NEURO-EVOLUTION IN PAST, PRESENT AND FUTURE VIDEO GAMES

Dom Sleightholme, 10574310, University of Plymouth

Abstract:

Abstract - This paper reviews the research on Neuro-Evolution and how this algorithm is applied to video games in the past, present, and future. It provides research on the origin of artificial intelligence and how this form of intelligence has manifested into our technology and advanced over the years. The paper describes the comparisons made between the artificial and biological neural network and how we can compare the process of human evolution to NE evolution using tests and research to support my argument. It discusses the algorithm NEAT (Neuro Evolution of Augmented Technologies) and how this has been applied to general game playing.

I discuss how NE is applied to general video games using neural networks and their two main contrasting applications: Regression and Classification, it analyses how NE is applied to three different examples of video games, each using NE in different ways to complete certain tasks or gather information for experiment purposes. Five experiments were conducted to analysis how an AI agent is used in the game, Space Invaders. Each experiment used different values to investigate how the controller will perform in the game. After conducting these experiments, the results were analysed to review how the controller is performed in the different experiments and how this implementation of AI can be taken further using other applications.

Introduction:

1.1 - Technology Overview:

Artificial Intelligence, also known as AI, what is it and how does it affect the human civilisation? There have been many definitions of AI, an early one being, "the ability of machines to understand, think, and learn in a similar way to human beings, indicating the possibility of using computers to simulate human intelligence." (Pan, 2016). The term artificial intelligence was first coined by John McCarthy in 1956 when he held the first academic conference on the subject (Smith, 2006). The concept of artificial intelligence stretched back to second world war where British computer scientist Alan Turing worked to crack the 'Enigma' code which the German forces used to communicate during the war, Alan Turing and his team created the Bombe machine which used machine learning to decipher Enigma's messages. Over the years artificial intelligence has advanced with new technology to mimic human intelligence and has increased learning of how to respond to certain actions.

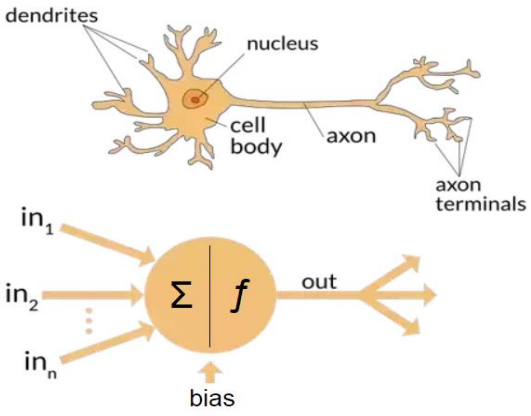

Figure 1: Image above displays the difference between biological neural network (top diagram) and an artificial neural network (bottom diagram) (Sharma, 2017)

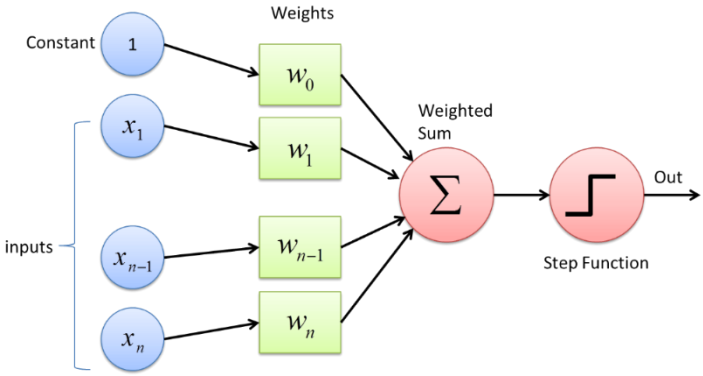

Figure 2: Perceptron. (Sharma, 2017)

In this paper we will be focussing on a form of artificial intelligence known as Neuro-evolution (NE). This form of artificial intelligence is a machine learning technique that takes inspiration from the biological nervous systems. Compared to other AI techniques, neuro-evolution is used to create artificial neural networks (ANN), parameters, topologies and rules which determine how the technique is used in computing. Neural networks (NN) can be described as either a network or circuit of neurons, today we could describe neural networks as an artificial neural network, comprised of artificial neurons or nodes, One of the characteristics of neural networks is the type of transfer function that is used by the neuron within the network (Hoekstra, 2011). The human brain can be described as a biological neural network—an interconnected web of neurons transmitting elaborate patterns of electrical signals (Shiffman, 2012), following on from this, in the paper 'A Logical Calculus of Ideas Immanent in Nervous Activity" the writers describe the concept of a neuron as "A logical calculus of ideas imminent in nervous activity".

Further expanding on neural networks, a perceptron was first introduced in the late 1950's by Frank Rosenblatt, but what is this type of neural network? This type of neural network is an algorithm which is used for supervised learning of binary classifiers, the binary classifier is implemented to determine whether an input belongs to a specific class. This single perceptron has only one node but works nearly identically to multi-layered perceptron (MLP). A perceptron is made up of four parts: Input Values or a single input layer, weight and bias, net sum and finally activation function. This neural network works like a multi-layered perceptron, all the inputs are accumulated and then multiplied by their weights, add all the multiplied values together defined as the weighted sum and finally apply this weighted sum to the correct activation function. A multi-layered perceptron is more than the single perceptron network, the multilayer perceptron is the hello world of deep learning: a good place to start when you are learning about deep learning (Nicholson, 2020). MLP can typically use two activation functions, this is because the activation function is continuous and differentiable, the two functions are: Sigmoid function and the Hyperbolic Tangent (tanh) function. When using this neural network there are two common techniques to help divide the data set into two: the training set which is the set where the MLP will be trained, and the testing set which is used to test the MLP's trained performance on a similar set of data which has not been used.

