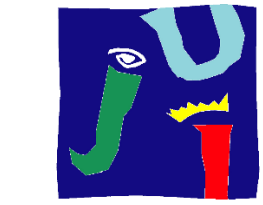
Videogame design and development degree

Final degree project's technical report



The ultimate story of a cube

Design and development of a narrative game with an immersive environment

Author: Raúl Cuadrado Alonso

Tutor: Miguel Chover Sellés

SUMMARY

This document collects the work realized to develop a narrative game with an immersive environment. In the game, the player controls a Cube through a side-scrolling movement and he has to talk with NPCs, interact with different items and solve puzzles.

The development of the project is divided in different blocks, such as modelling 3D objects and characters (with realistic and unrealistic shapes), programming the core of the gameplay as well as a dialog system suitable for different situations and the design of the story and the puzzles.

INDEX

TABLE OF FIGURES

1 INTRODUCTION

1.1 Objectives

The main objective of this project is the development of a game with an interesting story and smart and funny puzzles to solve, everything inside an immersive environment. The game relies on a good narrative design, with clever and humorous dialogues that, in this case, parodies the structure and style of the stories of conventional games. Puzzles are also important in terms of narrative (Koster, 2013) because they are connected to the story through an omniscient narrator that reacts to what the player is doing while he is solving a puzzle, sometimes even talking with Cube (the main character) explicitly.

To achieve this, a list of sub-objectives have been elaborated:

- Design of a good story with a lot of humor.
- Design of different puzzles that also have humor but that can also challenge the player without being excessively difficult to not slow down the flow of the story.
- Implementation of a dialogue system that allows us to show the text in different ways for various type of interactions, with auto-scalable dialogue boxes where the text displayed fits properly and that eases the process of localization of the game to different languages.
- Modelling of stages that give a feeling of mistery and suspense, mixing realistic looking models with basic shapes.
- Design of sound effects and composition of an original soundtrack that adds immersion to the experience.

1.2 Justification

Some years ago, in a talk about narrative in games, I asked to the speaker about the lack of matureness in games because, I said, they did not use to talk about important or controversial topics like politics or social issues, as literature or films do. He agreed with me and, minutes later, we both reached a new interesting conclusion: as well as games can be considered immature for not talking about 'serious' subjects, they can also be considered immature for not having humor. There are some good examples of funny games like *Portal* or *The Secret of Monkey Island*, but since the *boom of videogames* (2007-2008 approximately) until now, the trend has been to try to mainly write more serious plots with the intention of dignifying, somehow, the narrative side of games, also as a result of the own process of maturation that developers who started creating games 10 or 15 years ago (or even before) have experienced. Nevertheless, still nowadays people think that stories in games are irrelevant, in the sense of being only a mere excuse to keep killing the bad guys or doing whatever the games tells

you to do (in terms of mechanics). Of course there are good examples of 'serious games', like *Papers, please* or *This war of mine* (both indies), but in my opinion there is still much left to do to change the perception of games when we talk about narrative.

In this context, my way of trying to contribute with a new game is by parodying conventional games that take very seriously to themselves and end up being funny when their intention was the opposite. In general, that is why I decided to develop a game that creates apparently serious situations but what are actually full of humor (in purpose).

1.3 Initial planning

19 March - 25 March	**26 March - 1 April**
Documentation about existing dialog systems in Unity, different types of files to store text like JSON or XML and good practices to prepare a game to be localized. Narrative design of the game: main arcs, number of chapters and characters involved.	Modeling of the chapter 1 'realistic' environment, first tests for the soundtrack. Writing of the memory.
2 April - 8 April	**9 April - 15 April**
Implementation and testing of the custom dialog system. Writing of the memory.	Design and implementation of puzzles for the chapter 1. Development of the movement and physics of the player and his interaction with the world through the dialog system. Writing of the memory.
16 April - 22 April	**23 April - 29 April**
Production of the soundtrack and implementation of dialogs for the chapter 1. Testing of the dialogue system for the interaction with items. Writing of the memory.	Design and implementation of a Variable Camera System and implementation of conversations with NPCs using the dialogue system. Final scripting and creation of timelines for the chapter 1.
30 April - 6 May	**7 May - 13 May**
Light treatment and implementation of shaders for the chapter 1. Modeling realistic objects for the chapter 2. Writing of the memory.	Modeling, design and implementation of the environment and puzzles for the chapter 2. Writing of the memory.
14 May - 20 May	**21 May - 27 May**
Finishing the draft of the memory. Light treatment for the chapter 2. Implementation of dialogs for the chapter 2 and scripted moments.	Modeling, design and implementation of the environment and puzzles for the chapter 3. Soundtrack for the chapter 2.
28 May - 3 June	
Light treatment, dialogues and scripted moments for chapter 3 and writing of the final memory.	

The time needed for each type of task was also estimated:

25h	Documentation about existing dialog systems, particle systems, shaders and light treatment in Unity.
100h	Programming the core of the gameplay.
100h	Modelling the environment and the characters.
25h	Producing the soundtrack and sound effects.
20h	Designing puzzles and dialogs.
30h	Writing of the memory.

2 GAME DESIGN

The Game Design Document of the game is made up of the game, narrative and art design, but they have been separated in three big points for clarity. To write this document, some ideas have been taken from the GDD of Grim Fandango, which uses some interesting structures to explain the story of the game and an enjoyable style of writting, even telling some jokes along the document to make a more pleasant reading, which can be a nice strategy if your objective is to get the reader to reach the end of the document.

2.1 Overview

2.1.1 GAME CONCEPT

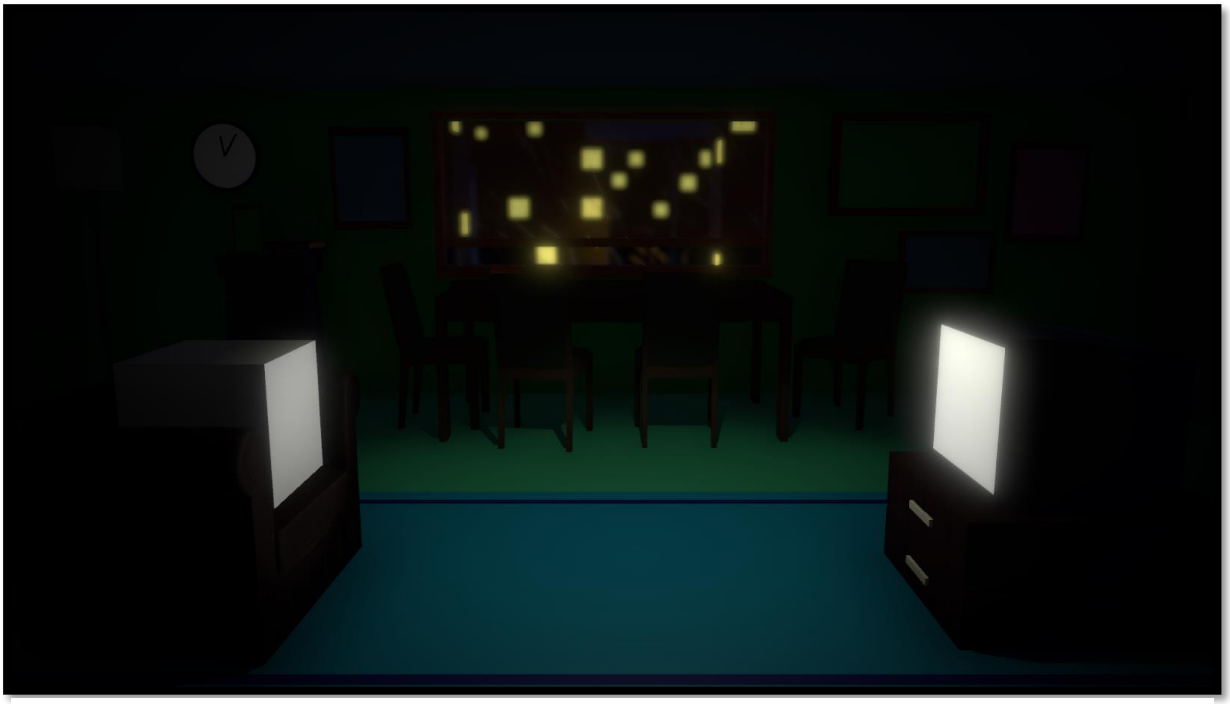


Fig. 1 – The introduction of the game

The Ultimate Story of a Cube (TUSC) is a 2.5D narrative game. It tells the adventures of Cube, a guy who lives in Cube Town and wants to leave the city with Penélope, his girlfriend, to run away from the stress and routine of their lives. But, one day, someone knocks the door of Cube's apartment (which can be seen in figure 1) and, suddenly, he is not in Cube Town anymore. It is a story-driven game and in terms of gameplay it is focused on talking with other characters and resolving puzzles by pressing buttons that perform different (and sometimes random) actions like disabling the collision detection of a wall so Cube can pass through it - the game constantly breaks the fourth wall to remind the player that even the game itself is not taking the story seriously.

2.1.2 GENRE AND TARGET AUDIENCE

It is mainly a narrative game, with a lot of conversations with different Non-playable-characters (NPCs), an omniscient narrator that appears in specific moments of the game (while the player has the control) and also visual elements of the environment that give to the player additional information of the story. Furthermore, it can be also considered a puzzle game, because the player will need to resolve puzzles to move the story forward.

The humor of the game and the simplicity of its dialogues allows it to have a wide target audience, but the jokes about narrative itself and the concept of breaking the fourth wall will be more interesting to people between the ages 20-30 approximately.

2.1.3 LOOK AND FEEL

The game has a low-poly aesthetic with a mix of flat colors and realistic textures such as wood or concrete for some objects. All the characters of the game will be primitive objects like spheres and different kind of prisms, but the environment will have more complex objects. For example, the apartment where Cube lives has 'realistic looking' furnitures in order to create a funny contrast in the game.

Talking about lighting, all the levels are mainly dark, it could be said that the game transmits a gloomy feeling in terms of visual input, but it is not an objective to make the player feel uncomfortable, but only make an interesting and intriguing world. This balance is achieved with the humor of the story.

2.2 Gameplay and mechanics

2.2.1 GAMEPLAY OVERVIEW

Taking into account that it is a 2.5D game, it mixes 3D models and side-scrolling movement. During the game, as explained before, the character moves through the different levels talking to NPCs, interacting with some items of the stage and resolving different types of puzzles that will be described later.

