DESIGN AND DEVELOPMENT OF AN EDUCATIONAL GAME ABOUT LOGIC CIRCUITS AND COMPUTER ARCHITECTURE

ANDORNOT

UNIVERSITAT JAUME I

Author: Antonio López Ruiz
Tutor: Sergio Barrachina Mir

Gracias.

A mi madre y a mi padre,
por haberme apoyado
siempre y haber dado
todo por mi.

A Cristian Montoya,
el hermano que nunca tuve.

A Maria del Olmo,
por hacer que los días
sean menos grises.

A todos los NH,
porque un minuto con vosotros
hacen que mis penas
sean más pequeñas.

Que haya sobrevivido ha sido gracias a vosotros.

Summary

The following document constitutes the Final Degree Project's Technical Report of the Video Games Design and Development degree carried out by the University Jaume I. The objective of the project is to provide to future Design and Development of Video Games students a gamified experience in the process of learning Computer Architecture.

The game has been developed with the game engine Unity 3D[1], custom art and non-copyrighted music.

Index

List of Figures

List of Tables:

1. Technical proposal

1.1. Final Degree Work Summary

The following document contains the technical proposal concerning the Final Degree Work of Design and Development of Videogames. This essay is about the development of a didactic game for the subject *Basic Informatics*[2] that must run in Linux[3] computers and that is focused in the teaching and practising of the different components of a computer, and how these elements interact. The videogame consist in both a series of test and a logic circuits simulator, that allow the student to build from simple circuits to a more complex association. This game pretends to make the Computer Architecture lessons more interactive and more visual.
It is a simulator/quiz game, in which the player has to accomplish several missions to advance through the game and learn the basics of Computer Architecture, it will be developed in Unity 3D.
Keywords: simulation, computer architecture, logic circuit, ARM, subject.

1.2. Introduction and motivation of work

The main goal is to develop a game that works as a reinforcer to *Basic Infomatics*, that allows the student to complement the information given at class by playing and experimenting with different levels of the game. To accomplish this, the game has to formulate a series of tests and recreate the different Logic Circuits in Unity 3D, it has to simulate from the very basics of Logic Circuits[4] to the more complex ones at a code level, using C#[5], one of the main programming languages of Unity 3D. The the games has to visually the different circuitry in 3D models or drawings and implementing them into Unity 3D. The final goal is to organize the resulting simulator and tests in a series of levels and problems to make the learning process progressive and intuitive.

I think that educational games[6] have a great potential, and in my opinion in the future, interactive systems will have a greater impact in the educational process. Since I discovered this alternative use of videogames I have wanted to make one, or get involved in the development of one of the kind, so I think this Final Degree

Work will be a good chance to get involved and make a functional educational game.

1.3. Related Subjects

- VJ1227 - Game Engines
- VJ1223 - Videogames Art
- VJ1222 - Conceptual Design of Videogames
- VJ1202 - Basic Informatics
- Vj1214 - Consoles and Videogame Devices

1.4. Final Degree Work Goals

- Develop a game which can be used to learn Computer Architecture[7].
- Create tests about the Computer Architecture.
- Create an original game with own animation and models, where they handle a series of test and visually recreate the use and interaction of Logic Gates.
- Use and the different Logic Gates, to "recreate" more and more complex Logic Circuits and understand its implementation.
- Make the Tests and Logic Circuits the key mechanic of the game, making it the key to solve the different levels and problems in the game.

1.5. Planning

The original planning is shown in Figure 1.

Action	Detailed action	Estimated time
Personal work	- Program of different elements of the game (Menu, Logic gates, Logic Circuits): **110** - Design of the levels (When and how the different problems are presented to the player): **30** - Testing (Test the work done to avoid future problems): **15** - Art (Create and implement the art, and if some external elements are used make sure it is free of copyright): **80** - Investigation (Solving code/art problems or search for different ways of implementing elements in the game): **15**	**250**
Deliverables	- Technical proposal - Game Design Document (GDD) - Memory - Definitive Memory - Project	**43**
Tutorial		**6**
Review		**1**

Figure 1

1.6. Expected Outcome

The expected outcome is a game programmed in C#, that test the students knowledge about Computer Architecture and that can recreate the logic gates, its use and its combinations. Also a visual representation of it, that allows to distinguish without a doubt the different parts and how they connect.
The outcome of this project is a game that can work as a pure simulator of Logic Circuitry, and as a tool to learn and understand it. The project must be able to work in Linux and Copyright free.

1.7. Tools

- Unity 3D,
- Adobe Photoshop[8],
- Adobe Illustrator[9],
- C#
- GitHub[10]

2. Game Design Document (GDD)

2.1. Introduction

This is the Game Design Document of **Andornot**. This game is for Linux (PC) Operative System that exemplifies and gamifies the content about Computer Architecture, in the subject *Basic Informatics*, of the Design and Development of Videogames degree in Jaume I University. This document has as a goal to point out the elements that **Andornot** must include, and to serve as a guide in case the game needs correction and/or further development.

2.1.1. Game concept

Andornot is a combination of a series of tests about Computer Architecture and simplified circuits design simulator, that presents the digital enginery in a more accessible way, where the student completes its education with tutorials and several challenges.

2.1.2. Genre

Andornot is the union of two genres, *Simulation Game* and *Quiz Game*:

- Simulator: This genre consists in the total, partial or adapted simulation of an action and/or phenomenon of the real world. In **Andornot** the working process of logic circuitry is being simulated and its represented in a way that each of the elements is easily identifiable.
- Educational game: A quiz is a form of game or mind sport, in which the players (as individuals or in teams) attempt to answer questions correctly. It is a game to test your knowledge about a certain subject. In some countries, a quiz is also a brief assessment used in education and similar fields to measure growth in knowledge, abilities, and/or skills.. In **Andornot** the quiz its manifested in the tests and in the sequence of the different challenges and levels.

2.1.3. Purpose and target audience

The main goal of **Andornot** is to offer the students and the teachers of the subject *Basic Informatic* a 2D Game developed in the Unity3D game engine for Linux. The game is meant to act as a complement for the subject matter. Its interest must lay down in the educational content.

The game is mainly directed to the students in the *Basic Informatics* subject to reinforce the information exposed in class and offer an interactive system that visually represents the different tests and logic gates.

2.1.4. Gameplay

Each level of **Andornot** offers a problem that the student must solve. To do it, the student must use its knowledge to solve the test or to identify several logical gates and combine them.

2.1.5. Visual style

Andornot will have a simple style, almost minimalist, to reinforce the importance of each element and make the game mechanics the main element of the game. The visual style that fits the best with this style is the "Programming Framework" or "Blank Workspace"

2.1.6. Reach

The main goal is to develop a solid game that can work on itself, and that is able to be expanded in the future with no problem, in case game must be modified (by adding mechanics of levels).

2.2. Game mechanics

In this section the different mechanics in which **Andornot is** based, are further detailed. The gameplay and the actions that the played can execute are detailed.

2.2.1. Gameplay

The different elements of the gameplay are described in this section:

- **Topic: Andornot** is divided in four differentiated parts that will divide the game:
    - Introduction: This part will be a series of test about basic Computer Architecture.
    - Logic Gates: This part will introduce Logic Gates and a few problems about them.
    - Processor[11]: This part will be focused in the Processor and its parts and functionality.
    - Memory[12]: This part will be focused in the Memory, its parts and its functionality

- **Levels:** Each of the levels of **Andornot** is a problem or and exercise related to the subject that must be solved by the students. The main goal will be determined by the Topic.
- **Elements:** To reach the main goal the player has at its disposal different logic gates (AND, OR, XOR, NOT) and the answers to the tests.
- **Intensity:** The difficulty of **Andornot** is progressive and its marked by the advance of the subject at class. The different levels are distributed to be played in sequential order, but the player is allowed to play any level at any time, to reinforce his knowledge or to catch up the class.
- **Progression:** In the first levels the amount of available elements is very reduced and then, as the game increases its difficulty and complexity more items will be available. This way the player learns how to use them progressively.

2.2.2. Game flow

And now the course of a game of **Andornot** will be detailed. The steps that a player must follow from the beginning of the game to when the level is concluded. In this section the mechanics are described, further the content of each section will be described.

The player starts **Andornot** and the *Main Menu* is displayed. If he wants to start a game the player will select *Play*. Next, the player will select the *Subject* that he wants to play in the *Topic* Section.

