End of degree project Memory

Full Vertical Slice development of a videogame to reach the publisher/influencer's attention

Gerard Ardit Castells

End of degree project

Videogame Design and Devlopment degree: course 2021-2022

Universidad Jaume I

May 11, 2022

Project tutored by: Michael Gould

To Esther

Acknowledgements

First of all, I would like to thank my final degree project director, Michael Gould, for following up on the work.

To Esther, my life partner who has accompanied me on this journey, who has helped and supported me at all times.

To my family, who have been a fundamental support throughout the whole process.

Lastly, I would like to thank Sergio Barrachina Mir for his «Plantilla LaTeX for the writing of the thesis report", which I have used as a starting point for the writing of this report. used as a starting point for the writing of this report.

Summary

In this work we are going to make a vertical slice of a 3D platform game with the intention of catching the attention of a publisher. Throughout the work you will be able to read about the course of development over the weeks.

Contents

Chapter 1

Introduction

Índice

1.1 Project motivation

The main reason for choosing this thematic for the final thesis is to satisfy some personal concerns related to entrepreneurship and self-improvement. Another reason why this thematic fits my preferences is the possibility of creating an employment opportunity to be able to finish college and start a job.

Regarding entrepreneurship, there has always resided in me a spirit that has pushed me to not only have ideas, but also to put them into practice and try to exploit them.

As for the opportunity to generate a job opportunity, it is true that one of the main motivations when choosing the subject was this, however, over the course of time, this need has already been satisfied since I have obtained a job opportunity in other ways. Even so, I still find interesting the idea of creating a project with these characteristics, because I think that if it turns out well, it would be a very interesting point to highlight in my future portfolio and a nice flag to plant in my upcoming career as a professional.

In addition, another of the motivations for doing the project, is to prove to myself that I am able to create, from scratch and by myself, the basic systems that a platform style game needs, not only to be playable, but to be good enough to cause good impressions, at all levels, to anyone who is going to play it.

Finally, there is also an intention to create a series of libraries and tools of my own from this project that I can use and extend to facilitate future game developments.

1.2 Aims of the project

The ultimate goal of the project is to make, from scratch, a product, albeit short, playable, and that can capture the attention of a publisher in a short period of time.

For this, it is necessary that the systems to be developed in the project work smoothly and satisfactorily and without many bugs. However, it is understood that this is an early version of the project and the appearance of some bugs is tolerated, although, of course, the fewer the better.

Also, the project is going to require an art that, although simple, it has to be pleasant to see for the player, implying that I do not intend to be the one in charge of making the art in a complete and finished version of the game, but I can be the one who controls and monitors the management of it, as I do have notions in the field.

Also, as mentioned above, from this project, I intend to create my own library of scripts, prefabs, functionalities and various tools that will allow me, in the future, to develop video games more easily.

Finally, it is also necessary that all this is accompanied by a previously thought, tested and quality design to make all of the above make sense and that all together form a memorable experience.

1.3 Environment and initial state

To develop the project we will only count on my work throughout the total hours stipulated by the course.

As mentioned above, we want to start from the crudest possible basis, that is, with the basic Unity 3D template. Without plugins or external tools.

From there, the different systems that will build the basis for a 3D platform videogame will be created. However, we do not rule out the option of adding some tools for Unity's own development, such as, for example, cinemachine, to lighten the time in case it is necessary.

For the development section, the VisualStudio 2019 [6]tool will be used together with the Unity version 2020.3.22f1. [4]

For the artistic section, in terms of visuals, Blender 2.8 will be used for the creation of 3D assets, while Adobe Photoshop CS6.v13.0 [8] will be used for the creation of textures. While we expect to use free online libraries for the sound part.

For the preparation of the Gantt chart, Google Sheets has been used. [3]

For the generation of this document, Overleaf has been used. [12]

To keep track of the tasks in more detail and to control their progress, Notion was used. [7]

For version management, GitHub was used. [10]

CHAPTER 2

Resource planning and evaluation

Índice

2.1 Planning

For the development of the vertical slice we have relied on the traditional methodology used in most of the companies that make up the video game industry, which is also the one that has been taught in some of the subjects of the degree. That is, starting the project from pencil and paper, designing every little aspect of what will be developed in the future, taking into account, of course, that it is virtually impossible to accurately predict all the content of a game and anticipate all the decisions to be taken, so, although the design part is the first thing that will be worked on, this may vary throughout the development. This phase will last, at least, during the 3-5 weeks of development, developing issues such as the general idea and theme of the game, conceptualize the scenarios, make a design of the basic mechanics of the game or design the levels.

Secondly, the basic and fundamental mechanics of the game will be developed in order to test the viability of the previously elaborated design. It is important to keep in mind that in this phase of the development, the viability of the design will be checked, so it does not matter too much if in the development of the mechanics some minor bugs arise, because this phase has to be dynamic and agile in order to be able to iterate in the design phase if the results are not as expected. This process can be developed in parallel with the first one, although with some caution, and would last approximately until the first 5-6 weeks of development.

