Development of a role-playing video game inspired in Maya Civilization

Alexandre Riera Martínez

Final Degree Work

Bachelor's Degree in
Video Game Design and Development

Universitat Jaume I

May 28, 2021

Supervised by: Miguel Chover Selles

To my parents, for being my main support during the development of this project.

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Miguel Chover Selles, for the advice on how to direct this project and the recommendations to follow.
Thanks to Aida for giving me the inspiration to create this story.
To Irene for motivating and supporting me when I needed it most.
To Guille for all the afternoons and nights in Discord doing this project and all the feedback.
I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this document.

Abstract

This is the memory of the final degree project in Video Game Design and Development. In this document there will be presented all aspects related to the concept, design and development of the video game K'an, a role-playing video game set in Maya civilization that tells the dystopian story of Princess K'an and her desire to unify the city-states of the Yucatan Peninsula to face the arrival of the Spanish invaders.
Special care has been taken to create an environment with tribal elements, a narrative and a level design in which the player has to visit all the relevant locations and the implementing of classic mechanics of Role-Playing Game (RPG), platforms and shooter video games.

Contents

CHAPTER 1

Introduction

Contents

This chapter explains the main objectives that have been reached through the development of this project, the knowledge of which degree subjects has been applied and what tools have been used.

1.1 Work Motivation

One of the motivations for carrying out this project was learning to develop typical Role-Playing Game (RPG) mechanics because of it's one of the most widespread genres and on which more different mechanics can be developed, being able to create open scenarios with elements of nature too.
It was also being able to create a game in Valencian since it is one of the official languages of the Valencian Country and there aren't many video games of this genre in this language either.

1.2 Objectives

The main objectives that have been achieved by carrying out this work are:

- The integration of the character models together with their main animations in a video game engine.
- The programming of basic elements of a role-playing video game: minimap, inventory, quest system, enemies, attacks, unlocking of new zones, unlocking of new habilities, collecting and using items.
- The elaboration of a narrative in an open world.
- The creation of a level design that allows maintaining a balance between the mechanics developed and the objectives.

1.3 Related Subjects

This section shows each of the degree subjects related to the development of the project and how their knowledge appears represented in the video game.

- **VJ1208 – Programming II**, for the class-oriented structure of all the scripts in the game.
- **VJ1216 – 3D Design**, for modeling and applying textures of the main character, the enemies, the non-player characters and the buildings.
- **VJ1224 – Software Engineering**, for the development of diagrams to organize the project such as Gantt charts, flow charts, beat charts, etc.
- **VJ1227 – Game Engines**, for using optimization techniques like baked lights, occlusion culling [1] and Fast approximate anti-aliasing (FXAA) [2].
- **VJ1231 – Artificial Intelligence**, for the implementation of various Artificial Intelligence (AI) in the game and states machines [3].

1.4 Resources

The following tools have been used to elaborate the project:

- **Unity**, as the main game motor where all the game objects, scripts, assets and logic of the game reside.
- **Blender**, for 3D modeling of the main characters and enemies.
- **Visual Studio Code**, for programming all scripts.
- **Mixamo**, for the animations of all the characters in the game.
- **Illustrator**, for designing the vector textures.
- **Procreate**, for reprinting the textures of the Non-Player Characters (NPC).

- **Github**, for managing the version control.
- **Visio**, for making the organizational charts.
- **LaTeX**, to layout this document.

CHAPTER 2

Game Design Document

Contents

This chapter presents the Game Design Document [4], the part of the development of a videogame that offers a guiding vision throughout the game creative process. It contains all the information about the story, the gameplay, the level design, the concept art, the user interface (UI) and the game controls.

2.1 Introduction

The title of the video game is K'an. It's a role-playing video game in Valencian with a 3rd person view for Personal Computer (PC). The player takes on the role of Princess K'an who must carry out quests in the city-state of Chichén Itzá.

The player will be able to walk, run, jump, attack using his powers, orient itself with a minimap, talk to NPCs, save items in his inventory and use them, unlock new abilities, etc. Two types of enemies will appear on the map (a bomb thrower that will

move towards the player and another with a bayonet that will consecutively hide in the ground and attack when it comes out) that will make completing these quests more difficult.

2.2 Story

The story is based on the following question: what would have happened if the arrival of the Spanish invaders to the Yucatan Peninsula had been a failure [5]? Thus, in a dystopian Chichén Itzá, Chilam, one of the shamans from the Caracol observatory [6], had a revelation from the gods where they warned that some "evil beings" would assault the entire peninsula. Seeing that without the collaboration of the other cities it would be impossible to face them, Chilam entrusted Princess K'an with the task of unifying the other city-states with which they were faced, and gave her the blessing of Hunab Ku [7] to begin her journey.

First of all, he commissioned K'an to make an offering to the gods before leaving towards Mayapan [8]. She went to the town to collect a shield from the blacksmith Kabáh and returned to the Cenote [9] to throw it and complete the ritual. Later, Chilam entrusted her with the task of collecting some sacks of corn to take to the prince of Mayapan as a gift since both tribes were at odds, but as they were speaking, they realized that some invaders had established their camp in the Kukulkan [10].

K'an went there to confront them and once she finished, she went straight to the town. There she spoke with Nicté, the farmer, who on behalf of the whole town thanked her for the help. Nicté told K'an that if she needed corn she could go to his farm and pick up whatever she needed. So K'an went to the farm, collected all the corn, and defeated one of the enemies that were lurking around. She then returned to the Caracol and Chilam confirmed that K'an was ready to leave.

He gave her the second blessing, this time from the god of fire Kauil [11], allowing the princess to launch fireballs and being able to destroy the obstacles that prevented the passage on the road to Mayapan. In this way, K'an left Chichén Itzá for other distant destinations.

Stories tell that the princess's journey lasted more than 10 years but she finally managed to unify all the tribes and thus avoided the conquest of the Spanish in a final battle in Cozumel [12].

2.3 Main Characters

The main characters that have a relevant role in the story are:

- **K'an**. She is the princess of Chichén Itzá and first in line to the throne. As it is seen in Figure 2.1, she has green hair with braids and a characteristic poncho that acts as a smock. She also wears a tribal mask that protects her face and also makes her go unnoticed in some places. Thanks to the blessings received by the gods, she has divine abilities such as casting spells with her hands. She is a bit

shy and her main hobbies are traveling the vast plains of Chichén Itzá, climbing mountains and playing ball [13].

Figure 2.1: K'An

- **Chilam**. He is the chief shaman of El Caracol. In Figure 2.2 it is shown he is dressed in an orange and blue poncho and mask, and his hair is tied up in a bun. He is a very wise man who is in constant contact with the gods who warn him of upcoming events. During the game he will always be at the door of El Caracol and will be the one who starts most of the quests.

- **Kabáh**. He is one of the inhabitants of the town of Chichen Itza. He has black hair and is styled in three bows. He wears a red poncho and the typical mask that all the villagers wear as it can be seen in Figure 2.3.

- **Nicté**. He's a farmer from Chichén Itzá. He lives in the town, but has a farm to the south of this. In Figure 2.4 it is shown he is dressed with a poncho and a yellow mask typical of farmers and on his head he wears a straw hat.

2.4 Enemies

There are two types of enemies that will appear scattered on the map:

- **Miner**. It is one of the invading Spanish warriors. In Figure 2.5, he is dressed in a morion [14] and suspenders. His main attack is to shoot with a bayonet and move underground where he cannot be hit.

Figure 2.2: Chilam

Figure 2.3: Kabáh

- **Thrower**. Another one of the invading Spanish warriors. In Figure 2.6, he's dressed in a vest and belt with many bombs. It's a heavyweight with a lot of life that will move towards the player to throw bombs that will explode on contact with him or after 3 seconds. While he's throwing bombs at the player, he will keep his distance with the player getting backwards in case he gets close.

Figure 2.4: Nicté

Figure 2.5: Miner

2.5 Level Design

2.5.1 Beat Chart

The Figure 2.7 shows each of the objectives that the player must complete. Each time one of them ends, the next one will be unlocked.

Figure 2.6: Thrower

2.5.2 Maya Ambient Justification

The design of the level in which the entire game takes place is a reduced version of the city-state of Chichén Itzá (See Figure 2.8). Thus, a search was carried out in order to represent some of the most characteristic elements of the time and everything that happened there.

The location where the game starts is El Caracol (See Figure 2.9), an ancient Maya observatory that allowed the Maya to observe changes in the sky.

The next important location is the Sacred Cenote (See Figure 2.10), a cavern with water in which the Maya made sacrifices and offerings to the gods. One of the quests that K'an will have to carry out is to make an offering by throwing a shield inside the Cenote.

Another characteristic location of Chichén Itzá, if not the most famous, is the Kukulkan (See Figure 2.11), a pyramidal temple erected to worship the Maya God Kukulkan. During the development of the game, some Spanish invaders set up their camp in front of it and K'an will have to bring them down.

There are other locations such as a corn farm [15] (See Figure 2.12) because of it was one of the foods that was most cultivated at that place, the town (See Figure 2.13) represented with straw huts, rocks and other characteristic elements where the commoners lived and the sacbés [16] or limestone paths that are they used to draw roads and connect towns with each other.

In conclusion, the important locations of the level are distributed following a logic since the river divides the map into three parts, making the player have to follow the route of the sacbés to advance with the events of the story, apart from the fact that it

also makes the player have to visit each of them.

2.5.3 Items and Enemies Location

Items and enemies are scattered throughout the map. As regards the firsts, both the quest items and the consumables items are in easy-to-see places so that the player can quickly identify them, such as near the sacbés, in the town, in the meadows and in relatively open spaces.
On the other hand, the enemies will always find themselves in the sacbes, blocking the player's path and motivating him to have to kill them in order to advance.

In this way, so that the combats aren't so difficult for the player, consumable items are also placed in the enemy areas so that he can collect and use them while he's in combat to improve their stats such as life, mana, move faster, be invincible, etc.

2.6 Gameplay

The gameplay follows a system with many elements of classic role-playing video games, but there are also shooter and platform parts. The player will be able to walk, run and move in general around the map, performing a series of quests where he must take into account their health and mana status.

He will be able to access an inventory to manage the consumable items he collects and the items from quests. He can also choose from his unlocked abilities to shoot the enemies using mana that will be progressively restored. Enemies will move within a specific area and attack the player as soon as they see him.

2.7 Concept Art

For the art part, it has been decided on a low-poly/medium-poly style with very live colors. Some references that have been taken from other video games are:

- **AER: Memories of Old** [17]. This Forgotten Key video game, released in 2017, has served as one of the main artistic references as can be seen in its art and all the aesthetics regarding forests, deserts and open spaces in general that it transmits in the player a feeling of calm that can also be seen in K'an.
- **The Legend of Zelda** [18]. Many references have also been taken from this saga for the magic part, the choice of different ranged attacks, the inventory, the enemies, the jump and movement mechanics of the player, the exploration of the open world and even dialogues with NPCs.

2.8 User Interface

In the Figure 2.14 it is shown all the elements of the HUD. While the player is in the middle of a game, the HUD will display a life bar (Elem. 1), a mana bar (Elem. 2), and

an icon [19] with the selected hability (Elem. 3) up to the left. Also, at the top right the current objective (Elem. 4) of the player will be written. In the inventory (Elem. 5) the player will be able to see the quest items (Elem. 6), the consumable items (Elem. 7) and the minimap (Elem. 8). The moment the player enters an important area, a sign (Elem. 9) will appear above saying which one it is. When the player approaches an NPC and talk to him, a panel [20] (Elem. 10) will appear showing the dialogue and the name with which he is speaking.
On the stage the player can see other elements of the In-Game UI such as the life bars of the enemies above their heads or glowing particles that reveal the location of important items.

In the main menu (See Figure 2.15), three buttons will appear to see start a game, see the about menu where the story and game controls appear, and the button to exit the game.

2.8.1 Graphic User Interface Objects

For a greater compression of the UI by the player, issues such as the colors of the potions were taken into account so that everything related to health would be green, those related to mana would be blue and colors like red, grey or just transparent for other effects like giving more velocity, being invencible or more damage to enemies (both In-Game and the HUD the player can see these colors easily). The quest items were also identifiable due to their characteristic flash effect and spin on themselves. In the inventory, a sprite, name and description was associated with the 3d object that the player had just been collected.

The colors of the enemies were also important so that it was possible to quickly differentiate what type it was (blue for the thrower and yellow for the miner).

2.9 Game Controls

In Figure 2.16 the controls to move around the game are displayed. As it is a video game for PC, both mouse and keyboard must be used.