NE is a promising approach to solving reinforcement learning problems for several reasons (Stanley & Miikkulainen, 2002): NE simplicity of design and structure, neuro-evolution has such a broad availability that this form of artificial intelligence can be used for supervised, unsupervised and reinforcement learning tasks (Sebastian Risi, 2014). Neuro-evolution algorithms have been around since the early 1990's where the algorithms implemented had weights that were represented using bit string and this type of algorithm also had a fixed topology. These types of neuro-evolution algorithms are often called conventional neuro-evolution (CNE) methods because these methods were the first successful attempts at neuro-evolution (Hoekstra, 2011). One of the most common uses of Neuro-evolution was the Neuro Evolution of Augmenting Topologies also known as NEAT, developed by Ken Stanley in 2002. This type of NE was a genetic algorithm that

was used for generation of evolving artificial neural networks, when implementing this algorithm on basic tasks, the NEAT algorithm has been known to often be quicker and more effective than such other techniques like neuro-evolutionary techniques or also reinforcement learning methods. Over the years, the concept of Neuro-evolution has advanced with new technology and increased demand due to its simplicity of design one might argue that it could go beyond this relatively narrow formulation of reinforcement learning (Sebastian Risi, 2014).

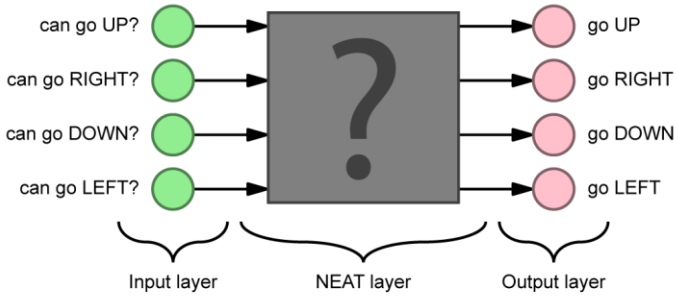

Figure 3: The input is what the brain senses. In this case the four neighbouring wall's state, whether it is a wall (1) or a free space (0) where it can go next. The output is what comes out of this neural network. (VBStudio.HU, 2019)

NE is mostly used in evolutionary robotics and general game playing, for example, neuro-evolution can be used for competitive car controllers in racing games like The Open Car Racing Simulator, also known as TORCS, NE can be used in evolutionary robotics to develop neural behaviour control systems which come closer to the abilities of biological nervous systems, a new class of challenges emerges (Pasemann, 2012), we will further analysis how NE is used in games in the next section.

1.2 - Related Work:

With the increase of demand for computer games over the years, the advancement of computer games technology has come a long way from the early days of Pong which was released in 1972, in our modern-day society we now have games that are commonly huge open world adventure or intense multiplayer matches that can hold up to one hundred players in one match. Current games today are not just fancy graphics, artificial intelligence has played a huge role in the advance of computer games over the years, neuro-evolution has been a part of this advancement in technology. Neuro-evolution has been used in the computer games industry for years from games such as Mario, the 2D platformer where you play as a plumber trying to save the princess, or even games like Quake II, a fast-paced first-person shooter or even NERO (NeuroEvolving Robotic Operatives), where you train robots to fight another team of virtual robots.

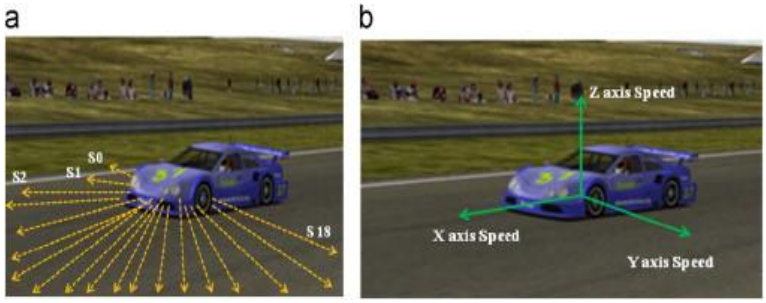

Figure 4: The diagram displays how neural networks are using games such as TORCS (Kyung-JoongKim, 2012)

When applying neural networks to games, there are two main areas of application: Control and Decision making, these contrasting areas play a key part to neural network application to video games. The control area uses a regression, this area of neural networks can predict an output variable known as a control signal from receiving a collection of inputs, the inputs used can be categorical or numeric types, but when using regression this process requires using a numeric dependent type to complete the process, If the output variable is a categorical variable (or binary) the ANN will function as a classifier (Boehmke, 2018). This area of NN can be used to steer a car in a game like TORCS, the vehicles have a set of inputs like speed or the heading factor which will be

initialized in the input layer, the inputs will be used to calculate the weighted sum of the inputs used, the weighted sum will add the bias and finally execute an activation function to get the final outputs which will be used by the vehicles to move or throttle, that's why it is linked with control when discussing the two main application areas NN uses for games.