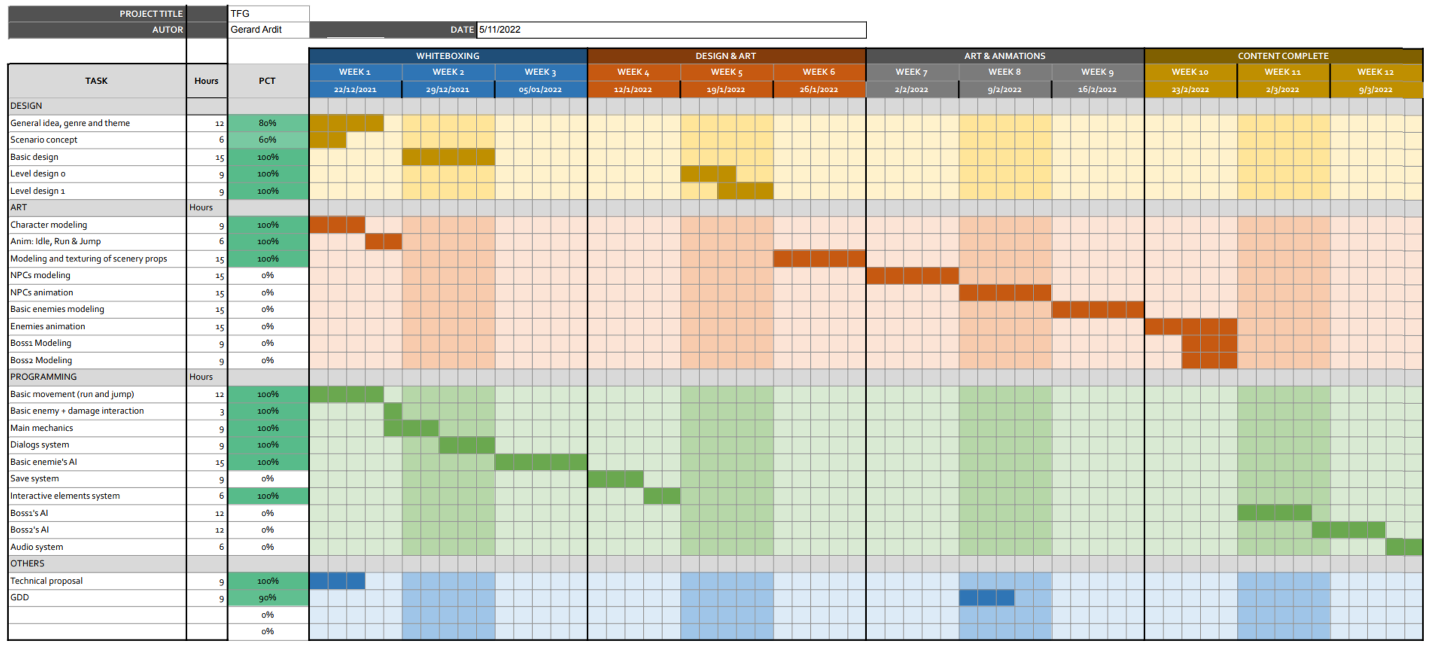
To continue, once the fundamental mechanics are already developed and we are satisfied with the results, we will make a round of confection of the most recurrent assets in order to give a little shape to the world, the main character, which involves modeling and main animations such as running, jumping, hitting... and even some basic enemies. This is so for a specific reason, and that is that, being that the development will be done by a single person, it is important to diversify the tasks, in order to make the process more enjoyable and avoid the "burnout" effect. Also, if for some reason, the project gets stuck in any of the future phases, at least there will be a part of it that will be easy to show. It is believed that the development of this phase would last from week 6 to week 10.

Once the artwork is done and implemented, the project will take shape and will have another face, it will start to look like a demo, although with more work to be done. The next phase will consist of the development of more secondary mechanics, more frequent enemies and elements that will appear later in the game, such as special enemies, or, for example, final bosses. It could also be used to polish some minor bugs that have been left pending in the first phase of development, although in the future a few weeks will be dedicated to the polishing of the game, where this type of tasks are included. This block corresponds to the following 5 weeks, that is to say, approximately, from week 10 to week 15 of development.

Following this week, it remains to make the remaining assets, less recurring assets, modeling and animations of secondary characters, including some extra animations for the main character, modeling and animations for final bosses. This phase of the project development could be summarized between 3 and 5 weeks.

Finally, the rest of the time left until the delivery of the project, efforts will be focused on polishing mechanics, implementing some sounds, generating a nice lighting for the project, finishing with the most important bugs, taking into account that this is still a demo/vertical slice, which allows the appearance of some minor bugs. In this phase we could also make some extra assets to give more life to the world, but we will avoid as much as possible everything related to the technical part, such as the implementation of new mechanics or enemies. This could lead to the appearance of bugs or unexpected errors that could lead to an undesired result.

A summary of the above, with a more detailed task breakdown, can be seen in the following Gantt chart (see Figure).



PROJECT TITLE	TFG		
AUTOR	Gerard Ardit	DATE	5/11/2022

			WHITEBOXING			DESIGN & ART			ART & ANMATIONS			CONTENT COMPLETE		
			WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10	WEEK 11	WEEK 12
TASK	Hours	PCT	22/12/2021	29/12/2021	05/01/2022	12/1/2022	19/1/2022	26/1/2022	2/2/2022	9/2/2022	16/2/2022	23/2/2022	2/3/2022	9/3/2022
DESIGN														
General idea, genre and theme	12	80%	🟨											
Scenario concept	6	60%	🟨											
Basic design	15	100%		🟨										
Level design 0	9	100%					🟨							
Level design 1	9	100%					🟨							
ART	Hours													
Character modeling	9	100%	🟧											
Anim: Idle, Run & Jump	6	100%	🟧											
Modeling and texturing of scenery props	15	100%						🟧						
NPCs modeling	15	0%							🟧					
NPCs animation	15	0%								🟧				
Basic enemies modeling	15	0%									🟧			
Enemies animation	15	0%										🟧		
Boss1 Modeling	9	0%										🟧		
Boss2 Modeling	9	0%										🟧		
PROGRAMMING	Hours													
Basic movement (run and jump)	12	100%	🟩											
Basic enemy + damage interaction	3	100%	🟩											
Main mechanics	9	100%	🟩	🟩										
Dialogs system	9	100%		🟩										
Basic enemie's AI	15	100%			🟩									
Save system	9	0%				🟩								
Interactive elements system	6	100%				🟩								
Boss1's AI	12	0%											🟩	
Boss2's AI	12	0%												🟩
Audio system	6	0%												🟩
OTHERS														
Technical proposal	9	100%	🟦											
GDD	9	90%								🟦				
		0%												
		0%												

Figure 2.1: Example of a Gantt chart (made with Google Sheets)

2.2 Resource evaluation

In this section, it is important to learn to differentiate between what would be the development of a complete project, and the vertical slice that is intended to develop in this TFG. They are two projects that, although related, are also very different in terms of the resources that must be allocated to each part. Thus, only the development costs will be mentioned with regard to the vertical slice.

Human costs: As mentioned in previous sections, the project will be developed by only one person over approximately 21 to 24 weeks, working, generally, a little less than half a day, which roughly sums up to about 315 hours at the end of the project. Given that the person in charge of the development can be classified as a junior developer, and taking into account the salaries offered in Spain to this profile, we will assume that the price to be paid for the development would be about 6.5€ per hour. Then, the human cost of that part would total, approximately, about 2000€.

