FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor's Thesis

Graphical User Interface for Rehabilitation Software

Lukáš Varga

PILSEN, CZECH REPUBLIC

2023

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor's Thesis

Graphical User Interface for Rehabilitation Software

Lukáš Varga

Thesis advisor
Doc. Ing. Libor Váša, Ph.D.

PILSEN, CZECH REPUBLIC 2023

Citation in the bibliography/reference list:
VARGA, Lukáš. *Graphical User Interface for Rehabilitation Software*. Pilsen, Czech Republic, 2023. Bachelor's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor Doc. Ing. Libor Váša, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš VARGA**
Osobní číslo:	**A19B0219P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Grafické uživatelské rozhraní pro rehabilitační software**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s softwarem pro rehabilitaci pacientů trpících roztroušenou sklerózou v prostředí virtuální reality vyvíjeným na KIV ZČU, zejména s mechanismem konfigurace terapeutických cvičení.
2. Navrhněte subsystém pro konfiguraci terapeutických cvičení prostřednictvím grafického uživatelského rozhraní. Dbejte na obecnost, návrh systému připravte na možné změny struktury parametrů cvičení používaných v terapeutickém softwaru. Konfigurační subsystém konstruujte tak, aby vytvářel a načítal konfigurační soubory ve struktuře XML navržené v současné verzi rehabilitačního softwaru. Případné nutné změny konzultujte s vývojovým týmem.
3. Navržené řešení implementujte a důkladně otestujte. Předložte řešení terapeutickému týmu a získejte zpětnou vazbu k funkčnosti a uživatelskému komfortu. Požadované změny zapracujte.
4. Vytvořené řešení důkladně zdokumentujte a popište možné směry pro budoucí vývoj.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Doc. Ing. Libor Váša, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Declaration

I hereby declare that this Bachelor's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

Pilsen, on 23 April 2023

. . . . . .

Lukáš Varga

Abstract

This bachelor's thesis deals with the extension of the arm rehabilitation application in virtual reality. The application is being developed as part of the study *Virtual reality in the treatment of patients with multiple sclerosis*. In the current version of the software, therapy exercises are set up manually by editing configuration files. The work aims to simplify the configuration of therapeutic exercises for the physiotherapy team using an intuitive graphical user interface. The focus is on backward compatibility with the original configuration mechanism and on possible changes to the structure of exercise parameters used in the therapy software.

Abstrakt

Tato bakalářská práce se zabývá rozšířením aplikace pro rehabilitaci paže ve virtuální realitě. Aplikace je vyvíjena v rámci studie *Virtuální realita v léčbě nemocných s roztroušenou sklerózou*. V aktuální verzi softwaru se terapeutická cvičení nastavují ručně pomocí editace konfiguračních souborů. Práce si klade za cíl zjednodušit fyzioterapeutickému týmu konfiguraci terapeutických cvičení za použití přívětivého grafického uživatelského rozhraní. Důraz je kladen na zpětnou kompatibilitu s původním mechanismem konfigurace a na možné změny struktury parametrů cvičení používaných v terapeutickém softwaru.

Keywords

VR rehabilitation • XML • Configuration • Multiple sclerosis • Unity • GUI

Acknowledgement

I would like to thank Doc. Ing. Libor Váša, Ph.D. for his guidance and helpful attitude during the writing of this thesis. My thanks also go to Ing. Jakub Frank for his assistance with software implementation and Bc. Lubomír Rodina and his therapeutic team for the feedback, which led to the improvement of the application. Last but not least, I would like to express my gratitude to my family and friends for their support, particularly to Duc Long Hoang for his encouragement and assistance with the LaTeX template.

Contents

1 Introduction **3**

2 Theoretical Part **5**

2.1 Multiple Sclerosis 5
2.2 Rehabilitation 6
2.2.1 Rehabilitation in MS 6
2.3 Virtual Reality 6
2.3.1 Physiotherapy using VR 7
2.4 Exercises 8
2.4.1 Movement 8
2.4.2 Game 9
2.5 Application Configuration 10
2.5.1 Data Serialization 10
2.5.2 Exercise Definition 11
2.5.3 Exercise Configuration 11
2.6 User Interface 14
2.7 Unity 14
2.7.1 User Interface Toolkit 15
2.7.2 Choosing the UI Toolkit 17

3 Implementation Part **19**

3.1 Integration to the Rehabilitation SW 19
3.2 Proposed Code Structure 19
3.2.1 Controller for Editing Configuration 20
3.2.2 Helper Functions 22
3.2.3 XML Handling 23
3.2.4 Class Diagram 24
3.3 Exercise Configurator Window 25
3.3.1 Global Settings 25
3.3.2 Exercise Definition 26
3.3.3 Exercise 27

3.3.4 Final Design 29
3.4 Additional Functionality 30
3.4.1 Floating Tooltip 30
3.4.2 Blinking Effect 30
3.4.3 Unity Layout 31

4 Testing **33**
4.1 User Manual 33
4.2 Test Scenario 33
4.3 Feedback 34
4.4 Incorporation of Improvements 35

5 Conclusion **39**

A Test Case Document **41**

B Feedback Document **43**

Bibliography **45**

List of Figures **49**

List of Listings **51**

Attachments **53**

Introduction 1

Patients with multiple sclerosis use rehabilitation to slow down the manifestations of the disease in combination with other treatments, such as medication. Thus, rehabilitation is considered to be one of the essential methods to enhance the overall quality of life for patients.

A team from the Third Faculty of Medicine at Charles University and the Faculty of Applied Sciences at the University of West Bohemia is developing software (SW) that aims to move treatment into virtual reality (VR). The application is being developed as part of the study — *Virtual reality in the treatment of patients with multiple sclerosis.* It aims to increase the motivation of patients to attend therapy.

This alternative method of rehabilitation still requires the involvement of a physiotherapist, at least for now. Patients visit a specialized therapeutic facility, where they perform a series of exercises in the presence of the therapist. The symptoms of the disease can vary depending on the individual. Therefore, the therapist creates a specific set of exercises that corresponds to the individual needs of the patient.

At the moment, the whole therapeutic plan is set up manually by editing configuration files. These files use the Extensible Markup Language (XML) format to describe exercises and their parameters. Editing configuration files, therefore, requires the therapist to have some technical knowledge and familiarity with the given format.

The goal of this work is to create a convenient graphical user interface (GUI) that will simplify the configuration of therapeutic exercises for the physiotherapy team. This would mean that therapists would not have to deal with unnecessary technicalities and could focus more on patients and the therapy itself. Ultimately, the implementation of the GUI can allow at least a portion of the therapy to be transferred to the home environment in the future [Fra22].

The GUI module for the exercise configuration will be designed with an emphasis on backward compatibility with the original solution. It will be properly tested by the therapeutic team and the proposed changes will be implemented in the final version.

Theoretical Part 2

2.1 Multiple Sclerosis

Multiple Sclerosis (MS) is a potentially disabling disease of the brain and spinal cord – the central nervous system (CNS). In MS, the body's immune system mistakenly attacks and damages the protective covering of nerve fibres called *myelin*, disrupting the connection between the nerves of the body and the brain. *Myelin* is a layer that forms around nerves, including those in the brain and spinal cord. It is made up of protein substances [Tob22].

This behaviour can result in a wide range of symptoms, such as inflammation of the optic nerve[1], movement disorders, sensitivity, balance disorders or difficulty urinating. The incidence of the disease usually occurs in individuals between the ages of 20 and 40. Some major factors that may contribute to the onset of MS include infection with the *Epstein-Barr* virus, vitamin D deficiency, tobacco smoking, and genetic predisposition. In general, it is not considered a fatal disease. Most patients live a normal life and reach an average age [Sib16].

The onset of the disease is usually not sudden and manifests itself in a variety of clinical manifestations. Nonspecific difficulties described by some patients, such as fatigue, loss of energy or malaise, may occur for several weeks to months before the development of neurological symptoms leading to accurate diagnosis [Fra22].

During the course of the disease, there are alternating phases of *relapse* and *remission*. During *relapses* (attacks), the patient's condition and neurological difficulties worsen. On the other hand, during *remissions*, there is a partial restoration of *myelin* and the disease does not manifest itself in any way [Rod20]. The basis of treatment is corticosteroids[2] and anti-inflammatory drugs during the *relapse* phase, combined with physiotherapy and rehabilitation.

[1] Blurred vision, pain when moving the eye, change in colour perception
[2] Synthetic drugs that mimic the effects of naturally occurring steroid hormones

2.2 Rehabilitation

Rehabilitation is a type of medical treatment that focuses on restoring physical and cognitive imbalance. The reason for this imbalance could be injury, illness, or disability. The initial phase is crucial in determining the precise diagnosis of the patient and identifying the exact procedures and priorities of treatment. Several tests are employed in rehabilitation to confirm the effectiveness of the therapy.

The goal of rehabilitation is to help individuals to achieve their maximum potential and to prevent further complications or disabilities. The process often requires a coordinated effort between healthcare professionals and patients. It may involve long-term care and ongoing support to ensure the best possible outcomes.

The text in this chapter is based on [Fra20].

2.2.1 Rehabilitation in MS

When rehabilitating patients diagnosed with multiple sclerosis, it is necessary to approach each of them individually. MS can have a variety of symptoms and its manifestations vary within the stage of the disease. In the phase of *remissions*[3], the patient is supposed to visit therapy at least once a year to learn therapeutic exercises. During the phase of *relapses*[4] it is suggested to attend the therapy more often in order to prevent complications.

During rehabilitation, methods based on neurophysiology are primarily used. These methods are built on the neuroplasticity[5] of the CNS. As a result, the damaged part of the brain can be at least partially replaced by an undamaged part in terms of functioning [Her21].

The therapy initially occurs with a therapist and then gradually transitions into an individualized program. The physiotherapy is based on elements of sensorimotor learning[6] and adaptation. Therefore, patients try to understand the possibilities of using their own body by simulating everyday activities, for instance, sitting, standing, standing up, sitting down, stepping and walking [Fra20].

2.3 Virtual Reality

Virtual reality aims to imitate some visual environment using a computer-generated model. In order to display the model, it is necessary to have hardware with sufficient performance to render the resulting image. A headset[7] has an integrated display, on

[3] An improvement of the patient's condition
[4] A worsening of the patient's condition
[5] The brain's ability to adapt to various changes in the internal and external environment
[6] A process of acquiring and improving motor skills through sensory perceptions
[7] A device worn over the head, commonly used for communication and entertainment purposes

which the generated image is displayed. The headset usually contains some technology for recognizing the position in space and then generating the corresponding scenery. An example of such a headset is shown in Figure 2.1.