2.2.2 GAME PROGRESSION

TUSC is a narrative-driven game, so its progression is linear. The character cannot die so there is no need of any kind of respawn system. The game is only divided into chapters (as part of the narrative design) and the player cannot take decisions to modify the story. Also, there is only one way to solve puzzles so in order to proceed the player just needs to move the character in the correct direction (right or left, depends on the level). The stages are separated into different files only for optimization issues.

2.2.3 MISSION/CHALLENGE STRUCTURE

Given its linear nature, there is not more than one mission at a time, if missions are understood as tasks like ‘solve this puzzle’ or ‘talk to this NPC’. The next mission to complete is enabled only when the immediately previous one is done. For example, if the player has to talk with an NPC, when he does it, a cutscene is triggered and Cube moves to another stage. There is not any type of scoring system, that is, the player cannot complete a mission better or worse.

2.2.4 OBJECTIVES

In terms of narrative, the main objective of the game is to escape from the ‘strange world’ (where Cube is stuck) to leave his hometown with Penélope, his girlfriend. Furthermore, it can be considered that in every chapter there are ‘micro-objectives’ like solving a puzzle, talking with an NPC, etc. The game never tells to the player explicitly what is the next thing to do, but is the narrative itself who does it.

2.2.5 MOVEMENT AND OTHER CONTROLS

The player moves the character using the left joystick of the controller, and the movement can be from left to right or vice versa. There is only one more button to use, B[1], which is used to start an interaction either with an item or an NPC. Talking about movement, the character can only be moved in a 2D space, but, in some way, it can be considered that vertical movement exists in some puzzles where the character can enter through a door that brings him to an upper place (that door can be used again to go down again).

2.2.6 PHYSICS AND OBJECTS

The character itself has physics as its movement it is controlled by force vectors. When a direction is indicated using the controller, a force is applied to the cube in the selected direction in an incremental way: the more the joystick is tilted, the more magnitude the force vector will have.

Some objects of the environment can collide with the character, such as other cubes or walls. The character cannot pick items (he has no inventory), hence there are two types of puzzles: those in which the character needs to physically interact with the items of the puzzle (like pushing a crate) and those in which the puzzle is solved pressing buttons. Both of them have physics since a trigger collider is used to detect if the character can press a button. The location of these triggers is used to vary where the character should be positioned to start an interaction. For example, interacting with an NPC means

[1]Considering the button mapping of the XBox controller.

talking to him, and the position of the character while the conversation is taking place can be defined using the position of the trigger.

2.2.7 ACTIONS

As explained before, button B is used to interact with everything. Interacting with an NPC means talking to him, and the NPC can tell something new or the last thing he said (if he have already said everything he had to). Conversations can have an impact in the game progression, that is, when they end something happens. For example, talking to the concierge in the chapter 1 enables a trigger that allows the player to open the door of the hall and leave the building. There are interactions with NPCs and items that do not affect to the progression, they just act like 'atrezzo', providing additional information of the story.

Interacting with an item can lead to different events. With some items, the character just says something about them. Other items can be 'modified', for example, the lamp of the living room of his apartment, which can be turned on or off. Finally, pressing a button is a little bit more complex, because each button behaves different. One button maybe does something when the player presses it, like opening a door, and then if the player presses it again, the button does the opposite. In this case, the button would close the door. Another case could be a button that needs to be pressed for a certain time. All different behaviors will be described in detail in the narrative design section.

2.2.8 SCREEN FLOW

The game is linear and the scenes are separated only due to performance issues. Camera cuts take place only when a chapter ends or when it is needed to switch from one scene to another. The camera is always at a fixed distance of the character and follows him in the horizontal plane, but its height and deepness can be modified by using special triggers that will be detailed in the technical aspects section.

2.2.9 GAME OPTIONS

The only option available inside the game will be the language (initially only spanish or english). Graphics, resolution and button mapping of the keyboard and other controllers can be modified in a previous window that appears when the game is executed.

Finally, unlike what games use to do, the player is not allowed to modify the sound of the game because it is an essential part of the experience as narrative or graphics are (Martínez, 2016).

2.3 Interface

The Head-Up-Display (HUD) of the game is only made up of the dialogue boxes (image and text) and an image of the B button that is showed up when the player can start an interaction with something. Taking into account that it is a narrative game and that an important objective is to create an immersive environment, the objective is to use all the possible elements from the diegetic universe to give information to the player, such as the sound effects, 3D models or dialogues.

2.3.1 MENUS

The main menu of the game will allow the player to do the following:

- **New game:** to start the game from the beginning.
- **Language**: pressing this button, the language is changed to another.
- **Exit**: close the game.

There is another in-game menu, while the game is running, that can be showed pressing *Escape* in the keyboard or *Start* in another controller. This menu just allows the player to resume the game or exit after saving the progress.

3 NARRATIVE DESIGN

In the justification of the project, in the point 1.2, the motivations of the style of the narrative have been partially explained. Apart from that, this section details how the story have been created and its structure. First, some games and films have been used as a reference.

The Stanley Parable (figure 2), made by Davey Wreden, is an important source of inspiration because of how the narrator of the game reacts to what the player is doing, sometimes even talking directly to him. It is a first-person-walker where the player can go through different rooms and when he enters in a room or interacts with an object, the narrator says something about that, even reacting if the player keeps waiting in a room, doing nothing. This design of the narrator is applied in our game in a similar but simpler way.

Fig. 2 - In-game footage of The Stanley Parable

Night in the woods (figure 3), by Infinite Fall, is a narrative game but what makes it interesting for this project is how its dialogues are displayed. The dialogue boxes of this game are resizable in function of the amount of text and the dialogue system itself has a feature that allows the user to introduce different emojis inside the dialogues that are not displayed, but used to trigger different animations in the character that is saying that dialogue. In our project this last feature is only used in a more simpler

Fig. 3 - In-game footage of Night In The Woods

way and it is described in the section 5 of this document, but the reference is explained here because it has narrative implications.

The last reference used is not a game. *Amanece que no es poco* (figure 4), a film directed by José Luis Cuerda, tells the story of a little village in Spain where a lot of weird things happen. This movie is one of the best sources of inspiration for surreal humor and how to constantly create moments where an apparently serious situation is filled with tons of random and funny things.

Fig. 4 - Shot from Amanece que no es poco

3.1 Main story and story flow

The Ultimate Story of a Cube is a narrative game where the main story doesn't matters. At the beginning is told that, one day, Cube discovered that he was not in Cubetown anymore. Outside of the building where he lives there were not streets, people and other buildings, and that was a problem because, as the game tells us, there was only one day left to leave the town with Penélope. That's everything the player knows about the story, the premise is already set and, after that, the game will not pay special attention to it. Cube's journey it is not important, what really matters are the different situations that take place during the adventure. This is a joke on its own, because the introduction generates an atmosphere of tension and suspense and then nobody in the game cares about it, even Cube.

After crossing the city, Cube falls inside a pipeline that leads to the City of Spheres, where he will meet Lemon, the doorman of a disco. At the beginning, Lemon seems like a rude guy who does not like to have friends. Cube just asks for help and Lemon treats him so bad, and then Cube talks with Lemon's boss and he gets fired. After that, Cube finds Lemon at the top of a hill, where they can see the whole city. Lemon apologizes for his behavior and he decides to help Cube to come back to Cubetown. While crossing the city, they will find some dificulties because, for some reason, spheres hate cubes. But Cube will be a hero for the city when after talking with the Public Debt of the city, which is another physical character. Public Debt is so sad because some economists have beat her up. Cube says to him 'hey, don't be sad', and she is not sad anymore.

After being a hero, he can leave the city and he finds a button that teleports him to Cubetown again. He say goodbye to Lemon and, when he presses the button, he appears again in his apartment, in the same scene than in the beginning. But, this time, when someone knocks the door, there is someone on the other side: his brother Manny. Cube tells him all his adventures and how he got to come back again to leave the city with Penélope. Rect asks to Cube, surprised, if he knows any Penélope. 'Not really', he answers, and after an uncomfortable silence, he says 'but it would have been cool' and the game ends, with a completely anti-climax ending.

As explained before, the intention is to parody those games that take very seriously to themselves but that have a very bad execution (in terms of narrative) and end up being funny. So, in order to achieve

this, the intention is to constantly create apparently serious situations and then make jokes about them.

3.2 Characters

In this section, the main characters of the game will be described. Their physical description is very simple since they are basic primitive objects, but they can be seen in the figures of Section 4.1.

Cube is the main character of the game. He is a silly guy that usually misunderstands everything he sees, reads or hears and cannot stop talking nonsense. He works as a step of a big stair in an important building. He does not use to show any kind of emotions and always keeps the same tone while talking. Is the hero of the game and the player plays as him through the entire game.

The concierge is a mysterious character that briefly appears at the beginning of the game. During a weird conversation with him, Cube realises that he is not in Cubetown anymore. In fact, he's not the concierge, but for some reason he talks as if he actually was and also Cube thinks so. He is so polite and looks like is constantly overexcited. This character is only used to set the premise of the story and he won't appear in the game anymore.

Lemon is a sphere that lives in the City of spheres. He works as a doorman at a disco, The disco of spheres. He acts like a tough man but he is actually a sensitive guy. Is the adventure partner of Cube and he will help him to come back to Cubetown. Lemon is the character with which Cube will talk more during the journey, and he is mentor and helper at the same time, because they both cooperate to overcome some situations but also he teaches Cube how to be a real hero. Unlike Cube, he is very explosive and is constantly showing his emotions. He also has a little touch of pimp, as a result of his childhood in the conflictive neighborhoods where he lived.

The omniscient narrator can be considered another character because he reacts to what Cube is doing and in some parts they even talk to each other. He is supposed to know everything but he makes mistakes and, in general, it could be said that he is not very professional.

3.3 About immersion

In the point 3.1 was explained that the main story is not important. The game is partly about talking with NPCs but, like the main story, characters are kind of irrelevant. And actually it could be said that the only thing that matters in this game is the humor. The lack of humor in other games is what makes this one to revolve around it all the time, leaving aside everything else. The player will never know too much about any character, including Cube. The game never talks about their background in deep, it only provides the exact information from them to make a particular joke. That is why, in fact, it would not be necessary to write a description of the characters, because their description is only made up of where they are and what they say in a particular moment. And this is one of the most important keys of the game, the simplicity. Because a lot of things are left to the imagination of the player, like why the characters are saying what they are saying, what are their motivations… and in a game of this kind

it is very useful because the player is not overwhelmed with unnecessary information, he just fills the gaps with the exact amount of things that he wants to imagine, no more no less.

