Charles Sturt University

Neural Networks on Nintendo Entertainment System

Software Development Project - Life Cycle Architecture Milestone (LCAM)

Submitted to:

James Tulip
School of Computing & Mathematics
Charles Sturt University

Prepared By,

Joshua Beemster *(11570593)*
Jasim Schluter *(11539147, git:ACoderLife)*
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 address a common problem in NN training, that of training prematurely converging on a local optimal solution, not the global optimal solution.

As seen in our preliminary research here:

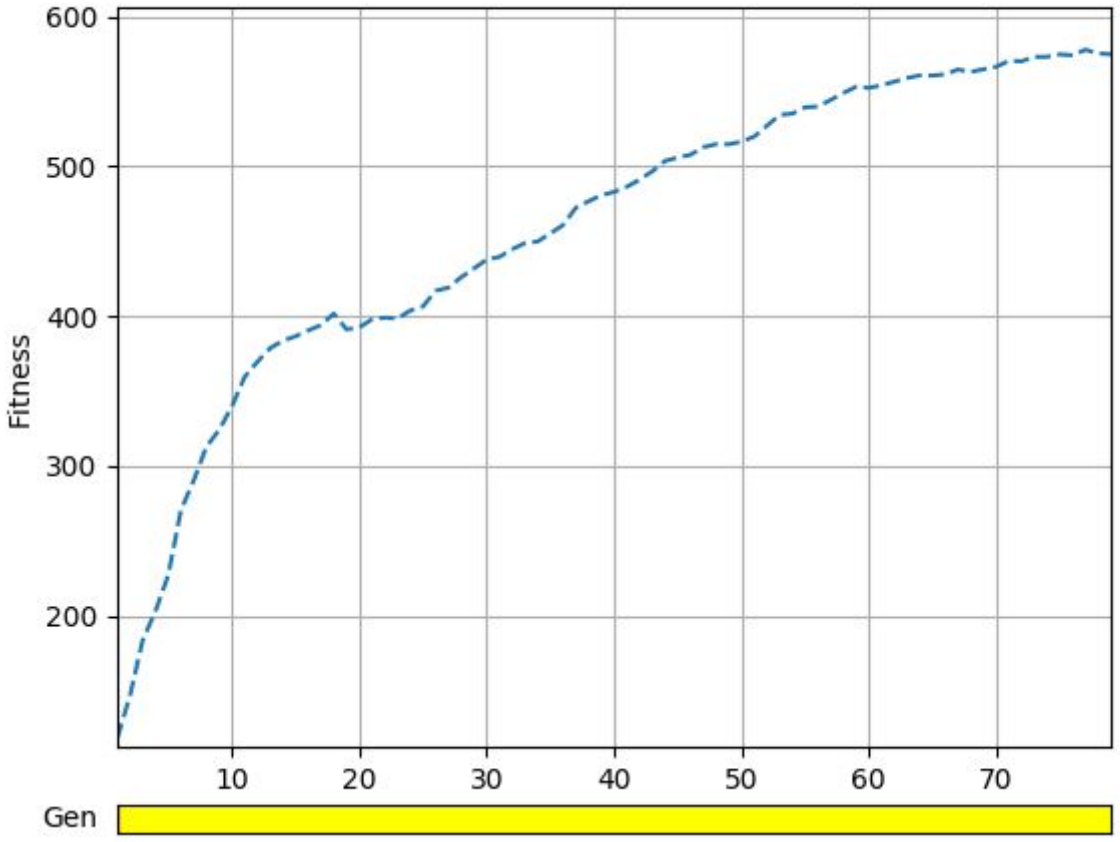


Figure of NN converging on a fitness of approx. 585. This is lower than the fitness which would indicate Mario has completed the level.

There are many reasons why NNs converge at local optimal solutions, and even more hypothesis on how to overcome them. While we are still formulating our solution, we intend to trial a strategy analogous to doing drills in sports. In sports a player will train on one specific skill, mastering it, and

[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.

moving on to focus on another skill, this is called “doing drills”. We hypothesize that if we detect areas that the NN is weak in (perhaps indicated by slower growth in the fitness), and have the NN train on those areas exclusively, we can improve the NNs overall fitness, and move it closer to the globally optimal solution.

While this research question raises other risks, for example, how well will the NN retain lessons learn in one drill, once it moves on to another? We believe that analysing this problem space, the research already done in it, and the practical skills gained by attempting to solve it, will give us real skills we can take into professional environments in the future.

