UNIVERSITAT JAUME·I

PROJECT'S MEMORY
RUFFIDIO'S ADVENTURE

Adrián Carboneras Mas

INDEX

CONTENT

Tables

Figures

1 Technical proposal

Summary

This document presents the memory of the final degree project which consists of the development of a platforming videogame in which exists an important AI component.

The main idea is that, instead of letting the player play the game to complete the levels from the platforming game, the main character controlled by an AI, will learn how to complete it based on a set of abilities taught by the player. Neural networks will be used to perform the automatic learning. The Ai will be trained with reinforcement learning.

The project will focus mainly on the programming part of the videogame leaving aside the art, making simple forms for everything or downloading the assets from the internet. Unity3d will be used to make the video game because it's a powerful but an easy to use tool with a lot of technical support. The results of the project will be tested by expert users.

Keywords

AI, machine learning, programming, Unity3D, Idle games, genetic algorithms, reinforcement learning.

1.Introduction

Most of the video games that exist have an AI. But due to the difficulty of implementing machine learning on a good designed game, not many apply it. As an example, the amiibos [1] from Nintendo on Smash Bros can get trained to fight better, but they are an extra as you need to buy external toys in order to use this function, so a lot of people won't use it. And that is why this final degree project is going to focus on the use of neural networks in order to implement an AI able to learn inside a video game designed for it, which hopefully will show ways of implementing machine learning on video games.

The problem in this final degree project that needs to be solved in order to create the game is the implementation of the neural network.

Even though the main work of this final project degree is the AI, a video game is going to be built as a technical demo. The video game will be based on idle games, which will be explained in the next section and will be about a monster, which will start inside a house. This is the only place where the player will be able to move the monster with the cursor in order to place it inside machines that teach the monster abilities or calm its necessities. The player will be able to buy more machines with money. Inside this part of the game there won't be any kind of machine learning. Once the player feels the monster it's ready, he will be able to change to a map that the monster needs to learn using the neural networks. Inside this map the monster will also be able to collect the money necessary to build the machines. The game finishes once the monster gets to the end of the map. The game is explained better on the third paragraph, preview of the videogame.

Google created something similar [12] with an artificial intelligence that learned to play Space Invaders or Breakout using reinforcement learning. In more than a half of the games it played, the AI performed better than an average human player. This AI also performed better on the 43 games tested than other AI.

The main focus of this project will be programming the AI. It will be based on the Mario AI challenges, where people provide Mario from the first Super Mario Bros an AI to cross the first level. Also, as an example of how the AI is going to work, MarI/O is a good example of something similar to the intended result of this videogame [11].

The chosen tool to create this videogame is Unity due to how easy it is to use.

The rest of the document is organized as follows:

Section 2 comments about the background of the project, what is based on and the explication of the technology that uses, section 3 is a preview of the video game that is going to be made, section 4 has the objectives that this final degree project is going to pursue, section 5 has the subjects that this final degree project has relation to in order to prove that this final degree project is justified, section 6 has the planning of the project, which is how many hours will this final degree project take, section 7 has the methodology that this final degree project is going to follow with the programming and revision of time, section 8 talks about the tools that this final degree project is going to take, section 9 the risks that this final degree project has and how to overcome them and finally section 10 has the bibliography.

2. Background

2.1 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are a system of interconnected neurons that collaborate between them to produce a result. They are inspired by the nervous system of animals, and are used to estimate functions that depend on a large quantity of inputs or are unknown.

Examinations [2] of humans' central nervous systems inspired the concept of artificial neural networks. In an artificial neural network, simple artificial nodes, known as "neurons", "neurodes", "processing elements" or "units", are connected together to form a network which mimics a biological neural network.

There is no single formal definition of what an artificial neural network is. However, a class of statistical models may commonly be called "neural" if it possesses the following characteristics:

1. contains sets of adaptive weights, i.e. numerical parameters that are tuned by a learning algorithm, and
2. capability of approximating non-linear functions of their inputs.

The adaptive weights can be thought of as connection strengths between neurons, which are activated during training and prediction. Neural networks are similar to biological neural networks in the performing of functions collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which individual units are assigned.

2.2 MARIO AI CHAMPIONSHIP

The Mario AI Championship [3] was created in order to benchmark AI techniques in video games. It was celebrated from 2009 to 2012 and consisted on the implementation of a Mario AI in a fan made Mario Bros video game, created by Notch, famous for creating Minecraft. Not every contestant used neural networks, but some of them did and this final degree project is inspired on the videos that showed this method.

In the paper, Togelius, Sergey Karakovskiy and Julian describe that the previous competitions with AI used to use chess as a benchmark game and that they didn't have any knowledge of a previous competition focusing of platforming games.

In order to provide a benchmark in a platforming game, they had to modify the Mario Game to make it suitable for the AI and the creation of an API. The API consisted on the following java interfaces:

1) The Environment interface: Describes the game state to the agent at each time step.
2) The Agent interface: This is the only interface that needs to be implemented in order to create a functional Mario-playing agent. The key method here is getAction, which takes an Environment as input and returns a five-bit array specifying the action to take
3) The Task interface: The Task defines certain aspects of the gameplay, including the presence or absence of sensory noise, whether there should be intermediate rewards and exactly which evaluation function is used.

MarI/O is similar to some of the entrants of this championship. In the video the author explains that he has added Mario vision in order for Mario to see the enemies, the floor and the wall.

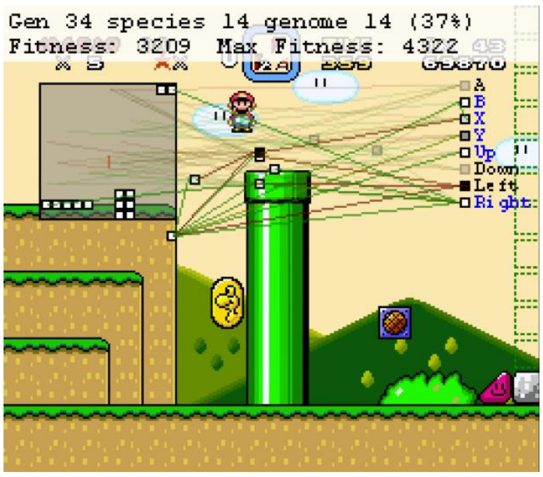

Gen 34 species 14 genome 14 (37%)
Fitness: 3209 Max Fitness: 4322

Figure 1

This image reflects the program. The interconnected white and black dots are the neurons which mean that they will react on the floor when they are white or on the objects when they are black. The lines that connect the neurons mean that, when they are green, they will react on the objects of the same colour the neurons are and when they are red, they will react on the objects of the opposite colour.

From that, the AI starts creating random neural on the first generation. Once a generation has ended, the bests neural networks (in this case, the ones who have taken Mario farthest) are selected and breed in order to produce better neural networks, and then, another generation starts until it completes the level. This is done using genetic algorithms. And, due to the use of reinforcement learning instead of supervised, the AI doesn't need examples, it just needs rewards. In this case, the fitness value is the reward.

2.3 UNITY

Unity3D it's a very powerful engine which has been taught during the degree. It's the most common used by the indie developers, so there's a lot of documentation on the internet. The code can be written in javascript or C#, the latest which has been taught during the degree. It is used to program video games for PC, consoles, mobiles and web sites [4]. Unity makes use of Direct3D, OpenGL or any kind of API suitable for the compilation of the device it's aimed to. Unity used to use mono as his solution to scripting but it has changed to IL2CPP recently in order to have better performance [5]. And in order to be able to create video games for game consoles the developer needs to have a license that needs to be asked to the company proprietary of that video game console.

Worldwide, Unity takes a plus 45% share of the full feature game engine market. The Unity game engine is far more popular amongst developers than any other game development software. The proportion of developers relying on Unity as their primary development tool and using Unity is growing all the time. Unity customers include Cartoon Network, Coca-Cola, Disney, Electronic Arts, LEGO, Microsoft, NASA, Nexon, Nickelodeon, Square, Ubisoft and Warner Bros. From large and small studios to independent professionals, more and more developers are moving to Unity. Unity touches 600 million gamers all over the world through games made using the engine [6].

2.4 IDLE GAMES

Idle games are video games whose gameplay consists of the player performing simple actions (such as clicking on the screen) repeatedly to gain currency. This can be used to obtain items or abilities that increase the rate at which currency accrues.

A common theme is to give the player sources of time-based income displayed as "buildings", such as factories or farms. Some incremental games require players to collect multiple currencies to progress through the game. In some games, even the clicking becomes unnecessary after a time, as the game plays itself, including in the player's absence, hence the moniker "idle game".

Incremental games gained popularity in 2013 after the success of *Cookie Clicker*, although earlier games such as *Cow Clicker* and *Candy Box* were based on the same principles. In 2015, the gaming press observed such games again proliferating on the Steam game distribution platform with titles such as *Clicker Heroes* [7]. To show its popularity, according to Steamcharts the maximum number of players that the game had playing at the same time it's 65560 [8]. But the game didn't start there, so if we go to Kongregate, a web page where the game was from the beginning, the game shows 25464788 times the game was played [9].

3.Preview of the videogame

The game that is intended to be created will be called Rufidio's adventure. This game will be an idle game which will consist on the player taking care of a creature. The player will have to feed it or make it sleep in order to make it happy. Once Rufidio is happy, it is able to run through a map in order to fulfil its destiny and defeat a dragon. The player will also need to teach Rufidio some abilities in order to overcome the dangers of that map. But, learning that map by heart, using the latest genetic algorithms, Rufidio will be able to get to the ending and defeat the dragon.

As we can see in figure 2, the player starts in a room with some machines to train the monster. Using the mouse, the player will be able to pick up the monster and drag it to the machines. The spring teaches to jump after some time, that treadmill teaches to run, etc. This part of the game will be an idle game, in which the player will just wait for the monster to end their training. Once the monster is trained the player will be able to go to an adventure map.

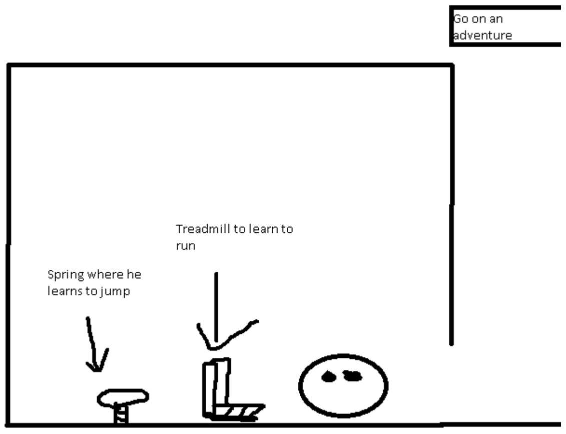


Figure 2

In figure 3, an example can be seen of the map that the monster must complete. Using the neural network, the monster will play the map until it gets tired or gets to the end. The monster will be able to rest when the player comes back to the house and puts the monster on a bed. Falling through a pit or staying stuck will restart the map. Coins will also be distributed through the map to purchase more machines or evolve them. Evolving a machine will make the player wait less time for the monster to recover.

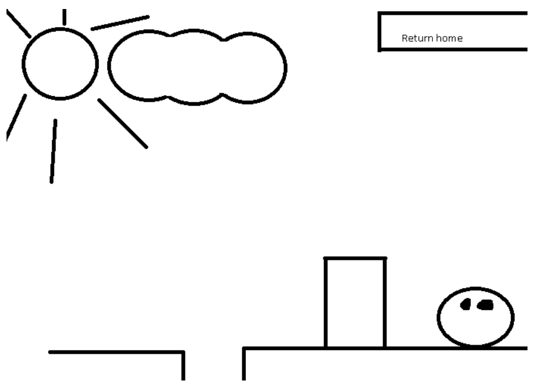


Figure 3

4.Objectives

As the subject's teaching guide indicates [10], the proficiency that must be acquired at the end of the subject is:

- The ability to bring to life individually and to present and defend in front of a university court an original exercise, which consist in a project inside the design and development of a videogame which synthesizes everything taught during the degree.

And the results of the learning process are:

- To plan and implement an original project which synthesizes everything taught during the degree.

- To write a memory in English and to present and defend orally, in front of a university court, an original and professional project inside the video games field which synthesizes everything taught during the degree.

To sum up, one of the main objectives of this project is going to be to learn to create a professional project, planning it realistically and adjust it to the time established to finish this project.

The main objective of this project is the creation of an AI that is able to learn. In order to do so, a game is going to be created where a player needs to train and take care of a monster and then send it to a map to learn how to complete the map. The AI will control the last part with a NN and a genetic algorithm.

Next, a list of objectives is going to be shown that this final degree project is going to meet.

O0: Learn how to develop a professional project.

This is a transversal objective from the first and second one.

O1: Learn to write a proposal of professional level:

Writing a technical proposal of a professional level is a very important feat in order to learn how to describe a project so other people can understand the main idea of any developed project.

O2: Put on practice the skills learned during the degree:

This is the reason why the project final degree is done at the end of every degree and that is why it's an objective.

O3: Investigate about machine learning techniques applied to video games:

Every technique of machine learning hasn't been taught in this degree, so investigating about neural networks, reinforcement learning and genetic algorithms is going to be a very important part of this project.

O4: To increase knowledge about Unity3d in order to apply it to a professional project.

O5: Develop a videogame

A video game must be created in order to prove the capabilities of the AI.

O6: Properly documentation of the work done:

The project needs to be properly documented with the final memory and the presentation in order to present it to the court.

5.Justification

This part of the document is going to explain the relation between the final degree project and the degree, in order to prove how this project is suitable to be considered a final degree project.

Mainly this project is going to use three subjects, VJ1231 Inteligencia Artificial, VJ1227 Motores de juegos and VJ1222 Diseño Conceptual de Videojuegos. In the rest of this section the relation between the subjects will be described:

5.1 VJ1231 INTELIGENCIA ARTIFICIAL

This subject taught how to build an artificial intelligence, making it vital to the existence of this project.

The learning results:

- IR06, IR08, IR15 - Know how to resolve videogame problems using artificial intelligence techniques.
- IR15 – Know how to define the different artificial intelligence techniques, especially for videogames.

5.2 VJ1227 MOTORES DE JUEGOS

This subject taught the inner workings of the video game engines and Unity was used to create a video game.

The learning results:

- E12, IR07, IR14 - Using video game engines to create videogames.

5.3 VJ1224 INGENIERÍA DEL SOFTWARE

This subject taught different agile methodologies to develop videogames.

The learning results:

- IR01, IR02, IR04, IR16, G01 – How to apply agile methodologies to the development of a videogame.

5.4 OTHER SUBJECTS

Because of the amount of programming and other necessities this project requires, these subject are also included:

- Programming I and II: As said before, this project needs programming.
- English: The technical proposal and the final memory need to be written in English.
- Algorithms and data structures: The videogame needs to be as efficient as possible so this subject is needed.

- Advanced interaction techniques: Even though this project doesn't cover the use of peripherals, this subjects taugh a few methods of unsupervised training, which the AI will need.

6.Planning

In this paragraph it is presented the list of tasks, the relation to the objectives, their lengths and a list of deliverables.

6.1 TASKS

In the list below it is presented the list of tasks to develop in this project

Block 1 Documentation of the project

- T1: Realization of a technical proposal where everything is explained clearly
- T2: Write a final memory where it is explained all the work done.
- T3: Create a presentation for the project defence.