Figure 2.1: HTC Vive Headset

The VR headset is typically paired with controllers that allow the user to simulate hand movements in a virtual environment. Other special accessories may include various sensors that can be attached to the body. These sensors monitor specific parts of the human body and hence enable a more accurate simulation in a virtual environment.

2.3.1 Physiotherapy using VR

During the study, the team of students primarily uses headsets from the *Vive* series, which were created by companies *Valve* and *HTC*. The device works on the *SteamVR* platform and needs to be connected to a computer that handles image rendering. Movement is tracked by the *SteamVR Tracking* system using body-mounted sensors, so-called trackers. These trackers enable tracking the movement of the entire body or its parts in a virtual reality environment [Fra22].

As mentioned earlier, the headset needs to be connected to the computer so the therapist can see the patient's full view in the desktop version of the rehabilitation SW. In addition, the therapist can configure certain parameters of the SW, edit the

therapeutic plan or see some statistics about the currently performed exercises on the computer screen.

In the VR rehabilitation SW, there are currently several exercises which aim to train the upper body. In the beginning, the patient puts on a headset and trackers are attached to the patient's body. After the initialization[8], the physiotherapist chooses the individual exercise plan using the computer. Then, the VR headset displays the first exercise from the plan and the patient is guided to perform the desired movement[9] till the end. After finishing the first exercise, the next exercise follows, until the patient completes all the exercises from the therapeutic plan.

2.4 Exercises

The VR rehabilitation application includes several predefined exercises which are used in the therapy. It helps patients to visualize the movements and motivates them to perform the desired physical activity. These exercises can be divided into two different categories — *movements* and *games*. The emphasis is on movements because they include basic principles of treating MS using physiotherapy. Games, on the other hand, provide more natural motions. Additionally, games can contribute to motivation by making the therapy more entertaining.

A VR physiotherapy session consists of different exercises which are put together into the individual plan created by a therapist. Each of these movements and games has its own specific parameters that can be adjusted. The order of exercises is also essential because the therapist may want some exercises to be done during the beginning of the therapy when the patient is full of energy, and vice versa.

The text in the following two chapters is based on [Fra22].

2.4.1 Movement

Movement is a type of exercise that involves following a specific hand trajectory in VR. Movements can be divided into two groups — *without return* and *with return*. A movement *without return* does not require the patient to return to the starting position. Such movement is, for instance, a spiral in Figure 2.2. A movement *with return* has the starting and ending point in the same place, and the patient must reach the so-called *apex*[10] during the exercise. An example of such an exercise is a one-handed diagonal in Figure 2.3.

Movements use several parameters which can be adjusted to the individual needs of the patient in the configuration file, as further explained later in Chapter 2.5.3.2.

[8] Configuring the trackers and putting the patient into a neutral starting position
[9] Using VR controllers and his whole body
[10] The highest point or peak of a movement, where direction changes

Figure 2.2: Spiral Movement

Figure 2.3: Diagonal Movement

These parameters are a phase target, a quality threshold and the number of repetitions of the given exercise.

2.4.2 Game

This type of exercise allows patients to play simple games so they can train their motor skills. Currently, there is one game implemented in the VR rehabilitation application. The name of the game is *Ball Connecting* and its objective is to collect balls displayed in virtual space using the patient's hand. Figure 2.4 illustrates the mentioned game.

Figure 2.4: Ball Connecting Game

2.5 Application Configuration

In order to adapt the previously mentioned exercises to the individual needs of patients, some kind of configuration is needed. The configuration is primarily made by physiotherapists with the help of programmers. Within the configuration, therapists must create a suitable combination of movements and games. The emphasis is also on the possibility to adjust the difficulty of exercises. The configuration uses files in XML format that will be described in more detail in the following chapter.

2.5.1 Data Serialization

Data serialization is a process to save an object's state and transfer it to a stream. The data stream created during serialization contains all the information needed to rebuild the object later. This process is widely used for storing persistent data on a disk in various formats, such as XML, YAML or JSON. The process of serialization enables exchanging data between different systems, languages and architectures [06].

In the rehabilitation SW, the configuration data are stored using Extensible Markup Language (XML). This format offers a universal method for representing various types of information. It is not a language specifically designed for data representation. Instead, it provides a highly versatile structure that allows for the creation of new formats to represent diverse sets of data. This language can be used to represent almost any data because it is highly generic. XML has gained significant popularity in the last two decades due to its universality and the fact that even HTML[11], which the entire internet is built on, is related to XML [NL14].

[11] HyperText Markup Language

An example of an XML structure is shown in Listing 2.1. XML documents form a tree structure that can be described as follows. On the first line, there is an obligatory prologue which sets the XML version and character encoding. On the second line, there is a mandatory root element. There can only be one root element in the whole document. The rest of described elements are only a sample to show the structure. The following line starts with a *<child>* element which has two attributes — *par1* and *par2*. Both of these attributes have their own values stated in quotation marks. On lines 4, 5 and 6, there are three elements of *<subchild>*. The final point to note is that each element must have a corresponding closing element[12].

Source code 2.1: Sample XML Code Snippet

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <root>
3     <child par1="val1" par2="val2">
4         <subchild>...</subchild>
5         <subchild>...</subchild>
6         <subchild>...</subchild>
7     </child>
8 </root>
```

2.5.2 Exercise Definition

In the VR rehabilitation SW, each movement or game needs to be defined before it can be used for therapy. Exercise definition can be thought of as a type of *class*. Subsequently, this class can be used to generate separate exercise *instances*. As mentioned in Chapter 2.4, these *instances* can be used to create a therapeutic plan by combining them in the list.

All movement definitions from the application are stored in the file *movement-definitions.xml*. It includes several pieces of information about the movement, such as its ID, the side[13], a filepath to the data and a filepath to the animation. As for the game definitions, they are defined separately. Currently, there is only one game implemented right now — *Ball Connecting*. Similarly, this game uses some parameters, as will be explained later in Chapter 2.5.3.3.

2.5.3 Exercise Configuration

Accordingly, the specific exercise instances can be used to create an individual therapeutic plan, which is specified in an XML file. The default file for the rehabilitation SW is *config.xml*[14], which is loaded during the application start. Because the mecha-

[12] Using the symbol '/' before the element name
[13] Left or right
[14] It is located in the *Assests/StreamingAssets/* directory by default

nism of exercise configuration is crucial for this bachelor's thesis, the configuration file and its structure will now be described in more detail, as can be seen in Listing 2.2.

The text in this section and its subsections is based on the following source [Fra22].

,

Source code 2.2: Therapeutic Configuration File

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <exercises>
3    <lang>english</lang>
4    <hide_hint_arrow>true</hide_hint_arrow>
5    <data_processor>combined</data_processor>
6    <tracker_model>Cube</tracker_model>
7    <hide_cubes_in_tunnel>true</hide_cubes_in_tunnel>
8    <show_tunnel_only_to_apex>false</show_tunnel_only_to_apex>
9    <hide_chair>false</hide_chair>
10   <exercise type="movement" id="diag1_l" hideChest="true"
      hideHead="true" hideHand="true" hideArm="true">
11     <phase_target>0.5</phase_target>
12     <quality_threshold>0.6</quality_threshold>
13     <repetitions>3</repetitions>
14   </exercise>
15   <exercise type="game" id="ball_r">
16     <name>sixballs</name>
17   </exercise>
18 </exercises>
```

The root element *<exercises>* encapsulates all the settings and individual exercises. There are several elements on lines 3-9 which modify the global settings of the whole application. Then, there are two exercises — a game starting on line 10 and a movement starting on line 13. Each exercise has common parameters — *type* and *ID*. *Type* can be either *movement* or *game*. *ID* then specifies the string name of the exercise.

2.5.3.1 Global Settings

Language. It is possible to change the language using this option. All the information in the application will be displayed in this language[15].

Hide Hint Arrow. The arrow in the *Ball Connecting* game can be shown or hidden to the user. This arrow is used for navigation when the user does not see the balls.

[15] Current localization options are *Czech* and *English*

Data Processor. Sets the data processor used. It is a deprecated artefact that was used during the development of earlier versions of the application.

Tracker Model. A model representing the current position of each tracker. *capsule* shows the tracker as an ellipsoid, while *cube* displays it as a cube.[16]

Hide Cubes in Tunnel. The trajectory displays cubes that visualize how the tracker should be rotated at that point. The cubes in the trajectory can be shown or hidden from the user.

Show Tunnel Only to Apex. Specifies how the trajectory will be visualized for movements with an *apex*, as mentioned in Chapter 2.4.1. If the value is selected, only the trajectory from origin to *apex* will be displayed during the entire motion execution. If the value is not set, then it will switch between the two trajectories.

Hide Chair. One of the VR trackers is attached to a chair that is used for the patient to sit down and stand up during the therapy session. For that reason, there is the parameter *Hide Chair* which can be used to show or hide the chair visualization. In the case of a hidden chair, it is not necessary to use the tracker for the chair.

2.5.3.2 Movement Parameters

A movement has a set of special parameters — *hideHead, hideChest, hideArm, hideHand*. These parameters define whether or not to display trajectories of each body part during the exercise by stating either *true* of *false*. The movement also has several sub-elements as listed:

Phase target. The element contains a decimal number from the interval [0, 1] and indicates to which phase the movement should be executed. The phase determines what part of the exercise the patient is in, spanning from the beginning to the end of the movement.

Quality threshold. The element contains a decimal number from the interval [0, 1] and indicates the level of quality that should be achieved for a given phase. The quality provides information about how far the patient is from the sample exercise.

Repetitions. The element specifies how many times the movement should be repeated before it is considered complete. The number belongs to the interval [0, 10] and it is a natural number.

[16] The model is implicitly set to a *cube*

2.5.3.3 Game Parameters

Filename. In the game, there is a limited number of balls and their locations are determined by the therapist using a separate application dedicated to recording the position of the balls. Thus, the game requires a configuration file with the data about balls in XML format[17].

2.6 User Interface

User Interface (UI) facilitates communication between human beings and computers. It includes components for input and output that allows users to interact with electronic devices or SW applications. When designing a UI, the focus is on showing information in a comprehensible way. This can be shown in abstract or representational forms to help people understand structures and processes using tables, forms, charts, maps, and diagrams.

In addition to UI, there is also User Experience (UX). It is a field that involves designing interactive experiences for viewers, customers, visitors or readers. UX design is not limited to graphic, media and web designers, or other related professionals. Instead, it contains all aspects of the human experience and incorporates them into computer-based products. As a result, related topics like customer-experience design, culture-driven design, cognitive-neuroscience-driven design and others have emerged.

Graphical User Interface (GUI) is closely connected to UI. The main difference is that GUI refers more specifically to a system of interactive visual components for controlling the SW, such as menus, windows, dialogue boxes, control panels, icons and tool palettes. An example of such a GUI is the Unity Editor, which is described in the following chapter.

