

TDK-dolgozat

Brand Ádám

Brand Ádám (MSc 2. évfolyam)

RESEARCH OF FINE MOTOR AND SPATIAL COORDINATION SKILLS, ANALYZING AND DEVELOPMENT POSSIBILITIES WITH INFORMATION TECHNOLOGY TOOLS

Témavezető: Dr. Szücs Veronika (adjunktus)

Műszaki Informatikai Kar, Villamosmérnöki és Információs
Rendszerek Tanszék

Lezárás dátuma: 2022. Április 25.

Pannon Egyetem
Intézményi Tudományos Diákköri Konferencia
2022



TABLE OF CONTENT

1. Introduction	
1.1 Digitalization of healthcare	1
1.2 Fine motor research	5
2. System plan	
2.1 System specification	12
2.2 Modelling of the key features and game running	14
2.3 Conceptual design of Graphical User Interface.....	24
2.4 Hardware and other requirements	29
3. Presentation of the game and the game engine	
3.1 Short review of the game.....	33
3.2 Unity	33
4. Implementation	37
4.1 Implementing the game world.....	38
4.2 Game scripting.....	47
5. Test	
5.1 Debugging	56
5.2 Performance.....	57
5.3 Functional test	58
6. Summary	
6.1 Conclusion.....	64
6.2 Options for further updates of the game.....	65
Bibliography.....	66
List of Figures	68
List of Tables.....	70

1. Introduction

1.1 Digitalization of healthcare

Digitalization has a big impact on our life. Think about how fast it is evolving, how many novelties we can see day to day. It is no different in healthcare either. Of course, keep that in mind that not only pros but also some of the disadvantages of this area known. Let us take some positive and negative effects of digitalization, including but not limited to (Table 1.1.).

Table 1.1. Effects of digitalization on our life

Some effects of digitalization on our life	
<i>positive</i>	<i>negative</i>
makes our life more comfortable	we become lazy
we can access to more information than before digitalization	depreciation of traditional books
on online platforms we can make friends, talk to them wherever they are	depreciation of traditional, personal relationships
we can reach the space and do deep space research to understand our place in the universe	more destructive war technology than ever before
we can save more life	more expensive healthcare

Hereinafter, I will focus only on the positive effects of digitalization in the healthcare area. The so called digitalized health care is built on the applications and technologies of the information technology of medical knowledge. With these tools there is a big improvement of medical care, supervision and rehabilitation. This project is focusing on the help of the rehabilitation phase of the patient.

There are millions of people in the world who has some kind of a movement-related disability. This disability can be congenital or acquired. congenital is like SMA (Spinal Muscular Atrophy), acquired is like a stroke, bone fracture, etc. This project is builds around mainly acquired, hand related fine motor skills. Fine motor skills is about small movements, working of small muscles. In a healthy statement it is a natural movement, but also very complex because of it requires the coordinated work of brain and muscles. We constantly use fine motor skills in our everyday lives for example for writing, typing on a keyboard, to grab something, etc. Spatial perception is closely related

to fine motor skills. Spatial perception is a process through which we can place the relative position of our body or other objects in the world. Through this process we obtain information such as depth, distance, size and so on. These information are all important and by their combined knowledge we can move and navigate the environment.

In the last decade, the number of applications that can be combined with health care and rehabilitation in some form has increased significantly. There are apps that can be downloaded by anyone from the Google Play store to the Android operating system or from the App Store for iOS systems, but there are also apps that aren't available to the public. The downside to downloading games from the popular smartphone app stores mentioned above is that there are few that are specifically designed just for sick people. The majority are games that can be used by anyone, but the game has a number of strengths that can help certain patients recover. This project also covers the making of such a game, i.e. the development of a game that can be played by anyone in general and can be fun for all ages, but can be of great help to patients who have impaired fine motor function for some reason but can be treated with some targeted exercise, so it can be rehabilitated from his illness. The Table 1.2. shows some well-known applications that may be useful for people with reduced motor mobility, which is of course only a fraction of similar applications on the market.

Table 1.2. Popular games. A: Android, I: iOS

Title	Platform availability		Actual rating out of 5		Actual downloads for Android	Company
	A	I	A	I		
Draw Line: Classic	+	+	4.5	4.5	14113	XLsoft
Flow Free	+	+	4.4	4.7	1443317	Big Duck Games
ZirkI	-	+	-	4.3	-	sysbee apps
Fruit Ninja	+	+	4.3	4.4	5346959	Halfbrick Studios
Angry Birds 2	+	+	4.3	4.6	5667316	Rovio Entertainment

The Angry Birds series is a very good game, its handling is easy to learn. There are a lot of levels in the game, and there are several episodes that appear with different themes. Due to the theme, look and purpose of the game, it provides good fun for young and old alike. Angry Birds is a game of skill based on the fine motor skills of our hands on the one hand and the skill of spatial coordination on the other. During the game, birds have to be shot from a slingshot so that we can hit all the pigs on the field. Targeting plays a very important role during shooting, as it is important to be able to gently move your finger to aim properly and to know where the target is located in space. This game is great for developing the skills mentioned above, in case your fine motor skills are not significantly impaired because you need very small moves to aim.

Managing Fruit Ninja is also easy to learn and can be played well by young and old alike. Thinking of the younger age group, a specific cutting tool in the game does not appear only in the form of light, so violent content does not appear in front of the players. This skill game builds on reflexes and fine motor skills. During the game, you have to cut down the falling fruits with your finger and thus get as many points as possible in a given time. The game can be a good choice for injured people with moderately impaired hand movements. In contrast to Angry Birds, the game also allows for larger volumes of movement here, but reflexes are much needed because of the fast-moving fruits, so it is less recommended for patients who have suffered a stroke.

Zirkel is a game available exclusively for iOS systems. The application builds on reflex and eye-hand joint coordination. The game is specifically designed to develop these abilities. The game is very simple to operate, just add one hand. You can see a clock-like object on the screen divided into four parts with different colors and a large pointer with one of the four colors of the circle. When the color of the pointer and the color of the circular article that the pointer points to are the same, you need to touch the display. This game can be a good choice for people whose fine motor ability of their hand is severely impaired because it helps motivate the patient to move their hand, yet it is enough to touch any part of the screen at the right time.

The games called Flow Free and Draw Line: Classic are also suitable for any age group not only for fun but also for skill development. These apps are also based on spatial vision and a bit of hand coordination. They are also a very simple games. Dots of different colors are placed on the track, and the goal is to connect these dots so that the lines do not intersect. The dots are relatively small, so you can help develop fine motor skills by having the patient move their hands with sufficient precision to be able to touch the dots,

hence this can also be a real challenge for injured people. However, for people who are not only injured in their movement, but also in their sense, these games may not be the most suitable, as properly connecting the dots requires a relatively large amount of brain work.

There are some of the games mentioned above where colors play an important role, such as Zirkel, Flow Free, Draw Line: Classic, so for patients who even have color blindness, these applications are not suitable for their rehabilitation in an understandable way. It is very important to emphasize that it is true for all the different applications that the basic need to use them is not to be blind. However, as mentioned earlier, the toys presented above are only a very small part of the market offering, in addition to these, thousands of other applications could be mentioned that could help patients heal, although not specifically designed for this purpose.

In addition to the above, I would mention a company called REHABILITY, which develops applications that are not available to the public, but are specifically for rehabilitation purposes, and the main platform for most of their applications is not for mobile, but for TV and PC. It is important to note that some of their games may also require some kind of motion sensor. They currently have six different game versions for people with different illnesses, including a game for movement development. [1]

Based on current trends, it makes a lot of sense to build various assistive applications in the healthcare field primarily on mobile platforms. Today, almost everyone has a smartphone in their pocket, their use is already part of our everyday lives. The advent of smartphones and healthcare applications has also been a huge step forward in doctor-patient communication, with the help of which communication has been simplified and made faster, and the doctor can monitor the condition of his patients from long distances.

Currently, the smartphone market is based on two mobile operating systems: Android and iOS. The range of devices available from Android-based smartphones covers a very wide spectrum from cheap to expensive, which is why more people use this system, while Apple only offers an expensive and premium option, which narrows the range of consumers. Because more people can afford an Android-based smartphone, I chose to have this project ready for this operating system as well.

Android is an open source operating system designed for smartphones. The kernel part of the operating system is based on the Linux system, which was written in C and C++, and later a GUI (graphical user interface) written in Java was added. Google acquired

a company called Android in 2005, and after a long development, the first version (Android 1.0) was released on October 21, 2008 under the license of Apache. Since then, many smartphones, tablets and other mobile devices have been running the Android operating system. Its development is still at an unbroken pace, with newer versions coming every year since 2010. Let's review the last three Android versions (Table 1.3.). [2] [3]

Table 1.3. Latest Android versions

Version	Code name	Release date	Useful features for the project
9	Pie	August 6, 2018	auto brightness, adaptive battery prediction, Vulkan 1.1 support
10	Android 10	September 3, 2019	adaptive battery improvements, focus mode
11	Android 11	September 8, 2020	permissions auto-reset
12	Android 12	October 4, 2021	improved system performance, easy phone switch

From what is summarized in Table 1.3., I would like to highlight adaptive battery prediction, which uses artificial intelligence to try to predict when to use applications during the day, thus shutting down applications that we are not used to at that time, thus saving battery time. Vulkan 1.1 is responsible for supporting high-performance 3D applications for multi-processor core systems. The focus mode is a useful features for patients who also have attention disorders, because with this tool we can mute other applications so that they are not disturbed, thus helping them to focus on the goals of the game developed within the framework of this project.

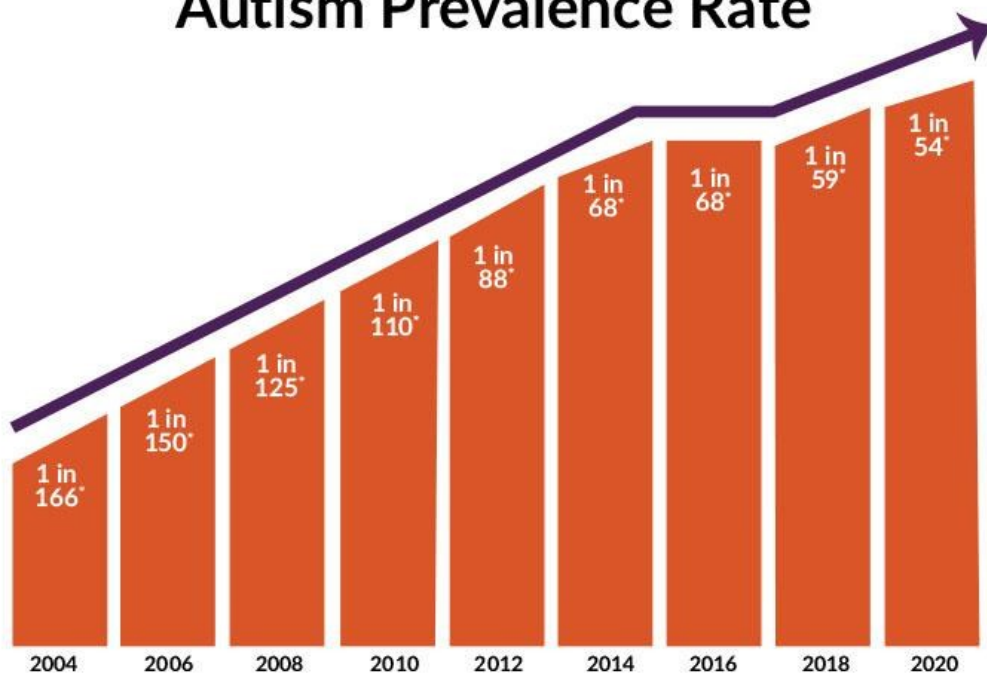
1.2 Fine motor research

Acquired or congenital disease can cause decreased fine motor skills. Acquired decreased fine motor ability is characterized by being generally temporary with a good chance of recovery. These include fractures, sprains, and direct damage to the muscles and bones. In the case of congenital disease, the chances of recovery depend on the disease. It is

characteristic of these diseases that the problem is not directly with the muscles and bones, but with the conduction of nerves and the control of the muscles. So in this case, the source of the problem is usually the brain, spinal cord, hand nerves. This category includes Autism Spectrum Disorder as well, which today plays a central role in analyzing and improving these skills with virtual, digital techniques.

I would single out autism because it causes fine motor problems in almost all cases, and the number of children diagnosed with the disease has risen sharply in recent years (1.4 figure).

Through the Years: Autism Prevalence Rate



*Centers for Disease Control (CDC) prevalence estimates are for 4 years prior to the report data. i.e 2020 figures are from 2016.

1.4 figure Growing tendency of Autism [8]

The first signs of autism can be observed around the age of eighteen months, although today no such diagnosis is made in Hungary under the age of two. Determining the onset of autism is not an easy task, especially since we are talking about children who in many cases do not even speak, so only by analyzing their social behavioral norms, movements and fine motor skills can this disease be diagnosed. [9]

All in all, decreased fine motor skills can cause serious challenges in a patient's daily life, even in everyday things like writing by hand or unscrewing a cap from a bottle. Therefore, it is very important to be able to work out the best possible solutions to the problem, but above all, the acquisition of knowledge is the most important thing. On the

one hand, it is necessary to get to know the disease that causes the symptom as well as possible, because in many cases the fine motor skills are also reduced with the cure of the disease. On the other hand, in cases where reduced fine motor skills persist after curing the underlying disease, it is necessary to develop methods to improve fine motor skills. Research shows that there are more and more effective methods for mapping and analyzing fine motor movement, so better and better rehabilitation and development methods are making their way.

One of the most common clinical tests today for fine motor skills is the Jebsen Taylor Hand Function Test (JTHFT). This is one of the oldest tests that meets a clinical standard and gives objective results. The test can be used to examine the fine and gross motor function of the hand, based on movements that are performed frequently during daily activities (1.5 image). [10]



1.5 image JTHFT test in practice [11]

The so called Nine-Hole Peg Test is well complement the JTHFT method, which is designed to measure the dexterity of the fingers in patients with various neurological diagnoses such as stroke, Parkinson's. The essence of the method is to insert the pegs into the holes one by one as quickly as possible in the nine-hole plate in front of the patient, and then take the pegs out of the hole one by one and put them back in their container. The evaluation is time-based, the faster the patient, the better the evaluated performance (1.6 image). [12]



1.6 image Nine-Hole Peg Test [13]

A research published in 2017 by KathrinAllgöwer and JoachimHermsdörfer studied nine parameters connected to fine motor skills using, inter alia, the tests mentioned above. Three factors that most influence fine motor skills have been successfully identified: grip force scaling, motor coordination, speed of movement. Together, these three factors can predict the outcome of JTHFT with an efficiency of 69%. [14]

Today's modern analyze methods put virtual reality at the forefront. This is because by connecting the patient to the digital space, we can extract much more and more detailed information from the tests. There are several ways to connect a patient to a virtual space, but one of the most effective interfaces for this is a special so-called Haptic-Gripper, which can be supplemented with VR goggles. With the help of a haptic gripper, the feeling of touch can be transmitted, with the help of which the movement of virtual objects creates a natural feeling. In terms of their design, there are many types that are usually tied to a target task.

A commercial, standard design for VR games is shown in the 1.7 image. This is a very good design especially for games where a weapon must also be used during the game. This design fits very well to grip a pistol. However, it is also clear that this design can only follow the gross motor function, not the fine motor movements like the

movements related to the fingers.



1.7 image Commercial haptic-grip for VR games[15]

In addition to the interface that creates the connection between the two worlds, there is a need for appropriate tasks in the virtual space that form the basis of the study. Because traditional test methods have been shown to provide adequate feedback on the patient, it is appropriate to virtualize the tasks used in these test methods. For example, the ordinary movement culture applied in the case of JTHFT can be applied very well in the virtual space as well. This means that a software solution that can even reproduce JTHFT tasks, such as moving objects from one location to another, can be easily reproduced. However, it can also be seen that the object lifting in the JTHFT method is no longer fully reproducible, as the weight of the object is lost by using the interface linked to the virtual reality. It was precisely this technique that sought the answer to study the fine motor behaviors of children with autism in a 2018 study. A special Haptic Gripper was used that could also measure the user's grip force. Although their results are preliminary and limited because of the fact that not everything can be simulated -for example the weight-, they were found that "medium to strong correlations between the proposed fine motor skill metrics and the scores achieved with a standardized fine motor skill test and improvements of participants in accuracy and steadiness of movement and force control." [16]

In a 2018 study specifically targeting the recovery of fine motor skills using virtual space, they thought in a completely different direction than the Haptic Gripper. The virtual environment was created with the help of Unity, their target group was

children aged 6-10. The interface used was the so-called Leap Motion Device (1.8 image). This device tracks the movement of the hand optically, so no physical contact on the part of the user is required. Their research shows that this technology scored 61% on the SEQ (Single Ease Question) usability test, which means it is suitable for use in rehabilitation. [17]



1.8 image Leap Motion Device, the pendrive sized object in front of the laptop [18]

So far, I've collected some techniques aimed at analyzing and developing people with impaired fine motor skills. However, not only do they need to improve fine motor skills, but they also need healthy people who need a higher-than-average degree of precision in their work. Such an area where a high degree of fine motor skills is expected is for example dentistry. A 2021 study addressed this issue. Learning motor skills requires a continuous interaction of cognitive, sensory, and neuromuscular processes. So the ability to fine-motor movement coordination is a very complex process. The research identified three factors that greatly influence the development of this ability: instructions, abilities of the learner, variation in practice. Research has found that verbal instructions have a strong effect in shaping motor skills learning. They also found that personal abilities were important, especially the memory of the individual. There are two types of memory: working memory and long-term memory. Both are much needed to develop fine motor skills. The information needed for a movement is stored in the working memory for a short time, approx. half a minute. In order for movement coordination to be maintained in long-term memory, multiple exercises are required. The better a person's memory ability, the more effectively they will be able to improve their fine

motor skills. The third factor, examining the variety of tasks, concluded that both repetitive tasks and variety are important in their own way. Repetitive tasks affect long-term memory, while random, varied tasks affect the working memory. If a task requires complex movement, repetitive practice is the more effective method, while a task requiring a simpler movement culture is more effective to use random practice method. [19]

Based on my research, I have come to the conclusion that it is not yet possible to simulate the whole real world. Virtual reality can be built to meet the necessary fine motor studies, but the problem of the lack of proper simulation is caused by the interface between real and virtual space. Each type of interface has its strengths and weaknesses, and these should be chosen with the right one in mind. A common feature of the virtual reality-based methods presented here is that they cannot represent the effort required to lift objects, so they cannot simulate weight and force. However, it can also be stated that by calling on virtual reality, it is possible to extract much more detailed and more information than with traditional methods such as JTHFT. So modern methods can only simulate a more limited part of the physical world, but they can provide much more detailed feedback on it. Previous research has shown that the Unity game engine is capable of creating virtual environments in which the right tasks can be compiled for the required fine motor tests, keeping in mind that tasks must include repetitive and novel challenges to ensure that both working memory and long-term memory are properly be stimulated. Since it is necessary to compromise in the field of interfaces, I will only concentrate on capturing finger movements. I find touch screens to be suitable for detecting finger movements, which are an integral part of smartphones and tablets, so the interface also defines a platform in the form of Android and IOS.