Block 2 design and development of a videogame

- T4: Create a Game design document.
- T5: Create the design of the application: interfaces and classes.
- T6: Implement the basic code of the game.
- T7: Evaluate the game testing it with players

Block 3 Artificial Intelligence

- T8: Investigate about machine learning
- T9: Implement the AI algorithms.
- T10: Train the AI.

Task 1 would be the realization of this document. The task 2 would be the final memory that needs to be handled once the project has been done in order to explain how the neural network was done. The task 3 will be a video of the AI learning the map and a few images of the implementation of the AI. Due to the time it takes to learn a neural network it may not have the whole process. The task 5 is where it's planned what classes will the video game need and the task 4 is where the video game will be completely described. the task 6 will be program the character, the levels, the menus and the machines to train the monster. The task 8 and 9 will be where this project is centred. This block needs to take the most time of the project. And the task 7 will be testing the game with real players.

Table 1 Relation between tasks and objectives

Objectives	Tasks
O1	T1, T2, T3, T4, T5
O2	Transversal
O3	T9
O4	T4, T5, T6, T7, T8, T9
O5	T4, T5, T6, T7, T8, T9
O6	T1, T2, T3, T4, T5

Table 2 shows the estimated time for each task as well the time that each task should consume divided in fortnights. It's adapted to the 300 hours the project must take.

Table 2 Planification of the project

Task	Hours	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
T1: Technical Proposal	50	20	20			10					
T2: Memory and Document	45		5	5	5	5	5	5	5	5	5
T3: Presentation	15										15
T4: GDD	12		7	5							
T5: Design	15			5	10						
T6: Basic code	24				17	5			2		
T7: Evaluate the videogame	27									27	
T8: Investigate AI algorithms	25			17		8					
T9: AI algorithms	65					5	28	16	16		
T10: Train	22							12	10		
Time per quarter	300	20	32	32	32	33	33	33	33	32	20

6.2 DELIVERABLES

Table 3 shows when is expected to hand the deliverables and the relation to the tasks and objectives. The time periods are in quarters.

Table 3: Deliverables

Deliverable	Task/objective	Quarter
D1: Technical Proposal	O0, O1, O2; T1	Q1
D2: Final memory	O0, O6, O2; T2	Q3
D3: Presentation	O0, O6, O2; T3	Q3
D4: GDD	O0, O1, O2, O3, O4, O6; T4	Q3
D5: Interfaces and classes	O0, O1, O2, O3, O4, O5, O6; T5, T6	Q3
D6: Videogames	O0, O3, O6; T7, T8	Q3

7. Methodology

The code will be commentated to explain what it does. Also, the project will use GitHub to save the game on the cloud in case something happens.

A methodology will be used to properly program in order to make the code easily readable.

The programming and evaluation of the neural network will be cyclical, which means that each time a significant change happens on the map or the neural network, the former will be trained in order to have an AI that can actually learn the last iteration of the map.

In order to follow the hours indicated in the section 6, the work times will be counted and, after each fortnight, the memory will get updated to reflect the work done, which will be part of the deliverables.

The code from MarI/O [11] will be studied thoroughly in order to properly implement the neural network. Although, instead of having to map directions on the memory, this project can interact directly with the environment putting the objects and the floor on different layers for the vision of the monster and controlling the inputs of the monster using Booleans.

8. Tools

The two main tools that will be used in this project are Microsoft Office and Unity 3D, although another variety of tools can appear in order to make the project easier.

1. Unity3D: Video game engine that has already been defined before. It will be used to create the video game due to the ease of use, and because there is a neural network on the store, which means that there is proof that this can be done.
2. Microsoft Office. To document the project and the presentation, Microsoft office 365 will be used. In particular, Word for the written documents and Powerpoint for the presentations.
3. Blender. Blender is a free 3d modelling and open source software that will be used to model the simple models that can't be found on the internet and also to model the map.
4. Gimp: Gimp is an image editor that will be used to create the simple textures of the game and the images from the technical documents, GDD and presentations. Photoshop was discarded because it's not free software and it started having compatibility issues with the computer that will be used to create the video game.

9 Risks and contingency plan

First of all, the art is not going to be a risk due to the simple shapes of the models in the game.

One of the risks that the game can have is the evaluation time that the neural network takes to learn the map. To solve this, the first thing that will be implemented is the ability to jump for the AI and once the AI learns the map with just the jumping ability, the game will be considered complete. And even if the game is barebones because of this decision, the main objective for this project will be considered complete.

Also the neural network can take time to implement, so the same contingency plan will be used. This means that a modular design approach has been chosen for the game, and the monster abilities will be implemented gradually.

2. Development

This part of the document will explain the fulfilled tasks and the amount of hours they took to be completed.

Block 1 Documentation of the project

- T1: Realization of a technical proposal where everything is explained clearly:
  This task had to do with the technical proposal. It suffered a few corrections and wasn't very well received, mainly because of the way it was redacted. The amount of hours it took was, more or less, 50, as it was planned.
- T2: Write a final memory where it is explained all the work done.
  Due to this task being the redaction of this document, it's difficult to comment on the realization of this part, but the planning done and the extraction of some pieces of the document from the GDD, the proposal and the class design, it's on track.

- T3: Create a presentation for the project defence.
  This task is in the same situation as the one above. It's planned to put images of the game, a video and some text explaining the game.

Block 2 design and development of a videogame

- T4: Create a Game design document.
  This task took 8 hours to be done instead of 12. The document wasn't very extensive, it only had drawings of the interface, a minimal exposition of the game and a comment about the music. It can be downloaded on the link below [13]

- T5: Create the design of the application: interfaces and classes.
  This task took 10 hours to complete due to the creation of a really simple UI and the decision to use Unity UI in the middle of development. As for the class design, it was more or less taken from a webpage that explained Neural Network so it didn't end up being that class design. It can be downloaded in the link below.

- T6: Implement the basic code of the game.
  This task took around 25 hours to complete. Although, the first quarter took 10 hours instead of seventeen, more work hours were put on the 8 and 9 quarter. The basic code of the game encompassed the training on

the machines, the player movement (moving to the left, right, jump), the economy of the game, the mini game screen (which was not planned from the beginning) and equilibrating the whole game. The game will be shown in future chapters of this document.

- T7: Evaluate the game testing it with players
  This is a task that will be explained thoroughly later. This task was done two times. Once in the early stages of the game, to see how the game ran on different pc configurations and later, once the game was finished, to evaluate the game itself. This task took 15 hours to complete, redacting the poll, and taking into account the feedback.

Block 3 Artificial Intelligence

- T8: Investigate about machine learning.
  This task was done in parallel with task 9. It took a lot more time than it was planned and I stopped tracking time halfway this task and the next one, but approximately it took 2 full weeks (8 hours work). I investigated how to make a basic neural network, the pros and cons of using a neural network in any videogame and also studied the genetic algorithm NEAT (neuroevolution of augmenting topologies).

- T9: Implement the AI algorithms.
  As said before, this task was done in parallel with task 8, but this one took more time to complete. It took one month to have a fully working AI plus the time it took to debug it. It was implemented a feedforward neural network, with NEAT as a genetic algorithm. The genes are the connections between each neuron and they are generated randomly, and then later breed with the fittest gene. Once the AI was mount, adding things like the attack for the monster only took 10 minutes on this task.

- T10: Train the AI.
  This took around 20 hours or more, due to having to wait, seeing that the AI was not training correctly, having to fix it, see that the gameplay is not equilibrated, fix it, delete the NN and start again, …

But, even with all the ups and downs, all the tasks have been achieved as it has been described in each of them.

Cases of use

Case of use identifier: CU01

Name of the case of use: Jump

Requirements: Be in the adventure map, learn the ability

 Actors: character

Description: Character jumps

Normal sequence steps:

1- Character jumps
2- Character lands [E1]

Alternative sequence: [E1] Character falls through a hole

Case of use identifier: CU02

Name of the case of use: Training the monster

Requirements: Be in the house map

 Actors: character

Description: Player clicks on the character and puts it on a training machine

Normal sequence steps:

1- Hold right mouse button on a monster
2- Bring him to a machine
3- Wait for the monster to learn the ability

Alternative sequence:

Case of use identifier: CU03

Name of the case of use: Move

Requirements: Being on the adventure map on a platform

Actors: character

Description: Character moves

Normal sequence steps:

1- Character moves [E1]

Alternative sequence: E1, if character is hungry or tired, it won't move.

Case of use identifier: CU04

Name of the case of use: Buying machines

Requirements: Be on the house map

Actors: character

Description: Buy more machines to train the monster

Normal sequence steps

1- Click on the button to buy objects
2- Select the object to buy [E1]
3- Machine appears or changes time.

Alternative sequence: E1, player doesn't have enough money, so it can't buy the machine.

Case of use identifier: CU05

Name of the case of use: Shoot fireballs

Requirements: Be on the house map, learn the ability

Actors: character

Description: The monster shoots fireballs

1- Monster shoots the fireball
2- Fireball hits the dragon and destroys him [E1]

Alternative sequence: [E1] The ball doesn't reach the dragon and ends up destroyed.

Application Design

In this chapter every interface, as well as the design of the videogame will be explained. There aren't a lot of objects in the videogame so all of the can be shown.

INTERFACES

Here it is shown two types of interfaces. A custom created and the one Unity uses programmatically. The last one is the OnGUI() function and was used to speed the development of the game, even though it use is slower due to having to enter on the OnGUI function every frame.

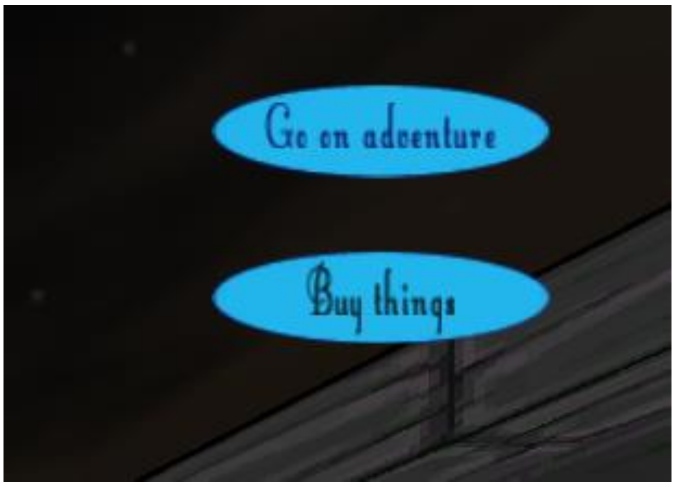


Figure 4

These are the custom buttons. The "go on adventure" takes the player to the adventure map and "buy things" launches the unity UI to buy things. The blue oval was created using Gimp and the font is Little Lord Fontleroy NF from Nick's fonts.

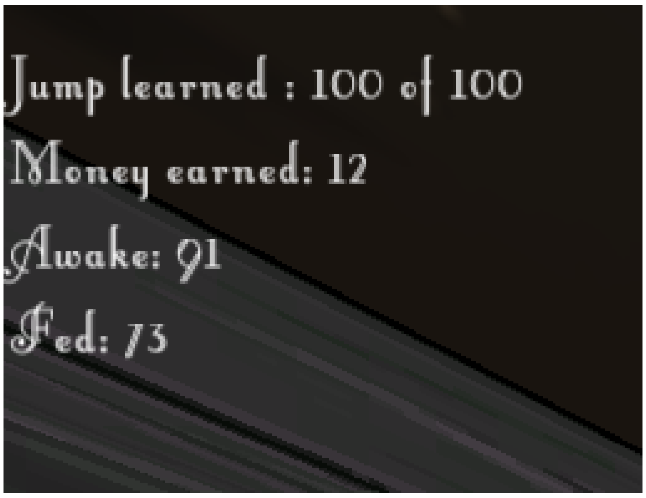

Jump learned : 100 of 100
Money earned: 12
Awake: 91
Fed: 73

Figure 5

This picture shows the information about the monster. If he is hungry or tired, the money and the abilities. Same font used.

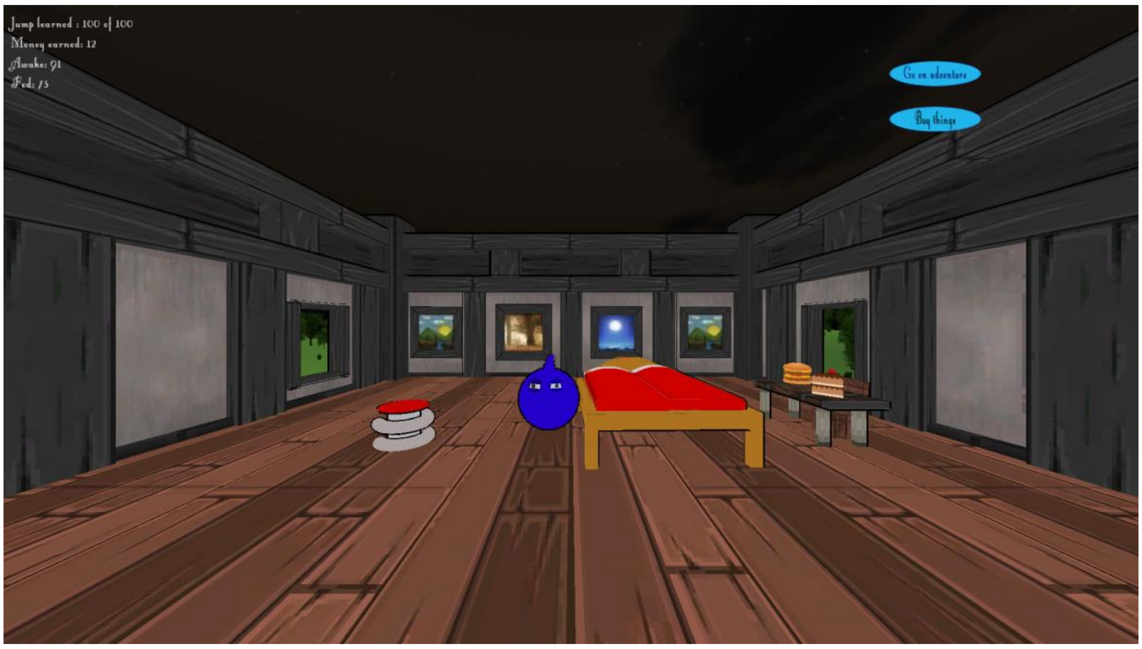

Jump learned : 100 of 100
Money earned: 12
Awake: 91
Fed: 73

Figure 6

The two screens are part of that window which is the house of the monster. In this screen the player can click the buttons, exactly as it has been said before, and drag the monster into the objects in order to interact with them. The buttons and text information are inside a Canvas. The button buy things only activates a variable that is checked during the OnGUI function and shows the shopping window. The go on adventure button calls a function which saves the game and loads the new map. Once the user clicks on the buy things button, the unity UI appears and looks like this:

