POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

ReFlex VR: a Virtual Reality game for upper limb rehabilitation of stroke survivors

Supervisors
Prof. Andrea Sanna
Prof. Aikaterini Bourazeri

Candidate
Marta Paoli

Academic Year 2024-2025

Acknowledgements

To my dad,
regardless of how it goes it will be a success.

Abstract

Stroke is a medical emergency that occurs when not enough blood is able to reach brain cells, leading to many complications and, if not promptly treated, even death. It is the main cause of adult disability, with stroke survivors often experiencing paresis in one upper limb, which can make them unable to perform many self-care activities, resulting in a drop in their quality of life and a burden on their loved ones.

By incorporating evidence from many previous studies on the topic and directly gathering opinions from experts in the field, this thesis' aim was to develop a Virtual Reality application, available through the Meta Quest 3 headset, that can help stroke survivors regain upper limb mobility by performing structured, highly repetitive tasks based on real life activities and inspired by the Graded Repetitive Arm Supplementary Program (GRASP). Users are expected to stretch their paretic limb in many directions and perform fine movements with their paretic hand.

After completing the prototype, the application was then tested in an experiment conducted with a group of patients from Centro Puzzle to try to objectively establish whether it could make a difference, in conjunction with traditional physiotherapy, as opposed to traditional rehabilitation by itself. The results were inconclusive, however the application was well liked by patients and the VR technology was well tolerated, with no side effects of note.

Contents

Chapter 1

Introduction

1.1 The impact of stroke

Stroke is the second leading cause of death in Europe, as per the European Cardiovascular Disease Statistics, as well as the second leading cause of death worldwide [47], and one of the main causes of adult disability, affecting mostly people over the age of 55 [31]. Ischemic stroke is the most common type of stroke, and it occurs when the brain's blood vessels become partially or completely blocked, either by blood clots, fatty deposits, or other kinds of debris typically coming from the heart. As a result, blood flow is drastically reduced to the point of preventing brain tissue from getting the necessary oxygen and nutrients; brain cells begin to die within minutes from the stroke's onset, and if not promptly treated this could lead to heavy complications and even death. Fig. 1.1 shows the global incidence of stroke and ischemic stroke between 1990 and 2017 [12]. Almost 70% of those people survive, but only a quarter of them recover completely with no ensuing complications, whereas the remaining 75% have to undergo a rehabilitation period [22] spanning from several weeks to months or sometimes years. The primary outcomes of a stroke include a degree of motor impairment in the upper limbs, resulting in a partial or complete inability to move one limb [1], which leads to experiencing significant difficulties performing reaching and grabbing tasks due to poor motor coordination and control over grip strength [2]. Aside from motor disabilities, stroke survivors may also be affected by unilateral neglect, which is the inability to detect or respond to physical stimuli received on the side of the body that is opposite to the hemisphere that has been hit by the stroke [38].

From a neuroscience standpoint, a stroke may cause an impairment in the primary motor cortices or areas responsible for praxis control, specifically those in the parietal regions [4], altering proprioceptive signals and the perception of peripersonal space [41]. Proprioception means a sense of relative position of one's body to the environment and the sense of one's strength of effor being applied while moving. In other words, stroke survivors experience a distorted proprioception of their body and an impaired embodiment ability of the injured limb, manifesting in everyday life as an inability to indipendently perform most self-care activities (such as eating, dressing, maintaining an adequate level of personal hygiene, and so on) which results in having to constantly rely on a caregiver.

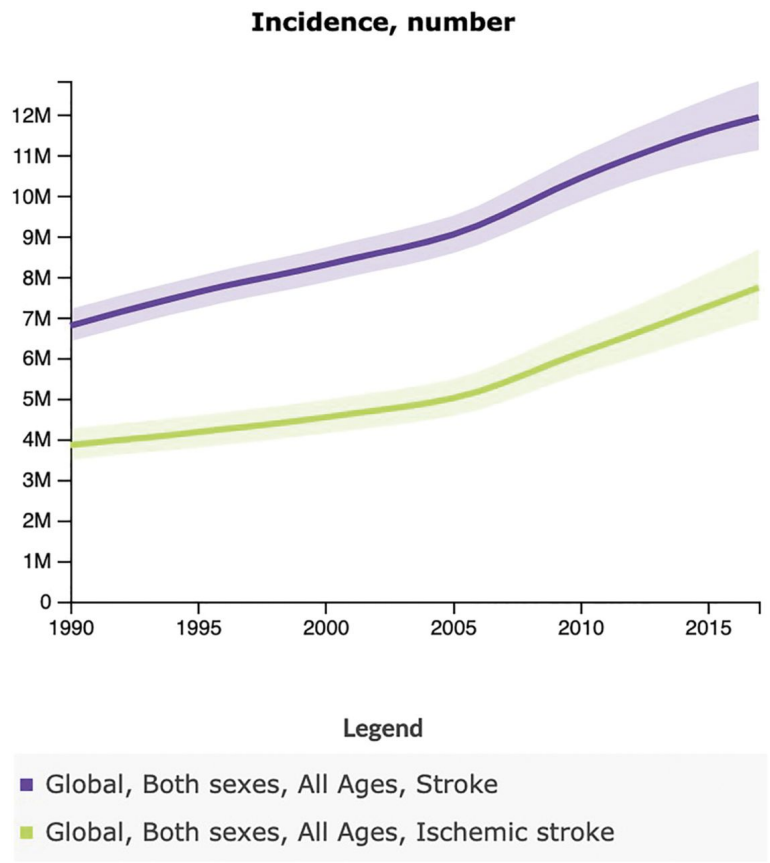


Figure 1.1. Global incidence of stroke and ischemic stroke from 1990 to 2017 (Source: Institute for Health Metrics and Evaluation; University of Washington, Seattle; 2015)

This is why consistent and effective rehabilitation to restore motor skills is fundamental, both to give back a sense of independence to the patients and also to alleviate the inevitable burden on their family members, that usually end up becoming their informal caregivers. In fact, motor dysfunction is the most prevalent impairment experienced by stroke survivors, with 9 out of 10 suffering from some degree of upper limb motor disability [48].

1.2 Available treatments

Neurorehabilitation is a multidisciplinary program that has been developed to help patients reattain functional independence [34]; it is typically administered in rehabilitation centers (which is why it is often referred to as occupational therapy), where patients may engage in many forms of therapy, including but not limited to physiotherapy and activities to improve mobility [5], speech therapy and cognitive rehabilitation to address any possible cognitive impairment accompanying the decline in motor abilities [9, 35], pain

and stress management [25] and emotional support [24] to support the patients' mental health as well as their physical health after such a life-altering event.

Sensorimotor training exercises are fundamental in the neurorehabilitation process to help patients improve their motor coordination and their sensory integration [39] through highly repetitive, high dosage, task-specific and personalized motor tasks. Other key elements in motor rehabilitation include multisensory stimulation, immediate and appropriate feedback, and increasing difficulty to lower boredom and to increase efficacy [26]. This type of training is usually administered either one-on-one or in groups, within environments specifically designed for practicing task-specific and context-specific exercises. Personalization of the various tasks by tailoring them to each patient according to their ability and needs plays a key part in ensuring that patients have stimulating experiences (since previous studies have shown that more engaging activities may lead to better results [7]), all while not becoming overwhelmed, as stroke and more generally speaking ABI (Acquired Brain Injury) survivors might perceive sensory stimuli as abnormal (sensory hyposensitivity) or too intense (sensory hypersensitivity) [17].

In some rehabilitation centers, healthcare professionals employ environmental enrichment techniques, which are based on ad-hoc environments reminding patients of their daily lives in order to further encourage physical, cognitive, and social activities [20]; these environments might include gyms, kitchens, stairs, or even outdoor environments such as bus stops and supermarkets. Such techniques, however, require many more resources and they also might be perceived as too demanding for some patients due to the increased amount of simultaneous stimuli.

Another approach is Constraint-Induced Movement Therapy (CIMT), which employs active task-oriented activities and behavioral strategies [15], and it is considered the most effective treatment for the rehabilitation of paretic limbs. It is based on the principle of learned non-use, meaning that stroke survivors tend to only use their unaffected limb simply because it is more convenient, therefore if patients are rendered unable to utilize their sane arm they will start using their paretic limb to perform required and purposeful movements. The original CIMT protocol consists of three main components: making the patient wear a padded mit on their unaffected hand for most of their waking hours, making them perform task-oriented and highly repetitive training with the paretic limb, and lastly applying behavioral practices to increase the patient's compliance with the protocol and also to help them transfer the skills they acquire during the training to real-life scenarios. This is still considered the golden standard for this type of treatment to this day, although some slight variations have been born, such as modified Constraint-Induced Movement Therapy (mCIMT), which doesn't adopt CIMT's three components but instead focuses only on the repetitive and task-specific training part, and forced use therapy, which is limited to the immobilization of the unaffected arm to force patients to use their paretic limb with increased frequency.

In order to address and ameliorate unilateral neglect another form of rehabilitation has been devised, called mirror therapy [49]: the patient is asked to sit in front of a table and to place both of their arms on top of it, and an appropriately positioned mirror will reflect their unaffected limb on the side where their paretic limb actually is, inducing them to believe that their paretic limb is, in fact, fully functional, increasing muscle activity

in the affected limb to bridge the gap between their kinesthetic and visual perception. Mirror therapy works by activating the mirror neuron system in the brain, and it has been proven to be an effective treatment to diminish unilateral neglect, with the potential to improve their quality of life.

Lastly, an additional treatment of note for stroke survivors is functional electrical stimulation (FES), which is not a standalone treatment but rather a procedure that can supplement and enhance the effects of other forms of therapy. It works by sending an electrical current to the muscles of the affected limb, simulating the signals that they received by the patient's brain before the stroke occurred, which stimulates them and makes them contract and move. This type of treatment cannot resolve a state of paralysis, but it can help strengthen the paretic limb's muscles which in turn should make it easier for the patient to perform any other form of motor rehabilitation, thus rendering it more effective [6].

1.3 Limitations of traditional physiotherapy

The main limitations imposed by traditional forms of physical rehabilitation for stroke survivors are due to the fact that they call for a high amount of manpower and funds, often making them inaccessible and difficult to keep up for prolonged periods of time. This is due to the fact that all of the available treatments require consistent and high volume practice, which would generally be hard for the patients to achieve independently, mostly because their motivation wanes overtime.

Such lack of drive to keep following motor rehabilitation programs can be attributed to the fact that said programs are quite repetitive by design and not very stimulating, as well as far removed from activities that patients might find more purposeful because they are not related in any way to their activities of daily living (ADL), such as brushing their teeth, cooking or putting on clothes.

The types of rehabilitation that would be more engaging require a significant amount of tools or specific environments which would be cumbersome and expensive to achieve at one's own home. As a result, often the only patients that can achieve complete recovery are those that had less severe disabilities to begin with, whereas survivors from more acute forms of stroke will usually be assisted in a rehabilitation center for a relatively short period of time and then their recovery will plateau, leading to life-long impairments and an overall diminished quality of life.

1.4 Aims of this thesis

Technology is becoming increasingly accessible and present in our homes and day-to-day lives, from smartphones to computers to tablets and even video game consoles. As such, a lot of research has been conducted in the last few decades in the field of neurorehabilitation to try to devise a technology-based training program for stroke survivors that is relatively cheap compared to traditional alternatives, easy to follow and master even without a caregiver's help, engaging and viable even in the comfort of one's own home.

Some notable findings specifically related to the use of various forms of Virtual Reality are explored in Chapter 2, including examples of both immersive and non-immersive VR and utilizing to this end many different technologies, among which there are some commercially available mediums such as PCs, the HTC Vive HMD (Fig. 2.4), LMC cameras (Fig. 2.3), and Kinect cameras (Fig. 2.5); but also hardware solutions that have been built for the specific purpose of upper limb rehabilitation, such as the Elements system (Fig. 2.2), the Virtual Reality Rehabilitation System-Handbox (Fig. 2.10), and the RAPAEL Smart Glove (Fig. 2.11).

The idea of this thesis is to expand on previous findings by creating a new VR application for the Meta Quest 3 headset that takes a gamified approach to deliver effective and enjoyable upper limb rehabilitation to people that have suffered a stroke and still present a certain degree of motor disabilities, with the aim of keeping financial demands, the necessity of additional help for setup and the user's learning curve to a minimum. The application resulting from this thesis' work had to be intuitive, aesthetically pleasing, and give users an experience as close to real life as possible to maximize the benefits stemming from the sense of familiarity and also the transference of re-learned skills to the real world. There had to be a scoring system to keep track of patients' progress and at the same time to give them appropriate feedback on their performance.

This thesis has been made possible by the collaboration between the Politecnico di Torino and the University of Essex: the latter provided crucial feedback on the neuropsychological aspects that had to be kept in mind in the making of the application, as well as providing all of the relevant material, from videos of common physiotherapy exercises for stroke patients' upper limb rehabilitation to information on all of the most important tests to keep track of patients' motor recovery to this day. Furthermore, when the application was completed it has been used for an experiment with a group of patients from the Centro Puzzle in Turin, which has been greenlit and supervised by Professor Zettin.

Chapter 2

State of the art

2.1 Background on VR

The limitations imposed by traditional physiotherapy have pushed researchers to look for new methods to help patients regain mobility and a sense of independence in their day-to-day life. Among those new methods, Virtual Reality has shown promise for its capacity to thoroughly engage the patients and to entertain them while they improve their mobility. A Virtual Reality application is a program, usually experienced through a headset, that transports its users in a computer-generated 3D environment that simulates the real world; this type of application can be particularly immersive thanks to visual, audio and even haptic feedback.

The first recorded trial for the use of VR in rehabilitation dates back to 1996 [11] which assessed whether VR is a viable option in this sense, and from there on many different solutions employing Virtual Reality have been explored. Virtual Reality-based systems employed in neurorehabilitation can be divided in two macrocategories: nonspecific Virtual Reality (N-SVR) and specific Virtual Reality (SVR) [27]. Systems of the former type have been extensively studied in the 2010s, and they include any kind of commercially-available gaming console, such as the Wii and the Xbox; their main advantage is the low bar of entry since most people already have at least one gaming console at home, making this solution the least expensive, however the games that can be played on them were not specifically designed with the goal of physical rehabilitation, and as such they can only bring limited benefits to the patients. Furthermore, there is no way of objectively tracking any relevant variables related to the patient's movement unless external devices are added to the equation, losing the advantages of being low-cost and low-effort.

SVR systems, on the other hand, are created with the specific goal of promoting motor recovery in mind, which makes them much more suited for this purpose. They take advantage of key aspects of Virtual Reality to make the exercises fun and engaging for the users, while keeping track of objective metrics and allowing for repetitive, intensive, task-specific training, in accordance with the principles of neurorehabilitation.

Virtual Reality can be either immersive or non-immersive: the first type is what we typically envision Virtual Reality to be, which is an immersive experience aided by

the use of a headset or all-encompassing display to give the user a sense of presence in the computer-generated world, whereas non-immersive Virtual Reality still allows users to interact with a virtual environment in somewhat naturalistic ways by using devices like joysticks or haptic tools, but said virtual environment is visible through a computer screen, significantly lowering the sense of immersion.

2.2 Previous studies

Quality of life assessment Stroke affects one's cognitive abilities and motor skills, and as a result of that it also meaningfully impacts one's quality of life and capacity to take care of themselves. In their study, Ogun et al. [29] decided to explore a VR application's capacity to improve not only stroke survivors' motor skills but also their functional independence. They included a total of 65 patients in the study, dividing them into two groups (VR and control): both groups received rehabilitation three days a week for six weeks, but for the VR group each session, lasting approximately 60 minutes, consisted of playing four different VR minigames (Fig. 2.1), each one for about 15 minutes, whereas the control group received 45 minutes of conventional physiotherapy and 15 minutes of passive VR (meaning they wore the headset and watched 3d scenes). The passive VR for the control group was meant to blind both the patients and the evaluators to the group distribution.