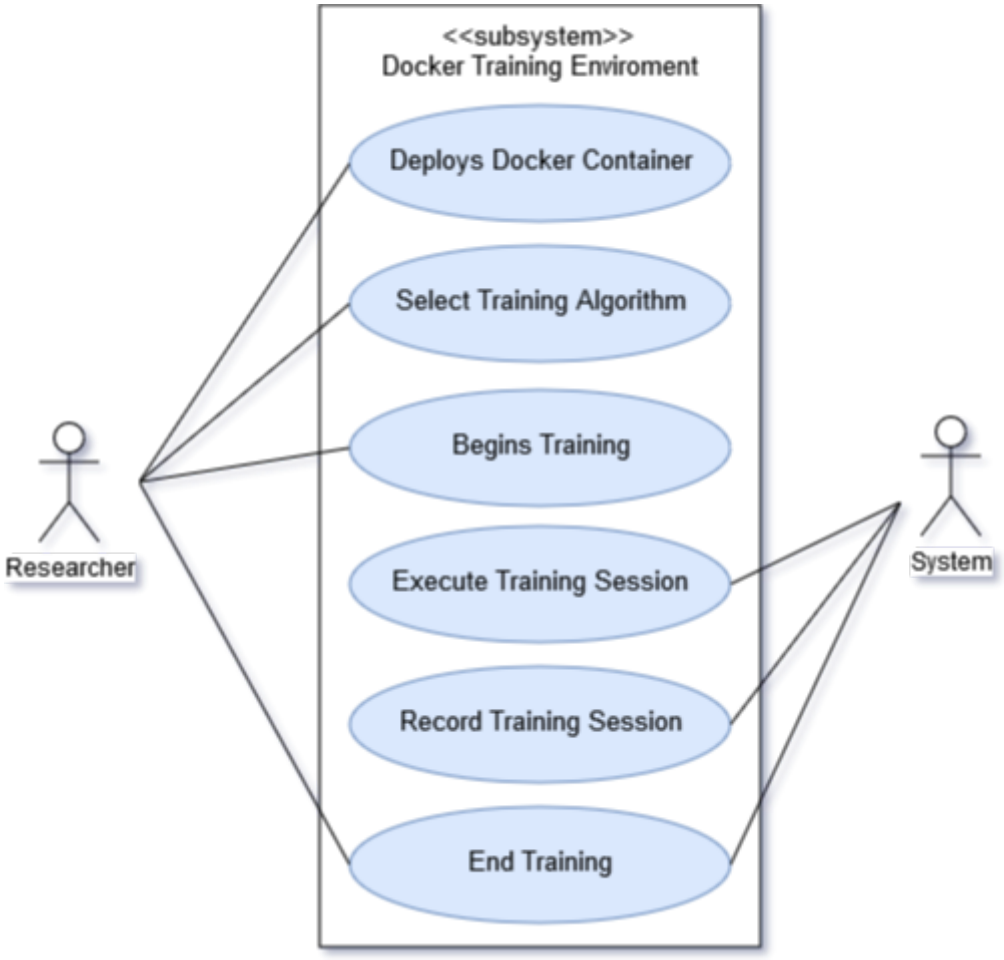
Our project is leveraging mainstream AI training tools, such as OpenAI’s training gyms. OpenAI’s gyms provide a extensible environment to train AI. We have uncovered that extensibility is a core requirement for iterating over many NN training experiments. While it is possible to build a end-to-end training solution, focusing on the experiments, and standardising on the environment we run the tests in, will allow us to focus on our research, while still working with industry recognised tools.

Research is a major component of this project, we have included a link to many of the research papers that we have found interesting and informative in formulating our project vision.
https://github.com/NES-NN/team-management/tree/master/research/References

2. Initial Requirement Model

Use Case Models

Benchmarking Neural Networks against their ability to play Super Mario Bros.



Description:

Record and compare the results of neural networks in their ability to learn to play Super Mario Bros. via key metrics such as speed, fitness and generality.

Actors:

NES-NN Research Team, testing or verifying potential solution to a research question.

Preconditions:

The user has access to a *nix machine and has deployed the docker container containing the testing framework.

Postconditions:

The user can visually compare the results of N neural networks against their ability to play Super Mario Bros.

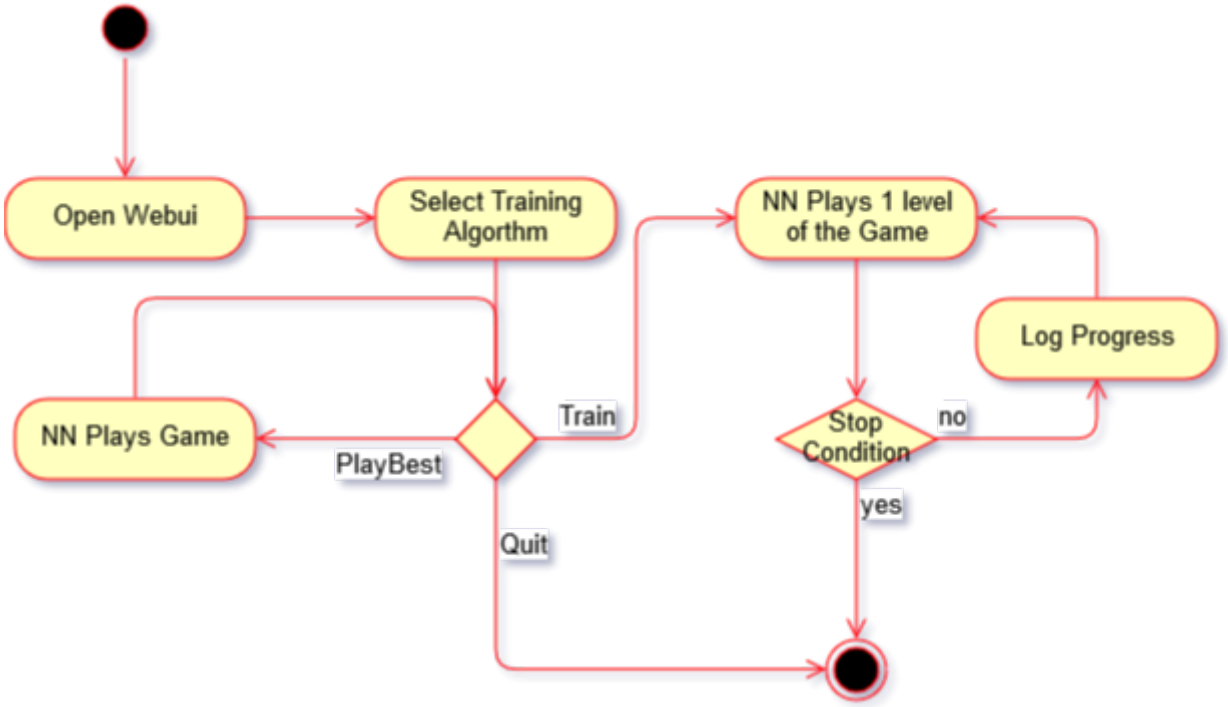
Course of action:

