UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Artificial Neural Networks

Technology & Adaptation to a Unity3D toolset

Treball Final de Grau
Grau en Desenvolupament de Videojocs

Surnames: Olivé Rubio
Name: Pau
Pla: Pla d'estudis 2014.
Director: Abdal, Sergi

Index

Summary

Artificial Intelligence (AI) has exploded in the last decade due to the increasing computation capabilities modern computers have achieved and the expanding availability of data. AI has had a huge impact in many fields and game play is one of them. Video games is a sector thriving for realism and new, inventive game challenges, two aspects AI can help to improve significantly.

Even though the use of Neural Networks in the video game industry has been reserved to specialized programmers and studios, we already see the emergence and use of open tools such as TensorFlow (*Ref.1*) which have been proven to be useful in some areas such as data analysis, eSport image processing, or even financial forecasting (*Ref.2, Ref.3*). However, these tools are still generic, lacking integration into game developing systems and limiting accessibility and adoption to the common game developer.

In this thesis, we will address this issue by covering not only the creation of an Artificial Neural Network (ANN) in the Unity Engine, but also the development of a Unity toolset that enhances the accessibility of said technologies to the common user. The use of the Unity Engine will later foster adoption of the technology by the non-expert as it is of the most popular and most accessible engines in the videogame industry in recent years.

This project will propose the creation of a Unity Tool that allows an easy and quick creation, edition and management of several ANNs, as well as its introduction to Agents and the subsequent training of said networks on user-defined behaviours. Also, as a testing tool, we will create a validation mini-game.

At the end, the project did show great results and utilities. And, even if it was lacking in some aspects, it was, for its initial proposed scope, a great success.

Key Words

- Neural Network
- Machine Learning
- Deep Reinforcement Learning
- Artificial Intelligence
- A.I.
- Training
- Learning
- Adaptative
- Unity
- Toolset
- C++
- C#

Links to the Project

Github Repository:
https://github.com/rcpauor32/ANNProject
Github Release:
https://github.com/rcpauor32/ANNProject/releases/tag/v1.0

Index of Tables

Index of Figures

Glossary

- **Artificial Neural Network (ANN):**
  Computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because it changes (or learns, in a sense) based on that input and output. ANNs are considered non-linear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found (*Ref. 48*).
- **Deep Reinforcement Learning:**
  Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps; for example, maximize the points won in a game over many moves. Reinforcement algorithms are penalized when they make the wrong decision and rewarded when they make the right ones (*Ref. 49*).
- **Machine Learning:**
  Machine Learning is part of research on artificial intelligence, seeking to provide knowledge to computers through data, observation and interacting with the world. That acquired knowledge allows computers to correctly generalize to new settings (*Ref. 50*).
- **Neural Network:**
  Name this project gives to define the Artificial Neural Network class.
- **Neural Node:**
  Name this project gives to define the Nodes or Neurons inside the ANNs.
- **Thought Process:**
  Name this project gives to a set of values extracted from the network. It contains the fitness and each bias and weight values from nodes and connections.
- **Agent:**
  An agent is an entity using an ANN to perform its tasks. Even if many agents use the same network, its values may and will be different.
- **Generation (ANN related):**
  A generation is a distinction between the network's evolution. When a generation changes it signifies the values have been updated taking into account previous results.
- **Parent (Agent / Network):**
  A parent is a set of results used to create new agents. On ANNs they are often used in pairs to create new values mixing both of them.
- **Offspring (Agent / Network):**
  An offspring is an agent created from the mixing of two agents from the previous generation. It contains values from both and is mutated to add some randomness to the process in order to achieve new results.

1. Introduction

1.1. Motivation

Artificial Intelligence (AI) has been expanding and revolutionary technology since it was first envisioned. It started on a theory with the objective to simulate and create a sense of intelligence. However, this type of AI was nothing more than 'tricks' to mimic traits of that intelligence. Since then, AI has evolved at an alarming rate with more and more capable machines and more and more uses and solutions for this technology, to a point we can only imagine what this technology could become in the future (*Ref.32*).

However, as it is right now the newest step is the improvement of the so-called Artificial Neural Network, a technology that has already showed great potential put to use. One can argue that this technology is, still, a set of complex algorithms that simulate the learning process. However, after seeing the results and processes of ANNs, one can easily see why it is deemed as a step towards true intelligence. The way it is inspired from our own brain structure and how it replicates the actual processes of learning from animals and humans (even if it is in a simpler way) creates a prediction for the future that was, until now, just possible on the Sci-fi genre.

However, making an ANN is not the whole challenge. A quick research can be done to notice an important issue on ANNs as they are right now. Even though there are lots of information about it, ANNs are still novel and there is not much done to bring this technology closer to people with less knowledge or that are interested on the topic and want to use ANNs for their purposes, but are not willing to make a huge effort to fully understand the theory and specifics behind ANNs. There are attempts to do so, but at the end, they are targeted to experienced programmers that are used to look for a great amount of documentation (*Ref.33-36*).

In order to solve this issue, a tool could be created to help people introduce ANNs to their own projects with as little effort as possible. This is what this project is based on.

1.2. Formulating the Issue

Artificial Neural Networks, and by extension the learning systems that make use of them (e.g. Deep Reinforcement Learning) are, nowadays, being introduced little by little into our society and culture as any other technology advance, the next and logical evolution of what we can create.

However, it is a complex technology that cannot be created or used by someone without great knowledge and programming experience.
This is exactly the issue we want to tackle in the context of video game development. It is true that some tools have been created to help developers put ANNs into their projects but they are either too specific, with too little customization or require a lot of programming knowledge to manage (*Ref.33-36*).

1.3. General Objectives

This project's aim is to develop an accessible toolset for the game engine Unity3D (which is one of the most popular and accessible game engines in the market) that helps developers to implement Artificial Neural Networks into their own projects with as little effort as possible. To do so, we have the following objectives:

Objectives:

1. To develop a code structure to handle Artificial Neural Networks (Network, Nodes, Connections, ...) and their training methods.
2. To allow any user to set up a customizable Artificial Neural Network.
3. To allow any user to create their own Output Behaviours and insert their Input Values.
4. To create a Unity3D toolset that allows a quick setup.
5. To create a small mini-game to test the effectivity of the project.

At the end of the project, the tool created should be:

- **Configurable:** The tool should allow any user to insert their own inputs and use their own outputs as well as be able to define and edit any network that has created.
- **Optimized:** The tool should be able to run 'in-game' consuming as little resources as possible, at least, during the 'Playing' phase (value won't be 'training').
- **Functional:** The tool should be able to provide the user with a functional ANN that trains agents.
- **Universal:** The tool should be able to be used for any project (or pretty much any project), independently from its sector or genre (if in video games).

1.4. Specific Objectives

As for the specific objectives of the project, there are several milestones that need to be properly implemented to be able to achieve the desired results:

Network Structure

- **Neural Network Class & Methods:** Creation of a C# Class that provides both a structure and framework to hold Neural Nodes & Connections as well as the pertinent Class Methods to perform the base ANNs functionalities.
- **Neural Node Class & Methods:** Creation of a Class that provides a sub-structure for the Neural Network. It will hold its different connections, as well as its layer value and the behaviours and algorithms.
- **Thought Process Class & Methods:** Creation of a Class that can store the different values provided by the nodes of the network. They are used in the training process.
- **Network Manager:** A Manager that acts as the main application of the toolset. It provides the more general data and methods such as holding all the created Networks.

Training System

- **Reinforcement Algorithm(s):** Algorithms aimed to tweak the Neural Nodes values in order to approach them to the desired behaviours depending on the input the network receives.
- **Creation of Agents Generations:** In order to train the network, several agents must keep spawning and use the Reinforcement Algorithms to create new values for its nodes.
- **New Thought Process Generation:** From the used Thought Processes we need to provide the new Generations of Agents with new values. However this values, far from random, must be generated using specific algorithms and the previous, more successful values.

Unity Tool Editor / User Interface

- **Neural Network UI:** Development of an interface to manage and edit any network created with the tool.
- **Neural Node UI:** Development of an interface to manage and edit the different nodes inside a network, independently.
- **Network Manager UI:** Creation of a UI to manage the high level aspects of the tool, such as the creation of a new network or to manage them.
- **Training UI:** Implementation of a UI to manage the training of the network.
- **Unity Inspector custom UI:** Creation of a custom inspector for Unity, since the default inspector provided by the engine is lacking some options this tool will need.

Others:

- **Validation Mini-Game:** The development of a minigame to test if the tool is working accordingly to the objective specified above [1.3. General Objectives].

1.5. Project Scope

The use of Artificial Neural Networks is a practice that has been growing and evolving during the last years. It is a tool mostly used on technological projects such as face or speech recognition and generation (*Ref.4*), and it is slowly transpiring into video games, on its search for more realistic artificial intelligences. It is through this sector that indie developers are appearing with an interest to create their own ANNs (*Ref.5*). However, this creates an issue, Indie developers are often short on time or / and resources and, in many cases, lacking the knowledge necessary to create a proper ANN for their projects.

This is who the project will be targeting.

Indie developers can benefit from the project by providing them a fast, easy tool to incorporate AIs in their games or applications that learn from their player or any other element they choose to. The tool will be able to be used in video games to add several features such as adjusting game difficulty or creating an enemy AI that learns from the players' actions.

Moreover, being a tool, developed in Unity, one of the most used, most accessible Game Engines in the indie community (and even on the triple A community) will grant a quick access to our product.

2. State of the Art

2.1. Artificial Neural Networks

Artificial Neural Networks are an area of Artificial Intelligence technology that has been around for some time, however, nowadays has attracted a great amount of attention and has exploded in use and advances. Neural Networks are inspired on the operation of the biological neurons in brains, and aim to replicate such behaviour into virtual networks of nodes and connections.

An ANN consists of a set of neurons and a set of connections between the nodes. The connections account for a given weight, so that a given set of inputs ends up translated into a set of outputs. The weight of the connections is assigned through an iterative process where the outputs for a given set of inputs are compared with the desired outputs (training phase). The difference between the obtained and expected outputs is used to correct the weights. In an interactive system, the *inputs* would be the data that is received by the network and *outputs* or *results* would be the different values the network would calculate through its structure, resulting in the different possibilities it has defined by the creator. Using this method, certain results are stimulated and more likely to be chosen gradually, granting the machines a process similar to the learning process (*Ref.8*).

In recent times, ANNs have become more and more popular, since its potential to perform incredibly complex tasks has been proven to be an outstanding feat. For example, the famous Apple's *Face ID* facial recognition technology integrated into mobile phones that teaches the device to recognize a face even from different perspective, a feat really common for human beings but incredibly difficult, and impossible until now, for machines (*Ref.6*).

Other examples are the self-driven cars, now being developed with this technology, using the ANN to recognize each element in front of the car, as well as other data such as the distance, to perform the best action possible in each case (*Ref.9, Ref.10*).

But it is not only used in these industrial environments, for example, ANNs have been used to analyse languages and ways to express them to be able to create whole books (*Ref. 11, Ref. 12*).

ANNs are a unique technology in the sense that are able to be used in a lot of areas, from technology to social sciences can benefit from it. It can create accurate, complex models with ease and has been found out to be really useful on problem-solving (*Ref.8*).

Example of many fields of applications of ANNs	Prediction	Pattern recognition	Classification	Total
Security	20	18	2	40
Science	25	25	2	52
Engineering	22	7	2	31
Medical science	10	5	2	17
Agriculture	3	3	2	7
Finance	10	15	2	27
Bank	5	15	2	22
Weather and climate	2	15	2	19
Education	30	15	2	47
Environmental	10	15	2	27
Energy	5	15	2	22
Mining	2	15	2	19
Policy	2	2	2	6
Insurance	5	4	2	11
Marketing	5	5	2	12
Management	40	2	2	44
Manufacturing	12	15	5	32
Other fields	52	11	10	71

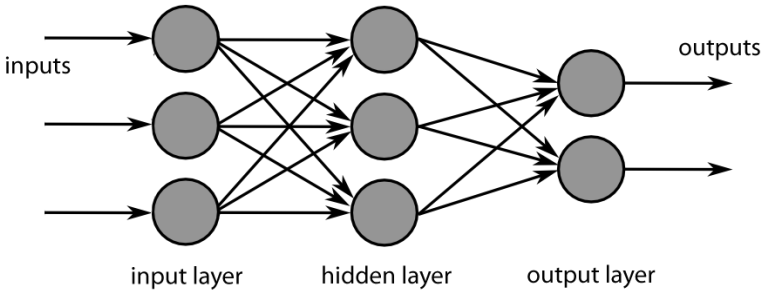
T2.A Comparison of different use areas of ANNs (from *Ref.8*)

2.2. Network Structure

There is an infinite amount of ways to arrange a Network. Each one has its *pros* and *cons*. In this section we are going to explore some of the most used for Neural Networks.

2.2.1. Feed Forward Structure (FF)

This structure is based on a series of layers of nodes, arranged in a way where there is no cycle structure, the network's flow goes from *inputs* to *outputs* (forward). It is one of the most, if not the most, used structure for ANNs (*Ref.13, Ref.46*).

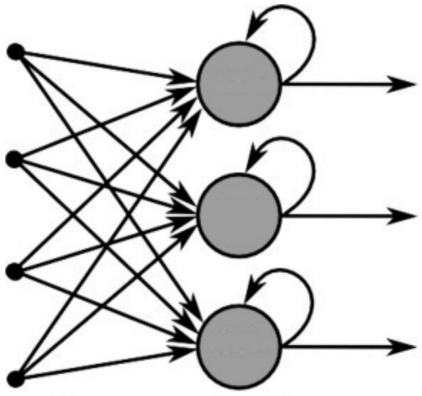


F2.A Feed Forward structure graph (from *Ref.16*)

There is a specific type of FF network that is called Fully Connected, where all nodes are connected to all nodes of the next layer (the one seen in the image above). It is the more used in ANNs since all nodes are connected and these connections are gradually strengthened or weakened by the learning process. This means that the ANN does not make any pre-assumption on the scenario and allows the training process to shape the neural network. For this reason, it is also the most resource-consuming (*Ref.13, Ref.30*).