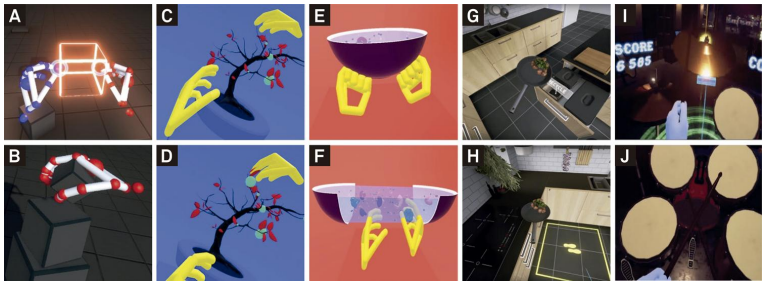


Figure 2.1. The four VR minigames: Cube handling (A and B); Decorating a tree (C and D) and picking up vegetables (E and F); Cooking in a kitchen (G and H); Drumming (I and J)

The patients in the VR group would be fastened to a chair for their sessions and they would wear an HTC Vive headset with a Leap Motion camera strapped to it to capture their arm and finger movements. Tests assessing both the patient's motor skills and their self-care skills were administered to all of the participants right before and right after the six weeks, and the comparison between the results showed a better performance for the VR group across the board, meaning that VR-based physical therapy for the recovery of upper limb function could be a viable option even without being paired with traditional physiotherapy.

This study presented two main drawbacks: the single-center design and the unusually high dropout rate, since almost 25% of the patients in both groups didn't conclude the

six week training period due to compliance issues. Another aspect to keep in mind with the VR games presented is that when shoulder movements were required the 10 to 15 minutes of uninterrupted exercise were reported to be exhausting, with the quality of movement progressively diminishing.

The *Elements* system Rogers et al. [21] conducted a pilot study to assess how useful could the *Elements* system be in conjunction with traditional physiotherapy to improve stroke survivors' motor skills, their cognitive abilities and also their quality of life. The study involved 21 patients which were divided in two randomly assigned groups: both groups received 3 hours of daily conventional physical rehabilitation for four weeks, and in addition to that the experimental group also participated in 12 VR sessions in total, each one lasting about 30-40 minutes.

The *Elements* system is an innovative solution consisting of a large tabletop touch-screen LCD panel with a built-in CPU, tangible user interfaces, and software to dynamically respond to the user's inputs. The user can engage with seven different tasks, each with its own base rules but all of them involving four hand-held objects placed on the display that need to be grabbed and moved in specific ways or placed in certain spots on the screen (Fig. 2.2). The tasks can be divided in two macrogroups: goal-directed tasks will have targets appear on the display, either following a specific pattern or randomly, and in order to win the user will have to grab the objects and place them on the targets, followed by both audio and visual feedback (for example the targets begin to glow more intensely and certain sound effects start playing), whereas exploratory tasks are meant to entice the user to explore the virtual environment and see what happens, for instance trying to create a melody by moving the different objects in a certain way. Furthermore, the *Elements* system allows the healthcare professionals to tweak the tasks' difficulty based on the patients' current physical and cognitive abilities.

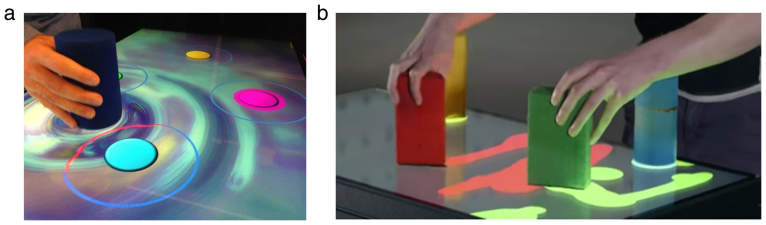


Figure 2.2. Two examples of *Elements* tasks, a goal-directed task (a) and an exploratory task (b)

The strength of this system lies in its use of so-called augmented feedback (AF), which includes all of the visual effects and the audio cues that react to the user's input via the hand-held objects; AF has been proven to promote the reacquisition of motor skills for people that have suffered a stroke, as well as the improvement of their cognitive abilities (since to complete the different tasks the patients had to firstly understand their demands and then act accordingly on a case-by-case basis). AF makes this system as engaging for the user as VR administered through a HMD, while having none of the

possible drawbacks, such as the cybersickness experienced by some acute neurological patients using a headset.

Out of the 21 patients recruited none of them dropped out of the study, and a comparison between the tests conducted right before the training period and those made right after shows that the experimental group's improvement far surpassed that of the control group both from a physical and a cognitive standpoint, and those improvements were maintained even a month after the end of the experiment, as shown by the results of the follow-up tests. Some tests regarding the participants' quality of life were conducted as well and they also showed a higher improvement for the experimental group, but the difference from the control group was not enough to be considered statistically significant and the fact that those tests are self-administered also has to be taken into consideration.

Among the limitations of this study, the researchers themselves cited the relative scarcity of the sample groups, the fact that none of the participants had severe impairments, the imbalance of training hours among the two groups (the experimental group received 7 more hours of training compared to the control group), the fact that the outcome assessors were not blinded to the study, possibly introducing some bias, and, perhaps most importantly, the cumbersome and expensive nature of the *Elements* system, which makes it less universally viable and definitely only applicable in a clinical setting.

HoVRS Another notable development in this field has been achieved with the creation of the Home based Virtual Rehabilitation System (HoVRS) [36]. The main concern this technology aims to address is the generally low adherence to unsupervised home exercise regimens for people with stroke. In order to increase their motivation to keep exercising independently, Qiu et al. decided to create a system that allows patients to comfortably play engaging exergames in the comfort of their own home while providing them with technical support and generating meaningful data to help clinicians track their improvement and make the necessary adjustments.

All users need is a computer to access the platform containing the games and a low-cost infrared camera, the Leap Motion Controller (LMC, Fig. 2.3), to track their movements. Using the LMC has many benefits: it is reliable and accurate, it does not restrain the user's movement in any way and does not need to be worn, and unlike other tracking devices (such as the Kinect, another popular choice in this field) it provides a ready-to-use finger tracking Software Development Kit (SDK). In addition to the camera, some of the patients participating in the study were also provided with a hip wedge, an arm support, or a forearm trough, to allow even those with severe impairments to use the system. The data collected by the LMC while the patient plays the games on their home computer are sent to a cloud-based system, which is responsible for elaborating and storing the data so that clinicians can access them whenever they need.

An interesting feature of this system are the algorithms that dynamically maintain each of the games within a certain range of difficulty, based on the data collected by the infrared camera, in order to always keep the user engaged, but not overly frustrated. Another peculiarity of this study is that it was truly conducted with a hands-off approach on the experts' side, only helping with the initial setup in each participant's home and then checking up on them on a weekly basis. However, patients could request assistance

Figure 2.3. The Leap Motion Controller

at any time through a button in the game and receive an immediate response, to minimize the drop in adherence to the study due to technical difficulties.

This study was successful in that patient participation remained consistent throughout its duration (three months in total), proving that a gamified exercise program with direct visual and audio feedback and with readily available technical support could be a great tool to keep people suffering from stroke motivated, improving their quality of life and their motor skills in the upper extremities overtime: the other positive outcome has been, in fact, the objective improvement of the patients, as seen by the better scores they received on the tests administered before and after the experiment. HoVRS still has some limitations (the more severely impaired patients cannot use it and it is not as immersive as other headset-based solutions), but it is definitely a notable step in the right direction.

HTC Vive VR A study with similar methods has been conducted just a year earlier [23]. Huang and Chen set out to evaluate the effectiveness of an immersive Virtual Reality system in upper limb rehabilitation by selecting 18 stroke patients from the Chung Shan Medical University Hospital in Taiwan and dividing them into two groups: the first (the control group) would perform 20 sessions of traditional physiotherapy over 8 weeks, while the second (the IVR group) would also have three 30-minute VR sessions per week. The VR game consisted of three different activities to make the user train their paretic limb (shooting balloons, shooting targets with a gun and making a circular device pass through an electrically charged rod without touching it) and it was administered through an HTC Vive headset plugged to a PC, while two HTC sensors were placed in the room to track the patient's movements (Fig. 2.4). In order to play the game, the patients had to use

the HTC controllers, meaning that their ability to flex their impaired arm's finger was not taken into account for this experiment.

Figure 2.4. Stroke patients using the HTC Vive to play an IVR game

The tests performed on patients in both groups after the intervention did, once again, indicate that VR games are effective in better improving patients' motor skills when combined with traditional physiotherapy; however, just as in the study mentioned above, the pool of available patients for this study was relatively small. The researchers also mentioned that the time since stroke onset for the patients was perhaps too long, potentially influencing the results, and also that in this experiment the VR training was not administered by itself but alongside traditional physical therapy, making it impossible to determine the standalone efficacy of the exergame.

Jintronix Nourouzi-Gheidari et al. [33] wanted to put another available system to the test when it comes to the physical therapy of stroke survivors, based on the evident need of a higher dosage of rehabilitation for stroke survivors than the amount they usually receive through traditional means. They recruited 18 patients to be divided in two groups: the control group would receive treatment as usual for 4 weeks, whereas the intervention group would add at least two VR session per week to the traditional physiotherapy for the same amount of weeks.

Each VR session lasted about 44 minutes, with half of that time dedicated to setting up, talking to the healthcare professional and taking short breaks. The patients were asked to sit in front of a computer display with a Kinect camera on top of it (Fig. 2.5) which is able to track the user's movements to interact with the exergame displayed on the screen without the need of controllers or any kind of wearable sensor, making the

experience less taxing for the patients.

Figure 2.5. The Microsoft Kinect camera

The exergame used for this study was the Jintronix system (Fig. 2.6), which has been explicitly designed to train the user's upper extremity movements through a series of minigames. Each task provides the user with direct audiovisual feedback and has a corresponding score, that can be monitored by a healthcare professional to track the patients' progress and to tailor each task's difficulty to their abilities and needs. There are many different dimensions that can be tweaked by the therapist to modify the levels' difficulty, such as precision, target size and speed. That is how this study has been conducted, although this system has the potential to be used by patients even without direct supervision, making it more versatile.

Figure 2.6. The desktop setup and some screenshots of the Jintronix rehabilitation system

By using the Jintronix system, patients in the intervention group were given the possibility to train their paretic limb against gravity for longer than those in the control group, although it should be noted that this system has no way of tracking hand movements,

meaning that the user would grab virtual objects in the game simply by moving their hand close to them.

In order to assess the results of the experiment, tests were administered to all participants right before the training period, right after it and 4 weeks after the last training session. The relevant recorded measures were feasibility, safety, and efficacy: the Jintronix system proved to be a feasible and safe solution for the physical rehabilitation of stroke patients, seeing as the participants seemed to respond positively to the technology and they experienced no adverse effects such as motion sickness or excessive fatigue. As for the efficacy measure, though, the results were inconclusive in the sense that the intervention group did show a greater level of improvement but it was not really statistically significant for most tests. These results could be due to the non-immersive nature of this system or to the relatively short amount of time the intervention group spent using the system. The researchers also recognized the small sample size of this study's participants and their lack of additional deficits other than physical impairments, meaning that the results of the study cannot be extended to all stroke survivors, but they did prove the feasibility and safety of such a solution to increase the amount of physical therapy stroke patients can receive while having fun.

MNVR-Rehab The solutions explored up to this point could only benefit mildly impaired patients; Mekbib et al. [8] set out to develop and test an immersive VR experience that would improve motor functions even in the most severe cases. Their VR rehabilitation system, called MNVR-Rehab (Mirroring Neuron VR Rehab), takes advantage of the role of the so-called mirror neurons in the recovery of motor function, as previous studies have shown that even just observing or imagining a certain activity could lead to an improvement over time. The game itself consists of sitting in front of a table with some tennis balls on it and having to grab said balls and place them inside a basket placed in the center of the table (Fig. 2.7A), with some adjustments that can be made by a clinician, such as modifying the height of the basket and the table, or placing a higher or lower number of tennis balls on the table. Patients who participated in the pilot experiment were instructed to sit on a chair (Fig. 2.7B) and were made to wear an HTC Vive headset for visual feedback, which would then be connected to a PC that would take care of the elaboration of data and processing required for the application to run. Patients did not need any kind of controller for the input, as an LMC camera (Fig. 2.3) was strapped to the headset itself, and two additional sensors were placed in front of the chair to track general movements of the patient.

The real innovation of this application lies in its UE (Upper Extremity) therapy mode selection: the clinician can choose whether the patient performs unilateral (grabbing the balls with one hand, Fig. 2.7C) or bilateral (grabbing the balls with both hands, Fig. 2.7D) UE therapy, and whether it should be mirrored or not. In the mirrored mode, a virtual limb parallel to the patient's more affected UE or both virtual arms will be displayed performing the ball grabbing exercise, depending on whether unilateral or bilateral training was chosen. This approach is meant for patients that are not able to move their affected arm at all, in order to facilitate their recovery through mirror neuron activation. Furthermore, if the patient is able to move their affected limb but has trouble

opening and closing their affected hand, the system will automatically attach the balls to their hand in an effortless way, so that they won't even notice it happening, making this application even more suitable for every conceivable degree of physical impairment.

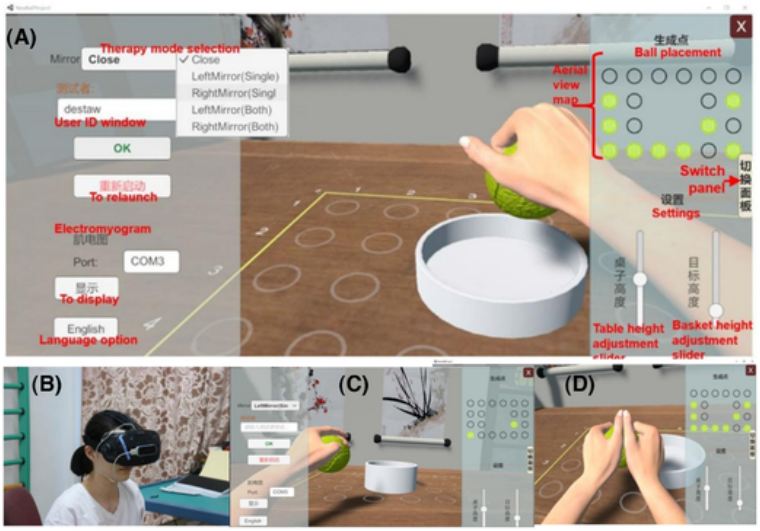


Figure 2.7. Some screenshots of the MNVR-Rehab system (A, C and D) and a picture of a patient wearing the HTC Vive headset (B)

The clinical study was conducted on 28 patients in the VR group, who received 1 hour of VR training with the MNVR-Rehab system and 1 hour of traditional physiotherapy per day, 4 days per week for 2 weeks, and 15 patients in the control group, who received the same amount of rehabilitation time-wise but consisting only of traditional physiotherapy. The results were satisfactory, seeing as of the patients that concluded the 2-week training period the ones in the VR group improved at a higher rate in comparison to those in the control group; however, the researchers recognize that the number of patients in the study was relatively small and that they did not administer any follow-up test to the participants, making the results less relevant overall. Moreover, MNVR-Rehab delivers on being task-oriented but lacks any kind of variety, consisting of a single, repetitive exercise that could become tedious for patients in the long run.

FarmDay Another study conducted in 2022 [3] aimed to address the lack of rigorous parameters in the existing literature for the creation of VR applications aimed at the rehabilitation of upper limb movement: Rojo et al. made a list of specific range of movement (ROM) targets for each involved joint, expressed in degrees, and then created a goal-based VR game, called FarmDay (Fig. 2.8), in which each task can be completed only if the user reaches those ROM targets.