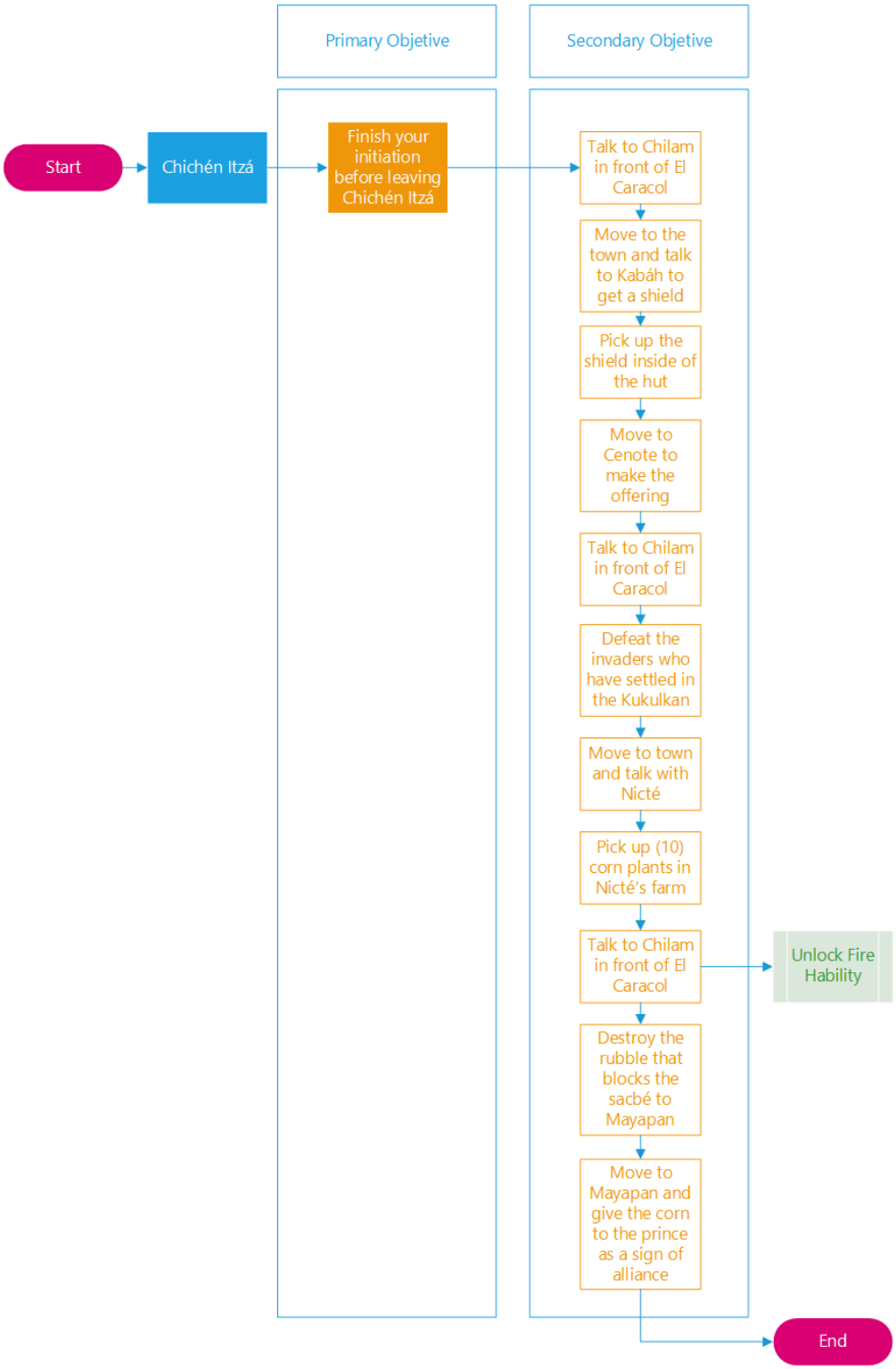


Figure 2.7: Beat Chart

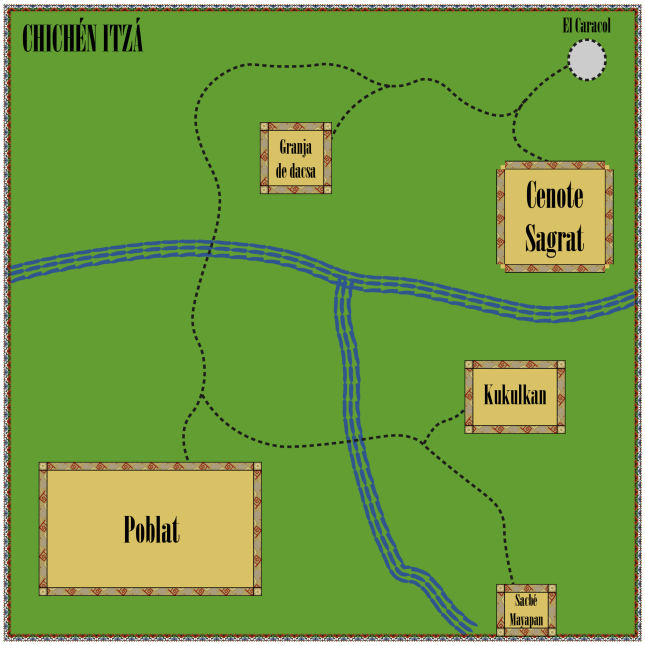


Figure 2.8: Minimap

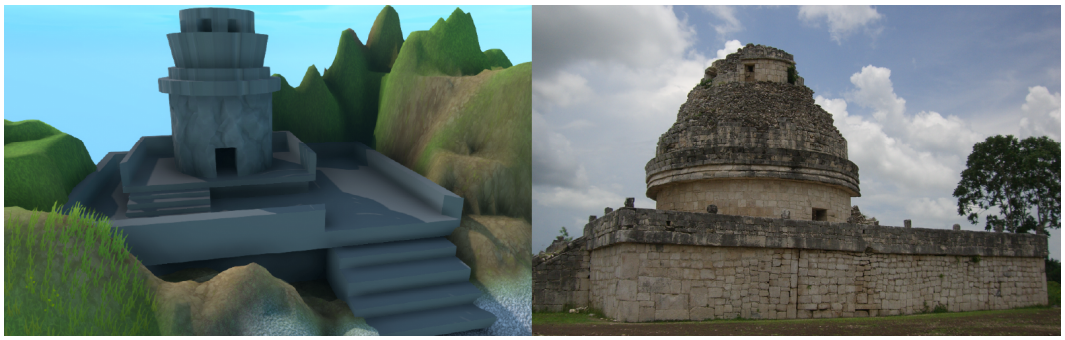


Figure 2.9: El Caracol In-Game and in real life

Figure 2.10: Cenote In-Game and in real life

Figure 2.11: Kukulkan In-Game and in real life

Figure 2.12: Nicté's Corn Farm

Figure 2.13: Maya Town In-Game and in real life

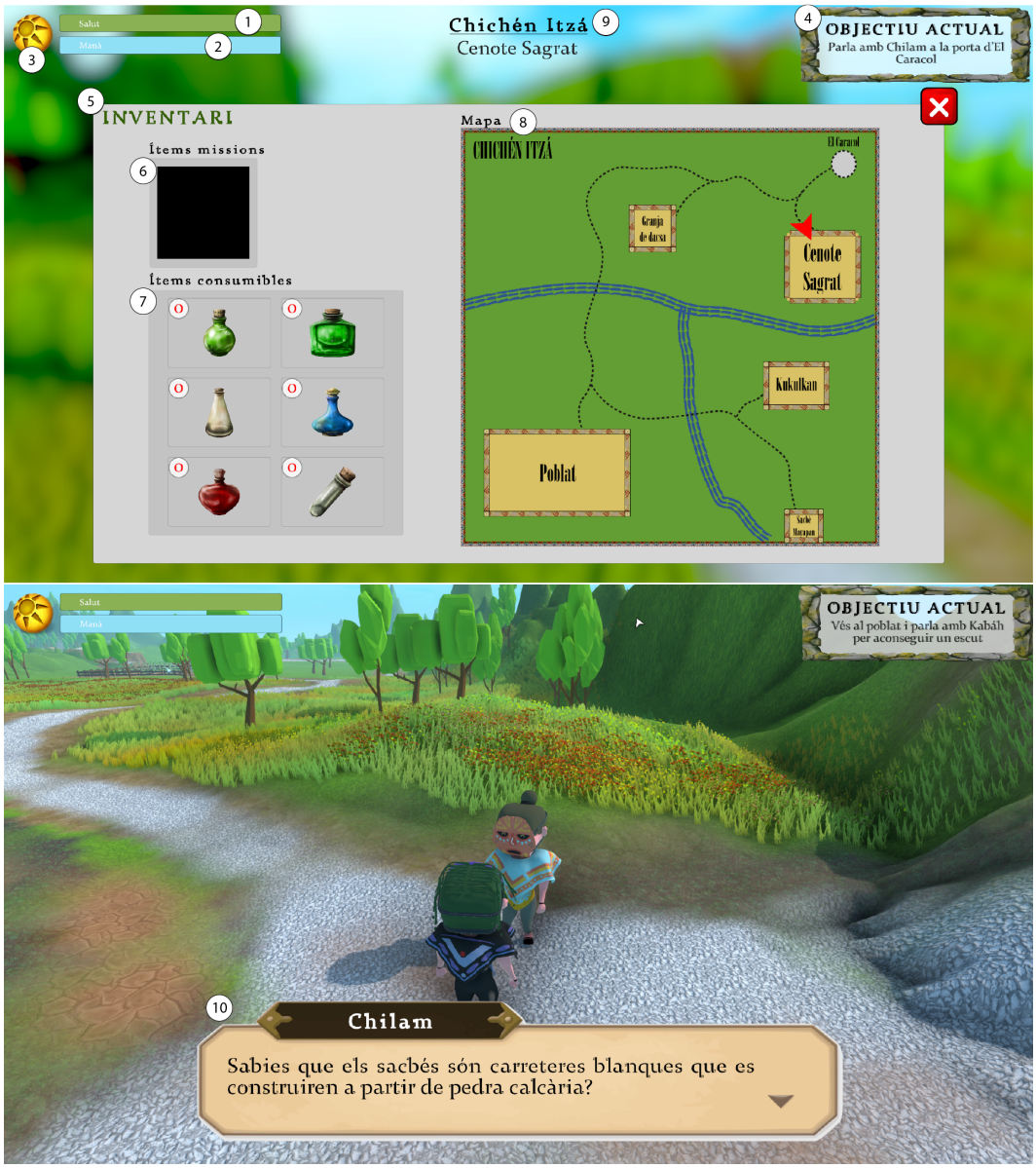


Figure 2.14: HUD

Figure 2.15: Main Menu

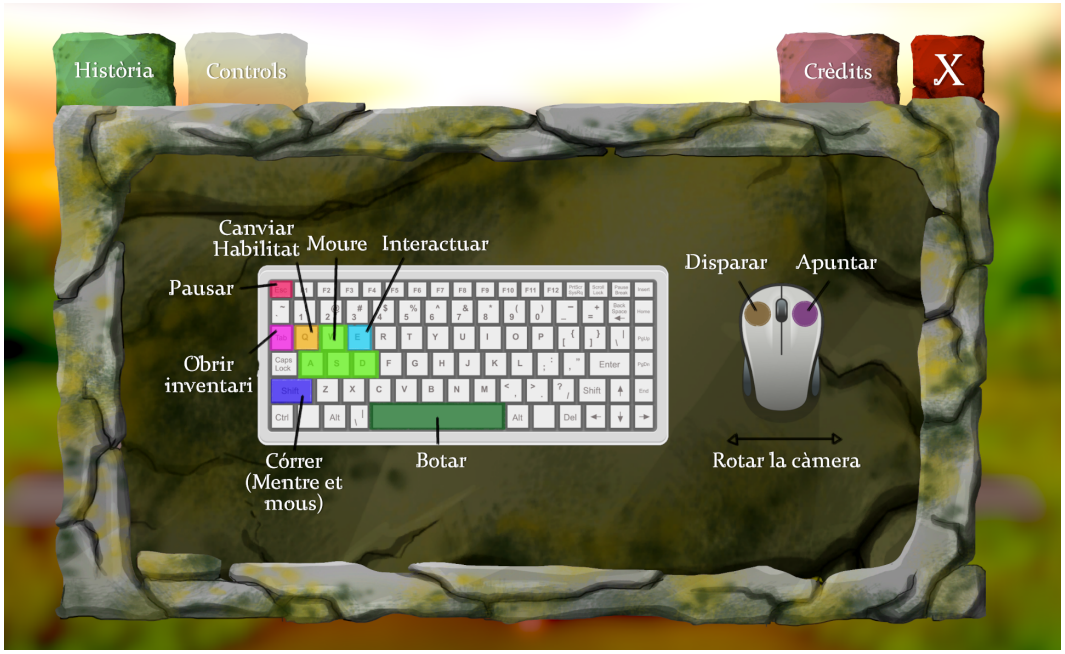


Figure 2.16: In-Game Capture of the Game Controls

CHAPTER 3

Functional and Technical Specification

Contents

This chapter presents everything related to the architecture that has been developed for the execution of the game mechanics, the objectives to be achieved, the art, the music, the level design and the functional requirements.

3.1 System Design

3.1.1 Flow Chart

The Figure 3.1 describes the actions that, depending on the situation that arises, the player may or may not perform during the execution of the level. In addition, it is included the way in which the system will process this data.