Once in the *Topic Section* the levels are displayed and organised in a sequential way to be played in this order, but in order to improve the flow of the game and the subject, every level is available from the start.

When the player starts a *Level,* depending on which *Topic* he chose, two different levels can be selected:

- A test, with a question and multiple possible answers that the player must select if he thinks that are good answers. Once he has selected the answers he considers to be right he can check the results.
- An empty workplace, and a goal that the player must complete are shown. The player will only complete the level if he beats the established goal. To do that the player can use several elements that are in the superior right part of the screen, that can use and connect within them. He will only be allowed to use the elements available for that level. The player is allowed to put these elements anywhere in the workplace and connect them.

There is no losing in this game, or a time limit. The only options are to complete a level or to exit to the *Level Selection Menu*. He can also repeat a level as many times as he desires.
When a player finishes a *Level* a "Completed" sign is showed, to point out that the level is concluded, and now the player can continue to the next level or return to level selection.
At any moment the player can return to previous *Levels*.

2.2.3. Mechanics and communication between elements

As **Andornot** has two different kind of mechanics, one for the tests that consists in a question with multiple answers, some of the good and some of them bad. Every interaction will be made by the mouse, by clicking the correct answers.
The other kind of mechanic is the *Logic Circuits* one. Logic Levels are played over a flat workplace, and the player can put, displace and delete as he wants every element of his choosing. Each element will have one or more ways IN, and one or more ways OUT, that the player can connect if the elements are compatible. Every interaction will be made with the mouse, by clicking the different elements of the game, and drag it or the connections along the screen.

2.3. Interface

In this section each one of the screens that form **Andornot** are specified and described with more detail. In addition, the transitions between them are further specified, as well as each element of the GUI (Graphical User Interface). The attached images are sketches that illustrate what is showed in each screen.

2.3.1. Flux diagram

The following diagram (Figure 2) of states shows the available screens through **Andornot** and the transition between them.

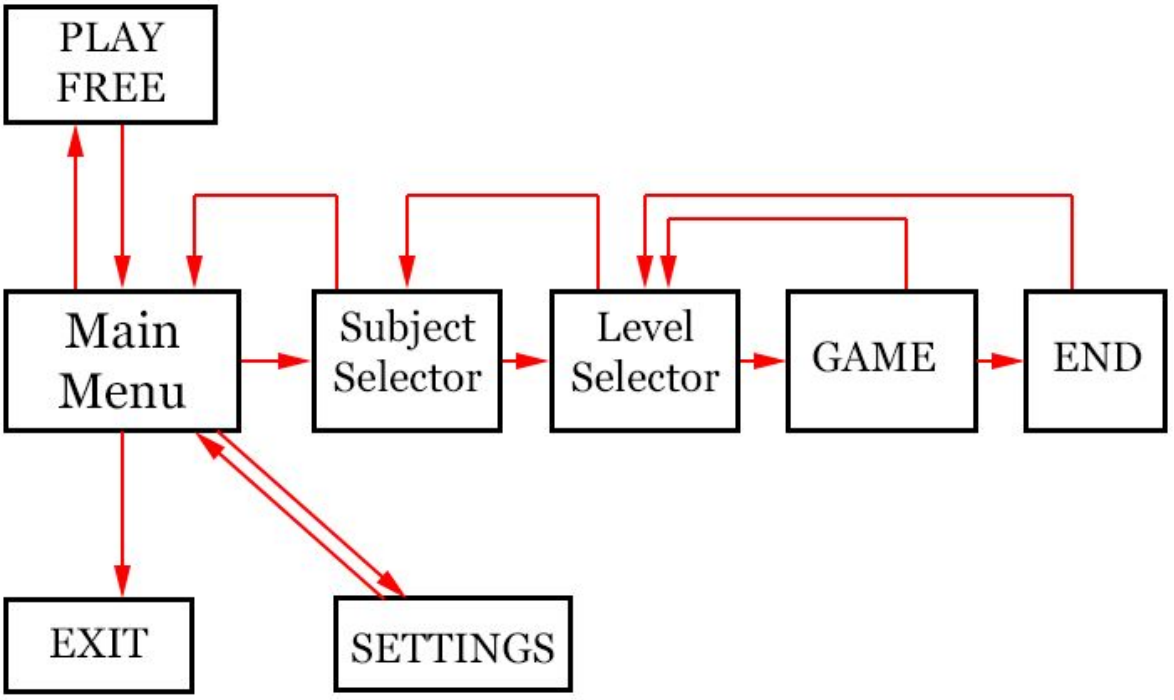


Figure 2

2.3.2. Main Menu

The sketch of the *Main Menu* screen is shown in Figure 3:

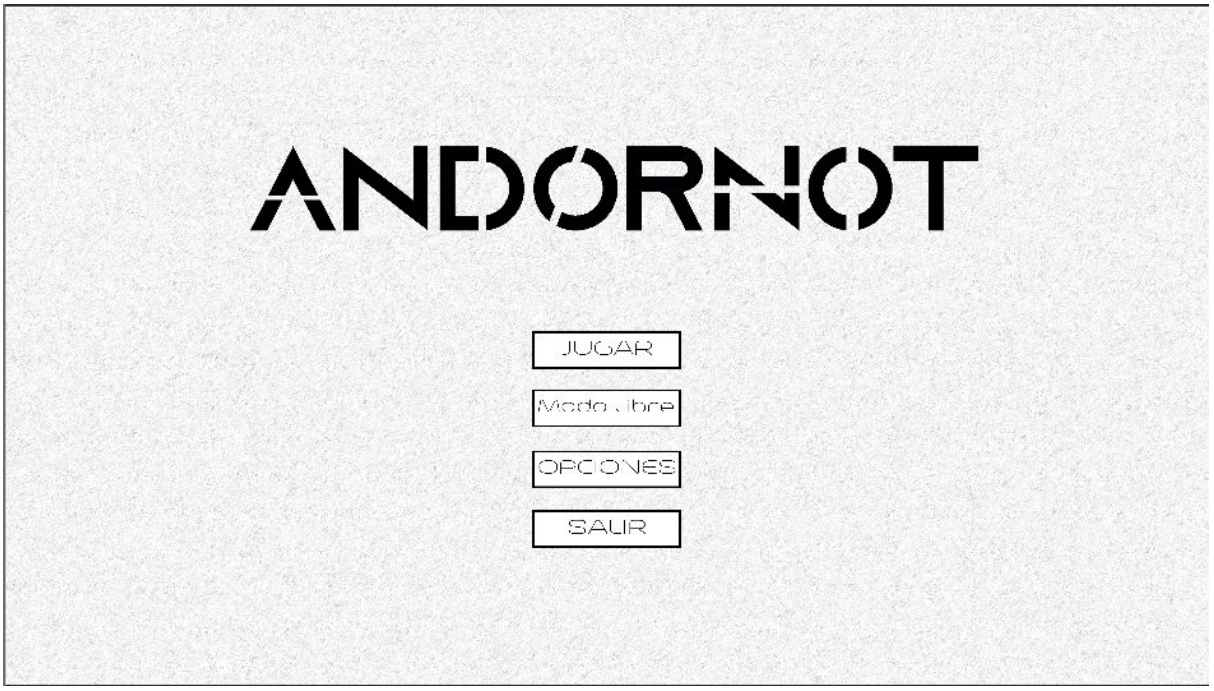


Figure 3

List and description of its components:

- **PLAY:** when clicked it takes to the *Topic Menu* screen
- **MODO LIBRE:** when clicked it changes the scene to *FreeMode*
- **SETTINGS:** it activates the *Settings Menu*
- **EXIT:** when clicked it exits into the Operative System

2.3.3. Level Selection

The sketch of the *Level Selection* screen:

Figure 4

List and description of its components:

- **Main menu:** it changes the scene to Main Menu
- **First Steps:** it changes the scene to Level 1 Level Selection
- **Logic Circuits:** it changes the scene to Level 2 Level Selection
- **Processor:** it changes the scene to Level 3 Level Selection
- **Memory:**it changes the scene to Level 4 Level Selection

2.3.4. Topic Selection

The following image (Figure 6) represents the Level Selection scene.

