UNIVERSITAT
JAUME·I

Development and training of an Artificial Neural Network in Unity3D, including a game to interact with it and observe its performance.

Final Degree Work
Bachelor´s Degree in Video Game Design and Development
Universitat Jaume I
June 8, 2020

Author: Sergio Valdelvira Pérez
Advisor: Angel Pascual del Pobil

A.N.N.43

Abstract

This document presents the Final Report of the Video Game Design and Development project. The work consists of creating an AI with the ANN (Artificial Neural Network) method, to face the player in different game modes and then training as best as possible with the Reinforcement learning method. Then, we will introduce our AI into different game environments the player may encounter. All this can be done using the Unity game engine.

Key Words

Artificial Intelligence, Machine Learning, Neural Networks, Brain, Perceptrons, Reinforcement Learning.

Index

Figure Index

1 - Introduction

Work motivation

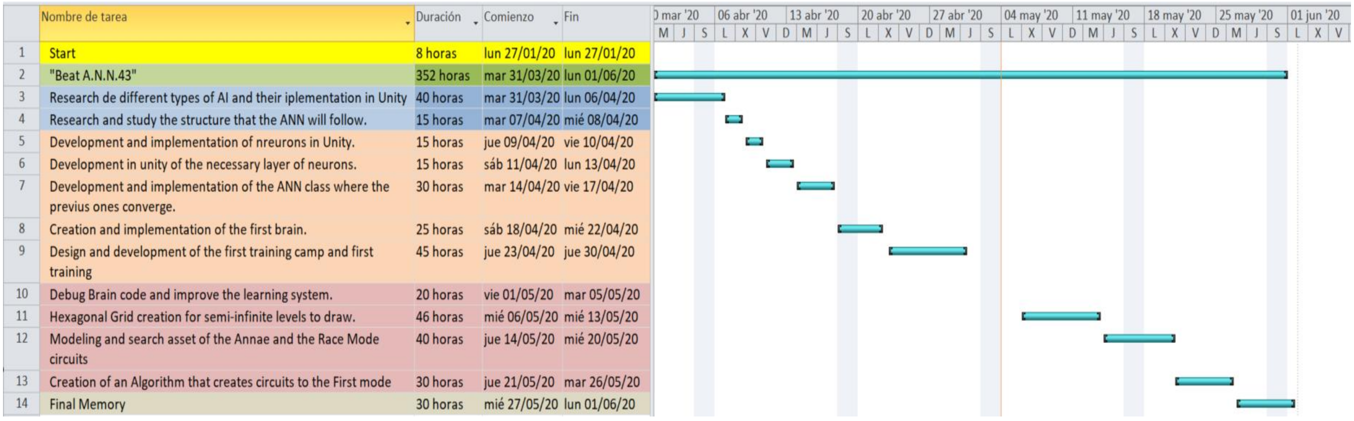
Computational Intelligence based on Artificial Neural Networks, also called ANNs, is not currently being exploited enough in the world of video games, but I believe that if they are well-developed, they have a lot of potential when it comes to generating AI for any video game.

Project's Objectives

- To develop an Artificial Neural Network.
- To train this ANN to pass a simple circuit using Reinforcement Learning.
- To improve training to pass any circuit set by a player.
- To develop a game for the player to interact with the AI.

2 – Planning

This is the planning of the different tasks that have been carried out and in their respective schedules, to achieve the completion of the development of both the ANN and the video game.



	Nombre de tarea	Duración	Comienzo	Fin
1	Start	8 horas	lun 27/01/20	lun 27/01/20
2	"Beat A.N.N.43"	352 horas	mar 31/03/20	lun 01/06/20
3	Research de different types of AI and their iplementation in Unity	40 horas	mar 31/03/20	lun 06/04/20
4	Research and study the structure that the ANN will follow.	15 horas	mar 07/04/20	mié 08/04/20
5	Development and implementation of nreurons in Unity.	15 horas	jue 09/04/20	vie 10/04/20
6	Development in unity of the necessary layer of neurons.	15 horas	sáb 11/04/20	lun 13/04/20
7	Development and implementation of the ANN class where the previus ones converge.	30 horas	mar 14/04/20	vie 17/04/20
8	Creation and implementation of the first brain.	25 horas	sáb 18/04/20	mié 22/04/20
9	Design and development of the first training camp and first training	45 horas	jue 23/04/20	jue 30/04/20
10	Debug Brain code and improve the learning system.	20 horas	vie 01/05/20	mar 05/05/20
11	Hexagonal Grid creation for semi-infinite levels to draw.	46 horas	mié 06/05/20	mié 13/05/20
12	Modeling and search asset of the Annae and the Race Mode circuits	40 horas	jue 14/05/20	mié 20/05/20
13	Creation of an Algorithm that creates circuits to the First mode	30 horas	jue 21/05/20	mar 26/05/20
14	Final Memory	30 horas	mié 27/05/20	lun 01/06/20

Figure 1 Gantt chart

Related Courses

- VJ1231 - ARTIFICIAL INTELLIGENCE
- VJ1222 - VIDEO GAME CONCEPTUAL DESIGN
- VJ1229 - STATISTICS AND OPTIMIZATION
- VJ1227 - GAME ENGINES

3 - Game Design

Create an ANN capable of passing any race circuit that the player can draw. However, to do this the ANN must first be created and trained with Reinforcement Learning to pass a basic circuit, and then the level must be raised to ascertain whether it can be improved.

The game that the player is going to use will be an editor where he can draw the circuit that the IA must beat (drawing a circuit that takes more than 1000 attempts to solve), watch training with new brains, and even compete against the previously trained ANN.

There is also a Race Mode where the player will face 3 more ANNs from a random list of circuits with different environments.

Overview

Game concept:

"Beat A.N.N. 43" is an experimental videogame, in which the main character is not you. It is about seeing the performance of "Annae," (acronym and name of A.N.N. 43) an ANN previously trained to beat any circuit and obstacle that the player can imagine. The action of the game can be contemplated both from a zenithal view of the circuit and in third person, using a PC as a platform.

The visual style of "Beat A.N.N. 43" is not one of its most important factors, although it is not to be underestimated either, since it is a low poly aesthetic with flat colors, leaving more pleasant and attractive environments and agents to the players.

The player can play in different game modes, and some of them with different difficulties, such as:

- Create circuits with obstacles, and try to make them unbeatable for Annae.
- Participate in races on different random circuits, competing with different versions of Annae and her different difficulties.
- See the training and learning process of an Annae unit from scratch.

Genre:

Logic & Race

Target Audience:

Users interested in both Artificial Intelligence and Artificial Neural Networks applied to video games, and even any arcade racing game player.

Look and feel / game experience:

The player must feel equal conditions against the ANNs at all times enjoying the competitive component against ANN, with simple and intuitive controls and pleasant scenarios and interfaces.

Gameplay and mechanics

Gameplay overview:

Many games use pre-programmed artificial intelligence, with a few actions pre-set by the developers, whether or not those are the best strategy to try to beat the player. In this case, we use a previously trained ANN who has tried thousands of strategies until she found the best one.

Game Modes: The player has different types of interactions with the game depending on the chosen mode:

Draw your circuit mode, which, as its name suggests, consists of preparing a circuit for “Annae” to try to make her fail. The player is given a grid of hexagons which he can select to shape the circuit that he has in his imagination.

Race mode, where we will select the difficulty of “Annae” and compete against her in different random circuits, driving another “Annae” unit.

Training mode, which is the least interactive because it consists of watching the training process of an “Annae” unit from the beginning.

Environments: We will have two types of very different environments, both in the Drawing Mode and in the Training Mode, which intend to give the impression of an experimental game and of working with robots. Here is where the different training tests are going to be performed, or there are even remains from previous tests. It is very bright so it gives a great feeling of spaciousness, with a certain industrial style where light colors predominate, in particular white.

On the other hand in the Race Mode, the player will be outside with much more natural scenarios, where the different random circuits will be inserted in different natural environments, with the asphalt of the circuit contrasting heavily with these environments.

Game progression:

As with any game that focuses on its game modes or mini-games, progression is not a priority. A more important element is to have fun exploiting these almost infinite modes to the fullest.

How do you know the goal? In the game's introduction you will meet Professor Roger Poplar, who will explain that he has developed a new robot and ask you to help him prove how advanced he has become. Poplar will ask you to defeat “Annae” in different tests.

So there is no progress? There is some progress in both circuit and race mode. In the circuit mode, you must defeat “Annae” at least once to be able to unlock the obstacles, and in the race mode, “Annae” will have three types of difficulties: Easy, Mid and Hard.

Mission / challenge / level structure

The main mission in all modes is to beat “Annae,” either by designing complex circuits that she is unable to overcome, or by reaching the finish line before her in the Race mode.

The challenge consists of defeating an artificial intelligence trained during tens of hours in different circuits, managing to improve it, based on the failed as well as the successful attempts of the player. This is because the AI will be storing all kinds of information, learning from all its mistakes.

Random circuits: In race mode, neither contestant will know the circuit in advance, adding a race game factor.

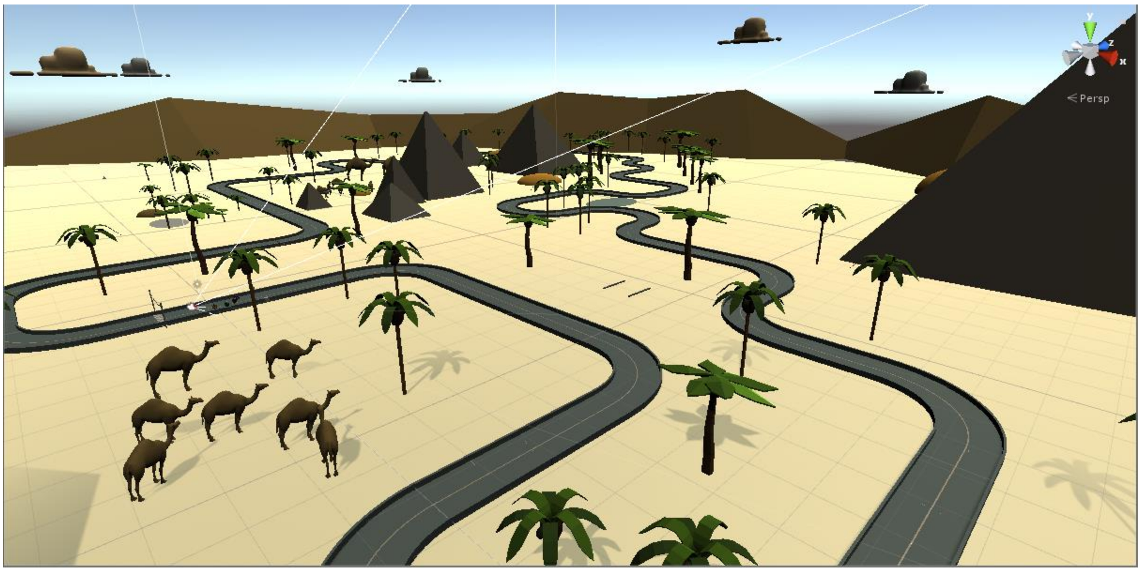


Figure 2 Random Circuit

Movements and controls:

The player will only be able to move in the Race Mode when competing against "Annae." In a very simple way, using the WASD keys:

up: Speed up
down: Brake
left: Turn left
right: Turn right

Figure 3 Controls

While in the Draw your circuit mode, the player will use the left mouse click to select the shape of the circuit as well as the positioning of the obstacles, and the right mouse click to move the camera.

Physics and objects:

The decorative objects will be static objects that, once chosen, their position will not be affected at all even if "Annae" crashes into them. They will simply be used by their colliders to detect if "Annae" has crashed.

Actions:

Although “Beat A.N.N. 43” is a simple game in terms of visuals and controls, it still has a fun and entertaining gameplay, while being re-playable.

Restricted areas: The laboratory (Draw your Circuit Mode) as a restricted area provides the possibility of making almost infinite circuits for “Annae,” plus the ability to set up obstacles at any point of the routes or not.
Players can access this title even if it focuses on the experimentation of Artificial Intelligence and the behavior of Artificial Neuronal Networks, because the actions and interactions needed to enjoy the game are accessible to any player.

Screenflow:

Depending on the selected game mode, we will have different viewpoints:

-The first in Draw your circuit mode is similar to the vision a scientist would have, watching as his mice try to reach the end of a maze to get the reward.

-With the second viewpoint in Race mode, the player will have the third person view, similar to that of another competitor like “Annae.”

Game flow:

- First Splash screen: Unity cinematic.
- Second Splash screen: Title cinematic
- Introduction
- Main menu: A screen appears with the following options.
    - Play
        - Draw your circuit
        - Race Mode
        - Training Mode
    - Options
    - Exit

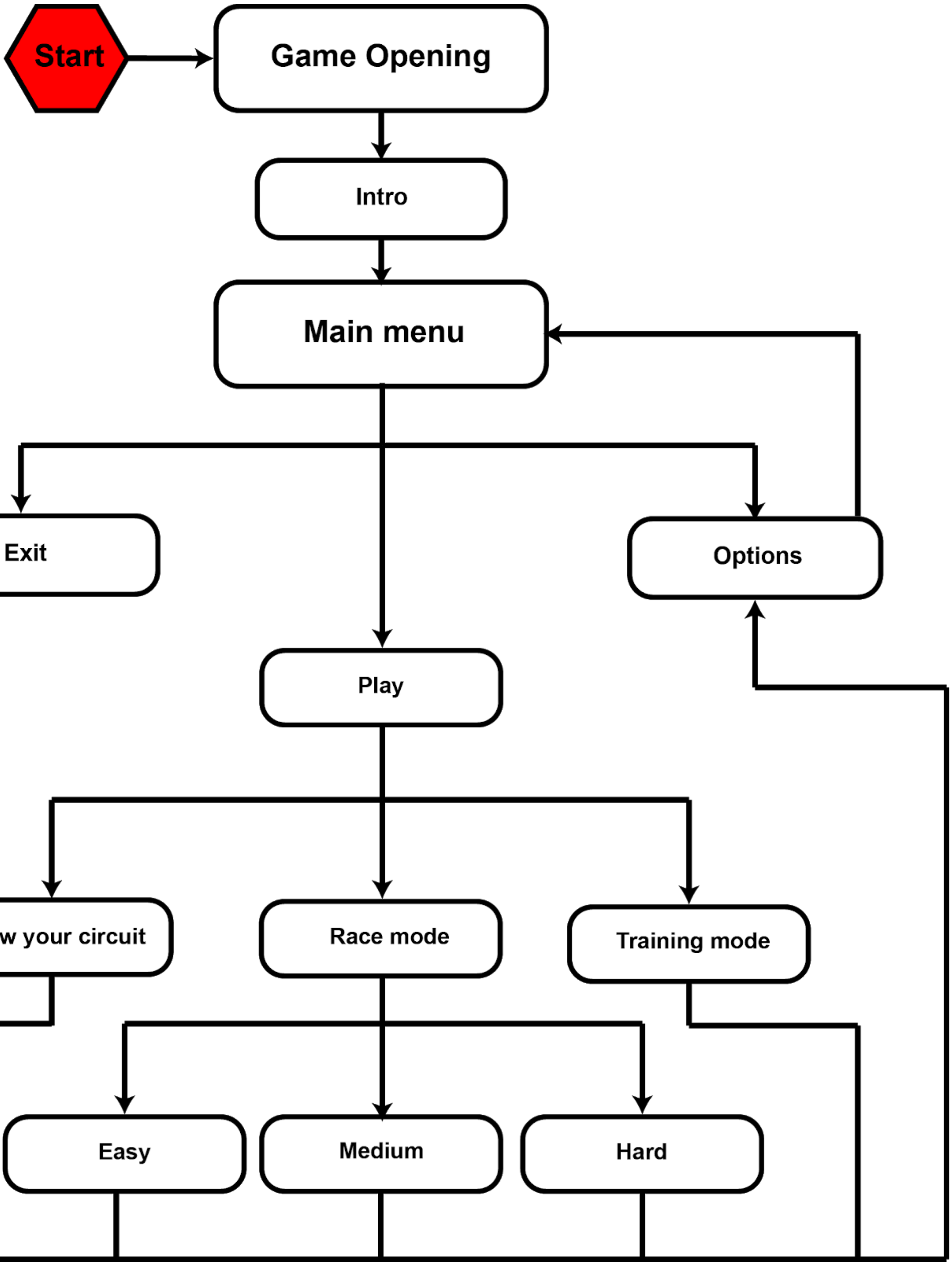


Figure 4 Game Flow

Replaying and saving:

The replayability is affected by the player, because he has an infinite number of options when drawing the circuit.
In the race mode the player will have at his disposal several circuit maps that will be loaded randomly.

In each race in any of the modes, “Annae” will assimilate the new data that she believes to be relevant for later confrontations.

Narrative design

Storyline:

Professor Roger Poplar has developed a robot with self-learning ability, A.N.N. 43, and he needs your help in his research.

Characters:

Roger Poplar: A scientist focusing on robotics with the objective that his creations evolve with experience.

A.N.N. 43: Unit developed by Prof. Poplar with the ability to learn.

Gameworld:

Beat A.N.N. 43 takes its name from the fact that it competes with and tests the robot developed by Professor Poplar. This robot is called A.N.N. 43 or “Annae” because it is an Artificial Neural Network followed by a number, which is not the first version developed by Poplar.

The game always takes place in controlled environments in the laboratory used in the Draw your circuit and Training modes as well as in the different outdoor scenarios of the Race mode.

BEAT A.N.N. 43

HELP PROFESSOR POPLAR IN HIS RESEARCH.

PROFESSOR ROGER POPLAR HAS DEVELOPED A ROBOT WITH SELF-LEARNING ABILITY, A.N.N. 43, HE NEEDS YOUR HELP IN HIS RESEARCH.

CHALLENGE AND COMPETE AGAINST A.N.N 43 TRYING TO BEAT HIM. ALL THIS WILL HELP THE UNIT TO EVOLVE AND CONTINUE LEARNING.

PLATFORM: PC
AGE: +12
GENRE: LOGIC + RACE

A SCIENTIST ORIENTED TO ROBOTICS WITH THE OBJECTIVE THAT HIS CREATIONS EVOLVE WITH EXPERIENCE.

UNIT DEVELOPED BY PROF. POPLAR WITH THE ABILITY TO LEARN.

PROF. ROGER POPLAR

A.N.N. 43

Functional and technical specifications

Tools:

unity	UNITY 2019 2.8f1
Visual Studio	VISUAL STUDIO 2017
Ai Adobe Illustrator	ADOBE ILLUSTRATOR 2018
Ps Adobe Photoshop	ADOBE PHOTOSHOP 2018
blender	BLENDER 2.8
Office	Microsoft Office 2019

4 - Work Development

What is an ANN? It is a set of both neurons arranged in layers creating a network or system so that the brain of an AI of the ANN may function. Starting from the bottom, there are Perceptrons or neurons.

Perceptrons/ Neuron

A Perceptron is an algorithm that imitates the operation and behavior of a neuron that is likewise capable of learning.