2. System plan

2.1 System specification

The features required for the game, which I designed and implemented are presented in the following Table 2.1. :

Table 2.1. Key functions

Feature name	Caller	Lifetime	Special requirement(s)
Main Menu	Player or System	scene	no
Back	Player	short	no
Play	Player	scene	no
Upcoming contents	Player	scene	internet connection
Rewards	Player	scene	complete specified achievements
Level selector	Player	scene	need to be unlocked
Player movement controller	System	scene	no
Level completed	System	short	touching of green platform
Saving level	System	short	triggered by level complete
Level loading	System	short to medium	calling a level
Player performance evaluator	System	short	triggering of level complete
Game object manipulation	System	scene	no
Player position reset	System	short	being in fall zone

Briefly summarizing the features:

- **Main Menu:**

The first call to the main menu is made automatically when the game starts. Later, the player has the option to call it from other submenus. This scene includes access to

submenus (Play, Upcoming contents, Rewards).

- **Back:**

With this feature, the player can return to the previous scene (enabled in submenus). In this game, it is not possible to return to the main sssmenu while playing. Once a player has started a level, he can only return to the main menu by completing it or closing the game. The reason for this is that from a rehabilitation point of view, interrupting the gameplay is not allowed because it could lead to an erroneous evaluation.

- **Play:**

It takes the player to the scene where he can select the level that he want to play and it also shows the progress of the player. The first level is automatically open to everyone, the next levels can only be accessed if the previous level has been completed (unlocked).

- **Upcoming contents:**

It takes the player to the game's website. Internet access is required for this feature. Since this is a non-critical feature, I don't add a strict requirement, so the launch of the game isn't tied to internet access either. You can play the game without internet access.

- **Rewards:**

This feature takes the player to a scene where he can view his milestones. This feature is especially important from a health point of view. I want to motivate the patient to play and try to perform as well as possible, in return the player can collect appreciative badges.

- **Level selector:**

This feature ensures that the player can only play on a level he has unlocked before. It reads the current state of the game from the encrypted file saved on the phone so that the level selector can set the locks on the proper levels.

- **Player movement controller:**

This is a basic and very important feature in the game, commonly known as virtual joystick that is familiar on touchscreen devices, complete with a jump option. A control interface for the player that allows him to move his dice in the game.

- **Level completed:**

This feature is called when the player touches the green platform at the end of the level with his dice. The level is then considered completed. If this happens, the player

must be deprived of their control interface and offered an option to move to the next level or return to the menu instead, and provide an assessment of the player's performance.

- **Saving level:**

This function is called when the level is completed. Have to save the current progress of the player, save it to an encrypted file and unlock the next level.

- **Level loading:**

Today's modern games use a so-called asynchronous loading method, so I chose that too. The point is that while the level and its constituent elements are loaded into the memory in the background, we show the player another screen on which we can place important information as well. When the actual level is ready for use, the loading screen is taken from the player and sent to the loaded scene. The game should also serve a rehabilitative purpose, so I placed motivational messages on the loading screen to help the patient persevere with recovery.

- **Player performance evaluator:**

At the end of each level, the player must be evaluated for his performance. I store the results locally on the device. There should be two ways to evaluate. A standard scoreboard and a list of player actions from the start of a level to its completion in txt format, which can then be taken over by a third party for further processing.

- **Game object manipulation:**

Manipulating the objects on the levels is necessary to create the right obstacles and interaction between the level and the player. There are several forms of this, but the goal is to make the mechanism known to the player and to provide adequate difficulty for the gameplay.

- **Player position reset:**

This mechanism allows the player to be penalized by returning the player to the starting point of the level. On the one hand, this gives players a good challenge, and on the other hand, it gives them the opportunity to correct mistakes.

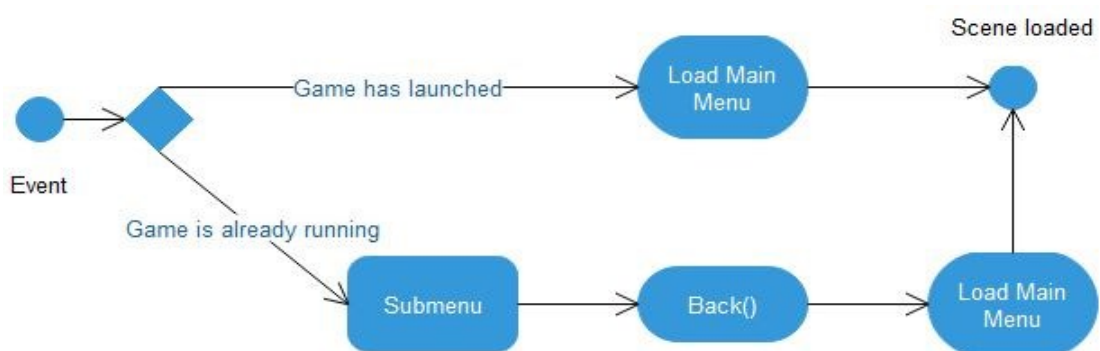
2.2 Modelling of the key features and game running

In the following, I present the models and a more detailed description of the key features and functions defined in the specification, noting that the functions in the models are continuously fine-tuned during development, if necessary expanded or simplified, but keeping in mind the properties set up in the models. As a result, the implemented solution

can be much broader or more complex than what the models represents. The models presented here are intended to help you understand how each feature works:

- **Main Menu** (2.2 diagram):

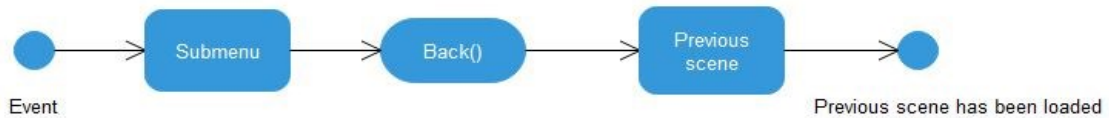
Loading the Main Menu is justified in two cases, it belongs to two different events. In one case, the game has been launched, so the first thing a player needs to display is the Main Menu. From here the player will be able to choose what he want to do next (submenus). In the other case, the game is already running and the player is in a submenu where he activates the back button. This will take him back to the main menu. In both cases, a scene load method must be performed. Here, I see casual, simple scene loading as sufficient, omitting the asynchronous method, which usually also includes the loading screen. This is because the minimal content of the main menu requires very little resources to render, so any mobile device today can load it in no time. I think it's important to point out that Unity's engine strives for constant optimization, so whenever a new scene has been loaded by default, the old one is lost and replaced with the new one requested. I consider this to be a very useful and powerful feature of Unity because it doesn't require too much memory handling. I would note that of course there is a way to additively load scenes, i.e. to preserve the content of the previous ones or certain parts of it - such as background music. In this case, due to the way the scene is loaded, the music will always restart, which belongs to this scene with each new load - this applies to all scenes anyway, which can of course be modified later if necessary.



2.2 diagram Main Menu

- **Back** (2.3 diagram):

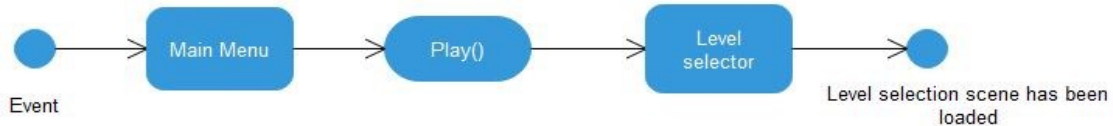
Each submenu (Play, Rewards, Upcoming contents) has the ability to access the Main Menu. Any submenu can only be accessed from the main menu, so if the player is in a submenu, the previous scene can only be the main menu. This function is assigned to the virtual back button. The function implements simple scene loading and is called when the player touches this button.



2.3 diagram Back

- **Play** (2.4 diagram):

Play is a submenu that can be accessed from the main menu. It is activated when the player touches the corresponding virtual button. It has two responsibilities. One is for the player to load the scene (simple scene loading) where he finds the playable levels and then launches them from here. The other task is to call the Level selector feature.

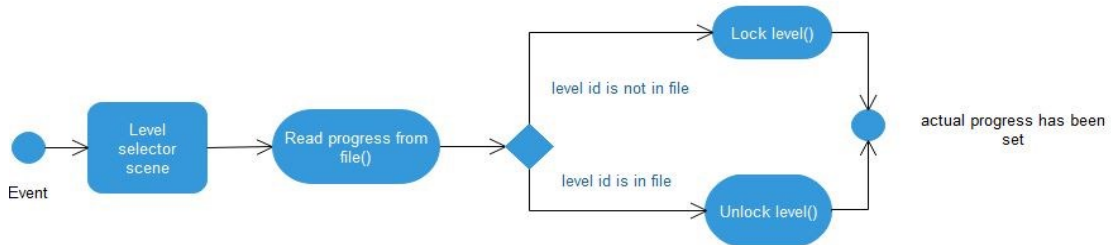


2.4 diagram Play

- **Level selector** (2.5 diagram):

This feature is called by touching the Play virtual button. At the same time as the player is transferred to the level selection scene, the availability for each level is determined. There are two possibilities: the game will call this feature for the first time or at least a second time. When the player first enters the level selection scene, a new binary-encoded file „PlayerProgressionV1.txt” must be created locally on the device’s storage at the persistent data path „./storage/emulated/0/Android/data/<packagename>/files” to store the number 1 -which means the first level- and made it available to the player, while all other levels must be locked. If this feature is not called for the first time, the file must already exist and the availability must be assigned to the levels according to the number read from the file. For

example, if the number 3 is read from the file, the first 3 levels must be made available to the player while all the others must be locked.

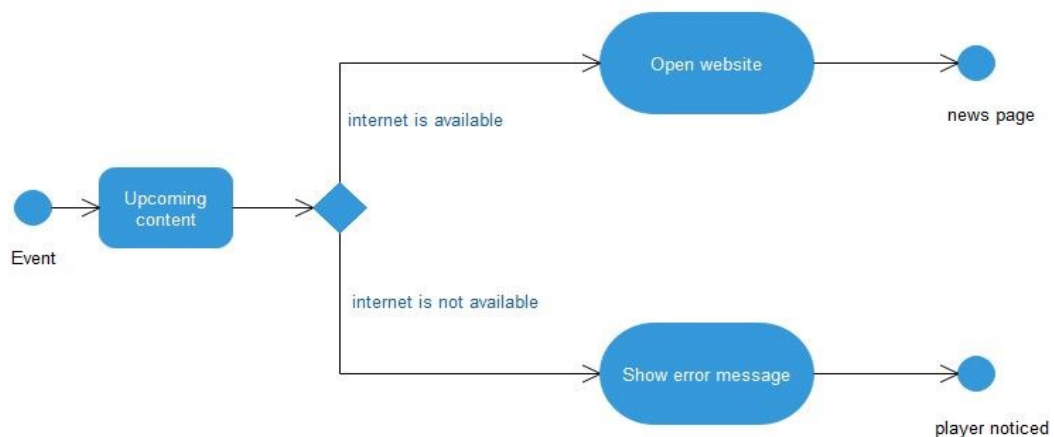


2.5 diagram Level selector

- **Upcoming contents (2.6 diagram):**

This feature belongs to the Upcoming contents submenu. If the player has active internet connection, the home page of the game's website will be loaded with the device's default browser. If there is no active internet connection, the player will get an error message informing him that the internet connection is required to load the requested content. The website should be simply transparent and concise so that the visitor receives the most information in the shortest possible time.

The game's website should include: a summary of the game, hardware requirement, contact, ingame footage, and a form for feedback. Noting that the design of this website is not the subject of this thesis, but belongs to the game and available at „<https://www.cubiedemonstration.weebly.com>”.



2.6 diagram Upcoming contents

- **Rewards (2.7 diagram):**

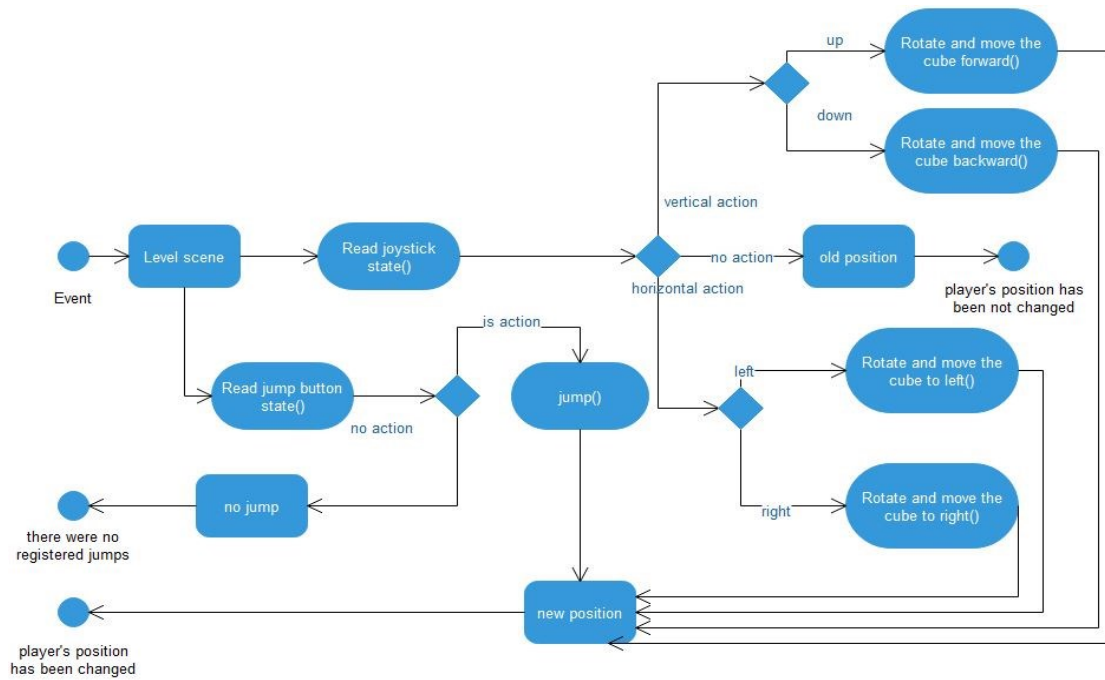
By activating the Rewards virtual button on the main menu, the player is taken to the scene where he can view his milestones so far. Along with loading the Rewards scene, the game progress will be read from the file stored on the device and the corresponding achievements become available as follows: the first achievement is after the completion of the 5th level, the second after the completion of the 8th level, and the third after the completion of the 11th level.



2.7 diagram Rewards

- **Player movement controller (2.8 diagram):**

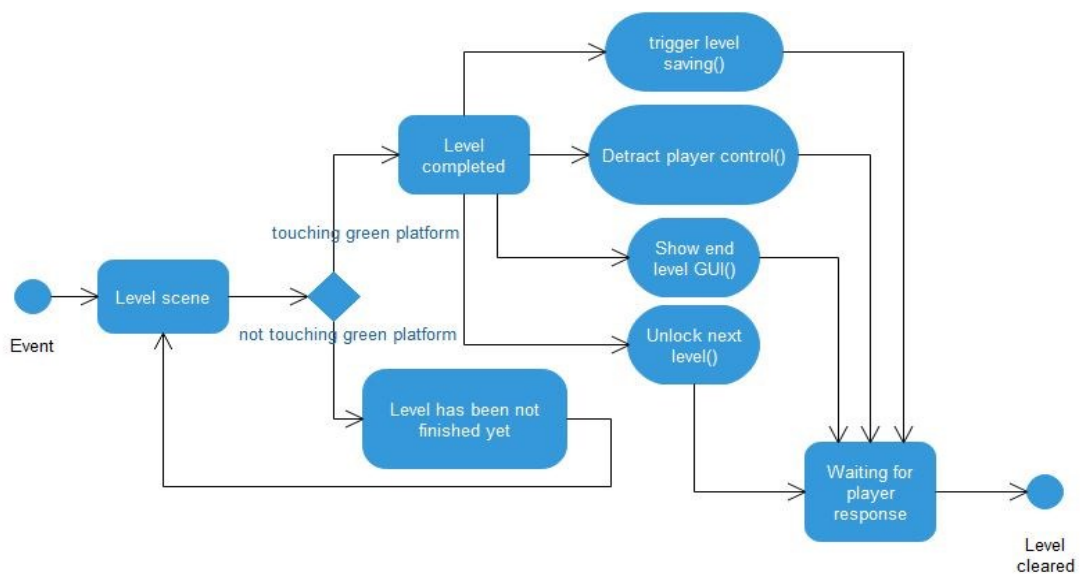
The player movement controller has two components: a virtual joystick and a jump button. These allow the player to control his dice. These interfaces should only be available in level scenes and only until completing the actual level. The joystick allows the player to move in four directions: forward, left, right, backward. The jump button moves the cube vertically for a short time and is only available if any surface of the cube is in proper contact with the ground. On some certain levels, the jump option must be deactivated. The usual operation is extended with a special feature that allows me to query the current position and state of the joystick and the jump button, which will be a necessary information to evaluate the player's performance.



2.8 diagram Player controller

- **Level completed (2.9 diagram):**

This feature constantly monitors the location of the player-controlled dice. The player must pass the dice to the green platform at the end of the levels. If this happens, the control interface will be deactivated and the Saving level feature must be called, the next level must be unlocked and a new GUI must be displayed. In the GUI activated in this way, the player can decide to return to the main menu or move to the next level, as well as an information about the performance.



2.9 diagram Level completed

- **Saving Level (2.10 diagram):**

This function is called after the player touches the end zone of a level. The current progress of the game is then saved by writing the number of the completed level to the file named „PlayerProgressionV1.txt”. The save was successful if the file was written successfully.