3.1.2 Functional Requirements

This section will deal with the functional requirements [21] of the video game, that is, how the game logic transforms certain actions or inputs into results or outputs and a description of what happens. So, the functional requirements are essentially specific functionality that defines what a system is supposed to achieve.

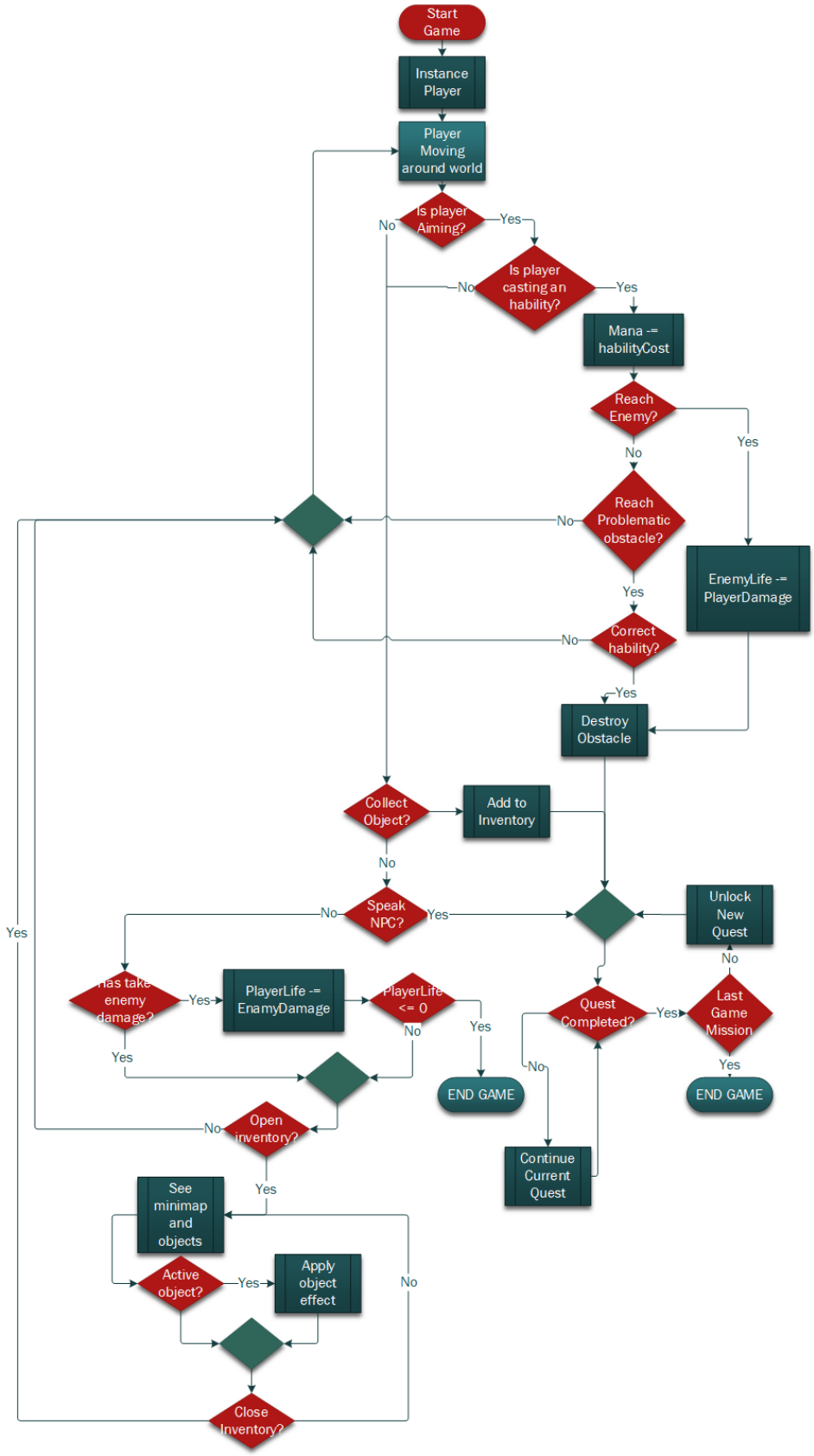


Figure 3.1: Flow Chart

Input:	AWSD input
Output:	Character moves
Each time the player hit A, W, S or D keys the character will move on X or Z Axis	

Table 3.1: Functional requirement «PLAY1. Movement»

Input:	Spacebar input
Output:	Character jumps
While the player is on the ground, the character will apply a positive force in the Y Axis when the player hit the spacebar	

Table 3.2: Functional requirement «PLAY2. Jump»

Input:	Right click input
Output:	Character aims
When the player hits the right click a zoom will be permormed and a reticle will appear, allowing him to cast an hability	

Table 3.3: Functional requirement «PLAY3. Aim»

Input:	Left click input
Output:	Character attacks
While the player is aiming and has enough mana, the character will cast an hability towards what is pointing at	

Table 3.4: Functional requirement «PLAY4. Attack»

Input:	Tab input
Output:	Open/Close inventory
Each time the player hits Tab key the game will be paused or resumed and the inventory will be shown or hidden	

Table 3.5: Functional requirement «PLAY5. Inventory»

Input:	Esc input
Output:	Pause/Resume Game Time
When the player hits Esc key the game will be paused or resumed	

Table 3.6: Functional requirement «PLAY6. Pause»

Input:	Life == 0
Output:	Game Over
When player's life is equal to 0 the game will be over and he'll go back to the start	

Table 3.7: Functional requirement «PLAY7. Death»

Input:	Character triggers an item
Output:	Add to inventory
If the character walks over an item it will be added to the inventory and will be able to see it when the inventory is opened	

Table 3.8: Functional requirement «INV1. Collect item»

Input:	Player clicks over item
Output:	Apply effect
If the player have 1 or more quantity of an item and clicks on it, the player showed in the item description will be applied to the character	

Table 3.9: Functional requirement «INV2. Use item»

Input:	Patrol time > 0
Output:	Move on stage
While the enemy didn't reach the patrol spot he's supposed to go he will move until he'll reach it. Then he'll find another one	

Table 3.10: Functional requirement «ENE1. Patrol»

Input:	Character is near
Output:	Attack
In case of character is close to the enemy he will perform his main attack	

Table 3.11: Functional requirement «ENE2. Attack»

Input:	Life == 0
Output:	Die
When the enemy life is equal to 0 because of the character attacks he will die	

Table 3.12: Functional requirement «ENE3. Death»

Input:	Character near
Output:	Display dialogue
If the character is close to an npc and hits the e key the dialogue of the npc will be shown	

Table 3.13: Functional requirement «NPC. Talk»

Input:	Player finish quest
Output:	Unlock new quest
When player talks to a quest npc, defeats enemies or makes any action in order to overcome the current quest a new one will be unlocked and showed in the HUD	

Table 3.14: Functional requirement «QUEST. Unlock»

3.2 Game Mechanics and Scripting

All the scripting part has been broken down in the project into different folders to speed up programming and make some classes independent from each other, with the exception of the Game Manager, which can be accessed by any class. First, it will present about how the movement and actions of the character was developed.

3.2.1 Player

The main character is made up of 4 scripts that control most of the actions he can perform:

PlayerController

It's a general class where all movement, jumping, aiming and throwing habilities are managed. All animations are also managed. This class depends on all the other classes of the character as it uses data such as life from the Player Stats, if the player is talking from the Player Dialogue Controller, if the player is using the inventory from the Player Inventory Controller, if the player is changing cameras when aiming from the Aim Con-

troller, etc.
Within the script the following functions are executed:

- **Move()** manages the AWSD inputs so that the character moves in the X-Axis and Z-Axis and also rotates the character where the camera is pointing when moving, and that the character looks where the camera is facing when he's aiming. It also manages the shift input to increase speed when held down and change the animation from walking to running.

```
void Move() {
  Vector3 velocity = rb.velocity;
  velocity.x *= drag;
  velocity.z *= drag;
  rb.velocity = velocity;

  float h = Input.GetAxis("Horizontal");
  float v = Input.GetAxis("Vertical");
  shiftPressed = isAiming || (h == 0 && v == 0) ? false : Input.GetKey(KeyCode.LeftShift);
  anim.SetBool("Running", shiftPressed);
  activeFeetParticles();
  if (!GetComponent < PlayerStats > ().canMoveFaster) speed = shiftPressed ? 25 : 10;
  else speed = shiftPressed ? 30 : 15;

  anim.SetFloat("Vertical", v);
  anim.SetFloat("Horizontal", h);

  transform.rotation *= Quaternion.AngleAxis(Input.GetAxis("Mouse_X") * 2, Vector3.up);
  followTransform.transform.rotation *= Quaternion.AngleAxis(Input.GetAxis("Mouse_X") * 2,
  Vector3.up);
  followTransform.transform.rotation *= Quaternion.AngleAxis(Input.GetAxis("Mouse_Y") * 2,
  Vector3.left);

  Vector3 angles = followTransform.transform.localEulerAngles;
  angles.z = 0;

  float angle = followTransform.transform.localEulerAngles.x;

  if (angle > 180 && angle < 350) {
    angles.x = 350;
  } else if (angle < 180 && angle > 60) {
    angles.x = 60;
  }

  followTransform.transform.localEulerAngles = angles;
  nextRotation = Quaternion.Lerp(followTransform.transform.rotation, nextRotation,
  Time.deltaTime * rotationLerp);

  if (Input.GetAxis("Horizontal") == 0 && Input.GetAxis("Vertical") == 0) {
    nextPosition = transform.position;
    if (Input.GetMouseButton(1)) {
      followTransform.transform.localEulerAngles = new Vector3(angles.x, 0, 0);
    }
    return;
```

```
45    }
46
47    Vector3 movement = new Vector3(h, 0, v).normalized;
48    rb.AddRelativeForce(speed * movement, ForceMode.VelocityChange);
49
50    transform.rotation = Quaternion.Euler(0,
51      followTransform.transform.rotation.eulerAngles.y, 0);
52    followTransform.transform.localEulerAngles = new Vector3(angles.x, 0, 0);
53  }
```

- **Jump()** handle the spacebar input so that the character applies a force vertically when it is touching the ground. It also has a small delay to synchronize the jump with the animation so that when the character pushes her legs, the force is applied.

```
1   void Jump() {
2     grounded = Physics.CheckSphere(feet.position, 0.5 f, groundLayer);
3     anim.SetBool("Grounded", grounded);
4
5     if (!isAiming && Input.GetButton("Jump") && grounded && !isJumpingAnimationActive) {
6       anim.SetTrigger("Jump");
7       Invoke("applieJumpForce", jumpTime);
8       isJumpingAnimationActive = true;
9     }
10  }
```

- **Aim()** manages the input of the right click causing one prefab or another to be instantiated depending on the hability selected. If the hability is changed, a new prefab would be instantiated.

```
1   void Aim() {
2     if (Input.GetMouseButtonDown(1)) isAttackinAnimationActive = false;
3     if (Input.GetMouseButton(1)) {
4       isAiming = true;
5       if (habilityInHand == null && !checkIfItsAttackingAnimation()) {
6         habilityInHand = GameObject.Instantiate(habilitiesPrefabs[currentHability],
7           hand.position, hand.rotation);
8         habilityInHand.transform.parent = hand.transform;
9       }
10    } else {
11      isAiming = false;
12      if (habilityInHand != null) {
13        Destroy(habilityInHand);
14      }
15    }
16    anim.SetBool("Aiming", isAiming);
17  }
```

- **ChangeHability()** manages the Q key input to switch habilities between those unlocked. The information of which habilities are unlocked is provided to us by the Game Manager.

```
void ChangeHability() {
  if (Input.GetKeyDown(KeyCode.Q)) {
    int currentHabilityBeforeChange = currentHability;
    if (currentHability == GameManager.instance.getNumHabilitiesUnlocked())
    currentHability = 0;
    else currentHability++;

    if (habilityInHand != null && currentHabilityBeforeChange != currentHability) {
      Destroy(habilityInHand);
    }
  }
}
```

- **Attack()** is the function that manages the input of the left click, adding a RigidBody to the prefab created previously and throwing it with a force towards where the camera is pointing in the case of having enough mana.

```
void Attack() {
  if (Input.GetMouseButtonDown(0) && !isAttackinAnimationActive
  && GetComponent < PlayerStats > ().EnoughManaToAttack()) {
    anim.SetTrigger("Attack");
    GetComponent < PlayerStats > ().UseMana();
    isAttackinAnimationActive = true;
  }

  if (isAttackinAnimationActive) {
    if (habilityInHand != null && anim.GetCurrentAnimatorStateInfo(0).normalizedTime > 0.1
    && anim.GetCurrentAnimatorStateInfo(0).normalizedTime > 0.11
    && checkIfItsAttackingAnimation()) {
      habilityInHand.transform.parent = null;
      habilityInHand.AddComponent < Rigidbody > ();
      habilityInHand.GetComponent < Rigidbody > ().AddForce(cam.transform.forward * 120 +
      transform.right * 2.2f * 2.5f + cam.transform.up * 10, ForceMode.Impulse);
      habilityInHand = null;
      isAttackinAnimationActive = false;
    }
  }
}
```

PlayerStats

It's the script that manages the life and mana of the character and the application of potions.