The game contains six different activities that are designed to make the user perform movements and actions similar to activities of daily living (ADL); it is an example of immersive Virtual Reality, since it was made to be experienced through a headset, which

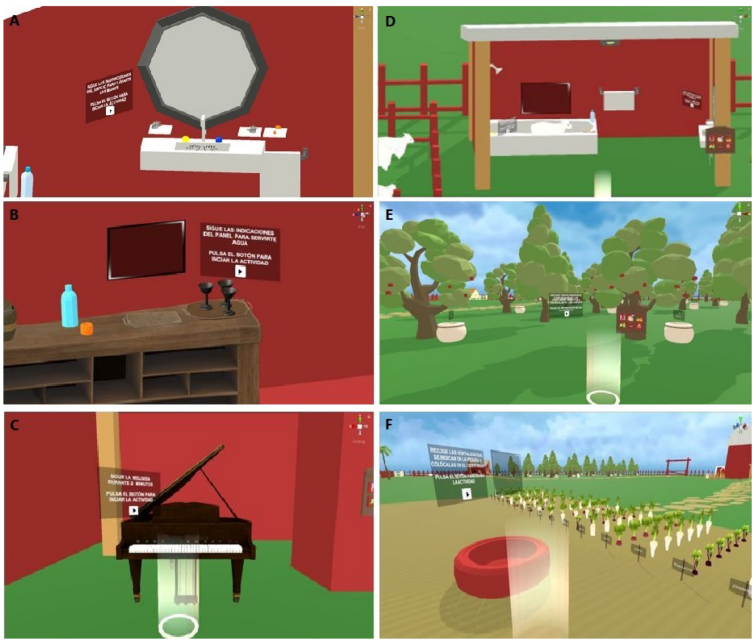


Figure 2.8. Some screenshots displaying the various activities in FarmDay

does not run the app natively but rather streams the data it receives from a powerful off-site server that is in charge of running FarmDay and incorporating the user input in a coherent manner.

The idea of determining a set of target positions and rotations of the arm to be met and then creating an ad hoc application based on those target positions is innovative and should be taken into consideration going forward. FarmDay has been tested by five physiotherapists specialized in neuromotor disorders to obtain an objective evaluation of its design and its usability: they agreed that the application was adequate for the most part, but they argued that some activities didn't feel realistic enough. Indeed, this software is meant to rehabilitate arm joints but does not take into account finger movements, since in order to play a pair of controllers (either the HTC Vive^{TM} or the Oculus Rift^{TM} controllers) must be used, greatly limiting the benefits when it comes to fine hand motions. Furthermore, the application has only been tried by professionals but hasn't been tested on actual patients, which is an issue that has also been mentioned by the researchers as well.

Humanized VR rehabilitation With the development of a new protocol and a preliminary study, Alves Marques et al. [19] tried to address the potential dehumanization patients could experienced with the automation of healthcare services associated with the use of VR solutions for physical rehabilitation. They selected ten patients with chronic stroke to undergo 15 sessions of VR therapy, each lasting between 30 and 45 minutes, with the presence and support of a healthcare professional for the entire duration of the

sessions. The VR game they played was the Harpy Game [13], and they wore a Myo armband (Fig. 2.9) to track their arm movements. In order to play the game, the patients would have to move their paretic arm in specific patterns, flexing and extending their elbow and rotating their shoulder.

Figure 2.9. The Myo armband

The patients were greeted by their caregivers and engaged in about 10 minutes of small talk before each session, promoting a relaxed and pleasant experience, after which they were made to sit on a chair in front of a television displaying the Harpy Game and to wear the Myo armband on their paretic arm, all the while conversing with the healthcare professional. The caregiver could finetune the difficulty of the game depending on each participant's level of physical impairment, and they could base this decision on whether the participant was able to complete each level in time and whether they felt happy and not too fatigued after therapy.

In this study, assessments to check the patients' ROM, spasticity, and quality of life were conducted right before and after each session: a comparison of the results led the researchers to the conclusion that a humanized approach to VR upper limb rehabilitation for stroke survivors is conducive to a greater improvement of ROM, a significant lowering of spasticity as well as social dysfunction, a decreased resistance to passive stretching and an overall improvement of the patients' quality of life. It should be noted, though, that the constant physical presence of a healthcare professional nullifies VR's benefit of being less expensive and more readily available; furthermore, the researchers noted the small sample size of this study, the lack of a control group to compare and contrast the results and the lack of a quantitative measure of the participants' emotions, leaving their interpretation to the healthcare professionals' subjectivity.

A protocol combining IVR and NIVR Ventura et al. devised a protocol [40] to improve sensorimotor and proprioception upper limb ability in patients with stroke by combining immersive VR exergames, administered through an HMD, and non-immersive VR activities, to be experienced through the Virtual Reality Rehabilitation System-Handbox

(Khymeia Group, Noventa Padovana, Italy) (Fig. 2.10). A prospective study based on this protocol should have 4 measurement times (pre-intervention, post-intervention, and two follow-ups at 6 and 12 months from the end of the intervention) and it should include two patient groups, the experimental group and the control group, each comprising of 30 patients. Both groups would follow a 4-weeks 3 days per week rehabilitation program, which would consist of traditional physiotherapy for the control group and VR rehabilitation (both immersive and non-immersive) for the experimental group.

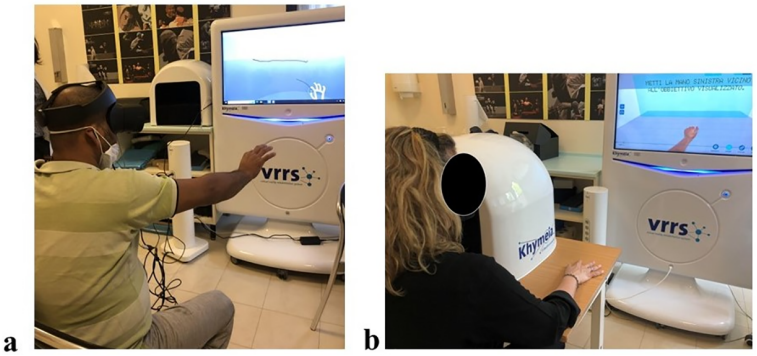


Figure 2.10. Patients in a VR session (a) and a 2D Handbox session (b)

VR sessions would go as follow: the patient would sit on a chair in front of the Handbox and would first play some exergames while wearing the HMD (such as playing with building blocks, positioning virtual objects in certain spots and coloring some pictures) and then they would remove the HMD and execute different exergames displayed on the screen in front of them. Their upper extremity movements would be captured by a Leap Motion Controller, and the therapist could modify the difficulty of each exercise by modifying the sensitivity of the kinematic sensors to make sure that each patient can play the games regardless of their level of motor impairment.

It should be noted that this is just a protocol that has not been actualized in any study yet; the researchers recognize that the recruitment of so many patients could be difficult and the duration of the experiment could lead to a high dropout rate. Furthermore, although the combination of IVR and NIVR is a novelty that could lead to a higher sense of variety and thus decrease boredom, this is an expensive and cumbersome solution that could not be implemented outside of a clinical setting, limiting its potential.

RAPAEL Smart Glove In a study conducted in Japan [16] Ase et al. divided 14 outpatients suffering from the effects of chronic stroke in two randomly selected groups and instructed one of them (the intervention group) to perform at-home rehabilitation exercises with the help of a VR device (see Fig. 2.11) and the other one (the control group) to follow a traditional rehabilitation program without the use of VR for the same amount of time.

This study was based on the importance of repetitive and task-specific practice for the functional recovery of upper limb motion, with the idea of making rehabilitation more

Figure 2.11. The RAPAEL Smart Glove^{TM}; NEOFECT Co., Yung-in, Korea

cost-effective and fun for patients by gamifying the whole process. The Smart Glove kept track of the range of movement of the user's affected arm and hand while the patient performed tasks related to daily life (such as squeezing an orange or pouring wine in a glass), with direct feedback in the form of an animated 2D scene displayed on a computer screen in front of them. An interesting feature of this study is that the various tasks to be performed were adjusted depending on the patients' current range of motion, tailoring the experience to their individual needs and making sure that the exercises were neither too easy nor too taxing.

The assessments performed on the patients before and after the training period were promising: the intervention group's improvement surpassed that of the control group and the patients that engaged in VR rehabilitation in addition to traditional therapy ended up using their paretic hand in day-to-day life much more than those who didn't, suggesting that repetitive and task-oriented exercises administered through VR can indeed be a valid option to ameliorate the upper-limb movement and quality of life of people who suffered a stroke. However, the researchers involved in this study stated that this type of rehabilitation is not suited for everyone, as it's only recommended in cases of mild paralysis, and also that more studies need to be done on what the ideal amount of time should be allotted weekly for VR exercises in order to obtain the maximum results. Furthermore, the fact that this type of VR is not immersive might lead to the users getting bored, and the necessity to use the Smart Glove in order to track the affected arm's movements makes this option less scalable and more expensive for the patients.

The following table outlines all of the articles explored in this chapter with each one's technologies, positive and negative aspects.

Table 2.1: Summary of the cited articles

Ogun et al. (2019)	Immersive VR, HTC Vive HMD for visual feedback, LMC camera to track the user's arm and finger movements	Immersive, accurate tracking, easy to use	No customization, fairly tiresome exercises
Rogers et al. (2019)	Non-immersive VR with Elements system	Engaging despite being non-immersive, intuitive, customizable	Very expensive and cumbersome
Qiu et al. (2020)	Non-immersive VR, LMC camera for tracking and computer screen for visual feedback, optional supports for accessibility	Algorithm-based difficulty adjustments, readily available technical support, cloud-based data collection	Lack of immersion, not ideal for severely impaired patients
Huang and Chen (2020)	Immersive VR, HTC Vive HMD for visual feedback, HTC Vive tracking stations for the user's location, PC to run the application	Immersive, fairly easy to use	No customization options provided, only suitable for mild cases, no finger tracking
Nourazi-Gheidari et al. (2020)	Non-immersive VR with a computer screen for visual feedback and Kinect camera to track the user's movements	Customizable, low-cost, easy to use	Lack of immersion, still requires healthcare professional supervision, no tracking of the finger movements
Mekbib et al. (2021)	Immersive VR, HTC Vive HMD for visual feedback, HTC Vive tracking stations for the user's location, LMC camera to track the user's arm movements, PC to run the application	Highly customizable, beneficial even for severely impaired patients	Cumbersome setup, might lead to boredom

	Methodology	Pros	Cons
Rojo et al. (2022)	Headset-based immersive VR with a cloud server host and controllers for tracking	Immersiveness, rigorously defined set of ROM targets	Lack of realism for some tasks due to the controllers, lack of testing with patients
Alves Marques et al. (2022)	Non-immersive VR, Myo armband to track the user's movements	User-centered, humanized protocol, easy to use	Lack of immersion, requires the constant presence of a therapist, no tracking of the finger movements
Ventura et al. (2024)	Immersive VR with HMD and non-immersive VR with the Virtual Reality Rehabilitation System-Handbox, LMC camera to track the user's arm and finger movements	Customizable, engaging	Expensive, not applicable outside of a clinical setting, lack of testing with patients
Ase et al. (2025)	Non-immersive VR, Smart Glove for tracking and computer screen for feedback	Customization, lack of adverse effects	Low immersion, recommended only for cases of mild paralysis

2.3 Differences between cited works and this project

All of the solutions explored in the previous section are valid recent innovations that explore VR's potential to enhance traditional physiotherapy's benefits when it comes to rehabilitating people that have suffered a stroke. However, most of them involve various forms non-immersive VR, which has yielded mostly underwhelming results due to its inherent inability to engage the user past a certain threshold; as for the cited examples of immersive VR, some of them lacked the customization options needed to tailor the training experience to each patient's needs, and they all required multiple different hardware in addition to the HMD in order to work, such as cameras or hand-held controllers to keep track of the user's movements and PCs to actually run the applications themselves. Furthermore, even though the option of utilizing these technologies in patient's own homes has been cited, in reality all of them have been used strictly in clinical settings and with healthcare professionals' constant help and supervision.

The application that was developed for this thesis is different from all the cited works in that it is meant to be experienced through a headset, making it an immersive and engrossing VR experience, without the need of any external sensors or even controllers,

since the SDK leveraged in this project makes it possible to track the user's hand movements through the HMD's own built-in sensors. Moreover, it is not necessary for the headset to be tethered to a computer because it contains a build of the application and is able to run it, as well as being able to keep track of all of the users' data and progress without needing access to a remote cloud server. All of the aforementioned details make this solution portable, easy to set up and use, and relatively low-cost in comparison to what is commercially available at this time. Lastly, this program can be used by mildly cognitively impaired patients with little to no supervision, therefore even though the experiment with the patients has been conducted in a clinical setting for convenience's sake this application does have the potential to be used anywhere, making it a valid option to keep training one's paretic limb even after the official rehabilitation period has ended to maintain and perhaps further any and all improvements.

Chapter 3

Game Design

3.1 The idea

The aim of this thesis was to create a serious game to be played with a VR headset either in a clinical setting, alongside a traditional physiotherapy program to amplify its benefits, or in the comfort of one's own home, to motivate the user to keep using their paretic limb while having fun. The choice to develop a VR application for a headset was made specifically to make this solution affordable, portable, and easy to use, ideally with little to no external intervention; furthermore, VR has been shown to be effective for the improvement of neuroplasticity and motor recovery due to its ability to be uniquely immersive through visual, audio, and even haptic feedback [18]. The application needed to be as lightweight as possible and with relatively low-poly scenes, so that the game could be played through a build installed on the headset without needing any external computing power or cloud server connection. The name chosen for this application was "ReFlex VR" to combine a prefix that implies repetition, recovery and rehabilitation to a reference to flexibility, movement and motor function (as well as, of course, the "VR" to indicate that the application is a Virtual Reality experience): ReFlex VR is a virtual rehabilitation journey focused on restoring reflexes, flexibility, and functional movement.

Once the medium and the name were established, the next step was to choose an appropriate setting for the application. Previous studies have shown that familiar environments and scenarios that mimic real life can increase acceptance and user experience [10, 14, 32], therefore that is the direction that was taken: ReFlex VR would be a goal-oriented exergame with multiple levels set in a home environment, each level consisting of an activity of daily life requiring the use of one or both upper limbs. This familiar setting could hopefully counteract any potential uneasiness or discomfort experienced by the users due to the technology, while keeping them engaged and motivating them to use their paretic arm more even in their day-to-day life. Moreover, since stroke does not only affect one's motor functions but also their cognitive abilities [30], another dimension to the application was added by having an "easy mode" that focuses on the motor rehabilitation by guiding the user through each level step by step, and a "hard mode" that requires the user to remember the steps by themselves to complete the levels without any help.

3.2 Target users and functional requirements

ReFlex VR is aimed at people who have suffered a stroke or any kind of traumatic brain injury that led to the impairment of one of their arms. However, physical impairment must not be so severe as to prevent them from lifting their arms or putting the headset on (unless external help is available) and the application is not recommended for people with cognitive disorders, severe aphasia (which would make it impossible to understand the levels' objectives), or any state of confusion defined by temporal and/or spatial disorientation, which would only be exacerbated by Virtual Reality.

Given the general idea and the target users, the functional requirements that were identified are:

- **Hand tracking**: the application must be able to detect and track the user's hand movements without any controllers or sensors external to the headset itself, in order to determine whether the user is performing the different tasks correctly not only as far as arm movements go, but also for fine hand gestures as well.

- **Interactions with objects**: the user must be able to grab, move and release virtual objects present in the scene using different hand movements, thus enhancing their physical rehabilitation.

- **Rehabilitation-like exercises**: the application must offer a series of exercises that stimulate UE physical rehabilitation through different hand and arm movements inspired by actual physiotherapy exercises, focusing on repetition and task-specificity while still being more enjoyable for the patients than traditional physiotherapy.

- **Visual and audio feedback**: the application must give users visual feedback (in the form of arrows indicating the targets or particle system being played) and audio feedback (specific sound effects) both to indicate to them if they have completed a certain task or correctly executed an arm/hand movement, but also to admonish them for example if they are using their non-paretic limb to play the game. These kinds of direct feedback are fundamental to maintain patients' frustration low and to keep them engaged, giving them a sense of accomplishment.

- **Detection of correct poses**: the application must be able to detect whether the movement performed by the user falls within the parameters considered correct for the exercise (for example their arm will be considered extended only if their hand exceeds a certain distance threshold from the headset, or their hand will be considered opened/closed if the finger parameters detected by the headset are in a certain configuration).

- **Level or exercise advancement**: the user must be able to access different levels or exercises with varying difficulties to aid their improvement, and there must be a points system to give them a tangible sense of progression.