1. The use case begins when a researcher wants to test different Neural Network algorithms playing Super Mario Bros.
2. The researcher accesses the training WebUI.
3. The researcher selects a training algorithm and inputs training parameters.
4. The researcher begins the training session.
5. The system performs the requested training session.
6. The system records the training session.
7. The system completes the requested training session.
8. The use case ends.

Alternative Flows

1. The user aborts the training session before it is complete.
    a. The use case ends.

Activity Diagram for "Benchmarking Neural Networks against their ability to play Super Mario Bros."



Training Subsystem - GYM toolkit

Name: Train a Neural Network to play Super Mario Bros.
Description: As a researcher I need to be able to train a Neural Network to play Super Mario Bros so that I can build a non-trivial Neural Network for analysis purposes.

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.

Name: Test a potential solution to the research question
Description: As a researcher I need to be able to test a proposed solution the the research question and compare the results to a baseline result.

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: 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.

WebUI 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: Evaluate fitness
Description: As a Researcher, I want to be able to Evaluate 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.

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. It should be able to leverage the advantages of a GPU to speed up training.	4
Usability	This research project should have a simple WebUI 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. Final 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 consists of a Docker container with multiple services available within it:

- A UI based on Wooey to launch training sessions in a “headless” mode
- A VNC server to view the live progress of the training
- The ability to create a direct shell session with the container

This lets us deploy and use the testbed from any system exposed to the internet and removes the need for direct shell or CLI access - however we have still retained the ability to create a shell session and if need be get access to a display so that we can directly visualise logs and emulators as the training progresses.

The Training Strategy itself is based around individual scripts that allow us to run different training strategies within a common OpenAI Gym and with a common logging system in the form of VINE. This makes the system extensible, and fit for using as a research test bed.

Each script is designed to run a particular “gym” which is a common abstraction created by the OpenAI team which supplies information about the game or system being trained and allows you to programmatically interface with said system and provide inputs to the system. In our case we have two scripts currently available:

1. A “SuperMarioBros” NEAT training
2. A “SuperMarioBros” Random training

Both use the same gym and output the same logging system and so can be directly compared. We also have the option of swapping out the underlying “gym” within these systems to train different games / systems as needed.

The scripts are also all available in a headless mode of operation which ensures that we can run large amounts of training performantly via the use of virtual display buffers - in essence we can direct the system to direct its display to a non-visible device which therefore does not incur any rendering cost.

4. Executable Architecture

https://github.com/NES-NN/OpenAI-Testbed

The executable architecture consists of a Makefile which builds and runs a single Docker container and which contains all required code and systems to run and analyse the output of training our neural networks.

Under the hood we are using:

1. A fork of this OpenAI Gym for Super Mario Bros: https://github.com/ppaquette/gym-super-mario
2. A deployment of Wooey which is a web interface for Python Scripts: https://github.com/wooey/Wooey
3. A VNC server for visualising the actual emulators during training and for visualizing the and investigating the output of the training via VINE logging: https://eng.uber.com/vine/

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 testbed

NOTE: Prerequisites for this include having a unix based system and having the Docker engine installed on your host system. This has been tested on both Ubuntu and macOS.

Depending on the speed of your network this could take between 5-30 minutes to complete.

1. Download and install Docker from https://docs.docker.com/install/
2. Git clone the following project and branch to your local system:
    a. https://github.com/NES-NN/OpenAI-Testbed
3. From the root of the project run `make`. This will build and launch the docker container on your system - you will need to have ports 5900 and 8000 available for the container to bind on.
4. Open up `localhost:8000` in your browser of choice
5. Select the `Train` script from the Wooey interface
6. Configure your options; for a quick test set `Generations` to 1 and `Numcores` to the maximum available on your host system
7. Select the “Submit” button from the bottom right corner to begin training

Interpreting training results

Once training completes you will see the training output as denoted by the following log output lines:

****** Running generation 0 ******

Population's average fitness: 79.25000 stdev: 54.58193
Best fitness: 143.00000 - size: (0, 1248) - species 1 - id 3
Species length: 1 totaling 4 individuals
Species ID : [1]
Species size : [4]
Species age : [0]
Species no improv: {1: 0}
Average adjusted fitness: 19.812
Spawn amounts: [4]
Species fitness : [19.8125]
Generation time: 5.171 sec
…

