Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Creación de inteligencias artificiales basadas en Machine Learning como alternativa a los métodos convencionales

Autor del proyecto: Sergio Jimeno Navarro

Director del proyecto: Antonio Iglesias Soria

Zaragoza, 11 de Septiembre de 2020

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma

Fecha

11 de Septiembre de 2020

Dedicatoria y Agradecimiento

Quiero dedicarle este proyecto a toda la gente que ha aportado su granito de arena en esto, directa o indirectamente.

A mi familia por apoyarme durante toda la carrera y por haberme dejado a mi aire mientras realizaba el proyecto.

A mi pareja, Isa, por aguantarme todos estos años, apoyarme en absolutamente todo lo que me propongo y por ayudarme a mejorar.

A mis amigos de Entalto Studios y 4FreaksFiction, los cuales me tenían que oír hablar de "conejitos" sin entender nada.

Al profesorado de la Universidad San Jorge y a la Universidad en general. Gracias en especial a Daniel Blasco y Antonio Iglesias por todas la ayuda dentro y fuera de clase que me habéis dado.

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations and Acronyms

AI: Artificial Intelligence

ML: Machine Learning

NN: Neural Network

BT: Behaviour Tree

NPC: Non Playable Character

FSM: Finite State Machines

GTD: Get Things Done

Resumen

Hoy en día, la Inteligencia Artificial es uno de los pilares principales a la hora de desarrollar un videojuego en la gran mayoría de los casos.

La evolución de la industria y de la informática en general ha hecho que las nuevas Inteligencias Artificiales necesiten ser mucho más complejas, lo que que a veces conlleva muchísimas horas de trabajo en diseño, implementación, testing e iteración.

Como alternativa, el aprendizaje máquina (Machine Learning) ha empezado a coger fuerza, y cada vez son más los desarrolladores que creen que este es el sistema que sustituirá al desarrollo convencional de Inteligencias Artificiales.

En este proyecto, analizaremos la herramienta ML-Agents, una API desarrollada para Unity con la que se pueden crear Inteligencias Artificiales basadas en Machine Learning, para comprobar si realmente este es el futuro de la Inteligencia Artificial.

Abstract

Nowadays, Artificial Intelligence is one of the main pillars when it comes to developing a video game in the vast majority of cases.

The evolution of the industry and of computing in general has meant that new Artificial Intelligence needs to be much more complex, which sometimes involves many hours of work in design, implementation, testing and iterations.

As an alternative, machine learning has begun to gain strength, and more and more developers believe that this is the system that will replace the conventional development of Artificial Intelligence.

In this project, we will analyse the ML-Agents tool, an API developed for Unity which you use to create Artificial Intelligence based on Machine Learning, to see if this is really the future of Artificial Intelligence.

1. Introduction

There is a lot written about Artificial Intelligences, but one of the best and earliest definitions was coined by John McCarthy [1], a Stanford's University computer scientist, in its article *What is Artificial Intelligence?* (1956), and it says: "*It is the science and engineering of making intelligent machines, especially intelligent computer programs"* [2].

How do we achieve this intelligence? What do we actually mean by intelligent machines? These are questions that the scientific community has been trying to answer for a long time too. In the textbook *Artificial Intelligence: A Modern Approach* (1994), authors Peter Norvig y Stuart J. Russell offered a new way of looking at it, they defined Artificial Intelligence as "*the study of agents that receive percepts from the environment and perform actions*." [3]. If we stop and think about it, humans, the most intelligent creatures found to the date behave like this, we have put most of our animal instincts aside and learnt to use our brain to choose what to do depending on our surroundings and situation.

Most of people value intelligence based on how something behaves, and it is logical that they label something as intelligent if that something behaves in a complex manner from a human-like base. If that something has a behaviour which reaches human standards or surpasses it, it will be considered as intelligent.

We still have a long way until we get computers with something that actually looks like real look alike human intelligence, maybe within our lifetimes but that is not one hundred percent sure [4]. So, how do we do it in video games? How can all those monsters, people, vehicles, everything we see on our screens be programmed and behave like the real deal? Easy, we don't, we just make it look like it.

The trick when creating artificial intelligences in video games is that they do not actually need to be intelligent, they need to look like they are. To achieve that we need to be creative and limit our Artificial Intelligences to do only what they must do, we do not need to make a foot-soldier in a war game to be able to know how to follow a cooking recipe for example.

We must also influence the player's perception by using visual or auditory feedback, to make the artificial intelligence look like is thinking. Another useful tool are randomizers, when we have several possible behaviours to execute or parameters which can be modified by randomizing them we can make the artificial intelligence more authentic for players. Gaming have a really inclusive way to look at AI: "Game AI is anything that contributes to the perceived intelligence of an entity, regardless of what's under the hood" [5].

With this information and purpose in mind, gaming artificial intelligence must do/be several things:

- It must appear intelligent but flawed, so it can look beatable.
- It must not have any weakness so it cannot be exploited repetitively.
- It must be configurable, that way developers can change and adapt it.
- It must be fun, entertaining, engaging, so players like to play with it or against it.
- It must not kill the game. A complex and hard AI can be a good idea, but AIs are costly to create both in terms of time and money, and, if too complex, it might not like the players.

In conclusion, artificial intelligence in video games has a very different objective and purpose than the one science is trying to achieve, in video games is more like a game of charades where the developer must make its AI to look as complex as they need it to look, without wasting resources and without implementing features that won't be notice or won't make a difference but only make the AI more difficult to process for the game.

Don't get the wrong idea, even if video games AIs are simpler that doesn't mean they are easy to create. Video game developers have been creating and implementing new methods and techniques to create more and more complex AIs with time, each one of them perfect for one type of game or another. With the creation of Machine Learning a new gate has been open for the gaming industry, one which may even make the programming of AIs easier for new coming developers.

Through the years we can see a clear evolution in the complexity of both developing techniques and the AIs themselves, beginning from simple scripted behaviours to AIs that learn by themselves how to act in environments and situations that the first developer could not even imagine.

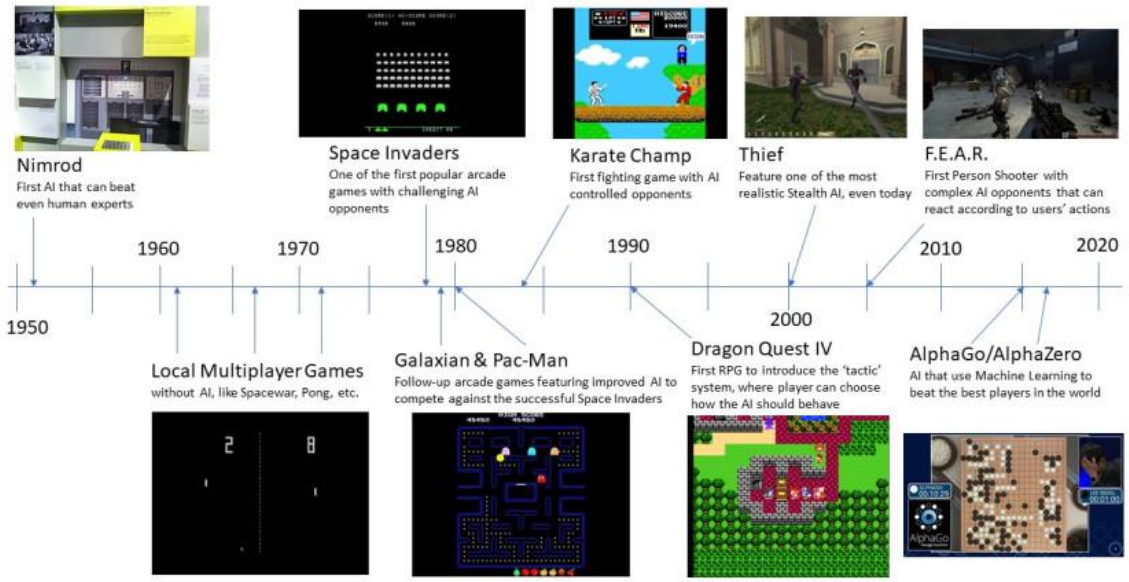


Figure 1: Timeline with the evolution of AIs in video games

But, are the new methods always better? Not necessarily, but in recent years the programming knowledge needed to create complex AIs has reached a point that not everybody can achieve, and maybe new tools like machine learning, will allow new developers to create equally complex AIs as with traditional methods but without the same amount of programming knowledge.

For that reason, in this project, I will create several artificial intelligences using traditional methods and after that I will re-develop them using Machine Learning in order to compared both methods and see which one may be the best option in the future.

2. State of the art

The main objective in AI investigation, as we mentioned, is to create an evolved enough AI that can reach and even surpass human intelligence standards, and this applies in a similar way when developing AIs for video games.

Today we have a lot of AI types with many different functions and objectives, but the original AIs that were programmed in the first video games had one single target, to compete against players, so they needed to be engaging and interesting to make the players keep playing.

Due to the technology of the time, these first video games were really basic in terms of programming, but with the appearance of microprocessors limitations were broken, allowing the industry to grow in complexity and therefore allowing developers to design and program new and more complex AIs for their video games.

The use of randomization was a huge step forward too, making AIs look "alive" and allowing players to achieve unique experiences with each playthrough, but not all can rely on randomization. With time, different techniques and methods were created to create more complex AIs which allow them not only to behave in some way or another, but to set different behaviours and actions and let them plan which ones to execute depending on the circumstances of the games.

Even tough new methods have appear and the technology keeps evolving, it doesn't mean that with every new method that is invented the old ones are no longer used. Each method and technique created is usually because developers are trying to create more complex games, but that does not mean that this new methods work better in simpler games either, so it is not surprise that almost all AI development techniques are still being used in one type of video games or another.

2.1. Scripted Behaviours

First videogames AIs like the one used in *Pong* (1972) were really simple and had only one behaviour, in this case, to move their racket to bounce the ball against the player.

This behaviour was achieved by calculating the current trajectory of the ball and moving the racket to match that trajectory, always, the difficulty setting didn't change this but only the speed of the racket, giving the player the illusion that it was better and maybe even more intelligent.

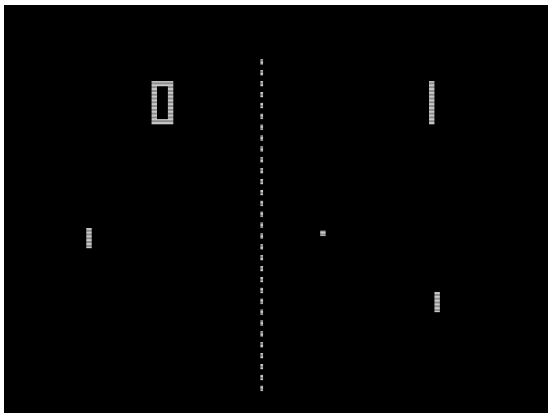


Figure 2: Image of the video game Pong (1972)

This is but an example, early video games used to employ AIs just like this, with just one single function or behaviour which would be oriented to compete against the player. Of course, the complexity could vary, but only within the actual single behaviour. To say it in other words let's use another example, *Pac-man* (1980).

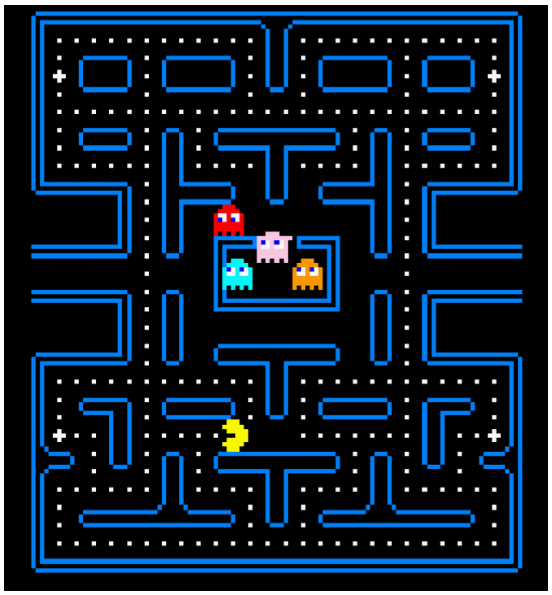


Figure 3: Image of the video game Pac-Man (1980)

Pac-man is a videogame where the player takes control of Pac-man, that little yellow creature in Figure 3, which objective was to eat all the dots in the labyrinth while avoiding the enemy ghosts. This whole map at the same time is divided in invisible cells, which are used to check the position of each element of the game, their status and other elements of the game.

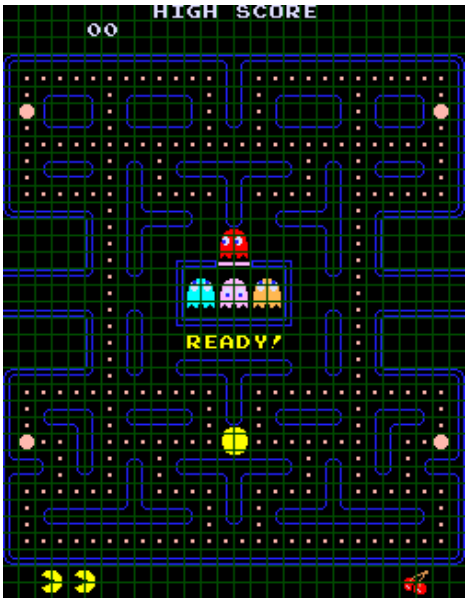


Figure 4: The different cells that divide the Pac-man's labyrinth

This ghosts are pretty good examples on how, even if an IA only have one behaviour, it can be more complex that the one in Pong, not only because the technology advanced but because the game had a completely different objective and therefore needed a completely different AI.

In *Pac-man*, there are four ghosts called Blinky (Red Ghost), Pinky (Pink ghost), Inky (Blue ghost) and Clyde (Orange Ghost). When the game starts, each one of these ghosts start coming out of their "safe room" and each one of them will behave in their own way. Blinky will find the shortest path to the player and follow it, it will chase the player wherever it goes. Pinky will do differently, it will check where the player is facing, get the fourth cell in front of him and go to it, in an attempt to get in front of the player and block his path. Inky will try to be in a symmetrical position from Blinky, in an attempt to surround Pac-man from both sides. And last, but not least, Clyde will move randomly around the map trying to avoid the Player [6].

This was the usual behaviour with one exception. The behaviour of Inky and Pinky were usually as I explained before, but it changed when Pac-man is facing up [46]. In the case of Pinky, for

example, instead of targeting the fourth cell in front of Pac-man he targeted the cell four positions in front and four positions to the left of Pac-man's position. The best part is that this was the result of a bug, it was not the intention of developers for this change of behaviour, teaching us that sometimes even human errors can make AIs more engaging and to look them even more complex.

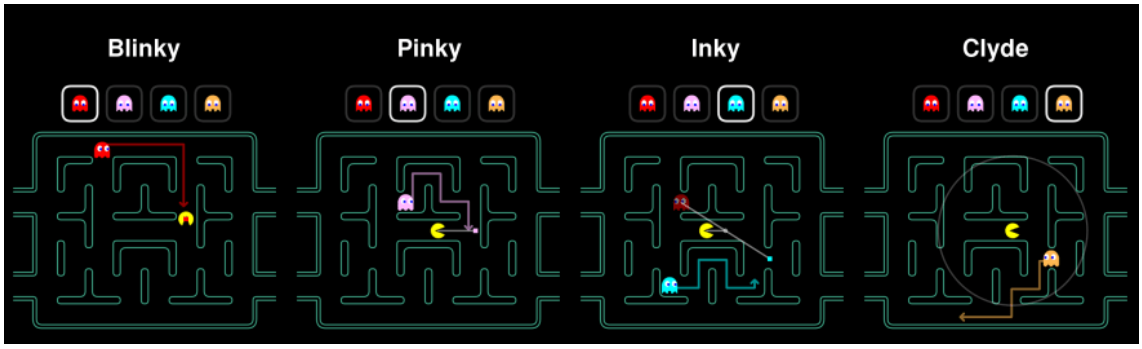


Figure 5: Pac-man's ghosts different AIs

As we can see, each ghost have a different single behaviour, scripted of course, but together they form a completely different experience. As we said, the target here is to simulate an AI, to simulate that they are behaving as living things, that there are something back there much more complex that it actually is, making the player engage and keep playing.

Going back a little bit, we mentioned this game is divided in cells, cells that are used for different things like when Pinky needs a target, but they are also used for a very important part of a lot of video games AIs, the pathfinding.

2.2. Pathfinding and search algorithms

Pathfinding is a process used to find the shortest path between two points. This is usually a vital part in a lot of video games, both for AIs and other mechanics like Search Trees and Behaviour Trees, but we will talk about them later.

Pathfinding uses node graphs, it starts in a specific node (the starting point) and explores adjacent nodes until the one where the destination resides is found. In videogames like Pac-man these graphs take the shape of cell matrixes, where each cell acts as a node.

There are several algorithms that allow us to find the connection between these nodes, being the most common ones the Disjkstra's algorithm and the A* algorithm.

Both algorithms start with the same setting: We have an initial node, a target node, a list of "closed nodes" and another list of "open nodes". The nodes, if connected, will have a determined weight or cost to travel from one to the other. This weight, which nodes and in which are added to one list or another will vary depending on the algorithm.

In the Dijkstra's algorithm we begin with the start node, which we add to the list of closed nodes, and all the nodes connected to it will be added to the open nodes list. From this point forward, the node in the open list with the lowest cost from the start will be added to the list of closed nodes and so on [7]. This method is not used a lot in videogames because it does not take in to account the position of the target to determine which nodes are closer to the target, but that's were A* enters.

A* algorithm works in a similar way, with the main difference that it will give to each node a value equal to the cost of the edge of the node (the deepness/distance from the start node) plus the remaining distance between this node and the finishing node [8]. This distance can be different depending on the heuristic that it has been used in the programming, like the Euclidean distance, which can be tampered in order to get the best heuristic for each program and environment [9].

Pathfinding algorithms are not only used to find actual paths in games like *Pac-man*, it can also be used to find the best resulting route inside search trees, like the ones used in the decision algorithm Minimax [10], used in most of turn-base board games like Chess or Checkers.

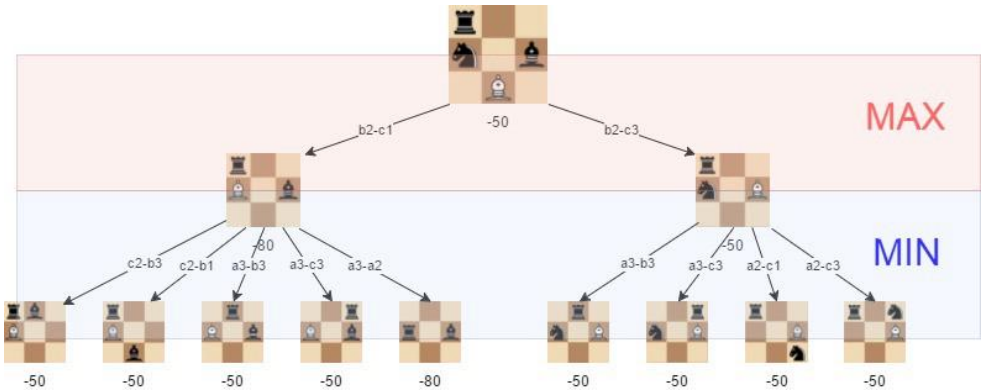


Figure 6: Example of Minimax moves in chess

2.3. State Machines and Planning

A state machine is an abstract machine which can be in one of the finite number of states which conforms it at any given time, reason why they are also called Finite State Machines (FSM).

This machines have the capability to change between states through external inputs in a process called transition. This transition will only happen when the current state machine and its conditions are met.

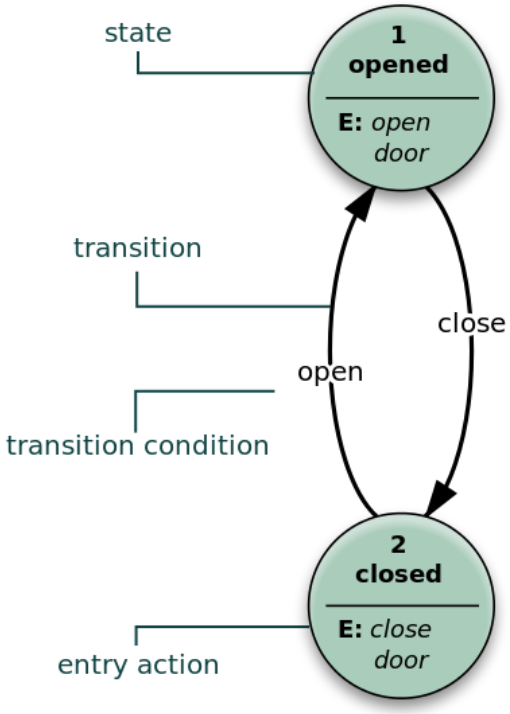


Figure 7: Example of Finite State Machine

In Figure 7 we can see a clear example of a FSM. This machine is composed of two states, Opened and Closed. To change between them we need the proper inputs, we need "open door" when the state is Closed and "close door" when the state is Opened. If both current state and input are the correct ones the state will change through the corresponding transitions of "close" and "open".

In video games, state machines are usually employed to program the behaviour of NPCs. From this point forward is all a matter of adding more states to make the state machine more and more complex, as complex as we need it to be.

A good example is the video game Wolfenstein 3D (1992), in which developers thought of every possible scenario their AIs could be in and created a FSM with each one of the actions needed to cover those situation.

Figure 8: Wolfenstein 3D (1992)

Unfortunately, there will be a point where the state machine is just too complex to follow or even to program in a clear way, and that's where we use planning.

When using planning, each state of a machine state becomes an independent action, decoupled from the rest. This is very useful because it allow us to make the coding modular, we can add more actions and the rest do not need to change.

To make it work, we must give to each action a set of preconditions and a set of effects. This conditions can be anything inside the AIs script, such as Boolean or float variables, and the same goes for the effects. This way, when the correct preconditions are met we change from one action to another, making more complex behaviours and being able to even plan several actions in advance.