2.2.2. Recurrent Structure

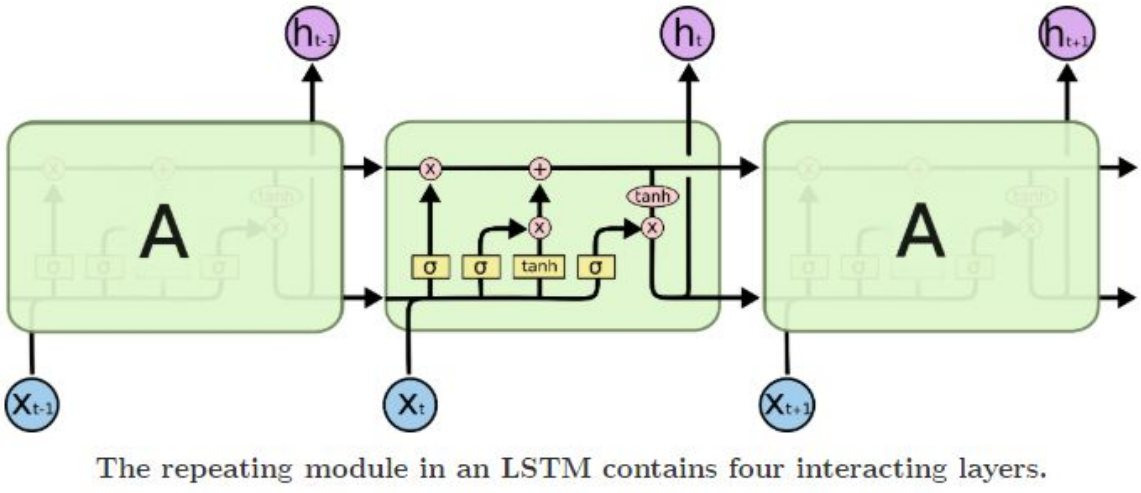
Recurrent structures allow nodes to have a temporal iterative behaviour inside a single node. What this means is that, instead of going forward, the flow can be redirected onto the same node again. This is useful to avoid using a great number of layers where the algorithm would be the same as for the last layer of nodes. It is widely used in problems where we want to learn temporal relationships (*Ref.14, Ref.15*).



F2.B Recurrent structure graph (from *Ref.16*)

2.2.3. Long Short-Term Memory Structure (LSTM)

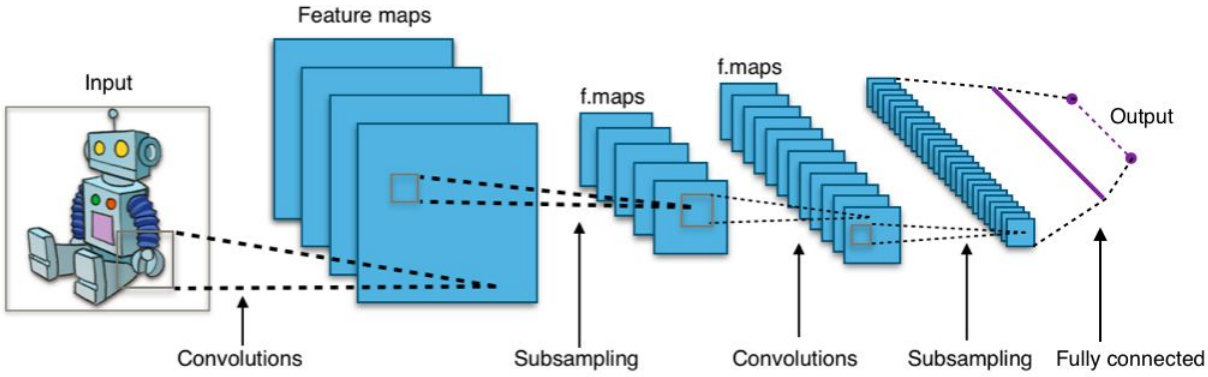
LSTM networks expand on the concept of the Recurrent structure. They work on the theory that, inside the recurrent cycles, some data from previous iterations would be needed on future iterations, for example, some data from the iteration 'it01' could be needed on the iteration 'it07' but, in a Recurrent structure the only valid data would be that of the 'it06'.
LSTM is a really complex system that incorporates an element of 'memory' to the network by making the different nodes interact in multiple ways instead of the typical 'one connection', each one with different algorithms (*Ref.15-17*).



F2.C LSTM graph (from *Ref.17*)

2.2.4. Convolutional Neural Network (CNN)

Convolutional Neural Networks are an attempt to mimic the functionality of the animal's visual cortex. This is a really complex structure where each node is in charge of a part of the vision field. This field gets smaller and smaller for each layer deep until it is just a pixel (a value) for the network to work on. It is used to replicate the image processing of biological beings and can be used on the application of filters and other image processing techniques (*Ref.29, Ref.47*). In summary, CNNs work well in cases where we want to learn spatial relationships (it is usually the case in image processing). (*Ref.47*)



F2.D CNN graph (from *Ref.47*)

2.3. Training Systems

There are several ways and techniques on how to train a Neural Network. In this list we outline some of the most used systems, their pros and their cons.

2.3.1. Supervised

Supervised training is one of the most common training systems. Its concept is simple (and pretty self-explanatory): It requires from a human supervisor who observes at the networks outputs and tells it, 'manually', how correct or how wrong it was on its predictions.

Pros:	The supervised nature of the system makes creates a more accurate system in less iterations and it is really useful for some problems such as Letters / Numbers recognition since it is comprehending human-made symbols.
Cons:	It is a really slow system compared to others, with much more less iterations with the same time. It is not useful for really complex task were a human would not be able to predict or have the correct answer beforehand.

2.3.2. Unsupervised

Unsupervised training is the exact opposite of the Supervised one. It lacks a human supervisor and the Neural Network is left *alone* to train itself. The network calculates its fitness through a series of probabilities and data analysis techniques and keeps doing this process automatically.

Pros:	It is a far faster system than supervised learning and has a higher adaptability to the data rapidly changing, being able to predict new results through its analysis.
Cons:	The unsupervised nature of the system makes its results unpredictable and it could evolve into undesired results. This is made worse with complex processes where, since humans would be unable

	to know if the results make sense, the machine could be giving wrong answers without the users knowing.

2.3.3. Reinforcement

Reinforcement training is sort of a mixture from the two previous systems. It does not supervise the network all the time nor it gives it full freedom. Reinforcement training lets the networks work by itself and automatically, however, when the fitness has to be calculated it gives a certain *stimulus* either negative or positive depending on the results. This means that neither is the network left to decide how appropriate was an answer by itself, neither is the 'considered wrong' path taken completely deemed wrong and cut from the network, giving the machine space to learn by itself.

Pros:	Can be used on complex matters and maintains a great part of the Unsupervised model speed. It has also a great degree of adaptability and at the same time its behaviour can be influenced by humans if it is desired / needed.
Cons:	It still requires a 'formula' given by humans. This means that even though it has much more space than the Supervised training, it still needs a certain degree of result prediction by the creator or at least has to know what would be considered *more correct* or *more incorrect* before hand.

2.4. ANNs in Videogames:

ANN is a pretty young technology and, as such, it is still working its way into video games where it is attractive due to its search for realism and challenge at the same time that still does not seem to fit properly. However, we already have some precedents on ANNs being developed in the sector.

Google's AlphaGo was an AI designed by the company on its research on the field. It used a Neural Network to be trained to play the boardgame 'Go', a traditional Asian strategy game (comparable to chess but way more complex in terms of the number of possible board positions, which makes it impossible to brute-force it). This AI used a variation of a Convolutional Neural Network (CNN), even though this type is used on image processing, the tree-like hierarchy of each move and its consequences benefited from this type of net.

This AI managed, with time, to beat several professional players of said game and, in 2016, finally beat the World's champion Lee Sedol. The AI was said to have done really inventive plays that surprised the champion (thereby showing capacity to infer new movements from past experience) and marked a crucial moment for Neural Networks similar to the match between Deep Blue and Kasparov in 1996 (*Ref.18, Ref.19*).

But it is not only limited to chess-like games. Elon Musk and a team of developers created the 'OpenAI' startup. It was an AI designed to learn how to play *DOTA 2* a really complex video game from the *MOBA* genre (similar to Action Real Time Strategy). It uses a really complex type of ANN based on the CNN (Convolutional Neural Network) type a useful choice since the AI is based on image treatment and processing to create its inputs and data.

In the same way as Google's AlphaGo, OpenAI beat several eSports champions with creative ways that none have thought of, for example, the AI was found to make bad decisions at the early stages of the game, playing with the enemies' self-confidence to trick them into underestimating their adversary.

Currently this AI is highly valued on eSports and used as training for highly competitive teams (*Ref.20, Vid.1*).

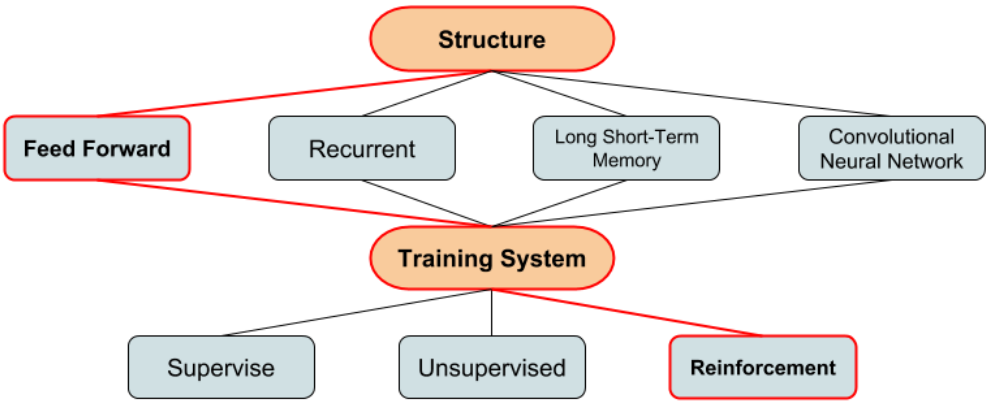
2.5. ANNs tools in Unity:

As mentioned above, video games is not a field known because of its integration of the ANN as it is right now. However, with a quick search in Unity Asset Store, we can see there are multiple projects dedicated to bring ANNs closer to developers. There are several versions, from more specialized such as digit recognition ANNs or post processing images applying ANNs to more general projects similar to our own objectives. The difference between those projects and ours is that our main objective is to create an accessible and universal ANN toolset. This means that,

while other tools could be more useful in some areas or more effective, our project will be as general as possible to be used by many people in many areas and tasks easily.
Despite of that, these projects can be taken as a source of inspiration for our own tool, observing where they succeeded and where they failed, to ensure a useful product.
It is observable that there is a clear desire to use and approach ANNs to video game developers. (*Ref. 40-42*)

2.6. State of the Art for the Project

For this project we will not need to apply the most complex, most specific state-of-the-art technology. However, the concept and core idea applied will be very similar.
For the structure we will use the Feed Forward model, specifically the Fully Connected graph type, for its popularity, generality, and usefulness in ANNs. For the Training algorithm, the Reinforcement system will be applied since it is the most adaptable to most situations, specifically on video games. In any case, the concept is applicable to any other type of training or neural network.

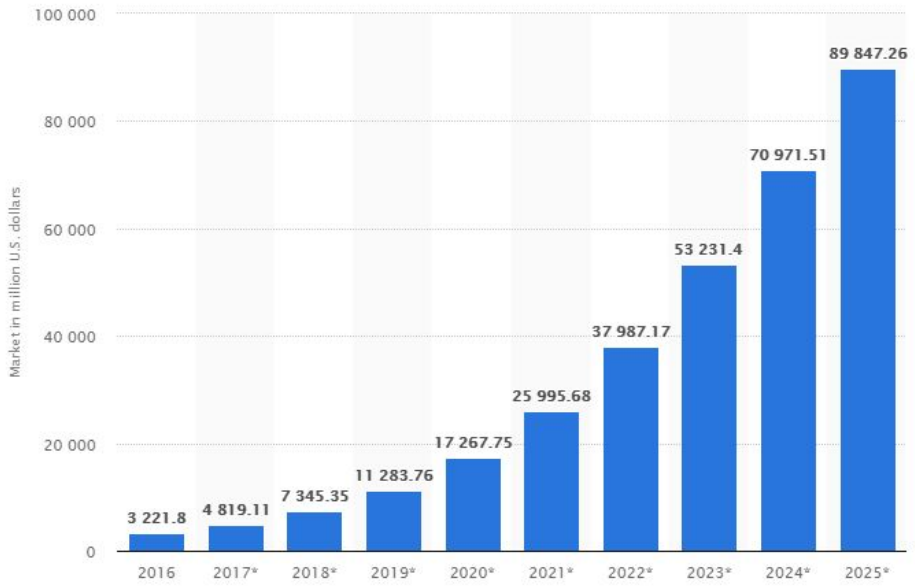


F2.E State of the Art used for the project

2.7. Market Study

To study our market, we have to focus on our target, our platform and our sector: The Indie Developer, the Unity Game Engine and the Artificial Intelligence.

We can observe in the chart below how the market for AI is not only growing, but also expected to keep growing exponentially on the next years.
As it has been discussed in the previous sections, AI is more and more present in our everyday live and more and more people and companies are finding new uses for specifically the Artificial Neural Networks' capability of learning.

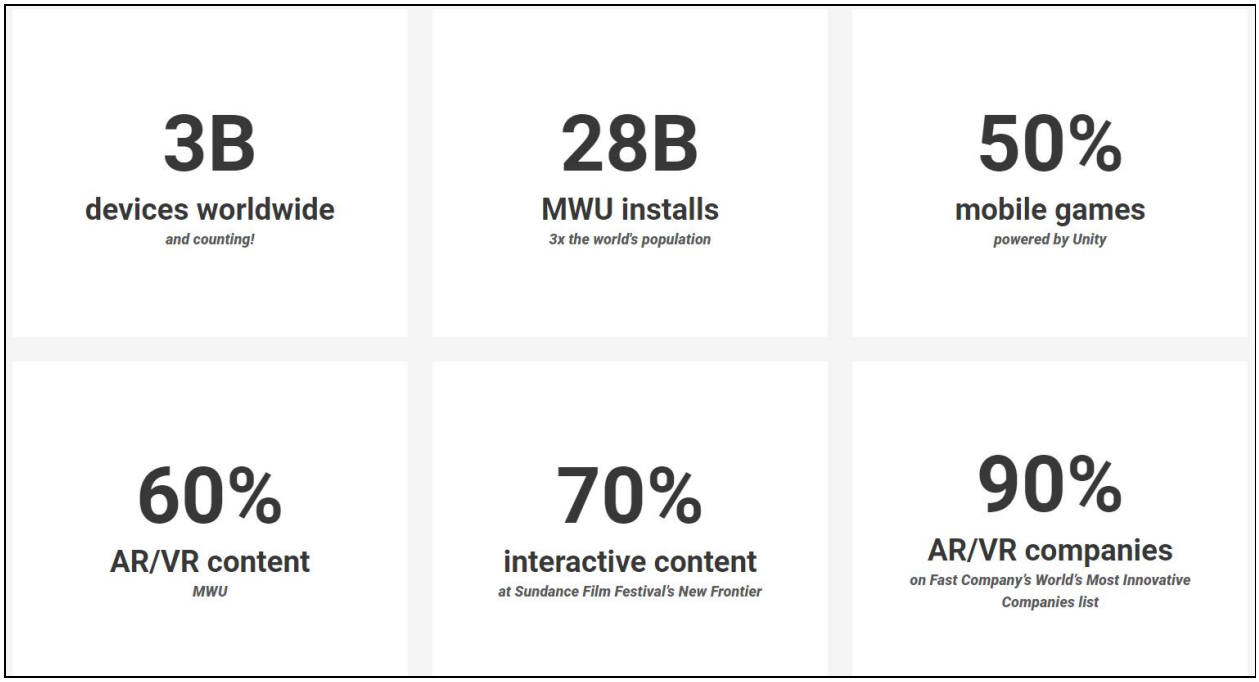


F2.F AI worldwide market growth and expectations (in millions of USD) (from *Ref.28*)

Apple, for example, has already seen the potential of ANNs and incorporated them into the so called *Apple's Neural Engine* used in the newest models of their phones and products (*Ref.6*).