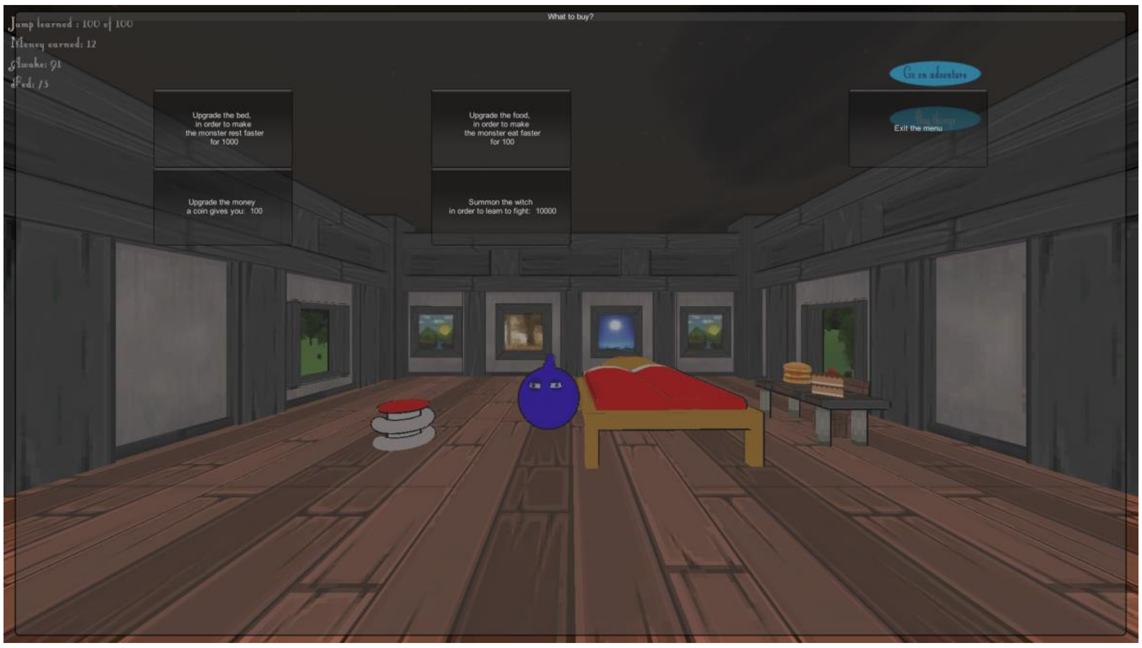

What to buy?
Jump learned : 100 of 100
Money earned: 12
Awake: 91
Fed: 75
Upgrade the bed, in order to make the monster rest faster for 1000
Upgrade the food, in order to make the monster eat faster for 100
Upgrade the money a coin gives you: 100
Summon the witch in order to learn to fight: 10000
Go on adventure
Buy things
Exit the menu

Figure 7

There is a button for each object, raising the level bed, food and coins and buying a witch to learn a fireball technique. This UI was used in order to save time, because the OnGui function is more expensive for the cpu than drawing a custom UI because the OnGUI function gets executed every frame. The button placement and the size of the windows takes into account the current resolution of the screen and sets the buttons accordingly.

As for the UI in the adventure map, we've got more or less the same elements:

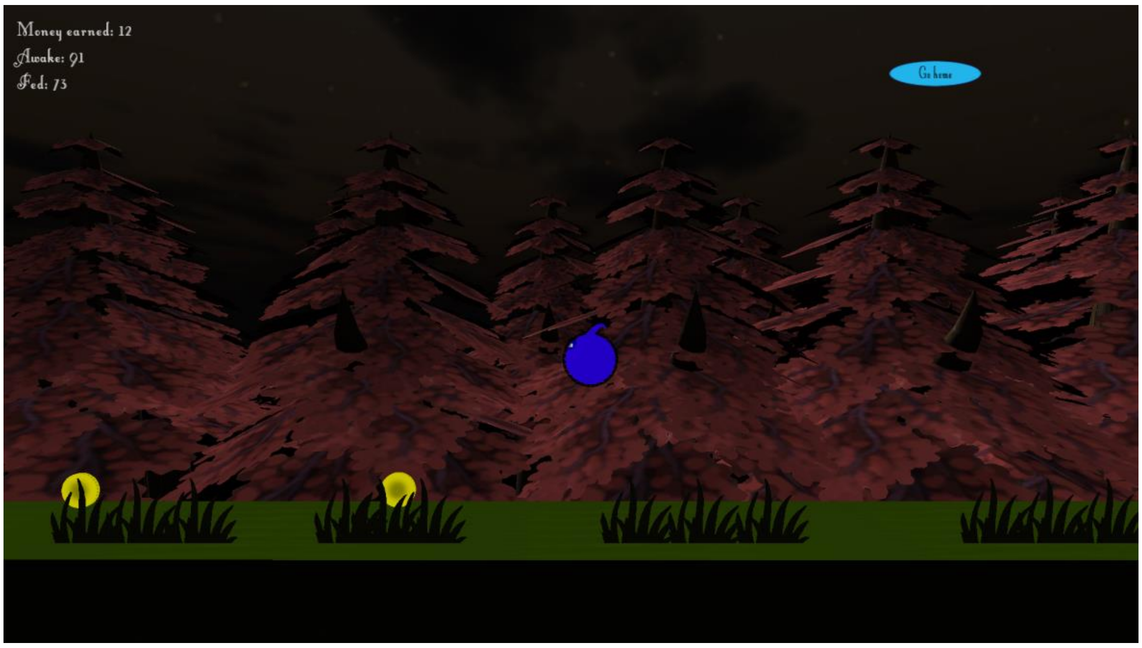
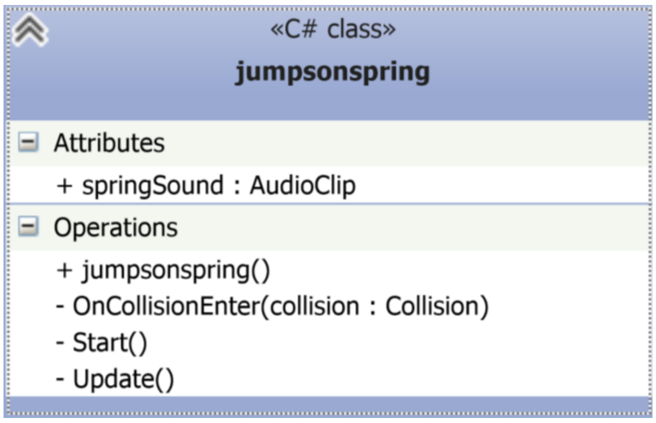


Figure 8

On the top right of the screen, there is a button which takes you back to the house and on the top left of the screen you can see the game info, which consists on the money and how well fed and tired is the monster. Same function as the other ones on the house map.

CLASS DIAGRAM



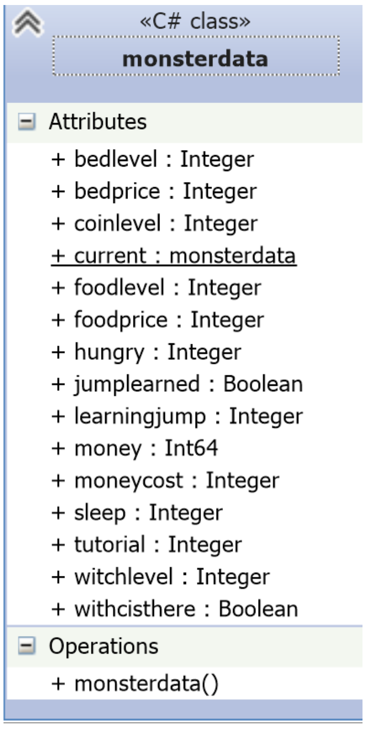
«C# class»
jumpsonspring

Attributes
+ springSound : AudioClip

Operations
+ jumpsonspring()
- OnCollisionEnter(collision : Collision)
- Start()
- Update()

Figure 9

This class makes the monster jump once it touches the spring and plays a sound. The start function and update are empty.


«C# class»
monsterdata

Attributes

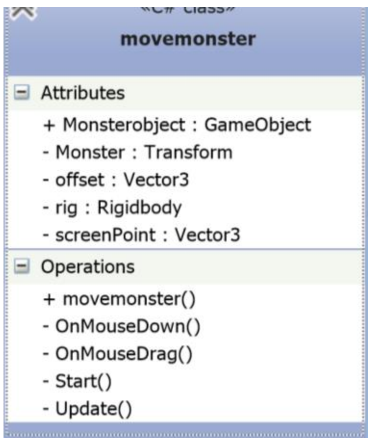
```
+ bedlevel : Integer
+ bedprice : Integer
+ coinlevel : Integer
+ current : monsterdata
+ foodlevel : Integer
+ foodprice : Integer
+ hungry : Integer
+ jumplearned : Boolean
+ learningjump : Integer
+ money : Int64
+ moneycost : Integer
+ sleep : Integer
+ tutorial : Integer
+ witchlevel : Integer
+ withcisthere : Boolean
```

Operations

```
+ monsterdata()
```

Figure 10

This class takes cares of defining all the attributes that are going to be saved to disk. Or, explaining better, this class uses the serialize parameter on the variables in order to be able to create a save game. It doesn’t have any function; it just stores data.



«C# class»

movemonster

Attributes

+ Monsterobject : GameObject
- Monster : Transform
- offset : Vector3
- rig : Rigidbody
- screenPoint : Vector3

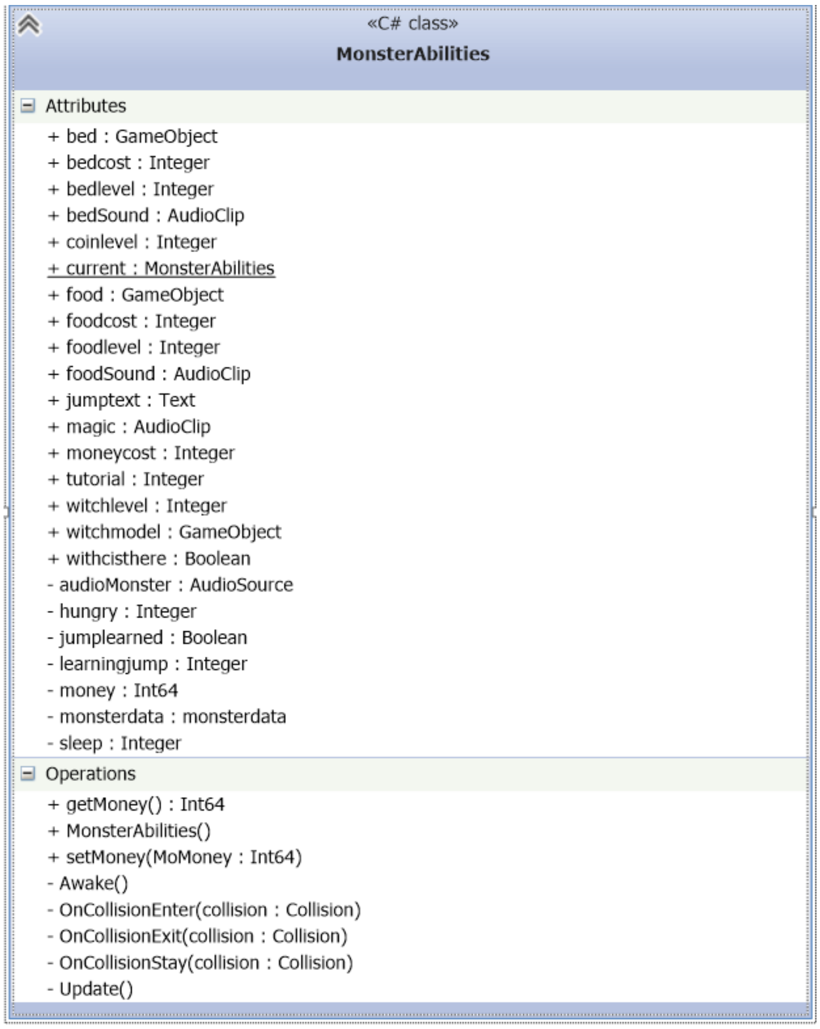
Operations

+ movemonster()
- OnMouseDown()
- OnMouseDrag()
- Start()
- Update()

Figure 11

This class takes care of moving the monster in the house once the players clicks on it.

- Start(): Assigns the rigidbody and the transform of the monster to the rig and Monster variables.
- Update() : Asks the rigidbody if it's sleeping in order to wake it up. If the rigidbody goes to sleep, it stops checking collisions and stops learning abilities.
- OnMouseDown(): Calculates the mouse position once the users clicks the left button.
- OnMouseDrag(): Adds to the position of the monster the actual position of the mouse minus the position the mouse and the difference of positions that the monster and the mouse had when the user clicked it in screen point space.


«C# class»
MonsterAbilities

Attributes

```
+ bed : GameObject
+ bedcost : Integer
+ bedlevel : Integer
+ bedSound : AudioClip
+ coinlevel : Integer
+ current : MonsterAbilities
+ food : GameObject
+ foodcost : Integer
+ foodlevel : Integer
+ foodSound : AudioClip
+ jumptext : Text
+ magic : AudioClip
+ moneycost : Integer
+ tutorial : Integer
+ witchlevel : Integer
+ witchmodel : GameObject
+ withcisthere : Boolean
- audioMonster : AudioSource
- hungry : Integer
- jumplearned : Boolean
- learningjump : Integer
- money : Int64
- monsterdata : monsterdata
- sleep : Integer
```

Operations

```
+ getMoney() : Int64
+ MonsterAbilities()
+ setMoney(MoMoney : Int64)
- Awake()
- OnCollisionEnter(collision : Collision)
- OnCollisionExit(collision : Collision)
- OnCollisionStay(collision : Collision)
- Update()
```

Figure 12

This class loads the save file, and takes care of the collisions and variables used on the savefile of the house scene.

- Start(): Loads the save file.
- Update(): Assigns the variables to the variable current that lets the game serialize monsterdata and save the variables to a file. This file is located in the folder appdata.
- GetMoney and SetMoney: Sets and returns the money variable.
- OnCollisionEnter: Raises the variable that checks if the monster has learnt the jump ability.
- OnCollisionStay: Does the same for the variable that teaches the monster to fight and for the variables that say if the monster is tired or hungry.
- OnCollisionExit: Tells the person who teaches the attack ability to stop the animation.