It is important to understand the **difference between immersion and consistency**. It is not needed to create a universe with well defined characters and the relationships between them, or logical rules about how the entire world works in a 'credible way' and that kind of things that role-playing games usually do. Immersion is not about believing in that fictional world because of its solidity. Immersion is about **complicity**. This can be achieved by trusting on the player, not explaining to him everything of what he is seeing; or believing (as narrative designers) that, in fact, he is going to be able to understand a lot of things that are not completely described and that makes the universe more interesting, as said before. In general, not treating him as he was stupid.

So, finally, it can be said that in relation to the intention of being a parody of other games as explained before, this game is full of surreal humor in defense of the player as an intelligent receiver that can understand situations where there is an important lack of information (needed to make surrealistic jokes). Hence, to create an immersive universe, the different elements described before (sound effects, music, good visual aesthetic) are needed and they all are put together through the complicity achieved.

3.4 Chapters

In this section the plot of the game that is in the game will be described in detail, as well as all the puzzles that appear along it and the concept of immersion applied to this game. Dialogues have been removed from the explanation for clarity.

3.4.1 CHAPTER 1

The game starts with Cube watching the TV. He gets down from the armchair and the player takes the control. Cube can move around the apartment and he can interact with the fridge and the wall at the end of the kitchen. After interacting with at least one of that items, if the player comes back to the living room, someone will knock on the door. Now the main door of the apartment can be opened. On the other side of the door there is a note that Cube can interact with. At the end of the corridor, the player has to call to the elevator to go down.

On the hall of the building, Cube can talk with the concierge and, after a weird conversation, the elevator disappears (so Cube cannot go back) and the main door of the hall is now interactable, so it can be opened. When Cube leaves the building, the narrator starts to tell the premise of the story and the puzzles begin.

Puzzle #1: the first puzzle is so simple and it is used for explaining the basic mechanic of interacting with a button. There is a button and a wall that blocks the way. If Cube gets close to the button and presses it, the wall moves aside and the player can proceed.

The puzzles are separated and while Cube is moving towards the next one, the narrator tells something about the story.

Puzzle #2: the second puzzle seems to be equal to the first one, with a button and a wall. In this case, if the button is pressed, the collision detection of the wall is disabled so Cube can go through it.

Puzzle #3: again, the puzzle is similar but now there is also a button on the other side of the wall. When Cube arrives to this puzzle the camera is updated to a static position, leaving the wall in the center of the image (if Cube goes back, the camera is updated again and begins to follow him). To solve this puzzle, it is needed to press the button that is on Cube's side. After that, the camera won't follow him if he goes back, but he will be teleported to the other side. Now the player needs to press the button of that side to enable the camera to follow Cube and proceed. If not, Cube will be teleported again to the other side.

Between the third and the fourth puzzle there is a bridge and, in the middle of it, a button. It is not needed to interact with it but, if so, Cube will say that he is not sure if he wants to press it, but he will do it anyway. After pressing it, a gate under him is opened but he does not fall down. 'Dad used to say that colliders never betray you', he says.

Puzzle #4: a button and a wall. But, in this puzzle, it is not needed to do anything, because when Cube presses the button he waits for a second and, since nothing happens, he decides to surround the wall and proceed.

Finally, the camera updates its location to a new static position, leaving Cube on the left and a button on the right of the image. When Cube tries to get close to the button, another gate is opened under him and now he falls down. If the button of the bridge was pressed, the narrator will say that 'Dad was not right after all'. The screen turns black after a fade out and the chapter 1 ends.

3.4.2 CHAPTER 2

After the fall, Cube is inside of what seems like a pipeline.

Puzzle #1: a button and a metallic gate. Like in the first puzzle of the chapter 1, it is needed to press the button to proceed.

Puzzle #2: a metallic gate and a button (on the other side). The narrator says 'Cube pressed the button to open the metallic gate', but Cube answers him saying that 'there is no button on this side'. The narrator realises that he made a mistake and after a sound effect of 'typing on a keyboard' the button disappears and appears on the correct side, so the player can now proceed.

Puzzle #3: in this puzzle the camera is positioned on a static location to get a general view of the puzzle. The door that is needed to be opened needs electricity and Cube has to activate the generator that provides it. To achieve it, he needs to go to the generator and keep the button pressed for at least 3 seconds and then reach the button that opens the door before the energy is lost again.

After this puzzle, Cube leaves the pipeline and he discovers a whole new town, City of Spheres, where after a little walk a fence will block the way, and a text appears in the screen: 'Coming soon'.

4 ARTISTIC DESIGN

As explained in the point 2.1.3, the game will have a dark aesthetic and will mix realistic looking and basic 3D models. This decision has been made for two reasons: the first one is that the low-poly is a very good visual style and also a mark of identity of indie games, and the second one is the result of a plannification for the development of the game where it is necessary to be realistic with the limitations of time and skills.

Fig. 5 - In-game footage of Inside

A few references can be mentioned as a source of inspiration. The first one is *Inside* (figure 5), which is a narrative game where there are no words. All the story is told with the visual input that the player receives. It is very dark and creepy and that fits with the desired style.

Little Nightmares (figure 6) is another game with a dark aesthetic and the interesting thing is the use of some textures that have a realistic look and which creates a nice contrast between the cartoon-style design of its characters and the realism of its textures.

Fig. 6 - In-game footage of Little Nightmares

4.1 Characters

One of the main jokes of this game is the basic shape of the characters. They all are primitive objects, such as cubes or spheres, but they live in places where apparently they cannot (how does Cube turns on the TV?). This is a deliberated decision because, apart from being funny, the process of design, modeling and rigging of characters would have taken a lot of time

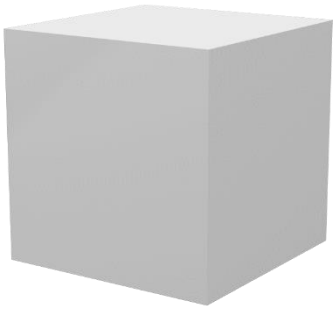


Fig. 7 - Cube

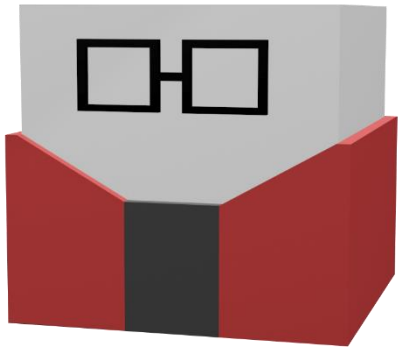


Fig. 8 - The concierge

that is needed to complete another tasks (such as programming or narrative design) and the environments are complex enough to gather the whole efforts designated to model.

Cube (figure 7) is the simplest character, because he is only a white cube. The concierge (figure 8) is a cube too but he wears glasses and a jacket. Some characters of the City of spheres, from the Chapter 2, wear hats or carry briefcases. They are not relevant because they are used as atrezzo for the game, the only important character from the Chapter 2 is a simple sphere (Lemon, remember?).

4.2 Game world

As said in the point 2.1.3, the world mixes simple 3D models and others with realistic-looking shapes and textures. In the chapter 1 this models are separated. Inside of the building where Cube lives, all of the models try to have a realistic looking (except Cube, obviously) but mixing flat colors and textures such as wood or tiles. But, after leaving the building, all that models are simple and with flat colors. After the chapter 2, the two types of models are mixed progressively in the scene, to incorporate puzzles in the realistic looking parts.

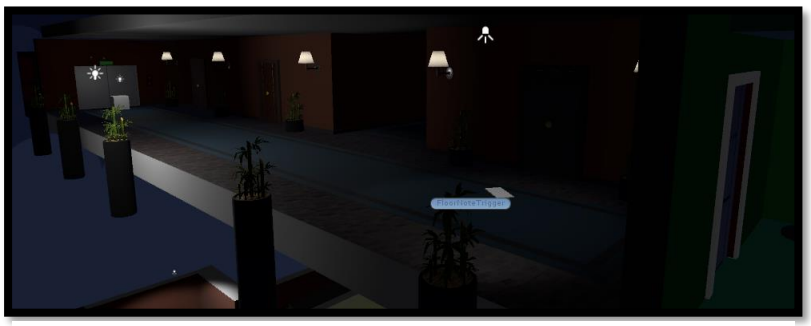


Fig. 9 - Buildings have a side without wall

None of the buildings are completely modeled, they all have a side free of walls because that space is used to put there the camera so the distance between the camera and the character can be freely updated without worrying about undesired obstacles that could block the view, as can be seen in figure 9. However there are objects between the camera and Cube to get a more immersive feeling, like plants while Cube walks through the corridor, or the TV in the apartment.

The building of the chapter 1 has been split in three big blocks to work better with them inside 3Ds Max (figure 10). The first one is the apartment, made up of two explorable rooms, the living room and the kitchen, the second one is the corridor and the elevator that brings Cube to the third block, the main hall of the building. Inside all of them there are some models downloaded from the Asset Store of Unity and others specifically modeled for the game. Since they are very basic, there is no need of an explanation of the process of creation.

Fig. 10 - The three blocks of the building

Table 1 shows all the models used for the Chapter 1 that have been downloaded from the Asset Store:

Table 1: Models downloaded for Chapter 1.

Chatper 1 – Aparment and kitchen
- Armchair (ref [1]). - Books (ref [2]). - Chairs and table (ref [3]). - Fridge (ref [4]). - Kitchen furnitures: broom, center table, stools (ref [4]). - Plates and glasses (ref [5]).
Chapter 1 – Corridor, elevator and hall
- Decorative plants (ref [6]). - Wall lamps (ref [7]).

Some models use realistic looking textures as explained before. The floor of the corridor and the hall uses a wood texture (ref [19]), the wall of the kitchen uses a texture of tiles (ref [19]) and the window of the hall uses a glass texture (ref [20]). The rest of the models have only flat colors. The mix can be seen in figure 12.

Fig. 12 - Mixing of realistic textures and flat colors

Finally, the second part of the Chapter 1, where the puzzles must be resolved, is very simple (figure 11). Walls are only white rectangular prisms and the buttons are made of a modified box that serves as the base, and a cylinider as the button.