The decision-making area uses classification, an algorithm where it is used to predict what category data belongs to by using a given set of inputs, this type of algorithm is used primarily for data science, such as when biologists categorize plants, animals, and other lifeforms into different taxonomies (DataRobot, 2020). The decision-making area is used in video games for non-player characters (NPC) whether it is enemies or friendly characters, they will all use classification to compute movement and other actions in game. In some video games today, we have experienced bugs where the NPCs perform completely against their set function, this is due to classification requiring a deep understanding of recent advances in machine learning to implement and while the expert AI players it produces can be challenging, they may not necessarily be fun (Digital Creativity Labs, 2020). This area of application will use neural networks, similar to regression where a set of inputs will be implemented, calculating together in the hidden layer and then output the calculated sum to use, an example using neural networks for decision making is where the NN uses attacking attributes and health as the input and then the calculated sum is whether to attack, flee or defend.

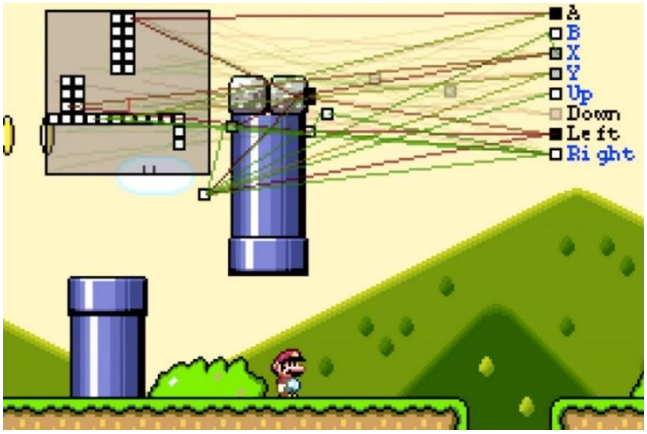

Figure 5: Neural Networks in MarI/0, (GAME ANIM, 2015)

NE is known to have such a broad applicability due to its simplicity of design we can apply NE in several ways when applying this algorithm into the games industry, NE enables new kinds of technology and design into video games, evolutionary computation here provides unique affordances for game design, and some designs rely specifically on NeuroEvolution (Sebastian Risi, 2014). NE can be applied in video games in several ways: TORCS, the car racing simulator uses NE for high-performing controllers used by computer controlled players, NE was used in the commercial game Creatures, the game allows players to breed and raise virtual pets, these virtual pets were controlled by ANN which allowed the pets to learn new behaviours from the player. As seen in current games the opportunities available from applying NE to video games are infinite.

For this research we will be focusing on Super Mario Bros, this platform game was released in 1985 from the game studio Nintendo who developed and published the title, the title was released for the Famicom in Japan and released for NES in North America and Europe. Super Mario Bros is still one of the best side-scrolling platformers (Mott, 2013). Mario games have been known to be easy sandboxes to play with interesting ideas when Japanese fans used the games sound effects to recreate the famous hit "Don't Stop Me Now" released by Queen, Youtuber SethBling developed a program called "MarI/O", the program was an AI controller using neural networks to learn how to play the video game. The neural networks adapted the movement of Mario using a fitness score, this type of score was controlled by every single movement Mario made, the higher the fitness score, the higher Mario travelled in the level. When Mario died, the program will restart the level and will add a

mutation to the AI, this mutation can be anything like making Mario jump or do a certain action at a certain point of the level. The MarI/O program will look for the highest fitness scores and breed them together including adding random mutations, this process was called ‘generations’ and is very similar to human evolution.

Figure 6: Screenshot of Quake II (DigitalFoundry, 2017)

Quake II is a first-person shooter video game which was developed by id Software and published by Activison, this game was released in December 1997 and was not a direct sequel to its previous successor, Quake. The fast-paced shooter was later re-released in June 2019 as remastered version of the original, incorporating the RTX graphics recently released by Nvidia. Quake is a good example of where neural networks have been applied in first-person shooters (FPS). Matt Parker and Bobby D Bryant used Quake II to apply NE differently to games such as NERO, they experimented using a combination of backpropagation and NE to train a neural network visual controller for agents in the game. Backpropagation is a well-known algorithm used for supervised learning of artificial learning using gradient descent; this algorithm is used by calculating the gradient of the error function using the neural network’s weights.

In the experiment they used an empty room which was dimly lit using varying shadows, this experiment included a single enemy, this setup was more realistic due to the design being like the actual game. In experiments previously to this, Parker and Bryant used the same setup but used a less visually complex map, using clear textures and no shadows to affect lighting. The experiment was used to make a comparison between using NE and using NE combined with backpropagation for Lamarckian adaption, the agent was spawned into the setup and was tasked to kill as many enemy bots as possible. The experiment results concluded that the tests that learned using backpropagation for Lamarckian NeuroEvolution were much more successful than the test that used only NeuroEvolution (Bryant, 2009), the controller used for both tests used a neural network with a visual retina input.