The text of this chapter is based on [Mar15].

2.7 Unity

The *Unity Editor* can be utilized to develop 2D and 3D games, applications, and interactive digital environments, so basically, it could be characterized as an integrated development environment (IDE). An IDE is a piece of computer SW that provides tools and features to make it easier to develop other pieces of SW. *Unity* offers a user-friendly interface, a powerful game engine, a visual editor, and support for multiple programming languages [Uni22c]. The *Unity Editor* is presented in Figure 2.5.

[17] The file location is the *Assests/StreamingAssets/* directory by default

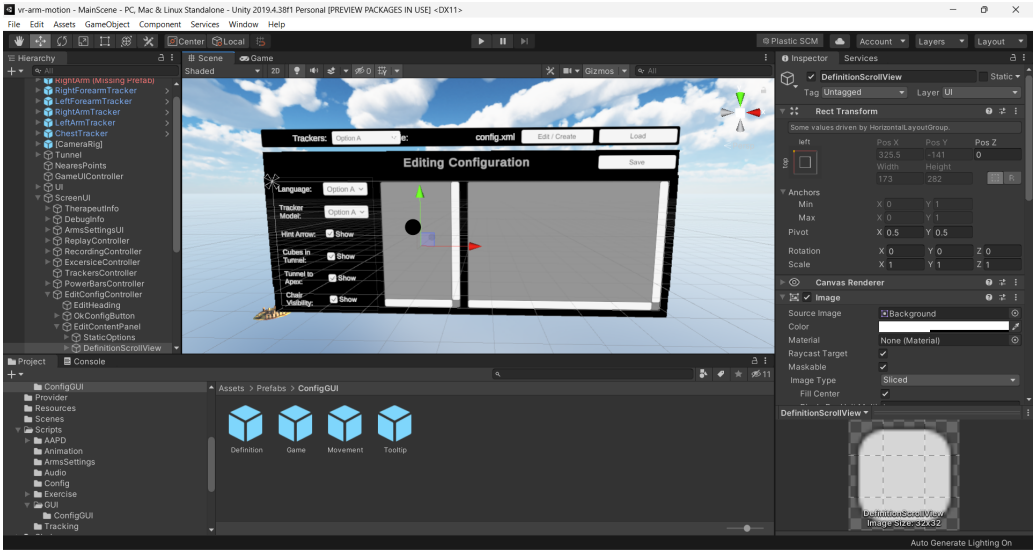

Figure 2.5: Unity Editor

The editor consists of several parts as follows. On the left side, there is the *hierarchy* with all the game objects used in the current application. At the bottom side, there is the *project tree* with all the files and also the *console*. In the middle of the screen, there is the *scene* for visualising game objects from *hierarchy* using the *camera*[18]. When any game object[19] is selected, some of its additional options and preferences will pop up on the right side.

One of the native languages that can be used for scripting in *Unity* is *C#*. It is a modern, object-oriented programming language designed and first introduced in 2000 by *Microsoft*. It is mainly used for developing *Windows* applications, web applications and games. *C#* is quite similar to languages such as *C++* and *Java*, but with some additional features [Mic23].

2.7.1 User Interface Toolkit

A very important point of the design was the selection of a suitable library for the creation of the UI. Within the *Unity* version in use[20], there are three different libraries, it can be chosen from — *UIElements*, *Unity UI* and *IMGUI*.

UIElements. User Interface Elements (*UIElements*) is a UI toolkit for developing user interfaces in the Unity Editor. The toolkit is based on recognized web technologies and supports stylesheets, dynamic and contextual event handling, and data

[18] A tool through which the player views the world
[19] Either from the *scene* or from the *hierarchy*
[20] Unity 2019.4 LTS

persistence. It is a UI toolkit that is no longer maintained by *Unity* and it is gradually being replaced by its descendant — *UI Toolkit*[21]. *UIElements* allows defining elements using a tree structure familiar to web technologies based on XML and CSS, as it is displayed in Figure 2.6.

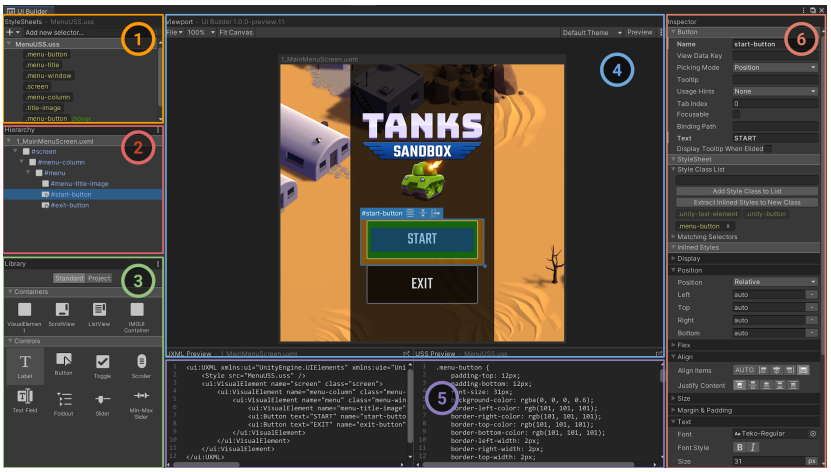

Figure 2.6: UIElements Example [Uni20b]

Unity UI. The Unity User Interface (*Unity UI*) package provides a simple UI toolkit for developing user interfaces for games and applications. *Unity UI* is a *GameObject*-based UI system that uses components to arrange, position, and style the user interface, as can be seen in Figure 2.7. This toolkit has the advantage of a mainly graphical approach. So far, it has been used for the development of graphic elements of VR rehabilitation SW. *Unity UI* is integrated into the Unity development environment, which is a great advantage among the others.

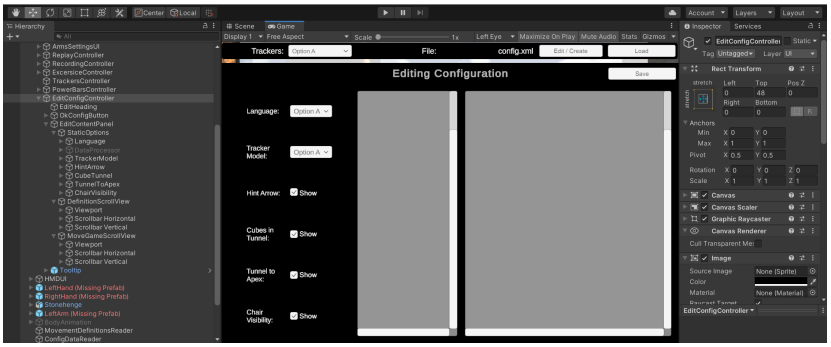

Figure 2.7: Unity UI Example

[21] Not presented the in Unity version of the VR rehabilitation SW

IMGUI. Immediate Mode Graphical User Interface (*IMGUI*) is a code-driven UI toolkit that is mainly intended as a tool for developers. *IMGUI* uses the *OnGUI* function (and scripts that implement the *OnGUI* function) to draw and manage its user interface. *IMGUI* is the most advanced tool, but also the most complex. Its big disadvantage is the aforementioned complexity, so Unity itself does not recommend it for building the UI of a game or an application. An example of IMGUI can be seen in Listing 2.3, where I used it for the blinking effect described later in Chapter 3.4.2.

Source code 2.3: IMGUI Example

```
public class ImageBlinkEffect : MonoBehaviour
{
    ...
    void Start()
    {
        imgComp = GetComponent<Image>();
        StartCoroutine(SelfDestruct());
    }

    void Update()
    {
        imgComp.color = Color.Lerp(startColor, endColor,
    Mathf.PingPong(Time.time * speed, 1));
    }

    private void OnDestroy()
    {
        imgComp.color = startColor;
    }
    ...
}

```

The text in this chapter is based on [Uni22a].

2.7.2 Choosing the UI Toolkit

Of the options listed, I chose to use mostly the *Unity UI* package. It seemed logical to build on the similar structure which was used during the previous development of the VR rehabilitation application. This toolkit contained sufficient tools from which I could model a graphical editor for configuring exercises.

In addition to that, I also used the last mentioned *IMGUI* package for a couple of advanced functionalities. Specifically, I used it for two interactive visual effects — the blinking effect and the floating tooltip[22].

[22] A graphical element that provides some information about a given item when the user hovers the mouse over it

Implementation Part 3

The entire GUI module was designed and implemented in the Unity development environment. More specifically, Unity version *2019.4 LTS* was used, since the whole VR rehabilitation application is based on this particular version. The object-oriented language *C#* was used as the programming language for scripting in *Unity*. The target platform of the application was *Windows* operating system.

3.1 Integration to the Rehabilitation SW

Before implementing the GUI module, I needed to examine the existing application structure so I could connect the module in a reasonable way. The most suitable place for linking was the class *ExercisesController*. This class handles the loading of *exercises* from a configuration file into the application environment. It already contains a list of *exercises* and *exercise definitions*, and a reference to the configuration file.

In the desktop version of the rehabilitation SW, the *ExercisesController* class is responsible for displaying the top panel, where therapists select the configuration file. A new button was added to the top right corner of the mentioned panel, as shown in Figure 3.1 (highlighted in yellow). It has two different functions — editing an existing configuration or creating a new configuration file.

3.2 Proposed Code Structure

In the following chapters, I am going to describe the proposed code structure of the GUI module for configuring exercises. In order to maintain code readability for other programmers, it is generally considered best practice to divide the code into multiple files and classes. Moreover, I personally consider this division to be crucial because it helped me to come up with better ideas when developing the application and it enables me to connect individual logical and functional blocks together. The code was divided into three main classes as follows — *EditConfigController*, *Helper* and *XmlHandle*.

