2021

Machine Learning Ecosystem

EXPLORATION OF A USE CASE FOR MACHINE LEARNING IN GAME DEVELOPMENT

KYLE DAKIN

Contents

Abstract

This study explores a potential use case for machine learning in the game development industry. The research compares different technologies that could be used for this project, as well looking at some of the biological aspects to evolution aiding in the development of the product. The aim is to create a proof-of-concept application, that creates different neural networks for each terrain system. The application should also be able to be scaled up to a more complex project if this research were to go any further than this project.

Introduction

Machine learning is a field of computer science that has seen growing interest, while slowing in recent years, in the last decade. Various industries are trying to explore it's potential use cases in their respective field, from Tesla's AutopilotAI (Tesla, 2021) to image based diagnostics in medicine (Nature, 2019), however the game development industry has been reluctant to follow this trend and have persisted with traditional artificial intelligence (AI) algorithms to handle their games AI.

Aim

The aim of this project is to explore a possible use case for machine learning in video games development. Artificial intelligence is one of the many tools developers use to create immersive worlds for the player to engrossed in. According to "The Psychology of Video Game Immersion" (Madigan, 2012), things "that create consistency between the things in that environment" facilitates immersion in the game world, which is where the aim of this project is focused. The goal by the end of this project, is to develop a collection of neural networks that have evolved across a selection of terrains. The different terrains replicate the variety of unique environments that exist within a single game world. Animals in the real world evolve and adapt to their local environments (Hershey, 2018), however animals in video game worlds use the same generic AI across the world and often use the same creature models. The approach this project takes explores how a neural network could be precomputed to generate and control different creatures based on the region within the game world.

Literature Review

Background

Machine learning is defined as a "branch of artificial intelligence (AI) focused on building applications that learn from data and improve their accuracy over time without being programmed to do so" (IBM Cloud Education, 2020).

The main use of machine learning in games has been algorithms learning to play a game to either beat the game, such as MarI/O (Benchoff, 2015), or beat human players, such as OpenAI's OpenAI Five. MarI/O used an evolutionary algorithm that controlled the player character 'Mario' guiding him through the level to the flag at the end of the stage. Meanwhile OpenAI Five was a much more complex case where it learned to play competitive five versus five team strategy game Dota 2, with the goal of beating the world champion team to demonstrate "that self-play reinforcement learning can achieve superhuman performance on a difficult task" (Berner, et al., 2021).

However, neither of these cases help in the development of games or could be included into modern games as either the character is controlled by a human, removing the need for machine learning AI, or the opposing AI is designed to be able to be beaten by the human agents. Since machine learning AI would create mostly unbeatable AI that would be very frustrating for new or low skilled players to play against.

Another case against the use of machine learning in video games is Alien: Isolation. Alien: Isolation is a survival horror game, developed by Creative Assembly, that developed a complex AI that convinced players that it was learning and therefore had to use a machine learning algorithm to control the hostile Alien's behaviour. However, Alien Isolation did not use any form of machine learning, instead using a very well-designed behaviour tree, an AI technique popularised in 2004 by Bungie's Halo 2 (Opsive, 2021). This made the Alien behave in a way that felt realistic, which helped create a greater sense of fear in the player, an emotion vital to the survival horror genre. Machine learning algorithms come with additional research, development, and training time to achieve the results that would be required in a game setting, and by simply using a cleverly designed behaviour tree, Creative Assembly were able to give the player the feeling that the AI was learning without needing to spend the additional resources developing a machine learning AI. Another negative in this situation, is that if Creative Assembly did use a machine learning algorithm and only trained it on the current player playing the game, they would not be able to guarantee any behaviour from the Alien, which could result in unplayable conditions as the Alien could simply learn that the best course of action would be to relentlessly follow the player until they are dead, which would the player would not be able to negate. Alternatively, the AI could just learn to ignore the player entirely completely removing the main cause of terror in a survival horror genre.

Motivations

The motivation for this project is noticing how machine learning has not much relevant utilisation in the game development industry. While there are articles that cover some ways that machine learning has been explored or is being utilised in games (Olckers, 2019), they often refer to machine learning being used to play games rather than being used for anything in the development cycle. There has been some research done by companies such as Nvidia and EA (Electronic Arts, 2021), that have produced research papers on uses of machine learning, these papers have seemingly yet to be utilised in the games industry. Furthermore, this project is motivated by products such as Nvidia's GauGAN which helps create photorealistic landscapes using machine learning (Salian, 2019), as this demonstrates a way that AI could be utilised in game development, by using precomputed neural networks to aid in the development process, rather than being used within the game developed.

Software Decisions

The focus of this project is around utilising machine learning in a games development setting rather than developing a game engine containing machine learning, so using a commercial engine such as Unity or Unreal is a logical decision as this saves a lot of development time. Furthermore, Unity and Unreal both comes with pre-existing libraries and tools that further aid the development of this project. There are alternatives to Unity and Unreal, such as PyGame for Python, however these come with additional development requirements as they have fewer in-built features, so choosing between Unity and Unreal makes the most sense for this project.

Unity and Unreal provide different features that benefit this project. For example, Unity introduced a visual scripting package (Unity, 2021), this package makes developing AI behaviour easier to understand visually. An additional benefit to the visual scripting package, is that it requires little to no coding which makes it ideal for games designers to use, so they can modify the AI's behaviour without having to understand C#. This is a useful feature for this project as one of the aims is to make the project expandable and reusable for future projects and other developers, therefore using the visual scripting package would make the project accessible to more people who may be interested in using this technology. However, the visual scripting system is new to Unity and only has a few resources available for beginners to learn how to use it, which does not guarantee that the project would be useable or scalable by anyone.

