CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Bachelor Thesis

Survival 3D computer game

Tomáš Pokorný

Field of study:	Computer Games and Graphics
Supervisor:	Ing. Ladislav Čmolík, Ph.D.

May 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Pokorný Tomáš**	Personal ID number:	**483433**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Graphics and Interaction**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Games and Graphics**		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Survival 3D computer game

Bachelor's thesis title in Czech:

Úniková 3D počítačová hra

Guidelines:

Get familiar with the principles behind computer game development. Analyze game principles utilized in the computer games played from the first-person view, especially in survival games where the goal is to escape from a specific area. Based on the analysis, design a 3D survival computer game. Further, create modular components, their materials, and textures. Based on the design, create at least three playable levels of the game utilizing the modular components. Test the game with a qualitative test with at least six players.

Bibliography / sources:

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.

Name and workplace of bachelor's thesis supervisor:

Ing. Ladislav Čmolík, Ph.D. Department of Computer Graphics and Interaction

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **02.02.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Ladislav Čmolík, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Ladislav Čmolík, who guided me through the whole process of working on this thesis and provided his valuable insight into the topic. I would also like to thank my family and friends for their unconditional and endless support.

Declaration

I hereby declare that I worked on this thesis independently and I cited all used sources with accordance to ethical principles for writing an academic thesis.

Prague, May 20, 2022

Abstract

This thesis analyzes the principles of creating computer games. Based on already existing computer games, it analyzes the game mechanics used in survival horror games. It analyzes the purpose and usage of modular components in games. In accordance with the analysis, a survival horror 3D computer game was designed. This thesis then briefly analyses Unity, which is the game engine used for developing the said game. This thesis describes the process of creating the game. The game was then tested by six users.

Keywords: survival horror, 3D game, Unity, modular components

Abstrakt

Tato práce analyzuje principy tvorby počítačových her. Analyzuje herní mechaniky používané v survival hororových hrách na základě již vytvořených počítačových her. Dále analyzuje účel a použití modulárních komponent v počítačových hrách. V souladu s touto analýzou byla navržena survival hororová hra. Tato práce krátce analyzuje Unity, herní engine, který byl použit pro vývoj zmíněné hry. Dále tato práce popisuje proces vytváření této hry. Tato hra byla poté otestována šesti uživateli.

Klíčová slova: survival horor, 3D hra, Unity, modulární komponenty

Contents

List of figures

1 Introduction

Gaming industry is one of the fastest growing entertainment industries in the world. The largest increase in the speed of growth of the gaming industry happened during the past 20 years. The pace at which the tech industry is growing is directly reflected upon the pace of the growth of gaming industry. More and more people are having computers or any other devices, such as gaming consoles or mobile phones, which can be used to play games. The widespread of these mentioned devices is one of the reasons why computer games are getting a lot more attention nowadays. Whether it is people who play games occasionally or people whose careers are undeniably tied to computer games, it is obvious that games take part in a lot of people's lives.

Because of the enormous popularity video games are getting, they are becoming more sophisticated, advanced, complex, and often introduce new ideas. Therefore, game development is not just programming, but it combines various areas of expertise such as 3D art, music, game design, story-telling and many other areas. Even though it is still possible to make a game single-handedly, take look at Minecraft – a game which was made by one person and only later when it became very popular it was sold to Microsoft for further development, it is more common for games to be developed in teams.

1.1 Motivation

I have always been fascinated by computers and computer games especially. As I was growing up, I was introduced to many computer games by my friends or relatives. I have always wanted to create my own computer game where only I have a say in what it is going to look like. During my early teenage years, I began to like survival horror games. This eventually turned into a deep passion, and it is the reason why I chose this game genre for my game.

1.2 Goals of thesis

The main goal of this thesis is to create a survival horror game. I begin by analyzing the process of creating computer games, followed by analysis of already existing survival horror games and the mechanics used in them. The next step is designing my own game based on the previous analysis. After analyzing Unity, the game engine in which I develop my game, I move onto the process of creating the game. The last step is to test the game with multiple people. Based on the results of the testing, I can then make adjustments to its gameplay and fix any potential bugs.

2 Analysis

This chapter focuses on analyzing the problematic of creating a 3D survival horror computer game. Some of these approaches are common for various game genres, whilst others are specific for survival horror games. This chapter briefly mentions some of the most known video game genres, giving careful attention to the survival horror genre. It mentions some of the already created computer games with survival horror or similar game genre and sheds light on some of the game mechanics used in them.

2.1 Games in general

What exactly are games? Games are not just one thing. Games can be puzzles, learning tools or simulations. Games should be entertaining, pleasurable experiences or activities. It is important to note that different people find different activities entertaining and it is very difficult to create a game that everyone is going to enjoy. Games present certain patterns which players learn over the time [1]. Some of these patterns are obvious, others can take a significant amount of time to be understood and mastered. By understanding the patterns of a game, players become better at playing the game. It is crucial for a game to balance between simple and complicated patterns. If the game is too simple, players will learn its patterns in no time and quickly become bored. On the other hand, an extremely complicated game is going to make it hard for players to grasp its purpose or meaning, often resulting in players giving up on the game before fully exploring its richness and possibilities. The ideal game would continuously bring new patterns and enhance the existing ones, so that a player is still learning new things and does not become bored.

One of the most important aspects of a game is its rules. They define how the game is supposed to be played, what is allowed and what is not. With a strict set of rules, games can turn out to be limited more than it is needed. On the contrary, having a rule set that is overly tolerant can make players tend to look for alternative ways of achieving the game's goal, in other words, cheat [1].

2.2 Computer game genres

Not all video games are alike. A video game, same as any other game, is characterized by its content, the key features, the controls, the purpose, and the things you can do in it. Like movies or tv series, we categorize games based on their genre. There are plenty video game genres and the categorization may slightly differ from one author to another. As the game industry is developing, some of the genres are becoming more and more popular. On the contrary, some genres are seeing a downfall in their popularity. First person shooter (FPS), multiplayer online battle arena (MOBA), strategic games, racing games, survival horror games, simulators and sport games, role playing games (RPG) and platformer are one of the most known game genres. Games often belong to multiple genres and some of the genres also overlap.

2.2.1 First person shooter

First person shooter is one of the most widely recognized video game genres. The “first person” in the name stays for the camera view used. In this case, a player plays the game as a character and sees the game environment through the character’s eyes. This usually makes the gaming experience more immersive; it makes the player feel like he is actually in the game himself. The “shooter” in the name refers to the game’s main idea. Player is equipped with a fire weapon and shoots enemies that pose a threat. FPS games are usually deathmatches or story based. In deathmatches the goal is usually to get the highest score, in other words, kill as many enemies as possible. In a story-based FPS game the player’s goal is to complete tasks, often referred to as quests, and kill the enemies that he or she encounters.

2.2.2 Multiplayer online battle arena (MOBA)

Multiplayer online battle arena is one of the genres that has seen immense increase in popularity over the last decade. That is mainly because of the games such as League of Legends, Dota2, Heroes of the Storm being extremely popular. The most common practice is two teams consisting of few players, usually two to five, fighting against each other in a battle arena. Generally, the goal is to capture a base, destroy enemy structure or kill the opposing team. Each player plays for their own character. Each character has a specific set of abilities. It is very common for MOBA games to have a camera view from the top, making the player see his own character as well as the surroundings. Winning a game usually rewards a player with some sort of rating. The rating is used to determine the skill level of a player. The best players can become professional esports players. They are picked up by a team, they train daily, receive a salary like in any other job and compete in tournaments all over the world for large prize pools.

2.2.3 Strategic games

Strategic games can be both single player as well as multiplayer. The main purpose of strategic games is not to impress the player with a lot of fast paced action, but rather to make the player plan the correct strategy in order to win a battle or the whole game. Good examples of strategic games are turn-based digital card games or games where you control a whole empire with thousands of soldiers and your goal is to conquer land and defeat the enemy empire.

2.2.4 Simulators and sport games

Simulators and sport games try to simulate real life activities in the most precise way possible. There are simulators simulating a truck driver, a farmer, gas station worker and many others. A lot of sports have their sport games nowadays. A great example is soccer or ice hockey. The most known soccer games are FIFA or Pro Evolution Soccer, the most famous ice hockey game is NHL. These games are usually published yearly, with every year’s version having slightly updated graphics, gameplay, and other areas. Players can play multiple game modes in these games, either playing as one specific athlete, as an entire team or even as a coach.

2.2.5 Role playing games (RPG)

Role playing games are games in which a player plays for one character. The player can often enhance his character by doing certain activities or by gathering better items such as clothing or weapons. These games are usually story rich. The player goes through dialogs with other characters in the game and can choose from multiple answers. The chosen answers can have a huge impact on the way the game progresses, sometimes resulting in a death of an important character. This makes the games special and entertaining, because the outcome is often different depending on who plays the game.

2.2.6 Platformers

Platformer games do not focus on the character or a story, but rather on running, jumping, climbing, and overcoming obstacles. Player is challenged to complete series of difficult jumps or dodge obstacles. These games are often separated into levels with increasing difficulty. Platformer is one of the oldest game genres. These games are often 2D and have a view from the side, allowing the player to see his own character and a big part of the level.

2.3 Survival horror game

Survival horror game is yet another of many video game genres. As the name of the genre says, it is a combination of two genres. It is the genre that I chose for my game.

Horror games are usually set in dark, gloomy areas, with both indoors and outdoors being a viable option. Their main goal is to make the player scared. They attempt to pull the player in the game's environment, making him feel as if he or she was in the game.

Survival games are mostly about managing resources [2]. If a player tends to waste his resources and use it when it is not necessary, he or she is likely to run out of it in the later stages of the game, which often results in the game character's death. Whereas saving the resources more than it is needed does not result in losing the game, it often makes the game harder. A perfect example of this is using a flashlight. Not using it in certain parts of a game can make it complicated for players to navigate in the environment, running out of batteries in an important combat or while being chased can result in dying or losing the game. Resources in survival games are often things like ammunition, health, stamina, batteries for a flashlight or all kinds of materials such as wood or stone.

Because these two genres are closely associated, it comes as no surprise that it eventually formed the survival horror genre. On top of the goals mentioned for horror and survival games, the goals for survival horror games can also be to escape some areas or buildings, avoid enemies or complete small tasks or quests. Some of the games from this genre also allow the player to fight the enemies instead of just avoiding them or escaping. Good visuals as well as sounds are key for any good survival horror game. More than in other game genres, the atmosphere in survival horror games is of critical importance.

2.4 Analysis of existing survival horror games

There are plenty of games that belong to the survival horror genre, some of them being more popular than the others. A good example of games from this genre that allow combat is the Resident Evil game series. The first game from this series is more than 20 years old and the whole saga has developed significantly since its first game was released. The game series features over 10 games (including remakes), some of them being multiplayer and also featuring one virtual reality game [3].

Quite the opposite, a game that does not allow you to fight enemies, is Outlast. One of the most famous games from the survival horror genre, which increased this genre's popularity massively. The game's main protagonist Miles Upshur goes on a mission to investigate inhumane experiments happening in a private psychiatric ward. With Miles having no special abilities or skills nor being able to fight with his enemies, this game ultimately comes down to running and hiding [4]. Players are required to think about their actions and always keep in mind that making a silly mistake can result in their character's death.

In the following three sections, I present three games from this genre that I have the most experience with. All of them are first person games, do not allow the player to fight enemies and share certain game mechanics. Some of these games specifically concentrate on the stealth mechanic, which is moving around the game's environment without being seen or without letting the enemies know about your presence. These games are the ones that introduced this genre to me. Even though I must admit that these games managed to repeatedly scare me to a point where I had to pause the game, they also made me fall in love with this genre and had a great influence on me choosing this topic for my thesis.

2.4.1 Penumbra: Black Plague

Penumbra: Black Plague was the first survival horror game that I have ever played. At that time, I had little experience with computer games in general. Penumbra immediately stunned me with its mechanics that were different from other games that I had played before. Being able to grab, hold and move a lot of objects in the game's environment was something that I had never experienced before. Opening drawers or lockers with mouse drag while looking for useful items or opening doors and hoping there are no enemies waiting behind it really builds up the atmosphere. Combined with solving little puzzles in order to move on in the game while continuously looking over your shoulder if there is an enemy that will start chasing you, it really creates a strong and unique experience.

Penumbra: Black Plague is the second entry in the Penumbra series. The story continues at the point where the previous game finished, Phillip, the game's main character that player controls, wakes up in a small room. This room is locked, therefore he needs to find his way out. Once he manages to escape the locked room, he finds himself in a research facility. Most of the staff is dead, the rest was infected by a virus and turned into hostile creatures. Phillip later realizes he is also infected, but his body reacts to the virus differently. The virus does not turn him into a hostile creature, instead he is being taunted by the mind of "the infected" during the game. He tries to cure his infection with a help of Dr. Amabel Swanson, who is locked in one of the labs, and tries to escape the research facility [5].

Figure 1 - Penumbra: Black Plague [6]

2.4.2 Amnesia: The Dark Descent

Amnesia: The Dark Descent is another game that I experienced playing. I had experienced playing Penumbra: Black Plague previously, thus some of the mechanics were not new to me. I knew what kind of gameplay works in survival horror games. I had a partial understanding of what was expected of me in some of the situations the game put me in. The horror atmosphere in Amnesia created by its game developers is sensational and truly horrifying. The game's main character, Daniel, finds himself in the Brennenburg Castle. From a note he wrote to himself, he finds out that he has erased his memory and that he is being chased by "the Shadow". Daniel tries to find his way to the Inner Sanctum in order to find and kill the castle's baron, Alexander [7].

While Amnesia is in a lot of ways similar to Penumbra, it also presented some mechanics that were new to me. One of them being sanity, a sort of mental health equivalent. The sanity indicator can be seen in the inventory. The more the player stays in dark areas of the game or looks at enemy creatures, the lower his sanity becomes. On the contrary, lighting up torches using tinderboxes, staying in lighter areas or avoiding enemies for longer periods of time increases sanity. With the player's sanity being low, he or she experiences all sorts of hallucinations and unpleasant visual effects. These effects affect his ability to play the game negatively. If player's sanity gets extremely low, it can even result in his death.

Figure 2 - Amnesia: The Dark Descent [8]

2.4.3 Slender: The Arrival

Slender: The Arrival is yet another game that caused rapid increase in popularity of the survival horror game genre. It is based on Slender: The Eight Pages which was an indie game. According to Wikipedia, "*an indie game, short for independent video game, is a video game typically created by individuals or smaller development teams without the financial and technical support of a large game publisher*" [9]. Slender: The Eight Pages is a game that focused on sneaking around and it also limited the amount of time for which a player can sprint [2]. Similar to Penumbra, the player in Slender: The Arrival is also equipped with a flashlight. Depending on the game's difficulty settings, the flashlight can or cannot run out of batteries. What is different is that in Slender: The Arrival, shining on certain enemies with the flashlight causes them to flee. However, this does not apply on Slender himself, who is the game's main antagonist. While Penumbra or Amnesia tend to keep the player scared continuously by building up the horror atmosphere, Slender: The Arrival is known for its jump scares, which are surprising horrifying events usually accompanied by a loud sound [10]. The gameplay is shorter compared to Penumbra or Amnesia; it averages around 2 hours of playtime [11].

Figure 3 - Slender: The Arrival [12]

2.5 Creating a computer game

While creating a simple computer game does not necessarily have to take a lot of time and effort, creating a good, advanced and successful game is often a long-sophisticated process. It involves multiple different tasks such as creating the game's concept, designing the game, prototyping, programming, testing, creating all the required 3D models, textures and so forth. Although it has been proven possible to do all these tasks as a single game developer, the most common practice is to develop games in teams.

2.5.1 Game concept

The game's concept is what is at the start of every computer game. It is an idea, an abstract visualization of how the game is going to look like. The game concept should not be about details, it should be about the game's core features, ones that define the game and make it enjoyable, perhaps unique from other games. In professional game development, it can be done in the form of short document or a presentation [13].

2.5.2 Game design

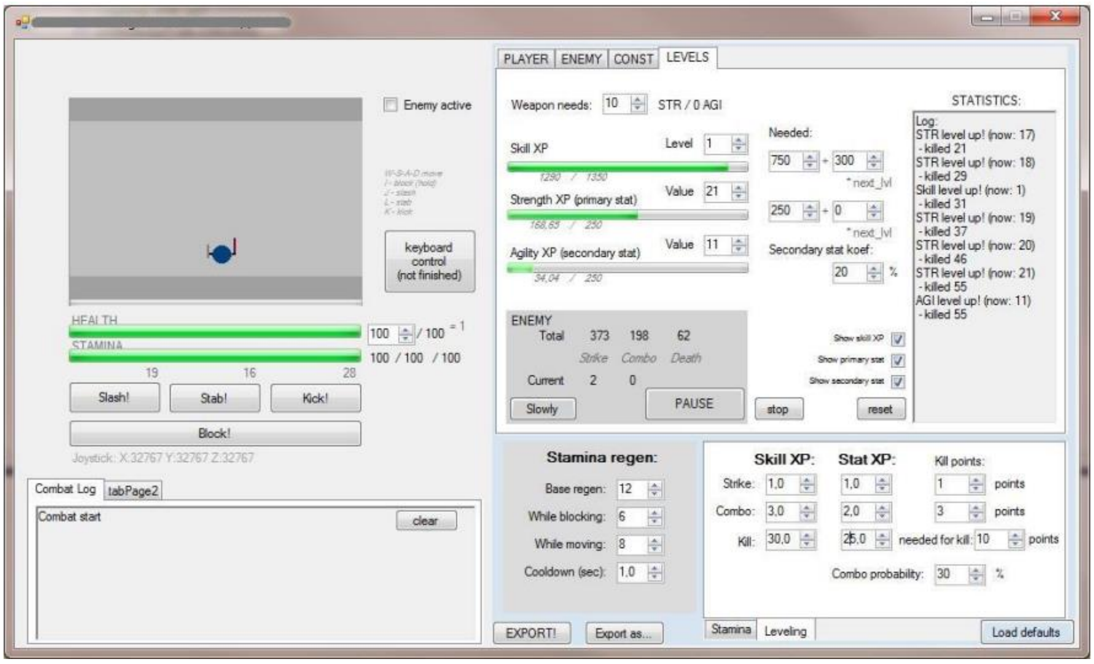
The next step on the road to making a video game is the game's design. It builds upon the game's concept, but it is far more detailed. Designing a game is the main task of game designers. They use the core ideas from the game's concept and design them in detail. It is their duty to come up with how these key features will be presented in the final project. Alongside these main ideas, game designers must also describe other features and mechanics that will be present in the game.

The size of game's design can be enormous, it is impossible be memorized. That is why game's design is almost always in a form of game design document (GDD). In later stages of the game development, programmers, testers and team members working on the game will use the game design document as a curriculum, a baseline for what they will be working on. It is important to note that the game design is a task that happens throughout the whole process of creating the game. Even though it would be great to have every single detail of the game on paper and then just create the game exactly as it is written down in the game design document, this is simply not possible. While the main concept should remain the same, it is almost certain that some of the game mechanics or features will have to be altered during the game's development. On top of that, there are certain things that cannot be designed in advance.