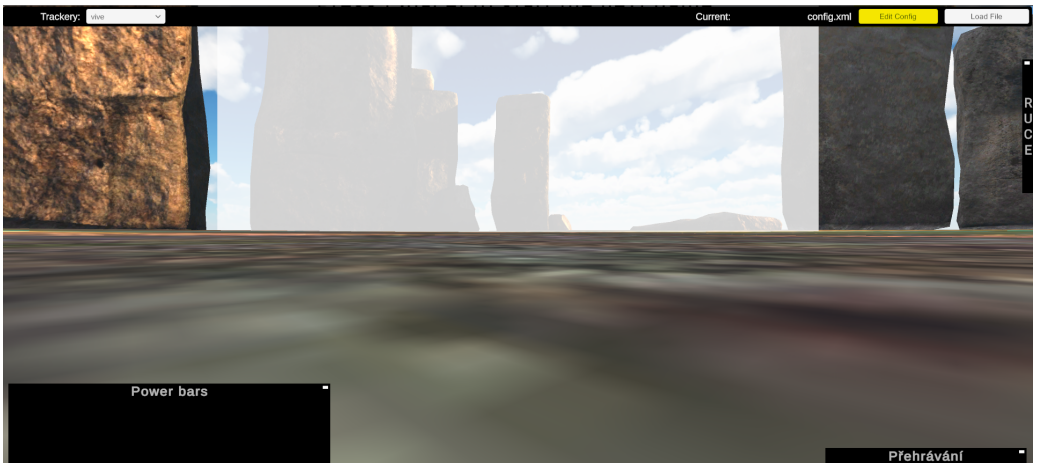

Figure 3.1: Exercise Controller Window

3.2.1 Controller for Editing Configuration

The class *EditConfigController* is a Unity class that inherits from MonoBehaviour[1]. In addition, I created a main *GameObject*[2] with an identical name to the class in the Unity Editor. This *GameObject* contains all the references to graphical elements used in the Unity scene, such as labels, buttons, dropdowns and toggles. *EditConfigController* class serves as a bridge between visual elements in Unity and their controlling scripts. That made it easier for me to implement my own functionality which was not offered by the editor.

The class *EditConfigController* is represented by a window that is displayed when a user wants to modify certain settings in the configuration file. Once the user clicks the editing button in Figure 3.1, the method *StartEditConfig()* is called[3]. The method's parameter is the list of *Exercise* elements which were loaded from the XML file to the application during the start[4]. This method serves as the top-level trigger for displaying all other graphical elements. The method can be imagined as a tree — the visual elements are displayed from the root through the branches to the last smallest leaves, as presented in Listing 3.1.

Source code 3.1: Creating the Configurator Window

```
1 public void StartEditConfig(List<Exercise> exercises)
2 {
3     Init();                    // Initializes the window
4     FillSettingLabels();       // Labels for settings
5     FillSettingInputs();       // Switches for settings
```

[1]The base class from which every Unity script derives [Uni22d]
[2]Base class for all entities in Unity Scenes [Uni22b]
[3]If the user creates a new config file instead, in addition, the method *CreateNewConfig()* is called
[4]The list of exercises is already present in the class *ExercisesController*

```
6       FillDefinitions();              // Draws definitions
7       FillExercises(_exercises);      // Draws exercises
8       SwitchVisibility();             // Shows the window
9   }
```

In the beginning, the method *StartEditConfig()* draws UI elements of the *global settings* and assigns them the data from the configuration file, as mentioned in Chapter 2.5.3.1. There is a group of toggles and dropdowns that can change several application settings. Afterwards, the method loads all the possible *exercise definitions* from the rehabilitation SW. The definitions are displayed in the Unity element called *Scroll View*[5], which will be introduced shortly. At last, the method draws another *Scroll View*. This one loads the *exercises* from the chosen XML configuration file. This includes *movements* and *games* with their parameters used in the therapeutic plan. The configuration window is created only once on the application start, but it can be hidden or shown to the user. Internally, it is implemented using the method *SwitchVisibility()* which uses Unity method *SetActive()* for hiding *GameObjects* from the scene.

It is also worth mentioning the methods responsible for closing the window. In the case of saving exercises and closing the configuration window, the method *SaveConfig()* is called. This method hides the window using the mentioned method *SwitchVisibility()*, updates the selected configuration file and clears *Scroll Views*. In the end, the updated configuration file is loaded into the application so the therapist can use the therapeutic plan immediately.

Detailed information about some other methods in this class will be described later in Chapter 3.3 because these methods are closely connected to individual graphic elements in terms of visualisation and behaviour. These methods are also responsible for creating additional visual effects, which will be explained further in Chapter 3.4.

3.2.1.1 Scroll View

During the development of the exercise configurator, I searched for a suitable Unity UI element which would be able to contain a varying number of items, in my case, it was *definitions* and *exercises*. In other words, I needed a graphic container enabling dynamic collection. After some searching, I have finally discovered the tool called *Scroll View* which fits the definition. It can simply display its contents inside a scrollable frame. A *Scroll View* is presented in Figure 3.2.

A *Scroll Rect* is a Unity *Component*[6] that is able present a lot of content in a small area and is often combined with other elements to create a *Scroll View*. Its essential

[5] A UI component for scrolling through large amounts of content within a limited screen space
[6] A modular piece of functionality attached to a *GameObject* to add behaviours and properties

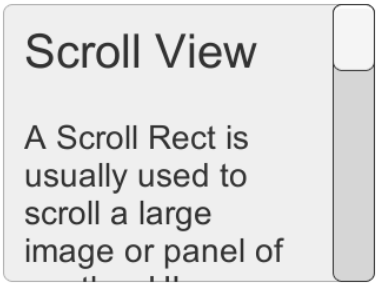

Figure 3.2: Unity Scroll View [Uni20a]

elements are the *Viewport*, the *Scrolling Content*, and vertical or horizontal *Scrollbars*. All scrolling content should be within a single content *GameObject*. The *Scrollbars* can hide automatically if the content doesn't need to be scrolled. Then, the *Viewport* can be automatically expanded in order to use more space [Uni20a].

3.2.2 Helper Functions

Unlike the previous class mentioned, the class *Helper* is not marked as a Unity class. It is a standard *C#* class which extends the functionality of *EditConfigController*. Essentially, it handles the programming logic for a wide range of elements in the GUI. Furthermore, it includes the internal code handling of methods in *EditConfigController* for the purpose of making the code more clear and more consistent.

The class is built using a software design pattern called *singleton*. The *singleton* is a class that has only one instance and provides a way to access a single global resource. To create and maintain a single instance, a class method is usually used, similar to a factory method. Singletons are often available via an idempotent[7] method, which returns a reference to the singleton instance and creates it for the first time if needed, although singletons may also be created at startup [09].

Thanks to that pattern, only one instance of the *Helper* is created inside the *EditConfigController*. That makes the usage of the class way safer and cleaner for future development. It is worth mentioning this class also has a reference to the instance of *XmlHandle* which is used to read from and write to the configuration file.

The class contains several methods with the prefix *Init-*. These methods are used to create and set up graphic elements, such as toggles, dropdowns, sliders, etc. In order to achieve generality, the methods pass UI elements as parameters, as shown in Listing 3.2. In this piece of code, the given slider is set to the default value.

[7] Refers to the property of a function or operation that can be repeated multiple times without affecting the result after the first execution.

Source code 3.2: Slider Initialization

```
1 public void InitSlider(Slider slider, float defaultValue)
2 {
3     if (slider != null)
4     {
5         slider.value = defaultValue;
6     }
7 }
```

In a similar way, there are methods with the prefix *On-* which handle all the different listeners and triggers for the graphic elements from *EditConfigController*. Likewise, these methods are built generically using input parameters so they can be easily used for future development and extension of the exercise configurator.

Once again, there are several methods for handling the internal logic of exercises and definitions, which I will describe in detail later in Chapter 3.3.

3.2.3 XML Handling

Similarly to the previous class, the class *XmlHandle* is a standard *C#* class without deriving from *MonoBehaviour*. One of the main objectives of this class is to gather all the code related to configuration file manipulation. A library used for manipulating XML data as if data were *C#* objects is *System.Xml*. Once again, the class uses a *singleton* design pattern which unites the instance manipulation in *EditConfigController*. Besides, it needs the reference to the class *ExercisesController* due to its dependency on the configuration filename.

When creating a new configuration file, the method *CreateEmptyXml()* is called. The method generates sample XML data[8] and saves it to the newly created configuration file in XML format. The file's name is *configHH-mm_dd.MM.yy.xml*, where datetime wildcards represent a current timestamp.

There are a few methods with the prefix *Save-*. These methods are used to save the *global settings* right after changing their values in the GUI. On the other hand, the therapeutic plan itself is saved after closing the window[9]. To do this, the *UpdateXmlFromList()* method is called, as displayed in Listing 3.3. It iterates over selected *movements* and *games* and writes the data into the XML file using methods *CreateXmlMovement()* and *CreateXmlGame()*. The writing is done separately, since *movements* and *games* have different parameters and attributes. Within the programming code, the class type is determined using the *is* operator[10] on lines 9 and 13.

[8] Including default values for the *global settings* and one sample *game*

[9] Saving dynamically changing exercises at the end of editing leads to improved application performance

[10] The *is* operator in *C#* checks if an object or expression can be cast to a specific type

Source code 3.3: Saving Exercises to XML File

```
public void UpdateXmlFromList(List<Exercise> exerciseList)
{
    ...
    foreach (var exer in exerciseList)
    {
        XmlNode node = doc.CreateElement(GlobalSettings.
    XML_EXERCISE);
        root.AppendChild(node);

        if (exer is Movement)
        {
            CreateXmlMovement(doc, node, (Movement)exer);
        }
        else if (exer is Game)
        {
            CreateXmlGame(doc, node, (Game)exer);
        }
    }
    ...
}
```

From my perspective, it was a good decision to separate the manipulation of XML data into its own class because it ultimately follows the basics of the *Model-View-Controller (MVC)* design pattern. In fact, all three classes in the GUI module follow the pattern in a way.

The *MVC* is a software architecture that splits an application into three components — model, view, and controller. The model contains or represents the data that users work with. In my case, it is presented by *XmlHandle*. The view renders the model as a UI which is showcased by *EditConfigController*. The controller processes incoming requests and performs operations on the model. This behaviour can be seen in *Helper*. The separation of tasks between each component leads to a well-defined code, making the application easier to maintain and extend over its lifetime [FS11].

3.2.4 Class Diagram

If visualizing the relationships between the individual classes is desired, then a UML[11] class diagram can be created, as can be seen in Figure 3.3. The diagram portrays the mentioned classes and their interdependencies. It is worth noting, there are several classes that derive from *Monobehaviour* class.