Unreal Engine also has a visual scripting system called Blueprints which has many resources and tutorials available which would make the project more accessible to anyone as they could follow existing tutorials to add more complex behaviour to expand the model. Additionally, Unreal comes with an in-built behaviour tree system. As previously discussed, behaviour trees are a popular way of controlling how an AI behaves and Unreal takes a visual scripting approach, similar to blueprints, that makes them game designer friendly as well as saving development time as Unity has no official support for behaviour trees, which means that either a paid third-party package would be required, or development time would have to be spent on developing the AI's behaviour tree in code. Another benefit of the Unreal's visual scripting approach in both Blueprints and behaviour tree system is that visual scripting can be a lot easier to debug as you can see exactly what code or behaviour is currently being executed during runtime, whereas coding requires debug logging in order to see anything, which can be a lot slower.

A major advantage to using Unity is that they have far greater support for machine learning projects, as Unity have an official ML-Agents package available in the Unity Package manager, which runs Python code in Unity through an open port. Unreal has the ability to use Python code through a port just like Unity, however Unity's official support for machine learning, with documented code, examples and tutorials has generated more interest in developing machine learning in Unity than Unreal, so there is more support and documentation for machine learning projects. Furthermore, Unity has a package called UnityNEAT, which is an adaptation of SharpNEAT for Unity, written by Daniel Jallov as part of his master's thesis (Jallov, 2015). From prior research, NEAT (NeuroEvolution of Augmenting Topologies) seemed like a suitable algorithm for this project, so Unity having a pre-existing template as a backup, while Unreal has no NEAT template, makes Unity a more logical choice for this project.

While Unity has little support for more traditional AI techniques like behaviour trees, implementing state machines and behaviour trees in code is a lot easier than having to do the same for machine learning algorithms, so the greater machine learning support in Unity makes it the better engine for this project.

Controlling Behaviour

There are a variety of ways of controlling the behaviour of an AI. The popular techniques in games development are state machines, behaviour trees and utility AI.

State machines are the oldest technique of the three but are still a very viable method of controlling actions and behaviours. Also knows as finite state machines (FSM), the basic idea of an FSM is to store a sequence of states and transition between the states depending on the inputs and if the inputs meet a specified criterion (Wilson & Mantooth, 2013). However, they are limited to only running one state at once, so if an AI needed to carry out multiple actions at once, a state machine would not be able to do this. Furthermore, state machines can lead to code getting too disorganised and complicated as the state machine grows and more states and transitions between states are added.

Behaviour trees address some of these flaws, as they can run multiple behaviours at once, while typically producing more organised code, as behaviour is managed into branches, with selectors and sequences being used to help select what behaviour sequence to execute. A behaviour tree is a "tree of hierarchical nodes that control the flow of decision making of an AI entity" (Samson, 2014). Behaviour trees are collection of tasks, that consist of actions, conditionals, composites and decorators. Actions are simply just an action to be carried out by the AI. Conditionals is an action that checks specific conditions, for an example if a gun has ammo in order to shoot, and the carries out an action appropriate to the condition. Composite tasks are tasks that are composed of other tasks in a sequence, for example a shooting composite tasks would have a conditional task that checks the ammo in the user's gun, and then have an action for firing the gun if there is ammo or reloading the gun if the ammo count is zero. Decorators are a task that modifies the behaviour of the sequence in some way. (Opsive, 2021). A tree is built up of these nodes, and they are evaluated at runtime, from top to bottom, left to right and actions are carried out based on what nodes in the tree get met.

As previously discussed, behaviour trees are a popular technique in industry having been popularised in 2004, and demonstrated to still be just as useful, if not more so, in 2020 in Capcom's Resident 3 Remake, which utilised behaviour trees to manage the zombie and Nemesis AI. Additionally, behaviour trees being an implemented feature into Unreal engine, mean that a both programmers and designers know how to use behaviour trees which aids this project being expandable by other people.

Utility AI is older than behaviour trees, having been first used in The Sims in 2000, however it became more popular in 2010 after a Games Developers Conference (GDC) given by Dave Mark and Kevin Dill (Mark & Dill, 2010). Utility AI works by identifying the possible actions of the AI, and then creating a score in that context. These scores are then evaluated to determine what action is best to carry out, and then that action is executed. For example, if an AI had a gun, it could have one of three actions, fire the gun, reload the gun, or move closer to an enemy. Each action would be scored based on some function, for instance if the gun has a max ammunition count of twenty but a current ammo count of one, the function will return a greater score than if the max ammo count were twenty and the current ammo count was also twenty, as there a greater need to reload the gun. (Huebner, 2017)

Due to the simplicity of the behaviour desired for this proof of concept, the decision was made to use a basic state machine. The organisms in this demonstration only have two main behaviours, flee, and consume food. While a utility AI or a behaviour tree could both work for this type of behaviour and would both allow for expansion if there is time available or if the project is revisited, the additional time required to develop the behaviour tree system or functions for scoring actions needed for both systems, are not justified considering the lack of complexity in the behaviour.