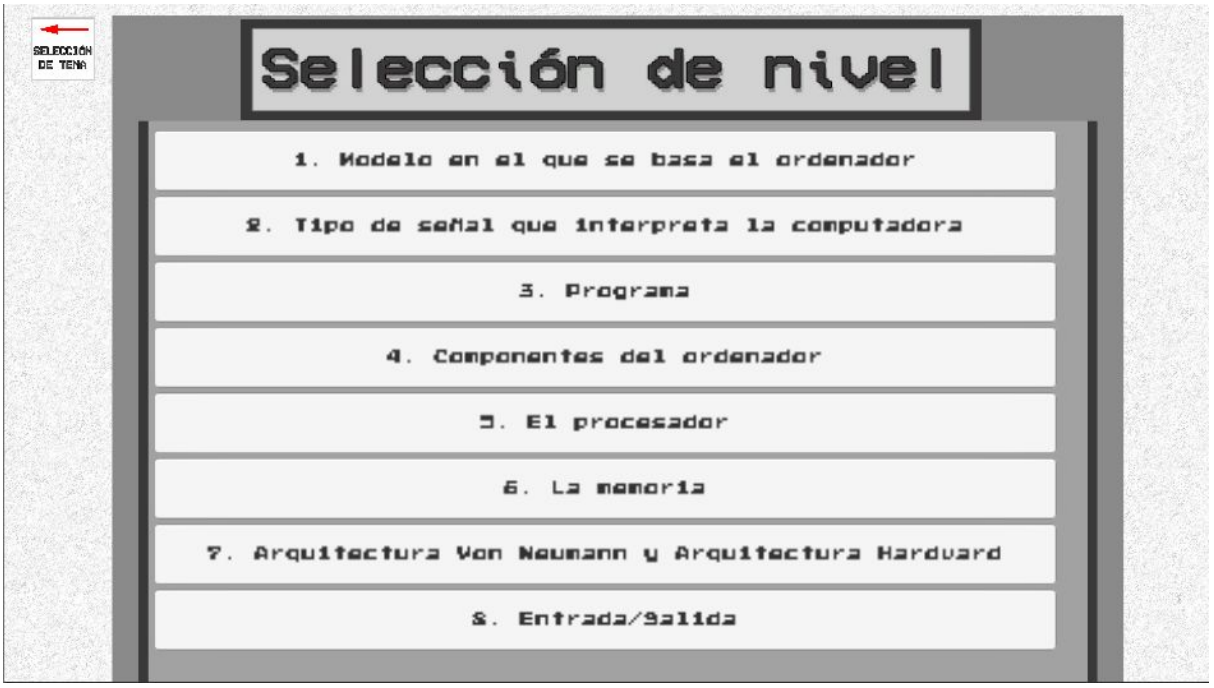


Figure 5

List and description of its components:

- **Topic Selection:** changes the scene to the chosen *Topic*
- **Level List:** a list that shows all the playable levels

2.3.5. Test Level

Test Level (Figure 6). List and description of its components:

- **Question:** The question that must be solved.
- **Answers:** Possible answers.
- **Task & Hints**: clues and tips that can help to solve the level and a description of the task to be done

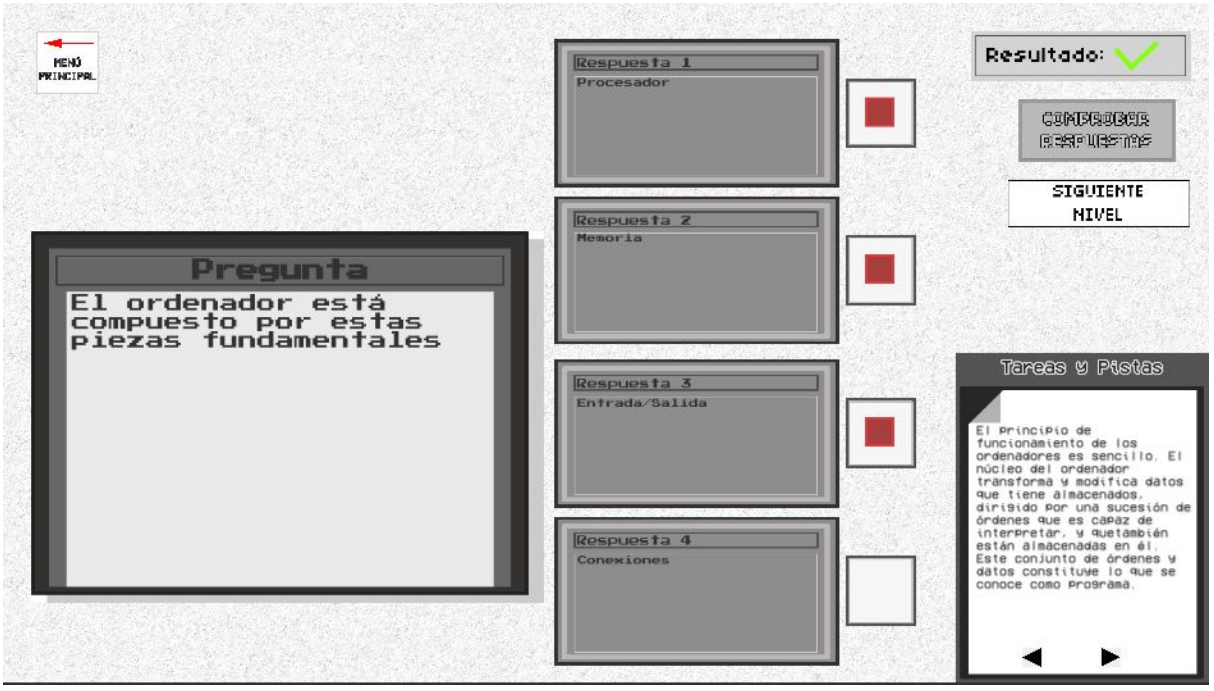


Figure 6

Logic circuit level (Figure 7). List and description of its components:

- **Components:** components available and with which the player can interact
- **Components interaction:** options to interact with the components
- **Scenario 2D:** game panel
- **Level Selection:** when pressed, it returns to the Level Selection screen
- **Task & Hints**: clues and tips that can help to solve the level and a description of the task to be done
- **Test:** checking the circuit status

Figure 7

2.3.6. Level End

The following image (Figure 8) represents the Level End

Figure 8

List and description of all its components:

- **Next Level Button:** pressing it starts the next level
- **Level Selection Button**: pressing it starts the Level Selection screen

2.3.7. Free Mode

The following image (Figure 9) represents the Free Mode

Figure 9

List and description of all its components:

- **Components:** all the logic gates are available to use them. The input and output value can be modified by the player
- **Components interaction:** options to interact with the components
- **Scenario 2D:** game panel
- **Level Selection:** when pressed, it returns to the Level Selection screen
- **Task & Hints**: clues and tips that can help to solve the level and a description of the task to be done
- **Change Bit Value:** Panel where the player can change the value of the input/output

3. Project Development

3.1. Computer Architecture and Logic Circuit analysis

In order to develop the game idea the first step was to study and understand the Educational Games genre and how to apply them to this subject-matter. There are many educational games, and they are usually very different, depending of what subject or matter or study it focus is on, so my tutor and I checked for other games that are Logic Circuit focused. There are not many in the mainstream platform services but we found a few in Steam:

- SHENZEN IO[13]
- Hardware Engineering[14]
- Ones and Zeroes[15]

These games helped me structuring the game canvas and the game progression.

Some information was needed to formulate the questions and answers to create the tests, therefore the information needed was gathered using the textbook *Introducción a la arquitectura de computadores con QtARMSim y Arduino*.
The information stored will be detailed in the Test Level Game Design

The next stop was to understand the different components of basic logic circuits and how they interact with each other. To achieve this goal, the information used was the subject-matter *Computer Architectures* given in the subject *VJ1214 - Consoles and Videogame Devices* and the different exercises proposed there. In the end the following logical components of circuits: AND, OR, NOT, XOR, NOR & NAND, where chosen. Their features and characteristics are explained in the Logic Circuit Level Design.

3.2. Game Design

After analyzing the information stored and playing Hardware Engineering, an idea about the game design was formed. Two different kind of mechanics would be established. One focused to a test simulator and the other mechanics would be

focused towards simulating a Logic Circuit. The main focus is to act as a reinforcement to the students progress.