The first part of the Chapter 2 takes place inside a pipeline (figure 13), which is just a cylinder that uses a texture of concrete (ref [19]). Walls here are replaced by metallic grids and buttons are embedded into the 'wall' of the pipeline. There is a big section at the end of this part, where the pipeline is much bigger. There are black square holes in the wall that Cube can use to move up and down between the different floors.

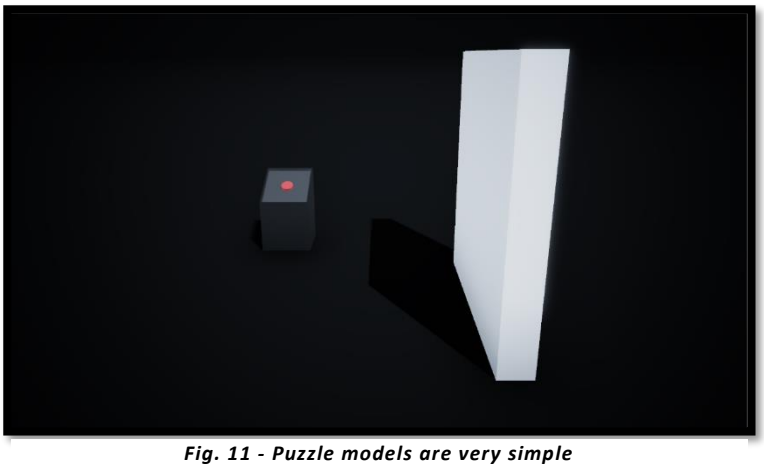


Fig. 11 - Puzzle models are very simple

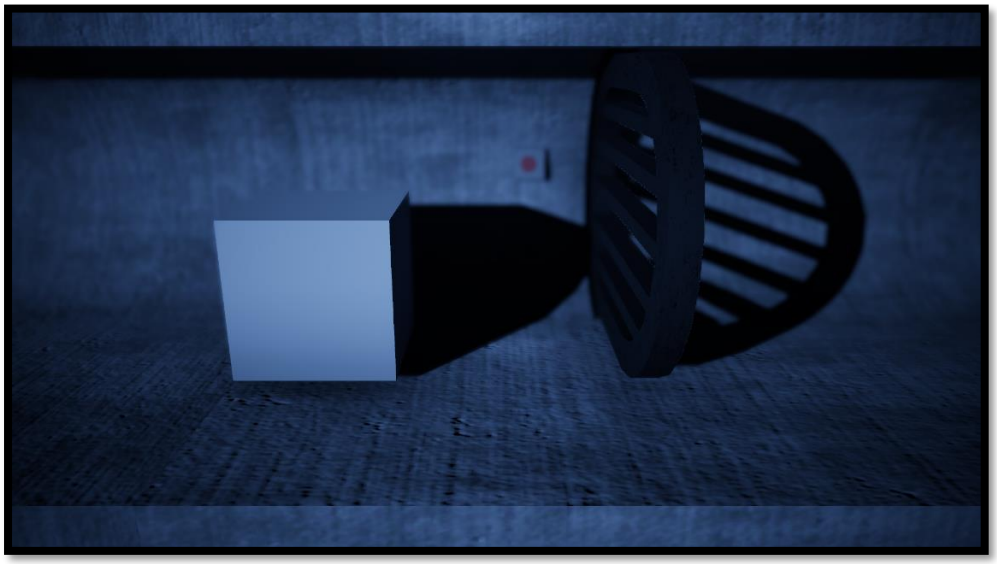


Fig. 13 - Cube inside the pipeline

Those holes are just black boxes that simulate darkness.

When Cube leaves the pipeline, a new big town is discovered, The City of Spheres, where the buildings in the background that have been downloaded from the Asset Store. The floor mixes a big modified box which is the road, and a terrain for which Cube moves that is made with the Terrain tool from Unity to put grass in it.

Table 2 shows the models used for the Chapter 1 that have been downloaded from the Asset Store:

Table 2: models downloaded for Chapter 2

Chapter 2 – City of spheres
- Buildings (ref [8]). - Streetlights (ref [8]). - Car 1 (ref[9]). - Car 2 (ref[10]). - Car 3 (ref[11]. - Fence (ref[12]).

4.3 Head-Up-Display

It have been already explained that the HUD of the game is very simple. It is just made up of the dialogues and the button image that appears when the player can interact with something.The dialo-

Fig. 14 - The HUD of the game

gue box is just a black image with rounded borders. It was exported in high resolution to avoid pixelation issues and taking into account that it can be resized. The font used for this game is *Averia Serif.* It can be seen in figure 14.

The image used to represent the button is a *PNG* of the B button of an Xbox Controller. It could be expanded and the image could be changed according to the controller used or if the player is using the keyboard.

4.4 Sound effects

The sound effects of the game have been implemented using *Wwise*, whose use and functioning will be explained in the section 5. They all have been modified to fit properly in the game. For it, a DAW (Digital Audio Workstation), *Cubase Elements 8,* has been used to apply them different effects like compression, distortion of equalization. Also, some sounds were combined to get a new sound. For example, the sound of the gate at the end of the Chapter 1 is made up of different sounds of a metallic door, and equalization and compression was also applied to them.

After the first modification of the sounds in Cubase, they are imported into Wwise to implement them into the game. There they can be also modified, but this time depends on the moment of the game. For example, when Cube is inside the pipeline, all the sounds that take place inside of it have an effect of reverb. The use of Wwise will be detailed in Section 5.

Table 3: Sound effects used

Chapter 1 sound effects
- Rain (ref [13]). - TV (ref [14]). - 'Knocking on the door (ref [15]). - Door opening (ref [16]). - Elevator doors opening and closing (ref [17]). - Elevator moving (ref [17]). - Hall main door opening (ref [16]). - Button pressed: created using a synthesizer. - Dark ambient (ref [18]). - Metallic gates opening (ref [19]).

Chapter 2 sound effects
- Metallic grid opening (ref [19]). - Dripping water (ref [20]). - Dark ambient (ref [18]). - Car engine (ref [21]). - Wind effect (ref [22]).

4.5 Soundtrack

The soundtrack of the game have been composed by me using different synths and VST (Virtual Studio Technology) instruments in Cubase. Jazz is the main style and the objective was to contribute to the dark feeling of the game with subtle melodies and gloomy armonies. To achieve this, the main instrument used is a VST that emulates electric pianos, which have a really warm sound that fits perfectly in the game. The tempo of the different songs has been choosed taking into account the moment where they will be played. For example, the song that sounds where someone knocks on the door is much faster than the song that was being played before that.

Since the production of music is very slow and the project had a very limited time, only three songs have been composed, recorded, mixed and mastered:

- ***Whispers in the night*:** the song that is played at the beginning of the game. It is a very minimalisitc and dark song that is meant to leave some space so the rain and the sound of the TV can also be heard.
- ***Who knocks:*** is a song that increases the tempo drastically to give a feeling of mystery and tension when the door is knocked. It is more complex and the bass adds the power needed to be the song that will accompany Cube through the rest of the introduction.
- ***A new place*:** is the song that is played when Cube arrives to The City of Spheres. It leaves behind the armonies used in the previous two songs to express different feelings: a new place to be discovered, the adventure begins...

5 FUNCTIONAL AND TECHNICAL SPECIFICATIONS

5.1 Tools

The game has been developed in a PC with the following specifications:

- OS Windows 10.
- Nvidia GeForce GTX 770.
- Intel i7 2600.
- 16 GB of RAM.

The tools used to develop the game are the following:

- Unity 2017.3.1f1.
- Visual Studio Community 2017.
- 3Ds Max 2017.
- Adobe Photoshop CC.
- Cubase Elements 8.
- Wwise.

5.2 Dialogue system

One of the main objectives of this project is the design and implementation of a dialogue system that eases the process of localization and also the insertion of text for the different types of dialogues that will be explained later.

5.2.1 TEXT INPUT AND DISPLAY

The first thing needed is to introduce text easily and process it automatically for displaying it properly. *JSON* files were chosen to store all the text of the game because they are human readable (so it can be modified fast) and also because all the text will be stored in dictionaries, so complex features from the files are not neded and *JSON* is very simple to use and parse. An example of the structure that will be used for *JSON*s can be seen in the figure 15.

```
{ "items":
        [
        {"key":"chapter1_floornote", "value":"There's a note on the floor, it says "I'm
        watching you". Great! It's always nice to have someone that take cares of you.
        Maybe the concierge knows who is this nice person!"},
        {"key":"chapter1_kitchen_fridge", "value":"I'm not hungry. But I like seeing the
        animation of the fridge opening."}
        ]
}
```

Fig. 15 - Structure of the JSON files

Dictionaries will store all the text using an arbitrary key that, for now, is defined by the user. As can be seen in the figure 16, the function receives a string, the name of the file, which will contain a termination that indicates the language (some lines of the code have been removed for clarity).

```
1.      Dictionary<string, string> LoadLocalizedText(string fileName)
2.      {
3.      string dataAsJson = File.ReadAllText(filePath);
4.      LocalizationData loadedData = jsonUtility.FromJson<LocalizationData>(dataAsJson);
5.      }
```

Fig. 16 - Loading dictionaries

Then the file is loaded into a *LocalizationData* variable (figure 17).

```
1.      [System.Serializable]
2.      public class LocalizationData
3.      {
4.              public LocalizationItem[] items;
5.      }
6.
7.      [System.Serializable]
8.      public class LocalizationItem
9.      {
10.             public string key;
11.             public string value;
12.     }
```

Fig. 17 - Localization Data class

LocalizationData is used since the data cannot be loaded directly into a dictionary (it have to be done later in a for loop) because it is not serializable. In the game will be possible to switch between spanish and english ('es' and 'en' terminations). The key of the dialogues in the dictionaries does not change for different languages (since it is only used by the developers):

Three dictionaries are needed, one for item interactions, another for conversations with NPCs and the last one for the text of the narrator. When all of them are loaded, it is needed to access to the suitable dictionary during an interaction to show the text. Now, let's explain how to display the text properly.

When an interaction starts, a string is obtained from the dictionary which is one of the dialogues written for that interaction. If only one character is going to say something, the whole dialogue is obtained.