2.5.3 Prototyping

Once the game is designed, it is time to start with some implementation. Before diving headfirst into implementing the entire game, it is a common practice to begin with a prototype. "*A prototype is a model built to prove a concept*" [13]. In other words, it is the most basic, simplified version of the game that is supposed to demonstrate the key game features. The prototype can be done in a game engine with the most primitive graphics, but it can also be a completely different application. For example, the prototype of a role-playing game can be a simple form or console application.

With the usage of prototyping, we can recognize some of the problematic parts in our game design. Knowing these issues in the very early stages of game development, game designers are able to make adjustments to the game design. These adjustments are usually not major, due to the time when they were revealed. On the other hand, skipping the prototyping part in the game's development process can make the game prone to contain several misconceptions in its design. These misconceptions, often unveiled in the later stages of game development, can result in extensive changes in some of the already finished parts of the game.



Enemy active

HEALTH 100 / 100 = 1
STAMINA 100 / 100 / 100
19 16 28

Joystick: X:32767 Y:32767 Z:32767

Combat start

Weapon needs: 10 STR / 0 AGI

Skill XP Level 1 1290 / 1350
Strength XP (primary stat) Value 21 168,63 / 250
Agility XP (secondary stat) Value 11 34,04 / 250

Needed:
750 + 300 * next_lvl
250 + 0 * next_lvl
Secondary stat koef: 20 %

ENEMY
Total 373 198 62
Strike Combo Death
Current 2 0

Show skill XP ✓
Show primary stat ✓
Show secondary stat ✓

```
Log:
STR level up! (now: 17)
 - killed 21
STR level up! (now: 18)
 - killed 29
Skill level up! (now: 1)
 - killed 31
STR level up! (now: 19)
 - killed 37
STR level up! (now: 20)
 - killed 46
STR level up! (now: 21)
 - killed 55
AGI level up! (now: 11)
 - killed 55
```

Stamina regen:
Base regen: 12
While blocking: 6
While moving: 8
Cooldown (sec): 1,0

	Skill XP:	Stat XP:	Kill points:
Strike:	1,0	1,0	1 points
Combo:	3,0	2,0	3 points
Kill:	30,0	25,0	needed for kill: 10 points

Combo probability: 30 %

Figure 4 - Kingdom Come: Deliverance prototype in the form of simple form application [14]

2.5.4 Implementation

Although some people consider implementation being just another word for programming, programming is not the only thing that belongs to implementation. Implementation is a wide term that encapsules multiple activities. In the context of creating video games, it means creating the game itself and its content based on the game's concept or GDD. Scripts, models, animations, textures or sound files, all of these are created during the implementation phase and together form the final game. The amount of time implementing a game is going to take can differ. One of the factors influencing the implementation length for a game is the game's genre. It is not surprising that implementing simple mobile platformer game is going to take less time than implementing a sophisticated story-rich RPG. Another factor, which influences the required amount of implementation drastically, is the game's length. In terms of game models creation, it depends on where the game is situated. A game situated indoors naturally tends to be less demanding when it comes to the number of required 2D or 3D models. On the contrary, an open world game situated outdoors generally requires more models.

2.6 Modular components

Creating an environment is one of the tasks that belongs to the game's implementation. Environments can vary from one game to another. Games have 2D or 3D environments, simple environments, high detailed environments, low poly style environments (low poly stands for low number of polygons, the basic geometric shapes from which game models are formed) and many others. Creating game environment can be one of the most time-consuming parts in the game development process, therefore there are techniques that can help 3D artists and game developers save their valuable time.

One of the techniques is using modular components. A modular component is an object, usually a 2D / 3D model, that is used in a game repeatedly. From large landmasses to small interior objects, modular models can be any size and shape. Take a building interior as an example. Instead of modeling all its walls, it is better to model just few types of walls, such as: straight wall, corner wall, wall with a window and wall with doors. It is then possible to assemble the interior walls from these mentioned modular components. Using modular components goes conjointly with using a grid that allows precise placement in the game world. Thinking modular when creating a game can save tremendous amount of time, however, it also has some downsides. Modular components bring certain restrictions to the game and their overuse can result in players noticing the repeating environment [16].

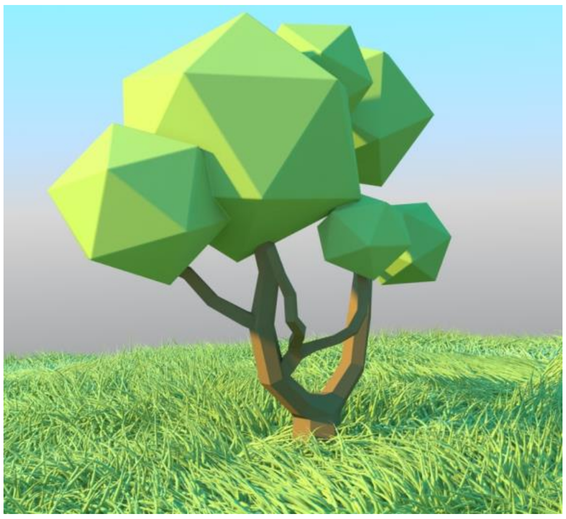


Figure 5 - Low poly style tree model [15]

2.7 Game engine

Video games are usually not developed from the ground up. They are developed using game engines. Game engine is a software specifically made for developing games. It offers a set of tools that makes developing a game easier and it also implements some of the features that games usually require, such as physics, game loop, sound system and more [17]. Choosing the right game engine that offers the needed features is a very important step and should not be taken lightly. In case the game engine is chosen correctly, it can make the development easier and faster, on the other hand, choosing a wrong game engine can make the development hard or even impossible.

Game engines can vary depending on their expected use. In the early days of game development, it was not uncommon for a game engine to be made just for one game. Whilst today the game engines are commonly either multipurpose or made for a specific game genre. Creating a game engine that can be used for any game seems rather impossible. Some of the bigger game companies develop their own game engines, engines that are specifically designed to suit the company's needs. Unreal engine and Unity engine are two examples of multipurpose game engines, Source engine is an example of engine developed by Valve company, primarily for their games.

2.8 Survival horror game mechanics summary

This chapter mentioned several mechanics that are defining for the survival horror game genre. One of them being the flashlight (or a lantern) as a means for the player to navigate in the often-dark areas. I find the flashlight very important and suitable for the game I am going to design and develop. The same can be said for the stealth mechanic. I find it very engaging, and it is something that differentiates survival horror games from most of the other game genres. For the survival part, managing resources has proven to be a vital concept. The most intuitive way of implementing this is with the usage of inventory and several resource items randomly scattered in the game's environment. I also particularly liked the ability to move some of the objects in the scene or opening doors by dragging the mouse, as if the player was holding the door himself. Putting a limit on how long a player can sprint makes the player think twice before sprinting. I also find this mechanic suitable for my game.

Being able to fight enemies can certainly enrich the possibilities of the game, however it also diminishes the importance of the stealth mechanic. Since I feel like the stealth mechanic is more defining for the survival horror genre than the ability to fight enemies is, I have decided that the ability to fight enemies is not suitable for my game. Scaring off enemies by pointing your flashlight at them is an interesting mechanic, however, because I am planning to only have one type of enemy in my game, I also do not find it suitable for my game.

3 Game design

This chapter focuses on designing the game I will be developing. Based on the analysis of survival horror games from the previous chapter, I will design my own survival horror game. This chapter mentions the game's story, purpose, and the idea behind it. It then mentions the mechanics that will be used in it and describes them in detail.

3.1 Idea

The game is a first-person survival horror game without the ability to fight enemies or anyhow participate in a combat. The main purpose is to make players feel drawn in by the game and to build up an atmosphere that creates a certain feeling of uncertainty and fear. Players are supposed to explore the game's environment and complete simple tasks. Completing these tasks unlocks other parts of the environment and allows players to advance to next levels. The ultimate goal is to escape from an underground complex where the game takes place. Players are supposed to use the stealth mechanic in order to avoid enemies and escape from the said underground complex.

3.2 Story

The game takes place in a secret underground research complex owned by a cult of people who perform inhumane experiments on people. The main protagonist of this game is a male, whose health was in perfect shape. He goes by the name of Cody. He was kidnapped and imprisoned in the underground research lab, because the cult wanted to perform deadly experiments on him. However, an unknown accident happened while he was in coma, and he woke up in the hostel area of the underground complex. Except for Cody, all of the people present in the underground complex turned into zombie-like creatures. These hostile creatures, although not very intelligent, are still able to hear and see. They are hungry for any human flesh. Cody needs to get out of the underground complex before these creatures manage to get him. All he thinks of is finding a way out of this nasty research lab back to the surface, so he can reunite with his loved ones and become The Last Escapee (TLE), which is also the game's name.

3.3 Controls

TLE is a first-person game. Players navigate in the environment using a combination of keyboard controls and mouse. Similar to other first-person games, players can use the **W** or **arrow up** key to move forward, the **S** or **arrow down** key to move backwards, the **A** or **arrow left** key to move to the left, and the **D** or **arrow right** key to move to the right. The movement direction is always relative, depending on where the player is currently facing. The facing direction determines what the player sees, and it is controlled by the mouse. There is no limit on turning left or right, in contrast to moving up and down, which is limited to 180°. This limit is there to mimic the up and down movement of a human head as well as to prevent the view from becoming upside down. No matter the direction a player is moving, his or her movement speed will be the same, unless he or she is sprinting or crouching. Players are able to sprint using the **SHIFT** key or crouch using the **CTRL** key. Players can also jump using the **SPACEBAR**.

The left mouse button (**LMB**) can be used to move some of the objects in the game. When a player is in proximity of an object that offers interaction and he or she looks towards the object, it is possible to interact with it using the **E** key. The **E** key is also used to pick up items and store them in the inventory. Once the player is equipped with a flashlight, he or she can enable it using the **F** key. Holding the **TAB** key allows the player to open the inventory. Navigation in the main menu as well as in the pause menu is done using the mouse. Pressing the **ESC** key while playing opens the pause menu and pauses the game.

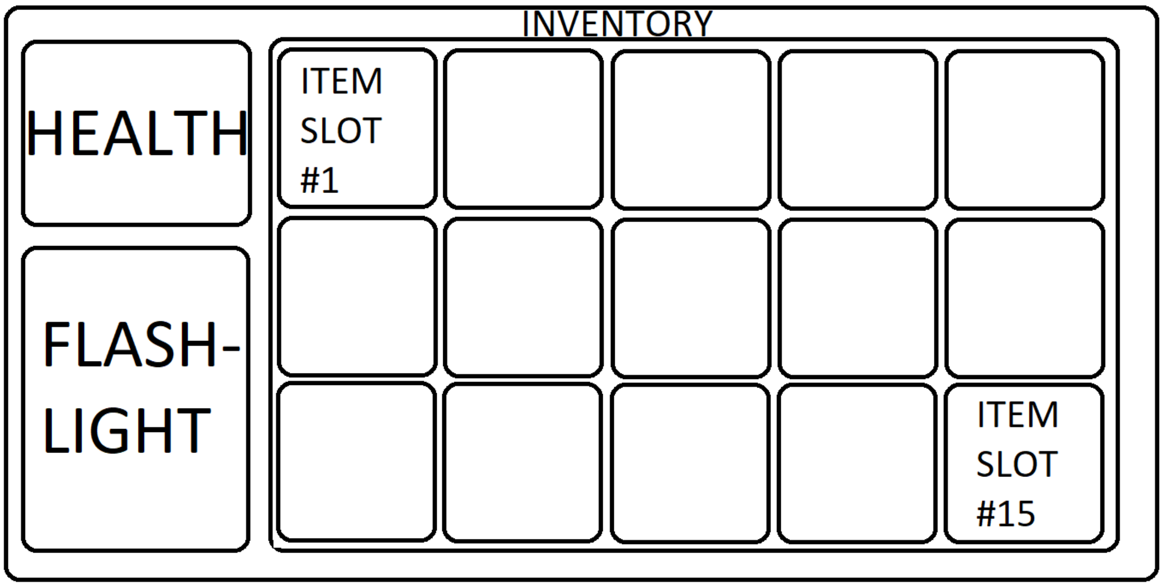


Figure 6 - Inventory layout

3.4 User interface and heads up display

TLE uses minimalistic heads-up display (HUD) in order to minimize the amount of disturbance while playing. This is a common practice for survival horror games. The default screen the player sees while playing has no HUD elements. There are several HUD elements which can appear as a reaction to certain events happening in the game, however, none of them is visible all the time. TLE uses non-diegetic user interface and HUD. Non-diegetic user interface is two dimensional, it is only visible for the player, and it is overlaid on top of the game.

A small hand cursor can appear in the center of the screen. When this happens, it means that the player is able to grab and move the object he or she is looking at. This HUD element works as an indicator that lets the player know about this possibility. When the player is no longer able to pick up the object, either because he or she is no longer looking at the moveable object or due to the distance between the player and the object, the hand cursor quickly fades away.

Whenever the player is able to interact with an object in the scene, he or she is notified about it by the HUD. The interaction key E is displayed alongside a short text that describes the interaction. This interaction text differs based on the object the player is interacting with. This HUD element appears instantly when the interaction is possible and also instantly disappears when the interaction becomes no longer possible. When the player tries to interact with an object, but it is not possible, there is additional info text displayed in the bottom center of the player's screen. It briefly mentions why the interaction is not possible and gives the player a small hint on what to do in order to make the interaction possible.

When the player is low on health, he or she is also notified about it by the HUD. There is no health bar or such thing in TLE. Alongside the border of the screen, there is a blood splash effect with drops of blood. It does not cover a big portion of the screen; however, it is still easily noticeable. This becomes visible once the player is below 50 health. The lower the player's health is, the more dominant this effect becomes. No matter if the player's health changes, this effect is dynamic and slowly alternates between being less and more visible.

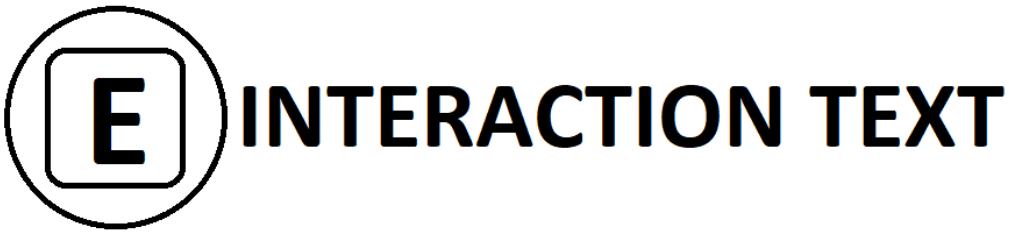


Figure 7 - Interaction UI element

There are two menus in TLE. There are buttons in both of the menus, with each button increasing its size when the player's mouse is over the button. The first menu is the main menu. There are four buttons in the main menu: Play, Levels, Controls and Exit. Play button starts the game by loading the first level. The levels button opens a submenu. This submenu lists all of the TLE levels. Clicking on a level button in this submenu starts the game by loading the corresponding level. By default, the player can only select the first level in this submenu. The subsequent level unlocks once the player successfully beats the previous one. The controls menu displays all the actions that are possible in TLE and the corresponding keys that are used to perform these actions. The exit button exits the game and returns the player to the desktop.

The second menu is the pause menu. It has three buttons: Resume, Controls and Quit. The resume button unpauses the game and returns the player back to the game. The controls button in the pause menu does exactly the same as in the main menu. The quit button returns the player to the main menu.

The last HUD element is the tutorial. It only works in the game's first level. For several of the game's mechanics, there is a hint. When one of these mechanics happens for the first time, the corresponding hint is displayed in the top center of the player's screen. These hints can not overlap, therefore when the next mechanics happens for the first time, the previous hint is replaced by the new one. Each hint, unless replaced by another one, stays visible for about 10 seconds, so the player has enough time to read it.

3.5 Enemies

The enemies are the main scary element in the game. They are also the only thing that can cause the player's character to die. There are multiple enemies moving in the game simultaneously, however, they are all of the same type – a mutated muscular zombie-like creature. These mutants are less intelligent compared to humans. They can see, however, it is quite limited in areas with no or very little light. In contrast, they are able to hear quite well. Enemies are wandering randomly in the game environment, going to all the places where they can while looking for any humans. When wandering in the level, enemies randomly select one of rooms and walk there. When an enemy spots or hears the player, it starts chasing the player. This can result in two scenarios. In case the enemy manages to close the gap between the player and itself, it attacks. In case the enemy loses track of the player while chasing, it goes on to explore the area where the player could possibly be hiding. If the enemy fails to find the player in the area it explores, it switches back to randomly wandering in the level. Due to their limited intelligence, enemies are unable to move objects in the scene. They are also unable to open doors or interact with interactable objects. Enemies also react to the light coming from the player's flashlight. When they spot it, they run towards it with the suspicion that something is probably going on there. If the flashlight shines directly on the enemy, the enemy starts chasing the player.

3.6 Mechanics

Some mechanics, such as the player movement, were already described in the section 3.3 Controls. Starting with the basic ones, there is player's health. Player's health is a number that ranges between 0 and 100. When player's health reaches 0, the character he or she is playing for dies and the game ends. Player can only lose health when attacked by an enemy. Because the main goal is to use the stealth tactic and avoid enemies, the player can only withstand few attacks before dying. A single attack deals around 20-50 damage depending on the distance between the player and the enemy. Similarly, there is only one way how to gain health. That is by using an item that replenishes health.

Since the player can sprint at times, the game uses a stamina mechanic. The stamina mechanic controls for how long the player can sprint. When sprinting, the stamina slowly decreases and once it reaches 0, the player is no longer able to sprint. There are two ways that allow stamina replenishment. One of them is automatic, when the player is not sprinting, his stamina slowly replenishes. The pace at which it replenishes depends on whether the player is moving or crouching. The second way that allows the player to replenish stamina is by using an item designed to do so.

The player has an inventory which he can use to store and use the picked-up items. Opening the inventory slows the game's time. If opening the inventory allowed for the game to pause completely, it would cancel building up the game's atmosphere significantly and it would serve as an unwanted interruption. On the other hand, if opening the inventory did not slow the game time at all, it would be hard for the player to use some of the items while being chased. Slowing the game time is a reasonable compromise.

There are a few items in TLE that the player can collect. Some of them have an active effect when the player uses them, others cannot be used but have a passive effect. The items in TLE are as follows:

- Flashlight
- Security card
- Batteries
- Medkit
- Stamina shot
- Key

Once **flashlight** is picked up, it is displayed in the inventory and the player can use it to navigate in dark areas of the game. The **security card** allows the player to open lab doors in the game's second level. **Batteries** are used to recharge the flashlight when it runs out of power. One batteries item recharges 50% of the flashlight's capacity. **Medkit** is used to heal up to 50 health. **Stamina shot** instantly replenishes all of the player's stamina. Having a **key** in the inventory allows the player to unlock any locked doors, however, the item is consumed, and the key stays in the door and cannot be picked up again.

