Development of a 3D action-adventure game with minimalist low-poly aesthetic and AI pathfinding

Ausias Garcia Torres

Final Degree Work
Bachelor’s Degree in
Video Game Design and Development
Universitat Jaume I
July 17, 2023

Supervised by: Enric Cervera Mateu

ACKNOWLEDGMENTS

First of all, I would like to thank my Degree Final Work supervisor, Enric Cervera Mateu for his help and supervision provided during these months.

ABSTRACT

This document represents the Final Degree Work report of Ausias Garcia Torres in Video Game Design and Development bachelor's degree at the Jaume I University.

This work consists of the prototype of an action-adventure game developed with Unity implementing the different mechanics of the game such as solving puzzles, exploring, and fighting enemies. In addition, the work consisted of developing assets of the game scenario modelling 3D assets and using techniques such as post-processing shaders and VFX to give it a more stylized artistic finish.

The project also uses an AI algorithm so that the enemies in the game know how to react to stimulations from other enemies and make a sense of group and coordination between the agents.

Key Words:
Stylized 3D Assets,Unity, adventure-action game, puzzles, AI.

Contents

CHAPTER 1

INTRODUCTIONS

Contents

This chapter is an explanation of what was the motivation for the development of this project, where the main idea comes from, which were the initial objectives and how it was going to be developed.

1.1 Game Concept

The Last Hope is a single player 3D action-adventure game with aspects of puzzles in which the player controls a young soldier who has to explore a ruined world and defeat its evil darkness by fighting and solving puzzles.

1.2 Work Motivation

As a great fan of action-adventure games like *The Legend Of Zelda* [1], *Tunic* [2] or *Death's Door* [3] and a student finishing the degree my purpose is to create a professional game applying the AI techniques that I had learned during the degree with stylized assets for show in my portfolio. The motivation behind this work is because I wanted to improve my AI programming and 3D modelling skills and I have a lot of experience with this game genre.

1.3 Objectives

- Create a prototype, original and professional game that can be shown in my personal portfolio
- Improve the design and production of 3D stylized assets in a professional way.
- Implement AI algorithms and techniques in order to create interesting interactions between the player and the npc's.
- Demonstrate the capacity to enrol in a professional project and finish a professional digital product for commercial purpose.

- Implement a Multi-agent System for the enemy interactions with the environment and the player.

1.4 Environment and Initial State

The project began with the idea of creating an action-adventure game inspired by games with a cartoon style and mechanics from this genre such as *The Legend of Zelda: Link's Awakening, Tunic,* or *Death's Door*. The first few weeks consisted of researching references, developing mechanics, level design, and meeting with the supervisor to finalise the project concept and organise the various tasks involved.

The game has been developed from scratch by me, except for the use of different tools like Unity, Visual Studio Code, Blender, and Substance Designer, the character models downloaded for free from the Unity Asset Store, created by Dungeon Mason and the music from OpenGameArt.org.

1.5 Related Subjects

VJ1231 - Artificial Intelligence
VJ1216 - 3D Design
VJ1227 - Game Engines
VJ1224 - Software Engineering

CHAPTER 2

PLANNING AND RESOURCES EVALUATION

Contents

This chapter shows the planning to complete the project and the resources and tools used to finish it.
The following table shows the planning of the different tasks of the project and the hours finally invested.

2.1 Planning

Task	Subtask	Hours	Hours invested
Design	Investigation	25 hours	15 hours
Design	Game Design	11 hours	13 hours
HUD	2D Art	2 hours	2 hours
3D Modelling	Scenario	5 hours	4 hours
3D Modelling	Assets	44 hours	45 hours
3D Modelling	Animate Interactable Assets	5 hours	3 hours
Programming	AI and Multi-Agent & Puzzle Programming	140 hours	140 hours
Unity	Setup Animations	8 hours	8 hours
Documentation	Writing Reports	50 hours	55 hours
Documentation	Final Presentation	10 hours	15 hours

Table. 2.1. Initial Planning Table

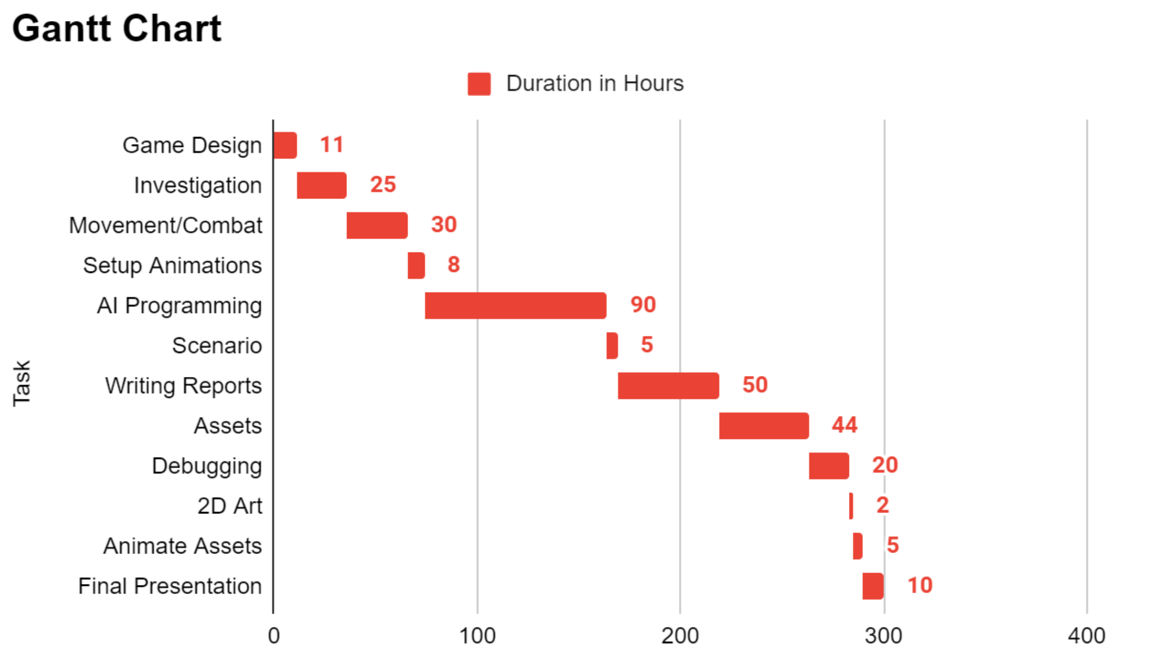


Figure 2.1: Gantt chart of the tasks

2.2 Resource Evaluation

A project of this characteristics would cost:

• Acer Aspire E 15 laptop PC, with I7-7500U Processor, NVIDIA GeForce 940MX, 8GB of RAM and 500 GB of hard disk. Cost: 550 € (it has not been taken into account because it can be developed with cheaper equipment)
• Unity 3D 2021.3.10f1 (free): The engine of the project [4].
• Visual studio code (free): Necessary to program the game. [5].
• Blender 3.2 (free): Necessary to model the 3D environment. [6].
• Substance Designer (free Student licence /20 €): For creating textures [7].
• GitHub (GitHub Desktop): a git repository where I will upload my work. It is a good tool to avoid losing progress in the project if a problem occurs on the computer[8].
• Trello: An organisation tool used to manage the different task of the project. Dividing the tasks on pending, doing and done and making more visually the work to do. [9]
• Unity Asset Store: a native Unity website for buying and selling assets. This page offers a very large amount of free assets, so it is very useful to download 3D models, animations or plugins and speed the project.

• Junior Programmer and Junior 3D Generalist or Freelance (same person): 300 hours of work * 11 = 3300€
The estimated price per hour of work has been my own assumption after reviewing several average salaries of junior programmers in Spain.

Total cost: 3870€

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

Contents

3.1 Requirements Analysis

This chapter lists and explains the functional and non-functional requirements of the project. To organise and finish the project it is necessary to make an analysis of its requirements, specify the artistic style and draw a line under how the game will work.

When running the game the player will start at the bottom of the map without any item. In front of him will be some signs explaining the inputs of the game and a chest that can be opened to unlock the sword and attack enemies(Figure 3.1). The player will encounter enemies that will attack and chase him. (Figure 3.2)

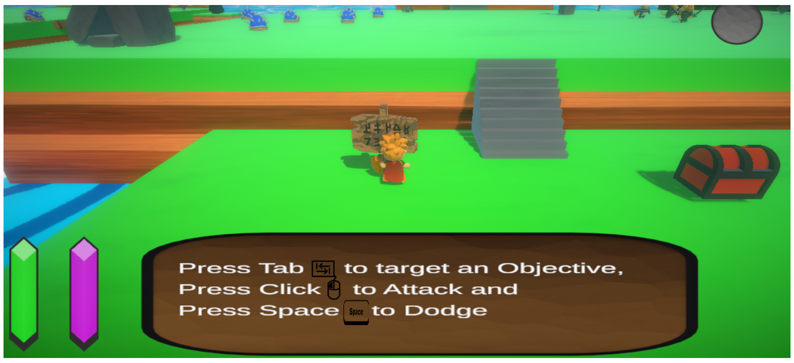


Figure 3.1: Sign explaining the inputs

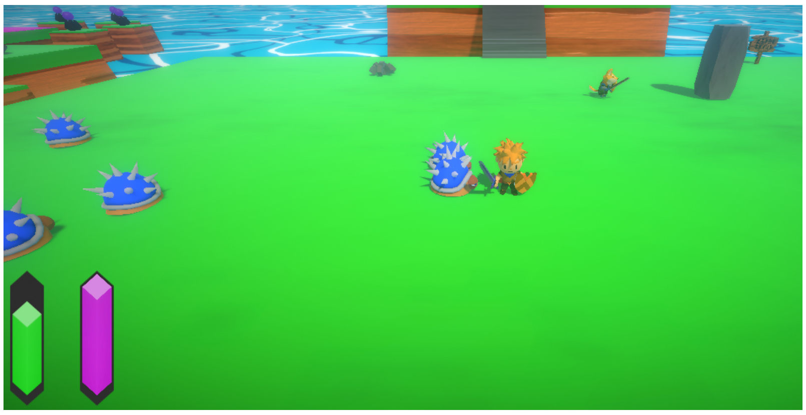


Figure 3.2: Enemies chasing the player.