3.3 Technologies used

ReFlex VR was created using the Unity game engine (version 6000.0.30f1) and the C# programming language through the Visual Studio Editor; some of the 3D models were made in Blender, while others were downloaded from sites such as Sketchfab and TurboSquid. Both Unity and Blender were chosen for their performance, customization, ease of use, and lack of restrictions from a licensing standpoint, since they're open source software. Furthermore, Unity comes with many useful SDKs (Software Development Kits) which allow for faster and more modular software development for many different platforms: in particular, the Meta XR SDK was crucial for the development of this application, as it served as the basis for all of the interactions of the user with the virtual environment.

ReFlex VR was developed specifically to run on the Meta Quest 3 (Fig. 3.1), though it should theoretically work on many other headsets with no issues, such as the older Meta Quest models, the Apple Vision Pro and the HTC Vive.

Figure 3.1. The Meta Quest 3 headset and controllers

Some animations for the virtual assistants were downloaded from the Mixamo website, and the audio tips in Italian and English were generated in ElevenLabs. Lastly, all of the royalty-free sound effects present in the application were downloaded from pixabay.

3.4 Gameplay

ReFlex VR can be experienced either standing up or sitting on a chair, with the only caveat being that this choice has to be made before starting the game and it has to be maintained throughout the whole game session, since the user's height in-game is calibrated based on the HMD's position relative to the ground as the game boots up.

At the start of the game the player goes through a "menu scene", where they can either create a new user, select a user created in a previous game session from a list, and even modify some parameters related to their specific user that will influence the gameplay. After that, they can choose one among three different levels (we will call them "kitchen level", "coffee level" and "living room level") and depending on which one they choose a different scene will load.

All three levels require the player to use their virtual hands to complete a series of tasks through grabbing, pushing, pulling and rotating motions to train their affected UE. After they complete all of the tasks in order, a UI (User Interface) element will pop up in front of them with some stars (going from a minimum of zero to a maximum of five) to tell them how well they did, after which they can choose to play another level or quit the game. If, for any reason, the player wants to go back to the menu scene or to quit the game before finishing the level, they can push a button, present in all three levels, which will make a UI show up to give them the option to do so (Fig. 3.2).

Figure 3.2. The exit button in the kitchen level and the menu that pops up when the button is pressed

In the kitchen level, the player's objective is to slice up a carrot and cook it on the stove: in order to do that, they will need to open up a drawer, grab and place a pan on the stove, cut the carrot with a knife, pick up each carrot piece and place it inside the pan, turn on the stove by twisting a knob, pick up a spoon to stir the carrot pieces as they cook, and lastly turn off the stove with the same twisting motion (Fig. 3.3).

In the coffee level, the player spawns in front of an automated coffee machine and the objective is to make some coffee. They will be able to achieve that by pulling up

Figure 3.3. Some tasks to be performed in the kitchen level: slicing the carrot, placing the carrot pieces inside the pan, turning on the stove and stirring with a spoon

the coffee machine lever, placing a coffee capsule in the allotted slot, pulling the lever back down, picking a saucer from a cupboard above the coffee machine and placing it underneath the dispenser, picking a cup from the same cupboard and placing it on the saucer, and finally pressing one of the buttons on the machine to make the coffee start flowing (Fig. 3.4).

Lastly, in the living room level the player is tasked to clean up their living room. In order to do so, they will firstly spray some spots on a window with a cleaning detergent to then scrub them off with a sponge, then they will have to use a vacuum cleaner (turning it on and handling it properly) to remove some dust from the floor, and lastly they will have to pick up some paper balls and origami from the desk and throw them in a paper bin (Fig. 3.5).

3.5 UI design

The UI elements throughout the whole game (Fig. 3.6) are based on the Meta Horizon OS UI Set available in the Meta XR Interaction SDK for Unity, to ensure visual uniformity and to make sure that they work well with the rig that has been set up for the user's interactions with the virtual environment. The player can interact with the UI buttons in

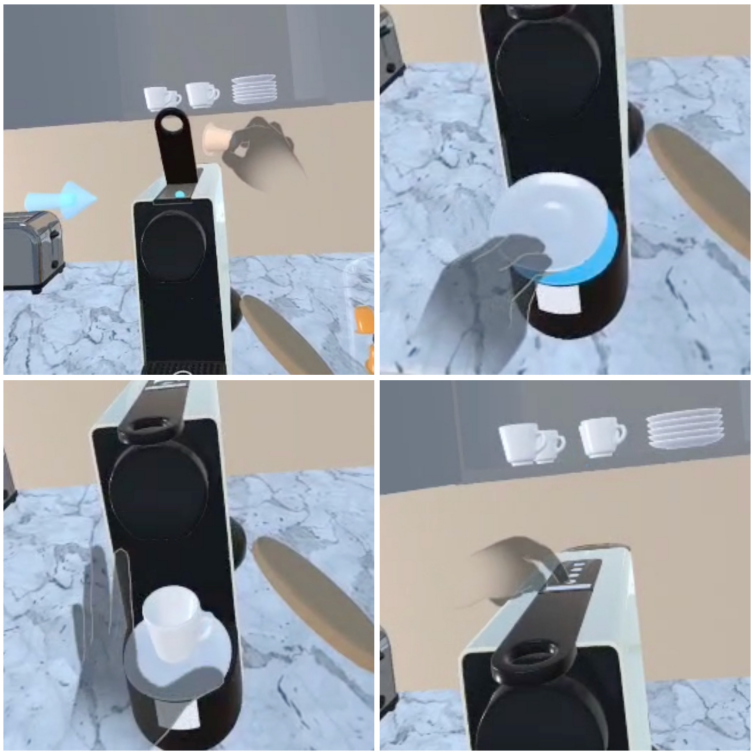


Figure 3.4. Some tasks to be performed in the coffee level: placing a coffee capsule inside the machine, placing a saucer under the dispenser, placing a cup on top of the saucer and pressing a button on the machine

several ways: they can use the controllers or the virtual hands to click them at a distance, or they can push the buttons with a virtual finger like they intuitively would a physical button in the real world.

There are also diegetic interfaces in all three of the levels, in the form of a bulletin board displaying a list of the different tasks the player is required to perform in order to complete the level, with a square box next to each one that gets dynamically ticked off as the player goes through them, to give them a sense of progression and to give them the possibility to remind themselves what they are supposed to do next at a glance. A diegetic interface is an integral part of the game world, blending seamlessly with its surroundings and creating a more prominent sense of immersion, whereas non-diegetic interfaces are what can be usually thought of as UI, since they are graphic overlays that are useful for the player but that the character in game is not aware of.

If it can safely be said that the bulletin boards are an example of a diegetic interface, the definition for all of the other UI elements in (Fig. 3.6) is a little more nebulous, since a VR game is experienced from a first-person perspective, effectively nullifying the difference in perspective between the "character" and the player, whereas in most other types of games there is a third-person point of view, making the distinction between what

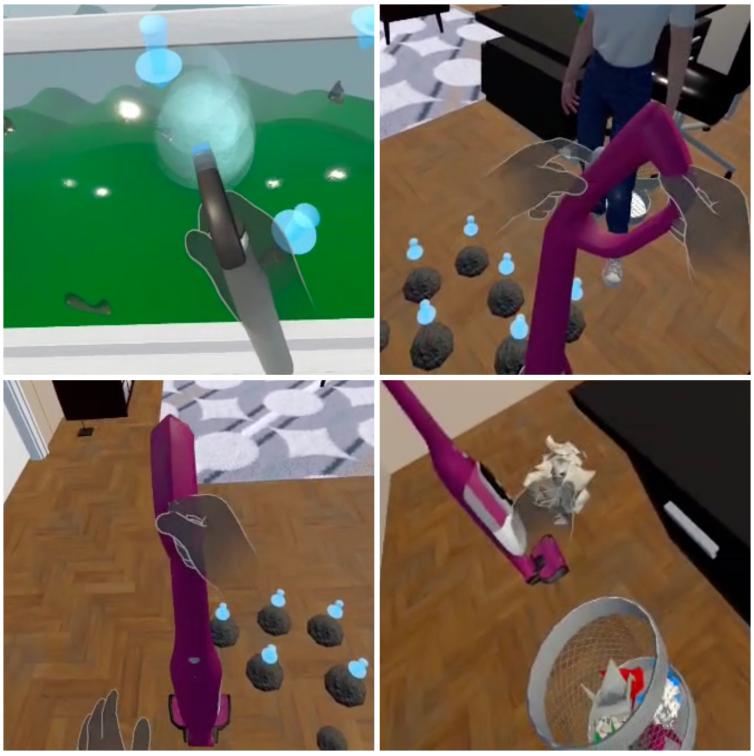


Figure 3.5. Some tasks to be performed in the living room level: spraying the spots on the window, turning on the vacuum cleaner, moving the vacuum around to remove the dust and throwing the paper balls in the bin

the character and what the player can see much clearer.

3.6 Game Environment

As previously mentioned, the setting chosen for this application is the interiors of a common house (mainly a kitchen and a living room), except for the initial menu scene where the environment is simple and essential in order to make the player concentrate on the menu itself. Most of the assets used to bring the game's visuals to life have been downloaded from the Unity Asset Store, but some of them have been imported to Blender and tweaked with the Decimate modifier to lower their polygon count and thus make it possible to play the game through a build downloaded in the headset without keeping it tethered to a PC. For the same reason, all of the lights in the scene have been set to "Baked" and lightmaps have been created, making the application even less resource-intensive. Some of the assets have been modeled on Blender when a viable option was not available in the Asset Store, while trying to keep their look consistent with the rest of the virtual environment. Although the overall polygon count is relatively low and the lighting is not real-time, the application still manages to achieve a realistic look and feel,

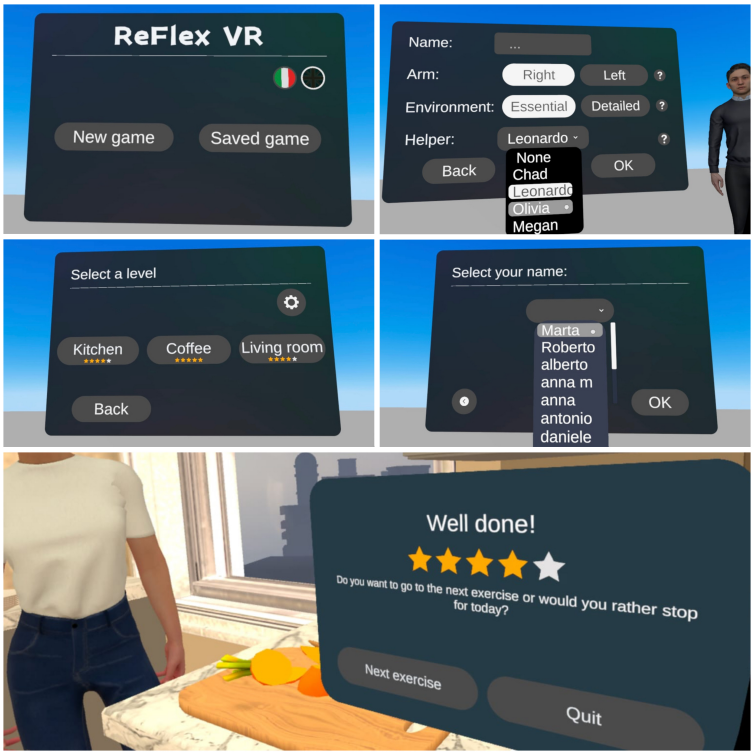


Figure 3.6. Some of ReFlex VR's UI elements and menus

making the whole experience for the players that much more immersive and enjoyable.

In a study conducted with neurorehabilitation experts to design the ideal VR environment for stroke patients [44] the importance of low-cognitively demanding virtual environments for patients with reduced cognitive abilities has been underlined, therefore in the user creation menu of this application there is an "Environment" section that gives the user the option to choose between a more essential look, with less superfluous objects that could potentially distract them from the main objective, or a more detailed room to engage them and keep up visual interest if the patient can handle it.

3.7 Sound Design

The aim for ReFlex VR's sound design was to make the user feel immersed while not distracting or confusing them: this is why no backing soundtrack was added to the game, making it possible for the user to still be somewhat cognizant of their surroundings while playing the game. All of the UI buttons in the application make a sound (borrowed from the Meta SDK Toolkit) when pressed, which is not conducive to the gameplay per se but it adds some nice polish to the application.

ReFlex VR is filled with various SFX to add to the realism: for example, when slicing

the carrot with the knife one among six different cutting sounds will be played, when turning on and off the stove the user will hear the gas coming out and spark sounds, when pressing the coffee machine's button a liquid pouring SFX will accompany the visuals of the coffee being poured in the cup, and when turning on the vacuum the familiar sound of a vacuum cleaner will fill the virtual living room. Some of the added SFX are used, in addition to increasing immersion, to indicate in an intuitive way whether the player is performing the tasks correctly: for instance, the vacuum's SFX will start playing only if the player has pushed the power button, and if they have not (thus the vacuum is not turned on) they will not be able to remove the dust from the floor. Adding this kind of auditory information to aid the player is fundamental in VR games, where the user cannot use their sense of touch having to rely on sight and hearing; furthermore, stroke patients typically already have an impaired sense of touch in their paretic limb to begin with, making audio cues even more essential for them.

There are two more audio-related cues to help the player navigate each level: the "success sound" and the helper tips. The success sound is a brief bell chime SFX that plays as soon as the user completes one of the tasks, making it immediately clear to them that they can move on to the next one; the helper tips are clear instructions to expand on what is written on the bulletin board, both delivered in written (as a "speech bubble" on top of the helper) and audio form, either in Italian or in English (the language can be chosen in the initial menu). The helper also admonishes the player if they use their healthy arm to complete tasks and congratulates them when they finish a level, hopefully making the application feel more lively and less intimidating.

Chapter 4

Game development

4.1 User-Game Interactions

As previously mentioned, ReFlex VR can be played in its entirety without the use of any controllers thanks to Meta's Interaction SDK, which is able to obtain certain data related to the user's real hands (such as their relative position and size) through the headset's built-in cameras and sensors and use such data to project a realistic and dynamic model representing the player's hands in the virtual world; not only that, but the SDK allows users to seamlessly and naturalistically interact with virtual objects in various different ways. The hand joints defined by Meta, shown in Fig. 4.1, are captured using algorithms based on machine learning and computer vision techniques, and that data is then available to be used by the SDK's various built-in scripts or by the developers themselves to make ad-hoc solutions when necessary.

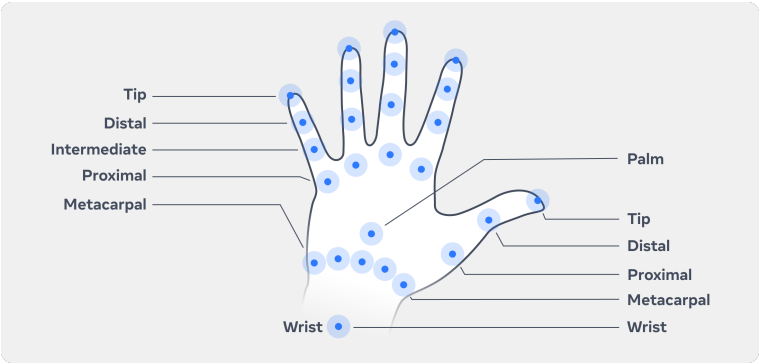


Figure 4.1. The hand joints as defined by Meta to be captured and elaborated through their SDKs

However, there are some limitations to this type of tracking that may arise due to, for example, occlusions (which is why the patients were asked at the beginning of the experiment to always wear short-sleeved tops or to have their sleeves rolled up before each training session) or the hands being out of the HMD's field of view (which actually

was not an issue for this application in particular because patients were naturally inclined to look at their own hands while completing the different tasks). This type of tracking is also obviously less precise than that obtained through wearable sensors, but it has been deemed accurate enough for the interactions designed for the application.