There are several objects in the game that the player can interact with. The interaction is possible when the player is nearby such object and looks towards it. The interactions are as follows:

- Turning on / off lamps or lights in the game
- Unlocking locked doors
- Opening lab doors or heavy gates
- Enabling / disabling electricity using the electric box

3.7 Sound

Sound is especially important in survival horror games. TLE uses various sounds to help build up the atmosphere. The most dominant sounds in TLE are the player sounds. There are two sounds that the player makes. The breathing sound is significantly affected by the player's stamina. When the player's stamina is full, his breath can be barely heard. Conversely, when the player is tired and his or her stamina is extremely low, the breathing sound becomes quite loud, and he or she also breathes very fast. The second sound that the player makes is the footstep sound. The volume and frequency of the footsteps varies depending on the player's current movement speed.

Enemies also make sounds. Similar to the player, enemies make the footstep sound when moving. Its volume and frequency also change depending on the enemy movement speed. Occasionally, enemies can snarl, grunt, moan, or growl. When the enemy spots the player, they also make one of these sounds.

The interactable objects also have a specific sound assigned to them. This sound is played once the player interacts with the given object. When an object collides with another object, one of the collision sounds is played randomly.

When playing the game, there are random scarry sounds such as unintelligible whispering or spooky blowing wind, coming from around the player. These sounds play occasionally. When in the main or pause menu, there is an ambient sound playing in the background.

4 Unity engine

This chapter focuses on the Unity engine. It describes its basic functionality, its user interface and the workflow often used to achieve certain results while using this software. This chapter also describes some of Unity's key features. I have decided to use Unity engine for the development of my game. It is a suitable game engine that fits the needs of my game and I also have prior experience using it. I have used it while developing a game for the B4B39HRY course as well as while working on some small projects of mine. The Unity engine is available for free and if your revenue in the last 12 months did not exceed $100.000, you can commercialize your games using the free version. There are also paid subscriptions in case your revenue exceeds this amount [18]. I will be using the Unity version 2020.3.21f1 for my project.

4.1 User interface

Unity's interface can be seen in the figure **below.** This is the default layout of Unity's user interface.

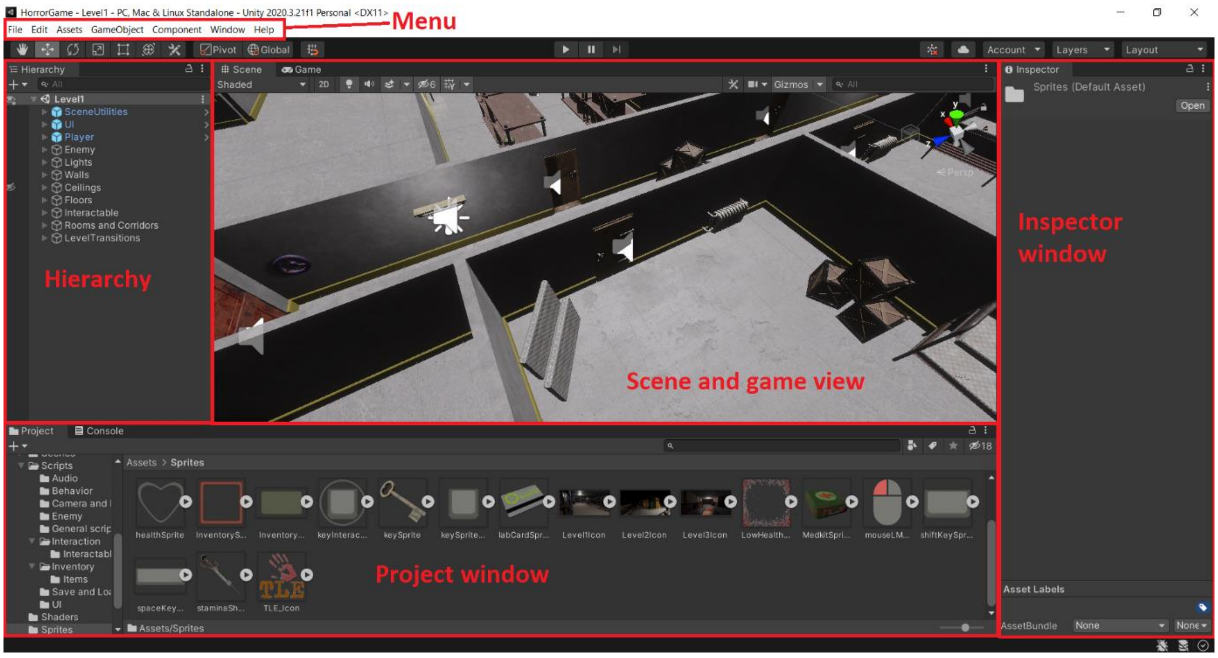

HorrorGame - Level1 - PC, Mac & Linux Standalone - Unity 2020.3.21f1 Personal <DX11>
Menu
File Edit Assets GameObject Component Window Help
Hierarchy
Scene and game view
Inspector window
Project window

Figure 8 - Unity user interface

The user interface is separated into multiple sections. The section on the left side is called **The Hierarchy window**. It shows the hierarchy of the objects in the currently opened scene. In this section, the user can target, create, destroy, or parent objects. In order to create an object, the user can either right click in the hierarchy and create an object from the menu that appears or drag an object from the project window to the hierarchy window. In order to destroy an object, the user can either right click on an object and destroy it from the menu that appears or left click on the object and press the delete key. Parenting objects is done by dragging one object onto another using left mouse button. The user can also make certain object invisible in the scene view or make them untargetable. It is crucial to keep the scene structured, especially when it contains hundreds or thousands of objects. A good practice on how to keep the scene structured is to use empty objects. The user can drag objects of the same kind to one empty object that will group them together and work as the parent object.

The bottom section is often referred to as **The Project window**. By default, it has two tabs. The project tab shows the project structure as well as the currently opened folder and its content. This tab is used to manage the project structure, add, remove, or adjust the game's assets. The second tab in this section that is displayed by default is the console tab. Whenever there is a problem with your project, the detailed information about this problem is displayed here. It is also used for debugging purposes.

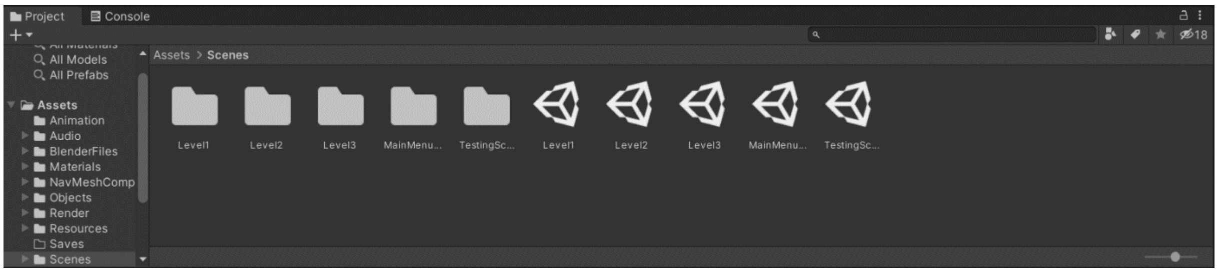


Figure 9 - The Project window in Unity

The section in the center, covering the largest portion of the screen, is referred to as **The Scene and Game view**. By default, there are two tabs: Scene and Game. Scene tab allows the user to assemble and edit the currently opened scene. The Game tab shows what the player is going to see. It is necessary to have at least one camera in the scene, otherwise the Game tab will not show anything.

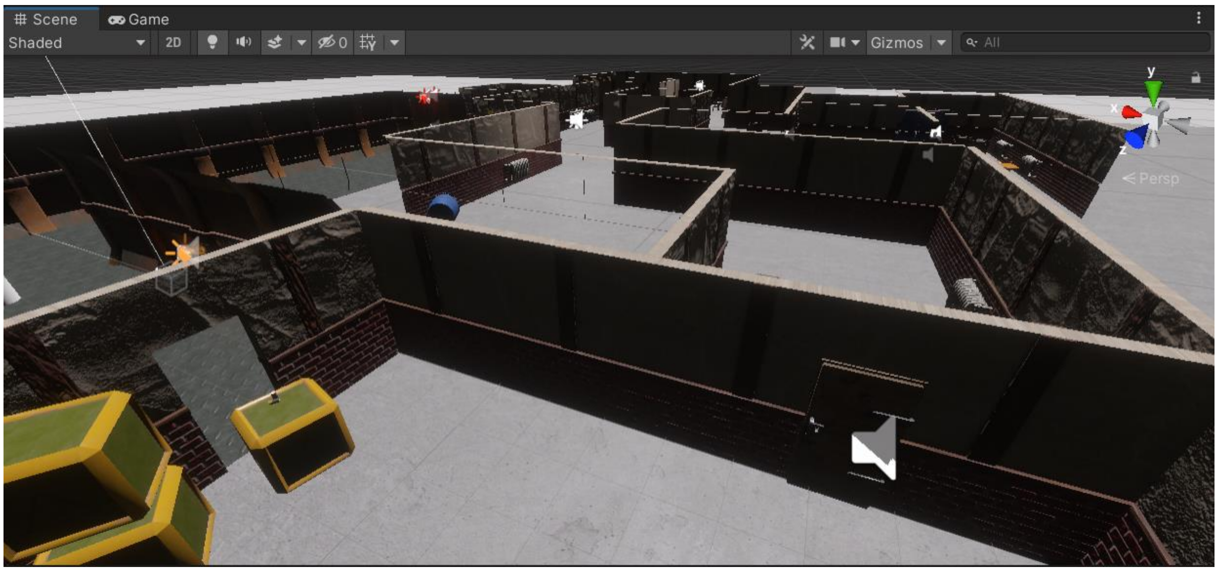
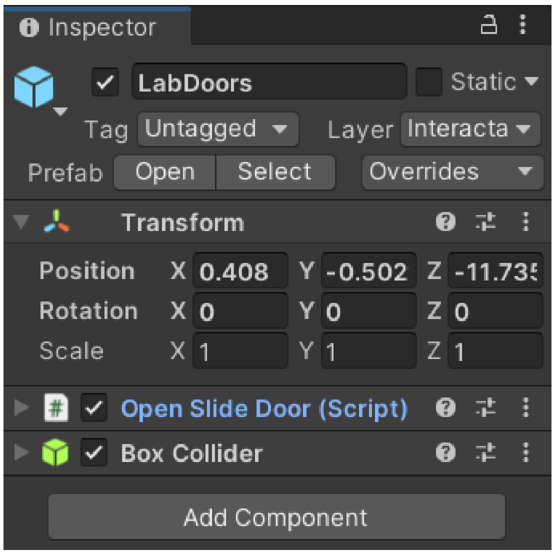


Figure 10 - The Scene and Game window in Unity

The section on the right side is called **The Inspector window**. The Inspector displays detailed information about the currently selected object. It shows the components that the object has, and it also allows the user to add new components or remove ones that the object already has.


Inspector

✓ LabDoors — Static

Tag: Untagged — Layer: Interacta

Prefab: Open | Select | Overrides

Transform

Position	X 0.408	Y -0.502	Z -11.73
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

✓ Open Slide Door (Script)

✓ Box Collider

Add Component

Figure 11 - The Inspector window in Unity

4.2 Assets

When developing games with Unity engine, users will work with assets. Asset is any item that is used to create the game, for example a 3D model, material, texture, or a sound. Assets can be created outside of the Unity engine and imported into the project. For example, it is a common practice to create 3D models outside of the Unity engine and import them into the project. On the other hand, it is also possible to create certain assets inside of the Unity engine. Materials, 3D primitives such as cube, animations, render pipeline assets or many others are examples of assets that can be created directly in the Unity engine. [19].

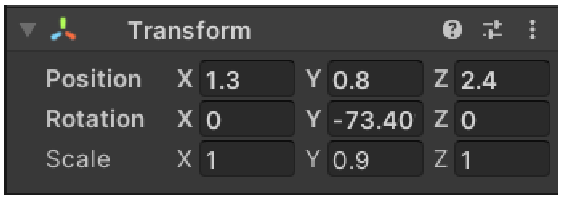
Scenes are also assets. According to Unity’s manual, “*scenes are where you work with your content in Unity*” [20]. Scene is a world where the game happens. Users can create multiple scenes in a project. For example, one scene can correspond to one level of the game. Users can add new game objects to the scene, remove or adjust the position or rotation of the already existing ones and more.

4.3 Components

Any object in the game is a game object. The contents of any scene in Unity engine are made of game objects. Each game object has a list of its components. The game object’s behavior is defined by the components it uses. Unity engine offers a wide spectrum of components for different purposes. The following section briefly mentions some of the most important components as well as the ones that are important for the game I have designed.

4.3.1 Transform

Transform is the component that every game object must have. It stores the values and properties that define an object in 3D space. Position, rotation, and scale are the three most important values which the Transform component defines. These values can be seen in The Inspector window. Transform also holds several other values and properties, such as the forward vector, which can be accessed from a script.

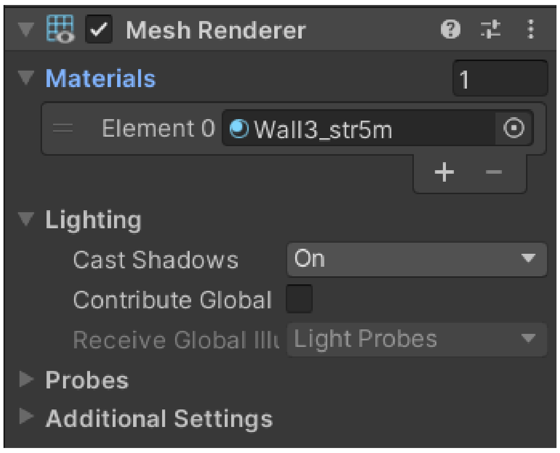

Transform

	X	Y	Z
Position	1.3	0.8	2.4
Rotation	0	-73.40	0
Scale	1	0.9	1

Figure 12 - The Transform component

4.3.2 Mesh renderer

Mesh Renderer is the component that Unity uses to render a mesh. A mesh is the shape of the 3D object. It consists of vertices, edges, and faces. If a certain 3D object does not have the Mesh Renderer component, it will not be visible in the scene. There is also the Skinned Mesh Renderer component which is used for meshes that can be deformed, such as characters or pieces of cloth.


Mesh Renderer

Materials 1
Element 0 Wall3_str5m

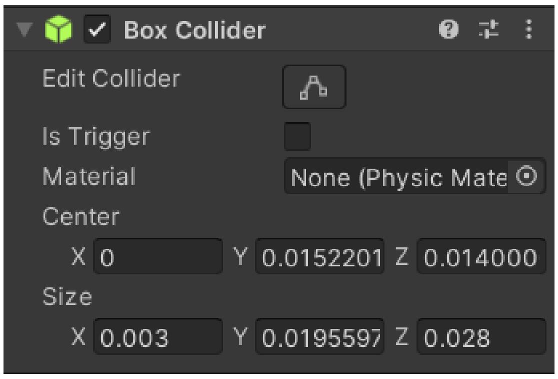
Lighting
Cast Shadows On
Contribute Global
Receive Global Illı Light Probes

Probes
Additional Settings

Figure 13 - The Mesh Renderer component

4.3.3 Colliders

Colliders are components that handle object collisions in the scene. If an object has no collider component, other objects or the player can simply move through it. There are multiple different collider components with different shapes. The most commonly used collider components are the Box Collider, the Sphere Collider, the Capsule Collider, or the Mesh Collider. Generally, it is a good practice to use simple colliders to approximate the shape of an object for collisions. Simple colliders such as the Box Collider take less computation time to compute whether or not the object is currently colliding. However, if we want to achieve more precise collisions, it is better to use the Mesh Collider.

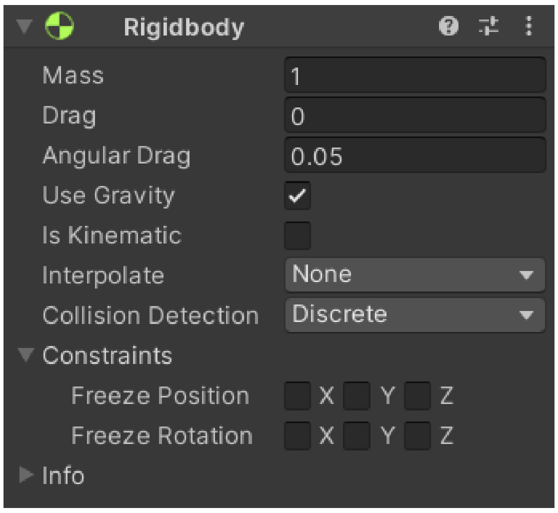

Box Collider

Edit Collider
Is Trigger
Material None (Physic Mate
Center
X 0 Y 0.0152201 Z 0.014000
Size
X 0.003 Y 0.0195597 Z 0.028

Figure 14 - The Box Collider component

4.3.4 Rigidbody

One of the things that the Unity engine handles is the physics. Adding the Rigidbody component to a game object allows it to be affected by Unity's physics. The user can define values such as mass, drag or angular drag, both from the Inspector window as well as from a script. Based on these values, Unity will compute this object's behavior whenever any force is applied to it. The user can also define multiple other parameters, such as freezing the object's position or rotation with respect to one of the axes.

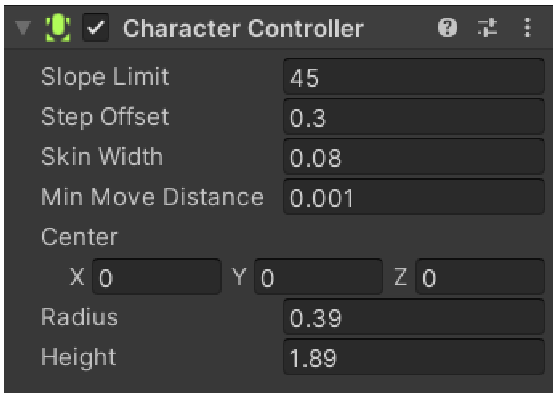

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	X Y Z
Freeze Rotation	X Y Z
Info	

Figure 15 - The Rigidbody component

4.3.5 Character Controller

The Character Controller component is one of the components that can be used to control a player in the game. Using the Character Controller allows the user to easily control player's movement in the game, however it does not make use of Rigidbody physics. Depending on the game, this can be both advantageous and disadvantageous. The Character Controller uses a built in Capsule Collider and the user can define where it will move from a script. The user can also define some of its parameters such as the Step Offset.

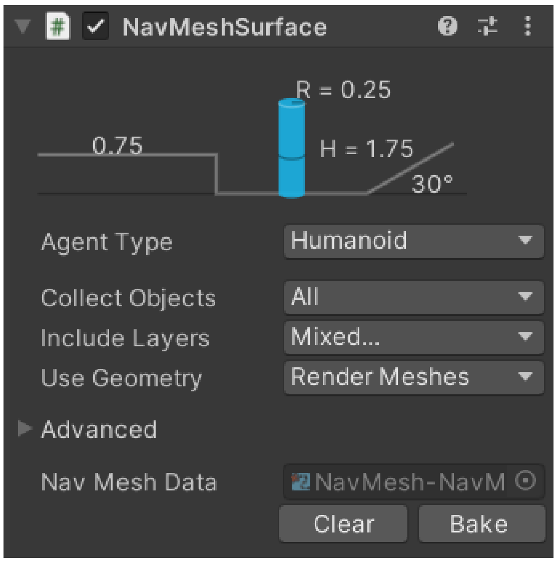

Character Controller

Slope Limit	45
Step Offset	0.3
Skin Width	0.08
Min Move Distance	0.001
Center	X 0 Y 0 Z 0
Radius	0.39
Height	1.89

Figure 16 - The Character Controller component

4.3.6 NavMesh components

Nav mesh components are used for navigating an agent in your scene. I use my own scripts and NavMesh components in my game for the enemy movement. Some of the NavMesh components are available directly in the Unity engine, however high-level NavMesh components must be downloaded separately [21][22].