2.10 diagram Saving Level

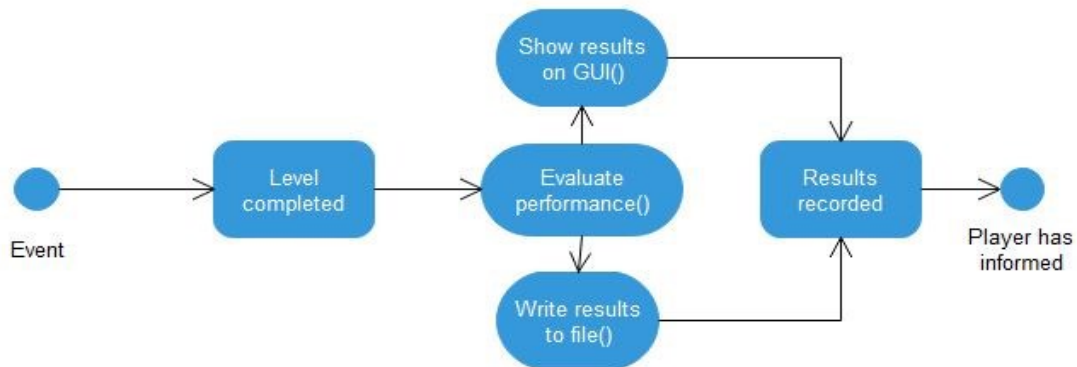
- **Player performance evaluating (2.11 diagram):**

This feature, which has two components, is one of the most important parts of the game, especially from a health point of view.

One function is to implement the well-known standard scoreboard. Its job is to give the player instant feedback at the end of each level about his overall performance, based on the time it takes to complete that level. This feedback is on a three-point scale: good, average, bad. Which time intervals belong to which assessment can be changed at will on each level, taking into account how long it takes an average person to complete the level.

The other function should give an unusual, more detailed evaluation of the player. For each level, this function saves a txt file - "levelnumber + date+random float number" - to the device's storage at „/storage/emulated/0/Android/data/<packagename>/files/Cubion_statistics” for how much time the player has been needed to complete that level, as well as a timestamp of when to perform the action, ie the position of the joystick and the use of jumps. Based on this information, graphs can be constructed to compare, for example, the eye-hand coordination of a person with impaired fine motor skills and a healthy person, which parts of the level were critical, from which one can also deduce one's weaknesses. It allows us to examine the development of a person's dexterity by examining whether the

player completes the same course over and over again and comparing this data with older data.



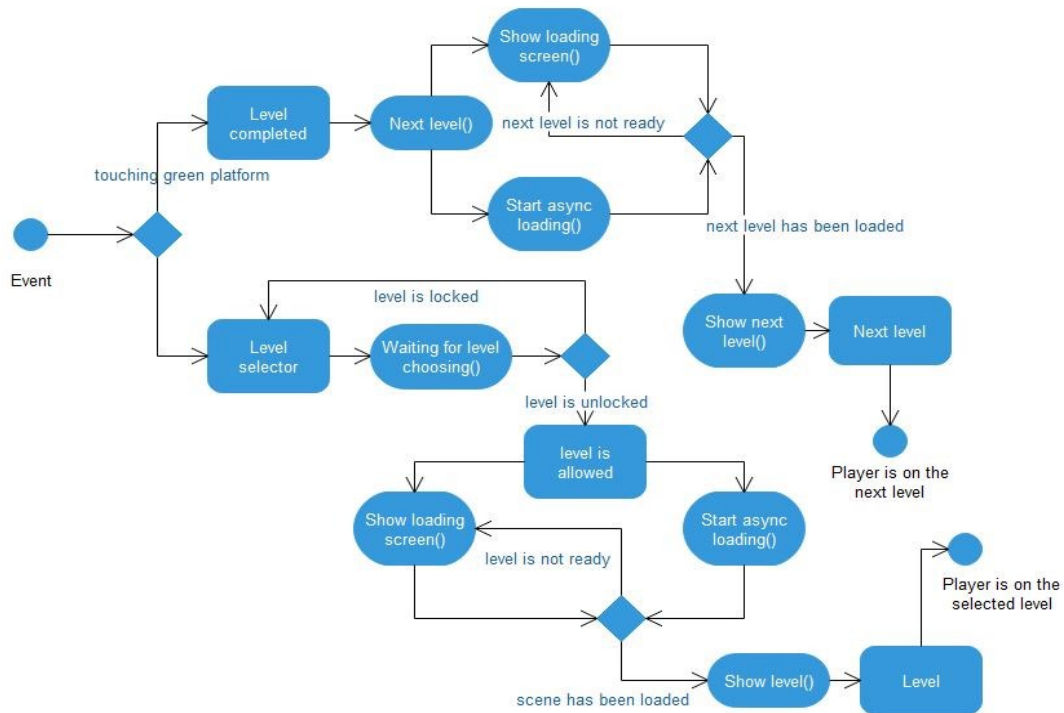
2.11 diagram Evaluating a player's performance

- **Level loading** (2.12 diagram):

This function is called by the intention to start a level or to move to the next level. Compared to other functions, it is more technically complex. When we use the simple load scene method, we are working with some simple scene with not too much content. In this case, there is minimal resource requirements to render the given scene and load it into memory, so the result looks almost instant on any device. Such a scene could be a plain background with some text-based captions. However, when the scene already includes several scripts, 3D models, effects, rendering and loading into memory is not so fast, especially for devices with less powerful hardware capabilities.

For such complex scenes, it is necessary to use the so-called async loading method, which can be used to improve the user experience by not letting the player on the level until all its elements are loaded properly, thus reducing lag, clutter and other inconveniences. The essence of the method is that when a level needs to be loaded, the player first switches to a simpler scene on the so-called loading screen, where the simple load method can be applied and in this scene the required level is called to be loaded in the background. When the level is properly ready for use, the player automatically switch from the loading screen to the scene of the current level. This technique is widely used in most games today. In addition to the fact that the player does not have to face lagging, missing textures or objects at the start of the current level, it is possible to display some shorter information on the loading screen. Such information could be, for example, tips,

information about the level or, in this case, good wishes.



2.12 diagram Async loading method

- **Game object manipulation (2.13 diagram):**

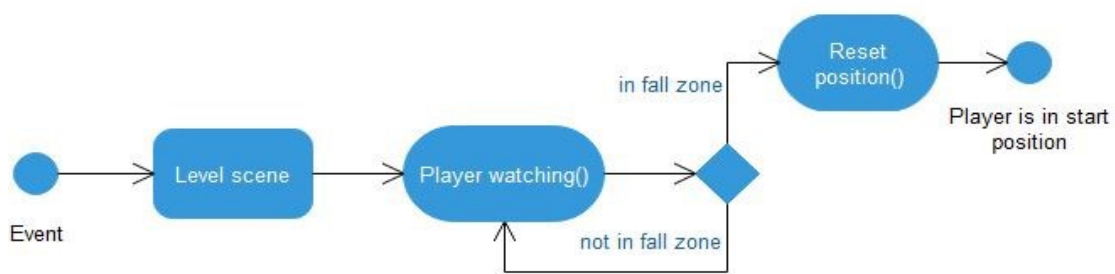
This feature is a collection of functions associated with manipulating objects in the game. This includes moving platforms, operating elevators, rotating objects in the game on different levels. These scripts start automatically when the async loading method is in complete status and run until the scene is destroyed because the player has exited the game or another scene is being loaded.



2.13 diagram Game object manipulation

- **Player position reset (2.14 diagram):**

It is a mechanism that will also be an integral part of every game in one form or another. This feature constantly monitors the player on the levels, and if the „y” coordinate of the dice is lower than the ground, it means that the player has fallen off the ground. In this case, it is a solution that is more likely to penalize the player if they fall off the platform by taking the player back to the starting point of the level. There is no checkpoint system on the levels on the one hand because the levels are not too long on the other hand so the game is more challenging. It also gives the player the opportunity to correct any mistakes he has made in the past and not to repeat them.



2.14 diagram Player position reset

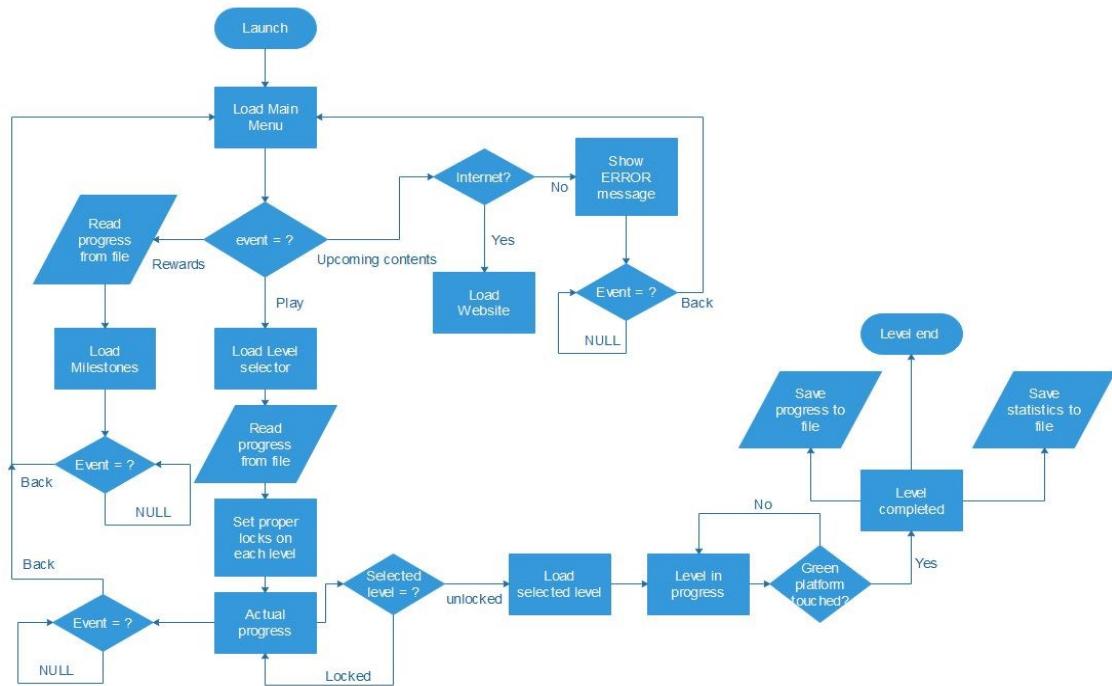
- **Running of the game (2.15 diagram):**

The model below shows how the game works from launch to completion of a level based on the features discussed above. You can see how each function is related, what order they define.

After starting the game, the player will be taken to the main menu after a short logo scene and a short loading time. In the main menu, the player can decide which submenu - Play, Upcoming contents, Rewards - to choose depending on what he wants to achieve.

Suppose the player enters the Play submenu for the first time, so when this scene is loaded, a new txt file will be created in the device's storage and the number "1" will be stored. The Level selector feature unlocks the first level and keeps the rest locked. The player would then like to start the only level he has available for the time being. The Level loading feature will be called, which will display a loading screen for the player. When the level is ready, the player is transferred to the current level from the loading screen. The Player movement controller feature is now available, giving the player the control interface for the dice. The first touch of the display starts a timer, then when the player has successfully rolled his dice to the green platform at the end of the level, the

timer stops. The Level completed and Saving level feature is then called. The control interface will then be deactivated for the player and a scoreboard will be displayed. A more detailed statistic should also be saved in a proper txt file containing the data needed for further evaluation, and the number of the next level should be overwritten in the corresponding txt file instead of the previous "1". Meanwhile, the player has the option to load the next level or return to the main menu.



2.15 diagram From launch to level completed

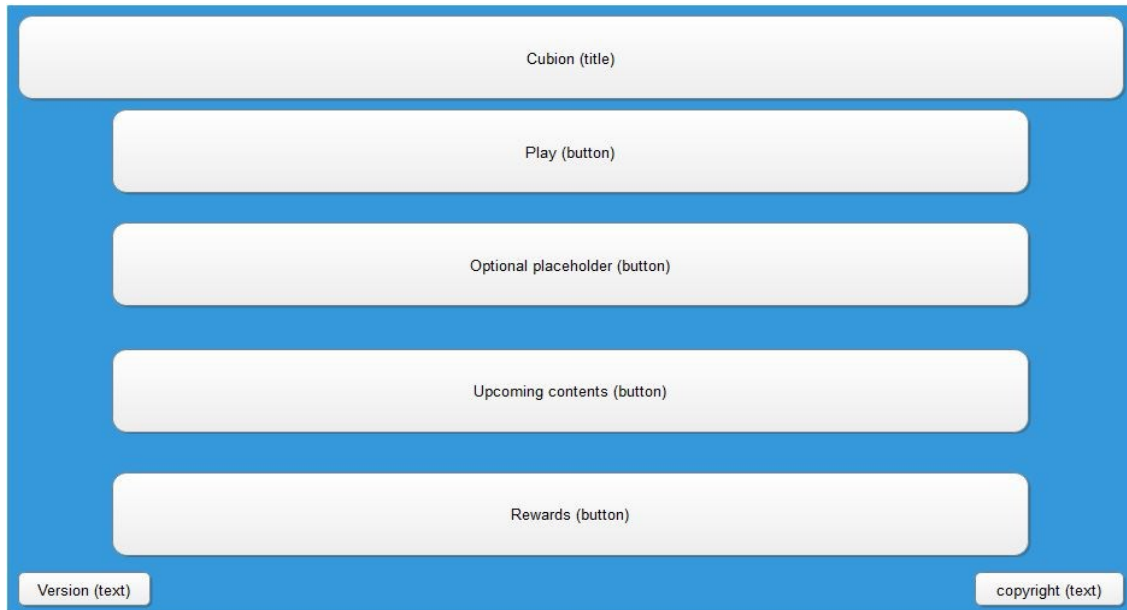
2.3 Conceptual design of Graphical User Interface

In this chapter I will present the conceptual designs for major scenes, i.e. the guiding sketch of the following graphical interfaces: Main Menu, Player UI, Rewards, Upcoming contents, Loading screen, Level selector, End of level.

- **Main Menu** (2.16 figure):

At the top of this scene is the game title. Below that is a button called Play. From top to bottom, the next in line is an optional placeholder for future updates like a Settings or How to play submenu. Below these are the Upcoming contents and Rewards buttons. The bottom left corner shows the current version number, which will be overwritten during build, and the lower right corner contains the copyright text. This structure is clear and accessible, offering easy navigation for any player who can read, since the main

menu is text based. The background design, which also applies to the submenus, includes the dice as the main motif, creating a unity between the design and the player's character.



2.16 figure Main Menu conceptual design

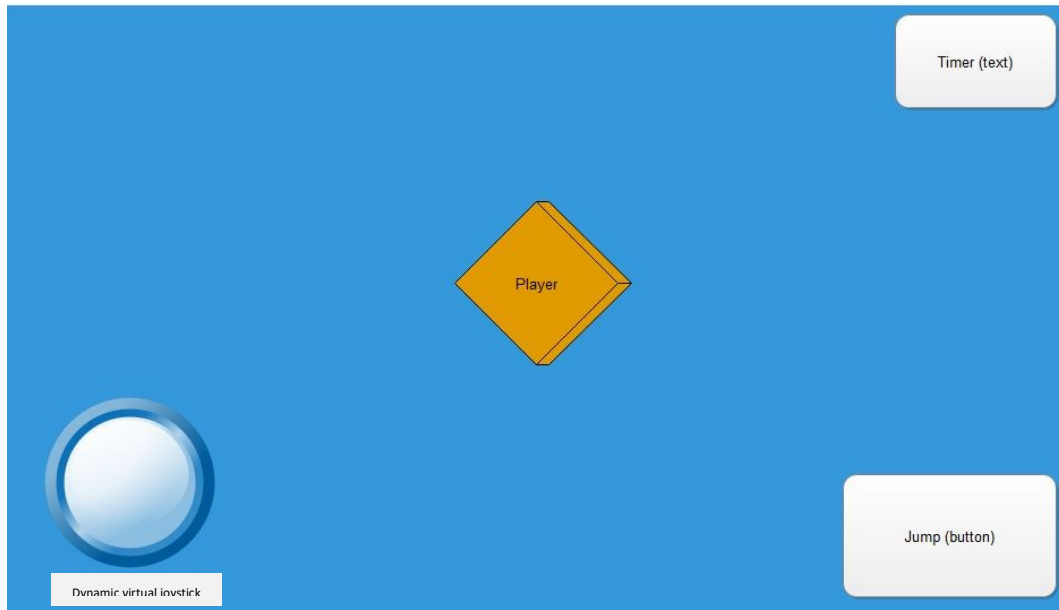
- **Player GUI (2.17 figure):**

The Player GUI is available when a level is loaded and is available until the level is completed. This is the interface that the player sees during the game. It has several components: Virtual joystick, jump button, timer. The interface is designed so that the character controlled by the player is always in the center and is surrounded by these GUI components.

A virtual joystick is called a dynamic joystick, which means that this control interface appears when and where the player touches the screen and only as long as a touch is made. This allows all players to play in a sufficiently comfortable position regardless of screen size. The look of the joystick follows the classic, familiar look, i.e. it consists of an outer circle representing the border of motion of the joystick and the control sphere itself in the middle, which can be moved in any direction. The color of this component is white because it is neutral to its environment and its size is fixed.

The position and size of the jump button is fixed in the lower right corner. The jump option is not available on some levels, so this button will not be visible to the player, that means it will be deactivated. The button can be of two colors: if the player's character touches the ground properly, it will turn green, which means that the jump ability is available otherwise it is white.

The position and size of the timer are fixed in the upper right corner. The timer must not distract the player, but it must be clearly visible.



2.17 figure Player UI conceptual design

- **Rewards (2.18 figure):**

In the Rewards scene the achievement achieved by the player can be viewed as a trophy. Three such trophies can be obtained by completing the marked levels. Each trophy includes a short text description and an animated cube with a unique skin. If the player has not yet completed any milestones, then in this case no trophy will appear, just a text based information about what is needed to reach the first milestone.