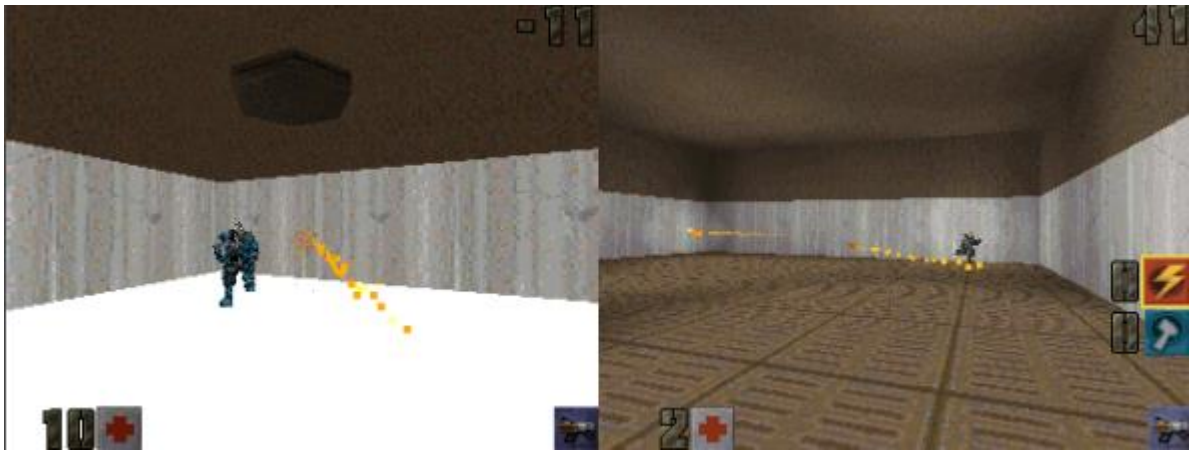

Figure 7: Experiments used by Parker and Bryant to compare NE and NE combined with Backpropagation for Lamarckian Approach (Bryant, 2009)

NERO is a unique take on NE, NERO was a result of an academic research project in artificial intelligence, this project was based on the rtNEAT (real-time NeuroEvolution of Augmenting Topologies), this type of method allows agents to adapt and improve during the game, in NERO this method is used where the player trains a team of robots through a series of customized tasks for combat against another team of robots but this time is controlled virtually. There are several modes to play solo or against friends but still following the same concept, for every time you are doing well, you are rewarded by evolution potential. Your agents evolve to perform better in what they are trained to do (M.Zinoune, 2012). As you improve the agents using training methods, the game will be advanced and complicated compared to previous training making the player think out advanced strategies to control the battlefield. As training of the robots continue to advance so will the enemy agents and battlefield, after fighting off moving turrets and manoeuvring around walls and harsh environments, the robots will be deployed to battle other user's trained teams which are using their own training methods. The developers of NERO have now moved away from the program and are currently developing OpenNERO, based on the previous title this project aims to be used more for research and education in artificial intelligence rather than being a strategy video game.

Figure 8: Screenshot of NERO in action (Willis, 2006)

Experiments and results:

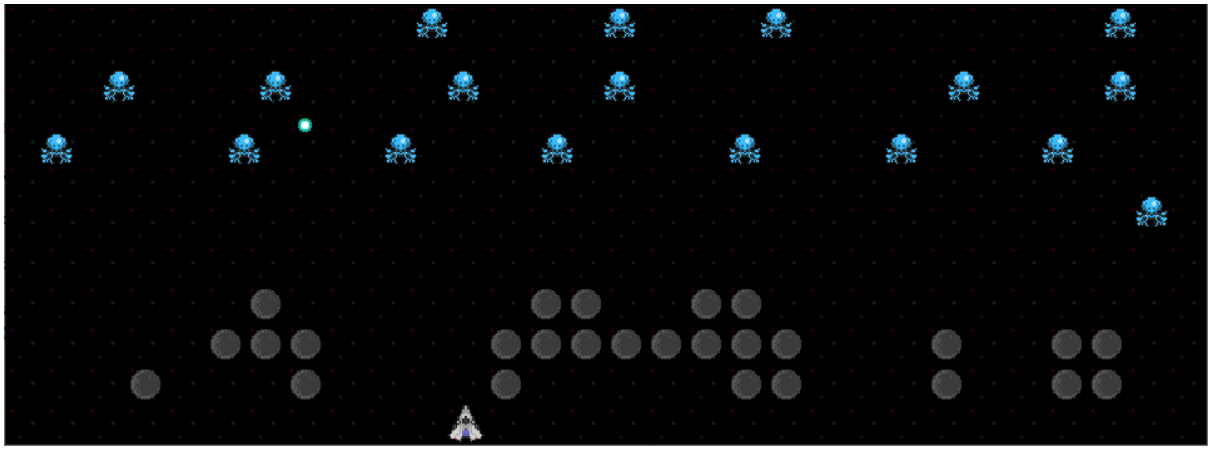
Figure 10: Screenshot from the Space Invaders game being used

For further understanding the technology of NeuroEvolution, a set of experiments were conducted to evaluate the evolution of an MLP Controller acting as an agent in the popular game, Space Invaders. To adapt the controller to compute in the application and to complete the experiments specified previously, modifications were made to the MLP controller: to compute with the input being the representation of the game grid using 1-to-N coding, modified the MLP to compute with the application using the number of hidden nodes, modified the output selection of the MLP using fitness measure, mutation rates, number of generations and the population size. In Figure 11 there are the modified values in the application with a short description of how they compute and range they will use in the experiments. For the research into NeuroEvolution, five experiments were conducted using

different values for the MLP controller and viewport dimensions in each experiment, each experiment will be tested in three different levels, for variation in results.