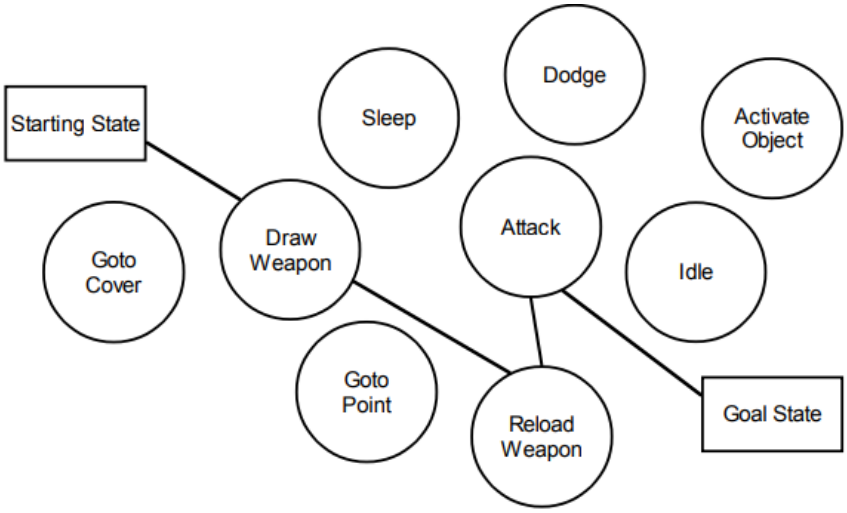


Figure 9: Example of plan. All actions are individual and only the ones conforming the current plan are connected.

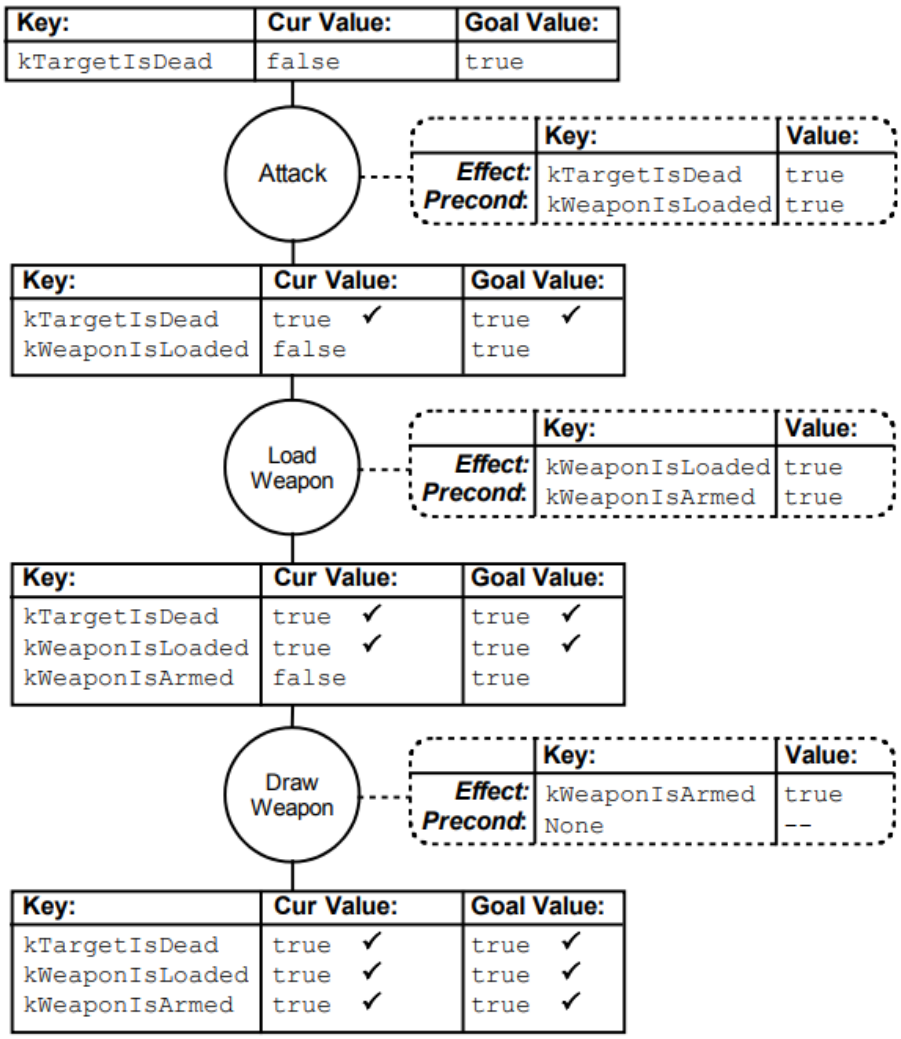


Figure 10: Example of planning with preconditions and effects

2.4. Behaviour Trees

As their name indicates, a behaviour tree is a tree of hierarchical nodes that controls the decision making flow of an AI [11].

Just like regular data structure trees [12], these trees will start in a root node, from which the different branches of nodes will grow. Depending on their position, these nodes can be inner nodes or outer nodes, also called leafs. When talking about behaviour trees, the nodes can be a composite node, a decorator node or a leaf node.

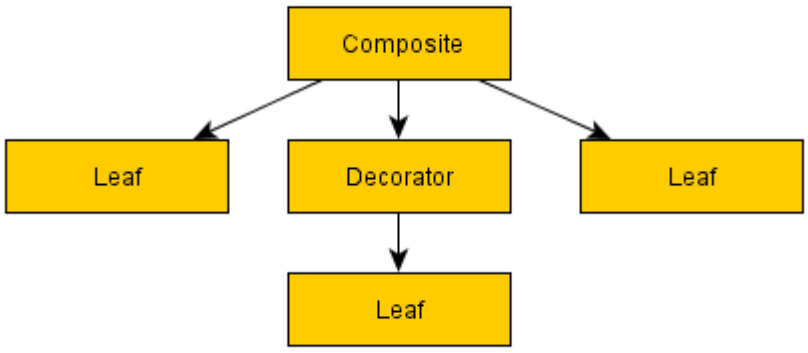


Figure 11: Behaviour Tree Types of Nodes

Composite nodes can have one or more children, and it will pass a signal of success or failure to its parent, a lot of times depending on the failure or success of its child nodes. There are several types of Composite nodes:

- Sequence: It will visit each child node, in order, starting with the first one until one of them fails or all of them have succeed. This is usually used when the AI tries to execute an orderly set of actions which must be followed to the letter in the proper order.

- Selector: This node will try each child node, starting with the first, until one of them returns success. This method is used when the AI have different options at hand and will execute the first one that is available.

Both types can be also be randomized, altering the order in which the nodes are visited.

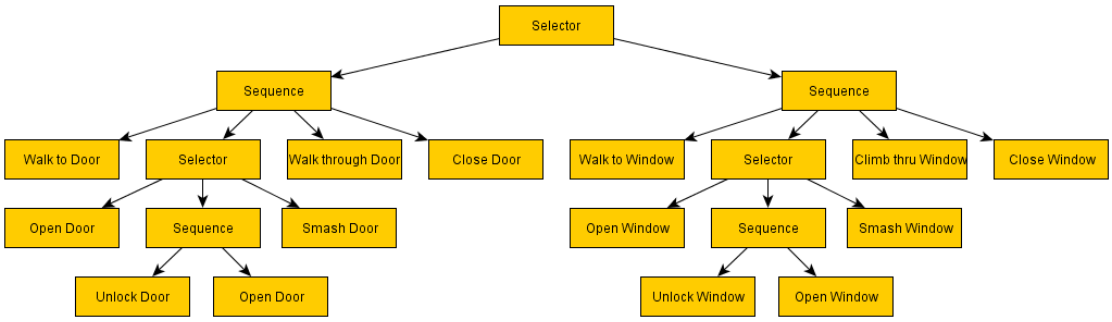


Figure 12: Example of different composite nodes in a behaviour tree

Decorator nodes only have one child and they usually transform the result they receive from their child node. There is also different types of decorator nodes:

- Inverter: This node will invert or negate the result that receives from the child node. It is usually used in conditional tests.

- Succeder: This node will always return success, no matter the result of the child node. This node is useful when we anticipate any kind of failure and we want to keep executing the branch sequence.

- Repeater: This node will reprocess its child node as long as the child returns a result. They are used when we want a behaviour tree to be executed repeatedly, maybe forever or only a set amount of times.

- Repeater until fail: This node works in a similar way as the repeater one, but it will stop executing its child node when this returns failure.

Leaf nodes are the actual actions that the AI will execute. This action of course will depend on the programming, they can be single pass actions like changing a Boolean or an action that will keep the node being executed until failure or success.

If we take for example a node called "Go to Target" we would get a success if the target has been reached, we would get a failure if the target can't be reached, for whatever reason, and we would get running if the AI is still trying to reach the target, making the node to keep being executed.

Behaviour trees can also use planning. When using it, a behaviour tree will calculate a plan and stack the different actions, which will try to execute in sequence through the behaviour tree until completed or something fails, in which case (unless instructed otherwise) it will make a new plan.

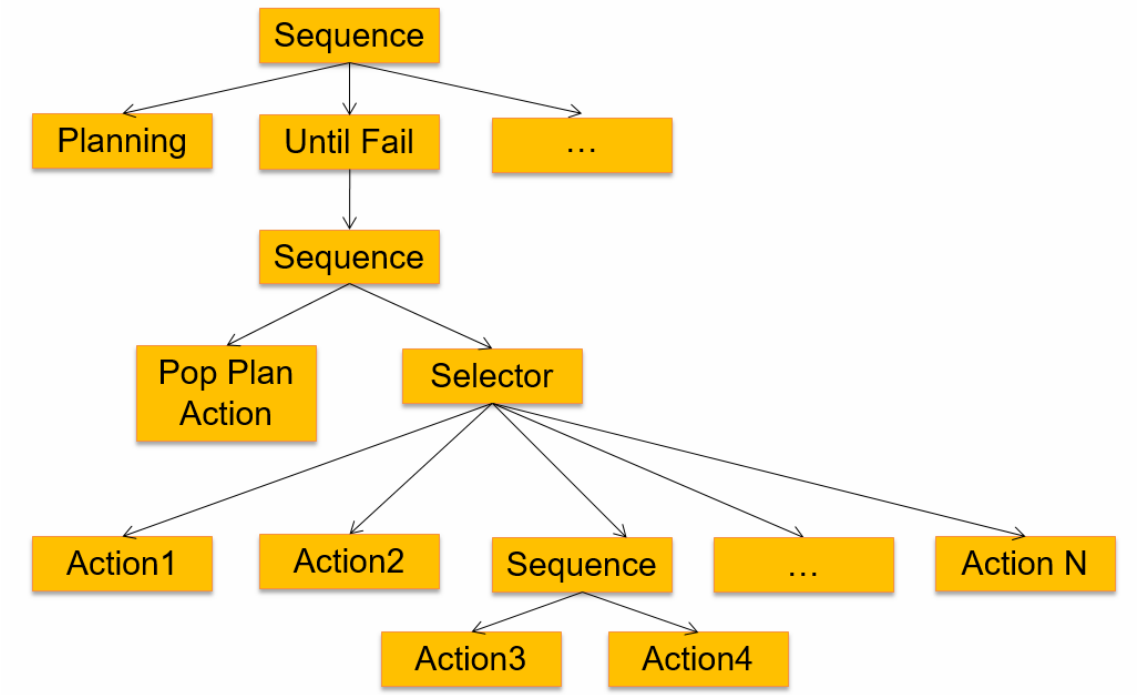


Figure 13: Example of behaviour tree with planning

An example of video game that uses behaviour trees is Alien: Isolation (2014). In this game the Alien will try to hunt the player by following noises, investigating areas, crawling through vents and many other actions, all of them controlled by a behaviour tree. But this game had something new that contributed to make the AI look even more complex that what it was.

Along with the Alien AI there is a Director AI which has information on everything going on the game: location of player, the Alien, control over gates, vents etc. This Director was in charge of evaluate the current state of the game and pace it by telling the Alien where the player is more or less. With that direction the Alien goes towards it and the behaviour tree starts working to find the player [44].

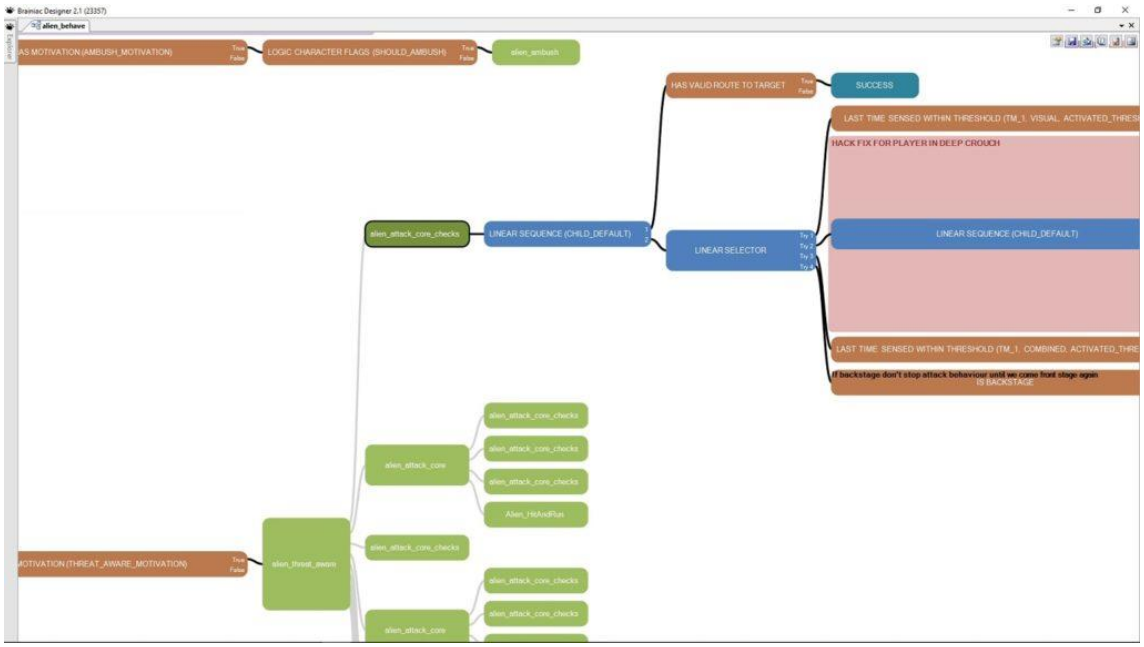


Figure 14: Actual Behaviour Tree used in Alien: Isolation

2.5. Machine learning

Machine Learning is an application of artificial intelligence that gives a system the ability to improve from experience [13]. This is achieved by creating what is called as Neural Networks, some short of artificial brains which we can get through this training process.

There are several machine learning methods like using supervised machine learning, unsupervised machine learning and semi-supervised machine learning, but the one that is used to train video games AIs is the reinforcement machine learning.

The reinforcement machine learning algorithm is a process in which we teach a neural network how to behave by setting a reward system. The agent will evaluate its current situation (variables, attributes, environment, what we want him to take in to account), it will take an action from a set of actions that we provide him, getting a positive or negative reward depending of the results of such action in that situation and environment [14].

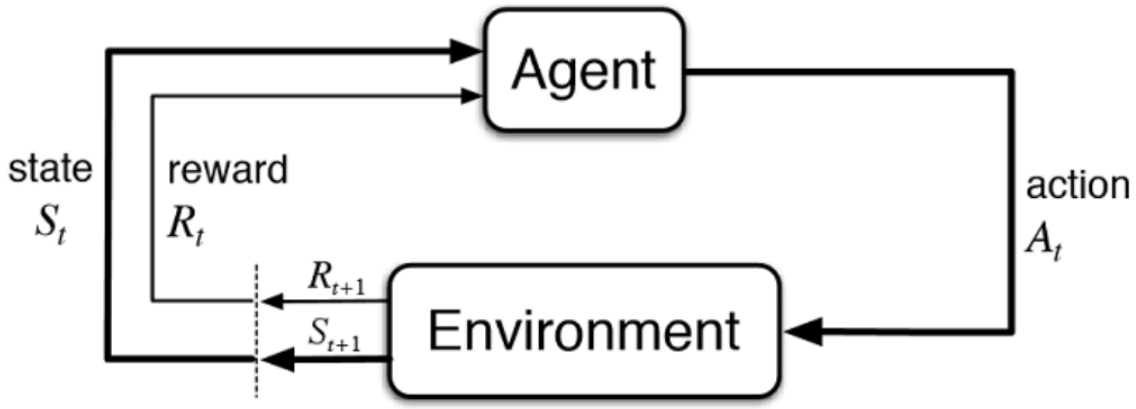


Figure 15: How reinforcement machine learning Works [15]

The agent will learn how to actually behave thanks to something called long-term reward. This reward is the actual goal of the agent, it will try to maximize it as much as it can, and the only way of doing it is by doing the correct actions in the correct situations. The agent will get short-term rewards after each training cycle, which put together will form the long-term reward. It is thanks to this cycles that the agent will start learning when to execute each action, adding at the end of the cycle the resulting short-term reward and evaluating if the cycle has been an improvement or not in its behaviour.

Machine learning has started to not only being used by some developers to create new AIs, but also for some game engines like Unity [16] to create their own system to train AIs as a complementary tool for the developers that use it.

Unity's machine learning tool is called ML-Agents [17], it launched its first full release in March 30, 2020. It is offered to the public as an open-source Unity project and allows developers to train AIs using several methods, like reinforcement learning, neuroevolution, imitation learning etc. by using a Python API.

The good side of this tool is that you don't need to know how machine learning or Python actually work, they have created a system were any person with a minimum knowledge of Unity can train new AIs using machine learning. Their extensive documentation and variety of examples, scripts and algorithms allow anybody to start tampering right away.

The project even includes several starter environments with different kinds of AIs and training grounds, which can be used and modified freely for developers to create their AIs, like the ones in Figure 16 (below). Even if the tool is young and still prompt to bugs and errors, it has proven to be very effective, opening a door to a new kind of development and AIs.

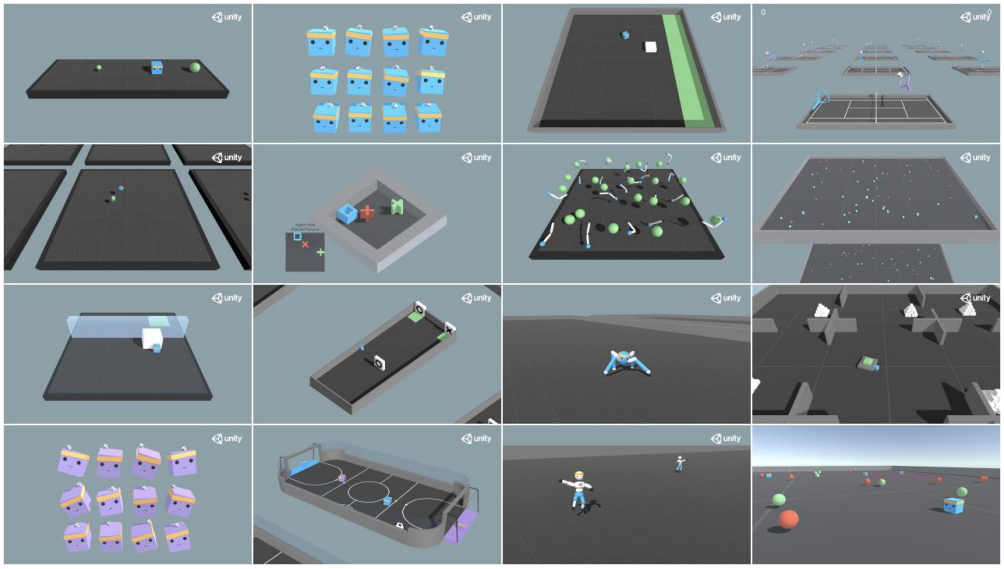


Figure 16: Starter environments

An example of a videogame that has used machine learning (specifically Unity's ML-Agents tool) is *Source of Madness* (2021) [45]. This game has used machine learning to train an AI that generates new creatures (different looking and with new behaviours) each time.

Figure 17: One creature created by the Source of Madness AI

3. Objectives

The objectives of this project are:

1. Study Machine Learning and systems modelling.
2. Implementation of Machine Learning systems in Unity.
3. Creation and implementation of Artificial Intelligence based on Machine Learning with different levels of complexity.
4. Comparative study between Artificial Intelligences trained using Machine Learning and Conventional Artificial Intelligence.

4. Methodology

4.1. Study Phases

This project is divided in two time periods, Pre-Study and During-Study.

The Pre-Study phase goes from January 2020 to July 2020. During this period of time I coursed the class *Artificial Intelligence applied to Video Games* in San Jorge University. In this class I learned core techniques, the role of AI in video games, a little bit of history and background a several AI design and programming techniques to create AIs.

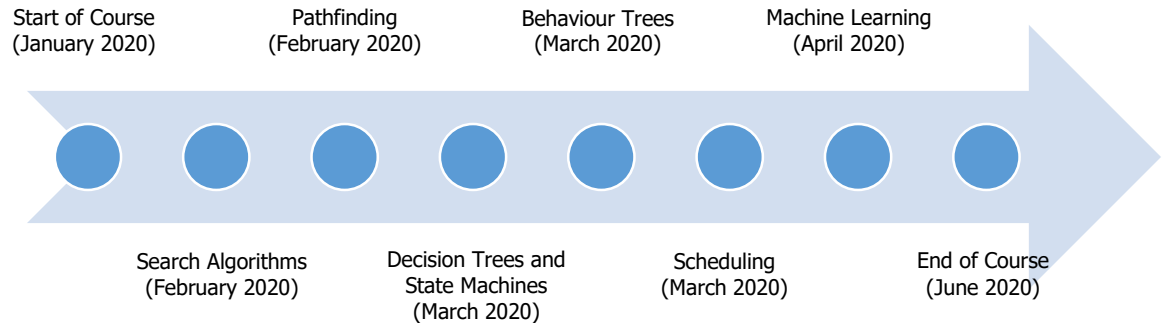


Figure 18: Pre-Study Phase Timeline

The During-Study phase is the work done since the start of this project on June 2020 to its delivery in September 11th 2020. This phase had a duration of 350 hours of work, divided in 24 hours each week. At the same time, the project was derived in four stages: Preparation, Development of AIs with conventional method, Development of AIs with Michele Learning and Documentation. The amount of time spent in every stage was: 48 hours of preparation, 224 implement both types of AI and 72 doing the documentation.