If there are more characters involved in the interaction, each dialogue will be obtained after the immediately previous one is displayed. For example, in a conversation like this:

Cube: Hey, this is an example!

Concierge: I like examples!

Cube: Me too!

The first dialogue of Cube is gotten and once it has been processed and showed, the dialogue of the concierge is obtained and, finally, again to the last Cube's dialogue. So, in this explanation, 'a dialogue' just means one piece from the whole conversation (or the whole dialogue if, as said before, only one character is going to talk).

Dialogues are displayed using a 'typewriter text effect', so the string is going to be showed character by character. To achieve this a foreach loop is used inside a coroutine as can be seen in the figure 18. The if-else condition checks if the player is pressing the B button. If so, the effect is accelerated and the characters are showed faster.

```
1.      StringBuilder sb = new StringBuilder();
2.
3.      foreach (char c in stringNodes[0].stringNode)
4.      {
5.              sb.Append(c);
6.              currentTextBox.text = sb.ToString();
7.              if (Input.GetKey(KeyCode.Joystick1Button1))
8.              {
9.                      yield return new WaitForSeconds(fastTypingSpeed);
10.             }
11.             else
12.             {
13.                     yield return new WaitForSeconds(TypingSpeed);
14.             }
15.     }
```

Fig. 18 - Typewritter text effect

But the system still needs some improvements. If the typewriter effect is executed now a problem will appear: if part of a word does not fit in a line, that word will be initially displayed on that line (remember that the string is showed character by character) but when the system reachs to a character that makes the string to be greater than the width of the dialogue box, this word will be removed from the line and will be displayed in the next line. This is a little bit annoying and to fix this it is needed to first split the whole string into different lines, so the system makes sure that the typewriter effect will display a bunch of strings that already fit in a single line.

Dialogues are showed in a default textbox component inside a Canvas. To store the strings that have been already split, a class called *StringNode* has been created. It has two member variables, the string itself and the width in pixels of that string (as an integer variable). So going back to the point, to split

the string it is needed to know the font used, the current size of the font, the current width of the textbox that is inside of the dialogue box and the width (in pixels) of every character. First, to get the width of a character, the methods of the figure 19 are used:

```
1.      font.RequestCharactersInTexture(currentString, currentTextBox.fontSize);
2.
3.      CharacterInfo characterInfo = new CharacterInfo();
4.      font.GetCharacterInfo(char character, out characterInfo, currentTextBox.fontSize);
```

Fig. 19 - Loading characters as textures

Basically what the system is doing here is obtaining a character as a texture. The result is stored in the *characterInfo* variable (as can be seen, it is passed as reference in the *GetCharacterInfo* function), and then it accesses to *characterInfo.advance*, which is the width of the character in pixels.

Once this is done, the string can be split. It is just needed to iterate the whole string character by character and add the width of each character in a variable. If the maximum width is reached (so the width of the string is equal or greater than the width of the textbox), the string has to be split there but with taking into account that:

```
-   If the character that does not fit in the line is part of a word,
    the function must find the space that is before that word and split
    there (removing the space since it is not needed anymore).

-   If the character that does not fit if in the line is a space, it is
    removed and the string is split there.
```

Fig. 20 - Splitting strings

Taking into account that these are no complex operations, there is no need to show the explicit code. Once the string has been split, it is added to a list and the system keeps iterating over the remaining dialogue. When finished, that function finally returns a list of *StringNode* objects that is used in the coroutine where the typewriter effect is. Basically it is only necessary to extend the function of the figure 18 and iterate the list of *StringNodes* until there are no remaining lines to show. It is important to pay attention to one thing: the number of lines that are displayed at a time inside the textbox are defined manually. In this case, the maximum number of lines has been defined to 3, so if a split dialogue have 5 lines, it will display 3 (character by character, line by line), then it will remove them and show the remaining 2.

But the displaying system does not ends here. The last feature to add is to automatically update the size of the dialogue box to contain the lines of the dialogue. For example, do not want to have the same size for a long dialogue than for one that says only ‘yes’. As said before, the maximum number of lines have been already defined, as well as the maximum width of the Dialogue Box. This is because it depends on the game; in this way it is possible to set the dialogue box to have a maximum size. So having this maximum values as a reference, a function that determines the best size for the dialogue size can be created.

To calculate the best size for a dialogue box, the text is first split. It will always be split assuming that the size of the dialogue box is the default; a more complex function would be needed to split the text with different sizes, and the only objective of this is to readjust the size to fit a little bit better with the dialogues. So once the text has been split:

```
- The function obtain the number of lines of the dialogue just che-
  cking the size of the string nodes list.

- When the number of lines has been already determined, it needs to
  check which one is the largest (remember that in the string nodes
  the string and its width are stored, so it is an easy operation) and
  the width of the largest line will be the width of the dialogue box.
```

Fig. 21 - Resizing dialogue boxes

It is important to bear in mind that the system will search for the largest string only considering the strings that will be displayed at the same time. So, for example, if a dialogue needs 5 lines to be displayed, at the first iteration the system will only care about the first 3 strings (because the default number of lines was set to 3), and then those strings will be removed from the list to display the remaining ones in the next iteration. The height of the dialogue boxes is calculated by simply multiplying the height of a line (*font.lineHeight*) by the number of lines which are going to be displayed.

Finally, the last thing to do are the transitions to show the dialogues smoothly. So, when they are hide and an interaction is triggered, a tween function (ref) is used to perform a fade in (updating the alpha value of the color of the dialogue box from 0 to 1) and the same for a fade out. Also, a transition is performed when the size of the dialogue box is updated, using the same tween system but, in this case, updating the width and the height values. The time needed for every type of transition is passed as a parameter of the function.

5.2.2 INTERACTION WITH ITEMS

All the functions needed to display the text have been already explained, and now they are used in the different type of interactions. When Cube interacts with some items, he says something about them, others just get modified (such as a lamp, which can be turned on or off). For the interactions where dialogues are involved, a prefab has been created and it has attached a trigger collider (to start an interaction Cube must be colliding with this trigger) and a script, *ItemDialogTrigger,* that stores a public string, which is the key for the dictionary. As said before, keys are defined by the user so when this prefab is put into a scene, it is necessary to write in the inspector the key defined in the JSON file for the desired interaction. There are also two more public values in the script, the X and Y offset for the 'Info Button'. The Info Button is an image of the button B of the Xbox controller that appears in the screen (rendered as an image of the Canvas) when an interaction can be started. Initially the position of this image is the same than the prefab (the world coordinates are translated to screen coordinates using the method of the Camera class *worldToScreenPosition*) and the X and Y offset allow to define its location in the screen having the prefab position as the center.

So when Cube enters the trigger collider, the button image is showed. If the button B is pressed, the item interaction begins and the first thing that is needed to do is to check whether the items dictionary has a value for the key that was written in the script. If so, the interaction begins and the function must do the following:

```
1.     Store the dialogue from the dictionary in a variable
2.     Split it and store it in a list of StringNode objects
3.     Get the number of lines needed
3.     Get the height and width for the dialogue box
4.     Show the dialogue box with a smooth transition
5.     While (there are lines in the list)
6.     {
7.             Show a line using the typewriter effect and remove it from the list
8.             If the maximum number of lines in the dialogue box is reached
9.             {
10.                    Wait for the player to press B to continue
11.                    Remove the lines from the textbox of the dialogue box
12.                    Get the new number of lines needed
13.                    Resize the dialogue box using a transition
14.            }
15.    }
16     Hide the dialogue box with a smooth transition
17.    Disable the trigger (items can be interacted just once)
18.    If not null, call to the AfterInteractionFunction
```

Fig. 22 - Item interaction flow

In the last line of the figure 22 the *AfterInteractionFunction* is called if it exists. This is a function defined by the programmer in every case and must be passed to the *ItemDialogTrigger* script at the beginning (when *Start* functions are executed) as an *Action* variable. For example, when Cube interacts with the fridge in his apartment, he says 'I like to see the animation of the fridge opening' so, when the dialogue ends, *AfterInteractionFunction* is called and the animation of the fridge is executed. By doing this, there is a lot of flexibility with the function that is needed at the end, because it can be defined in another script, with its own variables, attached to its own gameobject… And tons of references can be saved.

5.2.3 INTERACTION WITH NPCS

Interactions with NPCs work a little bit different because now there is more than one character involved, hence there are more dialogue boxes to handle and it is necessary to get more dialogues from the dictionaries, as explained in the point 5.2.1.

Another prefab has been created for this type of interactions and it also has a trigger collider and a script, *NPCDialogTrigger*, that has an X and Y offset for the Info Button. The key value is an array of strings, because it allows us to have different conversations with the same NPCs. How the array is used will be explained later. The last thing needed to set in the script are the NPCs involved in that interaction. For that, another script has been created, *NPCDialogue*, that is attached to every character that

is going to talk. This script just handles the position of the dialogue box of that character and its position offset in relation to the position of the character using, again, the *worldToScreenPosition* method from the *Camera* class. Once all the characters that are going to talk have been already set up as well as the keys that will be used to access to the dictionary, let's explain how the interaction itself works.

First, in the *JSON* file the dialogues are stored like this:

```
{"items":
        [
        {"key":"concierge_d0", "value":"Cube: Hi."},
        {"key":"concierge_d1", "value":"Concierge: Good evening, Mr. Cube. How can I help you?"},
        {"key":"concierge_d2", "value":"Cube: Somebody knocked on my door."}
        ]
}
```

Fig. 23 - Structure of the JSON files for dialogues

As can be seen, in the field *value* the dialogues start with the name of the character. It won't be showed in the game, it is just used in the code to check which character is going to talk to assign him the string obtained (that is, to his dialogue box) and, after that, the name is removed from the string. In the field *key* can be seen that the same key is used for all the dialogues of a conversation, and they end with a numeric value. This is because an integer counter will be used to access to them automatically, and for that it is just necessary to initialize the counter to 0 and set the key value to (in this case) 'concierge_d'. So to get the dialogue from the dictionary, the string 'key + counter' is used as the key.

Now that the use of the keys have been explained, let's see how the interaction works. At the beginning was said that the prefabs does not store only one key, but an array of them. This is because maybe Cube talks with an NPC and when the conversation ends, if he can talk to him again, the conversation can be different. This is achieved just by having a counter to track which conversation needs to be displayed. That counter is the index used to get a value from the array of keys. When the counter reachs the end of the array, it can be kept as the last possible index of the array (so if Cube talks again with that NPC, now he will always say the same thing) or it can be incremented and the interaction is disabled.