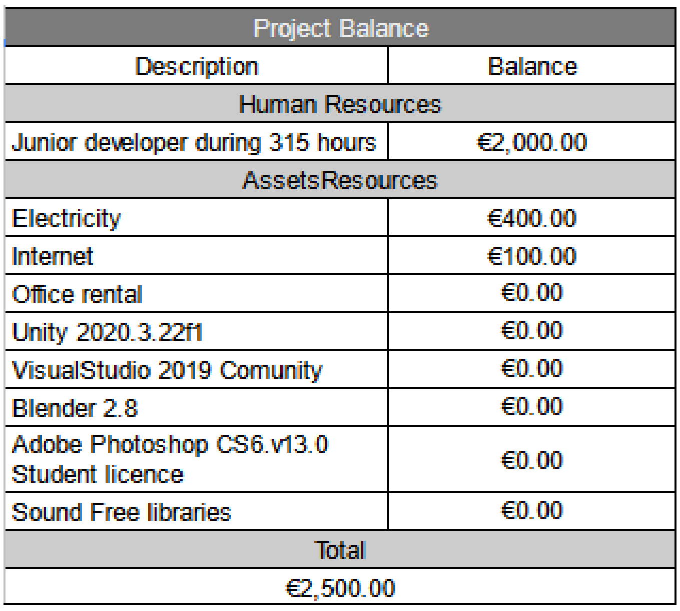
Equipment costs: Knowing the time in which the project will be developed, we can calculate approximately the costs of electricity and internet during the 5 or 6 months that the development will last, which would be about 500€ additional.

Taking into account that the development can be elaborated telematically, it would not be necessary to hire a physical office, so the costs of a rent could be discarded.

Regarding the material used for the development, student licenses and free licenses of the different software mentioned above are being used.

Initially, the financing of these costs will be provided by the use of the three F's: family, friends and fools.

The total sum of the different costs amounts to approximately 2500€.



Project Balance	
Description	Balance
Human Resources	
Junior developer during 315 hours	€2,000.00
AssetsResources	
Electricity	€400.00
Internet	€100.00
Office rental	€0.00
Unity 2020.3.22f1	€0.00
VisualStudio 2019 Comunity	€0.00
Blender 2.8	€0.00
Adobe Photoshop CS6.v13.0 Student licence	€0.00
Sound Free libraries	€0.00
Total	
€2,500.00	

Figure 2.2: Table the project balance (made with Google Sheets)

CHAPTER 3

System analysis and design

Índice

3.1 Requirements analysis

Before carrying out any development, it is necessary to perform a previous analysis of the analysis of the requirements of the project.

3.1.1 Functional requirements

In this project it is impossible to do without the following systems:

Input:	Horizontal and vertical movement input
Output:	Horizontal movement and jump
This system includes running, jumping in the air and jumping on the wall.	

Table 3.1: Functional requirement «Movement system»

Input:	Attack Input
Output:	The character will launch an attack in the direction where the character is facing
The player expects that if an enemy is encountered in the direction of the attack, it will take damage.	

Table 3.2: Functional requirement «Combat system»

Input:	Difficulty for sound implementation
Output:	Ease of implementation
System with which it is easy and simple so that the developer himself can implement any audio of any type in any place in an agile way.	

Table 3.3: Functional requirement «Sound system»

Input:	Any trigger
Output:	Effective storage without data loss
A system that does not allow the loss of progress	

Table 3.4: Functional requirement «Save system»

Input:	Any trigger
Output:	Efficient data save
A system that allows for the efficient updating of data in a timely manner.	

Table 3.5: Functional requirement «Checkpoints system»

Input:	Any trigger
Output:	Load the following scene and download the previous one
A system that allows concurrent loading and unloading of scenes without loading screens.	

Table 3.6: Functional requirement «Scene loading/unloading system»

Input:	Action input in front of a neutral NPC
Output:	Display of a text interface
A system that allows the player to interact with the different neutral npcs.	

Table 3.7: Functional requirement «Dialogue system»

Input:	Any input
Output:	An appropriate response to each input
A system that allows the player to interact with the game from any controller	

Table 3.8: Functional requirement «Hybrid control system»

Input:	Interaction input
Output:	A response adapted to each interaction
A system that allows the player to interact with any element that is desired.	

Table 3.9: Functional requirement «Interactables system»

3.2 System design

Motion system 3.1 : This is the most used system and the one with which the player will be most connected. It is important that these systems are responsive to the player's inputs, that the jump is fluid, trying to avoid the balloon effect. When the user will press the movement input, the result will be the displacement of the character in the world. While when he presses the jump input, he will get the character to jump vertically.

Combat system 3.2 : This is the second of the pillars that make up this video game. A combat system that is fair to the player must be developed, otherwise, this is one of the most frustrating systems. The user will press the input to attack, and the character will launch an attack towards the direction where he is looking. The player will expect that if an enemy is found in the direction where the attack has been launched, it will receive damage, so this is how the system should work. In addition, for the cases where the visual collision is adjusted, an extra margin will be given to the player with the physical collision of the attack.

Sound system 3.3 : A system must be developed with which it is easy and simple for the developer himself to implement any audio of any kind anywhere in an agile way. And, at the same time, it should be easy for the user to control the levels of each audio track, from the most general to the most specific.

Save system 3.4 : A system must be developed that does not allow the loss of the progress that the player has achieved so far. These, if they fail, are also systems that can generate a lot of frustration. Otherwise, if the save points are well designed and the system works well, it becomes very satisfying for the user.

Checkpoints system 3.5 : This system, although related to the previous one, at a technical level and, above all, in terms of resource consumption, has subtle differences when it is developed. This system must allow the developer to store only the necessary data at each checkpoint to avoid processing cost and memory load peaks.

Scene loading and unloading system 3.6 : As in the previous system, a good scene loading and unloading system can save the player from waiting times in obnoxious loading screens.

Dialog systems 3.7 : A system should be developed that allows the player to interact with the non-playable neutral characters in the game, the player expects to receive information in the form of conversations with the different characters in the game.

Hybrid control system 3.8 : The player must be able to choose the type of controller to play with via the virtual3.4 of the game, either keyboard and mouse or a controller.

Interactable system 3.9 : A system that allows the creation of all types of interactive elements in an agile way.

3.3 System architecture

The project can be divided into two main sections: Gameplay systems and functionality systems.