2.18 figure Rewards conceptual design

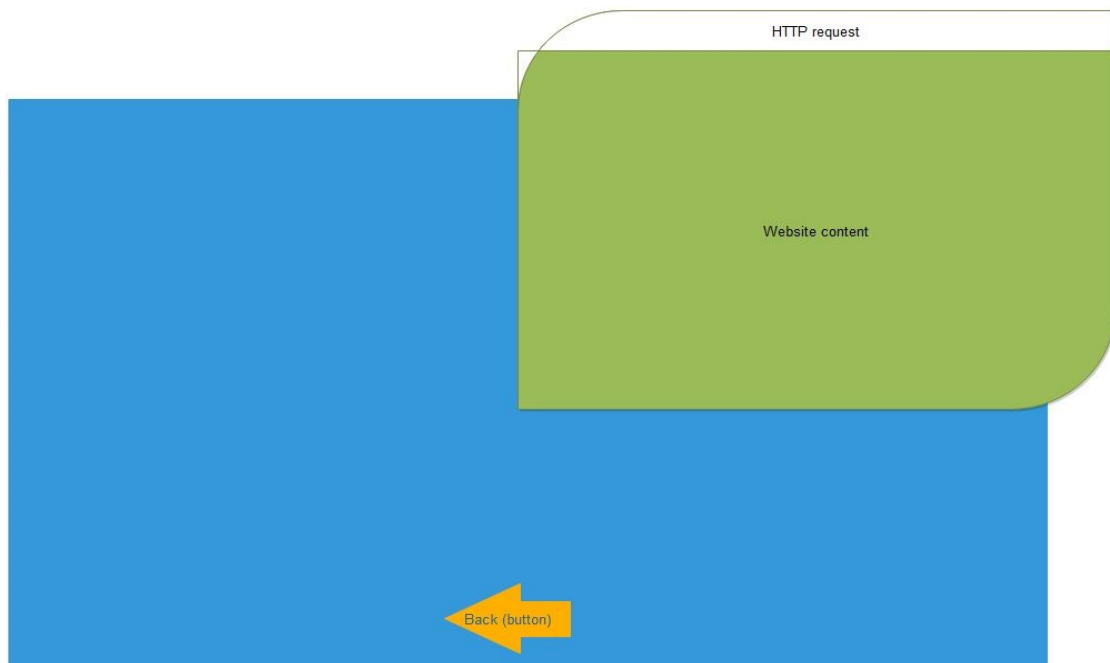
- **Upcoming contents** (2.19 figure):

The structure of this scene is the simplest. If there is no internet connection available, it will display a text-based error message to the player, if internet connection is available, the player will be redirected to the game's website using the default browser. The website consists of the following pages: Home, Upcoming contents, Contact. The pages should also be based on the theme that provides the background for the main and submenus of the game.

The Home page consists of five well-separated sections that present information about the game: name of the game and it's motto, a brief summary of the project, some ingame footage, hardware requirements, questionnaire.

The Upcoming contents page contains a brief summary of the upcoming updates. This brief is a text-based list without any multimedia content.

The Contact page contains the logo of the Faculty of Information Technology, referring to the University of Pannonia. On this page, you can send an email to the project developer Adam Brand using a simple contact form.

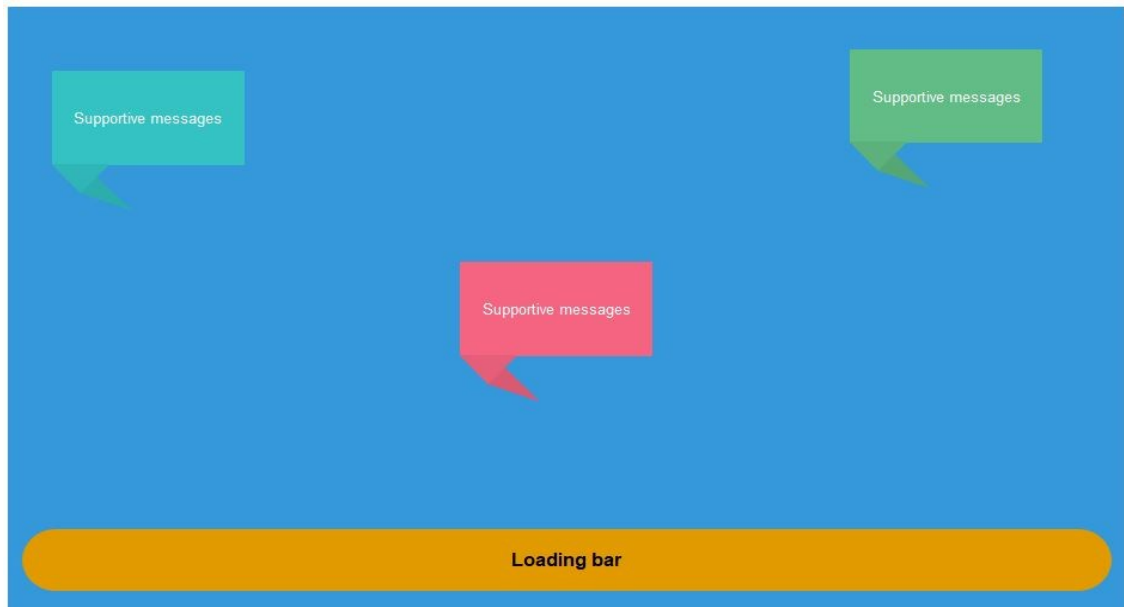


2.19 figure Upcoming contents conceptual design

- **Loading screen** (2.20 figure):

The theme of this scene is different from the other scenes in the game, in order to be eye-catching. At the bottom of the scene, almost to the full width, is a loading bar where the player can track the status of the current level, it provides a dynamic graphical

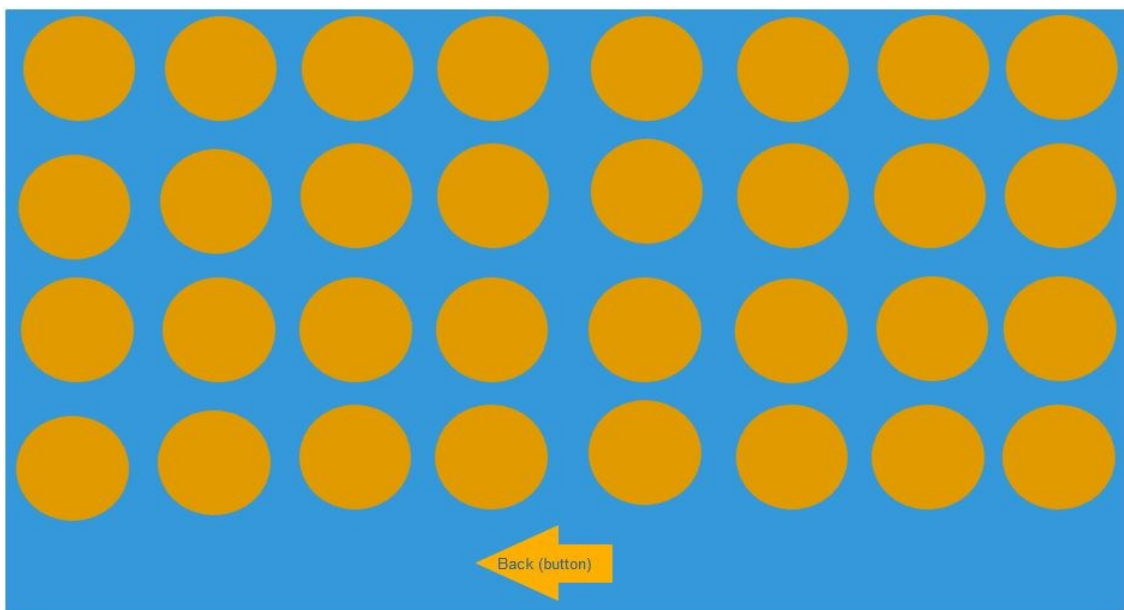
information for the player. The rest of the scene consists of static, pre-written texts. Together, these can reduce the „boring factor” while the player waits for the level to be ready.



2.20 figure Loading screen conceptual design

- **Level select menu** (2.21 figure):

In this scene, the player can choose a level. Here he can see how many levels there are in total and how many of them are available to him. The layout of this scene is matrix based. The levels are represented by their serial numbers, which increase in order from left to right, from top to bottom, as is customary in the Western world. Levels that are not yet available to the player are covered by a padlock.



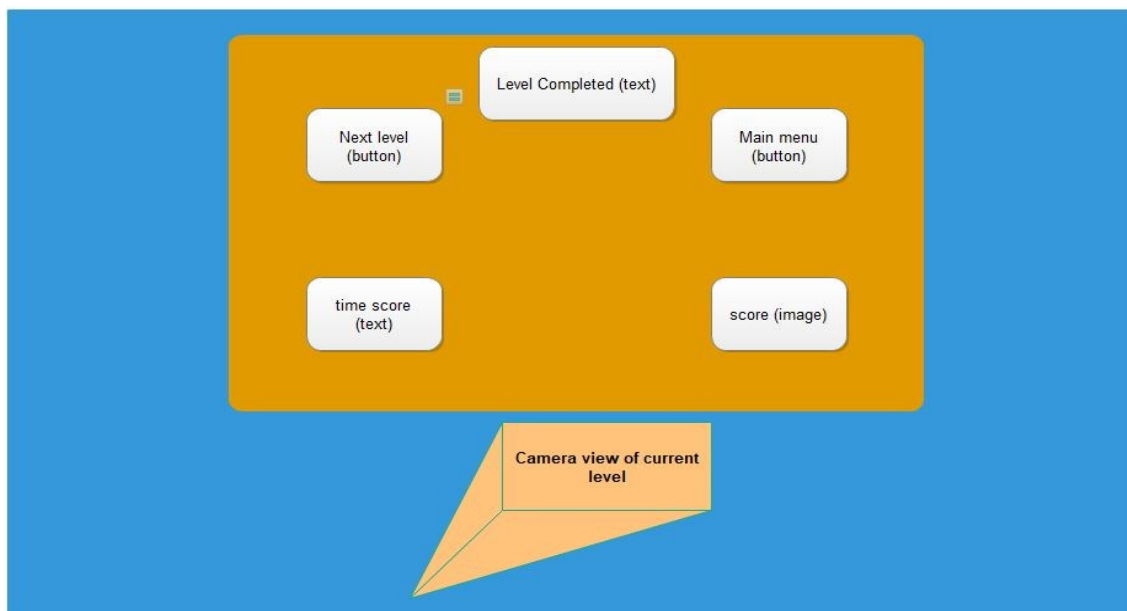
2.21 figure Level select menu conceptual design

- **End of level** (2.22 figure):

This GUI is the most complex in this game because this interface not only provides information but even has a scene loading feature. This interface only appears if the player has completed a level and only until the player chooses an action. The GUI itself is about a third the size of the entire screen, which has a fixed space at the top center. This layout allows me to create a more dynamic scene, for example, to show the current level from above in addition to this GUI. The color scheme of the GUI should stand out from the theme of the level to be clearly visible.

The lower part of the GUI serves as the information bar. The left side of this bar shows the time result, while the right side shows the graphical evaluation. The graphical evaluation should include hilarious icons whose themes match the basic theme of the game, i.e. they should be related to the cube. There are three types of results for each level: good, average, bad. These results have different icons and which one appears to the player depends on the time result. I made these icons individually specifically for this game. I set the interval distribution of these time results to my own average performance with differences of plus-minus about twenty percent.

At the top of the GUI are the buttons that allow the player to perform two types of actions. Use the Left button to initiate the loading of the next level, and the Right button to return to the main menu.



2.22 figure End of level conceptual design

2.4 Hardware and other requirements

Referring to unity's official documentation for running games made with it, the minimum system requirements required are as follows:

- Android version 4.4
- ARMv7 with Neon Support (32 bit) or ARM64
- OpenGL ES 2.0+ or 3.0+, Vulkan API support
- 1 GB of RAM
- Native Android OS, emulators excluded

You must also meet Google's Android compatibility definition, which includes mandatory, disallowed and recommended points. For handheld devices such as phones and tablets, it requires the following (highlighting key elements):

- battery
- minimum screen size is 2.5 inches (in diagonal size)
- touchscreen input support
- minimum 4 GB of non-volatile storage availability for application private data
- standard audio encoding and decoding support like AMR-NB, MPEG-4 AAC etc..

Fortunately, Unity provides most of the necessary system conditions already as far as the software part is concerned. [4]

In summary, Cubion's main hardware and software requirements are as follows:

- Android 9 or later OS
- minimum 2.5 inches touchscreen
- sound output
- 1GB of RAM
- 100 MB of free space (actual game is smaller)

Most manufacturers who build their smartphones around the Android operating system tend to support each device in software for up to two years. For most of the two years, it is valid for top models, lower end devices usually do not receive support for that long. This means that a flagship smartphone will receive two more Android updates from its launch, as a newer release of Android is usually released annually. For a lower-end Android smartphone, there is no guarantee that you will receive more than one update. From these, it can be said that the average life cycle of a phone is two years. This is

confirmed by the fact that for most mobile service providers, both at home and abroad, the loyalty period does not differ significantly from two years or more. Still, the majority can't afford to buy themselves a new cell phone every year, every two years. The average person tries to make the most of his mobile phone. Buying a new mobile usually happens when the old one is broken or a new device appears that can offer much more than the current one. So for most people, a phone lasts for more than two years, even though it no longer necessarily gets the latest Android updates. For this reason, Cubion is supported up to an older version of Android that is still in significant use. So this makes the game more widely available, even to those who can't afford the latest devices.

The touch screen and the Android operating system come together, so it is added to the operating system as a hardware component. What I would highlight here is that the size of the display doesn't matter, however, because I implemented an automatic resizing algorithm into the game so that Cubion can be played on smartphones of all sizes.

Sound output is not a critical requirement, of course it can be played without sound, but if you want to relax more, the game's music can also help. I've put together a selection of music that isn't intrusive for kids or adults alike, helping to add momentum to the gameplay, and helping to create a relaxed atmosphere.

One of the things I kept in mind when making Cubion is simplicity. There are two reasons for this. One is that you don't need a high end device to run the game, helping to make it accessible to almost anyone. The other is that since one of the goals of the game is to help with rehabilitation, the goal here is not to create a high end graphics and visuals, but functional usability. For these reasons, all smartphones with Android 9 or later versions are sure to have the minimum CPU and RAM specifications required to run.

My goal for the size of the game is less than 100 MB, but at least I don't want to significantly exceed that. This is not an easy task anyway, as only Unity game engine itself is nearly half the size of the game with only an empty scene. To this end, I sought to find a middle ground between the quality and size of the elements that make up the game. The player-controlled cube is a simple basic geometric object, so there was no need to import a model for it. The walls enclosing each level are made of a low poly model. I was a little more lenient with the textures, they are medium quality so that the game also has some visual minimum experience that it offers. Of course, in the case of music, the principle of saving space also applied, so small files also make up the sounds. I want to keep the size of the game lower because I know for myself how confusing it is when we have to wait a lot for downloads, installations, updates, and often have to make

room on our device to perform the desired operation. If I didn't take this into account, I could make the overall graphics values of the game much higher, taking advantage of the possibilities provided by the engine, but as I mentioned, I chose a middle ground. Furthermore, this principle provides another guarantee that the game can run smoothly on less powerful devices. This clearly shows one of the disadvantages of Unity, as a relatively large base file will be generated for each project, which also contains functions and elements that we may not need. Starting with the Pro version, we get a tool that allows us different compression methods. For more serious, large-scale projects, I think this is an essential feature.

3. Presentation of the game and the game engine

3.1 Short review of the game

The game was codenamed Cubion. The target platform is Android-based systems with a touch screen. During the development of the game, great emphasis was placed on having the necessary skills for both healthy users and people with disabilities in their fine motor movements, the former for fun, the latter for rehabilitation. Cubion is a 3D arcade game consisting of progressively more difficult levels, with a new challenge on each level.

The player must guide a cube through various obstacles to get to the green platform at the end of the level. This may sound easy at first, but control of the cube isn't that easy, it requires a lot of concentration, especially on higher levels where a lot depends on the timing and reflexes of the players as well. There is no time limit for completing the levels, but in order to move on to the next level, the player must complete the current one. From a health point of view, it is also important not to allow levels to be skipped, because in this way it is possible to avoid that until the patient has been able to complete the given level, he or she will not be able to get more difficult than at present. This technical detail can ensure phasing in rehabilitation. Furthermore, we can motivate healthy users with their curiosity because they only know what awaits them at the next level if they complete the current one first.

The game engine chosen to implement the game is Unity. There are several reasons for this. One of the reasons is that in terms of the structure of the game, Unity is great for accomplishing this, as my game consists of 3D objects that I endow with various properties using scripts. The other reason is that this game engine is extremely popular among game developers and not only for mobile platforms but also for PC. The third reason is based on personal preference, as I have been a member of the Unity community since the closed beta.

3.2 Unity

Unity is a cross-platform supported game engine that was released in 2005 by Unity Technologies. Its field of application is very wide. It supports the development of 2D, 2.5D, 3D, augmented reality, virtual reality based applications. Because the capabilities of the engine go beyond just game development, it is also used in many other areas, such as interior designers for visual design, simulation in the automotive industry, but we can

even make animated movies with it. Unity is constantly being developed and endowed with the latest technologies and capabilities. The Unity community, which is quite large, has a huge impact on these new updates, as developers base their suggestions on most of them. Unity is currently in its fifth major release. [5]

Table 3.1. shows some of the most important properties of the larger releases of Unity, noting that each release brought dozens of new features.

Table 3.1. Major releases of Unity

Version	Release date	Impact features
Unity 2	2007	real-time dynamic shadows, directional and spotlights, networking layer
Unity 3	2010	native font rendering, automatic UV mapping, extended android support
Unity 4	2012	DirectX 11 support, Adobe Flash support, Mecanim animation tool
Unity 5	2015	improved lightning and audio, real-time global illumination, Unity cloud, new particle effects, overall performance optimization

Unity is available to anyone for free under certain conditions. This is because the company has an extremely well-compromised licensing strategy. The game engine, understand development environment, exists in four variations: Personal, Plus, Pro, Enterprise. Anyone is free to purchase any version, however, in the event of a certain amount of revenue, it is mandatory to trigger the corresponding version. The Personal License can be used free of charge as long as the amount of revenue projected for the last 12 months does not exceed \$ 100,000. If your revenue is less than \$ 200,000, but you want to take advantage of some extra services such as splash screen customization, you are required to purchase the Plus license. If your revenue exceeds \$ 200,000 for the last 12 months, you will need to purchase a Pro or Enterprise license. It is important to emphasize that with the free Personal version we can create full-fledged applications, the difference between the individual editions tends to focus on technical services. For example, Enterprise Edition includes support to facilitate enterprise-level development and communication among developers, as well as diagnostic tools. In fact, this licensing strategy can also be seen as a kind of tax that has to be paid to the company on our revenues, but its rate is balanced, so it is worthwhile for both single developers and

companies to use this product. It's balanced and very fair because while your revenues are small from a market standpoint, you get a tax credit from the company, meaning you don't have to pay, helping your growth, giving your project a chance to grow into the "big ones". [6]

Furthermore, there is a very important and useful feature that Unity Technologies provides for free to all developers, and that is the Unity Asset Store. Content created by other developers, such as models, sounds, scripts, and more, can be downloaded from this store. We are free to upload content that we would like to share with others. The content of the content uploaded here is always decided by the uploader, ie the creator. You can give it for free, but you can also ask for money for it as you specify. I often use this store and have used it in this project as well. I mainly get textures and models from here, so the basics of environmental elements.