[11] Unified Modeling Language

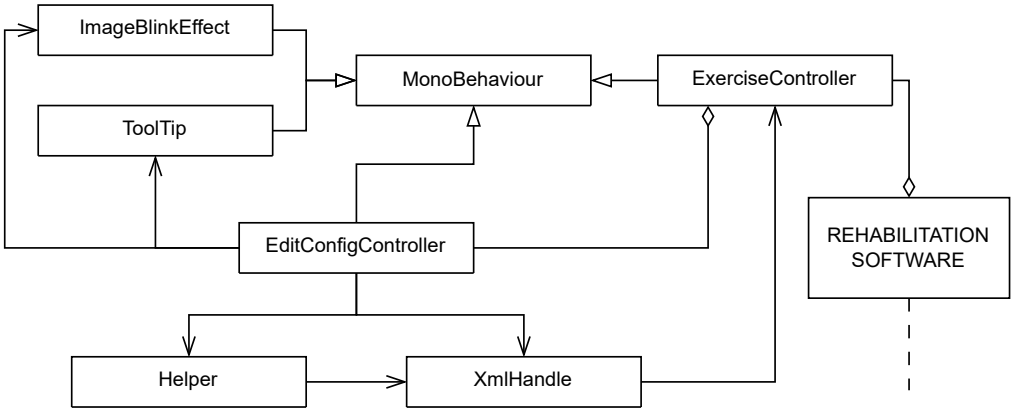

Figure 3.3: UML Class Diagram

3.3 Exercise Configurator Window

In this chapter, I am going to describe in more detail the internal logic and function of the graphical elements of the GUI including their appearance. Apart from *global settings* which are defined manually due to their uniqueness, *definitions* and *exercises* are modelled using Unity *prefabs*[12]. Thanks to prefabs, it is way easier to create any SW in Unity with an emphasis on a dynamic structure that can be effortlessly changed and maintained in future development. Prefabs are handled similarly to objects in programming languages, which means that every object must be instantiated before it can be used and destroyed at the end of its life cycle.

3.3.1 Global Settings

This set of parameters adjusts the settings of the whole environment in the rehabilitation SW. Since the parameters are used in a static way, there was no need to use prefabs while creating the controls. On the other hand, there is a possibility that new configuration settings will be added in the future. Therefore, I tried to design these elements descriptively with an emphasis on readability. I added new constants in the already existing class *GlobalSettings*. This class describes used XML tags and elements in the configuration file. To create a new parameter in the *global settings*, a new XML tag or element needs to be added to the *GlobalSettings*. By following this approach, the programmer is more likely to implement the new parameter correctly and not forget anything important. Controls for the *global settings* can be seen in Figure 3.4. This functionality was the first to be implemented so the given figure also shows the initial draft of the GUI module.

[12]A pre-made object template that can be used to create multiple instances of the same object

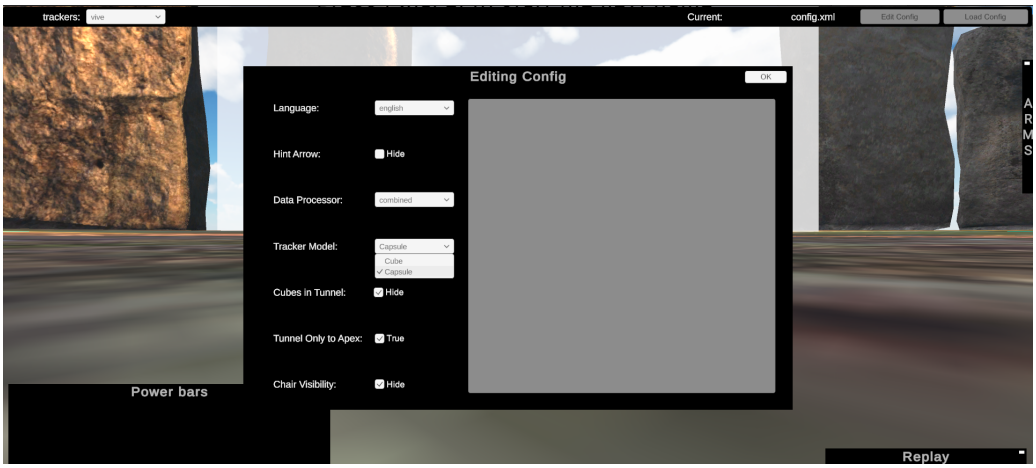

Figure 3.4: Global Settings Controls

As for the implementation itself, the text of captions and tooltips is assigned to the corresponding UI elements thanks to Unity's *TextMeshProUGUI*[13] in the method *FillSettingLabels()*[14] of the class *EditConfigController*. For each individual setting listed in Chapter 2.5.3.1, toggles and dropdowns are initialized and matching listeners are attached to them in *FillSettingInputs()*. For settings with value *true* or *false*, a Unity UI element called *Toggle* is used. It can be either checked or unchecked. For settings with the list of values, a UI element named *TMP_Dropdown* is used. It shows the currently selected value and, when clicked, displays a pop-up box that presents all the possible values for the user to choose from.

3.3.2 Exercise Definition

While designing a control panel for handling *exercise definitions*, I kept in mind its dynamic requirements. There is no static number of *definitions*, as was the case with *global settings*. Thus, I modelled the *definition* element using a generic prefab, as shown in Figure 3.5.

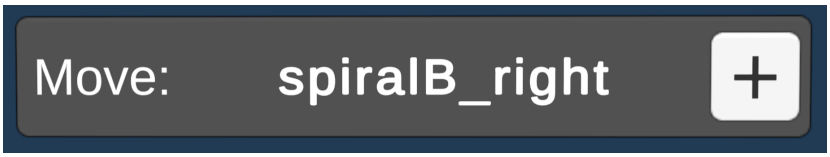

Figure 3.5: Definition Prefab

[13] A Unity asset that provides advanced text rendering and layout features for creating high-quality text in UI elements

[14] All the language texts, such as labels, captions or tooltips, are stored in the file *language.xml*

The implementation is done in Unity's file *Definition.prefab*. The whole element consists of three parts. On the left side, there is the label which shows the exercise type — *movement*[15] or *game*. In the middle, there is an *ID* that specifies the string name of the exercise. On the right side, there is a Unity UI element named *Button* which is used to add the given exercise to the current therapeutic plan. I also needed a GUI container that would be able to hold a dynamic number of these prefabs. Hence, I chose to use a *Scroll View* that was already mentioned in Chapter 3.2.1.1.

Instantiating the prefab is done in the function *InstantiateDefinition()* of the class *EditConfigController*. The instantiating is done separately for *movements* and *games*, since these two types have different parameters and attributes. Subsequently, linking data, captions and listeners to the instantiated prefabs is performed in methods *PopulateMoveDefinition()* and *PopulateGameDefinition()* of the class *Helper*. After clicking the Unity UI's *Button* with the text *'+'*, the methods *AddMovementFromDefinition()* and *AddGameFromDefinition()* add a new *exercise* to the therapeutic plan and instantiate it in the second *Scroll View* containing *exercises*.

A new *movement* with a different set of parameters can be added in the future using the XML file *movement-definitions.xml*. In the same way, a completely new game with different properties can be added to the rehabilitation SW. The generic approach, used during the development, enables making even slight changes easily without the need of rewriting the whole code. For instance, adding a new parameter describing the *movement* would result in modifying *Definition.prefab* and methods *PopulateMoveDefinition()* and *AddMovementFromDefinition*. On the other hand, adding a new *exercise definition* with the same set of parameters will automatically include the exercise in the SW without any required code modifications.

3.3.3 Exercise

Similarly to the *definitions*, there is no fixed number of *exercises*. Therefore, *exercises* were modelled using prefabs and they were grouped into another *Scroll View*. Individual *exercises* form together a therapeutic plan, which is represented by this *Scroll View* and refers to the XML configuration file. Speaking of prefabs, some of their controls are shared for *movements* and *games*.

An exercise is removed from the plan using the *'X'* button in the right corner of the prefab. To ensure that each therapeutic plan has at least one *exercise*, the program has been designed to prevent the removal of the last *exercise* in the plan. The removing is implemented in the method *OnRemoveClicked()* by calling Unity method *SetActive(false)*[16].

[15] The shorter form *move* is used as an abbreviation

[16] The method is preferred to *Destroy()* because it preserves the object's state, allows reactivation and improves performance [Uni23]

To change the order of the exercises, the user can use arrow buttons — *up* and *down* — which are located on the right side of each prefab. The arrow buttons do not allow the exercise to move up if it is in the first position, and vice versa. This functionality is implemented in the method *MoveByNumber()* by calling Unity method *SetSiblingIndex()*[17]. Once again, a Unity UI's *Button* is used for both removing and changing order.

The *exercises* are numbered on the left side of the prefab and they are performed in the given order when the VR application is launched. There are also labels for determining *type* and *ID* of the exercise.

3.3.3.1 Movement

An exercise of type *movement* is implemented in Unity's *Movement.prefab* file. At first, the prefab is instantiated in methods *PopulateMove()* and *AddControls()* of the class *Helper*. During that process, all graphical UI elements are created and event listeners and triggers are attached to enable interactivity. The *movement* element with controls and switches can be seen in Figure 3.6.

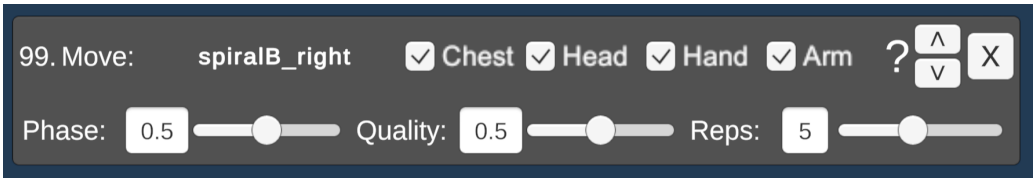

Figure 3.6: Movement Prefab

In the top row of the prefab, there are four Unity *Toggle* switches that define whether the trajectory of individual body parts should be displayed during the exercise or not. It is implemented using the generic method *OnDynamicToggleChanged()*, which is used multiple times for different parameters. The question mark displays a tooltip for these four switches.

In the lower part of the prefab, there are three segments with numerical parameters. These are: *target phase*, *quality threshold* and the number of *repetitions* (see Chapter 2.5.3.2). Changing values of these parameters can be done either by moving the *Slider* — in the method *OnMovementSliderChanged()* — or by direct numerical input into the *TMP_InputField* — in the method *OnMovementTextFieldEdit()*. Only a numerical input is allowed, and the *decimal point* is used as the separator. Both of these elements are part of Unity UI.

Thanks to reusable methods and prefabs the code is easily extendable. This generic approach implies that if another functionality needs to be added, such as

[17] Initially, reordering was done by destroying and re-instantiating all the prefabs instead, which was slowing down the response a lot

switching the arm colour, it is necessary to add a toggle to *Movement.prefab*. Afterwards, the logic of the toggle should be arranged in the *PopulateMove()* and the corresponding listener should be created or the generic one could be used in the method *OnDynamicToggleChanged()*.

3.3.3.2 Game

An exercise of type *game* is designed in Unity's *Game.prefab* file. First of all, the prefab is instantiated in methods *PopulateGame()* and *AddControls()* of the class *Helper*. In these methods, labels are initialized and triggers are attached. The prefab representing the *game* can be seen in Figure 3.7.