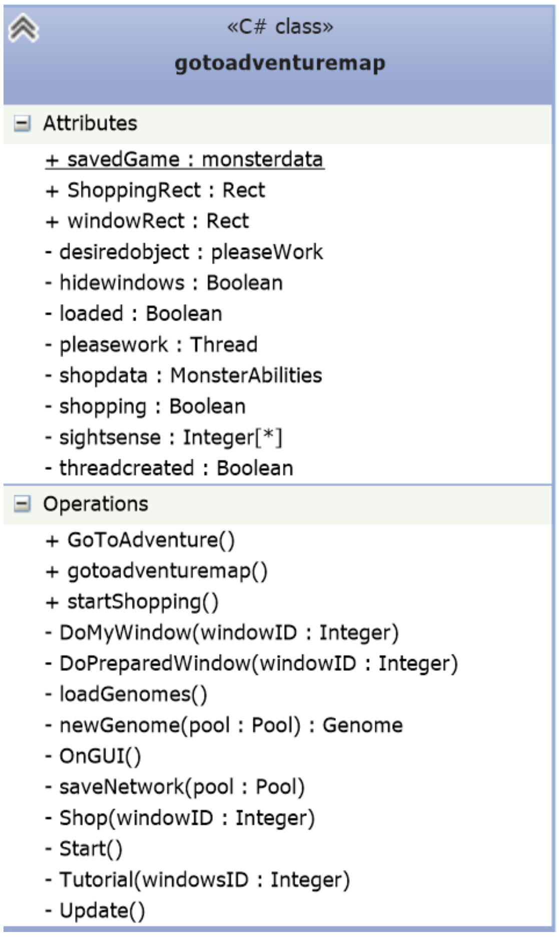
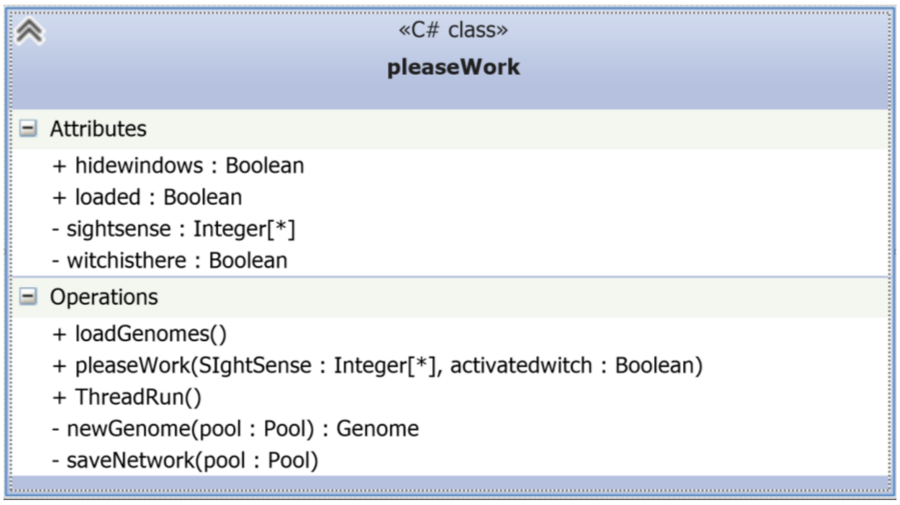

«C# class»
gotoadventuremap
Attributes
+ savedGame : monsterdata
+ ShoppingRect : Rect
+ windowRect : Rect
- desiredobject : pleaseWork
- hidewindows : Boolean
- loaded : Boolean
- pleasework : Thread
- shopdata : MonsterAbilities
- shopping : Boolean
- sightsense : Integer[*]
- threadcreated : Boolean
Operations
+ GoToAdventure()
+ gotoadventuremap()
+ startShopping()
- DoMyWindow(windowID : Integer)
- DoPreparedWindow(windowID : Integer)
- loadGenomes()
- newGenome(pool : Pool) : Genome
- OnGUI()
- saveNetwork(pool : Pool)
- Shop(windowID : Integer)
- Start()
- Tutorial(windowsID : Integer)
- Update()

Figure 13

This class takes care of managing the UI, creating the basic genomes and handling the upgrades of the objects in the house.

- Start: initializes the UI size and if there isn't a neural network created (searching for a xml file contaning it), starts a thread that creates it.
- OnGUI: Checks through boolean values what menu should be displayed(shop, if the monster is ready to go,…).
- Tutorial: OnGUI calls this function when the game is started for the first time, which displays the tutorial.
- Shop: OnGUI calls this function when the player has clicked the buying button. This will display the shop menu.
- DoPreparedWindows: This screen is always drawn if the neural network hasn't been created yet. It tells the player that the monster is not ready to go out.
- DoMyWIndow: Does the opposite of the DoPreparedWindows function. It tells the player that the monster is ready once a neural network has been created.
- SaveNetwork: Saves the network to a xml file.
- GoToAdventure: Saves the game and loads the next map.

The other functions where moved to the thread class, which will be explained now.

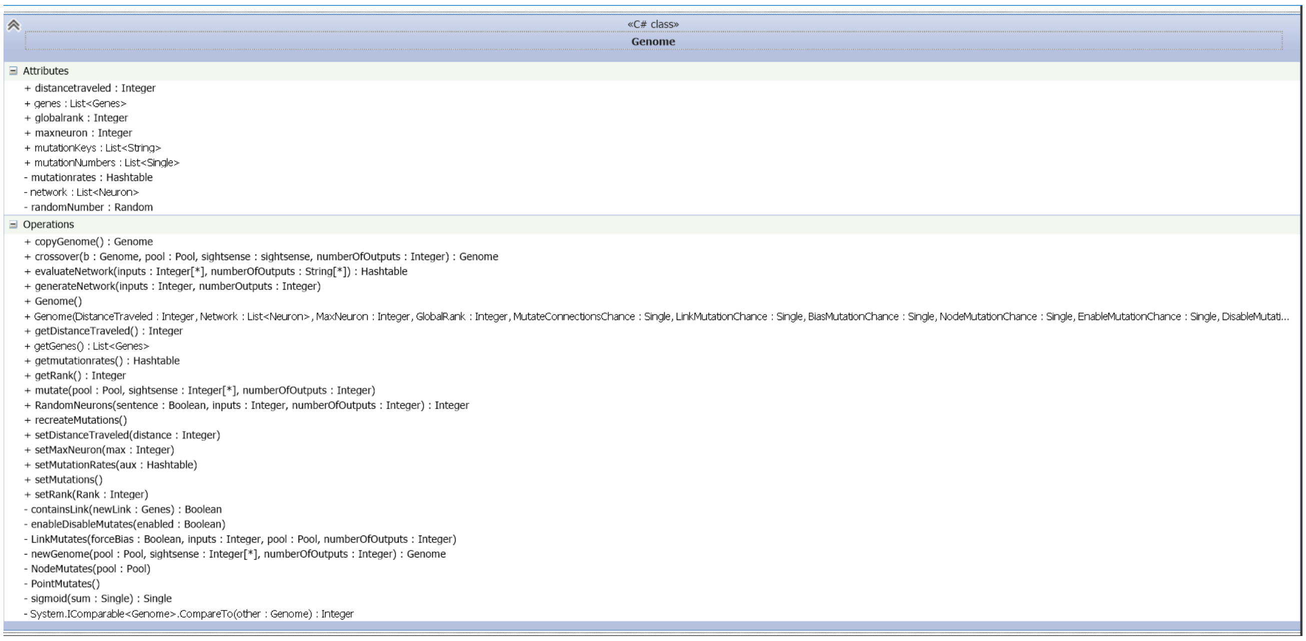

«C# class»

pleaseWork

Attributes
```
+ hidewindows : Boolean
+ loaded : Boolean
- sightsense : Integer[*]
- witchisthere : Boolean
```
Operations
```
+ loadGenomes()
+ pleaseWork(SIghtSense : Integer[*], activatedwitch : Boolean)
+ ThreadRun()
- newGenome(pool : Pool) : Genome
- saveNetwork(pool : Pool)
```

Figure 14

- PleaseWork: This creates the thread object and passes the information that each neuron needs to have.
- ThreadRun: This is the function called from adventuremap. It creates 200 genomes and then saves everything on an xml.
- newGenome(): It creates a new genome and inside that genome it creates a gene with a random mutation.



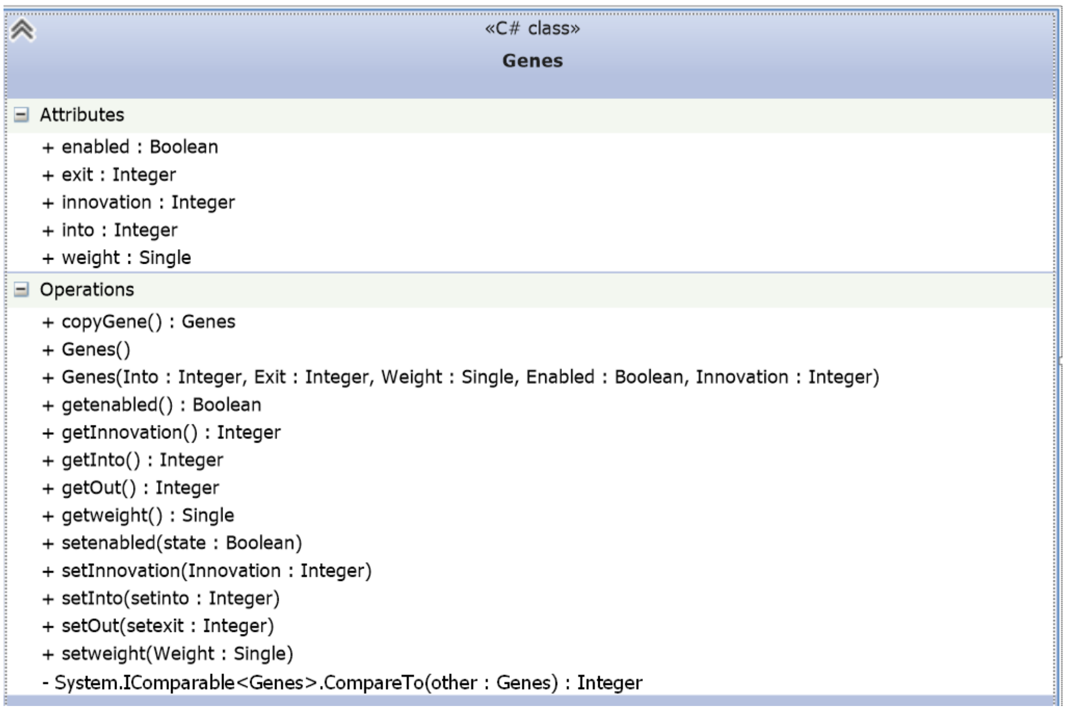
```
«C# class»
Genome

Attributes
  + distancetraveled : Integer
  + genes : List<Genes>
  + globalrank : Integer
  + maxneuron : Integer
  + mutationKeys : List<String>
  + mutationNumbers : List<Single>
  - mutationrates : Hashtable
  - network : List<Neuron>
  - randomNumber : Random
Operations
  + copyGenome() : Genome
  + crossover(b : Genome, pool : Pool, sightsense : sightsense, numberOfOutputs : Integer) : Genome
  + evaluateNetwork(inputs : Integer[*], numberOfOutputs : String[*]) : Hashtable
  + generateNetwork(inputs : Integer, numberOutputs : Integer)
  + Genome()
  + Genome(DistanceTraveled : Integer, Network : List<Neuron>, MaxNeuron : Integer, GlobalRank : Integer, MutateConnectionsChance : Single, LinkMutationChance : Single, BiasMutationChance : Single, NodeMutationChance : Single, EnableMutationChance : Single, DisableMutati...
  + getDistanceTraveled() : Integer
  + getGenes() : List<Genes>
  + getmutationrates() : Hashtable
  + getRank() : Integer
  + mutate(pool : Pool, sightsense : Integer[*], numberOfOutputs : Integer)
  + RandomNeurons(sentence : Boolean, inputs : Integer, numberOfOutputs : Integer) : Integer
  + recreateMutations()
  + setDistanceTraveled(distance : Integer)
  + setMaxNeuron(max : Integer)
  + setMutationRates(aux : Hashtable)
  + setMutations()
  + setRank(Rank : Integer)
  - containsLink(newLink : Genes) : Boolean
  - enableDisableMutates(enabled : Boolean)
  - LinkMutates(forceBias : Boolean, inputs : Integer, pool : Pool, numberOfOutputs : Integer)
  - newGenome(pool : Pool, sightsense : Integer[*], numberOfOutputs : Integer) : Genome
  - NodeMutates(pool : Pool)
  - PointMutates()
  - sigmoid(sum : Single) : Single
  - System.IComparable<Genome>.CompareTo(other : Genome) : Integer
```

Figure 15

Here we have the genome class. It takes care of managing everything that has to do with genomes.

- Genome: creates a genome with the information provided. This information is the distance travelled by the monster using this genome, the neural network (which is composed of what the monster sees in the world, an array of 49 positions with a 0 when there isn't anything in that position and a 1 when there is something, hidden layers that can be generated, and the inputs that these neurons must fire), the rank that this genome has between all the genome (they are classified by the distance the monster was able to travel), the list of genes, which are the connections between each neuron in the neural network and a Hashtable with the probability of any type of mutations for the genes.
- Mutate: This is called from the Thread that was already explained. This function takes care of mutating each gene from the genome. Depending on the value of a random number it will call any of these functions: pointmutates, linkmutates, nodemutates and enabledisablemutates.
- PointMutates: Takes care of making a mutation on the weight of each gene.
- LinkMutates: Grabs 2 random neuron and connects the one which is in a lower position with the other one or depending on a random value, with the last neuron on the network, then assigns a new innovation for it (which is the number of genes created) and a weight.
- NodeMutates: this one grabs a random gene and if it's activated, created two other genes from the information of this one, gives the last neuron as the exit to one gene and the entrance to the other gene and adds them to the list.

- enableDisableMutates: this one is given if we want to have an enabled or disabled gene. Then gets a random gene from the ones the genome has and that is disabled or enabled and sets it to the opposite of which it was.
- GenerateNetwork: This creates the neural network and the genes connections.
- EvaluateNetwork: This evaluates the network in order to find what button does the monster presses (left, right, jump or shoot, or any combination of them).
- Crossover: this function creates a new genome from crossing all the genes from two genomes.



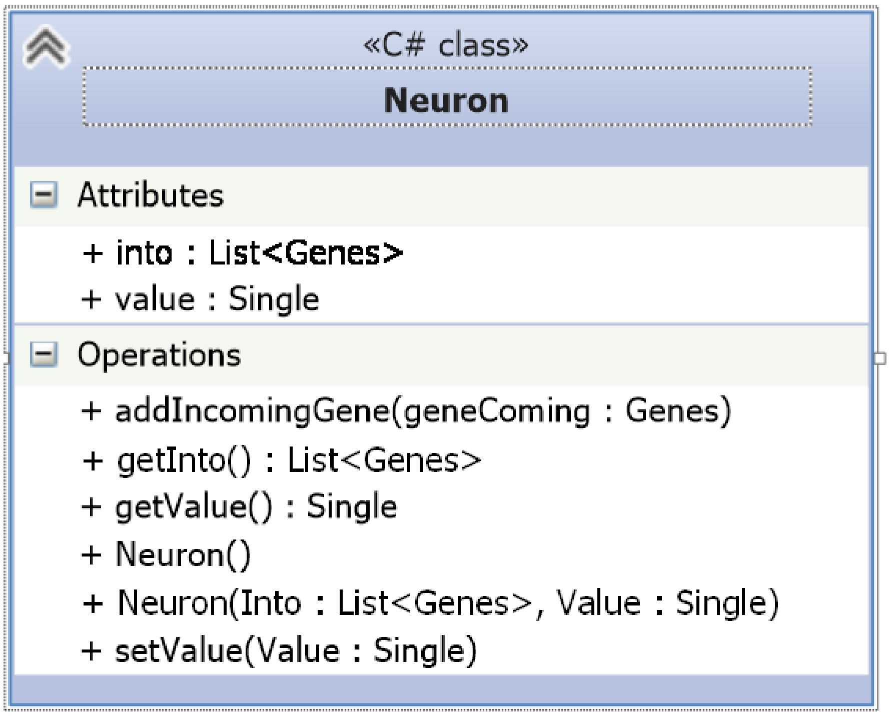
```
«C# class»
Genes

Attributes
  + enabled : Boolean
  + exit : Integer
  + innovation : Integer
  + into : Integer
  + weight : Single
Operations
  + copyGene() : Genes
  + Genes()
  + Genes(Into : Integer, Exit : Integer, Weight : Single, Enabled : Boolean, Innovation : Integer)
  + getenabled() : Boolean
  + getInnovation() : Integer
  + getInto() : Integer
  + getOut() : Integer
  + getweight() : Single
  + setenabled(state : Boolean)
  + setInnovation(Innovation : Integer)
  + setInto(setinto : Integer)
  + setOut(setexit : Integer)
  + setweight(Weight : Single)
  - System.IComparable<Genes>.CompareTo(other : Genes) : Integer
```

Figure 16

This class handles everything that deals with genes.