****** Training output ******

Number of evaluations: 40
Saving VINE statistics into: /opt/train/NEAT/snapshots
Created snapshot:snapshot_offspring_0000.dat
Created parent snapshot:snapshot_parent_0001.dat
Created snapshot:snapshot_offspring_0001.dat
Created parent snapshot:snapshot_parent_0002.dat
Created snapshot:snapshot_offspring_0002.dat
Created parent snapshot:snapshot_parent_0003.dat
Created snapshot:snapshot_offspring_0003.dat
Created parent snapshot:snapshot_parent_0004.dat
Created snapshot:snapshot_offspring_0004.dat
Created parent snapshot:snapshot_parent_0005.dat
Created snapshot:snapshot_offspring_0005.dat
Created parent snapshot:snapshot_parent_0006.dat
Created snapshot:snapshot_offspring_0006.dat
Created parent snapshot:snapshot_parent_0007.dat
Created snapshot:snapshot_offspring_0007.dat
Created parent snapshot:snapshot_parent_0008.dat
Created snapshot:snapshot_offspring_0008.dat
Created parent snapshot:snapshot_parent_0009.dat
Created snapshot:snapshot_offspring_0009.dat
Created parent snapshot:snapshot_parent_0010.dat
Saving best genome into: /opt/train/NEAT/neat_network.pkl

Along with this there will be several other files saved:

1. The actual Neural Network is saved locally in a file called `/opt/train/NEAT/neat_network` so that the next training session can resume where this one left off - meaning that training can be paused and resumed in many sessions

2. After each run we also save the best performing genome from the network in a file called `/opt/train/NEAT/neat_network.pkl` which will be used when the script is supplied with the `--play-best` option
3. For analysis purposes we dump out each generation in a VINE compatible snapshot format - this directory contains the result of every iteration of every generation for analysis and exploration purposes

To halt training prematurely simply run `make down` from the CLI to destroy the running container.

NOTE: If you opt to save data to another folder other than `/opt/train/` this data is ephemerally stored within the container only and will be lost on container destruction.

5. Risk List

Risk	Mitigation	Iteration(s)	Notes
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	Solved using RAM maps.
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	Solved as we can run using CLI
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	

Implementation of Accord.NET framework could be very difficult and much lower level than SharpNEAT	Investigating and attempting implementation of the framework	4,5	Solved by moving to openai gym
That the language we choose is not conducive to reusing the work already done in the AI NN area.	After comparing what is available to us in C# libraries to that of other languages, we have decided that python is the best language to develop out research on.	6	
That our work in building a Test Bed is not required since there are better test beds already built for what we want to test.	We have found that a general purpose environment for training AI against games exists in the work of OpenAI gyms. We have switched to that.	6	
Our current method of reading sprite locations from ram might be too simplistic to effectively train a network.	We may have to implement deep learning strategy to interpret the inputs before passing them to the NEAT network.		

**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.

Continuous Integration Testing:

With each push to the master branch we will have the environment rebuilt to validate to will work for everyone.

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.

Testing plan
- User acceptance tests covering the following:
  - All the command line params
- Validate training strategies:
  - That random player doesn't learn
  - That NEAT player does learn
  - That NEAT player config params work (a lot!)

Validate hypothesis:
- That there are local optimal conditions for neat.

Open questions:
- How do we plan to tackle reproducing the results of Ubers local optimal issue?

Manual test cases we ran:
- Testing if the training is working in parallel (freezes etc.)
- That the test environment is stable
- That emulator instances are shutting down correctly.
- That logs are produced and that they are correct.

Testing NFRs:
- Performance: impact of VINE vs. no VINE logging

7. Evidence of Testing

Testing Strategy 1: Command line args.

Test Case	Expected result	Actual result	Pass / Fail
--config-file Test Case 1: loading a valid config	should load the config file for the NEAT environment.	https://github.com/NES-NN/team-management/blob/master/testing/configFile/loadFromConfig.txt	Pass
Test Case 2: trying to load a config file that doesn't exist	incorrect input will show an error message.	https://github.com/NES-NN/team-management/blob/master/testing/configFile/wrongConfig.txt	Pass
--episodes Test Case 1: episodes set to 3	should run the same genome against the game x times. Picks the mean fitness result as the genomes fitness result.	https://github.com/NES-NN/team-management/blob/master/testing/EpisodesTest/episodesTest.txt	Pass
Test Case 2: no episodes set	Should default to one episode	https://github.com/NES-NN/team-management/blob/master/testing/EpisodesTest/noEpisodsSet.txt	Pass
--generations : The number of evolutionary runs of the species. Each generation will run all the species set in the gym_config pop_size setting. Test case 1: Generations count 1	Should run the training once	https://github.com/NES-NN/team-management/blob/master/testing/generations/generations1.txt	Pass