3.2.1. Test Level

In this type of level the player must respond correctly to the question posed by the game. For that matter he will be able to choose between some answers (also given by the game). Once the player has choose the answers he considers, he can check the results and if the outcome is correct he can go to the next level.
The Subject-Matter Scenes that belong to this type of level are:

INTRODUCTION: In this scene the content covered is basic information about Computer Architecture. The different levels that belong to Introduction and its content are displayed in the following table (Table 1):

Level	Information
1	Von Neumann Architecture
2	Digital Signal
3	Definition of a program
4	Fundamental parts of a computer
5	Processor
6	Memory
7	Von Neumann/ Harvard
8	Exit/Entry

Table 1

PROCESSOR: In this scene the main focus of information is about the Processor, the functionalities of it and how is it formed. All the levels and its contents are displayed in the following table (Table 2):

Level	Information
1	El procesador
2	Instrucciones
3	Tipos de procesadores
4	Tipos de procesadores II

5	Componentes del procesador
6	Elementos estructurales
7	Registros
8	Registros II
9	Unidades de transformación
10	Circuitos digitales y
11	Buses
12	Instrucciones
13	Orden de la instrucción
14	Lectura de la instrucción
15	Decodificación de la instrucción
16	Incremento del contador
17	Ejecución de la instrucción
18	Transformación de datos
19	Transferencia de datos
20	Control de flujo
21	Control de procesador

Table 2

MEMORY: In this scene the main focus of information is about the Memory, its functionalities and how is it formed. All the levels and its contents are displayed in the following table (Table 3):

Level	Information
1	Confusiones con la memoria
2	La memoria
3	Escritura/Lectura

4	Instrucciones y Registros
5	Transferencia de información
6	Transferencia de información II
7	Little Endian/Big Endian
8	Direcciones de memoria
9	Tipos de memoria
10	Combinaciones
11	Combinaciones II
12	Memoria Caché
13	Ordenadores Modernos

Table 3

3.2.2. Logic Circuits

The information used to form and design this level is the following:

Digital Signals

There are many signals and ways of communication, but in the digital world, the most basic form of information is a binary number which uses 0 and 1 as its options.

Digital Electronics

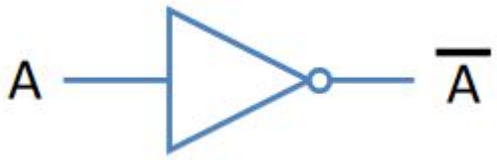
Digital electronics or digital circuits are electronics that operate on digital signals. They are usually made from large assemblies of logic gates, simple electronic representations of Boolean logic functions.

Boolean Logic

Boolean algebra is the branch of algebra in which the values of the variables are the truth values *true* and *false*, usually denoted 1 and 0 respectively. Instead of elementary algebra where the values of the variables are numbers, and the prime operations are addition and multiplication.
The main operations of Boolean algebra are the conjunction *and*, the disjunction *or*, and the negation *not*.

NOT

The logic gate NOT performs the boolean negation. Which returns the opposite number of the one inputted.



A	$\overline{A}$
0	1
1	0

Figure 10

OR

The OR logic gate performs the boolean addition. Which return 1 (true) if any of two components in the addition is 1, or 0 (false) if both are 0.

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 11

AND

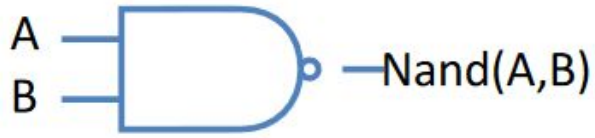
The logic gate AND performs the boolean multiplication. Which only returns 1 (true) if both components of the multiplication are 1, in the other cases the result is always 0.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Figure 12

NAND

A NAND gate is an inverted AND gate. A NAND gate is a universal gate, meaning that any other gate can be represented as a combination of NAND gates.

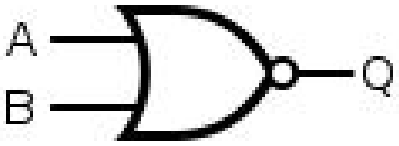


A	B	Nand(A,B)
0	0	1
0	1	1
1	0	1
1	1	0

Figure 13

NOR

A NOR gate is a logic gate which gives a positive output only when both inputs are negative. Like NAND gates, NOR gates are so-called "universal gates" that can be combined to form any other kind of logic gate.



Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

Figure 14

XOR

XOR gate is a digital logic gate that gives a true (1) output when the number of true inputs is odd.

A	B	Xor(A,B)
0	0	0
0	1	1
1	0	1
1	1	0

Figure 15

Once the performance of the Logic Gates is established, the level design is the following:
In this type of level the player must respond solve the logic circuit posed by the game. For that matter he will be able to choose between some logic gates given by the game. Then the player must connect them to get the final logic circuit. Once the player has completed the logic circuit, he can check the results and if the outcome is correct he can go to the next level.
The only Subject-Matter that belong to this level is Logic Gates. The following table represents its levels and content (Table 4).

Level	Content
1	Logic Gate: NOT
2	Logic Gate: OR
3	Logic Gate: AND
4	Exercise 1
5	Exercise 2
6	Logic Gate: NOR
7	Logic Gate: NAND
8	Logic Gate: XOR
9	Exercise 3
10	Exercise 4

Table 4

3.3. Programming and code

The game has been programmed to run in the Unity Game Engine using C# and Visual Studio[16] as an editor. The code uses the Game Objects[17] system of Unity, in which every Game Object which components must be given by the user. Game Objects can contain other Game Objects. Game Objects may be children of another Game Objects.

There are two systems designed for the game, one for the Test Levels and other for the Logic Gates Levels. Both systems control the flux of information in the scene, what the player is allowed to do and what the winning conditions are. They are also designed in a way that allows to easily create new levels.

3.3.1. Test Levels

All the interactions and control with the world in Test Levels are managed by a class called *QuestionManager.cs*.
This class receives as information an array of booleans, that reflects the correct answers. When the class is started (Start[18]) it generates another boolean array with the same size, for the purpose of storing the player responses. When the button[19] of an answer is pressed the game changes the created boolean array value in its right position. Finally, when the player decides to check the answers if compares both arrays and if they are both equal it means the answers are correct.

3.3.2. Logic Circuits

In similar way to Test Levels the iterations with the scene in Logic Circuits are managed by a class called *CircuitManager.cs*.
This class receives as information several byte[20] arrays that consist in the Truth Table[21] of the logic circuit. This class hosts all the different Logical Elements availables and how the user can interact with them.

3.3.2.1. Logic Circuit Interactions

The following elements form all the different options the user has to use the different Logical Gates. Each of the options buttons has a different function of *CircuitManager.cs*. attached to them which operation is disclosed in the following section:

- **Delete Element:** This function deletes the last selected Logical Element and its connections.

- **Save:** As the names points, this function is used to save the game. For this purpose, the game creates a list that saves every interactable element in the scene, its position and it assigns and id. Once this is done, FileStream is used to save the information in an archive.

- **Load:** This function is used to load the last circuit saved. If it is clicked while there is not saving file it saves the current logic circuit.
  It clears the current canvas and then it gets the serialized information, and it deserialices it. Finally it makes spawn every element saved in its position and with its connections.

- **Clear:**
  This function is used to clean the canvas by destroying all the Logical Elements.

- **Tool:**
  This enum determines how to interact with the Logical Elements (*public enum Tool{ Circuit, Translate, Move}*).

    - **Edit Circuit:** Circuit
      This allows to connect the output of a Logical Element with the input of another.

    - **Move Elements:** Translate
      The code checks if the Mouse is moved and if it clicks on a element, and then changes the position of the element until the click is stopped.

    - **Move Camera:** Move
      Interaction with the Logical Elements will become unavailable but the user can now interact with the camera (translate, zoom in, zoom out...)

- **Spawn:**
  This function makes spawn a Prefab[22] that represents a Logical Gate in the middle of the Camera[23].