Preparation phase: On this phase I prepared all the materials and assets I required to make the demo work. That includes to create the project, to create the materials and prefabs that are used on the project, develop the different scripts that are used to organize and make everything work, like different managers and other kind of scripts, and finally the importation of asset from the internet and past projects.

Development of AIs with conventional method phase: In this phase I took everything I learnt in the AI course I took and will apply it in the first set of AIs by trying several methods and techniques that may work in order to get the desired AIs.

Development of AIs with machine learning phase: In this phase took place the most experimental part of the study. In this phase I downloaded the several repositories of Unity's ML-Agents tool, install them and the different programs needed to make them run. After having everything in place, I created again the same AIs that I did but using Machine Learning, which took several iterations.

Documentation phase: In this phase I wrote this document. The document has been slowly written during the whole duration of the study. When the final AI was done, the remaining parts of this study were written, therefore leaving this document as finished and ready to present.

4.2. Working methodology

The working methodology employed in this study was *Get Things Done* or GTD [17]. GTD is actually a time management method, which consists in dividing our work in the different parts or tasks that conform it and start doing them one by one. When a new task appears, whatever the reason, is taken in to the pile of tasks and leave there until it is taken care of.

This method might seem lazy or inefficient, but that was not the case neither for me nor this project. This study has already a very natural established order: Design and create an environment four our AIs to train and be, create the conventional AIs, the machine learning AIs and compare them in this study. Each one of these big tasks can be divided in smaller tasks, the creation of the environment for example can be divided in creating the scene, placing the elements the AIs will interact with, obstacles etc. Each one of those actions are independent, they will form a whole but they do not rely immediately in one another, so it's easy to just start doing work non-stop and then assemble everything.

This method is not recommendable when working in teams, using GTD was only possible because I had a global vision of how the whole project was going to be, so I subconsciously knew which order to follow to get the best results.

4.3. Project structure

The project will follow a very define hierarchy to keep it organized. This project will use assets from the internet and other personal projects to which we will add new elements which we will to create in order to have this project working.

The main project will have six main folders: AISubject, AssetStore, ML-Agents, OriginalEcosystem, Resources and TFG. The content in them is the following:

- AISubject: Inside this folder we will have everything I did for the different works and tasks that I had to do during the AI course.

- AssetStore: In here I will keep all the assets I get from the Unity Asset Store [18].

- OriginalEcosystem: Inside this folder I will store several assets and props from an old project, like the animals, materials and several scripts.

- Resources: In here I will keep the prefabs of every animal that we implement so we can access them from any script.

- TFG: In here I will store all the elements I created for the project itself, like new scripts, prefabs, materials and the Neural Networks that work.

We must mentioned that, in order to train the Neural Networks, we will need a secondary project, one of Unity's ML-Agents repositories. This secondary project will be called from the command console so we can use the ML-Agents libraries and train our Neural Networks, keeping the different versions we train inside the project along with the hyperparameters of the training.

4.4. Coding Format

For the format of our code I will use some of the coding Guidelines of the video game developing company Kraken Empire plus my own. I adopted most of them during an internship with them. Besides, the teacher of the AI course I took is one of their CEOs and main programmer, so the code he provided us for the subject also follows that format, an extra reason to keep it.

- All variables will use names that are self-explanatory, regardless of length, as opposed to one-character names.
- All variables will be define in the beginning of the script, group by functionality.
- Public variables will use description tooltips to show in the inspector.
- All functions, structs and enumerators will start with uppercase.
- Functions brackets will be keep in the same line the function is defined.

5. Economic Study

In this point we will cover the economic cost of this entire essay, including software licenses and assets, hardware materials and the cost it would take to pay human resources for the development of this study in time and testing.

For the hardware I used the following elements:

- Computer HP Z1 Entry Tower G5 [19] – 800.88€
- Acer Full HD (1920 x 1080) IPS Ultra-Thin Zero monitor [20] – 75.94€
- VicTsing Wireless Mouse and Keyboard [21] – 19.40€

Total: 896.22€

Even if the hardware's specifications might seem a bit too much I decided to spent a little bit more of money in a powerful enough computer to run the neural networks training without any problems and be able to have a lot of AIs working together at the same time without any problems. Of course this won't cover any programming mistakes that might kill the computer (like infinite loops without exit conditions) but at least it won't give us problems product of long periods of times of training.

We must take into account the durability of the hardware too. Today's computers can work for several years and this study was no longer seven months. If we give the computer a lifespan of five years, due to its top quality, we should count for the study only a fragment of the actual cost. Instead of the cost of seven months we will calculate the cost of a year to round up the numbers, which would leave the total cost of the hardware in 179.25 €

During the Pre-Study phase of the project I took two courses, the course about Artificial Intelligence applied to Video games and an online course about Machine Learning, the costs of each one were:

- Artificial Intelligence applied to Video games by San Jorge University [22] – 954.00 €
- Reinforcement Learning: AI Flight with Unity ML-Agents by Adam Kelly [23] – 59.99€

Total: 1013.99€

Now we must check how much it would cost to hire a Software Developer to do this full study in the amount of time it took.

As we mentioned, in the Pre-Study period I took a course of Artificial Intelligence applied to video games, which had a duration of 150 hours plus 100 of individual work. And the online course about Machine Learning took around 40 hours, making a total of 290 hours of work between the two courses.

The development of this study can be divided in four stages and two periods of time. The stages would be Preparation, Development of AIs with conventional method, Development of AIs with Michele Learning and Documentation and the time periods would be Pre-Study and During-Study

The During-Study phase is the work done since the start of this project on June 2020 to its delivery in September 11th 2020. The project was derived in four stages: Preparation, Development of AIs with conventional method, Development of AIs with Michele Learning and Documentation. The amount of time spent in every stage was of: 48 hours of preparation, 224 implement both types of AI and 72 doing the documentation. Making a total of 350 hours of work.

If we sum up both periods of time we get around 640 hours of work. According to *Indeed* [24], the average salary of a Software Developer in Spain 16.86€ per hour, to which we need to add a 30% due to Spain's Social Security cost, resulting in 21.92€. If we take the total amount of hours it took for the completion of this study (640 hours) we get a cost of 14028.80€ total in that period of time to pay our Software Developer.

The resulting cost of this study through its development is the following:

Concept	Price
Hardware	179,25 €
Courses	1.013,99 €
Software Developers Salary	14.028,80 €
Total:	15.222,04 €

Table 1: Total cost of the study

6. Study Development

6.1. Summary

This study will consist in the implementation and comparison of two pairs of different technique developed based artificial intelligences, one pair being developed with conventional AI techniques and the other pair using the reinforcement machine learning technique.

With these AIs we will try to build a small ecosystem, being each pair of AIs a couple of animals, a rabbit and a fox, which will be placed in an environment with the shape of a forest where they will need to eat, drink and reproduce in order to survive and maintain a viable population of their species. We might mention the fact that one thing is to teach them how to survive individually, even how to interact with each other, but it can turn very difficult, and maybe impossible, to accurately simulate a whole ecosystem where both animal populations are kept alive by themselves and do not end up extinct.

Both animals will need, as I mentioned, to do three main actions, which are eating, drinking and reproducing. To eat they will need to find food, go to the food and eat it, same goes for water and mating company.

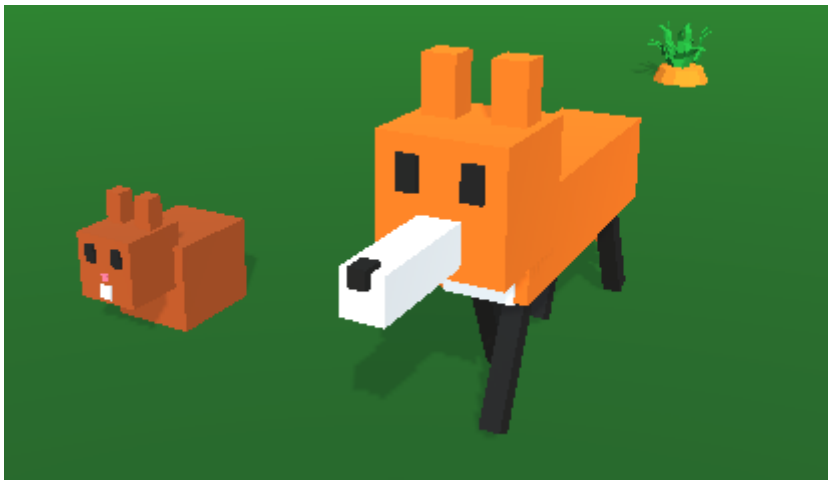


Figure 19: Models for the rabbit, fox and carrot

The only main difference between both AIs will be that the rabbits will eat carrots, inanimate objects which will not move, and the foxes will eat rabbits, that means other AIs, so they will need to chase them. Elements like how much time will each animal spend alive and how many children it will spawn when reproducing will be different too, but those variables will only affect the ecosystem as a hole, not the AIs themselves (in theory).

6.2. Unity and its tools

For the development of this study I will be using the game engine Unity. This is a very powerful tool and is the one I have been using the most in both academic and professional careers. Before diving in the actual development of each one of the tests it will be better to explain some key aspects of how Unity works and how we will use it.

Unity uses something called Scenes, the scenes are virtual environments, spaces where we will place all the elements that we will need in order to make our demo work, think of them as the different levels of a video game. In this study we will create twelve almost identical scenes, one for each type of locomotion system that we will implement.

The elements that will be placed inside the scene are called GameObjects, this elements will be everything that will conform the world, our forest in this case: The trees, the rocks, the light, the water… everything. GameObjects can be inside one another, constructing what is called as hierarchy. When inside a hierarchy, an object can be a parent and/or a child, depending on their position inside it.

Each GameObject will have different elements inside of them, which will give them different properties. Among those elements we should mentioned the most important for our study:

- Transform: This element keeps track and lets the user manipulate the position, rotation and scale of the GameObject. When inside a hierarchy, the transform properties will be relative of the parent. All GameObjects have a Transform.

- Rigidbody: This element gives physics properties to the GameObject, such as mass, air drag and even to be influenced by gravity.

- Collider: The colliders could be considered as the "skin" of the GameObject, they limit the bounds of it. It can have basic shapes such as spheres, cylinders and capsules, but they can also have the shape of a 3D mesh. Both Rigidbody and collider are needed for GameObjects to collide physically with one another.

- Scripts: Scripts are pieces of code which can affect almost every property of the GameObject and its elements. They are used to move GameObjects, create interactions between them and every behaviour and event that may happen in a video game.

- NavMesh Agent: This element allows a GameObject to use Unity's pathfinding algorithm to go from one point to another in our scene. To make it work, we must add a Navigation Mesh on the scene with the walkable areas that it will contain for our GameObject to displace on.

As we mentioned, scripts are used to control almost every aspect of our scene and GameObjects, this is done with the help of Unity Functions. This functions work as any other programming function that we might create, but there are part of Unity's engine and they tag specific aspects and attributes of the elements that form GameObjects. Functions like Transform.Translate and Rigidbody.AddForce for example will be used to move our GameObjects around the scene in the project, along with others which will have different functionalities.

Another important part will be Raycasting. Raycasting consist in creating an invisible ray between two point, which will detect and report collisions with any collider that happens to be between them, this is very useful to detect obstacles.

The next things that we will use are Assets. Assets are downloadable packages that you can get from multiple places like the Unity Asset Store or other websites. This packages, which can be free or not, contain different elements inside depending on what they offer, they can contain 3D models, scripts, sound effects etc.

Several repositories and code from the AI course will be used as well, this repositories will provide us with specific functionalities for pathfinding, planning and behaviour trees, but we will talk about them more when we use them. These functionalities will use their own terminology and system to work, but they are not part of Unity so we will dig more on them later.

All these elements will be mentioned along this project, with their corresponding bibliographic references, please feel free to look them if any further information is required for their understanding.

6.3. Environment

The idea of creating an ecosystem with different animals popped up in my head months before starting this study or any of the courses I took related to AI, it just seemed like an interesting idea. Before creating the AIs what we needed is the environment, a forest.

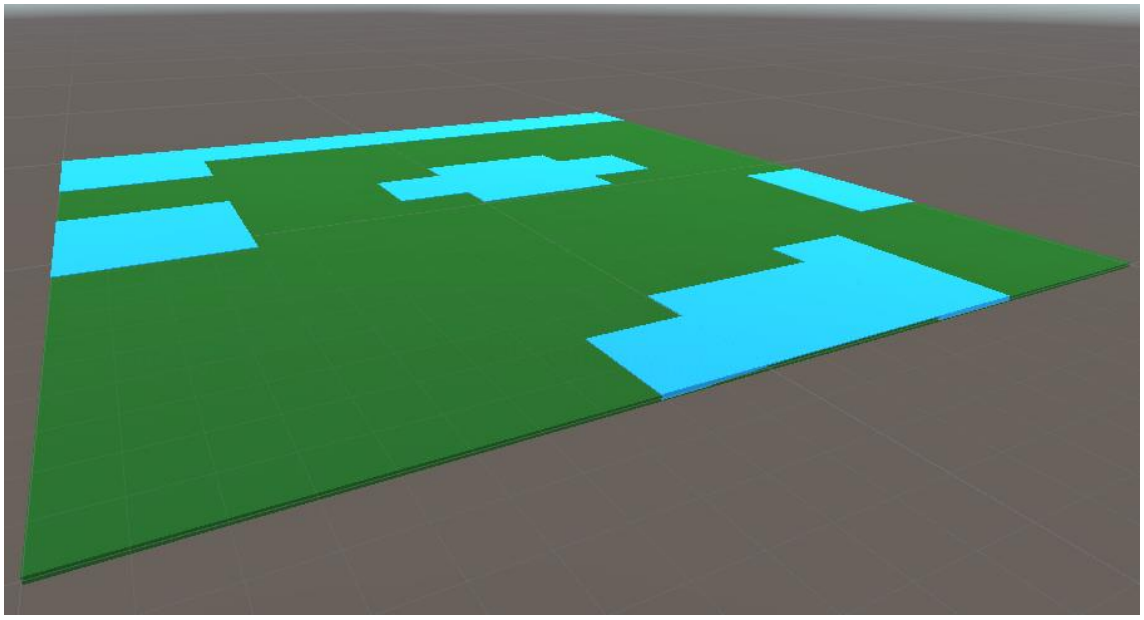


Figure 20: Forest Environment, first version

The first version of the forest was like in Figure 20, I placed a green plane to simulate the ground and placed several bodies of water, formed by different size cuboids of blue colour. The next step was to fill the land with everything I needed, so I took a free asset from the Unity Asset Store called *Nature Pack (Extended)* made by Kenney [25]. This asset had several trees and bushes

which I used to populate the forest, with the bushes acting as possible sources of food for herbivore animals.

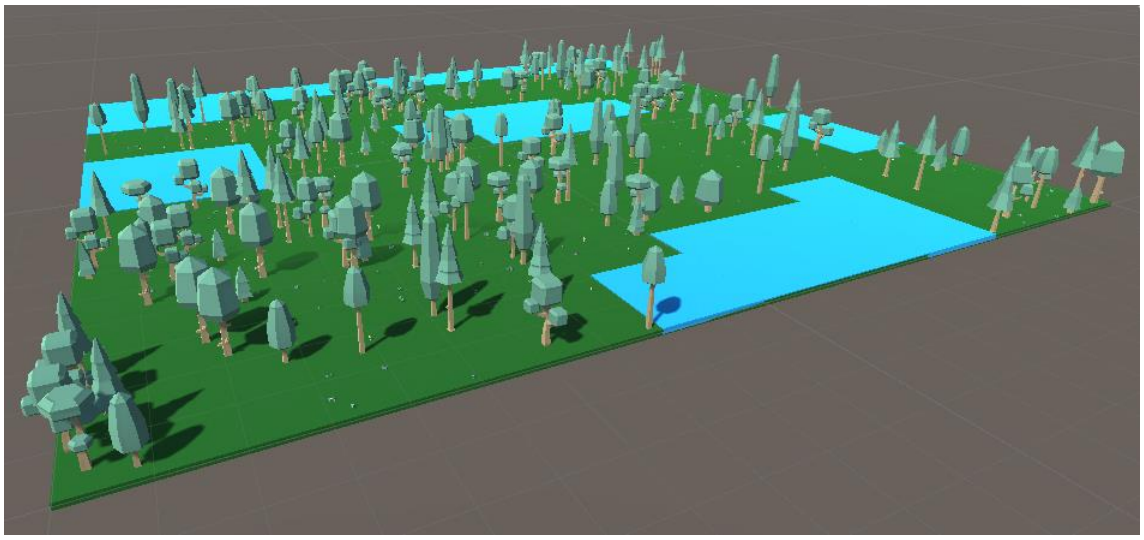


Figure 21: Forest Environment populated

To make the forest look more natural I wrote a small script which would take the surface of the scene, get a certain number of random points and place the amount of bushes and trees that I desired. To avoid overlapping I used raycasts [26], pointing to the ground in the random points, if those raycasts hit ground and not water or other trees and bushes meant that it was a viable spawning point, and placed the prefab there, making it look like Figure 21.

When I started implementing my first and more rudimentary AIs I noticed a huge problem with this design of my environment, the water. Each body of water is formed by a group of cuboids, each one with their own centre, and when I signalled my AIs to reach one of them I used these centres as the target. The problem is that this target could be in a whole completely different direction that the closest border of the water, so the animal would not reach the closest coordinate of the map with water. This can be solved by triangulating which point of the area of the cuboid is closest to the AI and set that as a target, but I had to leave the project for unrelated reasons until I started the actual study a year later.

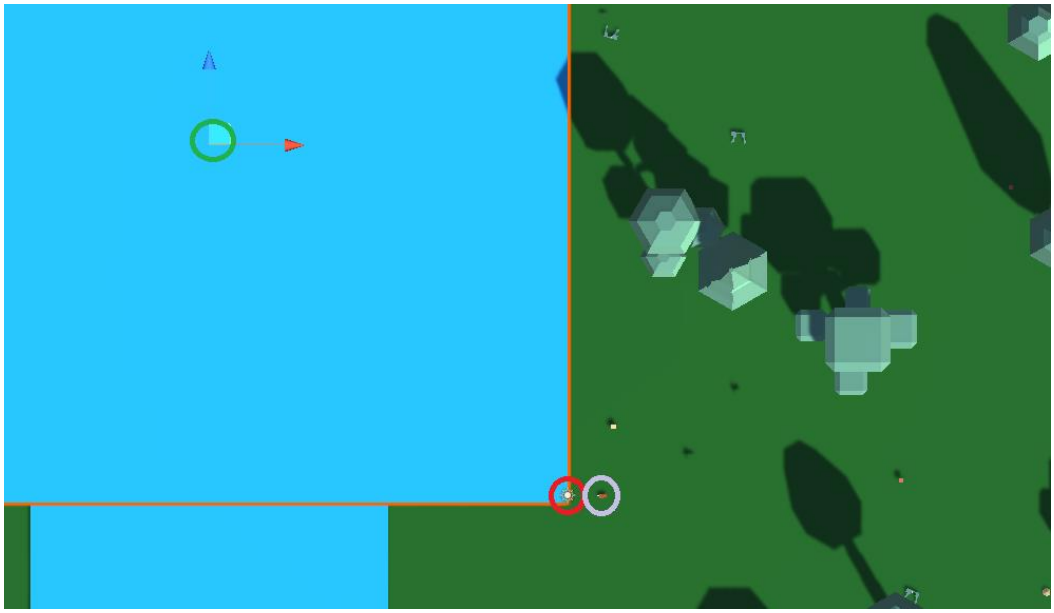


Figure 22: The problem - Where the rabbit is (Pink), where the rabbit should go (Red) and where the rabbit actually goes (Green)

Later, when I started this study, I took almost everything I did for this demo, I considered it a good base to start, and unfortunately there were several problems to deal with first.

The first problem was with the nature pack asset I was using. It happens that it is completely deprecated, and new versions of Unity make this asset to malfunction, so all my nature prefabs were useless and I needed to find new ones. After searching a little bit a found another free asset in the Unity Asset Store with everything I needed called *Polygonal Foliage Asset Package*, made by Aligned Games [27].

The next problem I solved was the one with the water. Instead of making things difficult and spent too much time working I just decided to sacrifice the aesthetics of the scenario a little. Instead of big bodies of water I just put several single and smaller cuboids in random points of the environment. This solution won't look as beautiful as the original design, but it will be completely functional.

Once those problems were solved, I started creating the actual map. The first thing I did was to make the map a lot bigger, that way I could have enough room for a lot of AIs at the same time and to train them when using Machine Learning. I limited the whole map with rocks and got into the next thing, populating the map.