- In the **Update()** it's constantly updating so that the value of mana and health appears correctly in the sliders. Each time mana is used it will be restored slowly. There's also a delay of 1 second so that the character in case he's injured is not constantly receiving damage. The function ends with a call to the **usePotions()**

function where the usage times of each potion are managed in case it was being used and the sliders that appear in the HUD.

```
private void Update() {
  healthBar.value = health;
  manaBar.value = mana;

  if (!canGetHurt) {
    damageDelay -= Time.deltaTime;
    if (damageDelay < 0) {
      canGetHurt = true;
      damageDelay = 1;;
    }
  }

  if (mana < 100) {
    mana += Time.deltaTime * 2;
  }

  if (health <= 0) {
    isAlive = false;
  }

  usePotions();
}
```

- The **Damage()** function is the function that when the delay has not been applied or if, on the contrary, an invincibility potion has not been consumed, it reduces the player's health and activates a small shake in the camera.

```
public void Damage(int damage) {
  if (canGetHurt && !isInvencible) {
    GetComponentInChildren < AimController > ().Shake(5 f, 0.1 f);
    health -= damage;
    canGetHurt = false;
  }
}
```

- The **whichPotionIsUsing()** function is called by the Game Manager when using a potion where the id of the one just used is passed to it. In **PlayerStats** a series of functions are called in which a corresponding Boolean variable is simply activated to say that one potion or another is activated. Particles are also activated in the character to show that there is an effect being applied.

PlayerDialogueController

It's the script that manages the actions of the player while he's talking with an NPC.

- The **StartDialogue()** function is executed as soon as a dialog is started to deactivate the **PlayerController**, preventing the player from moving and making also the player and the NPC look at each other.

```
public void StartDialogue() {
  GetComponent < PlayerController > ().enabled = false;
  GetComponent < Rigidbody > ().velocity = Vector3.zero;
  GetComponent < Animator > ().SetFloat("Horizontal", 0);
  GetComponent < Animator > ().SetFloat("Vertical", 0);
  GetComponent < Animator > ().Play("Walk", 0);
  transform.LookAt(new Vector3(NPC.transform.position.x, transform.position.y,
  NPC.transform.position.z));
  NPC.transform.LookAt(new Vector3(transform.position.x, NPC.transform.position.y,
  transform.position.z));
  NPC.GetComponentInChildren < QuestDialogueManager > ().
  dialogueTriggerCurrentQuest().cam.SetActive(true);
  NPC.GetComponentInChildren < QuestDialogueManager > ().
  dialogueTriggerCurrentQuest().buttonOverHead.SetActive(false);
  NPC.GetComponentInChildren < QuestDialogueManager > ().
  dialogueTriggerCurrentQuest().TriggerDialogue();
}
```

- On the contrary part, it is the **EndDialogue()** function that reactivates the player controller and deactivates the dialog camera that focuses on the NPC.

```
public void EndDialogue() {
  GetComponent < PlayerController > ().enabled = true;
  NPC.GetComponent < DialogueTrigger > ().cam.SetActive(false);
  dialogueIsStarted = false;
}
```

PlayerInventoryController

It's the script in charge of blocking or allowing the movement of the character while he's using the inventory.

- In the **Update()** the input of the tab is managed where while the player is alive and not in a dialogue the inventory will be opened or closed.

```
void Update() {
  if (Input.GetKeyDown(KeyCode.Tab) && GetComponent < PlayerStats > ().isAlive
  && !PlayerDialogueController.dialogueIsStarted) {
    if (!isOpened) {
      openInventory();
    } else {
      closeInventory();
    }
  }
}
```

- The **openInventory()** and **closeInventory()** functions are used to resume or pause the time each time inventory is used, make the cursor appear or disappear, and activate a Gaussian blur [22] on the HUD.

```
public void openInventory() {
  isOpened = true;
  cam.GetComponent < SuperBlurBase > ().enabled = true;
  inventory.SetActive(true);
  foreach(ShowItemInfo sii in inventory.GetComponentsInChildren < ShowItemInfo > ())
  sii.closeInfo();
  Time.timeScale = 0 f;
  Cursor.visible = true;
}

public void closeInventory() {
  Time.timeScale = 1 f;
  isOpened = false;
  cam.GetComponent < SuperBlurBase > ().enabled = false;
  Cursor.visible = false;
  inventory.SetActive(false);
}
```

3.2.2 Enemies

Enemies will attack the player whenever they see him and when they are defeated they will spawn again in a short period of time. There are several scripts common to both while each has a specific one that determines all its logic.

The scripts that all enemies have in common are:

CalculatePatrolPoint

As using a NavMesh [23] for their movement (See Figure 3.2), the **CalculatePatrol-Point** script is assigned to an empty object (parent of all enemies) that defines the maximum radius or space through which the enemies can move. It is composed of two functions:

- The first function is **RandomPoint()** in which a random point is extracted and returns true in case it is within the radius of space in which they can be moved.
- The second function is **GetRandomPoint()** in which **RandomPoint()** is called until a correct point is extracted and returns the position if the new random point extracted. This function is called in the MoveThrower and MoveMiner scripts to carry out the patrols.

```
bool RandomPoint(Vector3 center, float range, out Vector3 result) {
    Vector3 randomPoint = center + Random.insideUnitSphere * range;
    NavMeshHit hit;
    if (NavMesh.SamplePosition(randomPoint, out hit, 1.0 f, NavMesh.AllAreas)) {
      result = hit.position;
      return true;
    }
  result = Vector3.zero;
  return false;
```

```
}

public Vector3 GetRandomPoint(Transform point = null, float radius = 0) {
  Vector3 _point;
  bool pointFound = false;
  do {
    pointFound = RandomPoint(point == null ? transform.position : point.position,
    radius == 0 ? Range : radius, out _point);
    if (pointFound) {
      Debug.DrawRay(_point, Vector3.up, Color.black, 1);
      return _point;
    }
  }
  while (!pointFound);
  return point.transform.position;
}
```

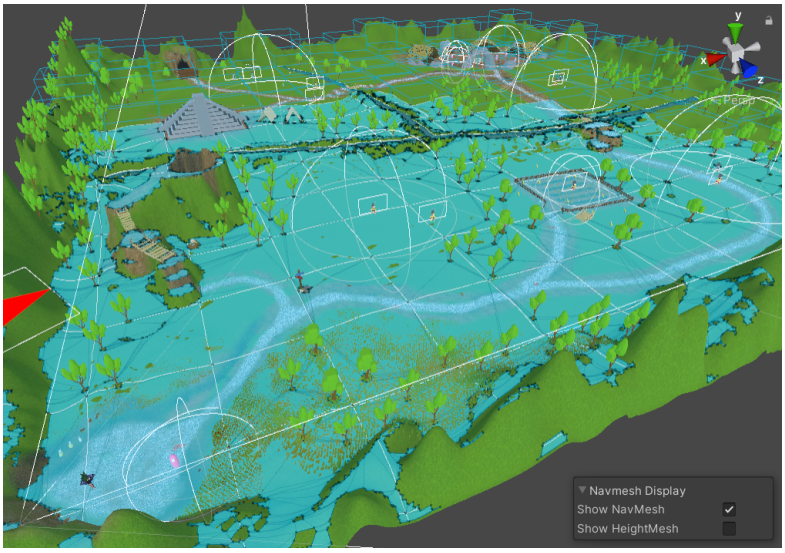


Figure 3.2: NavMesh

EnemyStats

In **EnemyStats**, the enemy's life is managed by updating in **Update()** his life variable with the one that appears in the slider above his head, and it also has an **EnemyGetsDamaged()** function that is called every time a hability thrown by the player touches an enemy.

```
private void Update() {
  chb._healthValue = health;
  if (health <= 0) {
    isAlive = false;
```

```
  }
}

public void EnemyGetsDamaged(int damage) {
  if (enemyCanGetDamaged) health -= damage;
}
```

Now, for each enemy exists the following specific scripts:

MoveMiner

In the **MoveMiner** all the logic of the miner is managed. Most of it is executed in the **FixedUpdate()** where it make calls to various functions:

```
private void FixedUpdate() {
  if (GetComponent < EnemyStats > ().isAlive) {
    if (!agent.hasPath) {
      minerIsStopped();
    } else {
      minerIsMoving();
    }
    manageAnimations();
  } else {
    minerDeath();
  }
}
```

- The **minerIsStopped()** function is executed when the miner has reached one of the calculated patrol points. Here, the canvas with the life bar would be shown and the **enemyCanGetDamaged** variable of the **EnemyStats** would be activated so the player could already harm him. During the execution of the function, the waiting time would be subtracted and, in the meantime, if the player was nearby, the miner would begin to attack.

  Once the wait time reaches zero, the "jump into floor" animation is activated, the wait time is reset for the next time the miner reach the patrol point, and therefore the new patrol point is recalculated.

```
private void minerIsStopped() {
  enemyLifeBarCanvas.enabled = true;
  GetComponent < EnemyStats > ().enemyCanGetDamaged = true;
  timeToWait -= Time.deltaTime;
  if (timeToWait < 0) {
    jumpInAnimationActive = true;
    anim.SetTrigger("Jump_in");
    foreach(ParticleSystem ps in allPs) ps.Play();
    resetTimeToWait();
    resetTimeToRecalculateRoute();
    agent.SetDestination(patrolArea.GetRandomPoint());
  } else {
    if (!executeOnce) jumpOutAnimationActive = true;
```

```
    foreach(ParticleSystem ps in allPs) ps.Stop(true, ParticleSystemStopBehavior.StopEmitting);
    canBeHurt = true;
    anim.SetBool("Shoot", playerIsNear);
    if (playerIsNear) {
      Attack();
    }
  }
}
```

- On the other hand, the **minerIsMoving()** function is executed while the miner is traversing the map until it reaches the assigned patrol point. While it is going, the life bar doesn't appear and it cannot be hurt either. In the case that by any chance he gets stuck somewhere or the place he has to go to is inaccessible, a new route will be recalculated after 10 seconds.

```
private void minerIsMoving() {
  enemyLifeBarCanvas.enabled = false;
  GetComponent < EnemyStats > ().enemyCanGetDamaged = false;
  timerToRecalculateNewPath -= Time.deltaTime;
  if (timerToRecalculateNewPath < 0) {
    resetTimeToRecalculateRoute();
    agent.SetDestination(patrolArea.GetRandomPoint());
  }
}
```

- In **Attack()** the miner constantly looks at the player and each time the firing animation is executed in 50% of the animation (that is, when it is seen that it is going to fire) an explosion and a bullet with a force are instantiated.

```
private void Attack() {
  transform.LookAt(new Vector3(player.transform.position.x, transform.position.y,
    player.transform.position.z));
  float nT = anim.GetCurrentAnimatorStateInfo(0).normalizedTime -
  (int) anim.GetCurrentAnimatorStateInfo(0).normalizedTime;
  if (nT > 0.53 f && nT < 0.56 f &&
    anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_ATTACK)) {
    explosionWhenPullTheTrigger();
    instantiateBullet();
  } else {
    bullet = null;
  }
}
```

- In **manageAnimations()** it is verified that the animations of entering and leaving the ground are running and that objects such as the SkinnedMeshRenderer, the helmet and the bayonet appear or not.

```
private void manageAnimations() {
  if (jumpInAnimationActive) {
    if (anim.GetCurrentAnimatorStateInfo(0).normalizedTime > 0.8 f &&
```

```
4       anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_JUMP_IN)) {
5       mr.enabled = false;
6       shotgun.SetActive(false);
7       helmet.SetActive(false);
8       jumpInAnimationActive = false;
9       executeOnce = false;
10    }
11  }
12
13  if (jumpOutAnimationActive) {
14    executeOnce = true;
15    mr.enabled = true;
16    shotgun.SetActive(true);
17    helmet.SetActive(true);
18    anim.SetTrigger("Jump_out");
19    jumpOutAnimationActive = false;
20  }
21 }
```

- In **minerDeath()** the death animation is simply executed and once it is finished the enemy is destroyed, existing a small chance of instantiating a potion on the ground.

```
1 private void minerDeath() {
2   anim.SetTrigger("Death");
3   if (anim.GetCurrentAnimatorStateInfo(0).normalizedTime > 0.8 f &&
4     anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_DEATH)) {
5     Destroy(gameObject);
6   }
7 }
```

MoveThrower

The **MoveThrower** script contains all the logic of the Thrower enemy. In this script a state machine has been implemented in Update() where the thrower can be in the PATROL, FOLLOW or ATTACK state.

While in PATROL, it will patrol at random points within the allowed zone and pause for a few seconds before calculating a new point. When the player enters the thrower's trigger, it will change its state to FOLLOW and it will move towards the player and, if it's at a certain distance, it will change its state to ATTACK. In this state, in case the player gets closer to the thrower, it would execute a call to the RunFrom() function and move backwards while maintaining the distance.