- **Test:**
  This function changes the values of the Input Logical Elements with values given by the Truth Table and checks that the Output Logical Elements are the same value that the value in the Truth Table. This method is shown in Figure 15.

```
public void Test()
{
    bool comprobacion = true;
    bool comprobacion2 = true;
    List<EditableOut> listaInputA = new List<EditableOut>();
    List<EditableOut> listaInputB = new List<EditableOut>();
    List<EditableOut> listaInputC = new List<EditableOut>();
    List<EditableOut> listaInputD = new List<EditableOut>();

    List<BitDisplay> listaOutputA = new List<BitDisplay>();
    List<BitDisplay> listaOutputB = new List<BitDisplay>();

    for (int i = 0; i < elements_parent.childCount; i++)
    {
        var ce = elements_parent.GetChild(i).GetComponent<CircuitElement>();
        if (ce is EditableOut)
        {
            EditableOut var = ce as EditableOut;
            if (var.returnTag() == "A")
                listaInputA.Add(var);
            else if(var.returnTag() == "B")
                listaInputB.Add(var);
            else if (var.returnTag() == "C")
                listaInputC.Add(var);
            else
                listaInputD.Add(var);
        }
        else if (ce is BitDisplay)
        {
            BitDisplay var = ce as BitDisplay;
            if (var.returnTag() == "A")
                listaOutputA.Add(var);
            else
                listaOutputB.Add(var);
        }
    }

    for (int i = 0; i < longitud; i++)
    {
        for (int j = 0; j < listaInputA.Count; j++)
        {
            listaInputA[j].value = inA[i];
        }
```

```
            for (int j = 0; j < listaInputB.Count; j++)
            {
                listaInputB[j].value = inB[i];
            }

            for (int j = 0; j < listaInputC.Count; j++)
            {
                listaInputC[j].value = inC[i];
            }

            for (int j = 0; j < listaInputD.Count; j++)
            {
                listaInputD[j].value = inD[i];
            }

            for (int j = 0; j< listaOutputA.Count; j++)
            {

                if (listaOutputA[j].getValue() != outA[i])
                {
                    comprobacion = false;
                }

            }

            for (int j = 0; j < listaOutputB.Count; j++)
            {
                if (listaOutputB[j].getValue() != outB[i])
                {
                    comprobacion2 = true;
                }
            }

        }

        if(comprobacion == true && comprobacion2 == true)
        {
            ChangeLevel();
        }

    }
```

Figure 16

3.3.2.2. Logic Gates Components

Each Logic Gate has its own properties and ways of working, but they all share a base, they have inputs and outputs, and share a base structure.

So a skeleton of the Logic Gates was created by using three classes, *LogicalElement.cs* which acts like a parent for all the Logical Gates and other

elements, InPoint.cs, which handles the logical gate inputs and OutPoint.cs which manages the element outputs.

LogicalElement.cs

As stated above, this class sets the base structure of the logical gates, it is composed by an array of InPoints (the data this logical gate receives) and an array of OutPoints (the data this logical gives) and an string that stores its type (and, or , not...).

This class handles the logical element since the moment it starts by starting a coroutine that checks if the value of the Logical Gate.

This class also handles the Inputs and Output connections when the LogicalElement is being moved by moving them too as well as when it is deleted.

OutPoints.cs

This is the class that handles the output representation and interactions for each output of every element. For this purpose each Output is made by the an InPoint, a LineRenderer that will be the visual representation and its value . When awaken, its assign a set of values to the LineRenderer[24].

It also controls mouse interactions on two phases. Only if CircuitManager is on Circuit "mode"

- **OnMouseDrag()**

While the mouse is dragging the output it sets the Line Renderer start point from the out point and the finish point to the mouse location.

- **OnMouseUp()**

In this code, it checks if the mouse position projection in the camera with a RaycastHit2D. If the mouse position "hits" with a GameObject, if it is not trying to connect with itself and it is "hitting" an Input element it connects the output with the input, it updates the input and output positions if they already have a value to null and then sets the current values. When completed it calls Draw().

The following figure (Figure 16) shows the OnMouseUp() function.

```
protected virtual void OnMouseUp()
    {
        drag = false;

        RaycastHit2D hit = Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.mousePosition),
Vector2.zero);

        if (CircuitManager.circuit_tool == CircuitManager.Tool.Circuit)
        {
            if (hit.collider != null && hit.collider.gameObject != gameObject &&
hit.collider.GetComponent<InPoint>())
            {
                InPoint end_point = hit.collider.GetComponent<InPoint>();
                if (end_point.In != this)
                {
                    if (Out != null)
                    {
                        Out.In = null;
                        Out = null;
                    }

                    if (end_point.In != null)
                    {
                        end_point.In.Out = null;
                    }
                    end_point.In = this;
                    Out = end_point;
                }
                else if (end_point.In != null)
                {
                    end_point.In = null;
                    Out = null;
                }
            }
            Draw();
        }

    }
}
```

Figure 17

InPoints.cs

This a very simple class that is made by the output information that each input receives, and a LogicalElement class.
It has a function that when called its LogicalElement gets assigned to the Logical Gate parent element and starts the coroutine in the parent.

3.3.2.2. Logic Gates Types

All Logic gates share an structure and components but each one of the has the specific part that determines how the value changes through that Logic Gate.
Every Logical Gate is a children from Logical Element but is defined in a separate class. For every element the byte value is handled in the ValueSet() that every class has, making the appropriate mathematical operations for each one.

And

The code checks the input array that receives the class, and checks each value. If it is not null it returns its own value, and if it is null it returns 1. For each one of those results, the ending byte is multiplied by the result, this allows to get a working AND Logical Gate, it only returns 1 if both inputs are 1.

Then when the end byte is obtained, the output is setted to this value. The code for And is refelected in the Figure

```
public class AndLogicalElement : LogicalElement
{
    protected override void ValueSet()
    {
        byte end = 1;

        for (int i = 0; i < inPoints.Length; i++)
        {
            end *= inPoints[i].In != null ? inPoints[i].In.value : (byte)0;
        }
        end = (byte)Mathf.Clamp01(end);

        for (int i = 0; i < outPoints.Length; i++)
        {
            outPoints[i].value = end;
        }
    }
}
```

Figure 18

Or

This class works very similar to the AND class, but instead of multiplying each other input number by the others and by the end byte, we add the inputs with each other. So we get a working OR Gate, as we get as a result 1 (true), if we have at least one input that is 1.

Not

As with the previous classes, first it gets the input value and return 0 if it is null or in the contrary the input value. Once it is obtained, if the result is bigger than 0 it sets the output to 0 or to1 if this condition is not met.

The result is a NOT Gate, that sets the byte value to 0 if the input is 1 and sets the value to 1 if the input is 0.

Xor

The code is very simple, and i just assigns the value of the output to 1 if both values are different. As a result we get a XOR Logical Gate.

```
public class XorLogicalElement : LogicalElement
{
    protected override void ValueSet()
    {
        base.ValueSet();
        byte end;

        if ((inPoints[0].In != null && inPoints[0].In.value == 0) && (inPoints[0].In != null &&
inPoints[1].In.value != 0))
            end = 1;
        else if ((inPoints[0].In != null && inPoints[0].In.value != 0) && (inPoints[0].In != null &&
inPoints[1].In.value == 0))
            end = 1;
        else
            end = 0; ;

        outPoints[0].value = end;
    }
}
```

Figure 19

Nand

It checks that both input points are equal to 1, if this goal is met sets the value of the end byte to 0. If the goal it is not met it sets the value of the end byte to 1.
Then it sets the output value to the end byte, and it sets as a result a working NAND Gate.

Nand

As with the OR class, it adds all of the input values. If the result is bigger than 0 then, the end byte will be 0, if else it will be 1.
This code mimics the working process of the NOR Gate, as it sets the value of the output to 1 only if both inputs are negative (0).

Equal

It works very similar to XOR Logical Gate, it sets the value of the output to 1 if both input values are the same and it sets the value of the output to 0 if they are different.

3.3.2.3. Logic Gates Extras

To complete the simulator, some other elements must be added to the scene. Such elements as the input bits, a button than changes its value while its pressed…

They all are childrens of the LogicalElement as the Logic Gates, so they can communicate with no problem with each other.

Bit Input

This class handles the inputs. By default the base value is 0, but it can change during the test and it stores the value given by Truth Table.

Bit Output

As the name suggest this class is responsible to set the result from a circuit or a piece of it. When awaken it gets the text which prefabs belongs to and changes it and its value from the input points. As the BIT FIELD it only possess an input value and not an output one, due that this is the end of the circuit and must test the result.

Splitter

This is a class that allows an output to be shared with multiple inputs. This is a Logical Element that has one input element and three output elements, this allows to distribute the information between three potential inputs.