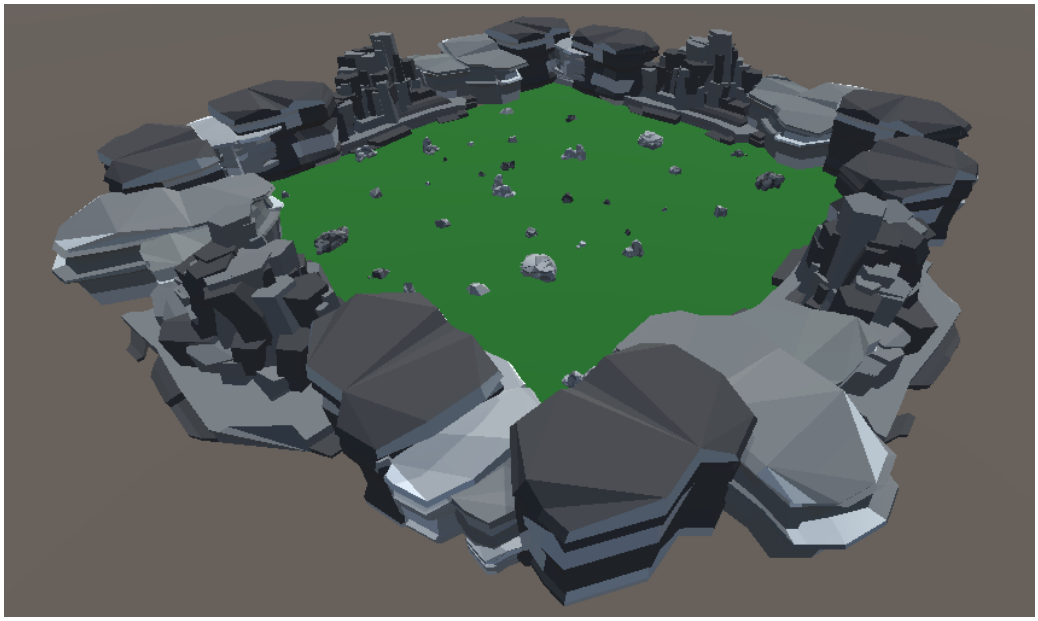


Figure 23: The new setting for the environment

I used the same random script I designed in the first version, but this time I did not spawned everything at the same time and I used another useful tool for it. In the AI class we learnt to use Pathfinding with some scripts which would create a grid of cells on top of the map. This tool was really useful because it also detected if there was any object in the cells, marking them as "unavailable". With this tool I just had to choose random points on the map, check the cell on that point, check if it was empty and if that was the case then I added the object.

Figure 24: Example of unavailable cells (Red), available cells (Grey) and water cells (Blue)

The first thing I added was the water because of its importance, and the next thing was the props. The asset provided me with several extra nature props like more types of trees, stumps, logs, flowers… so I decided to add them to the randomizer and have a much richer environment for the AIs to work around.

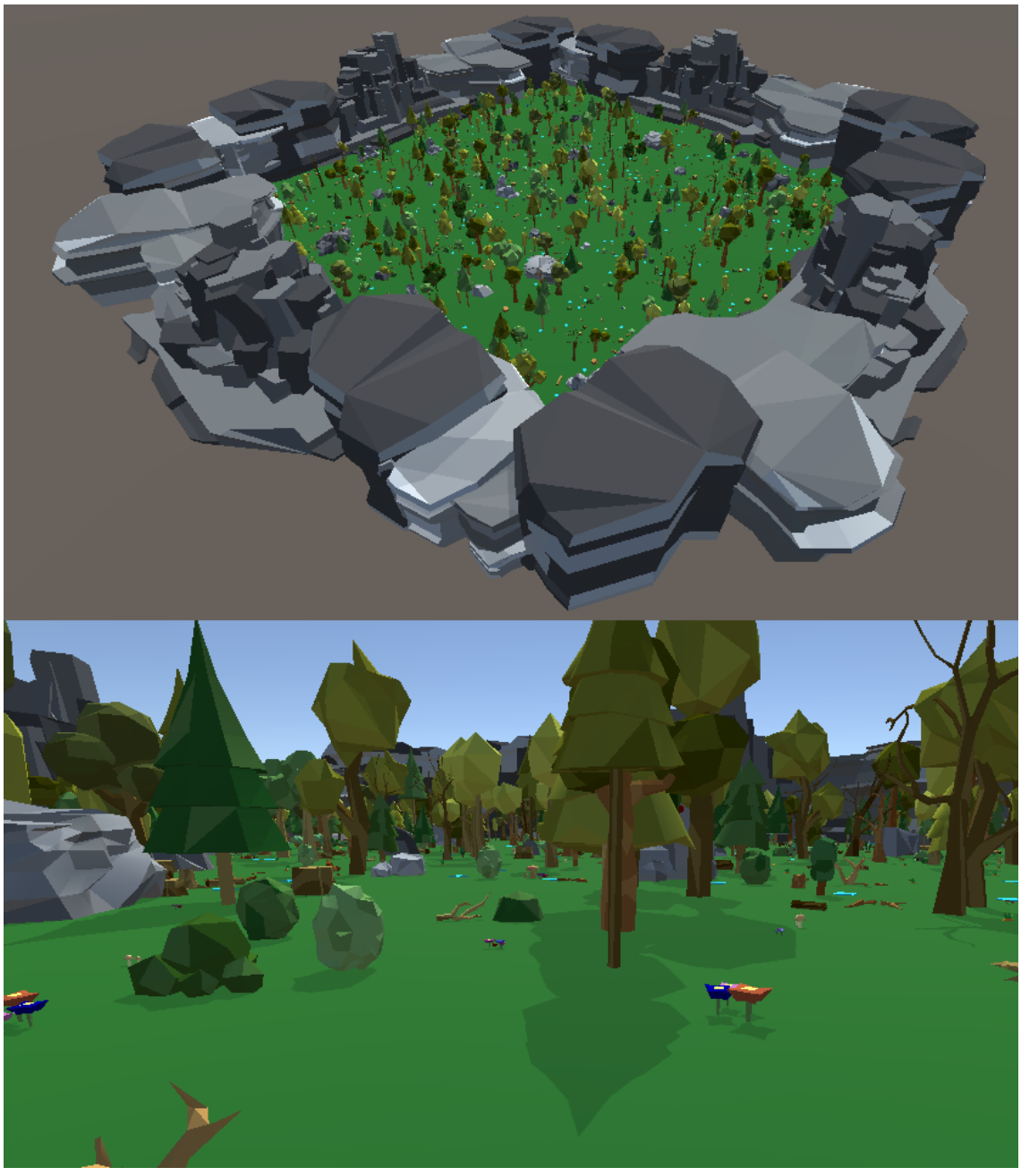


Figure 25: The final environment

After placing everything in order I just had to check that there was no overlapping between any body of water and the props, to avoid any type of problems in the future.

Figure 26: The different animals and carrot

As part of the old project I also created several animals using Unity's 3D geometric shapes and colouring them resulting in the animals in Figure 26. I kept the rabbit and the fox for the study and I created a carrot for the rabbits to eat, using a couple props from the Nature Pack by changing their size and colour to make them look like the different parts of a carrot.

6.4. AI development roadmap

After tampering with several methods in the AI course I notice two things: Unity's pathfinding system is way more optimized than the one I was using from the course and our AIs will use Behaviour Trees but no planning.

Our AIs will try to behave like animals, that means that they will try to fulfil their needs at any time when these change, for example, if an animal is more hungry than thirsty it will go eating, but if in the process of finding food the thirst gets higher it will stop searching for food and will go drinking. Planning won't be necessary because animals don't have long-term plans, they just need to know what they need at this very moment, so planning is not useful for these AIs. Besides, when you have a lot of AIs planning at the same time it is inevitable to find that your computer resources will quickly disappear.

The next thing I decided was to divide the AIs in different versions, each one with a different behaviour tree. In the case of the fox for example the first version will just eat, drink and wander when he is not hungry or thirsty. The second version will have a new branch with a scape function to escape from any nearby predators. The third and final version will add a reproduction function, completing the AI. The different behaviour trees are available in Annexes 1 and 2.

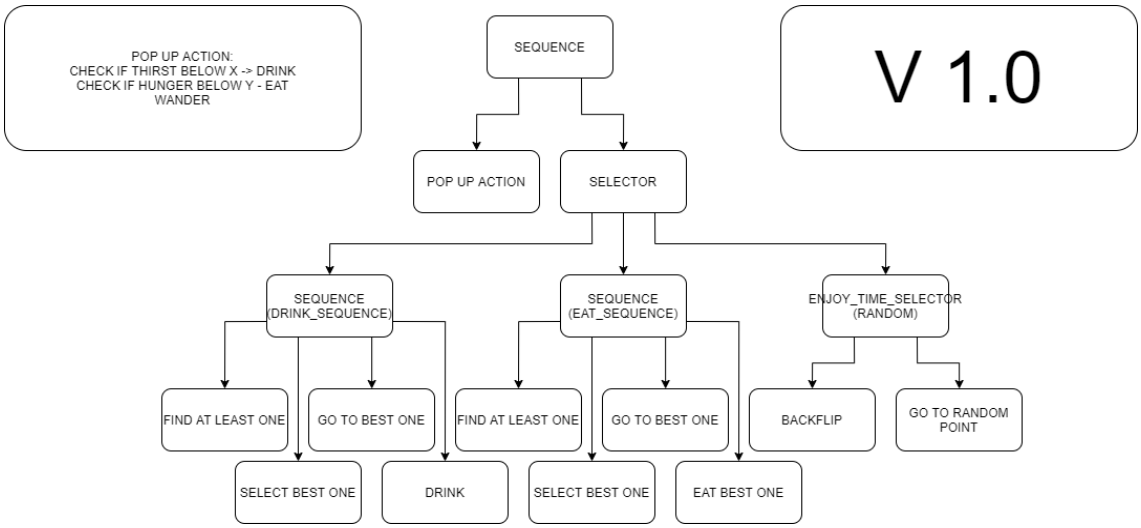


Figure 27: Rabbit Behavior Tree - V1.0

As we can see, the AIs will rely their behavior on each other, so this version-based development method will be useful and it will define the order in which we create each AI. Here is the roadmap for the development of each AI using traditional techniques, behavior trees in this case.

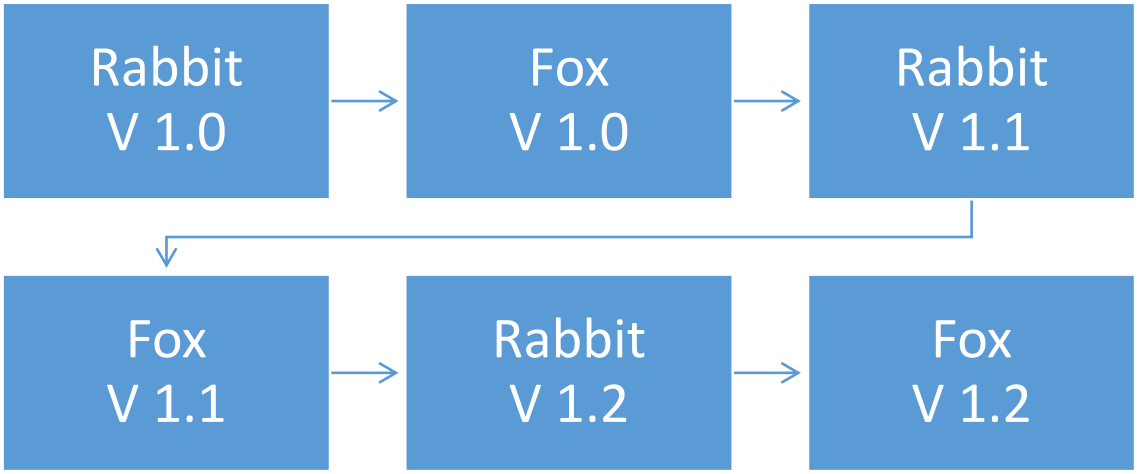


Figure 28: AI development roadmap

For the development of AIs in Machine learning I wanted to do a similar plan but I found that machine learning is far more unpredictable than I anticipated so I decided to make each AI individually and adapt them after each training was completed until I had the results I wanted.

6.5. Development of AIs using Traditional techniques

6.5.1. Introduction

As I mentioned before, the tool I ended up using for my AIs were behaviour trees, but this was only the result of trying other techniques before and deciding which one suited this type of AI the best.

As we can see in the roadmap in Figure 28 and in Annex 1 (Rabbit, V1.0) the first AI will be a rabbit with the ability to find food and eat it, find water and drink it and wander around the map or doing backflips when he is not hungry or thirsty. Each one of those actions will be triggered by several variables in our code, called **hungry**, **thirsty** and **happiness,** these three variables will be floats with a value between 0 and 100.

These variables will have a very specific functionality inside our AIs:

- Every second we subtract one unit from the happiness variable, one from the hunger variable and two from the thirst variable to simulate the biology of the animal.
- If the hunger is below 50 the animal will search for food.
- If the thirst is below 50 the animal will search for food.
- If the animal gets thirsty when already hunger it will get override and will search for water instead.
- When eating or drinking the animal will recover several units in the corresponding variable.
- If both hunger and thirst variables are above 50 the animal will wander or doing something "funny" (backflips in the case of the rabbit), which will increase the happiness variable.
- If any of the three variables reach 0, the animal dies.

Happiness will also be the determining variable on when the animals will be able to reproduce. An animal with a happiness value above 50 will be interpreted as healthy and able to find mating

companions. If that's the case, and a bit of time has passed since the last act of reproduction, the animal will be able to procreate.

Another variable will be used for the animal stats, called **timeToDie**. This variable will start with a value of 120 and decrease one unit at each second, killing the animal when reaching 0. It can be considered the life expectancy.

These values were and are still provisional, first we need a system that allow us to determine in which situations the animals will make one action or another, finding the correct values will come later to determine how the animals can keep the ecosystem alive.

With the stats finished, the next step was to make the animals moving. As I mentioned, one of the tools that the AI course provided me was several scripts to understand and use pathfinding by using the A* algorithm. This system would create a grid of cells on top of our map, marking cells filled with obstacles as "unavailable". With the A* algorithm, we would use the free cells to calculate a path between two points, for example between the rabbit and food.

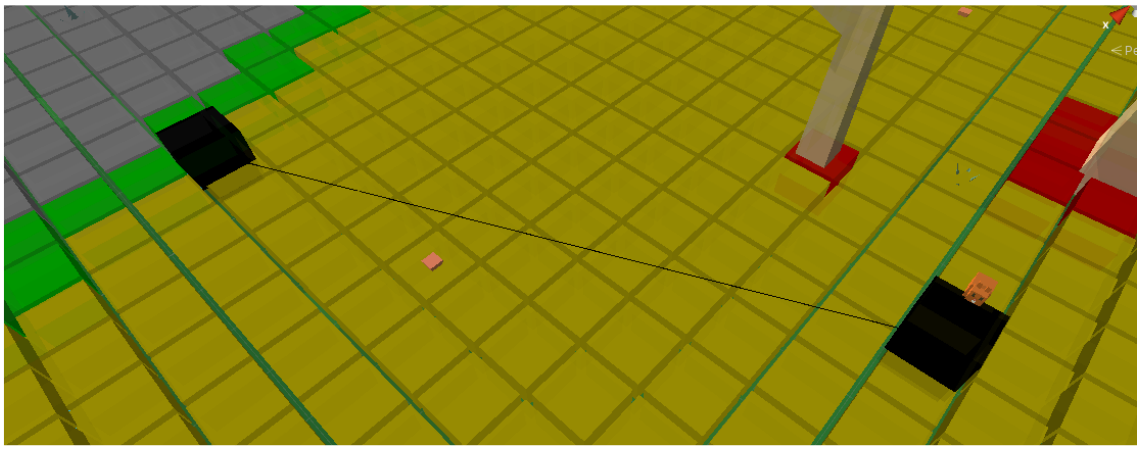


Figure 29: Example of pathfinding

The first technique I tried to use for the AI implementation was planning. To make planning I defined more actions that the ones I ended up with, and implemented them inside a search tree, which we would run through using our A* algorithm. The problem we faced with this system was that it took too long to get a plan.

Usually, in planning, you start in Action A, and you try to find the closest or more optimal path (whatever your heuristic is) inside the search tree to action B within the search trees limitations. Unfortunately our AIs would not look very real using that technique, because animals do not actually have plans.

When I implemented it, the script would explore the search tree for X deepness level instead of searching a plan from action A to action B. This method would get me an exponential number of possible plans, from which we would choose the one with the most optimal outcome, which would be the one were the animal got its stats at their higher values.

ACTION	DESCRIPTION
ACTION_TYPE_WANDER	Waste time, it increases happiness
ACTION_TYPE_GO_TO_WATER	The bunny goes to a body of water
ACTION_TYPE_ARRIVE_TO_WATER	The bunny arrives to the water
ACTION_TYPE_DRINK	The bunny drinks from the water
ACTION_TYPE_GO_TO_PLANT_FOOD	The bunny goes to a carrot
ACTION_TYPE_ARRIVE_TO_PLANT_FOOD	The bunny arrives to the carrot
ACTION_TYPE_EAT	The bunny eats the carrot

Table 2: Set of actions for planning

The problem was that because of this, I took too much time to find all the possible plans, and with dozens of AIs trying to plan at the same time the simulation would get temporarily stuck every few seconds. Because of this, I decided to go only with behaviour trees.

For the creation of behaviour trees I used NPBehave [28], an event driven Behaviour Tree Library for Unity. This library allowed me to use some of their already built behaviour trees and start creating my own, by using the same types of nodes I mentioned in section 2.4 of this study. The resulting scripts, attached our future prefabs, will be called **FoxBehaviourTree** and **RabbitBehaviourTree.**

The second problem I noticed aster running the first versions was that the pathfinding system that I was using cost too much to process, maybe because it was oriented as a learning tool more than for actual implementation. Instead of it, I used Unity's Navigation System [29], which only

required to determine which surfaces are walkable and which not (the obstacles) and is way more optimized for this, improving the processing time by a lot.

Figure 30: The Navigation map

6.5.2. Rabbit Prefab and RabbitBehaviourTree

Our rabbit prefab will have three main elements in his hierarchy. The parent GameObject and main prefab called RabbitPrefab, its child RabbitCollisions and its child's child Body, each one with a determined function.

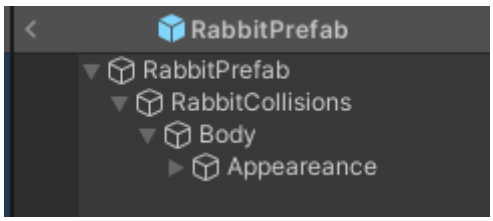


Figure 31: Rabbits prefab hierarchy

RabbitPrefab contains three components inside, the **RabbitBehaviourTree** script, the NavMesh Agent [30] component and a box collider [31] with the same dimensions as the actual body of the rabbit. The **RabbitBehaviourTree** script will be tasked with the overall behaviour of the prefab, it will store and update all the animal stats and will change its state depending on those stats and the environment. The NavMesh Agent component will be the one which will calculate

which path is the best one in order to reach the point we command the agent to go to. The box collider will be used for other AIs to see the agent.

The RabbitCollisions prefab will have three components too. Another box collider, a Rigidbody [32] and a script called **RadiusDectector**. These three elements will cooperate together for one single purpose, to be the animals eyes. The box collider will have the desired vision range we want to give our rabbit and the **RadiusDectector** script will be the one in charge of deciding if the things the collider detects are of interest or not. If they are things like water, carrots or foxes the script will add them to a list and remove them as soon as they get out of the vision range. The script will also update in the same way the lists inside the **RabbitBehaviourTree** script, so it is able to understand what there is around it.

The last element of the prefab is the Body. This is the element of the GameObject that we will move locally inside the hierarchy to make the rabbit look like bouncing without altering its collision boundaries.

Coming back to the **RabbitBehaviourTree** script, let's talk what it actually does. This script will manage all the stats mentioned before, losing or gaining in value depending on the action that is being carried. Inside of it there will be the behaviour tree using our NPBehave library.

```
new Action((bool wanderAction) => {
    if (nextAction == ActionPlanning.ActionType.ACTION_TYPE_WANDER) {
        bounceBool = true;
        if (checkForNewPlan) {
            limitTimePerPlan = 0f;
            checkForNewPlan = false;
            if (thirst < 50f || hunger < 50f || predatorsInSight.Count > 0) {
                needNewPlan = true;
            }
        }
        if (dead || needNewPlan) {
            needNewPlan = false;
            return Action.Result.FAILED;
        }

        bool actionCompleted = WanderAction();

        if (actionCompleted) {
            return Action.Result.SUCCESS;
        }

        return Action.Result.PROGRESS;
    }
    else {
        return Action.Result.FAILED;
    }
}) { Label = "Wander Action" },
```

Figure 32: Example of action using NPBehave library

As we can see in Figure 32 we see how an action would be programmed, in this case the "Wander" action which only target is to find a random point and go to it. The **bounceBool** variable will determine if the rabbit does the bouncing animation or not, in this case we turn it to True because it will bounce.

After that we need to check if a new plan is required. Every X seconds we will check if the rabbit's priorities have change (not all the time to save resources), if so a new plan will be asked to the **Pop_Action** node.

Once we do that we will check if the rabbit is either dead or needs a new plan and if so we will return the node as failure, exiting and restarting the behaviour tree. On the other hand, if nothing bad has happen, we will call our function **WanderAction**, which will find a valid random target point inside a limit area and move the rabbit to it using the NavMesh Agent component, returning true or false depending on if it has reached the target or no.

If the target has been reached the action has been successfully completed, otherwise it means the rabbit is still on the move, therefore we are still in progress of doing the action until further change.

This one of the most basic actions that we can find on the script, the rest follow a very similar structure adapted to each branch and action of the behaviour tree of the rabbit.

In those branches we can find the sequences to eat, drink, scape and reproduce, each one of them will rely in what the rabbit can see on its surroundings. These elements will be stored in four different lists called **foodInSight**, **waterInSight**, **predatorsInSight** and **sameSpeciesInSight**. All of these elements of this lists will be added from the **RadiusDetector** script when it detects them. All of the elements will be removed as soon as they are out of their line of sight but the water, after all the water is not moving and the animal can "remember" where the last source of water was in order to survive.

A full view of the resulting behaviour tree of the rabbit can be seen in the build-in debug window provided by the NPBehave library can be seen in Figure 33 (next page).