On one hand, the Perceptron takes an input, processes said input with its own function, and makes a decision that will come out through the Output. (Fig 5)

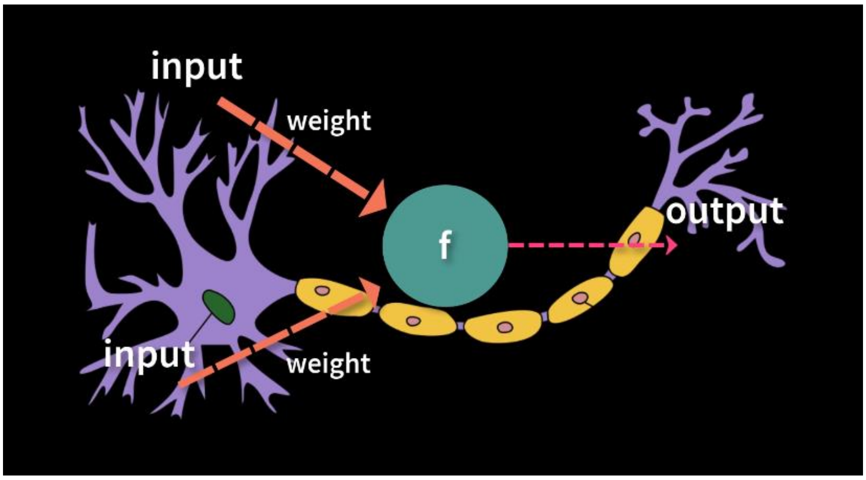


Figure 5 Neuron - Representation

Between inputs and function, there are weights involved in the final decision depending on the value of the weights, which signifies learning and being able to see ANN evolve. In addition to inputs, there is the bias for correct operation

The bias is a value that changes the value of the function regardless of inputs.

The functions, also called activation functions, vary according to the result to be obtained, e.g. from of 0 to 1 or from -1 to 1.

What we want to accomplish with the weights is for them to give the suitable result to the neurons, and this is done by multiplying the sum of inputs by their weights, adding the bias in the end. If the result is greater than 0, the neuron is activated; otherwise, it is not.

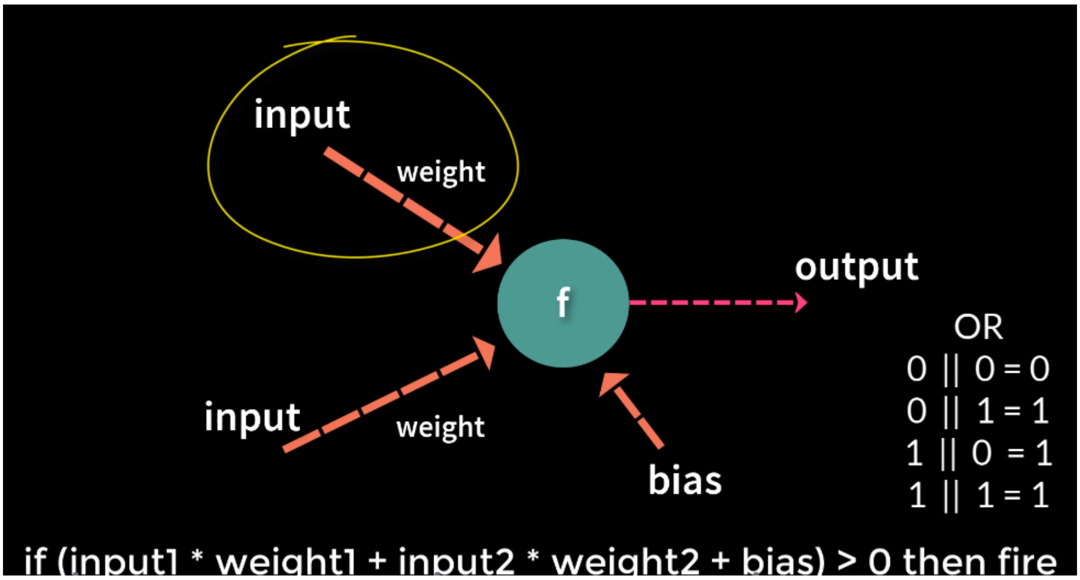


Figure 6 How Perceptron work

A.N.N

The concept of neuron is thrilling, but what is most interesting is when several of them are brought together, thereby creating an artificial neural network.

In a basic neural network, Perceptrons are arranged in three or more layers, the input layer, one or more hidden layers, and a final output layer, with all the neurons being connected between with the neurons in the next layer.

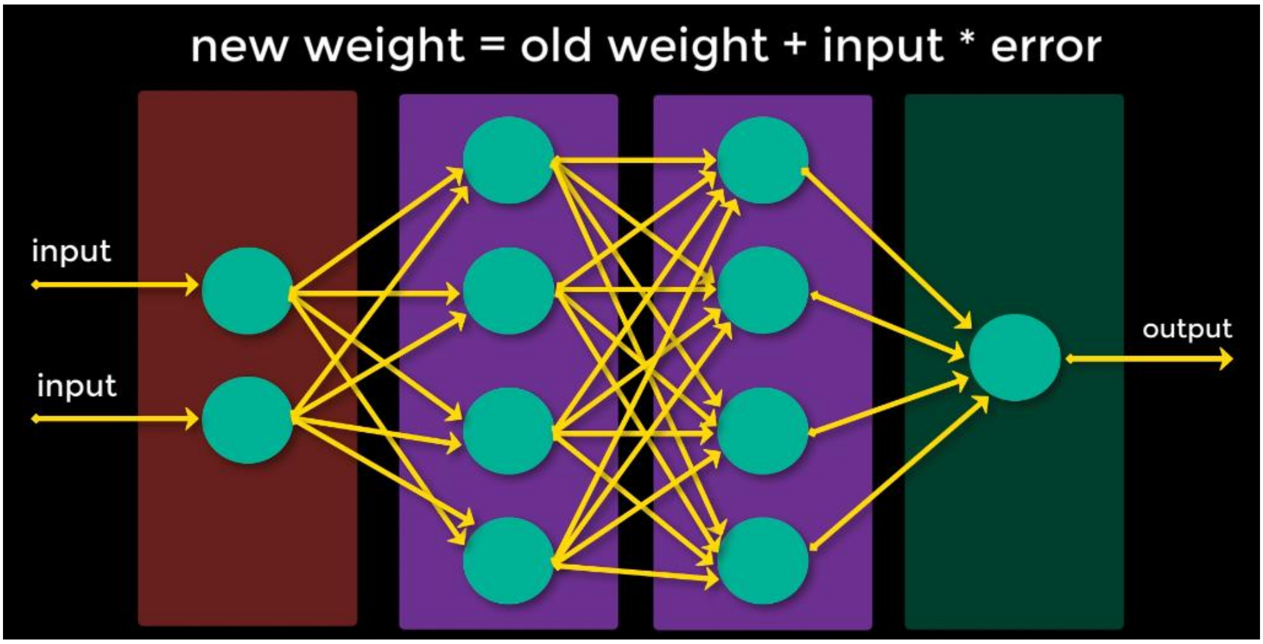


Figure 7 ANN Graph

The complete neural network basically works like a single neuron. The generated output is compared with the desired output, and the error is calculated and sent back with a technique called back propagation, which indicates where the error has been most decisive, giving priority to a modification of said neuron.

ANNs have an amazing processing capacity and have been used both in video games like in other applications outside the video game environment.

Brain

The ANN we already have is executed in the brain, like the learning process, which is Reinforcement Learning in this case. To start, we first must know how many inputs will be needed for our ANN and how many outputs and hidden layers.

In our case, since the game is based on a vehicle powered simply by a forward force, we know that two outputs are needed, one for turning to the left and another for turning to the right. In this example, we have chosen to place a type of eyes on our vehicle which is what carries the brain.

These eyes are five Raycasts:

One rotated 90º to the left.

One rotated 90º to the right.

One rotated 45º to the left.

One rotated 45º to the right.

One looking straight ahead.

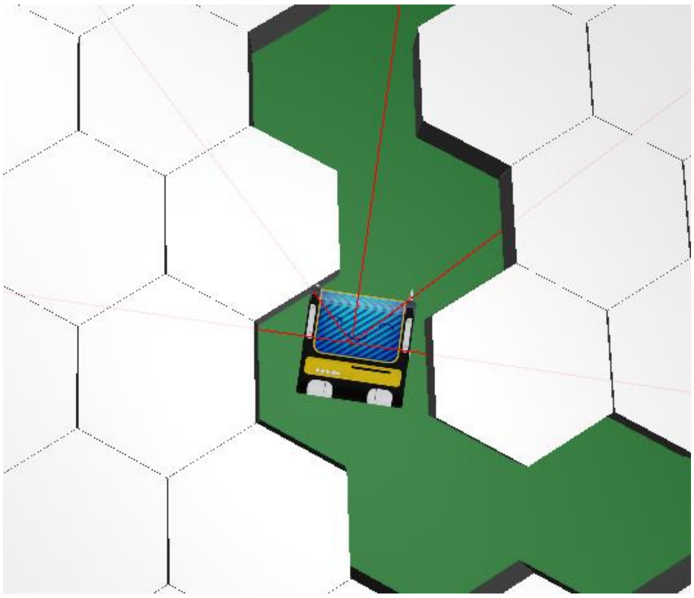
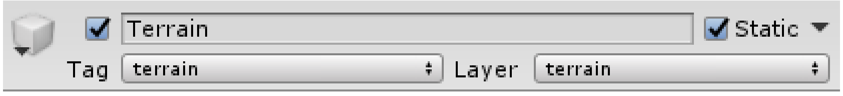


Figure 8 RayCast

These Raycasts (inputs), which are actually devolving, is the distance from the vehicle to any obstacle with the terrain layer that it may run into, thereby giving it the effect of vision with 5 eyes.



☑ Terrain ☑ Static

Tag terrain Layer terrain

Figure 9 Layer Terrain

Now training must take place. As discussed, we simply place in the scenario a circuit with a pair of curves, and we test the experiment.

It works yet can still be improved because it never turns around the circuit completely. To do that, we make the decision and train in the learning section.

Durante travel, if the vehicle crashes, it restarts in this position so that it can try to get through the circuit again, and the vehicle is penalized with -1. However, if the vehicle comes out alive, it is rewarded with 0.1. This entire learning process is performed when the vehicle crashes (BackPropagation).

To improve this learning algorithm, during travel we add waypoints which are added to the reward with 0.5 + 0.1 (reward: It is a variable, the higher it is, it means that it finds its way until in the end it does not make any mistake) for being alive every time it passes one of these waypoints, thereby finally resolving the circuit.

Now all that is left to do is save the weights so that they can later be loaded and the circuit resolved on the first attempt.

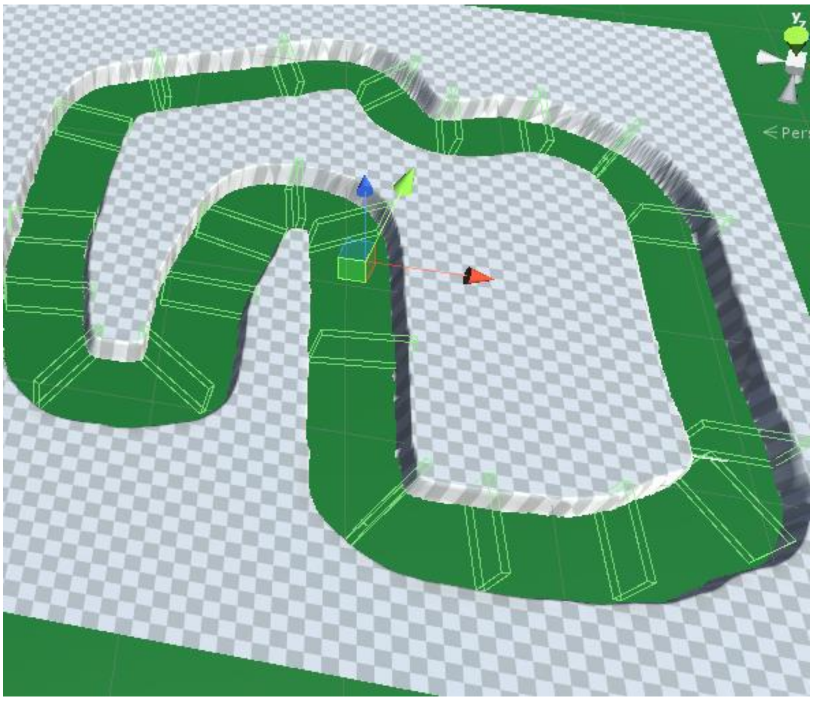


Figure 10 WayPoints

To save these weights, we need a Script for loading and saving weights. In this first part of the save Script, the word “Test” is used so it can be saved in a text file with the weights, and in the Brain itself we add the condition of checking to see if the circuit is saved, basically if the file called “Test” exists and the circuit is saved. Then it is loaded, going through the circuit in the first attempt without any errors. If the file does not exist, the Brain switches to training mode until it reaches the last of the waypoints to which we have assigned the ID “Finish Line” and the save Script is run, creating the file called “Test.”

Modeling and artist part

Before starting any of the game modes, we have to model the robot, our star player, and the objects we need, and we have to draw Dr. Poplar. This, along with free Asset and model downloads, can all be found in the art section of the video game.

The modeling part is done with the Blender 2.8 program together with the Box Cutter add-on. The drawing of Dr. Poplar is done with Photoshop, and the drawing of all the necessary Canvases, both keys and menus, etc., are done with Illustrator.

In Blender we also create the first piece of the first game mode that we are going to program, that is, a Cell which is basically is a hexagon with volume.

The robot animations modeled in Blender were done with the Unity animation editor, requiring no more than two animations, IDLE and RUN, and the suitably separated parts of the robot. Therefore, rigging the body for better animations is not necessary.