Procedural Generation

One of the ways planned for the creatures to evolve is to use procedural generation to create the body of the creature. This would allow for factors such as a creature's height or its number of legs to be relevant to its evolution. However, procedurally generating an entire creature's mesh and having that mesh be functional is an incredibly complex task. Video games have used procedural generation to generate the creatures that the player encounters. Games like these, such as No Man's Sky, are one of the inspirations for this project, as the creatures are just generated randomly using a library of body parts. The random assignment of creatures to each area may sometimes cause some creatures to feel out of place, as they may resemble a creature that exists in real life but is found in an environment that is not where its real-life counterpart would exist. So, having procedurally generated creatures that have evolved to their environments allows for organisms to feel like they belong in that environment, enhancing the players immersion, while still maintaining the random element of procedural generation.

However, these generation algorithms often used a collection of pre-modelled body parts that are put together procedurally. With no access to a library of models like this, and the additional complexity added by trying to make completely procedural creatures, the decision was made to not implement this idea to ensure the project could be completed. If there is enough time of the end of development, this feature will be revisited.

Machine Learning Algorithms

One of the reasons previously provided for choosing Unity over Unreal, is Unity's existing support and interest in machine learning projects. The official Unity package for machine learning is called ML-Agents which uses Python and Tensforflow (Bonzom, 2020). ML-Agents is comprised of three components: the learning component, the Python API (Application Programming Interface) and the external communicator. The learning component is the actor in the scene with a 'Brain' component. The brain communicates with the 'Academy' which controls the agents and their decision-making processes. The academy is also what communicates with the Python API (Simonini, 2020). ML-Agents primarily supports reinforcement learning and imitation learning. Reinforcement learning is "the training of machine learning models to make a sequence of decisions" (Osiński & Budek, 2018). This type of learning is not the type of learning desired for this project as the goal is not to enforce or create a specific behaviour pattern, but instead to let one evolve as a pattern naturally would in the wild. Imitation learning is a technique designed to mimic human behaviour in a given task (Hussein, Gaber, Elyan, & Jayne, 2017). Similar to reinforcement learning, imitation learning is not a suitable form of learning for this project as it does not learn naturally. Both reinforcement learning and imitation learning could be very useful if the project were to be adapted to simulate the behaviour of pre-existing real-life creatures, so if this project were to be revisited and have a greater focus on one organism, both of these learning algorithms would work very well.

As previously discussed, UnityNEAT is a pre-existing NEAT package written by Daniel Jallov (Jallov, 2015). The aim of this project is to implement the NEAT algorithm without the use of a third-party package, however having an open-source package protected by the MIT License available as a backup plan ensures that the project can at least be completed. NEAT is a Neuroevolution algorithm developed in 2002 by Kenneth Stanley and Risto Miikkulainen. Neuroevolution is the "artificial evolution of neural networks using genetic algorithms" (Stanley & Miikkulainen, 2002). The reason why NEAT was decided upon was because the way it learns closely mirrors biology, which is partially the aim of the project, to mirror biology, this aligns with my goal to create a realistic as possible, and somewhat biologically accurate programme.

Biology research

While this project is a computer science research project, one of the project aims was to simulate evolution and adaptation in the wild to create a more natural attributes and behaviour patterns for AI in video games, so in order to simulate adaptation as best as possible biological research is required. However, I am not a biology student so I have a very limited education in biology so this project could benefit from collaborating with an evolutionary biologist as their knowledge when it comes to what effects evolution and how systems like the metabolic system change in animals. Additionally, due to my lack of knowledge in the field of biology, some of the concepts relevant to this project are too advanced for my knowledge, which have led to the concept being simplified in code, which affects how accurately the project represents natural evolution and adaptation in the wild.

One issue found is that there is no generic formula for calculating an animal's metabolic rate. The formula for calculating metabolic rate is $BMR = aM^b$ (White & Seymour, 2003) where BMR is the basal metabolic rate, a is, according to research, the Bergmann rule constant (ScienceDirect, 2021), and M is the mass. According to research the exponent b is a point of contention regarding what the value is, however for this project Kleiber's Law is used which makes b = 0.75 (Niklas & Kutschera, 2015). The issue of the value of the exponent affects the accuracy of the evolution of the neural network, therefore it would benefit from future work with biologists to potentially create a formula or method that allows this method to adapt in this kind of experiment.

Furthermore, research was carried out into the energy gained from eating particular foods as what food is available affects a creature's evolution. However, this is a very complex field that does not have a consistent formula for, as the energy value of a food source is dependent on what that food is comprised of, and this project is intended as a proof of concept in computer science, so the model is being restricted to one food source with an arbitrary energy return. However, if this project is revisited and expanded upon, a suitable approach could be to create food sources that occur naturally in the specific biomes, so the model would more accurately represent how an organism would evolve in that specific ecosystem with the resources available to it.

Finally, research into ecosystems can be very complex due to the many components of an ecosystem, from its climate to its elevation, which all play a role in how that ecosystem functions (Myers, 1992). Due to the aim being simplified to a proof of concept, a lot of factors that would naturally affect evolution in an ecosystem have to be omitted, which could affect the accuracy of the final neural networks. These could be included later, either if there is time in the project or if the project is revisited, through heatmaps and additional resources, such as water, for the creature to adapt to.