To track the user's head and hand movements, as well as to allow hand interactions, the Meta Interaction SDK's "OVR Interaction Rig" has been used in all of the scenes, which has the "OVR Camera Rig" and "OVR Manager" scripts already configured and various children GameObjects containing the scripts in charge of the tracking (mainly "OVR Hand" and "OVR Skeleton") and those that enable various types of interactions ("Poke Interactor", "Hand Grab Interactor" and "Hand Grab Use Interactor"). The main difference from the standard OVR Interaction Rig is the "Tracking Origin Type" variable in the "OVR Manager" script, which for this application has been set to "Eye level" to make it possible to play it either sitting down or standing up: with the "Eye level" option the user's height relative to the virtual world is given by the rig's height relative to the virtual ground, regardless of the user's real-world position, as opposed to the "Floor level" mode and the "Stage" mode which base the rig's position in the virtual world on the user's position relative to the actual floor, making the player's position in the game vary depending on whether they are sitting down or standing up.

The virtual object's counterparts to the "Interactor"-type scripts are the "Interactables", which are necessary to actually make the interactions between the user and the virtual world occur: for example, all of the UI elements that are placed close to the user in ReFlex VR (such as the victory screen at the end of each level), as well as the buttons on the coffee machine and the one on the vacuum cleaner, have a "Poke Interactable" component that makes it possible to push the buttons with an index finger; all of the GameObjects that the player can pick up with their hands have the "Hand Grab Interactable" script attached to them; and the spray bottle in the living room level contains a "Hand Grab Use Interactable" component, which makes it possible to not only grab the GameObject but to then make said GameObject do something by performing a specific action while holding it (in this case, curling one's index finger while holding the spray bottle triggers a particle system and a sound effect simulating the action of spraying). Other notable scripts that have been borrowed from Meta's SDKs are the "Snap Interactable"/"Snap Interactor" (to make it easier for the patients to, for example, place the pan on the stove in the kitchen level or put a coffee capsule in the machine's slot in the coffee level), the "One Grab Translate Transformer" (to simulate the movement of drawers or cabinet doors) and the "One Grab Rotate Transformer" (to make it possible to rotate the stove's knob in the kitchen level and the coffee machine's lever in the coffee level).

4.1.1 User-Game Interactions: Kitchen level

In the kitchen level, the player must open a drawer in front of them to grab a pan and place that on the stove; they then are required to place a carrot on the cutting board and grab a knife from the same drawer to slice up the carrot. The slicing logic is contained in its entirety in the "SlicingScript" component attached to the knife GameObject: basically, the script checks at a fixed interval whether the knife's edge has come in contact with

the carrot using a raycast (Listing 4.1) and, if it has, it triggers a coroutine to actually separate the carrot mesh in two pieces (Listing 4.2).

```
1  void FixedUpdate()
2  {
3      bool hasHit = Physics.Linecast(startSlicePoint.position, endSlicePoint.
       position, out RaycastHit hit, sliceableLayer);
4
5      if (hasHit)
6      {
7          if (currentTarget != hit.transform.gameObject)
8          {
9              currentTarget = hit.transform.gameObject;
10
11             if (slicingCoroutine == null)
12             {
13                 slicingCoroutine = StartCoroutine(DelayedSlice(currentTarget, hit)
       );
14             }
15         }
16     }
17     else
18     {
19         currentTarget = null;
20
21         if (slicingCoroutine != null)
22         {
23             StopCoroutine(slicingCoroutine);
24             slicingCoroutine = null;
25         }
26     }
27 }
```

Listing 4.1. A snippet of the "SlicingScript" script: the variables involved and the FixedUpdate method

The slicing part has been implemented through a coroutine to introduce a slight delay between the knife touching the carrot and the cut happening in order to simulate some level of resistance and thus making the experience feel more realistic: the knife's edge needs to stay in contact with the carrot's surface for a certain amount of time (established through the `delay` variable) in order for the cut to happen, which happens in line 15 in Listing 4.2 when the `Slice` method is called on the carrot, and the current number of carrot pieces (saved in the `pieces` variable) is increased. If the number of pieces cut exceeds 15 it means that the patient has successfully completed the slicing task, therefore the following events happen: the success sound plays, the checkbox next to the corresponding task on the bulletin board gets ticked off, the text above the helper gets updated and the next audio tip is played through the `PlayAudioWithDelay` coroutine (in this case a coroutine has been introduced to avoid any overlap between the success sfx and the audio tip). In this if statement another method is called, `TallyPoints`, but it will be explored in more detail in the next section.

```
1  IEnumerator DelayedSlice(GameObject target, RaycastHit hitInfo)
2  {
3      float delay = 0.01f;
4      float elapsedTime = 0.0f;
5
6      while (elapsedTime < delay)
7      {
```

```
8           if (!Physics.Linecast(startSlicePoint.position, endSlicePoint.position,
        out RaycastHit newHit, sliceableLayer) || newHit.transform.gameObject !=
        target)
9           {
10              yield break;
11          }
12          elapsedTime += Time.fixedDeltaTime;
13          yield return new WaitForFixedUpdate();
14      }
15      Slice(target);
16      pieces++;
17      if (!slicingInProgress)
18      {
19          sliceStartTime = Time.time;
20          slicingInProgress = true;
21      }
22      if (pieces >= 15 && !hasPlayed)
23      {
24          sliceEndTime = Time.time;
25          success.Play();
26          checkbox.sprite = checkedBox;
27          task02Text.SetActive(false);
28          task03Text.SetActive(true);
29          TallyPoints();
30          StartCoroutine(PlayAudioWithDelay(1f));
31          hasPlayed = true;
32      }
33      slicingCoroutine = null;
34  }
35
36  IEnumerator PlayAudioWithDelay(float delay)
37  {
38      yield return new WaitForSeconds(delay);
39      localizedAudio?.PlayLocalizedClip(3);
40  }
41
42  public void Slice(GameObject target)
43  {
44      Vector3 velocity = velocityEstimator.GetVelocityEstimate();
45      Vector3 planeNormal = Vector3.Cross(endSlicePoint.position - startSlicePoint.
        position, velocity);
46      planeNormal.Normalize();
47      SlicedHull hull = target.Slice(endSlicePoint.position, planeNormal);
48
49      if (hull != null)
50      {
51          HandGrabInteractor grabbingHand = null;
52          HandGrabInteractable originalInteractable = target.GetComponent<
        HandGrabInteractable>();
53          if (originalInteractable != null)
54          {
55              grabbingHand = originalInteractable.Interactors.FirstOrDefault();
56          }
57
58          GameObject upperHull = hull.CreateUpperHull(target, crossSectionMaterial);
59          SetUpSlicedComponent(upperHull);
60          upperHull.layer = target.layer;
61
62          GameObject lowerHull = hull.CreateLowerHull(target, crossSectionMaterial);
63          SetUpSlicedComponent(lowerHull);
64          lowerHull.layer = target.layer;
65
66          if (grabbingHand != null)
67              grabbingHand.ForceRelease();
68
69          Destroy(target);
70          if (slicingSounds.Length > 0 && audioSource != null)
```

```
71          {
72              PlayRandomAudio();
73          }
74          else
75          {
76              Debug.LogWarning("AudioClips or AudioSource is missing!");
77          }
78      }
79  }
```

Listing 4.2. A snippet of the "SlicingScript" script: the DelayedSlice, PlayAudioWithDelay and Slice methods

In the `Slice` method, the two new carrot pieces resulting from the cut are determined by calculating the normal with the knife's edge and its velocity at the time of the cut and then creating a SlicedHull variable, called `hull`, by calling the `Slice` method from the "SlicedHull" script (an open source script downloaded from the ezy-slice repository from DavidArayan on GitHub). If the resulting hull is not null, the HandGrabInteractable and HandGrabInteractor involved in the grab (if the carrot was grabbed while being cut) are acquired in two variables to make it possible to force release the grab before destroying the original carrot piece and creating the two new pieces in order to avoid any possible issues arising from an interaction with a GameObject that is not in the scene anymore. The two new pieces are called `upperHull` and `lowerHull`: they are spawned respectively through the `CreateUpperHull` and `CreateLowerHull` methods (from the "SlicedHull" script) and they are then set up by calling the `SetUpSlicedComponent` method on them (Listing 4.3); lastly, their layer is set to be the same as the original piece's layer in order to make it possible to cut them as well. To further add on to the realism, the `Slice` method also calls the `PlayRandomAudio` method each time a cut happens, which picks a random number between 0 and the length of the `slicingSounds` array (through the `Random.Range` method) and uses that number as an index to randomly select and play one of the audio clips in `slicingSounds`, which, as the name suggests, contains six different slicing sfx.

```
1  public void SetUpSlicedComponent(GameObject slicedObject)
2  {
3      Rigidbody rb = slicedObject.AddComponent<Rigidbody>();
4      rb.linearVelocity = Vector3.zero;
5      rb.angularVelocity = Vector3.zero;
6      rb.Sleep();
7      rb.mass = 0.1f;
8      rb.linearDamping = 1f;
9      rb.angularDamping = 1f;
10     rb.collisionDetectionMode = CollisionDetectionMode.Continuous;
11
12     MeshCollider col = slicedObject.AddComponent<MeshCollider>();
13     col.convex = true;
14
15     Grabbable grab = slicedObject.AddComponent<Grabbable>();
16     grab.enabled = true;
17     var gfield = typeof(Grabbable).GetField("_rigidbody", System.Reflection.
       BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance);
18     if (gfield != null)
19     {
20         gfield.SetValue(grab, rb);
21     }
22
23     HandGrabInteractable handGrab = slicedObject.AddComponent<HandGrabInteractable
       >();
```

```
24      handGrab.Initialize(rb, grab);
25
26
27      slicedObject.tag = "Carrot";
28      slicedObject.layer = LayerMask.NameToLayer("CarrotPieces");
29
30      RespawnOnDrop resp = slicedObject.AddComponent<RespawnOnDrop>();
31      var field = typeof(RespawnOnDrop).GetField("_yThresholdForRespawn", System.
        Reflection.BindingFlags.NonPublic |
32          System.Reflection.BindingFlags.Instance);
33      if (field != null)
34      {
35          field.SetValue(resp, 0.3f);
36      }
37      else
38      {
39          Debug.LogWarning("Failed to set _yThresholdForRespawn via reflection.");
40      }
41  }
```

Listing 4.3. A snippet of the "SlicingScript" script: the SetUpSlicedComponent method

The `SetUpSlicedComponent` method is in charge of adding all of the relevant components to the newly spawned carrot pieces, namely a Rigidbody to make them responsive to gravity and any other physics forces (lines 4-9 of Listing 4.3 have been added to limit any glitches or unwanted "explosions" after a cut as much as possible), a convex MeshCollider to make them collide with other GameObjects in a realistic manner, Grabbable and HandGrabInteractable components to make it possible for the user to grab them and a RespawnOnDrop component (courtesy of the Meta Interaction SDK) to make the carrot pieces respawn on the counter in the event of the player accidentally dropping them on the floor, where it would become difficult to pick them up.

After completing the slicing task, the player must pick up the carrot pieces one by one and place them on the pan: in order to keep track of the number of carrot pieces currently inside of it, check whether the player is using the correct hand to execute the task, and eventually trigger the completion logic and update the points, an empty child GameObject of the pan with a trigger box collider has been created and the "CarrotsInPanObjective" script has been attached to it, with `OnTriggerEnter` and `OnTriggerExit` methods to keep track of the carrot pieces (Listing 4.4) and `CheckHandednessAndUpdatePoints` and `TallyPoints` methods to handle the handedness and points logic (which, again, will be explored in the next section). In this case, the win condition corresponds to placing 10 carrot pieces in the pan.

```
1   private void OnTriggerEnter(Collider other)
2   {
3       if (other.CompareTag("Carrot"))
4       {
5           if (!taskInProgress && numPieces == 0)
6           {
7               taskStartTime = Time.time;
8               taskInProgress = true;
9           }
10          numPieces++;
11          CheckHandednessAndUpdatePoints(other.gameObject);
12          if ((numPieces >= 10) && !hasPlayed)
13          {
14              taskEndTime = Time.time;
15              success.Play();
16              checkbox.sprite = checkedBox;
```

```
            oilArrow.SetActive(true);
            task03Text.SetActive(false);
            task04Text.SetActive(true);
            TallyPoints();
            StartCoroutine(PlayAudioWithDelay(1f));
            hasPlayed = true;
        }
    }
}

private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Carrot"))
        numPieces--;
}
```

Listing 4.4. A snippet of the "CarrotsInPanObjective" script: the OnTriggerEnter and OnTriggerExit methods

After that, the player must turn on the stove by twisting the knob; both the logic to handle this task and that pertaining the last task of the level, which is to turn the stove back off, can be found in the StoveFireHandler script (Listing 4.5). Its behaviour hinges on calculating the knob's rotation at every frame and comparing it to a certain target angle (defined through the `targetAngle` variable): if the knob's angle exceeds said target angle and the stove has not been turned on yet (this condition is ensured by the `hasPlayedSuccess` boolean variable), then the task completion logic is triggered and, if they were not already playing before, the stove effects (some particle effects simulating a small fire and an sfx audio clip) start playing. On the other hand, if the knob's angle is below the target angle the particle systems and the audio clip are stopped, and if all of the other tasks before the last one have been completed the level victory logic is triggered, because it means that the player has turned the stove off after stirring the carrots, thus effectively completing the level.

```
void Update()
{
    currentAngle = transform.localEulerAngles.x;
    currentAngle = NormalizeAngle(currentAngle);

    if (currentAngle > targetAngle)
    {
        if (!hasPlayedSuccess)
        {
            turningOn = true;
            success.Play();
            drawerArrow.SetActive(true);
            checkbox.sprite = checkedBox;
            task04Text.SetActive(false);
            task05Text.SetActive(true);
            StartCoroutine(PlayAudioWithDelay(1f, 5));
            hasPlayedSuccess = true;
            spoonOutline.SetActive(true);
            if (currentAngle < 90f)
                PointsManager.Instance.Points += 2;
            else
                PointsManager.Instance.Points += 1;
            scoreGrabThing.DoYourThing(spoon, spoonHDComponent, spoon.transform);
        }

        if (!isPlaying)
        {
            fire.Play();
```

```
29              fireADD.Play();
30              fireAlpha.Play();
31              fireGlow.Play();
32              stoveSound.Play();
33              fireLight.SetActive(true);
34              isPlaying = true;
35          }
36      }
37      else
38      {
39          if (isPlaying)
40          {
41              fire.Stop();
42              fireADD.Stop();
43              fireAlpha.Stop();
44              fireGlow.Stop();
45              stoveSound.Pause();
46              fireLight.SetActive(false);
47              isPlaying = false;
48              if (hasPlayedSuccess && !hasPlayedEnd && stirringCarrots.
    hasStirredCarrots())
49              {
50                  turningOn = false;
51                  success.Play();
52                  checkbox02.sprite = checkedBox;
53                  task06Text.SetActive(false);
54                  task07Text.SetActive(true);
55                  hasPlayedEnd = true;
56                  if (currentAngle == initialAngle)
57                      PointsManager.Instance.Points += 2;
58                  else
59                      PointsManager.Instance.Points += 1;
60                  PointsManager.Instance.SaveCurrentUserPoints();
61                  winCanvas.SetActive(true);
62                  StartCoroutine(PlayAudioWithDelay(1f, 7));
63              }
64          }
65      }
66  }
67
68  float NormalizeAngle(float angle)
69  {
70      if (angle > 180f) angle -= 360f;
71      return angle;
72  }
```

Listing 4.5. A snippet of the "StoveFireHandler" script: the Update and NormalizeAngle methods