Once defeated, he will find two blocked paths and one that will lead him to a puzzle with levers that he must solve by activating the levers in the correct order. By solving this puzzle, another chest will appear and unlock the ability to attack from a distance with a magic rod. (Figure 3.3) This will allow him to go down the previously blocked left path by activating a switch and deploying a bridge. (Figure 3.4)

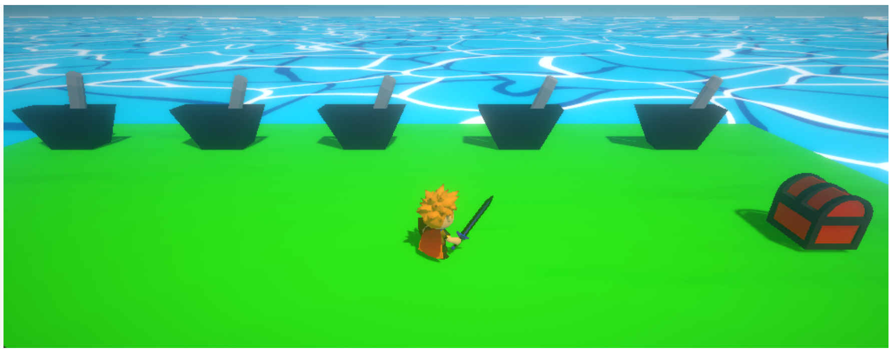


Figure 3.3: After solving the lever Puzzle

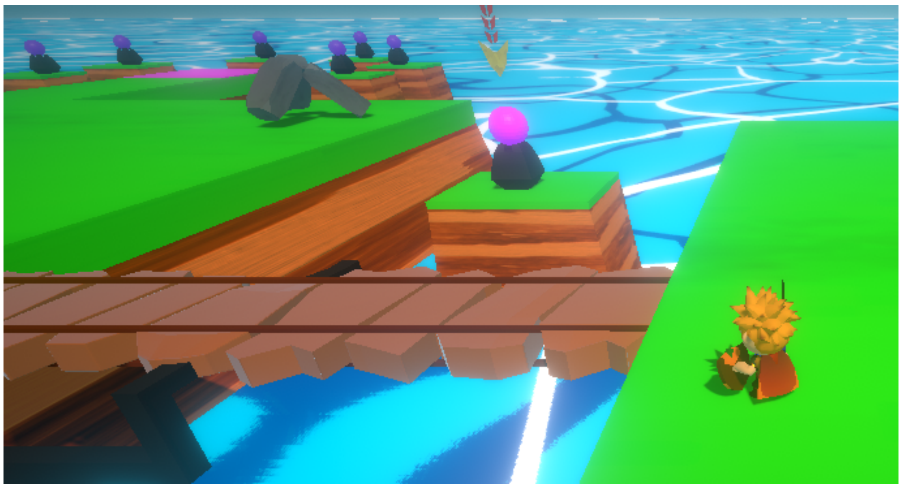


Figure 3.4: Deployed bridge after activating the switch

There, once again, they will have to defeat enemies and activate a series of switches within a certain time limit. (Figure 3.5)
In both puzzles, if the player makes a mistake, they will reset, and enemies will appear nearby. Once both puzzles are solved, a door will unlock, granting access to the final path where a dragon will summon enemies and have a more complex attack pattern. Once defeated, a victory screen will appear, followed by images that explain a bit of the game's story.

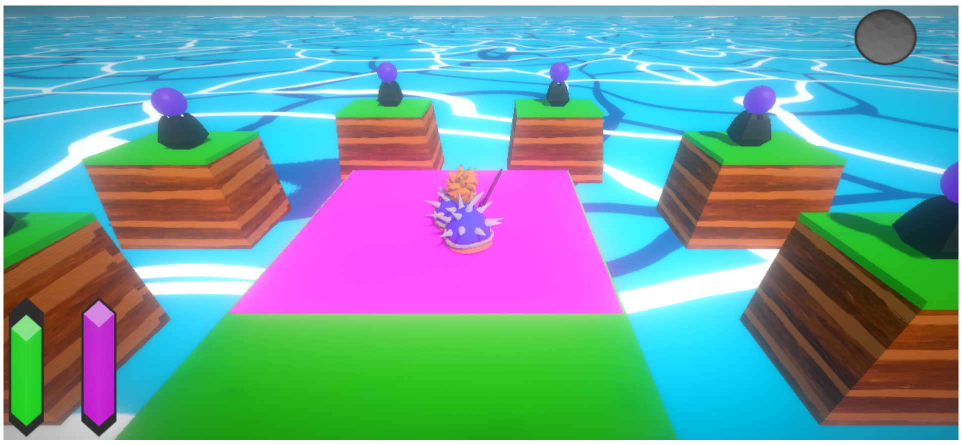


Figure 3.5: Switches Puzzle

The combat of the game will consist in dodging the enemies attacks by rolling in the ground and attacking them. If the player loses some life he can recover some life by breaking the vases of the game, they will drop some pick up life.
If the player loses, the game will show a game over screen and the player will appear at the beginning of the map but with the weapons and the puzzles (that have been solved) solved.

The artistic style will be a low poly aesthetic inspired by similar indie and triple A games of this genre (Figure 3.6). The characters will be cartoonish, the colours will be vivid and the textures will be very simple.

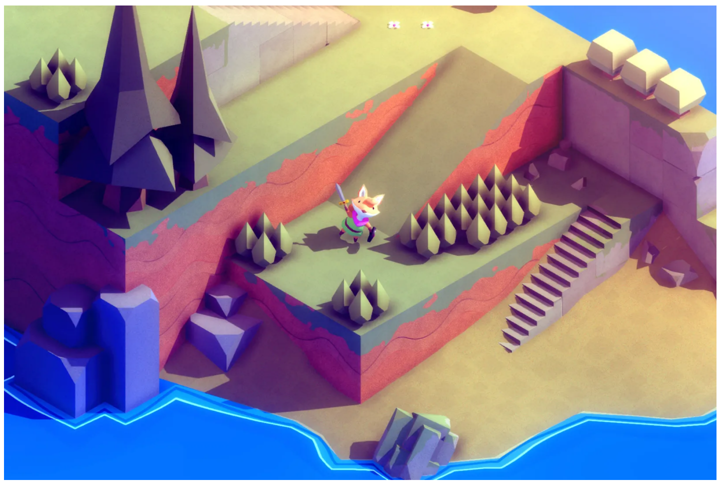


Figure 3.6: Example of the artistic style references (*Tunic*)

Functional requirement:

Once the previous explanation is clear, it is easy to identify which are the functional requirements:

- **R1**: the player can start the game.
- **R2**: the player can move through the level.
- **R3**: the player can open chests.
- **R4**: the player can fight enemies.
- **R5**: the player can check instructions and signs.
- **R6**: the player can interactuate with puzzles.
- **R7**: the player can target the objectives.
- **R8**: the player can shoot to objectives.
- **R9**: the player can break objects in the scene.
- **R10**: the player can collect items from the ground.
- **R11**: the system will be able to generate collectible items in the scene.
- **R12**: the system will be able to reset a puzzle if it is not solved.
- **R13**: the system will be able to move platforms and open doors.
- **R14**: the camera will follow the player.

Non-functional requirement:

Non-functional requirements impose conditions on the design or implementation. In this project, the non-functional requirements are:

- **R15**: the game will be playable on PC.
- **R16:** the game will use low poly models.
- **R17:** the game will have fictional characters and environments.
- **R18**: the mechanics will be easy to learn.
- **R19**: the UI and sound of the game will have good feedback to the player.
- **R20**: the camera will be placed with an aerial perspective.
- **R21**: the game will have dynamic and fluid movement.

3.2 System Design

This section presents the logical and operational design of the system. The logical design is represented through a use case diagram (see Figure 3.7). All the cases are described in these tables.

Requirement:	R1
Actor:	Player
Description:	The Player can start the game by clicking the start button.
Preconditions:	1. The player must be in the main menu screen.
Normal sequence:	1. The player clicks the start button from the main menu. 2. The system loads the level screen.
Alternative sequence:	None.

Table. 3.1: Functional requirement <<CU01. Start game>>

Requirement:	R2
Actor:	Player
Description:	The Player can move through the level and interact with physics.
Preconditions:	1. The player must be on the level screen.
Normal sequence:	1. The player presses the directional arrow keys from the Keyboard. 2. The player is in the level screen.
Alternative sequence:	None.

Table. 3.2: Functional requirement <<CU02. Move Character>

Requirement:	R3
Actor:	Player
Description:	The Player can open a chest and get a weapon.
Preconditions:	1. The player must be on the level screen. 2. The player must be in front of a chest.
Normal sequence:	1. The player presses the Q key from the keyboard. 2. The chest makes an animation. 3. The player get a new weapon.
Alternative sequence:	1. The system doesn't makes the animation because the player isn't in front of a chest.

Table. 3.3: Functional requirement <<CU03. Open chest>>

Requirement:	R4
Actor:	Player
Description:	The Player can fight and kill enemies.
Preconditions:	1. The player must be on the level screen. 2. The player must have a weapon equipped. 3. The player must be in front of an enemy.
Normal sequence:	1. The player clicks the left click button. 2. The player makes an animation attack 3. The system reduces the enemy health.
Alternative sequence:	1. The player haven't a weapon equipped and the system doesn't do anything. 2. The player isn't in front of an enemy and the system doesn't do anything.

Table. 3.4: Functional requirement <<CU04. Attack to enemy>>

Requirement:	R5
Actor:	Player
Description:	The Player checks a sign.
Preconditions:	1. The player must be on the level screen. 2. The player must be in front of a sign.
Normal sequence:	1. The player presses the Q key from the keyboard. 2. The system show a UI with a image and the respective text.
Alternative sequence:	1. The player isn't in front of a sign and the system don't do anything.

Table. 3.5: Functional requirement <<CU05. Interact with sign>>

Requirement:	R6
Actor:	Player
Description:	The Player interacts with puzzles
Preconditions:	1. The player must be on the level screen. 2. The player must be in front of a puzzle. 3. The puzzle must be reset.
Normal sequence:	1. The player presses the Q key from the keyboard. 2. The system changes the state of the puzzle to activate.
Alternative sequence:	1. The player isn't in front of a puzzle 2. The puzzle is blocked because it isn't reset. 3. The system doesn't do anything.

Table. 3.6: Functional requirement <<CU06. Interact with a puzzle>>

Requirement:	R7
Actor:	Player
Description:	The Player movement and rotation target a objective and a UI appears.
Preconditions:	1. The player must be on the level screen. 2. The player must be closer to an objective.
Normal sequence:	1. The player presses the tab key in the keyboard. 2. The system fixes the movement and rotation of the player to the objective. 3. An UI appears on the objective.
Alternative sequence:	1. The player isn't closer to an objective. 2. The system don't do anything.

Table. 3.7: Functional requirement <<CU07. Target an objective>>

Requirement:	R8
Actor:	Player
Description:	The Player shoots a magic projectile in front of him.
Preconditions:	1. The player must be on the level screen. 2. The player must have the magic rod equipped.
Normal sequence:	1. The player clicks the left click button. 2. The system generates a shoot object with different components (collider, particle system etc). 3. The system moves the shoot with a direction and a constant velocity. 4. The system destroys the object with the shoot collides with another collider or in 5 seconds. 5. The system generates a particle system. 6. The system destroys the particle system.
Alternative sequence:	1. The player hasn't equipped the magic rod. 2. The system don't do anything

Table. 3.8: Functional requirement <<CU08. Shoot>>

Requirement:	R9
Actor:	Player
Description:	The Player destroys a destructible object.
Preconditions:	1. The player must be on the level screen. 2. The player must be in front of a destructible object.
Normal sequence:	1. The player clicks the left click buttonThe system destroys the destructible object in front of the player. 2. The system generates a collectible object.
Alternative sequence:	1. The player isn't in front of a destructible object.