To start a conversation the process is the same than for items. So, once the player has pressed the button B and it have been already checked that the key used exists in the dictionary, a function needs to do what figure 24 shows:

```
1.     Get the key from the array of keys using the keysCounter
2.     While (key + dialogueCounter exists in the dictionary)
3.     {
4.     Get the string from the dictionary and store it
5.     Check which character is going to talk
6.     Get the dialogue box and textbox of that character
7.     Remove the name from the string
8.
9.     //Now, the text is displayed using the same process than in item interaction
10.    //(from line 2 to line 15)
11.    }
12.
13.    //Conversation ended
14.    Reset dialogueCounter to 0
15.    Hide dialogue box with a smooth transition
16.    If desired, udpate the keysCounter
17.    If keysCounter has reached the end of the array, keep it there or disable the interaction
```

Fig. 24 - Conversations flow

In this case there are no *AfterFunctionCalled* function like in the items interactions because they were not needed in the game, but it could be easily implemented just adding it to the end of the method.

However, a different feature has been added to the interaction with NPCs: pauses. In some moments of the game, for narrative reasons, a character waits for a determined number of seconds before its dialogue box is showed. The time that will take a pause is defined in the *NPCDialogTrigger* script as a float number. But the pause itself is set in the *JSON* file like this:

```
{"items":
        [
        {"key":"concierge_d0", "value":"Cube: Hi."},
        {"key":"concierge_d1", "value":"Pause"},
        {"key":"concierge_d2", "value":"Cube: Somebody knocked on my door."}
        ]
}
```

Fig. 25 - Special commands in JSON files

As can be seen, the pause is treated as a normal dialogue, but it has no name attached. The pause affects to the next character. In this case, the concierge would have to wait. It is only necessary to add a special condition in the code when the name of the character is checked. If is a pause, *yield return new WaitForSeconds(pauseTime)* is used inside the condition, since the code is in a coroutine.

5.2.4 INTERACTION WITH THE NARRATOR

Interactions with the narrator are the simplest, because it is not necessary to care about dialogue boxes and sizes, as can be seen in figure 26. The text will be displayed in a textbox positioned in the lower part of the screen as a subtitle and the split function is partly similar.

First of all, the typewriter effect won't be used, the text will be showed everything at once. Second, the split function will be 'hacked', because the text box will show two lines. Taking into account that the string will not be displayed character by character, the width of the textbox can be set as if it was the double. But instead of passing the width as *textbox.width * 2* to the function, it will be multiplied for a smaller value, such as 0.8, to avoid problems of words that maybe does not fit.

Fig. 26 - Narrators dialogue box

Once the text is split, the process of displaying it is completely different from item and NPC interactions, because the player won't need to press any button. The text will appear in the screen when Cube collides with a trigger. A prefab has been created and it has attached a trigger collider and a script, *NarratorDialogueTrigger*, which has a public string 'key', as would be expected, and also a float called 'TimeActive'. This variable specifies the time that has to pass before the dialogue is hidden. So when Cube collides with a trigger, the dialogue is automatically showed making a smooth transition, then the collider is disabled (the text will be displayed only once) and after the specified time, the dialogue will dissappear.

Finally, it is important to take into account that Cube could collide with other narrator triggers while a dialogue is being showed. In that case, it is necessary to first hide the current dialogue. It is easy to do since all the scripts of the narrator prefab references to the same textbox. To avoid two scripts from trying to modify the textbox at the same time, a static boolean called *isBeingUpdated* is used and every script will check it before doing something and, if it is true, the script will wait for it to be false.

5.3 Camera positioning system

The main movement of the camera is very basic, it is only necessary to use the *SmoothDamp* function from Unity to follow Cube smoothly. To create a more immersive experience, a system that updates the camera position while it still follows Cube has been programmed. It allows to have different shots in different parts of the levels. For example, when passing accross a bridge over a deep abyss, the camera position is updated to be away from Cube to 'feel the height' (figure 27).

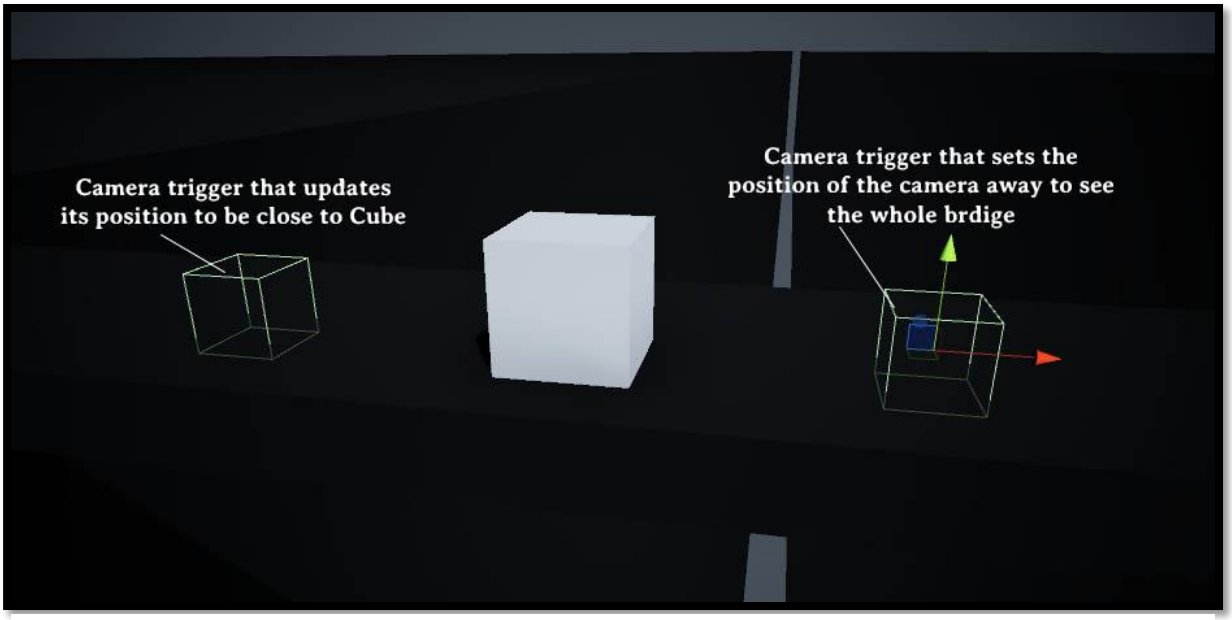


Fig. 27 - Trigger camera system

Again, trigger colliders are used to indicate where to update the position of the camera. These triggers are collided by Cube, not the Camera. A prefab was created with a trigger collider and a transform component that is the new position of the camera, so when the collision is triggered, it is only necessary to make a transition from the current position of the camera to the position of the prefab. It is important to make sure that only the Y and Z values are updated –the X must remain following Cube.

5.4 Player movement

Cube is moved using forces. It is a very simple system that just gets the input of the controller in the horizontal axis:

```
1.      //Input movement
2.      float moveHorizontal = Input.GetAxis("Horizontal");
3.      Vector3 movement = new Vector3(moveHorizontal, 0, 0);
4.      //Basic movement
5.      movement = new Vector3(moveHorizontal, 0, 0);
6.      rb.AddForce(movement * Speed);
```

Fig. 28 - Cube movement

A value from 0 to 1 is returned and it is multiplied by the speed value (manually defined). Using this technique a smooth movement is achieved, with acceleration and deceleration.

The last thing remaining is to rotate the Cube. He has no textures but his 'face' is simulated, so he must look towards the direction of the movement (see figure 29).

```
1.      //Rotation
2.      if (movement != Vector3.zero)
3.      {
4.              Quaternion lookRotation = Quaternion.LookRotation(movement);
5.              transform.rotation = Quaternion.Lerp(transform.rotation, lookRotation, rotationSpeed *
6.                                                                                      Time.deltaTime);
7.      }
```

Fig. 29 - Cube rotation

5.5 Scripting for puzzles

Buttons react different in each one of the puzzles of the game and custom scripts have been created for them, but since in order to save time a base class was created for common behaviors.

This class, *ActionButton,* is attached to a prefab that also has a trigger collider and it takes care of things that are common to all the buttons. First, it manages the process of interaction, showing the 'Info Button' image when Cube can interact with it (that is, when Cube is colliding with the trigger) and handling the input of the controller. When the player interacts with the button, the method *doSomething* is called. Like in the item interactions, this method is an Action variable, so a custom function can be defined for each button and it can be sent to each script, as can be seen in the figure 30.

```
1.      Action doSomethingFunc;
2.
3.      void doSomething()
4.      {
5.              doSomethingFunc();
6.      }
7.
8.      public void setActionFunc(Action func)
9.      {
10.             doSomethingFunc = func;
11.     }
```

Fig. 30 - Custom function for buttons

ActionButton also defines public functions for the animation of pressing and releasing the button. Button animations are made via code, using the Tween class to update their position smoothly. So, when pressed, the Y of the local axis of the button is updated (the button moves down when pressed and up when released) and it also changes its color from red to green if it is pressed and vice versa..

The functions for pressing and releasing the button must be used inside the *doSomething* function, whereas it is defined. This gives us flexibility to decide how the button reacts: maybe the intention is to press and release the button after a determinated time, or to keep it pressed while the player has the button of the controller pressed… etcetera. All of the custom scripts created for each puzzle are very simple, so it is not necessary to explain how to program them in detail, and they all are described in the point 3.4.

5.6 Working in Unity

5.6.1 LIGTHING AND POST-PROCESSING

To achieve a better visual aesthetic, some tweaks have been made to the lighting options in Unity. First, the use of mixed lighting. The game has to be very dark in some parts of the levels and for this, directional lights cannot be used. But this leads to have a very dark game in general and, to avoid this, the baked lighting is necessary to get a little bit of bright by using the indirect lighting that is gotten from the different source lights that are placed in the scene.

The light and colors of the scene are finally improved by using the post-processing tool (see figure 31) from the standard assets of unity. To add even more immersion feeling and darkness the vignette and ambient occlusion effects are used, which add more shadows in their own way. Finally, bloom and depth of field are used to give more presence to the lights.