Between the two tasks regarding the stove, the only task left to discuss involves grabbing a wooden spoon from the counter and stirring the carrots for a set amount of time: all of the game logic tied to this task resides in the "StirringCarrotsObjective" script attached to the same trigger collider inside the pan that has been used to detect the carrot pieces. In this script, as shown by Listing 4.6, the `OnTriggerEnter` and `OnTriggerExit` methods are used to change the value of a boolean variable, `isInside`, which is then continuously checked in the `Update` method: as long as the spoon is inside of the trigger collider a timer variable is incremented by adding `Time.deltaTime` to it, and when said variable reaches a certain value (set in `timerThreshold`) the success logic is triggered, with the `hasPlayed` variable used to allow said logic to fire off only once. In this script there is also a first reference to the "SimpleHandDetection" script, which will be talked about in detail in the following section.

```
1  private void Update()
2  {
3      if (isInside)
4      {
5          timer += Time.deltaTime;
6          if ((timer >= timerThreshold) && !hasPlayed)
7          {
8              success.Play();
9              checkbox.sprite = checkedBox;
10             task05Text.SetActive(false);
11             task06Text.SetActive(true);
12             StartCoroutine(PlayAudioWithDelay(1f));
13             hasPlayed = true;
14             if (simpleHandDetection.hasUsedWrongHand)
15                 PointsManager.Instance.Points += 1;
16             else
17                 PointsManager.Instance.Points += 2;
18         }
19     }
20 }
21
22 private void OnTriggerEnter(Collider other)
23 {
24     if (other.CompareTag(targetTag))
25     {
26         isInside = true;
27         simpleHandDetection = other.GetComponent<SimpleHandDetection>();
28     }
29 }
30
31 private void OnTriggerExit(Collider other)
32 {
33     if (other.CompareTag(targetTag))
34     {
35         isInside = false;
36     }
37 }
```

Listing 4.6. A snippet of the "StirringCarrotsObjective" script: the Update, OnTriggerEnter and OnTriggerExit methods

4.1.2 User-Game Interactions: Coffee level

As for the coffee level, the first task to be completed involves grabbing the coffee machine's lever and rotating it upwards; there is then a specular task, as is the case for the stove in the kitchen level, that requires pulling the lever back down once the player has correctly positioned the coffee capsule inside the machine. The logic for both of these is contained in the "LeverRotationDetection" script attached to the lever itself. Basically, in the `Start` method of this script the lever's initial rotation is saved in a variable aptly named `initialRotation`, which then is used in the `Update` method to check whether the lever's rotation has reached a certain threshold (Listing 4.7, line 3). The following if-else tree in the `Update` method has the function of checking whether the player is using their paretic limb to rotate the lever, and if not an audio clip is played to remind them to try to use their affected arm as much as possible and the `hasUsedWrongHand` boolean is set to true, which will influence the points tallying later on (see sections 4.2 and 4.3 for more details).

```
1  void Update()
2  {
```

```
    float angleDifference = Quaternion.Angle(initialRotation, transform.
    localRotation);
    if (_interactable != null)
    {
        var interactors = _interactable.Interactors;
        if (interactors.Count > 0)
        {
            _interactor = interactors.FirstOrDefault();
            if ((_interactor != null) && _interactor.IsGrabbing)
            {
                IHand hand = _interactor.Hand;
                if (hand != null)
                {
                    if (PointsManager.Instance.Arm == "Right")
                    {
                        if (hand.Handedness == Handedness.Left)
                        {
                            if (!localizedAudio.IsClipPlaying())
                                localizedAudio?.PlayLocalizedClip(7);
                            hasUsedWrongHand = true;
                        }
                    }
                    else
                    {
                        if (hand.Handedness == Handedness.Right)
                        {
                            if (!localizedAudio.IsClipPlaying())
                                localizedAudio?.PlayLocalizedClip(7);
                            hasUsedWrongHand = true;
                        }
                    }
                }
            }
        }

        if (!hasTriggered && (Mathf.Abs(angleDifference - targetAngle) <=
    angleThreshold) && (interactors.Count == 0))
        {
            OnLeverFullyRotated();
            hasTriggered = true;
        }

        if (capsuleSnapEventsHandler.hasPlayed && hasTriggered && (angleDifference
     <= 91) && !hasPulledDownLever && (interactors.Count == 0))
        {
            OnLeverPulledDown();
            hasPulledDownLever = true;
        }
    }
}
```

Listing 4.7. A snippet of the "LeverRotationDetection" script: the Update method

Finally, the last two ifs in the `Update` method serve the purpose of triggering the win condition respectively for the first task of the level and for the third one (Listing 4.8). The `StartRotation` method is never called inside of the script itself, but it fires off as soon as the lever is grabbed by the user thanks to the "Interactable Unity Event Wrapper" component (courtesy of the Meta Interaction SDK) attached to the lever: this component accepts an "Interactable View" variable that is the "interactable" component that needs to be checked (in this case the lever's Hand Grab Interactable component) and it has a series of events that can be raised when said interactable is hovered on by an interactor, is selected and so on. For this specific case, an element has been

added to the "When Select ()" list of the component, which makes it possible to call `LeverRotationDetection.StartRotation` as soon as the lever is grabbed, setting the initial time of the rotation (which will be then used to calculate the user's points).

```
void OnLeverFullyRotated()
{
    success.Play();
    checkbox01.sprite = checkedBox;
    outline.SetActive(false);
    capsuleOutline.SetActive(true);
    elapsedTime = Time.time - startTime;
    CalculatePoints();
    armExtensionDetector.SetTimer(20f, true);
    lidHGI.enabled = true;
    StartCoroutine(PlayAudioWithDelay(1f, 1));
}

void OnLeverPulledDown()
{
    success.Play();
    checkbox02.sprite = checkedBox;
    cupboardArrow.SetActive(true);
    plateOutline.SetActive(true);
    CalculatePoints();
    armExtensionDetector.SetTimer(20f, true);
    StartCoroutine(PlayAudioWithDelay(1f, 3));
}

public void StartRotation()
{
    startTime = Time.time;
}
```

Listing 4.8. A snippet of the "LeverRotationDetection" script: the OnLeverFullyRotated, OnLeverPulledDown and StartRotation methods

Aside from pulling the lever up and down, in this level the player is also required to grab a coffee capsule from a jar and place it in the coffee machine, grab a saucer from a cupboard in front of them and place it under the coffee dispenser, and take a cup from the same cupboard and put it on the saucer. All three of these actions involve a *snap interaction*: the capsule snaps to the coffee machine, the saucer snaps to the dispenser and the cup snaps to the small plate. However, if the win condition were to be triggered by the snap interaction itself, it would fire off as soon as each GameObject reaches the "snap area" due to how snap interactions have been built by Meta, but this solution would not be ideal because if, for any reason, the user moves the interactable into that area and then away without ever releasing it, thus with no snap actually occurring, the task would still be considered successfully completed. This is why the "SnapEventsHandler" script was created: it is a script that makes sure that the win condition logic is triggered only after the player releases the interactable in the interactor's area, and it works for all three snap interactions mentioned (Listing 4.9). The `Update` method simply checks at every frame whether the interactable is in the interactor's area, starting and stopping a coroutine accordingly, whereas the `WaitUntilReleasedWhileSnapped` method contains all of the relevant logic: the while loop keep tracks of what happens as long as the interactable is being held, breaking out of the coroutine if the object is moved away from the snap interactor while still being held; then, the if statement after the while loop checks whether the interactable has snapped to the interactor after being released, and if so it triggers

the "snap condition" which is different depending on the specific GameObject that has been snapped (the condition is based on the GameObjects' tags).

```
void Update()
{
    if (_snapInteractor == null) return;

    var current = _snapInteractor.Interactable;
    if (current != _lastInteractable)
    {
        if (current != null)
        {
            if (waitForReleaseCoroutine != null)
                StopCoroutine(waitForReleaseCoroutine);
            waitForReleaseCoroutine = StartCoroutine(WaitUntilReleasedWhileSnapped
    ());
        }
        else
        {
            if (waitForReleaseCoroutine != null)
            {
                StopCoroutine(waitForReleaseCoroutine);
                waitForReleaseCoroutine = null;
            }
        }
        _lastInteractable = current;
    }
}

IEnumerator WaitUntilReleasedWhileSnapped()
{
    while (_interactable.Interactors.Count > 0)
    {
        if (_snapInteractor.Interactable == null)
        {
            yield break;
        }
        yield return null;
    }

    if (_snapInteractor.Interactable != null && !hasPlayed)
    {
        hasPlayed = true;
        machineArrow.SetActive(false);

        if (gameObject.CompareTag("CoffeeCapsule"))
            leverRotationDetection.StartRotation();
        else if (gameObject.CompareTag("Plate"))
            OnPlatePositioned();
        else if (gameObject.CompareTag("Cup"))
            OnCupPositioned();
    }
}
```

Listing 4.9. A snippet of the "SnapEventsHandler" script: the Update and WaitUntilReleasedWhileSnapped methods

After correctly placing all three objects, the player must press one of the buttons on the coffee machine (through a *poke interaction*, made possible by the "Poke Interactable" component attached to the buttons) to make it begin dispensing coffee inside of the cup. In order to obtain this behavior, an "Interactable Unity Event Wrapper" component was added to the buttons (Fig. 4.2), with a list of items in the "When Select ()" event to activate a particle system simulating coffee flowing from the dispenser into the

cup, play a coffee machine sfx, make the victory sound play and call a helper function, `FinishCoffeeLevel`, from the "CoffeeMachineHelperFunctions" script, which contains all of the end-of-level logic. This event also deactivates two GameObjects called "ButtonsOutline" and "ArrowButtons", but these will be discussed in more detail in section 4.3.

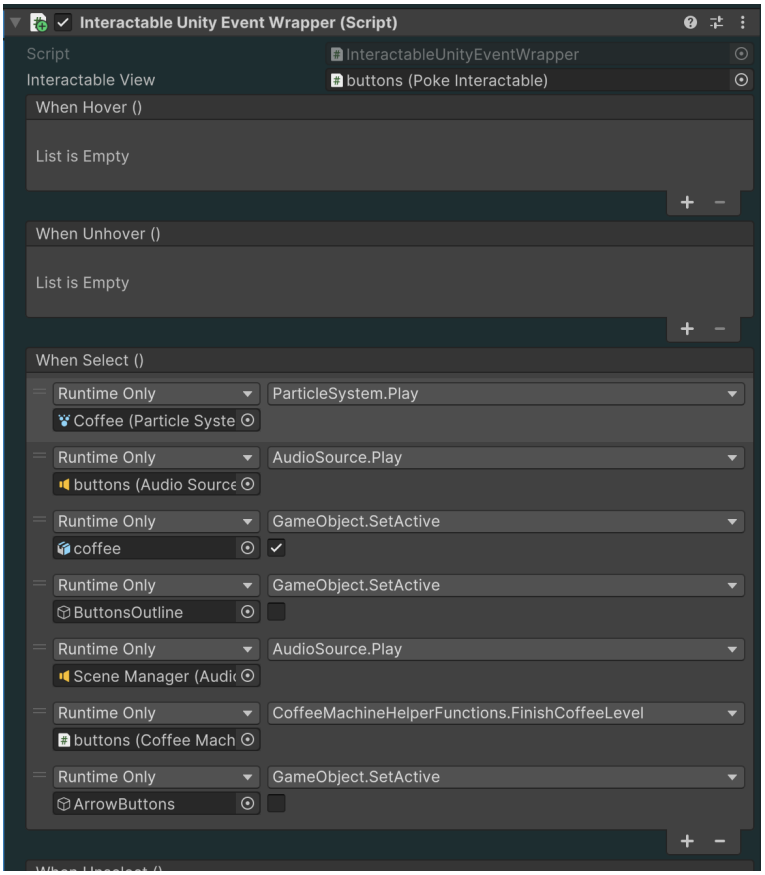


Figure 4.2. The coffee machine buttons' Interactable Unity Event Wrapper component in the Unity inspector

4.1.3 User-Game Interactions: Living room level

Lastly, in the living room level the player spawns in front of a window with some spots on it that need to be scrubbed off, and their first objective is to grab a spray bottle on the windowsill and spray all of them down, requiring a more complicated movement that combines grabbing with curling the index finger. As previously mentioned, this mechanic was made possible through the Hand Grab Use Interactable and a script for the particle system and sfx logic, "WaterSpray", that was provided in a demo scene from the Meta Interactable SDK. In order to keep track of which spots on the window have been hit by

the spray and which haven't yet, two scripts have been created, "SprayCleanerHandler" and "DirtOnGlassHandler", that work in synergy to bring this first task to life.

"SprayCleanerHandler" has been added to the spray bottle's particle system: it contains a private integer to keep track of the number of spots that have been hit by the spray and an `OnParticleCollision` method (Listing 4.10) that, in the event of a collision, checks whether the object the particle system has collided with is a spot on the window, if that is the case it then checks if that particular spot had already been sprayed before, and if not it increments the script's internal variable to count the sprayed spots (`spotsHit`) and it does two more things before checking `spotsHit`'s current value to possibly trigger the win condition logic. First of all, it disables any non-trigger collider on the hit spot: this is due to the fact that it has been necessary to add on each spot two separate colliders, a trigger and a non-trigger one, because `OnParticleCollision` is only able to detect collisions with non-trigger collider, whereas because of the way it has been set up the sponge's collisions later on would only work properly with trigger colliders.

```
1  void Start()
2  {
3      localizedAudio = FindAnyObjectByType<PlayLocalizedAudio>();
4      cnu = GetComponent<CanvasNumberUpdater>();
5  }
6
7  void OnParticleCollision(GameObject other)
8  {
9      if (other.CompareTag("Spot"))
10     {
11         dogh = other.GetComponent<DirtOnGlassHandler>();
12         if (!dogh.hasBeenHit)
13         {
14             dogh.hasBeenHit = true;
15             spotsHit++;
16             BoxCollider[] colliders = other.GetComponents<BoxCollider>();
17             foreach (BoxCollider col in colliders)
18             {
19                 if (!col.isTrigger)
20                 {
21                     col.enabled = false;
22                 }
23             }
24             cnu.currentTally++;
25             cnu.UpdateTallyOnClipboard();
26         }
27         if ((spotsHit == 10) && !hasPlayed)
28         {
29             SprayCleaningCompleted();
30         }
31     }
32 }
```

Listing 4.10. A snippet of the "SprayCleanerHandler" script: the Start and OnParticleCollision methods

The other event that happens in this if statement is shown in lines 24 and 25, and it involves another script, "CanvasNumberUpdater": in order to enhance the player's sense of progression and to make what they need to do even clearer to them, in this level the bullet board contains not only a list of tasks, but also some numbers next to each task that requires them to show how many steps involved in said task have already been completed. Listing 4.11 shows the script's code and Fig. 4.3 shows an example

of the numbers on the board being updated to depict that the player has gone from 0 to 4 spots sprayed. What "SprayCleanerHandler" does is update "CanvasNumberUpdater"'s `currentTally` variable and call the `UpdateTallyOnClipboard` method, which modifies the number contained in the TMP_Text component through the use of a regular expression.

```
public class CanvasNumberUpdater : MonoBehaviour
{
    public TMPro.TMP_Text canvasText;
    [HideInInspector]
    public int currentTally = 0;

    public void UpdateTallyOnClipboard()
    {
        canvasText.text = System.Text.RegularExpressions.Regex.Replace(canvasText.
    text, @"(\()([^\:]+:\s*)\d+/\d+(\))",
        match => match.Groups[1].Value + match.Groups[2].Value + currentTally + "
    /10" + match.Groups[3].Value);
    }
}
```

Listing 4.11. The "CanvasNumberUpdater" script

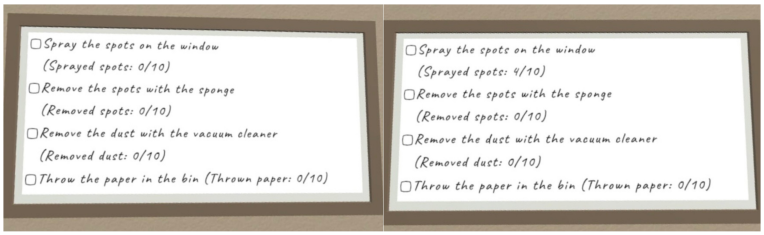


Figure 4.3. Two screenshots from the game showing the numbers on the board dynamically changing

"DirtOnGlassHandler" is a script that is attached to all of the spots on the window and its role does not end in relation to the spray objective, but it extends to the following task of scrubbing off all of the spots with a sponge. The logic relating to the first task is in the `Update` method (Listing 4.12): the variable `hasBeenHit` is set to true by the "SprayCleanerHandler" script once the particle system comes in contact with that specific spot; the first if statement sets the `allSpotsSprayed` variable to true (which will be discussed again later on) once the particle system has reached the win condition, therefore having sprayed all of the spots on the window, while the second if-then-else statement will be explored in section 4.3.

```
void Start()
{
    originalScale = transform.localScale;
    rsowo = FindAnyObjectByType<RemoveSpotsOnWindowObjective>();
}

void Update()
{
    if (sch1.hasPlayed || sch2.hasPlayed)
        allSpotsSprayed = true;
```

```
    if (hasBeenHit && arrowStillActive)
    {
        arrow.SetActive(false);
        arrowStillActive = false;
        timer = 0f;
    }
    else if (hasBeenHit && !arrowStillActive && allSpotsSprayed)
    {
        timer += Time.deltaTime;
        if (timer >= 10f)
        {
            arrow.SetActive(true);
            timer = 0f;
        }
    }

    if (numOfRubs > 3)
    {
        rsowo.dirtSpotsRemoved++;
        gameObject.SetActive(false);
        rsowo.UpdateTallyOnClipboard();
    }
}
```

Listing 4.12. A snippet of the "DirtOnGlassHandler" script: the Start and Update methods

The last if statement, as well as the `OnTriggerEnter` and `PlayRandomAudio` methods (Listing 4.13), have to do with the second task instead: the `Update` method checks at every frame whether the specific spot the script is attached to has come in contact with the sponge at least three times, in which case a variable in the "RemoveSpotsOnWindowObjective" attached to the sponge is increased, the spot in question is removed from the scene, and the removed spots tally on the board is updated accordingly; as for the `OnTriggerEnter` method, it first checks if the spot is colliding with the sponge and whether it has been previously hit with the spray (otherwise the sponge will not be able to scrub the spot off), and if so it randomly plays one of six possible sponge rubbing sfx (with the `PlayRandomAudio` method), it increases the variable that keeps track of how many times the spot has been rubbed with the sponge, and it progressively reduces the spot's scale by one third each time it gets rubbed.

