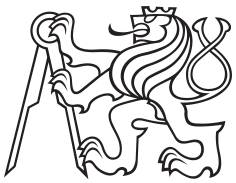
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Logic 3D game

Petr Varga

Supervisor: Ing. Ladislav Čmolík, Ph.D.
May 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Varga** Jméno: **Petr** Osobní číslo: **483768**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Logická 3D hra

Název bakalářské práce anglicky:

Logic 3D game

Pokyny pro vypracování:

Seznamte se s principy návhu počítačových her. Proveďte analýzu herních principů používaných v 3D logických počítačových hrách. Dále se seznamte modulárními komponentami a jejich využitím při návrhu úrovní. Na základě analýzy vytvořte design dokument pro 3D logickou počítačovou hru. Dále vytvořte modulární komponenty, ze kterých se budou skládat jednotlivé úrovně, jejich materiály a textury. Dle design dokumentu vytvořte s využitím modulárních komponent alespoň tři hratelné úrovně hry. Výslednou hru otestujte pomocí kvalitativních testů alespoň s šesti hráči.

Seznam doporučené literatury:

[1] R. Koster. Theory of Fun for Game Design, 2nd edition, O'Reilly Media, 2013.
[2] J. Schell. The Art of Game Design: A book of lenses. CRC Press, 2008.
[3] B. L. Mitchell. Game Design Essentials, John Wiley & Sons, 2012.
[4] S. Rogers. Level up! the Guide to Great Video Game Design, John Wiley & Sons, 2014.
[5] E. De Nucci and A. Kramarzewski. Practical Game Design : Learn the art of game design through applicable skills and cutting-edge insights, Packt Publishing, 2018.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **18.03.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank Ing. Ladislavu Čmolíkovi, Ph.D., for his guidance, willingness to help, and for giving me the opportunity to work on the game. Furthermore, I would like to thank my mother, who provided emotional support. Lastly, I would like to thank all the people who helped me test the game, provided new points of view, and discovered many bugs.

Declaration

I declare that I made this work independently and that I cited all sources I have used.

In Čelákovice, 21. May 2021

Abstract

This work analyses the process of designing games, mechanics of logic games, and creating of modular components. I use the analysis to design a game of my own and develop it in the Unity game engine. Afterward, I test the game.

Keywords: Unity, logic game, 3D game, modular components

Supervisor: Ing. Ladislav Čmolík, Ph.D.
Praha 2, Karlovo náměstí 13, E-418

Abstrakt

Tato práce analyzuje proces navrhování her, mechaniky logických her a vytváření modulárních komponent. Tuto analýzu následně použiji k navržení své vlastní hry a vytvoření této hry v Unity. Následně tuto hru otestuji.

Klíčová slova: Unity, logická hra, 3D hra, modulární komponenty

Překlad názvu: Logická 3D Hra

Contents

Figures

Chapter 1

Introduction

Nowadays, computers are everywhere. People are using them for work, communication, or spending their free time. There are many things a person can do on a computer to get rid of boredom, from watching videos on the internet to playing some computer games. Computer games are played across the population, no matter the gender or age. For many, games are a massive part of their lives, and for some, playing games is their work.

1.1 Goals of Thesis

The main purpose of this work is to create a 3D logic game. The first step is to analyze how to design and create a game and the necessary means to do so. The second step is to design the game and create a design document. After that, the creation of models and implementation follows. And during the process of implementing the game, test it so that the test results can lead to an improvement of the game during the development stage.

1.2 Motivation

The main reason for this thesis is my obsession with games and my dream to create one. The reason for developing a logic game is that there are many

3D games full of action that might contain a few puzzles, but there are not that many 3D games whose primary purpose is to solve puzzles with a bit of action on the side. Another reason is the opportunity to improve my abilities of game creation by creating a game alone and trying almost every step of the game development process.

Chapter 2

Analysis

There are many steps that a person has to make to create a game. The first step is to design the game. Then it is necessary to decide how the game will be implemented. I decided to use the Unity game engine[Unic]. The main reason for that was my previous experience with it. Another important step is the creation of models and textures for the game. In this chapter, I analyze the process of designing a game, what the modular components are, and rendering in Unity.

2.1 Game Design

People might feel that it is an easy task to create a game when they have a good idea. It can be an idea of an environment that no one had ever thought of before or a fantastic main character. These things are essential for a game, but they are not the most important.
The most important things for a game are its rules, objectives and a story if the game has it[Mit12]. Rules are defining how the game will be played. They tell how the player can achieve his goal. Objectives are the goals of the game, things that have to be done to win the game if it is possible. It might not be possible as there are some games that do not have any end. For example, Tetris, a game where the player is trying to build rows from falling blocks. When a row is complete, it disappears. A player can lose by not having any space left to place the blocks, but he cannot win. In such a game, the player is usually trying to beat some score.
The last thing is the story. Certain designers say that it is an essential part

of the game as it increases the player's engagement. Others do not think so[Rog14]. Some games have practically no story at all; again, Tetris is a good example. However, there are games with whole new worlds, such as World of Warcraft, where the story is significant, and as the player progresses, the story evolves. Some players enjoy the lore of the game, and they will spend a lot of time just trying to uncover everything there is, whereas others will skip every cut scene or dialog possible because they do not care about it. If the story of the game is important, it is necessary to include the essential parts in the gameplay, so those who skip everything will still be able to follow. And this is why there are game designers.

The game designer's job is to decide what the game will be like[Sch08]. There are many decisions that the designer has to make. Deciding on all of the things mentioned before and many more. Some others are: how will the environment look, what will the main character be like, what kind of background music will be used... Usually, all of these decisions are written down in a game design document.

The process of designing does not stop until the game is finished. Many decisions are not even possible to make at the very beginning until the designer sees the game in action. Therefore, the design document evolves as the decisions are made[Sch08].

The goal of a game is to entertain a player in some way. There are many ways the player could get bored or frustrated and leave the game even though he did not finish it. One of the reasons for a person to drop the game is that its difficulty is not appropriate. A game cannot be too easy, neither can it be too hard. If a player can do everything without any difficulties, he will get bored, and if he cannot do anything because it is too hard, he will get frustrated.

A game usually starts with some tutorial where the player is presented with the basic game mechanics, and he has time to learn them. After that, the game should get gradually more complex. The player should still be able to solve everything by using the things he learned. However, at some stage of the game the gameplay might become too repetitive. There are ways to avoid repetitiveness, one of which is the addition of something new. It could be either some new type of enemy, a new obstacle, a new player's ability, or a new way of interaction.

Before committing to creating the game design, a designer should check if there are any similar games. If there are, the designer should decide how will his game be different from those already out there, what will be unique, as there is no reason to create something that is already on the market.

2.2 Similar Games

There are many logic games, but most of them are 2D. Some games focus solely on the puzzles, and there is no story at all. Those would be predominantly mobile games, but there are some computer games as well. There are many forms of puzzles. Some of them are based solely on the player's abilities. Every level in these games is always possible to complete on the first try a good example would be a game like Cut the Rope[Zep], where the player needs to get a candy into a creature's mouth by cutting ropes. Other form of puzzles include the necessity of luck. The player cannot always complete the stage, for example, games like Candy Crush Saga[Kin] or Bejeweled[Stu], both represent the match-three genre, where a player has to match three or more identical symbols to clear them out. For 2D games, the closest match to my idea is Aargon, which I describe more later.
There are two big titles for the 3D logic games: Portal and The Talos Principle, both of them are using a first-person camera view and have some similarities to my idea. In addition, I have found Archaica: The Path of Light, which is not using a first-person camera view but has some mechanics similar to those I would like to use. I describe these three games later.
There are many games that use puzzles and traps, like Tomb Raider, The Elder Scrolls V: Skyrim[Bet] or even World of Warcraft[Bli]. These games use the mechanics to make the gameplay more various, but they are not the core features. For example, in Skyrim, there are many types of puzzles and traps inside the dungeons similar to those I would like to use. There are masks that spit fire, corridors with swinging blades, or simple pressure plates that can activate an arrow trap. However the core mechanics of Skyrim are fighting with dragons and other monsters.

2.2.1 Aargon

Aargon[Gam], published in 1999, is a 2D logic game, where the goal is to pass colored laser beams through coins of the same color[Mobb]. To do so, there are many objects which the player can drag into the grid. These objects interact with the lasers, each in a unique way, from changing the color to bending it in a specific direction.
Later in 2002, Aargon Delux was published. It included levels from the original game and added new features like toxic barrels or slimes.

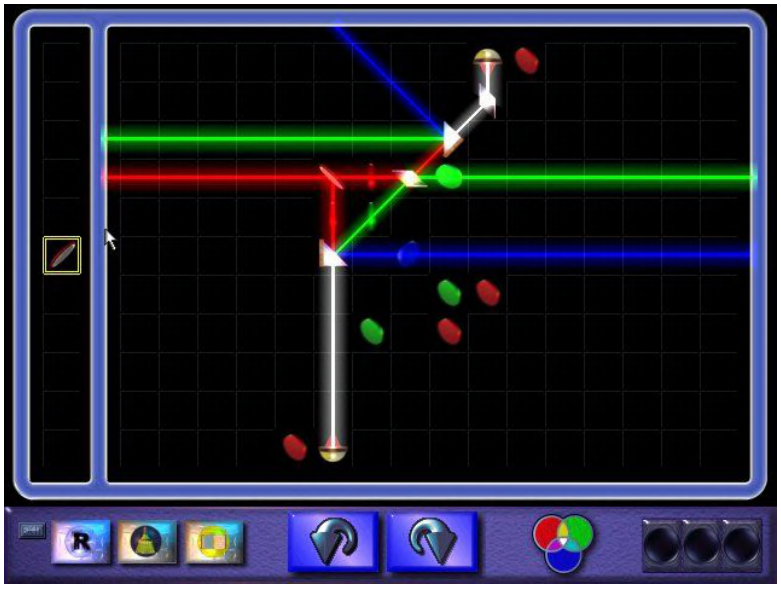


Figure 2.1: Aargon[Moba]

2.2.2 Portal

Portal[Val], released in 2007 by Valve, is a 3D puzzle game where the player uses a portal gun to solve puzzles. The portal gun can create two portal ends. One is orange, and the other one is blue. The portal connects the two locations visually and physically. Both portal ends are an exit and an entrance at the same time. The important thing is that when a moving object enters the portal, it comes from the other side of the portal with the same speed and direction. This allows the player to cross gaps by jumping down and coming from a wall(Figure 2.2). It is also possible to launch objects over long distances. When leaving a portal, the player is always reoriented based on gravity. Whenever a new portal end is created, the old one of the same color shuts down. Some obstacles pose a danger to the player and can kill him, for example, crushers or turrets. The player takes the role of Chell, who is subjected to tests by GLaDOS, an artificial intelligence. After Chell completes the final test chamber, GLaDOS attempts to kill her. Chell escapes and tries to destroy the artificial intelligence. The game was well-received, and the subsequent Portal 2, released in 2011, was even more successful.