Other potential use cases for this project

As discussed in the biology research section, there is a lot of overlap between this project and evolutionary biology, so if this project were to not be useful in games development, it could still prove to be useful to biologists to be able to model ecosystems and test ecological theories. For example, the value for the exponent in the metabolic rate formula has been debated since it's conception as it proves to be a reasonable explanation for birds and mammals, but does not work for multicellular plants and other organisms, so by further developing this project by making it more specific to individual organisms, biologists could test their theories for what the best value is, or perhaps if there is no best value, then what is an appropriate function for the exponent. Typically, in biology this is done by collecting data about specific organisms and then testing the data mathematically. However, this method is limited due to small sample sizes versus the total population. Using machine learning can theoretically test a much larger population to see if the maths produces the organism in the way they expect.

Furthermore, with ecosystems changing due to global events like human expansion and climate change, biologists are having to predict what the effects of these changes could be on ecosystems and the organisms inhabiting the area. However, with this type of research, biologists could model the new environments and test how the existing wildlife would adapt to the new ecosystem without having to endanger any living animals. This would help identify creatures that would be most at risk of extinction, so they could receive greater priority. On the topic of human expansion, SpaceX and Elon Musk aspire to send 1 million people to Mars and colonise the planet (Duffy, 2021). Mars is currently an uninhabited planet so there are no models of what could survive on Mars, and there is currently no way of testing what would survive without taking organisms up to Mars to find out. This project could potentially solve that issue. By accurately representing Mars as an environment, different modelled organisms could be tested in the environment, to test whether they would be able to successfully adapt to the planet's environment.

Professional, Legal, Ethical, Social Issues

Due to the nature of this project, there are not many issues to address in this area. The main one is the professional and ethical issue of using third party code. In this project, a third-party engine is being used as well as third party packages, however all of these are protected under license agreements such as the MIT License. The MIT License grants permission, free of charge, to anyone obtaining the software and documentation without restriction (Open Source Initiative, 2021). All work that is not my own, for example UnityNEAT, is referenced and credited to the original author, and is all licensed under open-source code protections. Unity engine comes with a personal license which covers projects developed in Unity that generate less than $100,000 in revenue (Unity, 2021).

There are no major social or legal issues to be concerned about in this topic as the project is just modelling evolution, which is not a morally or legally dubious about this topic, and there is no cyber security or controversial issues being studied. The project also does not produce any technology that may affect people in a significant way in their life, as the only products are neural networks that show the evolution of creatures in ecosystems and a Unity project file that could be used to expand the model.

Workflow

Creating Terrain

The terrains decided upon for this project were a flat grassland, a forest area, and a mountainous area. These terrains were chosen as they are very common in video games and simple to produce without being a games designer. They also have different ecosystems and pose different challenges for the AI to deal with. The decision was also made to keep all the scenes at ground level. Pathfinding in 3D space would be required to navigate a creature through the air and water. Furthermore, more research would be required into the biology and adaptation of sea-based animals as the biological research carried out focussed on land-based mammals.

Figure 1 Forest scene from Legend of Zelda Breath of the Wild by Nintendo. Example of forest scene in games (Zelda Wiki, 2021)

To produce the terrain, there were two options; one was to use Unity's Terrain system and the other is to procedurally generate the terrain. The decision was made to use procedural generation. Procedural generation in Unity is aided with inbuilt functions and libraries like Mathf.PerlinNoise which is a noise function that generates a noise map based on the Perlin Noise algorithm that can be used to procedurally generate terrain. Using procedural generation saves development time in the future, as any changes that need to be made just require changing some values. Furthermore, if a new terrain were added to the project, the terrain builder would require a new terrain chunk and for all the details to be painted on, whereas the procedural algorithm will already have been written, making it a faster process to produce the new terrain. The procedural generation system uses a biome system, that controls where trees as well as where the plants, which act as food, can spawn on the terrain. To navigate the terrain, Unity has an inbuilt pathfinding system called NavMesh and

Figure 2 Scene from Elder Scrolls 5 Skyrim by Bethesda. Example of mountain scene in games (Datadas, 2014)

NavMeshAgents. Once the terrain has been generated, a NavMesh is baked using the terrain mesh. The NavMeshAgent component is attached to the creature GameObject. At runtime, the NavMeshAgent component uses the NavMesh to navigate around the terrain. There is also a NavMeshObstacle component which is attached to the trees in the forest scene so the NavMeshAgent must navigate around it. For the mountainous terrain, the mountains act as an obstacle without need the obstacle component as the slope is too steep is some places for the NavMeshAgent to navigate. The angle of the slope that the agent can navigate can be changed, and in a future revisit of the project, the slope that a creature can climb will be included into its evolution as there are animals that live at ground level, while other animals live higher in the mountains.

Controlling Behaviour

As previously mentioned, the technique chosen to control the AI's behaviour was the state machine method. The behaviour decided upon for this project has been intentionally kept simple. The states needed for the state machine are Sleeping, Wandering, Moving To Food, Fleeing and Eating. To control this behaviour only a small number of decisions are made, which is what makes the state machine a suitable choice for this project.