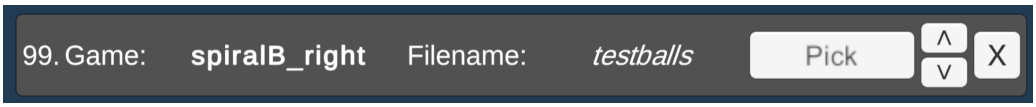

Figure 3.7: Game Prefab

In the middle of the prefab, there is a description specifying the name of the file from which game data will be loaded. The filename is given without the XML extension. The user can select the file by clicking a Unity *Button* with the text *Pick* located on the right side. When clicked, the user is prompted to select an XML game data file. The implementation is written in the method *OnGameFileClicked()* of the class *Helper*.

In a similar way, a completely new game can be added to the rehabilitation SW in the future. That would require adjusting the prefab *Game.prefab*, the initializing method *PopulateMove()* and the listener *OnGameFileClicked()*. Alternatively, a new prefab and methods similar to the one mentioned could be created.

3.3.4 Final Design

Putting together all the information about prefabs, controls and graphic elements resulted in the final implementation of the GUI module for the exercise configurator. A screenshot of the module running in the rehabilitation SW can be seen in Figure 3.8.

On the left-hand side, there are *global settings* that are able to adjust the parameters and properties of the whole environment in the VR rehabilitation application. In the middle, there is a list of *definitions* for all available movements and games, which can be added to the selected therapeutic plan. The list is represented by the *Scroll View* with *definition* prefabs. On the right-hand side, there is a list of individual *exercises* representing the therapeutic plan, which is loaded from the XML configuration file. Again, this list is created using the *Scroll View* with *exercise* prefabs.

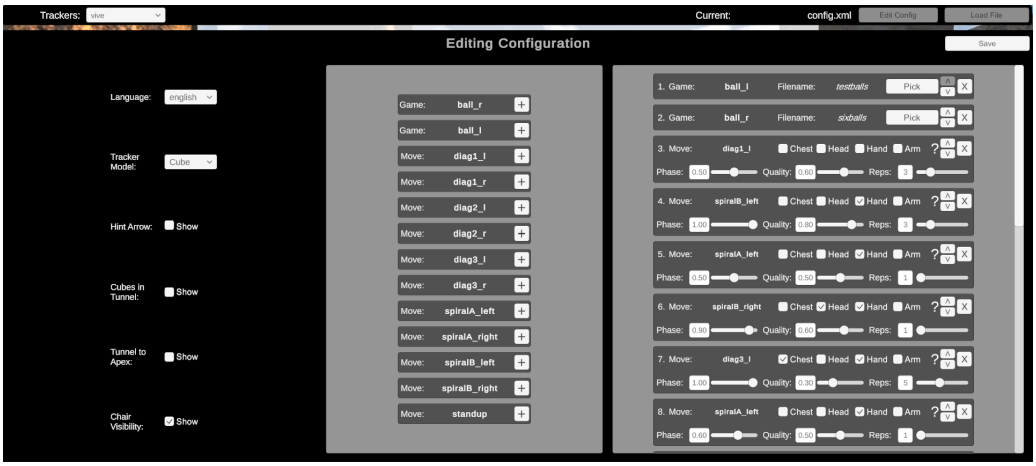

Figure 3.8: Final Implementation of the Exercise Configurator

After modifying the exercise plan, it is necessary to confirm the changes. This is done with the *Save* button, which is located in the upper right corner of the window. After pressing it, the changes are saved to the selected configuration file and the updated file is directly loaded into the application, so there is no need to restart it.

3.4 Additional Functionality

3.4.1 Floating Tooltip

A tooltip is a graphical element providing additional information about a given item or feature, which is typically displayed when the user hovers the mouse over that item. In the GUI module of the exercise configurator, it is necessary that therapists understand the meaning and behaviour of the UI elements correctly, so they can create the desired therapeutic plan.

Therefore, I implemented the tooltip functionality in the Unity class *ToolTip* and in the file *Tooltip.prefab*. It can display an auxiliary text on a white background[18], as shown in Figure 3.9. The text is assigned to corresponding UI elements in the method *AttachTranslatedToolTip()* of the class *EditConfigController*. I based the implementation on [Cod20].

3.4.2 Blinking Effect

While developing the GUI module, I noticed that adding or reordering *exercises* happens almost instantly, so I often lost track of which *exercise* I have just modified.

[18]All the text for tooltip captions is given in the file *language.xml*

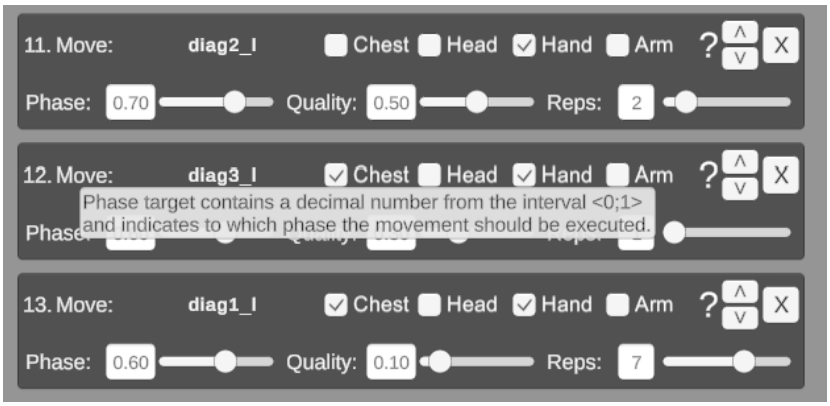

Figure 3.9: Tooltip for Movement

In order to correct this behaviour, I decided to add a graphical effect that would increase the quality of the UX design.

Information about which exercise has just been edited is displayed using the yellow blinking effect[19], as can be seen in Figure 3.10. The effect is implemented in the Unity class *ImageBlinkEffect* and this component is added to corresponding exercises in the method *BlinkYellow()* of the class *EditConfigController*. This functionality was adapted from [Uni21].

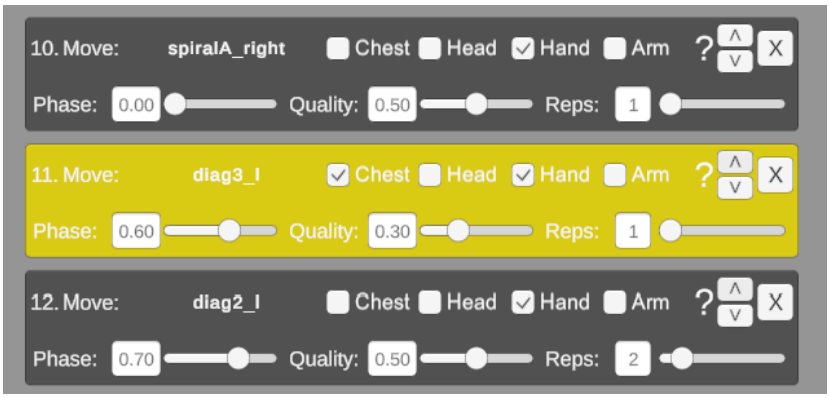

Figure 3.10: Blinking Effect

3.4.3 Unity Layout

In order to design a user-friendly GUI, I needed to come up with a layout structure that can arrange all the UI elements (*GameObjects*) logically in the window, regardless of the screen pixel count. In Unity, I combined the following components

[19]The blinking is stopped after a few seconds so that a therapist is not excessively distracted

to achieve the desired result — *HozirontalLayoutGroup*, *VerticalLayoutGroup* and *LayoutElement*.

Both layout groups allowed me to position elements side by side or on top of each other. In addition, *LayoutElement* was used to maximize the area of both *Scroll Views* when using computers with different screen sizes. As presented in Figure 3.11, all the *GameObjects* were put into the Unity hierarchy (left), combined with mentioned components, to create a friendly GUI (right).

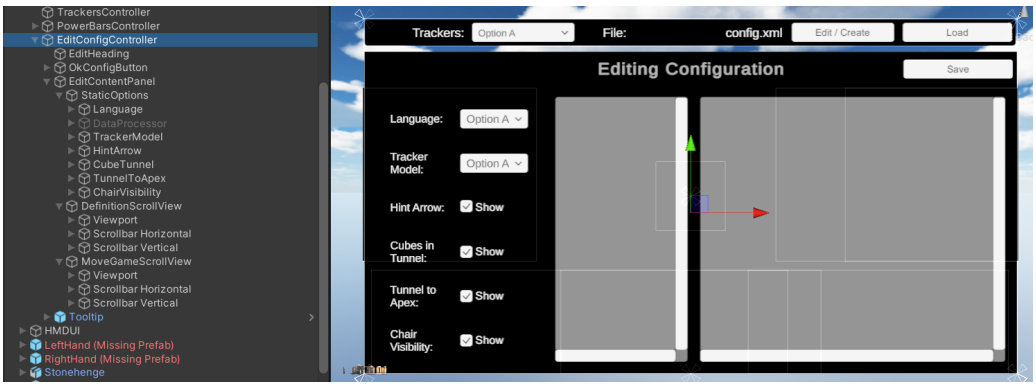

Figure 3.11: Unity Hierarchy

Testing 4

4.1 User Manual

Before the testing itself, it was necessary to familiarize the therapists with the environment of the GUI module for exercise configuration, so that the therapists could more easily understand the functionality of the individual controls. For this reason, two files were created. Both of these documents are listed in Chapter *Attachments* and the used language is Czech.

The first of them is a traditional user manual in text form. Therapists who rely on a more detailed and descriptive approach can get acquainted with the exercise configurator through a brief PDF document. There are several chapters describing individual parts of the GUI module and demonstrating an illustrative use of the application.

The second one is an instructional video in MKV[1] format. This concise video manual is suitable for therapists who prefer a more illustrative demonstration of the functionality. The video shows a description of the graphic elements and an example of the configuration.

4.2 Test Scenario

The VR rehabilitation SW is designed to be used at both University Hospital Královské Vinohrady and Thomayer University Hospital. The physiotherapeutic department team of the first mentioned hospital was given the application for testing purposes. During the testing process, Bc. Lubomír Rodina acted as a mediator, so he facilitated the new version of the SW to several therapists. Additionally, he created a test case, which was then provided to the therapists, as can be seen in Appendix A.

This test scenario was designed to cover various aspects of the exercise configuration. The scenario includes adding exercises to an XML configuration file, changing the parameters and order of these exercises and renaming the given con-

[1] Matroska Video File

figuration file. The desired output of this test case in rehabilitation SW is shown in Figure 4.1.