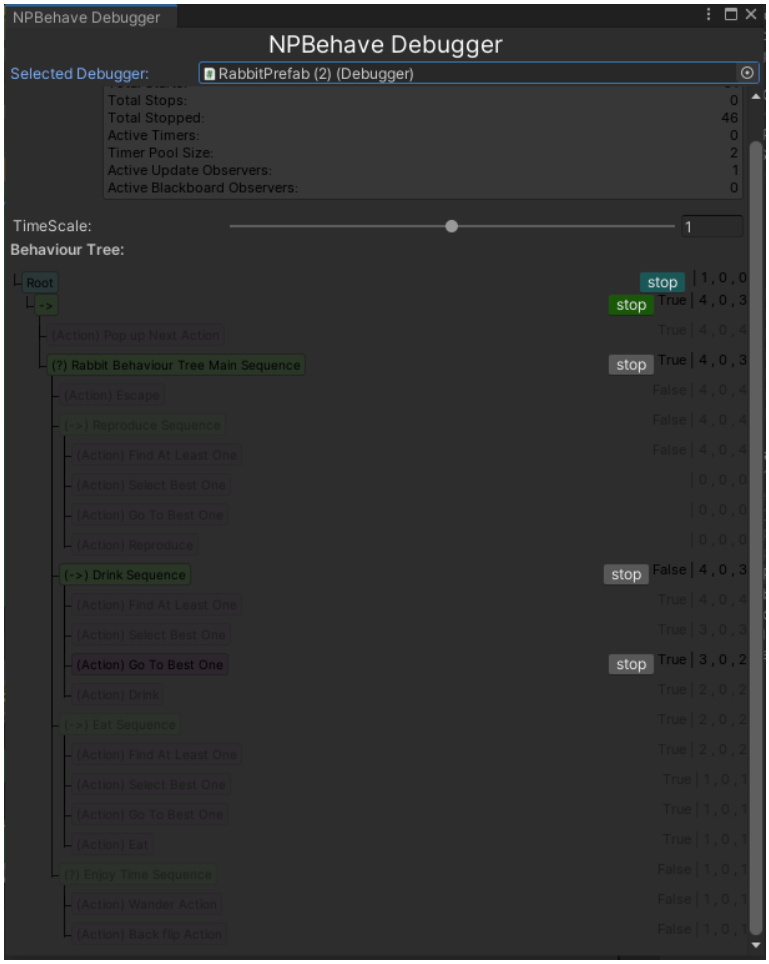


Figure 33: Debug window with the rabbit behaviour tree in runtime, trying to go to the closest body of water

6.5.3. Fox Prefab and FoxBehaviourTree

After proving effective with the rabbit prefab, the same process and almost identical behaviour tree was implemented for the fox prefab, with only a few noticeable changes:

- The name of the behaviour tree script will be, as mentioned, **FoxBehaviourTree**.
- Its displacement functions have been adapted to use moving GameObjects (the rabbits) as targets, increasing in such case the speed of the NavMesh Agent to simulate running.

- The **RadiusDetector** has been updated, introducing a system that detects which type of behaviour tree the prefab has (fox or rabbit) so it detects rabbits as food and other foxes as same species.

- Instead of bouncing we will change the animation. The legs of the fox will be rotated over time in angles of 90 degrees using sinusoidal functions, making the fox look like if it was walking/running.

- The stats will also have different values to simulate the biologic difference between the two species, like giving them more life expectancy or lowering the amount of cubs the foxes can have when mating.

6.6. Development of AIs using Machine Learning

6.6.1. Introduction

The tool that used for the machine learning process is called ML-Agents. ML-Agents is an open source project with its own libraries that allow developers to design and train Neural Networks using some of the most common machine learning techniques.

The first step was to install everything. This has been for a lot of people the hardest step of all and not because it is difficult, but because of its dependencies. After cloning the project from their repository [17] the next step is to install Python 3.6.1 [33] or a higher version. Unity's ML-Agents uses the Python API for the training, and we might have Python already installed in our computer from other programs, causing conflicts because of the different versions and forcing us to uninstall everything and re-install it.

Once we manage to have Python correctly installed, we need to create a Virtual Environment for our trainings using the command console (Anaconda Prompt), which I called **ml-agents1**. This environment will need to be activated every time we need to train our agents because it will be there where we will also install the mlagents Python package using the command "pip3 install mlagents" [34]. Once all of this has been done, we can begin with the training of AIs.

The most important part of ML-Agents is the Agent itself. The agent is our prefab, an actor that observes and takes actions in our environment, in our case our rabbits and foxes in our forest.

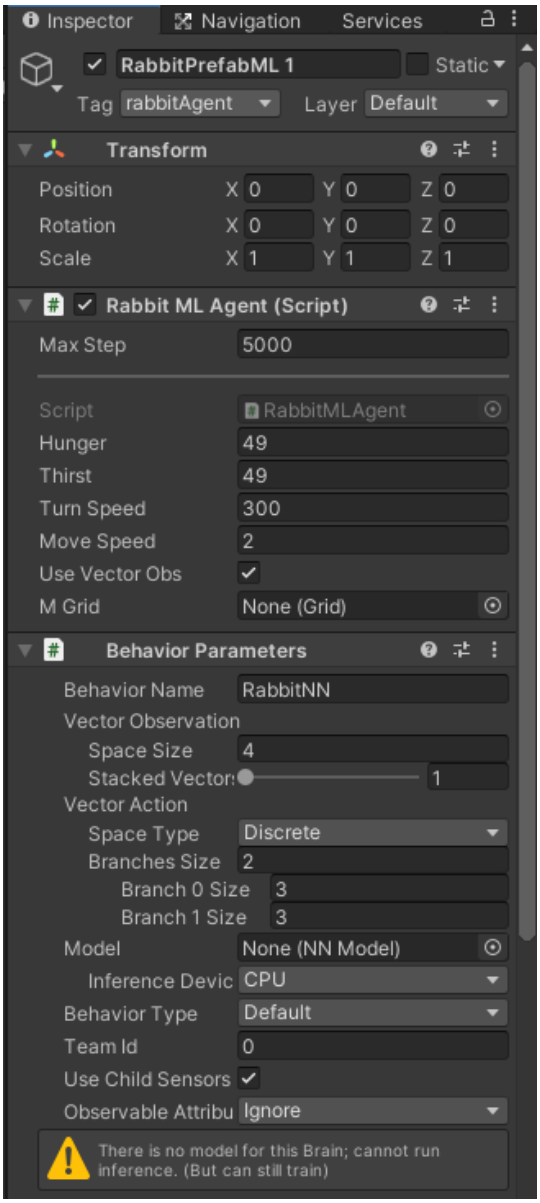


Figure 34: Rabbit Prefab using ML-Agents components

As we can see in Figure 34, our agent has several components which will be vital for its functionality. The first component is the script **RabbitMLAgent**, this script will determine the overall behaviour of the agents (the rabbit in this case), its stats, the maximum number of steps that can occur before the agent's training episode ends and more important, the reward system etc. It will be derived from the class Agent, and therefore adding to our agent a new component automatically, the script **BehaviourParameters** [35].

The **BehaviourParameters** script is very important, in here we will determine which type of behaviour the AI will have. If we pay closer attention in Figure 34 we will see several critical elements:

- Name: This will be the name the Neural Network will adopt once it is created.

- Vector Action: Agents will be able to choose from and array of actions which one to carry. This Vector array will be used to store the information of such actions. For example, if the actions were rotation on the X and Z axis, we should need a space size of 2.

- Vector Observation: Before doing any action or decision, the agent will observe the world's state. This state can be either its surroundings or even internal stats of the agent and must be stored in here. For example, if we need to store two whole vectors, we will need a space of 6.

- Model: This is where we will put the Neural Network. If a NN is place here the agent will follow what it learnt from its trainings, if empty it will mean that the agent is going to train a new NN (or continue one).

- Behaviour Type: Depending on this option the agent will one system or another for the decisions making process. If a NN is provided it will use Inference, otherwise it will use Heuristic, unless we make him choose one of them by force.

Besides of the agents, a manager of some sort is required, which will manage the agents and the environment if needed. We won't be creating one, instead of that we I modify one of the examples provided by the ML-Agents project, the FoodCollector script, which I will explain later in this study.

Once we have our environment and agents, we can begin the training by placing one or more agents on the scene. We must make sure that the presence of several agents does not affect the training, because it could end up in them competing against each other against our will. To avoid that we can create copies of the environment inside the scene and make each one of them train in their own space.

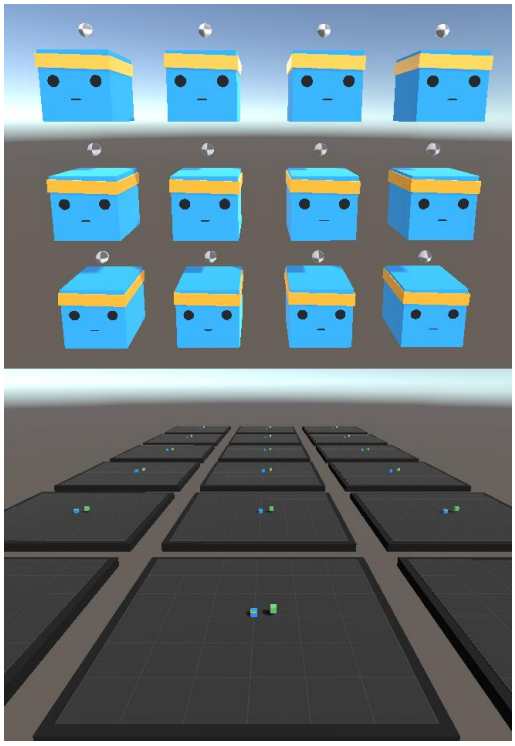


Figure 35: Agents training together (Up) and agents training in separated environments (Down)

The next and last component our agent will need to use is the Ray Perception Sensor 3D component [36]. This component will substitute our old box collider and **RadiusDetector** script that we used in our AIs with behaviour trees, it will be their eyes. This script will throw a series of rays with a desired length, which will be able to collide and detect elements from the environment. These detectable elements will be GameObjects which tags must be defined inside a tag-list, otherwise they will be ignore. For example, en Figure 36 (below), we can see our rabbit rays having collisions with trees and rocks but not bushes,

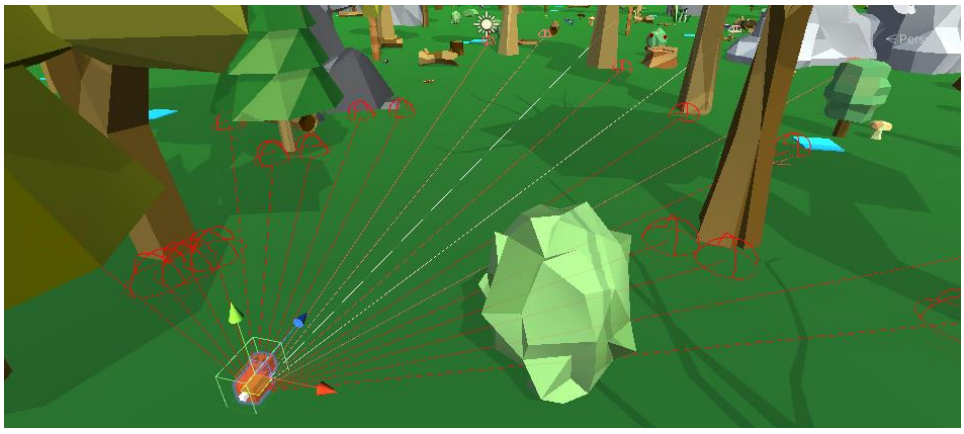


Figure 36: Example of Ray Perception Sensors (Red colliding, White not colliding)

Once we have everything ready we can begin the training. For that we must open our command console like Anaconda Prompt and activate the virtual environment in which we keep our mlagents installed. After that, we must write a command line with a specific structure [37].

Let's use an example like "mlagents-learn config/ppo/3DBall.yaml --run-id=first3DBallRun". In this command line we can see the three main parts of the command:

- mlagents-learn: This is the actual command which will start the training sessions.
- config/ppo/3DBall.yaml: This is the path to the training configuration file. Obviously it will change depending on which folder we are using inside the command console and where our file actually is inside our computer.
- run-id=first3DBallRun: This will be just the name we will give to the training session, in this case first3DBallRun.

If everything is going according to plan, the command console should tell us to start the training by pressing the start button in the Unity Editor. Once we do that the training will run until finished or until we stop it.

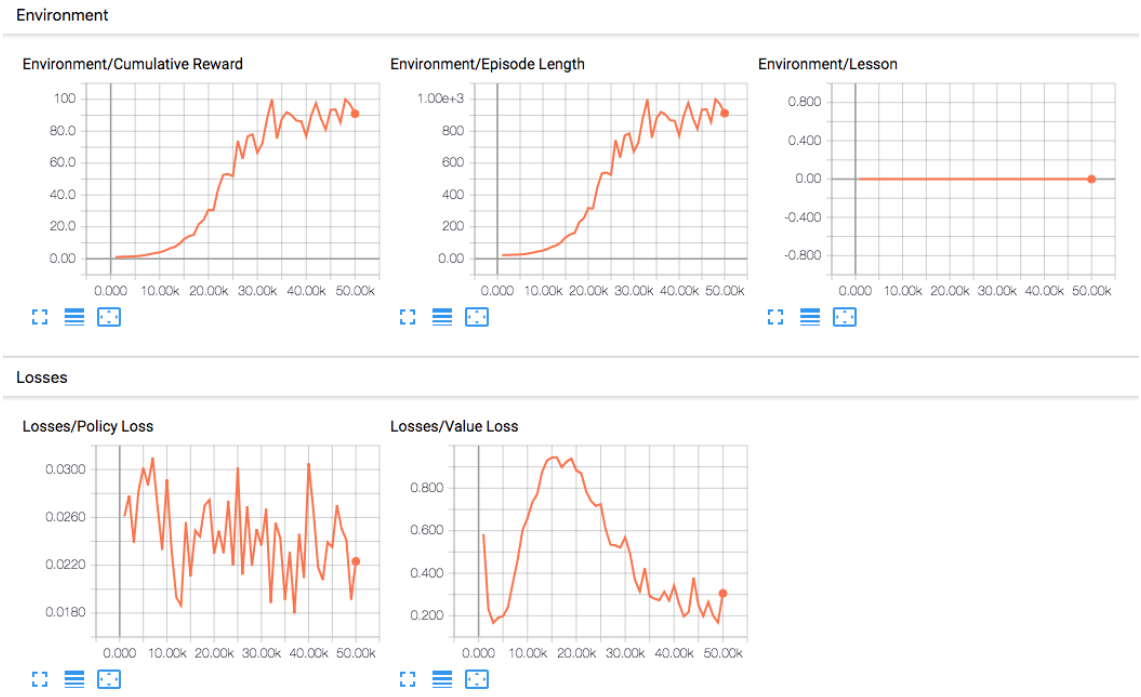


Figure 37: Example of the Tensorboard statistics

The final tool that we will be using will be the TensorFlow [38] utility called, Tensorboard [39]. The command mlagents-learn that we will be using will save training statistics in a folder called results using the run-id name we give to the training. By executing the command "tensorboard --logdir results --port 6006" in our virtual environment, we will be able to open all training statistics inside the virtual environment by opening a browser window and connecting to localhost:6006. An example can be seen in Figure 37 in the previous page.

6.6.2. Development of AIs

For both AIs I created using ML-Agents I used one of the starting environments that the ml-agents project provides called FoodCollector. FoodCollector is a Neural Network specialized in search for "good food" and avoid "bad food" among other things, very similar to what we want to achieve, so I took the scene and imported it to my project.

The development will follow a similar order than the one used in the other AIs. That is teaching the rabbit to eat and drink, same for the fox using static targets, then the rabbit to scape, the fox to chase, the rabbit to reproduce and finally the fox to reproduce too. So, the first step was to make our rabbits to eat carrots and drink water.

The resulting script based on the FoodCollector model is called RabbitMLAgent. Rhis script would have the same hunger and thirst stats that we used before in our behaviour trees AIs and also lose points from each of them after each second, causing the death of the agent if reaching zero. The next thing I implemented was the type of movement the agent would have, giving him only 2 options, to either rotate in its y axis or to go forward. I didn't want the agent to rotate in any other axis and I didn't want to make it go backwards either.

Following the FoodCollector model, I thought that the next step was to make the agent do two things: Find water and carrots and avoid all obstacles. With that premise in mind I set the Ray Sensors to detect absolutely everything.

To make the training more effective (or at least that is what I thought) I set the reward system to something like this:

EVENT	REWARD
If collides with an obstacle	-0.5f
If collides with a carrot, the agent is more hungry than thirsty and the hunger value is between 50f and 80f	+0.5f
If collides with a carrot, the agent is more hungry than thirsty and the hunger value is between 30f and 50f	+0.25f
If collides with a carrot, the agent is more hungry than thirsty and the hunger value is between 0f and 30f	+0.1f
If collides with a carrot but the agent is more thirsty than hungry	-0.5f
If collides with water, the agent is more thirsty than hungry and the thirst value is between 50f and 80f	+0.5f
If collides with water, the agent is more thirsty than hungry and the thirst value is between 30f and 50f	+0.25f
If collides with water, the agent is more thirsty than hungry and the thirst value is between 0f and 30f	+0.1f
If collides with water but the agent is more hungry than thirsty	-0.5f
If hunger or thirst reaches zero or below	-1.0f

Table 3: Reward system for the first version of the RabbitMLAgent

Unfortunately my agents did not achieve the desired objectives after finishing their trainings, they ended up with erratic behaviours like only drinking, only eating, wandering aimlessly o even staying in one place rotating constantly until dying. My attempt to solve this was to change the values of the rewards to higher values, but that didn't change things at all.

After investigating a little bit I found out two things that may have been causing the problems. One it was obviously my reward system, specifically what actions and situations gave or removed reward points, and the other was the values of the rewards.

The ML-Agent developers recommend to limit our reward systems to values between -1f and 1f to avoid problems, higher values may cause the programming to fail and cause unpredictable secondary effects in our training [40]. So the first thing I did from that point forward was to limit my reward values, establishing -1f and 1 as the lower and higher values possible.

But how could my reward system to cause the agent to, for example, spin out of control until dying? One of the most dangerous things Machine Learning has is that it's a little bit unpredictable because we might not be taking in to account all the variables that he is actually using.

As we can see in Table 3, I gave our agent negative rewards in four cases: When dying, when he eat when he should be drinking, when he drink when he should be eating and when he bumped into an obstacle. During training, the agent will start doing random actions from the available list we gave him, and the first thing that will happen as the product of it is that it will start colliding with the environment. The theory was "If we punish him heavily when colliding with an obstacle he will learn to avoid them really quickly" and yes, that's what he learnt to do, but inside its brain he decided that the risk to move and get positive rewards from eating and drinking was not worth it, it was clearly better, according to my reward system, to avoid everything and just die by avoiding any collision with obstacles. Besides I forgot to include the hunger and thirst variable into the observation vector of the agent, so that didn't help either.

I needed to make drastic changes in my code. The first thing I did was obviously to add the hunger and thirst variables into the observation vector, so it could see the relationship between reward and stats. The second thing I did was to remove all the reward system, leaving only the dying one, giving it a value of -1f.

The next step was to determine when the actor should get positive rewards. After my little investigation I learnt that a different point of view was required [41]. When creating an AI with reinforcement learning we must ask the question "When is the AI doing it great?", and the answer in this case is "When it is both without thirst nor hunger". The only time we need to give the agent positive rewards is as long as both stats are in their best shape, which I decided was when

each value was above the value of 50f. If both values are above 50f we will give the agent a very small positive reward at each step of the program.

Also, in the previous version, when colliding with water or food I forced him to drink/eat it, and I think this was also a mistake so I changed it. In the new version the agent will only drink or eat when the corresponding variable is below 50 and it is what he needs to do in the moment, no matter the collisions.

EVENT	REWARD
If hunger or thirst goes reach zero or below	-1f
As long the hunger and thirst variables are between 50f and 100f	+0.1f

Table 4: Reward System of the first functional rabbit agent

The last change I made was to put a total of ten agents training together inside the environment. The big size of the environment allowed them to be alone enough to train by themselves, but also will force them to interact between each other when the different versions of the AIs are implemented, versions in which they need to chase/scape each other or find a reproduction companion. This also has the benefit of speeding up the whole training process because the individual training of each agent is taken into account for the development of the neural network.

With this new system I finally got my rabbit agent to actually drink and eat and survive as long as it found sources of food around it, as we can see reflected in Figure 38.

Figure 38: Cumulative Reward of the Rabbit Agent with the eating and drinking behaviours

After using the same system with the fox AI with some modifications I advanced a little, reaching the point of having to implement the reproduction system.

In the first version of the reproduction system I implemented a float variable called **timeToReproduce** which would work as a reproduction cool down timer and as a signal for the reward system. The logic would be that if the animals hunger and thirst values are optimal (above 70f) it means the animal is healthy enough to reproduce. When that happens and the **timeToReproduce** value is below zero we allow the AI to reproduce. Once it is done it we will turn the value of the **timeToReproduce** variable to 30f, to leave it "resting" for a while.

If we keep with the mentality of "When is the AI doing it great?" we arrive to the conclusion that the animal now it will be doing great as long as the value of **timeToReproduce** is above 0, because it means I has reproduced and that's exactly what we want the AI to learn, so we need to add it to our reward system. I also changed the reward of the old "eat and drink" condition to match the new one in value, because I theorized that if we give the AI a more positive reward for eating and drinking it will come to the conclusion that reproduction is not needed because it won't have any negative effects if it doesn't do it.

The resulting reward system for the AIs having implemented its reproduction system would be the following.