So we have the *Sector* part covered. But what about the target and the platform?

Unity Engine is already the most used engine for Indie Game Developers so we can tackle both issues at the same time. Observing Unity's data about their usage we can find they are in the 28 billions (americans) of installs and operative in more than 3 billion devices. The engine is not only extended, but adapted to be used for the most common platforms, from PCs to Mobile and even on AR/VR sets. (*Ref.21*)



F2.G Unity Statistics (from *Ref.21*)

A useful data for our project is that 49% of the studios operating with Unity are 2 years or less old and 67% are fully independent. This means that indie studios are most likely in need of tools to make their processes easier and faster, such as our own tool, making them our perfect target (*Ref.22*).

So now we have all parts covered except for one: the competitors.

Unity has an incorporated eStore called *Unity Assets Store* where people can publish their tools or assets at any fee (including for free). This seems like a really good place to put our tool; however we are not the only ones that have thought of making ANN projects.

With a quick search we can find some tools that have a similar objective than the one we are creating. This is both a bad and good sign: They will most likely take a part of our potential audience, however, after making as much research as possible we can learn from what makes them popular and what makes them fail to apply that knowledge to our own product and create a highlightable tool that stands out between our competitors (*Ref.40-42*).

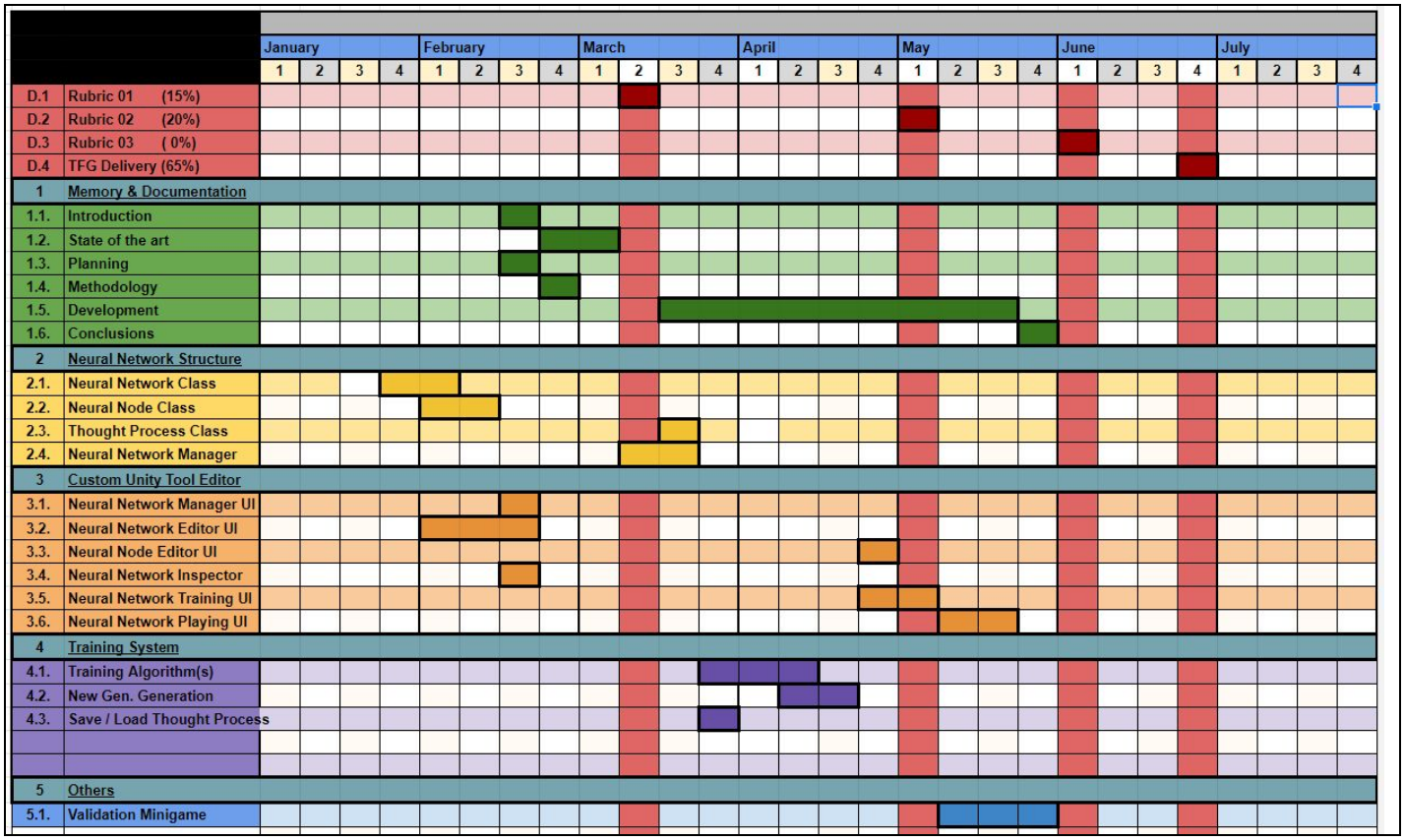
3. Project Management

3.1. Tools & Procedures to manage the project

As in any project, some tools will be needed to manage the whole process in an efficient way. The following tools, are the ones needed for this specific task:

3.1.1. Initial Planning- GANTT

The following is the initial plan for the development of the project, represented in a GANTT diagram:



F3.A. GANTT Diagram

3.1.2. Planification Revision (03/05/2019)

For the most part, the planning has been followed accurately, however, there are some aspects that needed some revision.

1. **Removing Neural Network Manager (and UI):**

   During the development of the project, several issues were faced, most of them regarding the engine used, Unity. During the investigation to solve those problems, a new kind of object was discovered in the engine which could improve both the usability and code of the project.
   The integration of this new type of object did not incur a great time cost. The time spent developing this part would later prove to be compensated by the fact that the Network Manager and its UI could be removed since its functionality (a manager for all the networks) can be now handled by the Unity Resource System instead thanks to this improvement.

2. **Extending Training system:**

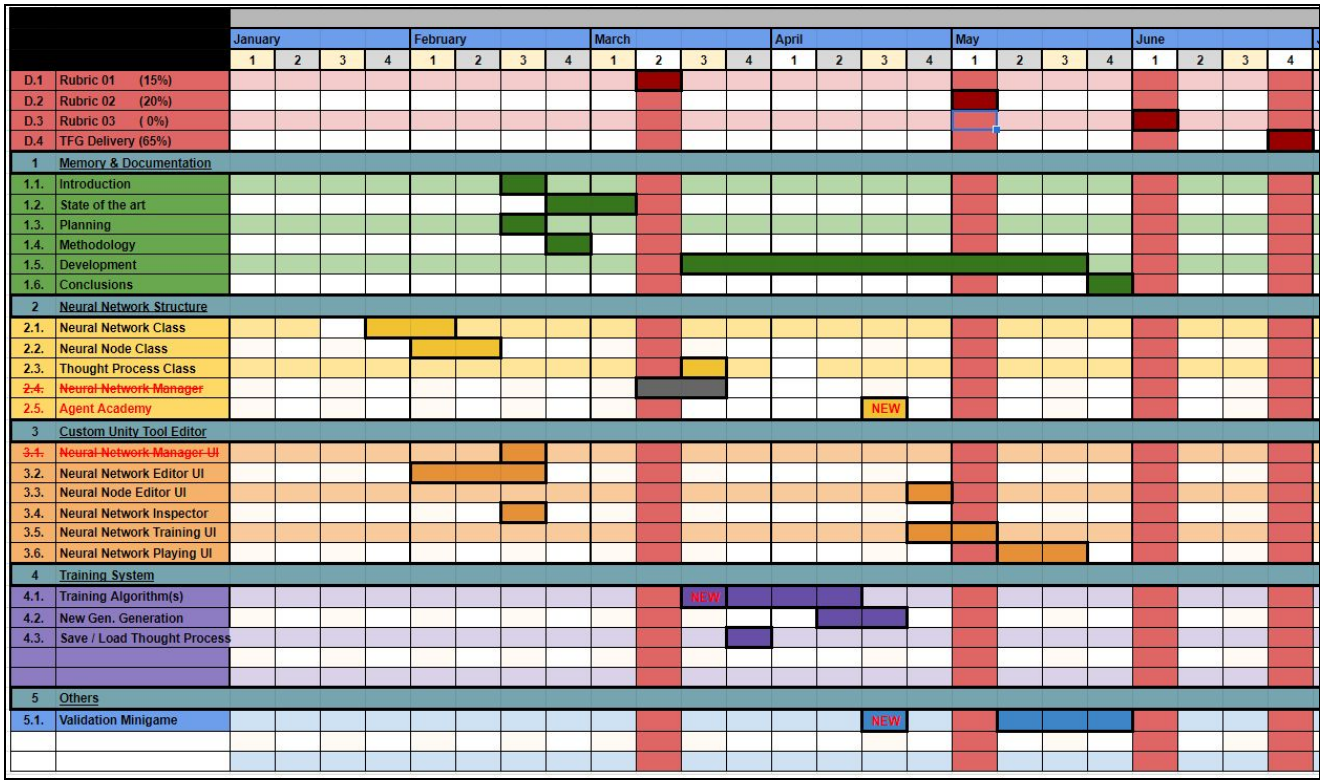
   As a consequence of Removing the Neural Network Manager from the planning, the Training System time could be extended, taking some of the time the Manager was planned to take.
   This is very useful since the Training System is the core part of the project and one of both the most complex and most time consuming parts.

3. **Adding Agent Academy:**

   When developing the training system, it was found out that a new structure needed to be created. The Agent Academy was needed so the training of the agents could be properly implemented and managed.
   The time cost of this class is minor since its functionality is really simple (replicating agents).

4. **Advancing the Validation Minigame development:**

   While developing the Training System, the learning of the agents needed to be checked. Even though the actual code structure and flow was developed, the development of a minigame was crucial for checking the actual networks' evolution.
   Due to this, the development of a minigame was advanced.
   The rest of the planning regarding this has stayed exactly the same since this first game was developed just as a very simple prototype to perform tests.



F3.A.2 GANTT Revision

3.1.2.1. State of the Project (01/05/2019):

The initial planning has been followed with slight modifications as stated above. The following states are:

- Creating the UI for the user to be able to create an ANN intuitively.
- Creation of more advanced minigames to demonstrate the ANNs capabilities.

3.1.3. Unity Cloud

Unity Cloud(*Ref.23*) is a really useful Version Control System (VCS)(*Ref.25*) tool similar to Git (*Ref.24*) to keep track of the different stages and changes in your Unity project. It will be used as the VCS tool for this project since it is already implemented in Unity.

3.1.4. Google Drive

Google Drive (*Ref.26*) is a VCS tool for writing documents and sheets. Documents can be shared to be commented , viewed or even edited by any user who has been given the proper rights. This tool will be used to create all documents and also share them with the advisor in order to review the progress in a fast, easy way, through comments.

3.2. Validation Tools

To validate the functionality of the project, a mini-game will be developed to quickly test and debug the features implemented. This mini-game will have at least one ANN Agent with a network generated from the tool and the whole training process will be analysed and/or recorded as proof of the result.

Also, to test efficiency and optimization, data will be gathered from the code to create graphs, charts and figures regarding (specially) the training process.

3.3. SWOT Analysis

Strengths	Threats
- The proposed asset is a flexible 'product' that can be used on different genres and fields (Video games, Augmented Reality, etc...). - Experience of the programmer with Unity3D and the Unity UI system.	- Great companies already working on ANN tools. - Some Unity Tools that already work on ANNs.
Opportunities	**Weaknesses**
- Not many / not really useful Unity Tools regarding ANNs. - Very wide potential adoption base. - A field with lots of information and studies.	- Not experienced on creating ANNs. - Short amount of time to develop the project.

T3.A. SWOT Analysis

3.4. Risks & Contingency Plan(s)

3.4.1. General Risks:

Risk	Plan
● **Lack of time:** It is possibly the most likely risk to actually happen. The project complexity mostly leads to a problem of time..	● The project has been planned leaving some room for unexpected issues to happen. ● Milestones in the project serve as

	guide of the state of its development. • The project can be re-dimensioned at those points according to the actual progress. Modularity helps in this regard.
• **Lack of knowledge:** Another risk is the lack of knowledge on the matter, since, as a student, I do not have any experience in ANN or DRL.	• I have chosen the Unity engine to not only have an accessible tool, but also use an engine I am experienced with. • Researched a lot of information about the ANN topic (there is a lot) to be sure to have the proper knowledge and skills to develop this project.
• **Lack of optimization:** Optimization is certainly a risk since ANNs are, usually, a CPU consuming technology.	• Researched about the use of Threads on the Unity Engine to be able to use them in case they are necessary. • Left some space on the project planning to perform optimization fixes if necessary.

T3.B. General Risks and Contingency Plans

3.4.2. Specific Tasks Risk:

Risk Task	Plan
• **Network Structure:**	**Risk Level:** Medium The network structure is constructed in an encapsulated way so, if something needs to be changed or fixed, it will not compromise the rest of the project.
• **Network Training:**	**Risk Level:** High This is not only a Core point of the project, but also the part where most issues can arise.

	Most of the time will be dedicated in this part of the project to reassure everything works fine. In fact, most of the project will be encapsulated to be as non-dependent from the code from this part in case its processes fail.
• **Custom Editor:**	**Risk Level:** Low The Editor is a key part of this tool, however most core functionalities will be developed at the same time as the network functionality to avoid its construction compromising the process. For all other functionality, they will act more as a 'Polish' phase, so, if there is no time / knowledge to develop them, they can be reduced in scope easily.
• **Validation Mini-Game:**	**Risk Level:** Low The mini-game will act as a validation tool, so, it will be developed in a really simple manner to not act as a risk factor for the project. Even in the case there is no time to create it, a quick demonstration can be done through Unity with almost no coding.

T3.C. Specific Risks and Contingency Plans

3.5. Initial Cost Analysis

Table T3.D shows the costs of the project. General and average costs and salaries of a Software Developer, which could develop tasks similar to those required in this project, have been taken into account. We considered 5 months as the total length of the project. Linear amortization has been used to extrapolate the costs of equipment.