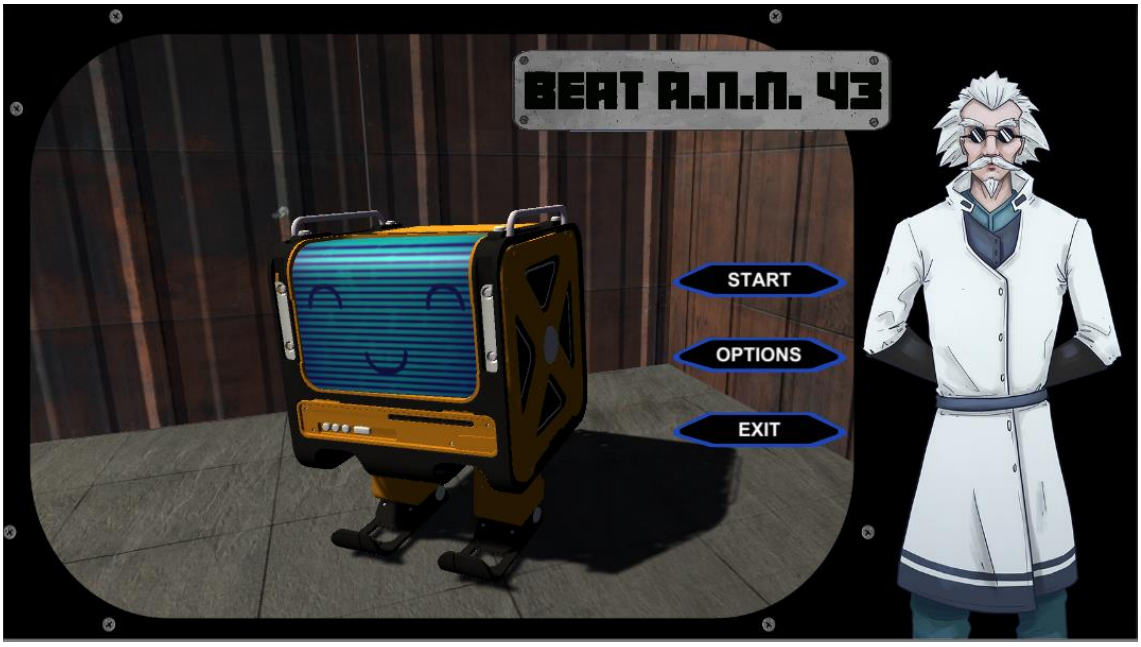


Figure 11 Main Menu 1

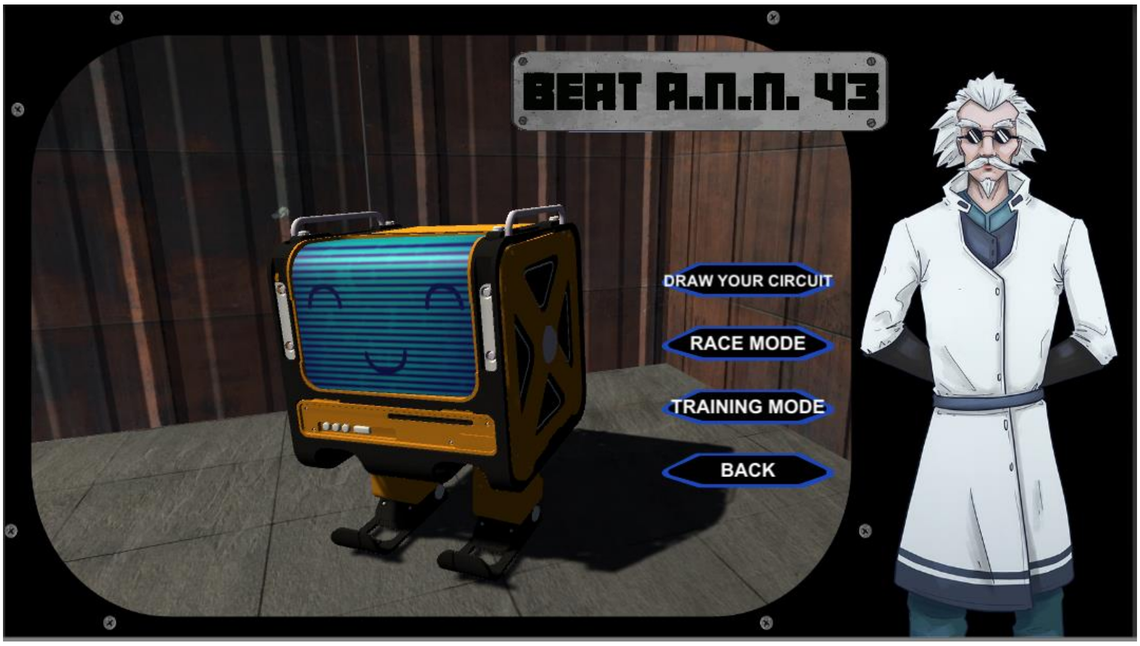


Figure 8 Main Menu 2

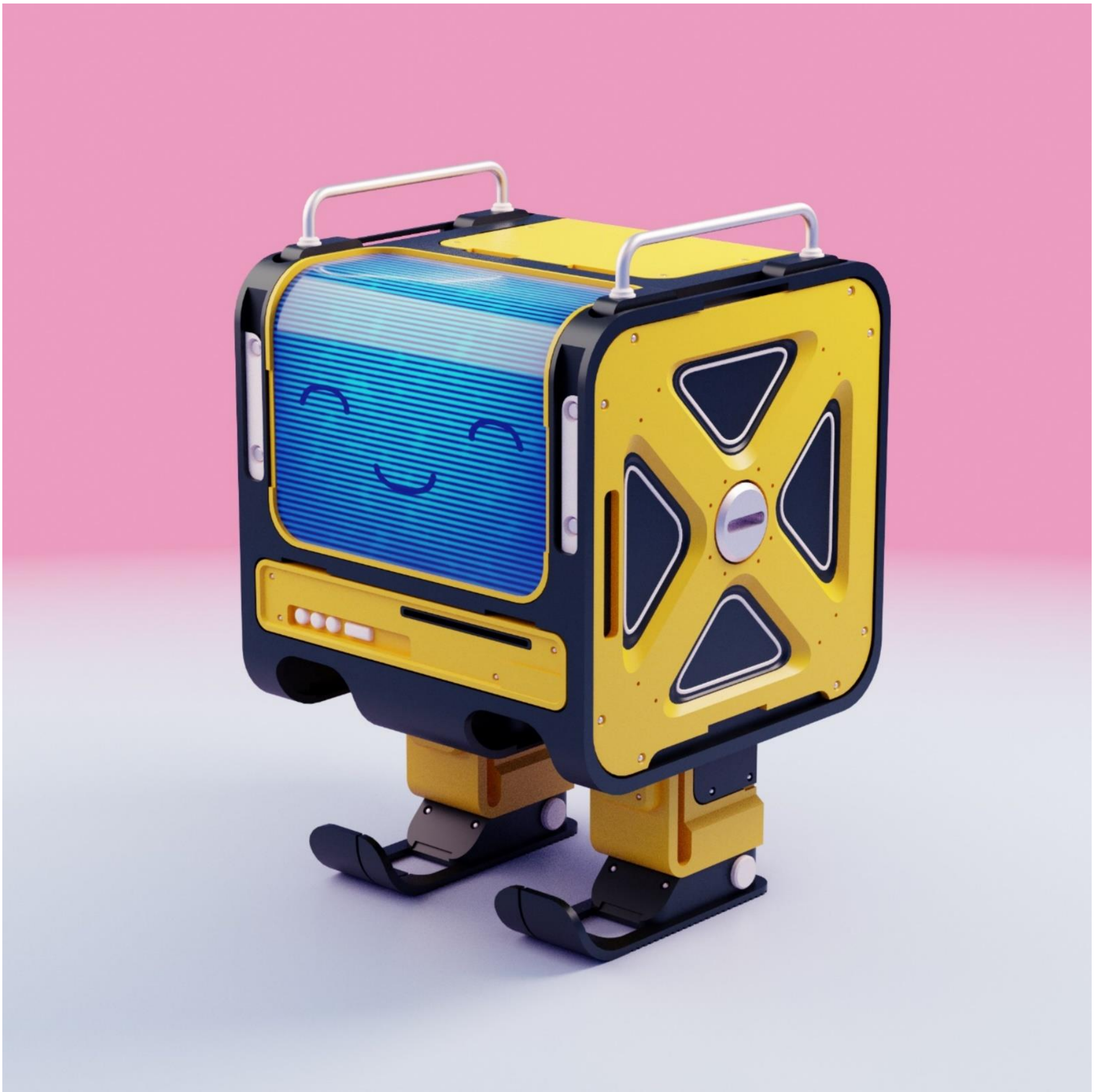


Figure 9 "A.N.N. 43"

“Draw your Circuit” Mode

To start, we import into Unity our Cell object modeled in the preceding point. There is a new script added to it called “Cell.cs,” where we create the ID variables, whether or not it is selected, whether or not this is the starting point, and a list of neighboring cells.

Next we create a matrix with these hexagonal cells using Grid.cs Script.

Since the cells are hexagonal, we have to add an offset in the horizontal displacement of the cells in even rows for the result to be visually appealing. However, this causes an error, since the matrix is hexagonal. With this offset, the cells are disarranged, the current neighboring cells are not the right ones, for example [0,0] might not have thereunder [1,0], and that error is transmitted to all the cells in the matrix.

To fix this, we simply create two for loops to run through the matrix. Boundaries, or better yet a black frame, are put up, and they run through it differently according to whether they are even or uneven, the right neighboring cells are gradually added to the lists of neighboring cells of each of the cells.

After that, cell [3,3] is painted red and selected as an output cell because that cell are going to be the start and the end of the circuit.

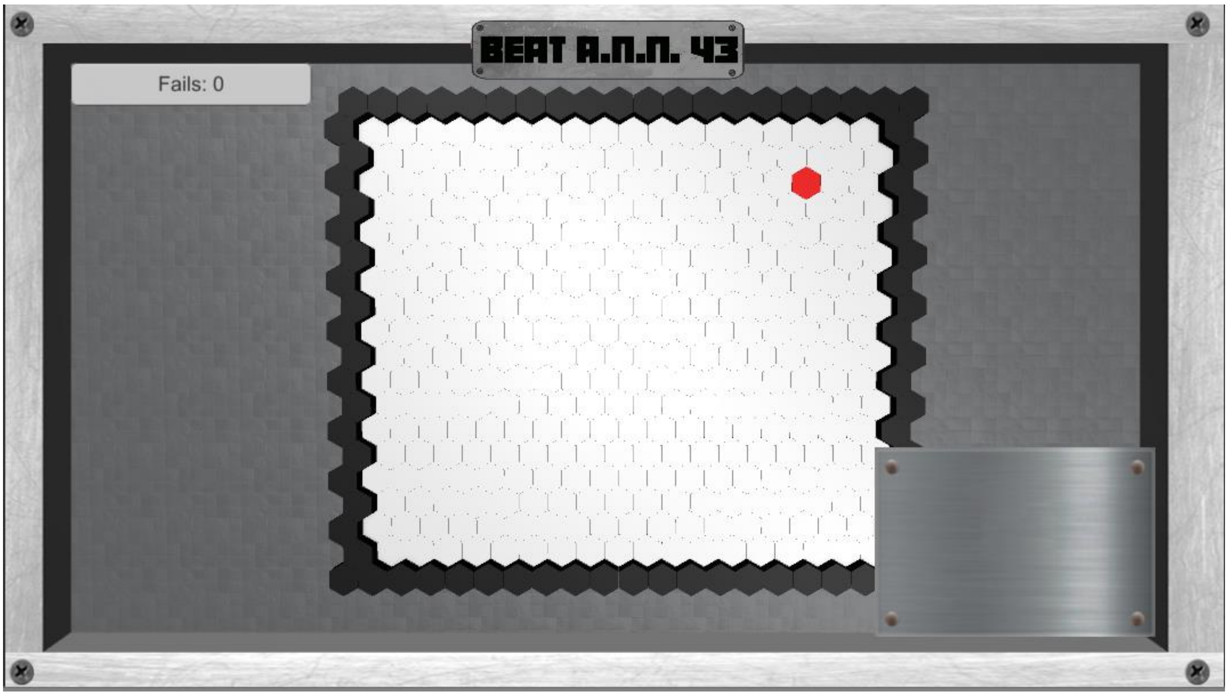


Figure 12 Hexagonal Grid

With the grid and the starting point created, it is now time to give the player the tool to draw the circuit he wants in this grid, always beginning at Start.

We have for that purpose the SelectionAction.cs code, where it is possible to find the following variables: a list of selected cells, a current cell, a number of waypoints and a list of waypoints.

The player will make the selections using a Raycast going from his pointer to the place where is a cell and if it is connected with a selected cell, the player will be allowed to continue selecting. However, if at any time the left mouse click is not pressed, the list of selected cells will be erased. This is to prevent players from drawing impossible circuits or circuits forked paths. There is also the rule that any single cell can have two selected neighboring cells.

Conversely, if the player keeps the left mouse click pressed and does not try to add a third neighbor, anywhere in the list, he has complete freedom to draw the circuit he wants in the grid and to close it by reuniting with the Start cell.

Once a valid circuit is recognized, all the unselected cells are scaled on the "Y" axis and have the tag and layer indicating "terrain," leaving the path of selected cells that are not augmented in size. A waypoint every three cells is also instantiated in this list, which is converted into a circuit.

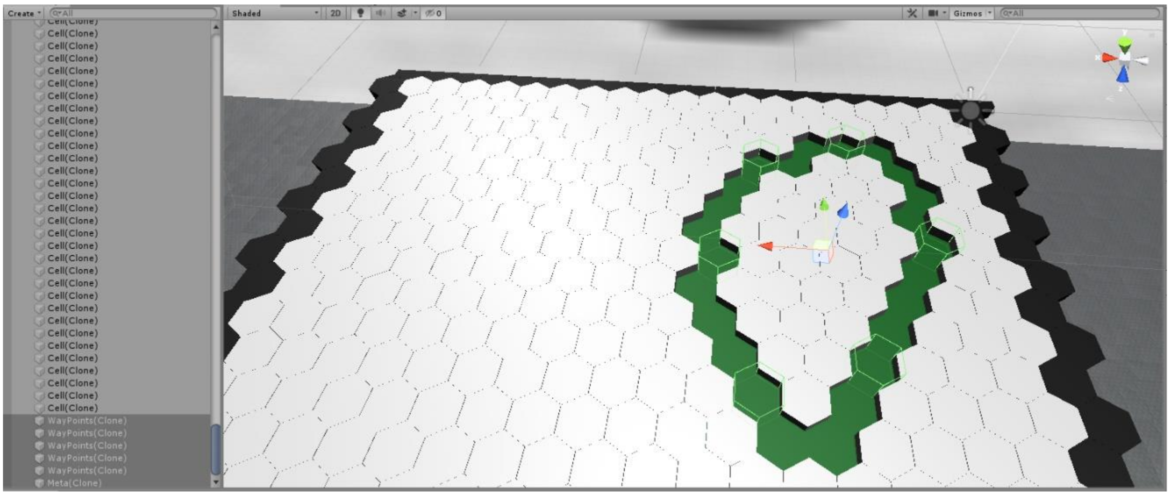


Figure 13 Valid Circuit

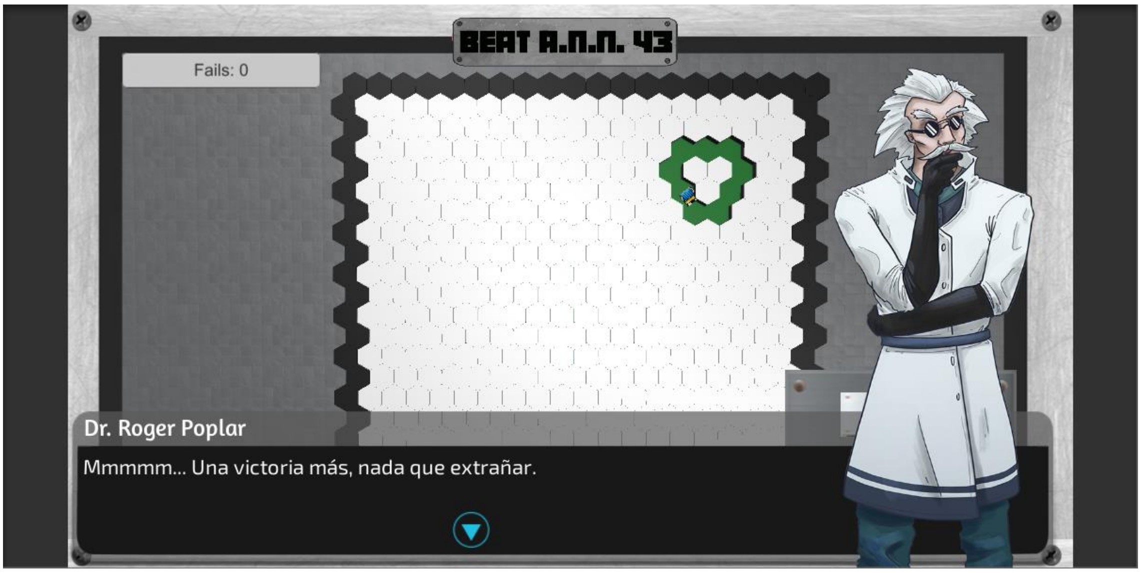
Once everything is ready, the bot is instantiated looking at the first cell of the list, and training begins. The last finish line of this list is instantiated as the finish line.

How to save circuits needs to be implemented, and from this point on, all of those generated in the grid will be given an ID or name, followed by the weights needed to resolve the circuit once it has been completed.

The ID or name is the set of all the IDs of the cells making up the list of the circuit so that it can be identified in the future, and training does not have to be repeated, otherwise load the suitable weights from the Circuits.txt file where they are all saved.

To convert all this into a small game, we use the Fungus Asset for Dr. Poplar to challenge you to create a circuit that is complicated enough for the robot to take over 1000 tries to surpass it (victory condition).

In both cases, whether you succeed or not, Dr. Poplar makes a comment about it and if he has won he stays repeating the circuit. While if the player loses, stays in training mode until complete.



Dr. Roger Poplar

Mmmmm... Una victoria más, nada que extrañar.

Figure 14 Winning

The possibility of having two cameras, a zenithal and a third person camera, which can be alternated by pressing “v” as a result of the Cinemachine Asset providing the player with more dynamic cameras, are added to this game mode.

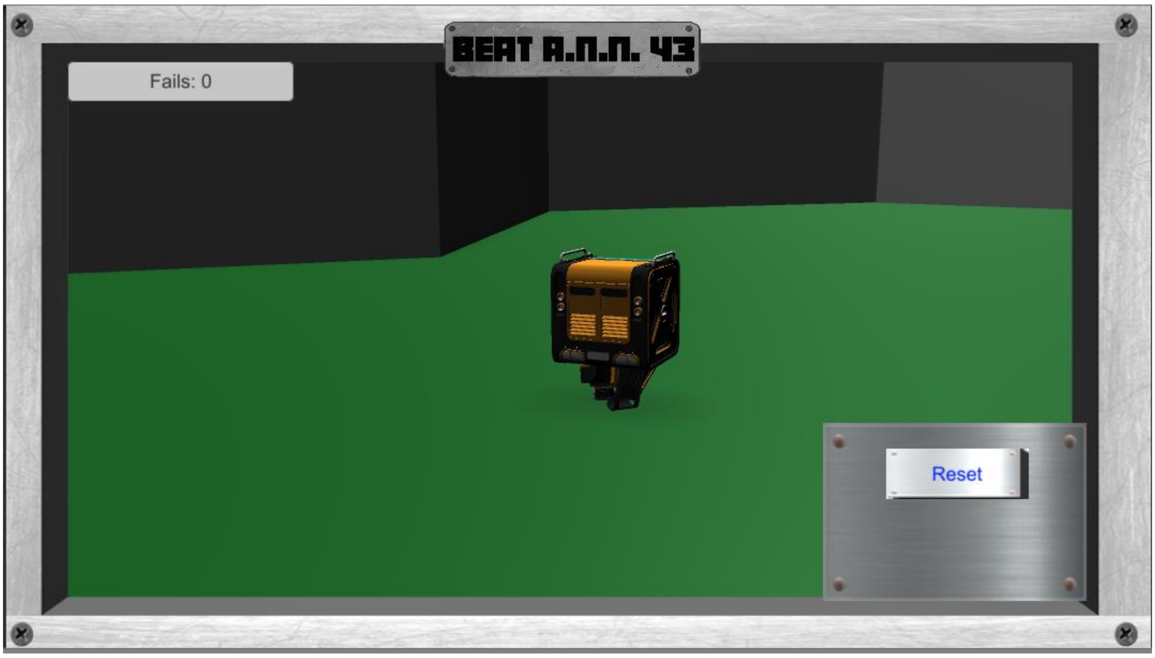


Figure 15 Third View

The circuit implementation system is also updated once a valid circuit is recognized, where two keys are shown

- Start: To start the training mode or loading mode, if this circuit has already been passed through.
- Reset: If the drawn circuit does not convince the player and he prefers to draw it again.

A pause menu is also added with the following options:

- Resume: Eliminates the pause on the game.
- Menu: Takes you to the main menu.
- Help: Dr. Poplar offers a small tutorial on how this game mode works.

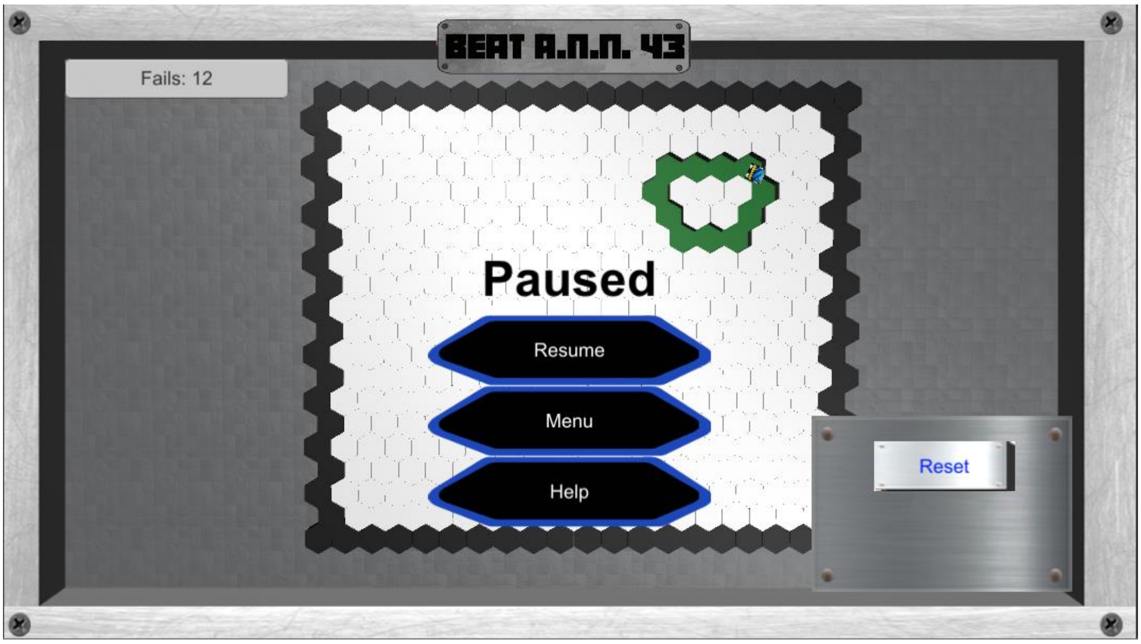


Figure 16 Pause Menu

Race Mode

The Race Mode is made up of four scenes with different circuits and scenarios, where the objects needed to give the right atmosphere, as well as the modular pieces of the circuit, have been downloaded or modeled. In general, however, these four scenarios have exactly the same programming, with the variable the bots have presenting a minor new variable in order distinguish the circuits.

In the main menu, when we select the Race Mode option, one of the circuits is randomly chosen so as not to have to always start with the same one.

The working of one of these Scenes discussed above, all of which work in a similar manner, will now be explained.