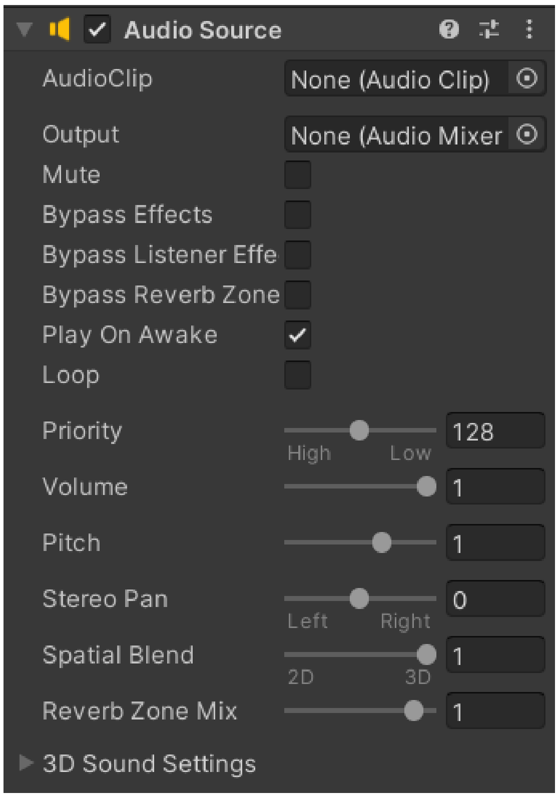

NavMeshSurface

Agent Type	Humanoid
Collect Objects	All
Include Layers	Mixed...
Use Geometry	Render Meshes
Advanced	
Nav Mesh Data	NavMesh-NavM

Figure 17 - The NavMeshSurface component

4.3.7 Audio components

Audio is an important part of a vast majority of games. In Unity, audio is also handled by the use of corresponding components. The two main components that control audio in Unity are Audio Listener and Audio Source. The Audio Listener component is used to capture the sound in the scene, which is then played to the player. It works just like a microphone. A common practice is to add the Audio Listener component to the Camera [23]. In order to play a sound for the Audio Listener to capture, Unity uses the Audio Source component. The Audio Source plays a sound in the scene. Based on the settings of the Audio Source, the sound can be 2D or 3D. The user can also define several other parameters such as volume, pitch and a curve which defines how fast the sound's volume fades out with distance.


Audio Source

AudioClip	None (Audio Clip)
Output	None (Audio Mixer
Mute	☐
Bypass Effects	☐
Bypass Listener Effe	☐
Bypass Reverb Zone	☐
Play On Awake	✓
Loop	☐
Priority (High – Low)	128
Volume	1
Pitch	1
Stereo Pan (Left – Right)	0
Spatial Blend (2D – 3D)	1
Reverb Zone Mix	1

3D Sound Settings

Figure 18 - The Audio Source component

4.4 Rendering

"*Rendering is the process of producing a 2D image from a description of a 3D scene*" [24]. Based on the position and the forward vector of the camera, this process takes the scene's objects, lights, textures and more and creates the output image out of it. This output image is then displayed to the user. The dynamic scene which the user can see is a result of constantly rendering new images (frames) based on the scene. Unity uses physically based rendering (PBR) to render the scene. PBR tries to mimic the behavior of light in the real world, and it follows the principles of energy conservation. When a light hits a surface, it can be reflected, refracted, diffracted, scattered, or absorbed. When an incoming light ray is reflected, it hits the object and bounces off. In this case, the angle of incidence (the angle between the incoming light ray and normal vector of the surface at the given point) and the angle of reflection (the angle between the reflected light ray and normal vector of the surface at the given point) are the same. This is also called the specular reflection. A refracted light ray goes through the surface, but it changes its direction based on the surface's index of refraction. Diffraction happens when the light ray is bent around the shape of an object it collides with. A light ray can also be absorbed, although it is often just a part of the color spectrum of the ray that is absorbed. Absorbed light is turned into heat or a different form of energy. A light ray is scattered when it hits an object with a lot of surface imperfection. It bounces off in many directions at a time [25]. Scattered light is also called the diffuse reflection. Based on the roughness of the surface, the amount of specular and diffuse reflection differs. Due to the principles of PBR, the overall energy of reflected light cannot be greater than the energy of the incoming light.

4.5 Render pipeline

The rendering process is performed by a render pipeline. It is a series of actions that the computer needs to perform in order to render the final image [26]. Unity gives its users the opportunity to choose from different render pipelines. Built-in Render Pipeline, Universal Render Pipeline (URP) and High Definition Render Pipeline (HDRP) are the three render pipelines that Unity offers. Built-in Render Pipeline is the default one and it offers limited amount of customization. URP works on many platforms. It offers more options and better customization. HDRP is the best in terms of visuals, however it is not available on all platforms. When it comes to hardware, HDRP is more demanding than the other two.

5 The process of creating my game

This chapter describes the process of creating my game. Once the game was designed, it was time to turn the design into an actual game. In order to do that, I had to complete several different tasks. I needed to create assets such as 3D models, textures, or animations. I also had to create scripts to define the game's logic and behavior. With all these assets, I created 3 different levels and assembled the game in Unity engine.

5.1 My workflow for creating 3D assets

I created all 3D models and textures in my game myself. I used the same sequence of steps for creating all of these 3D models, with an only exception being character models. Modeling 3D characters for games is a complicated and time-consuming task. It also comes with creating the character's rig, which is used for animation. I have no experience with creating 3D characters for games, therefore I decided to use Autodesk Character Generator [27]. The Autodesk Character Generator allows users to create 3D characters for games based on existing templates. I chose a suitable template and customized it according to my needs. The customization included things like changing the eye color, body shape, skin color or clothes.

The rest of the 3D models were created using the following sequence of steps. The first step was to create the model's geometry, The model's geometry is the shape, often referred to as mesh. Mesh is created in a 3D modeling software. There are several 3D modeling applications. I have tried using Blender and Maya [28][29]. I have experience using both, however, I have greater experience using Blender, therefore I created meshes for my 3D models in Blender. The only exception was a sci-fi incubator model used in the second level of the game, which I created in Maya.

Depending on the object I was modeling, I used different modeling techniques. For the vast majority of objects, I started with a primitive object such as cube or cylinder. The primitive object usually represented the basic shape of the object. I started adding more detail to the object, usually by using modifiers, inserting faces, extruding regions or by using the bevel tool. Once The mesh had been created, I checked it and potentially cleaned it if it had some imperfections. The next step was to unwrap the mesh to a 2D space. Although textures can be 3D, the most commonly used textures are two-dimensional. They are 2D images and in order to be applied on the mesh correctly, the mesh must be unwrapped. I mostly used unwrapping using seams. I manually defined the seams, in other words cuts, on the mesh. Based on these seams, the object's geometry was then cut and placed in the 2D texture space. I also used unwrapping using projection for some objects. Based on the camera position and its forward vector, the mesh is projected onto a plane. The last step in Blender was to paint the vertices. This step was not necessary, but it helped me significantly later on, when making textures for this model. Once this had been finished, I exported the model in the **.fbx** file format.

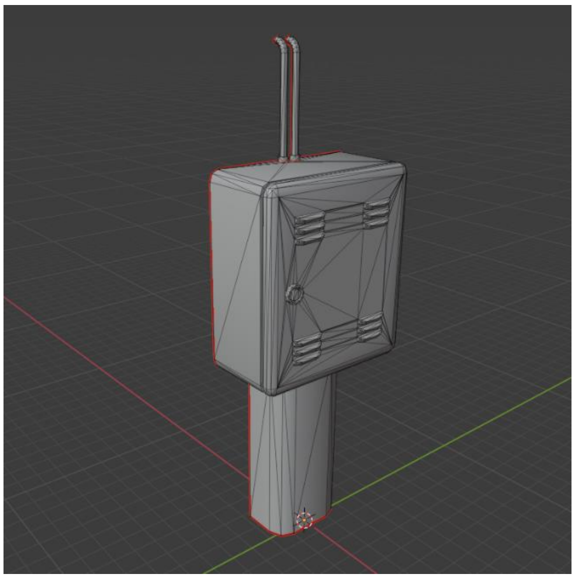


Figure 19 - Electric box mesh

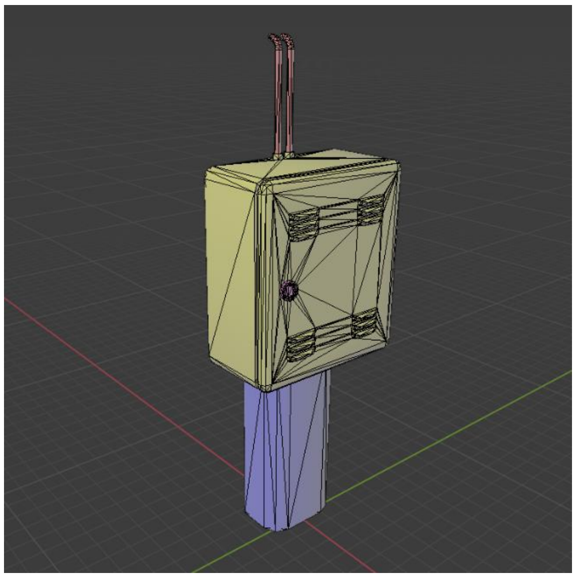


Figure 20 - Electric box vertex paint

When I had completed all the above-mentioned steps in Blender, I moved onto creating textures for my model. The software I used for this purpose was Adobe Substance 3D Painter [30]. This software has a paid subscription, however, as a student I was able to obtain a student license, which allows me to create textures for my student related projects. This software offers multiple features such as painting on the surface of a 3D object, working with layers, masks and a built-in library of materials and textures with adjustable attributes. I created a separate material for each distinct part of the model, e.g., wooden part of the model, metal part of the model or plastic part of the model. Because I had prepared the vertex painting in blender, I was easily able to mask materials, so they are only applied on the desired parts of the model. Once everything had been set, I exported the textures.

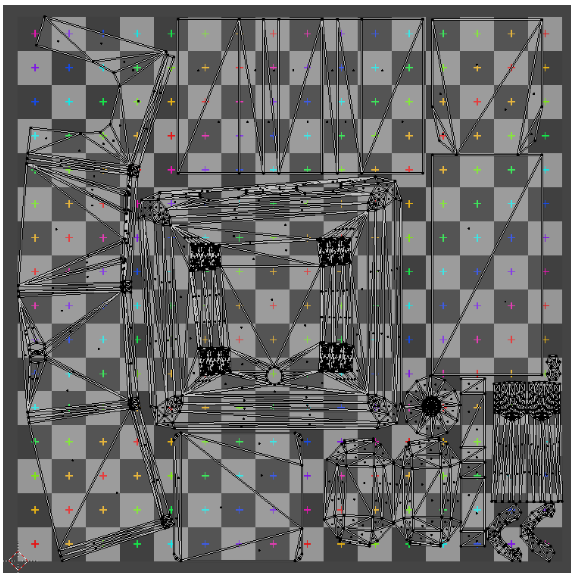


Figure 21 - Electric box unwrapped mesh in UV space

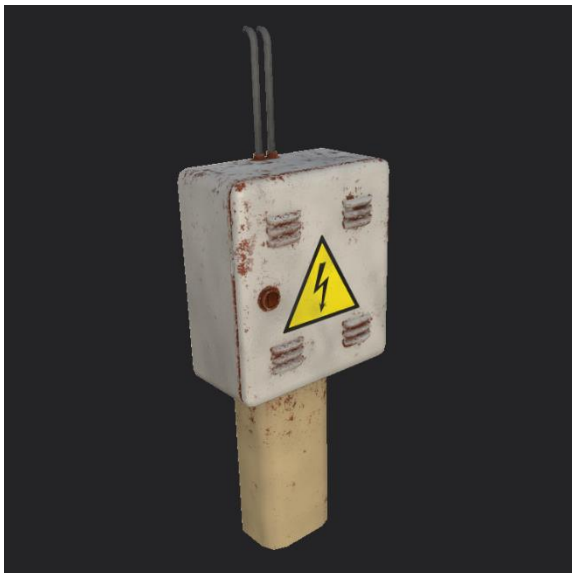


Figure 22 - Textured electric box

The next step was to import the model and the textures into Unity. With everything imported in Unity, I created an HDRP Lit shader, assigned the correct textures to it and applied it on the mesh. Furthermore, I created a prefab out of the object so it can be used in any scene of the game.

5.2 Modular components

I used modular components to build the basic shape of the environment. Because my game is only situated indoors, I needed modular walls to assemble the levels. I created three different sets of modular walls, each for one level. In order to create these modular components, I used the same sequence of steps described in the previous section. When creating a set of modular walls, I had to make sure that every piece in the set goes well together with the remaining pieces in the set. This means that the components should have precise proportions and that there should not be any visible seams when they are placed next to each other.

The first set of modular walls was used in the first level. I created 3 types of walls – straight wall, wall with a door hole and a wall with a large hole, used in corridors. They are concrete walls with a little ledge at the bottom, each wall being three meters long.

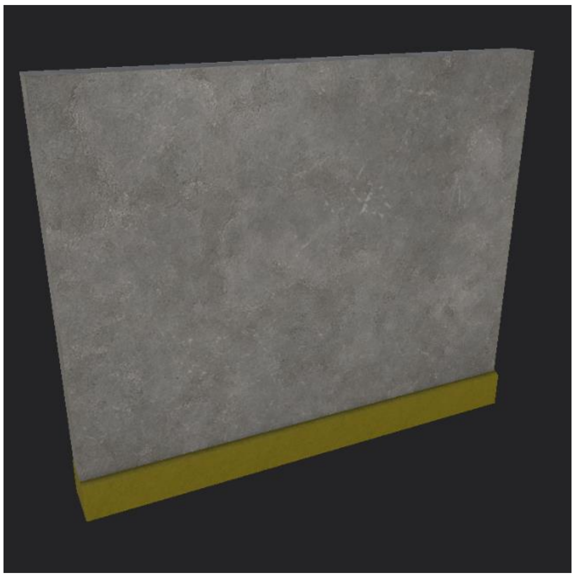


Figure 23 - Level1 straight modular wall

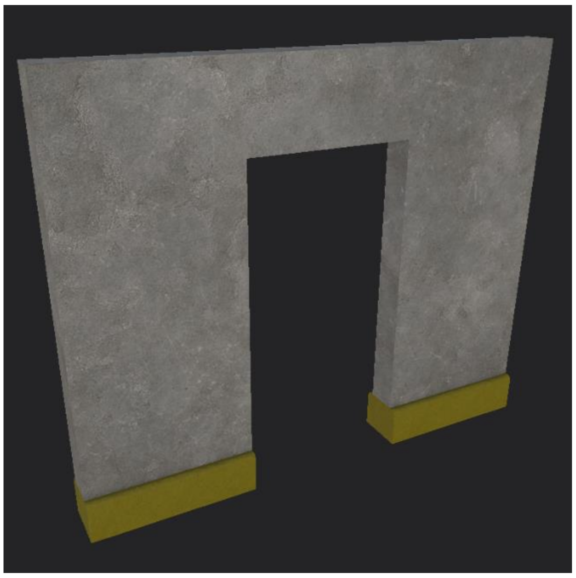


Figure 24 - Level1 wall with a door hole

The second set of modular walls was used in the laboratory level, which is the game’s second level. Compared to the first level, the laboratory level is overall more spacious, thus the walls are also larger. I wanted the corners in the second level to be round, therefore I had to create more walls for this set. Except for the straight wall and the wall with a door hole, I created variations of walls with either left or right corner bent in or out.

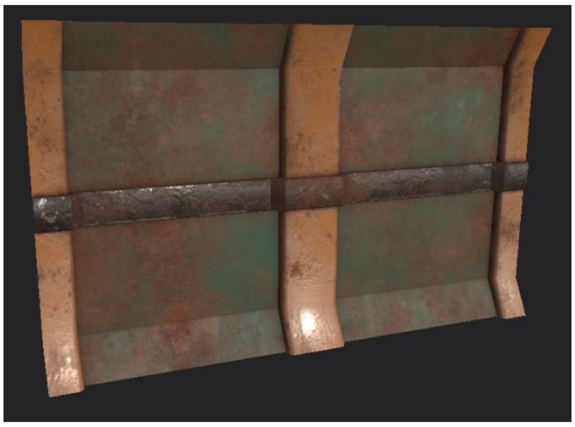


Figure 25 - Level2 straight wall

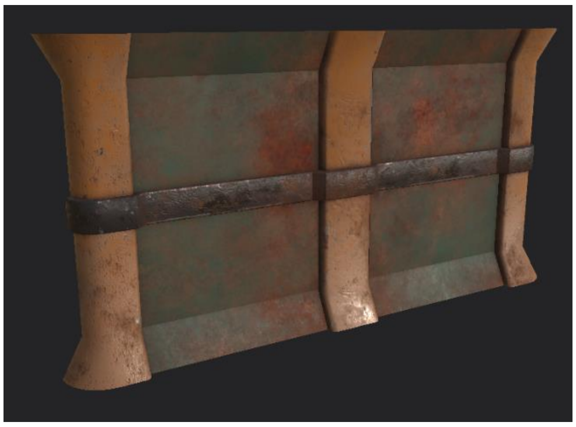


Figure 26 - Level2 wall with corners bent in

The last set of modular walls was used in the game's third level. These walls are a combination of brick and mortar, with a wooden ledge and occasional wooden planks. I created five types of walls - straight wall, wall with a door hole, wall with a large hole for a gate, left corner wall and right corner wall.