- Genes: this fills the gene with the information. Into, which is the number of the neuron this connection is coming from, exit, which is the number of the neuron this connection goes to, innovation, which is the number of the gene created, enabled, if it is enabled or not and weight, which is the weight this connection has, if it is too low, it's deactivated.

Everything else is just getters and setters.

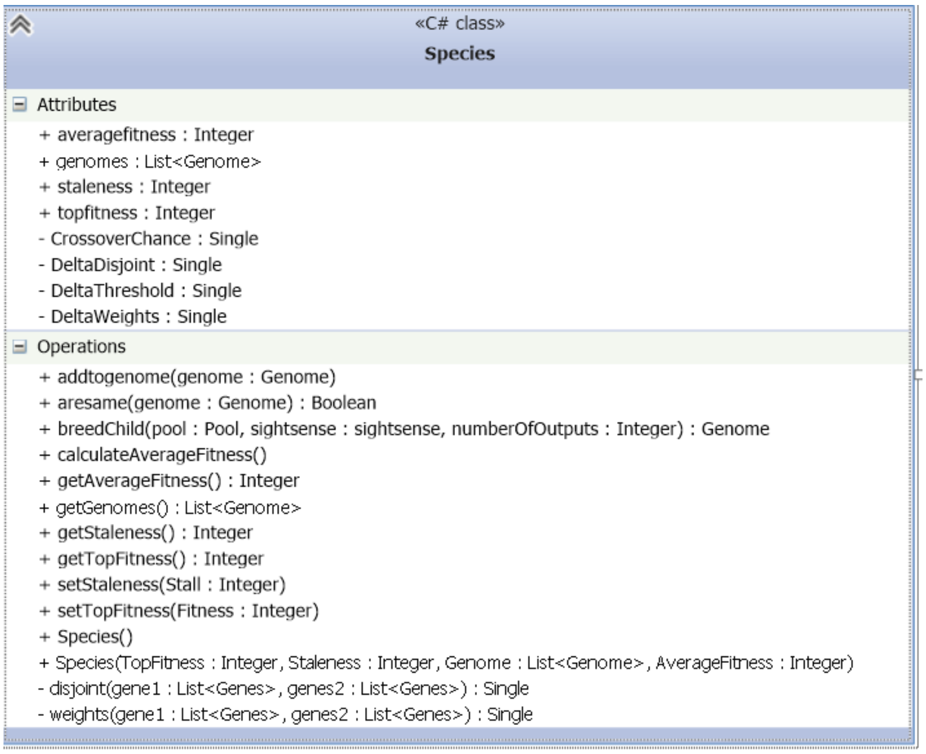


```
+ into : List<Genes>
+ value : Single
```

```
+ addIncomingGene(geneComing : Genes)
+ getInto() : List<Genes>
+ getValue() : Single
+ Neuron()
+ Neuron(Into : List<Genes>, Value : Single)
+ setValue(Value : Single)
```

Figure 17

This is the class that handles each neuron, it only contains the value of the neuron and which genes have this neuron as an exit.



«C# class»

Species

Attributes

```
+ averagefitness : Integer
+ genomes : List<Genome>
+ staleness : Integer
+ topfitness : Integer
- CrossoverChance : Single
- DeltaDisjoint : Single
- DeltaThreshold : Single
- DeltaWeights : Single
```

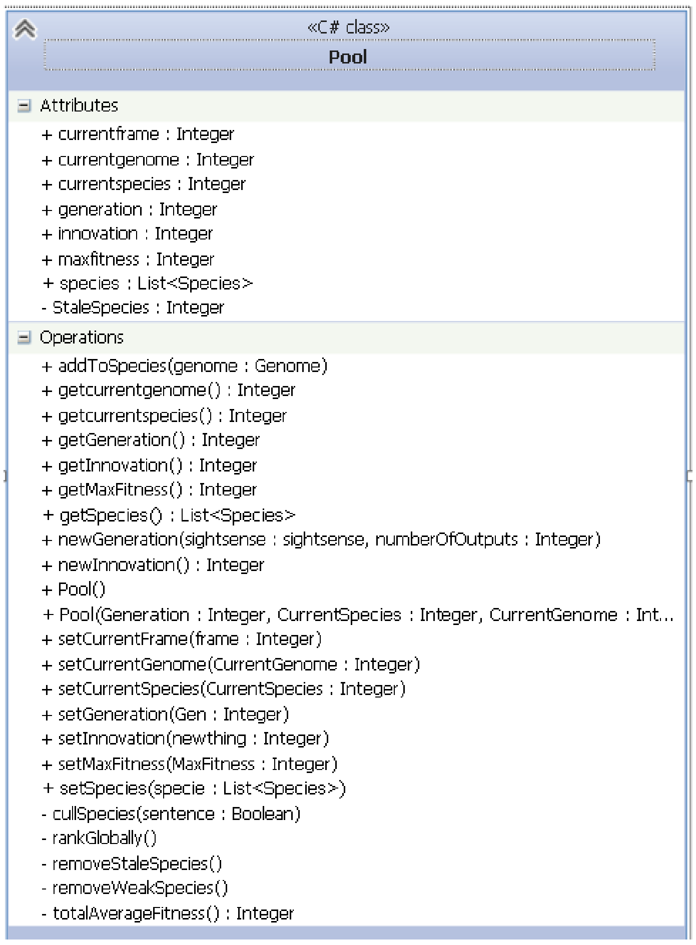
Operations

```
+ addtogenome(genome : Genome)
+ aresame(genome : Genome) : Boolean
+ breedChild(pool : Pool, sightsense : sightsense, numberOfOutputs : Integer) : Genome
+ calculateAverageFitness()
+ getAverageFitness() : Integer
+ getGenomes() : List<Genome>
+ getStaleness() : Integer
+ getTopFitness() : Integer
+ setStaleness(Stall : Integer)
+ setTopFitness(Fitness : Integer)
+ Species()
+ Species(TopFitness : Integer, Staleness : Integer, Genome : List<Genome>, AverageFitness : Integer)
- disjoint(gene1 : List<Genes>, genes2 : List<Genes>) : Single
- weights(gene1 : List<Genes>, genes2 : List<Genes>) : Single
```

Figure 18

This class has all the genomes of a specie. It's formed by the list of genomes, staleness, a number which tells us when it's needed to erase all the genomes except the one which made the monster run farthest. There is also the longest space the monster has travelled in this specie, the average space, and global variables to use when it's needed to see if two genomes are the same and to cross two genomes.

- Aresame: this function finds if two genomes are the same, calling the disjoint and weights function, adding the result and then finding if the result is under the delta threshold number.
- Disjoint. this function tells the quantity of genes that are different between two genomes.
- Weights: this divides the sum of the difference between the weights of each gene and the divides it by the quantity of genes which should give the coincident genes.
- Breedchild: depending on a random number it crosses two random genomes or copies one random genome on to another and the mutates that genome.



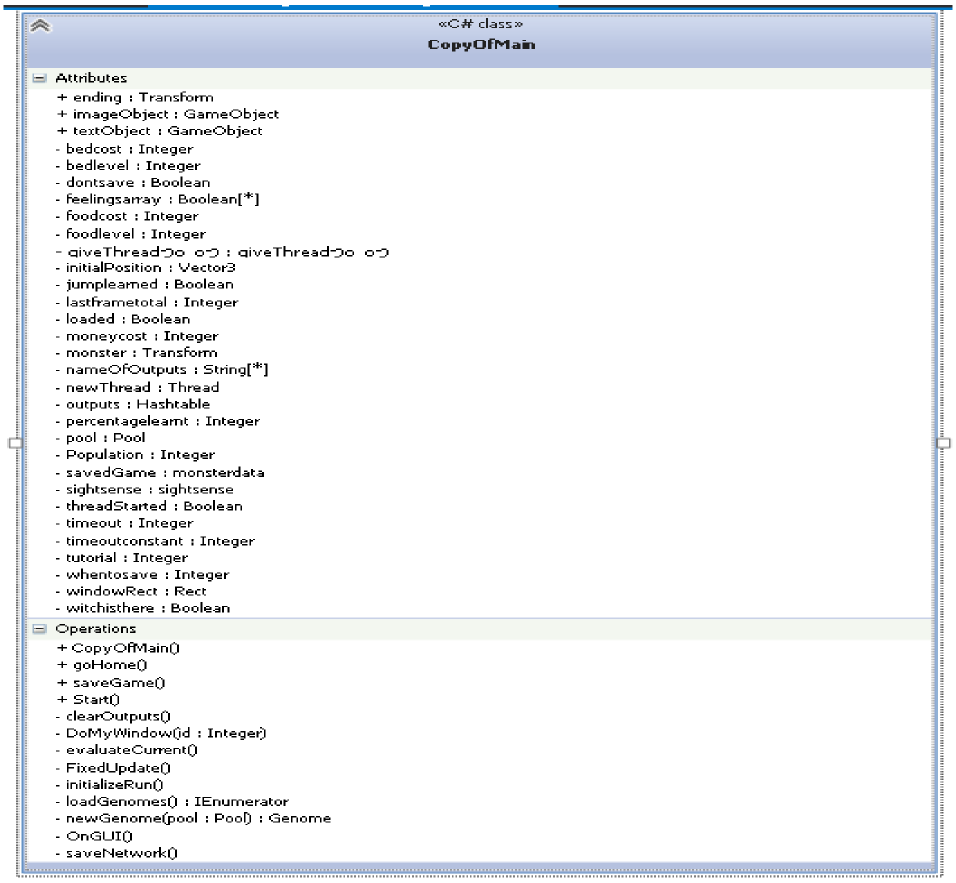
```
«C# class»
Pool

Attributes
+ currentframe : Integer
+ currentgenome : Integer
+ currentspecies : Integer
+ generation : Integer
+ innovation : Integer
+ maxfitness : Integer
+ species : List<Species>
- StaleSpecies : Integer

Operations
+ addToSpecies(genome : Genome)
+ getcurrentgenome() : Integer
+ getcurrentspecies() : Integer
+ getGeneration() : Integer
+ getInnovation() : Integer
+ getMaxFitness() : Integer
+ getSpecies() : List<Species>
+ newGeneration(sightsense : sightsense, numberOfOutputs : Integer)
+ newInnovation() : Integer
+ Pool()
+ Pool(Generation : Integer, CurrentSpecies : Integer, CurrentGenome : Int...
+ setCurrentFrame(frame : Integer)
+ setCurrentGenome(CurrentGenome : Integer)
+ setCurrentSpecies(CurrentSpecies : Integer)
+ setGeneration(Gen : Integer)
+ setInnovation(newthing : Integer)
+ setMaxFitness(MaxFitness : Integer)
+ setSpecies(specie : List<Species>)
- cullSpecies(sentence : Boolean)
- rankGlobally()
- removeStaleSpecies()
- removeWeakSpecies()
- totalAverageFitness() : Integer
```

Figure 19

The pool takes is the class that tells which genome is the monster using, in which specie is at the moment, which generation, the number of genes (innovation), the maximum amount of space the monster has been able to travel, and a list of species.

- newGeneration: this function creates a new generation. First of all, it removes the first half of genomes that each specie has with the cullspecies function, then it ranks them depending on the distance the monster has been able to travel, with the rankGlobally function, then depending on the staleness, it leaves only the genome that travelled farthest, removes a few or leaves the list intact with removeStaleSpecies, then the genomes are ranked again, and the average that has been travelled with these genomes of each specie is calculated. After that it removes the genomes that couldn't travel from each specie until a calculated distance, using removeWeakSpecies, then it calculates the total average fitness, with totalAverageFitness, which is the average fitness of each specie, then leaves only one genome on each specie again using cullSpecies, which is supposed to be the best one and then breeds 200 genomes more between these genomes and creates a new specie with these genomes.

All these were the classes that dealt with the neural network and the genetic algorithms. Now it is going to be explained the classes that deal with the gameplay on the adventure map.



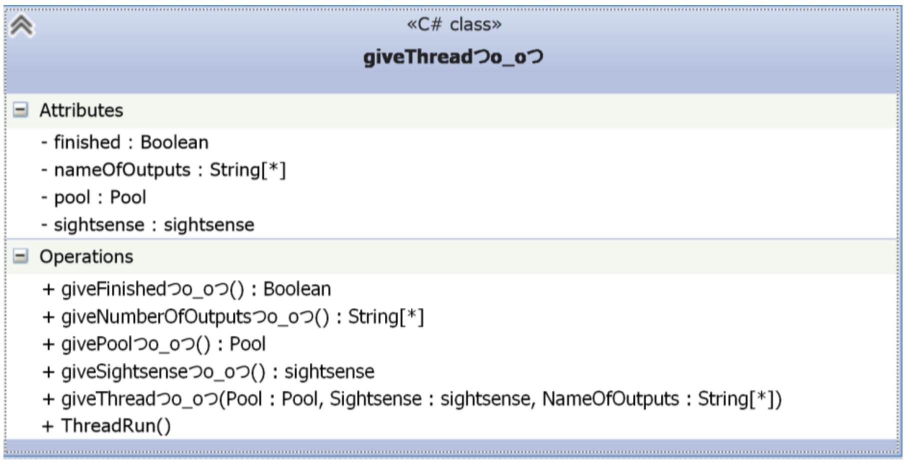
```
«C# class»
CopyOfMain

Attributes
+ ending : Transform
+ imageObject : GameObject
+ textObject : GameObject
- bedcost : Integer
- bedlevel : Integer
- dontsave : Boolean
- feelingsarray : Boolean[*]
- foodcost : Integer
- foodlevel : Integer
- giveThread؟o o؟ : giveThread؟o o؟
- initialPosition : Vector3
- jumplearned : Boolean
- lastframetotal : Integer
- loaded : Boolean
- moneycost : Integer
- monster : Transform
- nameOfOutputs : String[*]
- newThread : Thread
- outputs : Hashtable
- percentagelearnt : Integer
- pool : Pool
- Population : Integer
- savedGame : monsterdata
- sightsense : sightsense
- threadStarted : Boolean
- timeout : Integer
- timeoutconstant : Integer
- tutorial : Integer
- whentosave : Integer
- windowRect : Rect
- witchisthere : Boolean

Operations
+ CopyOfMain()
+ goHome()
+ saveGame()
+ Start()
- clearOutputs()
- DoMyWindow(id : Integer)
- evaluateCurrent()
- FixedUpdate()
- initializeRun()
- loadGenomes() : IEnumerator
- newGenome(pool : Pool) : Genome
- OnGUI()
- saveNetwork()
```

Figure 20

- Start: Creates the array of outputs (which is the keys that the AI presses, left, right, jump and attack), stores the sightsense class, loads the save file to know what things the monster knows, the money it has, etc. and loads the neural network.
- OnGUI: This one only activates once the monster gets to the ending and display a congratulations message for a few seconds.
- goHome: Sends the player to the house scene
- FixedUpdate: calls evaluateCurrent which is the function that calls to evaluateNetwork in order to know which buttons does the monster press, checks if the monster has moved and if it hasn't moved for 5

seconds, the travelled distance is saved, the monster is rewarded money if it gets to the ending and gets to the next genome. If all the genomes have been used, a giveThreadↃo_oↃ is started in order to create a new specie while a minigame where the user has to press the Z button to get coins appears on screen. After finishing it initializes a new run with the new genomes. Also, if the sum of feed plus awake divided by 2 is less than fifty, it will call to initializerun depending on a random number, which will make the monster go back to the beginning without changing genomes, so it will force the player to go back home to take care of the monster.