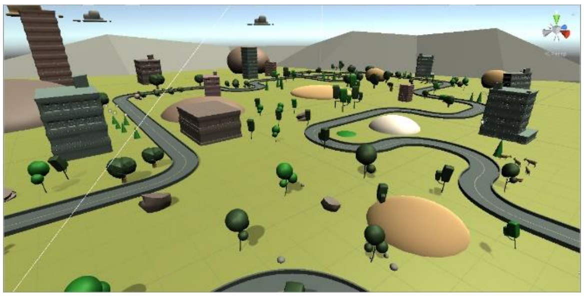


Figure 18 Circuit 1

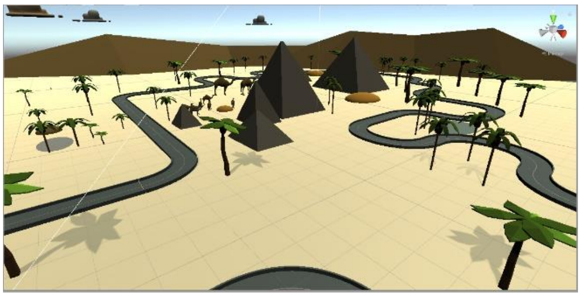


Figure 19 Circuit 2

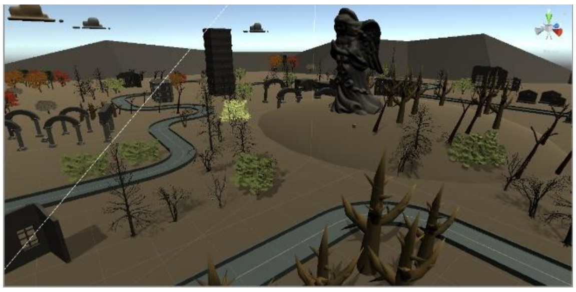


Figure 17 Circuit 3

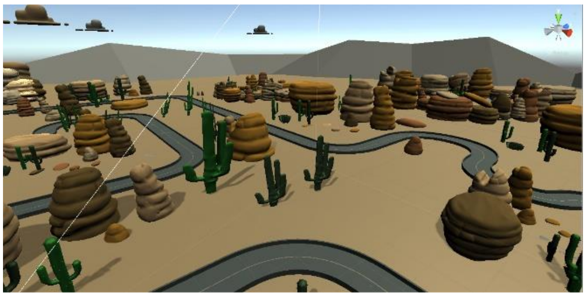


Figure 20 Circuit 4

We start by making a circuit with the modular tracks saved in an object given the same name. When we have the circuit completed, we choose the point of departure and create the WayPoints Position.cs Script, where in the center of each of the pieces of the Modular Track a Waypoint of the same kind seen in the Draw your Circuit section is instantiated, but this time with rectangular shapes, with the last one being indicated as the Finish Line.

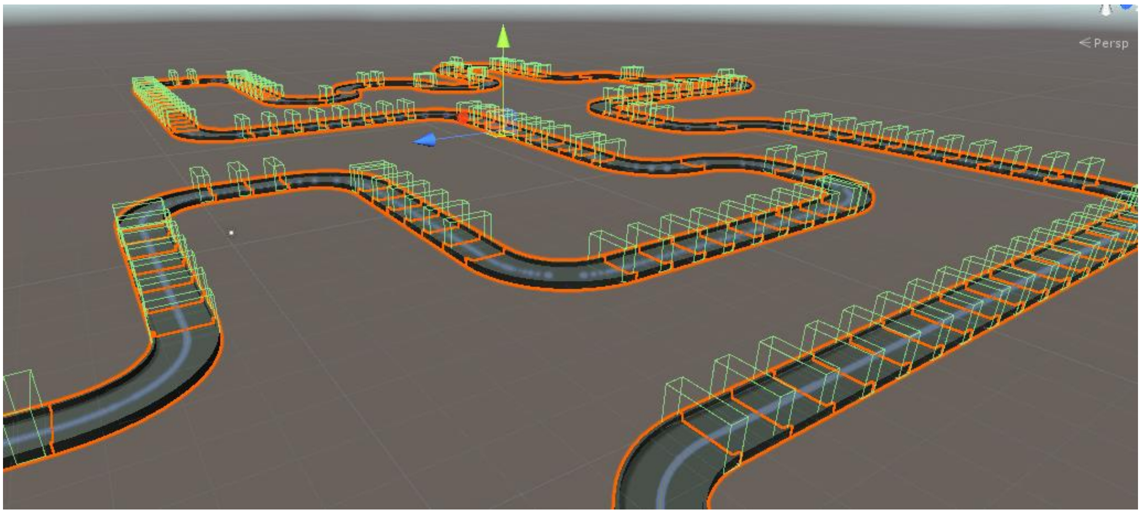
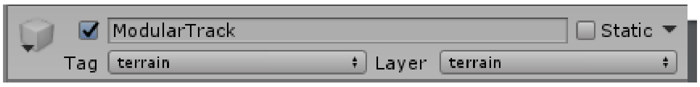


Figure 21 WayPoints in Modular Track

To continue preparing the circuit, both the tag and the layer are added to all the pieces of the modular track as terrain, and an invisible plane is added on the road, but a few millimeters above it, so that in the later bot training mode, they can only crash into the barrier of the circuit, as a result of the Mesh Collider in each of the pieces, and not into the ground as a result of this invisible plane.



☑ ModularTrack			□ Static
Tag	terrain	Layer	terrain

Figure 22 Layer and Tag from de Modular Track

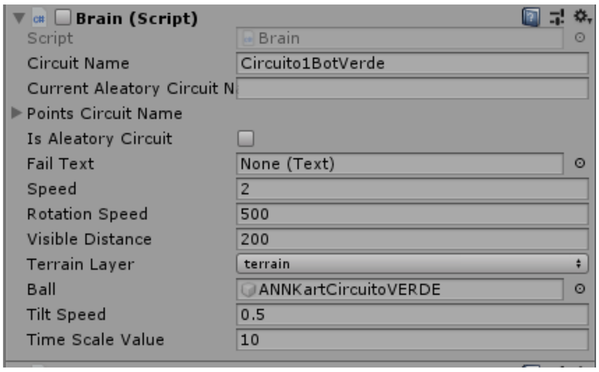
With the finish line and the starting point established, all that is left is to place both the bots, with three in all the circuits, and the robot controlled by the player.

The Player Movement.cs code is very simple. Using the forward and backward arrows, we can move forward and backwards, while the left and right arrows allow us to rotate in these directions. Little work is applied to the movement code because it has to be fair in order to compete against it, with both having the same movement speed.

On the other hand, the Cinemachine is used again as the Main Camera in third person, providing us with a much more dynamic camera.

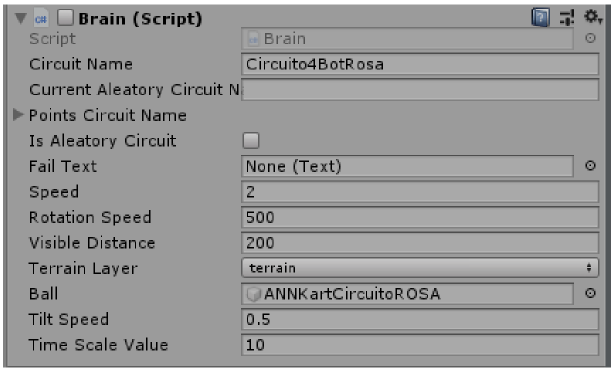
Now we have to train the bots, and to do this their Brain.cs has to be improved by adding restrictions in order to know if they are in the Draw your Circuit or in one of the modeled circuits. Therefore, they already distinguish being in the Race Mode.

By adding another restriction, an ID is added for the robot to know who it is and the circuit it is in.



□ Brain (Script)	
Script	Brain
Circuit Name	Circuito1BotVerde
Current Aleatory Circuit N	
Points Circuit Name	
Is Aleatory Circuit	□
Fail Text	None (Text)
Speed	2
Rotation Speed	500
Visible Distance	200
Terrain Layer	terrain
Ball	ANNKartCircuitoVERDE
Tilt Speed	0.5
Time Scale Value	10

Figure 24 Circuit 1 Green Bot



□ Brain (Script)	
Script	Brain
Circuit Name	Circuito4BotRosa
Current Aleatory Circuit N	
Points Circuit Name	
Is Aleatory Circuit	□
Fail Text	None (Text)
Speed	2
Rotation Speed	500
Visible Distance	200
Terrain Layer	terrain
Ball	ANNKartCircuitoROSA
Tilt Speed	0.5
Time Scale Value	10

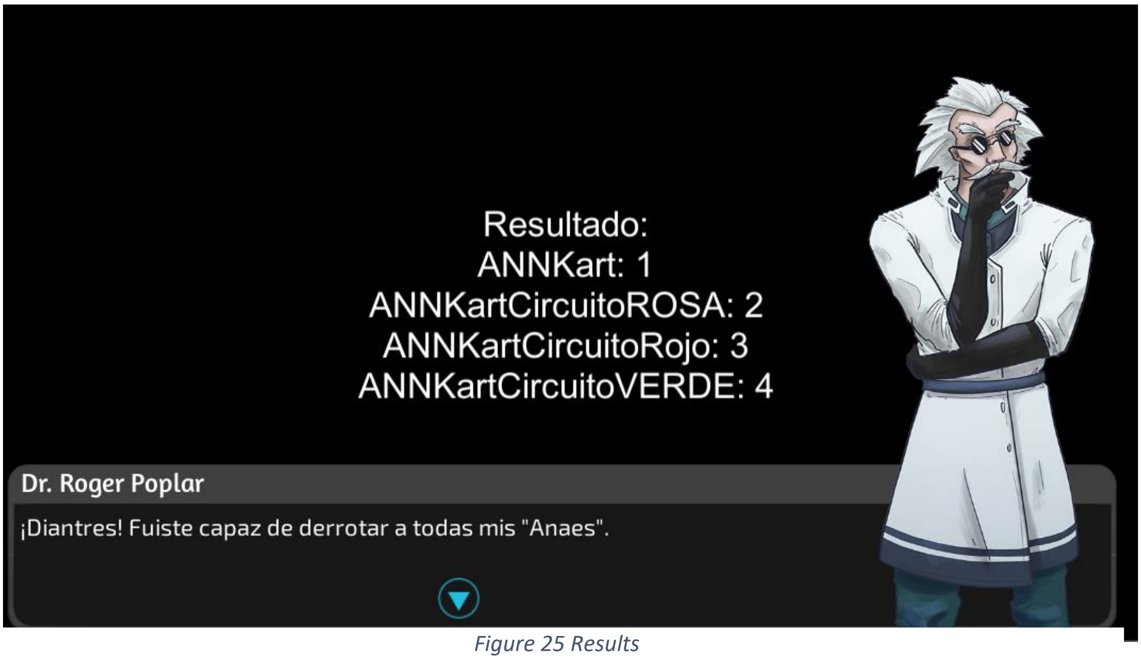
Figure 23 Circuit 4 Pink Bot

Now all that is left is to let them train until they learn the circuit, creating text files with their ID and weights, ending their training.

We now have to program the game logic. To start, we add a countdown using the Canvas where the Brain.cs and Player Movement are deactivated. When the countdown reaches zero and GO appears, the aforementioned Scripts are activated and the race begins since each of the bots will try to resolve the circuit on their own, with the weights previously saved during training.

The player, in turn, must try to reach the Finish Line before the bots do.

When all the participants in the race reach the Finish Line, the image fades to black with the results of the arrival positions and a commentary from Dr. Poplar, depending on if you were able to defeat his bots or if they defeated the player.



Resultado:
ANNKart: 1
ANNKartCircuitoROSA: 2
ANNKartCircuitoRojo: 3
ANNKartCircuitoVERDE: 4

Dr. Roger Poplar

¡Diantres! Fuiste capaz de derrotar a todas mis "Anaes".

Figure 25 Results

Once this commentary ends, the player will be sent to the main menu. Then we would add the pause menu and regard the logic of this game mode as finished.

Trainning Mode

The concept of the Training Mode was to gradually extract the paths in the same grid as the one in Draw your Circuit, and this was to be used to have more circuits saved, thereby increasing the difficulty of the Draw your Circuit mode.

The bot would be instantiated looking towards the first cell of the circuit and would go through its training process until reaching the Finish Line, where it would be deinstantiated, another random circuit not stored in the Circuits.txt would be loaded, and the bot would be instantiated again generating an automatic process, without the player needing to intervene.

However it has the same options as in the Draw your Circuit mode, moving the zenithal camera, zoom, changing cameras, and even pausing the game.

The problem appears when trying to create an algorithm which generates random circuits *in situ*. The random circuits are loaded, the bot learns the, and the algorithm is run again in a reasonable time, since it would be a process requiring loading screens between the different stages.

After several days of trying to extract the algorithm, one idea is ruled out for another idea, which is to deploy all the possible paths that can be drawn in Draw your Circuit, and these are stored in a .txt. Each of the circuits is stored in each line, and with this file, Unity simply loads them later, line by line, giving rise to the SacaCaminos algorithm.

“SacaCaminos” Algorithm

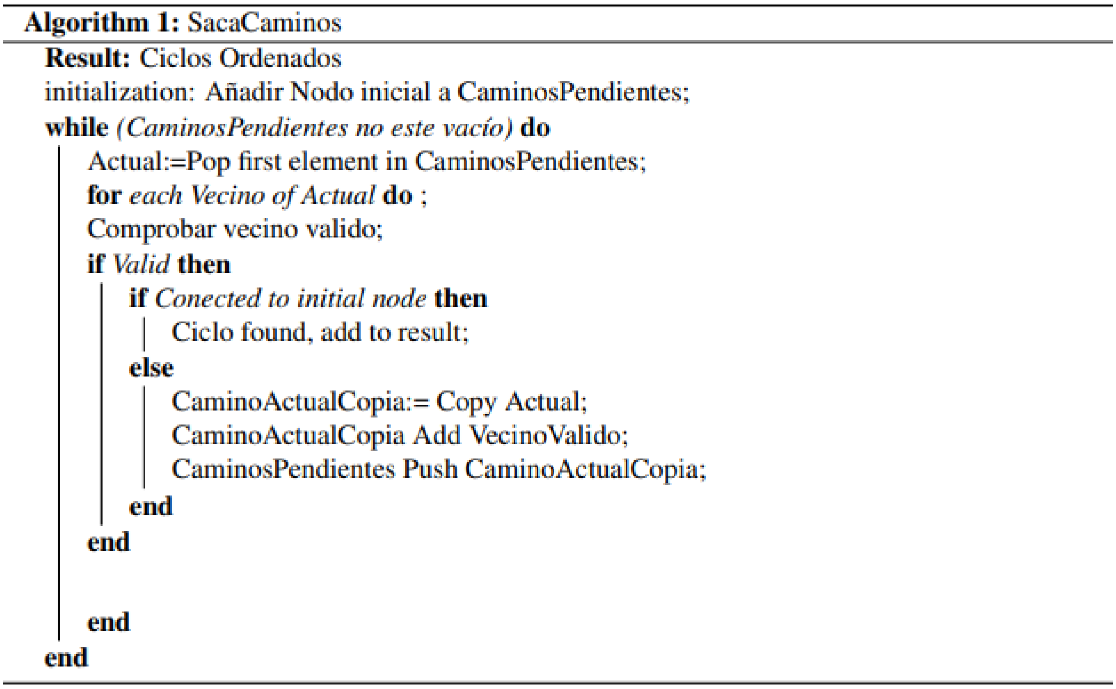
We will resolve the given problem, an initial cell, and we want to extract all the possible paths in a hexagonal grid that depart from and end in said initial cell, with the restrictions mentioned in the Draw your circuit section, that is, paths without forks and having only two neighbors at most.

This problem will be solved using an approach with graphs. We will interpret the grid as a graph with four hundred nodes interconnected with their neighbors by means of the restrictions.

Translated into graphs, the preceding problem consists of finding all the simple cycles (a series of vertexes such that from each of these vertexes there is an edge leading to the next vertex, where a vertex is repeated only once, appearing twice as the beginning and the end of the cycle) by using or taking into account the initial cell of the grid with id 63 (Cell [3][3] start and finish Cell) as the initial and final node and the condition that a vertex may only be connected to two vertexes at most.

Given that we want to find all the cycles, a brute force algorithm is used. Here, the run time is exponential because each node has 6 neighbors and we are going to scan all of them, and then the neighbors of those 6 neighbors, and so on and so forth. This solves our problem, but the time is too long.

The algorithm proceeds as follows: there is a list of paths in which we save the initial node and the last one added to the list, such that for each path we scan the vertexes neighbors of the last node added (corresponding to neighboring hexagons of the grid). With each of these neighbors, we will check to see if the node is valid for being added to the path, i.e., it is checked in order to determine that this node is not a neighbor of any of the nodes already present on the path. To do this, we go through the neighbors of this node and check to see if they are no longer on the path. If the neighbor is valid, there are two possibilities. In the first, it checks to see if the neighbor is connected with the initial vertex, in which case we have found a valid path because the neighbor can be added to the previous path and is connected with the initial node, thereby finding a cycle like the one described above. In the second possibility, in the event that the neighbor is not connected, a new path variable is created and contains the previous path with this neighbor as the last one added to the path. This new path is then left to be later processed. The algorithm is shown below in pseudocode.



```
Algorithm 1: SacaCaminos
  Result: Ciclos Ordenados
  initialization: Añadir Nodo inicial a CaminosPendientes;
  while (CaminosPendientes no este vacío) do
      Actual:=Pop first element in CaminosPendientes;
      for each Vecino of Actual do ;
      Comprobar vecino valido;
      if Valid then
          if Conected to initial node then
              Ciclo found, add to result;
          else
              CaminoActualCopia:= Copy Actual;
              CaminoActualCopia Add VecinoValido;
              CaminosPendientes Push CaminoActualCopia;
          end
      end

      end
  end
```

Figure 26 Pseudocode

A class called “Camino” is therefore defined, and it will represent the paths. This class has a final integer which represents the last one added to the path and a list of integers which represents the remaining nodes of the path in the order in which they are passed.

After this first approach of the algorithm, it could be seen that it was inviable to run said algorithm in memory due to the large number of paths that were accumulated in the stack to be visited, overrunning the internal memory internal, generating a fault.

To correct this error, files instead of the internal memory are used as a stack, such that functions are implemented that allow reading and writing the paths in texts files.

After this solution, the algorithm is run with complete normalcy, but the file used as a stack can become quite large, slowing down input and output operations (because to read one row and then delete it, the entire file must be loaded and re-written without that line). To solve this, a maximum stack size of 5 mb is established, such that when the stack file is about to exceed this size, a new file will be created and it will become the current stack until it is emptied. For this method to work properly, functions which allow checking to see if such file already exists and the size of that file are implemented.

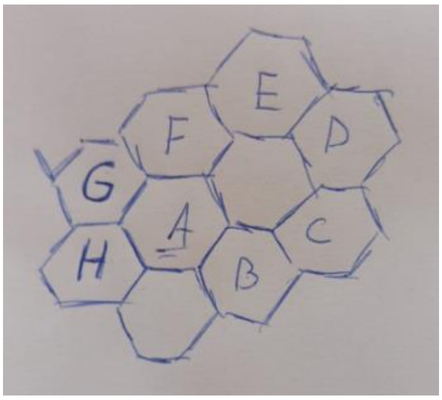
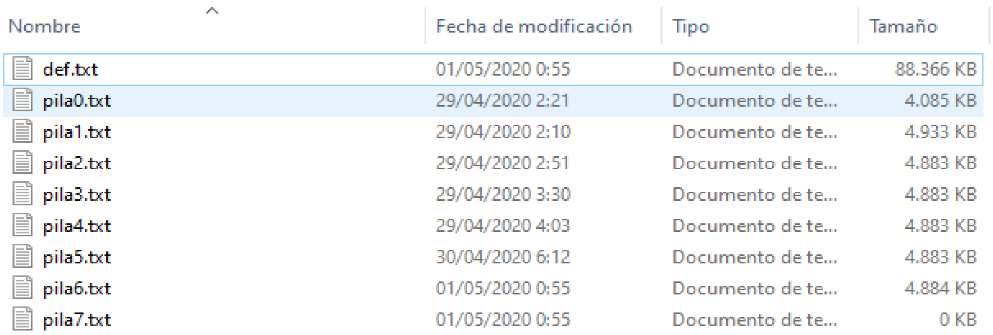


Figure 27 Duplicate problem Sample

After the latest modifications to the algorithm, we are faced with another problem: the paths duplicate. Since only the initial node is indicated, the valid paths appeared in both directions as being the same one, that is, path ABCDEF is the same as AFEDCB (Fig. 17), but the algorithm detected them as different paths.