EVENT	REWARD
If hunger or thirst goes reach zero or below	-1f
As long the hunger and thirst variables are between 50f and 100f	+0.01f
As long as the **timeToReproduce** variable is above zero in value	+0.01f

Table 5: Reward System of the first functional version of the reproduction system

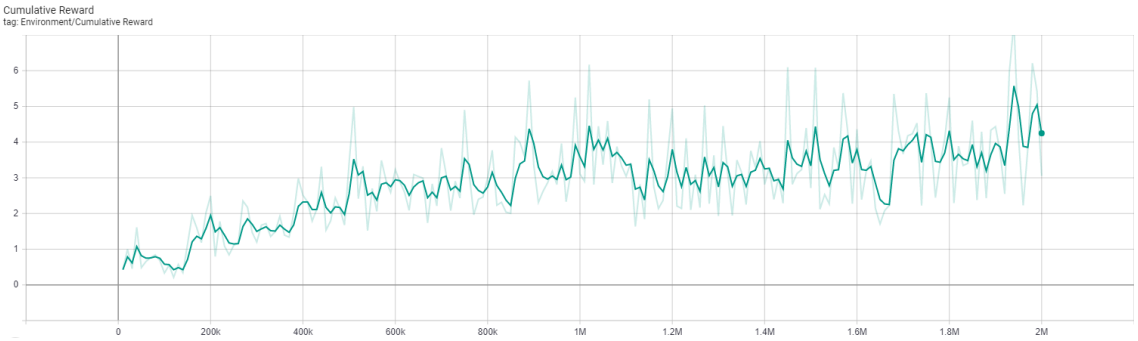


Figure 39: Cumulative Reward of the Fox Agent after implementing reproduction

At this point something crossed my mind, even thought I was happy because my foxes already did everything I hoped for I thought something. If we need optimal values for reproduction, why not use that as the only positive reward for the agent? In my model, for reproduction, the agent needs two things as I mentioned, for the cool down variable to be below zero and the values of both the hunger and thirst variables to be above 70, their "optima/healthy" values.

So, what if just leave the reproduction reward? My theory was that the animal would start trying things, would start eating and drinking, the negative reward would teach him to avoid dying, and at some point of the training it would just happen that the animal accidentally reproduces. By giving it a high reward it might ended up understanding that reproduction is the best thing that it can do, and because of its conditions it will force him to feed and drink until reaching optimal levels instead in only doing it to stay alive.

I decided to give it a go, implementing the "new" reward system seeing in Table 6.

EVENT	REWARD
If hunger or thirst goes reach zero or below	-1f
As long as the **timeToReproduce** variable is above zero in value	+0.5f

Table 6: Reward System of the new experimental neural network.

As I expected, it took a little bit for our AI to figure out the wonders of reproduction, but once it understood what was going on it started to do it as often as it could, figuring out the process more and more. I ended up doubling the duration of the trainings to make sure the AIs truly understood what they needed to learn. If we take a look at Figure 40 we can see how only at the very beginning we get a negative cumulative reward (meaning the animal died) but soon we start seeing spikes of positive rewards, meaning the animal is starting to figure out how to reproduce, increasing in frequency until mastering it.

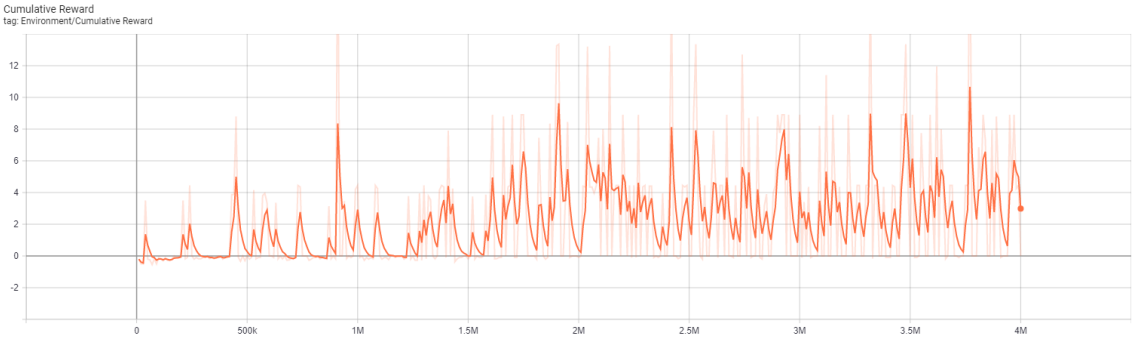


Figure 40: Cumulative reward of our new experimental neural network

With this new experiment being successful I adapted and used it on the rabbit AI as well, having as result both AIs to do all their functions successfully.

The evolution and iteration of the different Neural Networks developed can be found in Annex III of this study.

7. Results

7.1. Functionality

When we first compare and think about both methods used in this study we might think that there is no difference in functionality because all of them ended up fulfilling their objectives, this is no exactly accurate.

The AIs made using behaviour trees have several pros on their side. The graphs are a really clear and useful way to design the AI behaviour and decision making. The logical flow also make behaviour trees predictable and easy to follow when the AI are executing them, especially if we count with tools like the Debugging Window from the NPBehave library. They are highly modular, you can remove or add new branches and even whole behaviour trees within other behaviour trees, it only depends on how complex you want the AI to be. The addition of new elements and actions in new branches of the trees won't affect the main tree if doing properly. And finally, they are widely used in video games AI development, so you can count with multiple libraries and documentation everywhere to help you.

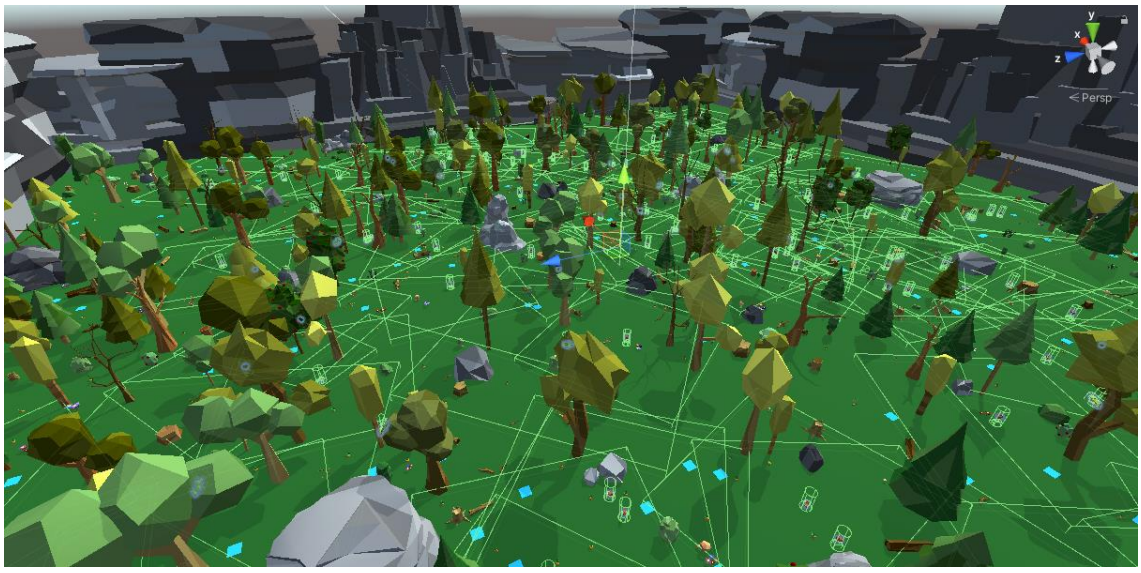


Figure 41: Scene view with AIs using Behaviour Trees (All AIs selected)

But no everything about behaviour trees is positive, among the main problems of using behaviour trees is precisely how complex can they become. There might be a point in the development

process where the tree is just too big, with too many dependencies, and even if it might seem clear when looking at the graph you might find a hard time with the implementation. In other words, it may take too much time and work to design properly and to implement a complex behaviour tree.

In the same way, machine learning also have its own pros and cons. I won't be talking about other machine learning techniques or libraries but only about ML-Agents, the open source project made by Unity. The obvious advantage is how easy it is to start using it, you only need some basic programming notions and the ML-Agents project. Its potential is limitless, this specific tool allows developers to create a huge variety of AIs and other different neural networks, leaving it all to their own imagination and time. And in a personal note, I found it very funny to use and see the AIs evolving.

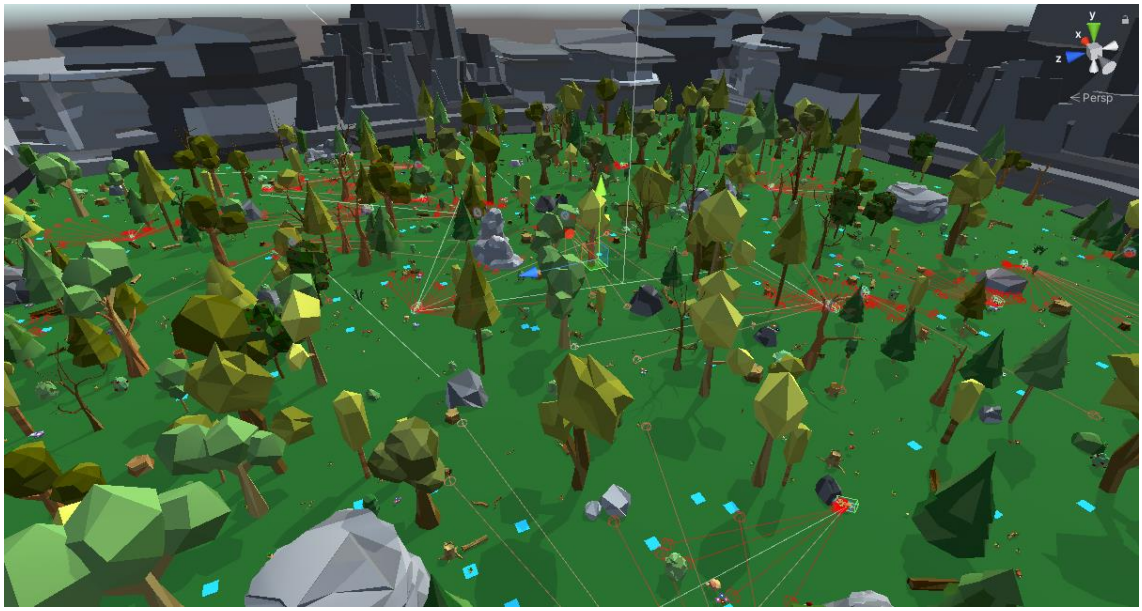


Figure 42: Scene view with AIs using Neural Networks (Foxes AIs selected)

Unfortunately, I think it has more disadvantages. Machine learning is still very unpredictable, it does not give the developer a real process to follow in order to get the results he wants, only a few guides and notes. Even if you learn to create a standard neural network in one day you may need a lot more time to find our which combination of rewards, observations and actions you need to achieve your objectives. The AIs actual behaviour is far more difficult to control and to fix, because it may need re-training. In our case the AIs were simple enough to avoid major problems once we figured out the formula but for more complex behaviours it might be far more problematic.

7.2. Time of production

The time it took for both methods (behaviour trees and machine learning) can't even be compared.

Learning how to use behaviour trees, even with an already built library such as NPBehave, took me around one month to fully understand and dominate. To that we must add the actual development time, which took around 150 hours of work. In that time I designed the different AIs, their graphs, the process, the different versions and finally the development. As I mentioned before, this is one of the disadvantages of using behaviour trees, it takes too much time, not to mention that even if I use one AI as a base to create another one I must do a lot of changes in order to fulfil the requirements of the new AI, like targets, implement chasing/scape etc.

Machine learning was easier in comparison, I learnt to use it with the help of the online course that I took in less than 20 hours, and the development of the final version of the AIs took me about 72 hours. Thanks to their design, both AIs have virtually the same behaviour, so the training for both of them can be the same only changing a couple parameters in key places, so the development is far quicker than when using behaviour trees.

Although quicker, using machine learning, specifically Unity's ML-Agents, might present installation problems due to conflicts between Python versions, virtual environment etc. I did not count this problem in working hours because it was more like a personal problem with my computer, but we must take it into consideration.

7.3. Performance

To compare the performance I tampered the simulation a little. Instead of an unpredictable ecosystem with the number of prefabs constantly changing due to dead or reproduction I changed both ecosystems, the one using behaviour trees and the one using machine learning, applying the following changes:

- We will instantiate a total of 100 rabbits and 20 foxes in both scenes.
- All AIs will live a total of 120 seconds, no less, so we have time to check their performance.

- No rabbit will be able to be killed by any fox, the fox will "eat" but won't kill the rabbit in the process so the amount of AIs don't change.

- For the same reason, reproduction has been disabled, they will "reproduce" but no new AIs will be instantiated.

With this setting in place we tested both scenes and check their performance using Unity's analytic tool, the Profiler [42].

The first batch of results are from the scene where the AIs are using Behaviour Trees. As we can see in Figure 43 we have a really good performance considering the amount of AIs in the scene, with some spikes of work that might reach about 3.5ms of processing time at worst.

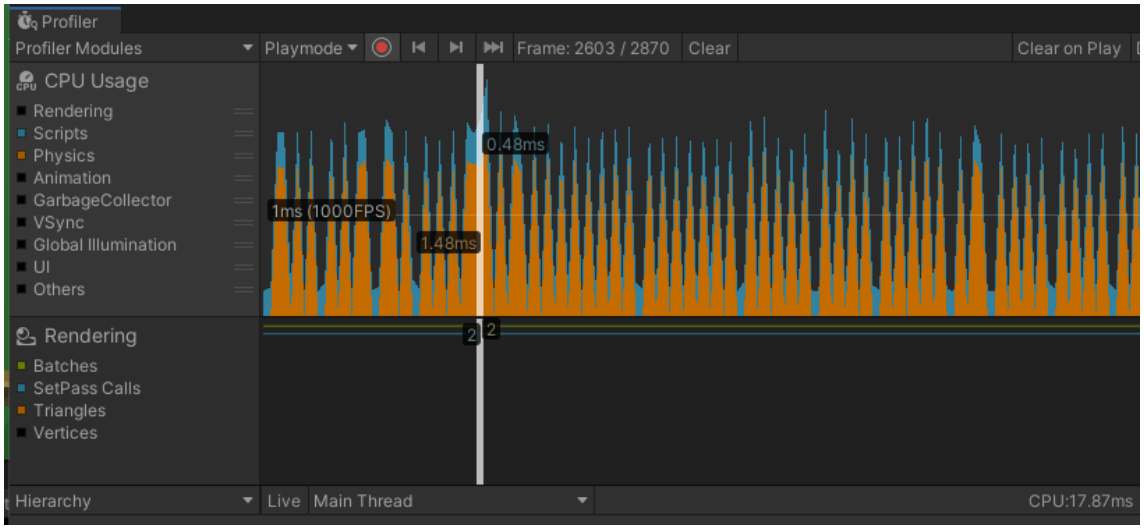


Figure 43: Overall performance using Behaviour Trees

If we look at Figure 44 we can see a breakdown of the performance, and we can see that the thing that consumes more memory are physics, rounding the 5.6KB in that specific time. On the other hand, the actual AI of the different animals barely costs anything, only 224B. This values are not constant obviously but the pattern is as we can see in Figure 43, so this percentages are always very similar.

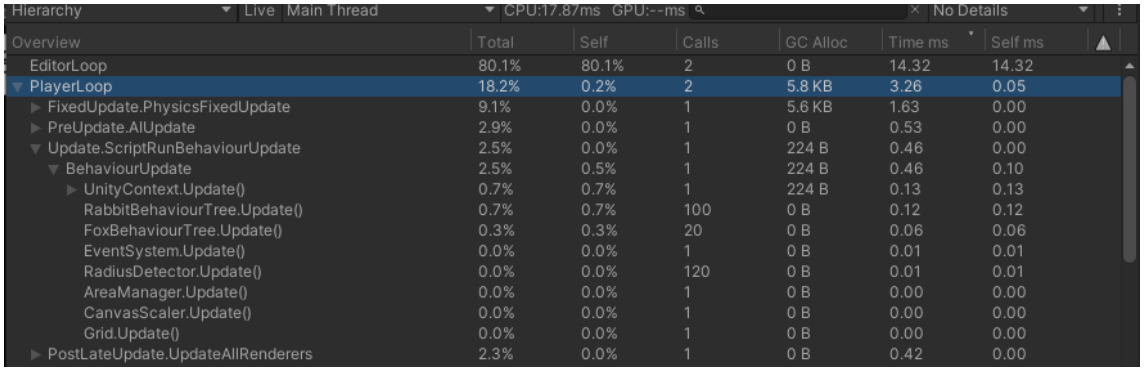


Figure 44: Performance break-down using Behaviour Trees

Moving on to the next scenario we have our AIs using machine learning, and we are getting very different values. Taking a look at Figure 45 we can see that this system have spikes with major costs than the ones appearing when using behaviour trees. These spikes are less frequent but they can have a bigger impact if we increase the number of AIs, causing FPS drops at key moments even.

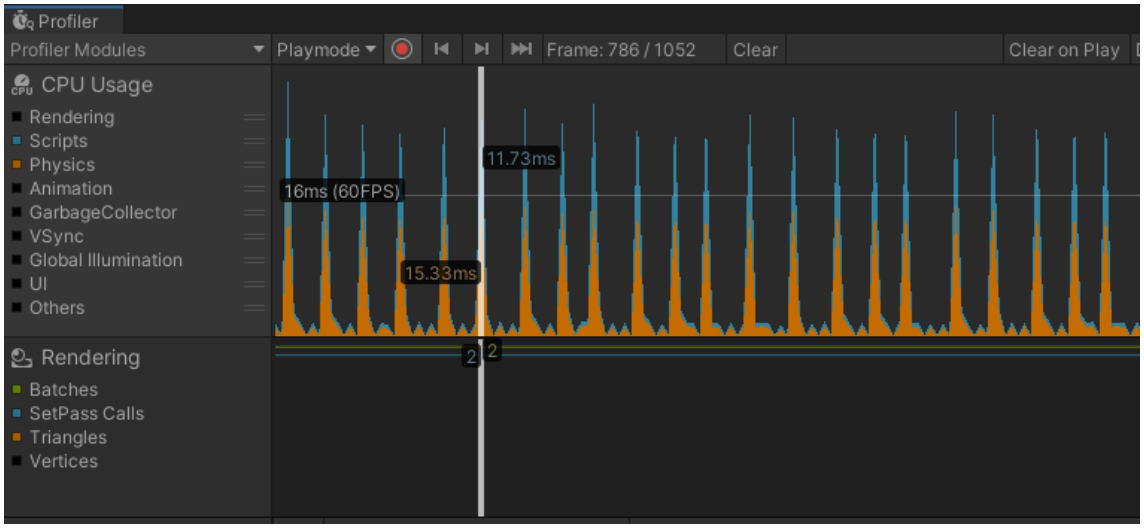


Figure 45: Overall performance using Machine Learning

If we look at Figure 46 we will understand why. By looking at one of those spikes we can see that almost all the processing capacity goes for the machine learning academy, which is in charge of selecting and give to each agent which action to take, reaching around 70KB alone. For this

simulation we might not perceive the spikes in runtime yet but in a non-controlled version, with more AIs instancing (or directly a scene with more AIs working) it might be problematic.

Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
PlayerLoop	65.7%	0.1%	2	67.0 KB	28.10	0.05
FixedUpdate.ScriptRunBehaviourFixedUpdate	60.3%	0.0%	1	67.0 KB	25.81	0.00
FixedBehaviourUpdate	60.3%	0.0%	1	67.0 KB	25.81	0.00
AcademyFixedUpdateStepper.FixedUpdate()	60.3%	0.0%	1	67.0 KB	25.81	0.03
root.DecideAction	53.1%	0.1%	1	39.1 KB	22.75	0.05
root.AgentSendState	6.6%	0.7%	1	27.4 KB	2.82	0.30
root.AgentAct	0.4%	0.4%	1	0 B	0.19	0.19
GC.Alloc	0.0%	0.0%	8	408 B	0.00	0.00
FixedUpdate.PhysicsFixedUpdate	2.9%	0.0%	1	0 B	1.24	0.00
PostLateUpdate.UpdateAllRenderers	1.0%	0.0%	1	0 B	0.45	0.00
updateScene.Invoke	0.4%	0.0%	1	0 B	0.19	0.00
Update.ScriptRunBehaviourUpdate	0.4%	0.0%	1	0 B	0.18	0.00
BehaviourUpdate	0.4%	0.0%	1	0 B	0.18	0.03
RabbitMLAgent.Update()	0.2%	0.2%	100	0 B	0.09	0.09
FoxMLAgent.Update()	0.0%	0.0%	20	0 B	0.03	0.03
EventSystem.Update()	0.0%	0.0%	1	0 B	0.01	0.01
AreaManager.Update()	0.0%	0.0%	1	0 B	0.00	0.00
CanvasScaler.Update()	0.0%	0.0%	1	0 B	0.00	0.00
FoodCollectorSettings.Update()	0.0%	0.0%	1	0 B	0.00	0.00
Grid.Update()	0.0%	0.0%	1	0 B	0.00	0.00
UGUI.Rendering.RenderOverlays	0.0%	0.0%	1	0 B	0.01	0.00
PostLateUpdate.PlayerUpdateCanvases	0.0%	0.0%	1	0 B	0.00	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.0%	0.0%	1	0 B	0.00	0.00

Figure 46: Performance break-down using Machine Learning

7.4. Overall view

All four IAs seem to do their work pretty fine, having only some small issues besides possible performance problems.

Behaviour trees AIs, at an individual level, have the main problem of taking too long to program, just too many variables and things to take in to account. One of my implementations to improve performance was to check if a new plan/action was needed every 5 seconds, those are 5 seconds that the animal might be wasting doing something it should not, but it might cost too much to check at every step of the program so I thought this was the best way to go.

Machine learning AIs, also at an individual level, seem to behave like true animals due to the random nature of machine learning training, but that's a two sided sword. Even if the AIs are able to perform all the tasks I wanted them to learn they achieved them in some peculiar ways, preferring to rotate in one direction over another for example, making the AIs to look "dumb" sometimes. These AIs are also very erratic and fast, they do not lose any time and are always moving as fast as they can, occasionally drifting on the map and colliding with the environment.