In the gameplay systems, we find the GameManager, the PlayerController and the enemies. These systems are located in the AlwaysScene, which is a persistent scene throughout the gameplay. This structure allows us to elaborate the functional requirement of loading and unloading scenes 3.6 without abrupt performance jumps.

The GameManager will allow us to manage the game status, that is, the whole system will be able to know at all times if the game is paused, in dialogue state or in normal execution. In addition, the GameManager will contain the UI Manager, from which the interface and menus are managed and also the SoundManager, which will make it accessible to the developer.

On the other hand, the PlayerController, will be in charge of managing everything related to the player, containing the MainCamera, the PlayerMovement and also all its Combat part.

Finally, there are systems that are independent from others, but bring a lot of value to the project.

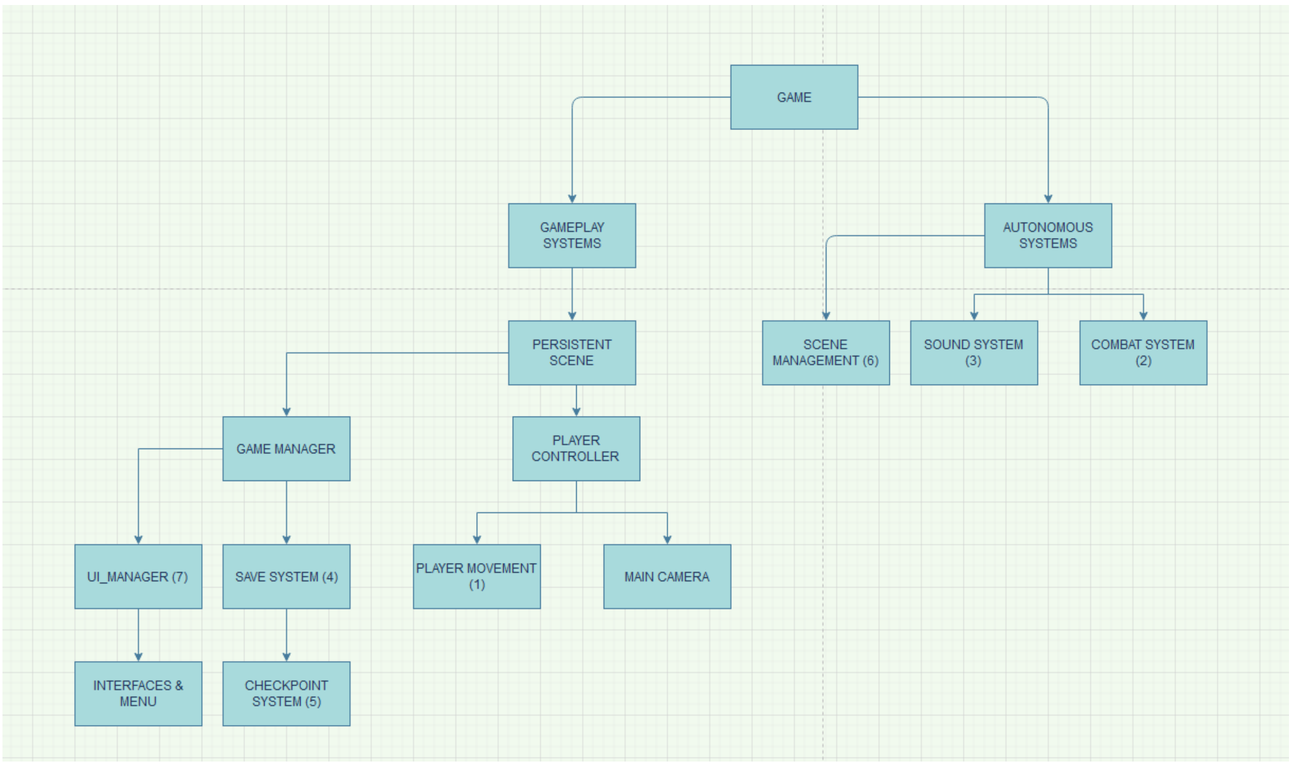


Figure 3.1: Diagram of the main project structure (made with Diagrams.net) [2]

3.4 User Interface Design

For the game interface, we intend to maintain a clean and neat user interface with as little information on screen at the same time as possible with the intention of generating the greatest possible immersion in the game, making each part of the game enjoyable to its fullest potential. To do this, the necessary information will be shown and hidden at all times.

To indicate the player's remaining life, the number of lives left will be shown only when it is relevant, that is, once the player is about to enter combat, or in an area where we know it will be dangerous for the player.

For additional information such as the player's damage, speed or other possible statistics that will be added in the future, they will be visible in the pause menu, which will open with a button assigned to this menu. In addition, from this same menu, the player will be able to return to the main menu.