Type	Subject	Cost	Type	Amortization(y)	Total Price
Personal	Salary	€9.60	per Hour		€2,880.00
Equipment	Desk	€100.00	Unique	4	€10.42
	Chair	€60.00	Unique	4	€6.25
	Computer	€1,000.00	Unique	3	€138.89
	Screen	€60.00	Unique	3	€8.33
	Mouse	€30.00	Unique	2	€6.25
	Keyboard	€20.00	Unique	2	€4.17
Software	Visual Studio	€0.00	Monthly		€0.00
	Unity Pro License	€125.00	Monthly		€625.00
Indirect Costs	Electricity	€30.00	Monthly		€150.00
	Water	€20.00	Monthly		€100.00
TOTAL					€3,929.31
Duration:	5	months	300	hours	

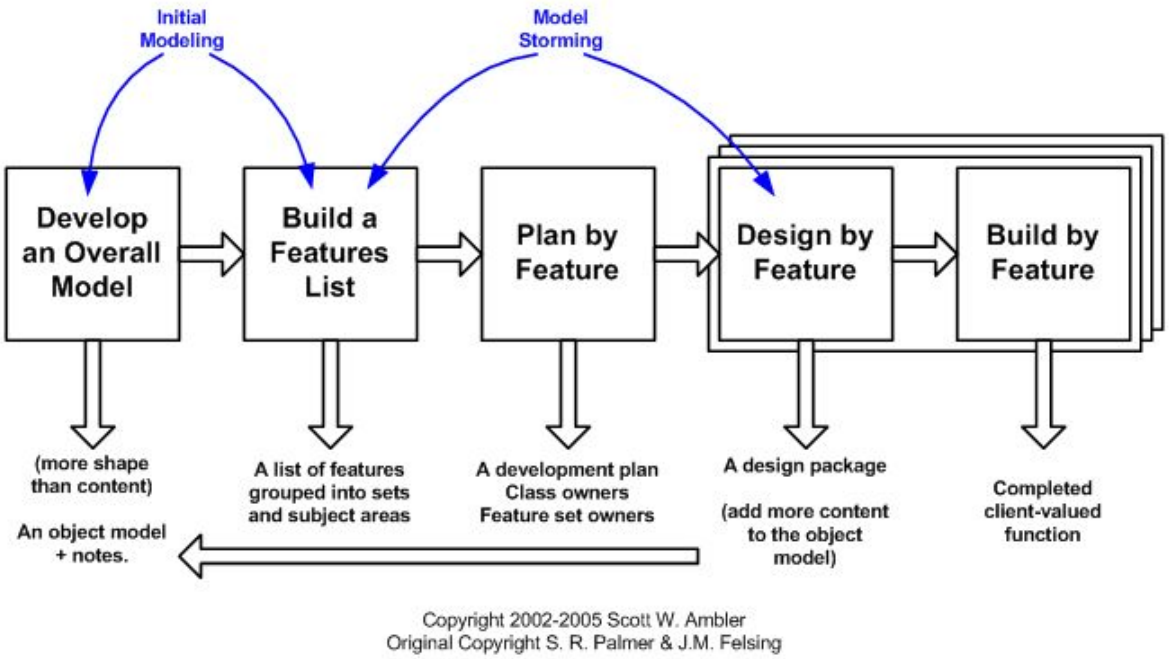
T3.D. Initial Costs Analysis

4. Methodology

As a software development project, 'Agile' is one of the most common methodologies, and, in a 'real environment' of a company the logical methodology to use would be 'Scrum'. However, since this a one-person project and Scrum is more oriented to team projects, I will use another type of methodology known as 'Feature Driven Development'.

4.1. Feature Driven Development:

Feature Driven Development (FDD) is a project methodology for software development that focuses on the design and development of features, understanding a feature as a "client-valued function" (p.e. "Create a network", "Train the network", "Edit the menu", ...). This process starts with a general model and gets more specific with each feature. This way, the software developed is always functional but has more and more functionalities until you reach the desired product (*Ref.27*).



F4.A Feature Driven Development schema (from *Ref.27*)

4.2. Applying the Methodology:

Following this project, the first phase will be something similar to what is known as the *preproduction phase*.
First, we have to design and plan what will be the final goal we want to achieve, in this case, everything stated on the General Objectives part [1.3. General Objectives]. Then, from this general plan, we split each objective into little milestones, needed to create the software. Those will be our *features* [1.4. Specific Objectives], which we will organize depending on their time cost, effort and, especially, their dependencies on each other. For example, we cannot reach the *training system* part of the ANN without having a good structure of said Network.

Once each required feature is thought out, planned and organized, we will proceed to the development phase. For this phase the procedure is simple. We start by the feature we have planned to do first and design the more specific parts of this feature and actually incorporate it to the product. Once it is considered finished, it is tested and reviewed to be sure everything works as it should. If the feature passes this test, it can be incorporated and we can continue with the next feature, repeating the process. If the feature does not pass the test, we get back to the design or coding phase to fix everything that has to be fixed on that feature.

At the end we will have a complete product made of lots of features that work perfectly together.

What we accomplish with this method is, first of all, organization for the developer: it is easier to manage and to know what is causing issues. It also creates encapsulated, non-dependent features that do not compromise the whole project in case something goes wrong.

It is worth noticing that the list of features can change at any time. Thanks to the nature of this methodology, removing or adding features should not be a liability.

4.3. The Concept of Done:

There is a topic in every development process that seems really obvious, but also very dangerous if it is not tackled properly. This is the definition of the *Concept of Done*.
Basically this concept responds to a single, simple question: ¿When can we say a feature is finished?

This is not as easy to answer as it seems so we have to define when, each feature, will be declared as 'Done' (obviously, each feature is subject to changes and polishes, however, we will need to have some references).

- **Neural Network Structure:**
  - **Neural Network Structure:**
    A network structure that supports nodes, connections and layers. Each node should be connected to all nodes from the next layer.
  - **Neural Node Structure:**
    A node structure that is able to calculate and hold input and output values as well as a Bias value.
  - **Thought Process Structure:**
    A structure able to hold all the relevant values from each node, be stored into memory and used by the next generation of the same network.
  - **ANN Manager Structure:**
    An external class that can hold all the Networks created by the user as well as provide general functionalities.

- **Training System:**
  - **Reinforcement Algorithms:**
    The network's values are automatically changed depending on the reinforcement algorithm and its inputs and outputs, adjusting more and more to the desired or logical output behaviour.
  - **Agent Generation:**
    The system should be capable of spawning more than one agent with the desired network. Each agent should have its own lifecycle but with the same values as the rest of the same Generation. The values should be able to be stored and used to generate the next Generation.
  - **Thought Process Generation:**
    Thought Processes should be automatically generated before spawning the next generation of agents. The result should take on account the most fitting values of the last Generation.

- **Custom UI:**
  - **ANN Manager UI:**
    The user can easily create new networks, store and manage them. It can also access to any networks editor. This is done through the Unity's ScriptableObject elements.
  - **Neural Network UI:**

The user can Add, delete and manage nodes as well as connections. The Node UI can be opened from it.

- **Neural Node UI:**
  The user can quickly see the interesting data of a node, such as the Bias and Input/Output value.
- **ANN Component UI:**
  This component can be added to a Unity GameObject and the user can choose any Neural Network to add to it. The Network UI can be opened from it.
- **Training UI:**
  The user has to have the capability to supervise the evolution of the training phase, even though he should not interfere. Also, he should be able to change several options such as the number of Agents per Generation or the Neural Network Mode (Design / Training).

5. Project Development

5.1. Unity Engine

As it has been mentioned previously, this project uses the Unity Engine as a platform to be developed on. It offers great tools to help speed up the process, such as the UI system, Scripting and the Scriptable Objects class.
It also one of the most popularised and easy to use game engines, which makes it perfect for our target audience.

5.1.1. UI System

The Unity UI System is an integrated library to create your own UI for the Unity Editor, which is where our users will work on while creating their ANNs.
It is worth noticing that, when referring to the Unity UI System, it is not an "in-game" UI, it is actually an "in-editor" UI system that can be customized to adjust to our project.
This part will be tackled more in depth on 5.4. UI System were the different elements used will be explained.

5.1.2. Scripting

Unity Engine is based on a Scripting system. Scripting is really useful part of coding that, in short, is based on creating scripts and classes that are attached to specific objects in the unity editor. Once the game is in 'play mode', specific methods of this scripts are called (Start(), Update(), ...) to apply the behaviours that have been coded.

5.1.3. ScriptableObjects

Scriptable Objects are a type of C# class integrated in Unity. This is built on top of the Scripting part of Unity. Scriptable Objects are scripts or classes that do not need to be attached to objects like normal scripts do. This proves to be extremely useful to code both networks and nodes since, even though they could be created through normal scripting, it is way quicker, useful and comfortable for the user.

5.2. Network Structure

To start with the Artificial Neural Network (ANN) development, we need to first have a structure to support all the behaviors we need to perform.

The following is both an explanation of each core class in the project and what it took to create them.

5.2.1. Network

The Network is the main class of this structure. It is based on the models presented in 2.2. Network Structure, more specifically on the Feed Forward (FF) structure, which does not make any assumptions on the relations of data and will be easier to adapt to any situation.
It has a graph structure with different nodes (or neurons). This class handles the main methods and behaviours regarding the ANN such as calculating the output values, loading nodes, setting up connections and generating new generations from the stored thought processes amongst many other functionalities.

Layers:

Layers are an important concept in ANNs, specially on FF structures. There are 3 main layers:

- **Input Layer:**
  The input layer is the "first" layer of the network. It is composed by the input nodes which handle the input of data into the system. It connects directly to the first hidden layer.
- **Output Layer:**
  The output layer is the "last" layer of the network. It is composed by the output nodes which handle the output of the data generated by the system. Its information comes directly from the last hidden layer.
- **Hidden Layers:**
  This layer is composed by the hidden nodes. This is a more complex layer than the others since it has multiple layers within. Basically, it is actually not a single layer but a collection of many, connected to the adjacent ones.
  Depending on the network, the amount of hidden layers can be adjusted. This is used to give the network the ability to perform more complex tasks. For example, it is used to make the network 'memorise' something that is not currently being sensed, which a one-layered network wouldn't be able to do.

5.2.2. Node

The node is a core unit in the ANN system. It handles basic operations like the Weighted Sum and Activation computation which we will see more in depth in 5.3.2. Functionality.
It also stores the Bias Value and the Activation Method (5.3.1. Important Elements).
Each node is connected to all nodes from adjacent layers, receiving input from the last layer and giving output to the next layer.

Input Node:

This type of node is at the first layer of the network. It has an input value that is received from the Agent. This value is sent to the next layer when the network is computing.

This node will be defined by the user.

Hidden Node:
The hidden node is in the hidden layer. Its main aspects are the bias value and the activation method (even though the output nodes also have them).
They are intermediary nodes that help the network computations, and, even if it is true that an ANN could work without them, they are needed to perform complex operations and a must-have on most ANNs.

These nodes are created transparently to the user, but its amount can be defined.

Output Node:
These nodes are at the last layer of the network. They perform a similar operation to the hidden nodes but also store the Output Value computed by the system until the Agent asks for it.

This node will be defined by the user.

5.2.3. Connection

A connection is a structure that handles the transmission of data between two nodes. In many networks the functionalities of this structure is handled by the nodes themselves, but for this project, separating it into its own class helped both organizing the code and making the code easier to handle.
It gets the output data computed by the first node and passes it to the second one so it can perform his. This structure has a Weight Value which is needed for each node to compute its data (5.3.1. Important Elements).

5.2.4. Thought Process

The so called Thought Process is actually a structure that stores each weight and bias value of each node and connection and also the fitness of the whole process.
It is used to train the net and create new generations.
It will be explored in more depth in 5.4.2. Storing Thoughts and 5.4.3. Genetics & Mutations.

5.2.5. Agent

The Agent class is the one that serves as a bridge between the network, the environment and the behaviour.
It is actually a class thought to be parent class. This means that is designed in a way that the player can make its actual script an inherited class from the Agent class.
By doing this, the user can assign it a network and defined its own behaviours as well as give and ask for the inputs and outputs of the network.

The Agent script must be attached to the desired Object that will perform its behaviours.

5.2.6. Academy

The Academy is a needed class, specially for the Multi-Agent Training (5.4.5. Training Type). It is actually external to the network's processes but is used to handle all the agents spawned by the training sessions.
An agent Prefab or Object needs to be assigned to the Academy to use it as the base object for the agents that will train and modify the original network.

5.3. Network Functionality

5.3.1. Elements

Input/Output Value:

The input value of the networks are all the data received by the network. Each input node has one and can be assigned through the agent. It is used for the network to start computing its output values.

The output values are the data generated by the network. Each output node has one and can be read from the agent to implement its behaviours depending on its value.

Bias Value:

The bias value is in each node and it is used when computing its own value that will be passed on to the next one. It acts as a sort of filter for the node to prevent the node from triggering which is a needed functionality for the network to perform correctly.

Weight Value:

The weight value is on each connection basically acts as a multiplier for the value that is being inputted through that connection. For example, a node with a connection with W=1 will have less impact that a connection of W=10.

It is used when computing the value of a node (See 5.3.2. Functionality).