Otherwise, the thrower would continue to spawn bombs towards the player. Within the same Update() the operation to throw the bombs is executed in case the transition between the animations fails. Also within the Update() it's executed that in the event that the enemy died, the death animation would be carried out and later it would be destroyed, being able to instantiate a potion.

```
void Update() {
  if (GetComponent < EnemyStats > ().isAlive) {
    float nT = anim.GetCurrentAnimatorStateInfo(0).normalizedTime
    - (int) anim.GetCurrentAnimatorStateInfo(0).normalizedTime;
    if (currentState == State.PATROL) {
      agent.speed = 10;
      anim.SetBool("Attack", false);
      anim.SetBool("isWalking", true);
      agent.isStopped = false;
      if (!agent.hasPath) {
        anim.SetBool("Attack", false);
        anim.SetBool("isWalking", false);
        timeToWait -= Time.deltaTime;
        if (timeToWait < 0) {
          resetTimeToWait();
          agent.SetDestination(patrolArea.GetRandomPoint());
        }
      }
    } else if (currentState == State.FOLLOW) {
      agent.speed = 30;
      anim.SetBool("Attack", false);
      anim.SetBool("isWalking", true);
      agent.isStopped = false;
      agent.SetDestination(player.position);
      if (Vector3.Distance(transform.position, player.position) < 25) {
        currentState = State.ATTACK;
      }
    } else if (currentState == State.ATTACK) {
      if (Vector3.Distance(transform.position, player.position) < 15
      && Vector3.Distance(transform.position, player.position) > 5) {
        RunFrom();
      } else Attack();
    }
    if (nT < 0.45
    && anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_ATTACK)) bombThrowed = false;
    if (nT > 0.45 && nT < 0.47 && !bombThrowed
    && anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_ATTACK)) {
      GameObject bullet = GameObject.Instantiate(bulletPrefab, hand.position, hand.rotation);
      bullet.GetComponent < Rigidbody > ().AddForce(transform.forward * 1500
      + new Vector3(0, 250, 0), ForceMode.Force);
      bombThrowed = true;
    }
  } else {
    agent.isStopped = true;
    anim.SetTrigger("Death");
    if (anim.GetCurrentAnimatorStateInfo(0).normalizedTime > 0.8 f
    && anim.GetCurrentAnimatorStateInfo(0).shortNameHash.Equals(HASH_DEATH)) {
      Destroy(gameObject);
    }
  }
}
```

3.2.3 Dialogue System

Now it will present the three scripts that have managed the dialogs:

Dialogue

The **Dialogue** script is simply a class formed by the name of the NPC and some sentences that it has to say. This class will be used to build dialogs more easily.

```
public Dialogue(string name, string[] sentences) {
  this.name = name;
  this.sentences = sentences;
}
```

DialogueManager

The **DialogueManager** script appears as a single instance and is the one in charge of writing the **Dialogue** in the HUD. Its main functions are **StartDialogue()** to put all the sentences in a queue, **DisplayNextSentence()** to unqueue the sentences that then **TypeSentence()** will take care of writing slowly. Finally, **EndDialogue()** is in charge of closing the dialog canvas and, if the dialog that the player just had was from a quest, the next one would be unlocked.

```
public void StartDialogue(Dialogue dialogue) {
  dialogueBoxAnimator.SetBool("isOpen", true);
  isDialogueQuest = false;
  textName.text = dialogue.name;
  sentences.Clear();
  foreach(string sentence in dialogue.sentences) {
    sentences.Enqueue(sentence);
  }
  DisplayNextSentence();
}
public void DisplayNextSentence() {
  if (sentences.Count == 0) {
    StartCoroutine(EndDialogue());
    return;
  }
  string sentence = sentences.Dequeue();
  StopAllCoroutines();
  StartCoroutine(TypeSentence(sentence));
  //textSentence.text = sentence;
}
IEnumerator EndDialogue() {
  dialogueBoxAnimator.SetBool("isOpen", false);
  yield
  return new WaitForSeconds(0.5 f);
  FindObjectOfType < PlayerDialogueController > ().EndDialogue();
  if (isDialogueQuest && !newQuestUnlocked) {
    newQuestUnlocked = true;
    GameManager.instance.unlockNextQuestDialogueNPC();
```

```
29    }
30  }
31
32  IEnumerator TypeSentence(string sentence) {
33    textSentence.text = "";
34    foreach(char letter in sentence.ToCharArray()) {
35      textSentence.text += letter;
36      yield
37      return new WaitForSeconds(0.03 f);
38    }
39  }
```

DialogueTrigger

Finally, the **DialogueTrigger**, that is the script that each of the NPCs has. Simply every time a dialogue is going to be executed, the **TriggerDialogue()** function will be called which, depending on whether it is a quest dialogue, will execute one function of the **DialogueManager** or another. In case it is not a quest dialogue, the **chooseRandomDialogue()** function will be called so that one of the predefined dialogs is written.

```
1  public void TriggerDialogue() {
2    PlayerDialogueController.dialogueIsStarted = true;
3    if (isDialogueQuestActive)
4      FindObjectOfType < DialogueManager > ().StartDialogueQuest(new Dialogue(this.name,
5      dialogueQuest.sentences));
6    else
7      FindObjectOfType < DialogueManager > ().StartDialogue(chooseRandomDialogue());
8  }
9
10 private Dialogue chooseRandomDialogue() {
11   return dialoguesRandom[Random.Range(0, dialoguesRandom.Length)];
12 }
```

3.2.4 Inventory

For the consumable items management part of the inventory, three scripts have been used:

ConsumableItem

It's the script that is assigned to each of the potions that appear distributed throughout the map. With it the system can identify key variables such as its id, its name, its description and the associated image so that when taking one or the other in the HUD it is displayed correctly.

```
1  public class ConsumableItem: MonoBehaviour {
2    public int itemID;
3    public Sprite itemImage;
4    public string itemName;
```

```
  public string itemDescription;
  void OnTriggerEnter(Collider other) {
    if (other.gameObject.CompareTag("Player")) {
      GameManager.instance.setNewConsumableItem(itemID);
      Destroy(gameObject);
    }
  }
}
```

ConsumableInventory

It's the script associated with the canvas where, when starts the game, loads the prefabs of the empty items in the HUD grid layout and, each time a consumable is used or collected, the inventory is updated with the **refreshItemQuantity()** function.

```
private void Start() {
  consumablesInstantiated = new GameObject[prefabsConsumables.Length];
  for (int i = 0; i < prefabsConsumables.Length; i++) {
    GameObject go = Instantiate(prefabConsumableUI, gridLayout);
    Image[] images = go.GetComponentsInChildren < Image > ();
    TextMeshProUGUI[] texts = go.GetComponentsInChildren < TextMeshProUGUI > ();
    images[1].sprite = prefabsConsumables[i].itemImage;
    texts[0].text = GameManager.instance.getQuantityOfAnItem(i).ToString();
    texts[0].color = Color.red;
    texts[1].text = prefabsConsumables[i].itemName;
    texts[2].text = prefabsConsumables[i].itemDescription;
    go.GetComponent < InventorySlotIdClick > ().id = prefabsConsumables[i].itemID;
    consumablesInstantiated[i] = go;
    consumablesInstantiated[i].GetComponent < Button > ().interactable = false;
  }
}

public void refreshItemQuantity(int id) {
  consumablesInstantiated[id].GetComponentsInChildren < TextMeshProUGUI > ()[0].text =
   GameManager.instance.getQuantityOfAnItem(id).ToString();
  consumablesInstantiated[id].GetComponentsInChildren < TextMeshProUGUI > ()[0].color =
   int.Parse(consumablesInstantiated[id].GetComponentsInChildren < TextMeshProUGUI > ()[0].text) > 0
   ? Color.black : Color.red;
  consumablesInstantiated[id].GetComponent < Button > ().interactable =
   int.Parse(consumablesInstantiated[id].GetComponentsInChildren < TextMeshProUGUI > ()[0].text) > 0
   ? true : false;
}
```

ShowItemInfo

It's a simple script to manage when the player places or not the cursor on an item in the HUD to show the information of this.

3.2.5 Quest System

One of the fundamental pillars of the video game is the quest system. As the quests are basically of three types (talking with NPCs, collecting/transporting items and defeating enemies) a **Quest** script composed of an integer as the quest id and an active boolean was used to know if the quest was active or not. In this way, the logic of the Game Objects that had this script would only be executed at the moment in which the id of the current quest matches its associated id, with the Game Manager in charge of managing the change of quests.

```
public class Quest : MonoBehaviour
{
    public int id;
    public bool active;
}
```

QuestDialogueManager

A **QuestDialogueManager** script was also created so that any NPC could have different dialogues depending on which quest they were on. Thus, to do this, the same number of **DialogueTrigger** scripts as **Quest** were attached to each NPC so that the first **DialogueTrigger** coincided with the first **Quest**, the second with the second, etc.

So, the next piece of the **QuestDialogueManager** shows how depending on the **Quest** the character is in, the NPC will activate a **DialogueTrigger** or another.

```
public void ActiveNextDialogueQuestInNPC() {
  if (questPlayerIsGonnaExecute < quests.Length - 1) {
    quests[questPlayerIsGonnaExecute].enabled = false;
    dialogueTriggers[questPlayerIsGonnaExecute].enabled = false;
    questPlayerIsGonnaExecute += 1;
    quests[questPlayerIsGonnaExecute].enabled = true;
    dialogueTriggers[questPlayerIsGonnaExecute].enabled = true;
  } else {
    dialogueTriggers[questPlayerIsGonnaExecute].disableDialogueQuest();
  }
}

public DialogueTrigger dialogueTriggerCurrentQuest() {
  return dialogueTriggers[questPlayerIsGonnaExecute];
}
```

Finally, as each quest was executed differently, various scripts were created for each of them, thus isolating them from each other to avoid conflicts (PickShield.cs, MakeOffer.cs, FightKukulkanEnemies.cs, PickCorn.cs, UnlockFireHability.cs and BurnWood.cs).

3.3 Art

In this section it will be presented the modeling of the characters, the terrain creation, the elements of the HUD, the animations and all the assets and elements that make up

the artistic part of the video game.

3.3.1 Modeling

For the modeling of the main character and the enemies, **Blender**'s modifiers were used such as **Subdivision Subsurface** to soften the edges of the figures and the **Mirror** to facilitate modeling making the objects completely symmetrical.

Main character

For the creation of the main character (See Figures 3.3 and 3.4), it wanted to make a characterization so that with just a few elements that tribal aspect, indigenous to Mesoamerica of that time, was represented. In this way, the 3 elements that give K'an personality are **her hair, her mask and her poncho**. As she had to attend to a low poly style, there isn't much detail in terms of textures since it simply use flat colors and a very similar chromatic range with cold tones in the suit, poncho and mask.

Figure 3.3: K'an Model

So that the character wasn't so "static", physics were applied (See Figure 3.5) to both the poncho and the hair using a **Cloth** [24] modifier and instantiating a series of collider in each of his bones or in areas where necessary in order to generate that fluid and realistic movement.

Enemies

Both enemies were modeled with a similar structure since at the beginning a "base" model (See Figure 3.6) was created with a poor characterization so that changing a few

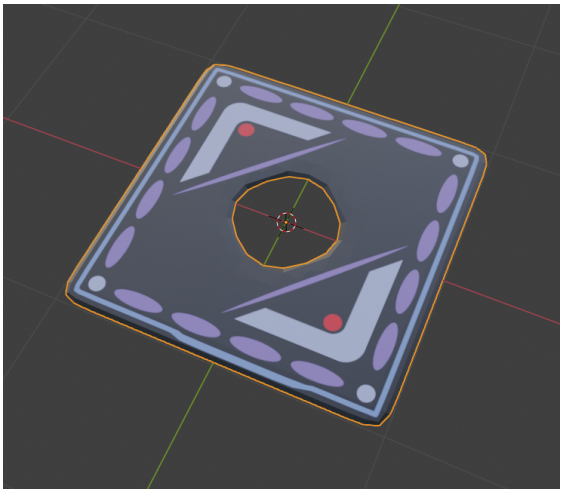


Figure 3.4: Poncho Model

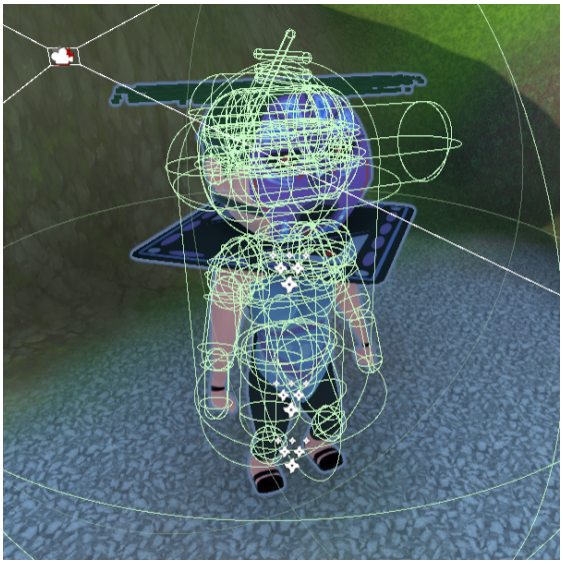


Figure 3.5: Colliders attached to K'an Skeleton

things would create different characters but with the same essence (See Figures 3.7 and 3.8). It was understood that the enemies had that cartoon and low poly aspect that can be seen throughout the game but that they also imposed fear on the player.