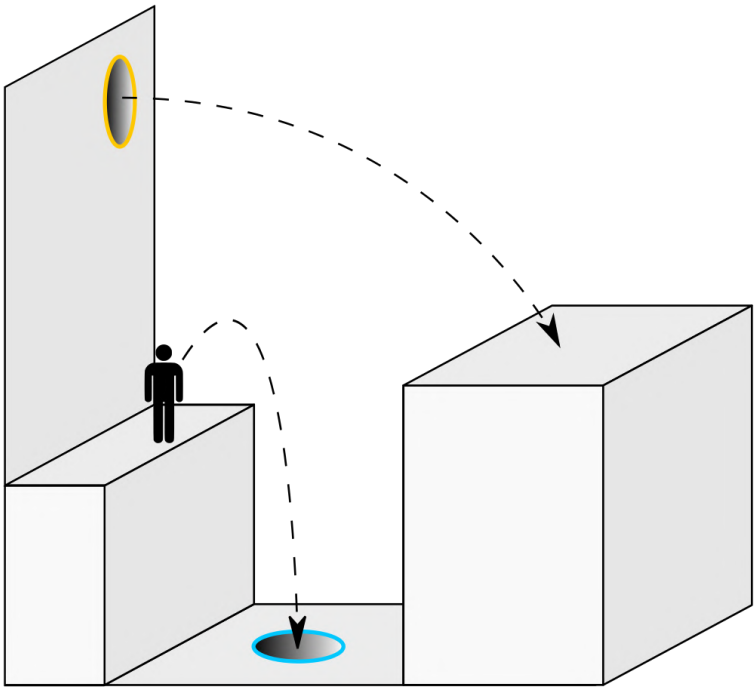


Figure 2.2: The mechanics of a Portal[Wik]

2.2.3 The Talos Principle

The Talos Principle[Cro], released in 2014 by Croteam, is another first-person puzzle game. The player takes the role of a robot with a human-like consciousness. The robot awakes and is instructed to explore the world and solve puzzles to collect sigils but is warned not to climb a tower in the center. As the player progresses through the game, he learns that the space he is in is a testing ground for new artificial intelligence entities. The robot should show intelligence by completing the puzzles and free will by disobeying the order to climb the tower. There are multiple endings to the game based on the decisions of the player. There are many puzzles accessible, so the player can choose the order in which he will complete them. There are puzzles where the player needs to activate light-based switches by using portable refractors, and there are also puzzles that pose a danger to the player. If the player dies, he is respawned at the beginning of the puzzle. The game was praised for the elements of philosophy in its story and the challenging puzzles.

Figure 2.3: The Talos Principle[Steb]

2.2.4 Archaica: The Path of Light

Archaica: The Path of Light[Two] is a puzzler with lasers and mirrors released in 2017 by TwoMammoths. The goal of each level is to illuminate the crystals. This can be done by placing objects on a grid and rotating them around. There are multiple objects that interact with the lasers in different manners. On each level, there are hidden objects that reveal more of the game's lore. It is an indie game created by two brothers. Although it is not a well-known game, those who played it left mostly positive reviews.

Figure 2.4: Archaica: The Path of Light[Stea]

2.2.5 Summary

Generally, all puzzle games have one thing in common. They start with simple levels, where the player learns how to use or interact with a particular object. Later, this interaction is tested in more complicated tasks. After a while, a new feature appears, and the players again learn how to use it. Then the tasks are composed of the new features combined with the old one. This process happens many times. It is easier for the player to grasp how to use and combine the mechanics when introduced this way. If the mechanics were introduced all at once, the player would be confused and not comprehend how to utilize them together. For most of the games, the story is a substantial part of the game experience.
All of the described games(Portal just if I count Portal 2 as well) work with the idea of light beams and the goal of navigating them to specific places. The games that uses the idea of changing colors of the beam is Aargon and Archaica. In my game, there are objects that can either add or remove a colour from a beam, where as in Aargon the colors are usually made by mixing or splitting the beams and in Archaica there are objects that just change the color. Aargon uses red, green, and blue as the basic colors, where as, I use red, blue, and yellow. I decided to use these colors, because I think it is easier to comprehend that yellow and blue gives green, instead of that green and red gives yellow. It should be easier, because as children people are thought, how to mix colors together. All the games but Archaica have the possibility for the player to die and when it happens the player is respawned at the start of the current puzzle.

2.3 Modular Components

Every 3D game is made out of models which together create the world of the game. One of the approaches for constructing game worlds is the usage of modular components. To describe what modular components are, we can look up the word "modular" in a dictionary. It means "consisting of separate parts or units that can be joined together"[Oxf10]. This description fits the concept of modular components well, as the idea behind these components is that separate objects are being used to form more complex scenes and environments. A complex scene can be compared to creations from the well-known toy Lego, where a person can use small plastic parts to construct a building, vehicle, or some other thing. The advantage of using modular components is that it is less demanding to create small pieces and then use them to build the scene rather than to make every single object separately. Additionally, it requires less memory to store them. There are some disadvantages as well. When

Figure 2.5: Modular dwarven kit from Skyrim[Burb]

the pieces are being used frequently, the scene might start to look repetitive and boring. To avoid repetitiveness, some decorative elements can be used. Moreover, there is the possibility to use a hero element. That is a component that is not modular and is specific for the place.
Using modular components in big projects can mean that instead of tens of artists, only a few are needed. These artists create modular kits, and then level designers use them to put the level together. For example, in the development of Skyrim, only two artists were required to create the modular kits[Bura]. The kit for dwarven dungeons can be seen in figure 2.5.

2.4 Physically Based Rendering

When creating components, it is necessary to create materials for them. Materials define how to draw the surface of the component. Unity uses Physically Based Rendering (PBR), a concept of trying to simulate lightning as close to the real-world lighting as possible. It is possible to simulate realistic lighting as the algorithms use the knowledge of how light behaves and how it interacts with materials in the real world. When talking about light behavior, we know that a light ray can be either reflected, refracted, absorbed, diffracted, or scattered[ASA]. When a ray hits the interface between two environments, it is reflected or refracted. The reflected ray bounces off at the same angle as it hits the surface. Refracted rays change direction based on the index of refraction. Absorption usually affects just part of the color spectrum, the energy of the ray is converted to some other kind of energy, usually heat. As a result of absorption, the color of the ray changes. Diffraction occurs when a ray hits an obstacle, and it bends around its corners. When a light ray changes direction inside the same environment, it is called scattering.
When we want to distinguish the light reflecting from a surface, we talk about specular and diffuse reflections. Specular reflection is the light that is reflected. Diffuse reflection is the scattered light that makes its way out

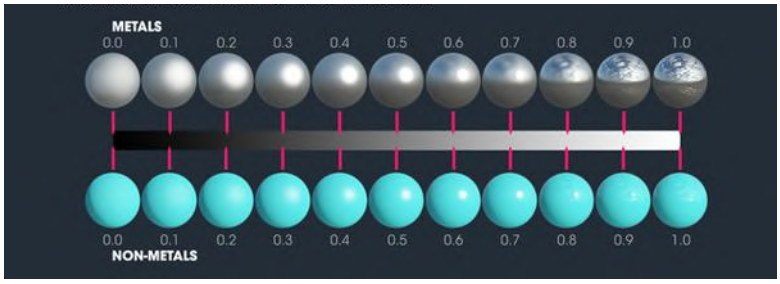


Figure 2.6: The effect of smoothness[Unib]

of the surface. The scattering is so chaotic that it can be said that the light comes out in all directions[Rus].
One of the most important concepts is the energy conservation concept. This ensures that an object can not reflect and refract more light than it receives unless it is an object which emits light on its own. It means that the more specular reflection a surface has, the less diffuse reflection occur.
Unity uses materials to set what color an object should have and how it should reflect the light. When creating materials, there are many parameters that can be set. These parameters can be set as one value for the whole material, or it is possible to use a map. A map is a texture, where for each texel, we can set a color. A color can be represented in four channels: red, green, blue, and alpha. The alpha channel is used for transparency. The texture is a function that can be used to represent a certain property of the surface of an object. The most used are 2D textures, which are raster images. Here are the parameters that Unity uses:

Albedo Represents the color of diffusely reflected light. In short, what color the object has. An example can be seen in the figure 2.7 on the left.

Metallic Shows if the material is a conductor and reflects the environment or an insulant that almost does not reflect the environment at all.

Smoothness Most surfaces are not completely smooth. There are very small imperfections, called microsurfaces, which are not visible by the eye. Even though they are so small, they have a notable effect on reflection. It is not possible to evaluate each microsurface as it would require a lot of computation and memory usage, so instead of describing each microsurface on its own, there is smoothness, a measure that represents how smooth the material is. With lower smoothness, the light will be scattered more evenly. If a map is used for the metallic trait, the smoothness is taken from the alpha channel. The effect of smoothness can be seen in figure 2.6.

Normal Map A much more detailed surface can be represented without

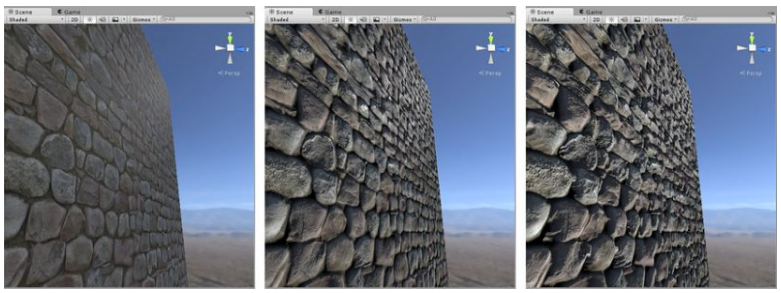


Figure 2.7: From left to right: object with just albedo, object with albedo and normal map, object with albedo, normal map and height map[Unia]

the need of adding extra geometry. It changes how the light is reflected from the surface, and by doing so, it can simulate bumps or crevices. A normal map is a texture, which stores normal vectors for each surface, where the RGB values of each texel represent the X, Y, and Z values of a direction vector. An example can be seen with the combination of an albedo in the middle of figure 2.7.