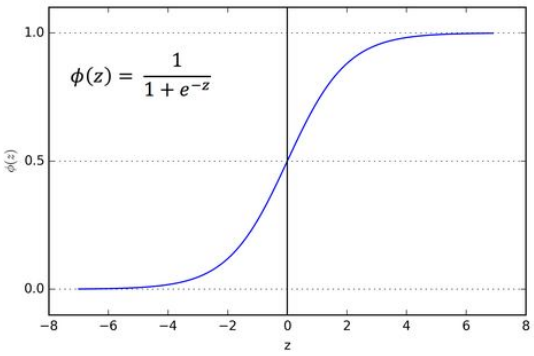
Activation Method:

The activation method is an algorithm applied to the computed value of the node, after applying the weights multipliers. It is an algorithm that tells the network if the neuron is being "fired" or not and adjust the value to suit the network goals. There are an infinite amount of possible activation methods but the following are the most common and the ones implemented into the project:

- **Sigmoid:**

  Useful to encapsulate the value between 0 and 1. It creates extreme values such as the very positive values are very close to 1 and the very negative values are very close to 0.

  $$V = \frac{1}{1+e^{x}}$$

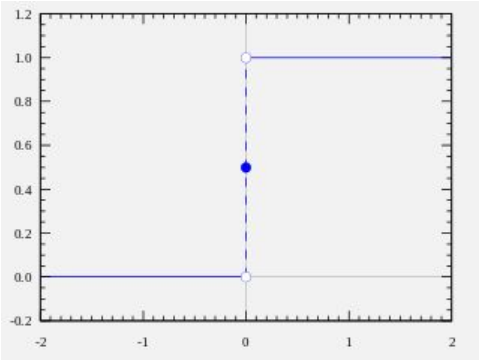


F5.A Sigmoid Function Representation (from *Ref.45*)

- **Step:**

  The step method sets a threshold value. If the node's value is lower than that, the value will be 0 (it is not triggered), if it is bigger, it becomes 1.

$$V = (x > th)\ ?\ 1\ :\ 0$$

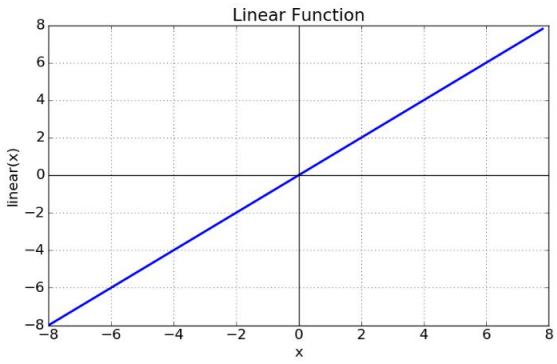


F5.B Step Function Representation (from *Ref.44*)

- **Linear:**

  The linear function will return the value multiplied by the damp value. It is the simplest activation method.

$$V = mx$$

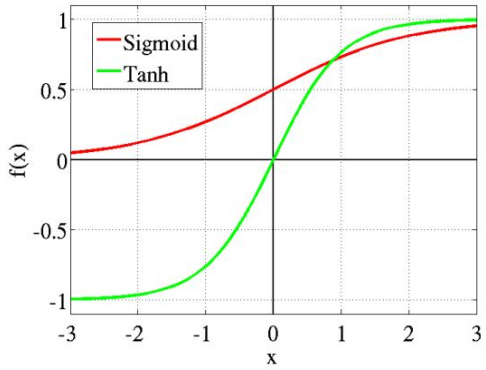


F5.C Linear Function Representation (from *Ref.45*)

- **Tanh:**

  The Tanh algorithm is very similar to the Sigmoid but it can generate values between -1 and 1 apart of having a slightly different curve.

$$V = \frac{2}{1+e^{-2x}}$$

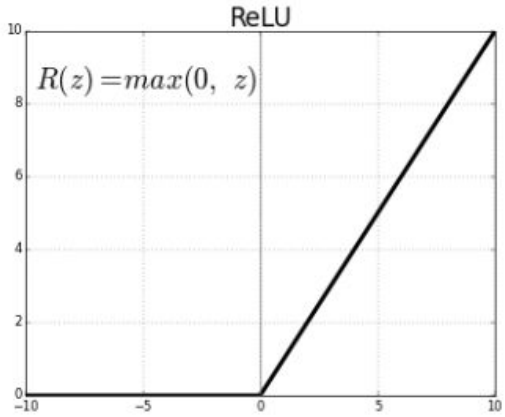


F5.D Tanh Function Representation (from *Ref.45*)

- **ReLu:**

  The ReLu method will return the same value if it is positive and a 0 if it is negative.

$$V = max(0,\ x)$$



F5.E ReLu Function Representation (from *Ref.45*)

Fitness Value:

The fitness value is a 'punctuation' given to the current network's process. It is used in order to let the network know how well it did with its last process. This is a key part of the Reinforcement learning.

An example of fitness value could be the amount of time that the agent endured before hitting a wall.

It is assigned to the network when it ends the process.

5.3.2. Functionality

The Artificial Neural Network concept is actually fairly simple. The agent receives a set of different inputs that have been programed by the user. These inputs are then passed into each input node (p.e. The distance to the next milestone is passed to the node named "distance" and the speed of the agent is passed to the node "speed"). When the nets receives these inputs, it starts a 'chain reaction' between its nodes, computing its values one-by-one and layer-per-layer. At the end that computation reaches the output nodes and their output values, which are received by the agent, just as they did with the input values, to change its behaviour, defined by the user.

Computing the Node Value:

The process is simple, however, each node has to compute its own value and pass it to the next layer so it can compute its values as well.

This value is divided in three parts:

1. **Weighted Sum:**

The weighted sum is the sum of all values inputted through each connection, multiplied by the connection's weight value.

$$WS = \Sigma(i_i \cdot w_i) = i_1 w_1 + i_2 w_2 + \ldots + i_n w_n$$

2. **Bias appliance:**
   When the weighted sum is computed, we apply the bias value to it.
   $$Biased = WS + B$$

3. **Activation Function:**
   To end with the computation, the activation method is applied and the final value is passed to the next node, so it can repeat this process. (See 5.3.1 Important Elements - Activation Method)
   $$Value = ActivationMethod(Biased)$$

5.4. Network Training

Once the network and its functionalities are set up, we need to aggregate the 'Training' part of the ANN. This is the trickiest part of them and can become kind of an 'abstract' matter since it is never clear if the network is performing as it should.
To avoid that, I set up a couple of very simple games (*Flappy Birds* and *Temple Run* like games, see more on 5.6. Validation Minigames) were I could predict the expected behaviours or at least tell, after some time of training, if the agent was 'learning' properly.

It is worth noticing before explaining the training process, that this project uses Genetic Algorithms (GA) to train its network. This process is inspired by the natural process of evolution and *survival of the fittest*. This, however, does not mean it is neither the only nor the best method.
There are several training techniques for the reinforcement model, such as Q-Learning. The training method could differ depending on the type of behaviour the user wants to perform, despite of that, we have chosen Genetic Algorithms since it does not take any assumptions and is better fit for general purposes, being slower than most techniques but way more adaptable.

The training starts with a first random generation and, from there, it will pick couples of processes from the last generation to create an offspring agent with combined values from both parents. After some generations and mutations, if everything is working properly, we should be able to see an improvement on the fitness value of each generation.

5.4.1 Generation ∅:

The very first generation has no thought processes to generate new agents. This is solved by creating agents with random values so, not only we can create a generation from nothing, but also make a great pool of values for the next generations to use for their mutations.
Here, we can notice that, at the beginning, most generations will have an erratic behaviour and most likely they will achieve close to no improvement. Due to the randomness of this state, it is impossible to predict the amount of time it will take the network to generate an actual improving agent, however, it is an expected behaviour of this kind of algorithms and once the first improving agent spawns, the networks starts learning way faster.

5.4.2. Storing Thoughts Processes:

Before we start generating agents, we need to first store every process generated by the last generation.
This is achieved through the ANNThoughtProcess structure (see 5.2.4. Thought Process).
We generate a thought process for each agent in the generation and inside of it we store the bias of each node and the weights of each connection as well as the network's fitness.
After every agent has generated its thought process, they are assigned a probability based on its fitness (in this project this probability is stored in the process itself for commodity, but it is not needed).
The way we assign the probability can change, however it was decided to use an exponential algorithm so the processes with more fitness would have way better chances to be chosen as parents.

$$Prob = Fit^2 / (\Sigma Fit_i^2)$$

5.4.3. Genetics & Mutations:

After we have given a probability to each thought process, we have to start producing offspring.
To do so we have to follow these steps:

1. Chose a couple of processes as parents.
2. Apply genetic algorithms (crossovers).
3. Adjust new network's values from resulting process.

To choose a pair is fairly simple, we produce a random number, and, depending on the probabilities we grab one from the list, then we repeat this process one more time to get the second parent.

After we have the chosen couple, we apply the genetic algorithm.

Genetic Algorithms:

The Genetic Algorithms are actually a way to generate a new string of values from two other strings. There are several ways to do so, but for this project we use the Slice Crossover and the Random Crossover methods, both based on actual genetics of chromosomes.

- **Slice Crossover:**
  This is the actual method used by chromosomes.
  A position in the string is chosen, then we grab a number of values from it. The resulting string is the first parents string but with a slice of them switched by the grabbed values from the other parent.

```
for(int i = 0; i < processLength; ++i)
{
      if( i > randIndex && i < randIndex + sliceLength)
      {
            offspring [ i ] = parent_B[ i ];
      }
      else
      {
            offspring[ i ] = parent_A[ i ];
      }
}
```

Parent_A = A A A A A A A A
Parent_B = B B B B B B B B

Resulting Offspring = A A B B B A A A

- **Random Crossover:**
  The Random Crossover method is a variation of the Slice Crossover method.
  The main difference is that the switched values are chosen randomly, and not from a slice of values.

This produces less stable offspring but is quicker learning or adapting and generates more unexpected results, which is a good thing to avoid the Local Minimum issue (See 5.4.4. Potential Issues).

```
for( int i = 0; i < processLength; ++i )
{
     offspring[ i ] = rand(0, 1) > 0.5 ? parent_A[ i ] :
parent_B[ i ];
}
```

Parent_A = A A A A A A A A
Parent_B = B B B B B B B B

Resulting Offspring = A B A A B B A B

Mutation Algorithm:

The mutation applied need to be computed through an algorithm since, even if we mutate a value, we want it to mutate it with a certain logic behind, giving a random value but not too different from the actual one.
The mutation algorithm used is the following one:

$$V = x + x \cdot (rand(0, 1) - 0.5) + 3 \cdot (rand(0, 1) - 0.5)$$

It is based on the Gaussian Mutation algorithm (*Ref.31*).

Tips & Tricks:

When training the network, it is possible that it does not achieve the expected behaviours or that it "unlearns" away of the best outputs. This is handled by doing some of these tricks:

- **Adapt Mutation Rate:**
  The mutation rate is reduced as when the fitness of the generation is greater or close to the best fitness of the last generation. At the same time, it can be augmented when it is way lower (in both cases in a percentual way depending on the values obtained).
  This is used to avoid the network unlearning because of the amount of mutations it suffers, as well as avoiding more erratic behaviours.

- **Force Crossovers:**

Another way to help the network learn more efficiently is to force the network to produce some agents from the best two processes (regardless of the probabilities).
This creates a mutated version of the best agents so it has more probabilities to perform better than it predecessors.

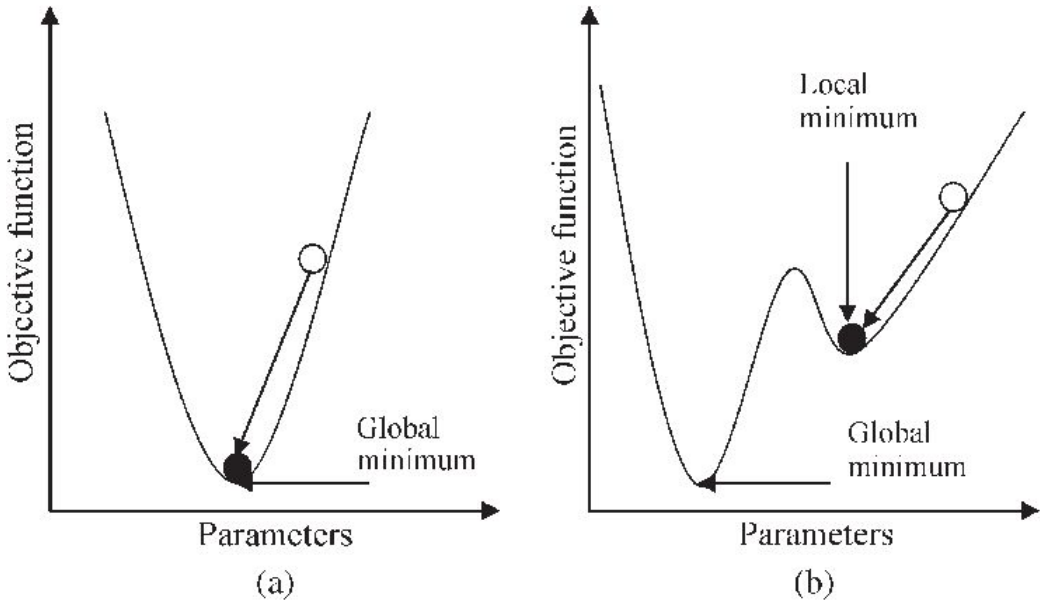
- **Direct Copy:**
  The Direct Copy method is similar to the Force Crossovers but, in this case the agent generated is not a mutated offspring but a direct copy of the best process. It is useful to be sure the network will always have a good agent to start making agents and, even though it can accentuate the Local Minimum issue (5.4.4. Potential Issues) if we reduce the number of copies to a minimum percentage of the number of generated offsprings (in this project is just 1 copy) its effect on that issue is minimum.

5.4.4. Potential Issues:

When creating an ANN there is a number of potential issues you could encounter. The following is a list of the most common and how to solve them:

Local Minimum:

The Local Minimum issue is basically when the network arrives to an optimal solution but the most optimal is actually "hidden" between non-optimal solutions.
For example, if an agent has to decide between two paths, one that gives them 1 point every step and another that has a huge reward at the end (which the agent does not know of), the agent will always choose the first one, since it has more apparent rewards.



F5.F Local Minimum Representation (from *Ref.43*)

This can be solved in many ways; these are the most popular:

- **Mutations:**
  This is the most used, and the one used in this project. When generating new agents from the last generation, it does not only combine the values of two parent processes but it also modifies some values depending on some percentages and a Mutation Rate value.

- **Add Random Processes**
  When generating the new agents, shuffle a number of randomly generated processes to add randomness to the offspring creation and be able to add new values to the pool.

5.4.5 Training Type

Training a network is a slow matter, if, for example, we have a number of agents per generations of 50 and each agent lasts for 2 seconds, we can already make quick maths to realise it can take several hours for the network to make a significant amount of cycles.
This is solved by running several copies of the network at the same time, going through a whole generation in the same span of time a single agent would.
This is a good method to speed the training up. However, when playing the game, we do not want all those agents running at the same time, but just one improving little by little.
This project has been developed so it can accept both types of training.

Multi-Agent:
The Multi-Agent method is the first that has been mentioned. By using the Academy, several instances of the original agent, with its respective networks, are created. Then, the agents are thrown into the environment do their respective behaviours and functions. When their cycles end, once each one of them has ended, all the processes generated are passed into the original network and the process is repeated with new offsprings.

Single-Agent:
The Single-Agent method is the type of training prepared to be used already “in-game”. There is just one agent and this keeps improving little by little, going through each cycle by itself.
This is a slow method but if the network has been previously trained with the Multi-Agent method, it can create great results while playing the game.

5.5. UI System

5.5.1. Elements:

Network Tab:

This 'Tab' or window can be opened by pressing the '*Network Tab*' button on the Network Inspector.
This shows a graphic representation of the networks that has been created:

Input Nodes (Graphic):

Color: BLUE
Header: [node_name : input_value]

Hidden Nodes (Graphic):

Color: ORANGE
Header: [Node_name : bias_value]

Output Nodes (Graphic):

Color: GREEN
Header: [Node_name : output_value]

Connections (Graphic):

Graphic representation of the actual connections between nodes. Their size depends on the amount of strength / weight they have assigned.

Information Panel:

Network Name:
Shows the Network's name (char string).

'*Generation*':
Shows the number of the current Generation of the net (From 0 - inf).