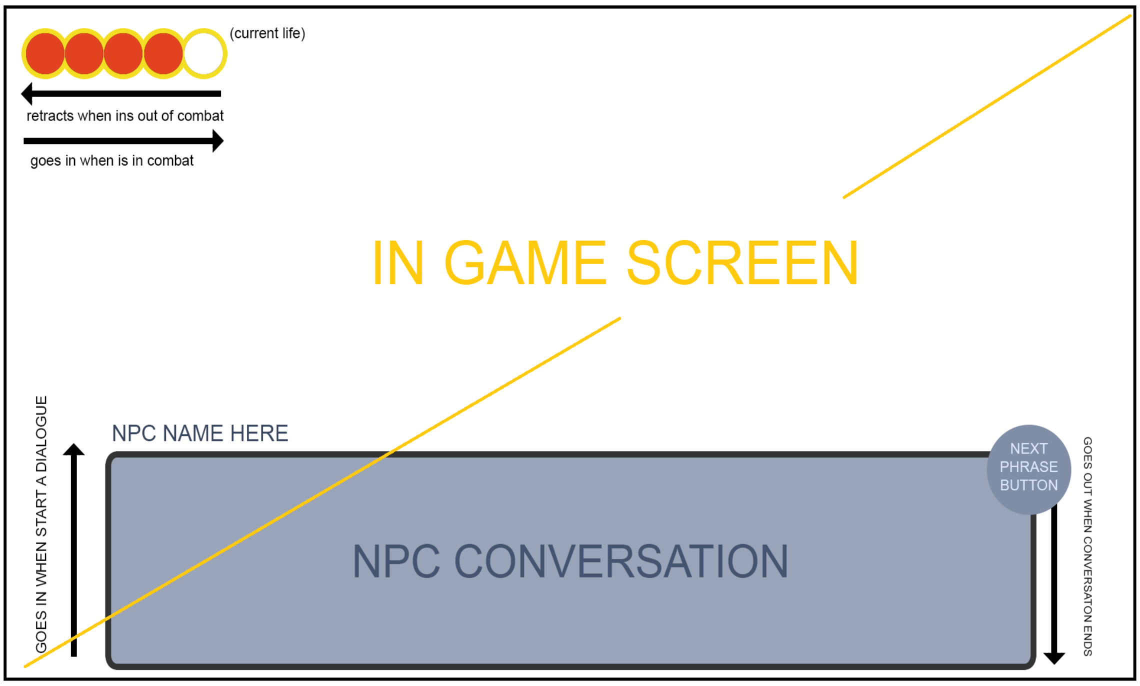

(current life)
retracts when ins out of combat
goes in when is in combat
IN GAME SCREEN
GOES IN WHEN START A DIALOGUE
NPC NAME HERE
NEXT PHRASE BUTTON
NPC CONVERSATION
GOES OUT WHEN CONVERSATON ENDS

Figure 3.2: Design of the inGame UI (made with Photoshop) [8]

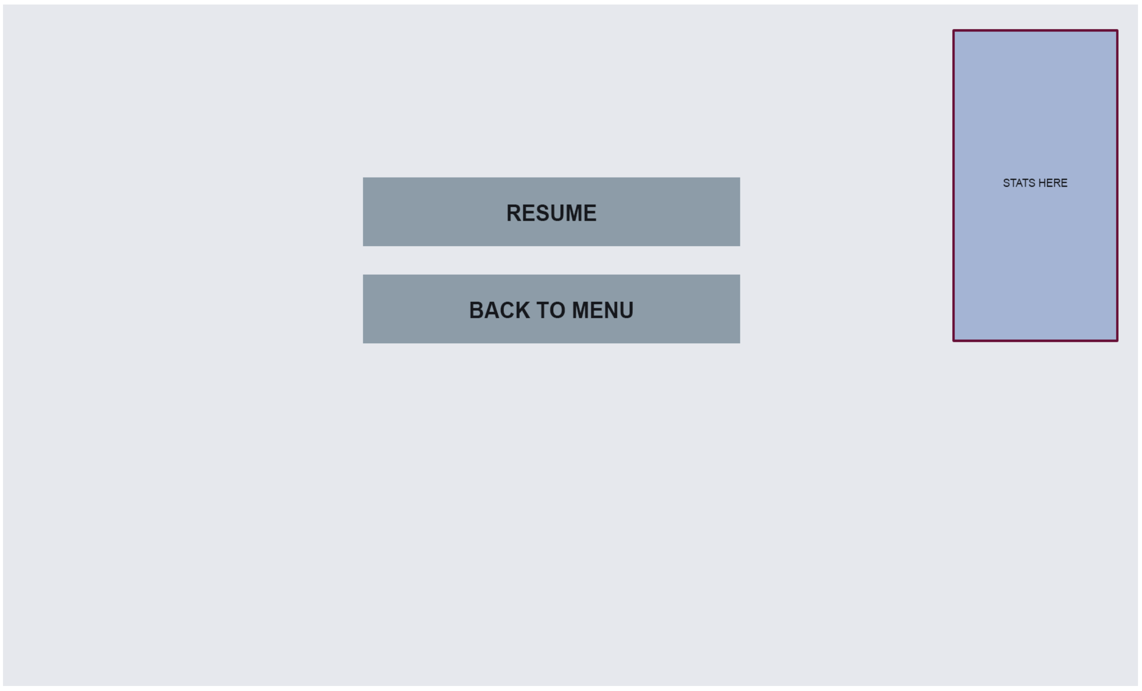


Figure 3.3: Design of the Pause menu (made with Photoshop) [8]

CHAPTER 4

Work development and results

Índice

4.1 Work development

This section will provide a detailed description of the work carried out chronologically. From the beginning until the moment this document is being written.

4.1.1 Design phase

As intended in the planning phase, the project was started from pen and paper, conceptualizing a design that can appeal to the player. What the design phase involves is thinking about the most important aspects you want the game to have, knowing exactly where the game is going to start and how far the game is going to go.

In this project, we knew that we wanted to make a short game, with the most fundamental systems but at the same time we wanted to see the potential of a future game. So it was concluded, through the design phase, that a main character in humanoid form would be the link that would unite the player with the virtual world. In addition, it was concluded that the game had to have a fluid, responsive and satisfying control, both in jumping and with movement. Then, through certain references mentioned and analyzed in the GDD, that a stealth mechanic was going to be implemented, since it was known that the target publisher was sympathetic to this type of mechanics. Then the map where the action would take place was designed. Giving rise to 4 or 5 zones in which the different elements would be gradually introduced. Then, the types of enemies that would appear in the game were designed, resulting in two basic enemies, an enemy that patrols horizontally, and another enemy that flies, circling randomly in an area. Finally, a series of interactive elements such as NPCs, death zones, healing elements, walls and breakable elements were devised along with the story that will give meaning to the world.

4.1.2 First technical phase

Once the design phase is over, the technical phase of development begins, so, we started with the creation of the repository, where the version has been updated so as not to lose the progress.

Once the repository was created in GitHub, we started creating the basic mechanics of movement and jumping, this mechanics was created simply using the physics engine provided by Uinity, resulting in the PlayerMovement.cs script, which will collect all the features related to the movement of the character.

Next, the development of the basis of what would be an enemy and its interaction is continued, which implies the CombatUnit.cs script, although not much more progress was made in this aspect.

The development of the interacting elements continued, giving rise to the InteractuableBase.cs script.

It should be noted that these systems were developed very quickly without much depth or attention to detail, they were the first steps in the project and really aimed to have moving elements on the screen and to be able to play something.

4.1.3 First Artistic Phase

Once some of the most fundamental systems were functional, it was time to make some assets so that the game could begin to be seen in its earliest phase.

We started with the modeling of the main character, which was modeled from scratch using Blender 2.8, a humanoid with proportions that were intended to be cartoon style. The quality of the modeling did not matter too much, because, as in the previous section, the intention was to be able to implement some art, although later it would be retouched and improved, in order to outline certain details of the development.