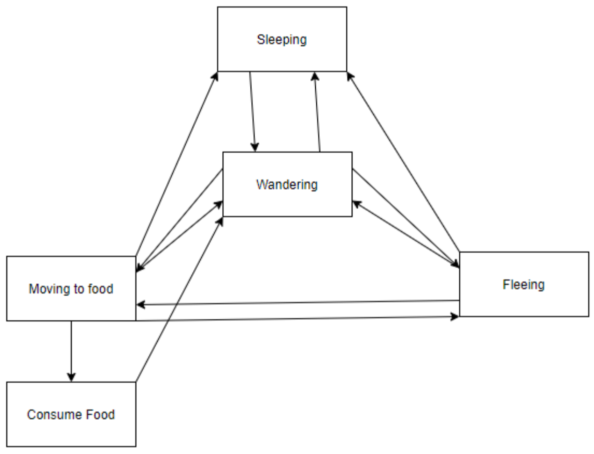

Figure 3 Model of AI State Machine structure

There is a basic day night cycle in the project, which is primarily in place to respawn the food that gets eaten during the day. When it is night-time, all the creatures will be asleep, so they can't hunt or flee.

The behaviour is designed around the 'Prey' creature which can both eat food and be eaten. If the creature sees food, then it will move to eat the food, and once it has arrived it will then eat the food. If they creature sees a predator, then it runs away in the opposite direction at twice the speed. There is a condition that balances whether the AI choses to prioritise food or fleeing as a creature may see food, but then see a predator, but the choice may not always be the same. The decision making here is very simple and if the project were to be revisited, the additional complexity would be implemented. The way that this choice has been implemented is simply just a weight system, so if the creature weighs finding food more highly, then it will favour going to the food, than fleeing from the predator. This weight is adjusted whenever the creature dies based upon its cause of death. If the creature starved to death, then it will favour finding food over fleeing, and if it is eaten by a predator, then it will increase the weight for fleeing. The current implementation also does not consider the distance between the creature and the food, and the creature and the predator, which would be something an animal would most likely consider in real life. The animal also does not try to flee for cover, instead it currently just runs away until it is no longer able to see the predator.

If the creature cannot see food or a predator, then it wanders about the NavMesh, until it finds either food or a predator.

Constantly throughout the day, the creature's energy is drained based up its metabolic rate as well as its activity level. For this project, the metabolic rate has been set to $1 * mass^{0.75}$ where 1 is the Bergmann rule constant, and 0.75 is the Kleiber's Law constant. 1 was chosen as the constant as there is no generic value for all land-based mammals and a future implementation would require input from a biologist who has a better understanding for the biological aspects of this research. The activity level value is based on the creature's current speed but has a minimum value to account for the constant use of energy required to keep bodily functions active.

Implementing NEAT

At the beginning of the project, the aim was to implement the NEAT algorithm without resorting to using a third-party package. Several attempts were made to implement the NEAT algorithm, using both the original paper on NEAT as well as looking at how SharpNEAT was written. However, due to Unity using its custom Monobehaviour library as part of its engine, it does not work the same way as traditional C#. Furthermore, there is a lack of educational resources on how to implement these types of algorithms as most of the content produced for NEAT educationally is around how to use it, typically in Python as well. For this reason, the decision was made to use the UnityNEAT package.

Implementing the UnityNEAT package was relatively straight forward. The two main classes to incorporate are the 'Optimizer' class and the 'UnitController' class. The optimizer class is attached to a GameObject in the scene and acts as a spawner for the Unit that is being trained. The optimizer class four variables that needed to be assigned, the number of trials, the trial duration, the stopping fitness and the unit that needs to be trained. Additionally, within the script there integer variables that needs to be assigned that represent the number of inputs and the number of outputs the neural network. Furthermore, one additional variable was added which was just a string to control the file name to be based upon the terrain that the network was generated on.

The number of trials was set to one due to performance reasons, as various values were tested the performance quickly got worse as the number of trials increased. The trial duration was set to five hundred, as the day cycle duration was about fifty seconds so one trial would last approximately ten days, which seemed like a fair trial time as a creature that is able to survive ten days in a basic environment. And finally, the stopping fitness was set to ten, this value was chosen as a result of testing different values. When the metabolic rate was implemented, the creatures were starving to death almost instantly and the stopping fitness was at fifty. So different values were tested to see what would be appropriate so the AI could learn and adapt instead of immediately dying. Unfortunately, there is little to no resources that discuss how to determine an appropriate stopping function, which means that this is something that just has to be constantly tested until a reasonable result is found.

The UnitController script simply has to be inherited by a class to function. It has three methods that needs to be overridden. The methods simply tell the NEAT algorithm when the agent has started and stopped as well as returning the agents fitness. The agent's fitness is calculated by the number of days the creature has been alive plus a portion of the units energy when the method was called. The result is then multiplied by a weight that is set by whether the creature was dead or alive when the method is called, this is ideally to promote the creature surviving.

Changes Made

Throughout the development process a number of changes had to be made in order to complete the product as well as achieve the result.

One of the main changes was made was swapping the input and outputs of the neural network. The inputs to the neural network used to be the genetic traits of the creature, such as its height, weight, and movement speed. However, this would have required to height and weight to have been changed each generation and trying figure out how to change these factors based on the environmental conditions, led to realising the better way to handle the inputs would be to have the decision weights and the creatures energy and metabolic rate as the inputs, and then having the outputs be the creatures height, weight and speed variables as the output so it would try and produce values for the creatures physicals stats that would be appropriate to the environment.

Another main change was that the original intention was to have both the predator and prey creatures learn, however the UnityNEAT package does not seem to support training multiple units at once, so the decision was made to just train the prey creatures as they had to both eat food to survive as well as avoiding the predators that are trying to eat them. This is something that needs revisiting in a future project as the predators were not adapting alongside the creatures which is not how adaptation works in nature, so working on the NEAT algorithm to allow for training more than one unit seems very important.