As stated the class get the byte value from the input, and assigns it to every output. This class is presented in Figure 19.

```
    public class SplitterLogicalElement : LogicalElement
{
    protected override void ValueSet()
    {
        byte value = inPoints[0].In == null ? (byte)0 : inPoints[0].In.value;

        for (int i = 0; i < outPoints.Length; i++)
        {
            outPoints[i].value = value;
        }
    }

}
```

Figure 20

Button

This class works in a similar fashion to the BIT FIELD, but instead of reading the bit value from a source, it changes its value from 0 to 1 and vice versa everytime the button is pressed.

```
public class XorLogicalElement : LogicalElement
{
    protected override void ValueSet()
    {
        base.ValueSet();
        byte end;

        if ((inPoints[0].In != null && inPoints[0].In.value == 0) && (inPoints[0].In != null &&
inPoints[1].In.value != 0))
            end = 1;
        else if ((inPoints[0].In != null && inPoints[0].In.value != 0) && (inPoints[0].In != null &&
inPoints[1].In.value == 0))
            end = 1;
        else
            end = 0; ;

        outPoints[0].value = end;
    }
}
```

Figure 21

3.4. Art

The art of the game is mainly made with Photoshop and Illustrator. Each scene has been divided in multiple parts and then redistributed in the Unity Scene[25].
The objective of the art is to make the scenes understandable and intuitive.
The different elements that compose each scene will be described in each section.

3.4.1. Main Menu

This scene is made by a big image that acts as background (a white image that simulates an used paper), the game logo and 4 other images assigned to the button. All the components are shown in Figure 21.

Figure 22

The background and logo are not playable but the 4 images are each one linked to one of the buttons in the scene.

3.4.2. Topic

This scene is composed by the same background as Main Menu, two images to form the place to hold the subject matters, four images that are linked to buttons, each one of these images give information about the Topic that represents and another image that acts as button but for returning to Main Menu.

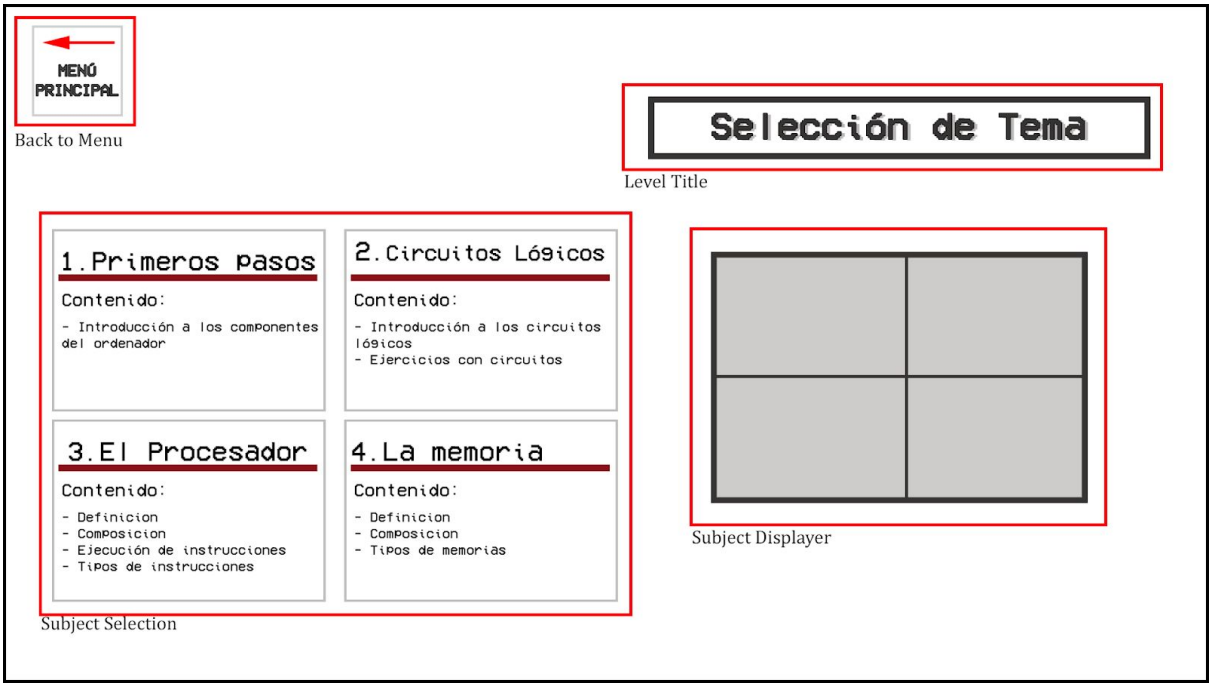


Figure 23

3.4.3. Level Selection

This scene has 4 images: the same background as before, an image that represents a selection field and an image that represents a button to go to the Topic scene.

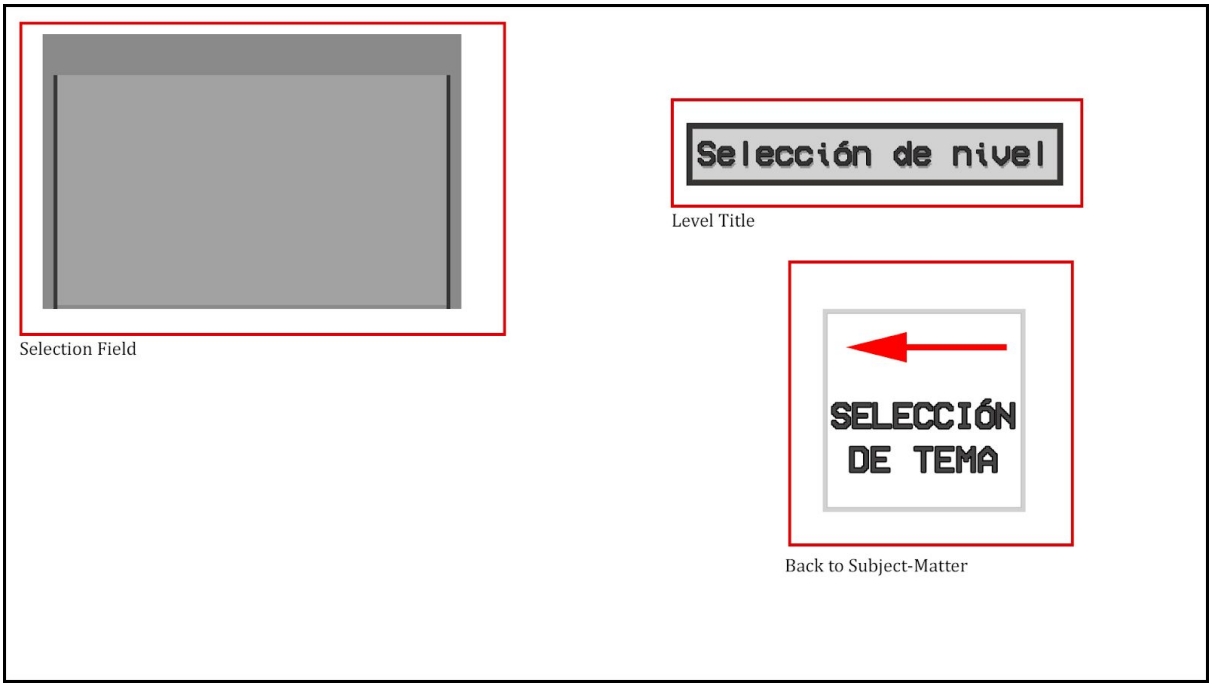


Figure 24

3.4.4. Quiz Test

This scene has a multitude of images:

- The same background,
- One to post the questions by Text in the scene,
- One to post the answers,
- An image that represents a button that goes to the Main Menu,
- One to post the Task & Hints,
- An image that represents a button that checks the answers
- One image that shows when the answer is correct
- One image that shows when the answer is incorrect