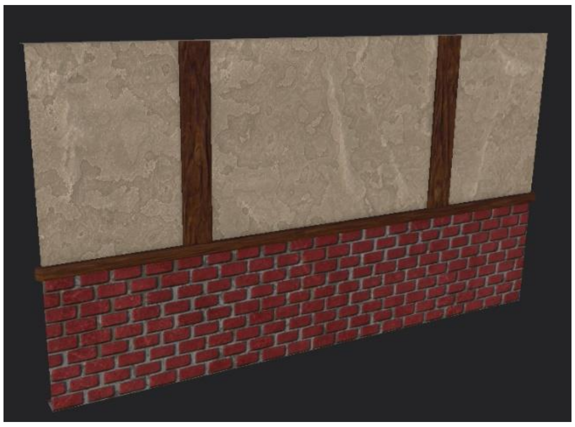


Figure 27 - Level3 straight wall

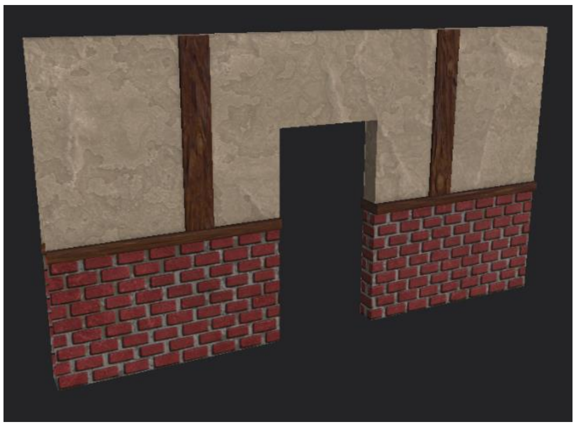
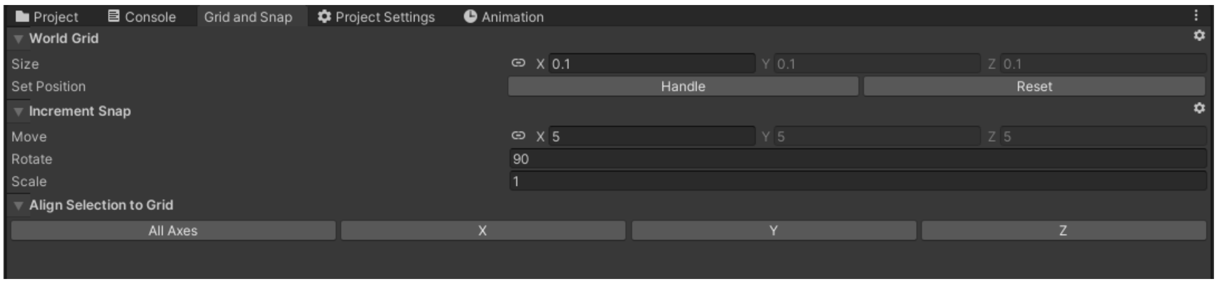


Figure 28 - Level3 wall with a door hole

5.3 Levels creation

With modular walls finished, I was able to start working on the levels. I started by working on the modular parts of levels. In order to work with modular components with ease, I used Unity's grid feature. I opened Unity's **Grid and Snap** window from the **Edit -> Grid and Snap** menu. Depending on the set of modular walls I was working with, I set the values in this window. As I was only placing walls which are either parallel or perpendicular to each other, I set the rotation increment value to 90 degrees.



World Grid
Size X 0.1 Y 0.1 Z 0.1
Set Position
Increment Snap
Move X 5 Y 5 Z 5
Rotate 90
Scale 1
Align Selection to Grid

Figure 29 - Unity's Grid and Snap window

I started by placing one modular wall in the scene and I aligned it properly. Then I copied it and placed it using the move tool. When the grid snapping option is enabled, it automatically places objects on the grid. It can only move by the increment defined by the grid's size. Similarly, when holding CTRL, the move tool only moves the selected object by the move increment amount. Using a combination of these two snapping techniques, I was able to create the base shape of each level. I then continued by adding other models, enemies, interactables and so forth.

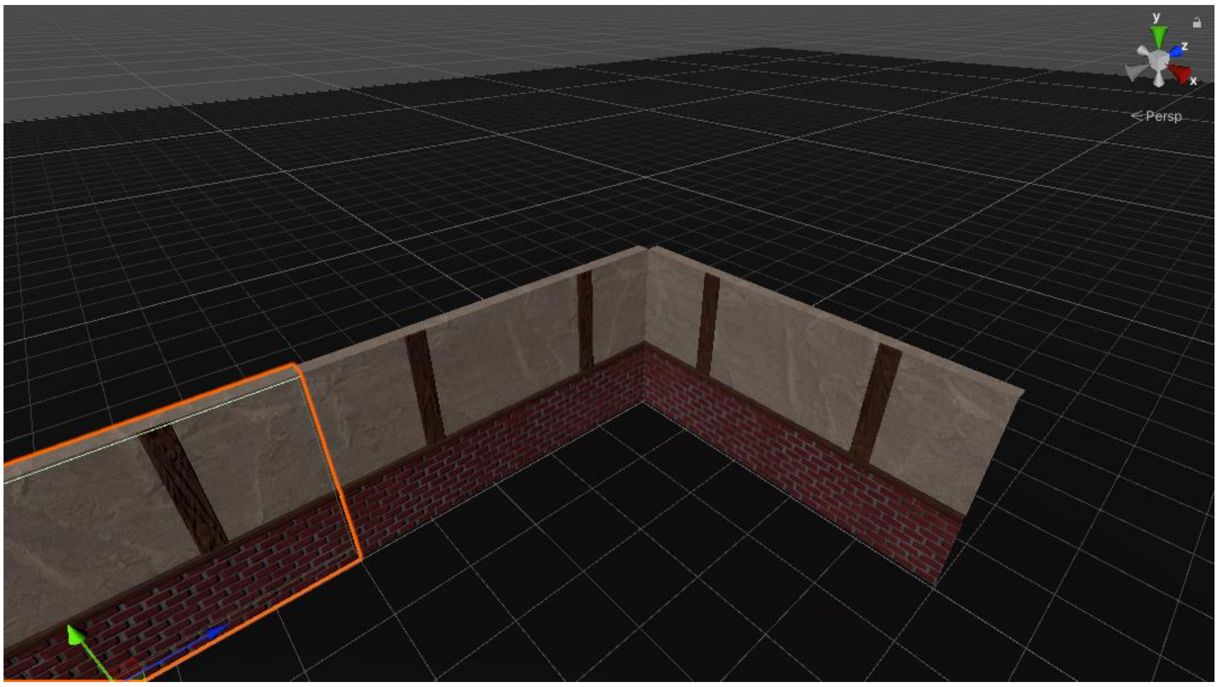


Figure 30 - Modular walls placed on the grid

5.4 Sprites

Sprites are 2D graphical elements used in games. Sprites can be images or animations. They are often used in 2D games and in pixel art stylized games. Sprites are also used for user interface. I created various sprites for my game's user interface and HUD. I had to create sprites for the inventory items as well as other UI elements. I created the vast majority of sprites in Blender. For the inventory items, I imported the models and textures to Blender, set up the camera, material, lighting and rendered an image with transparent background. For the UI elements, I usually created an object with the required shape in

Blender, added a material with just a base color to it and proceeded with the same steps as for the inventory items.

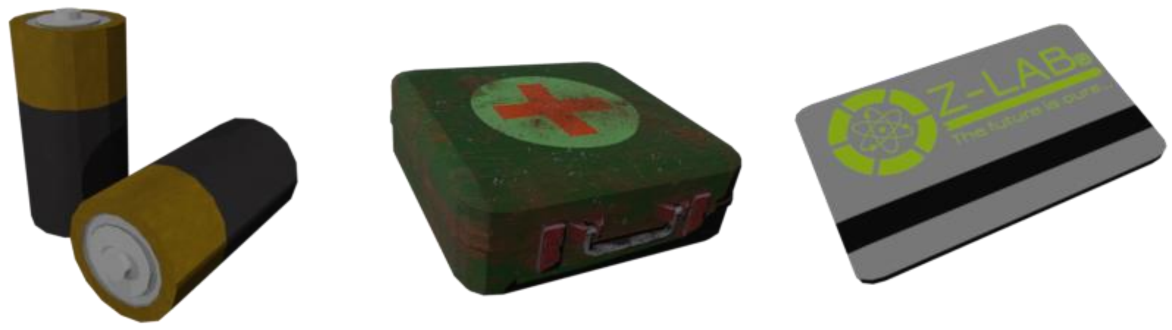


Figure 31 - Item sprites

5.5 User interface

With all the sprites ready, I created the user interface. The base component for any UI elements in Unity is the Canvas component. For the main menu UI, I created a Canvas, which automatically creates an empty object with Rectangle Transform, Canvas, Canvas Scaler and Graphics Raycaster components on it. These components are required in order for the UI to work in Unity. I added a child empty object to the Canvas object and added a custom script to it. This script handles all the button clicks in the menu. I created an empty object for every submenu in the menu and added them as children to the empty object with the script. In every submenu, I created an empty object for every item in it. If an item in the submenu is interactive, it has a Button component. The text of each button uses a specific style with keyboard key images. Every letter of the text consists of an image of a single keyboard key and a text element with the given letter.

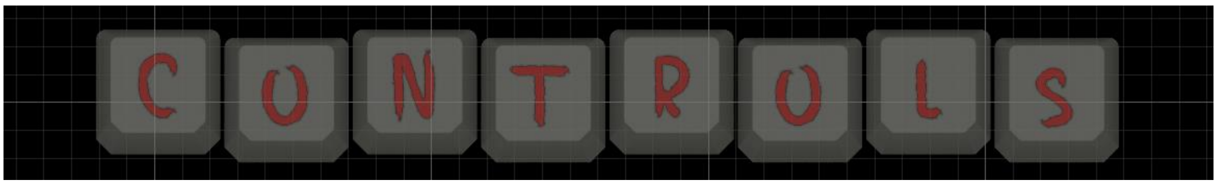


Figure 32 - Controls button using the keyboard key style

Using the modular components and other 3D models from my game, I also created a scene for the main menu. This scene serves as a background for the main menu and there are several flickering lights to add some dynamicity to it. I created the pause menu in a similar way. The pause menu does not use the main menu scene, but it is displayed on top of the scene the player is currently in when he or she pauses the game. The font used in the entire game is the Horror Memories font [31]. This font was created by Faisal Yudha Nugraha. I have obtained it from the dafont website [32].

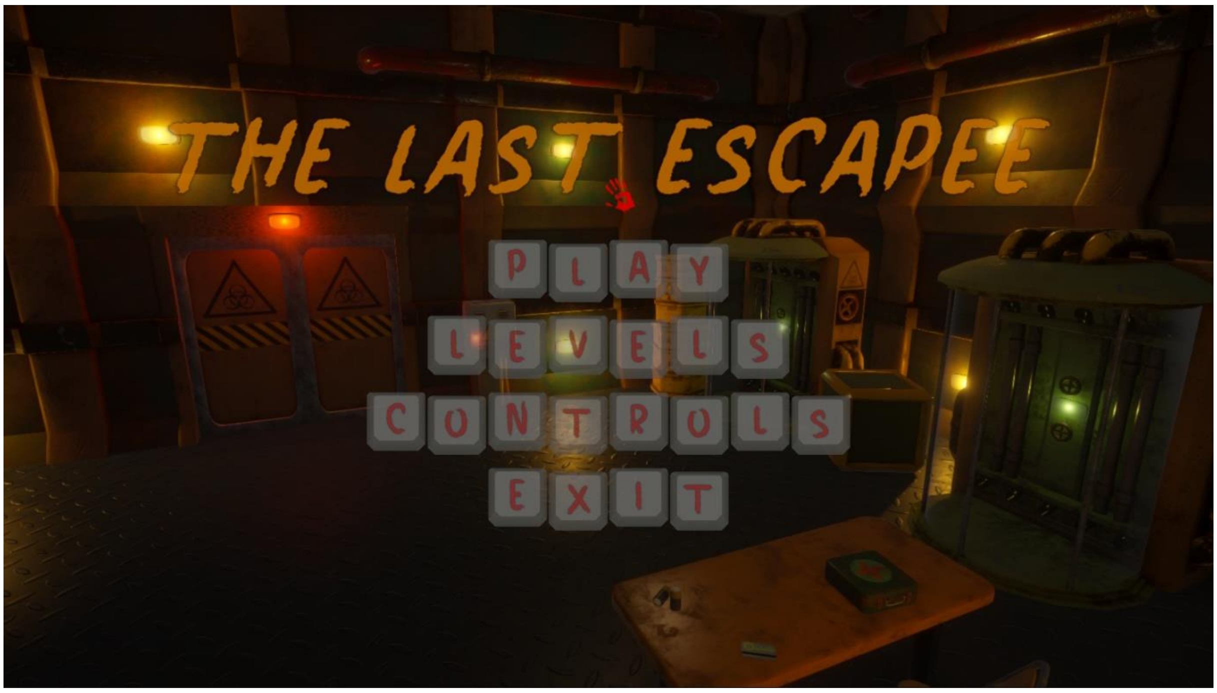


Figure 33 - The main menu

The inventory UI is separated into these sections: health, flashlight, and item slots. For each section, I created an empty object and added it as a child to the inventory UI object. I added two heart shaped sprites in the health section. One of them is slightly bigger, it is static, and it is not filled. For the smaller one, I selected the image type to be **filled** and fill method **vertical**. The fill amount is controlled from a script, and it serves as health indicator. The flashlight section is similar. The item slots section has 15 item slots in it. Each item slot is an empty object with an item slot script, an inventory slot sprite, and a button. When the item slot is filled with an item, the corresponding button uses the item's sprite as its source image.

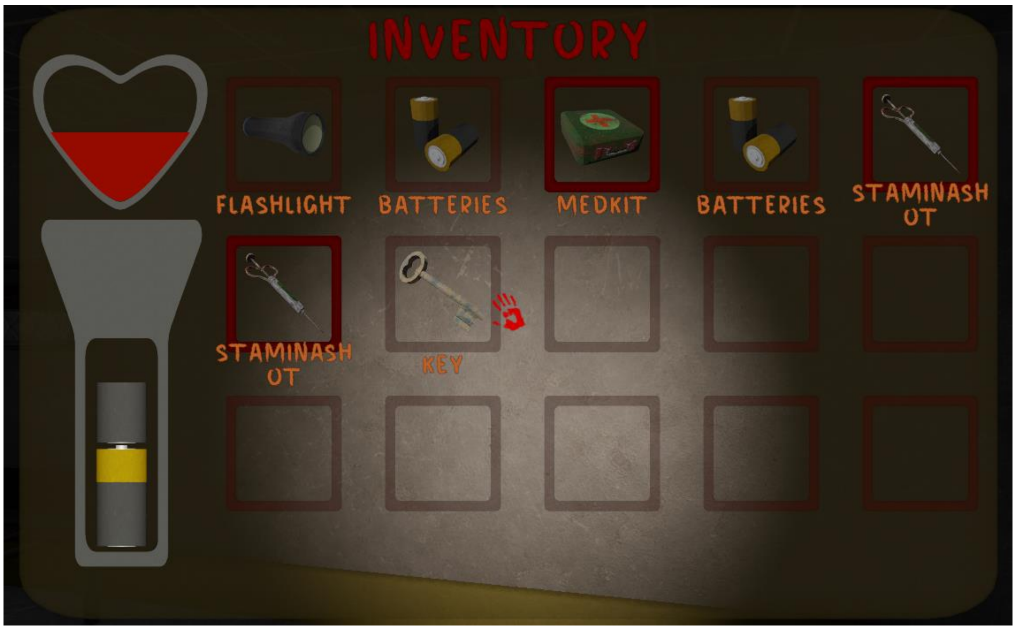


Figure 34 - The inventory

5.6 Scripts

This section mentions the scripts I created for my game. Scripts add functionality to the game and define objects' behavior. In Unity, a script is a piece of code written in C# programming language. Although it is not a necessity, scripts in Unity are often attached to objects and work just like components.

5.6.1 General scripts

The GameManager script handles basic game logic like pausing the game or player's death. This script also allows other scripts to access the game's main variables such as health, stamina, battery level or movement speed.

The PostProcessing script handles all script-controlled post-processing. It allows other scripts to easily access the post-processing overrides and change its values using this script's functions.

The LevelTransition script handles level transitions. At the end of each level, there is an empty object with a collider component. This collider does not affect physics, however, when the player collides with it, the LevelTransition script saves the player's inventory items and game variables. The script then loads the following level's scene and transitions the items and variables.

The SceneTransitionData script serves as a structure that encapsules the inventory items and game variables. It is created when the player leaves the previous level. When a new level is loaded, it is used to set the game variables and inventory items.

The DisableDirectionalLight script disables any directional lights present in the scene. TLE does not have any use for directional lights, however, I used directional lights to make designing levels easier.

The FPSLimit script sets the application's target framerate. TLE does not limit the target framerate to the screen's refresh rate, which is often 60Hz. This script allows for higher FPS (up to 100), but at the same time it does not allow wasting resources by having ridiculously high FPS on high-end hardware.

5.6.2 Player controls

The PlayerMovement script handles the player movement and everything related to it. The player is always in one of the following states: normal, sprinting, crouching or uncrouching. This script takes keyboard input. Based on the pressed keys, the script changes the player's state. Based on the state, player's position and facing direction, it computes magnitude and direction of a movement vector in 3D space. This vector is then passed to the Character Controller component. This component then handles the player's movement in the scene.

The LookAround script handles the player's facing direction. It takes the mouse input. Based on the mouse input and the player's current rotation, this script computes the resulting rotation and rotates the player's transform component accordingly.

The FlashlightScript handles the flashlight, its flickering light, and its battery consumption.

5.6.3 Interaction

The FindInteraction script finds items and interactable objects in the scene. It uses Raycast to determine whether the player can pick up an item or interact with an interactable object. It casts an invisible ray from the camera (player's eyes) in the same direction as the camera's forward vector. The length of this ray is determined by the *maxPickupDistance* variable. When the player can pick up an item or interact with an object and chooses to do so, this script adds the item to inventory or calls the *Interact()* method of the corresponding interactable object.

The PickupObject script handles moving moveable objects in the scene. Similar to FindInteraction, it also uses Raycast to find out whether the player can move an object. When the player presses left mouse button and there is an object that can be picked up in front of him, the *pickupObject()* function is called. This function creates an empty object named *holder* and adds the moveable object as its child. It also disables the moveable object's gravity. The *holder*'s position is updated in every frame based on the player's position and facing direction and the moveable object changes its position with it. Because this script does not directly set the moveable object's position, the moveable object is still affected by collisions. When the player releases left mouse button, or the moveable object collides in a way that it is too far from the *holder*, the *dropObject()* function is called. This function returns the moveable object back to its place in the scene's hierarchy, enables its gravity and destroys the *holder* object.

The Interactable script serves as a parent class for other interactable classes. It has variables that are used by all the other interactable classes, and it also has virtual methods that can be overridden.

The ElectricBox script handles electric box interaction. Depending on whether the electric box is currently turned on or off, it enables or disables possible interaction for the *interactingGameObject*.

The OpenGate script handles gate opener's interaction. When its *Interact()* method is called, it opens the connected *Gate* and rotates the gate opener wheel.

The OpenSlideDoor script handles the slide door interaction. When its *Interact()* method is called, the script either opens or closes the slide door, depending on its current state.

The SwitchLamp script handles the switch interaction. When its *Interact()* method is called, the script either enables or disables all lights which are children of the assigned *lampParent* object.

The UnlockDoor script handles the door interaction. When its *Interact()* method is called, the script checks whether the player has a key. If he or she does, the door is unlocked, and the key is consumed.

5.6.4 Enemies

The EnemyChase script handles enemies. Each enemy has one EnemyChase script attached to it. An enemy is always in one of the following states: patrol, chase or explore area. There is a set of patrol points in every level. Every patrol point covers a part of the level, usually an entire room or its part. All these patrol points combined with the paths connecting them cover the entire level. The default state is patrol. When in patrol state, the enemy is randomly wandering in the level. The enemy randomly chooses one of the patrol points and sets it as its destination. Once the enemy reaches its destination, it randomly chooses next patrol point and walks towards it again. The enemy switches to chase state if it sees the player, hears the player or sees the light of player's flashlight. When the enemy is in the chase state, it continuously updates its destination to match the player's position or the visible flashlight's light position. Whether the enemy can see the player depends on three factors.