Table. 3.9: Functional requirement <<CU09. Destroy destructible object>>

Requirement:	R10
Actor:	Player
Description:	The Player collects a item from the ground
Preconditions:	1. The player must be on the level screen.
Normal sequence:	1. The player moves to collide with an collectible item. 2. The system restores the health of the player. 3. The system destroys the collectible item.
Alternative sequence:	None.

Table. 3.10: Functional requirement <<CU10. Collect a collectible item>

Requirement:	R11
Actor:	System
Description:	The system generate collectible items
Preconditions:	1. The player must be on the level screen. 2. The player destroys a destructible object.
Normal sequence:	1. The player destroy a destructible item 2. The system generates a collectible item.
Alternative sequence:	None.

Table. 3.11: Functional requirement <<CU11. Generate a collectible item>>

Requirement:	R12
Actor:	System
Description:	The system reset a puzzle if it is not solved.
Preconditions:	1. The player must be on the level screen. 2. The player must interact with a puzzle.
Normal sequence:	1. The player interacts with a puzzle until it is no longer intractable or a certain time passed. 2. The system reset the puzzle to its initial state.
Alternative sequence:	None.

Table. 3.12: Functional requirement <<CU12.Reset a puzzle>>

Requirement:	R13
Actor:	System
Description:	The system move platforms and open doors when required.
Preconditions:	1. The player must be on the level screen.
Normal sequence:	1. The player solves a puzzle or interacts with a door with a key . 2. The system moves a platform or destroy a door.
Alternative sequence:	None.

Table. 3.13: Functional requirement <<CU13. Move platform and open doors>>

Requirement:	R14
Actor:	System
Description:	The system moves the camera to the player position.
Preconditions:	1. The player must be on the level screen.
Normal sequence:	1. The player moves to a position. 2. The system moves the camera to his position with a certain delay.
Alternative sequence:	None.

Table. 3.14: Functional requirement <<CU14. Move camera to follow player>>

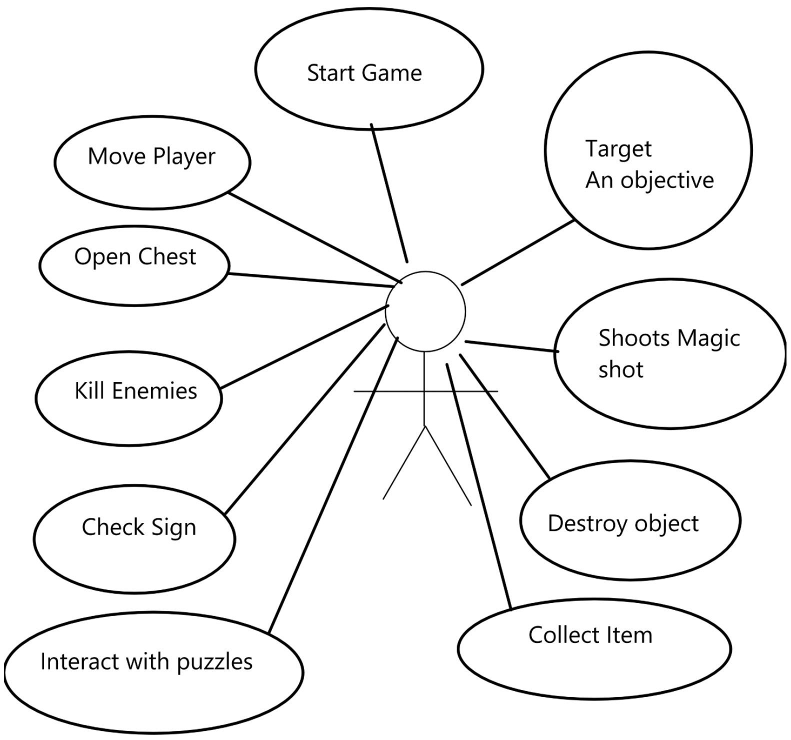


Figure 3.7: Use Case Diagram

3.3 System Architecture

This section describes the architecture of the system needed to play the game. The video game is made with Unity3D engine with 2021.3.10f1 version. For running games with this version the minimum system requirements are:

- Operating system
  - Windows 7 SP1+
  - MacOS 10.13+
  - Ubuntu 16.04+
- CPU: x86,x64, SSE2 instruction set support.
- GPU:
  - Windows: Graphics card with DX10.
  - Linux: OpenGL 3.2+

This information is taken from Unity's documentation. The game has been developed on a computer with these hardware and software features:

- Operating system: Windows 10
- Processor: Intel i7-7500U
- Memory: 8GB
- Graphics: NVIDIA GeForce 940MX

3.4 Interface Design

The game interface will have: (Figure 3.8)

- Indicators of life and energy at the top left that will serve to know the status of the character.
- Indicator of the equipped weapon on the upper right (which can be changed by pressing the E key on the keyboard).
- A text box that will appear or disappear depending on whether the player interacts with an object with text (example a sign)

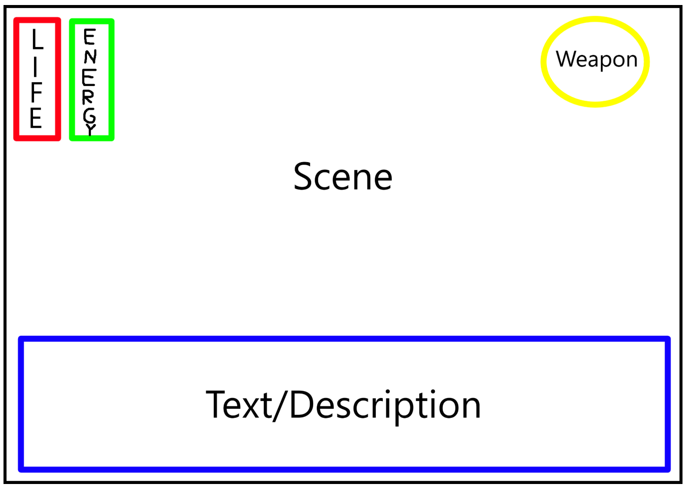


Figure 3.8: Preliminar Interface

3.5 Level Design

The normal Scenario will consist in:

In the game there will be different vases, if the player hits them he will receive life or magic. The enemies will be distributed in different areas within the level in groups of three at most. The stage will have a square or rectangular shape, although the different areas will be rounded so that it does not give the impression that the stage is artificial.(Figure 3.9) (Figure 3.10)

Challenges organization:

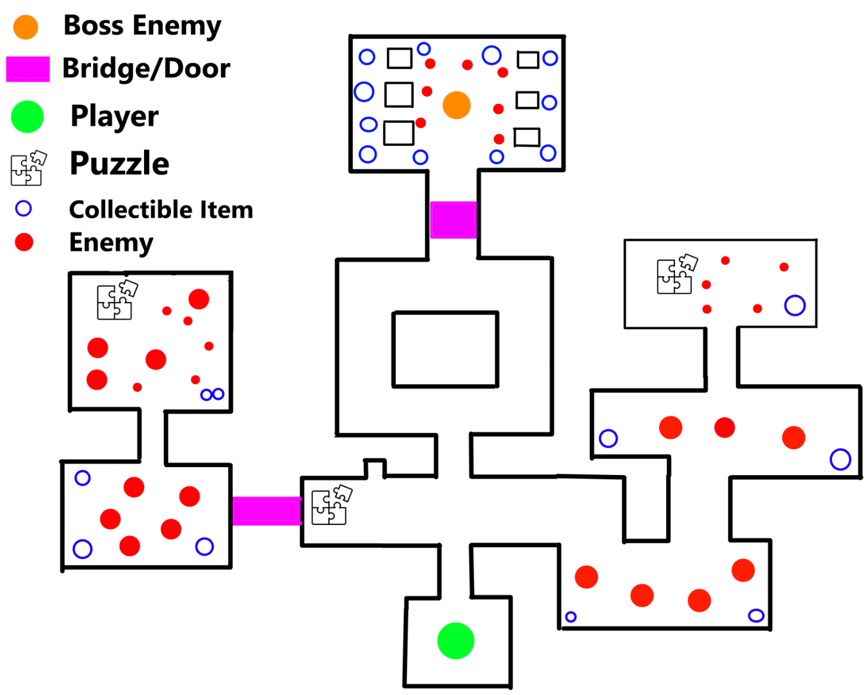


Figure 3.9: Preliminar Level Design

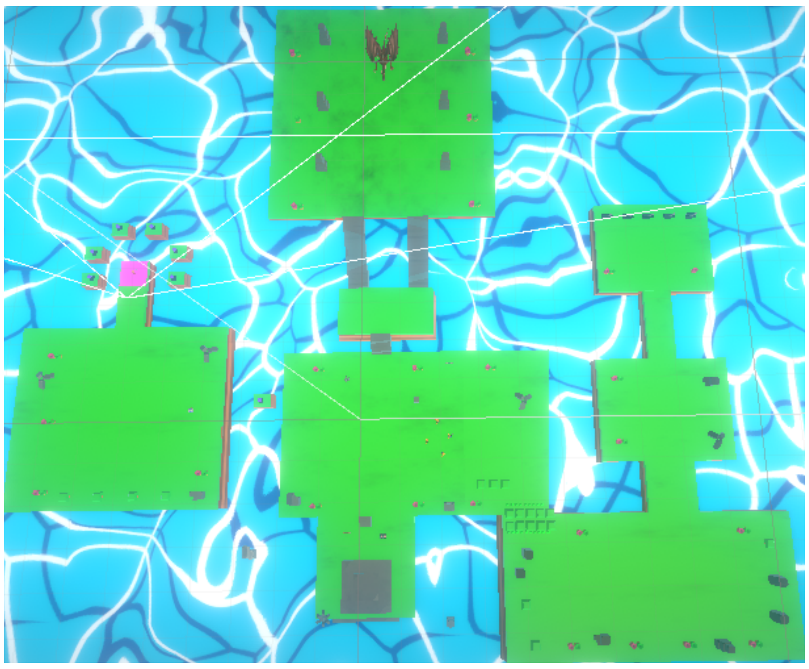


Figure 3.10: Final Level Design

The bosses level will consist in:

Closed areas where the player will not be able to exit.The scene will have vases scattered throughout the level, the final boss in the middle of the level, and randomly distributed enemies. (Figure 3.11)

Organisation

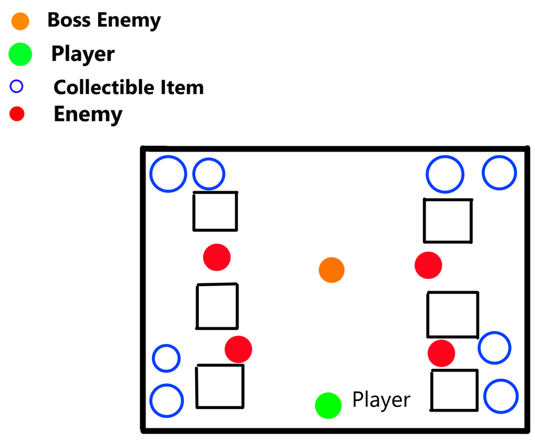


Figure 3.11: Preliminar Boss Level Design

3.6 Sound Design

Open Game Art [10] is a media repository that allows the developers to use freely licensed artwork.
All the content found on this repository is licensed under free Creative Commons licences like CC0 "public domain" or CC BY-SA 3.0 that allows the use of the content if the original author is recognized.
We will mainly use public domain sound effects and some with CC By-SA licence.
We will need background music, one for each type of fight in the game and one for the start and end menus of the game.
We will also need sound effects for when the player attacks or shoots, is attacked, solves a puzzle, gets an item, opens a chest and is defeated.
The following tables lists the sounds used in the game with their respective download links.