These parts are reflected in the Figure 24

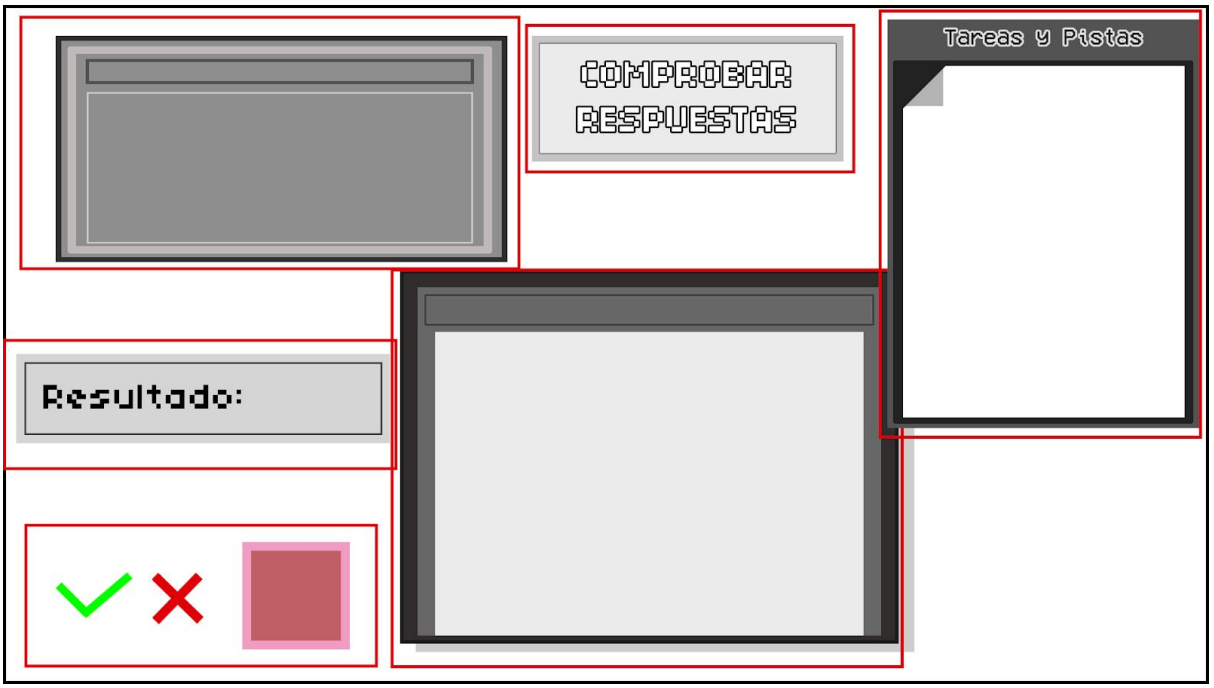


Figure 25

3.4.5. Logic Circuits

This scene has a multitude of images:

- Some of them represent the Logic Gates available
- Some of them represent the Logic Gates mode
- An image that represents a button that goes to the Main Menu,
- One to post the Task & Hints,

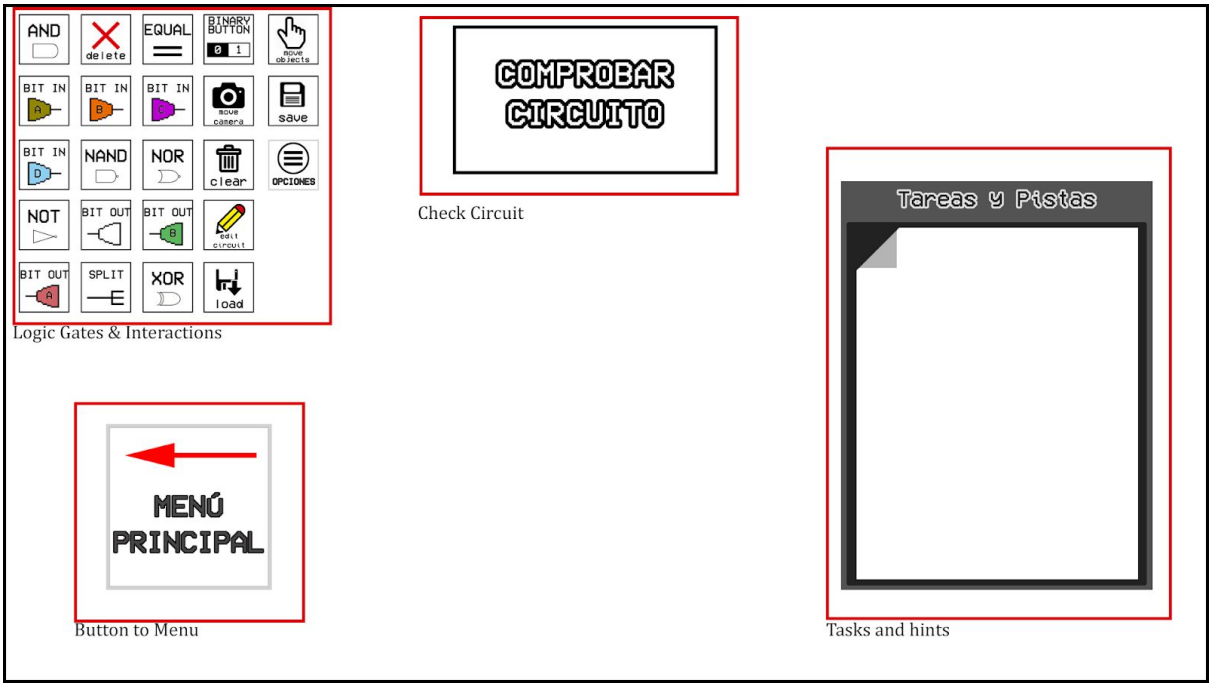


Figure 26

3.4.6. External assets

All the images in the game are copyright free due that they are original, but some of the assets in the game are external.

As Soundtrack for the game a Copyright Free song has been placed to be in loop: Chill Lofi Hip Hop Instrumental Music - Days Like These[26].

Also two Copyright Free text fonts:

- Azonix Font[27]
- Trench [28]

3.4.7. Art Table

The following Table (Table 5) portrays all the art made for the game and a link to the image.

Image Name	Link
And Gate	https://imgur.com/GRdeqMp
Equal Gate	https://imgur.com/fLpfKaz
Bit A In Gate	https://imgur.com/mk9kTAr
Bit B In Gate	https://imgur.com/AsfNytt
Bit C In Gate	https://imgur.com/BMkvOhM
Bit D In Gate	https://imgur.com/IsnUK8e
Background	https://imgur.com/rjgfhQZ
Bit Out Gate	https://imgur.com/46jFEnX
Bit A Out Gate	https://imgur.com/CW7ti89
Bit B Out Gate	https://imgur.com/MjKVAwA
Nand Gate	https://imgur.com/UVtfOV2
Nor Gate	https://imgur.com/fBamOP8
Not Gate	https://imgur.com/rSlDfwX
Or Gate	https://imgur.com/A2uTejX
And Button	https://imgur.com/MqvdujE

Delete Button	https://imgur.com/0xDZX4R
Equal Button	https://imgur.com/wIISGLT
Bit In Button	https://imgur.com/UOXAeix
Bit A In Button	https://imgur.com/CliQPXt
Bit B In Button	https://imgur.com/sQUACZS
Bit C In Button	https://imgur.com/u2PjP45
Bit D In Button	https://imgur.com/lfMqclP
Nand Button	https://imgur.com/NXdJ7ne
Nor Button	https://imgur.com/0fvFwoe
Not Button	https://imgur.com/GISoHYd
Settings Button	https://imgur.com/6uc2ZaD
Or Button	https://imgur.com/NkfiUio
Bit Out Button	https://imgur.com/8xFkmas
Bit A Out Button	https://imgur.com/AyH4zh8
Bit B Out Button	https://imgur.com/8xNQsaa
Split Button	https://imgur.com/PgT1m0e
Binary Button	https://imgur.com/fWj0btC
Load Button	https://imgur.com/X712Ofh
Save Button	https://imgur.com/Qe5Wj9w
Move Elements Button	https://imgur.com/4uEW2V9
Move Camera Button	https://imgur.com/Mi79csX
Clear Button	https://imgur.com/LXrx33u
Edit Circuit	https://imgur.com/8eDzRnr
Xor Button	https://imgur.com/KnbfFdj
Main Menu Button	https://imgur.com/aZnrFla
Level Selection Button	https://imgur.com/s39MqaB
Next Level Button	https://imgur.com/vkHpTMa

Test Button	https://imgur.com/k5lute3
Change Bit Value Panel	https://imgur.com/zUcBC3M
Check Answers Button	https://imgur.com/8eDzRnr
Exit Game Button	https://imgur.com/fjTkMjv
Free Play Button	https://imgur.com/jtd8dkQ
Change Task&Hints Page Button	https://imgur.com/W7trDrx
Task & Hints Panel	https://imgur.com/AYwZyvE
Logic Completed Panel	https://imgur.com/7YYNJ1n
Options Menu	https://imgur.com/PFfASTh
Next Level Button 2	https://imgur.com/gkbZRoU
Level Accomplished	https://imgur.com/Pja2kt1
Main menu settings button	https://imgur.com/J204y4z
Main menu play button	https://imgur.com/pHrrst3
Element placer panel	https://imgur.com/OITAd4T
Topic placer panel	https://imgur.com/NOGwORQ
Level placer panel	https://imgur.com/9owpUwJ
Correct answer image	https://imgur.com/H56s9F1
Wrong answer image	https://imgur.com/HNJKyl7
Question holder panel	https://imgur.com/JTk1ExY
Topic holder panel	https://imgur.com/jYVvwqM
Level title image	https://imgur.com/fryNGkQ
Topic title image	https://imgur.com/7AAheC3
Answer chosen image	https://imgur.com/MNNDhvl
Processor Level Finished	https://imgur.com/ptoTya9
Memory Level Finished	https://imgur.com/mverL4r
Topic 1 Button	https://imgur.com/AVPFpLs
Topic 2 Button	https://imgur.com/mwwfZpt

Topic 3 Button	https://imgur.com/wGqNRFV
Topic 4 Button	https://imgur.com/84tJbm8
Question Holder Panel	https://imgur.com/o4fPVok
Check Circuit Button	https://imgur.com/0DHlmhS

Table 5

3.5. Testing

Testing each level and mechanic is a key task for the correct result of the project. In the following sections, the testing process is described.