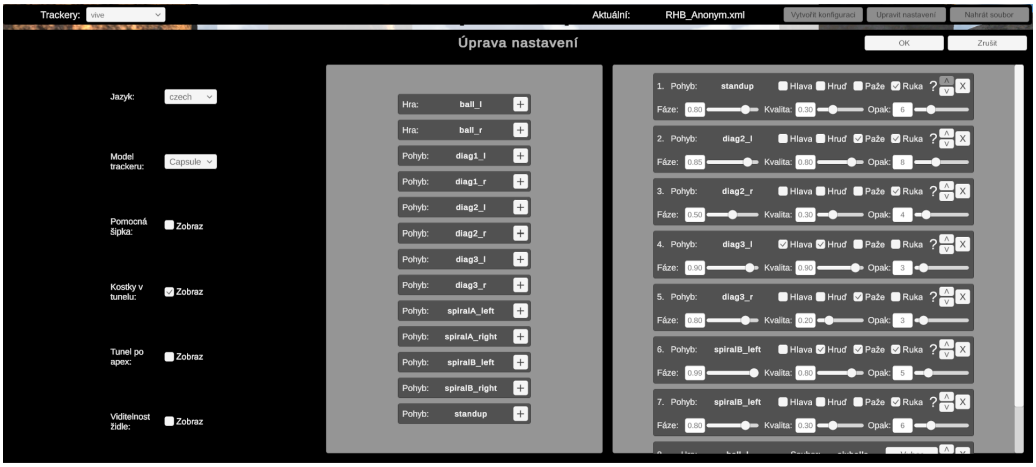

Figure 4.1: Desired Test Case Result from Appendix A

4.3 Feedback

Based on the above test scenario, the team of therapists provided feedback in the form of comments and ideas for improvement, as shown in Appendix B. Furthermore, the document emphasizes in bold the key points that Bc. Lubomír Rodina considered essential, as the primary person responsible for utilizing the VR rehabilitation SW. As some remarks were repeated in the feedback document, I also considered these points to be quite important.

As stated in the feedback, the therapeutic team often mentioned the demanding creation of a new XML configuration file. The implemented process of deselecting the loaded config file in order to create a new one using only one button is indeed a bit complicated. Another point of feedback was a difficult file renaming together with complex navigation in the folder structure of the application. One of the other requirements was to have the ability to revert the configuration back to its original state without saving any changes made. Other remarks were, for instance, a typo in one of the tooltips, different alphabetical order for *game* and *movement* definitions and an increase in the maximum number of repetitions. I incorporated the above changes into the new version of the application, as described in detail in Chapter 4.4.

In the feedback document, there were also some demands, which I did not include in the new version of the SW. One of them was a request to drag and drop elements from the definition *Scroll View* into the exercise *Scroll View* using the mouse.

This feature would be excessively difficult to implement, since mouse clicking is already recognised as an event for scrolling through the *Scroll View*. In other words, the whole logic of events and listeners related to adding a new exercise would need to be reworked. On the other hand, this feature might be a good idea for future improvements of the application. Another feedback point, which I did not address when implementing the new version, was storing exercise records in a folder by patient name. This request is outside the scope of this work, since it is not directly related to the exercise configuration process. Rather, this request is related to the process of saving exercise records within the rehabilitation SW. Therefore, the request was forwarded to the development team, specifically to Ing. Jakub Frank, and it will be addressed in the future.

4.4 Incorporation of Improvements

Regarding the incorporation of new features in the application, I will describe each point briefly, primarily focusing on implementation and ordering them based on their relative importance.

Creating and Renaming a New Configuration File. One of the most frequently mentioned feedback comments concerned the creation of a configuration file. Apart from the complicated process of creating a new file, there was also no option available to rename the file in the provided version of the rehabilitation SW, so it needed to be renamed in the *Windows File Explorer*. When it comes to implementation, I added a new *Create* button in the top panel of the *ExercisesController*. The button logic is provided by the *FilenameDialogShow()* method, which uses Unity *StandaloneFileBrowser*. The method prompts the user to create a new XML file directly in the folder *StreamingAssets* with the possibility of choosing a filename, as seen in Figure 4.2.

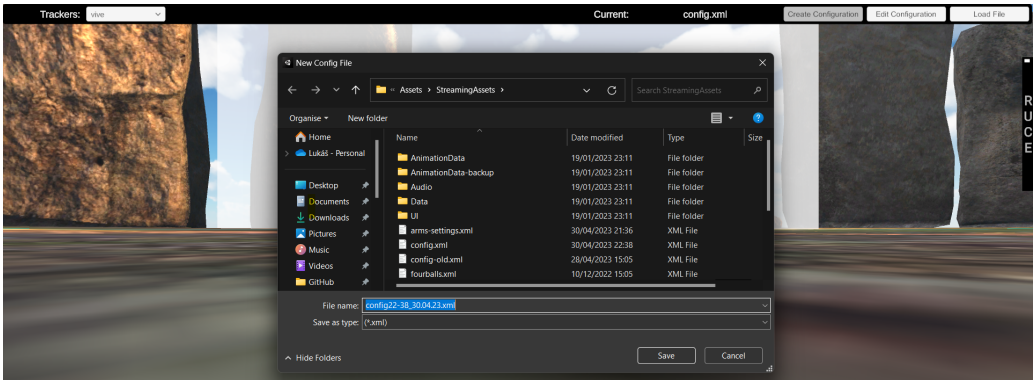

Figure 4.2: File Browser for Creating and Renaming a New Configuration

Reverting Configuration Changes. This remark concerned the possibility to restore the configuration back to its original state. In other words, a user could close the editing window without saving any changes made using a *Cancel* button. The idea behind the implementation was to create a deep copy of the *global settings* and the list of *exercises* at the start of configuring. After clicking the mentioned button, this deep copy containing original data is then loaded into the selected XML file in the method *CancelConfig()* and the editing window is then closed, as seen in Figure 4.3 (highlighted in yellow). It is worth mentioning that the application saves the *global settings* after each change, whereas the list of *exercises* is saved all at once after pressing the *OK* button, as was discussed in Chapter 3.2.3. Therefore, if the application crashes unexpectedly in the middle of editing, the selected file will contain the original list of *exercises* before saving, so the exercise data will remain consistent.

Figure 4.3: Reverting the Configuration Using the Cancel Button

Alphabetical Order in the Definition Panel. This comment was related to reordering elements in the left definition *Scroll View*. The solution required sorting temporary lists of *game* and *movement* definitions in the method *FillDefinitions()*. Because elements in the lists were not basic data types, but instances of *GameDefinition* and *MovementDefinition*, I decided to sort the lists using a lambda expression to access the string name of each definition, as shown in Listing 4.1

Source code 4.1: Reordering of Elements in the Definition Panel

```
1 public void FillDefinitions()
2 {
3     ...
4     gameDefs.Sort((x, y) => x.ID.CompareTo(y.ID));
5     movementDefs.Sort((x, y) => x.ID.CompareTo(y.ID));
```

```
6       ...
7   }
```

Order of Trajectory Animation Switches. In this case, it was desired to reorder the movement toggles inside the right exercise *Scroll View*. In the first version of the GUI module, the toggles follow this randomly selected order — *Chest, Head, Hand and Arm*. The main goal was to create a more intuitive and meaningful sequence for the toggles' position, for instance using an order inspired by the human body, as seen in Figure 4.4. To do so, it was necessary to edit the *Movement.prefab* file.

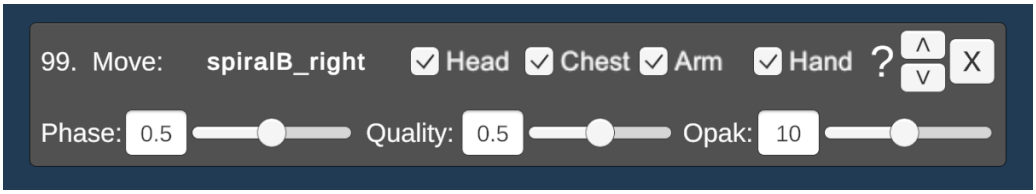

Figure 4.4: Altering the Movement Switches Order

An Increase in the Maximum Number of Repetitions. Since it is planned that VR rehabilitation will be at least partially moved to the home environment in the future, it would be beneficial to allow a larger range in the number of repetitions. Therefore, I increased this number from *10* to *20* in both methods that manipulate the number of repetitions of a movement — *OnMovementTextFieldEdit()* and *OnMovementSliderChanged()*.

Typo in One of the Tooltips. Besides implementing new functionality, I also corrected a few typos in the tooltips. These changes required revising the file *language.xml* that includes the text of labels in Czech and English.

Decimal Point and Decimal Comma. One of the comments addressed non-intuitive entering decimal numbers for parameters *target phase* and *quality threshold*. In the initial version of the SW, input was only possible using a decimal point as the separator. An ability to use a decimal comma as well was implemented in the method *OnMovementTextFieldEdit()*, where an entered comma is replaced with a point and the number is then further processed. The use of both separators does not depend on the locale used in *Windows*.

Conclusion 5

A new graphical user interface module was successfully implemented in the VR rehabilitation SW in this work. The implementation was preceded by the selection of a suitable *Unity* UI toolkit, and analyzing the structure of the application and exercise format. This led to more efficient integration of new functionality into the already developed rehabilitation SW without the need to change anything in the rest of the application.

The GUI module now allows the creation and editing of an individual therapeutic plan directly within the desktop version of the SW. As a result, it is no longer necessary to manually write configuration files in order to customize the therapeutic plan. The module provides users with a comprehensible editing environment using visual representation. There is also a user manual describing individual parts of the GUI, as well as a video tutorial that demonstrates basic functionality.

The application was handed over to the therapeutic department of University Hospital Královské Vinohrady in Prague. The new functionality was made available to the therapeutic team for testing and the team was encouraged to share their thoughts. The final version of the GUI module implemented some of the features that emerged from the feedback, such as simplifying the creation of a new configuration file or introducing a *Cancel* button in the editing window.

The SW is still backwards compatible with the original configuration structure, so any direct changes to the XML file will affect the therapeutic plan as in previous versions. In future development, newly created exercise definitions will be automatically included in the GUI module without requiring code modifications. On the other hand, programmers might add a new exercise type or a new set of parameters which requires some changes in the code. However, these potential adjustments will not be particularly challenging to implement because the module is designed in a generic way using object-oriented programming principles. Moreover, GUI elements in Unity were also created generically using Unity prefabs.