Related sounds with mechanics:

Mechanic	Sound
Hit	https://opengameart.org/content/10-impactshield-blocks
Death	https://opengameart.org/content/game-over-bad-chest-sfx
Collect item & Resolve Puzzle	https://opengameart.org/content/level-up-power-up-coin-get-13-sounds
Magic Shot	https://opengameart.org/content/magic-sfx-sample
Opening Chest	https://opengameart.org/content/open-chest-sfx

Table 3.15: Sounds used

Non related sounds with mechanics:

Ambient Sounds	Sound
Ambient Sounds	https://opengameart.org/content/outdoor-ambiance https://opengameart.org/content/nature-sounds-pack
Final Boss Fight Music	https://opengameart.org/content/mml-voces-desde-el-cielo https://opengameart.org/content/durchriss
Fight Music	https://opengameart.org/content/last-angel-soundtrack
Menu Music	https://opengameart.org/content/underwater-theme-ii

Table 3.16: Sounds used

4

ARTISTIC DESIGN & DEVELOPED ART

Contents

This chapter explains the art style of the project, the external resources used and the delivered art and developed shader to complete the project.

4.1 Artistic Design

Based on the references, it is concluded that the 3D models of the characters and the environment should be low poly, with a fantastic and cartoonish aesthetic. Additionally, the interface should be simple & concise. Furthermore, considering the level design, there should be a texture or shader of the sea in the background to enhance immersion.

4.2 Characters

We need characters to follow this art style so, as the project does not focus on the development of characters and animations, to speed up the work we will use characters imported from the Unity Asset Store with their textures and animations and developed by Dungeon Mason [11].

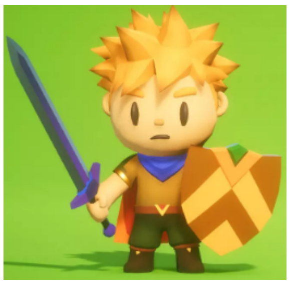


Figure 4.1. The Protagonist, imported model from Dungeon Mason (Asset Store RPG Polyart)

After being shipwrecked, this brave soldier from distant lands will wake up on this mysterious island and must defeat the evil that lies on it. He has the appearance of a child and even though he is not armed (he later finds a weapon) he is agile and can quickly dodge enemies by cartwheeling on the ground. (Figure 4.1)

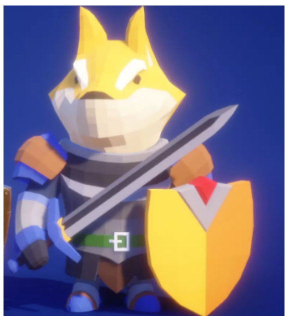


Figure 4.2. Dog Knight, imported model from Dungeon Mason (Asset Store Modular Animal knight)

Basic enemy, former citizen of the world. Classic medieval knight of short stature but with an animal appearance. He is armed with a short sword and a shield.(Figure 4.2)

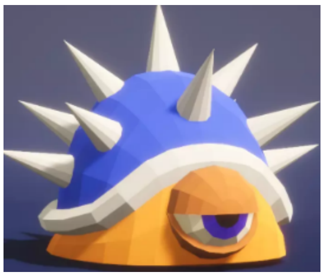


Figure 4.3. Turtle Shell, imported model from Dungeon Mason (Asset Store RPG Monster Duo)

Strange rounded creatures that inhabit the world stand out for being weak but fast. (Figure 4.3)

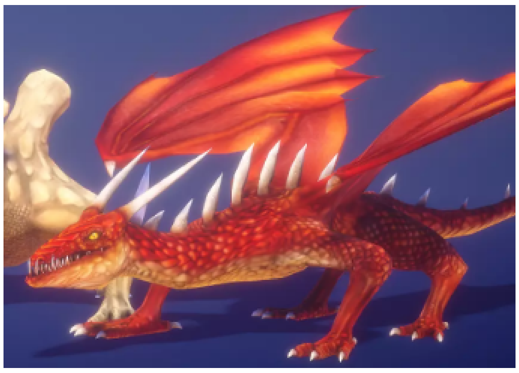


Figure 4.4. Dragon Boss, imported model from Dungeon Mason (Asset Store RPG Monster Duo)

Final Boss of the game with fast attack, the capacity to fly and throw fire. You need to defeat him to finish the game.(Figure 4.4)

4.2.2 Character animation process

To implement the dodging mechanic, you need to create a dodge animation since the character from the imported resources to the project doesn't have it. To do this, simply import the 3D model into Blender and modify the bone positions of the model, especially the back and head parts, and save their positions in different keyframes. Some of these keyframes will be automatically adjusted due to interpolation. (Figure 4.5)

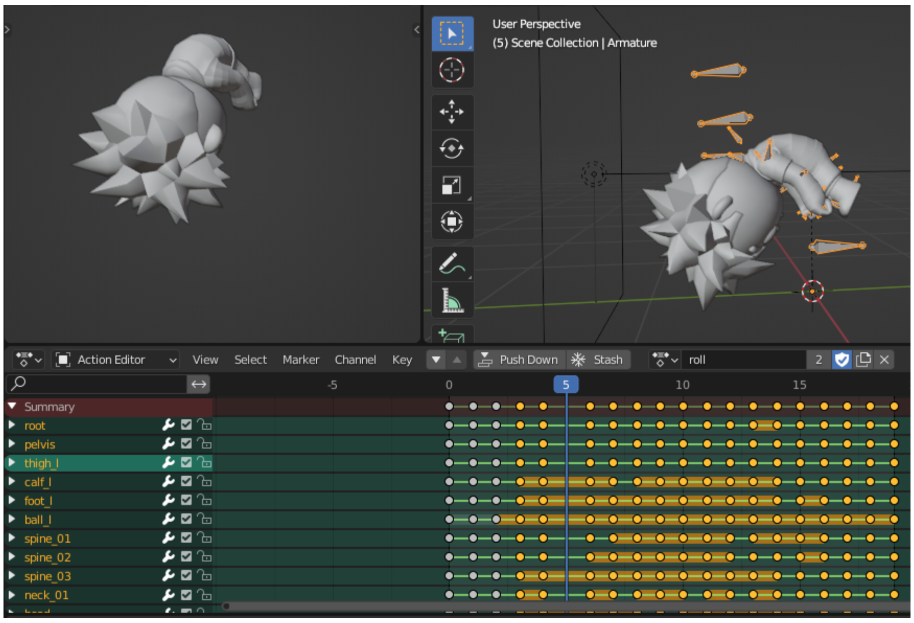


Figure 4.5: Dodge Animation In Blender and Keyframes

4.3 Props and animations

4.3.1 Work Delivered

The following images show the assets made and their references. The references have been taken from the *Tunic* and *The Legend of Zelda* or made by me for the GDD of this project

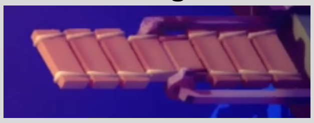
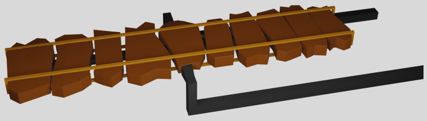
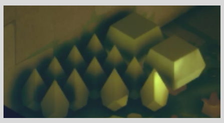
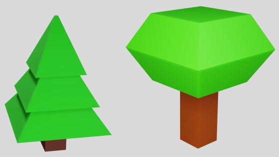
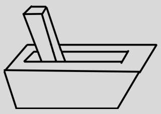
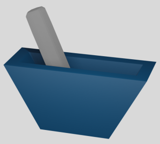
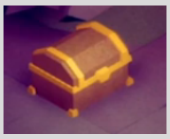
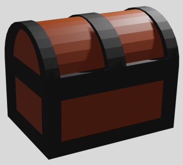
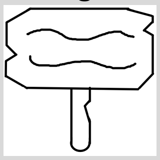
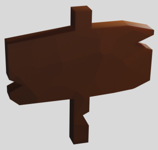
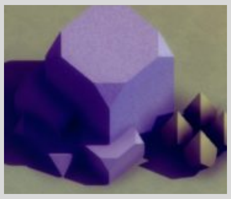
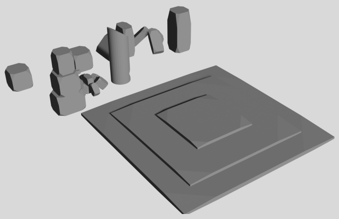
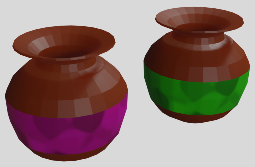
Reference	Work Delivered
Bridge 	
Trees 	
Lever 	
Chest 	
Sign 	
Rocks 	
Pot	

Table 4.1: Delivered 3D work done during the project.