Fig. 31 - Comparison: left without post-processing, right with post-processing

5.6.2 TIMELINES

Timelines were introduced in the 2017 version of Unity and although they are a bit limited they are so powerful for games like that. They are used in every cutscene of the game to move Cube and other elements such as doors. Sometimes it is necessary to place Cube in a determined position before executing a timeline to avoid cuts (Cube suddenly appearing in the initial position set in the timeline), and the Tween class is used for this. The camera script that follows Cube cannot be used while timelines are being executed because some problems of jittering can appear, so the camera position must be updated using the timeline too.

5.6.3 WWISE

Wwise is the tool used to integrate audio inside the game, both sound effects and the soundtrack. This plug-in is very simple to use and implements useful functions to modify sounds at runtime through events that can be customized. Some sound effects are played when Cube collides with that trigger and others are called through code.

More complex events modify all the sounds (except the soundtrack) if Cube is in two particular places: passing over the bridge of the chapter 1 and inside the pipeline of the chapter 2. In both of them is applied a Low-pass filter and all the sounds are passed through an auxiliar bus that has an effect of reverb.

6 PROJECT MONITORING

In the first point of this document an initial planning describes the different tasks to be done in several months, split in weekly work. Each week, the planning was reviewed to check if all the work was been done succesfully and to make changes if needed. During the development of the game, there has been modifications of the planning because of some changes in the structure of the story and also because of problems with the timing —some tasks took more time than expected.

At the beginning, all the chapters of the story was designed to be divided into two different parts: one part purely narrative, only to talk with NPCs and with realistic looking 3D models, and the other part only focused on solving puzzles. This structure can be found in the first chapter, which has not been modified, but the second one has a mix of puzzles and narrative. This had a big impact in the planning because the sections of puzzles were initially designed to have a very basic environment (in terms of 3D modelling) but the decision of mixing the two parts did not affect only to the narrative but also to the environment, which mixes now realistic looking and basic models. These issues delayed the project and finally the Chapter 3 of the game had to be removed from the development plans and the Chapter 1 got bigger. The last two weeks of the planning where scheduled to focus on the development of this last chapter, but they were finally used to fix and polish the experience in the first two chapters. Apart from that, talking about programming, the planning has been followed succesfully and all the systems needed were created in time.

7 RESULTS

In this section, the results of the game will be showed. The full gameplay of the game can be seen in this video. Also, the full source code of the game can be examined in Gitlab.

At the beginning, when the game is executed, the menu appears. If the player presses the button 'New game', it will load the introduction (figure 32).

Fig. 32 - Main menu

The player can explore the apartment and interact with some items (figure 33 and 34).

Fig. 33 - Cube in the kitchen

Fig. 34 - Cube interacting with a wall

After that, when someone knocks the door, Cube passes through the corridor (figure 35) and there is a little cutscene while he is in the elevator (figure 36).

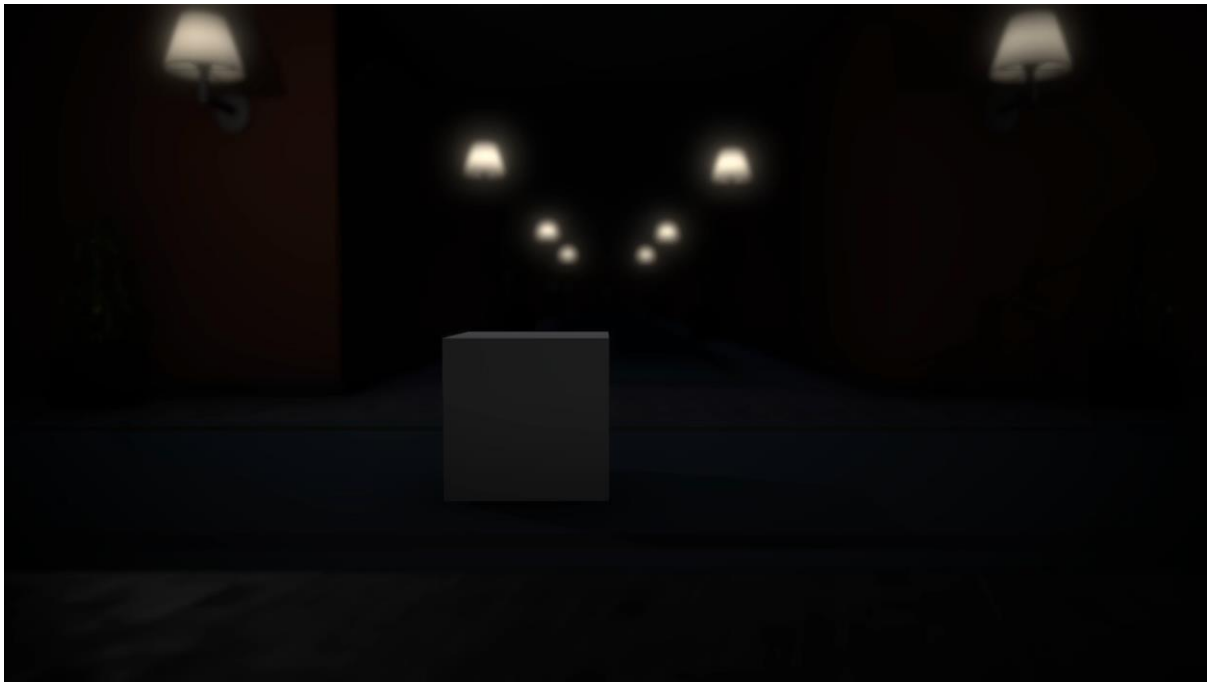


Fig. 35 - Cube in the corridor of the building

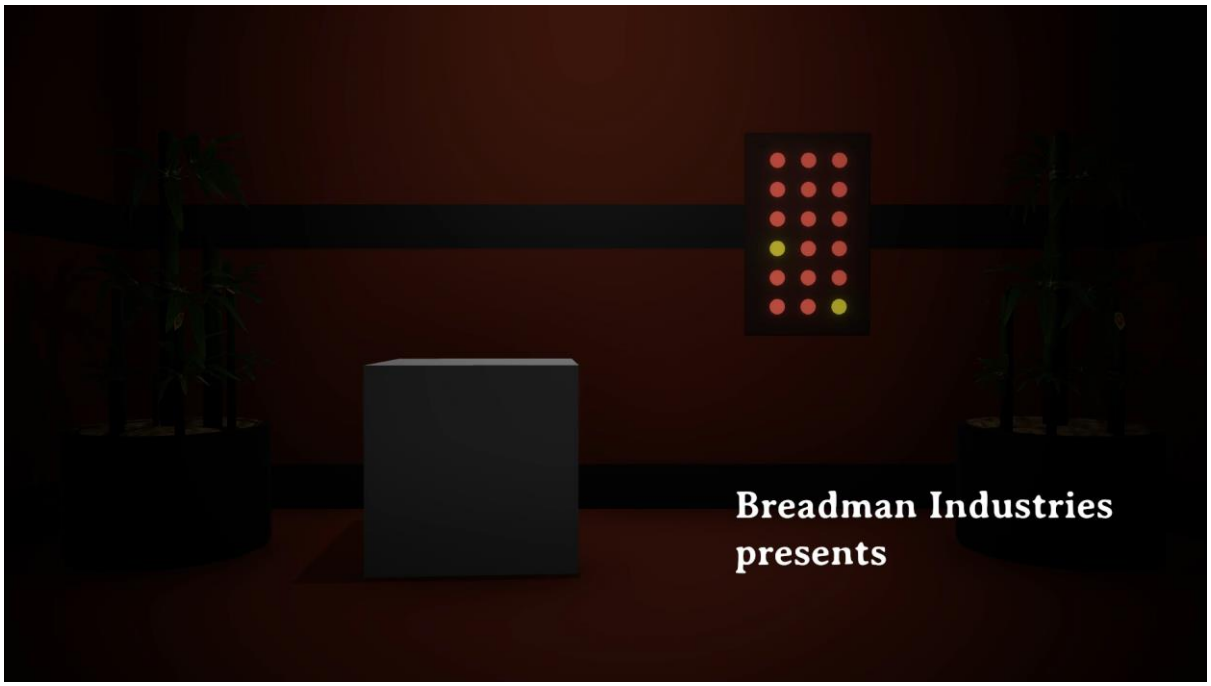


Fig. 36 - The elevator

Then Cube has to talk with the concierge (figure 37) and, after that, outside the building, the narrator will start to tells the player the story (figure 38).

Fig. 37 - Conversation with the concierge

Fig. 38 - The narrator

Outside, there are some puzzles that Cube needs to resolve (figures 39 and 40).

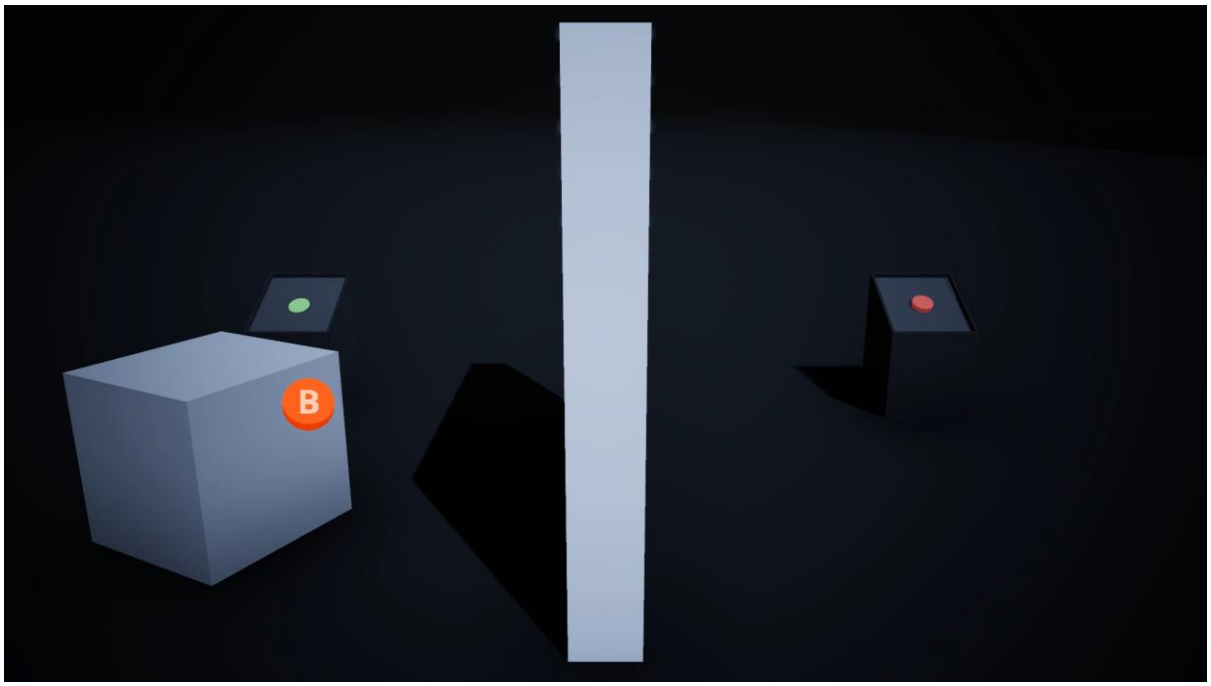


Fig. 39 - A puzzle of the chapter 1

Fig. 40 - The bridge, and optional puzzle

In the chapter 2, Cube is inside a pipeline (figure 41) that leads to City of Spheres (figure 42).