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Sponge") && hasBeenHit)
    {
        timer = 0f;
        arrow.SetActive(false);

        if (numOfRubs == 0)
        {
            spongeAudioSource = other.gameObject.GetComponent<AudioSource>();
        }
        if (spongeRubSounds.Length > 0 && spongeAudioSource != null)
        {
            PlayRandomAudio();
        }
        numOfRubs++;

        float shrinkFactor = Mathf.Clamp01(1f - (numOfRubs / 3f));
        transform.localScale = originalScale * shrinkFactor;
    }

}
```

```
23
24 public void PlayRandomAudio()
25 {
26     int randomIndex = Random.Range(0, spongeRubSounds.Length);
27     spongeAudioSource.clip = spongeRubSounds[randomIndex];
28     spongeAudioSource.Play();
29 }
```

Listing 4.13. A snippet of the "DirtOnGlassHandler" script: the OnTriggerEnter and PlayRandomAudio methods

Another script is attached to the sponge, "RemoveSpotsOnWindowObjective", which is in charge of the overall task logic: it contains the `dirtSpotsRemoved` integer that is increased by "DirtOnGlassHandler" every time that a spot collides with the sponge for at least three times, while its `Update` method checks whether that variable has reached 10 and, if so, it triggers the task completion logic. This time, that logic contains, among the activating and deactivating of certain hints, certain audio clips being played and points being updated, a method from another script, called "ScreenFader", that has the function to move the player's rig to another spot. That is necessary because in the living room level the player will find themselves in three different spots in the virtual room to complete the various tasks (in front of a window, facing the room and sitting at a desk), unlike the previous two levels where the rig is stationary in one place throughout. In order to achieve this, three empty GameObjects have been added to the scene that function as "teleport anchors" and they have been placed where the rig is supposed to teleport; the "ScreenFader" simply modifies the rig's coordinates by copying one of the three anchors' coordinates and it adds a fade in-fade out effect through the use of a coroutine and a black canvas, to make the sudden change of scenery smoother and hopefully less disorienting for the patients.

As for the vacuum cleaning task, there are once more two different scripts that work together to achieve the desired game logic: first of all, each ball of dust on the ground has a trigger collider and the "VacuumCleaningHandler" attached to it, which mainly manages the arrow logic (see section 4.3 for more details), progressively diminishes the dust ball's scale as it comes in contact with the vacuum cleaner, with the same logic as the spots on the window, in an `OnTriggerEnter` method, and when it collides with the vacuum cleaner more than three times the GameObject is removed and the `UpdateDustRemovedVariable` method from the "VacuumCleanerHandler", which is the script attached to the vacuum cleaner, is called. This method increases a variable, called `dustRemoved`, that is necessary for the script to keep track of the number of dust balls vacuumed and to eventually trigger the task completion logic, and then it calls the `UpdateTallyOnClipboard` method from the "CanvasNumberUpdater" script to dynamically update the number displayed on the board.

Another method of note from this script is `TurnVacuumOnOrOff` (Listing 4.14): in the first part of the task the player is required to turn the vacuum on by pressing a button on it, therefore a button GameObject was added as a child to the vacuum GameObject and all of the relevant components to turn it into a poke interactable were added to it, as well as an Interactable Unity Event Wrapper component.

```
1 public void TurnVacuumOnOrOff()
2 {
```

```
3       if (!isOn)
4       {
5           audioSource.PlayOneShot(vacuumStartSfx);
6           if (!hasPlayedHint)
7           {
8               hasPlayedHint = true;
9               StartCoroutine(PlayAudioWithDelay(1f, 3));
10              vacuumText.SetActive(false);
11              vacuumCleanText.SetActive(true);
12              foreach (GameObject arrow in dustArrows)
13              {
14                  arrow.SetActive(true);
15              }
16          }
17          isOn = true;
18      }
19      else
20      {
21          if (audioSource.isPlaying)
22              audioSource.Stop();
23          audioSource.PlayOneShot(vacuumEndSfx);
24          isOn = false;
25      }
26  }
```

Listing 4.14. A snippet of the "VacuumCleanerHandler" script: the TurnVacuumOnOrOff method

In this component, in the "When Select ()" event, other than the hints regarding the button being deactivated a call to `TurnVacuumOnOrOff` was added for multiple reasons: first of all, three different audio clips simulating the sound of a vacuum cleaner (`vacuumStartSfx`, `vacuumSfx`, and `vacuumEndSfx`) were added to the script as variables to be played in different occasions, specifically `vacuumStartSfx` when the button is pressed to turn the vacuum on, `vacuumEndSfx` when the button is pressed again to turn the vacuum off, and `vacuumSfx` as long as the vacuum is turned on (Listing 4.15, lines 3-17); secondly, it signals to the "VacuumCleaningHandler" script whether the vacuum is turned on, thus making it possible to remove the dust (through a call to the `VacuumIsOn` method, Listing 4.15, that returns the current value of the `isOn` boolean); and lastly, it manages both the visual and audio hints regarding this specific task.

```
1   void Update()
2   {
3       if (isOn)
4       {
5           isPlaying = true;
6           if (!audioSource.isPlaying)
7               audioSource.PlayOneShot(vacuumSfx);
8       }
9       else
10      {
11          if (isPlaying)
12          {
13              if (audioSource.isPlaying)
14                  audioSource.Stop();
15              isPlaying = false;
16          }
17      }
18
19      if ((dustRemoved == 10) && !hasPlayed)
20      {
21          hasPlayed = true;
22          victorySound.Play();
```

```
23          checkbox.sprite = checkedBox;
24          foreach (GameObject arrow in paperArrows)
25          {
26              arrow.SetActive(true);
27          }
28          TurnVacuumOnOrOff();
29          vacuumText.SetActive(false);
30          vacuumCleanText.SetActive(false);
31          paperText.SetActive(true);
32          StartCoroutine(PlayAudioWithDelay(1f, 4));
33          vacuumHGI.enabled = false;
34          MoveHelper(helper.transform, newHelperPosition.transform);
35          MoveHelper(lc.GetClone().transform, newClonePosition.transform);
36          screenFader.FadeToBlackAndTeleport(playerRig.transform, deskPosition.
    transform);
37          if (vacuumSHD.hasUsedWrongHand)
38              PointsManager.Instance.ThirdLevelPoints += 1;
39          else
40              PointsManager.Instance.ThirdLevelPoints += 2;
41      }
42  }
43
44  public bool VacuumIsOn()
45  {
46      return isOn;
47  }
48
49  public void UpdateDustRemovedVariable()
50  {
51      dustRemoved++;
52      cnu.currentTally++;
53      cnu.UpdateTallyOnClipboard();
54  }
```

Listing 4.15. A snippet of the "VacuumCleanerHandler" script: the Update, VacuumIsOn and UpdateDustRemovedVariable methods

In addition to dealing with the sfx logic, the `Update` method also contains the task completion logic, by checking whether all of the dust balls have been removed (through the `dustRemoved` variable) and, if so, it removes and activates the relevant hints, it checks the checkbocks relative to the vacuum cleaning task, it plays the victory sound and it updates the player's points. Furthermore, it moves the player's rig to the desk position in the same way it was shifted to the vacuum cleaner's position and it also moves the virtual helper to be in front of the desk with the `MoveHelper` method, which simply takes two Transforms and copies the second one's position and rotation into the first one ("newHelperPosition" and "newClonePosition" are two more empty GameObjects that were added to the scene for this specific purpose).

The last task in the living room level involves picking up 10 individual paper balls or origami placed on the desk in various different spots and throwing them into a bin next to the player: all of the logic regarding this task is encapsulated in the "PaperInBinObjective" script attached to the bin GameObject. From a usability standpoint, it was important to make sure that the bin would be placed on the player's side closer to their paretic limb, in order to further entice them to use that one to complete the task, which is why in the script's Start method (Listing 4.16, lines 3-12) an if-else statement was added to check the patient's "Arm" variable and position the bin GameObject accordingly.

```
1   void Start()
2   {
```

```
3       if ((PointsManager.Instance != null) && (PointsManager.Instance.Arm == "Right"
        ))
4       {
5           transform.position = binPositionRight.transform.position;
6           transform.rotation = binPositionRight.transform.rotation;
7       }
8       else if ((PointsManager.Instance != null) && (PointsManager.Instance.Arm == "
        Left"))
9       {
10          transform.position = binPositionLeft.transform.position;
11          transform.rotation = binPositionLeft.transform.rotation;
12      }
13
14      audioSource = GetComponent<AudioSource>();
15      localizedAudio = FindAnyObjectByType<PlayLocalizedAudio>();
16      cnu = GetComponent<CanvasNumberUpdater>();
17  }
```

Listing 4.16. A snippet of the "PaperInBinObjective" script: the variables and the Start method

Furthermore, in order to keep track of the number of objects correctly thrown in the bin a trigger collider encompassing its entire volume was added to it and two methods were added to the script, `OnTriggerEnter` and `OnTriggerExit` (Listing 4.17). The first particular to note is that the only GameObjects that are considered for this trigger collisions are ones tagged as "Paper": this is to avoid any unwanted behavior, for example the user accidentally entering the trigger collider with their virtual hand. The first if statement in the `OnTriggerEnter` method and the if statement in the `OnTriggerExit` method could be considered specular, seeing as what is done in the first one gets undone by the second one: in the `OnTriggerEnter` the paper ball is considered to have entered the bin, therefore the variable that internally keeps track of the number of thrown paper balls, `piecesInBin`, is incremented, as well as the number displayed on the board (lines 6-7), whereas if an `OnTriggerExit` call is triggered it means that a paper ball has entered and then exited the bin, which is why both `piecesInBin` and the tally on the board are decremented accordingly. The addition of the `OnTriggerExit` to this script was motivated by a need to cover all possible cases, even those where the ball falls on the edge of the bin, technically entering the trigger collider, but then it drops outside of the bin. The first if statement of the `OnTriggerEnter` method also updates the user's points according to whether they have used the correct hand to throw away the paper ball or not (lines 10-14) and it deactivates the paper ball's "RespawnOnDrop" script: this was necessary because the paper ball GameObjects, much like every other GameObject in ReFlex VR, have the "RespawnOnDrop" component attached to them to make sure that the patients will be able to grab them again even if they dropped them by mistake, but obviously this behavior is not desirable for this task, since the paper balls that fall inside of the bin need to stay there; the `OnTriggerExit` method, on the other hand, reactivates that script so that the paper balls that have mistakenly fallen out of the bin can respawn on the desk. `OnTriggerEnter`'s second if statement contains the win condition logic, that is called only if all of the 10 paper balls have been correctly positioned in the bin.

```
1   private void OnTriggerEnter(Collider other)
2   {
3       if (other.gameObject.CompareTag("Paper"))
4       {
```

```
5           piecesInBin++;
6           cnu.currentTally++;
7           cnu.UpdateTallyOnClipboard();
8           rod = other.gameObject.GetComponent<RespawnOnDrop>();
9           rod.enabled = false;
10          pbh = other.gameObject.GetComponent<PaperBallHandler>();
11          if (pbh.HasUsedWrongHand())
12              PointsManager.Instance.ThirdLevelPoints += 1;
13          else
14              PointsManager.Instance.ThirdLevelPoints += 2;
15      }
16      if ((piecesInBin == 10) && !hasPlayed)
17      {
18          hasPlayed = true;
19          victorySound.Play();
20          checkbox.sprite = checkedBox;
21          paperText.SetActive(false);
22          victoryText.SetActive(true);
23          StartCoroutine(PlayAudioWithDelay(1f));
24          victoryCanvas.SetActive(true);
25          PointsManager.Instance.SaveCurrentUserPoints();
26      }
27  }
28
29  private void OnTriggerExit(Collider other)
30  {
31      if (other.gameObject.CompareTag("Paper"))
32      {
33          cnu.currentTally--;
34          cnu.UpdateTallyOnClipboard();
35          piecesInBin--;
36          rod = other.gameObject.GetComponent<RespawnOnDrop>();
37          rod.enabled = true;
38      }
39  }
```

Listing 4.17. A snippet of the "PaperInBinObjective" script: the OnTriggerEnter and OnTriggerExit methods

4.2 Points and Star System

4.2.1 The PointsManager class

With the idea of making ReFlex VR customizable and score-based came the need to make user data persistent across different play sessions. The approach chosen to solve this problem was to create a script, called "PointsManager", that contains a serializable class (Listing 4.18) that makes it possible to write all of the user's data in a JSON file. The information that is saved in a file includes: the username (a string containing the patient's name), the kitchen level's current points, the coffee level's current points and the living room level's current points (all three are integer values), the patient's affected arm (a string containing either "Left" or "Right"), the type of enviroment that has been chosen (a string containing either "Simple" or "Detailed") and the helper that has been chosen by the patient, if any (represented by an integer, 0 means no helper). The class needs to be serializable (line 1) in order to be able to work with JsonUtility, a Unity class that makes writing and reading JSON files simple through ad-hoc methods.

```
1   [System.Serializable]
2   class SaveData
```

```
3  {
4      public string Username;
5      public int CurrentPoints;
6      public int CoffeeLevelPoints;
7      public int LivingRoomLevelPoints;
8      public string AffectedArm;
9      public string ChosenEnvironment;
10     public int SelectedHelper;
11 }
```

Listing 4.18. A snippet of the "PointsManager" script: the SaveData class

In the menu scene an empty GameObject has been added, called "Points Manager", and the aforementioned script has been attached to it; aside from the SaveData class, which is necessary to actually make user data persistent, this script also creates a static PointsManager instance (Listing 4.19) that makes it possible to instantiate a PointsManager GameObject once in a game session and access it from any scene contextually to that same session. The script's variables (lines 3-9) are used to save the player's current settings, either by reading them from the user's JSON file if they already existed or by creating them in the current game session: they could be considered the interface between the game and the SaveData class. `CurrentUsername` (line 10) is used to track the active user, whereas `OnPointsChanged` (line 12) is an event used to update some text-based UIs and `charSel` (line 13) is used to load the current user's selected helper when loading and already existing save file.