- InitializeRun: puts the monster on the initial position, generates the network with what the monsters see at that moment, and evaluates the network.
- evaluateCurrent: As said before, evaluates the network to know which buttons need to be pressed.



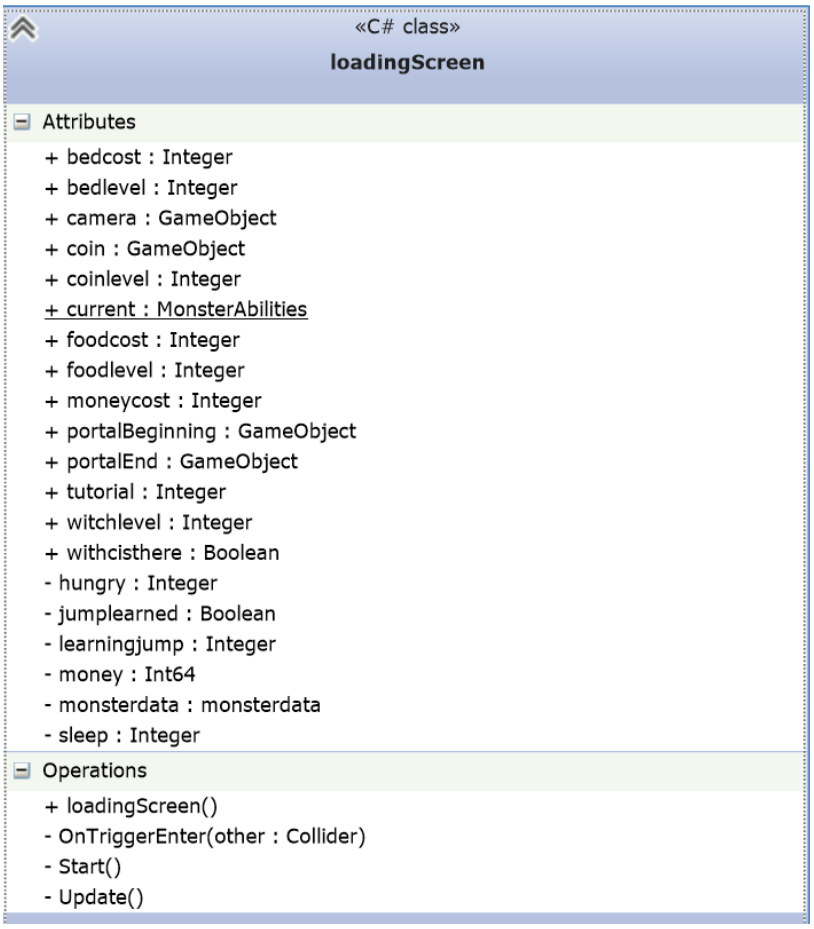
«C# class»
giveThreadↃo_oↃ

Attributes
- finished : Boolean
- nameOfOutputs : String[*]
- pool : Pool
- sightsense : sightsense

Operations
+ giveFinishedↃo_oↃ() : Boolean
+ giveNumberOfOutputsↃo_oↃ() : String[*]
+ givePoolↃo_oↃ() : Pool
+ giveSightsenseↃo_oↃ() : sightsense
+ giveThreadↃo_oↃ(Pool : Pool, Sightsense : sightsense, NameOfOutputs : String[*])
+ ThreadRun()

Figure 21

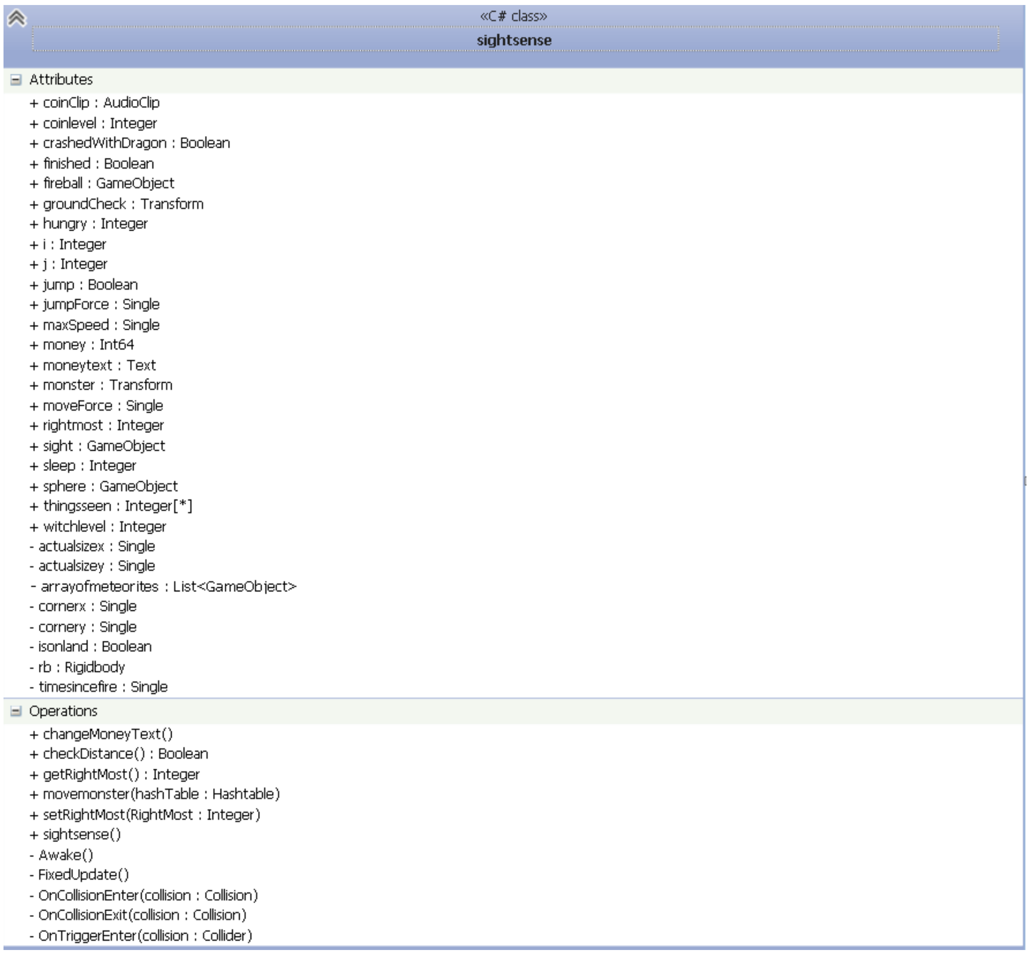
This is the thread that takes care of creating a new specie. And that is everything it does, it runs pool.newGeneration on ThreadRun.



```
«C# class»
loadingScreen

Attributes
  + bedcost : Integer
  + bedlevel : Integer
  + camera : GameObject
  + coin : GameObject
  + coinlevel : Integer
  + current : MonsterAbilities
  + foodcost : Integer
  + foodlevel : Integer
  + moneycost : Integer
  + portalBeginning : GameObject
  + portalEnd : GameObject
  + tutorial : Integer
  + witchlevel : Integer
  + withcisthere : Boolean
  - hungry : Integer
  - jumplearned : Boolean
  - learningjump : Integer
  - money : Int64
  - monsterdata : monsterdata
  - sleep : Integer
Operations
  + loadingScreen()
  - OnTriggerEnter(other : Collider)
  - Start()
  - Update()
```

Figure 22

This is the class function that takes care of the minigame. It just loads the savefile, checks if the player pressed the Z button, then moves forward the monster, and, if the player got a coin, it’s added to the money of the monster.


```
«C# class»
sightsense

Attributes
+ coinClip : AudioClip
+ coinlevel : Integer
+ crashedWithDragon : Boolean
+ finished : Boolean
+ fireball : GameObject
+ groundCheck : Transform
+ hungry : Integer
+ i : Integer
+ j : Integer
+ jump : Boolean
+ jumpForce : Single
+ maxSpeed : Single
+ money : Int64
+ moneytext : Text
+ monster : Transform
+ moveForce : Single
+ rightmost : Integer
+ sight : GameObject
+ sleep : Integer
+ sphere : GameObject
+ thingsseen : Integer[*]
+ witchlevel : Integer
- actualsizex : Single
- actualsizey : Single
- arrayofmeteorites : List<GameObject>
- cornerx : Single
- cornery : Single
- isonland : Boolean
- rb : Rigidbody
- timesincefire : Single
Operations
+ changeMoneyText()
+ checkDistance() : Boolean
+ getRightMost() : Integer
+ movemonster(hashTable : Hashtable)
+ setRightMost(RightMost : Integer)
+ sightsense()
- Awake()
- FixedUpdate()
- OnCollisionEnter(collision : Collision)
- OnCollisionExit(collision : Collision)
- OnTriggerEnter(collision : Collider)
```
Figure 23

- Awake: Here, instead of start, an awake function is used in order to be executed before the Start function of CopyOfMain. Everything this does is set up the array that holds the things the monster sees.
- Checkdistance: checks if the monster has gone farther than the farthest position he has gone in that genome.
- FixedUpdate: It fills the sight array with the things the monster sees, decrease the time and feed variables, and checks if the attacks that the monster launches need to be destroyed.
- OnCollisionEnter: Checks if the monster is on the floor or if it collided with a dragon that guard the ending. If the monster crashes with the dragon, it changes to the next genome and restarts.
- OnCollisionExit: Checks if the monster jumped.
- OnTriggerEnter: Checks if the monster got a coin or if it got to the end.
- Movemonster: moves the monster depending on the result of the evaluation of the neural network

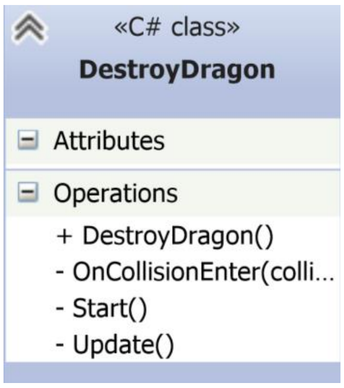

«C# class»
DestroyDragon
Attributes
Operations
+ DestroyDragon()
- OnCollisionEnter(colli...
- Start()
- Update()

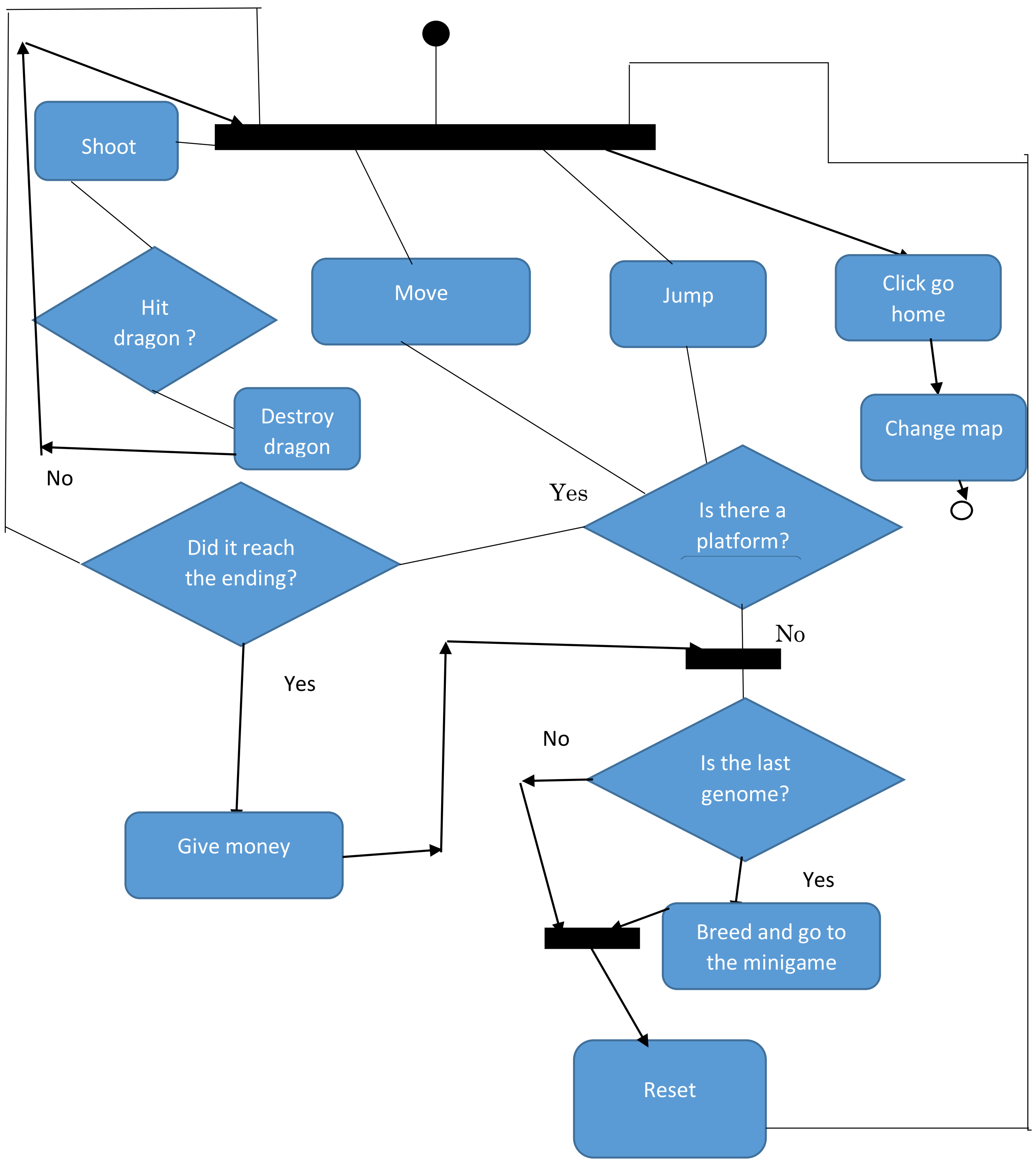
Figure 24

This checks if any attack hit the dragon and then destroys the dragon and the attack.