Fig. 41 - Puzzle inside the pipeline

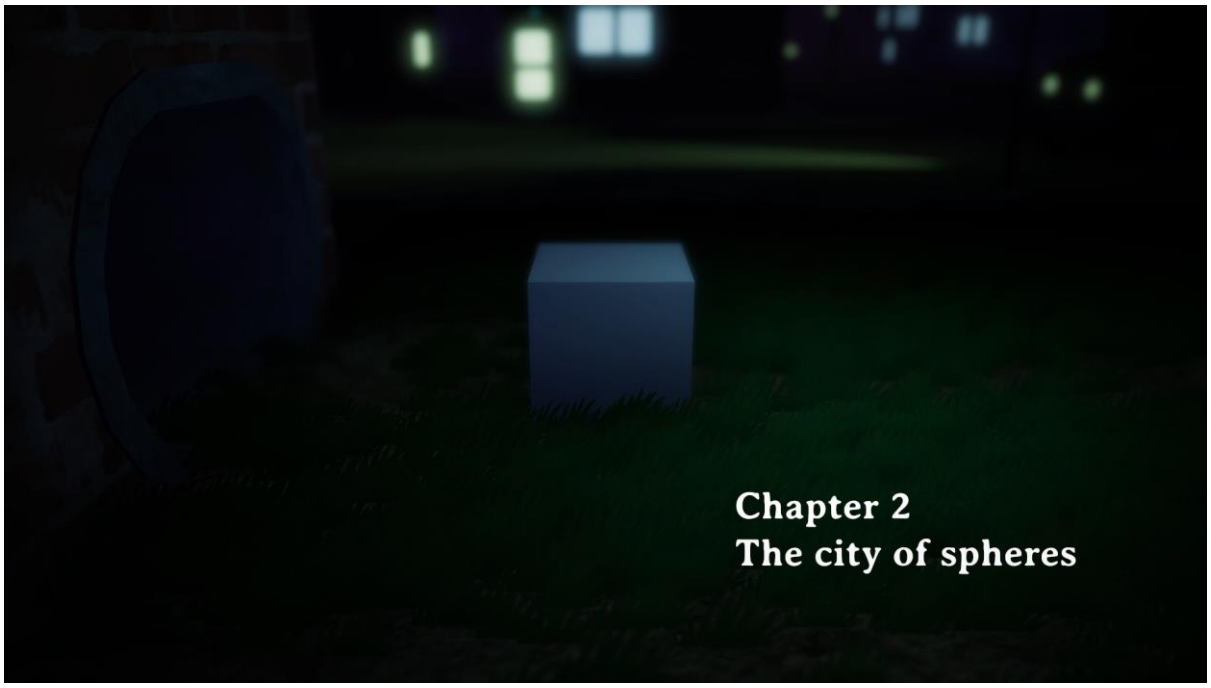


Fig. 42 – The city of spheres

8 CONCLUSIONS

Developing narrative games has been always a complex challenge because they require a huge amount of work to create only a few minutes of gameplay, and maybe they seem to be very easy to program since they do not have difficult fighting systems or that kind of things. But, in fact, narrative games do need to work very smoothly and handle a lot of different possibilites of how the player can play the game and resolve those situations in a subtle and elegant way, because in the attempt of generating an immersive world, every mistake that could appear in the game can ruin this objective.

The Ultimate Story of a Cube creates a universe where the story is told to the player through a lot of systems and different elements that work together to achieve immersion, and we can conclude that in a satisfactory manner. Those elements are:

- A good narrative with an interesting and funny story.
- Well designed puzzles that are connected to the story.
- A suitable and scalable dialogue system that allows us to create different types of interactions.
- A camera positioning system that permits to achieve different shots to get the desired feeling in every moment of the game.
- A good aesthetic that mixes realistic looking models and basic shapes.
- An accurate design of sound effects and a soundtrack that transmits tension and suspense.

Talking about narrative, the style of the story designed for this game was achieved but with some exceptions due to the time limitations –the chapter 3 could not be developed and the main arc of the story is not concluded properly, but the chapter 1 is bigger and the whole introduction more powerful. The puzzles are difficult enough to be a challenge but without leaving the player stuck in one puzzle too much time.

To create the immersive world, 30 objects were modeled (apart from the downloaded ones), 28 different sounds effects have been mixed between them and used, and finally 4 songs were composed, recorded and mixed.

The systems needed for this game were successfully developed. At the end, the dialogue system allows the developer to write dialogues easily in a file and they are automatically split and displayed properly, according to some settings that give flexibility to the way they are showed (the maximum size of the dialogue box, the maximun number of lines displayed at the same time, the speed at which the characters appear…). Also, the use of this files permits to switch the language easily in the main menu of the game by loading the desired file, since they all use the same key for the dictionaries but their values are in the determined language. In terms of localization, there is a good advantage of this dialogue system: it is not

necessary to worry about the length of the dialogues in different languages since the dialogue box updates automatically so the text fits properly.

Finally, the camera system works smoothly and the game changes its perspective at runtime updating the camera position as well as the post-processing values of field of depth, allowing to have more dynamic scenes. It can also be used for the design of puzzles, like the puzzle #3 explained in section 3.4.1.

For this game has been created 40 scripts with aproximately 6000 lines of code.

9 BIBLIOGRAPHY

References

Koster, R. (2013). *A theory of fun.* Scottsdale: Paraglyph Press.

Martínez, V. (2016). A propósito de las opciones de sonido. En P. Sànchez, F. Pinto, & V. Martínez, *El peor año de la historia de los videojuegos* (págs. 49-56). Barcelona: AnaitGames.

Assets

[1] Next Level 3D. (2015, September 3). *PBR Armchair*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/furniture/pbr-armchair-44819

[2] Vis Games. (2012, May 11). *Books.* Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/interior/books-3356

[3] Vertext Studio (2017, December 14). *Big furniture pack*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/furniture/big-furniture-pack-7717

[4] Studio Krokidana (2016, July 26). *Kitchen creation kit*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/environments/kitchen-creation-kit-2854

[5] Jake Sullivan (2017, Mar 2). *Kitchen Props Free*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/interior/kitchen-props-free-80208

[6] Nobiax/Yughues (2015, Mar 27). *Yughues Free Decorative Plants*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/interior/yughues-free-decorative-plants-13283

[7] Flatriver (2016, Aug 20). *Simple Wall Lamp.* Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/props/simple-wall-lamp-69411

[8] Polygon Land (2018, Mar 13). *Simple City pack plain*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/3d/environments/urban/simple-city-pack-plain-100348

[9] CACTUSCREATIVES PVT. LTD. (2014, Jul 19). *Car*. Retrieved June 03, 2018, from: https://assetstore.unity.com/packages/3d/vehicles/land/car-20128

[10] PIXTIM (2016, Jun 17). *Car'Toon: The Sport Car with interior*. Retrieved June 03, 2018, from: https://assetstore.unity.com/packages/3d/vehicles/land/car-toon-the-sport-car-with-interior-62697

[11] WEISSEGAMES (2015, Apr 20). *Old military car*. Retrieved June 03, 2018, from: https://assetstore.unity.com/packages/3d/vehicles/land/old-military-car-20945

[12] Kobra Game Studios (2016, Oct 18). *Chainlink Fences*. Retrieved June 03, 2018, from: https://assetstore.unity.com/packages/3d/chainlink-fences-73107

Sounds

[13] Lebaston100 (2014, July 27). *Heavy Rain*. Retrieved May 20, 2018, from: https://freesound.org/people/lebaston100/sounds/243627/

[14] Timbre (2014, Jan 20). *Like static from old cheap portable analogue television*. Retrieved May 20, 2018, from: https://freesound.org/people/Timbre/sounds/214757/

[15] Anagar (2015, Mar 23). *Knock the door*. Retrieved May 20, 2018, from: https://freesound.org/people/anagar/sounds/267931/

[16] Amholma (2016, April 26). *Door open close*. Retrieved May 20, 2018, from: https://freesound.org/people/amholma/sounds/344360/

[17] Trautwein (2015, Jan 28). *Elevator3*. Retrieved May 20, 2018, from: https://freesound.org/people/Trautwein/sounds/262574/

[18] Strathamer (2018, Jan 19). *Dark ambient 3*. Retrieved May 20, 2018, from: https://freesound.org/people/strathamer/sounds/415889/

[19] Jorickhoofd (2012, June 30). *Metal heavy mechanics*. Retrieved May 20, 2018, from: https://freesound.org/people/jorickhoofd/sounds/160048/

[20] InspectorJ (2016, April 22). *Dripping fast*. Retrieved May 20, 2018, from: https://freesound.org/people/InspectorJ/sounds/343764/

[21] Diramus (2016, Aug 3). *VW Touareg.* Retrieved May 20, 2018, from: https://freesound.org/people/Diramus/sounds/351421/

[22] InspectorJ (2017, October 25). *Realistic Wind*. Retrieved May 20, 2018, from: https://freesound.org/people/InspectorJ/sounds/405561/

[23] Pixel Processor (2018, April 18). *Substance materials Lite*. Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/vfx/shaders/substances/substance-materials-lite-115293

[24] Michael Kremmel (2018, May 4). *MK Glass Free.* Retrieved May 20, 2018, from: https://assetstore.unity.com/packages/vfx/shaders/mk-glass-free-100712