or illogical in their function. For the moment we have simply disabled these buttons with the aim to refactor and bring them back in future iterations of the project.

The CLI implementation handles the parallelism much better but seems to struggle to actually progress the neural network. Even with many generations it does not seem to get better at playing Super Mario Bros which points to an underlying flaw in the implementation of either the fitness calculator or the evaluation loop which requires more investigation.