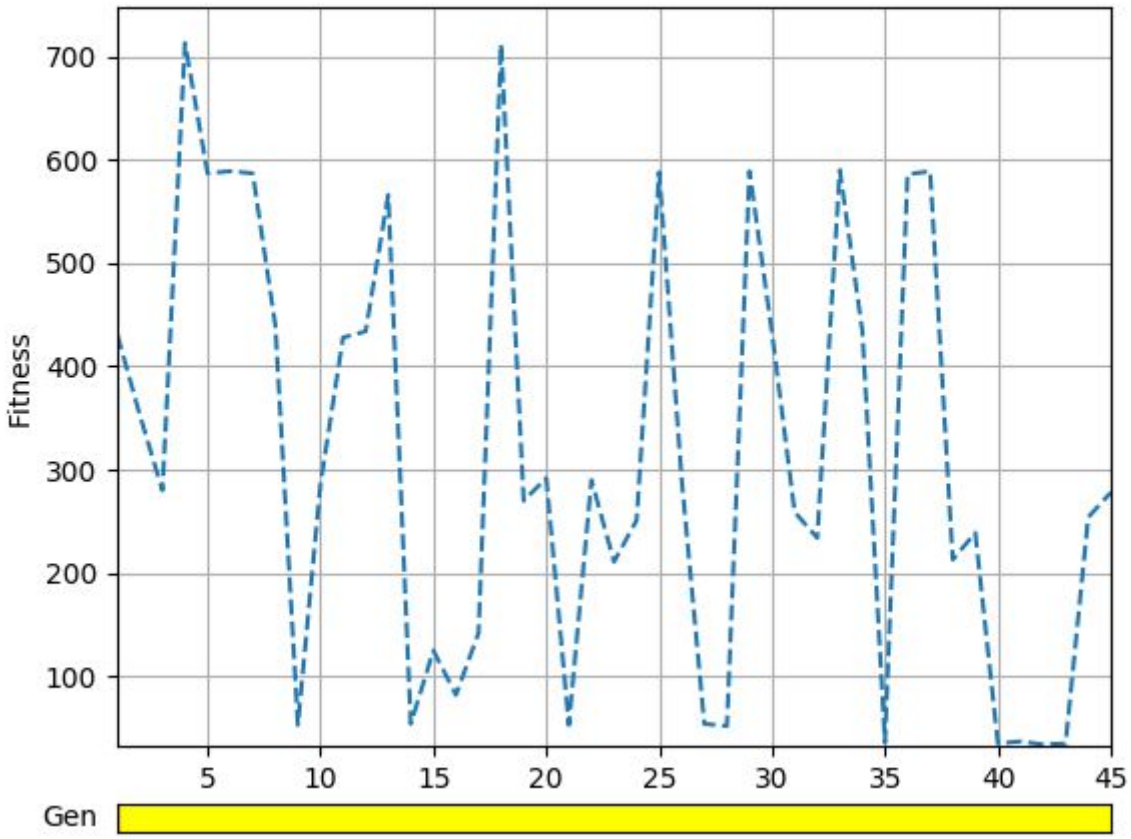
Test case 2: Generations count 2	Should run the training twice	https://github.com/NES-NN/team-management/blob/master/testing/generations/generations2.txt	Pass
--save-file : This saves the simulation at the end of the run of generations. Test case 1: save-file not set	should save the default file named: neat_network	https://github.com/NES-NN/team-management/blob/master/testing/CreatesSaveFile/defaultSnapshotTest.txt	Pass
Test case 2: save-file set to customSnapshot	a custom file name should create a snapshot with that name	https://github.com/NES-NN/team-management/blob/master/testing/CreatesSaveFile/customSnapshotTest.txt	Pass
--play-best : This param will put the Trainer in play best mode, which will load the file specified by the –save-file param, or neat_network if not specified. Test case 1: play-best, without save-file specified, with no prior best saved:	Expect error message	https://github.com/NES-NN/team-management/blob/master/testing/PlayBest/playBestWithNoPriorBestSaved	Pass
Test case 2: play-best, without save-file specified, with prior best saved:	Should pay last best file	https://github.com/NES-NN/team-management/blob/master/testing/PlayBest/playBestWithDefaultPrior	Pass

Test case 3: play-best, with save-file specified incorrectly:	Expect error message	https://github.com/NES-NN/team-management/blob/master/testing/PlayBest/playBestWithWrongSaveFile	Pass
Test case 4: play-best, with save-file correctly specified:	Should load that best file, not the default one	https://github.com/NES-NN/team-management/blob/master/testing/PlayBest/playBestWithRightSaveFile	Pass
--num-cores : This runs training in parallel batches of the size set as num-cores size. It doesn't check the number of cores on the system. Test case 1: not set should run one instance at a time.	Expect to see one emulator running at a time	https://github.com/NES-NN/team-management/blob/master/testing/numCores/default1.png	Pass
--num-cores set to 2	Expect to see two emulators running at the same time	https://github.com/NES-NN/team-management/blob/master/testing/numCores/2.png	Pass
--num-cores set to 4	Expect to see four emulators running at the same time	https://github.com/NES-NN/team-management/blob/master/testing/numCores/4.png	Pass
--parallel-logging-file : The location that the temporary logging file for VINE is stored	log file should be created at /opt/train/NEAT/parallel_info.ndjson	https://github.com/NES-NN/team-management/blob/master/testing/ParallelLoggingFile/parallel_info.ndjson	Pass