For a long time, three languages were natively available for programming and scripting in Unity: C#, Boo and UnityScript. The latter two have already been removed from the repertoire for the fifth version because they are obsolete and have less potential than C#. Incidentally, in terms of the popularity of programming languages, C# is at the forefront, so the camp of language users is also crowded, so there is a lot of help available to developers in various forums. Today, the C# programming language is also referred to as the programming language for video games. It follows directly from the above that Cubion was also written in C#. However, for those who for some reason would not be suitable for this language, there are also various third-party solutions available to them that allow them to connect other languages to Unity, such as C++ or even Python. It's worth keeping in mind that most support and technical feasibility comes with C#. [5]

The Unity editor interface may seem complicated to those who haven't used a similar program yet. We have a lot of information and panels in front of us that give us a lot of information, but we should not get lost in the little things. The thing is, the editor can be used very effectively after a little getting used to. The different modules are quite handy, making the work dynamic. Of course, for those who don't like the default layout, it is possible to customize it to suit individual tastes. For me, the default layout is right, that's what I get used to, so I usually use that.

Getting to know the Unity classes is very important for scripting. All classes come from the MonoBehaviour class, both the ones we make and the ones we already have available. The MonoBehaviour namespace is the "soul" of Unity beside UnityEngine namespace. This namespace includes elementary functions to handle the scenes such as

Update or coroutine calls and every other event that can happen. The other important thing we take advantage of in scripting is that actually everything we see or do in the editor exists with a class of related variables and functions.

From a developer perspective, Unity provides even more support than any other game engine and development environment. This is nothing more than full market support for the games we make. By this I mean that with the development environment comes the opportunity to publish our game in the App Store or the Google Play store, furthermore it natively supports the placement and management of paid advertisements with several types of options. If we use advertising, which can generate our revenue, we also get a web-based account where we can withdraw the money we earn, as well as a wide range of statistics to help us with our work and market strategy. I would note that Cubion does not contain any paid content or advertising, as the application is for health purposes in part, however, I do not rule out the possibility that a modified version of this game will be available in the Play store in the future with extra content and in-app advertising.

I believe that Unity gives everything that is needed to be successful in the market, the rest is up to the developer to figure out how to take advantage of it, how creative it is, how ambitious it is to its goal. Moreover, in addition to the technical toolbox, the company also publishes tutorial videos and webinars, from beginners to advanced, to try to help everyone and pass on new ideas and knowledge.

4. Implementation

Game development usually consists of two major parts that can have additional components. These two main components are: game world, which includes visuals and graphics, music, storyline, player character and the other component is the code that brings the game to life. In game studios, separate groups and subgroups work in these two parts, as different competencies are required to design and implement them. This is one of the biggest challenges for indie game developers, as they have to excel at different things and do in one person what a studio does with up to a hundred or more developers.

There are two different ways of scripting: code separation and code encapsulation. The essence of code separation is to create many small, logically separated scripts, that is, to treat each function as a separate entity. In the case of the encapsulation approach, we have fewer but larger scripts, the functions between which there is any connection are implemented in one script. I myself prefer the code separation way, so I used this as usual in this project. The reason for this is that it is more practical to treat each desired feature as a separate script, as this will give me better overview of the game's structure and make fixes faster if I need to reach into the scripts. Also, if I need to modify one of the scripts, I don't have to review hundreds of lines for just a fraction of it. It is also positive that if I reach into a script, I am less likely to put a bug in the code, for example due to a lost dependency. For these reasons, among others, even AAA-renowned studios use this approach in the field of game development for the most part.

I would note that since I am an independent game developer, I don't have to follow any corporate coding standards, I can enjoy the full freedom of creation and creativity. In the case of independent game developers, the development and coding can be seen by one person: the developer himself, unless he creates game elements and scripts specifically for sale or sharing these with the community. Applying these principles, the development process and coding technique is based on my logic, according to my own taste.

The starting point for development in Unity is the so-called "Scene". Everything we work with: game components, scripts, etc. belong to a Scene. In fact, this is where we work. A Unity project has a minimum of one Scene and the more complex the game, the more Scenes will be needed.

Cubion consists of a total of 17 Scenes, these are the 11 levels, the loading screen and menu scenes. Of course, each scene has graphics and programming related components and they together form the game. The implementation of the main components is presented in the following subchapters. One subchapter is about how the graphic elements, music, character so the game world were implemented, while in the other subchapter I only present the scripting related work.

4.1 Implementing the game world

Game development usually begins with the import of existing, already available components into the project, such as models, sounds, materials, and so on. It is necessary to create a starting point to start building the game. In practice, it is natural that as game development progresses, new components - including music, models, scripts - will be implemented into the game being developed. So in general, as the game develops, its size increases.

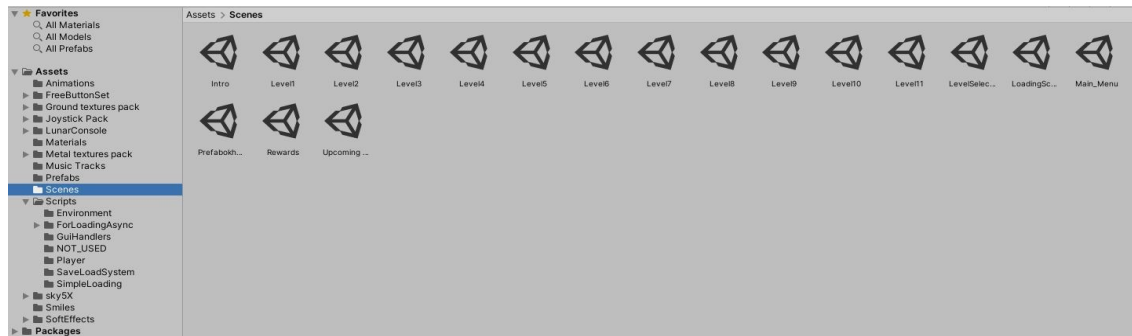
In the case of the Cubion, since one of the goals of the graphic design is to be easy to see and understand, it should be minimalist but also „tasteful”. One positive effect of this is that the game will be able to run on hardware with fewer resources. In this case, importing all the components needed to start the development was satisfactory.

- **Implementation of the levels and the player’s character:**

In the Unity editor, we can create basic 2D and 3D objects, such as spheres, cubes, on which we can apply some transformations as desired, such as resizing, rotations. Cubion is made up of most of these elements, including the player character, except for the walls on the levels, which is an imported 3D model. This model was purchased from the Free3D community, which has a so-called royalty-free license, so it can be used freely, even for paid content. It is also important to note that this model falls into the so-called low-poly category, with a small size of 32 KB, which supports low resource requirements, as it does not take much time and resource to render it per frame. If we get any model from a modeling community, then in addition to the fact that it is available in several formats, it is customary for the maker to add a texture, which actually gives the gray, bare model itself colors and graphics. In this case, the wall in the game was originally made of concrete cordon, but I didn’t use the texture made for it, but instead endowed it with a more colorful, friendlier color. In addition to the wall model, a royalty-

free music, ground textures and a royalty-free skybox from Unity store have also been added to the list of imported game environmental assets.

The import method itself is very easy: in the project view, right-click import New Asset, but the grab and drop method is more convenient for me, so I just drag the necessary files into the project. It is advisable to organize the components in the project view, to group the components into folders, especially in the case of large projects, so that we can get a clear view of the assets (4.1 picture).

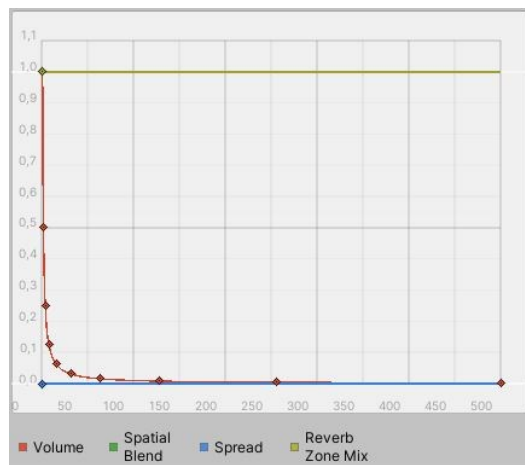


4.1 picture Assets and Scenes are organized

After importing most of the assets a new scene for Level 1 is needed: file → New scene (ctrl + N). With the new scene, we always get two things by default, which of course we can delete if we want. One is a so-called Main Camera component and the other is a Directional Light. I won't delete them because I need them. The camera is actually our eyes, it is responsible for the rendering of the frames, so if we don't assign a camera to a scene, we don't see anything but a black image when we enter the game. Once we have a scene, so a workplace, we can start building up the level. I use a so-called Plane component for the game surface on which the player can move. This is available from the GameObject → 3D Object → Plane menu. From the Planes I create the path of the level with the help of the corresponding transform component (scaling, position). Then I drag the 3D cordon model from the projectview to the scene, so it becomes part of the level. From this model I make the walls of the surface. At the end of each level, which symbolizes the goal, a special Plane serves. I will make this functional later. In the following, I will add a so-called Mesh collider component to the Planes by clicking on the object from the Inspector → Add component menu. This is necessary for Unity's physical engine to reach this component, so our player, for example, won't go through it, but will stay on top as if it were solid ground. Like this, I add a Mesh Renderer component to this game object, and I assign a texture called Dry ground pattern to it's Material. As a result, I get an earthy-like soil from the gray Plane. This type of texture

can be found up to level 6 and from level 7 this texture is replaced by Grass ground pattern to maintain the graphic variety. For the Planes that symbolizes the end platform it's material color is set to green and these Planes do not have any special textures. A box collider component must also be added to the wall model so that it cannot be traversed, and a material component that I set to orange.

What else belongs to the level is the sound. In `GameObject` → `Audio` → `Audio Source` I add such a game object to the scene and an audio listener component to the camera. I then assign one of the imported music to the Audio Source component. I note that I have a total of 3 types of songs available and I only use one of these three types of songs in all scenes. However, they all sound a little different, so you may hear there are more songs than that. The trick to this is that Unity also has a basic audio editing interface that is integrated into the audio source. Using this tool, I change the sound frequency and speed of the songs on each scene (4.2 picture).



4.2 picture A detail of the sound mixer

In what follows, I set up the skybox so that it is not a smooth light blue, but a more stylish starry dark sky. To do this, set the Clear Flags of the camera component to Skybox, then add the desired skybox under `Window` → `Rendering` → `Lighting` → `Environment`.

There are rotating wiper objects on some certain levels, which are also made with the help of Unity's built-in base toolbar. These objects are made of cubes which can be accessed from the following menu: `GameObject` → `3D Object` → `Cube`. The material color of these wipers are set to blue and red otherwise they do not have any special textures.

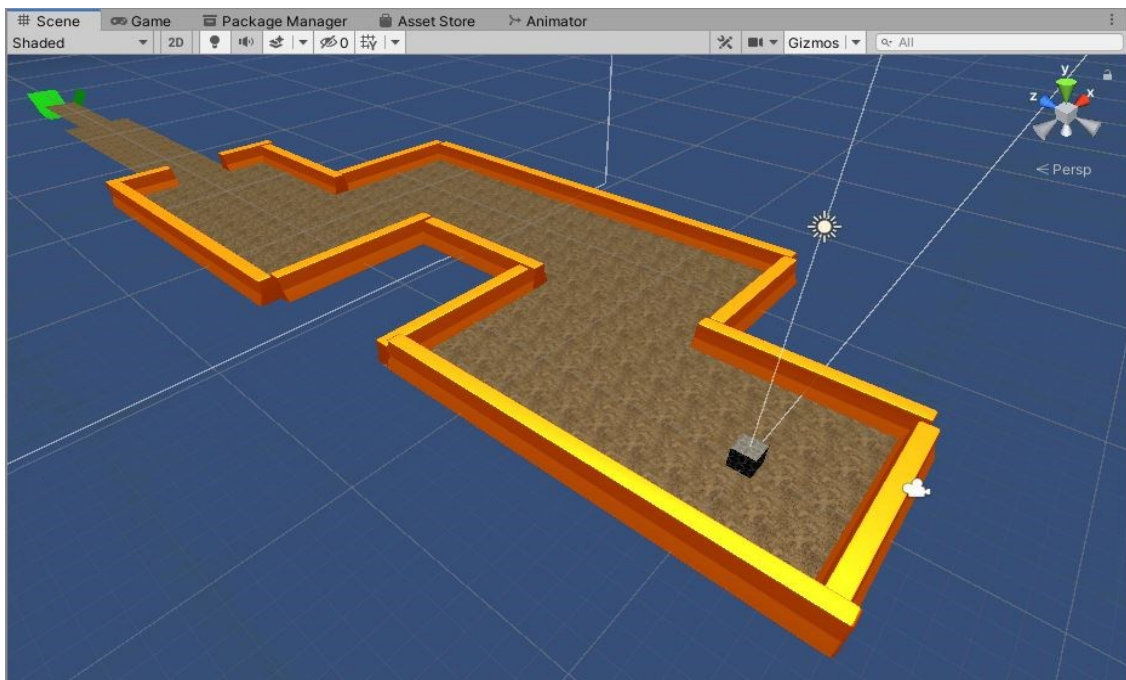
In addition, some levels contain special effects designed to simulate fog, which are implemented by particle systems, which is also available from the `GameObject` menu.

The most important element that can be found on every level is, of course, the player's character. The character consists of the same cube as the rotating wipers. This game object also has a Mesh Renderer component with a SpecialMetal texture. What makes this game object special from the others is that it has even got a Rigidbody component, that applies physics provided by the game engine (4.3 picture).



4.3 picture Rigidbody settings for the character

After the first level is implemented (4.4 picture), the making of additional levels is faster, as all other levels consist of game objects already implemented on the first level. The very mechanism that speeds up the implementation of the other levels is the use of so-called prefabs, which can be used to save parameterized game objects, which can then be used as ready-made building blocks. This feature is extremely useful if the game has



4.4 picture Scene view of Level 1

a lot of similar game objects and I want to change something about it later then I don't have to modify each such game object in each scene one by one, just modify the prefab and it applies to all the objects derive from it. By analogy with Object Oriented Programming, I have actually made a class of the game element and the next time I add it to a scene, it will be instantiated. To do this, grab the desired game object and drag it into the project view, and a so-called prefab will be created. It is even customary to create a special scene where I create prefabs, but this scene is not added at the end when building the game, it is only used to create these prefabs.

- **Implementation of GUI:**

The user interface is based on the so-called Canvas, which we can be added to Scene from `GameObject` → `UI` → `Canvas`. Canvas functions as a placeholder, where such various elements can be placed that allow the player to intervene in the game or just get information from the game. In the case of Cubion, there are two Canvas for each level. One Canvas provides the player's control interface, animated text and the stopwatch, while the other provides an information interface available at the end of the level, complete with move to the menu and the next level buttons.

I got the joystick pack for the control interface from Unity store. It is available for free and it is also recommended to use this package for mobile games because most Unity based mobile games have implemented this joystick package, be it iOS or Android. The operation and appearance follows the standard, usual one. Note that there are three types of joystick variations included in the package:

- fixed: in this case, the position of the joystick is fixed and does not move during the game.
- dynamic: the starting position of the joystick is fixed, but can be moved during the game as the player reaches the outer border of the controller.
- floating joystick: the starting position of the joystick is not fixed, it always appears where the player has just touched the screen.

All three types have their own place of use. In the case of Cubion, the floating joystick is the one that implemented to make the controlling of the player's character easier on a larger screen, as well as the user interface to flexibly adapt to the player's needs. The first part of the joystick implementation is that dragging the prefab associated with the proper joystick type onto the Canvas. By adding the joystick prefab to this Canvas, this interface has been implemented visually. In order for it to work, the associated script –which is

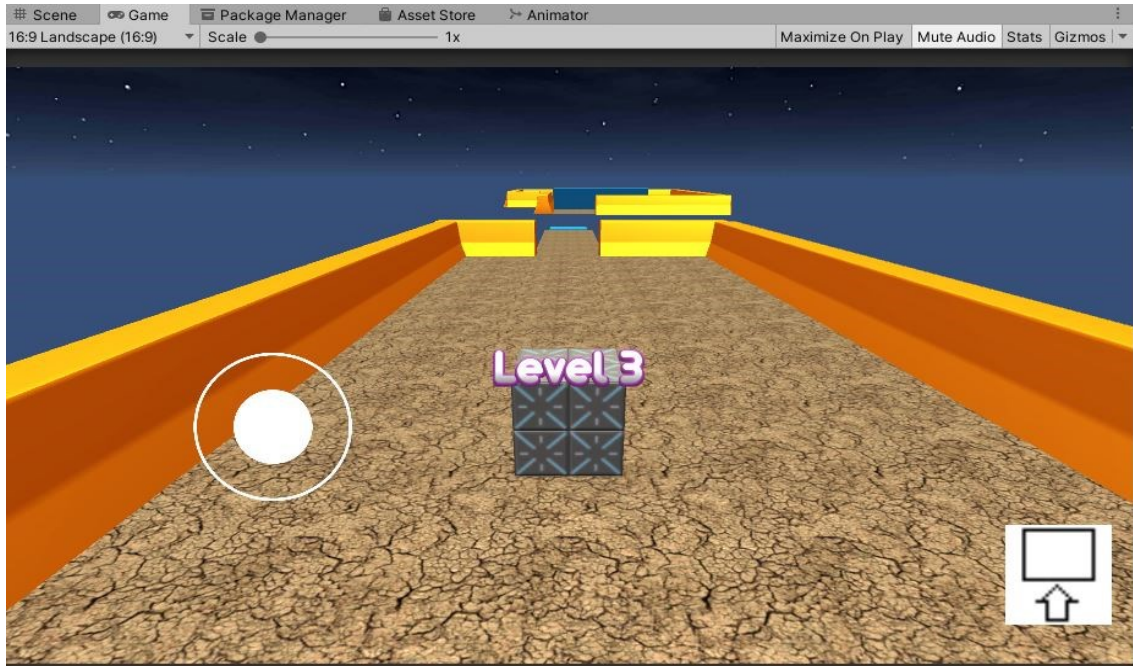
included in the package- need to be assigned to the player's character as a script component.