Furthermore, the genes for each creature were originally unique to each unit, however this caused problems as the NEAT package generates new units each generation, so when a new generation would be produced, that creature would then just generate its own genes again, nullifying anything learnt by the previous generation. To manage this, a GeneController class was produced that was a singleton static class that kept reference to the genes for the prey species. This allowed the genes to be altered when the creature died, and then when the optimizer spawned the new generation, the new creatures would call the GeneController class to get their genes and would then apply them before beginning to try and survive the environment.

As previously mentioned in the literature review, one of the original intents was to use procedurally generated creatures. However, there were attempts made at implementing this technology in Unity, but due to the complexity of that form of work from experimentation and researching the topic, I felt that the best decision was to not implement the procedural creatures until the end of the project if there was time to develop it. A future revisit to this project would either include the procedural creatures or would go more into specific creatures to create natural behaviours in specific species.

Additionally, the original idea behind the day and night cycle was to attempt at evolving nocturnal creatures. However, through biological research it was hard to come up with a way of being able to achieve this effect naturally as the research just hypothesises suggesting that nocturnal evolution dates back to prehistoric times where creatures adapted to survive dinosaurs (Gerkema, Davies, Foster, Menaker, & Hut, 2013), so because there seem to just be theories, it is hard to implement a mechanic for creatures to adapt to being nocturnal on their own naturally.

Evaluation of Neural Networks

The neural networks produced by the three terrains are shown by appendix one, two and three. These networks represent the best creature produced by the NEAT algorithm. In each XML file, nodes one to four are input where node one is the creatures fleeWeight, node two is the creatures hungerWeight, node three is the creatures metabolicRate and node four is the creature’s energy level. While nodes five to eight are the outputs where node five sets the height, node six sets the mass, node seven sets the speed and node eight is the creatures visionRange. The vision element of this project has not been discussed during this project review, as the code is not very complex and is not as relevant as the other classes to this project. It uses Unity’s Physics.OverlapSphere method to find anything within the visionRange. If there are objects that are within the sphere, the angle between the creatures forward vector and the targets position is checked to see if it falls in a range. If the objects is within range and within view angle, a ray is fired out to the targets position to test if there is an object in the way that would result in the target not being visible to the creature.

All three neural networks ended with roughly the same fitness at around a value of six. However, the forest scene did produce a fitness closer to seven, however this value still may be within margin of error, so it is hard to say if the algorithm performed any better in that environment.

One result that was not expected was the mountain terrain produced a simpler network than the flatland did. I was expecting the neural network for the flatland to be the simplest as there were no obstructions or difficult terrain to navigate. The forest scene produced the most complex network which was predicted as the scene was the most complex due to all of the trees that blocked version and pathing restricting access to food as well as paths to use to flee from the predators.

Overall, it seems that the different terrains produced different neural networks, but with them having very similar fitnesses, that would suggest they had similar genetic values. So there is some proof that the terrains did affect the evolution of creatures but did not affect the final outcome, which makes sense in some applications as there are animals that exist in multiple different ecosystems so they would have adapted different to each environment but would have had the same result roughly overall as a species. I predict that if this project were to be continued, with additional work such as more complex environments with heatmaps and procedural creatures, that there could be significant results showing different evolutions in different terrain settings. And this could be used in game development in games similar to No Mans Sky that have huge ecosystems with an assortment of creatures.

Evaluation of the Project Process

I learnt a lot from this project process. Reflecting on my attitude at the beginning of this project, I severely underestimated how complicated some of the tasks would be to implement for the project. For example, I aimed to have implemented to NEAT algorithm myself, once I had done research and found that the task that I was trying to do as part of my project, was nearly Daniel Jallov's entire master's thesis. I still attempted to implement NEAT knowing that this was going to be a difficult task, but ultimately, I had to resort to using the UnityNEAT package.

Another aspect of this project that I learnt a lot from was the biological overlap. I feel that the overlap made me look too deep into biological research, using up research and development time as well as leading me to try and implement ideas that were beyond my understanding to try and keep the project as realistic as possible which was my initial goal. So being able to understand what research is relevant in a computer science context is something that I need to learn as it would lead to projects in the future getting too complicated and trying to solve too many problems at once. Overall, I would say that biology research helped me understand concepts that affect evolution but being able to isolate that information and using it to create suitable methods for the project, is a nuance that I need to work on.

Also, as a games programming student, having the opportunity to learn more about AI outside of a games development stand point was very interesting. Learning how AI developers were using some of the same tools, such as Unity, made me think differently about how AI and games development tools could be utilized outside of just making games.

Conclusion

In conclusion, I believe that the project has been a mixed success. The different neural networks show that the environment caused the creatures to evolve in different ways. However, those differences caused only small differences in the resulting creatures. This conclusion makes sense as evolution works like this in the real world where animals have adapted to live in a variety of climates, so the ecosystem did affect how they evolved, but how these adaptations presented themselves was somewhat insignificant. Furthermore, I think that the simplicity of the product and the model itself is a limiting factor to the result, so if this project were to be revisited with more complex procedural creatures, more complex behaviours and more complex environments to more accurately represent nature in the real world, I think there would be greater results produced by the NEAT algorithm demonstrating how a terrain affects the evolution of a creature. In regards the game development issue, I think a project like this could be utilized but it would be in very niche games that have these types of conditions where there are a variety of different ecosystems with different species in every area, which only one major game has really met this criterion in recent memory, which was No Man's Sky. A lot of games simply use the same models and the same simple AI to control their wildlife across the entire landscape, so a neural network that produces only slight differences is not worth the additional time to add this type of AI over the pre-existing techniques. Which means that this product has more usability in biology research than games development. However, having carried out this research and seeing the developments of packages like ML-Agents for Unity, I still believe that machine learning has a role to play in games development, but more exploration in this field is required before its use case is solved.