Height Map A way to simulate depth and height information based on the position of a camera. It uses shades of grey to represent data. It is easier to paint on in comparison to the normal map. A heightmap is usually used together with a normal map to make larger changes on the surface[Doc]. An example can be seen with the combination of a normal map and albedo on the right of figure 2.7.

Occlusion A black and white texture that represents places that would be shadowed by the rest of the surface. It is also known as the Ambient Occlusion map[Rom15].

Emission A way to make a surface emit light. A map can be used if only some parts of the surface should produce light.

These are the most important parameters of a material. There is some more option such as adding secondary maps or changing the tiling and offset.

2.5 UV mapping

To make it possible to apply the material to an object, the object has to have a UV map. UV mapping is a process that determines how a 2D texture can

be projected on a 3D object. The process consists of taking the 3D object and unwrapping it into the 2D space. The unwrapping could be described through an example of a box. If I would like to unwrap a box, I could take scissors, cut the folds, and lay it flat on the ground. Now, If I would look from the top, I would be able to see the box in just two dimensions[Bleb]. The letters "U" and "V" represent the axis in a 2D space as "X" and "Y" are already used for the axis in the 3D space. After the UV mapping is done, we can use any texture and project it to the 3D object based on the result of unwrapping, where each texel represents a point on the surface of the object in the 3D space.

2.6 Universal Render Pipeline

When playing a game, everything the player can see has to be rendered. Rendering is the process of drawing anything from the scene on the screen of a computer. A rendering pipeline is a sequence of steps that makes this possible.
Unity offers the possibility of choosing between three different pipelines. Built-in Render Pipeline, Universal Render Pipeline, and High Definition Render Pipeline. Unlike the Built-in renderer, the other two have more possibilities for customization. Universal Render Pipeline, shorter URP, offers an option to create custom shaders using the shader graph, which allows creating shaders by connecting nodes in a framework rather than writing a code. A shader is a script that contains algorithms for calculating the color of a rendered pixel based on the material and lightning input. The advantage of the shader graph is that a person can instantly see the changes, and it is easier to understand for users with low or no experience with shaders. URP is optimal for games that do not require the best possible graphics. For those who want better graphics, the High Definition Render Pipeline would be the choice, but it is more performance-heavy. It includes many features that URP currently does not have. URP is getting regular updates and should replace the Built-in Renderer Pipeline in the future.

Chapter 3

Design

In this chapter, I show parts of the game design document, mainly the idea of the game, the game elements, controls, story, and a detailed description of the levels.

3.1 Game Idea

The game is a puzzle game with some action features, but there is no combat. The main focus is the puzzles which involve changing the color of light beams and getting them to a certain location. This game should include a story where the player discovers the secrets of the temple and what happened to the old civilization.

3.2 Game Elements

The main feature of this game is the light beams, which have to be changed, blocked, or navigated to certain places. There are three basic colors of a beam: red, blue, and yellow. There are also combinations of these colors: orange(red and yellow), green (blue and yellow), purple(red and blue), and white(red, blue, and yellow). There are structures that can add or remove a color from the beam. For example, if a purple beam goes through a field

that adds yellow, it will become white. If it goes through a field with a color that it already has, nothing will happen. There are masks on the walls that have different eye colors corresponding to all possible colors. These masks are like triggers. They react if the right colored beam enters its mouth. One mask can open a door, and another mask can move a part of a wall. There are pillars with masks on them that will turn the light to the right, left, or change the height at which the beam currently is. The structures and pillars can usually be moved, but there are also some that cannot be moved at all. To avoid having all rooms filled by light beams, there are also rooms that pose a danger to the player. A room where parts of the ceiling drop and the player has to walk through it by finding safe spots and figuring out the ceiling's timers, or a corridor filled with lasers that disintegrates the player on touch.

3.3 Game Controls

To play this game, the player needs to use both a mouse and a keyboard. Navigation in menus is by a mouse. The player starts in walking mode. The direction the player is facing can be changed by the mouse and to move player uses the W, A, S, D controls. Pressing E while being near an object, which can be interacted with, causes different interactions. When the player interacts with a structure that changes colors or a pillar with masks, it changes the walk mode to pushing mode. The change will snap the current direction to one of the four global directions (right(1,0,0), left(-1,0,0), forward(0,0,1), backward(0,0,-1)) based on the position of the player according to the object. In the pushing mode, the player can move only forward or backward, and the movement speed is much lower. By pressing E again, the mode changes back to walking. By pressing ESC, the player can pause the game or stop the pause and return to the game.

3.4 Game Story

My parents were archeologists. They were the ones that discovered the temple devoted to the goddess Azshi. When they entered the temple, it rose from its slumber, and all the mystical things inside rose to life. Beams of light started to shine from masks on the walls, and strange structures with unknown energy fields appeared. They found that this place was a testing ground, where the goddess chose her avatar. Anyone could try to pass the tests, and only those worthy would succeed.

For many years, the temple was under lockdown, and only a few people chosen by the government could enter. Whenever a person returned from the testing ground, they would know that they failed, but they did not remember what happened inside. People tried to take records, but there was nothing on the recordings when they came back. Recently, the lockdown was lifted, and anyone could try the test. People from all over the world came, but everyone failed.
Today is the day. I am going inside to try my luck. Will I succeed where everyone failed?

3.5 Model Design

As the whole game is in an old temple, the player will be mostly inside. As there are unknown powers inside the testing grounds, there is a possibility to create levels in practically any environment. However, to make it easier to make, I will keep it in corridors beneath the ground. The source of light in these corridors will predominantly be the torches. There will be multiple different masks, which will interact with the light beams in different ways. One will be the place where the beams are coming from, and the other will be the one that unlocks a door. Another two masks will be on the turners. Each of these masks has to be unique so that the player can distinguish them(Figure 3.1). Furthermore, there will be two color changers, one of which will add aspects of light and the other one will remove them. These again have to be distinguishable.

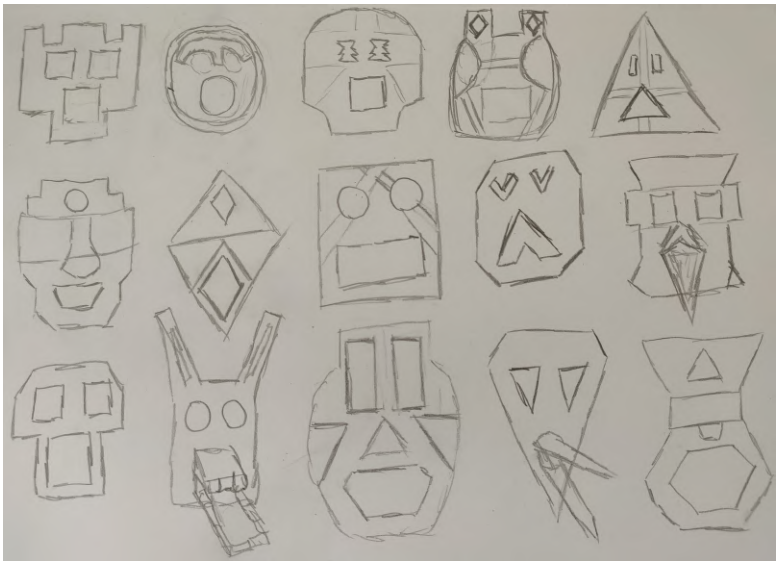


Figure 3.1: Sketches of masks

3.6 Level Design

Based on the analysis of other games, the goal was to make the game progressively more complex. That is why the first level is mainly just an introduction to the game mechanics, and it is trying to test if the player understands them well. Other levels are using and evolving all previously used mechanics.

3.6.1 Level One

In the first room, there are two beams, one is green, the other one is orange, two masks with yellow eyes, and two objects that have the same shape, but the color of the field is different. I will call these objects color changers. One of the fields is red, and the other one is blue. These color changers can be moved by the player in four directions. The goal of this room is to introduce the player to the color changer that removes aspects of the light.
It is easier to recognize these changes if the player focuses on the particles flying around the beams. By intersecting the lights, the red changer can change the orange beam to yellow, and the blue changer can change the green beam to yellow. If the red changer interacts with the green beam, nothing happens because there are no red particles in a green beam. The same applies to the blue changer and the orange beam. The second room introduces the second type of color changers. It has a different shape, but the color field looks the same. This time there are red and yellow beams, and the player needs purple and orange beams to pass. One of the changers is blue, the other one is red. Blue changer interacting with the red beam creates a purple beam, the red changer interacting with the yellow beam leads to an orange beam. The third room puts the player's knowledge to the test. There are several changers of many colors and two beams going straight to masks. On the right side, the beam is blue, but the mask requires a yellow color. The left beam is orange, and the eyes of the left mask are blue. To successfully pass, the player needs to remove orange and yellow from the left beam and add blue. However, it is important to add blue before removing the other colors, as if a beam loses every color, it does not go through the object. On the right side, it is necessary to add yellow and remove blue.
The game could become repetitive if everything were just about the light beams. That is why the next room is filled with falling ceilings, which can crush the player if he gets hit. Every ceiling has a different timer, so by watching them fall, the player can figure out when it is safe to pass.
The fourth room is there to introduce a new mechanic, an object that can turn the beams in a certain direction. I will refer to it as a turner. To make

it easier to interact with it, there is a crystal, which begins to glow when the light is going through. In this room, only two turners have to be put in the right places to get the beam to the mask. After the introduction is done, there is a room full of the turners. The task is to get two beams from one side of the room to the masks on the other side to unlock the door. In this room, the player should learn that the beams will not go through each other when they collide. Later this knowledge is necessary to solve another room. The second corridor, full of falling ceilings, awaits the player. This time there are no gaps between the crushers. The way to go through it is to move one crusher forward each time they go up. There is one more room with beams. It is more complicated as there are color changers as well as the turners. In addition, it is necessary to block one beam with another one to get a third beam through. The solution for this room can be seen on figure3.2.
Another danger is introduced. There are lasers that can disintegrate the player. When the laser is off, the mask's eyes, from which the laser is coming, are flashing, so it is possible to count how many flashes there are before the laser turns on again. The first set requires finding the moment to enter between the beams and continue through them unharmed. It is possible to pass this set by walking just in one direction. The second set requires the player to walk through a corridor and, meanwhile, to go from side to side to avoid being hit. After passing the lasers, there is one final room with a button that, after pressing, activates an elevator which takes the player to the next level.