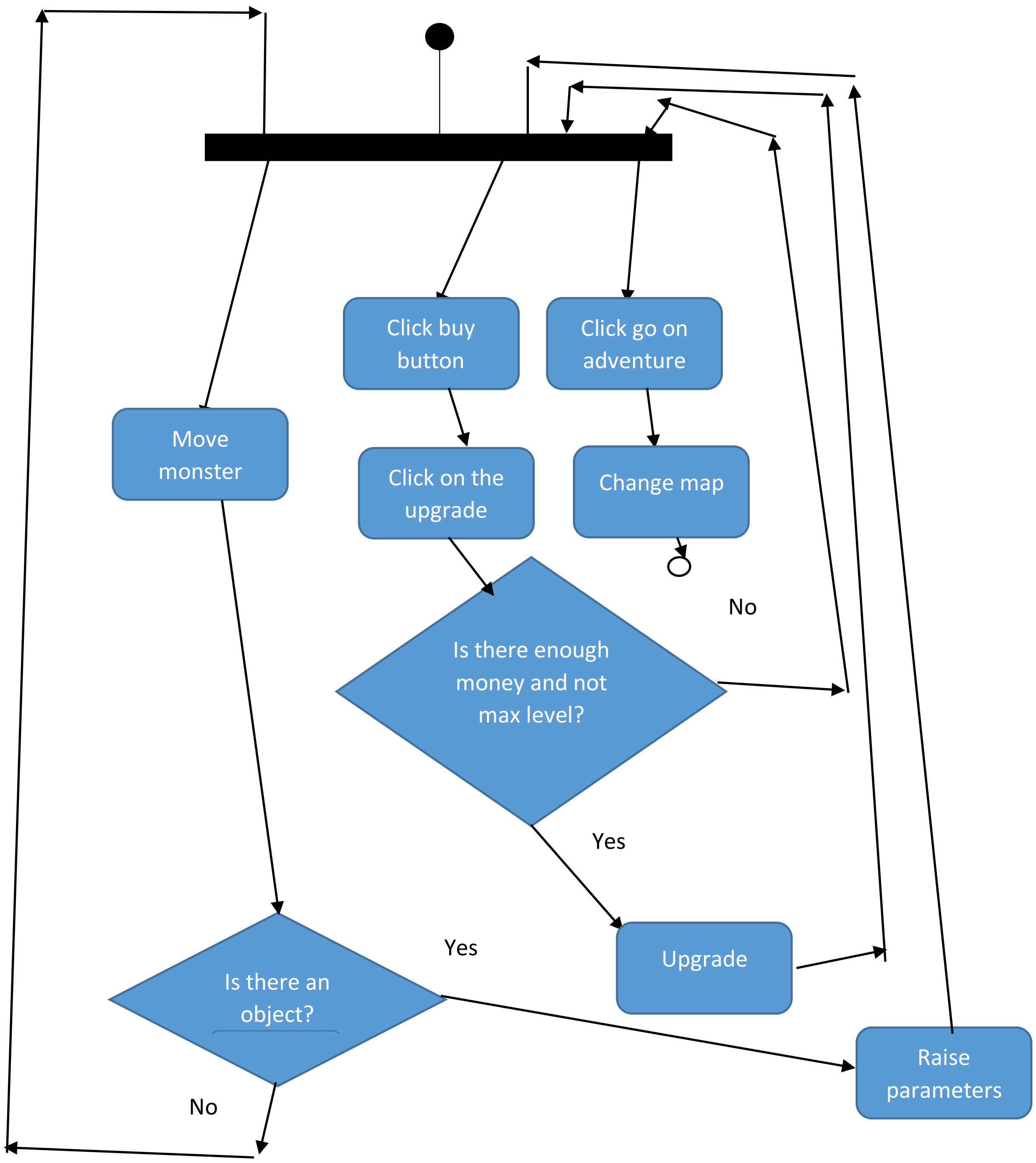
3. Results

Activity diagram

This is the activity diagram that the monster uses to move through the adventure map.



The next one explains the player actions in the house:



Implementation

In this section some code will be shown of the application and a brief explanation. Not everything is going to be shown due to the huge amount of it.

First of all, here's the code of how to load a savefile.

```
void Awake () {
        monsterdata.current = new monsterdata();
        audioMonster = GetComponent<AudioSource>();
        if (File.Exists(Application.persistentDataPath + "/savedGames.gd"))
        {
            BinaryFormatter bf = new BinaryFormatter();
            FileStream file = File.Open(Application.persistentDataPath +
"/savedGames.gd", FileMode.Open);
            monsterdata = (monsterdata)bf.Deserialize(file);
            file.Close();
            jumplearned = monsterdata.jumplearned;
            learningjump = monsterdata.learningjump;
            money = monsterdata.money;
            sleep = monsterdata.sleep;
            hungry = monsterdata.hungry;
            bedlevel = monsterdata.bedlevel;
            foodlevel = monsterdata.foodlevel;
            bedcost = monsterdata.bedprice;
            foodcost = monsterdata.foodprice;
            moneycost = monsterdata.moneycost;
            coinlevel = monsterdata.coinlevel;
            withcisthere = monsterdata.withcisthere;
            witchlevel = monsterdata.witchlevel;
            tutorial = monsterdata.tutorial;
            }
    }
```

As it can be seen it's just opening a file, and get all the information from the current variable. The same but the other way if anyone wants to save the game.

And here is the creation of the genomes.

```
public void ThreadRun()
    {
        Pool pool;
        if (witchisthere)
        {
            pool = new Pool(0, 0, 0, 0, 0, 4, new List<Species>());

        }
        else
        {
            pool = new Pool(0, 0, 0, 0, 0, 3, new List<Species>());

        }
        int i = 0;
        while (i < 200)
        {

                Genome genome = newGenome(pool);
                pool.addToSpecies(genome);
                i++;


        }
        saveNetwork(pool);
        loaded = true;
        hidewindows = false;
    }
```

It just goes through the number of the desired population, creates a genome and adds it to the specie.

Next and last, it will be explained the interesting code, CopyOfMain, the fixedupdate, which is where everything happens, even if it as already explained before.

```
void FixedUpdate () {
        //print(pool.getspecies().Count);
        if (loaded && threadStarted == false )
        {
            Genome genome =
pool.getSpecies()[pool.getcurrentspecies()].getGenomes()[pool.getcurrentgenome()]
;
            if (Time.frameCount % 5 == 0)
            {
                evaluateCurrent();
            }
            sightsense.movemonster(outputs);
            if (sightsense.checkDistance())
            {
                timeout = timeoutconstant;
            }

            if (dontsave)
                initializeRun();
            timeout--;
            float timeoutBonus = (Time.frameCount - lastframetotal) / 4;.
```

This part is fairly simple, if the game has been loaded and there is no thread running, the current genome gets selected and each 5 frames the monster presses a combination of buttons by evaluating the network. Then the distance is checked and if the player has moved, timeout stays at 5 seconds, if not it jumps and timeout will decrease. That condition on the variable dontsave only gets executed when the monster needs to rest or to eat. Then timeout decrease.

```
if (timeout + timeoutBonus <= 0 || monster.position.y < ending.position.y ||
sightsense.crashedWithDragon || (sightsense.finished && Time.time % 5 == 0))
            {
                sightsense.crashedWithDragon = false;
                int fitness = (int) sightsense.getRightMost() - (int)
initialPosition.x;
                if (sightsense.getRightMost() > ending.position.x)
                {
                    //game has ended, so give a better fitness than others
                    fitness += 1000;
                    sightsense.money += 1000 * (int)Mathf.Pow(2,
sightsense.coinlevel);

                }
                if (fitness == 0)
                {
                    fitness = -1;

                }
                sightsense.finished = false;
                    genome.setDistanceTraveled(fitness);
                    if (fitness > pool.getMaxFitness())
                    {
                        pool.setMaxFitness(fitness);
                        //saveNetwork();
                    }
```

After that, this condition will only happen if the player has reached the ending, has gotten stuck or died. Then it saves the space the monster has run and if it was in the ending, coins get added to the player and a score of 1000 get added to the player fitness(the distance travelled) in order to differentiate from the other genomes better. If the monster didn't move the space gets changed to -1 because the current genome gets calculated by which is the first genome which has 0 as fitness. Then the distance travelled gets assigned and it is checked if this distance was the biggest one in order to assign it to the max distance.

```
pool.setCurrentGenome(0);
                    pool.setCurrentSpecies(0);
while(pool.getSpecies()[pool.getcurrentspecies()].getGenomes()[pool.getcurrentgen
ome()].getDistanceTraveled() != 0)
                {
                    pool.setCurrentGenome(pool.getcurrentgenome() + 1);
                    if (pool.getcurrentgenome() >=
pool.getSpecies()[pool.getcurrentspecies()].getGenomes().Count)
                    {
                        pool.setCurrentGenome(0);
                        pool.setCurrentSpecies(pool.getcurrentspecies() + 1);
                        if (pool.getcurrentspecies() >= pool.getSpecies().Count)
                        {
                            saveGame();
UnityEngine.Object.DontDestroyOnLoad(this.gameObject);
                            SceneManager.LoadScene("Loading Screen");
                            giveThreadↄo_oↄ = new giveThreadↄo_oↄ(pool,
sightsense, nameOfOutputs);
                            newThread = new Thread(new ThreadStart(giveThreadↄ
o_oↄ.ThreadRun));
                            threadStarted = true;
                            try
                            {
                                newThread.Start();
                                break;
                            }
                            catch (ThreadStateException e)
                            {
                                Console.WriteLine(e);  // Display text of
exception
                            }
                            catch (ThreadInterruptedException e)
                            {
                                Console.WriteLine(e);  // This exception means
that the thread
                                                       // was interrupted during
a Wait
                            }
                        }
                    }
                }
```

This function sets the currentgenome and species to 0 in order to start the search to find which genome is the first that hasn't been tested. If all are used, then the game is saved, the minigame is started and the thread that creates a new species gets called. Threads aren't compatible with Unity so every function that comes from Unity can't be used on the thread. And the reason why a Thread needed to be used was because putting the creation of a specie on the FixedUpdate made the game freeze during this process, which takes 5 minutes more or less, and putting it on a Coroutine created micro stuttering, which isn't desirable. But, as long as anything from Unity isn't used, everything is fine. For example, every time a random number appeared it had to be used from the System namespace instead of UnityEngine.

```
if (threadStarted == false)
                {
                    initializeRun();
                }
                whentosave++;
                if (whentosave == 50)
                {
                    whentosave = 0;
                    if (jumplearned)
                    {
                        saveNetwork();
                    }
                }
```

Here, if the game hasn't started the thread, which means that there isn't a new species and it continues with a new genome, it starts a run again. There is also a condition to save the network only when the monster has learnt to jump, in order not to destroy the learning process.

```
if (threadStarted == true)
        {
            if (giveThreadつo_oつ.giveFinishedつo_oつ())
            {
                if (jumplearned)
                {
                    saveNetwork();

                }
                SceneManager.LoadScene("adventure map");
                Destroy(this.gameObject);
            }

        }
        dontsave = false;
```

If the thread has been started, the function enters here. And checks if the thread has finished in order to save the network, now with one more specie, load the scene again to solve some bugs and destroy the actual scene.

As for how initializerun and evaluatecurrent look, they look like this.

```
void initializeRun()
    {
        lastframetotal = Time.frameCount;
        monster.position = initialPosition;
        sightsense.setRightMost((int) monster.position.x);

        pool.setCurrentFrame(0);
        timeout = timeoutconstant;
        clearOutputs();
        Genome genome =
pool.getSpecies()[pool.getcurrentspecies()].getGenomes()[pool.getcurrentgenome()]
;
        genome.generateNetwork(sightsense.thingsseen.Length,
nameOfOutputs.Length);
        evaluateCurrent();

    }
```

It puts the monster to the initial position, sets the time to 5 seconds, clears the outputs, gets the current genomes and gives a set of outputs.

```
void evaluateCurrent()
    {
Genome genome =
pool.getSpecies()[pool.getcurrentspecies()].getGenomes()[pool.getcurrentgenome()]
;
        int[,] input = sightsense.thingsseen;
        outputs = genome.evaluateNetwork(input, nameOfOutputs);
        int falsenumbers = 100 - /*totalfalsenumbers -*/ (sightsense.hungry +
sightsense.sleep) / 2;
        //totalfalsenumbers = falsenumbers;
        for (int i = 0; i < falsenumbers; i++)
        {
            feelingsarray[i] = false;
        }
        if ((bool)outputs["Left"] && (bool)outputs["Right"])
        {
            outputs["Left"] = false;
            outputs["Right"] = false;
        }
        if (!jumplearned)
        {
            outputs["Jump"] = false;
        }
        if (sightsense.witchlevel < 1000)
        {
            outputs["Attack"] = false;
        }
        if (feelingsarray[UnityEngine.Random.Range(0, feelingsarray.Length)] ==
false && falsenumbers > 50)
        {
            outputs["Left"] = false;
            outputs["Right"] = false;
            outputs["Jump"] = false;
            dontsave = true;
        }
        sightsense.movemonster(outputs);
    }
```

It gets the array with the things that the monster sees, it constructs the network with them and gets the outputs, if right and left are pressed at the same time they get deactivated, if jump or attack haven't been learnt yet they get deactivated and if the monster is tired or hungry all the buttons get deactivated. The full game code is able to be downloaded in a link at the end of the document [13]

Tests of the game

Two types of test were done to this game. Testing to prove if everything worked fine and letting other people play to see if they liked the game.

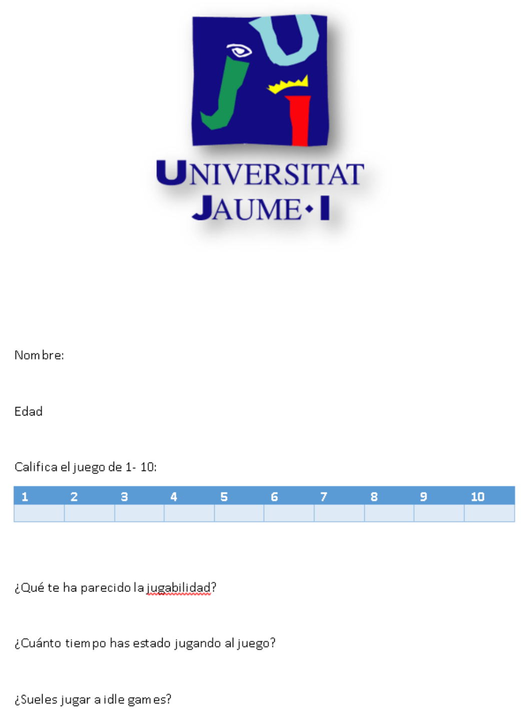
The tests that were done to prove if the game worked were:

- Making a human play the platforming part. With this test I was able to prove that the physics were alright.
- Letting the game play by itself a few hours. With this test, I was able to see that the first time the monster wasn't learning because the genetic algorithm wasn't deleting the bad genes and breeding them to create worse genes. But this was fixed.
- Make the monster collide with everything in order to prove that it doesn't fall. This discovered that, if the monster crashed against a wall it got stuck, but now is fixed.

- A test to see if the economy of the game was fair by cheating the amount of money needed to buy everything. From that I thought that it was a good idea to make the player be able to increase the amount of money each coin gives.
- A test to see if the monster could finish the game.
- To prove how much did it take for the monster to complete the game, several tests were done. Due to the level not being really complicated, it normally takes 10 generations to complete it, but due to needing to get money to buy the person who teaches the attack technique, the game takes more time to complete than that. Usually a few days.

As for the tests that were done to see the quality of the game:

4 people were chosen to test this game. This form was send to the people that too this test.



Nombre:

Edad

Califica el juego de 1- 10:

1	2	3	4	5	6	7	8	9	10

¿Qué te ha parecido la jugabilidad?

¿Cuánto tiempo has estado jugando al juego?

¿Sueles jugar a idle games?

Figure 25

Opinion of the first person: This person rated the game with a 6. He said that the game was okay but advancing was really slow. He tested the game for 50 minutes and plays idle games.

Opinion of the second person: This person rated the game with a 7 after the corrections that the first person helped to fix. He also said that advancing through the game was slow. He tested the game for 2 hours and plays idle games.

Opinion of the third person: He rated the game with an 8, saying that the key to complete the game was making the monster go through the spring many times, played for 10 minutes and doesn't play idle games.

Last person: She gave the game an 8,saying that the beginning the game run slow, but she liked it.