Test Case Document

Sestavte cvičební sestavu podle zadání:

Jazyk: *čeština*, Model trackeru: *capsule*, Pomocná šipka: *Ne*, Kostky v tunelu: *Ano*, Tunel po apex: *Ne*, Viditelnost židle: *Ne*

1)	Ball_l	sixballs			
2)	Ball_r	threeballs			
3)	Diag2_l	ruka/paže	fáze: 0,85	kvalita: 0,80	Opakování: 8
4)	Diag2_r	ruka	fáze: 0,5	kvalita: 0,30	Opakování: 4
5)	Diag3_l	hruď/hlava	fáze: 0,9	kvalita: 0,9	Opakování: 3
6)	Diag3_r	paže	fáze: 0,8	kvalita: 0,2	Opakování: 3
7)	SpiralB_l	ruka/paže/hruď	fáze: 0,99	kvalita: 0,8	Opakování: 5
8)	SpiralB_r	ruka/paže	fáze: 0,6	kvalita: 0,5	Opakování: 5
9)	Standup	ruka	fáze: 0,8	kvalita: 0,3	Opakování: 6
10)	Umístěte pohyb Standup na pozici 1) a hru Ball_l a Ball_R na pozici 8) a 9)				
11)	Soubor uložte a poté přejmenujte na: RHB_Anonym				

Feedback Document B

Míša Raková

- **V nabídce výběr cviků: Neprohazovat pravou a levou stranu v pořadí. Míčky jsou na levou končetinu, poté na pravou končetinu a u pohybu je to obráceně, prvně levá a poté pravá končetina.**
- Při vpisování hodnoty fáze či kvality se musí psát tečka (0.5), zda případně neumožnit čárku (0,5). Nebo variantu obojího.
- **Změnit pořadí zobrazování křivek: Hlava, Hruď, Paže, Ruka. Bude to více intuitivní.**
- Přetahování cviku z prostředního sloupce do pravého sloupce za pomoci myši a rovnou jej moct umístit i na námi vybranou pozici.
- **Pokud si otevřu config a měním parametry, ale poté se rozhodnu, že nechám původní verzi, nemohu ze souboru zpět. Mám jedinou možnost uložit nové nastavení anebo vypnutí celého programu, aby se změna neuložila.**
- **Chyba v popisku. U otazníku překlep ve slově: *Jednotlivých***

Lubomír Rodina

- **Možná navýšit počet opakování z 10 na 15-20.** Pokud bude aplikace užívaná pro domácí RHB, je možné, že pacienti budou plnit úkoly s vyšším počtem opakování.
- **Tlačítko pro vytvořit nový konfigurační soubor.**
- **Upravit pořadí zobrazování křivek: Hlava, Hruď, Paže, Ruka.**
- **Jednodušší přejmenování konfiguračního souboru.**

MUDr. Barbora Miznerová

- Při přetažení cviku 1diag_l a nastavení parametrů, aby se automaticky nastavili stejné parametry i pro cvik 1diag_r. *(Urychlení nastavení cvičební sestavy, ale s možností poté měnit parametry.)*
- Ukládání záznamu pohybu do složek podle názvu configu (např. Novák), ve které by byli záznamy 1-15 terapie.
- **Složité vytváření nového konfiguračního souboru.**

Franziska Vosenová

- **Prohodit strany míčků.**
- Složité hledání ve složkách, kde je uložený soubor config a verze míčkových her.
- **Přejmenování souboru config v aplikaci. Obtížné hledání ve složkách aplikace.**

Sára Hukaufová

- **Jednodušší přejmenování vytvořeného souboru config.**
- **Tlačítko odkazující rovnou do složky sestav cviků, aby se nemuselo hledat v podsložkách.** *(Pro většinu lidí, kteří neměli zkušenost s pravidelným používání aplikace matoucí, dlouho hledali, kde je uložený konfigurační soubor.)*

Anna Herynková

- **Složité ukládání a vybírání .xml souboru.**

Bibliography

[Cod20] CODE MONKEY. *Dynamic Tooltip in Unity! (Resizable, Follows Mouse, Edge Detection)*. 2020. Available also from: https://unitycodemonkey.com/video.php?v=YUIohCXt_pc.

[Fra20] FRANK, Jakub. *Sběr 3D dat pro rehabilitační software ve virtuální realitě*. 2020. Available also from: http://hdl.handle.net/11025/41806. University of West Bohemia.

[Fra22] FRANK, Jakub. *Rozšíření aplikace pro rehabilitaci paže ve virtuální realitě*. 2022. Available also from: http://hdl.handle.net/11025/49092. University of West Bohemia.

[FS11] FREEMAN, Adam; SANDERSON, Steven. The MVC Pattern. In: *Pro ASP.NET MVC 3 Framework*. Berkeley, CA: Apress, 2011, pp. 63–88. ISBN 978-1-4302-3405-0. Available from DOI: 10.1007/978-1-4302-3405-0_4.

[Her21] HERYNKOVÁ, Anna. *Virtuální realita v rámci fyzioterapie pacientů s roztroušenou sklerózou mozkomíšní*. 2021. Available also from: http://hdl.handle.net/10467/97904. Czech Technical University in Prague.

[Mar15] MARCUS, Aaron. Dare We Define User-Interface Design? In: *HCI and User-Experience Design: Fast-Forward to the Past, Present, and Future*. London: Springer London, 2015, pp. 21–29. ISBN 978-1-4471-6744-0. Available from DOI: 10.1007/978-1-4471-6744-0_4.

[Mic23] MICROSOFT. *A tour of the C# language*. 2023. Available also from: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[NL14] NOLAN, Deborah; LANG, Duncan Temple. An Introduction to XML. In: *XML and Web Technologies for Data Sciences with R*. New York, NY: Springer New York, 2014, pp. 19–52. ISBN 978-1-4614-7900-0. Available from DOI: 10.1007/978-1-4614-7900-0_2.

[Rod20] RODINA, Lubomír. *Virtuální realita v rámci rehabilitace pacientů s roztroušenou sklerózou mozkomíšní*. 2020. Available also from: http://hdl.handle.net/20.500.11956/124048. Charles University.

[Sib16] SIBŘINOVÁ, H. *Co je to roztroušená mozkomíšní skleróza*. 2016. Available also from: https://www.ereska-aktivne.cz/. The article on multiple sclerosis.

[09] Singleton Pattern. In: *Learn Objective-C for Java Developers*. Berkeley, CA: Apress, 2009, pp. 429–432. ISBN 978-1-4302-2370-2. Available from DOI: 10.1007/978-1-4302-2370-2_23.

[Tob22] TOBIN, Oliver. *Multiple Sclerosis*. 2022. Available also from: https://www.mayoclinic.org/diseases-conditions/multiple-sclerosis/symptoms-causes/syc-20350269.

[06] Understanding Object Serialization. In: *Pro VB 2005 and the .NET 2.0 Platform*. Berkeley, CA: Apress, 2006, pp. 555–571. ISBN 978-1-4302-0160-1. Available from DOI: 10.1007/978-1-4302-0160-1_19.

[Uni20a] UNITY TECHNOLOGIES. *Scroll Rect*. 2020. Available also from: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-ScrollRect.html. Unity UI 1.0.0.

[Uni20b] UNITY TECHNOLOGIES. *The main UI Builder window*. 2020. Available also from: https://docs.unity3d.com/2019.4/Documentation/Manual/ScriptingToolsIDEs.html. UI Builder 1.0.0.

[Uni22a] UNITY TECHNOLOGIES. *Comparison of UI systems in Unity*. 2022. Available also from: https://docs.unity3d.com/2019.4/Documentation/Manual/UIToolkits.html. Unity User Manual (2019.4 LTS).

[Uni22b] UNITY TECHNOLOGIES. *GameObject*. 2022. Available also from: https://docs.unity3d.com/2019.4/Documentation/ScriptReference/GameObject.html. Unity User Manual (2019.4 LTS).

[Uni22c] UNITY TECHNOLOGIES. *Integrated development environment (IDE) support*. 2022. Available also from: https://docs.unity3d.com/2019.4/Documentation/Manual/ScriptingToolsIDEs.html. Unity User Manual (2019.4 LTS).

[Uni22d] UNITY TECHNOLOGIES. *MonoBehaviour*. 2022. Available also from: https://docs.unity3d.com/2019.4/Documentation/ScriptReference/MonoBehaviour.html. Unity User Manual (2019.4 LTS).

[Uni23] UNITY TECHNOLOGIES. *GameObject.SetActive*. 2023. Available also from: https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html. Unity User Manual (2019.4 LTS).

[Uni21] UNITY3D SCHOOL. *Creating Image Blink effect by Changing Color | UI | Color Lerp | Unity Game Engine*. 2021. Available also from: `https://u3ds.blogspot.com/2021/09/creating-image-blink-effect-by-changing.html`.

List of Figures

2.1 HTC Vive Headset 7
2.2 Spiral Movement 9
2.3 Diagonal Movement 9
2.4 Ball Connecting Game 10
2.5 Unity Editor 15
2.6 UIElements Example [Uni20b] 16
2.7 Unity UI Example 16

3.1 Exercise Controller Window 20
3.2 Unity Scroll View [Uni20a] 22
3.3 UML Class Diagram 25
3.4 Global Settings Controls 26
3.5 Definition Prefab 26
3.6 Movement Prefab 28
3.7 Game Prefab 29
3.8 Final Implementation of the Exercise Configurator 30
3.9 Tooltip for Movement 31
3.10 Blinking Effect 31
3.11 Unity Hierarchy 32

4.1 Desired Test Case Result from Appendix A 34
4.2 File Browser for Creating and Renaming a New Configuration 35
4.3 Reverting the Configuration Using the Cancel Button 36
4.4 Altering the Movement Switches Order 37

List of Listings

2.1 Sample XML Code Snippet 11
2.2 Therapeutic Configuration File 12
2.3 IMGUI Example 17
3.1 Creating the Configurator Window 20
3.2 Slider Initialization 23
3.3 Saving Exercises to XML File 23
4.1 Reordering of Elements in the Definition Panel 36

Attachments

The attached archive has the structure as listed:

Application_and_libraries

A folder containing the application build, source code and both user guides. To run the Unity project, it is recommended to use the *UnityHub* application and the Unity version, specified in the project. After starting the application build, it is necessary to press the space bar to confirm tracker initialization.

VR_Rehabilitation_3.3. A directory containing the SW build of the rehabilitation application with the added GUI module for configuring exercises. The *VR Arm Rehabilitation_Data/StreamingAssets* subfolder contains all the configuration files.

VR_Rehabilitation_Project. A directory containing the Unity project of the rehabilitation application with the added GUI module for configuring exercises. The project repository is located at `https://gitlab.com/frankkuba/vr-arm-motion`.

Uživatelská příručka pro konfiguraci cvičení. A PDF document that describes the configurator of exercises. This document serves as a textual user manual.

Video návod pro konfiguraci cvičení. An MKV video file that demonstrates the basic functionality of the configurator of exercises. This file serves as a video guide.

Text_thesis

A folder with the LaTeX source code and a PDF file containing the text of this paper.

Latex/img/cust/. A directory with images used in the text.

Latex/img/pdf/. A directory with PDF documents used in the text.