4.3.2 Prop animation process

The animation process has been carried out using Blender's 3D bone system, along with the assistance of its interpolation animation.
For the lever animation, a keyframe has been animated where the lever is in a static position (which also serves as the idle animation), and another keyframe rotates the lever positively along the X-axis for the activation animation and negatively for the deactivation animation.
The same approach has been used for the chest animation, animating the top part. In this case, there is an idle animation and an animation for opening the chest.(Figure 4.6)

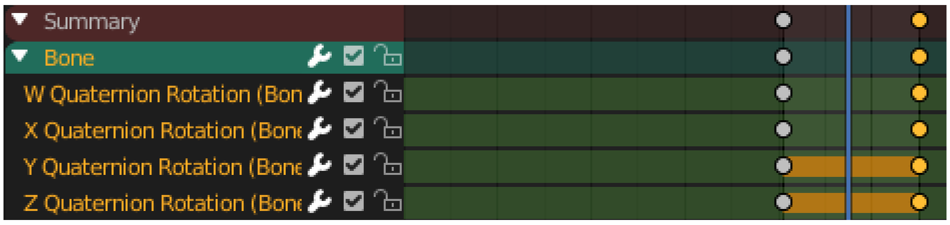
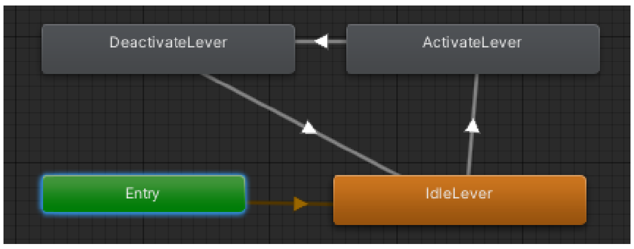
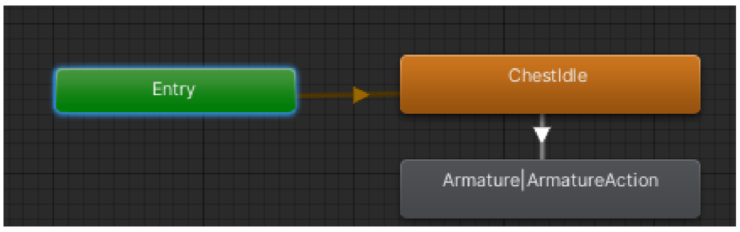

Summary
Bone
W Quaternion Rotation (Bon
X Quaternion Rotation (Bone
Y Quaternion Rotation (Bone
Z Quaternion Rotation (Bone

Figure 4.6:Keyframes in the Blender Action Editor

Later on, an Animator component was added, and different variables were declared to control the execution of the animation.(Figure 4.7, a-b)



(a) Animator State Machine (lever)



(b) Animator State Machine (chest)

Figure 4.7:Unity's Animator State Machines

4.4 Textures

For texturing, Adobe's Substance Designer program has been used, which is widely used in the professional production of 3D texturing and VFX in the gaming industry. It provides image editing through nodes and procedural textures (or user-provided textures), which can later be layered, distributed randomly as patterns, and more. Substance Designer is similar to Unity's Shader Graph but specifically designed for the production of textures (such as Color, Normal, Ambient Occlusion, etc.) only.

For the texture of the sign(Figure 4.8), two procedural textures have been used: B&W Spots and Grunge Map. They have been smoothed out using the levels node (which makes black

pixels grayer and white pixels grayer depending on the settings). They have been merged in the Blend node and converted to colour using the Gradient Map.
For the ground texture(Figure 4.9), the same process has been followed but with a single texture.

For the Mountain Texture (Figure 4.10) you use a Directional Warp node to make a curly rectangle, then a Tile Sampler to expand the same texture in columns and rows and next you can mask the texture with a blend node and a noise node and convert the result to colour with the Gradient Map.

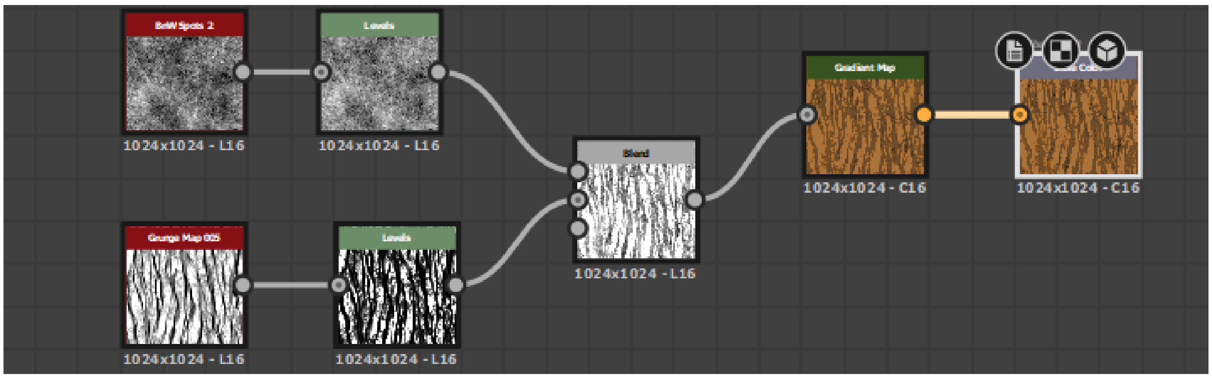


Figure 4.8:Graph of the Sign texture

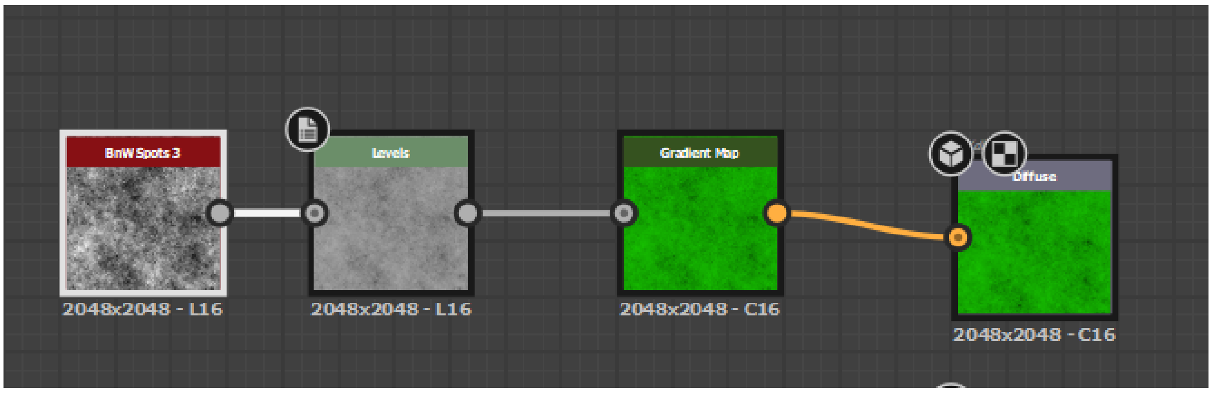


Figure 4.9:Graph of the Ground texture

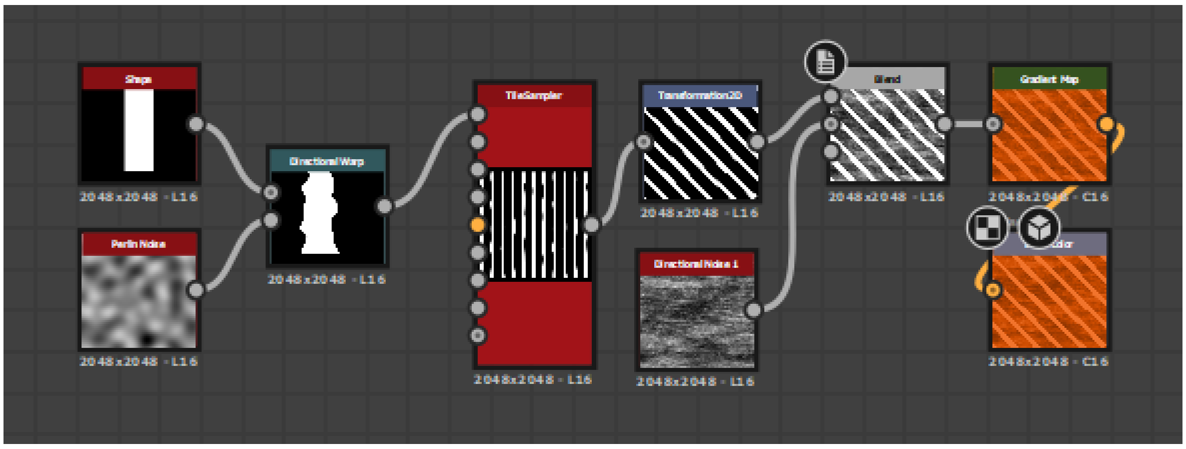


Figure 4.10:Graph of the Mountain Texture

4.5 UI/Interface

The UI images have been made by rendering them in Blender and modeling them in the same way as the assets.

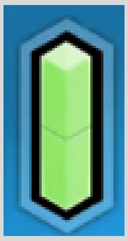
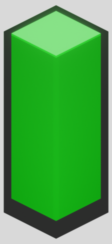
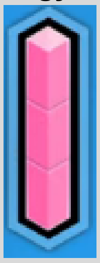
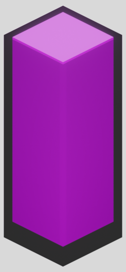
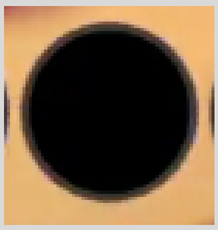
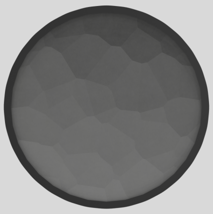
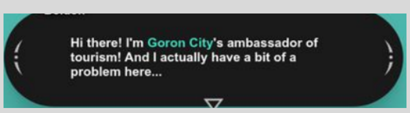
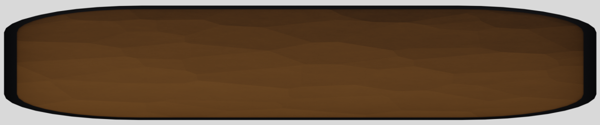
Reference	Work Delivered
Health Bar 	
Energy Bar 	
Weapon Menu 	
Text Box  Hi there! I'm Goron City's ambassador of tourism! And I actually have a bit of a problem here...	

Table 4.2: Delivered 2D work done during the project.

4.6 Shaders

As an extra feature I wanted to refine the game by adding shaders as well as adding a post processing effect with the Unity tools, this section explains how these tools have been used for the final result.

To give a more stylized effect to the sea, a water shader inspired by *The Legend Of Zelda*, the Shader Graph tool from Unity and the free resources from Daniel Ilett's tutorial [12] have been used, for this you have to add this tool in the Package Manager window.

Once this is done and with the FlowMap and Voronoi textures, we create a material and an unlit Shader Graph and open the Shader Graph, change the Active Target from Universal to Built-in in the Graph-Inspector window and mark Allow Material Override as true .

Once this is done we add the different variables, float Color and Texture2D as in the tutorial, these will help us to adjust the speeds, size, depth and colour of the Shader.