Setting Name	Range	Parameter Function
Viewport Dimensions		
viewWidth	5- 10	This parameter decides the width of the grid used as the input for the MLP Controller, the width will combine with the grid height and the numberCategories in the numCells calculation.
viewHeight	5 - 10	This parameter decides the height of the grid used as the input for the MLP Controller, the grid follows the player. This parameter is used in the numCells calculation.
numberCategories	4	This parameter is used for the different cell options in the application, for example, Alien, Missile, Obstacle, Empty, Left Window and Right Window.
numCells	144 - 324	numCells is the calculation result of (ViewWidth * numberCategories) * viewHeight, this parameter is used in 1-to-N coding.
MLP Topology		
numberInputNodes	144 - 324	This setting provides information from the viewport dimensions to be calculate in the hidden layer of the MLP Controller.
numberHiddenNodes	10 - 200	numberHiddenNodes takes in the input and calculates an output using an activation function.
numberOutputNodes	5 - 100	The values of the numberOutputNodes are the inputs calculated with the hidden layer, the outputs are used by the agent in the game.
Activation Function		
Sigmoid Function	N/A	Sigmoid is the activation function used in the MLP Controller, this function limits the output to have a range between 0 and 1.
Evolution Parameters		
populationSize	20	This parameter uses mutationMagnitude and mutationProbability to calculate a value used by the MLP controller.
numberGenerations	10	The MLP controller will evolve the same amount of times as the value of this parameter.
numberElite	5 - 10	This parameter is used when removing this parameters value from population size to calculate a value used in the Controller.
mutationMagnitude	10 - 100	This parameter is used in the MLP Controller to calculate movement alongside mutationProbability.
mutationProbability	10 - 100	This parameter is used in the MLP Controller to calculate movement alongside mutationMagnitude.

Figure 11: Parameters Used Table

The activation used in our MLP Controller for our experiments is known as the Sigmoid function, this activation function, known to have an “S”-shaped curve, also known as a sigmoid curve, the Sigmoid function is known to be used when using 1-to-N coding. The function limits the output results of the MLP controller to either 0 or 1, these are used in the 1-to-N coding representation in which the controller is used. For the output representation of my controller, I choose to use 1-to-N coding, this

representation uses the Sigmoid activation outputs to then be further used for the controller's actions. When modifying the MLP Controller, there was the option to modify the number of categories used in the controller, these categories are used in the numCells calculation for the grid of the AI agent. When modifying this parameter, the controller used four categories in the MLP Controllers, with these categories in place the agent was able to recognise the grid boundaries, aliens, obstacles, and empty tiles. The final modifications completed before experiments was modifying the scoring of the game, there was a lot of creative freedom here and alternative concepts of scoring were investigated when conducting the experiments.

Experiment One

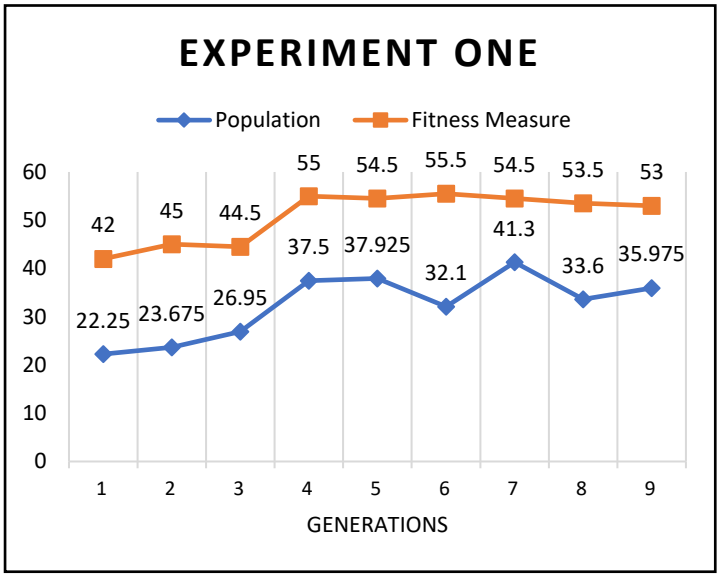

Figure 12: Experiment One

For the first experiment to investigate the implementation of an MLP controller acting as an agent in the application known as Space Invaders, the controller was assigned low values to each of the parameters including the viewport dimensions. The decision for low values for this experiment was to compare the MLP controller using low values for all parameters including viewport dimensions compared to the MLP controller, using the same parameters except viewport dimensions where there will be an increase of value for the viewport values in experiment two. The results presented in Figure 12 showed a steady increase for fitness measure until generation four where the value started to decrease a small amount each generation during the evolution. Population stats shown in Figure 12 increased and decreased multiple times during the evolution process. When the MLP controller was presented to the three chosen levels for next stage of the experiment, the controller was successful in completing all levels used in the experiment, the most successful level of the experiment being level two, the level was completed in 388 timesteps and the overall score for the controller was 45. Overall, this was a successful experiment to show how the controller would compute using the lowest values for all parameters including the parameters for the viewport.