For this, they are very large humanoid beings with very large eyes who are dressed in clothes that could refer to the costumes that the Spanish used during colonization with their morions, braces, etc.

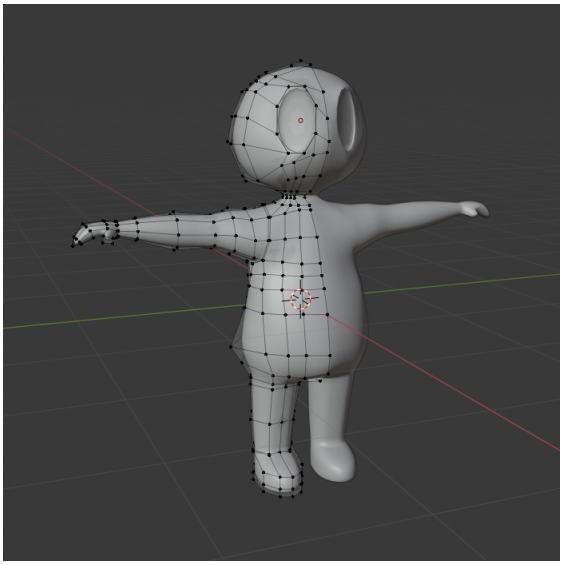


Figure 3.6: Enemy Base

NPCs

For the creation of the NPCs, an Unwrap was applied to the main character's maya to make a series of reprints (See Figure 3.9) on the poncho, mask and clothing. Thus, with few changes, different characters could be built (See Figure 3.10). For example, different hairstyles were also added to them or simply that they wore a hat on their head.

Locations

To generate the map decorations, only the El Caracol was modeled (See Figure 3.11), all other elements being third-party assets, which will be explained later.

3.3.2 Terrain Creation

For the creation of the level, the Terrain [25] object from Unity was used in which, with brushes, the terrain could be flattened or lifted, painted with textures such as grass, limestone, earth, etc. Sprites of plants and flowers and 3d models of trees (See Figure 3.12) were also taken from various assets that were instantiated in the same way with

Figure 3.7: Miner Model

Figure 3.8: Thrower Model

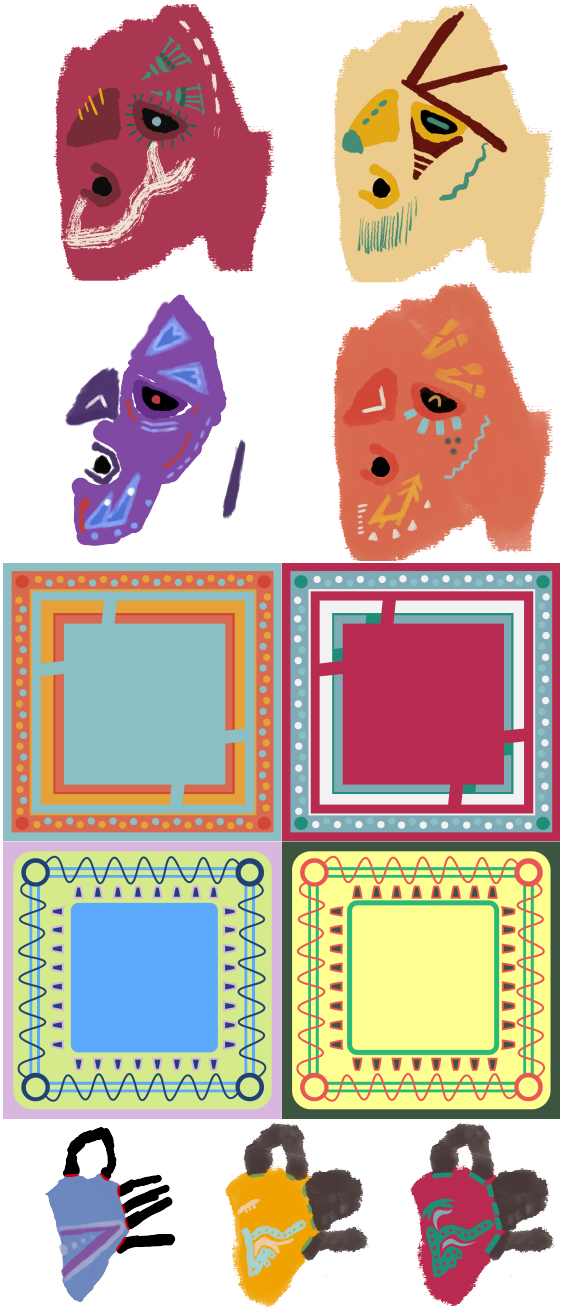


Figure 3.9: Texture reprints of the mask, the poncho and the cloths

Figure 3.10: Examples of different NPCs In-Game

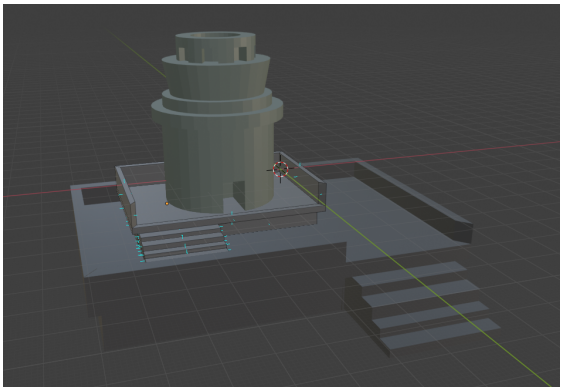


Figure 3.11: Caracol Model

the brushes.
The entire stage was designed in such a way that the details appeared in a proportionate way and didn't disturb the gameplay, but was merely an accompaniment to the stage.

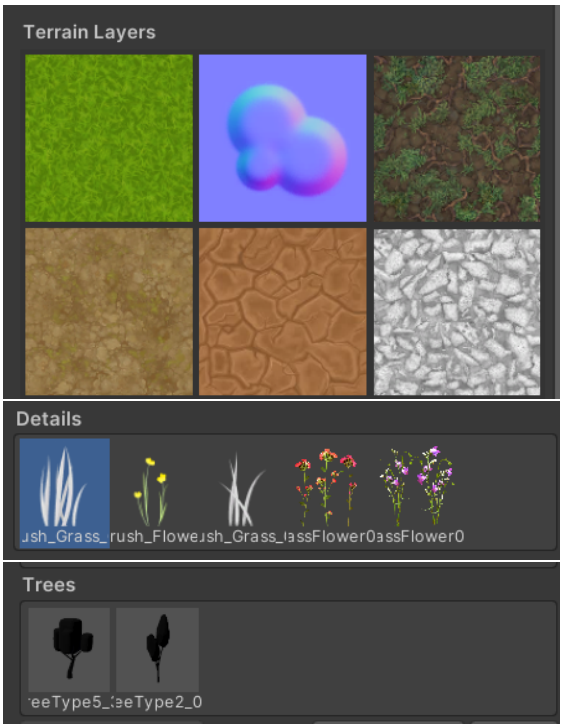


Figure 3.12: Terrain Textures, Flower Details and Trees

3.3.3 Character Animations

For the animation, all of them have been taken from the website **Mixamo**. There, the meshes were uploaded and an autorigging [26] was performed so any animation could be attached easily. Every character has an Animator Controller (See Figure 3.13, 3.14 and 3.15) filled with various animations that change between them depending an specific parameters managed in the scripts.

What's more, some of these animations were put together in blend trees [27] to make it easier to handle and thus avoid putting too many transitions and parameters for very similar movements. For example, as it can be seen in the Figure 3.13, a Blend Tree was used for the walking and running animation, and then other loose animations such as jumping, falling, aiming, shooting or dying.

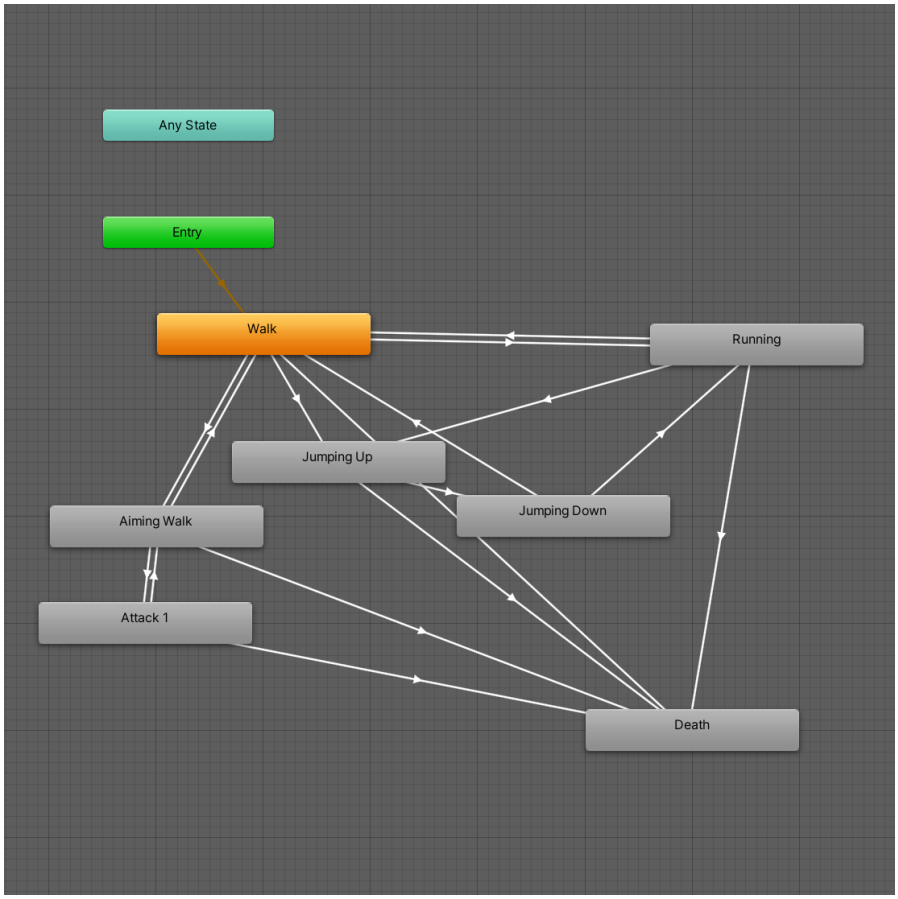


Figure 3.13: K'an Animator Controller

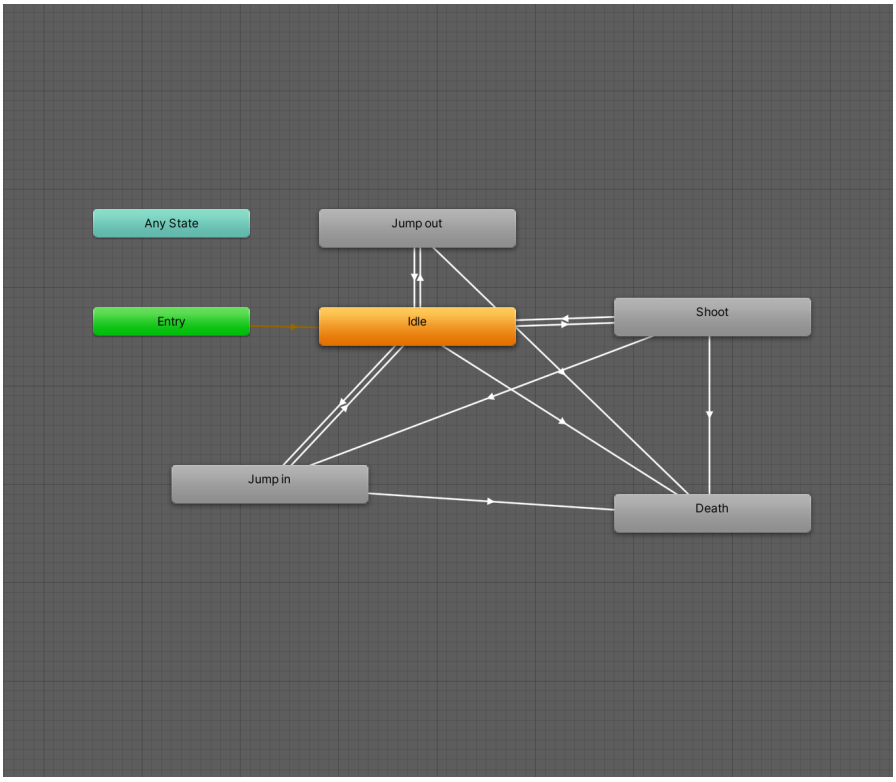


Figure 3.14: Miner Animator Controller

3.3.4 Minimap

For the creation of the minimap (See Figure 2.8), it was drawn a vector image with indigenous and tribal artistic motifs and a typeface similar to that used in westerns. For the programming of this, an orthographic camera was positioned on the stage where the K'an icon was simply rendered (to determine the position and direction) within a Render Texture.
This texture was shown in the inventory within a Raw Image, so putting the image made in Illustrator underneath it gave the feeling that the character was actually moving through the image.