The jump function requires a virtual button, which can be added from the GameObject → UI → Button menu. This component is assigned a simple image created in Paint, which refers to the function of the button.

The primary GUI has two more game objects: a stopwatch and a welcome text at the start of each level. Both gameobjects type is text, that can be added to the Canvas from the GameObject → UI → Text menu. The font of every text in the game are set to "Arista Sep 2.0". The difference between the two game object on this Canvas is that the value of the stopwatch clock is null by default which is changes during runtime, while the value of the welcome text is fixed. Additionally, the welcome text includes an animator component, because I want to give this game object a transition effect. I created the proper animation using Unity's built-in animator module, which is otherwise key frame based and the company has put a lot of emphasis on developing this module over the years, but if you don't want to use the built-in feature, you can import animation files made in third party applications like Blender or Maya as well. I would point out that animation endows the game object with a predefined behavior, which makes the behavior of an animated gameobject less flexible than a scripted one, but in this case, I only expect a constant behavior from this game object.

The primary GUI implemented as discussed above is shown in the 4.5 picture. Notice that the stopwatch is not visible in the picture, because, as I mentioned earlier, the default value of this game object is set to null - until it is overwritten by the corresponding script - and the developer game view shows the actual scene with the initial values.

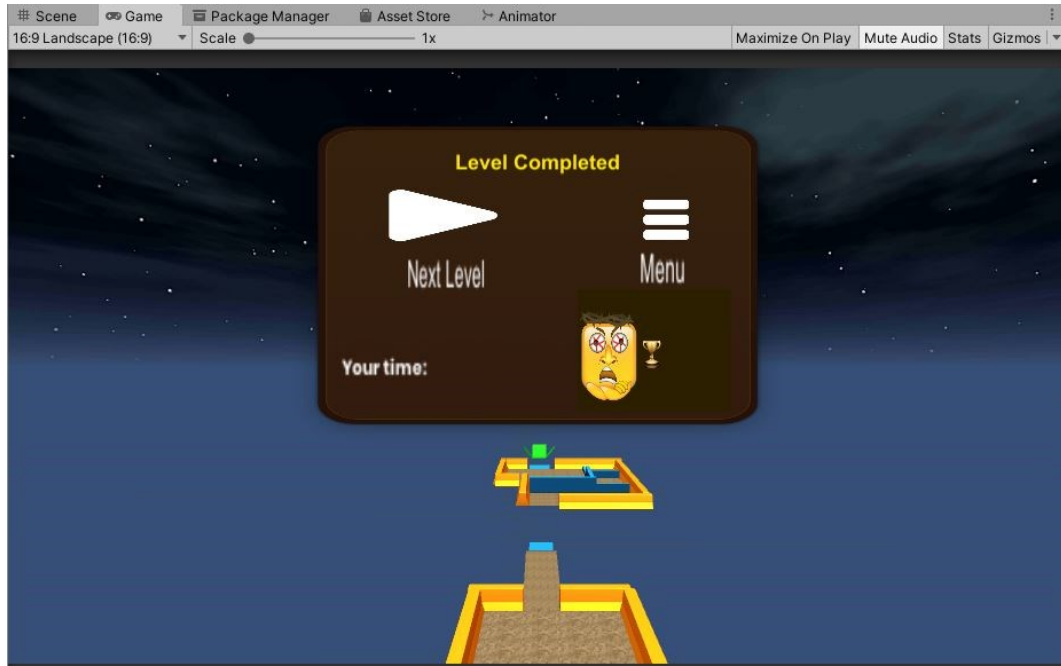
The other Canvas or as I call it secondary GUI that appears after completing a level is very similar in structure to the primary GUI. The only difference between them is that the secondary GUI does not have a joystick controller element, but it has Image elements, which the primary GUI does not.



4.5 picture Implemented primary GUI design

The elements belonging to the secondary GUI are placed in front of an Image component as a placeholder. This game object can be added to Canvas from the GameObject → UI → Image menu. The image I associated with this object is a square with a rounded side in a brown hue. I added two virtual buttons to this Canvas that are the same type as the jump button on the primary GUI. I assigned to these buttons the standard menu and next level icons already familiar in various games, which I downloaded from the Unity store for free. Furthermore this Canvas has two text objects with fixed value - "Level completed" and "Your time"- and a text element that displays the time result with default value set to null, like the stopwatch, which is overwritten by the appropriate script. Three more Image game objects are required to return the rating for the achieved result. To these three objects I added the smileys I made, which I placed on top of each other. Which one is displayed when the level is completed is controlled by the corresponding script. The 4.6 picture shows the implementation of the secondary GUI in developer game view. It can be seen that the top smiley which is the last placed one appears in the GUI in developer mode.

From a technical point of view, it is also important that the two GUIs have two separate cameras, which are responsible for displaying the corresponding GUI to the player, which is controlled by the appropriate script. Of course, it is enough to create these GUIs once as a prefab, and then add them to any Scene at will, as I did.



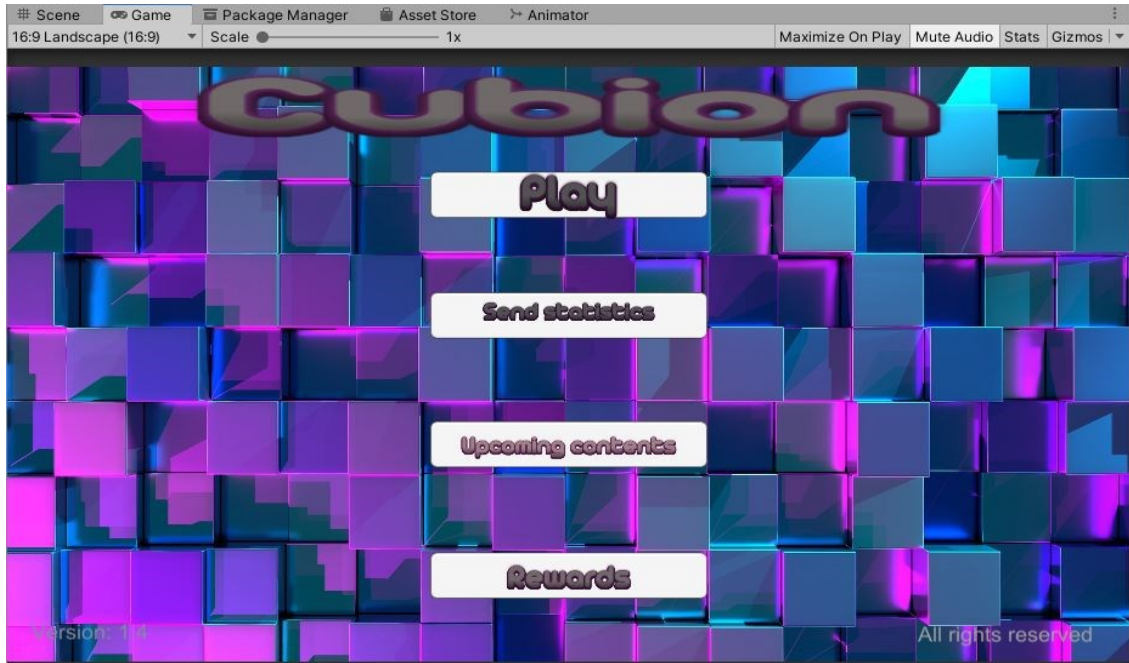
4.6 picture Implemented secondary GUI design

- **Implementation of the menu and loading screen:**

The Scenes for the menu and loading screen were created using the same technique I used to implement the GUI, since they actually belong to the GUI, but in a different approach. So the base here too is a Canvas, on which I place the necessary game objects. During the development, there was a demand to make the statistics created in the game available not only on the given device, but also to be able to be forwarded to a specified e-mail address if necessary.

Each Scene that is part of the menu has an Image component that serves as a background image. In front of this wallpaper, each menu scene has the appropriate virtual buttons for navigation based on the conceptual designs.

In the case of the main menu, the conceptual design includes an optionally usable placeholder for a virtual button. It is necessary to use this placeholder to satisfy the newly created data sending function, so I have inserted a virtual button called "Send statistics" in place of this placeholder. The main menu implemented according to the conceptual design and the new demand is shown in the 4.7 picture.



4.7 picture Implemented Main Menu supplemented with the new demand

The main function of the loading screen is to provide feedback on the status of the level. To express the loading status, I chose a graphical solution, i.e., a progress bar rather than a numeric format. I built this Scene by adding an Image component to the Canvas to display a green gradient background, and then placing the Text objects that contain the motivational messages and the progress bar itself in front of that background. I made the progress bar from two Image game objects. One only serves as an empty frame. I shaped this into a rectangle of the right size, which I placed at the bottom of the



4.8 picture Implemented Loading screen

screen. Then I added another Image game object to this element as a child component, which role is masking. I added a blue gradient picture to this mask. The progress bar works by controlling the completion of this mask between 0 and 100 % according to the loading status. The implementation of loading screen is shown in the 4.8 picture.

4.2 Game scripting

In this subchapter I present the coding of the game, ie bringing the game to life, during which I try to highlight only the essential parts, but the full coding can be found in the appendix. I would like to mention a few more important things about game scripting in Unity, such as the fact that this is a smaller project compared to larger games, so there is less use of object-oriented programming. The other important thing is related with the access modifiers. A program written in C# generally uses many private variables, but in Unity usually developers avoid it because only the public variables are visible in the editor at runtime, which helps a lot in debugging. However there is another way to make this available, if someone still insists on private variables, the modifier should be preceded by "[SerializeField]". It can be seen that in the latter case we get longer and harder to read codes. Games made in Unity are single-threaded, despite the fact that the C# language and its framework supports multithreading, but according to the documentation, we need to avoid from creating additional threads in Unity, because it can result unpredictable operation, which is because the game engine itself calls the scripts at the right time using an internal state machine and we can override this if we try to use multithreading. It is also important to point out that every script we write must inherited from the MonoBehaviour class, otherwise it will not be called by the game engine.

The game engine calls the functions in a specific order and mode. With the special built-in functions, we can control when our code should execute (4.9 picture). The most commonly used such functions are Awake(), Start(), Update(), FixedUpdate().



4.9 picture Calling order of MonoBehaviour class functions

The Awake() and Start() functions are called only once when a given Scene is loaded, so these are usually used for initialization tasks. Update() and FixedUpdate() are called during each frame, the difference being that if the script also applies the game engine's physics in some form, FixedUpdate() is used to work properly.

I divided the scripts into six groups according to which part of the game they are associated with, and I created the structure of the assets folder accordingly. The six groups are Environment, ForLoadingAsync, GUIHandlers, Player, SaveLoadSystem, SimpleLoading. In the following, I present the essential part of the scripts according to this grouping.

- **Environment group:**

There are four scripts in this group, which are: ColissionDetect, LiftPlatform, MPlatformRigidbody, RotateObject.

The RotateObject script rotates a specific type of game object around its y axis at a set speed. To set the speed, I created a float type variable called rotationSpeed and then called the rotation function for each frame using this variable for speed.

//Rotating game object around y axis

```
transform.Rotate(0,rotationSpeed * Time.deltaTime,0);
```

The function of the LiftPlatform script is to move the game object continuously between two coordinate points at a specified speed. The direction of the movement is vertical, i.e., only the y coordinate of the 3-dimensional vector space needs to be manipulated. The Vector3 class is suitable for storing coordinates. I use the MovePosition function to move the game object.

//lift is going down

```
liftbody.MovePosition((transform.position + new Vector3(0, -1, 0) * Time.deltaTime * liftspeed));
```

//lift is going up

```
liftbody.MovePosition((transform.position + new Vector3(0, 1, 0) * Time.deltaTime * liftspeed));
```

The MPlatformRigidbody script is very similar to the LiftPlatform script, but its function is to move the game object horizontally between two coordinate points.

//Move to destination

```
objectRigidBody.MovePosition(transform.position + transform.forward * Time.deltaTime * movespeed);
```

//Move to start position

```
objectRigidBody.MovePosition(transform.position - transform.forward *
Time.deltaTime * movespeed);
```

The function of CollisionDetect script is a little more complex than the previous ones. Its main function is to detect whether the player's character touches the designated finish zone. Other tasks include switching between GUIs and unlocking the next level. As long as the player's character not touches the green platform at the end of the level, it displays the primary GUI. Once the player has reached the finish line, it switches to the display of the secondary GUI and unlocks the next level. In order to detect a collision between objects, it is necessary that these game objects have some kind of collider component, which is otherwise given to all 3D game objects by default. The OnCollisionEnter (Collision collision) function can be used to detect collision between such game objects. This function is called only if a collision has occurred between the objects.

```
//If cube entered then switch GUI and unlock next level
```

```
public void OnCollisionEnter(Collision collision)
```

```
{
    If_cube_entered();
}
```

```
//This function handles the GUI switch and level unlock
```

```
private void If_cube_entered()
```

```
{
    maincamera.enabled = false;
    secondarycamera.enabled = true;
    userController.enabled = false;
    finish = true;
    time_Canvas_value.text = Stopper.time_for_global_use;
    if (leveltoUnlock > SaveLoadSystem.Progress)
    {
        SaveLoadSystem.Progress = leveltoUnlock;
        SaveLoadSystem.SaveProgress();
    }
}
```

- **SimpleLoading group:**

There are two scripts in this group: SimpleLoad, LoadSpecificSceneTimed. These scripts implements simple scene loading without a loading screen, the only difference between them is the latter has a delay that I use for the intro scene. I use the SimpleLoad script for every button in the menu system, except the level choosing. The level name is the same as the scene name, for which I created a string variable called levelToLoad. I set the value of this variable directly in the editor by always typing the name of the next level in the current level.

//On button touch call the game engine's scenemanager method

```
public void OnPointerDown(PointerEventData eventData)
{
    SceneManager.LoadScene(levelToLoad_Simple);
}
```

- **Player group:**

This group contains two scripts: PlayerController and PlayerJump. These scripts handling the interaction with the player.

The PlayerController script calls the FloatingJoystick and according to the position of the joystick applies force to the player's rigidbody.

//Get joystick position and apply force to the character

```
Vector3 joystickDir = new Vector3(floatingJoystick.Horizontal, 0,
floatingJoystick.Vertical);
player.AddForce(joystickDir * forceMultiplier * Time.deltaTime,
ForceMode.VelocityChange);
```

This script contains variables for accessing joystick positions which will be needed to save in the activity. Moreover this script implements the respawn method, as this script is linked to the player. Since I designed the game so that the respawn does not happen immediately, but there is a short delay after the player falls off the ground, the Respawn function itself must be called in a Coroutin.

//If player under -5.0f set position to starting position

```
void RespawnPlayer()
{
    if (player.position.y < -5.0f)
    {
```

```

        player.transform.position = startposition;
    }
}

```

The PlayerJump script applies an impulse force to the y axis of the player when the jump button has touched and the player has connection to the ground.

//Apply impulse force to the y axis of the player

```

    if (ppl.grounded == true)
    {
        rb.AddForce(jump * jumpForce, ForceMode.Impulse);
        jumpstate_for_global_use = "1";
    }

```

In addition, if the player has contact with the ground, the jump button will turn green, indicating that the jump is available, otherwise it will be white. I achieve this by using the ColorLerp() function. The script contains a variable to store the state of the jump which will needed to save in the activity.

- **ForLoadingAsync group:**

There are four scripts in this group, which are: SceneLoader, SceneLoadData, ProgressBar, LoadSpecificScene_Loading.

The ProgressBar script handles the proper filling of the loading bar based on the SceneLoader status.

//Control of the loading bar's mask

```

fill = (float)current / (float)max;
mask.fillAmount = fill;

```

The other three scripts build on each other. Whenever I use Async loading, I first call the loading screen. This scene receives the name of the scene that need to be load as a string, and then calls the Async loading method in the background. When the Async loading is ready, I switch from the loading screen to the desired scene. The SceneLoadData script contains the requiredSceneToLoad string, which I set directly in the editor on each scene that contains Async loading. The LoadSpecificScene_Loading script gets this string and calls the loading screen with this string. The loading screen calls the SceneLoader script that starts the Async loading method with the string it gets from LoadSpecificScene_Loading script.

//Start Async loading and disable scene activation

```

loaderOperator =

```

```

SceneManager.LoadSceneAsync(SceneLoadData.requiredSceneToLoad);
loaderOperator.allowSceneActivation = false;
//If loading state is > 0.89f (0.9 is the maximum = ready) activate the loaded scene
if (loaderOperator.progress > 0.89f)
{
    loaderOperator.allowSceneActivation = true;
}

```

- **GUI group:**

This group contains seven scripts: AutoScale, IntroScale, AndroidExit, CameraFollower, Internetconnection, Stopper, SmileyControl. These scripts are related to parts of the GUI and the camera.

With the AndroidExit script, I implement the function to close the game with the physical button of the device. This script runs continuously when the main menu is loaded.

```

//If Keycode.Escape triggered
Application.Quit();

```

In the Internetconnection script, I implemented that if there is active internet connection then open the game's web page using the default browser. The address of the website is stored in a string variable.

```

//If internet is reachable then call the browser with the specified string
private readonly string website = "https://cubiedemonstration.weebly.com/";
Application.OpenURL(website);

```

With the IntroScale script, I lend a little dynamics to the intro scene. I adjust the magnification of the camera continuously during the two-second life of this scene.

```

//Apply scaling in x and y axis of the camera
Vector3 newscale = new Vector3(actScale.x + 2 * Time.deltaTime, actScale.y + 2 *
Time.deltaTime, actScale.z);
imageIntro.transform.localScale = newscale;

```

The AutoScale script is part of every camera object. Its purpose is to provide a 16: 9 aspect ratio.

```

//Determine the game window's current aspect ratio
float windowaspect = (float)Screen.width / (float)Screen.height;
//Current viewport height should be scaled by this amount
float scaleheight = windowaspect / targetaspect;

```

In the CameraFollower script I implement that the camera follows the player at a fixed tracking distance without changing the orientation.

```
//Set a fix offset and lerp the camera with it without orient change
Vector3 desiredpos = target.position + offset;
Vector3 smoothpos = Vector3.Lerp(transform.position, desiredpos, followspeed);
transform.position = smoothpos;
```

With the the Stopper script I control the value of the stopwatch clock that attached to the primary GUI. It is based on a simple timing that starts when the player touches the display at first on a level and ends when the level is completed.

```
//Calculating time members
float actual_time = Time.time - start_time;
string minutes = ((int)actual_time / 60).ToString();
string seconds = (actual_time % 60).ToString("f2");
```

The SmileyControl script is responsible for controlling the graphical feedback about the player's performance on the secondary GUI. There are three categories for evaluating player performance - Good, Avarage, Bad - with different smileys depending on the time achieved. I created time intervals for the three categories, which can be customized for each level in the editor. On each level, I set these intervals based on my own performance supplemented by some margin of error.

```
//Create the intervals and proper smiley control
if ((Stopper.time_minutes_for_global_use <= good_time_minutes) &&
(Stopper.time_seconds_for_global_use <= good_time_seconds))
{
    goodsmiley.enabled = true;
    r_good = true;
}
else if ((Stopper.time_minutes_for_global_use >= bad_time_minutes) &&
(Stopper.time_seconds_for_global_use >= bad_time_seconds))
{
    badsmiley.enabled = true;
    r_bad = true;
}
else
{
```

```
avaragesmiley.enabled = true;
r_avarage = true;
}
```

- **SaveLoadSystem:**

There are five scripts in this group, which are: SaveLoadSystem, StatisticsHandler, LoadUnlockManager, RewardsUnlockHandler, EmailHandler.