Nº of Nodes:
Shows the number of total nodes inside the network (InputNodes.size() + HiddenNodes.size() + OutputNodes.size())

Input Nodes:

Shows the number of Input Nodes inside the net.

Output Nodes:
Shows the number of Output Nodes inside the net.

Hidden Nodes:
Shows the number of Hidden Nodes inside the net.

Connections:
Shows the number of connections inside the net.

'Tick Net' Button:
Will 'Tick' the net on click. This means the net will recalculate the output value with the current values. This is a mostly debug function since this 'Tick' is triggered by the network automatically when the Agent is training.

'Reset Net' Button:
This resets the net's values. All values in each node will be given the default value (most of them are random by default). This is a debug functionality since it replicates the 'Generation 0' setup.

'Reload Net' Button:
This will load the net from the last save file. This replicates what the application does when we press the 'Preview/Play' Unity button. This is a solution to an issue treated on 5.9. Issues with the Unity Engine.

Layer Markers:

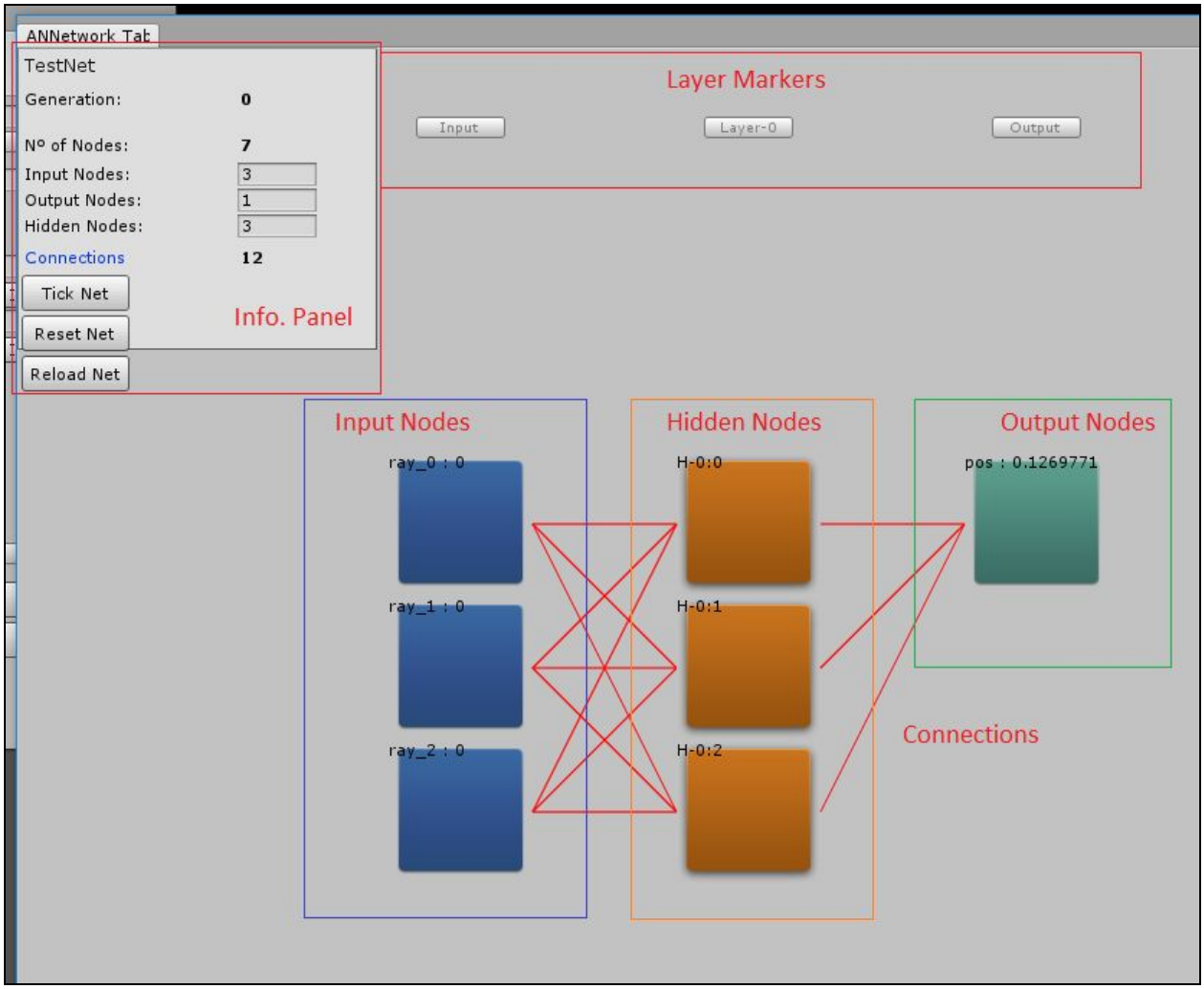
Layer markers or a graphic rectangle that is always on the same Y axis position but will keep its X axis position where the layer they are marking is. This is to quickly visualize the number or name (in the case of the 'Input' and 'Output' layers) of a Layer.

Extra Functionalities:

Move Network Graphics: [MOUSE DRAG]
On mouse drag, the networks graphics will move with the mouse movement. This is to be able to see the whole representation of the network easily.

Open Node Inspector: [LEFT CLICK on Node]

On mouse left click on a node, on the Unity Inspector Window will appear the clicked node's inspector.



F5.G Network Tab

Network Inspector

The Network Inspector is a custom version of the default Unity Inspector. It shows important net information as well as offer options and functionalities that make the network setup easier and easily understandable.

'Cycles per Generation':

Adjusts the number of cycles a network will do before updating to the next generation.
On Single-Agent, this translates to the number of times the Agent has to 'end'.
On Multi-Agent this translates to the number of Agents that will be spawned on a single cycle.

Network Training Type:

Adjusts the type of training the network will follow. There are 2 type:

Multi-Agent:
Various agents will be spawned to proceed with the behaviours and the generation won't end until every single one has ended (See 5.4.5. Training Type).

Single-Agent:
A single agent will be spawned and will proceed with the behaviours. It will need to end X times before updating to next generation (See 5.4.5. Training Type).

Nº of Hidden Layers:

Adjusts the number of hidden layers the net will have (See 5.2.1. Network).

'Manual Nº of Hidden Nodes' Checker:

The number of hidden nodes is determined by an algorithm to be as optimal as possible, however, it can be manually introduced (See 5.2.1. Network).

Checked:
Manual Mode is activated.

Unchecked:
Automatic / algorithmic mode is activated.

'Add Nodes' Menu:

This part of the inspector is designed to easily add multiple nodes to the network.

Add Input / Output Node Field:

Drag & Drop a node to its respective field or click on it to open the asset inspector and choose one. The node will remain on the field until the 'Add Node' button is pressed.

'Amount to Add':

Adjusts the number of instances of the selected node the user wants to add. If multiple nodes were to be added, their names will add the suffix '_[number]' where [number] is the correspondent index of the added node (p.e.: Add 3 instances of the node 'Position' -> List: Position_0, Position_1, Position_2).

'Add Input / Output Node' Button:

Will add the selected node on click, taking on account the 'Amount to Add' value. This will also clear the 'Add Input / Output Node' field.

Number of Input / Output Nodes:

Shows the amount of Input and Output nodes the Network has.

'Hide Names' Checker:

Checked:
The list of nodes' names will be visible.

Unchecked:
The list of nodes' names will be non-visible.

Input / Output Nodes Lists:

Show the list of names of all nodes in the network. It is visible by default.

'Delete' Button:
On click will delete the selected instance of the node and remove it from the network (does not destroy the original asset of the node).

Tick Type:

The tick type of the network is an important matter regarding optimization since the 'Tick' functionality (recalculation of the output values) is one of the heaviest parts of the network's code.
It is on 'Tick on Output' by default since most Agents tend to have more inputs than outputs requests.

Tick on Output:
The net will 'Tick' when an output value is requested (GetOutput(node_name) method).

Tick on Input:
The net will 'Tick' when an input value is set (SetInput(node_name, value) method).

Manual Tick:
The net will not automatically 'Tick'. If this type is selected, the user has to call the method 'network.Tick()' on its project whenever he wants to recalculate the networks' output value.

'Network Tab' button:

Opens the 'Network Tab' for the inspected net. (See 5.5. UI System)

'Save Network''button:

Saves the net configuration and nodes. This has been coded so it is done automatically by the tool on most changes to the net, but a manual button has been implemented if the user desires to use it.

Generation Graph:

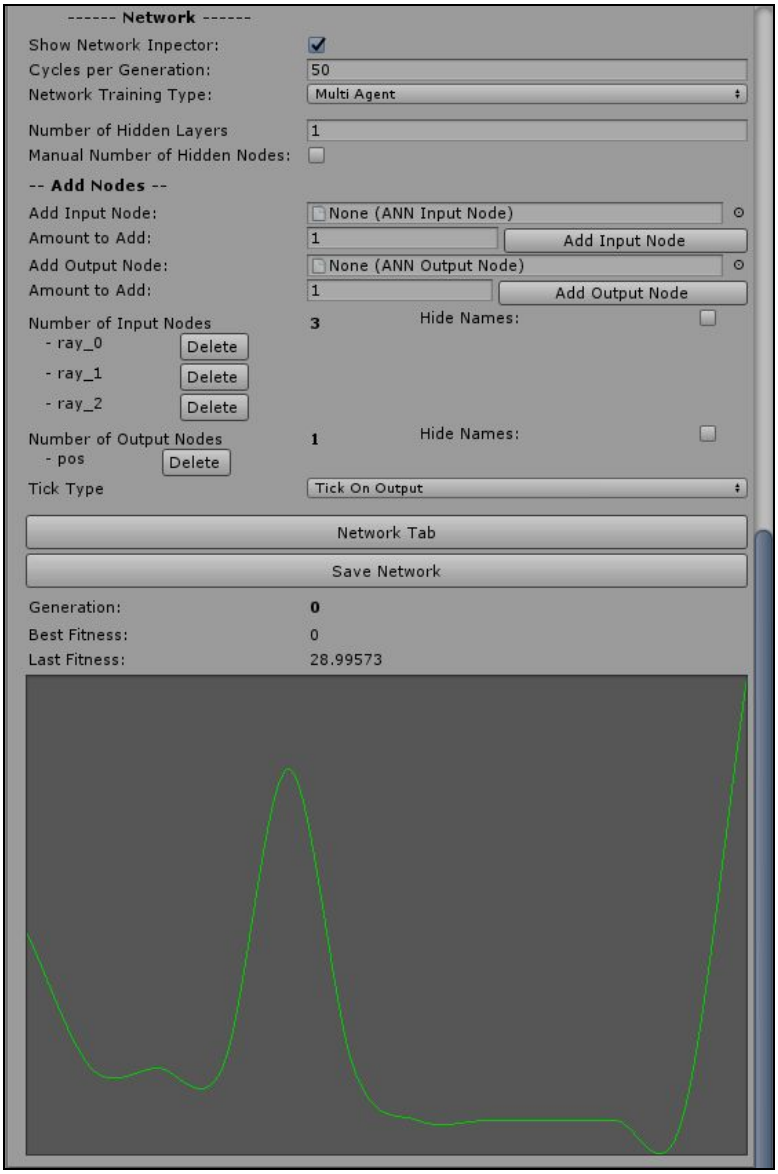
This is a very important part of the inspector when training the network.
It will show the progress of each generation of the network that is being currently trained, showing the top fitness of each gen.

Generation:
Shows the number of the generation is currently been training.

Best Fitness:
Shows the top fitness of all generations.

Last Fitness:
Shows the top fitness of the last generation.

Performance Graphic:
Shows the progress of each generation on a graph so it is easily visualized and comparable by the user.



F5.H Network Inspector

Node Inspector:

Shows and adjusts the different values inside a node.

Network:

Shows the network this node is currently in.

Bias Value:

Shows the Bias Value of the selected node.

Activation Method:

Adjusts the activation method of the selected node (See 5.3.2. Functionality).

'Apply to Layer' Button:

Applies the activation method of this node to all nodes inside the same layer of the parent network.

'Apply to All Nodes' Button:

Applies the activation method of this node to all nodes inside the parent network.

F5.I Node Inspector

Agent Inspector:

Script Panel:

Shows all the script variables as the usual Unity Inspector for a script (all 'public' or marked as 'SerializeField' variables) (*Ref. 51*).

Agent Panel:

Shows the Agent-related variables (inherited from ANNAgent).

'Network' field:

Drag & Drop or click to open the Asset menu and choose the network this Agent will have attached to it.

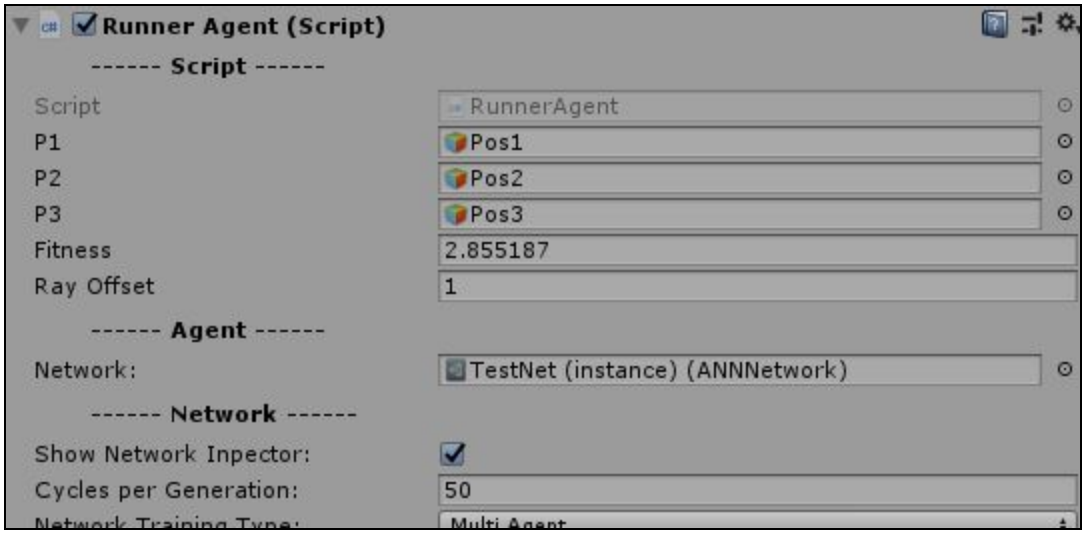
'Show Network Inspector' Checker:

Checked:
Shows network inspector.

Unchecked:
Hides network inspector.

Network Inspector:

Shows the Network inspector inside the Agent inspector to easily access to all attached network's functionalities and information (See Network Inspector).



F5.J Agent Inspector

5.5.2. Code Elements:

- **Core Methods:**
  - ***void SetInput(name, value)***

    This method is used to set the Input Value of the named node.

  - ***float GetOutput(name)***

    This method is used to get the output value of the named node. In default mode it will trigger the computation of the value.