References

Benchoff, B. (2015, June 14). *Neural networks and MarI/O*. Retrieved from HackADay: https://hackaday.com/2015/06/14/neural-networks-and-mario/

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., . . . Zhang, S. (2021, March 10). *Dota 2 with Large Scale Deep Reinforcement Learning.* Retrieved from OpenAI: https://arxiv.org/pdf/1912.06680.pdf

Bonzom, T. (2020, July 15). *An Introduction To Unity's ML-Agents*. Retrieved from GameDev Academy: https://gamedevacademy.org/unity-ml-agents-tutorial/

Clarey, K. (2020, February 25). *Interest in machine learning and AI up, though slowing, one platform reports*. Retrieved from HRDive: https://www.hrdive.com/news/interest-in-machine-learning-and-ai-up-though-slowing-one-platform-report/572947/

Datadas. (2014, October 9). *Skyrim Mountain Range*. Retrieved from Nexus Mods: https://www.nexusmods.com/skyrim/images/463046

Duffy, K. (2021, Feb 8). *Elon Musk says SpaceX will get humans to Mars in 2026*. Retrieved from Business Insider: https://www.businessinsider.co.za/elon-musk-spacex-starship-humans-mars-mission-2026-experts-question-2021-2

Electronic Arts. (2021). *SEED*. Retrieved from EA: https://www.ea.com/seed

Gerkema, M. P., Davies, W. I., Foster, R. G., Menaker, M., & Hut, R. A. (2013, August 22). *The nocturnal bottleneck and the evolution of activity patterns in mammals*. Retrieved from US National Library of Medicine: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3712437/

Hershey, A. (2018, April 19). *Examples of Evolutionary Adaptation*. Retrieved from Sciencing: https://sciencing.com/examples-evolutionary-adaptation-6131133.html

Huebner, D. (2017, April 2). *Indie AI Programming: From behaviour trees to utility AI*. Retrieved from Gamasutra: https://www.gamasutra.com/blogs/DavidHuebner/20170214/291420/Indie_AI_Programming_From_behaviour_trees_to_utility_AI.php

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017, April). *Imitation Learning: A Survey of Learning Models*. Retrieved from ACM Digital Library: https://dl.acm.org/doi/10.1145/3054912#

IBM Cloud Education. (2020, July 15). *Machine Learning*. Retrieved from IBM: https://www.ibm.com/cloud/learn/machine-learning

Jallov, D. (2015). *UnityNEAT*. Retrieved from Github: https://github.com/lordjesus/UnityNEAT

Madigan, J. (2012, July 03). *The Psychology of Video Game Immersion*. Retrieved from Psychology Today: https://www.psychologytoday.com/us/blog/mind-games/201207/the-psychology-video-game-immersion

Mark, D., & Dill, K. (2010). *Improving AI Decision Modeling Through Utility Theory*. Retrieved from GDCVault: https://www.gdcvault.com/play/1012410/Improving-AI-Decision-Modeling-Through%C2%A0

Myers, N. (1992). *Ecological Complexity*. Retrieved from CIESIN: http://www.ciesin.org/docs/002-109/002-109.html

Nature. (2019, April 18). *Ascent of machine learning in medicene.* Retrieved from Nature: https://www.nature.com/articles/s41563-019-0360-1.pdf

Niklas, K. J., & Kutschera, U. (2015, July 10). *Kleiber's Law: How the Fire of Life ignited debate, fueled theory, and neglected plants as model organisms*. Retrieved from US National Library of Medicine National Institutes of Health: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4622013/

Olckers, A. (2019, Hune 18). *Does machine learning have a place in game development?* Retrieved from Medium: https://medium.com/@TheGeekiestOne/does-machine-learning-have-a-place-in-game-development-b34b0352d78a

Open Source Initiative. (2021). *The MIT License*. Retrieved from Open Source Initiative: https://opensource.org/licenses/MIT

Opsive. (2021). *What is a behaviour tree?* Retrieved from Opsive: https://opsive.com/support/documentation/behavior-designer/what-is-a-behavior-tree/

Osiński, B., & Budek, K. (2018, July 5). *What is reinforcement learning? The complete guide*. Retrieved from Deep Sense AI: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

Salian, I. (2019, March 18). *Stroke of Genius: GauGAN Turns Doodles into Stunning, Photorealistic Landscapes*. Retrieved from Nvidia: https://blogs.nvidia.com/blog/2019/03/18/gaugan-photorealistic-landscapes-nvidia-research/

Samson, C. (2014, July 7). *Behavior tree for AI: How they work*. Retrieved from Gamasutra: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

ScienceDirect. (2021). *Bergmann Rule*. Retrieved from ScienceDirect: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/bergmann-rule

Simonini, T. (2020, January 30). *An Introduction to Unity ML-Agents*. Retrieved from Towards Data Science: https://towardsdatascience.com/an-introduction-to-unity-ml-agents-6238452fcf4c