Once the modeling was finished, the basic idle, running and jumping animations were elaborated, also using Blender.

In addition, an attempt was made to model some scenery elements such as stones or pillars, but after a couple of failed attempts due to the lack of resemblance to the target style in the result obtained, these assets were discarded.

4.1.4 Second technical phase

Once the model and animations of the main character were available, we proceeded to polish and complement the scripts developed in the first technical phase, that is, to implement the animations and add other more secondary systems such as footsteps.

When the animations were already implemented, testing the game in a small improvised scenario, we could see that the result of the movement, which was given so much importance in the design of the game, was not exactly as intended, since it was neither responsive nor fluid, and the balloon effect appeared. So the next step would be to review and redesign the script in charge of the movement to make it work as we wanted.

First, it was decided to do without Unity's physics engine, because the inertias that this engine generates made the system respond slowly to the player's input. So a simple movement system was programmed that was intended to simply increase or decrease the position value according to the input that was pressed. This should work, but for the combat system and for the footsteps system we made use of Unity's collision system, and this in turn is closely related to the physics system, so if you make use of these collision systems, you can not modify the position of a transform that has a Rigidbody and a collider in the same object, so we had to discard this first implementation.

So, we had to make use of the physics system for the movement of the character, if we didn't want to implement our own collision system as well. The decision was made not to go that far, discarding the option of developing a new collision system.

The final decision was to make use of Unity's physics system, but limiting it and making use of certain tricks so that the accumulation of inertia would not be a problem.

For the jump, in the ascent phase, it was intended that the speed on the vertical axis did not draw a parabola, because we wanted the ascent speed to be continuous, that would make it much easier for the player to predict how far his character is going to ascend. So instead of using an impulse force when pressing the input, a constant force was used while holding down the jump button, in each frame a small vertical force would be applied until the input was released or until the jump time was over. For the descent phase, as for the ascent phase, we wanted a constant speed, that would make it much easier for the player to predict where his character is going to fall. For that, once the character has reached the maximum point of his jump (isGrounded = false jumping = true velocity.y == 0), in that instant the speed of fall is assigned to the desired value, and it is checked in each frame that this value is not lower in any case until touching the ground. If so, the value would be reset again.

For the horizontal movement, use is also made of the constant forces, thus accumulating the inertia that at first gave problems. Now a solution had to be found. The solution found was as follows: When changing direction, not only was going to change the direction of the velocity applied to the movement (addForce(-v)), but to the change of speed, would be added change of direction, the current speed at which the character goes, but in the opposite direction (addForce(-v + currentVelocity.x). This solved the problem of inertia and turned out to be a good solution for both controller and keyboard. Resulting in the movement system that was sought at first.

4.1.5 Second artistic phase

With the movement system already satisfactory, it was time to polish the artwork, improving the character modeling and animations. It was also time to start modeling some assets to start generating a more immersive scenario.

Once in Blender, the current animations are improved and the attack animation is added.

For a second time, we tried to generate a series of assets suitable to recreate the designed levels and also to create an immersive environment in the scenario. However, given the lack of skill, the generated assets did not meet the minimum quality standards expected. So those assets were discarded and the option of looking for free and freely available assets was considered.

Finally, after a lot of searching, we took advantage of an offer to buy a pack of assets for the creation of scenarios.

After acquiring the pack, a world building phase began, shaping the world in which the game would be played, and starting to make a final version of the scenarios updated in the design phase.

4.1.6 Last phase of development before the paradigm change

In this last phase of development, base patrol enemies were implemented using placeholder assets and added to the world design, also a round of bugfixing was done, solving some of the minor technical issues that arose. Finally, we added detail to the world created using the purchased assets pack, added dynamism to the camera using Unity's cinemachine tool and generated lighting to the scenery, giving a very finished look to the game.

4.1.7 Change of paradigm

At this point, after having tried to contact the influencer proposed in the technical proposal but without success. The decision was made to aim a little lower and try to contact some smaller, but still influential streamers/influencers with interesting numbers.

So, taking advantage of my position as an active developer and a situation of interest for the streamer el Yuste with 88mil followers on his twitter @inyustificado [14], 64mil followers on the twitter of his project @Esportmaniacos, 42mil followers on his Youtube channel, and an average of 5mil live viewers every day on Twitch, an agreement was reached to implement a mechanic that he wanted in his video game, and from there the possibility of further developing the video game would be assessed.

In addition, on the other hand, it was possible to contact a content creation club called Inevitables with 110,000 followers on their Tiktok account @inevitablesmk [5] whose videos reach, in some cases, 700,000 and 800,000 reproductions.

With this party, they are already negotiating the conditions for further development, contributing illustrators, music producers and project managers and, above all, how the project will be monetized.

4.1.8 After the changes

After the events concerning the publishing and its subsequent negotiations, it was assumed that one of the possible future publishers required to add a specific mechanic and, on the other hand, the other publisher is already prepared to start including varied content, both illustration and sound, a decision had to be made.

The decision to be taken, at least for the moment, is to keep the main branch for the Yuste variant. While for the Inevitables variant a new branch will be created from the current state of the game and, from there, adapt the project to what Inevitables wants to do, both with the story and with the direction of the project.

These events result in a shortening of the previously planned project, because given the publishers' proposals, the decisions to be taken will be different from what has been planned so far.

4.2 Results

After the development process of the final degree project, most of the objectives set at the beginning of it have been achieved.

In the first place, the aim of this project was to capture the attention of a publisher who, given the right conditions, could generate a business opportunity when the time came. In this case, although we have failed in the attempt to capture the attention of the targeted publisher, we have managed to contact two other potential publishers, of which one seems motivated to start working as soon as possible, and another one is open to be convinced.

In addition, we have managed to develop a project of sufficient quality to be able to capture the attention of these publishers.

However, it is true that the intention was to develop a short vertical slice, but, even so, it was expected that the length and the amount of content included in the project would be a little more extensive.

Finally, it was intended to generate, from this project, a series of assets, scripts, prefabs and development tools that could be extrapolated to allow, in the future, to develop our own videogames in a more agile way. Regarding this point, as in the previous point, we have achieved a library of elements such as combat scripts, enemy prefabs, character movement scripts, footsteps and others. Even so, it is true that it was expected to generate a larger library according to the expectations of the extension of the game.