  - ***void EndNetCycle(fitness_assigned)***

    This method is called when the process of this net can be terminated and stored, for example, when the agent dies or hits a wall.

- **Customizable Methods:**

  The following methods can be customized by overriding the inherited method from the ANNAgent class.

  - ***void OnGenerationEndSingle() / OnGenerationEndMulti()***

    It is called when all the agents of a generations have been ended. It is useful to, for example, re-start or setup the environment whenever the next generation has to be created.

  - ***void OnAgentEndSingle() / OnAgentEndMulti()***

    This method is called when the agent has ended (has called EndNetCycle() method). It is useful to destroy or deactivate the agents that no longer need to be actively training.

5.6. Validation Minigames

To check the functionality of the network, some validation minigames were developed.
These games are very simple games with very simple mechanics, used to check if everything worked as expected and at a good rate.

5.6.1. 'Flappy Bird':

The 'Flappy Bird' minigame consists in a set of bars that come towards the agent. These bars have random variable height.
The Agent gets 4 inputs:

- Horizontal Distance towards next bar (hdist)
- Vertical Distance to next bar's height (ddist)
- Vertical Distance from floor (ypos)
- Agent's speed from last frame (speed)

From this data, the agent generates 1 output:

- "Jump" value, between 0.0 and 1.0 (click)

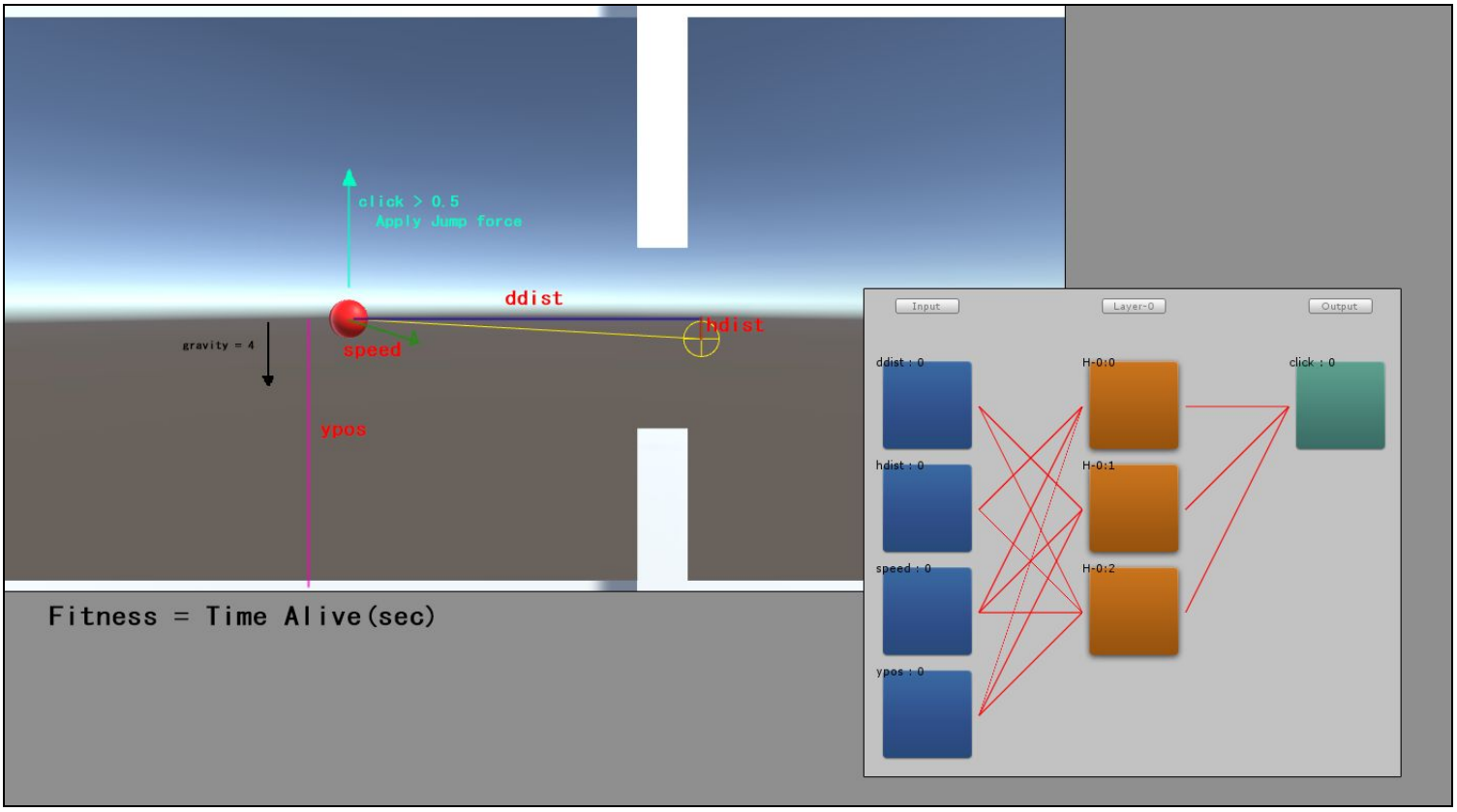
If the Jump value is greater than 0.5, the Agent will make a little jump, augmenting its height to be able to pass through the bars.

Number of hidden nodes: 1 layer with 3 nodes.

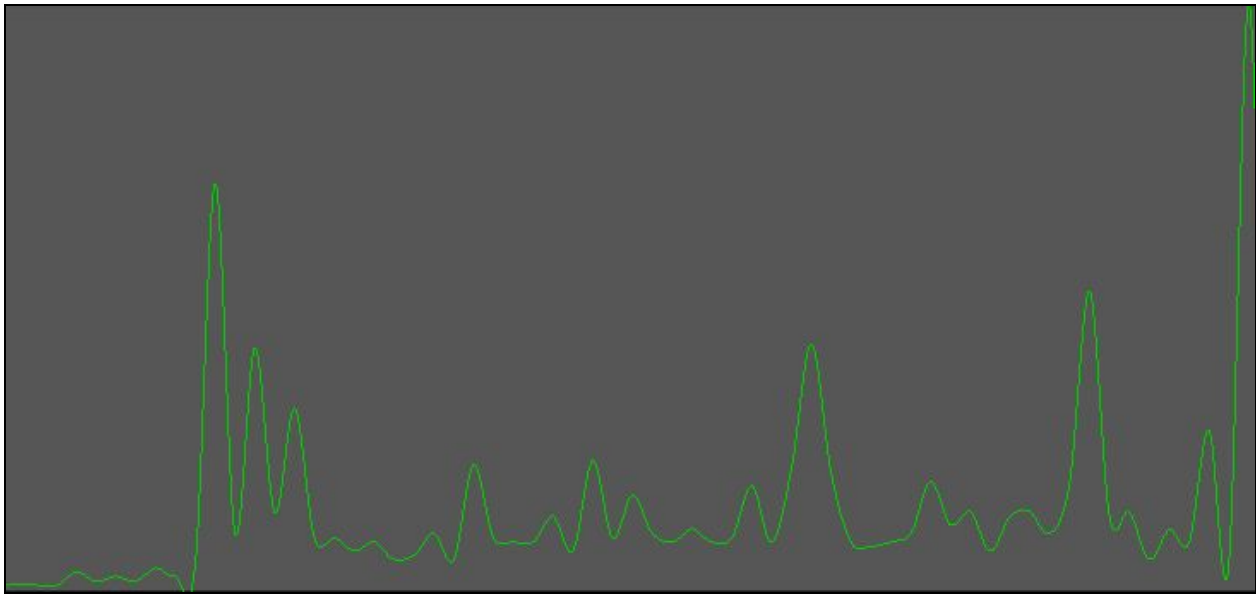
The fitness value is equal to the amount of time the agent has kept without hitting any wall or bar.

It must be pointed out this minigame has physics embedded like gravity and force which makes the process much more complex than a task with higher certainty.

Results: The Agent is able to stand for long times avoiding the bars coming at them.



F5.K Minigame Setup Diagram



F5.L Bird Results Graph

5.6.2. 'Runner':

The 'Runner' mini game consists on a platform with sets of obstacles that keep spawning and going towards the player. At the same time, some of these obstacles have a chance to be coins, which gives a +25 bonus to the player.
The Agent has 3 positions it can stand.
Inputs:

- Distance to nearest object from Pos01
- Distance to nearest object from Pos02
- Distance to nearest object from Pos03
- Type of nearest object from Pos01
- Type of nearest object from Pos02
- Type of nearest object from Pos03

Outputs:

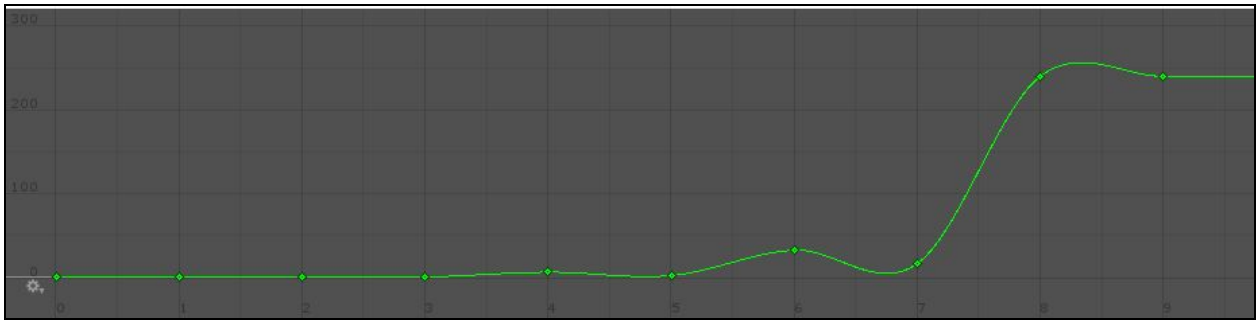
- "Position" value, between 0.0 and 1.0:
    - if x > 0 && x <= 0.33 -> Stand on Pos01
    - If x > 0.33 && x <= 0.66 -> Stand on Pos02
    - If x > 0.66 -> Stand on Pos03

Number of hidden nodes: 1 layer with 6 nodes.

The fitness value is equal to the amount of time the agent kept without hitting any obstacle plus 25 points per each grabbed coin.

```
Fitness = TimeAlive + NumCoins * 25
```

Results: The network works especially well for this case. The best result registered is +2.000.000 points (sec). Since it has no physics involved, the network is able to work its way around the task with far more ease.



F5.M Runner Results Graph

5.7. Usability

The following are a collection of guides and features to use the ANNProject toolset.

5.7.1 How To Use ANNProject

Creating a Network:

Creating a network is really simple:

1. Mouse Right-Click on Assets directory
2. Create > Network

A new object called 'New Network' will appear, this is the network that has just been created.

Creating a Node:

Creating a node is similar to creating a network:

1. Mouse Right-Click on Assets directory
2. Create > Node > Input / Output

A new object called 'New Node' will appear.

Important -> Name the node knowing this name will be later used to SetInput and GetOutput methods.

Setting up a Network:

Once the needed nodes are created, click the network object. The inspector menu of the selected net will open at the right part of the editor. Grab each node and put them inside the 'Add Input/Output Node' field.

It is also possible to modify the number of hidden layers, hidden nodes per layer and the number of Agents per Generation. Each one of them can be changed like you would in the actual Unity Editor UI.

After that, we have a net set up, however, we lack an Agent.

Creating an Agent:

To create an Agent you need to create a new Unity Script and change the parent of the class from 'MonoBehaviour' to 'ANNAgent'. With this we make our class inherit from the ANNAgent class so its code can be used by the network.

Note: All standard methods (such as Start() or Update()) work exactly like they would in a MonoBehaviour class since ANNAgent actually inherits from it.

In this class we created the behaviours can be coded as they would in a normal Unity Script or GameObject.

To set the inputs of the assigned network, we can use *void SetInput(node_name, value)* and to get the outputs *float GetOutput(node_name)* with these values, the different behaviours that have been coded inside the agent scripts can be tweaked, for example changing the speed of the agent.

To define the end of the Agent we use *void EndNetCycle(fitness_assigned)*. This requires to be assigned the fitness of the Agent so it can be used by the network. So the fitness value must be calculated however the user wants inside or outside the agent script, but, at the end, it must be assigned in this function.

Other important functions that we can override are:

- OnAgentEndSingle() / OnAgentEndMulti()
- OnGenerationEndSingle() / OnGenerationEndMulti()

These methods have to be overwritten with the 'override' keyword to be used.

Here we can define what happens when an agent ends its cycle and what happens when the whole generation has ended, respectively. They can also distinguish between Single-Agent and Multi-Agent type. (See more at 5.5. UI System - Customizable Methods)

Setting up the Academy:

The Academy is a needed class so the agent can be both replicated and set for the behaviour to be played.
Grab the ANNAgentAcademy asset from the Assets directory and drag it into the scene.
This object already has a script attached to it, only needs to be assigned the asset or prefab that will be trained.
Note: The object assigned must have been previously saved as an asset inside the assets directories.

Play the Network:

At this point, everything is set up for the agent to work and train. Press the 'Play' button inside unity editor and everything will start training and learning.

5.8. Extra Functionalities

5.8.1. Serialization

Most elements in the application are serialized automatically by the tool itself and then loaded when needed. It uses a system of references to avoid unnecessary weight into the application.
This is a core functionality that even though it doesn't stand out for the user, was a really heavy task to perform working in tandem with the Unity serialization methods.
This was also used to solve an issue with the Unity Engine itself, treated on 5.9. Issues with the Unity Engine.

5.8.2. Generation Load

Using the serialization of the networks' elements, it is possible to load generations, providing great functionalities into the application. When the system detects the last generation's processes list is empty, it will automatically try to load the last generation of the current networks. This is really useful, for example, when using the Single-Agent training type since it can work upon the training previously done by the Multi-Agent type. A generation to load can be manually chosen by the user using the '*Load Generation*' button on the Network Inspector.

5.9. Issues with the Unity Engine

The Unity Engine provides a great platform to create many different projects and videogames as well as tools such as ANNProject, however, it also has its own limitations and rules that create some issues when developing on it. Most problems are similar to those one would encounter when using a game engine but there was one key issue that really affected development. The way Unity works, there are two different 'scenes', the editor and the play or preview. Normally, there is no difference between them but, what Unity does on the background is that, when entering the 'play' scene, it serializes every element on the 'editor' scene and then loads it into preview. This works most of the time but there is one very key situation where it fails. Unity can only serialize elements it 'knows' off, this means that when using elements created by the user from scratch (such as many elements from this project) they cannot be serialized and loaded automatically by the engine, breaking the preview scene.
This issue was solved by implementing an automatic serialization to every element that needed it, such as the Network, Nodes and Thought Processes. This technique was simple but at the same time escalated quickly into a huge, core, issue to handle that took more time and effort than expected and every change could affect the behaviour of the net itself, since the loading process was key to put the network into play. (5.8.1. Serialization)

6. Conclusions & Future Work

At the end of this project, it is noticeable that the first feat that has been achieved is a deep understanding of all the elements and features that compose an Artificial Neural Network (ANN) as well as its workflow and utilities. It has been proven at first hand that even when it is a simple concept, it has really complex ramifications and issues that need a lot of work and effort to be solved, but this has served both as a challenge to learn from and a confirmation of the necessity or at least the helpfulness of the existence of a tool like the one that has been developed. Another element to notice is the amount of work that supposed the construction of not only an ANN but an Editor Tool to create them. The *generalist* aspect of this project has been a greater obstacle than what was expected on the initial phases, however, it has been shown such effort makes the tool way more accessible and usable as well as differentiable from the rest.