Experiment Two

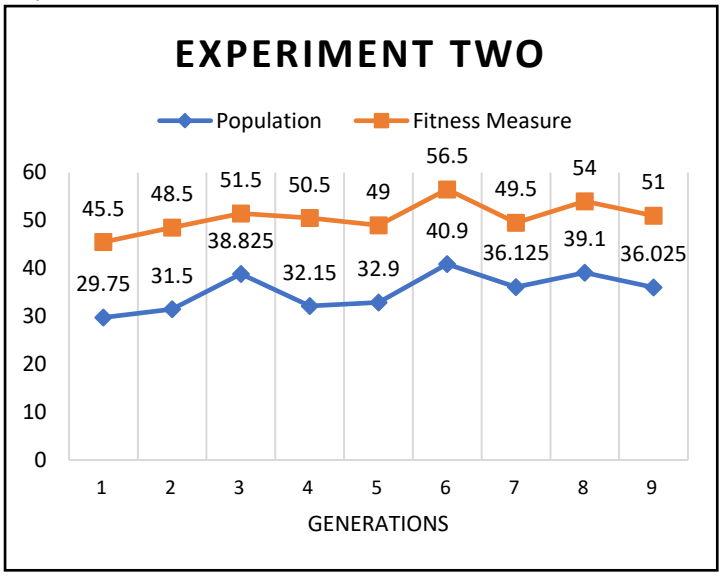

Figure 13: Experiment Two

This experiment will be used to compare the performance of the MLP controller using low values for all parameters except the viewport which will use high values, this experiment will be used to compare the difference in results when the MLP controller is using a high value viewport compared to a low value viewport. The results presented in Figure 13 showed a steady increase in value for the fitness measure except in generation five, in this generation the value increased by a large amount but in the continuing generations the value resumed the steady increase. After reviewing the population stats for this experiment, the value did not increase by a large amount but had a few high values increases in generations three and six. When the MLP controller was presented to the three chosen levels for next stage of the experiment, the controller was successful in completing all levels used in the experiment, the most successful level of the experiment being level three, the level was completed in 361 timesteps and the controller's overall score was 41.

Experiment Three

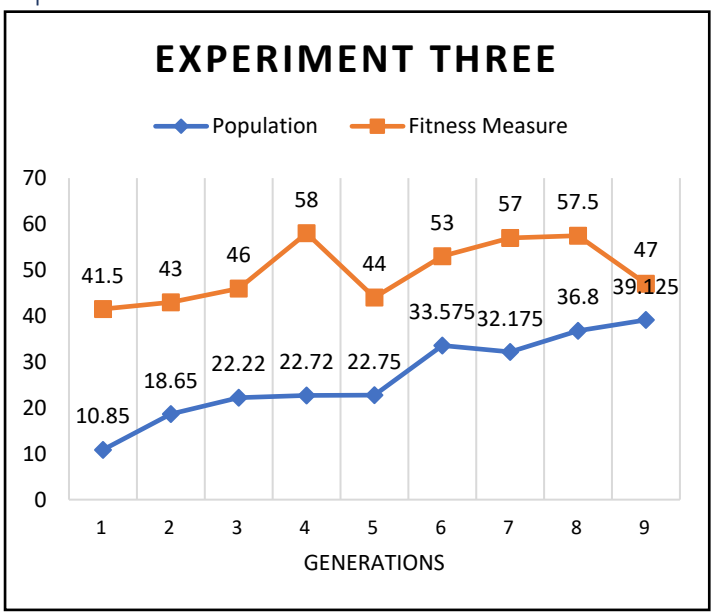

Figure 14: Experiment Four

This experiment will be used to compare the performance of the MLP controller using high values for the controller's parameters and lower values for viewport, this experiment will be used to compare the difference in results when the MLP controller is using a low value viewport compared to a high value viewport. The results presented in Figure 14 showed a steady increase in population stats, at generation eight the value increased by a high amount but after that continued at a steady amount. The fitness measure increased at generation four but then jumped in value, after this drop the fitness measure began to increase again until the generation where it decreased in value again. in value for population stats compared to fitness measure which continued to decrease slightly. When the MLP controller was presented to the three chosen levels for next stage of the experiment, the controller was successful in every level used except level one where the MLP controller was destroyed by the enemy AI, the most successful level was the level three, this level was completed in 434 timesteps and the overall score for the controller was 42. Overall, this was a

successful experiment to show how the controller would compute using the highest values for all parameters except the viewport which used low values.

Experiment Four

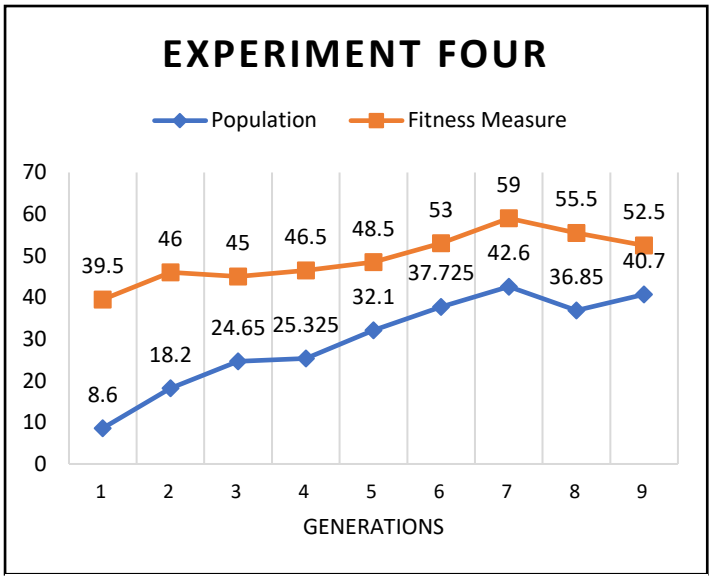

Figure 15: Experiment Four

This experiment will be used to compare the performance of the MLP controller using high values for the controller's parameters and viewport dimensions compared to the previous, experiment three where that evolved controller used high values for the parameters except the viewport which used low values for the dimensions. The results presented in Figure 15 showed a steady increase in value for both the population stats and the fitness measure, there was a slight drop at generation eight for both values but there was an increase in value for population stats compared to fitness measure which continued to decrease slightly. When the MLP controller was presented to the three chosen levels for next stage of the experiment, the controller was successful in completing each level without being destroyed by the enemy AI, the most successful level being level one where the amount of timesteps the level was completed in was 430 and the controllers overall score was 49. Overall, this was a successful experiment to show how the controller would compute using the highest values for all parameters including the viewport.