Once here, the different nodes of the graph are added, to make the UVs of the texture move and give a sensation of moving foam, the Time node is used and it is added to a UV node, to change the speed, the Time node is used. Flow_speed variable (Float 0.04) is divided by size (Float 2) and this is multiplied by Time as shown in the image. (Figure 4.11)

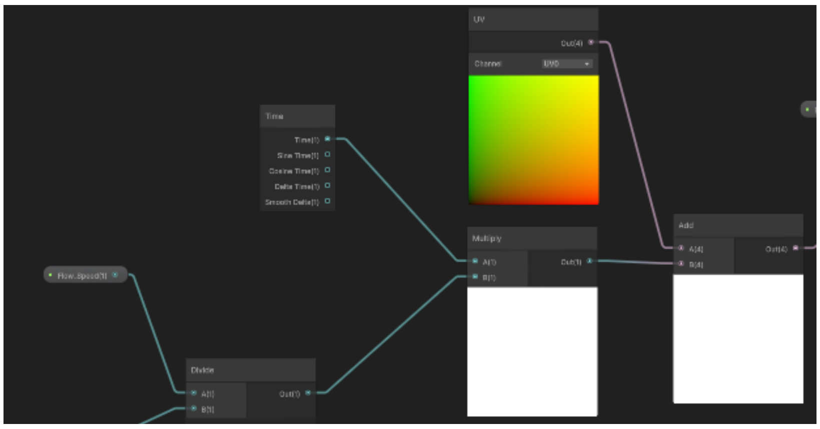


Figure 4.11: Time node connected to the UV

This Time node is applied to a normal Texture (FlowMap) to give that movement to the foam with a Flow Strength (Float 0.01). (Figure 4.12)

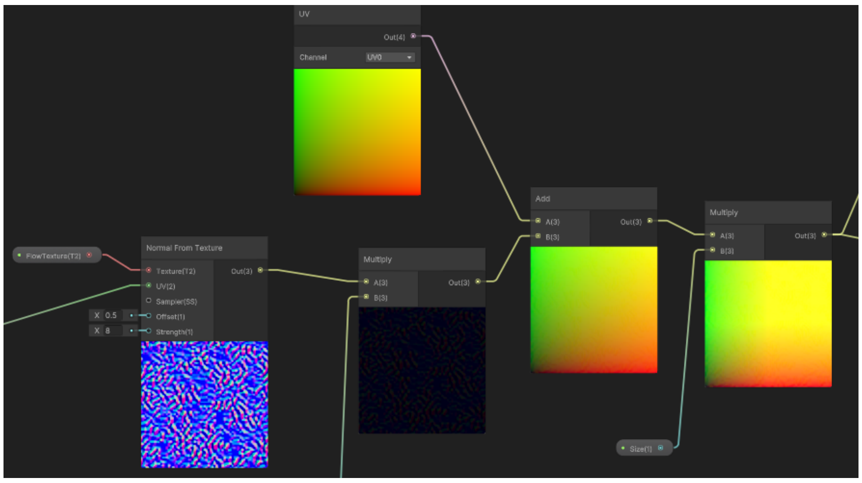


Figure 4.12: FlowMap Texture

Finally, to add colour to the shader, we use the sampleTextures (Voronoi) and Lerp nodes that interpolate the colour of the texture and the colour that we want, in this case dark blue, light blue and white, the last node is connected to the Color base of the Fragment Shader. (figure 4.13)

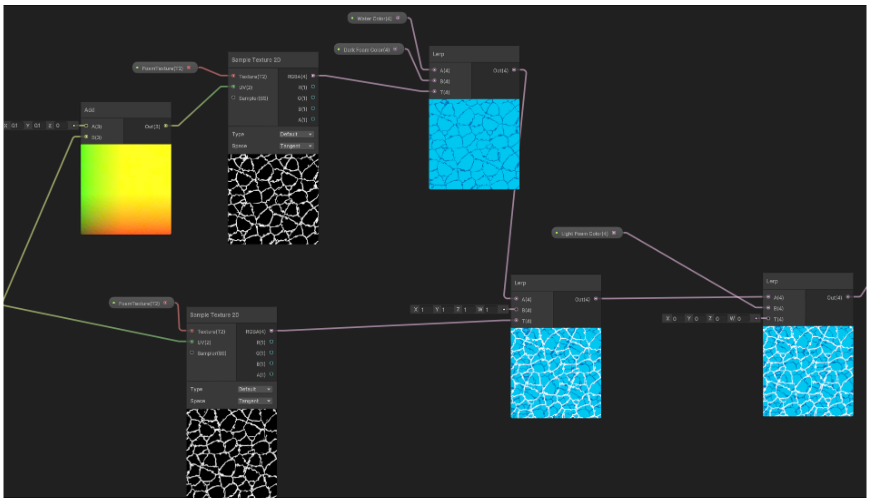


Figure 4.13: Lerp nodes

As an addition, if we want there to be a wave that moves on the Y axis, the position of the Object with the position node must be made to move on this axis at a certain time, therefore the Time node is used, the effect with the Choppiness variable (Float 0.01) and to only take the Y axis of the object a Vector 3 node is used, this is connected to the position of the Vertex Shader. (Figure 4.14)

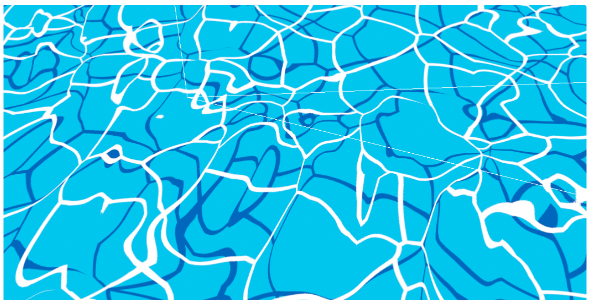


Figure 4.14 Water Shader Result

4.7 Unity Post-Processing Volume

To add a Post Processing object in Unity we must add an empty object to the hierarchy and add the Post-Process Volume component and create a profile in Profile New, if this component is not found, the Post Processing package must be added in the window Package Manager.
Next we add to the project a layer for the post processing and we put the created object in this layer. In our camera we add the Post-Processing Layer component with our camera in the variable Trigger and layer of the Post-Processing.
Once this is done we add to the Post-Process Volume the effects:

Bloom: To have a more interesting lighting. The intensity at 9, the threshold at 0.9 and Fast Mode true so that it doesn't consume so many resources.
Vignette: Adds a dark border to the screen. Intensity to 0.2 and Smoothness to 1
Ambient Occlusion: To add less flat shadows. Mode Multi Scale Volumetric Obscurance, Intensity 3, Thickness Modifier 1, Ambient only true.
The next figures show the difference before and after adding the post-processing (Figure 4.15) (Figure 4.16)

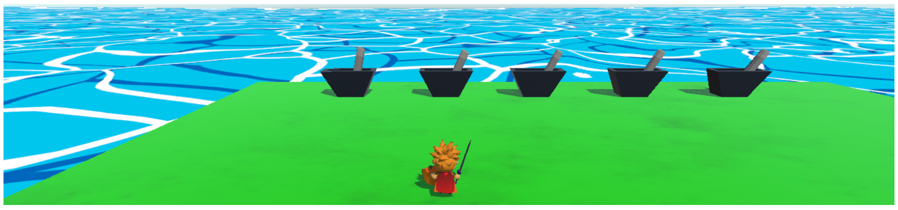


Figure 4.15: Game without Post-Processing

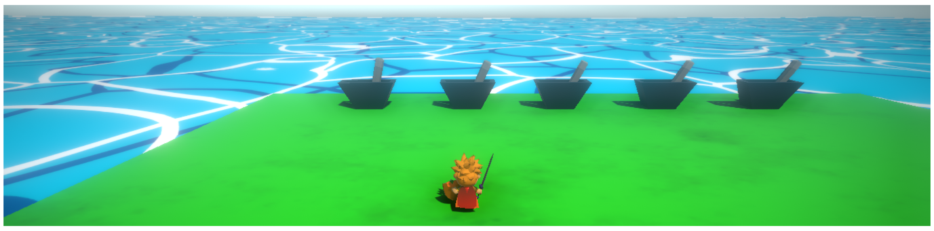


Figure 4.16: Game with Post-Processing

CHAPTER 5

PROGRAMMING WORK DEVELOPED AND RESULTS

Contents

This section explains the programming techniques, programming tools and code used to achieve the functional and non-functional requirements of the game.

5.1 Camera Positioning System

The camera needs to follow the player horizontally as the player progresses through the level, from an isometric perspective, plus the player does not have the ability to rotate it around him.

To achieve this result, a Unity tool called Cinemachine has been used to automate this process.
To add this tool to Unity you need to open the Package Manager window and in the Unity Registry tab search and install the tool. This can be done from Unity version 2019.4 onwards.

Although I have used this tool once I don't have so much experience with it, so at first I used the FreeLook Camera but later I realised that the Virtual Camera was easier since the Free Look Camera has three Axis that control the vertical camera and horizontally and lowering the camera speed to 0 vertically was not optimal, therefore the Virtual Camera was used.

To make the camera follow the character and centre its position on the screen, it is only necessary to add the Transform component of the character to the variable that is given by the tool.

Then you have to add the cinemachine Brain component to the main camera and set the Blend Update Method to Late Update since the character will move in the update method and moving the camera in the FixedUpdate method can cause refresh problems.

To have an isometric perspective of the character, you can modify its position, the FOV, etc.

5.2 Player Movement/Combat Mechanics

Character Controller

I encountered several difficulties while programming the movement. Initially, I implemented the character's movement using Rigidbody physics, but I faced challenges in managing it effectively. Moreover, I found that using Rigidbody physics alone couldn't handle traversing steps unless specifically programmed. Eventually, I switched to using the character controller, which presented its own set of problems (such as the player climbing on top of enemies), but I was able to address and manage them more effectively.

To move the character around the scene, you need to capture their horizontal and vertical inputs. You can then create a `Vector3(movementDirection)` using the horizontal input for movement along the X-axis and the vertical input for movement along the Z-axis.

Afterward, you can move the character using the function `characterController.Move (movementDirection)`.

Furthermore, as a gameplay mechanic in adventure games, a dodge or dodge spin move was implemented to allow the player to evade attacks. The player can move as described before if they are not dodging. However, if the space key is pressed, a subroutine is triggered. During this subroutine, an animation is played, and the character's collider is reduced, making it more difficult to hit him.(Figure 5.1, a-b-c)

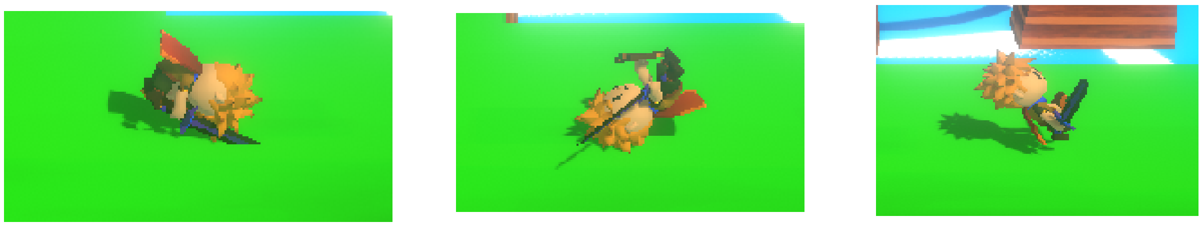
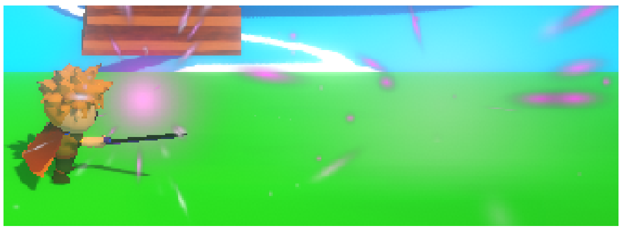

(b) Animation Keyframe (b) Animation keyframe (c) Animation keyframe

Figure 5.1 Dodge animation

Magic Shot

To make the shoot I made use of the SM Channel particle System[13] and programmed the logic to detect when the shot collides with the Sphere Puzzle or with the enemies.
The way to do it is to make an instance of a shooting prefab, which when created it moves forward with a force and creates a particle system that is destroyed when it impacts with another object or after 5 secondsThis generates a problem if the character has extra colliders, so I had to fix it by putting the shot in a separate layer and disabling the collisions of that layer with the character layer in the Physics tab of the project settings.