Chapter 5

Conclusions and future work

Índice

5.1 Conclusions

Once finished the development process of the final degree project, the experience of developing the final degree project over these months has made me a better developer and has allowed me to draw a number of conclusions which I will express below.

First of all, although I already had certain notions about how complicated it is to plan in terms of time and amount of content a multidisciplinary project such as the development of a video game, the experience with this project has reaffirmed it. That leads me to think that only a team of flexible and experienced developers are able to adapt to the multiple unforeseen events that these projects are subject to in many cases.

Another lesson that I have reinforced with this development, is the complete admiration I have for developers who are able to bring out a game with only one person. For me, it is so difficult to do a decent job in so many different fields and disciplines, that I can only admire those who manage to do it.

Finally, in this project, it was important for me to prove to myself that I could do what I had set out to do, to prove to myself that I was good enough at it to know how to develop the fundamental mechanics at a level that is not the minimum, giving a leap of quality in terms of functionality. Now that it's finished, I must say that I expected the process to be simpler, I didn't expect to get stuck and encounter problems in something that was supposed to be as simple as movement. That made me doubt myself. Nevertheless, I managed to overcome the situation and get the expected results, even if it took longer than expected.

5.2 Future work

This final degree project was designed from the beginning to provide opportunities to continue working on this project. Given the results, everything indicates that it will be possible to continue developing projects based on the project that has been developed in this final degree project.

On the one hand, the project will be expanded, adding a specific mechanic in order to convince one of the publishers that have been contacted, in the hope of continuing to convince the publisher and continue working with this publisher.

On the other hand, regarding the Inevitables club, we are already negotiating the conditions under which we will continue with the project, adding illustrators, writers, music producers and other profiles to the project.

Bibliography

[1] Blender Foundation. (s. f.). Download. Blender.Org. Last visited: May 11, 2022, from https://www.blender.org/download/

[2] Diagram Software and Flowchart Maker. (s. f.). Diagrams dot net. Last visited: May 11, 2022, from https://www.diagrams.net/

[3] Fulls de càlcul de Google: creeu i editeu fulls de càlcul en línia de franc. (s. f.). Fulls de càlcul de Google. Last visited: May 11, 2022, from https://www.google.com/intl/ca_es/sheets/about/

[4] Get - Download Archive. (s. f.). Unity. Last visited: May 11, 2022 https://unity3d.com/es/get-unity/download/archive

[5] I. (s. f.). Inevitables (@inevitablesmk) TikTok | Watch Inevitables's Newest TikTok Videos. TikTok. Last visited: May 11, 2022 https://www.tiktok.com/@inevitablesmk

[6] Microsoft. (s. f.). Visual Studio 2022 | Descargar gratis. Visual Studio.Last visited: May 11, 2022, from https://visualstudio.microsoft.com/es/vs/

[7] Notion Desktop App for Mac Windows. (s. f.). Notion. Last visited: May 11, 2022, from https://www.notion.so/desktop

[8] Official Adobe Photoshop | Photo and design software. (s. f.). Official Adobe Photoshop | Photo and Design Software. Last visited: May 11, 2022, from https://www.adobe.com/products/photoshop.html

[9] Stylized Ancient Ruins Environment | 3D Environments. (2022, 12 marzo). Unity Asset Store. Last visited: May 11, 2022, from https://assetstore.unity.com/packages/3d/environments/stylized-ancient-ruins-environment-195760

[10] T. (2021, 29 diciembre). GitHub - Taspaya. GitHub.Last visited: May 11, 2022, from https://github.com/Taspaya/TFG_Project

[11] Technologies, U. (s. f.). Cinemachine. Unity. Last visited: May 11, 2022, from https://unity.com/es/unity/features/editor/art-and-design/cinemachine

[12] Templates - Journals, CVs, Presentations, Reports and More. (s. f.). Overleaf, Online LaTeX Editor.Last visited: May 11, 2022, from `https://www.overleaf.com/latex/templates`

[13] Unity Asset Store - The Best Assets for Game Making. (s. f.). Unity Asset Store. Last visited: May 11, 2022, from `https://assetstore.unity.com/`

[14] Yuste (@inyustificado) | Twitter. (s. f.). Twitter. Last visited: May 11, 2022, from `https://twitter.com/inyustificado`

Appendix A
Source Code

BaseEnemy.Cs

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

public class Cs_BaseEnemy : Cs_CombatUnit
{
    public override void Attack()
    {
        throw new System.NotImplementedException();
    }
    // Start is called before the first frame update
    void Start()
    {
        maxLife = 2;
        currentLife = maxLife;
    }
}
```

CombatUnit.cs

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

public abstract class Cs_CombatUnit : MonoBehaviour
{
    public int maxLife { get; set; }
    public int currentLife { get; set; }
    public int strength { get; set; }
    public Vector3 direction { get; set; }
    public bool canMove { get; set; }


    [Header(" ======= COMBAT =========")]
    [System.NonSerialized]
    public bool isStunned;

    public int currentDamage = 1;

    private void Awake()
    {
        currentLife = maxLife;
    }

    public abstract void Attack();
    public void RecieveDamage(int n) {
        Debug.Log(gameObject.name + ": Ouch");
        currentLife -= n;
        DeathChecker();
    }

    public void Init_BasePatrol()
    {
        maxLife = 1;
        currentLife = 1;
        strength = 1;
        canMove = true;
    }
    void  DeathChecker()
    {
        if (currentLife <= 0) Destroy(gameObject);
    }
    public void Heal(int n)
    {
        currentLife += n;
    }

    public void DealDamage(Cs_CombatUnit other)
    {
        other.RecieveDamage(currentDamage);
    }
}
```

InteractableBase.cs

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

```

```
public abstract class InteractableBase : MonoBehaviour
{

    public bool compareTag = false;

    [SerializeField]
    string tagToCompare = "";