Experiment Five

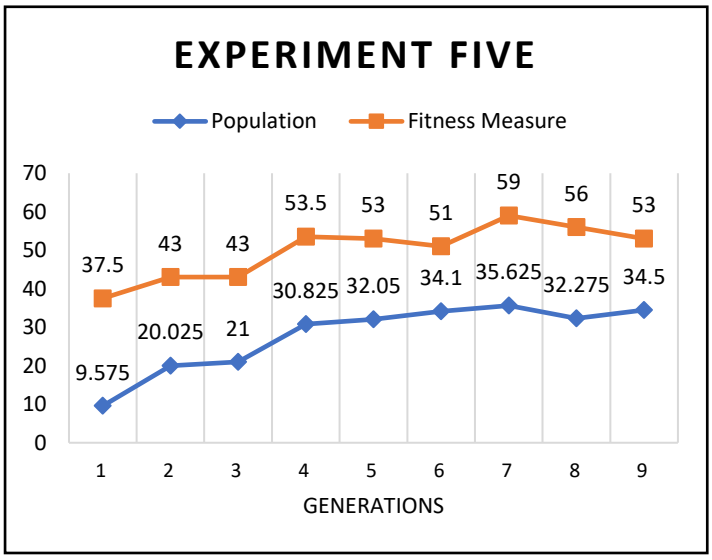

Figure 16: Experiment Five

For the fifth and final experiment, the controller will be assigning the average value between the high and low values previously used as values for this experiment. The decision to use the average of values of the high and low values used previously was to analysis how the MLP controller would function using the average values compared to the lowest and highest values used previously. The results presented in Figure 16 showed a steady increase of population stats value each generation compared to the fitness measure where this value had a slight drop in the middle and end of the evolution process. When the MLP controller was presented to the three chosen levels for next stage of the experiment, the controller was successful in completing each level without being destroyed by the enemy AI, the most successful level being level

three where the amount of timesteps the level was completed in was 361 and the controllers score was 41. Overall, this was a successful experiment to show how the controller would compute using the average values of the previous experiments.

Discussion:

All results obtained were successful in completing the evolution of generations and obtaining data on the population stats and fitness measure for all generations, with these experiments completed successfully there is a clear indication that after each generation when evolving, all experiments increased value for both measures when comparing their values at the start and end of the evolution process. After completing the five experiments, there were not enough values for each of the experiments to determine positive or negative changes when analysing the results for comparisons with the experiments conducted, due to the lack of results to use for analysis, the experiments were conducted again using ten generations instead of the previous set which used only five generations in each evolution. One of the key factors from analysing all the levels completed for each experiment was that three out of the five experiments conducted completed level three compared to the rest of the levels used in the experiments, these experiments being; experiment two, three and five. After reviewing the three experiments which all had level three as their best score, there was not a common factor as each experiment used different values for the controller and the viewport.

After analysing the results there was one unusual factor which occurred in experiment three, this factor being that the controller failed to complete level one after the evolution process, this factor was unusual because in all the other experiments the controllers were able to complete the level and defeat the enemy AI. From analysing the unusual result further, experiment three used high values for all parameters except the viewport in which low values were used, the result of this could be the reason for level failure, the high values for the MLP parameters could not compute with the small grid size of the viewport. After further analysing the results the lowest average timesteps out of all experiments was experiment two, this experiment used low values for all parameters except viewport which used high value parameters. After further analysis, the fitness measure was at its highest value in experiments five and one with a value of 53. The population stats were highest in value in experiment four with a value of 40.7, experiment three was close behind with a value of 39.125

After analysing the results, both the population stats and the fitness measure for experiment two did not increase in value in comparison to the rest of the experiments after the evolution process, this experiment in particular was the experiment where low values were used for all parameters except the viewport in which high values were used, this factor leads to the idea of the MLP controller having a large amount of the viewport accessible to use but low values in input, the controller was not successful in using the viewport to its advantage. After further analysis, experiment four seemed to increase in both data measures the highest compared to the other experiments which were conducted, both data measures increased in high value after each generation except generation eight, this generation was where both data measures were at their highest, after generation eight, the data measures began to decrease by a low value but managed to still have a good value for each data measure, this experiment used high values for all parameters for MLP controller except the viewport which used low values. Experiment five was close to be the most successful evolution compared to experiment four, this experiment used the average values of the highest and lowest values used in the previous experiments so far.

After reviewing the results and finding the experiments four and five to have the best results after the evolution, the experiments showed that the higher the parameter values are for the MLP controller, the greater the results. The viewport dimensions in these experiments did not use the highest value compared to the MLP controller parameters which used the highest values and the average values of the previous highest and lowest values used. One of MLP's weaknesses is setting parameters as this process is known more as an art rather than a science, this method of setting parameter values was used highly during all the experiments. After all the experiments were conducted and reviewing their results, the findings prove that using the NeuroEvolution technique for an AI agent in Space Invaders is a good example of AI being applied to video games. The idea of AI being applied to video games in the way this agent has been in the Space Invaders application has been used for several other games, these games being; Chess, Pong and several other games to conduct how AI can be used to play a video game. Overall NeuroEvolution has shown how an AI agent can be applied to a video game and outperform a biological player using generation evolution, the process of picking the fittest of each generation, breeding them together and adding random mutations very closely matches the process of biological evolution which took single cell organisms and produced intelligent humans (Brockman, 2016), that is the power of neuro-evolution.