To solve it instead of indicating a single starting vertex, starting triplets are indicated consisting of an input node, an intermediate node and an output node. The input node correspondence to the initial of the road structure, the exit to the last aggregate and the intermediate will be in the accumulated vector that corresponds to the initial node of the grid, 63.

Therefore, the nodes will always be output nodes and others will be input nodes when both are present, thereby preventing the duplication of paths.
For the preceding example in Fig. 17 the FAB triad would be added, so only the ABCDF path would result and not the inverse because B is for output and F for input.



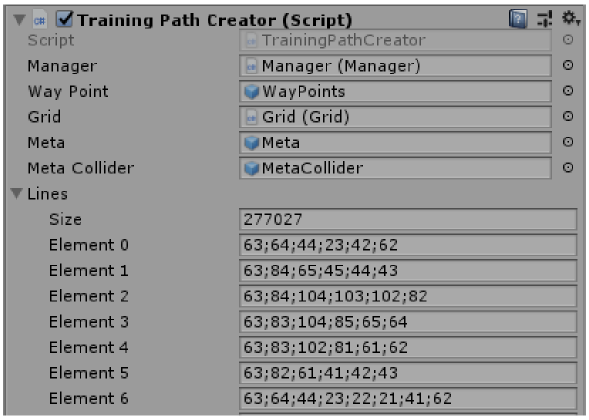
Nombre	Fecha de modificación	Tipo	Tamaño
def.txt	01/05/2020 0:55	Documento de te...	88.366 KB
pila0.txt	29/04/2020 2:21	Documento de te...	4.085 KB
pila1.txt	29/04/2020 2:10	Documento de te...	4.933 KB
pila2.txt	29/04/2020 2:51	Documento de te...	4.883 KB
pila3.txt	29/04/2020 3:30	Documento de te...	4.883 KB
pila4.txt	29/04/2020 4:03	Documento de te...	4.883 KB
pila5.txt	30/04/2020 6:12	Documento de te...	4.883 KB
pila6.txt	01/05/2020 0:55	Documento de te...	4.884 KB
pila7.txt	01/05/2020 0:55	Documento de te...	0 KB

Figure 28 Paths and piles

After leaving the algorithm running for more than four days, the def.txt file is obtained, where, as can be seen in Fig. 28, there are still seven stacks that have been created. This means that not all the possible paths have been extracted yet.

However, taking into account the run time and the fact that more than 270,000 paths have been extracted and that the mean learning of the ANN is between 50 and 300 attempts to learn a path, learning these 270,000 paths would take considerable time. It can therefore be left like that and we can start with the Unity logic.

The logic can be recycled from the Draw your Circuit Grid.cs, as occurs with the cameras and managers, but in this case the user is deprived of the drawing interaction and we create the Training Path Creator.cs Script from where we can read the def.txt file imported into Unity.



☑ Training Path Creator (Script)

Script	TrainingPathCreator
Manager	Manager (Manager)
Way Point	WayPoints
Grid	Grid (Grid)
Meta	Meta
Meta Collider	MetaCollider
Lines	
Size	277027
Element 0	63;64;44;23;42;62
Element 1	63;84;65;45;44;43
Element 2	63;84;104;103;102;82
Element 3	63;83;104;85;65;64
Element 4	63;83;102;81;61;62
Element 5	63;82;61;41;42;43
Element 6	63;64;44;23;22;21;41;62

Figure 29 Training Path Creator.cs

Besides reading the file line por line, it remembers the line where it left off so that the game can be closed and it will open again where it left off.

In this Script we insert both the WayPoints every three cells and the Finish Line, as the last element of each line, and we achieve the desired result of, once the bot reaches the Finish Line, the weights and the circuit being saved in Circuit.txt, the bot is deinstantiated, the next line of the def.txt is loaded, and the bot in the training process is instantiated again. All this is done in time real and without any loading screen, ultimately achieving the result that was desired from the beginning, but doing so in a different way.

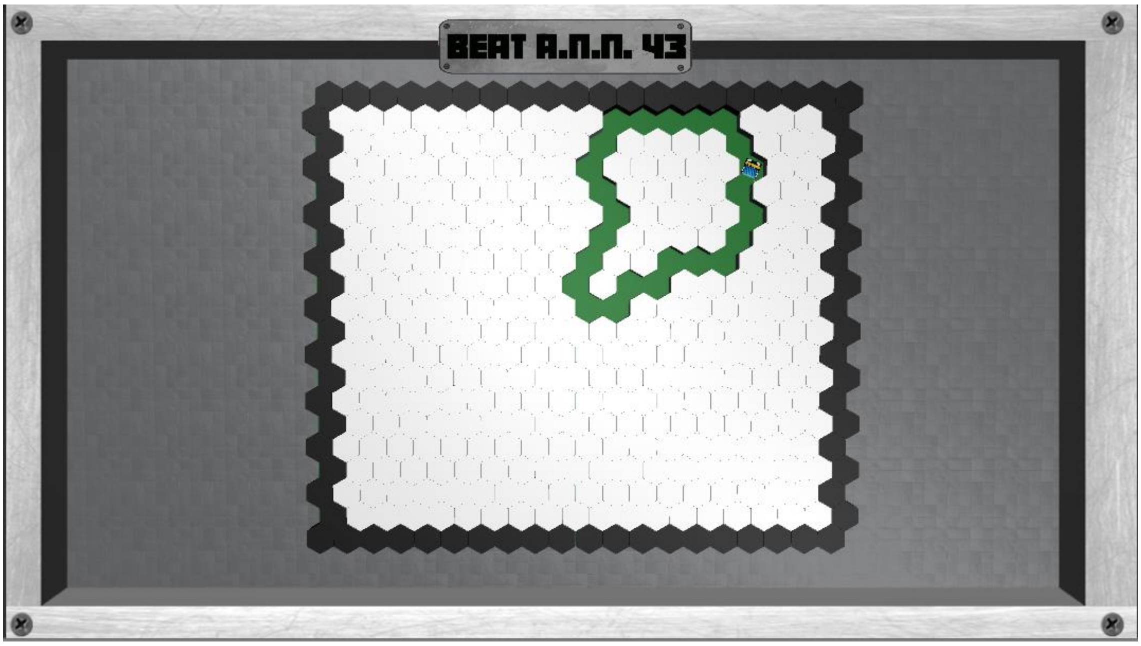
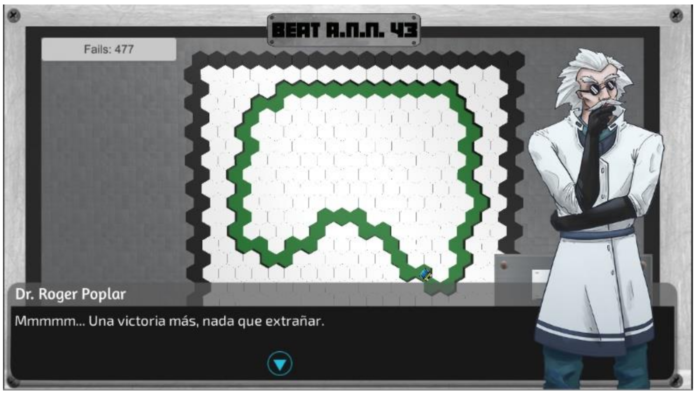


Figure 30 Training Mode

Validation

This ANN when starting with random weights leaves similar results when learning the circuits. A complicated circuit usually takes between 300 and 1000 attempts but as you can see sometimes the circuit can be done on the third (Fig. 31 and Fig 32 compare fails).

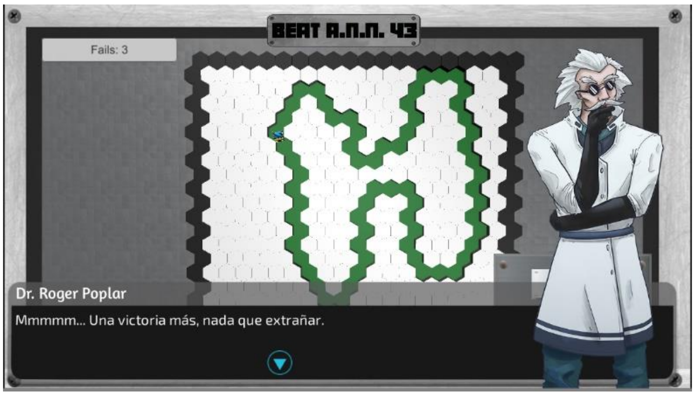


Fails: 477

Dr. Roger Poplar

Mmmmm... Una victoria más, nada que extrañar.

Figure 31 Test 1



Fails: 3

Dr. Roger Poplar

Mmmmm... Una victoria más, nada que extrañar.

Figure 32 Test 2

Results

- A functional ANN has been created.
- Different game modes have been created for interacting with said ANN, together with Dr. Poplar's commentaries; there is also a narrative interconnection between the different game modes.
- Several scenarios with different characterizations have been achieved using flat colors.
- An automatic training mode has been developed using Reinforcement Learning.

Project Access

GitHub - https://github.com/sevalper/Beat-A.N.N.-43.git

YouTube – https://youtu.be/ZDibuewxy88

Conclusions

"Beat A.N.N. 43" has been, without question, the most important and complicated project I have worked on in these years of study. It has been an enormous challenge to learn so much from scratch to create an artificial intelligence that was able to satiate my curiosity. I am very satisfied with the obtained result because my knowledge about artificial intelligence was very sparse at the beginning of this project, and I was able to combine said artificial intelligence with a game with which one can interact.

I believe that ANNs can still be further developed in the world of video games in order to create much more dynamic and different experiences.

Future work

In the future I would like to work on new, more ambitious projects involving ANNs so as to apply them to video games and other possible project or challenges left for me to discover.

Audio

Having the "Options" menu option opens a small window with a slider to select the volume of all the scenes in the game.

For this we have the codes of "SaveAndLoad.cs" to save the volume indicated in the slicer and the "VolumeController.cs" that each of the AudioSouces has to broadcast the different scenes of the game without copyright and with this "VolumeController.cs" we load the float of the volume chosen in the main menu, maintaining in all the scenes the volume selected in the main menu, set to 0.1/1 of base.

Bibliography

- Unity Technologies. [Online]. Available: https://unity.com/es

- Unity Learn Tutorials. [Online]. Available: https://unity3d.com/es/learn/tutorials.

- Unity User Manual. [Online]. Available: https://docs.unity3d.com/es/2018.3/Manual/UnityManual.html

- TextMesh Pro. [Online]. Available: https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126

- GitHub. [Online]. Available: https://github.com/.

- Reddit, Unity3D. [Online]. Available: https://www.reddit.com/r/Unity3D/.

- A Beginner's Guide To Machine Learning with Unity. [Online]. Available: https://www.udemy.com/course/machine-learning-with-unity/

- Reinforcement Learning: AI Flight with Unity ML-Agents. [Online]. Available: https://www.udemy.com/course/ai-flight/

- Book: Artificial Intelligence For Games. - by: Ian Millington & John Funge

- Fungus. [Online]. Available: https://assetstore.unity.com/packages/templates/systems/fungus-34184

- Karting Microgame. [Online]. Available: https://assetstore.unity.com/packages/templates/karting-microgame-150956

Appendix – Additional documentation

“Neuron.cs”

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Neuron {

        public int numInputs;
        public double bias;
        public double output;
        public double errorGradient;
        public List<double> weights = new List<double>();
        public List<double> inputs = new List<double>();

        public Neuron(int nInputs)
        {
                float weightRange = (float) 2.4/(float) nInputs;
                bias = UnityEngine.Random.Range(-weightRange,weightRange);
                numInputs = nInputs;

                for(int i = 0; i < nInputs; i++)
                        weights.Add(UnityEngine.Random.Range(-weightRange,weightRange));
        }
}
```

“Layer.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Layer {

        public int numNeurons;
        public List<Neuron> neurons = new List<Neuron>();

        public Layer(int nNeurons, int numNeuronInputs)
        {
                numNeurons = nNeurons;
                for(int i = 0; i < nNeurons; i++)
                {
                        neurons.Add(new Neuron(numNeuronInputs));
                }
        }
}
```

“A.N.N.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ANN{

        public int numInputs;
        public int numOutputs;
        public int numHidden;
        public int numNPerHidden;
        public double alpha;
        List<Layer> layers = new List<Layer>();

        public ANN(int nI, int nO, int nH, int nPH, double a)
        {
                numInputs = nI;
                numOutputs = nO;
                numHidden = nH;
                numNPerHidden = nPH;
                alpha = a;

                if(numHidden > 0)
                {
                        layers.Add(new Layer(numNPerHidden, numInputs));

                        for(int i = 0; i < numHidden-1; i++)
                        {
                                layers.Add(new Layer(numNPerHidden, numNPerHidden));
                        }

                        layers.Add(new Layer(numOutputs, numNPerHidden));
                }
                else
                {
                        layers.Add(new Layer(numOutputs, numInputs));
                }
        }

        public List<double> Train(List<double> inputValues, List<double> desiredOutput)
        {
                List<double> outputValues = new List<double>();
                outputValues = CalcOutput(inputValues, desiredOutput);
                UpdateWeights(outputValues, desiredOutput);
                return outputValues;
        }

        public List<double> CalcOutput(List<double> inputValues, List<double> desiredOutput)
        {
                List<double> inputs = new List<double>();
                List<double> outputValues = new List<double>();
                int currentInput = 0;

                if(inputValues.Count != numInputs)
                {
                        Debug.Log("ERROR: Number of Inputs must be " + numInputs);
                        return outputValues;
                }

                inputs = new List<double>(inputValues);
                for(int i = 0; i < numHidden + 1; i++)
                {
                                if(i > 0)
                                {
                                        inputs = new List<double>(outputValues);
                                }
                                outputValues.Clear();

                                for(int j = 0; j < layers[i].numNeurons; j++)
                                {
                                        double N = 0;
                                        layers[i].neurons[j].inputs.Clear();

                                        for(int k = 0; k < layers[i].neurons[j].numInputs; k++)
                                        {
                                            layers[i].neurons[j].inputs.Add(inputs[currentInput]);
                                                N += layers[i].neurons[j].weights[k] * inputs[currentInput];
                                                currentInput++;
                                        }

                                        N -= layers[i].neurons[j].bias;

                                        if(i == numHidden)
                                                layers[i].neurons[j].output = ActivationFunctionO(N);
                                        else
                                                layers[i].neurons[j].output = ActivationFunction(N);

                                        outputValues.Add(layers[i].neurons[j].output);
                                        currentInput = 0;
                                }
                }
                return outputValues;
        }

        public List<double> CalcOutput(List<double> inputValues)
        {
                List<double> inputs = new List<double>();
                List<double> outputValues = new List<double>();
                int currentInput = 0;

                if(inputValues.Count != numInputs)
                {
                        Debug.Log("ERROR: Number of Inputs must be " + numInputs);
                        return outputValues;
                }

                inputs = new List<double>(inputValues);
                for(int i = 0; i < numHidden + 1; i++)
                {
                                if(i > 0)
                                {
                                        inputs = new List<double>(outputValues);
                                }
                                outputValues.Clear();

                                for(int j = 0; j < layers[i].numNeurons; j++)
                                {
                                        double N = 0;
                                        layers[i].neurons[j].inputs.Clear();

                                        for(int k = 0; k < layers[i].neurons[j].numInputs; k++)
                                        {
                                            layers[i].neurons[j].inputs.Add(inputs[currentInput]);
                                                N += layers[i].neurons[j].weights[k] * inputs[currentInput];
                                currentInput++;
                            }

                            N -= layers[i].neurons[j].bias;

                            if(i == numHidden)
                                layers[i].neurons[j].output = ActivationFunctionO(N);
                            else
                                layers[i].neurons[j].output = ActivationFunction(N);

                            outputValues.Add(layers[i].neurons[j].output);
                            currentInput = 0;
                        }
                }
                return outputValues;
            }


            public string PrintWeights()
            {
                string weightStr = "";
                foreach(Layer l in layers)
                {
                    foreach(Neuron n in l.neurons)
                    {
                        foreach(double w in n.weights)
                        {
                            weightStr += w + ";";
                        }
                        weightStr += n.bias + ";";
                    }
                }
                return weightStr;
            }

            public void LoadWeights(string weightStr)
            {
                if(weightStr == "") return;
                string[] weightValues = weightStr.Split(';');
                int w = 0;
                foreach(Layer l in layers)
                {
                    foreach(Neuron n in l.neurons)
                    {
                        for(int i = 0; i < n.weights.Count; i++)
                        {
                            n.weights[i] = System.Convert.ToDouble(weightValues[w]);
                            w++;
                        }
                        n.bias = System.Convert.ToDouble(weightValues[w]);
                        w++;
                    }
                }
            Debug.Log("LoadWeights: " + weightStr);
            }

            void UpdateWeights(List<double> outputs, List<double> desiredOutput)
            {
                double error;
                for(int i = numHidden; i >= 0; i--)
                {
                    for(int j = 0; j < layers[i].numNeurons; j++)
                    {
                        if(i == numHidden)
                        {
                            error = desiredOutput[j] - outputs[j];
                            layers[i].neurons[j].errorGradient = outputs[j] * (1-outputs[j]) * error;
                        }
                        else
                        {
                            layers[i].neurons[j].errorGradient = layers[i].neurons[j].output * (1-layers[i].neurons[j].output);
                            double errorGradSum = 0;
                            for(int p = 0; p < layers[i+1].numNeurons; p++)
                            {
                                errorGradSum += layers[i+1].neurons[p].errorGradient * layers[i+1].neurons[p].weights[j];
                            }
                            layers[i].neurons[j].errorGradient *= errorGradSum;
                        }
                        for(int k = 0; k < layers[i].neurons[j].numInputs; k++)
                        {
                            if(i == numHidden)
                            {
                                error = desiredOutput[j] - outputs[j];
                                layers[i].neurons[j].weights[k] += alpha * layers[i].neurons[j].inputs[k] * error;
                            }
                            else
                            {
                                layers[i].neurons[j].weights[k] += alpha * layers[i].neurons[j].inputs[k] * layers[i].neurons[j].errorGradient;
                            }
                        }
                        layers[i].neurons[j].bias += alpha * -1 * layers[i].neurons[j].errorGradient;
                    }

                }

            }


            double ActivationFunction(double value)
            {
                return TanH(value);