- The player needs to be in the enemy's field of view. If the enemy is facing the opposite direction of where the player is, it can not spot the player.
- There must not be any obstacle in the way. This is determined by casting a ray from the enemy's eyes to the center of the player's body.
- The distance between the player and the enemy must be lower than the *visibilityDistance*. The *visibilityDistance* changes based on the *playerVisibility*.

Whether the enemy can hear the player depends on the distance between the enemy and the player. Implementing hearing in a way that it would reflect how sound is transmitted in real life would be extremely difficult and also performance heavy. Therefore, I have decided to approximate it by using two spheres. The enemy has an imaginary sphere around it with the *hearingRange* being its radius. This sphere demonstrates the area in which the enemy can hear any noise. The player also has an imaginary sphere around himself or herself with the *noiseRange* being its radius. This sphere represents the area where the noise made by the player can be heard. The *noiseRange* changes constantly, based on the player's movement speed. If the distance between the player and the enemy is smaller than the sum of the two radiuses (the two imaginary spheres collide), the enemy can hear the player and sets the player's position as its destination. If there is a wall between the player and the enemy, the *noiseRange* is significantly lower.

Whether the enemy can see the light of the player's flashlight is determined by Raycast. The light of the player's flashilight is a cone. It is approximated by seven rays coming from the light. One is in the center of the cone, the other six are on a circle which is the cone's base.The *flashlightHitPoints* are the points where these rays collide with the environment. For each of these points, if it is in the enemy's field of view, a ray is cast from the enemy's eyes to this point. If the ray does not collide with any object on its way, the enemy can see the flashlight's light and runs towards it. If the player shines directly on the enemy, it also starts chasing the player.

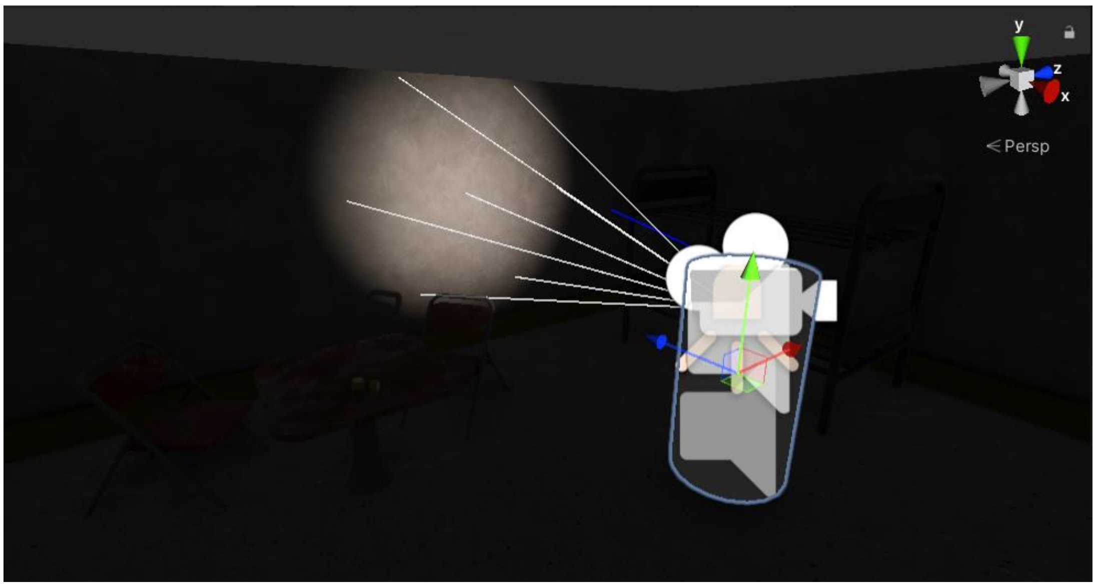


Figure 35 - The light of player's flashlight approximated by seven rays

The EnemyUtilities script is a helper script for enemies. It computes the *playerVisiblity*. Each light in the scene has a certain range. If the player is in the light's range, the EnemyUtilities script casts a ray from the light to the center of the player's body to determine whether the light shines on the player. If it does, the *playerVisibility* is increased according to the light's intensity and the distance between the player and the light. This happens for every light in the scene. This can be seen in the figure below. Green ray represents a light that currently has an influence on *playerVisiblity*, red ray represents a light that does not have influence on *playerVisilibty*. Lights which are too far from the player do not cast any rays. This script also computes the *flashlightHitPoints*.

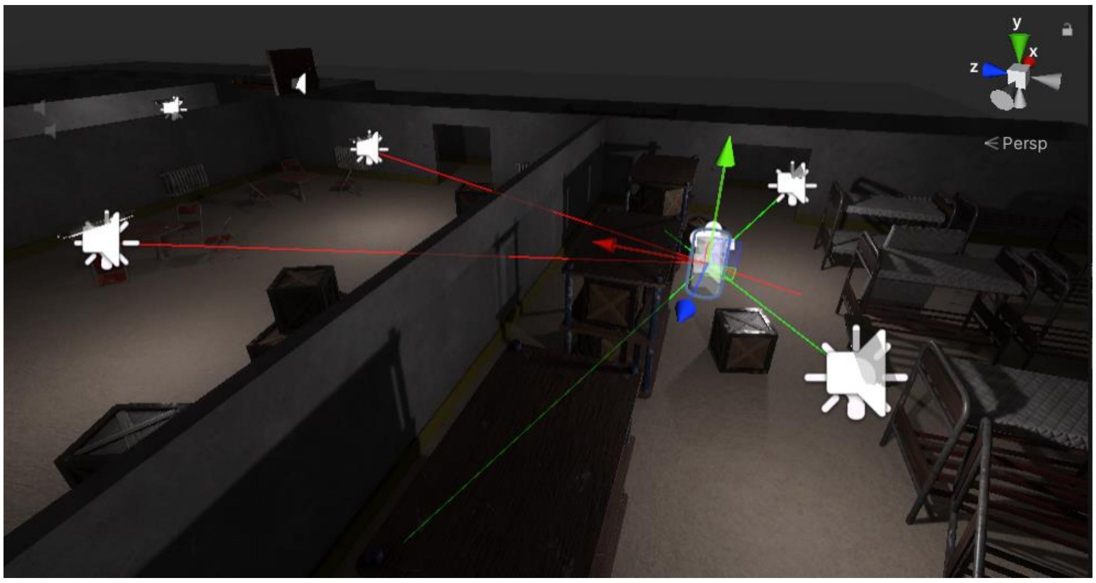


Figure 36 - Scene lights and their influence on playerVisibility

5.6.5 Inventory and items

The Inventory script handles the logic behind inventory. It uses a *List<Item>* to store the items and implements methods such as *addToInventory*, *removeFromInventory* or *hasItem*.

The Item script serves as a parent class to other items. It has variables that are used by all the other item classes, and it also has virtual methods that can be overridden.

The FlashlightItem, the BatteriesItem, the MedkitItem, the StaminaShotItem, the KeyItem and tbe SecurityCardItem are scripts representing items. Each of these scripts inherits from the Item script, stores necessary values and implements the corresponding item's functionality.

5.6.6 Audio

The AudioManager script handles most of the game's audio. It implements the *playAudio* method, which other scripts call in order to play an audio. This method is overloaded and has three different variants. Depending on which one is used, the AudioManager either plays the audio at the player's position, at the provided position or using a provided AudioSource component. This script also handles the player's and enemies sounds as well as ambient sounds. It is attached to an empty object in the scene, and it has control of all the audio files. All of the audio files were obtained from freesound website [33]. Specifically, the credit goes to the following authors: 137SOUNDS, AderuMoro, birdOfTheNorth, Breviceps, dragonmin, DrMinky, DWOBoyle, EminYILDIRIM, EverHeat, FullMetalJedi, InspectorJ, Jacco18, joedeshon, LightYarn, meroleroman7, missozzy, MrPokephile, nebulasnails, newlocknew, OtisJames, Sami_Hiltunen, Sergenious, SpaceJoe, stereostereo, stk13, tonsil5 and VKProduction.

The Audio script encapsules the AudioClip class alongside few other variables like volume or pitch. The AudioManager script has a list of Audios which are defined in the inspector.

The CollisionSound script handles the collision sound of moveable objects. It is attached to every moveable object in the game, and it uses the *OnCollisionEnter* method to randomly play one of the collision sounds whenever it collides with another object.

The DoorAudio script handles the squeaking door audio sound. Whenever the door is moving, this script plays the squeaking sound. The volume changes based on the door's angular velocity.

The EnemyAudio script serves as a structure for the variables required for the enemy audio.

5.6.7 User interface

The InventoryUI is the script handling the logic behind the game's UI and HUD. It enables, disables and updates the UI elements based on callbacks. It also handles the cursor as well as the inventory UI elements.

The MainMenuVM script handles the logic behind the main menu UI.

The Tutorial script handles the tutorial. It shows and hides the tutorial UI elements based on certain events happening in the game. It is only present in the first level.

The UIHoverText, the HealthHoverText and the FlashlightHoverText handle the info text displayed in inventory when the mouse is over the given UI element.

The HoverButtonHighlight script handle button highlight. When a mouse is over a given button in menu, the button's size increases and it also plays a sound.

5.7 Animation

Because TLE is a first-person game, where at no point in the game there is any part of the player's body visible, I did not have to create any player animations. The only animations I had to make were the enemy animations. I created three different enemy animations - the walk animation, the run animation and the attack animation. I used Unity's built-in animation tab and animator. I created all the animations in the animation tab. I defined positions of the rig's joints for several keyframes. The animation clip then interpolates between these keyframes. Using the Animator, I defined the default animation as well as transitions between animations.

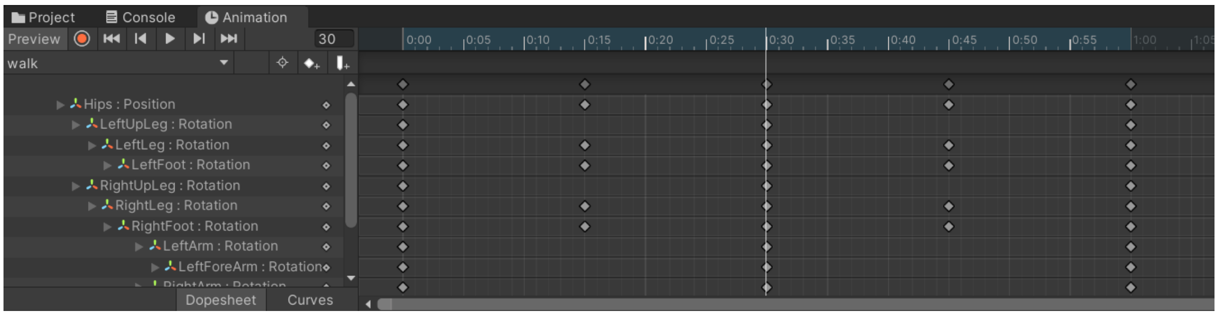


Figure 37 - Animation tab with keyframe animation

5.8 Optimization

The vast majority of optimization in TLE comes from its design. Because I created all the 3D models myself, I was able to decide how much detail there will be in the model's mesh and how many polygons each model will have. I tried to keep a reasonable balance between having good-looking 3D models and not using a high number of triangles. The game is separated into multiple scenes. Although this means that there is a short load time when switching between levels, it helps optimize the game significantly. If I had used only one scene, I would not have been able to put as much content in it.

There are objects in the game which can be moved anywhere in the scene. Due to that, the game only uses realtime lights. Realtime lights are costly when it comes to resources, especially when casting shadows. Their lighting is calculated every frame. Having a lot of realtime lights in the scene can affect the performance heavily. In order to reduce the performance hit, I set a distance limit for shadows and volumetrics. When the distance between the player and the light is greater than this distance limit, the light no longer casts shadows nor has impact on volumetrics. I tried to set this limit in a way that it is barely noticeable for the player, but it also helps optimize the game considerably.

There are a lot of other optimization techniques used in computer games such as level of detail (LOD) or baking detail into normal maps from high detailed mesh. I did not use them simply due to lack of time. There is undeniably a lot more optimization that can be done in TLE as I have only scratched the surface when it comes to applying optimization techniques.

6 Results

This chapter presents the results of the process of creating my game. It shows the finished levels and some of the assets.

6.1 Models

Throughout the process of creating my game, I created more than seventy 3D models. These models differ in size, shape and their purpose. Some of these models were relatively simple to create and did not take a lot of time, the other ones were more advanced and took more time to create.

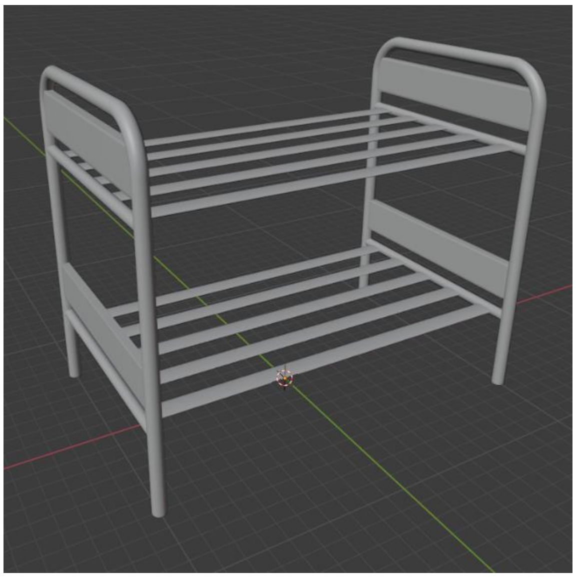


Figure 38 - Bunk bed model

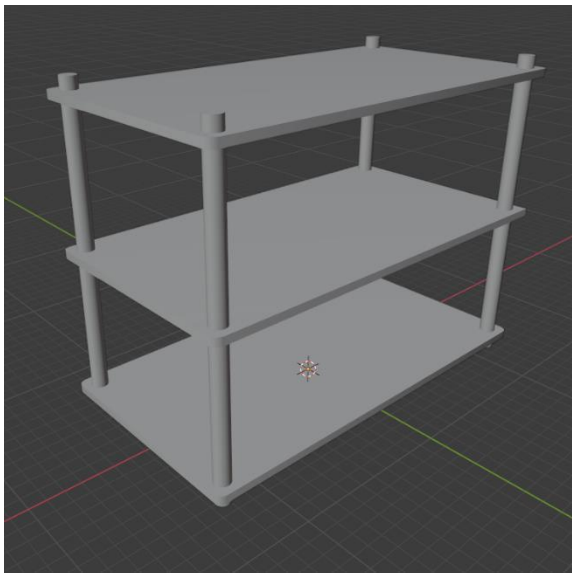


Figure 39 - Rack model

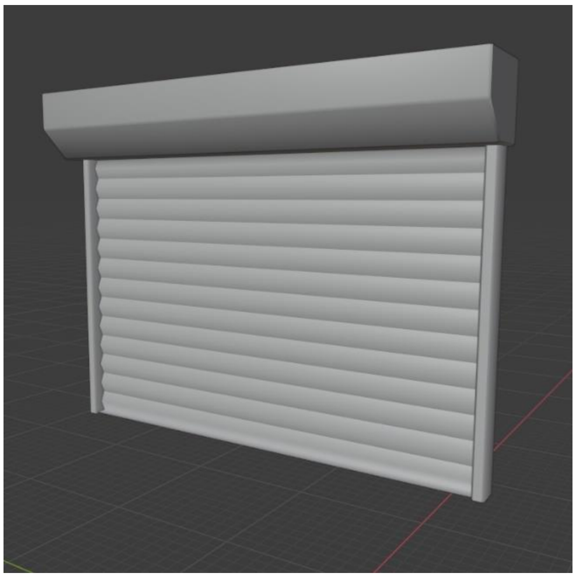


Figure 40 - Blinds mechanism model

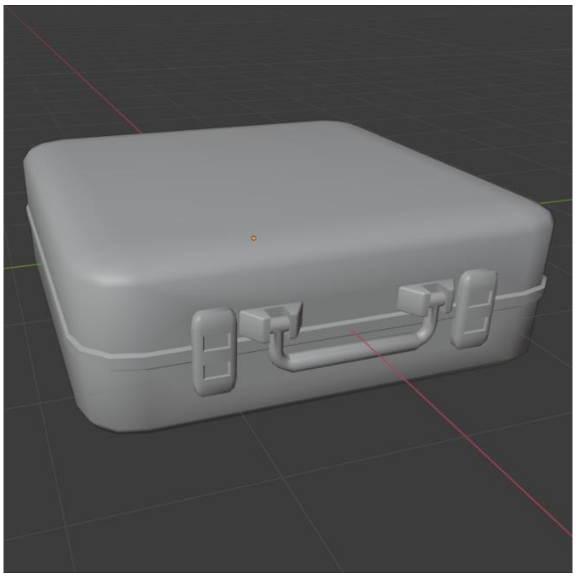


Figure 41 - Medkit model

6.2 Textures and materials

For each 3D model, I created a set of three textures. These three textures are **base map**, **mask map** and **normal map**. **Base map** defines the model's base color. **Mask map** defines the properties of the model's surface. It combines multiple textures into one, where every channel represents a different texture. The red channel stores the metallic texture, the green channel stores the ambient occlusion texture, the blue channel stores the detail mask texture, and the alpha channel stores the smoothness texture. I did not use detail mask in my textures. **Normal map** adds more detail to the model. From these textures, I created a material and applied it to the model.