References

Boehmke, B., 2018. *UC Business Analytics R Programming Guide.* [Online]
Available at: http://uc-r.github.io/ann_regression
[Accessed 15th April 2020].

Brockman, J., 2016. *Life: The Leading Edge of Evolutionary Biology, Genetics, Anthropology, and Environmental Science.* 1st ed. New York: Harper Perennial.

Bryant, M. P. a. B. D., 2009. Lamarckian Neuroevolution for Visual Control in the Quake II. *Congress on Evolutionary Computation,* p. 2630–2637.

DataRobot, 2020. *Classification.* [Online]
Available at: datarobot.com/wiki/classification/
[Accessed 15th April 2020].

Digital Creativity Labs, 2020. *Decision Making AI for Games.* [Online]
Available at: https://www.digitalcreativity.ac.uk/projects/decision-making-ai-games
[Accessed 20th April 2020].

DigitalFoundry, 2017. *DF Retro: Quake 2.* [Online]
Available at: https://www.digitalfoundry.net/2017-05-14-df-retro-quake-2
[Accessed 19th April 2020].

GAME ANIM, 2015. *ARTIFICIAL INTELLIGENCE LEARNS MARIO LEVEL IN JUST 34 ATTEMPTS.* [Online]
Available at: https://www.gameanim.com/2015/06/19/artificial-intelligence-learns-mario-level-in-just-34-attempts-2/
[Accessed 18th April 2020].

Hoekstra, V., 2011. *An overview of neuroevolution techniques.* Amsterdam: s.n.

Kyung-JoongKim, J.-H. S. J.-G. P. J. C. N., 2012. Neurocomputing. *Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis,* Volume 88, pp. 87-99.

M.Zinoune, 2012. *Neuro-Evolving Robotic Operatives | Friday Game.* [Online]
Available at: https://www.unixmen.com/nero-evolving-robotic-operatives-friday-game/
[Accessed 17th April 2020].

Mott, T., 2013. *1001 Video Games You Must Play Before You Die Paperback.* 1st ed. London: Cassell.

Nicholson, C., 2020. *A Beginner's Guide to Multilayer Perceptrons (MLP).* [Online]
Available at: https://pathmind.com/wiki/multilayer-perceptron
[Accessed 08 April 2020].

Pan, Y., 2016. Heading toward Artificial Intelligence 2.0. *Engineering,* 2(4), pp. 409-413.

Pasemann, C. R., 2012. An Interactively Constrained Neuro-Evolution Approach for Behavior Control of Complex Robots. In: Heidelberg, ed. *Variants of Evolutionary Algorithms for Real-World Applications.* Berlin: Springer, pp. 305-341.

Sebastian Risi, J. T., 2014. State of the Art and Open Challenges. *Neuroevolution in Games:.*

Sharma, A., 2017. *What is the differences between artificial neural network (computer science) and biological neural network?.* [Online]
Available at: https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network
[Accessed 13 March 2020].

Sharma, S., 2017. *What the Hell is Perceptron?.* [Online]
Available at: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53
[Accessed 28 March 2020].

Shiffman, D., 2012. *The Nature of Code.* California: Free Software Foundation.

Smith, C., 2006. *The History of Artificial Intelligence.* [Online]
Available at: https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf

Stanley, K. O. & Miikkulainen, R., 2002. Evolving Neural Networks through. *The MIT Press Journals,* 10(2), pp. 99-127.

VBStudio.HU, 2019. *Growing an AI with NEAT.* [Online]
Available at: https://vbstudio.hu/en/blog/20190317-Growing-an-AI-with-NEAT
[Accessed 28 March 2020].

Willis, N., 2006. *AI versus AI: N.E.R.O. on Linux.* [Online]
Available at: https://www.linux.com/news/ai-versus-ai-nero-linux/
[Accessed 20th April 2020].

Appendix:

Experiment Results:

Experiment Number	Fitness Measure	Population Stats	Level 1		Level 2		Level 3		Best	
1	42 45 44.5 55 54.5 55.5 54.5 53.5 53	22.25 23.675 26.95 37.5 37.925 32.1 41.3 33.6 35.975	597	64	388	45	449	42	388	45
2	45.5 48.5 51.5 50.5 49 56.5 49.5 54 51	29.75 31.5 38.825 32.15 32.9 40.9 36.125 39.1 36.025	409	59	608	51	361	41	361	41
3	41.5 43 46 58 44 53 57 57.5 47	10.85 18.65 22.22 22.72 22.75 33.375 31.175 36.8 39.125	855	53	633	49	434	42	434	42
4	39.5 46 45 46.5 48.5 53 59 55.5 52.5	8.6 18.2 24.65 25.325 32.1 37.725 42.6 36.85 40.7	430	49	837	51	835	55	430	49
5	37.5 43 43 53.5 53 51 59 56 53	9.575 20.025 21 30.825 32.05 34.1 35.625 32.275 34.5	565	61	555	48	361	41	361	41

Key:

Red = The Controller failed to complete the level

Yellow = The Controllers best score compared to the other levels used in the experiment.