3.3.5 Assets

Most of the third-party imported assets that have been used in the game have been mainly for decorative elements of the map and to improve the aesthetic part of the video game. Thus, for the decoration of the environment, asset packs have been used [28, 29, 30, 31], which included 3d models of rocks, trees, logs, flowers, plants, carts, cabins, tents, etc. Examples of this assets can be seen in Figure 3.16.

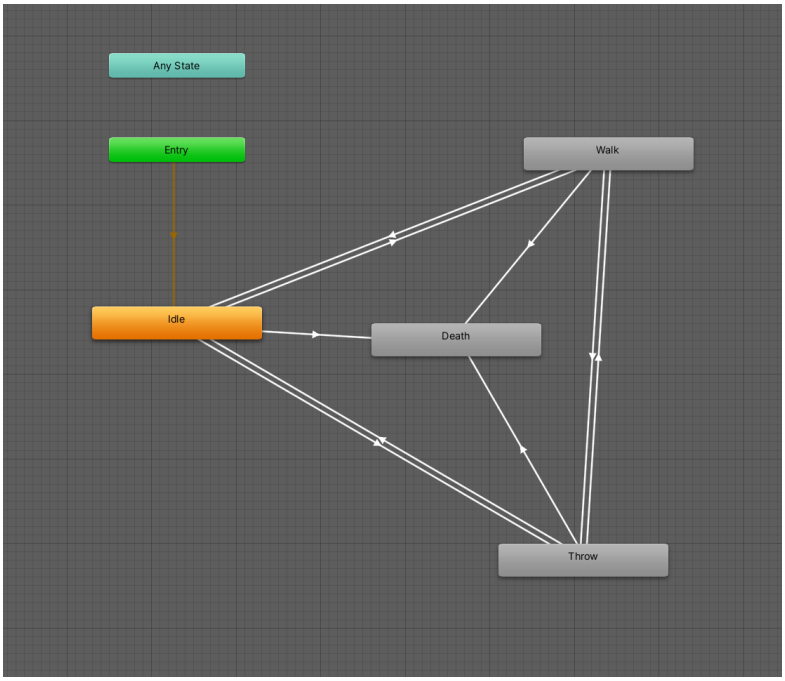


Figure 3.15: Thrower Animator Controller

Various assets have also been used for visual effects such as particles [32, 33, 34, 35] for explosions, fire, magic and camera effects. Examples of this assets can be seen in Figure 3.17.

3.4 Music and Sounds

3.4.1 Overall goals

For the sound section, an intro music is played in the main menu [36] and a quiet background music during the execution of the game [37]. For the SFX, some of effects represents actions of the player like when he moves [38], when he jumps [39], when receives damage [40] or when he talks to a NPC [41]. In the same way, there are some effects for actions of the enemies like shooting [42], exploding a bomb [43] or dying. [44]. Some relevant events in the game have also an associated sound like pressing a button [45], collecting an item [46], using a potion [47], completing a quest [48] and unlocking a new hability [49].

All these sounds have been extracted from YouTube where the value of the search was to find that they recreate that tribal and indigenous environment that is shown in the game.

Figure 3.16: Examples of Imported Assets

Figure 3.17: Examples of Particles

Chapter 4

Project Monitoring

Contents

In this chapter, all aspects related to the planning of the project, changes in the mechanics and construction of the game, problems that arose during its development and the solutions that were found will be presented.

4.1 Planning Control

The Figure 4.1 shows how the planning of the entire project has been structured, indicating the possible delays and the estimated times for each of the tasks.

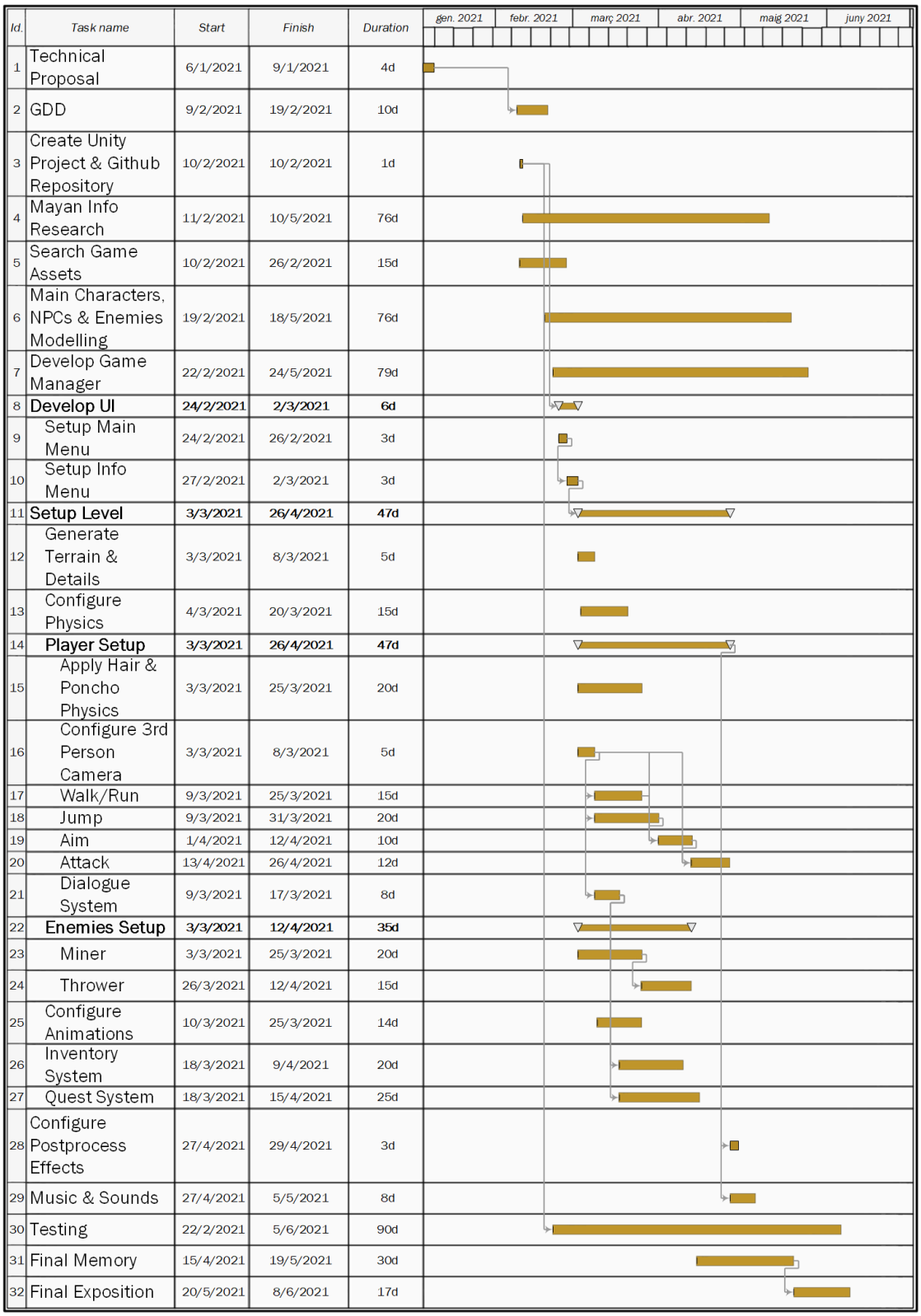
4.2 Changes and Evolution

From the beginning of the project, the idea was to make a video game that was set in nature. In the first instance it was going to be simply a walking simulator type adventure where a landscape would be shown, the character would be made of leaves and a story would be told.

Later the idea changed to a slightly more dynamic video game in the style of the Flower video game (Thatgamecompany, 2009) [50] where the player was made up of leaves and was avoiding obstacles in order to reach the end of a level with the maximum number of leaves possible. Finally, the idea of the project was re-evaluated since it was

a little bit poor of content and it opted for simply making a role-playing video game, maintaining at all times the first idea of placing it in an open stage set in nature.

With this proposal already established, a search for information was made since the idea was to make a video game with indigenous tribes and it seemed a good option to find out if throughout history some tribes had united in order to defeat a foreigner. In this way, a dystopian story was made narrating the opposite that happened during the fall of Maya civilization.



Id.	Task name	Start	Finish	Duration
1	Technical Proposal	6/1/2021	9/1/2021	4d
2	GDD	9/2/2021	19/2/2021	10d
3	Create Unity Project & Github Repository	10/2/2021	10/2/2021	1d
4	Mayan Info Research	11/2/2021	10/5/2021	76d
5	Search Game Assets	10/2/2021	26/2/2021	15d
6	Main Characters, NPCs & Enemies Modelling	19/2/2021	18/5/2021	76d
7	Develop Game Manager	22/2/2021	24/5/2021	79d
8	**Develop UI**	**24/2/2021**	**2/3/2021**	**6d**
9	Setup Main Menu	24/2/2021	26/2/2021	3d
10	Setup Info Menu	27/2/2021	2/3/2021	3d
11	**Setup Level**	**3/3/2021**	**26/4/2021**	**47d**
12	Generate Terrain & Details	3/3/2021	8/3/2021	5d
13	Configure Physics	4/3/2021	20/3/2021	15d
14	**Player Setup**	**3/3/2021**	**26/4/2021**	**47d**
15	Apply Hair & Poncho Physics	3/3/2021	25/3/2021	20d
16	Configure 3rd Person Camera	3/3/2021	8/3/2021	5d
17	Walk/Run	9/3/2021	25/3/2021	15d
18	Jump	9/3/2021	31/3/2021	20d
19	Aim	1/4/2021	12/4/2021	10d
20	Attack	13/4/2021	26/4/2021	12d
21	Dialogue System	9/3/2021	17/3/2021	8d
22	**Enemies Setup**	**3/3/2021**	**12/4/2021**	**35d**
23	Miner	3/3/2021	25/3/2021	20d
24	Thrower	26/3/2021	12/4/2021	15d
25	Configure Animations	10/3/2021	25/3/2021	14d
26	Inventory System	18/3/2021	9/4/2021	20d
27	Quest System	18/3/2021	15/4/2021	25d
28	Configure Postprocess Effects	27/4/2021	29/4/2021	3d
29	Music & Sounds	27/4/2021	5/5/2021	8d
30	Testing	22/2/2021	5/6/2021	90d
31	Final Memory	15/4/2021	19/5/2021	30d
32	Final Exposition	20/5/2021	8/6/2021	17d

Figure 4.1: Gantt Chart

CHAPTER 5

Results

Contents

In this chapter, the results that have finally been achieved are presented, establishing the limitations that have been found at the time of development, in what way the testing has been carried out as functionalities were included and most of the problems that appeared. In Figure 5.2 it can be seen some game captures of the final result where an image of a combat versus a miner, of a dialogue with Kabáh in town and two of the landscape and environment of the level are shown.

5.1 Testing

For the integration of all the functionalities, all the elements of the game have been built little by little, keeping a version control in a repository on Github. Firstly, it was started with the modeling of K'an, importing her into Unity and developing the PlayerController, creating the walking, running and jumping mechanics along with their corresponding animations. As physics had to be used for the main movements, it was tested for several days in order to solve problems with the RigidBody.

Afterwards, it began with the modeling of the enemies, their import into Unity and the development of the patrol and attack scripts, also putting and synchronizing their animations. Once both the character and the enemies were working correctly, the stage began to be set up. Various tests were carried out on the movement of the enemies since,

as they had to move through a NavMesh, there were times that they did not recalculate the spots well and ended up staying fixed without moving.

With the stage set up, the patrols of the NPCs and the dialogues were programmed. From there the entire inventory system began and finally with the quest system. With the level already in the Alpha phase, some bugs with the enemies were finished and the reprinting of the NPCs was finished.

5.2 Limitations

Some limitations that have been found all along the project are:

- **Time** has been one of the main limitations since, having all the basic programming developed, with a longer time it would have been possible to build more levels and continue expanding the story.
- The **High Definition Render Pipeline (HDRP)** [51], due to the fact that at the beginning of the project an interactive shader for grass was implemented [52] but in the end it could not be included since when changing to the Universal Render Pipeline [53] it did not allow to compile it.