            }

            double ActivationFunctionO(double value)
            {

                return Sigmoid(value);
            }

            double TanH(double value)
            {
                double k = (double) System.Math.Exp(-2*value);
            return 2 / (1.0f + k) - 1;
            }

            double ReLu(double value)
            {
                if(value > 0) return value;
                else return 0;
            }

            double Linear(double value)
            {
                return value;
            }

            double LeakyReLu(double value)
            {
                if(value < 0) return 0.01*value;
                else return value;
            }

            double Sigmoid(double value)
            {
            double k = (double) System.Math.Exp(value);
            return k / (1.0f + k);
            }
        }
```

“BallState.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```
public class BallState : MonoBehaviour {

    public bool dropped = false;
    public bool point = false;
    public bool meta = false;

        void OnCollisionEnter(Collision col)
        {
                if(col.gameObject.tag == "terrain")
                {
                        dropped = true;

                }



    }








    private void OnTriggerEnter(Collider col)
    {
        if (col.gameObject.tag == "point")
        {
            ID id = col.gameObject.GetComponent<ID>();
            if (id != null && id.tipo == tipoCheckpoint.Meta) { meta = true; }
            point = true;


        }


    }

    private void OnTriggerExit(Collider col)
    {
        if (col.gameObject.tag == "point")
        {
            point = false;
        }
    }
}
```

“Brain.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
using System;
using Fungus;
using UnityEngine.UI;

public class Replay
{

    public List<double> states;
    public double reward;

        public Replay(float fDist, float rDist, float lDist, float r45Dist, float l45Dist, double r)
        {
                states = new List<double>();

        //fDist = 200;
        //rDist = 200;
        //lDist = 200;
        //r45Dist = 200;
        //l45Dist = 200;

        states.Add(fDist);
        states.Add(rDist);
        states.Add(lDist);
        states.Add(r45Dist);
        states.Add(l45Dist);


        reward = r;

        }
}


public class Brain : MonoBehaviour {

    private Flowchart flowchart;
    [SerializeField] private string circuitName = "Circuito1";
    private bool WIN = false;

    public string currentAleatoryCircuitName;
    public List<float> pointsCircuitName;


    public static Action<int> onFail;
    public bool _isAleatoryCircuit;
    public Text failText;
    public static event Action Death;

    public float speed = 50.0f;
    public float rotationSpeed = 500.0f;
    public float visibleDistance = 200.0f;
    public LayerMask terrainLayer;

    public GameObject ball;                          //object to monitor
    private Manager manager;
    private ManagerCircuits managerCircuits;


        ANN ann;

        float reward = 0.0f;                                                     //reward to associate with actions
        List<Replay> replayMemory = new List<Replay>(); //memory - list of past actions and rewards
        int mCapacity = 10000;                                                   //memory capacity

        float discount = 0.99f;                                                  //how much future states affect rewards
        float exploreRate = 100.0f;                                              //chance of picking random action
        float maxExploreRate = 100.0f;                                   //max chance value
    float minExploreRate = 0.01f;                                        //min chance value
    float exploreDecay = 0.0001f;                                        //chance decay amount for each update

        Vector3 ballStartPos;                                                    //record start position of object
    Quaternion ballStartRot;
    int failCount = 0;                                                       //count when the ball is dropped
        public float tiltSpeed = 0.5f;                                               //max angle to apply to tilting each update
            //make sure this is large enough so that the q value
            //multiplied by it is enough to recover balance
            //when the ball gets a good speed up
        float timer = 0;                                                                 //timer to keep track of balancing
        float maxBalanceTime = 0;                        //record time ball is kept balanced
                                                 // Use this for initialization
    private string weightsString;

    public float timeScaleValue = 25.0f;


    private void Awake()
    {
        manager = FindObjectOfType<Manager>();
        managerCircuits = FindObjectOfType<ManagerCircuits>();
        flowchart = FindObjectOfType<Flowchart>();
        if (_isAleatoryCircuit) failText = manager.getFailText();
    }

    public void Start () {
                ann = new ANN(5,2,1,10,0.5f);

        var aux = PlayerPrefs.GetString("Weights");
        //if (aux != null)
        //ann.LoadWeights(aux);
        if (_isAleatoryCircuit)
        {
            if (manager.GetDictionaryCreate() && manager.GetNameDictionary(currentAleatoryCircuitName))
            {
                Debug.Log(currentAleatoryCircuitName);
                Debug.Log(manager.GetDataDictionary(currentAleatoryCircuitName));
                ann.LoadWeights(manager.GetDataDictionary(currentAleatoryCircuitName));
            }

            /*
            if (SaveAndLoad.SaveExists(currentAleatoryCircuitName+".txt"))
            {
                Debug.Log("Cargar");
                List<string> loadContent = new List<string>();
                loadContent = SaveAndLoad.Load<List<string>>(currentAleatoryCircuitName+".txt");
                Debug.Log(loadContent[0] + "\n" + loadContent[1]);
                ann.LoadWeights(loadContent[1]);
            }*/
        }
        else
        {
            if (managerCircuits.GetDictionaryCreate() && managerCircuits.GetNameDictionary(circuitName))
            {
                Debug.Log(managerCircuits.GetDataDictionary(circuitName));
                ann.LoadWeights(managerCircuits.GetDataDictionary(circuitName));
            }
        }

        ballStartPos = ball.transform.position;
        ballStartRot = ball.transform.rotation;
        Time.timeScale = timeScaleValue;        //
    }

        GUIStyle guiStyle = new GUIStyle();
        void OnGUI()
        {
        /*

                guiStyle.fontSize = 25;
                guiStyle.normal.textColor = Color.white;
                GUI.BeginGroup (new Rect (10, 10, 600, 150));
                GUI.Box (new Rect (0,0,140,140), "Stats", guiStyle);
                GUI.Label(new Rect (10,25,500,30), "Fails: " + failCount, guiStyle);
                GUI.Label(new Rect (10,50,500,30), "Decay Rate: " + exploreRate, guiStyle);
                GUI.Label(new Rect (10,75,500,30), "Last Best Balance: " + maxBalanceTime, guiStyle);
                GUI.Label(new Rect (10,100,500,30), "This Balance: " + timer, guiStyle);
        GUI.Label(new Rect(10, 125, 500, 30), "Reward: " + reward, guiStyle);
        GUI.EndGroup ();*/
        }

        // Update is called once per frame
        void Update () {
        if (Input.GetKeyDown("space"))
        {
            maxBalanceTime = 0;
            ResetBall();
        }

        if (Input.GetKeyDown("j"))
        {

            Debug.Log(ann.PrintWeights());
        }


        Debug.DrawRay(transform.position, this.transform.forward * visibleDistance, Color.red);
        Debug.DrawRay(transform.position, this.transform.right * visibleDistance, Color.red);
        Debug.DrawRay(transform.position, -this.transform.right * visibleDistance, Color.red);

        Debug.DrawRay(transform.position, (Quaternion.AngleAxis(-45, Vector3.up) * this.transform.right) * visibleDistance, Color.red);
        Debug.DrawRay(transform.position, (Quaternion.AngleAxis(45, Vector3.up) * -this.transform.right) * visibleDistance, Color.red);

        weightsString =  ann.PrintWeights();


            //

    }

    private void OnApplicationQuit()
    {
        PlayerPrefs.SetString("Weights", weightsString);
    }

    void FixedUpdate () {
                timer += Time.deltaTime;
                List<double> states = new List<double>();
                List<double> qs = new List<double>();

        RaycastHit hit;

        float fDist = visibleDistance, rDist = visibleDistance, lDist = visibleDistance, r45Dist = visibleDistance, l45Dist = visibleDistance;



        if (Physics.Raycast(transform.position, this.transform.forward, out hit, visibleDistance, terrainLayer))
        {
            fDist = Vector3.Distance(transform.position, hit.point);

        }

        if (Physics.Raycast(transform.position, this.transform.right, out hit, visibleDistance, terrainLayer))
        {
            rDist = Vector3.Distance(transform.position, hit.point);

        }

        if (Physics.Raycast(transform.position, -this.transform.right, out hit, visibleDistance, terrainLayer))
        {
            lDist = Vector3.Distance(transform.position, hit.point);
        }

        if (Physics.Raycast(transform.position, Quaternion.AngleAxis(-45, Vector3.up) * this.transform.right, out hit, visibleDistance, terrainLayer))
        {
            r45Dist = Vector3.Distance(transform.position, hit.point);
        }

        if (Physics.Raycast(transform.position, Quaternion.AngleAxis(45, Vector3.up) * -this.transform.right, out hit, visibleDistance, terrainLayer))
        {
            l45Dist = hit.distance;
        }

       // Debug.Log("Frontal: " + fDist + ", Derecha: " + rDist + ", Izquierda: " + lDist + ", Derecha45: " + r45Dist + ", Izquierda45: " + l45Dist);

        states.Add(fDist);
        states.Add(rDist);
        states.Add(lDist);
        states.Add(r45Dist);
        states.Add(l45Dist);

        qs = SoftMax(ann.CalcOutput(states));
                double maxQ = qs.Max();
                int maxQIndex = qs.ToList().IndexOf(maxQ);

                //exploreRate = Mathf.Clamp(exploreRate - exploreDecay, minExploreRate, maxExploreRate);


       //if(Random.Range(0,100) < exploreRate)
        //      maxQIndex = Random.Range(0,2);


        float translation = speed * Time.deltaTime;

        this.transform.Translate(0, 0, translation);








        if (maxQIndex == 0)
                        this.transform.Rotate(Vector3.up,tiltSpeed * (float)qs[maxQIndex]);
                else if (maxQIndex == 1)
                        this.transform.Rotate(Vector3.up, -tiltSpeed * (float)qs[maxQIndex]);


        if (ball.GetComponent<BallState>().dropped)
        {
            reward = -1.0f;
            //reward = 0;
        }

        else if (ball.GetComponent<BallState>().point)
        {
            reward = 0.5f;

        }
        else
            reward = 0.1f;// + 0.01f;



        Replay lastMemory = new Replay(fDist,rDist,lDist,r45Dist,l45Dist,
                                                                reward);

                if(replayMemory.Count > mCapacity)
                        replayMemory.RemoveAt(0);

                replayMemory.Add(lastMemory);

                if(ball.GetComponent<BallState>().dropped)
                {
            ResetBall();    //Para que no se quede pillado al no tener archivo.

                        for(int i = replayMemory.Count - 1; i >= 0; i--)
                        {
                                List<double> toutputsOld = new List<double>();
                                List<double> toutputsNew = new List<double>();
                                toutputsOld = SoftMax(ann.CalcOutput(replayMemory[i].states));

                                double maxQOld = toutputsOld.Max();
                                int action = toutputsOld.ToList().IndexOf(maxQOld);

                            double feedback;
                                if(i == replayMemory.Count-1 || replayMemory[i].reward == -1)
                                        feedback = replayMemory[i].reward;
                                else
                                {
                                        toutputsNew = SoftMax(ann.CalcOutput(replayMemory[i+1].states));
                                        maxQ = toutputsNew.Max();
                                        feedback = (replayMemory[i].reward +
                                                discount * maxQ);
                                }

                                toutputsOld[action] = feedback;
                                ann.Train(replayMemory[i].states,toutputsOld);
                        }



                        timer = 0;

                        ball.GetComponent<BallState>().dropped = false;
                        this.transform.rotation = Quaternion.identity;
                        ResetBall();
                        replayMemory.Clear();
                        failCount++;
            if (_isAleatoryCircuit)
            {
                if (failCount == 1000)
                {
                    flowchart.ExecuteBlock("LOSE");
                }
                //Debug.Log( "Fails: " + failCount);
                onFail?.Invoke(failCount);

            }

            //reward = 0;////////////////////////////////
                }

        if (ball.GetComponent<BallState>().meta)
        {

            ball.GetComponent<BallState>().meta = false;
            //string pesos = PlayerPrefs.GetString("Weights");
            string pesos = ann.PrintWeights();



            if (_isAleatoryCircuit)
            {
                if (!WIN && failCount <= 1000)
                {
                    if(flowchart!=null)
                        flowchart.ExecuteBlock("WIN");
                    WIN = true;
                }

                /*List<string> saveFileContent = new List<string>();
                saveFileContent.Add(currentAleatoryCircuitName);
                saveFileContent.Add(pesos);
                SaveAndLoad.Save(saveFileContent, currentAleatoryCircuitName + ".txt");*/

                manager.SaveNewDataDictionary(currentAleatoryCircuitName, pesos);
                Debug.Log(currentAleatoryCircuitName);
            }
            else
            {
                managerCircuits.SaveNewDataDictionary(circuitName, pesos);
            }
            /*
            if (!_isAleatoryCircuit && maxBalanceTime <= 0)
            {
                pesos = ann.PrintWeights();
                SaveAndLoad.Save(pesos, CIRCUITO1);
            }
            else if(!_isAleatoryCircuit && maxBalanceTime > timer)
            {
                pesos = ann.PrintWeights();
                SaveAndLoad.Save(pesos, CIRCUITO1);
            }*/
            maxBalanceTime = timer;
            Debug.Log(maxBalanceTime);
            timer = 0;

        }



    }

        void ResetBall()
        {
        /*
        Death();
        Destroy(this.gameObject);//QUITAR ESTO
        */
        /*      Esto se puede utilizar cuando funcione perfectamente el cargado
         * if (manager.GetDictionaryCreate() && manager.GetNameDictionary(currentAleatoryCircuitName))
        {
            Debug.Log(manager.GetDataDictionary(currentAleatoryCircuitName));
            ann.LoadWeights(manager.GetDataDictionary(currentAleatoryCircuitName));
        }*/

        ball.transform.position = ballStartPos;
        ball.transform.rotation = ballStartRot;

                //ball.GetComponent<Rigidbody>().velocity = new Vector3(0,0,0);
                //ball.GetComponent<Rigidbody>().angularVelocity = new Vector3(0,0,0);
        }

        List<double> SoftMax(List<double> values)
    {
      double max = values.Max();

      float scale = 0.0f;
      for (int i = 0; i < values.Count; ++i)
        scale += Mathf.Exp((float)(values[i] - max));

      List<double> result = new List<double>();
      for (int i = 0; i < values.Count; ++i)
        result.Add(Mathf.Exp((float)(values[i] - max)) / scale);

      return result;
    }
}
```

“ID.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public enum tipoCheckpoint
{
    Meta,
}

public class ID : MonoBehaviour
{
    public tipoCheckpoint tipo;
}
```

“SaveAndLoad.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

public class SaveAndLoad : MonoBehaviour
{
    public static void Save<T>(T objectToSave, string key)
    {
        string path = Application.persistentDataPath + "/save/";
        Directory.CreateDirectory(path);
        BinaryFormatter formatter = new BinaryFormatter();
        using (FileStream fileStream = new FileStream(path + key, FileMode.Create))
        {
            formatter.Serialize(fileStream, objectToSave);
        }
    }

    public static T Load<T>(string key)
    {
        string path = Application.persistentDataPath + "/save/";
        BinaryFormatter formatter = new BinaryFormatter();
        T returnValue = default(T);
        using (FileStream fileStream = new FileStream(path + key, FileMode.Open))
        {
            returnValue = (T)formatter.Deserialize(fileStream);
        }

        return returnValue;
    }


    public static bool SaveExists(string key)
    {
        string path = Application.persistentDataPath + "/save/" + key;
        return File.Exists(path);
    }

    public static void DeleteAllSaveFiles()
    {
        string path = Application.persistentDataPath + "/save/";
        DirectoryInfo directory = new DirectoryInfo(path);
        directory.Delete();
    }

}
```

“AlgoritmoCaminoInf.cs”

```
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Linq;


public class AlgoritmoCaminosInf : MonoBehaviour
{
    public class SaveAndLoad
    {
        string ruta = "C:/Users/sergio/Desktop/AI Projects/GUARDADO/CircuitosTotales/";
        private int ultimo;

        public SaveAndLoad()
        {
            //Aqui se busca cual es el último archivo de pila creado, para que se utilice ese
            ultimo = 0;
            if (File.Exists(@ruta + "pila" + 0 + ".txt"))
            {
                int auxiliar = 1;
                while (File.Exists(@ruta + "pila" + auxiliar + ".txt"))
                {
                    auxiliar++;

                }

                ultimo = auxiliar - 1;

            }



        }

        public bool existe()
        {
            return File.Exists(@ruta + "pila" + 0 + ".txt");
        }

        public string Leer(string arch)
        {
            long tamanio = new System.IO.FileInfo(@ruta + arch + ultimo + ".txt").Length;
            if (tamanio == 0) ultimo--;

            string res;
            using (StreamReader archivo = new StreamReader(@ruta + arch + ultimo + ".txt"))
            {
                res = archivo.ReadLine() ?? "";
            }
            // string resultado = File.ReadLine(@ruta+arch+".txt").First();
            var lines = File.ReadAllLines(@ruta + arch + ultimo + ".txt");
            File.WriteAllLines(@ruta + arch + ultimo + ".txt", lines.Skip(1).ToArray());




            return res;
        }

        public void Escribe(string testo, string arch)

        {
            long tamanio = 0;
            if (File.Exists(@ruta + "pila" + ultimo + ".txt"))
            {
                tamanio = new System.IO.FileInfo(@ruta + arch + ultimo + ".txt").Length;
            }


            if (tamanio > 5000000) ultimo++;

            using (System.IO.StreamWriter archivo =
                   new System.IO.StreamWriter(@ruta + arch + ultimo + ".txt", true))
                archivo.WriteLine(testo);



        }

        public void Escribe2(string testo, string arch)

        {




            using (System.IO.StreamWriter archivo =
                new System.IO.StreamWriter(@ruta + arch + ".txt", true))
                archivo.WriteLine(testo);
        }





    }



    public class Prueba
    {
        private int num = 20;




        public class camino
        {


            /*
                   Inicial almacena la celda inicial del camino, ultimo el ultimo nodo añadido
                   y acumulado el resto de nodos del camino en orden.