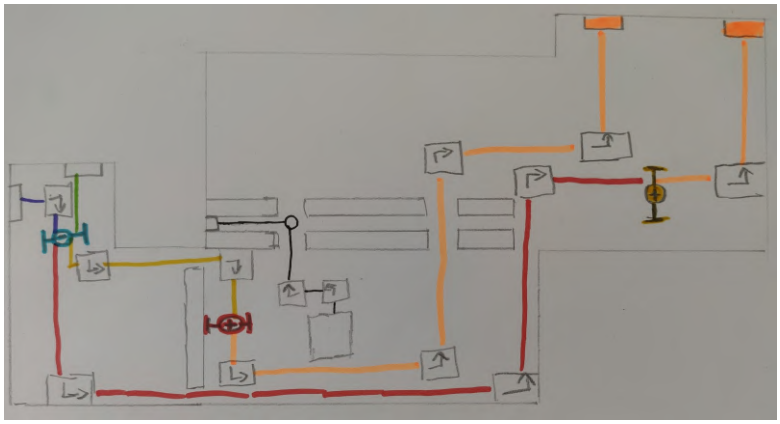


Figure 3.2: Level one, room nine

3.6.2 Level Two

This level starts by introducing a new mechanic, where the beams can now be in different heights. The turner that was previously used just to turn the beam can now also change its altitude. The first room shows that there are three main heights, and the player has to lower the beam two times to get it

under other beams. There are four different beams in the second room, and the player has to navigate them to the masks on the other side of the room. The solution for this room can be seen in figure 3.3. The squares with arrows represent turners, and the numbers represent at which height the beam is.
To solve the next room, the player needs to get a beam through a small gap in the wall. However, the beam will be blocked by a second beam. It is necessary to use a third beam to block the second one, to let the first one pass. Furthermore, the player needs to remove yellow and add blue aspects to the beam on the way.
The next corridor is filled with crushers with different sequences from the first level. This time, there are some more challenging places where the player needs to choose the timing accurately.
After the crushers, there is a room on the same base as the second one. One difference is that there are unmovable pillars that can also bend and change a beam's altitude. The player has to solve how to make the light beams pass through. A good strategy here is to pick a color, get it to the mask, and repeat this process until the room is solved. The solution for this room can be seen in figure 3.4. The squares with arrows represent turners, those with circles represent unmovable turners. The numbers represent at which height the beam is. The last room of the level is once again filled with lasers. This time it is a bit harder to pass as there are places where lasers are coming from more than one side.

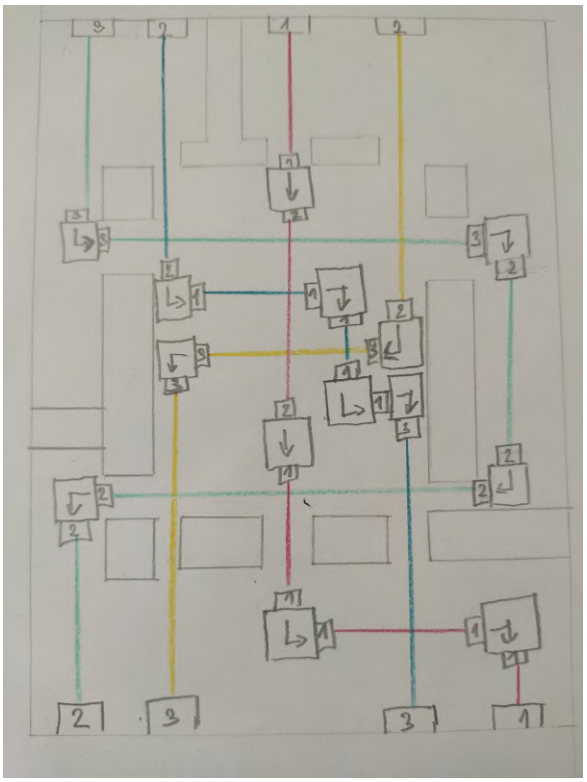


Figure 3.3: Second room in second level

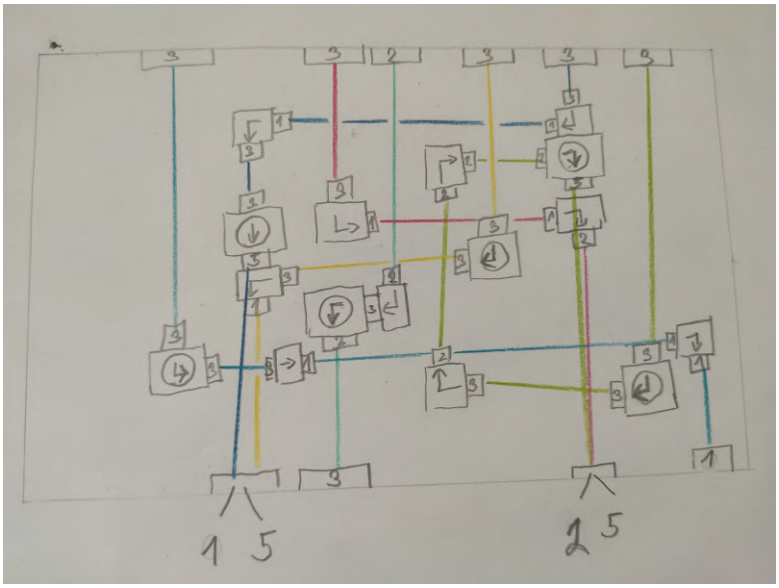


Figure 3.4: Fifth room in second level

3.6.3 Level Three

The player starts at one side of a giant maze(Figure 3.5). There are two possible ways to get out of the maze. The better one is finding out that there are barely visible arrows on crossings. By following the arrows, the player can get out quite easily. However, if the player does not find out that there are these arrows, it might take a while as it is a really big maze. When the player gets out of the maze, he finds himself in a room with three doors. One of the doors is blocked by two differently colored fields. The other two doors are accessible. The player needs to complete the puzzles in each of these doors to get rid of the fields and be able to open the last door. On one side, there are beams, and on the other side there are lasers. In the first room with lasers, the player learns that he can turn them on and off by using a lever. In addition, he finds out that a laser can collide with another laser. The second room with lasers is a bit harder. The player needs to figure out the order of switching the levers on and off to access a button. When he presses the button, a cut scene appears where he can see that one of the fields disappears. In the rooms with the beams, the player guides the beam to a new mask. When this mask is receiving a light beam of the correct color, a part of the wall opens and shows a hidden part of the room. There is a turner inside, thanks to which the player can open the next door. The next room is a little more complicated. The wall is blocking a blue and an orange beam. There are two masks near the door that require green and red colors. Furthermore, there is a green beam, three turners, and a yellow light remover. There is the new mask in the corner that requires a blue beam. The idea is to first

make the green beam blue and turn the blue beam into the new mask. This will move the wall that was previously blocking the two beams. After that, it is necessary to quickly remove the turner and let the blue beam hit the blue mask before the wall moves back again. Finally, the player has to get the green beam to the green mask and use the yellow remover to make a red beam from the orange one.
The last room might be hard for someone. In this room, there is no color changing, and all masks cause parts of the wall to move. Everything requires red beams. First, there are two turners by which the player can open the first hidden alcove. There is another beam and another mask, and a new turner. Now the player needs to use the new beam to hold the first mask activated and the first beam to activate the new mask. Another alcove opens with again one beam, one mask, and a turner. This time the player needs to use the third beam to activate the first mask and use the second beam to activate the third mask. When the third mask is activated, it opens the last alcove, where the button is hidden. When the player presses it, another cut scene is played, and the second force field disappears. Now the player can return to the main room and enter the previously barricaded door. In there is the exit of the third level.

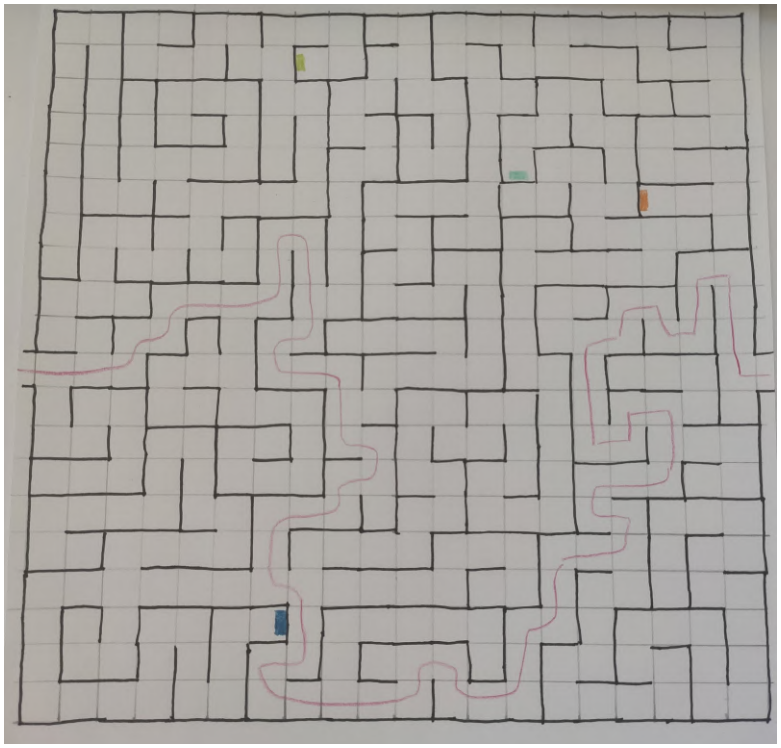


Figure 3.5: The maze

3.7 User Interface

When the game starts, there are three buttons: Start, which lets the player choose a level and then starts the game. Options, which brings the player to an options menu where he can set the volume and mouse sensibility. And Quit, which closes the application.
There is another menu that can be accessed during the game by pressing ESC. The player can resume the game, go back to the main menu, access the options menu, access a simple help, where is the basic information about the light beams, restart the room he is in. This can be useful when the player gets stuck or wants to start the puzzle from the beginning. There is also an exit button to close the application.