Stanley, K. O., & Miikkulainen, R. (2002). *Evolving Neural Networks through.* Retrieved from MIT Press: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

Tesla. (2021, April). *AutopilotAI*. Retrieved from Tesla: https://www.tesla.com/en_GB/autopilotAI

Unity. (2021, March). *Can I make a commercial game with Unity Free/Personal Edition?* Retrieved from UnitySupport: https://support.unity.com/hc/en-us/articles/205253119-Can-I-make-a-commercial-game-with-Unity-Free-Personal-Edition-

Unity. (2021). *Visual Scripting*. Retrieved from Unity: https://unity.com/products/unity-visual-scripting

White, C. R., & Seymour, R. S. (2003, April 1). *Mammalian basal metabolic rate is proportional to body mass2/3*. Retrieved from Proceedings of the National Academy of Sciences of the United States of America: https://www.pnas.org/content/100/7/4046

Wilson, P., & Mantooth, A. (2013). *Model-Based Engineering for Complex Electronic Systems*. Retrieved from ScienceDirect: https://www.sciencedirect.com/book/9780123850850/model-based-engineering-for-complex-electronic-systems

Zelda Wiki. (2021). *Forest of Spirits*. Retrieved from Zelda Wiki: https://zelda.fandom.com/wiki/Forest_of_Spirits

Appendix 1 – Flatland Champion XML

```
  <?xml version="1.0" encoding="UTF-8"?>
- <Root>
  - <ActivationFunctions>
        <Fn prob="1" name="SteepenedSigmoid" id="0"/>
    </ActivationFunctions>
  - <Networks>
    - <Network id="30119" fitness="6.4003868103027344" birthGen="753">
      - <Nodes>
            <Node id="0" type="bias"/>
            <Node id="1" type="in"/>
            <Node id="2" type="in"/>
            <Node id="3" type="in"/>
            <Node id="4" type="in"/>
            <Node id="5" type="out"/>
            <Node id="6" type="out"/>
            <Node id="7" type="out"/>
            <Node id="8" type="out"/>
        </Nodes>
      - <Connections>
            <Con id="34" wght="3.6302389157935977" tgt="5" src="0"/>
            <Con id="35" wght="-2.0891293740233592" tgt="7" src="1"/>
            <Con id="38" wght="-5" tgt="6" src="2"/>
            <Con id="39" wght="-5" tgt="6" src="1"/>
        </Connections>
      </Network>
    </Networks>
  </Root>
```

Appendix 2 – Forest Champion XML

```
 <?xml version="1.0" encoding="UTF-8"?>
- <Root>
  - <ActivationFunctions>
        <Fn prob="1" name="SteepenedSigmoid" id="0"/>
    </ActivationFunctions>
  - <Networks>
    - <Network id="21009" fitness="6.7476191520690918" birthGen="526">
      - <Nodes>
            <Node id="0" type="bias"/>
            <Node id="1" type="in"/>
            <Node id="2" type="in"/>
            <Node id="3" type="in"/>
            <Node id="4" type="in"/>
            <Node id="5" type="out"/>
            <Node id="6" type="out"/>
            <Node id="7" type="out"/>
            <Node id="8" type="out"/>
            <Node id="178" type="hid"/>
            <Node id="205" type="hid"/>
            <Node id="266" type="hid"/>
            <Node id="276" type="hid"/>
            <Node id="300" type="hid"/>
        </Nodes>
      - <Connections>
            <Con id="35" wght="3.01755916679973" tgt="5" src="1"/>
            <Con id="47" wght="-3.9977268803903288" tgt="7" src="3"/>
            <Con id="48" wght="2.5195393481349129" tgt="5" src="0"/>
            <Con id="180" wght="3.1712647488459647" tgt="5" src="178"/>
            <Con id="197" wght="-4.21987455381954" tgt="6" src="178"/>
            <Con id="206" wght="-1.3764380773510521" tgt="205" src="178"/>
            <Con id="254" wght="3.4890932489988473" tgt="178" src="2"/>
            <Con id="267" wght="2.7600921661199713" tgt="266" src="178"/>
            <Con id="301" wght="4.4021585137719246" tgt="300" src="205"/>
            <Con id="302" wght="4.9836969804310547" tgt="276" src="300"/>
            <Con id="325" wght="2.8945988789200783" tgt="8" src="266"/>
        </Connections>
      </Network>
    </Networks>
 </Root>
```

Appendix 3 – Mountain Champion XML

```
  <?xml version="1.0" encoding="UTF-8"?>
- <Root>
  - <ActivationFunctions>
      <Fn prob="1" name="SteepenedSigmoid" id="0"/>
    </ActivationFunctions>
  - <Networks>
    - <Network id="26620" fitness="6.1769247055053711" birthGen="666">
      - <Nodes>
          <Node id="0" type="bias"/>
          <Node id="1" type="in"/>
          <Node id="2" type="in"/>
          <Node id="3" type="in"/>
          <Node id="4" type="in"/>
          <Node id="5" type="out"/>
          <Node id="6" type="out"/>
          <Node id="7" type="out"/>
          <Node id="8" type="out"/>
        </Nodes>
      - <Connections>
          <Con id="31" wght="-5" tgt="7" src="3"/>
        </Connections>
      </Network>
    </Networks>
  </Root>
```