3.5.1. Testing Mechanics

At its most basic level the different mechanics have been tested, this includes: the Bit Field, the Button, the different logical gates (AND, OR, NOT, NAND, NOR, XOR, EQUAL), the Splitter and the Bit Display.

Once the individual elements are checked to work correctly, i created more complex circuits that i copied from a college assignment , and which Table of Truth i posses, so i could check that every result is correct.

Also the Test mechanic have been tested with every combination of answers.

3.5.2. Testing Levels

All the test have been tested, and all the expected results have been correct. The levels have been tested in a linear way (Level 1 -> Level 2 -> Level 3….), and also in a not linear way, by selecting every level from his menu.

3.5.3. Balance Testing

As this game is oriented towards the students of Basic Informatics, one of the biggest problems is the difficulty and the lack of references he might experience playing the game. To solve this, all questions have been placed in a progressive order so the student goes step by step.

4. Results

In this section the results obtained with the project are summarized

4.1. Andornot

One of the results is the game per se. An executable game in multiplatform, 4 differentiated Topics with more than 10 levels each.

4.2. Creation of levels

The Andornot Unity project can also be used to create more levels. Two scenes in the game are set up as Templates, one for the quiz and one for the Logic Gates.

Quiz Template: User can determine the number of questions and in the scene, as well as formulate them and select the correct ones.

Logic Gates Template: User can establish as far as 4 Input Bit and 2 Output Bit, as well as the Truth Table that would solve the problem.

In both scenes the user can determine what is shown up in the Task & Hints object.

5. Conclusions

5.1. Project Balance

At the beginning of the project one of the barriers that made me fear the most was been unable to represent the logic gates on the game properly. The next tables show the total work executed.

The following table (Table 5) shows the Game Analysis Work

Work	Time (h)
Computer Architecture	4
Processor	3
Memory	3
Logic Circuits	4
Total	**14**

Table 6

The following table (Table 6) shows the Level Design Work by Subject Matter

Work	Time (h)
Level 1 (Basic Computer Architecture)	7
Level 2 (Logic Gates)	16
Level 3 (Processor)	10
Level 4 (Memory)	10
Total	**43**

Table 7

The following table (Table 7) shows the Programming Work

Work	Time (h)
Circuit Manager	33
Question Manager	12
Logical Element	17
InPoint/Outpoint	11
Logic Gates (AND, OR, NOT...)	15
Total	**88**

Table 8

The following table (Table 8) shows the Assembling Levels Work by Subject

Work	Time (h)
Computer Architecture	6
Processor	12
Memory	10
Logic Circuits	8
Total	**36**

Table 9

The following table (Table 9) shows the Art Work by Scenes

Work	Time (h)
Main Menu	4
Subject-Matter Scene	5
Level Selection Scene	4
Test Scene	12
Logic Gates Scene	12
Total	**37**

Table 10

The following table (Table 10) shows the Testing Work by Subject

Work	Time (h)
Level 1 (Basic Computer Architecture)	7
Level 2 (Logic Gates)	10
Level 3 (Processor)	7
Level 4 (Memory)	7
Total	**31**

Table 11

The following table (Table 11) shows the Memory Work

Work	Time (h)
Technical Proposal	5
Game Design Document	5
Final Memory	45
Total	**55**

Table 12

The following table (Table 12) shows the Total amount of work

Work	Time (h)
Game Analysis	14
Level Design	43
Programming	88
Assembly Levels	36
Art	37
Testing	31
Memory	55
Total	**304**

Table 13

On the personal level this project has allowed me to get the study the educational games and replicate how they work. This made me understand a little bit better the position of a teacher toward the students.

One the biggest problems in my planning was that the project was more complex than I expected. This was because implementing the logic gates, thus the final solution was simple, was very hard to structure in my mind and reflect that in the game.

Nonetheless Andornot is a game that include 4 different block with multiple levels each one, and a Logic Circuit Simulator. And at the same time, a system that allows anyone to create more levels. For the future of the project I would like to get more images into to Task & Hints so everything becomes more visual.

5.2. Objectives

Objective 1: Analyse and understand the Computer Architecture
Objective 2: Determine which of this content is worth asking in a Test
Objective 3: Create a Logic Gate simulator
Objective 4: Design and implement levels and block of the videogame based in the previous Objectives.

In my opinion I think I have accomplished the goals, I have gathered information about Computer Architecture, decided what were the most important parts and based on that constructed multiple levels. I have also implemented a Logic Circuit Simulator that is also included in some other levels.

5.3. Project and Executable

- Unity Project:
  https://github.com/AntonioLopezRuiz/TFGLopezRuiz_Antonio
- Video:
  https://youtu.be/-n6kIL8XDcE
- Executable:
  https://drive.google.com/drive/folders/1HfpiclML1FSzo6MMZbJVLK1MALO1qrTV?usp=sharing
- Art of the game with source docs:
  https://drive.google.com/drive/u/1/folders/1JXM9zrzGOr3VpLVxvr8At9RnzXvGLnte

6. Annex

---

[1] Unity 3D: https://unity.com/es
[2] Basic Informatics:
https://ujiapps.uji.es/sia/rest/publicacion/2018/estudio/231/asignatura/VJ1202
[3] Linux: https://www.linux.org/
[4] Logic Circuits: https://en.wikipedia.org/wiki/Logic_gate
[5] C#: https://es.wikipedia.org/wiki/C_Sharp
[6] Educational Game: https://en.wikipedia.org/wiki/Educational_game
[7] Computer Architecture: https://en.wikipedia.org/wiki/Computer_architecture
[8] Adobe Photoshop: https://www.photoshop.com/
[9] Adobe Illustrator:
https://www.adobe.com/es/products/illustrator/free-trial-download.html
[10] GitHub: https://github.com/
[11] Processor: https://en.wikipedia.org/wiki/Processor
[12] Memory: https://es.wikipedia.org/wiki/Memoria_(inform%C3%A1tica)
[13] Hardware Engineering:
https://store.steampowered.com/app/525610/Hardware_Engineering/
[14] Shenzen IO: https://store.steampowered.com/app/504210/SHENZHEN_IO/
[15] One and Zeroes: https://store.steampowered.com/app/879600/Ones_and_Zeroes/
[16] Visual Studio:
https://visualstudio.microsoft.com/es/?rr=https%3A%2F%2Fwww.google.com%2F
[17] Game Object: https://docs.unity3d.com/es/current/Manual/GameObjects.html
[18] Start: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html
[19] Button: https://docs.unity3d.com/ScriptReference/UI.Button.html
[20] Byte:
https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/integral-numeric-types
[21] Truth Table: https://en.wikipedia.org/wiki/Truth_table
[22] Prefab: https://docs.unity3d.com/es/current/Manual/Prefabs.html
[23] Camera: https://docs.unity3d.com/ScriptReference/Camera.html
[24] Line Renderer: https://docs.unity3d.com/Manual/class-LineRenderer.html
[25] Scene: https://docs.unity3d.com/Manual/CreatingScenes.html
[26]Chill Lofi Hip Hop Instrumental Music - Days Like These:
https://www.youtube.com/watch?v=DMyudUrFlQQ
[27] Azonix: https://www.dafont.com/azonix.font
[28] Trench: https://www.1001fonts.com/trench-font.html