Chapter 4

Creation of the game

There are many things that have to be done to turn the game design document into a playable game. I needed models, textures, sounds, and, most importantly, scripts for the game's logic.

4.1 Models

I used Blender[Blea] for the creation of the game objects. The reasons for choosing it were previous experience, availability of many tutorials, and that it is a free software. I started by creating a wall, a simple plain with some added edge loops to create rows for stones. A cube scaled to fit between the two edges was an easy start to make the stones, but the bricks could not be just simple cubes. After multiple subdivisions and the application of a subdivision surface, the cube looked much better. However, it still missed details, and creating a wall by using only one same rock would not look nice. By using multiple different brushes in the Sculpt mode, I made eight various stones.
The process of making the entire wall consisted of duplicating the stones, rotating them, and moving them to fill the rows. Every other row has the stones offset, so there are no visible vertical lines. This resulted in having some bricks that were overreaching the edges of the plane. On the rows where the stones were overreaching, it was necessary to use the same rock as the first and last one, so later, when using the same wall, it would tile seamlessly. When the wall was constructed, I created a single object by joining all the

bricks together, a high-poly wall(Figure 4.1).

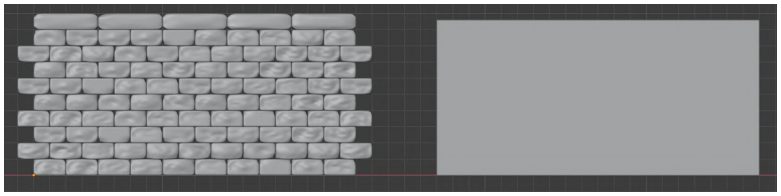
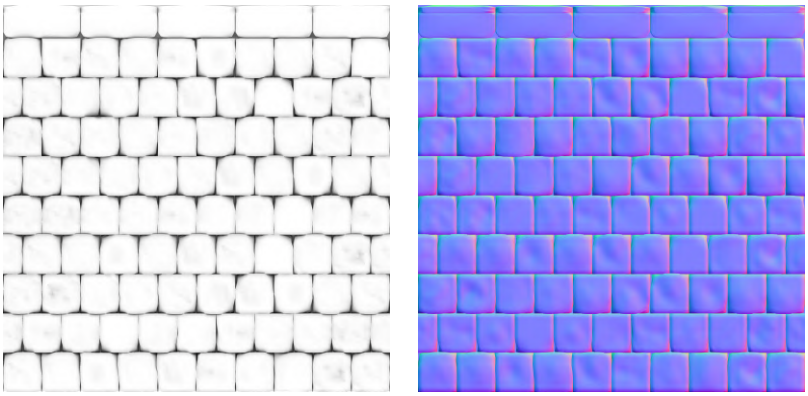


Figure 4.1: High-poly and low-poly model of wall

High-poly means that a model consists of a massive number of polygons. It would not be a problem to use high-poly objects to create some animated videos or photorealistic images. However, computer games should use models with a lower number of polygons, as it requires less performance to render them. It is possible to get the details of the high-poly model to the low-poly one by a normal map.
Before the creation of a normal map, the object must have a UV map. By using the smart UV project function, I created the UV map for the wall. With the UV map ready, I could start the process of creating a normal map. A normal map can be baked. Baking is a process of calculating specific data and saving them into a texture. Usage of a cage can help with the baking, where instead of using the mesh normals, it uses the normals of the cage. It can prevent glitches that would appear on the edges of the object. It is possible to make more maps than just a normal map. To create the albedo texture, I also generated an Ambient Occlusion map(Figure 4.2).
In Gimp (free software for editing images)[Gim], I used the texture from the Ambient Occlusion bake and multiplied it with a rock texture that I acquired from a portal with free images[Texb] To create some differences between each brick, I added a new layer of soft light and painted most of the brick in some color. The bricks that were just partially on the texture (those that were overlapping the edges) were not painted. So when I would create other objects with the same pattern, there would be no seams when they snap together(Figure 4.3).
The first wall I created was eight units long(Figure 4.4), and I needed smaller walls to be able to make corridors of any size. It was possible to repeat the same process for each wall, but as the textures were quite big (4098x4098), I decided to reuse the same texture. A new edge loop had to be created to be able to reuse the texture, as it was necessary that the left side of the wall had the left side of the texture and the right side of the wall the right side of the texture, but it was not possible to fit the whole texture on the smaller wall as it would not look good. Therefore, instead of using the whole texture, I just used some parts of it.
The corner piece and door frame have their own textures. The process of creating them was pretty much the same. It is important to have the row's

height coordinates exactly the same as the coordinates of the walls and that the height coordinates on both UV maps match because if these coordinates are not the same, there will be seams when these two objects snap together. When the walls were done, I created a plane that would represent the floor and the ceiling. Additionally, I created the model for a torch and all the objects I needed for my puzzles.



(a) Ambient map (b) Normal map

Figure 4.2: Baked maps of the wall

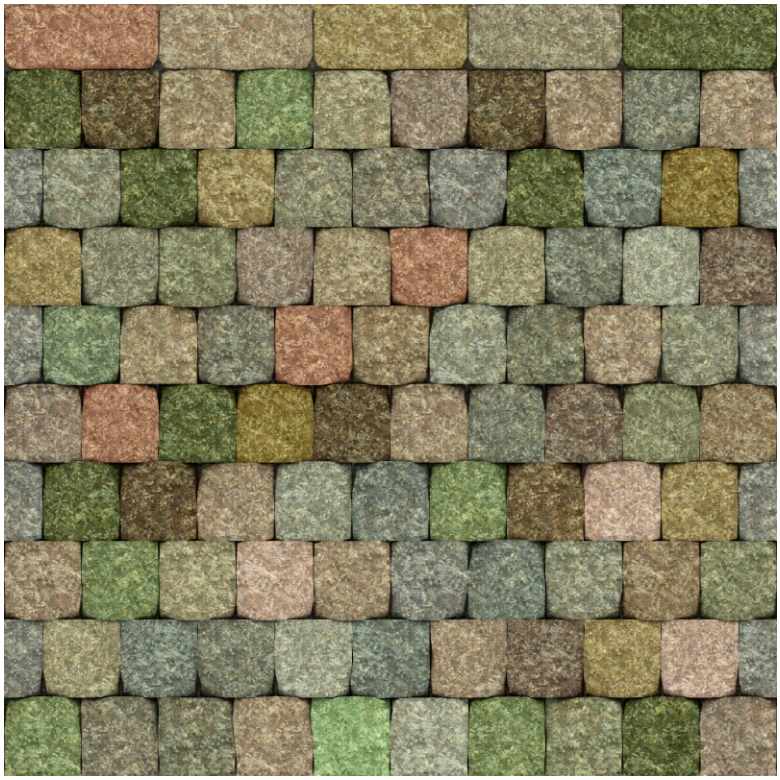


Figure 4.3: New texture for the wall

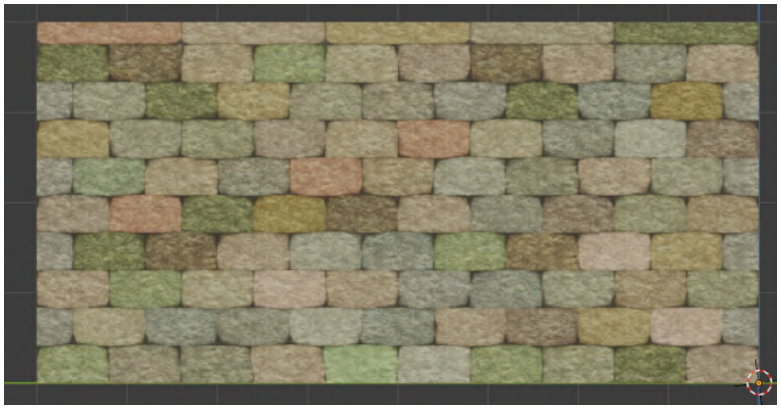
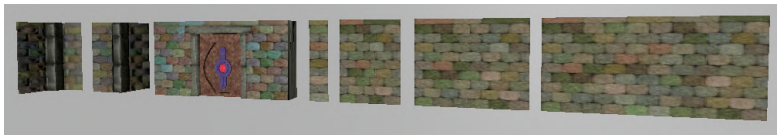
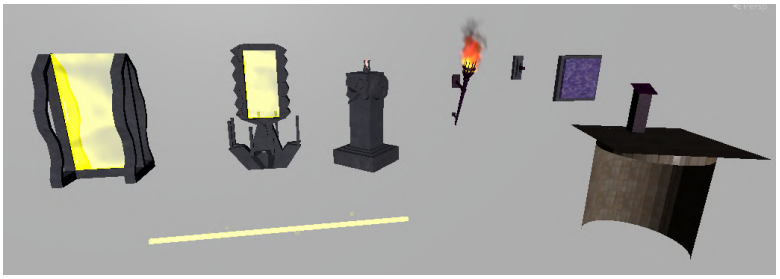


Figure 4.4: Low poly model with applied texture



(a) Walls

(b) Masks

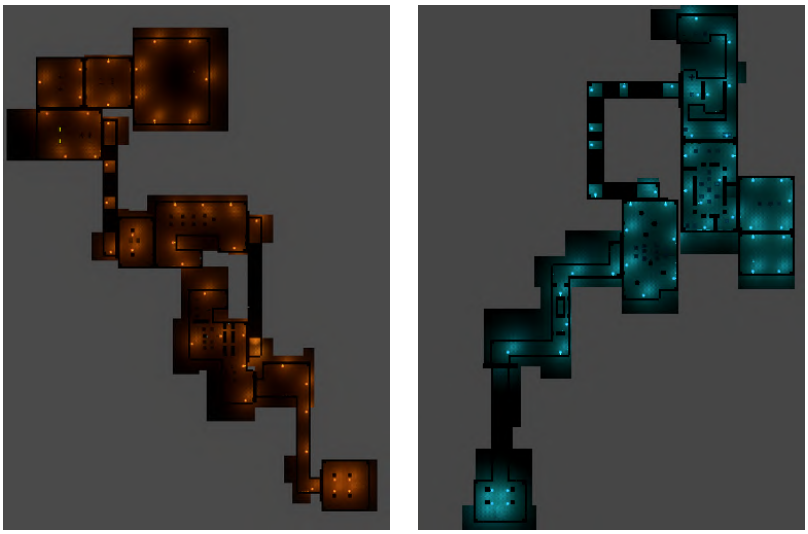


(c) From left to right: color remover, color adder, turner, torch, lever, button, end of level button, and at the bottom light beam

Figure 4.5: Models

4.2 Level Creation

When all the components were created, I imported them into Unity as assets. In there, I created prefabs. Prefab is a reusable asset that stores all the components of a GameObject, its values, and even its children. This is essential for modular components, as when there is a need for a change, changing the prefab will change all the objects created from this prefab. I added colliders to the walls, floors, and others assets that would be moved. Colliders are bounds that can be used to detect collisions between objects. Additionally, I created materials using the maps I created or downloaded[Texa] and applied them to the prefabs. After that, I started creating rooms based on my design by simply dragging the prefabs into the scene view. I used grid snapping to move the objects in the scene. This allowed to precisely place the walls next to each other. When the walls were done, I placed the floor, duplicated it, rotated it around the x-axis by 180 degrees, and moved it up on the y-axis. Then I placed torches so that the room would be well lit. After that, I created puzzles from the rest of the objects based on my design. There is a maze on the third level, and for that, I could not use torches because there would be too many point lights, and the performance would drop drastically. Thus, instead, I created a light that floats above the player when he is inside the maze. When this was done, I started working on the scripts.