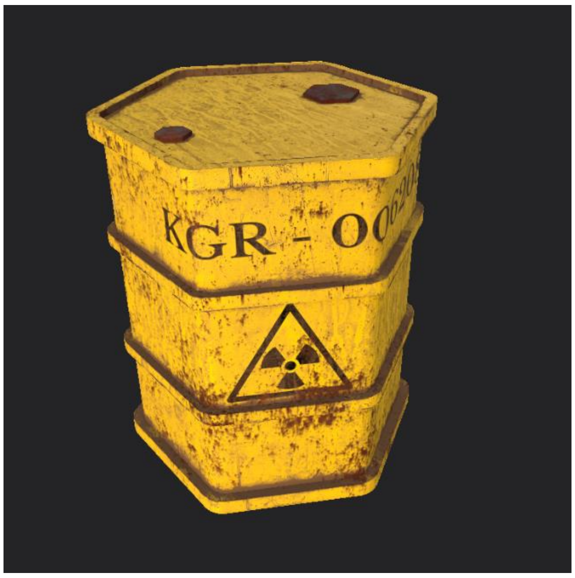


Figure 42 - Barrel with applied material

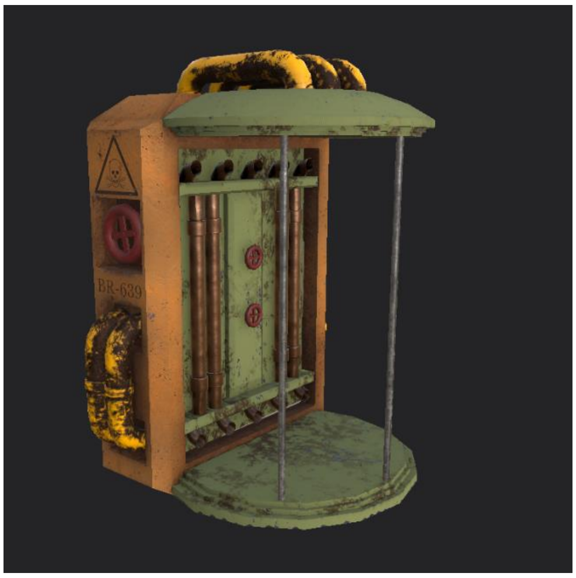


Figure 44 - Incubator with applied material

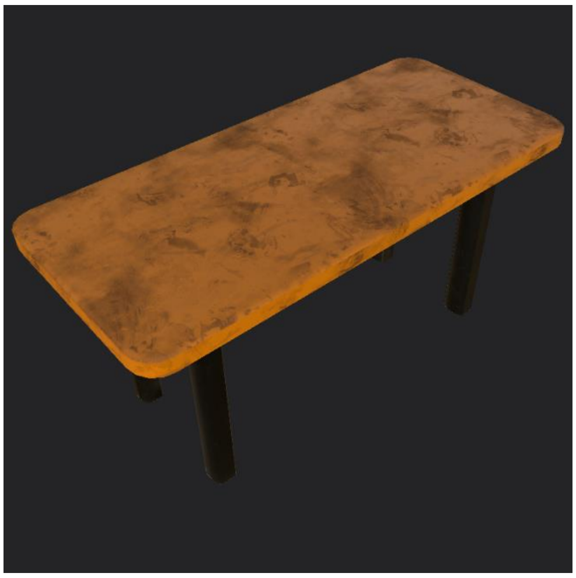


Figure 45 - Lab table with applied material

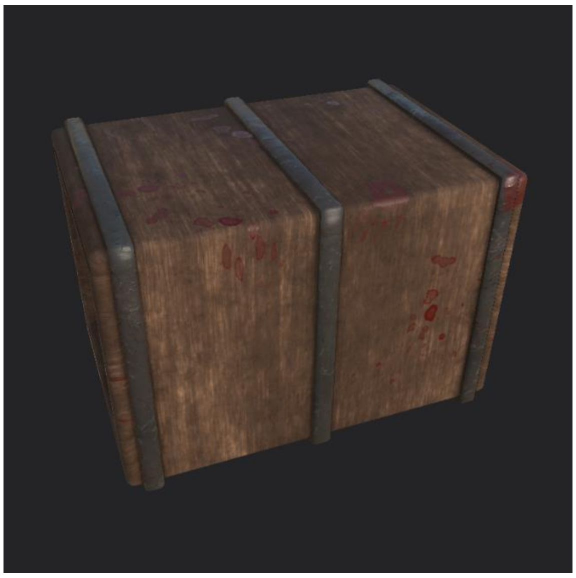


Figure 43 - Chest with applied material

6.3 Levels

I created three levels for my game. The mechanics in all three levels were similar. The levels differed in their size, structure and appearance. The following three figures show the levels' corresponding floor plans.

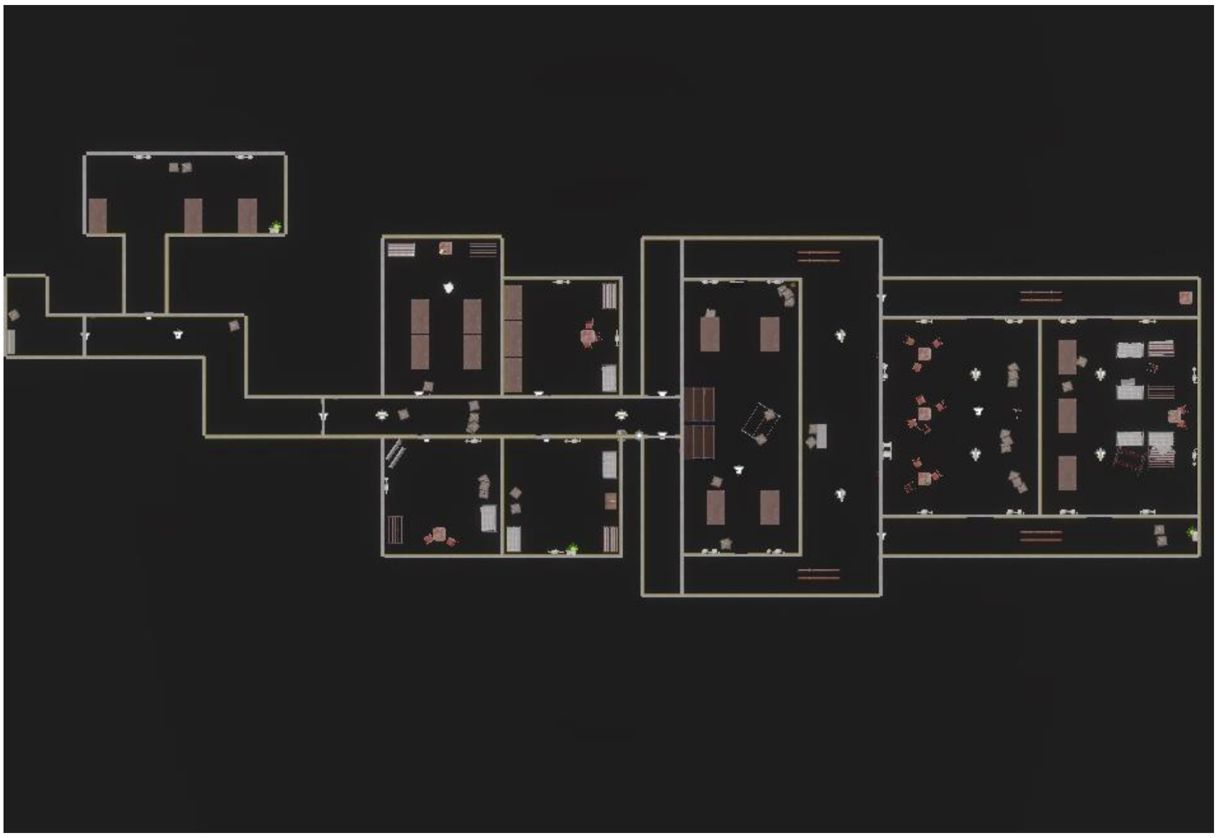


Figure 46 - Level1 floor plan

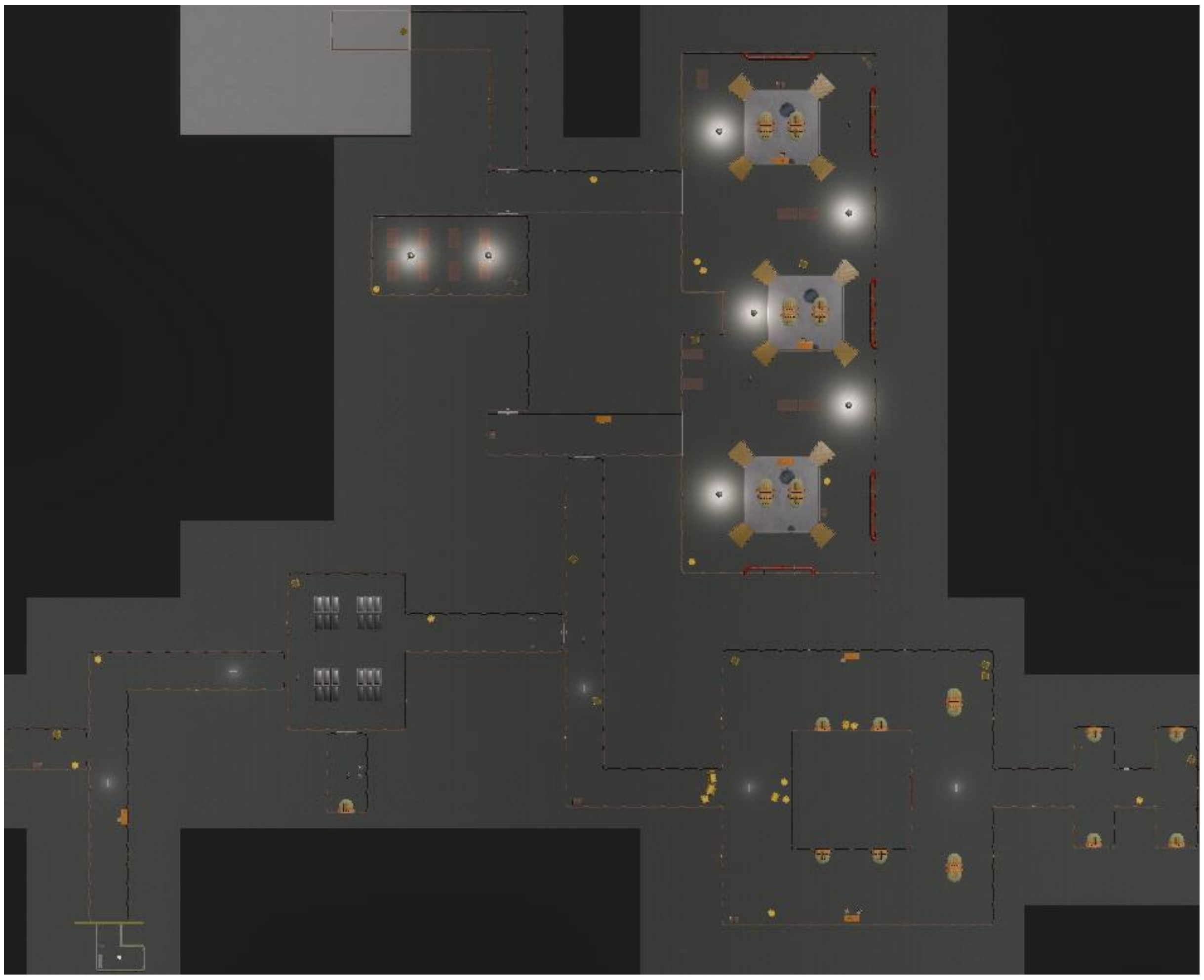


Figure 47 - Level2 floor plan

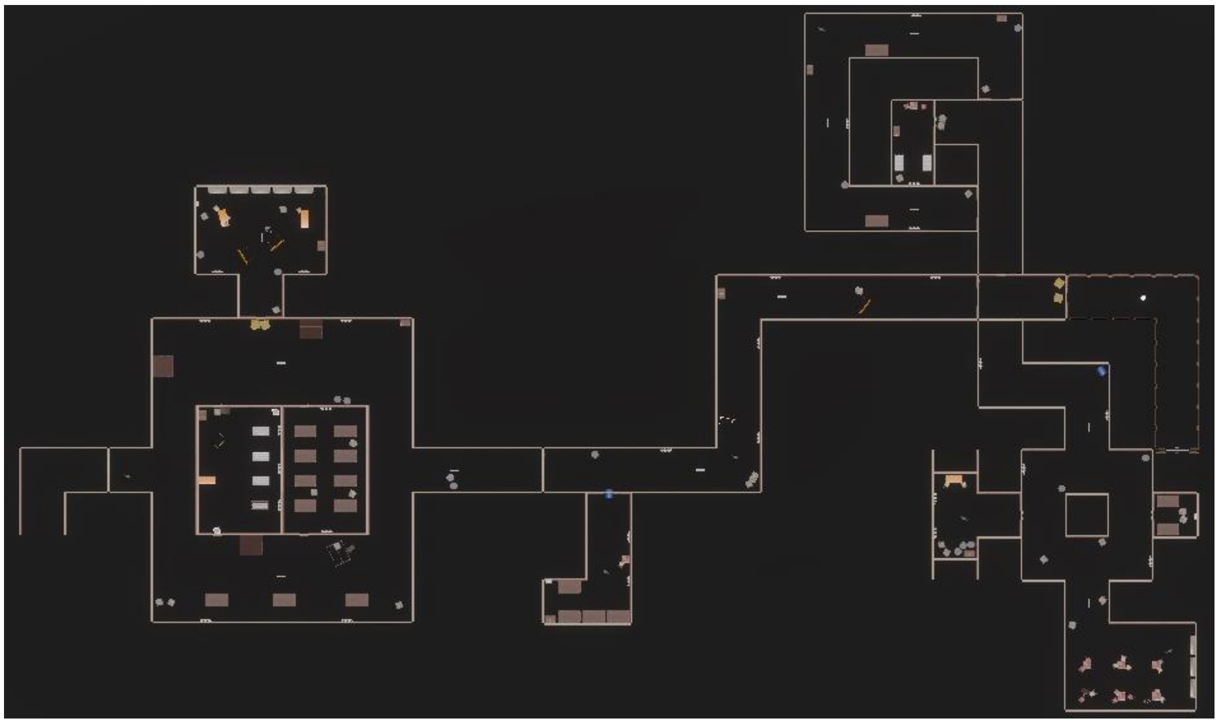


Figure 48 - Level3 floor plan

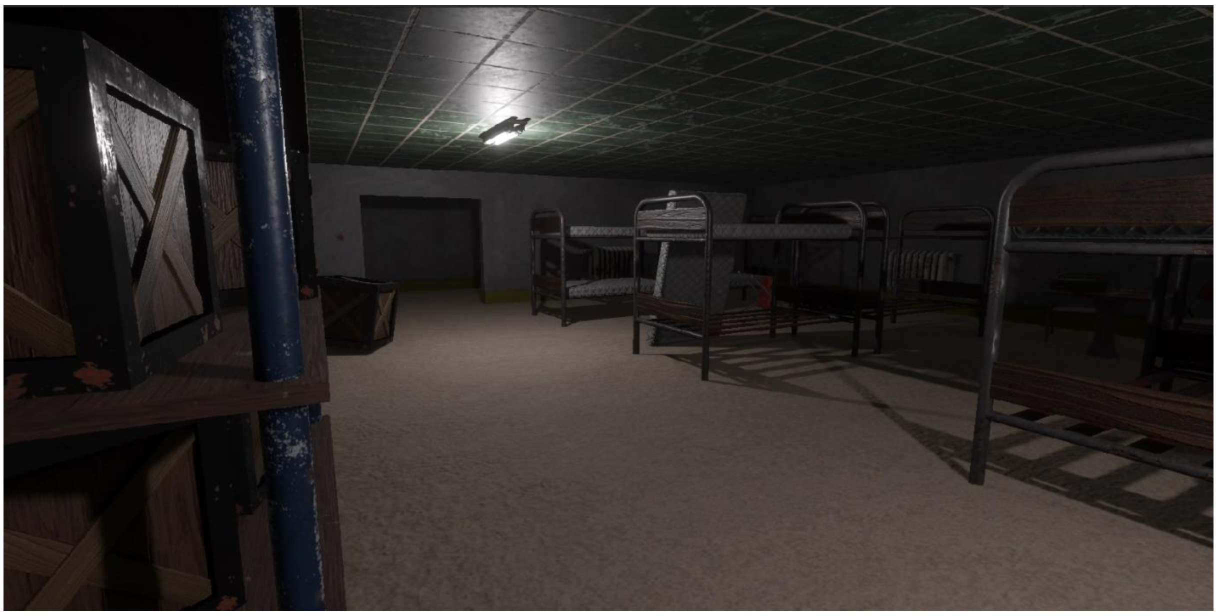


Figure 49 - Screenshot from the first level

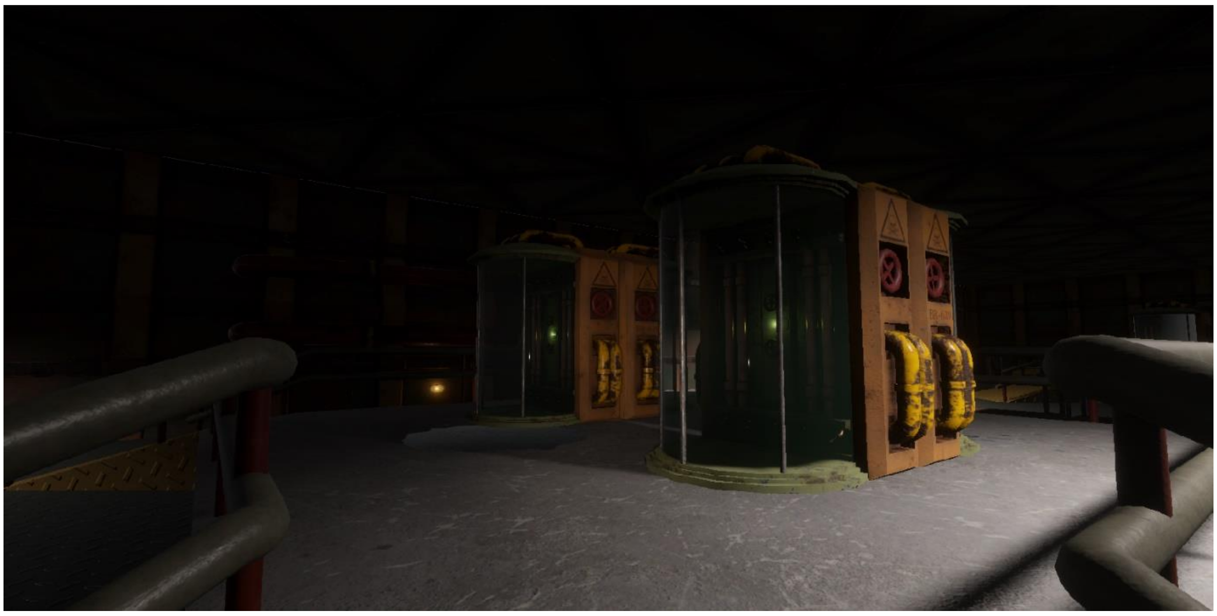


Figure 50 - Screenshot from the second level

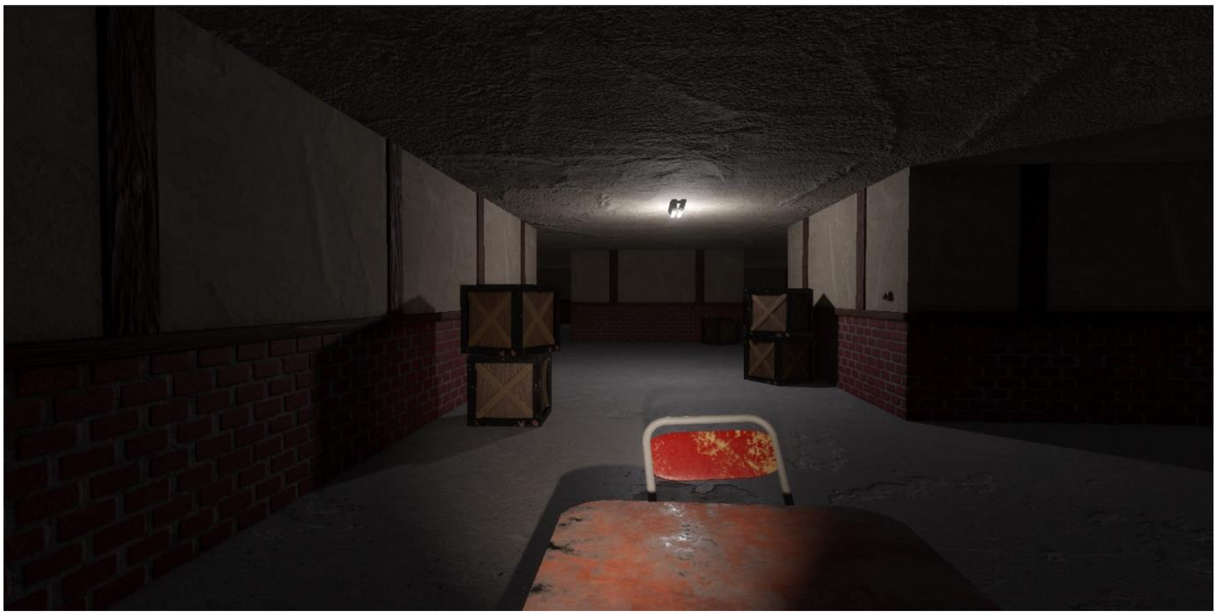


Figure 51 - Screenshot from the third level

7 Testing

This chapter focuses on testing the game. Throughout the game's development, I was continuously testing it. I was trying out all of its features as well as the gameplay. While this testing helped me reveal a lot of complications, bugs and design imperfections, it was still extremely subjective. Because I designed and implemented the game and its mechanics, I knew how things work, which often limited my ability to objectively test it. That is why I also tested the game with other people. These people did not know a single thing about the game, which is exactly what I needed. I wanted to find out how do others react to the game, whether they find it hard, intuitive, entertaining, and so forth.