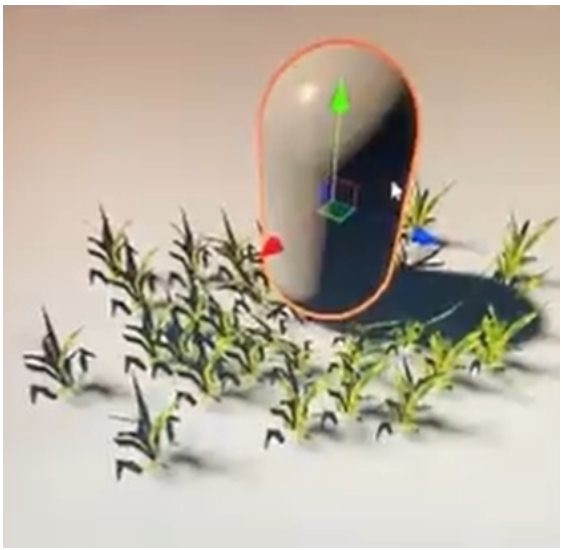


Figure 5.1: Example of the Interactive Grass shader

5.3 Problems and Solutions

Now the list of problems that have occurred during the development of the project will be presented in chronological order. In the first place, one of the problems that have

been present since the beginning of the project has been the **jump and move** of the character since, as physics had to be used, many times the synchronization of the jump with the applied force was not carried out correctly or wrong transitions were made towards other different animations. As the terrain was used, there were many times that it did not detect when it was grounded and could jump indefinitely. Many changes were made in the RigidBody parameters (mass, drag, freeze rotations, etc) and in PlayerController variables (jump force, jump time, grounded) until finally there was an good jump.

Another problem was the initial use of **HDRP** since it gave many compatibility problems and there were times that the camera did not render certain objects or all the materials of the scene were simply erased and they turned pink. Also the HDRP gave problems with the Terrain since it did not allow details such as flowers or plants to be placed.

Also mention that a lot of hours were dedicated to **the poncho and the hair** since they caused many problems when making collisions with other objects but in the end it was quite natural.

Finally, comment that there were quite a few bugs with the **movement of the enemies and the attacks**, especially with the thrower since many times it instantiated more bombs than it should or simply did not point towards the player, but they have been fixed in the final version of the game.

Figure 5.2: In-Game Captures of Final Result

Chapter 6

Conclusions

Contents

In this chapter, the conclusions of the work, as well as its future extensions are shown.

6.1 Objectives achieved

As a final conclusion, it can be seen that all the objectives described in section 1.2 have been achieved in the project.
First of all, it has **integrated the modeling of the main character, the enemies and the npcs** within the game engine, providing them with animations and assigning specific actions to each one through scripting.

In the same way, it has been possible to implement the **basic elements of role-playing video games** such as the complete movement of the character, the quest system, the character's abilities, the inventory with the minimap and the collectible items, the attacks and movements of the enemies, the dialogue system and the unlocking of new areas.

It has also made **the developing of a plot** that involves different characters and places in an open world through dialogues and in-game events.

And as the last objective achieved, it has been possible to **create a level design** that allows the player to go through all the important locations, maintaining a balance between the mechanics that the player can permorm and the quest that he has to complete.

In this way, the game shows up a large amount of programming of the different mechanics and visual elements that follow a specific aesthetic, applying so many of the knowledge learned in the degree. It is definitely a game that it could be considered *academically balanced* because of most of the knowledge learned in the degree of programming as well as art and narrative is applied.

6.2 Future work

For future development, **new levels** could be created to continue with the main narrative of the video game since there is already a large amount of programming explicitly prepared to create more levels.

New mechanics could be included such as a save system and control points to facilitate user gameplay and allow quitting the game at any time, and a decision system with multiple paths where the dialogues with the NPCs are more complex and the player has the option to decide how K'an will act, with each act having repercussions in the future.

A **monetization system** could also be integrated where the player has to collect coins around the map or simply drop them by enemies when they die. With these coins, there could be shops where the player can buy potions, new abilities or different outfits for the main character.

Finally, it could be interesting to integrate new enemies or a **final boss** at the end of each level where the player puts into practice all the habilities that he has developed throughout the game.

Bibliography

[1] U. Documentation, "Occlusion culling." https://docs.unity3d.com/560/Documentation/Manual/OcclusionCulling.html. Accessed: 2021-05-22.

[2] Wikipedia, "Fast approximate anti-aliasing." https://en.wikipedia.org/wiki/Fast_approximate_anti-aliasing. Accessed: 2021-05-22.

[3] Wikipedia, "Finite-state machine." https://en.wikipedia.org/wiki/Finite-state_machine. Accessed: 2021-05-22.

[4] Wikipedia, "Game design document." https://en.wikipedia.org/wiki/Game_design_document. Accessed: 2021-05-21.

[5] Wikipedia, "Spanish conquest of the maya." https://en.wikipedia.org/wiki/Spanish_conquest_of_the_Maya. Accessed: 2021-05-22.

[6] Wikipedia, "El caracol, chichen itza." https://en.wikipedia.org/wiki/El_Caracol,_Chichen_Itza. Accessed: 2021-05-22.

[7] Wikipedia, "Hunab ku." https://es.wikipedia.org/wiki/Hunab_Ku. Accessed: 2021-05-22.

[8] Wikipedia, "Mayapan." https://en.wikipedia.org/wiki/Mayapan. Accessed: 2021-05-22.

[9] Wikipedia, "Sacred cenote." https://en.wikipedia.org/wiki/Sacred_Cenote. Accessed: 2021-05-22.

[10] Wikipedia, "El castillo, chichen itza." https://en.wikipedia.org/wiki/El_Castillo,_Chichen_Itza. Accessed: 2021-05-22.

[11] Mitologia.info, "Kauil." https://mitologia.info/kauil-dios-maya-del-fuego/. Accessed: 2021-05-22.

[12] Wikipedia, "Cozumel." https://en.wikipedia.org/wiki/Cozumel. Accessed: 2021-05-22.

[13] Wikipedia, "Mesoamerican ballgame." https://en.wikipedia.org/wiki/Mesoamerican_ballgame. Accessed: 2021-05-22.

[14] Wikipedia, "Morion." https://en.wikipedia.org/wiki/Morion_(helmet). Accessed: 2021-05-22.

[15] E. camino más corto, "The sacred corn in the life and food of the mayans." https://en.elcaminomascorto.es/maiz-alimentacion-mayas/. Accessed: 2021-05-22.

[16] Wikipedia, "Sacbe." https://en.wikipedia.org/wiki/Sacbe. Accessed: 2021-05-22.

[17] D. Entertainment, "Aer memories of old - release trailer." https://www.youtube.com/watch?v=ldJr6nUDqAs&ab_channel=DaedalicEntertainment. Accessed: 2021-05-22.

[18] Nintendo, "The legend of zelda: Breath of the wild - nintendo switch presentation 2017 trailer." https://www.youtube.com/watch?v=zw47_q9wbBE&ab_channel=NintendoNintendo. Accessed: 2021-05-22.

[19] Findicons, "Orbz icon pack." https://findicons.com/pack/164/orbz. Accessed: 2021-05-22.

[20] Pngwing, "Dialog box." https://www.pngwing.com/en/free-png-kelsv. Accessed: 2021-05-22.

[21] Wikipedia, "Functional requirement." https://en.wikipedia.org/wiki/Functional_requirement. Accessed: 2021-05-22.

[22] Wikipedia, "Gaussian blur." https://en.wikipedia.org/wiki/Gaussian_blur. Accessed: 2021-05-21.

[23] U. Documentation, "Building a navmesh." https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html. Accessed: 2021-05-22.

[24] U. Documentation, "Cloth." https://docs.unity3d.com/es/2018.4/Manual/class-Cloth.html. Accessed: 2021-05-22.

[25] U. Documentation, "Creating and editing terrains." https://docs.unity3d.com/Manual/terrain-UsingTerrains.html. Accessed: 2021-05-22.

[26] Wikipedia, "Skeletal animation." https://en.wikipedia.org/wiki/Skeletal_animation. Accessed: 2021-05-22.

[27] U. Documentation, "Blend trees." https://docs.unity3d.com/Manual/class-BlendTree.html. Accessed: 2021-05-22.

[28] A. Bringer, "Simplistic low poly nature." https://assetstore.unity.com/packages/3d/environments/simplistic-low-poly-nature-93894. Accessed: 2021-05-21.

[29] Elcanetay, "Low poly nature - free vegetation." https://assetstore.unity.com/packages/3d/vegetation/low-poly-nature-free-vegetation-134006. Accessed: 2021-05-22.

[30] S. Night, "Low poly pack - environment lite." https://assetstore.unity.com/packages/3d/props/exterior/low-poly-pack-environment-lite-102039. Accessed: 2021-05-21.

[31] . S. Robots and Counting..., "Free low poly desert pack." https://assetstore.unity.com/packages/3d/environments/free-low-poly-desert-pack-106709. Accessed: 2021-05-21.

[32] M. Carnivore, "Particle ribbon." https://assetstore.unity.com/packages/vfx/particles/spells/particle-ribbon-42866. Accessed: 2021-05-21.

[33] PavelDoGreat, "Super blur." https://github.com/PavelDoGreat/Super-Blur. Accessed: 2021-05-21.

[34] R. Knight, "Fantasy skybox free." https://assetstore.unity.com/packages/2d/textures-materials/sky/fantasy-skybox-free-18353. Accessed: 2021-05-21.

[35] M. Inc., "Effect textures and prefabs." https://assetstore.unity.com/packages/vfx/particles/effect-textures-and-prefabs-109031. Accessed: 2021-05-21.

[36] Fantasy and W. M. by the Fiechters, "Tribal jungle music - mayan pyramids." https://www.youtube.com/watch?v=QYUdwh2nTWs&ab_channel=Fantasy%26WorldMusicbytheFiechters). Accessed: 2021-05-23.

[37] Fantasy and W. M. by the Fiechters, "Rpg towns music and rpg villages music." https://www.youtube.com/watch?v=xu2pESvXcmM&ab_channel=Fantasy%26WorldMusicbytheFiechtersFantasy%26WorldMusicbytheFiechters. Accessed: 2021-05-23.

[38] S. E. SoundEffectsONE, "Steps and stepping." https://www.youtube.com/watch?v=A2JGpBHqucM&ab_channel=SoundEffects-SoundEffectsONESoundEffects-SoundEffectsONE. Accessed: 2021-05-23.

[39] S. E. Free, "Jump sound effect." https://www.youtube.com/watch?v=z8O8mu365Rc&ab_channel=SoundEffectsFreeSoundEffectsFree. Accessed: 2021-05-23.

[40] S. E. N. Copyright and F. Download, "Damage sound effect pack." https://www.youtube.com/watch?v=Uwh5iyQhy1w&ab_channel=SoundEffects-NoCopyright%26FreeDownloadSoundEffects-NoCopyright%26FreeDownload. Accessed: 2021-05-23.

[41] B. Tris, "Dialogue blip." https://www.youtube.com/watch?v=KG4Y27IeSV4&ab_channel=BTris. Accessed: 2021-05-23.

[42] Q. Productions, "Free gunshot sound effects." https://www.youtube.com/watch?v=KXYfW8AUh0c&ab_channel=QuandaryProductions. Accessed: 2021-05-23.

[43] Rikki, "Explosion sound effct." https://www.youtube.com/watch?v=Ruwoj_qzCLA&ab_channel=rikki. Accessed: 2021-05-23.

[44] S. Lah, "Growtopia death sound effect." https://www.youtube.com/watch?v=wVcTdCPFTug&ab_channel=SidanLahSidanLah. Accessed: 2021-05-23.

[45] A. Alake, "Sound effect - button press." https://www.youtube.com/watch?v=hXD78SxXsD4&ab_channel=AdamAlakeAdamAlake. Accessed: 2021-05-23.

[46] S. E. Database, "Arcade game collect item sound effect." https://www.youtube.com/watch?v=Zg_20JvQxFU&ab_channel=SoundEffectDatabaseSoundEffectDatabase. Accessed: 2021-05-23.

[47] S. Source, "Collect item sound effect." https://www.youtube.com/watch?v=BVXUp8RaVWI&ab_channel=SFXSource. Accessed: 2021-05-23.

[48] C. Fluke, "Level up sound effect." https://www.youtube.com/watch?v=P_u0k2uElHI&ab_channel=CPhTFlukeCPhTFluke. Accessed: 2021-05-23.

[49] 2MirrorsDialogue, "Magic sound effects." https://www.youtube.com/watch?v=O5Y1QGG0q90&ab_channel=GamingSoundEffectsGamingSoundEffects. Accessed: 2021-05-23.

[50] Wikipedia, "Flower (video game)." https://en.wikipedia.org/wiki/Flower_(video_game). Accessed: 2021-05-22.

[51] U. Documentation, "Getting started with the high definition render pipeline." https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/Getting-started-with-HDRP.html. Accessed: 2021-05-22.

[52] K. B. 14, "Unity 3d interactive grass shader graph." https://www.youtube.com/watch?v=zm7rKXEPa9M&ab_channel=KAYB14. Accessed: 2021-05-22.

[53] U. Learn, "Introduction to urp." https://learn.unity.com/tutorial/introduction-to-urp. Accessed: 2021-05-22.