(a) Level one (b) Level two

Figure 4.6: Images of levels one and two

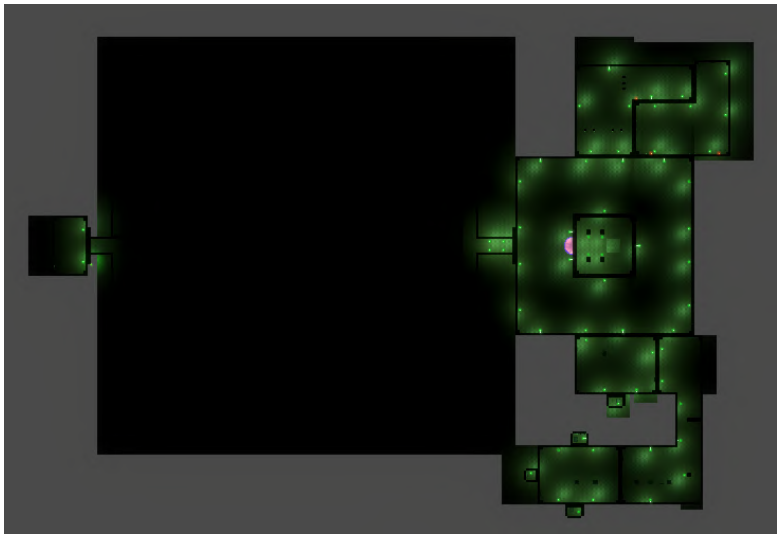


Figure 4.7: Level three

4.3 Scripts

Everything in the game, from moving to defining how objects interact with each other, is done through scripts. Scripts are text documents that contain a code that represents the logic of the game. Unity uses C# as the programming language. Here I will describe the more complicated logic and how I implemented it.

4.3.1 Movement and player interactions

The first thing I implemented was player movement. Because I decided that the game will be from a first-person view, I had created two classes, one for the camera, which took care of rotation. Rotating up and down rotates just the camera, but rotating left and right also rotates the body unless the player is pushing something. When the player is pushing, the rotation left and right affects only the camera, but it is limited. This allows the player to look to the side while pushing. The other class is called Movement, and it processes the keyboard input and uses it to make the player move, check for triggers, and initiate actions with other objects. The player can move to the left, right, forward, or backward based on the direction of the forward vector. When the player is pushing an object, the movement speed is lower, and he can move only forward and backward. To make the game easily expandable, the logic

of the interaction is not in the player script but in the script of the object instead. I used a raycast to detect which object is in front of the player and check if it has a script derived from an abstract class Interactive. If it does, it will call the Interact method. Each object can have it implemented differently, so if a new interactive object is added to the game, it just needs to derive from the Interactive class.
Currently, there are several objects the player can interact with. There are passive interactions and active interactions. Passive interactions are those that happen even without the consent of the player and do not use the Interactive class. The laser and crusher(falling ceiling) are able to kill the player whenever he touches the deadly parts. The walls could be counted in passive interactions, as they prohibit the player from passing through them. On the other hand, active interactions happen when the player chooses to do so, those are the objects he can interact with and are using the scripts derived from the Interactive abstract class. Currently, those are the object he can move, a lever, and a button. For the moveable objects, the Interact method snaps the player to a specific location based on the player's angle when he starts the interaction with the object. Then the object saves its formal parent and sets the player as its parent. In addition, it activates the pushing mode. For the lever, The Interact method plays the animation to make it go up or down and calls the Switch method of the controlled object. The first time the player interacts with a button, an animation is played.

4.3.2 Beam of light and its interactions

The other important thing is the light beam. To make the beam length precise, I used a raycast and used the distance as a scale in the direction of the beam. Instead of using collision to make the interactions of the beam with other objects, I used the raycast to implement the same thing I did with the player. This again should allow easy expansion. There is an abstract class, BeamInteractive, and it is working on the same principles as the Interactive class. When the ray hits an object for the first time, it calls the BeamEnter method, every time it hits the same object again, it calls the BeamStay method. When it hits another object or when it becomes disabled, or when it is destroyed, it calls the BeamExit method. Each beam has three particle systems, one for each of the three basic colors(red, blue, and yellow). Whenever a beam's color is set, the particle systems are disabled or enabled based on the color.
Throughout all the scripts, the color is set by a string. To convert the string to a color, a dictionary from a static class, where all the information is, is used. The class also includes methods that decide what will happen when a color is added or removed from the beam.
There are many objects that are interacting with the light beam, and even

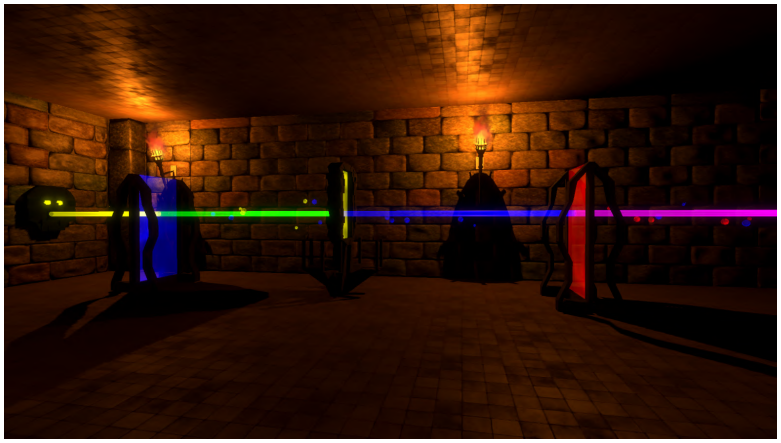


Figure 4.8: Interaction of a beam with the color changers

the light beam can interact with another light beam. When two light beams collide, a small sphere appears to represent a collision. There are two objects that can add or remove colors from the light(Figure 4.8). They both use the same script. To determine which is which there is a boolean. The appropriate method is called based on the boolean, and a new color for a beam is determined. It is not possible to change the color of the beam and let it pass through with the new color, so instead, a new beam is created, and it starts where the incoming beam ended. The position is counted based on the forward vector of the incoming beam. If the forward vector has its z value one or minus one, the new beam uses the x and y incoming beam position values and the z value from the changer's position. If the forward vector has its x value one or minus one, the new beam uses z and y values from the incoming beam and the x value from the changer's position instead. The changers hold a list of all the incoming beams so more than one beam can go through. Every time the BeamStay method is called, the position of the new beam is recalculated. When the BeamExit is called, the new beam is destroyed.

Turner is working on a similar concept, but it only takes one beam at a time, so there is no need for a list, nor is there a need for instantiating new beams. There is a prepared beam in the mouth of the exit mask, and it is turned active when the BeamEnter method is called. Furthermore, the color changes to the same as the incoming one. There is no need for the BeamStay function as the local position of the turned beam is set, and it is not changing. The BeamExit method just deactivates the new beam. The turner is also used for changing the altitude of beams. This is done by changing position of the exit mask(Figure 4.9)

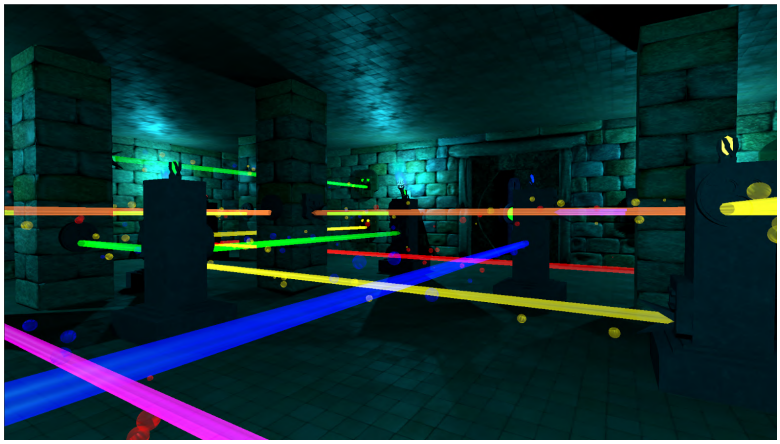


Figure 4.9: Beams with different altitudes

There are masks that check if a certain color is entering their mouth. When the BeamEnter method is called, it checks if the color of the incoming beam is the same one as the color it requires. If it matches, the emissive index of the eye changes, so the player can see that it is hit. When the beam leaves, it changes the index back.
I had some problems with the interaction of two beams. When they were facing each other, the raycast would hit a different place every update call as the scale would change every update call. To resolve this, I added a small delay, where whenever a raycast is made, there is a short pause of a random length before another raycast is cast. This resolved in the desynchronization of the raycasts, so now the collision position stays in one place.