When trying to simulate the ecosystems with several AIs it does not seem to have any impact in the AIs or how they behave actually, they will follow the program to the letter. Although there is a problem, I found out that even if the AIs work properly it is really difficult to calibrate their stats to make a stable ecosystem in both scenes, it will require more time and even more study to achieve that, without causing overpopulation or mass extinction.

8. Conclusions

After working on this project I have experience by first-hand how complex video games AI programming actually is. In the past I had to program different AIs but I used worst techniques if any, resulting in real messes. Anyhow, I have learn several things thanks to this experience.

Traditional AI development techniques exist for a reason, they are result of a lot of years of industry and experimentation and they have survive because they are actually useful. Even if we are able to use our own methods it is always important to consider using techniques like Planning or Behaviour Trees.

Behaviour Trees have proven to be a little exhausting to program, but not actually hard. Even though you need some programming knowledge to take advantage of their full potential I think this technique is clear enough for new developers to implement in their projects if they follow the correct process at all time in a clearly fashion.

I have seen that not only they are useful, but also really cheap in terms of processing cost compared with other systems. Also, implementing new actions from one AI version to the next did not prove to be difficult at all if, as I said, you do it clearly and without rushing. Same thing for debugging and fixing, behaviour trees can be so well organized that when any type of problem occurred I immediately knew where the cause might be located.

Machine Learning on the other hand might not be as good of an option for video game AI development. The training process can be very difficult to grasp at first, you need to find out the precise pattern and reward system that will get your AI to actually learn what you want, and that might take a lot of time if even possible. This makes this technique very unpredictable and unstable, causing a lot of rare problems during training. Making small changes are almost impossible, usually having to re-train from zero the whole neural network every time you need to change or add anything, that makes debugging and the control of small and specific actions really hard to achieve. On top of that it seems that the decision making progress of the resulting neural networks are much more expensive that traditional methods, but it also depends on the AI behaviour and to which method you compare it with.

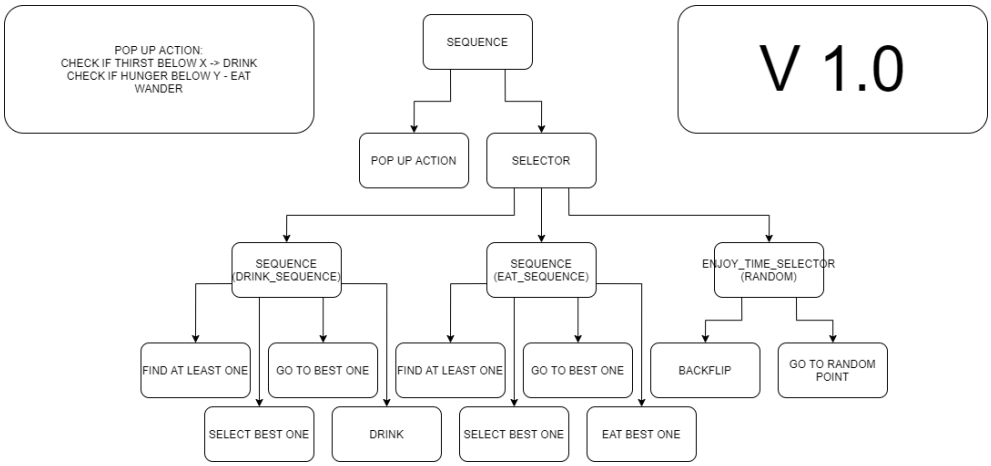
On the other hand, I had a really good time using machine learning. Even with the frustration of the installation process and the unpredictable behaviours I really liked the whole process of figuring out the correct settings for my AIs to work, as well as to see them improve when they

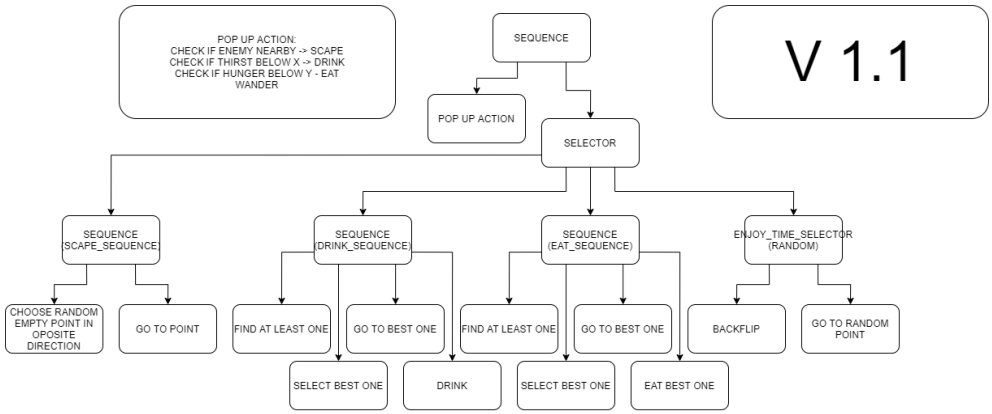
were training. The amount of code I had to implement is but a fraction of the one I had to use using behaviour trees, that plus the help from the starting environments from ML-Agents made the process a lot easier. The final result is also satisfactory, after all the AIs have learnt to do what I wanted and the randomness in their behaviour make them look more "alive" than using the other method.

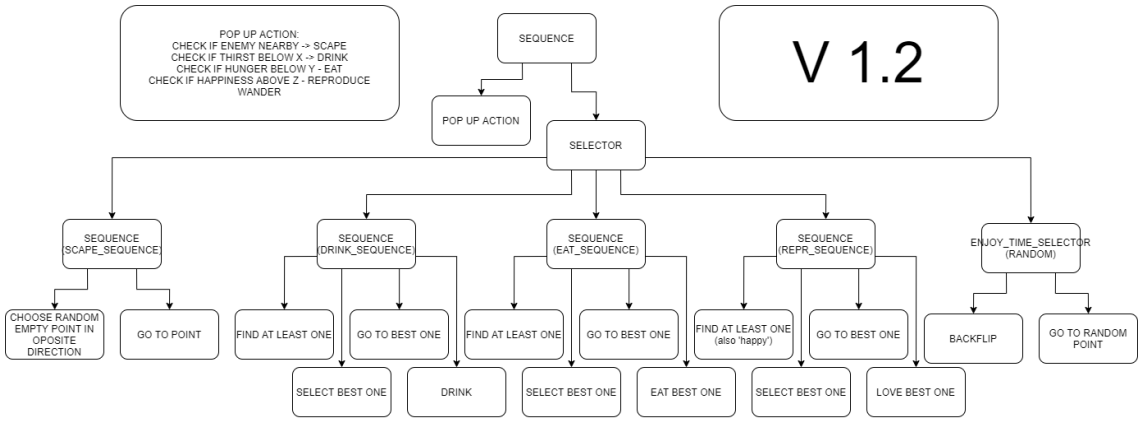
I think machine learning will be a useful tool for a lot of developers out there who want to create AIs from scratch and with random behaviours, but its young and unpredictable nature makes it, for now, to work better as a research and experimenting tool to train other types of AIs instead of video games AIs. Developers usually need AIs with very specific behaviours independently on how complex they might be and in a limited amount of time because they are developing a product with a release date, and machine learning has been proven to be more powerful when you let it train a lot, not only a couple of hours.

As today I think the best option is to keep using more traditional techniques instead of machine learning, but who knows, it all depends on the time and the design of the AI what will decide that.

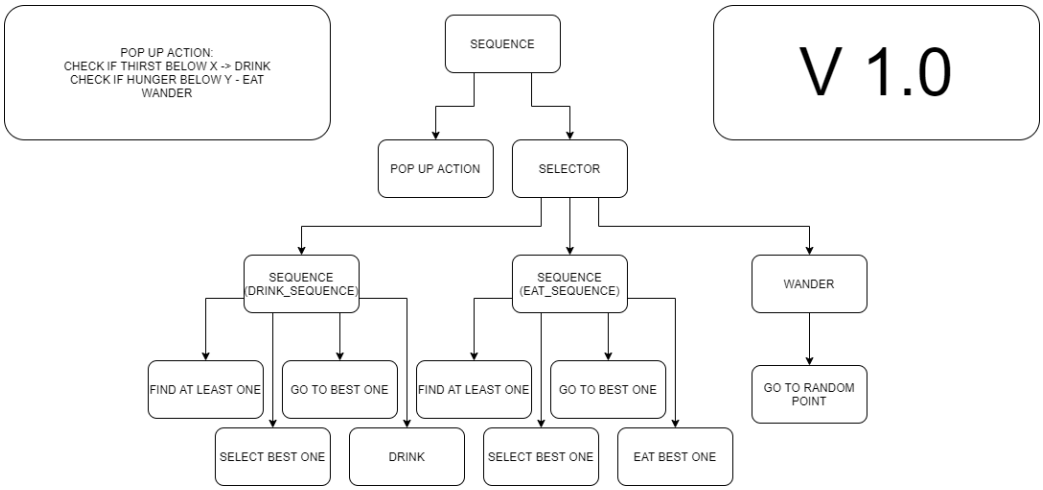
Annex I – Rabbit Behaviour Trees

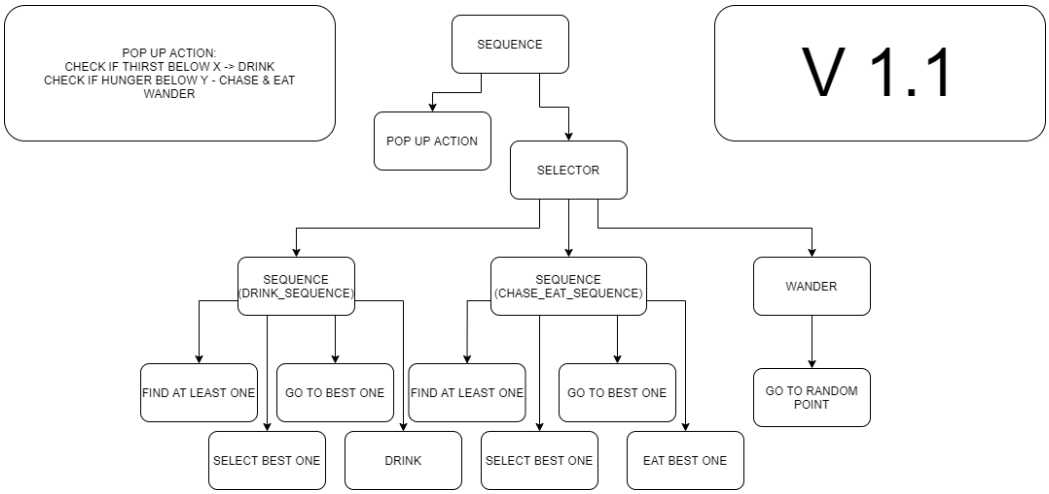


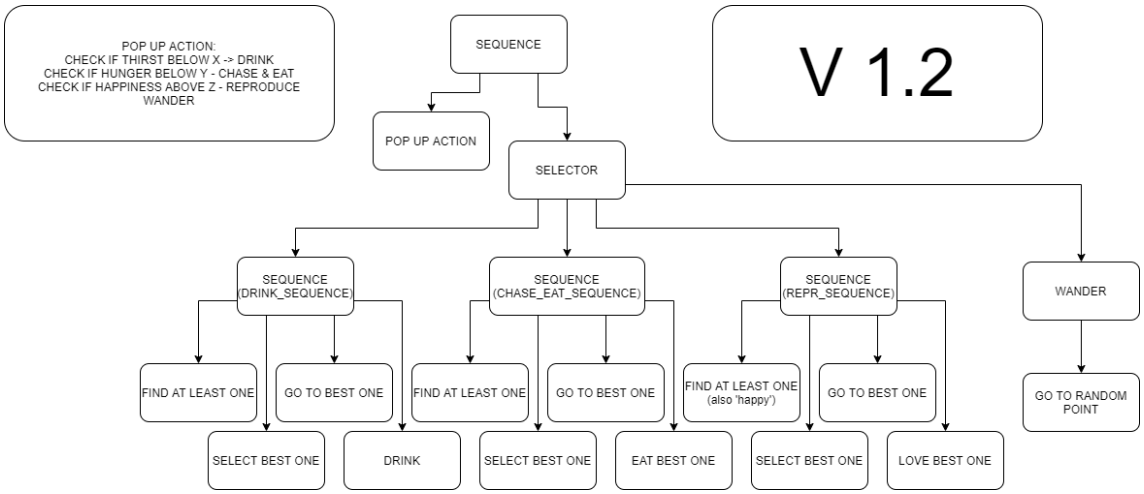




Annex II – Fox Behaviour Trees







Annex III – ML-Agents training evolution notes

Behaviour Name: RabbitNN_1.0
Class Name: RabbitMLAgent.cs

Objectives:

- Stay alive by drinking and eating.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - carrot

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject Carrot	If(hunger <= 50) hunger = hunger + 40 Carrot.OnEaten()
OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f

Training Notes:

o	Step	Description
1	560000	Mean Reward: 0.405 – Agents seem to achieve both targets so far, barely going down 50 in both hunger and thirst.
2	750000	Mean Reward: 0.405 – Behaviour keeps the same, rewards growing.
3	1120000	Mean Reward: 0.413 – Rewards are going up and down between 3.X and 4.X, but the agents seem to be doing fine, maybe falling from map? Give solution to that in next training. Seems promising, possibility of keep improving, let the training finish and evaluate if re-training.

Final Result: Success

Agents move constantly through the whole environment, and as soon they have the need to consume either water or food, they do it.

Changes:

None, clear for next stage.

Behaviour Name: RabbitNN_2.0
Class Name: RabbitMLAgent.cs

Objectives:

- Stay alive by drinking and eating.
- Reproduce.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.
- If Collides with another agent within reproduction conditions, do it. Give Reward to make it do it.
- Use a counter that once goes below 0 allows reproduction, reset to X value when reproduction takes place.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
  - TimeToReproduce (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - carrot

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject Carrot	If(hunger <= 50) hunger = hunger + 40 Carrot.OnEaten()

OnCollisionEnter	GameObject RabbitAgent	If(hunger > 70 && thirst > 70 && timeToReproduce <= 0)
		timeToReproduce = 30f;
		Reproduce();

OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f
If(timeToReproduce > 0)	+0.01f

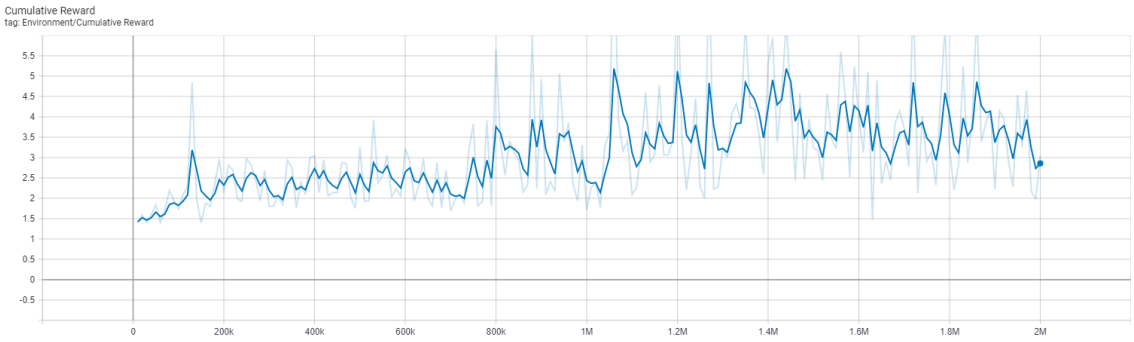
Training Notes:

No notes, had a meeting and could not pay enough attention to the training, the result will be only evaluated when the Neural Network is implemented and we see the results.

o	Step	Description
1		
2		
3		

Final Result: Success

The agents have learn to eat and drink again and they try to reproduce whenever they have the chance.



Changes:

None, clear for next stage.

Behaviour Name: FoxNN_1.0
Class Name: FoxMLAgent.cs

Objectives:

- Stay alive by drinking and eating non-moving rabbits.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - foxAgent

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject rabbitAgent	If(hunger <= 50) hunger = hunger + 40 rabbitAgent.OnEaten()
OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

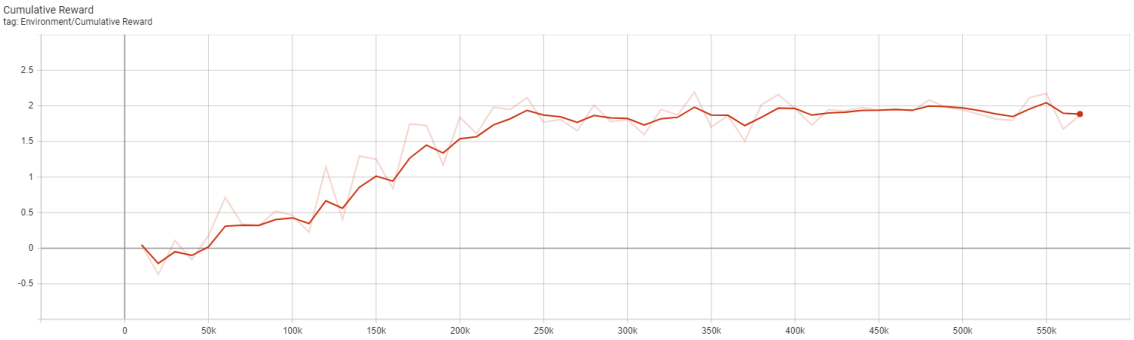
Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f

Training Notes:

o	Step	Description
1	300000	Mean Reward: 2.016 – Agents seem to achieve both targets so far, barely going down 50 in both hunger and thirst, not reaching 40. They explore the map. Maybe the increase in rewards compared to rabbits is due to the difference of speed (Rabbits 2f, Foxes 3f).
2		
3		

Final Result: FAIL

The agents have learn to eat and drink again but the training has been interrupted.



Changes:

None, re-train required due to problems.

Behaviour Name: FoxNN_1.1
Class Name: FoxMLAgent.cs

Objectives:

- Stay alive by drinking and eating moving rabbits.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - foxAgent

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject rabbitAgent	If(hunger <= 50) hunger = hunger + 40 rabbitAgent.OnEaten()
OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f

Training Notes:

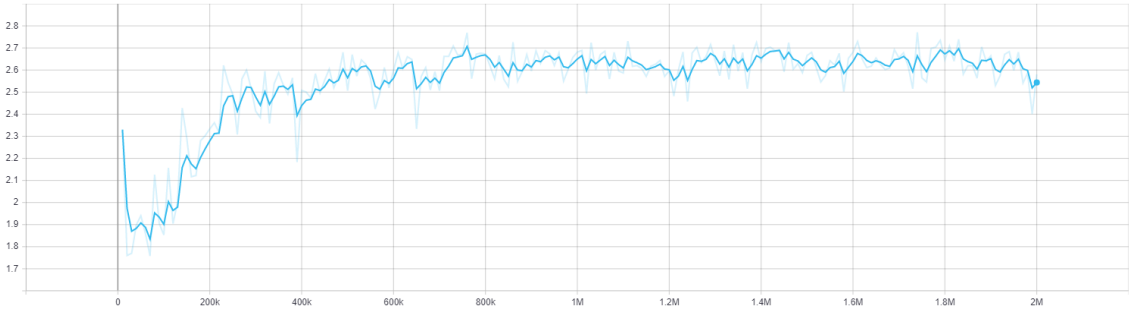
No notes.

o	Step	Description
1		
2		
3		

Final Result: ~~Success~~ **FAIL**

~~The agents have learn to eat and drink, full trained this time.~~

The agent model had several colliders inside, so the raycasting was colliding with them and therefore it was barely getting any information from them.



Changes:

Modify the agent until raycasts are operational, then re-train.

Behaviour Name: FoxNN_1.2
Class Name: FoxMLAgent.cs

Objectives:

- Stay alive by drinking and eating moving rabbits.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - foxAgent

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject rabbitAgent	If(hunger <= 50) hunger = hunger + 40 rabbitAgent.OnEaten()
OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

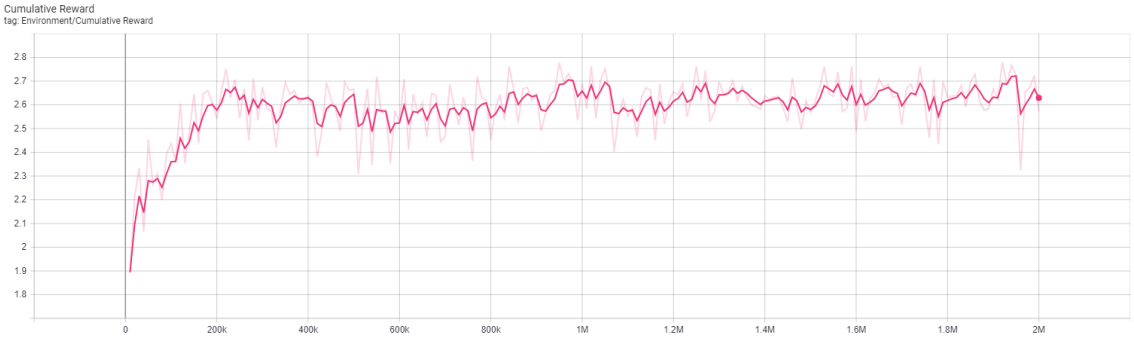
Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f

Training Notes:

o	Step	Description
1		
2		
3		

Final Result: SUCCESS

Agents move constantly through the whole environment, and as soon they have the need to consume either water or food, they do it.



Changes:

None, clear for next stage.

Behaviour Name: FoxNN_2.0
Class Name: FoxMLAgent.cs

Objectives:

- Stay alive by drinking and eating.
- Reproduce.