--snapshots-dir : Location where VINE log files will be stored	folders for each generation should be created, with a offspring and a parent log file inside, except the gen 0 which has no parent and the last gen which will not have offspring. This should be located at: /opt/train/NEAT/snapshots	https://github.com/NES-NN/team-management/tree/master/testing/snapshotsDir/snapshots	Pass
--v : Verbose logging, off by default, shows episodes and genome fitness if on. Test case 1: train with no setting	Output should not have episode or genome fitness info	https://github.com/NES-NN/team-management/blob/master/testing/verbose/verboseOff.txt	Pass
Test case 2: train with setting	Output should have episode or genome fitness info	https://github.com/NES-NN/team-management/blob/master/testing/verbose/verboseOn.txt	Pass

Testing Strategy 2: Validate training strategies

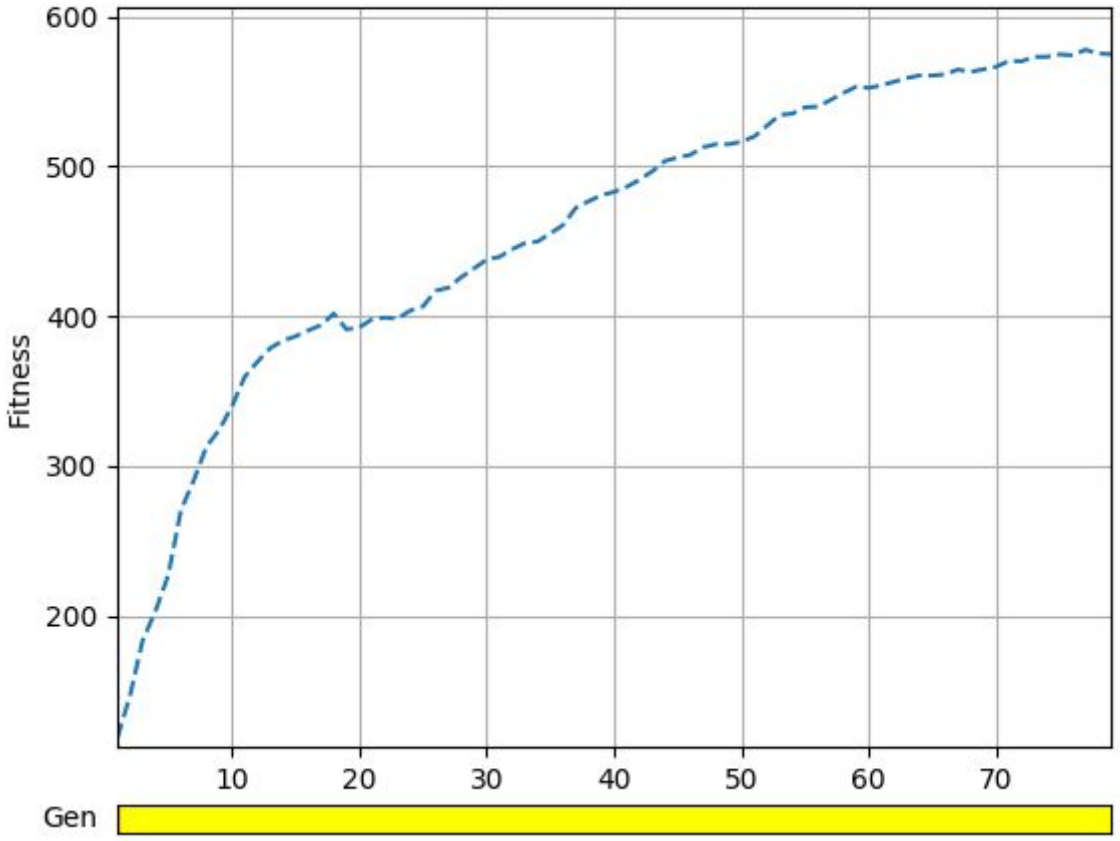
Test Case 1: That random player doesn't learn:
After running the Random player for 45 generations and plotting it's fitness in VINE we get the following graph:



Notice that the Random Player is not increasing fitness over the generations. This is because the Random Player doesn't learn.
(See: https://github.com/NES-NN/team-management/tree/master/testing/RandomTrainingResults for test data)