4.3.3 Game functionalities

To achieve better performance, I implemented a script called Game, which deactivates everything except the room where the player is or two rooms when the door is open. I am using point lights that cast shadows, so if I would not do this, the performance would drop heavily, as counting the lighting is performance heavy. When an object is deactivated, it is still contained in the memory, but all the scripts and components are deactivated as well, so Unity is neither calling Update methods for deactivated objects nor are they rendered. All the objects, but the player and doors, are sorted into rooms, and when the game starts, the game scripts find all the rooms and doors and add them to two lists, one for each, respectively. It also sets the first room as the active one. Whenever a door closes, a method is called that disables all rooms and doors which the player can not see. The script for the door is closely tied to the Game class. Each door holds a reference to the rooms

that it connects, and when a player is at a certain distance from it, it will first check if all the masks are receiving the right color. If they do, it will set these rooms to active and call the method to activate the doors in the next room. While the door is open, every update, the game also calls a script from the Movement class, where it checks in which room the player currently is. The Game script is also taking care of resetting the game when the player dies. It finds a checkpoint in the current room and gets the new position and rotation for the player. While testing, the FPS(frames per second) never dropped under 60.

4.4 User interface

The game starts with a simple menu(Figure 4.10). There are three buttons. A Play button, which lets the player choose a level. An options button that leads to an options menu where the player can change the mouse's sensitivity and the volume of sounds and a quit button that exits the application. A mouse is used to navigate the menu.
When the player is playing some level, he can pause the game and access the pause menu(Figure 4.11) by pressing ESC. Here the player can resume the game by the button or by using ESC again, go to the menu, go to the options, access help, reset the current room, or quit the game.

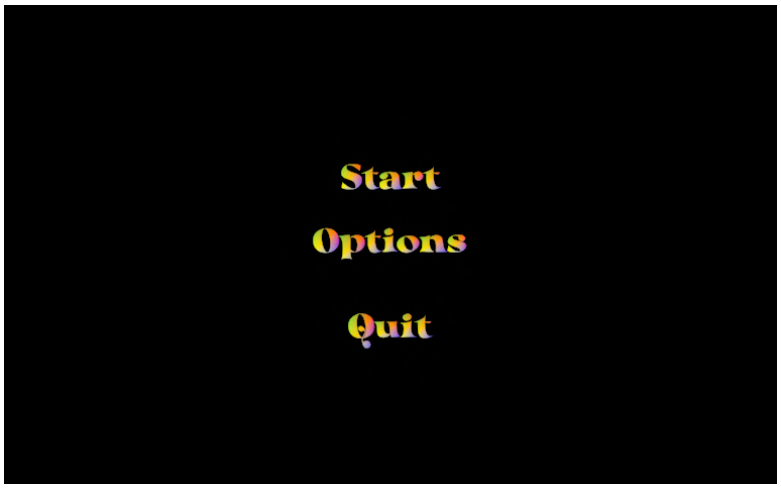


Figure 4.10: Main menu

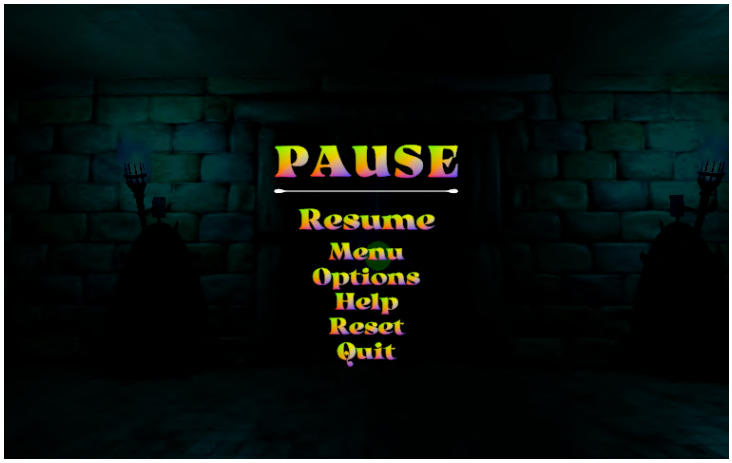


Figure 4.11: Pause menu

4.5 Sounds and Music

Every game needs background music and some sound effects. I created the background music with Waveform 11(a free digital audio workstation)[Tra]. I used the sforzando plugin[Plo] to be able to use the Virtual playing orchestra[Orc] .sfz files, which can be used for free. I created a minute-long track that will be looped in the background.
I made the sound effects in Audacity(free audio software)[Aud]. I recorded the sounds with my microphone and then applied some effects that are in Audacity to create suitable sounds for my game. The quality of my microphone is not as good as I would like it to be, so the sounds are not perfect. To create the door opening and closing sounds, I rubbed two stones with a rough surface. For the sound of pulling or pushing an object, I used one stone with smooth and one with a rough surface. To create fire crackling, I used a small plastic bag, and for the sound of the crushers, I used a drum. First, I had to get rid of the noise from all of these recordings, I sampled the part where only the noise could be heard and applied noise reduction to the whole clip. After that, I changed the pitch and speed of the recordings until I was happy with the sound. For the torch and pulling sound, I had to make the sound seamless, so the player cannot hear when the sound begins to play again from the start. I cut the beginning of the recording and put it at the end of the clip. Then I applied the crossfade effect.
When the sounds were ready, I imported them and the background music to Unity. As the background music is playing all the time, I keep it on an object that is not destroyed when a scene changes. I created Audio Sources for the objects that are making the sounds and set when should they be played in the scripts.

Chapter 5

Testing

Testing is a crucial part of the process of developing a game. It gives the developer important information on how to progress. When another person tries the game for the first time, he does not know anything about it. That is perfect as the developer can see if the implemented system is working, if the tester understands everything, and if the game is intuitive. The developer can think that everything is easy, makes sense, and people will not have any problems while playing, but sometimes the opposite is the reality. The first few tests were conducted to try out if the idea of the game is interesting enough to make the players enjoy the game. One or two testers would try the game, and I would watch them play it. Then they would tell me their opinion.

5.1 First Test

The first time I asked someone to try out my game, it was filled just with cubes and planes(Figure 5.1). There were no textures, no animations, no sounds; just the basic idea of the game was implemented. Furthermore, the game was quite different from its current state. There were no light beams, and the idea was just to pass through obstacles, move pillars to buttons, or block lasers with them. I wanted to determine if the direction I was taking was the right one. To my surprise, nobody really liked it. Everyone said that there was nothing interesting about it. The whole gameplay was walking through the same kind of traps the entire time. And furthermore, none of them thought that it is a logic game.

This test resulted in me redesigning the whole game. I still stuck with my idea of exploring a temple full of puzzles and traps, but I took a different approach. I needed something that would catch the eye of potential players. And that is when I first came to the idea of the light beams. I deleted almost everything in my design document and started anew. I salvaged some of the code and reused it in the new version, but most of it was not very useful for my next try.

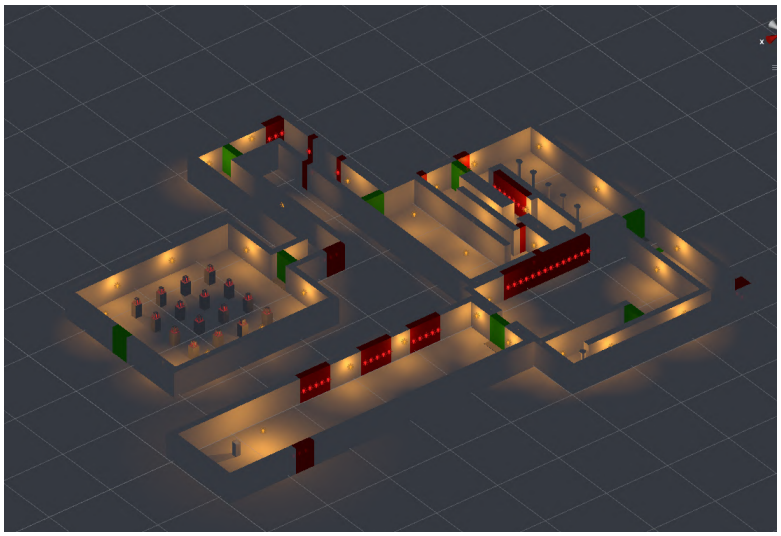


Figure 5.1: The look of the game during the first test

5.2 Second Test

The second round of testing came after a few weeks. I wanted to test if the light beams would have any success. This time, the testers liked the idea, but the logic of changing the colors was strange, and they did not understand it. At that time, I had just one structure that would change the color based on a table that I created and thought was logical. The tests showed that no one could tell which color will emerge unless they tried all the combinations before.

5.3 Third Test

I concluded the third test when I reworked the mechanic of changing the light. At this time, there were two color changers as in the current version,

where one would add, and one would remove colors from the beam. There were people who did not have any problems, but some people did not know the combinations of colors. For example, they did not know that orange color can be created by mixing red and yellow.
While the light beams were interesting, the testers did not like that there were only beams. "I would expect some danger when going into an old temple, not just walking around and solving puzzles," one of them said. However, overall, this test was a success. The reviews were better, and I had the main feature of my game. I did not want people to struggle with learning what the colors are made of, so I came up with particles that are flying around the beams(Figure 5.2) to help those who would have problems with color mixing. In addition, after hearing that the traps would be appreciated, I started working on them as well.

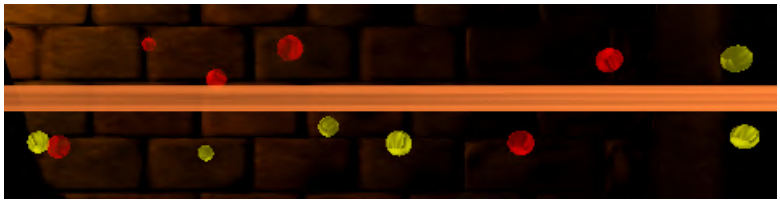


Figure 5.2: Beam particles

5.4 Fourth Test

This time I had an entire level done. I had most of my textures done, the models were ready, and I did not know about any bugs. I had seven testers this time, and I watched each of them play through it and asked them to speak their thoughts out while playing. Some of the testers were people who play computer games often. Some of them do rarely or not at all. Also, two of them already had some experience from previous tests.
During the test, it was really easy to see the differences between players and no-players. Even though there was a tutorial, some of them needed a little help to understand the concept of the beams as the introduction was too brief. After the little help, all of them were able to understand the mechanics and enjoyed it. All but one of them were able to pass the traps. I had to create a new version of the game where the traps were not working so the rest of the rooms could be tested by this individual. However, the tester did know how to pass the obstacles. He just was not capable of moving the character precisely. This tester was one of those who do not play computer games that often. In this version, some of the rooms on the first level were quite hard. And some of the testers struggled, so I expected bad reviews. The expected time to finish the first level was about 30 minutes. The faster players finished in around 40 minutes, where the slower ones played it for over an hour.