           */
            public int inicial;
            public int ultimo;
            public List<int> acumulado;

            /*Constructor con una cadena de texto, no se realizan las comprobaciones necesarias, asumimos que se le
             pasa como parámetro una cadena de texto con los valores codificados conforme la función print. Dada una
             cadena de este tipo la codifica separando los nodos y adecuandolos a la estructura de representación interna.
            */
            public camino(string val)
            {
                string[] vect = val.Split(';');
                inicial = int.Parse(vect[0]);
                ultimo = int.Parse(vect[1]);
                acumulado = new List<int>();

                for (int i = 2; i < vect.Length; i++)
                {
                    acumulado.Add(int.Parse(vect[i]));
                }
            }

            //Constructor de copia
            public camino(camino valor)
            {
                ultimo = valor.ultimo;
                inicial = valor.inicial;
                acumulado = new List<int>(valor.acumulado);
            }

            //Al añadir un nodo al camino este pasa a ser el ultimo añadido
            public void aniade(int nodo)
            {
                if (ultimo != inicial) { acumulado.Add(ultimo); }
                ultimo = nodo;

            }
            //Constructor de camino dada una terna salida-medio-entrada, salida corresponde al primer nodo que se expandira,
            //entrada al de llegada que finalizará el camino. Se asume que los nodos estan conectados no se comprueba.
            public camino(int salida, int medio, int entrada)
            {
                inicial = entrada;
                ultimo = salida;
                acumulado = new List<int>();
                acumulado.Add(medio);
            }

            public int GetInicial()
            {
                return inicial;
            }

            public List<int> getAcumulado()
            {
                return acumulado;

            }

            //Tamaño del camino
            public int getTamanio()
            {
                return acumulado.Count;

            }

            //Que esto devuelva el id de la ultima celda añadida al camino,
            public int idUltima()
            {
                return ultimo;

            }

            /*Construimos una cadena de texto representativa del camino para que sea sencillo del almacenar y leer posteriormente,
             se añade el identificador en orden de cada elemento del camino separados por el carácter ';', y los dos primeros nodos
             representan el inicial y el último, lo hacemos así para que sea sencilo de leer para la representación de la clase camino.*/
            public string toprint()
            {
                string resultado = "";
                resultado = resultado + inicial + ";";
                resultado = resultado + ultimo + "";
                for (int i = 0; i < acumulado.Count; i++)
                {
                    resultado = resultado + ";" + acumulado[i];
                }
                return resultado;
            }
```

```
            /*Al igual que la función anterior, esta construye una cadena de carácteres que representa el camino, difiere de la anterior
             en que en este caso el primer nodo que aparece es el primero de la lista acumulado y asi en orden hasta el último que será el
             nodo de entrada, esta representación es más sencilla para ser leída por una persona u otro programa que no conozca la representación
             interna de la clase.
             */
            public string toprint2()
            {
                string resultado = "";
                resultado = resultado + acumulado[0];


                for (int i = 1; i < acumulado.Count; i++)
                {
                    resultado = resultado + ";" + acumulado[i];
                }
                resultado = resultado + ";" + ultimo;
                resultado = resultado + ";" + inicial;
                return resultado;
            }

            public bool Comprueba(int candidata)
            {
                //Comprobamos si la candidata esta ya en los vecinos
                //Opción eficiente guardar copia del camino como vector ordenado
                bool resultado = acumulado.Contains(candidata);
                return !resultado;

            }
        }

        private bool InRange2(int number) => 0 < number && number < num;

        //Crea una lista con los ids de las celdas vecinas a la pasada como parámetro acuerdo a las restricciones que se tienen
        //del grid
        public List<int> SacaVecinos(int celda)
        {
            List<int> vecinos = new List<int>();
            int x = celda / num;
            int y = celda - num * x;
            int aux;

            if (y % 2 == 0)
            {
                if (InRange2(x) && InRange2(y - 1))
                {
                    aux = y - 1 + num * (x);
                    vecinos.Add(aux);
                }
                if (InRange2(x - 1) && InRange2(y - 1))
                {
                    aux = y - 1 + num * (x - 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x - 1) && InRange2(y))
                {
                    aux = y + num * (x - 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x + 1) && InRange2(y))
                {
                    aux = y + num * (x + 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x - 1) && InRange2(y + 1))
                {
                    aux = y + 1 + num * (x - 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x) && InRange2(y + 1))
                {
                    aux = y + 1 + num * (x);
                    vecinos.Add(aux);
                }
            }
            else
            {
                if (InRange2(x + 1) && InRange2(y - 1))
                {
                    aux = y - 1 + num * (x + 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x) && InRange2(y - 1))
                {
                    aux = y - 1 + num * (x);
                    vecinos.Add(aux);
                }
                if (InRange2(x - 1) && InRange2(y))
                {
                    aux = y + num * (x - 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x + 1) && InRange2(y))
                {
                    aux = y + num * (x + 1);
                    vecinos.Add(aux);
                }
                if (InRange2(x) && InRange2(y + 1))
                {
                    aux = y + 1 + num * (x);
                    vecinos.Add(aux);
                }
                if (InRange2(x + 1) && InRange2(y + 1))
                {
                    aux = y + 1 + num * (x + 1);
                    vecinos.Add(aux);
                }
            }


            return vecinos;
        }









        public void main()
        {
            //Este valor booleano controla que queden valores por leer, si no quedaran se establece a true.
            bool finito = false;
            //Clase que se encarga del guardado y carga de los caminos
            SaveAndLoad salvador = new SaveAndLoad();
            camino actual;
            //Si ya existe un archivo pila, carga los caminos desde él.
            if (salvador.existe())
            {

                actual = new camino(salvador.Leer("pila"));

            }
            //En caso de que no exista, añadimos las ternas válidas.
            else
            {
                actual = new camino(84, 63, 43);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(84, 63, 62);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(84, 63, 82);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(83, 63, 64);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(83, 63, 43);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(83, 63, 62);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(82, 63, 64);
                salvador.Escribe(actual.toprint(), "pila");


                actual = new camino(82, 63, 43);
                salvador.Escribe(actual.toprint(), "pila");

                actual = new camino(64, 63, 62);
            }



            while (!finito)
            {

                //Obtenemos los vecinos del último nodo del camino.
                List<int> vecinos = SacaVecinos(actual.idUltima());

                //Para cada vecino, comprobamos si seria válido dentro del camino(que no este conectado con ningún nodo
                //ya incluido en el camino) y de ser asi, añadimos a la pila el camino anterior con este vecino válido.
                for (int i = 0; i < vecinos.Count; i++)
                {
                    int vecinaactual = vecinos[i];

                    List<int> vecinasvecina = SacaVecinos(vecinaactual);

                    bool valida = true;

                    for (int j = 0; j < vecinasvecina.Count && valida; j++)
                    {
                        valida = actual.Comprueba(vecinasvecina[j]);


                    }

                    if (valida)
                    {
                        bool auxiliar = (vecinasvecina.Contains(actual.GetInicial()) && actual.getTamanio() > 1);
                        if (auxiliar)
                        {

                            camino caminoauxiliar = new camino(actual);
                            caminoauxiliar.aniade(vecinaactual);
                            salvador.Escribe2(caminoauxiliar.toprint2(), "def");

                        }
                        else
                        {
                            camino caminoauxiliar = new camino(actual);
                            caminoauxiliar.aniade(vecinaactual);
                            salvador.Escribe(caminoauxiliar.toprint(), "pila");


                        }








                    }

                }

                string aux = salvador.Leer("pila");
                finito = string.IsNullOrEmpty(aux);
                if (!finito)
                {
                    actual = new camino(aux);
                }


            }
        }



        // Start is called before the first frame update
        void Start()
        {
            Prueba main = new Prueba();
            main.main();
        }

        // Update is called once per frame
        void Update()
        {

        }
    }
}
```

“CameraControl.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Cinemachine;

public class CameraControl : MonoBehaviour
{

    static readonly float Field_Of_View_Min = 80f;
    static readonly float Field_Of_View_Max = 20f;

    static readonly float Velocity_Zoom = 5f;

    static readonly float Drag_Speed = 2f;

    public float minX, minY, maxX, maxY;

    public Vector3 dragOrigin;

    public Camera myCamera;

    private CinemachineVirtualCamera _cameraVirtual;

    private void Awake()
    {
        _cameraVirtual = GetComponent<CinemachineVirtualCamera>();
    }

    // Update is called once per frame
    void Update()
    {
        float zoom = Input.GetAxisRaw("Mouse ScrollWheel");
        if (zoom < 0f && myCamera.fieldOfView < Field_Of_View_Min)
            _cameraVirtual.m_Lens.FieldOfView += Velocity_Zoom * Time.deltaTime;
        else if (zoom > 0f && myCamera.fieldOfView > Field_Of_View_Max)
            _cameraVirtual.m_Lens.FieldOfView -= Velocity_Zoom * Time.deltaTime;

        //Arrastrar cámara
        if (Input.GetMouseButtonDown(1))
        {
            dragOrigin = Input.mousePosition;
            //Debug.Log("Arrastrar");
            return;
        }
        else if (!Input.GetMouseButton(1))
            return;

        DragCamera();
    }

    void DragCamera()
    {
        Vector3 pos = myCamera.ScreenToViewportPoint(Input.mousePosition - dragOrigin);
        Vector3 move = new Vector3(pos.x * Drag_Speed * Time.deltaTime, 0f, 
pos.y * Drag_Speed * Time.deltaTime);

        float sumX = transform.position.x + move.x;
        float sumY = transform.position.z + move.z;


        if (sumX > minX && sumX < maxX && sumY > minY && sumY < maxY)
            transform.Translate(-move, Space.World);

    }
}
```

“CanvasManager.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class CanvasManager : MonoBehaviour
{


    public void NewScene(int index)
    {
        SceneManager.LoadScene(index);
    }
}
```

“Cell.cs”

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Cell : MonoBehaviour
{
    public static event Action<Cell> StartCellNeighbourSelected;

    [SerializeField]private int ID;
    private bool _selected;
    private bool _start;
    private List<Cell> _neighbourCells;
    private Vector2 _position;

    public bool GetStart() => _start;
    public void SetStart(bool s)
    {

        _start = s;
    }

    public Vector2 GetPosition => _position;
    public void SetPosition(Vector2 pos, int id)
    {
        ID = id;

        _position = pos;

        float offset = 0;

        if (pos.y % 2 != 0.0f)
            offset = 1.75f / 2f;
        float x = pos.x * 1.75f + offset;
        float y = pos.y * 2f * 0.75f;

        transform.SetPositionAndRotation(new Vector3(x,0,y),Quaternion.Euler(new Vector3(-90,0,0)));
    }

    public bool GetSelected() => _selected;
    public string SetSelected(bool s) { _selected = s;  return (ID + ""); }
    public string GetID() => ID.ToString();

    public List<Cell> GetNeighbourCell() => _neighbourCells;
    public void SetNeighbourCells(List<Cell> cells)
    {
        _neighbourCells = cells;
    }

    public bool CheckCell(Cell _actual)
    {
        if (_selected) return false;

        var encontrada = false;
        var start = false;

        var count = 0;
        foreach (var cell in _neighbourCells)
        {
            if (cell == _actual) encontrada = true;

            if (cell.GetStart())
            {
                start = true;

            }
            if (cell.GetSelected())
            {
                count++;
            }
        }

        if (start && encontrada)
        {
            Invoke(nameof(ThrowStartCellNeighbourSelectedEvent),0.01f);
            return true;
        }
        return count == 1 && encontrada;
    }

    private void ThrowStartCellNeighbourSelectedEvent()
    {
        StartCellNeighbourSelected(this);
    }
}
```

“ChangeScene.cs”

```
using Fungus;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ChangeScene : MonoBehaviour
{
    [SerializeField] private Flowchart flowchart;

    private void Awake()
    {
        if (!PlayerPrefs.HasKey("RaceMode"))
        {
            flowchart.ExecuteBlock("Intro");
            PlayerPrefs.SetInt("RaceMode", 1);

        }
        else
        {
            PlayerPrefs.SetInt("RaceMode", PlayerPrefs.GetInt("RaceMode") + 1);
            ChangeRanddomScene();
        }
    }

    [ContextMenu("DeletePlayerPrefs")]
    public void DeletePlayerPrefs() { PlayerPrefs.DeleteKey("RaceMode"); }

    public void ChangeRanddomScene()
    {
        SceneManager.LoadScene(Random.Range(2, 6));
    }
}
```

“CountdownEvent.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CountdownEvent : MonoBehaviour
{
    [SerializeField] private GameObject[] items;
    [SerializeField] private float time;
    [SerializeField] private ManagerCircuits manager;

    private int count;


    private void Start()
    {
        count = 0;
        InvokeRepeating(nameof(NextItem), time, time);
    }

    private void NextItem()
    {
        if (count == 0)
        {
            items[count].SetActive(true);
        }
        else if (count < 4)
        {
            items[count].SetActive(true);
            items[count - 1].SetActive(false);
        }
        else
        {
            items[count - 1].SetActive(false);
            manager.GO();
            CancelInvoke(nameof(NextItem));
        }
        count++;

    }
}
```

"ExitApp.cs"

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExitApp : MonoBehaviour
{
    public void AppQuit()
    {
        if (!Application.isEditor)
            Application.Quit();
        else
            UnityEditor.EditorApplication.isPlaying = false;

    }
}
```

"FailsTextActualize.cs"

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class FailsTextActualize : MonoBehaviour
{
    [SerializeField] private Text failText;

    private void OnEnable()
    {
        Brain.onFail += ActualizeText;
    }

    private void OnDisable()
    {
        Brain.onFail -= ActualizeText;
    }

    private void ActualizeText(int fail)
    {
        failText.text = "Fails: " + fail;
    }
}
```

“Grid.cs”

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Grid : MonoBehaviour
{
    [SerializeField] private int gridSizeX;
    [SerializeField] private int gridSizeY;
    [SerializeField] private GameObject cell;
    [SerializeField] private SelectionAction selAct;
    [SerializeField] private Manager manager;
    [SerializeField] private GameObject buttonContainer;
    [SerializeField] private GameObject floor;

    private int min = 1;

    private Cell[,] grid;

    public void SetInit() { selAct.SetActual(grid[3, 3]); }

    public Cell GetCell(string id)
    {
        for (int x = 0; x < gridSizeX; x++)
        {
            for (int y = 0; y < gridSizeY; y++)
            {
                if (grid[x, y].GetID().Equals(id))
                {
                    return grid[x, y];
                }
            }
        }
        return null;
    }

    private void OnEnable()
    {
        grid = new Cell[gridSizeX,gridSizeY];
        Manager.FinishPath += UP;
    }

    private void Start()
    {
        buttonContainer.SetActive(false);
        selAct.enabled = true;
        manager.Clear();
        for (int x = 0; x < gridSizeX; x++)
        {
            for (int y = 0; y < gridSizeY; y++)
            {
                var newCell = Instantiate(cell);
                grid[x, y] = newCell.GetComponent<Cell>();
                grid[x, y].SetPosition(new Vector2(x,y), y + (x * gridSizeX));

            }
        }

        List<Cell> neighbourCells = new List<Cell>();

        for (int x = 0; x < gridSizeX; x++)
        {
            for (int y = 0; y < gridSizeY; y++)
            {
                if (OnBounds(x, y))
                {
                    continue;
                }
                if (y % 2 == 0)
                {
                    if (InRange(min, gridSizeX, x) && InRange(min, gridSizeY, y - 1))
                    {
                        neighbourCells.Add(grid[x, y - 1]);
                    }
                    if (InRange(min, gridSizeX, x - 1) && InRange(min, gridSizeY, y - 1))
                    {
                        neighbourCells.Add(grid[x - 1, y - 1]);
                    }
                    if (InRange(min, gridSizeX, x - 1) && InRange(min, gridSizeY, y))
                    {
                        neighbourCells.Add(grid[x - 1, y]);
                    }
                    if (InRange(min, gridSizeX, x + 1) && InRange(min, gridSizeY, y))
                    {
                        neighbourCells.Add(grid[x + 1, y]);
                    }
                    if (InRange(min, gridSizeX, x - 1) && InRange(min, gridSizeY, y + 1))
                    {
                        neighbourCells.Add(grid[x - 1, y + 1]);
                    }
                    if (InRange(min, gridSizeX, x) && InRange(min, gridSizeY, y + 1))
                    {
                        neighbourCells.Add(grid[x, y + 1]);
                    }
                }
                else
                {
                    if (InRange(min, gridSizeX, x + 1) && InRange(min, gridSizeY, y - 1))
                    {
                        neighbourCells.Add(grid[x + 1, y - 1]);
                    }
                    if (InRange(min, gridSizeX, x ) && InRange(min, gridSizeY, y - 1))
                    {
                        neighbourCells.Add(grid[x, y - 1]);
                    }
                    if (InRange(min, gridSizeX, x - 1) && InRange(min, gridSizeY, y))
                    {
                        neighbourCells.Add(grid[x - 1, y]);
                    }
                    if (InRange(min, gridSizeX, x + 1) && InRange(min, gridSizeY, y))
                    {
                        neighbourCells.Add(grid[x + 1, y]);
                    }
                    if (InRange(min, gridSizeX, x) && InRange(min, gridSizeY, y + 1))
                    {
                        neighbourCells.Add(grid[x , y + 1]);
                    }
                    if (InRange(min, gridSizeX, x +1) && InRange(min, gridSizeY, y + 1))
                    {
                        neighbourCells.Add(grid[x +1, y + 1]);
                    }
                }
                grid[x, y].SetNeighbourCells(neighbourCells);

                neighbourCells = new List<Cell>();
            }
        }
        selAct.SetActual(grid[3,3]);
        grid[3, 3].SetSelected(true);
        grid[3, 3].SetStart(true);
        grid[3, 3].gameObject.GetComponent<Renderer>().material.color = Color.red;
    }

    private bool OnBounds(int x, int y)
    {
        if (x == 0 || y == 0 || x == gridSizeX - 1 || y == gridSizeY - 1)
        {
            grid[x,
y].gameObject.GetComponent<Renderer>().material.color = Color.black;//Esto es estética
            Vector3 scale = new Vector3(100,100,500);
            grid[x, y].gameObject.tag = "terrain";
            grid[x, y].gameObject.layer = 8;
            grid[x, y].transform.localScale = scale;
            return true;
        }
        return false;
    }

    private void UP()
    {
        floor.SetActive(true);

        for (int x = 0; x < gridSizeX; x++)
        {
            for (int y = 0; y < gridSizeY; y++)
            {
                if (!grid[x, y].GetSelected())
                {
                    Vector3 scale = new Vector3(100, 100, 500);
                    grid[x, y].gameObject.tag = "terrain";
                    grid[x, y].gameObject.layer = 8;
                    grid[x, y].transform.localScale = scale;
                }
                else
                {
                   // grid[x, y].gameObject.GetComponent<Renderer>().material.color = Color.white;
                }
            }
        }
        buttonContainer.SetActive(true);
    }

    private bool InRange(int min, int max, int number) => min <= number && max - 1  > number;

    public void ResetPath()
    {
        for (int x = 0; x < gridSizeX; x++)
        {
            for (int y = 0; y < gridSizeY; y++)
            {
                Destroy(grid[x, y].gameObject);
            }
        }
        enabled = false;
        grid = new Cell[gridSizeX, gridSizeY];
        Start();
    }
}