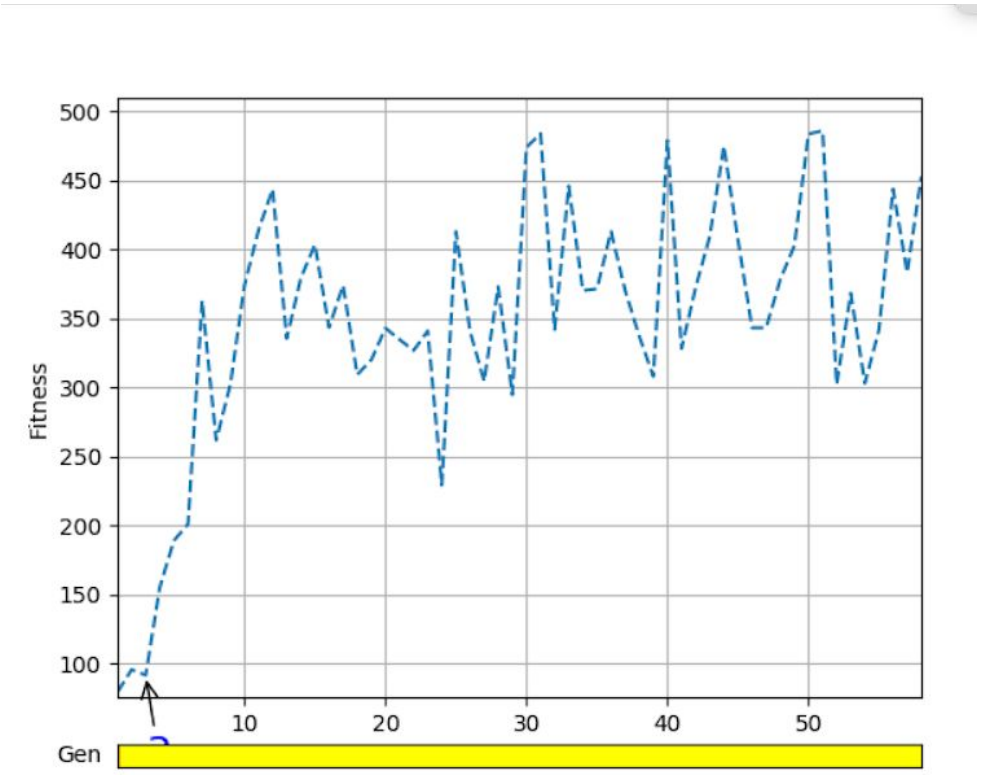
Test Case 2: That NEAT player does learn:

After running the NEAT trainer for 79 runs and visualizing the results in VINE we get the following graph:



As you can see the NEAT player improved almost each generation of training. This proves that NEAT learns and improves.

Test Case 3: That NEAT is changing behaviour based on the gym_config. We usually run NEAT on 20..150 genomes per species. To test the gym_config, here is the results of setting the pop_size to a very low 5:



Notice the increased volatility when there are so few members of the species. Small changes to their individual fitness have bigger weight on the overall.

Testing Strategy 3: Large scale multi-core testing

In order to test if our training environment scales, we have tested NEAT with +50 generations of a 150 genomes each running 5 times per generation, taking the average fitness of those 5 runs as the fitness of the genome for that generation. This was run over 8 parallel instances for over 12 hours, and created a player that can make it through almost 70% of the first level.

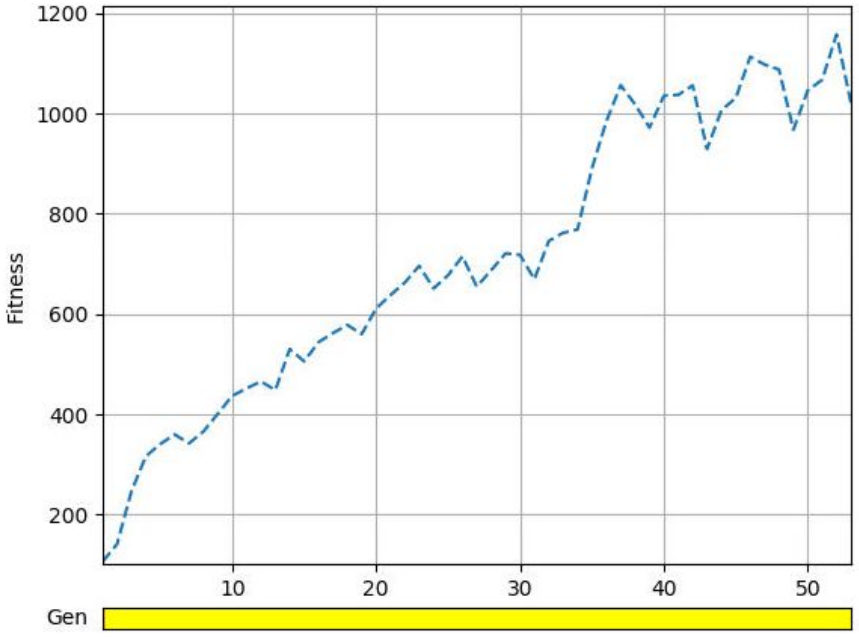


Fig: Mario trapped after completing nearly 70% of level one.

Testing Strategy 4: Continuous Integration testing

To ensure that changes to our environment do not break anything we test the ability to build and launch our container via Travis CI everytime a change is merged.

This ensures that our container, or testbed, can be built on a fresh environment and removes the “it works for me but not for you” issue seen all too often in collaborative programming.

The builds can be seen here: https://travis-ci.org/NES-NN/OpenAI-Testbed

The current build status is reflected on the project Github page here:
https://github.com/NES-NN/OpenAI-Testbed/blob/master/README.md

8. Initial Project Plan

Inception phase

Phase 1 : This iteration we plan to get an existing NES NN working, then to reproduce it as a C# implementation.This will put us in a position to rapidly iterate on Neural Network designs in future iterations.

Phase 2 : This iteration we are going to step back and review the current literature on Neuro Evolution and Game AI. We are going to formalise our vision of this project, along with documented major architectural components and the decisions behind them. Looking into the future we will come up with some high level goals for future iterations. We will document the whole thing and submit it as LCOM1.