In addition, to give a more realistic effect to the shot, another particle system is generated when it hits something as can be seen in the next figures. (Figure 5.2 a-b)


(a) Initial particle effect of the shot

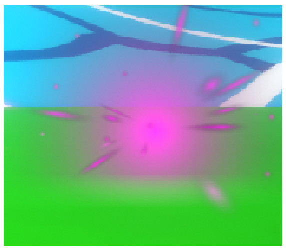

(b) Intermediate Particle effect

Figure 5.2. Particle effect of the shot

Target Lock or (Z Targeting) mechanic

This game mechanic from action games consists of blocking the camera or the movement of the player so that he can attack a specific enemy.

To do this in Unity you need to detect the nearby targets by inserting each gameObject into a layer and collecting the nearby colliders with an array using the `Physics.OverlapSphere` function. `Physics.OverlapSphere(Vector3 position, float radius,int layerMask );`
Then the new direction to which the character will have to rotate is calculated by subtracting the position of the target from the position of the player.
And the rotation of the player is changed with `Quaternion.LookRotation` and `Quaternion.RotateTowards`. This only happens when the target is blocked, when it isn't blocked the rotation of the player is about itself, that is, the one introduced by the player's controls (`Input.GetAxisRaw`).

To indicate that an enemy has been fixed I have added an animation to a child object of the objective that will be activated and through a subroutine it will be activated or deactivated.
During the subroutine if the Tab key is not pressed the subroutine will stop with yield return null. (Figure 5.3)

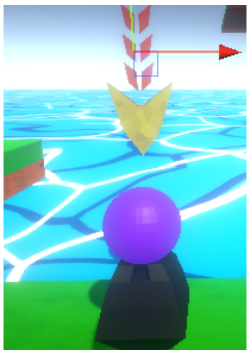


Figure 5.3. Target UI

5.3 AI Enemies

One of the project ideas was to implement a flocking system, which involves different agents acting as a group and exhibiting separation, cohesion, and alignment behaviours. Separation allows agents to maintain a certain distance from other agents, preventing overcrowding or collisions. Cohesion enables agents to come together and find the average distance between multiple agents in order to position themselves accordingly. Alignment provides agents with the ability to align their movement with that of other agents.
This information can be found in: [14][15]
After several hours of work and researching information from projects that implemented this in Unity, it was generally used as more of an experimental feature or for 2D projects.

It was possible to make the agents detect nearby agents and separate from them at a distance to avoid collisions.A MoveToGoal class was also implemented, which takes into account the player's position. This was necessary because when considering only the positions of other agents, the enemies would pass through the player. However, I was not satisfied with the outcome and struggled to get the rest of the features to work properly. As a result, I set that part of the project aside for a while. However, at the last moment, I decided to use a simpler AI behaviour that involves individual pathfinding towards the player, as I found it too difficult to continue the project using the previous approach. The logic for avoidance between agents can be found in the AiAvoidance class of the project.
(Figure 5.4)

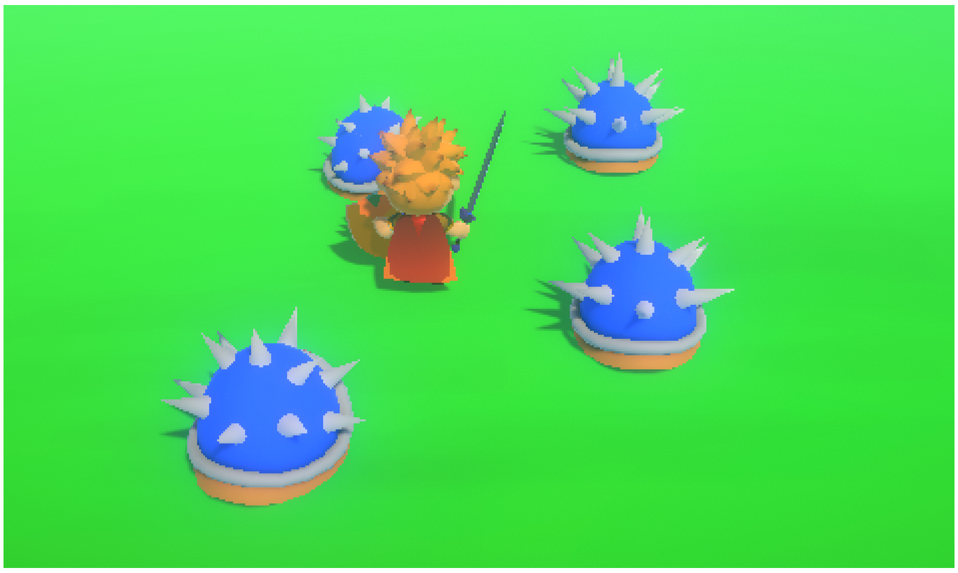


Figure 5.4. Avoidance results

NavMesh

The logic followed for the pathfinding with NavMesh is:
When enemies are not close to the player, they will have a random movement towards a nearby position.A timer will change the state of the enemy from being stationary to being in movement. To prevent the enemy from going to an unreachable position, it will first calculate if it can complete that trajectory, and if it can, it will move to that position. Thanks to the NavMesh Obstacle component, it will detect other agents as obstacles and take this into account when calculating the trajectory.

When the enemy is close to the player, it will alert nearby enemies using an overlapSphere check, and they will start attacking the player. When calculating the new route towards the player, the position of other enemies will be taken into account by utilising the NavMesh Obstacle component, which will be updated as they pursue the player.

5.4 Puzzles and Interactable Objects

Lever Puzzle

In this puzzle, 5 levers are positioned on the stage and to solve the puzzle the player needs to activate the levers in the correct order, if they are activated in the correct order a chest will appear on the stage with an object for the player, if not, they will reset to their initial state and spawn enemies across the stage (as seen in the Figure 3.3)

To manage this, on the one hand we will have the levers with the components, Animator: to manage the animation, Rigidbody and Box Collider to manage the interaction with the character, audiosource to play a sound when the lever is interacted with, and LeverSystem that will have two boolean variables. boolchecker which will enable or disable the toggle and flag which will block its interaction (explanation of why to block it is explained in the system controller below).
Upon contact with the character it will be activated with the boolchecker variable and it will be blocked with the flag variable, an animation will also be played and a sound will be played, since we want the sound to only be played once we will also use a flag for the Audio.

On the other hand, to know if they have been activated in the correct order, they all communicate with a controller of the lever system called the Switch System, which has access to the levers and a string which will check if the order entered by the player is correct, if it is it will make the chest appear and play a sound and if it is not it will unlock all the levers with "flag = false" it will reset its state with "leverSystem .boolchecker = false".

So that the controller knows that lever x corresponds to number x, an if is used for each lever in which the lever number is passed to string once it has been activated and a counter checks the combination if all 5 have been activated.

Switch to deploy bridge

To access the left side of the level, a switch must be activated that will deploy a bridge. To achieve this, the bridge needs to move slowly to a specific position. Therefore, when the switch is activated, the bridge will calculate its position and the midpoint position it needs to move to (using an invisible object located there), and it will move to that new position using the translate method. By multiplying the movement by Time.deltaTime, the animation will be performed based on the game's current frames, avoiding instant movement. To activate the switch, the collision between the player's shot and the switch will be detected, and if the shot impacts, the Renderer of the switch will change the colour of the material to indicate that it has been activated. (Figure 5.6)

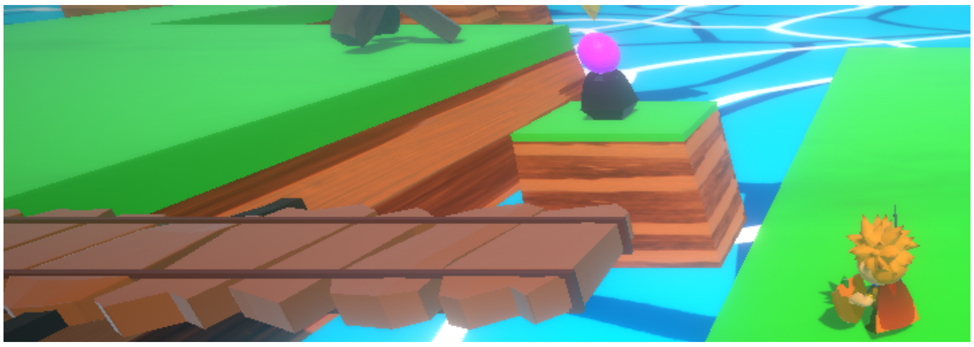


Figure 5.6. Deploying bridge

Switch puzzle

Another puzzle has been implemented based on the previous information. This puzzle involves activating all the switches within a time limit. If the player fails to do so within the specified time, the switches will reset, and enemies will appear. To accomplish this, the SwitchTimerUnit class is responsible for individually detecting whether a switch has been activated or deactivated and changing its color accordingly. The SwitchTimerManager class is responsible for determining whether all the switches are activated or not. Each time the first switch is activated, a coroutine will be initiated that will count a few seconds using the function "yield return new WaitForSeconds(seconds);". If the player fails to activate all the switches within that time, they will be deactivated, and this information will be communicated to the previous class. Also, a platform has been added that will reload the player's shots to prevent him from being unable to complete the game. (Figure 5.7)