```

"Manager.cs"

```
using Fungus;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Manager : MonoBehaviour
{
    public static event Action FinishPath;

    [SerializeField] private SelectionAction action;
    [SerializeField] private Grid grid;
    [SerializeField] private GameObject bot;
    [SerializeField] private Vector3 pos;
    [SerializeField] private GameObject meta;
    [SerializeField] private GameObject ResumeMenuContainer;
    [SerializeField] private Cinemachine.CinemachineVirtualCamera followCamera, zenitalCamera;
    [SerializeField] private Cinemachine.CinemachineBrain cameraBrain;
    [SerializeField] private Text failText;

    [SerializeField] private Flowchart fc;

    public Cinemachine.CinemachineBlendDefinition cameraDefinition;

    private int _initStartSelectedTimes;
    private bool _isFinished;
    private GameObject _currentBot;
    private GameObject _meta;
    private Vector3 _pointToView;

    private string circuitName;
    private List<float> _circuitPoints = new List<float>();

    Dictionary<string, string> _circuitsNameDictionary = new Dictionary<string, string>();
    static readonly string DICTIONARY = "Circuits";

    public bool GetFinished() => _isFinished;
    public void Clear() { _initStartSelectedTimes = 0; _isFinished = false; grid.SetInit(); Destroy(_meta); circuitName = ""; _circuitPoints = new List<float>(); }

    private void Start()
    {
        if (!PlayerPrefs.HasKey("Dibujado"))
        {
            if (fc != null)
            {
                fc.ExecuteBlock("Intro");
                PlayerPrefs.SetInt("Dibujado", 1);
            }

        }
        else
        {
            if (fc != null)
                PlayerPrefs.SetInt("Dibujado", PlayerPrefs.GetInt("Dibujado") + 1);
        }

        followCamera.enabled = false;
        zenitalCamera.enabled = true;

        cameraBrain.m_DefaultBlend = cameraDefinition;

        if (SaveAndLoad.SaveExists(DICTIONARY))
            _circuitsNameDictionary = SaveAndLoad.Load<Dictionary<string, string>>(DICTIONARY);
    }

    [ContextMenu("DeletePlayerPrefs")]
    public void Delete() { PlayerPrefs.DeleteKey("Dibujado"); }

    public void SaveNewDataDictionary(string name, string data)
    {
        if (!GetNameDictionary(name))
        {
            _circuitsNameDictionary.Add(name, data);
            SaveAndLoad.Save<Dictionary<string, string>>(_circuitsNameDictionary, DICTIONARY);
            Debug.Log("primera vez");
        }/*
        else
        {
            _circuitsNameDictionary.Remove(name);
            SaveAndLoad.Save<Dictionary<string, string>>(_circuitsNameDictionary, DICTIONARY);
            Debug.Log("otras");
        }*/
    }

    public bool GetDictionaryCreate()
    {
        return SaveAndLoad.SaveExists(DICTIONARY);
    }

    public string GetDataDictionary(string name)
    {
        string newValue;
        _circuitsNameDictionary.TryGetValue(name, out newValue);
        return newValue;
    }

    public bool GetNameDictionary(string name)
    {
        foreach (string a in _circuitsNameDictionary.Keys)
        {
            if (name == a)
                return true;
        }
        return false;
    }

    internal Text getFailText()
    {
        return failText;
    }

    private void OnEnable()
    {
        Cell.StartCellNeighbourSelected += StartCellNeighbourSelected;
        Brain.Death += Respawn;
    }

    public void Update()
    {
        if (Input.GetKeyDown(KeyCode.P))
        {
            ResumeMenuContainer.SetActive(true);
            Time.timeScale = 0f;
        }

        if (Input.GetKeyDown(KeyCode.V))
        {
            if (_currentBot == null)
                return;

            followCamera.enabled = !followCamera.enabled;
            zenitalCamera.enabled = !zenitalCamera.enabled;

            if (followCamera.enabled)
                FollowInstance();
        }
    }

    public void AddCellToCircuit(string c)
    {
        circuitName += c + ";";
        float value;
        float.TryParse(c, out value);
        _circuitPoints.Add(value);
    }

    public void Resume()
    {
        Time.timeScale = 25f;
    }

    public void KillANN()
    {
        if (_currentBot != null) Destroy(_currentBot);
    }

    public void Respawn()
    {
        _pointToView.y = pos.y;

        Debug.Log(_pointToView);

        //var botInst = Instantiate(bot, pos, Quaternion.LookRotation(_pointToView - pos, Vector3.up));
        var botInst = Instantiate(bot, pos, Quaternion.identity);
        botInst.transform.LookAt(_pointToView);
        botInst.GetComponentInChildren<Animator>().SetBool("Running", true);
        Brain brain = botInst.GetComponent<Brain>();
        brain._isAleatoryCircuit = true;
        brain.currentAleatoryCircuitName = circuitName;
        brain.pointsCircuitName = _circuitPoints;
        _currentBot = botInst;
        FollowInstance();
    }

    public void SetPointToView(Vector3 p)
    {
        _pointToView = p;
    }

    private void FollowInstance()
    {
        if (_currentBot == null)
```

```
            return;

        Vector3 newPosition = new Vector3(0f, 0.11f,-0.5f);

        var transposer = followCamera.GetCinemachineComponent<Cinemachine.CinemachineTransposer>();
        transposer.m_FollowOffset = newPosition;
        followCamera.m_Lens.FieldOfView = 37f;
        followCamera.m_Follow = _currentBot.transform;
        followCamera.m_LookAt = _currentBot.transform;
    }

    private void StartCellNeighbourSelected(Cell cellScript)
    {
        _initStartSelectedTimes++;
        if (_initStartSelectedTimes == 1)
            _pointToView = cellScript.transform.position;
        else if (_initStartSelectedTimes == 2)
        {
            var position = cellScript.GetPosition;

            float offset = 0;

            if (position.y % 2 != 0.0f)
                offset = 1.75f / 2f;
            float x = position.x * 1.75f + offset;
            float y = position.y * 2f * 0.75f;

            _meta = Instantiate(meta, new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0)));

            action.enabled = false;
            FinishPath();
            _isFinished = true;
        }
    }

    public void FinishedTrainingPath() { FinishPath(); }

    public void Help()
    {
        Time.timeScale = 25f;
        Invoke(nameof(ExecuteHelp), 1);
    }

    private void ExecuteHelp() { fc.ExecuteBlock("Help"); }
}

```

“ManagerCanvas.cs”

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Player
{
    public int id;
    public int score;
    public string name;

    public Player(int id, int s, string n) { this.id = id; score = s; name = n; }
}

public class ManagerCanvas : MonoBehaviour
{
    private List<Player> players = new List<Player>();
    public Text text;
    public string[] names;
    public GameObject pauseMenu;

    //finish
    public Text textoFin;
    public Image finishCanvas;


    private void Awake()
    {
        finishCanvas.canvasRenderer.SetAlpha(1.0f);
        finishCanvas.CrossFadeAlpha(0.0f, 3, false);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.P))
        {
            pauseMenu.SetActive(true);
            Time.timeScale = 0f;
        }
    }

    public void Resume()
    {
        pauseMenu.SetActive(false);
        Time.timeScale = 10f;
    }

    /*
    private void Start()
    {
        for (int i = 1; i < 5; i++)
        {
            players.Add(new Player(i,0, names[i - 1]));
        }
    }
    */
    public void ActualizeCanvasPositions(int id, int score)
    {
        foreach (var player in players)
            if (player.id == id)
                player.score += score;

        text.text = GetInOrder();
    }

    private string GetInOrder()
    {
        var aux = players.OrderByDescending(x => x.score);

        string txt = "";
        var count = 1;
        foreach (var play in aux)
        {
            txt += $"-{count}- {play.name} \n";
            count++;
        }
        return txt;
    }

    public void Finish(List<FinishPos> positions)
    {
        var txt = "Resultado: \n";
        foreach (var pos in positions)
        {
            txt += $"{pos.name}: {pos.pos} \n";
        }

        textoFin.text = txt;
        finishCanvas.CrossFadeAlpha(1.0f, 3,false);
    }

}

```

"ManagerCircuits.cs"

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ManagerCircuits : MonoBehaviour
{
    Dictionary<string, string> _circuitsNameDictionary = new Dictionary<string, string>();
    public string DICTIONARY;

    [SerializeField] private List<Brain> brains;
    [SerializeField] private PlayerMovement playerMov;

    public void GO()
    {
        foreach (var brain in brains)
            brain.enabled = true;
        playerMov.enabled = true;
    }

    public void STOP()
    {
        foreach (var brain in brains)
            brain.enabled = false;
        playerMov.enabled = false;
    }

    private void Awake()
    {
        if (SaveAndLoad.SaveExists(DICTIONARY))
            _circuitsNameDictionary = SaveAndLoad.Load<Dictionary<string, string>>(DICTIONARY);
    }

    private void OnEnable()
    {
        Time.timeScale = 10f;
    }

    public void SaveNewDataDictionary(string name, string data)
    {
        if (!GetNameDictionary(name))
        {
            _circuitsNameDictionary.Add(name, data);
            SaveAndLoad.Save<Dictionary<string, string>>(_circuitsNameDictionary, DICTIONARY);
        }
    }

    public bool GetDictionaryCreate()
    {
        return SaveAndLoad.SaveExists(DICTIONARY);
    }

    public string GetDataDictionary(string name)
    {
        string newValue;
        _circuitsNameDictionary.TryGetValue(name, out newValue);
        return newValue;
    }

    public bool GetNameDictionary(string name)
    {
        foreach (string a in _circuitsNameDictionary.Keys)
        {
            if (name == a)
                return true;
        }
        return false;
    }
}

```

“ManagerMainMenu.cs

```
using Fungus;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ManagerMainMenu : MonoBehaviour
{
    [SerializeField] private Animator animatorMenuButt;
    [SerializeField] private Animator animatorStartButt;
    [SerializeField] private GameObject poplar;
    [SerializeField] private Flowchart flowchart;

    // Start is called before the first frame update
    void Start()
    {
        if (!PlayerPrefs.HasKey("START"))
        {
            flowchart.ExecuteBlock("Intro");
            PlayerPrefs.SetInt("START",1);

        }
        else
        {
            PlayerPrefs.SetInt("START",PlayerPrefs.GetInt("START")+1);
            OnFinishFungus();
        }

        Time.timeScale = 1.0f;
    }

    [ContextMenu("DeletePlayerPrefs")]
    public void Delete() { PlayerPrefs.DeleteKey("START"); }

    public void OnFinishFungus()
    {
        poplar.SetActive(true);
        animatorMenuButt.SetBool("Start", true);
    }

    public void OnClickStart()
    {
        animatorMenuButt.SetBool("Finished", true);
        animatorStartButt.SetBool("Initied", true);
        animatorStartButt.SetBool("Finish", false);
    }

    public void OnClickBack()
    {
        animatorMenuButt.SetBool("Finished", false);
        animatorStartButt.SetBool("Initied", false);
        animatorStartButt.SetBool("Finish", true);
    }

    public void OnClickGameMode(int sceneIndex)
    {
        SceneManager.LoadScene(sceneIndex);
    }
}

```

"Meta.cs"

```
using Fungus;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FinishPos
{
    public string name;
    public int pos;

    public FinishPos(string n, int p) { name = n; pos = p; }
}

public class Meta : MonoBehaviour
{
    private List<FinishPos> positions = new List<FinishPos>();
    public int bots;
    public ManagerCircuits manager;
    public ManagerCanvas manCanvas;

    public Flowchart flowchart;

    private int count = 0;

    private void OnTriggerEnter(Collider other)
    {
        if (other.GetComponent<Brain>() != null || other.GetComponent<PlayerMovement>() != null)
        {
            count++;
            positions.Add(new FinishPos(other.gameObject.name, count));
        }
        if (count == 4)
        {
            manCanvas.Finish(positions);

            if (Win())
                flowchart.ExecuteBlock("WIN");
            else
                flowchart.ExecuteBlock("LOSE");


            manager.STOP();
        }
    }

    private bool Win()
    {
        foreach (var p in positions)
            if (p.name == "ANNKart" && p.pos == 1)
                return true;
        return false;
    }
}

```

"PlayerMovement.cs"

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    [SerializeField] private Animator anim;
    [SerializeField] private float turnSpeed;
    [SerializeField] private float speed;

    void Update()
    {
        transform.Translate(Vector3.forward * speed * Time.deltaTime * Input.GetAxis("Vertical"));
        transform.Rotate(Vector3.up * turnSpeed * Input.GetAxis("Horizontal"));
        if (Input.GetAxis("Vertical") == 0)
            anim.SetBool("Running", false);
        else
            anim.SetBool("Running", true);
    }

    private bool InRange(float value, float max, float min) { return value < max && value > min; }
}
```

"SelectionAction.cs"

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SelectionAction : MonoBehaviour
{
    [SerializeField] private Manager manager;
    [SerializeField] private GameObject wayPoint;

    private List<Cell> selectedCells = new List<Cell>();
    private Cell _actual;
    private int _wayPointCount;
    private List<GameObject> wayPoints = new List<GameObject>();

    public void SetActual(Cell c) { _actual = c; }

    public bool canInteract;

    void Update()
    {
        if (!canInteract) return;
        if (Input.GetMouseButton(0))
        {
            RaycastHit hit;
            int layerMask = 1 << 9;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hit, Mathf.Infinity, layerMask))
            {
                var cellScript = hit.transform.gameObject.GetComponent<Cell>();
                if (cellScript != null)
                {
                    if (!cellScript.CheckCell(_actual)) return;
                    else
                    {
                        string aux = cellScript.SetSelected(true);
                        manager.AddCellToCircuit(aux);
                        cellScript.gameObject.GetComponent<Renderer>().material.color = Color.blue;
                        _actual = cellScript;
                        selectedCells.Add(cellScript);
                        TryInstantiateWayPoint(cellScript);

                    }
                }
            }
        }
        if (Input.GetMouseButtonUp(0))
        {
            if (!manager.GetFinished())
            {
                foreach (var cell in selectedCells)
                {
                    if(cell!= null)
                        cell.gameObject.GetComponent<Renderer>().material.color = Color.white;
                        cell.SetSelected(false);
                }
                manager.Clear();
                selectedCells.Clear();
                ClearWayPoints();

            }
        }
    }

    private void TryInstantiateWayPoint(Cell cellScript)
    {
        _wayPointCount++;
        if (_wayPointCount <= 3) return;

        _wayPointCount = 0;

         var position = cellScript.GetPosition;

        float offset = 0;

        if (position.y % 2 != 0.0f)
            offset = 1.75f / 2f;
        float x = position.x * 1.75f + offset;
        float y = position.y * 2f * 0.75f;

        wayPoints.Add( Instantiate(wayPoint, new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0))));
    }

    public void ClearWayPoints()
    {
        foreach (var wp in wayPoints)
            Destroy(wp);
        wayPoints.Clear();
    }
}

```

“TrainingFinisEvent.cs”

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TrainingFinishEvent : MonoBehaviour
{
    public static Action onFinishTraining;

    private bool IN = false;
    private int count = 0;

    private void OnTriggerEnter(Collider other)
    {
        if (IN) return;

        if (other.GetComponent<Brain>() != null)
        {
            count++;
            IN = true;
            if (count == 1)
            {
                onFinishTraining();
                count = 0;
            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (!IN) return;
        IN = false;
    }
}

```

“TrainingPathCreator.cs”

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;

public class TrainingPathCreator : MonoBehaviour
{
    [SerializeField] private Manager manager;
    [SerializeField] private GameObject wayPoint;
    [SerializeField] private Grid grid;
    [SerializeField] private GameObject meta;
    [SerializeField] private GameObject metaCollider;

    private List<Cell> selectedCells = new List<Cell>();
    private Cell _actual;
    private int _wayPointCount;
    private List<GameObject> wayPoints = new List<GameObject>();

    public void SetActual(Cell c) { _actual = c; }

    public string[] lines;

    private void Start()
    {
        ReadData();
        if (!PlayerPrefs.HasKey("Training"))
            PlayerPrefs.SetInt("Training", 0);

        Invoke(nameof(getNextCircuit),3f);
    }

    [ContextMenu("DeletePlayerPrefs")]
    private void DeletePlayerPrefs() => PlayerPrefs.DeleteKey("Training");

    private void ReadData()
    {
        var sr = new StreamReader(Application.dataPath + "/" + "DATA/def.txt");
        var fileContents = sr.ReadToEnd();
        sr.Close();
        lines = fileContents.Split("\n"[0]);
    }

    private void getNextCircuit()
    {
        var count = PlayerPrefs.GetInt("Training");
        bool first = false;
        var name = "";
        Debug.Log(lines[count]);
        var positions = lines[count].Split(";"[0]);

        for (int i = 0; i< positions.Length; i++)
        {
            if (i == positions.Length - 1)//Quitando el ultimo carácter no válido de la cadena
            {
                var arrayChars = positions[i].ToCharArray();
                var res = "";
                for (int j = 0; j < arrayChars.Length - 1; j++)
                    res += arrayChars[j];

                var cell = grid.GetCell(res);
                cell.SetSelected(true);

                name += res ;

                //Crear meta
                var position = cell.GetPosition;

                float offset = 0;

                if (position.y % 2 != 0.0f)
                    offset = 1.75f / 2f;
                float x = position.x * 1.75f + offset;
                float y = position.y * 2f * 0.75f;

                wayPoints.Add(Instantiate(meta,new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0))));

                //Suscribo al evento de meta
                TrainingFinishEvent.onFinishTraining += FinalizadaVuelta;

                wayPoints.Add(Instantiate(metaCollider, new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0))));
            }
            else
            {
                name += positions[i] + ";";
                var cell = grid.GetCell(positions[i]);
                cell.SetSelected(true);
                if (i == 1) manager.SetPointToView(cell.transform.position);

                //Crear waypoints
                if (i != 0 && i % 3 == 0)
                {
                    var position = cell.GetPosition;

                    float offset = 0;

                    if (position.y % 2 != 0.0f)
                        offset = 1.75f / 2f;
                    float x = position.x * 1.75f + offset;
                    float y = position.y * 2f * 0.75f;

                    wayPoints.Add(Instantiate(wayPoint, new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0))));
                }

            }
        }

        manager.FinishedTrainingPath();
        manager.Respawn();


        GameObject.FindObjectOfType<Brain>().currentAleatoryCircuitName = name;
    }

    private void FinalizadaVuelta()
    {
        Debug.Log("Terminada la vuelta");

        foreach (var obj in wayPoints)
            Destroy(obj);

        wayPoints.Clear();

        TrainingFinishEvent.onFinishTraining -= FinalizadaVuelta;

        manager.KillANN();
        manager.Clear();
        grid.ResetPath();

        PlayerPrefs.SetInt("Training", PlayerPrefs.GetInt("Training") + 1);

        Invoke(nameof(getNextCircuit), 10f);
    }

    private void TryInstantiateWayPoint(Cell cellScript)
    {
        _wayPointCount++;
        if (_wayPointCount <= 3) return;

        _wayPointCount = 0;

        var position = cellScript.GetPosition;

        float offset = 0;

        if (position.y % 2 != 0.0f)
            offset = 1.75f / 2f;
        float x = position.x * 1.75f + offset;
        float y = position.y * 2f * 0.75f;

        wayPoints.Add(Instantiate(wayPoint, new Vector3(x, 0, y), Quaternion.Euler(new Vector3(-90, 0, 0))));
    }

    public void ClearWayPoints()
    {
        foreach (var wp in wayPoints)
            Destroy(wp);
        wayPoints.Clear();
    }
}
```

“WayPointsPositioner.cs”

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class WayPointsPositioner : MonoBehaviour
{
    [SerializeField] private GameObject modularTrackContainer;
    [SerializeField] private GameObject wayPoint;
    public MeshCollider[] modularTracks;
    string meta = "Meta";

    void Start()
    {
        modularTracks = modularTrackContainer.GetComponentsInChildren<MeshCollider>();

        foreach (var piece in modularTracks)
        {

            var wayPt = Instantiate(wayPoint, piece.transform.position, Quaternion.identity, piece.transform);
            wayPt.transform.localRotation = Quaternion.Euler(new Vector3(0, 0, 0));

            if (piece.name.Contains(meta))
            {
                //Es meta
                var script = wayPt.AddComponent<ID>();
                script.tipo = tipoCheckpoint.Meta;
            }
        }

    }
}

```

“OptionControls.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class OptionControls : MonoBehaviour
{
    public Slider sliderVolume;
    public AudioSource audioSource;

    public void SaveVolume()
    {
        SaveAndLoad.SaveVolume(sliderVolume.value);
        audioSource.volume = sliderVolume.value;
    }
}
```

“VolumeController.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(AudioSource))]
public class VolmeController : MonoBehaviour
{
    AudioSource _audioSource;

    private void Awake()
    {
        _audioSource = GetComponent<AudioSource>();
    }

    private void Start()
    {
        if (SaveAndLoad.SaveVolumeExists())
        {
            _audioSource.volume = SaveAndLoad.LoadVolume();
        }
    }
}
```