Game description

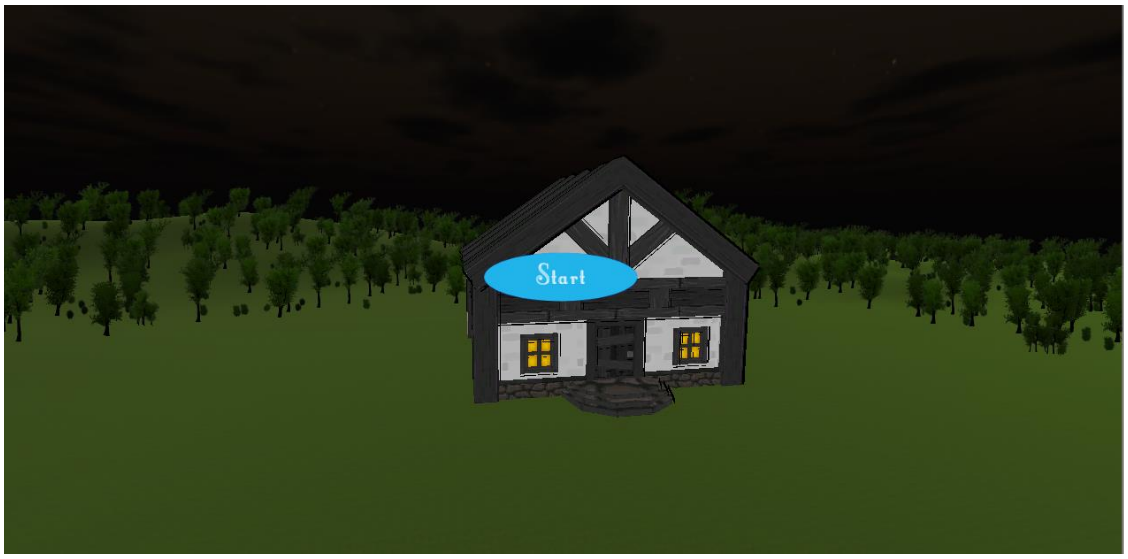
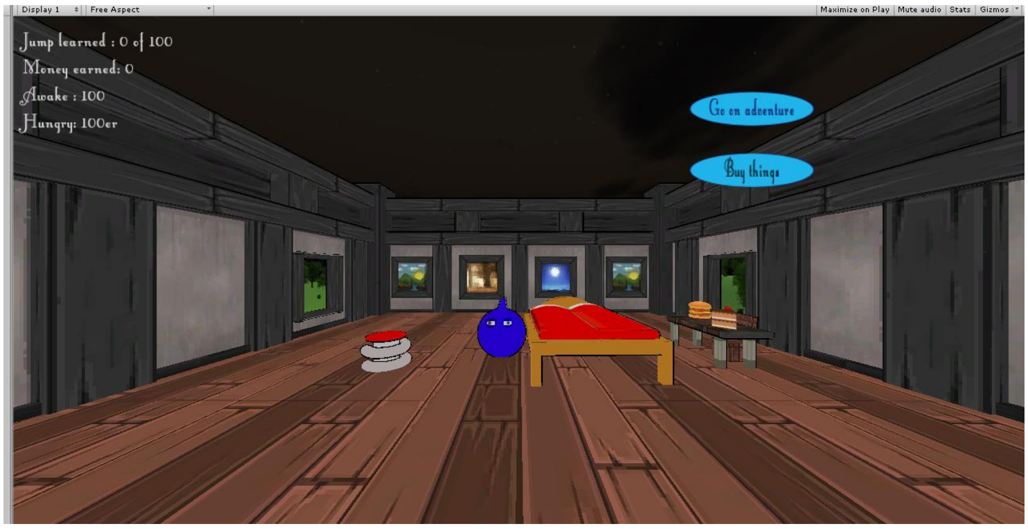


Figure 26

This is the first screen the player will see. Nothing happens in it, the player has to click on Start and the game will start. Here it was intended to load multiple save files, but it couldn't be done due to time.



Jump learned : 0 of 100
Money earned: 0
Awake : 100
Hungry: 100er

Figure 27

This is the second screen the player will see. As said before, the player will be able to drag the monster or click the buttons. The player can see this in movement in the video ruffidio, tutorial + learn to jump at the link in the bibliography [13].

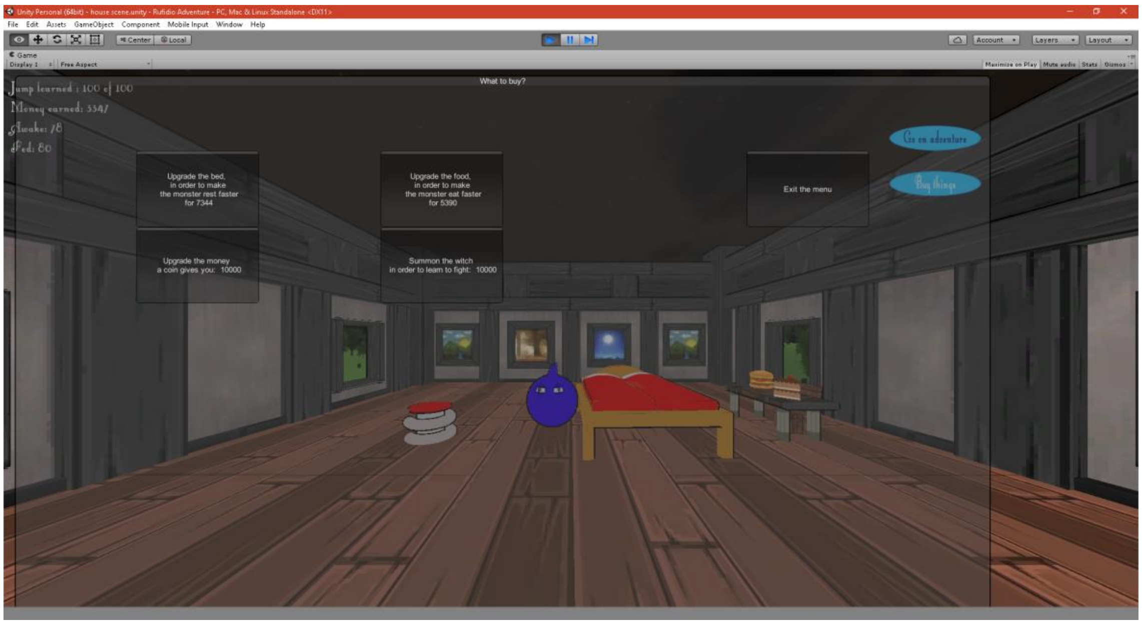

Unity Personal (64bit) - house scene.unity - Rufidio Adventure - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Center Local
Account Layers Layout
Game
Display 1 Free Aspect
Maximize on Play Mute audio Stats Gizmos
Jump learned : 100 of 100
Money earned: 5547
Awake: 78
Fed: 80
What to buy?
Upgrade the bed, in order to make the monster rest faster for 7344
Upgrade the food, in order to make the monster eat faster for 5390
Exit the menu
Upgrade the money a coin gives you: 10000
Summon the witch in order to learn to fight: 10000
Go on adventure
Buy things

Figure 28

That is what appears when it is clicked. Clicking each button of this menu will make the numbers grow up, a message appear that says that the machine is already upgraded to the maximum level or nothing.

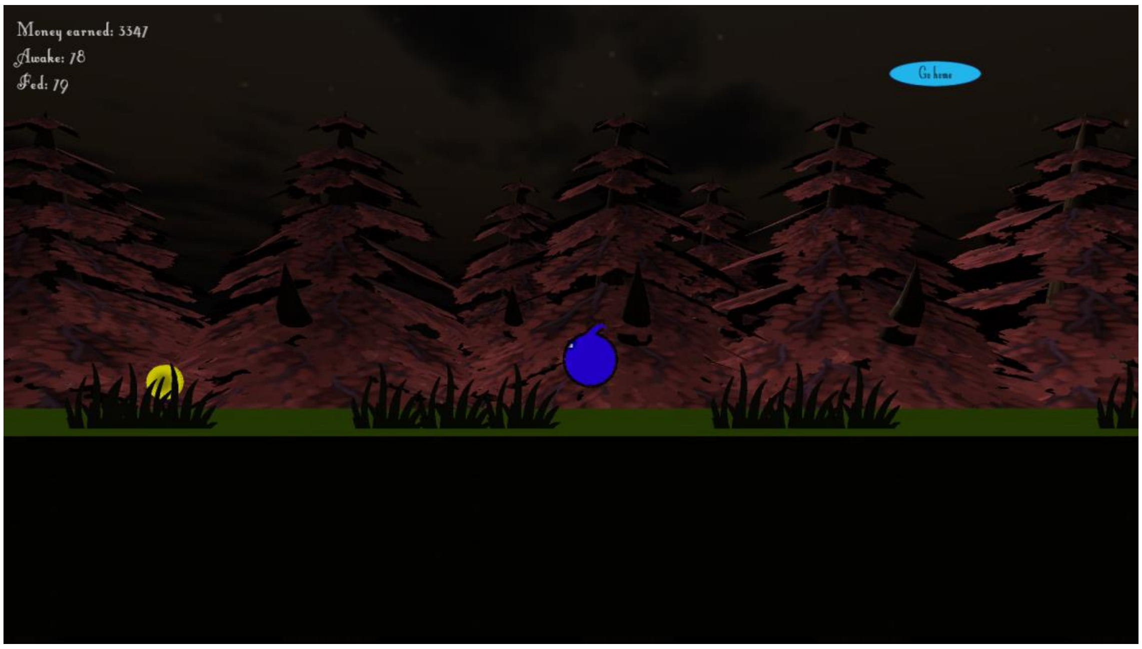


Figure 29

This screen is the one the player sees once it clicks on the go on adventure button. The only thing the player can do here is click Go home to go back to the previous screen.

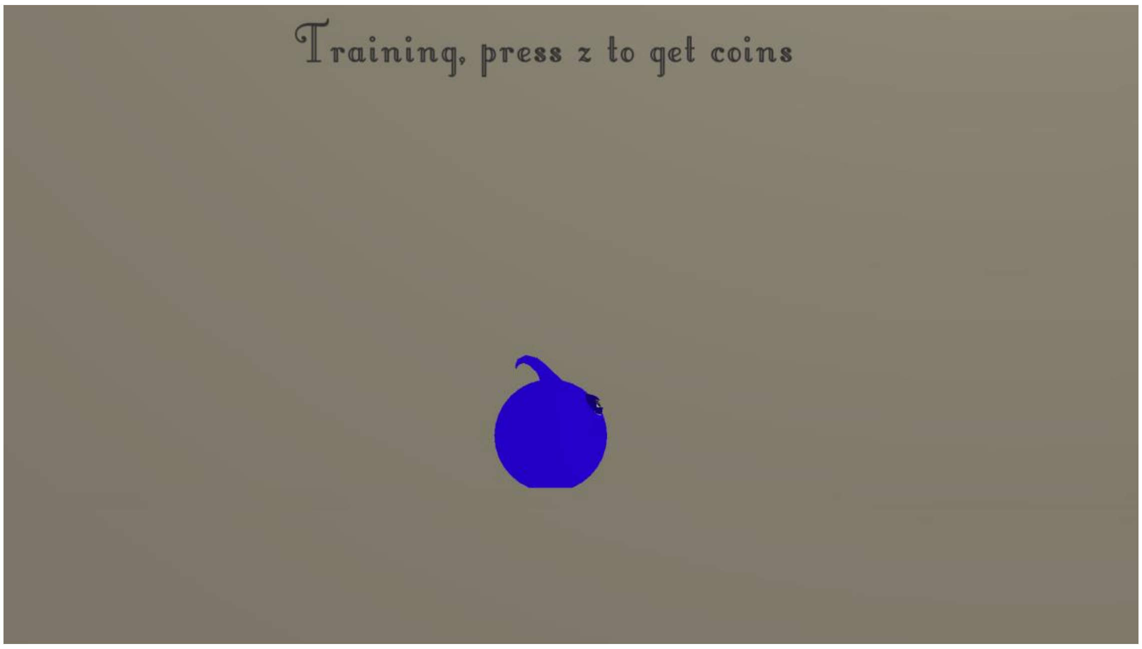


Figure 30

Finally, this is the loading screen. This is the screen that the player will see when the game is creating a new specie. Everything the player can do here is press z to move the monster and get coins. Once it finishes creating the specie, it goes back to the previous screen.

This three screens are shown in the video Rufidio, learning + mingame + buy + rest at the link in the bibliography [13]. What happens after an upgrade is shown in the video Ruffidio, upgrading coins effect and teaching the monster to attack and the effects is shown in the video ruffidio, learn attack + attack all in the link [13] in the bibliography.

4.Conclusions

The objectives that were accomplished here are:

O0: Learn how to develop a professional project.

I learnt how to develop a professional project. Hitting all the deadlines helped to do this.

O1: Learn to write a proposal of professional level:

This objective wasn't fully accomplished due to the doubts the correctors expressed about the writing, but I learnt a simple structure for a proposal.

O2: Put on practice the skills learned during the degree:

This objective was also accomplished because I used everything I learnt.

O3: Investigate about machine learning techniques applied to video games:

Every technique of machine learning hasn't been taught in this degree, so investigating about neural networks, reinforcement learning and genetic algorithms is going to be a very important part of this project.

O4: To increase knowledge about Unity3d in order to apply it to a professional project.

I increased my knowledge about Unity3d. I've learnt about the limitations on multithreading, the amount of bugs that appear if a tree is added, and also the strengths of the game such as how easy is to start making a game.

O5: Develop a videogame

This objective has been accomplished.

O6: Properly documentation of the work done:

The writing of this document clearly shows that the work has been documented.

My final conclusions about this project are:

Even though the game used Unity, it wasn't the best suited for this kind of job. Unity doesn't use multithreading so any heavy processing made the game stop playing for a few second while it processed everything. I was able to overcome this limitation by not using Unity libraries inside the threads. In addition to all this information, I have also learnt that genetic algorithms aren’t suitable for every type of gameplay. For every different action that it wants to be done with the AI, a new neural network needs to be created, which consumes too much memory to be feasible in a lot of the videogame types. In this one it was only desired for the AI to learn to travel the map, but, for example, on a shooter, it would be needed a neural network to learn the map, another to know when, and where to shoot, and another to learn how to cover in the shooting sections. Nonetheless, applying machine learning to a videogame was an enriching experience and I’m really happy for how the final product ended up looking.

As for future things I would like to add to this videogame, one would be a map editor. It would just place blocks on the screen and coins and then, everything would be saved to a xml, and the player would be able to load any created map. Also the ability to have multiple save files.

5.Bibliography

[1] “http://smashboards.com/guides/comprehensive-amiibo-training-table.541/,” [Online].

[2] “https://en.wikipedia.org/wiki/Artificial_neural_network,” [Online].

[3] S. K. a. J. Togelius, ““The Mario AI Benchmark and Competitions” ,”.

[4] “http://unity3d.com/es/unity/multiplatform/,” [Online].

[5] “http://blogs.unity3d.com/2014/05/20/the-future-of-scripting-in-unity/,” [Online].

[6] “http://unity3d.com/public-relations,” [Online].

[7] “https://en.wikipedia.org/wiki/Incremental_game,” [Online].

[8] “http://steamcharts.com/app/363970#All,” [Online].

[9] “http://www.kongregate.com/games/playsaurus/clicker-heroes,” [Online].

[10] “http://e-ujier.uji.es/pls/www/!gri_ass.lleu_asignaturas_ob_g?p_titulacion=231,” [Online].

[11] “https://www.youtube.com/watch?v=qv6UVOQ0F44,” [Online].

[12] “http://www.elmundo.es/ciencia/2015/02/25/54edcd2ae2704e04558b4576.html”.

[13] Adrian, “Code, GDD and game,” [Online]. Available: https://drive.google.com/folderview?id=0B6rsjhW1bXvvZDZDUUpEYzZSaVk&usp=sharing.