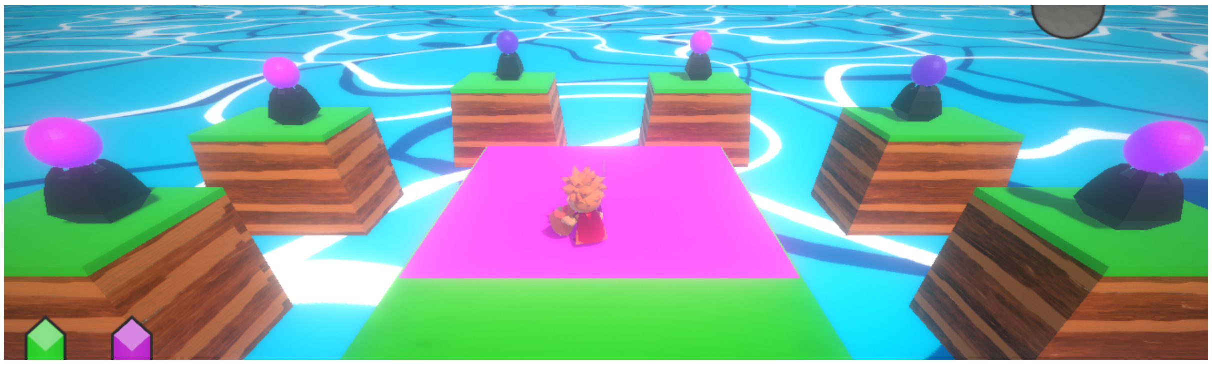


Figure 5.7. Switch Puzzle

5.5 Interface Display

Sign Box

To display text on the screen, the Unity Canvas has been utilised, with various texts indicating the game controls or providing hints on how to progress if the player gets lost. Different signals have been distributed throughout the game, and when the player approaches them, their position is detected using triggers, triggering the display of the corresponding text on the screen. (Figure 5.8)

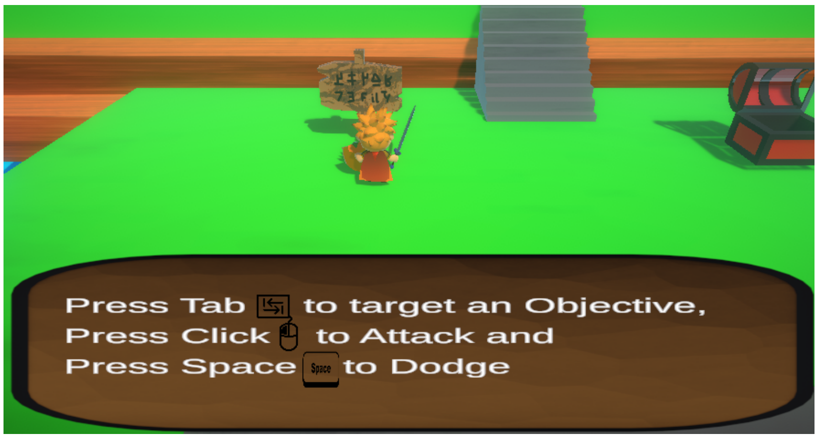

Press Tab to target an Objective,
Press Click to Attack and
Press Space to Dodge

Figure 5.8. Text indicating the game controls

Feedback from interactable objects

To let the player know which objects they can interact with, a system has been implemented using colliders. When the character is near an object, an icon will appear indicating the key they should press. (Figure 5.9) The image is a free icon from iconos8.es

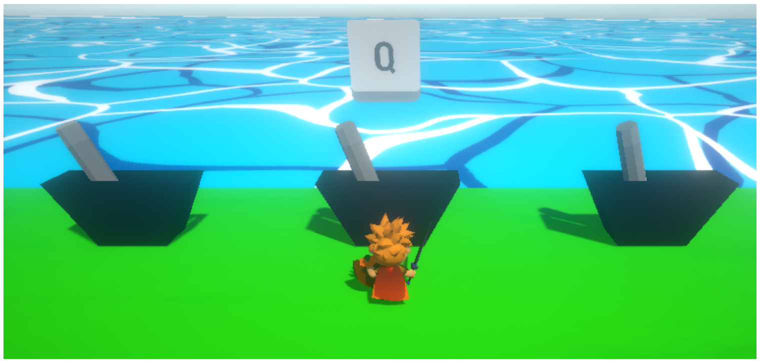
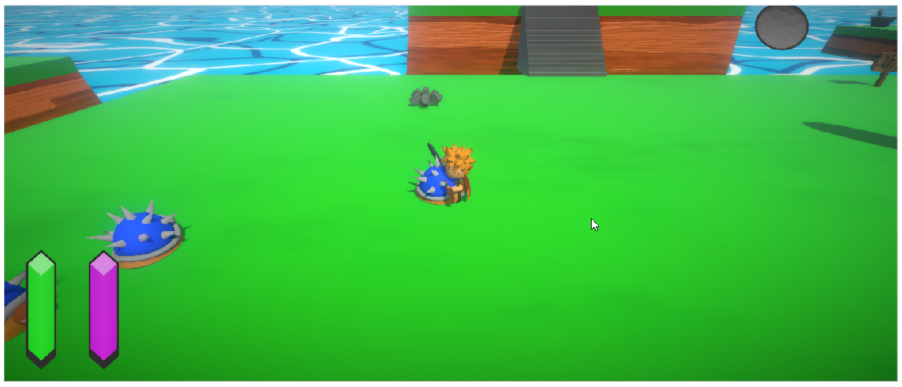


Figure 5.9. Icon to indicate that the player can interact

Life and magic feedback

To indicate when the player has been hit, the PostProcessManager class has been modified to apply a brief red "Vignette" effect (Figure 5.10) to the camera, along with a shaking animation of the Cinemachine coordinates, triggered when the player gets hit.

In addition, a system has been implemented where a portion of the green health bar disappears when the player is hit, and it reappears when the player collects a health item.


(a) Default UI

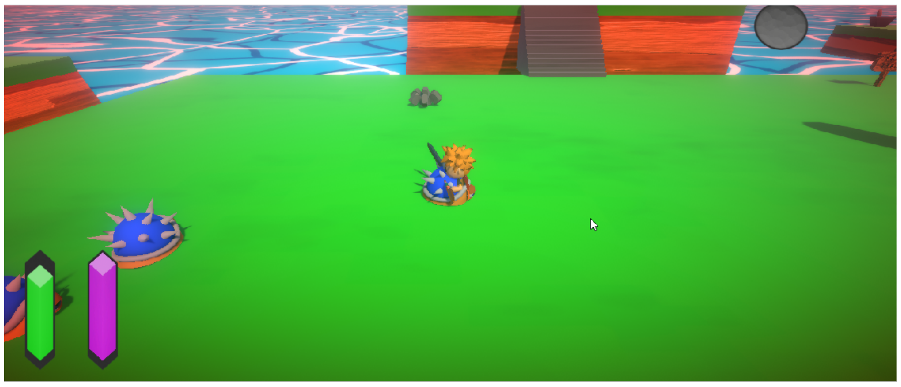

(b) UI when receiving a hit

Figure 5.10. UI when the player receive damage

In addition, when the player is attacked, they will lose one unit of health. However, if they collide with a collectible item, they will regain health.

Switching between weapons

To keep the player informed about the currently equipped weapon at all times, an icon of the weapon will appear in the top right corner. When the player has both weapons, they can press the E key to swap between them.

CHAPTER 6

Conclusions and future Work

Contents

6.1 Acquired Results

As mentioned in section 5.3, the project started with the objective of creating an action game with a cartoon art style, incorporating group-based AI patterns. However, due to complications in terms of experience and time, the decision was made to simplify the AI and focus on the game mechanics and artistic aspects.

The result is a small game that can be easily expanded upon, featuring intuitive mechanics and average-level art. The project has also provided an opportunity to utilise and enhance the skills learned during the degree program, while gaining experience in working on a personal project, product design, task organisation, and handling different disciplines within game development. Specifically, significant learning took place in the areas of C# programming, shader programming, and AI.

A playable demo, and the source code of the project can be found at https://github.com/al397425/TFG_Ausias_Garcia_Torres_The_Last_Hope[16]

6.2 Conclusions

As a conclusion, the developed game successfully meets both the functional and non-functional requirements. The different assets have been created appropriately, and the project has been thoroughly documented, demonstrating effective utilisation of agile methodology.
Personally, there are plans to continue adding content to the game.

6.3 Future Work

As the game has a reduced duration, it would be beneficial to include a greater variety of interactable objects and gadgets for the player to utilise (such as bombs, enemy stunning devices, enemy attractors, or repellers) to expand and design levels around them.

Additionally, the implemented puzzles should be made more challenging, as they currently exist in their most basic and functional form.
It is also recommended to develop an AI for enemies with more complex attack patterns and design battles that leverage the player's new gadgets to defeat them.

Narrative elements should also be added to the beginning and end of the game through image animations or cinematics, as outlined in the GDD but due to time constraints, it was not possible to include them.
In the beginning, explaining the state of the world or the character's context, and at the end of the game, waking up at home and explaining that the adventures he had experienced were just a dream.

Of course, these future additions would be developed by the student.

BIBLIOGRAPHY

[1] *The Legend of Zelda Link's Awakening*
https://www.nintendo.es/Juegos/Juegos-de-Nintendo-Switch/The-Legend-of-Zelda-Link-s-Awakening-1514327.html
Accessed: 2023-03-03.

[2] *TUNIC*
https://store.steampowered.com/app/553420/TUNIC/
Accessed: 2023-03-03.

[3] *Deaths Door*
https://store.steampowered.com/app/894020/Deaths_Door/
Accessed: 2023-03-03.

[4] Unity3D 2021.3.10f1
https://unity.com/es/releases/editor/archive
Accessed: 2023-03-03.

[5] Visual Studio Code
https://visualstudio.microsoft.com/es/downloads/
Accessed: 2023-03-03.

[6] Blender 3.2
https://www.blender.org/download/
Accessed: 2023-03-03.

[7] Substance Designer
https://www.adobe.com/es/products/substance3d-designer.html
Accessed: 2023-03-03.

[8] Github Desktop
https://desktop.github.com
Accessed: 2023-03-03.

[9] Trello
https://trello.com
Accessed: 2023-03-03.

[10] Open Game Art
https://opengameart.org
Accessed: 2023-03-05

[11] Dungeon Mason Unity Asset Store Page
https://assetstore.unity.com/publishers/23554
Accessed: 2023-03-05

[12] Zelda Shader Graph Tutorial and Voronoid Textures
https://www.youtube.com/watch?v=NHy3rSKtRmc&ab_channel=DanielIlett
Accessed: 2023-05-05.

[13] SM Channel Particle System
https://www.youtube.com/watch?v=fNGEevaGHMI&ab_channel=SMChannel
Accessed: 2023-04-03.

[14] Reynolds, C. W. (1999), March). Steering behaviours for autonomous characters. In Game developers conference (Vol. 1999, pp. 763-782).

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.636&rep=rep1&type=pdf.
Accessed: 2023-03-03.

[15] Boids by Craig Reynolds
http://www.red3d.com/cwr/boids/
Accessed: 2023-03-03.

[16] Github Repository
https://github.com/al397425/TFG_Ausias_Garcia_Torres_The_Last_Hope
Accessed: 2023-03-03.