Unexpectedly, everyone said they had enjoyed the experience and did not find anything they would not like. Some of them preferred the light beams, others the traps, and all agreed that they were combined well throughout the level. The two testers from previous tests did like the new particles around the beams and agreed that they make it easier to understand.
However, I did not like that they struggled in the first level, so I decided to lower the difficulty. I wanted to make the beginning better, so people would grasp the idea faster and without problems. In addition, I saw that there were some things that I could improve. For example, people could not say if the beam was going through, while pushing the turner from some sides. That led to the addition of a small crystal on the top that begins to shine when a beam is going through.
During the playtesting, a few bugs were discovered. Sometimes when pushing an object, it would get stuck between two colliders on the floor, or when pushing near a wall, it would get stuck between two walls. I found out that the colliders of different floor game objects were not at the same height, and for the walls, they did not have the same width and length. I remade the colliders, but still sometimes it would get stuck. After some research, I found out that it were a ghost collision. A rigid body that uses continuous collision detection can cause this. By lowering the default contact offset in the project setting, I was able to get rid of it.
Another bug that was discovered during the test was that the sphere that appears when beams collide would not be destroyed. The implementation was not good, as it would instantiate and destroy the object on every new collision when it was not possible to have more spheres at the same time. Now there is just one sphere that just deactivates when it should not be visible.
One of the testers argued that in one of the rooms, a mask that moves a wall when hit by a beam of the right color looks the same as the masks which open the door. It was a valid complaint, and I created a new type of mask(Figure 5.3).

Figure 5.3: New mask

5.5 Final Test

In the final test, all levels were ready. I asked the testers to play all of them, as I made a lot of changes to the first one. This time I had only one tester who did not test the game previously. He did not have any trouble comprehending how the beams are working. All testers agreed that the difficulty of the first level is adequate.
The second level brings the possibility of having beams with different altitudes and the testers liked it. Few of them did not like that in the second room there is not that much space to push the turners around. None of them had any problems solving the puzzles. One of the testers had problems with the crushers as in previous tests, so I created a version without them for him.
The last level starts with a maze. None of the testers found the arrows on the walls which could lead them through. Almost everyone used the rule of the left or right hand. After showing them the barely visible arrows, everyone said that they would not find them, if they would not know about them. I created a new script for the arrows and now, after being in the maze for one minute, every twenty seconds the arrows become more visible. For the rest of the rooms there were no bigger problems, although the last one in the beam section was a bit too hard for someone. However, because it is the last level, I decided to keep it there. None of the testers had a problem to comprehend what the new mask that moves the walls does, neither how the lever that can turn the laser on or off works. Two testers even said that from the whole level they enjoyed the levers the most.
Overall the game was received positively. All of them found the game enjoyable, although sometimes a bit frustrating(when the last set of crushers would kill them for the third time).
There were two big bugs that the testers found. One of them was that every time the player died in a room with laser masks in levels one and two. The set of invokes that are turning lasers on and off would be initialized again, but the previous invokes would not stop, so for each time the player died a new set of invokes started. This bug was easily fixed by turning the old invokes off. The second bug they found was that a cut scene would not play in the third level after pressing the button. Unexpectedly, the cut scene did not work just for one of the two buttons. This took me quite a while to fix. The reason for it happening was that getComponentInParent does not search in disabled objects. However, getComponentsInParent can search in disabled objects, so I used it instead.

Chapter 6

Conclusions

The goal of this work was to design a 3D logic game based on the analysis of logic game principles, create modular components, create three playable levels, and test it on at least six players.
I started with the design document. Then I created modular components for some of which I also created textures. I then used the components to construct levels based on the design and implemented all the game's mechanics. I tried to implement them in a way that would allow easy expansion. I concluded several tests and based on the results changed the game appropriately. I also created some sounds and background music for my game.
For the future, I plan to add a third dimension to the movement of the player by introducing stairs or platforms. I would also like to model the character and animate it so the hands can be seen all the time and the interactions with objects are more visible. Also, I would like to add some voice lines for the main character and maybe for the goddess. Unfortunately, the game lacks a story at the moment, which I think is an important part of the game, so I will have to work on it and fit it into the game.
The work on this game improved my skills a lot. I have tried many things that I have not done before, from creating textures to composing background music. I will definitely continue my work on this game, and hopefully, one day, I will publish it.

Appendix A

Bibliography

[ASA] National Aeronautics and Science Mission Directorate Space Administration, *Wave behaviors*, http://science.nasa.gov/ems/03_behaviors, ([Online] Accessed 30/04/2021).

[Aud] Audacity, *Audacity*, https://www.audacityteam.org/, ([Online] Accessed 17/05/2021).

[Bet] Bethesda, *The elder scrolls v: Skyrim*, https://elderscrolls.bethesda.net/en/skyrim, ([Online] Accessed 13/05/2021).

[Blea] Blender, *Blender*, https://www.blender.org/, ([Online] Accessed 17/05/2021).

[Bleb] _____, *Uv editor introduction*, https://docs.blender.org/manual/en/latest/editors/uv/introduction.html, ([Online] Accessed 30/04/2021).

[Bli] Blizzard, *World of warcraft*, https://worldofwarcraft.com/en-us/, ([Online] Accessed 13/05/2021).

[Bura] Joel Burgess, *Skyrim's modular approach to level design*, http://blog.joelburgess.com/2013/04/skyrims-modular-level-design-gdc-2013.html, ([Online] Accessed 14/05/2021).

[Burb] Joel Burgress, *Image of the dwarven modular kit*, http://2.bp.blogspot.com/-q7M9ZyKUWpI/UXAcJT3pSUI/AAAAAAAAAak/FvXRSHtOsik/s640/DwarvenKitShot.png, ([Online] Accessed 17/05/2021).

[Cro] Croteam, *The talos principle*, https://store.steampowered.com/app/257510/The_Talos_Principle/, ([Online] Accessed 14/05/2021).

[Doc] Unity Documentation, *Heightmap*, https://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html, ([Online] Accessed 14/05/2021).

[Gam] Twilight Games, *Aargon*, http://www.twilightgames.com/, ([Online] Accessed 13/05/2021).

[Gim] Gimp, *Gimp*, https://www.gimp.org/, ([Online] Accessed 17/05/2021).

[Kin] King.com, *Candy crush saga online*, https://www.king.com/game/candycrush, ([Online] Accessed 13/05/2021).

[Mit12] B. L. Mitchell, *Game design essentials*, John Wiley & Sons, 2012.

[Moba] Mobygames, *Aargon, image from the game*, https://www.mobygames.com/images/shots/l/14887-aargon-windows-screenshot-shot-of-a-basic-level-deliberately.jpg, ([Online] Accessed 17/05/2021).

[Mobb] MobyGames, *Aargon(windows)*, https://www.mobygames.com/game/aargon, ([Online] Accessed 12/05/2021).

[Orc] Virtual Playing Orchestra, *Virtual playing orchestra*, http://virtualplaying.com/virtual-playing-orchestra/, ([Online] Accessed 17/05/2021).

[Oxf10] Oxford, *Oxford studijní slovník*, Oxford University press, 2010.

[Plo] Plogue, *Sforzando*, https://www.plogue.com/products/sforzando.html, ([Online] Accessed 17/05/2021).

[Rog14] S. Rogers, *Level up! the guide to great video game design*, John Wiley & Sons, 2014.

[Rom15] Pierre-Armand Nicq Romain, Caudron, *Blender 3d by example*, Packt Publishing, 2015.

[Rus] Jeff Russell, *Basic theory of physically-based rendering*, https://marmoset.co/posts/basic-theory-of-physically-based-rendering/, ([Online] Accessed 30/04/2021).

[Sch08] J. Schell, *The art of game design: A book of lenses*, CRC Press, 2008.

[Stea] Steam, *Archaica: The path of light, image from the game*, https://cdn.cloudflare.steamstatic.com/steam/apps/550590/ss_d08bbefba9e1e95974cbe174f5869871f5d9a942.600x338.jpg?t=1510283845, ([Online] Accessed 17/05/2021).

[Steb] _____, *Talos principle, image from the game*, https://cdn.cloudflare.steamstatic.com/steam/apps/257510/ss_b42acabe63d45a11580a2949e34f305e1bd10fc7.600x338.jpg?t=1601561095, ([Online] Accessed 17/05/2021).

[Stu] PopCap Studios, *Bejeweled*, https://www.ea.com/cs-cz/games/bejeweled, ([Online] Accessed 13/05/2021).

[Texa] TextureHeaven, *Source of free textures*, https://texturehaven.com/textures/?c=floor, ([Online] Accessed 17/05/2021).

[Texb] Textures.com, *Rocksmooth0045*, https://www.textures.com/download/RockSmooth0045/13130, ([Online] Accessed 12/04/2021).

[Tra] Tracktion, *Waveform free*, https://www.tracktion.com/products/waveform-free, ([Online] Accessed 17/05/2021).

[Two] TwoMammoths, *Archaica: The path of light*, https://store.steampowered.com/app/550590/Archaica_The_Path_of_Light/, ([Online] Accessed 14/05/2021).

[Unia] Unity, *Image of the effect of normal and height map*, https://docs.unity3d.com/uploads/Main/StandardShaderParallaxMap.jpg, ([Online] Accessed 17/05/2021).

[Unib] _____, *Image of the effect of smoothness*, https://unity3d.com/sites/default/files/learn/updated-7.3-copy-of-staying-on-track-25.jpg, ([Online] Accessed 17/05/2021).

[Unic] _____, *Unity game engine*, https://unity.com/, ([Online] Accessed 17/05/2021).

[Val] Valve, *Portal*, https://store.steampowered.com/app/400/Portal/, ([Online] Accessed 13/05/2021).

[Wik] Wikipedie, *Portal, image of the mechanic*, https://upload.wikimedia.org/wikipedia/commons/thumb/4/41/Portal_physics.svg/220px-Portal_physics.svg.png, ([Online] Accessed 17/05/2021).

[Zep] ZeptoLab, *Cut the rope game*, https://www.cuttherope.net/, ([Online] Accessed 13/05/2021).