   [SerializeField]
   public UnityEvent customEvent;

    public virtual void ExecuteAction()
    {
        customEvent.Invoke();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (compareTag)
        {
            if (other.tag == tagToCompare) ExecuteAction();
        }
        else ExecuteAction();
    }
}
```

PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour
{
    Animator myAnimator;
    Rigidbody myRb;
    float horizontal;
    bool isGrounded = true;



    //Gravity implementation
    [Header(" ======= Gravity Settings =========")]
    const float FALL_SPEED = -10;

    //Jumping variables
    bool isJumping = false;
    float initialJumpVelocity;
    float maxJumpHeight = 1;
    float currentJumpTime;
    float jumpTimeCounter;

    // ======= Run Settings =========")
    float speed = 2;
    float currentSpeed;


    [SerializeField]
    Transform feetPos;

    float jumpTime;
    [SerializeField]
    float checkRadius = 1;
    [SerializeField]
    LayerMask groundMask;
    [SerializeField]
    float jumpForce = 3;

    [Header(" ======= Other Settings =========")]
    [SerializeField]

    [Tooltip("Used to flip the mesh")]
    GameObject playerMesh;
    private bool jumping;

    private void Awake()
    {
        myAnimator = playerMesh.GetComponent<Animator>();
        myRb = GetComponent<Rigidbody>();
        speed = PlayerController.Instance.speed;
        jumpTime = PlayerController.Instance.jumpTime;
        currentSpeed = speed;
        currentJumpTime = jumpTime;
    }

    // Update is called once per frame
    void Update()
    {
        isGrounded = Physics.OverlapSphere(feetPos.position, checkRadius, groundMask).Length > 0 || (myRb.velo

        FlipPlayer();
        ManageAnimations();
        ManageWallJump();
        jumping = Input.GetButton("Jump");

        if (isGrounded && Input.GetButtonDown("Jump"))
        {
            isJumping = true;
            myRb.AddForce(transform.up * jumpForce, ForceMode.VelocityChange);
        }
        else if (isGrounded && !Input.GetButtonDown("Jump")) isJumping = false;

        if(!isGrounded && Input.GetButton("Jump") && currentJumpTime > 0)
        {
            myRb.AddForce(transform.up * 0.5f, ForceMode.VelocityChange);
        }

        if (isGrounded && currentJumpTime <= 0) currentJumpTime = jumpTime;
        HandleGravity();
    }

    private void FixedUpdate()
    {
        if (!isGrounded && Input.GetButton("Jump") && currentJumpTime > 0)
        {
            currentJumpTime -= 0.1f;
        }

            ManagePlayerMovement();
        if (!isGrounded && myRb.velocity.y <= 0 &&
            !PlayerController.Instance.GetIsLeftLimited() &&
            !PlayerController.Instance.GetIsRightLimited())

            myRb.AddForce(-transform.up * (jumpForce * 0.1f ), ForceMode.Acceleration);
    }


    void ManageWallJump()
    {
        if(PlayerController.Instance.GetIsLeftLimited() && !isGrounded && Input.GetButtonDown("Jump"))
        {
            Vector3 walljumpForce = new Vector3(jumpForce, -myRb.velocity.y + jumpForce * 2, 0);
            myRb.AddForce(walljumpForce, ForceMode.Impulse);
            FlipPlayer();
        }
    }

    void ManagePlayerMovement()
    {
        if (!isGrounded) currentSpeed = speed / 2;
        else currentSpeed = speed;

        if (PlayerController.Instance.GetCanWalk()) horizontal = Input.GetAxis("Horizontal");
        else
        {
            horizontal = 0;
            myRb.velocity = new Vector3(0, myRb.velocity.y, 0);
        }

        if (horizontal < 0.2f && horizontal > -0.2f) myRb.velocity = new Vector3(0, myRb.velocity.y, 0);

        if (!PlayerController.Instance.GetIsLeftLimited() && !PlayerController.Instance.GetIsRightLimited())
            MovePlayer();
        else if (PlayerController.Instance.GetIsLeftLimited() && horizontal > 0)
            MovePlayer();
        else if (PlayerController.Instance.GetIsRightLimited() && horizontal < 0)
            MovePlayer();
        //myRb.velocity = myRb.velocity + new Vector3(horizontal * currentSpeed, 0, 0);

    }

    void MovePlayer()
    {
        float factor = 10;
        //Going Left
        if(myRb.velocity.x < 0)
        {
            //Direction change
            if(horizontal > 0) {
                factor = 10;
            }
            //Same direction
            else if(horizontal < 0) {
                factor = 1f;
            }
        }
        //Going Right
        else if(myRb.velocity.x > 0)
        {
            //Direction change
            if (horizontal < 0)
            {
                factor = 10;
            }
            //Same direction
            else if (horizontal > 0)
            {
                factor = 1f;
            }
        }

        myRb.AddForce(new Vector3(horizontal * currentSpeed * factor, 0, 0), ForceMode.Impulse);

    }

    void FlipPlayer()
    {
        if (PlayerController.Instance.GetCanWalk())
        {
            if (horizontal > 0 && transform.rotation.y != 90) playerMesh.transform.rotation = Quaternion.Euler
            else if (horizontal < 0 && transform.rotation.y != -90) playerMesh.transform.rotation = Quaternion
        }
    }

    void ManageAnimations()
    {
        myAnimator.SetBool("isRunning", ((horizontal < -0.5f || horizontal > 0.5f || horizontal != 0)));
        myAnimator.SetBool("isGrounded", isGrounded);
    }

    void HandleGravity()
    {
        if (!isGrounded && (myRb.velocity.y > FALL_SPEED && myRb.velocity.y < 0)) myRb.AddForce(-Vector3.up, F

        if (Input.GetButtonUp("Jump"))
        {
            myRb.AddForce(new Vector3(0, myRb.velocity.y,0));
        }
    }

}
```

UIManager.cs

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

public class UI_Manager : MonoBehaviour
{
    [SerializeField]
    GameObject dialogueCanvas;
    [SerializeField]
    GameObject objectiveCanvas;
    [SerializeField]
    Text currentDialogueText;

    [System.NonSerialized]
    public SimpleDialogue currentSimpleDialogue;
    public void ShowUIDialogue()
    {
        dialogueCanvas.SetActive(true);
    }
    public void HideUIDialogue()
    {
        dialogueCanvas.SetActive(false);
    }
    public void WriteDialogue(string text)
    {
        currentDialogueText.text = text;
    }

    private void Update()
    {
        if (GameManager.Instance.GetCurrentGameState() == GameManager.GameState.Dialogue)
            if (Input.GetButtonDown("Attack")) currentSimpleDialogue.NextDialogue();
    }

}
```