Elaboration phase

Phase 1 : Now that the Technical Competency Demonstrator is over, we will rebuild the system from scratch, with a new training UI and following the proposed architecture. This iteration we look at extending the NEAT NES implementation with other Learning algorithms. The SharpNEAT library allows for other training params that we haven't tried, so we will start with those. Some issues with the SharpNeat library, for example: XML config, may be improved with a Winforms UI. As with all our Iteration plans, Jims review will give us an opportunity to fine tune it on Wednesday. Technically we are going to split out the training system as a command line tool to speed up training times.

Phase 2 : Now that we have completed the abstraction piece for being able to plug in multiple neural networks we are going to begin the implementation of other neural networks to allow for our comparative study of neural networks. This will done using the Accord.NET framework in place of the SharpNEAT framework.

Phase 3 : In this iteration we will be porting our code to python to Python to take advantage of more advanced/industry standard neural network frameworks. We will also be implementing Uber Vine logging framework.

Phase 4 : In this iteration we will finalise and debug our training environment. Clean up the codebase and finalise GitHub Readmes and wikis.

Research phase

Phase 1 : Reproduce an open research question on our testing framework and record the results using the vine logging framework.
Phase 2 : Test our proposed solution on our testing framework and compare the results to our original findings.
Phase 3 : If successful; write up the results of our findings into a research paper. If unsuccessful try one last strategy.

Elaboration Phase Project Status Assessment

Elaboration Iteration 1

Iteration Plan:
Now that the Technical Competency Demonstrator is over, we will rebuild the system from scratch, with a new training UI and following the proposed architecture.
This iteration we look at extending the NEAT NES implementation with other Learning algorithms. The SharpNEAT library allows for other training params that we haven't tried, so we will start with those. Some issues with the SharpNeat library, for example: XML config, may be improved with a Winforms UI. As with all our Iteration plans, Jims review will give us an opportunity to fine tune it on Wednesday.
Technically we are going to split out the training system as a command line tool to speed up training times

Main concern for the team is the difficulty in working with the SharpNEAT library and the inability to get it train performantly - we are now formally moving our
efforts away from this framework and towards the Accord.NET framework. This also means we are abandoning the work for Hyper-NEAT

Assessment:
This was intended to become the foundation for our testing platform. We started again having learnt the issues in the library's we were using to build a fast reliable headless training system.

We ended up shifting focus to OpenAI and industry standard tools to reduce the amount of work we had to do ourselves and to be able to leverage the vast amount of community resources available in terms of code, documentation and examples. In short the shift let us focus on the algorithms and training instead of being bogged down by having to build the entire environment as well.

Elaboration Iteration 2

Iteration Plan:
Now that we have completed the abstraction piece for being able to plug in multiple neural networks we are going to begin the implementation of other neural networks to allow for our comparative study of neural networks. This will done using the Accord.NET framework in place of the SharpNEAT framework

1.1 Have not had time to dive deeply into this - I think based on the outcome of our research into local optimas and solving for this we should be able to build a model of what we are trying to understand now - but this needs to be defined in some detail first before looking at what we are trying to log.

1.2 This is a pretty massive undertaking and I (Josh) should not have put myself on it - time constraints are a real issue and this a very involved task - for the next iteration we need to figure out a way to share this load.
2.0-2.4 Work on the UI was put on hold due to uncertainty in which direction the project will go. A potential switch to python from C# would render any work on the UI useless.
3.1-3.3 We have revised the vision roughly around work on solving the local optima problem. However work on fleshing out these documents was delayed due to
not having visibility of Jim's response in a timely fashion. On the basis of our meeting on Wednesday the 9th of May we will have a structured vision and can progress from there on LCAM specific work

Assessment:
We started to formalise our interesting research question in this iteration. We started to understand what a limitation C# was in terms of reusing the industries standard libraries training gyms etc. We started to consider a wholesale switch to python here.

This was a difficult iteration as we were very uncertain about future project direction and were struggling quite a lot with the implementation we had built. The UI work was wasted time if we switched to Python and any work with other C# libraries would also be wasted. We ended up doing much more investigation to come up with the correct approach moving forward.

Elaboration Iteration 3

Iteration Plan:
As mentioned above LCAM work has been put somewhat on hold until our next meeting with Jim to make sure that our project expectations are aligned and that we have the correct templates to work with.
Python port is taking slightly longer than expected - the environment was fairly straightforward to setup within a Docker container but getting the training to work as before is proving challenging. This is likely all down to configuration issues which we will have resolved within the next sprint cycle.

Assessment:
We didn't want to make the mistake of LCOM and try to shoehorn our research project into the development project assessment guide. We reached out to Jim to better understand how to tackle the LCAM as a research assessment.

We started on the major rewrite of out platform, in python with new libraries and hosted in Docker. This also meant switching to a linux environment for dev and test.

Elaboration Iteration 4

Iteration Plan:

Finish off LCAM work, and stabilise the Executable Architecture. If possible it would be good to get logging working for multi-threaded training. There is an issue with the emulator closing, this is causing a big slow down in training.

Assessment:

We were able to fix the issues with with closing the emulator. All the logging issues were addressed and we were able to initiate heavy multi-threaded training. We were able to generate detailed graphs from the test runs.

This iteration came together well with everything we needed to complete being completed and being able to thoroughly test our systems on a fairly powerful server. While there is still some minor work and improvements to be done the overall system functions as expected and can train as required.