How To:

- Avoid dying (Negative Reward)
- Keep stats above minimum levels (Above 50, Positive Reward)
- Make the agent search, not avoid.
- If Collides with another agent within reproduction conditions, do it. Give Reward to make it do it.
- Use a counter that once goes below 0 allows reproduction, reset to X value when reproduction takes place.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
  - TimeToReproduce (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - foxAgent

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject RabbitAgent	If(hunger <= 50) hunger = hunger + 40 RabbitAgent.OnEaten()

OnCollisionEnter	GameObject FoxAgent	If(hunger > 70 && thirst > 70 && timeToReproduce <= 0)
		timeToReproduce = 30f;
		Reproduce();

OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(hunger > 50 && hunger <= 100 && hunger > 50 && hunger <= 100)	+0.01f
If(timeToReproduce > 0)	+0.01f

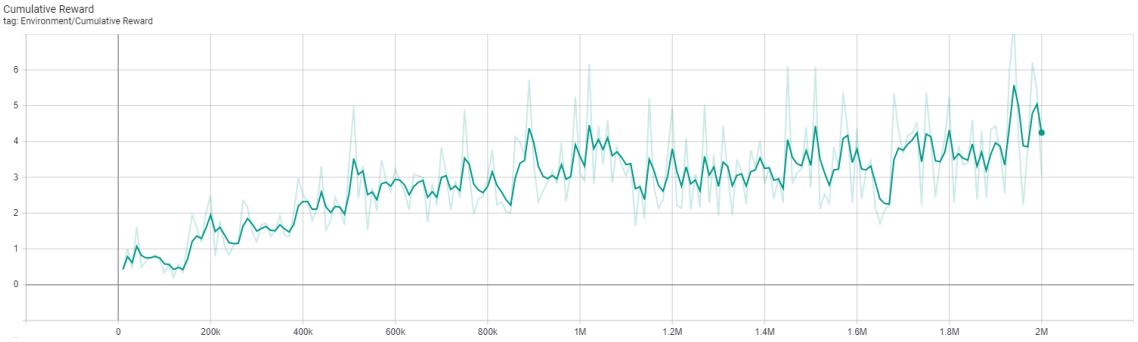
Training Notes:

No notes, had a meeting and could not pay enough attention to the training, the result will be only evaluated when the Neural Network is implemented and we see the results.

o	Step	Description
1		
2		
3		

Final Result: Success

The agents have learn to eat and drink again and they try to reproduce whenever they have the chance.



Changes:

None, clear for next stage.

Behaviour Name: FoxNN_Esperimental **(Spelling error)**
Class Name: FoxMLAgent.cs

Objectives:

- Reproduce. Reproduction need to have both hunger and thirst in optimal values (above 70), so in order to reproduce, the agent will need to learn to eat and drink before, learning everything we need in the process.

How To:

- Avoid dying (Negative Reward)
- If Collides with another agent within reproduction conditions, do it.
- Use a counter that once goes below 0 allows reproduction, reset to X value when reproduction takes place.
- Positive reward as long as that counter is positive.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
  - TimeToReproduce (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - foxAgent

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject RabbitAgent	If(hunger <= 50) hunger = hunger + 40 RabbitAgent.OnEaten()

OnCollisionEnter	GameObject FoxAgent	If(hunger > 70 && thirst > 70 && timeToReproduce <= 0)
		timeToReproduce = 30f;
		Reproduce();

OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

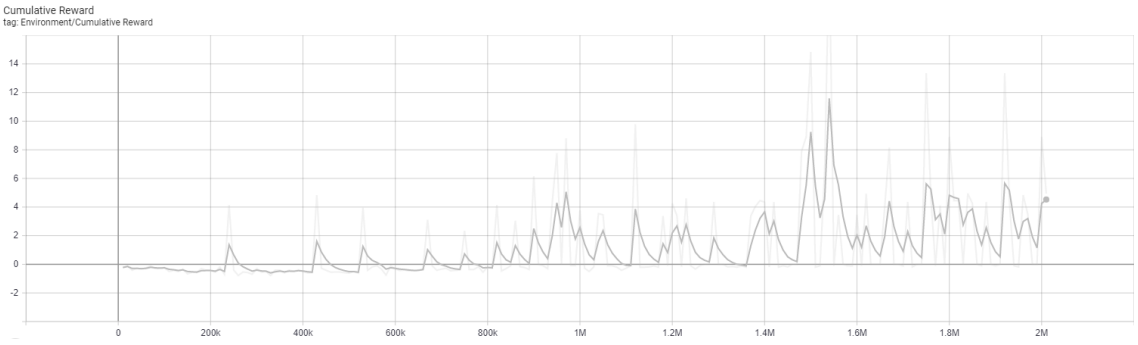
Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(timeToReproduce > 0)	+0.5f

Training Notes:

o	Step	Description
1		
2		
3		

Final Result: Success

The agents have learn to eat and drink again and they try to reproduce whenever they have the chance with it as only focus.



Changes:

None, the Neural Network is a complete success, train a new NN for the Rabbit agents following the same logic.

Behaviour Name: RabbitNN_Experimental
Class Name: RabbitMLAgent.cs

Objectives:

- Reproduce. Reproduction need to have both hunger and thirst in optimal values (above 70), so in order to reproduce, the agent will need to learn to eat and drink before, learning everything we need in the process.

How To:

- Avoid dying (Negative Reward)
- If Collides with another agent within reproduction conditions, do it.
- Use a counter that once goes below 0 allows reproduction, reset to X value when reproduction takes place.
- Positive reward as long as that counter is positive.

Observations:

- VectorSensor
  - m_AgentRb.velocity.x
  - m_AgentRb.velocity.z
- Attributes
  - Hunger (float)
  - Thirst (float)
  - TimeToReproduce (float)
- RayPerception
  - rabbitAgent
  - obstacle
  - water
  - carrot

Actions:

- Vector3 Rotation
- Vector3 forward

Interactions:

Type	With what	Result
OnCollisionEnter	GameObject Carrot	If(hunger <= 50) hunger = hunger + 40 Carrot.OnEaten()

OnCollisionEnter	GameObject RabbitAgent	If(hunger > 70 && thirst > 70 && timeToReproduce <= 0)
		timeToReproduce = 30f;
		Reproduce();

OnTriggerEnter	GameObject Water	If(thirst <= 50) thirst = thirst + 40

Rewards:

Condition	Reward
If(hunger <= 0 || thirst <=0)	-1f
If(timeToReproduce > 0)	+0.01f

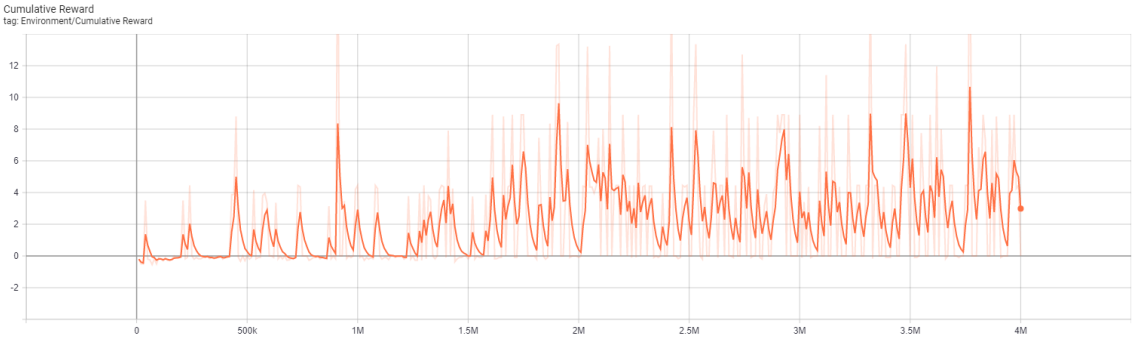
Training Notes:

No notes.

o	Step	Description
1		
2		
3		

Final Result: Success

Like the Fox Agent, the rabbits have successfully learn all traits by just encouraging reproduction as an end.



Changes:

None, the Neural Network is a complete success, keep using this training method if new agents need to be implemented.

Annex IV – Propuesta TFG

1. TÍTULO DEL PROYECTO

Creación de inteligencias artificiales basadas Machine Learning como alternativa a los métodos Convencionales

2. DESCRIPCIÓN Y JUSTIFICACIÓN DEL TEMA A TRATAR

Hoy en día, la Inteligencia Artificial es uno de los pilares principales a la hora de desarrollar un videojuego en la gran mayoría de los casos. La evolución de la industria y de la informática en general ha hecho que las nuevas Inteligencias Artificiales necesiten ser mucho más complejas, lo que que a veces conlleva muchísimas horas de trabajo en diseño, implementación, testing e iteración. Como alternativa, el aprendizaje máquina (Machine Learning) ha empezado a coger fuerza, y cada vez son más los desarrolladores que creen que este es el sistema que sustituirá al desarrollo convencional de Inteligencias Artificiales. En este proyecto, analizaremos la herramienta ML-Agents, una API desarrollada para Unity con la que se pueden crear Inteligencias Artificiales basadas en Machine Learning, para comprobar si realmente este es el futuro de la Inteligencia Artificial.

3. OBJETIVOS DEL PROYECTO

Los objetivos de este proyecto son:

1. Estudio Machine Learning y modelado de sistemas.
2. Implementación de sistemas de Machine Learning en Unity.
3. Creación e implementación de Inteligencias Artificiales basadas en Machine Learning con diferentes niveles de complejidad.
4. Estudio comparativo entre Inteligencias Artificiales entrenadas utilizando Machine Learning e Inteligencias Artificiales convencionales.

4. METODOLOGÍA

La metodología se establecerá en las primeras fases del proyecto.

5. PLANIFICACIÓN DE TAREAS

Las tareas quedan predefinidas de manera global en los objetivos. Serán fijadas de forma concreta durante el desarrollo del proyecto.

Annex V - Tutorías

1. **Fecha**: Lunes 23 de Septiembre de 2019. **Modo:** Presencial. **Lugar:** Universidad San Jorge, Zaragoza.

   Contenido: En esta primera toma de contacto le propuse a Antonio Iglesias que fuera mi tutor para el Trabajo de Fin de Grado de Informática. Tras aceptar, nos reunimos para tener una pequeña charla en la que le expuse las bases de mi idea, es decir, un estudio comparativo sobre IAs en un ecosistema aprovechando la clase que el imparte como base.

   Decimos volver a reunirnos en Octubre para darle el visto bueno a la propuesta del TFG y centrar los objetivos del estudio.

2. **Fecha**: Domingo 29 de Septiembre de 2019. **Modo:** Online. **Lugar:** Discord.

   Contenido: Le envío a Antonio mi propuesta inicial del Trabajo de Fin de Grado. Tras hablar un poco termino de redactar la propuesta gracias a sus consejos y la envío.

   Terminamos diciéndole yo que voy a presentarme directamente a la convocatoria de Septiembre para poder centrarme en el resto de asignaturas de la carrera. Acordamos continuar en Junio.

3. **Fecha**: Miércoles 8 de Julio de 2020. **Modo:** Online. **Lugar:** Discord.

   Contenido: En esta reunión hablamos de cómo voy a orientar el trabajo, que metodología voy a emplear y cómo voy a utilizar lo aprendido en su asignatura para el TFG.

   Nos intercambiamos disponibilidad para tutorías durante el verano, acordando por mi parte mandarle pequeños informes sobre el estado actual del desarrollo, para mantenerlo informado del avance y para avisarle sobre la necesidad de futuras tutorías.

4. **Fecha**: Jueves 27 de Agosto de 2020. **Modo:** Online. **Lugar:** Discord.

   Contenido: Solicito tutoría a Antonio para ponerle al día de cómo va el TFG y de algunos problemas que me he ido encontrando para preguntarle si mis soluciones están bien llevadas o no.

   Tras aclarar dudas y ver el estado acordamos seguir e n contacto para lo que necesite.

5. **Fecha**: Lunes 7 de Septiembre de 2020. **Modo:** Online. **Lugar:** Discord.

   Contenido: Le envío a Antonio la primera versión de mi TFG a falta de pasar a limpio bibliografía y completar un par de apartados para los cuales necesito tener la demo cien por cien funcional.

Bibliography

[1] STANFORD UNIVERSITY. John McCarthy [Online]. John McCarthy, Stanford University. October 2011. Available from: http://jmc.stanford.edu/

[2] MCCARTHY, John. *What is artificial intelligence?* [Online]. Computer Science Department, Stanford University, November 2007. 15 p. Available from: http://www-formal.stanford.edu/jmc/whatisai.pdf

[3] RUSSEL, Stuart; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. Third edition. Upper Saddle River, New Jersey 07458, US. Prentice Hall, 2009. 1151 p. ISBN-13: 978-0-13-604259-4, ISBN-10: 0-13-604259-7.

[4] VINCENT, James. *This is when AI's top researchers think artificial general intelligence will be achieved* [Online]. The Verge. November 2018. Available from: https://www.theverge.com/2018/11/27/18114362/ai-artificial-general-intelligence-when-achieved-martin-ford-book

[5] BROOKLY COLLEGE. *Artificial Intelligence: Agents, Architecture, and Techniques* [Online]. Chapter 5.3. Computer and Information Science, Brooklyn College, New York. Available from: http://www.sci.brooklyn.cuny.edu/~meyer/CISC3600/Materials/5_3AIAgents.pdf

[6] CATTELAN, Dylan. *Evolution of AI in Video-Games from 1980 to Today* [Online]. The Medium. December 2019. Available from: https://medium.com/@DylanCa_/evolution-of-ai-in-video-games-from-1980-to-today-e3344acaed4c

[7] IGLESIAS, Antonio. *Dijkstra Algorithm Representation* [Online]. GYPHY. Artificial Intelligence in Video Games, San Jorge University. February 2020. Available from: https://media3.giphy.com/media/LP6F4gZtzSpmsUKf6M/giphy.gif

[8] IGLESIAS, Antonio. *A* Algorithm Representation* [Online]. GYPHY. Artificial Intelligence in Video Games, San Jorge University. February 2020. Available from: https://media2.giphy.com/media/Ur8LzmiXwjOS089TSB/giphy.gif

[9] IGLESIAS, Antonio. *A* Algorithm with Heuristics Representation* [Online]. GYPHY. Artificial Intelligence in Video Games, San Jorge University. February 2020. Available from https://media4.giphy.com/media/RibTTSJT7pVjEB71sL/giphy.gif

[10] EPPES, Marissa. *Game Theory — The Minimax Algorithm Explained* [Online]. Towards Data Science. August 2019. Available from: https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1

[11] SIMPSON, Chris. *Behavior trees for AI: How they work* [Online]. Gamasutra. July 2017. Available from: https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

[12] LEANDRO, TK. *Everything you need to know about tree data structures* [Online]. Free Code Camp. November 2017. Available from: https://www.freecodecamp.org/news/all-you-need-to-know-about-tree-data-structures-bceacb85490c/

[13] EXPERT SYSTEM TEAM. *What is Machine Learning? A definition* [Online]. Expert System. March 2017. Available from: https://expertsystem.com/machine-learning-definition/

[14] ALTEXSOFT INC. *Reinforcement Learning Explained: Overview, Comparisons and Applications in Business* [Online]. The Medium. March 2019. Available from: https://medium.com/gobeyond-ai/reinforcement-learning-explained-overview-comparisons-and-applications-in-business-7ecc8549a39a

[15] SUTTON, Richard S; BARTO, Andre G. *Reinforcement Learning: An Introduction*. First Edition. MIT Press, Cambridge, MA, 1998. 352 p. ISBN: 9780262193986.

[16] UNITY TECHNOLOGIES. *Unity* [Online]. Unity. Available from: https://unity.com/

[17] UNITY TECHNOLOGIES. *Unity ML-Agents Toolkit* [GitHub]. Unity Technologies ML-Agents GitHub. August 2019. Available from: https://github.com/Unity-Technologies/ml-agents

[18] UNITY TECHNOLOGIES. *Unity Asset Store.* [Online] Available from: https://assetstore.unity.com/

[19] HP. *HP Z1 Entry Tower G5* [Online]. HP Store. Available from: https://store.hp.com/us/en/pdp/hp-z1-entry-tower-g5-p-8ag71ut-aba-1

[20] ACER. *Acer Full HD (1920 x 1080) IPS Ultra-Thin Zero monitor* [Online]. Amazon. Available from: https://www.amazon.com/-/es/Acer-SB220Q-Ultra-Thin-Frame-Monitor/dp/B07CVL2D2S/ref=sr_1_2?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=monitor&qid=1597936970&sr=8-2

[21] VICTSING. *VicTsing Wireless Mouse and Keyboard* [Online]. Amazon. Available from: https://www.amazon.com/-/es/VicTsing-Keyboard-Adjustable-Independent-Indicator/dp/B07TT3VN4X/ref=sr_1_5?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=DIYG9P3ZQ34D&dchild=1&keywords=mouse+and+keyboard&qid=1597937010&sprefix=mouse+and+key%2Caps%2C239&sr=8-5

[22] SAN JORGE UNIVERSITY. *Inteligencia artificial aplicada a videojuegos – Guía* [Online]. San Jorge University. Available from: https://gdweb.usj.es/VerHtml?web=1&plan=35&idGuia=12754&version=5.0&idioma=3

[23] KELLY, Adam. *Reinforcement Learning: AI Flight with Unity ML-Agents* [Online]. Udemy. Available from: https://www.udemy.com/course/ai-flight/

[24] INDEED. *Salarios para empleos de Desarrollador/a de software en España* [Online]. Indeed. August 2020. Available from: https://es.indeed.com/salaries/desarrollador-de-software-Salaries?period=hourly

[25] KENNEY. *Kenney Free Gam Assets* [Online]. Kenney. Available from: https://www.kenney.nl/

[26] UNITY TECHNOLOGIES. *Raycast* [Unity]. Unity User Manual (2019.4). August 2020. Available from: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

[27] ALIGNED GAMES. *Polygonal Foliage Asset Package* [Online]. Unity Asset Store. Aprl 2020. Available from: https://assetstore.unity.com/packages/3d/environments/polygonal-foliage-asset-package-133037

[28] NKUEBLER. *NPBehave - An event driven Behavior Tree Library for code based AIs in Unity* [Github]. Meniku GitHub. July 2019. Available from: https://github.com/meniku/NPBehave

[29] UNITY TECHNOLOGIES. *Navigation and Pathfinding* [Unity]. Unity User Manual (2020.1). August 2020. Available from: https://docs.unity3d.com/2020.1/Documentation/Manual/Navigation.html

[30] UNITY TECHNOLOGIES. *NavMesh Agent* [Unity]. Unity User Manual (2019.4). August 2020. Available from: https://docs.unity3d.com/es/2019.4/Manual/class-NavMeshAgent.html

[31] UNITY TECHNOLOGIES. *Box Collider* [Unity]. Unity User Manual (2019.4). August 2020. Available from: https://docs.unity3d.com/es/2019.4/Manual/class-BoxCollider.html

[32] UNITY TECHNOLOGIES. *Rigidbody* [Unity]. Unity User Manual (2020.1). August 2020. Available from: https://docs.unity3d.com/es/2020.1/Manual/class-Rigidbody.html

[33] VAN ROSSUM, Guido. *Python* [Online]. Python. Available from: https://www.python.org/downloads/

[34] UNITY TECHNOLOGIES. *Unity ML-Agents Installation* [GitHub]. Unity Technologies ML-Agents GitHub. August 2019. Available from: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Installation.md

[35] UNITY TECHNOLOGIES. *Class BehaviourParameters* [Unity]. Unity User Documentation. August 2020. Available from: https://docs.unity3d.com/Packages/com.unity.ml-agents@1.1/api/Unity.MLAgents.Policies.BehaviorParameters.html

[36] UNITY TECHNOLOGIES. *Class RayPerceptionSensor* [Unity]. Unity User Documentation. August 2020. Available from: https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Sensors.RayPerceptionSensor.html

[37] UNITY TECHNOLOGIES. *Unity ML-Agents Getting Started* [GitHub]. Unity Technologies ML-Agents GitHub. August 2019. Available from: https://github.com/Unity-Technologies/ml-agents/blob/376168bbb55deac540a572617fb70effffe98cd5/docs/Getting-Started.md

[38] GOOGLE. *Tensorflow* [Online]. Tensorflow. November 2015. Available from: https://www.tensorflow.org/?hl=-419

[39] UNITY TECHNOLOGIES. *Unity ML-Agents Tensorboard* [GitHub]. Unity Technologies ML-Agents GitHub. August 2019. Available from: https://github.com/Unity-Technologies/ml-agents/blob/376168bbb55deac540a572617fb70effffe98cd5/docs/Using-Tensorboard.md

[40] BERGES, Vincent-Pierre. *When to Add reward* [GitHub]. Unity Technologies ML-Agents GitHub. November 2018. https://github.com/Unity-Technologies/ml-agents/issues/1417#issuecomment-438360022

[41] KERBRAT, Sebastién. *Agent Only Learns to Rotate rather than move* [GitHub]. Unity Technologies ML-Agents GitHub. December 2018. https://github.com/Unity-Technologies/ml-agents/issues/1457#issuecomment-446478927

[42] UNITY TECHNOLOGIES. *Profiler* [Unity]. Unity User Manual (2019.4). August 2020. Available from: https://docs.unity3d.com/Manual/Profiler.html

[43] ALLEN, David. *Getting Things Done: The Art of Stress-Free Productivity.* City of Westminster, London, England. Penguin Books. 2001. 267 p. ISBN: 978-0-14-312656-0V.

[44] THOMPSON, Tommy. *Revisiting the AI of Alien: Isolation* [Online]. Gamasutra. May 2020. Available from: https://www.gamasutra.com/blogs/TommyThompson/20200520/363134/Revisiting_the_AI_of_Alien_Isolation.php

[45] CARRY CASTLE. *Source of Madness* [Online]. Steam. 2021 Available from https://store.steampowered.com/app/1315610/Source_of_Madness/

[46] HODGES, Don. *Why do Pinky and Inky have different behaviors when Pac-Man is facing up?* [Online]. DonHoges.com. December 2008. Available from: http://donhodges.com/pacman_pinky_explanation.htm