I tested the game with six people. This testing was conducted in later stages of the game's development. Three of these people tested the game remotely, sharing their computer screen and communicating with me using VoIP software. The reason behind that was that I wanted to find out whether there will be any complications with running the game on a different hardware. The testing with the other three was conducted in person. All the testers were presented with a working version of the game with most of its features already finished. Every tester tested just the first level of the game, even though all three levels were already finished. Playing through all three levels could take a lot of time, which some of the testers did not have. The game mechanics are almost identical across all three levels. The main difference between the levels is the visuals. Because I wanted the testing to bring consistent results, I decided to limit the testing to the first level only.

In order for the testing to bring consistent results, I also had to use the same approach every time I tested it with one of the testers. I established a testing routine according to which I then tested it with every tester. The first part of this routine was introducing the tester to the testing process and letting him or her know what is required of him or her. Every tester was guided to think loud during the testing and describe what are they currently doing as well as why they are doing it. The only thing I told the testers about the game was that it is a survival horror game. I did not elaborate any further, because I was interested in how intuitive the game is and whether people will understand what they are supposed to do in it. The second part of the testing routine was the testing itself. In case the tester was present in person, I sat next to him or her and I observed him or her playing the game as well as his or her reactions to it. In case the tester was testing the game remotely, my observance of their reactions was limited to verbal only. I was making notes of their observations, reactions and struggles. The last part of the testing routine was getting feedback from the tester. This was done in two forms. Firstly, I asked every tester about the game and their overall impressions. Secondly, I also gave every tester a questionary with several questions which they answered. The following sections describe testing the game with six testers.

The recommended system requirements are as follows: Windows 10, Intel Core i5-4460 processor or better, 8GB RAM, dedicated GPU GeForce GTX 1650 SUPER 4GB or equivalent, headphones, keyboard, mouse.

7.1 First test

The first test happened remotely. I was observing the tester play the game while he was sharing his screen. After exploring the main menu, the tester started playing the game. He immediately understood the basic movement controls and started navigating in the game's first level. After a while, he noticed that a cursor sometimes appears on the screen, and it caused a little confusion. This was because he either did not pay attention to the help displayed in the top part of the screen or because he did not notice it at all. After a while, this confusion was cleared. He then picked up a flashlight in the starting room. This revealed the first bug in the game. He did not know how to turn the flashlight on, so he opted to pause the game and open the controls menu. The flashlight control was missing, therefore I stepped in and let him know which key is used to control the flashlight. He explored the first room and collected several items. After a relatively short amount of time, he managed to find the key needed to escape the first room. After a while, he managed to explore the starting area, find and collect some items, and open a gate, allowing him to explore the first level further. His encounter with the first enemy clearly caught him off guard. Based on his verbal reaction and the sudden screen shake, it was clear that the enemy scared him. Although he managed to escape enemies a couple of times, he managed to die four times. As he was becoming more familiar with the game's mechanics, he started to use moveable objects to trap enemies in rooms. On his fourth attempt, he was noticeably better at playing the game and he was successfully avoiding, escaping, or trapping enemies. The first testing also revealed a bug with multiple screens, as the mouse was able to leave the game window and move to the second screen. According to the questionary, the first tester has little experience with survival horror games, and it is not a genre that he likes a lot. Despite that, his overall experience playing the game was good. He liked TLE's mechanics, visuals, and user interface. The thing he enjoyed the most was trapping creatures in the rooms. On the contrary, he did not find the controls very good, and he found the game quite hard. The thing he was annoyed with the most was opening the doors.

7.2 Second test

The second test was conducted in person. I was sitting next to the tester while he was playing the game. According to the questionary, he plays games regularly, he plays survival horror games quite often and it is a genre that he really likes. He immediately started playing the game without exploring the main menu. After a brief moment, he used the ESC menu in order to check the game's controls. He easily managed to comprehend the game's controls and basic mechanics. He escaped the first room after finding the key and started exploring the level further. After a while, he wanted to recharge his flashlight by using the batteries item. This revealed a bug in the game. When the battery level is exactly 50, he was still not able to use the batteries item. This tester was using the inventory items in a smart way, and he was navigating in the scene with ease. He often barricaded himself in a room so he could safely explore it. He also managed to trap enemies in rooms several times. Although he died three times while playing the first level, he managed to successfully complete it. He found the game relatively hard, but he said he does not mind the deaths, because it makes the game challenging. According to the questionary and his verbal feedback, his overall experience when playing the game was amazing, he loved the visuals and the audio. He especially pointed out the audio, running from enemies and looking for new ways to beat the enemies after dying. He disliked the health indicator in inventory, and he had a hard time jumping over boxes when being extremely close to it. He suggested adding a key binding for items such as medkit or stamina shot.

7.3 Third test

Similar to the first test, the third test was conducted remotely. According to the questionary, the tester plays games regularly, but does not have a lot of experience with survival horror games. He started by exploring the main menu and immediately said "I really like the mouse cursor". After some time, he started playing the game. He paid attention to the tutorial and understood the basic controls immediately. He managed to find the key and his way out of the first room without any trouble. As he opened the first door he said he really appreciates the mechanic where he has to drag the mouse to simulate the door movement. Compared to the previous two testers, he chose to practice the stealth tactic more. After his first death, he realized that it is important not to be spotted by enemies. He mentioned that the overall visibility is quite good and that he often does not need to use the flashlight at all. He managed to discover a bug where the enemy was able to hit the player through a door. His playstyle was prudent, which resulted in him often being able to sneak around enemies without being seen. His experience playing the game was very good. He said the sounds were amazing and he liked the controls and game mechanics. He especially liked the pathing of the AI and the usage of doors as tools to trap enemies in rooms. He disliked the absence of an item which the player could throw in order to bait an enemy to go to a certain area.

7.4 Fourth test

The fourth test was conducted in person. The tester plays computer games regularly. He does not play a lot of survival horror games, as it is not his favorite game genre. After going through the main menu, he said he really likes it. He also noticed the flickering light effect in the main menu scene and found it very nice. He understood the movement controls right away. The tester collected the flashlight and expected it to turn on automatically. He paid attention to the tutorial and really enjoyed the ability to move objects in the scene. At first, he tried to drag batteries in the inventory and put them onto the flashlight, but he quickly realized that he just needs to click on the item to use it. He managed to make his way out of the first room and started exploring other parts of the first level. While manipulating with a mattress object, he said that it behaves a bit too rigid. Based on his reactions, it was obvious that he was scared by the enemies on multiple occasions. Because he did not realize opening the inventory slows game time, he did not use stamina shot while being chased. After his first death, he complained about this issue. I stepped in and notified him that opening the inventory slows the game time. He liked the game's tutorial and found it very helpful. He sometimes struggled with opening the doors from the corridor. His overall experience when playing the game was good. He found the game mechanics and user interface amazing, and he especially enjoyed trapping enemies in rooms. On the contrary, he found the game very hard.

7.5 Fifth test

The fifth test was also conducted in person. The tester plays computer games regularly and he has adequate experience with survival horror games. Starting in the main menu, he immediately noted that he really likes the menu music. Upon loading the first level, he paid attention to the tutorial and understood the controls without any problems. He enjoyed the ability to move the objects in the scene. After fully exploring the first room, he managed to find the key and move to the next part of the first level. While moving the mattress, he noticed that it sometimes drops on its own, making it hard to manipulate. He tried pushing moveable objects with his body, but shortly after he realized that it is not possible. He also disliked that the moveable objects keep their rotation. When using the inventory, he noticed that each item has its description, which he found nice and informative. In relation to the stamina shot item, he said that there should be stamina indicator in the menu. His playstyle was slow paced, he used the stealth mechanic a lot, which worked in his favor. When being chased by the enemies, he often found it hard to lose them, which sometimes resulted in his death. He also noted that the visibility is pretty good and that he often does not need to use the flashlight to navigate in the level. He used moveable objects and doors to block enemies in rooms or corridors. His experience when playing the game was good, he especially liked the sound effects and the game mechanics. On the other hand, he thought that the enemies can hear way too good and that there should be less enemies.

7.6 Sixth test

Like the first and the third test, the sixth test was conducted remotely. According to the questionary, the tester plays games regularly, but does not have a lot of experience with survival horror games. He checked the game controls in the main menu and started playing the game right away. After getting familiar with the movement controls, he managed to find the key and move on from the first room. He liked the ability to move objects in the scene, but he also sometimes struggled with opening the doors. On several occasions, he did not expect the interaction with an enemy and according to his reaction, he clearly got scared. He died shortly after the first interaction with an enemy because he did not sprint at all. After this death he checked the controls again and realized that he can sprint. While playing, he compared TLE to Amnesia. Unlike most of the other testers, he utilized a very fast paced playstyle, which to my surprise also worked pretty well. He was really impressed by the sounds and said that it makes for a great atmosphere. He also utilized the doors and moveable objects as a tool to trap enemies. According to the questionary, he found the visuals and sounds amazing. He found the fast-paced gameplay very entertaining, but he disliked the struggle with opening doors.

7.7 Testing summary

The testing has proven to be extremely helpful. It helped me reveal multiple bugs. I managed to address and fix some of the bugs. I fixed the bug with multiple screens by choosing different cursor lock mode, which forbids the mouse from exiting the game window. I also fixed the issue which was making it complicated to jump over obstacles when the character was right next to it. The issue was a wrong choice of step offset in the Character Controller component. When the player jumps, it now sets the jump offset to zero and it returns the old value once the player lands. I added the flashlight control description to the controls menu as it was missing. The next thing I managed to address was the issue with using the batteries item when the battery level is exactly 50%. I fixed this issue by simply replacing the strict inequality with a regular inequality in the code. After testing the game myself, I found out what caused the bug where enemies were sometimes able to hit the player through the door. This was caused by a tiny gap between the door and wall collider. I adjusted the colliders in a way that this should not happen again. Because usage of flashlight is something that should be required to complete the game and some testers stated that they did not have to use it at all in some parts of the game, I added a settings section to the game. This allows users to adjust the brightness as well as volume level.

It seemed that the difficulty level I had set for the enemies was a bit too difficult, therefore I decided to slightly worsen their hearing ability. To my surprise, the overall feedback was positive, and everyone enjoyed the game, even though most of the testers do not consider survival horror genre to be their favorite. The game's visuals and sounds received overwhelmingly positive response. After finishing the testing, some of the testers also asked me if they could finish the game's other levels.

8 Conclusions

The main goal of this thesis was to create a survival horror game. I began by analyzing the principles of games. The next part of the analysis was to research computer games in general, and the way they are created. Followed by that, I analyzed the survival horror game genre. I examined already existing computer games from this genre and analyzed the game mechanics used in them. Based on this analysis, I deemed which of these mechanics are suitable for my game.

The next step was to design my own game based on the previous analysis. I designed a first-person 3D survival horror game. The game is situated in an underground complex. The player plays for a character named Cody, who was imprisoned. The main goal of the game is to escape the said underground complex without being killed by hostile creatures. I described the game's controls, user interface, enemies and mechanics in detail.

Following the game design, I analyzed Unity. Unity is the game engine I chose for my game. I began by analyzing Unity's user interface. Followed by that, I described the asset workflow in Unity. I inspected Unity's most commonly used components as well as components important for the creation of my game. I briefly described rendering and render pipelines available in Unity.

The following step was to create the game. Based on the game design, I created a first-person survival horror game in Unity. I began by creating modular components and other 3D models. I created textures and materials for these modular components and models. With these assets, I created three separate levels for the game. These levels differ in their size and appearance. I created various scripts which define the game logic and enemies' behavior. I added multiple sounds to the game to help build up the atmosphere while playing.

The last step was to test the game. I tested the game with six people. Three of the people tested the game remotely while sharing their screen with me, the other three tested it in person. The testing revealed several bugs. I was able to fix some of these bugs, resulting in overall better gaming experience.

Bibliography

[1] Koster Raph. *Theory of Fun for Game Design*. Paraglyph, Incorporated, 2004. ISBN: 9781932111972

[2] Bycer Joshua. *Game Design Deep Dive: Horror*. Taylor & Francis Group, 2021. ISBN: 978-1-032-05806-1

[3] Wikipedia. *Resident Evil.* [online]. URL: https://en.wikipedia.org/wiki/Resident_Evil (visited on 4.5.2022)

[4] Wikipedia. *Outlast.* [online]. URL: https://en.wikipedia.org/wiki/Outlast (visited on 5.5.2022)

[5] Wikipedia. *Penumbra: Black Plague.* [online]. URL: https://en.wikipedia.org/wiki/Penumbra:_Black_Plague (visited on 6.5.2022)

[6] Steam. *Penumbra: Black Plague, image from the game.* [online]. URL: https://cdn.cloudflare.steamstatic.com/steam/apps/22120/0000007042.jpg (visited on 6.5.2022)

[7] Wikipedia. *Amnesia: The Dark Descent.* [online]. URL: https://en.wikipedia.org/wiki/Amnesia:_The_Dark_Descent (visited on 6.5.2022)

[8] Steam. *Amnesia: The Dark Descent, image from the game.* [online]. URL: https://cdn.cloudflare.steamstatic.com/steam/apps/57300/ss_fbf8289efe7ff6ce01d2e209f0300ecffb8fcfc4.jpg (visited on 6.5.2022)

[9] Wikipedia. *Indie game.* [online]. URL: https://en.wikipedia.org/wiki/Indie_game (visited on 6.5.2022)

[10] Urban Dictionary. *Jump scare.* [online]. URL: https://www.urbandictionary.com/define.php?term=Jump%20Scare (visited on 6.5.2022)

[11] HowLongToBeat. *Slender: The Arrival.* [online]. URL: https://howlongtobeat.com/game?id=13981 (visited on 6.5.2022)

[12] Steam. *Slender: The Arriva, image from the game.* [online]. URL: https://cdn.cloudflare.steamstatic.com/steam/apps/252330/ss_74c39a1d7b8523fa7fe5c1dbac97a81b89aa71b5.jpg (visited on 6.5.2022)

[13] De Nucci Ennio, Kramarzewski Adam. *Practical Game Design.* Packt Publishing, Limited, 2018. ISBN: 978-1-78712-179-9

[14] Bocan Viktor. *Kingdom Come: Deliverance prototype screenshot.* [online]. URL: https://cw.fel.cvut.cz/b191/_media/courses/b4b39hry/protected/02-design.pdf (published with author's permission, visited on 6.5.2022)

[15] Telezhkin Simon. *Low Poly tree 3D model image.* [online]. URL: https://cdn.hum3d.com/wp-content/uploads/2018/06/25/low_poly_tree_600_lq_0001.jpg (visited on 4.5.2022)

[16] Unreal Engine Documentation. *Modular Level and Component Design.* [online]. URL: https://docs.unrealengine.com/udk/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf (visited on 5.5.2022)

[17] Endsley Kezia. *Video Game Design.* Cavendish Square Publishing LLC, 2014. ISBN: 9781502601131

[18] Unity. *Unity Terms of Service.* [online]. URL: https://unity3d.com/legal/terms-of-service/software (visited on 9.5.2022)

[19] Unity Documentation. *Asser Workflow.* [online]. URL: https://docs.unity3d.com/Manual/AssetWorkflow.html (visited on 10.5.2022)

[20] Unity Documentation. *Scenes.* [online]. URL: https://docs.unity3d.com/Manual/CreatingScenes.html (visited on 10.5.2022)

[21] Unity Documentation. *NavMesh Building Components.* [online]. URL: https://docs.unity3d.com/Manual/NavMesh-BuildingComponents.html (visited on 13.5.2022)

[22] Unity technologies. *NavMesh Components, GitHub repository.* [online]. URL: https://github.com/Unity-Technologies/NavMeshComponents (visited on 11.1.2022)

[23] Unity Documentation. *Audio Listener.* [online]. URL: https://docs.unity3d.com/Manual/class-AudioListener.html (visited on 11.5.2022)

[24] Pharr Matt, Humphreys Greg. *Physically Based Rendering: From Theory to Implementation*. Elsevier Science & Technology, 2010. ISBN: 9780123750792

[25] National Aeronautics and Space Administration, Science Mission Directorate. *Wave Behaviors.* [online]. URL: https://science.nasa.gov/ems/03_behaviors (visited on 11.5.2022)

[26] Haines Eric, Akenine-Möller Tomas. *Real-Time Rendering.* CRC Press LLC, 2002. ISBN: 9781568811826

[27] Autodesk. *Autodesk Character Generator.* [online]. URL: https://charactergenerator.autodesk.com/ (visited on 17.11.2021)

[28] Blender Foundation. *Blender.* [online]. URL: https://www.blender.org/ (visited on 4.11.2021)

[29] Autodesk. *Maya.* [online]. URL: https://www.autodesk.cz/products/maya/overview (visited on 24.3.2022)

[30] Adobe. *Adobe Substance 3D Painter.* [online]. URL: https://www.adobe.com/products/substance3d-painter.html (visited on 4.11.2021)

[31] Ahargun C Design, Faisal Yudha Nugraha. *Horror Memories font.* [online]. URL: https://www.dafont.com/horror-memories.font (visited on 15.5.2022)

[32] DaFont [online]. URL: https://www.dafont.com/ (visited on 15.5.2022)

[33] Free Sound. [Online]. URL: https://freesound.org/ (visited on 15.5.2022)

Appendix A: Abbreviations

AI:	Artificial Intelligence
FPS:	First Person Shooter
GDD:	Game Design Document
GPU:	Graphics Processing Unit
HDRP:	High Definition Render Pipeline
HUD:	Heads-up Display
LMB:	Left Mouse Button
MOBA:	Multiplayer Online Battle Arena
RPG:	Role Playing Game
TLE:	The Last Escapee
UI:	User Interface
URP:	Universal Render Pipeline

Appendix B: Supplementary material

The zip archive with supplementary material contains two directories:

- **Build** – this directory contains the game's build files with the executable file which is used to start the game (file The Last Escapee.exe)
- **Project** – this directory contains the Unity project and all its files, the required Unity version to run this project is: **Unity version 2020.3.21f1**