The SaveLoadSystem script is used to save and read the game progress through the static int progress variable. During save, the number of the level that is the highest and completed is saved in the PlayerProgressV1.txt file. During the reading of the progress, the value of the progress variable has been set to the file. During the reading of the progress, the value of the actualProgress variable has been set according to this file.

//Writing the PlayerProgressV1.txt

```
FileStream stream = new FileStream(path, FileMode.Create);
coder.Serialize(stream, progress);
```

//Reading from PlayerProgressV1.txt to progress

```
FileStream stream = new FileStream(path, FileMode.Open);
progress = (int) decoder.Deserialize(stream);
```

LoadUnlockManager is responsible for setting the availability of levels based on the value of the progress variable after reading PlayerProgressV1.txt. I implemented the accessibility mechanism itself by modifying the accessibility of the buttons that connected to the levels. The evaluation of accessibility was implemented in a switch-case structure with the actualProgress variable.

//evaluation of accessibility 2-11 according to levels

case 2:

```
l2.enabled = true;
unlock_L2.enabled = false;
break;
```

RewardsUnlockHandler is very similar to LoadUnlockManager. This script handles the unlocking of the three achievements in the Rewards menu. In this case, I do not set the availability of buttons, but the texts and moving game objects for the achievements. Also, the value of the actualProgress variable is used for evaluation, but here I do not use a switch-case structure, but an interval-like division with an if-else structure.

//if actualProgress >= 5 && actualProgress < 8 then enable first achievement

```

first_Mile_Stone.SetActive(true);
first_Text.enabled = true;
no_Reward.enabled = false;

```

The task of the StatisticsHandler script is to save the player's actions on each level to the device. The script first looks at the existence of this folder named CubionStatistics in the game directory and if it does not exist then creates it. In this folder the player's actions in txt format are saved for each level. The structure of statistics file can be divided into three parts: metadata, detailed actions, summary. The metadata contains the name of the level and when the player started it. Detailed actions contains the joystick positions, the state of the jump button and the value of stopwatch clock for the i. rendered frame. I would also like to point out that in this script I also implemented that each file name gets a unique identifier.

```

//ensure that statistics files have unique identifier.
r_random = System.Math.Round(Random.Range(0.0f, 10.0f), 2);
datapath = $"{folderpath}/{actual_active_scene}_{date}_id{r_random}.txt";
//format of detailed actions
statistics_data.Add($"{index};{Stopper.time_minutes_for_global_use};{Stopper.time_seconds_for_global_use};{PlayerController.joystick_position_x};{PlayerController.joystick_position_z};{PlayerJump.jumpstate_for_global_use}\n");

```

In the EmailHandler script I implement the new demand I got during development. Access to this feature is tied to a secret password that I hard-coded in the script, which I set to Admin. The script first collects the statistics files stored on the device, then compiles an email using the .Net Email class and adds all such found files as attachments. The email is being sent using SMTP, which I set up for a gmail account based on google documentation. The sender and recipient of the email are the same address: Cubionreports@gmail.com

```

//set up SMTP for google
SmtpClient client = new SmtpClient("smtp.gmail.com", 587);
client.EnableSsl = true;

```

5. Test

5.1 Debugging

One of the most common and easiest to fix errors in object-oriented programming is `NullReferenceException`. This occurs when the developer try to access a member on a type which value is null. The source of the error is minor inattention on the part of the developer. I highlight this erro because I made it myself during the development of the game.

I made the mistake of not setting the correct component with the `ColissionDetect` script. The child of the secondary camera named `Camera_end` is `LevelCompleted` canvas. I used the script to set the camera component and not the canvas. This caused that when I deactivate the `Level_end` component at starting of the level its child's gets Null value and when the level has been completed I would refer to this child component but its value is Null. To fix this bug all I have to do is refer to the parent component in the script instead of the child component.

It is easy to fix this type of error, but it is difficult to find the source of it, because it is a runtime error. When I started the game in Unity during development, I got the following error:

```
NullReferenceException: Object reference not set to an instance of an object  
ColissionDetect.Start () (at <9621ed5885ce4e21aa7ec9268b37fad0>:0)
```

From this, I did not get the exact source of the error, only which script causing it. To get a more accurate picture of the error, I had to use the so-called online debugging method. A Debug version –Development Build- had to be created for this game, which is available from the Build menu. On the device side where the game is running, developer mode must be enabled, which allows the device to send real-time data to the computer via USB. Access to this feature varies by manufacturer. I used a Samsung device that made it easy to access this feature. After these it is necessary to type the following command at a command prompt on the computer side to start monitoring:

```
adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG
```

In the debug environment set up like this, when the error appeared I got the following:

```
11-28 09:27:39.043 24756 24822 E Unity : NullReferenceException: Object reference  
not set to an instance of an object  
11-28 09:27:39.043 24756 24822 E Unity : at ColissionDetect.Start () [0x000ad] in
```

Z:\Unity\Cubie\Assets\Scripts\Environment\ColissionDetect.cs:38

Based on this, it was easy to fix the error.

5.2 Performance

In this chapter, I will examine whether the game meets these requirements, which have been specified from the side of the device. Such a requirement was that the size of the game should be less than 100 MB if possible, but in no way deviate much from it (Table 5.1.). The other and most important requirement was that the game should not require a lot of hardware resources, so that older, cheaper devices would be able to run the game as well. The other and most important requirement was that the game should not require a lot of resources, so that older, cheaper devices would be able to run the game as well. I used CPU and RAM usage as the basis for examining this criterion.

Table 5.1. Comparison of devices

Device	Android ver.	Size after install
Samsung	8	92,48 MB
Sony	9	92,43 MB

It can be stated that the game does not exceed the intended size, although I note that with subsequent expansions, such as new levels, it is natural that it will exceed this intended number.

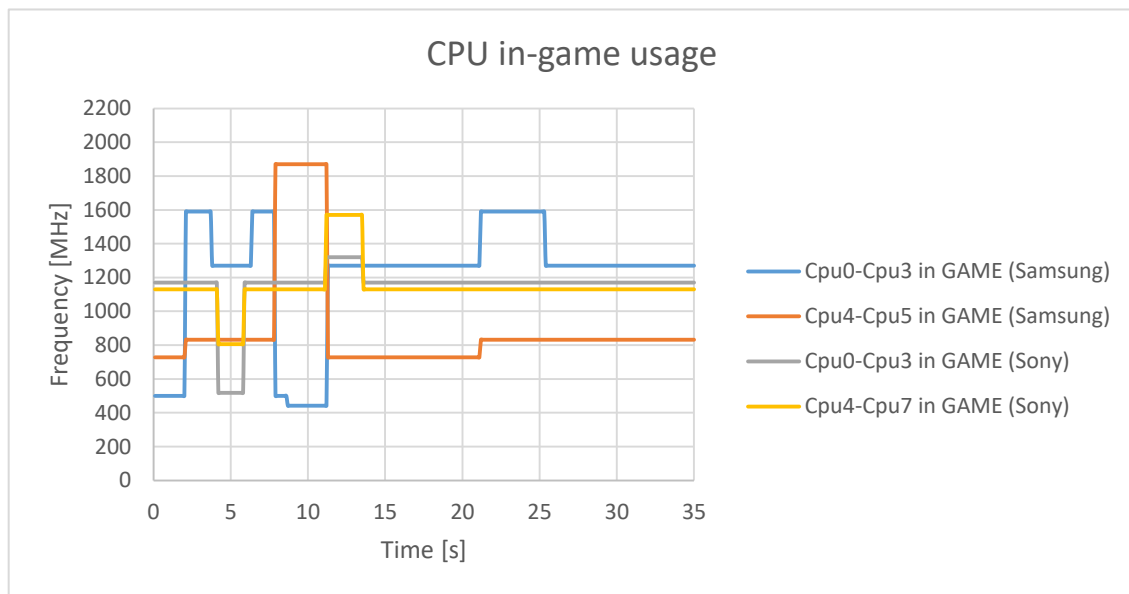
I tested the game on two devices: Samsung Galaxy S7 edge, Sony Xperia XZ. The test itself is not exhaustive, but it is representative of the extreme performance of the game, which is sufficient in this case, as examining the worst-case scores is sufficient to determine whether or not it is suitable to run on less hardware-intensive devices or not. One of the most complex level is Level 5, because this scene contains lot of game objects and effects, so this level is one of the most representative in the mean of extreme value of performances.

To measure resource usage, I used an application called “System Monitor Float Free by 98Apps” available for free from Google Play, which I installed on the test devices. I measured two values for each resource, one is idle usage when no application is running and the other is when the Level 5 scene is in use.

The results of the CPU test (5.2 diagram) show that Sony is trying to distribute the load more evenly than Samsung. Sony has two more cores available, so this device can run the game at a lower clock speed, while Samsung compensates for the fewer cores

with a higher clock speed. As a result, Samsung has a slightly higher heat up than Sony due to the higher CPU clock.

It can also be observed that in both cases most of the changes are between 5 and 15 seconds. This is because most particle effects are found at the beginning of the level, which require more resources to render than the rest of it. When the player leaves this area and does not need to render these particles, it is seen that the clock drops to a near constant value. Overall, these values are adequate, far from the maximum clock and load capacity for these devices, especially on newer devices, so it shouldn't be a problem for the CPU to run the game even on older devices.



5.2 diagram In-game CPU usage comparsion

The RAM usage is around 1.80 GB in both cases, noting that the system itself uses 1.71 GB in both cases when idle. A very slow upward trend can be observed, which is rooted in the StatisticsHandler script mechanism, but it takes long minutes for the increase in memory usage to be significantly visible. However, in a later update, this could be optimized by storing and writing statistics in smaller time blocks, thus reducing and keeping memory usage constant. All in all, 2GB of RAM is enough to run the game, which is what even older devices have.

5.3 Functional Test

In this chapter, I examine whether the data extracted from the actions saved in the statistical files during the game are suitable for showing the difference between people with impaired and healthy people in their fine motor skills, and for monitoring the

patient's progress.

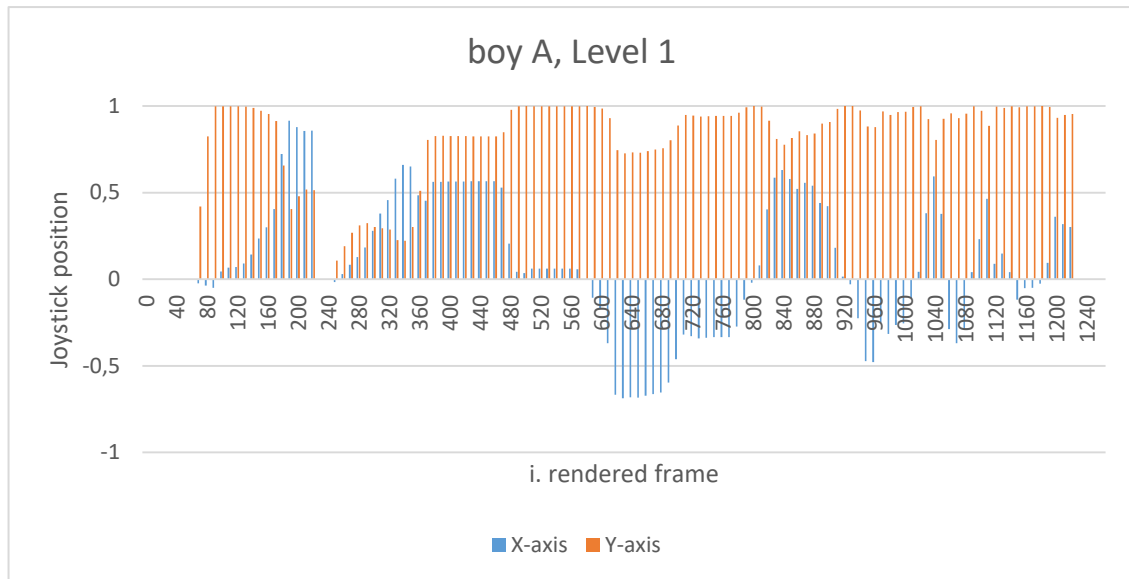
The subjects of the test are two eight-year-old boys. One of them is healthy (boy A) and the other has an acquired, temporal fine motor impairment (boy B), which happened to him at the end of the ski season. The type of injury of boy B is a closed phalangeal fracture, which is the avulsion or rupture of the ulnar collateral ligament of the first metacarpophalangeal joint. Boy B did not require surgery, but it was necessary to wear plaster for four weeks. Physiotherapy was recommended for him once a week for six weeks after removing the plaster. Boy A played the game once, boy B twice, at the start of the physiotherapy and six weeks later at the end of the physiotherapy. [7]

I narrowed the testing down to two levels, according to the difficulty level, to two extremes: one easy and one difficult. The easy one is Level 1, while one of the hardest is Level 8. Based on these, it can be examined whether the difference between the performance of boys A and B can be observed on the basis of the measured data, and if so, this difference can be observed globally even on easily completable levels or only on more difficult levels.

As a reminder to interpret the test results, the joystick coordinates can be between -1 and 1. For the horizontal or X axis, -1 means the joystick is in the fully left position and 1 means it is in the right position. For a vertical or Y axis, -1 means the joystick is fully backward, while for 1 it is forward. If both horizontal and vertical values are displayed at a certain time, the joystick has moved in a diagonal direction. If no value belongs to a certain time, i.e. the value of the axes are equal to zero, the joystick is released.

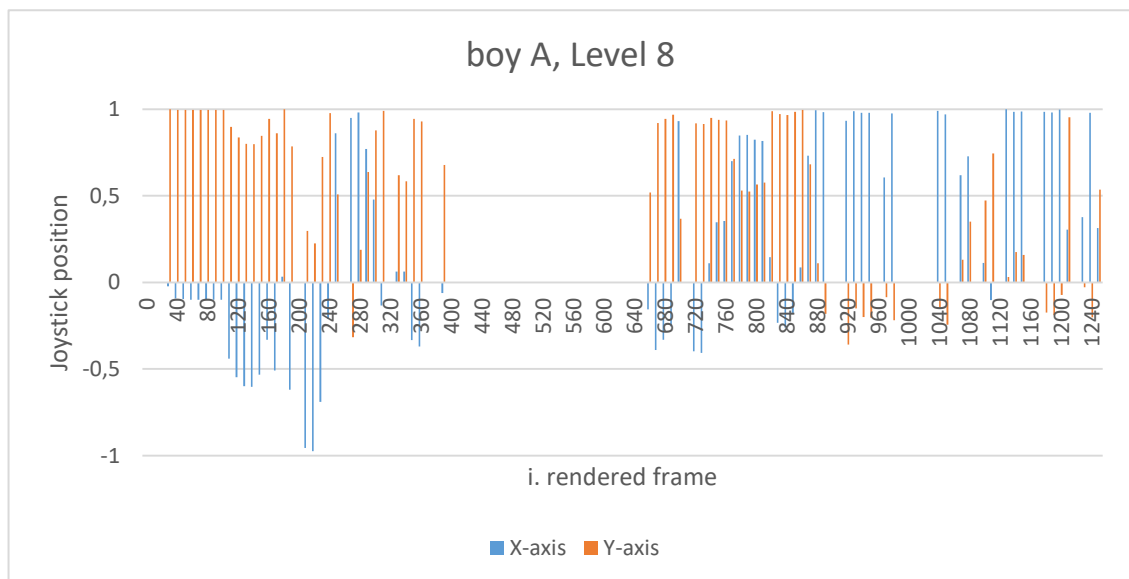
Let's look first at the test results of boy A for the two levels and what observations can be made from it, and then look at the results for boy B before and after the physiotherapy treatment for this same two levels.

The results of boy A for Level 1 are shown in the 5.3 figure. The first thing that can be seen is that this boy completed this level in 1250 rendered frames. This level has no obstacles and is mostly straight except for the S-bend at the beginning of the level. This bend can be well observed at the X coordinates of the joystick at the frames 250-900. Another characteristic feature is that there is no zero coordinate on the graph -except for a short section around frame 230-, which means boy A did not release the joystick during the game.



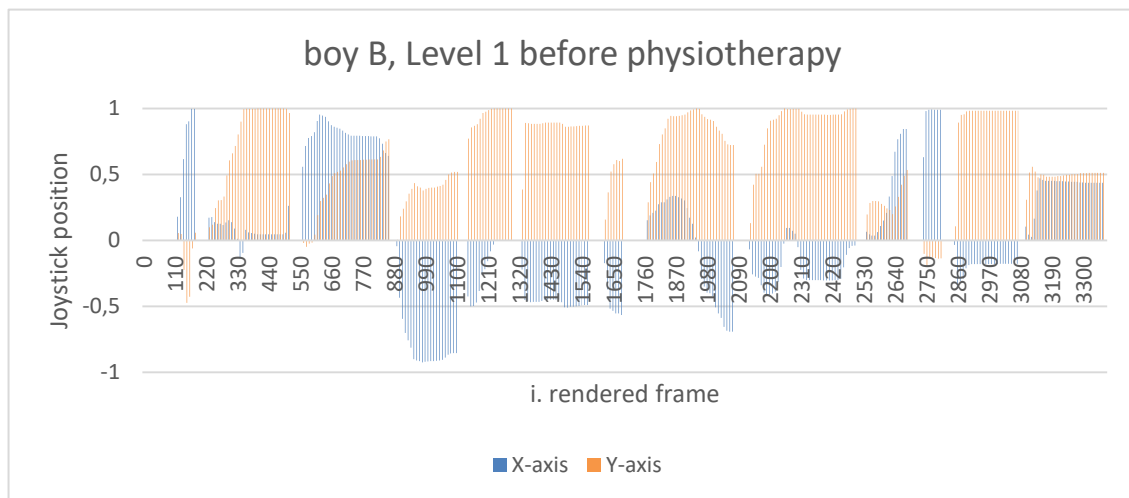
5.3 figure boy A test results for Level 1

The results of boy A for Level 8 are shown in the 5.4 figure. He also completed this level in almost the same time as Level 1. This level has three characteristics: a rotating obstacle at the beginning of the level, an elevator in the middle of the level, and the require of three right sided jumps at the end of the level. From the 50 to 250th frames of the graph, it can be seen that boy A bypassed the rotating obstacle from the left. From frame 400 to frame 650 the player waited for the elevator to arrive while releasing the joystick. The three jumps required at the end of the level can be seen on the graph between 1050 and 1200 frames and it can be also seen that the player released the joystick for a short time between the jumps.