On the results matter, the progress that has been shown by the ANNs created from this tool is, at least from my understanding, a success. Agents show a clear learning development when tested and an 'understanding' of the inputs, outputs and consequences of each value.

However, there are, of course, some negative aspects to it. The first thing we can observe is that, sometimes, the agents take a long time to start giving hopeful results. This is an issue since it affects the development of the net, still, it is a complex issue to solve since every decision done at the time of planning and developing the tool has been focused on making a general network that can be used for many situations. Due to this, the agents are slower learners but way more adaptable. A good way to solve this type of situation would be offering the option of creating another kind of net or even another type of training such as the supervised learning method were the user could correct the agent's behaviour as it is trained, a feature that could be implemented on future works.

On the objective matter, now the project has been completed, the initial goals can be compared with the ones achieved (See 1.3. General Objectives and 1.4. Specific Objectives).

6.1. Achieved Objectives

Objective:
To develop a code structure to handle Artificial Neural Networks and their training methods.
Achieved:

An expandable structure for networks, nodes and connections with both core and utility methods to be used by both the tool and the user.

Objective:
To allow any user to set up a customizable Artificial Neural Network.
Achieved:
Easy and understandable way and UI to set up and configure every element on a neural network. It can be also done through code but in a more complex manner.

Objective:
To allow any user to create their own Output Behaviours and insert their Input Values.
Achieved:
Adapted agents into Unity's MonoBehaviour / Scripting system to be easily used by Unity users. Easy and comfortable way for the agents to be input data and return their output values to the user's code (One method to input, one method to output).

Objective:
To create a Unity3D toolset that allows a quick setup.
Achieved:
Use of the Unity's ScriptableObject elements to allow a rapid setup of an ANN as well as a handful of configuration options, all quickly accessed.

Objective:
To create a small mini-game to test the effectivity of the project.
Achieved:
Creation of two mini-games (FlappyBird-like / Runner) to test the networks capabilities.

Objective:
Highly configurable Artificial Neural Networks.
Achieved:
Added many configurable options that can be accessed through the tool's UI. However, way more options could be added in the future such as custom activation methods.

Objective:
Optimized agent behaviour and code.
Achieved:
Lack of some optimization on the 'Multi-Agent' training, noticeable when creating a high number of agents. On 'Single-Agent' the network shows to be working properly without frame drops or high delta times between frames.

Objective:
Functional Artificial Networks able to train and learn.
Achieved:
Networks created by the tool show a learning process and curve able to adapt to the situations they have been tested on. Slow evolution on some tasks but expected due to the network's nature.

Objective:
Universal tool to be used by many genres or even sectors.
Achieved:
Highly general network creation that can be applied on many different projects independently of their nature.

6.2. Future Work

Is finally worth bringing focus on the possible future work that could be put into the tool and new features that could be implemented to improve its purpose.

For future works new configurable options could be implemented, as the tool stands it has already many important values to configure, however this or just values or algorithms to change for the network. A really impactful change that could be made is the implementation of the possibility to create any type of ANN of the ones stated on 2.2. Network Structure. Right now this project can just create Feed Forward type networks, but, with the implementation of important types such as the Convolutional Neural Network or the Long Short-Term Memory network, which are really relevant nowadays, the applications could offer interesting options to developers at the same time that these complex methods would be done more accessible.

Another important feature that would add to the project's value is the ability to create custom algorithms for both Activation and Genetic methods. This was discussed and even the code's structure was, at first, developed to hold that kind of options, but was discarded due to complexity and time issues. Now the time restrain of the project has been lifted, it can be revised and modified to make networks even more customizable.

Also, it can be adapted and optimized for image processing software, different training types, holding memory, use external databases, ...

To summarize, the list of possibilities to add is almost limitless. This technology has been developed for years and this tool is lacking many features compared to state-of-the-art technologies, however, giving the scope it was initially given, this project has been a great success, both for utility and technological results and, especially, on learning purposes.

7. Bibliography & Webgraphy

7.1. References

1 Abadi et al. "*TensorFlow: A System for Large-Scale Machine Learning."* n.d. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf . Accessed 27 Feb 2019.

2 Mahanta, Jahnavi. "*Introduction to Neural Networks, Advantages and Applications.*" n.d. https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207 Accessed 27 Feb 2019.

3 Tainim. "*Elon Musk's Artificial Intelligence gets into videogames to show it can beat human teams.*" *XATAKA eSports*, 21 June 2018. https://esports.xataka.com/xataka-esports/inteligencia-artificial-elon-musk-llega-videojuegos-para-demostrar-que-puede-ganar-equipos-formados-humanos Accessed 1 March 2019.

4 Vincent, James. "*This person does not exist.*" *The Verge*, 15 Feb 2019. https://www.theverge.com/tldr/2019/2/15/18226005/ai-generated-fake-people-portraits-thispersondoesnotexist-stylegan Accessed 1 March 2019.

5 Greene, Katie. *"How AI is used in video games and what we should expect in the future."* *KultureHub*, 29 Feb 2018. https://kulturehub.com/ai-games-artificial-intelligence/ Accessed 1 March 2019.

6 "*Face ID*", *Apple*. N.d. https://www.apple.com/iphone-xs/face-id/ Accessed 1 March 2019.

8 Oludare et al. "*State-of-the-art in artificial neural network applications.*" Elsevier Ltd. 2018.
https://ac.els-cdn.com/S2405844018332067/1-s2.0-S2405844018332067-main.pdf?_tid=1341b4cd-bc9d-4022-9b14-b7aae6f5b11c&acdnat=1552206623_4d687b13a988acada0cf5daa2ab1f139
Accessed 2 March 2019.

9 "*Future of Driving: Autopilot.*" *Tesla, Inc.*
https://www.tesla.com/autopilot
Accessed 2 March 2019.

10 The Tesla Team. "*All Tesla Cars are being produced now have self-driving hardware.*" *Tesla, Inc.* 19 Oct 2016.
https://www.tesla.com/BLOG/ALL-TESLA-CARS-BEING-PRODUCED-NOW-HAVE-FULL-SELF-DRIVING-HARDWARE
Accessed 2 March 2019.

11 McCall, Rosie. "*AI Attempts to write Harry Potter and goes wrong.*" *IFLScience*, 14 Dec 2017.
https://www.iflscience.com/technology/ai-attempts-to-write-harry-potter-and-it-goes-hilariously-wrong/
Accessed 2 March 2019.

12 *Botnik* n.d.
https://botnik.org/
Accessed 2 March 2019

13 *"Feedforward Neural Network."* Wikipedia, n.d.
https://en.wikipedia.org/wiki/Feedforward_neural_network
Accessed 3 March 2019.

14 "*Recurrent neural network.*" *Wikipedia*, n.d.
https://en.wikipedia.org/wiki/Recurrent_neural_network
Accessed 3 March 2019.

15 Sak, Haslim; Senior, Andrew; Beaufrays, Francoise. "*Long Short-Term Memory based Deep Recurrent Neural Network architectures for large scale acoustic modeling.*" 2014.
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf

Accessed 3 March 2019.

16 Donges, Niklas. "*Recurrent Neural Networks and LSTM.*" *TowardsDataScience*, 26 Feb 2018.
https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5
Accessed 3 March 2019.

17 Olah, Chris. "*Understanding LSTM Networks.*" *Colah's blog*, 27 Aug 2015.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Accessed 3 March 2019.

18 "*AlphaGo research.*" *Deepmind*, n.d.
https://deepmind.com/research/alphago/
Accessed 6 March 2019.

19 Rus, Cristian. "*'AlphaGo' the Netflix documentary about what supposed the Google's AI victory against the Go's world champion.*" *XATAKA eSports,* 5 Feb 2018.
https://www.xataka.com/cine-y-tv/alphago-es-el-documental-de-netflix-que-mejor-explica-lo-que-supuso-la-victoria-de-la-ia-de-google-al-campeon-de-go
Accessed 6 March 2019.

20 Vincent, James. "*OpenAI's Dota 2 defeat is still a win for Artificial Intelligence.*" *The Verge*, 28 Aug 2018.
https://www.theverge.com/2018/8/28/17787610/openai-dota-2-bots-ai-lost-international-reinforcement-learning
Accessed 6 March 2019.

21 "*Public Relations.*" *Unity, n.d.*
https://unity3d.com/public-relations
Accessed 6 March 2019.

22 "*The Unity Game Studio Report 2018*" *Unity,* 2018.
https://unity3d.com/game-studio-report-2018
Accessed 6 March 2019.

23 "*Cloud Build.*" *Unity*, n.d.
https://unity3d.com/es/learn/tutorials/s/cloud-build-0
Accessed 6 March 2019.

24 *Git,* n.d.
https://git-scm.com/
Accessed 6 March 2019.

25 "*Version Control.*" *Wikipedia,* n.d.
https://en.wikipedia.org/wiki/Version_control
Accessed 6 March 2019.

26 "*Using Drive.*" *Google*, n.d.
https://www.google.com/intl/es_ALL/drive/using-drive/
Accessed 6 March 2019.

27 "*Feature Driven Development (FDD) and Agile Modeling.*" *Agile Modeling,* n.d.
http://agilemodeling.com/essays/fdd.htm
Accessed 7 March 2019.

28 "*Revenues from the artificial intelligence (AI) software market worldwide from 2018 to 2025.*" *Statista,* 2019.
https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/
Accessed 7 March 2019.

29 "*Convolutional neural network*" *Wikipedia,* n.d.
https://en.wikipedia.org/wiki/Convolutional_neural_network
Accessed 9 March 2019

30 Gupta, Tushar. "*Deep Learning: Feedforward Neural Network.*" *Towards Data Science,* 5 Jan 2017.
https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7
Accessed 9 March 2019.

31 Mikola, et al. *"How to implement the gaussian mutation operator for a genetic algorithm in Java."* 25 June 2011.
https://stackoverflow.com/questions/6275827/how-to-implement-the-gaussian-mutation-operator-for-a-genetic-algorithm-in-java
Accessed 16 April 2019.

32 Khaled, Mariam. "*Artificial Intelligence.*" *American University of the Middle East,* Dec 2017.
https://www.researchgate.net/publication/323498156_Artificial_Intelligence
Accessed 4 Jun 2019.

33 *torch.ch,* n.d.
http://torch.ch/
Accessed 4 Jun 2019.

34 *TensorFlow,* n.d.
https://www.tensorflow.org/
Accessed 4 June 2019.

35 *Caffe,* n.d.
http://caffe.berkeleyvision.org/
Accessed 4 June 2019.

36 *Theano,* n.d.
http://www.deeplearning.net/software/theano/index.html
Accessed 4 June 2019.

37 "*OpenAI Five.*" *OpenAI*, 25 June 2018.
https://openai.com/blog/openai-five/
Accessed 4 June 2019.

38 "*Generative Models*" *OpenAI,* 16 June 2016.
https://openai.com/blog/generative-models/
Accessed 4 June 2019.

39 Weidman, Seth. "d*The 3 Tricks That Made AlphaGo Zero Work.*" *Hackernoon,* 7 Jan 2018.
https://hackernoon.com/the-3-tricks-that-made-alphago-zero-work-f3d47b6686ef
Accessed 4 June 2019.

40 [VIRTUALSTAR] *"Perceptron Unity Asset." Unity Asset Store,* n.d.
https://assetstore.unity.com/packages/add-ons/machinelearning/ann-perceptron-117766
Accessed 4 June 2019.

41 [SWORD-MASTER] *"Artificial Neural Networks Unity Asset." Unity Asset Store,* n.d. https://assetstore.unity.com/packages/tools/ai/artificial-neural-networks-93236 Accessed 4 June 2019.

42 [VIRTUALSTAR] "*ANN&TOOLS Unity Asset.*" *Unity Asset Store,* n.d. https://assetstore.unity.com/packages/add-ons/machinelearning/ann-tools-138836 Accessed 4 June 2019.

43 Bi, Luzheng. "*Using the Support Vector Regression Approach to Model Human Performance.*" *Beijing Institute of Technology,* May 2011. https://www.researchgate.net/publication/220510315_Using_the_Support_Vector_Regression_Approach_to_Model_Human_Performance Accessed 6 June 2019.

44 Sharma, Avinash. "*Understanding Activation Functions in Neural Networks.*" *The Theory of Everything*, 30 March 2017. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 Accessed 6 June 2019.

45 Sharma, Sagar. "*Activation Functions in Neural Networks.*" *Towards Data Science,* 6 Sep 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 Accessed 6 June 2019.

46 Tahmasebi, Pejman; Hezarkhani, Ardeshir. "Application of a Modular Feedforward Neural Network for Grade Estimation." *Natural Resources Research*, 21 Jan 2011. https://www.researchgate.net/publication/225535280_Application_of_a_Modular_Feedforward_Neural_Network_for_Grade_Estimation Accessed 3 March 2019.

47 Solegaonkar, Vikas. "*Convolutional Neural Networks.*" *Towards Data Science,* 18 Feb 2017. https://towardsdatascience.com/convolutional-neural-networks-e5a6745b2810 Accessed 6 April 2019.

48 "*What does Artificial Neural Network mean?*" *Technopedia,* n.d. https://www.techopedia.com/definition/5967/artificial-neural-network-ann Accessed 23 June 2019.

49 "*A Beginner's Guide to Deep Reinforcement Learning*" *Skymind,* n.d.
https://skymind.ai/wiki/deep-reinforcement-learning
Accessed 23 June 2019.

51 Fagella, Daniel; Dr. Bengio, Yoshua "*What is Machine Learning?*" *Emerj,* 19 Feb 2019.
https://emerj.com/ai-glossary-terms/what-is-machine-learning/
Accessed 23 June 2019.

51 "*The Inspector Window*" *Unity Technologies*, n.d.
https://docs.unity3d.com/Manual/UsingTheInspector.html
Accessed 23 June 2019.

7.2. Videography

Vid.1
[Mashable] (31 Aug 2017). *Elon Musk's 'Dota 2' is disrupting eSports in a Big Way - No Playing Field.* Retrieved from URL https://www.youtube.com/watch?v=jAu1ZsTCA64
Accessed 6 March 2019.