5.4 figure boy A test results for Level 8

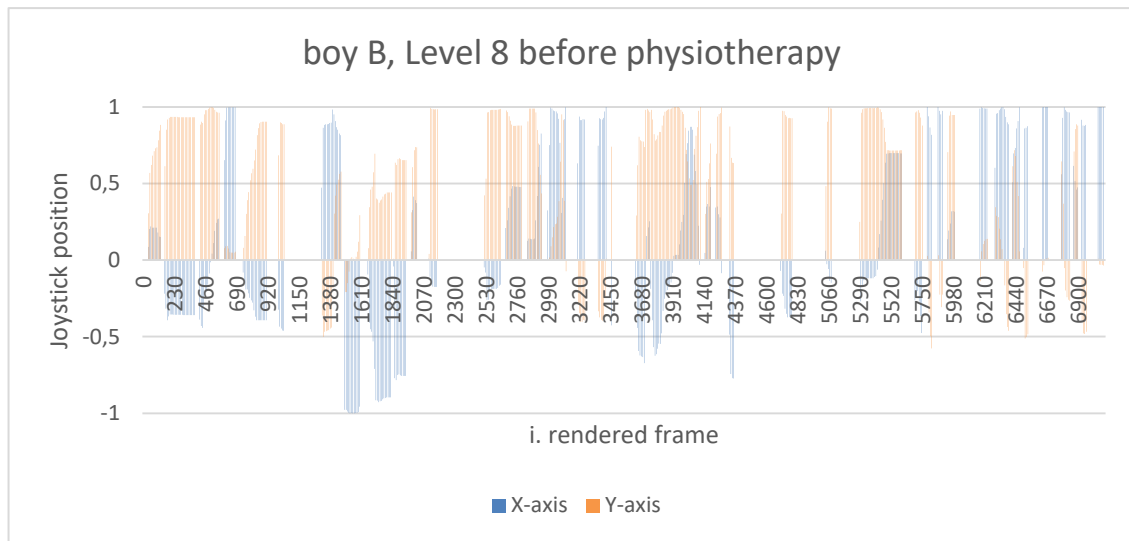
The results of boy B before physiotherapy for Level 1 are shown in the 5.5 figure. The first conspicuous data that can be read from the chart is that boy B completed the first level in more than 3330 frames, which means he needed three times more time to reach the finish line than boy A. Another characteristic that can be observed in the case of boy B is that he released the joystick several times for a shorter time periods as opposed to boy A. The S-bend on this level can also be observable on the chart, which is covered by the 550-2800 frames. From this it can be seen that the S-bend was the biggest challenge for boy B on this level, as he spent the most time with it.



5.5 figure boy B test results for Level 1 before physiotherapy

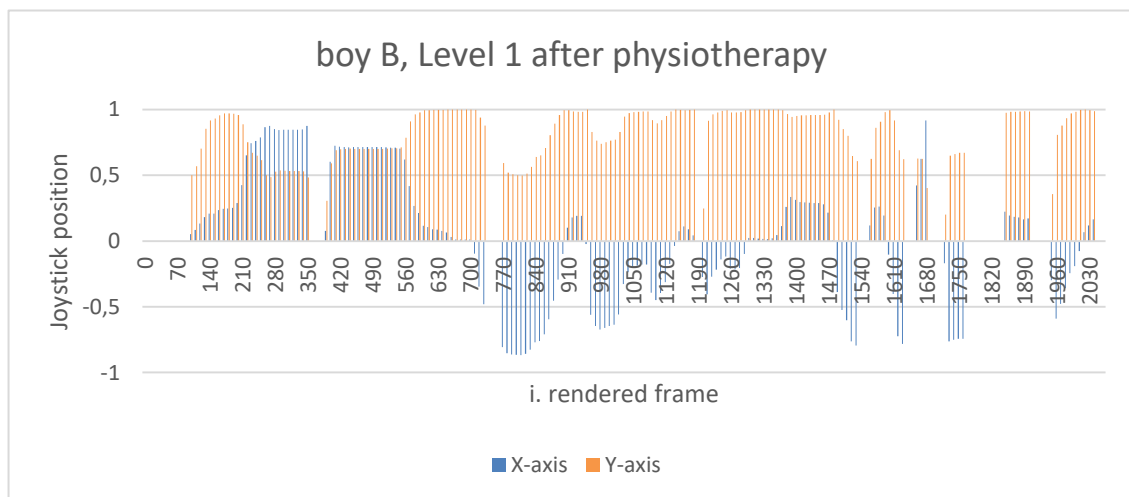
The results of boy B before physiotherapy for Level 8 are shown in the 5.6 figure. Boy B completed this level in more than 7030 frames, which means he took nearly six times more time than Boy A. It can be seen that B boy released the joystick several times on this level as well. From the X coordinates of 130-520 frames, it can be seen that boy B also bypassed the rotating object from the left. The graph shows two longer pauses for frames 1080-1320 and 2160-2520. Presumably, these longer breaks mean waiting for the elevator. It could have happened that the player had just missed the elevator when he got to the waiting area, while the rotating object was approaching him, which necessitated a small position correction, and then waited for the elevator to come again. Frames 3000-3480 show the right jumps required at the end of the level, but between 3480-3600 there is a relatively longer pause and then a section up to frame 5280 is shown, which is typical of the activity at the beginning of the level. The explanation for this that the player fell off the ground, so he was automatically returned to the start position, so he had to start over the level, which can be seen that he did it in a similar way as the first time. It can be seen that for the second time he has managed to complete this level, as there is no

further stage on the chart that is typical of the beginning of the level.



5.6 figure boy B test results for Level 8 before physiotherapy

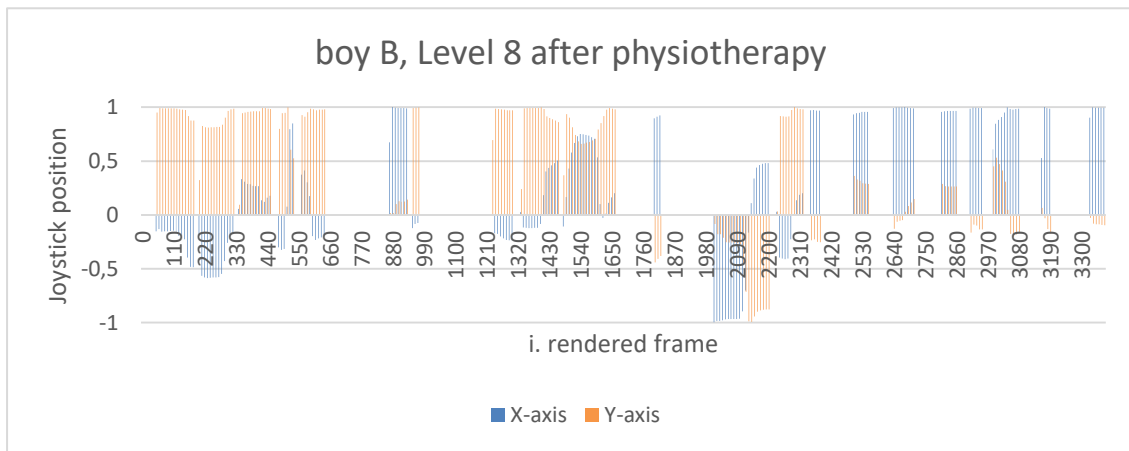
The results of boy B after physiotherapy for Level 1 are shown in the 5.7 figure. There are large differences in the results before and after physiotherapy treatment. It can be seen that boy B completed this level in more than 3330 frames before treatment and in 2060 frames after treatment. This is a significant improvement over himself, although he still lags behind boy A. Another big difference is that he released the joystick much less often after the treatment. It can be clearly seen in the chart that the S-bend was now overcome much more continuously by the player after the treatment, instead of the fragmented, intermittent movement that he did earlier.



5.7 figure boy B test results for Level 1 after physiotherapy

The results of boy B after physiotherapy for Level 8 are shown in the 5.8 figure. This time, the player completed Level 8 in 3380 frames, which was previously 7030 frames. His results show a significant improvement over himself, although he still lags

behind boy A on this level as well. It can also be observed on this level that after the physiotherapy treatment he released the joystick less times, he used his finger much more continuously and dynamically. The longer release of the joystick can be linked to waiting for the lift and jumps at the end of the level, besides these he has only released the joystick for only a few frames, such as around the 480th frame. It can be seen that there is no repetition of a section that is typical of the beginning of the level, so this time the player did not fall off the ground. Furthermore, it can be seen that a very similar diagram belongs to the A and B boys for Level 8, if we look at the results of boy B after the physiotherapy treatment, the difference is mostly in the number of frames, ie in playing time.



5.8 figure boy B test results for Level 8 after physiotherapy

The functional test shows how much information and observation can be extracted only from the joystick position. I note that based on the system specification, the game saves not only this data, but also the elapsed time, the number of jumps that could be used for further analysis, not to mention how much other information could be extracted during the game. However, the tests presented here show that only by using the coordinates associated with the joystick can be used to distinguish between a healthy player and a player with impaired fine motor skills whose decreased ability affects the fingers the most, and even the improvement of the player with impaired fine motor skills can be tracked during rehabilitation. Moreover, in addition to tracking the player's improvement, the game can even complement the rehabilitation process itself in a playful way. For a child today, it is much more appealing through such a game like Cubion to perform the exercises needed for rehabilitation, although it is not yet a substitute for traditional methods. Based on these, I consider the functional test successful.

6. Summary

6.1 Conclusion

In this project, I reviewed what is characteristic of people with impaired fine motor skills specially whose decreased ability affects the fingers the most, how digitalization affects rehabilitation methods, and what modern diagnostic methods are available. Based on these, I have developed the functions and elements needed for a game that can help people with impaired fine motor skills whose problem affects the fingers the most, and a method that captures the movement of the fingers during the game. I designed the GUI system and appearance of the game with child-friendly principles in mind.

Based on the plans, I implemented the game using the Unity engine. I documented the main moments of game development in detail, both from a graphical and scripting point of view. The original game concept changed during development due to the emergence of a new demand for forwarding statistical files via email. Of course, this demand has also been implemented. In the implementation chapter, I tried to present common problems and a possible method of implementation that come up during the development of almost every game.

I did not have the opportunity to test the game on a person with innate, permanent impaired fine motor skills therefore, testing the unhealthy side was represented by an eight-year-old boy who had a thumb fracture. This did not affect the test method or the quality of the extracted data. I have shown that a graph can be created from the data saved during the game, with the help of which the player can be analyzed and compared with his previous data. It can also be seen from the tests that the sensitivity of the extracted data is high, so even very small movements were recorded, so the game can be used for diagnostic purposes as well regardless of the disease, as long as it is focused on the fingers. The game is easy to use, provides enough challenge and interest, so the game can be used for diagnostic and rehabilitation purposes from pre-school age onwards.

I have fulfilled the goals set for the game, the game is suitable for the tasks for which it was made. This project was a next milestone for me, during which I also learned new things, gained a higher level of routine in game development. The success of the project is also evidenced by the successful qualification for the 2022 GDWC Game Development World Championship.

6.2 Options for further updates of the game

I see three areas where the game could be improved:

- Increase the number of levels
- Expand the type of data to extract
- New menu design

More levels are needed to get even more level layouts and obstacles into the game, which can help highlight more specialized fine motorskills problems, more levels provide more data. In addition from a player's perspective, it's important to maintain the interest and give the player new challenges.

There are still untapped opportunities for extracted data. All smartphones have a built-in gyroscope that I think would be worth using because they would allow the test to be extended from the fingers to even the entire hand, giving us information on how much the player has used his hand, specifically his wrist.

Because of the younger age group, I believe that supporting the menu system with icons or even replacing the text in its entirety with icons that indicate the feature would have a beneficial effect on the game.

I consider it necessary to continue the functional test. It should be examined, for example, whether the intermittent release of the joystick is typical of those players who have impaired fine motor skills, examine whether joystick handling is age-related. Gyroscopic data should also be included in the studies, examining whether this provides sufficient additional information for diagnostics, if so, how it can be used in addition to the joystick coordinates.

Finally, I think that user feedback is very important for all software, but especially for games, and that it have to be taken into account in future developments. There are many negative examples in the industry of when developers didn't heed the request of players and that's why their game failed.

Bibliography

- [1] "Company called REHABILITY" [Online]. Available: <http://www.rehability.me/#gallery>. [Accessed 04 22 2022].
- [2] „Android operating system” [Online]. Available: <https://www.javatpoint.com/android-operating-system>. [Accessed 04 22 2022].
- [3] „Android versions” [Online]. Available: <https://socialcompare.com/en/comparison/android-versions-comparison>. [Accessed 04 22 2022].
- [4] „Unity’s minimum system requirements for Android ” [Online]. Available: <https://docs.unity3d.com/2020.1/Documentation/Manual/system-requirements.html>. [Accessed 04 22 2022].
- [5] „History of Unity” [Online]. Available: <https://educationecosystem.com/guides/game-development/unity/history>. [Accessed 04 22 2022].
- [6] „Unity plans” [Online]. Available: <https://store.unity.com/compare-plans>. [Accessed 04 22 2022].
- [7] „Closed phalangeal fracture” [Online]. Available: <https://radiopaedia.org/articles/fractures-of-the-thumb>. [Accessed 04 22 2022].
- [8] „Prevalence of Autism” [Online]. Available: <https://autismcenter.org/prevalence-autism-increases-10-1-54-children>. [Accessed 04 22 2022].
- [9] „First signs of Autism” [Online]. Available: http://medicalonline.hu/gyogyitas/cikk/az_autizmus_spektrum_zavar_korai_tunetei. [Accessed 04 22 2022].
- [10] „JTHFT test” [Online]. Available: https://www.physio-pedia.com/Jebsen-Taylor_Hand_Function_Test. [Accessed 04 22 2022].
- [11] „JTHFT test in use” [Online]. Available: <https://www.euromedical.co.uk/jebsen-taylor-hand-function-test/p/332>. [Accessed 04 22 2022].

-
- [12] „Nine-hole Peg test” [Online]. Available: https://www.physio-pedia.com/Nine-Hole_Peg_Test. [Accessed 04 22 2022].
- [13] „Nine-hole Peg test in use” [Online]. Available: <https://www.devinemedical.com/A8515-9-Hole-Peg-Test-Kit-p/a8515.htm>. [Accessed 04 22 2022].
- [14] „JTHFT research” [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1388245717308969>. [Accessed 04 22 2022].
- [15] „Commercial Haptic Controller” [Online]. Available: <https://uploadvr.com/tactical-haptics-reactive-grip-vr-controllers-are-now-production-ready/>. [Accessed 04 22 2022].
- [16] „Haptic Gripper and Autist children research” [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3231938>. [Accessed 04 22 2022].
- [17] „Leap Motion research with 6-10 years old Autist children” [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-77712-2_98. [Accessed 04 22 2022].
- [18] „Leap Motion technology” [Online]. Available: <https://devpost.com/software/universal-drone-controller>. [Accessed 04 22 2022].
- [19] „Fine motor skill learning in Dentistry” [Online]. Available: <https://www.hindawi.com/journals/ijd/2021/6674213/>. [Accessed 04 22 2022].

List of Figures

1.4 figure Growing tendency of Autism [8]	6
1.5 image JTHFT test in practice [11]	7
1.6 image Nine-Hole Peg Test[13]	8
1.7 image Commercial haptic-grip for VR games[15]	9
1.8 image Leap Motion Device, the pendrive sized object in front of the laptop[18]....	10
2.2 diagram Main Menu	15
2.3 diagram Back	16
2.4 diagram Play	16
2.5 diagram Level selector	17
2.6 diagram Upcoming contents	17
2.7 diagram Rewards	18
2.8 diagram Player controller	19
2.9 diagram Level completed	19
2.10 diagram Saving Level	20
2.11 diagram Evaluating a player's performance	21
2.12 diagram Async loading method	22
2.13 diagram Game object manipulation	22
2.14 diagram Player position reset	23
2.15 diagram From launch to level completed	24
2.16 figure Main Menu conceptual design	25
2.17 figure Player UI conceptual design	26
2.18 figure Rewards conceptual design	26
2.19 figure Upcoming contents conceptual design	27
2.20 figure Loading screen conceptual design	28
2.21 figure Level select menu conceptual design	28
2.22 figure End of level conceptual design	29
4.1 picture Assets and Scenes are organized	39
4.2 picture A detail of the sound mixer	40
4.3 picture Rigidbody settings for the character	41
4.4 picture Scene view of Level 1	41
4.5 picture Implemented primary GUI design	44

4.6 picture Implemented secondary GUI design	45
4.7 picture Implemented Main Menu supplemented with the new demand.....	46
4.8 picture Implemented Loading screen	46
4.9 picture Calling order of MonoBehaviour class functions.....	47
5.2 diagram In-game CPU usage comparsion	58
5.3 figure boy A test results for Level 1	60
5.4 figure boy A test results for Level 8	60
5.5 figure boy B test results for Level 1 before physiotherapy	61
5.6 figure boy B test results for Level 8 before physiotherapy	62
5.7 figure boy B test results for Level 1 after physiotherapy	62
5.8 figure boy B test results for Level 8 after physiotherapy	63

List of Tables

Table 1.1. Effects of digitalization on our life.....	1
Table 1.2. Popular games. A: Android, I: iOS	2
Table 1.3. Latest Android versions.....	5
Table 2.1. Key functions	12
Table 3.1. Major releases of Unity	34
Table 5.1. Comparison of devices	57