```
1  public static PointsManager Instance;
2
3  public int Points = 0;
4  public int SecondLevelPoints = 0;
5  public int ThirdLevelPoints = 0;
6
7  public string Arm = "Right";
8  public string Environment = "Simple";
9  public int Character = 0;
10 public string CurrentUsername { get; private set; }
11
12 public event Action OnPointsChanged;
13 public CharacterSelection charSel;
14
15 private void Awake()
16 {
17     if (Instance != null)
18     {
19         Destroy(gameObject);
20         return;
21     }
22
23     Instance = this;
24     DontDestroyOnLoad(gameObject);
25     LoadPoints(CurrentUsername);
26 }
```

Listing 4.19. A snippet of the "PointsManager" script: the static Instance, the variables and the Awake method

Listing 4.20 shows the CharacterSelection class, which handles the game's helper selection logic in the starter menu: the player can choose a helper when they create a new save data (or by going into the settings tab of their already existing save data) and the selection is done through a dropdown menu, in which each option triggers the `SetCharacter`

method when clicked, saving the current choice both in the `selectedCharacter` variable and in the PlayerPrefs (a Unity class that stores player preferences) and also changing the helper model shown to the player next to the UI to be the one that they chose. This script also contains a method, `StartGame`, that is called from the level buttons with the appropriate string to load their respective levels.

```
public class CharacterSelection : MonoBehaviour
{
    public GameObject[] characters;
    public int selectedCharacter = 0;
    public TextMeshProUGUI subtitle;
    public string levelName;
    public string noHelperLevelName;

    public void SetCharacter(int charIndex)
    {
        characters[selectedCharacter].SetActive(false);
        selectedCharacter = charIndex;
        characters[selectedCharacter].SetActive(true);
        PlayerPrefs.SetInt("selectedCharacter", selectedCharacter);
        subtitle.SetText($"{characters[selectedCharacter].name}");
    }

    public void StartGame(string lvlName)
    {
        if (SceneManager.GetActiveScene().name == "NewStarterMenu")
            PlayerPrefs.SetInt("selectedCharacter", selectedCharacter);
        SceneManager.LoadScene(lvlName);
    }
}
```

Listing 4.20. The “CharacterSelection” script

Going back to the “PointsManager” script, the main method responsible for writing the current user’s data to a JSON file is `SavePoints` (Listing 4.21), which creates a new `SaveData` variable, updates all of its fields by copying the values of the corresponding variables into them, and finally formats this information and saves it to a file. In order to do so, firstly it converts the `SaveData` variable into a JSON-compatible string (through the `JsonUtility.ToJson` method), then it creates a unique path for the file (`Application.persistentDataPath` provides a folder that will persist even when the game session ends), and then it creates a save data file in that path with the `File.WriteAllText` method. `SavePoints` is called in two different methods shown in Listing 4.21, `AddPoints` (which accepts a username and an amount to add to the kitchen level’s points, adds that amount, triggers the `OnPointsChanged` event and then calls the `SavePoints` method with the received username) and `SaveCurrentUserPoints`, that doesn’t modify any current value but it simply saves the current values by calling `SavePoints` with `CurrentUsername`, if it was defined.

```
public void AddPoints(string username, int amount)
{
    Points += amount;
    OnPointsChanged?.Invoke();
    SavePoints(username);
}

public void SaveCurrentUserPoints()
{
    if (!string.IsNullOrEmpty(CurrentUsername))
```

```
    {
        SavePoints(CurrentUsername);
    }
    else
    {
        Debug.LogError("No username set! Cannot save points.");
    }
}

public void SavePoints(string username)
{
    SaveData data = new SaveData();
    data.Username = username;
    data.CurrentPoints = Points;
    data.CoffeeLevelPoints = SecondLevelPoints;
    data.LivingRoomLevelPoints = ThirdLevelPoints;
    data.AffectedArm = Arm;
    data.ChosenEnvironment = Environment;
    data.SelectedHelper = Character;

    string json = JsonUtility.ToJson(data);
    string path = Application.persistentDataPath + $"/savefile_{username}.json";
    File.WriteAllText(path, json);
}
```

Listing 4.21. A snippet of the "PointsManager" script: the AddPoints, SaveCurrentUserPoints and SavePoints methods

`LoadPoints` is `SavePoints`' complementary method (Listing 4.22): firstly it creates a string containing the path obtained from the username that it receives, and then it checks whether it already exists in the if-else statement. If it does, then this method reads the file's contents to a string (through `File.ReadAllText`) and converts it to a SaveData variable (with `JsonUtility.FromJson<T>`), to then copy all of the newly created SaveData variables' values to the script's variables; if it does not, the method simply "resets" the values of the variables pertaining to each level's points back to zero.

```
public void LoadPoints(string username)
{
    string path = Application.persistentDataPath + $"/savefile_{username}.json";
    if (File.Exists(path))
    {
        string json = File.ReadAllText(path);
        SaveData data = JsonUtility.FromJson<SaveData>(json);

        Points = data.CurrentPoints;
        SecondLevelPoints = data.CoffeeLevelPoints;
        ThirdLevelPoints = data.LivingRoomLevelPoints;
        Arm = data.AffectedArm;
        Environment = data.ChosenEnvironment;
        Character = data.SelectedHelper;
        PlayerPrefs.SetInt("selectedCharacter", Character);
        charSel.selectedCharacter = Character;
    }
    else
    {
        Points = 0;
        SecondLevelPoints = 0;
        ThirdLevelPoints = 0;
    }
}
```

Listing 4.22. A snippet of the "PointsManager" script: the LoadPoints method

This method is called by `SetCurrentUser` (Listing 4.23), which receives a string representing a username and calls `LoadPoints` with that string. `SetCurrentUser` is called in the menu scene from a dropdown menu containing all of the existing usernames that appears if the player clicks on the "Load game" button; this dropdown menu is populated through the `GetSavedUsernames` method, which creates a list of strings containing all of the existing usernames, which in turn are extracted from the save files in the folder created by `Application.persistentDataPath`.

```
public void SetCurrentUser(string username)
{
    CurrentUsername = username;
    LoadPoints(CurrentUsername);
}

public List<string> GetSavedUsernames()
{
    List<string> usernames = new List<string>();
    string[] files = Directory.GetFiles(Application.persistentDataPath, "savefile_
    *.json");

    foreach (string file in files)
    {
        string filename = Path.GetFileNameWithoutExtension(file);
        string username = filename.Replace("savefile_", "");
        usernames.Add(username);
    }

    return usernames;
}
```

Listing 4.23. A snippet of the "PointsManager" script: the SetCurrentUser and GetSavedUsernames methods

Finally, "PointsManager" contains getter and setter methods for all of the available variables (`Arm`, `Environment`, `Character`, `SecondLevelPoints` and `ThirdLevelPoints`) and two more methods, `AddSecondLevelPoints` and `AddThirdLevelPoints`, that are very similar to `AddPoints` except for the fact that the updated variable is, respectively, `SecondLevelPoints` and `ThirdLevelPoints` instead of `Points`.

Because of the static instance definition, every other script can access "PointsManager"'s methods without any using statement: this makes it possible to personalize the gameplay based on the current settings, for example by adding or removing props depending on the `Environment` variable's value or moving objects to either side of the rig based on the `Arm` variable's value. The latter is also used when tallying points, since the player will be rewarded with more points if they complete tasks with their paretic limb. As an example, Listing 4.24 shows the `CheckHandednessAndUpdatePoints` method, which is called every time a carrot piece enters the pan's trigger collider (see Listing 4.4 for more details): when that happens, this method obtains the IHand variable from the interactable that is allowing for the grab interaction with that carrot piece and compares it with the value of `PointsManager.Instance.Arm`. If the player was grabbing the carrot piece with its paretic hand, then he gets a point; otherwise, an audio clip is played urging the patient to use their affected arm in order to get more points. This is not the only way to get points for this task though, since it is also a timed exercise: `TallyPoints` gets called when the player has successfully placed 10 carrot pieces inside the pan, and it checks the amount of time it took by calculating the difference between the ending time

and the starting time, adding two more points if the user has managed to complete the task in under 10 seconds, one point if it took more than 10 seconds but less than 20, and no points otherwise.

```
private void CheckHandednessAndUpdatePoints(GameObject obj)
{
    _interactable = obj.GetComponent<HandGrabInteractable>();
    if (_interactable != null)
    {
        var interactors = _interactable.Interactors;
        if (interactors.Count > 0)
        {
            _interactor = interactors.FirstOrDefault();
            if ((_interactor != null) && _interactor.IsGrabbing)
            {
                IHand hand = _interactor.Hand;
                if (hand != null)
                {
                    if (PointsManager.Instance.Arm == "Right")
                    {
                        if (hand.Handedness == Handedness.Right)
                        {
                            PointsManager.Instance.Points += 1;
                        }
                        else
                        {
                            if (localizedAudio != null && !localizedAudio.
    IsClipPlaying())
                            {
                                localizedAudio?.PlayLocalizedClip(9);
                            }
                        }
                    }
                    else
                    {
                        if (hand.Handedness == Handedness.Left)
                        {
                            PointsManager.Instance.Points += 1;
                        }
                        else
                        {
                            if (localizedAudio != null && !localizedAudio.
    IsClipPlaying())
                            {
                                localizedAudio?.PlayLocalizedClip(9);
                            }
                        }
                    }
                }
            }
        }
    }
}

public void TallyPoints()
{
    float totTime = taskEndTime - taskStartTime;
    if (totTime < 10)
        PointsManager.Instance.Points += 2;
    else if (totTime < 20)
        PointsManager.Instance.Points += 1;
}
```

Listing 4.24. A snippet of the "CarrotsInPanObjective" script: the CheckHandednessAndUpdatePoints and TallyPoints methods

4.2.2 "SimpleHandDetection" and "ArmExtensionDetector"

A script that was created to make the handedness checking more modular and streamlined is "SimpleHandDetection" (Listing 4.25): the logic pertaining to that verification is inside said script's `Update` method, with the same structure as the one in the "CarrotsInPanObjective" script. If the grabbing hand's handedness does not coincide with the value of `PointsManager.Instance.Arm`, then the `WrongHandUsed` method is called, which continuously plays an audio clip asking the player to use their other hand for as long as they keep holding on to the GameObject this script is attached to with the "wrong" hand and setting the `hasUsedWrongHand` boolean variable to true. This variable is what makes it possible to correctly update the level's points from other scripts: an example of this can be seen in Listing 4.6, where a reference to the "SimpleHandDetection" component is obtained in the `OnTriggerEnter` method, to be able to tell whether the patient is grabbing the spoon with their paretic hand, and then in the win condition logic the value of `hasUsedWrongHand` is checked and the points are updated accordingly.

```
void Update()
{
    if (_interactable != null)
    {
        var interactors = _interactable.Interactors;
        if (interactors.Count > 0)
        {
            _interactor = interactors.FirstOrDefault();
            if ((_interactor != null) && _interactor.IsGrabbing)
            {
                IHand hand = _interactor.Hand;
                if (hand != null)
                {
                    if (PointsManager.Instance.Arm == "Right")
                    {
                        if (hand.Handedness == Handedness.Left)
                        {
                            WrongHandUsed();
                        }
                    }
                    else
                    {
                        if (hand.Handedness == Handedness.Right)
                        {
                            WrongHandUsed();
                        }
                    }
                }
            }
        }
    }
}

void WrongHandUsed()
{
    if (!localizedAudio.IsClipPlaying())
    {
        if (currentLevel == 0)
            localizedAudio?.PlayLocalizedClip(9);
        else if (currentLevel == 1)
            localizedAudio?.PlayLocalizedClip(7);
    }
    hasUsedWrongHand = true;
}
```

Listing 4.25. A snippet of the "SimpleHandDetection" script: the Update and WrongHandUsed methods

In Listing 4.8 there is an example of another script's method being called, specifically "ArmExtensionDetector": this script has been written in order to make it possible to evaluate the patients' ability to extend or contract their paretic limb within a certain amount of time and to award them with more or less points depending on the results. As shown in the "LeverRotationDetection" script, its logic is activated by calling its `SetTimer` method (Listing 4.26), which accepts a float value, that will be used as the maximum amount of time given to the patient to elongate or retract their paretic limb, and a boolean value, which indicates whether in the specific instance the user is supposed to extend (true) or contract (false) their arm. When this method is called, the timer starts running (lines 32-34) and the two variables that indicate success are reset (lines 36-37).

```
1  void Update()
2  {
3      if (!timerIsRunning)
4          return;
5      if (timerIsRunning)
6      {
7          if (timeRemaining > 0)
8          {
9              elapsedTime = Time.time - startTime;
10             if (lookingForExtension && !hasExtendedArm)
11             {
12                 DetectArmExtension();
13             }
14             else if (!lookingForExtension && !hasContractedArm)
15             {
16                 DetectArmContraction();
17             }
18             timeRemaining -= Time.deltaTime;
19         }
20         else
21         {
22             timeRemaining = 0;
23             timerIsRunning = false;
24             hasExtendedArm = false;
25             hasContractedArm = false;
26         }
27     }
28 }
29
30 public void SetTimer(float time, bool e)
31 {
32     startTime = Time.time;
33     timeRemaining = time;
34     timerIsRunning = true;
35     lookingForExtension = e;
36     hasExtendedArm = false;
37     hasContractedArm = false;
38 }
```

Listing 4.26. A snippet of the "ArmExtensionDetector" script: the Update and SetTimer methods

The `Update` method checks whether the timer is currently running and whether there is still time remaining, and if so it calls one of two possible methods: `DetectArmExtension`

(Listing 4.27, lines 1-19) is called if the specific exercise requires the patient to extend their arm and they have not done it yet, otherwise `DetectArmContraction` is called (Listing 4.27, lines 21-39).

```
1  void DetectArmExtension()
2  {
3      if (hasExtendedArm) return;
4
5      if (PointsManager.Instance.Arm == "Right")
6      {
7          float rightHandDistance = Vector3.Distance(rightHandTransform.position,
       headTransform.position);
8          bool isRightHandExtended = rightHandDistance > extensionThreshold;
9          if (isRightHandExtended)
10             Success();
11     }
12     else
13     {
14         float leftHandDistance = Vector3.Distance(leftHandTransform.position,
       headTransform.position);
15         bool isLeftHandExtended = leftHandDistance > extensionThreshold;
16         if (isLeftHandExtended)
17             Success();
18     }
19 }
20
21 void DetectArmContraction()
22 {
23     if (hasContractedArm) return;
24
25     if (PointsManager.Instance.Arm == "Right")
26     {
27         float rightHandDistance = Vector3.Distance(rightHandTransform.position,
       headTransform.position);
28         bool isRightHandContracted = rightHandDistance < extensionThreshold;
29         if (isRightHandContracted)
30             Success();
31     }
32     else
33     {
34         float leftHandDistance = Vector3.Distance(leftHandTransform.position,
       headTransform.position);
35         bool isLeftHandContracted = leftHandDistance < extensionThreshold;
36         if (isLeftHandContracted)
37             Success();
38     }
39 }
40
41 void Success()
42 {
43     string lvlName = SceneManager.GetActiveScene().name;
44     if (lvlName == "ApartmentScene")
45         PointsManager.Instance.Points += 1;
46     else
47         if (elapsedTime <= 10)
48         PointsManager.Instance.SecondLevelPoints += 2;
49     else if (elapsedTime > 10 && elapsedTime <= 20)
50         PointsManager.Instance.SecondLevelPoints += 1;
51     if (lookingForExtension)
52         hasExtendedArm = true;
53     else
54         hasContractedArm = true;
55     timeRemaining = 0;
56     timerIsRunning = false;
57 }
```

Listing 4.27. A snippet of the "ArmExtensionDetector" script: the DetectArmExtension, DetectArmContraction and Success methods

In either case, the elongation or contraction of the arm is determined by the relevant hand's position relative to the headset, which is calculated with `Vector3.Distance` and then compared to a certain threshold which can be modified in the inspector; it has been empirically determined that a value of 0.5f is adequate to distinguish between an extension of the arm and a contraction. When the given condition is met, then the `Success` method is called, which updates the points of the appropriate level, modifies the relevant boolean value to signal to the `Update` method that it should not keep calling the previously mentioned method, and finally it resets the timer and it sets `timerIsRunning` to false.

4.2.3 Points transposition in the Star system

Most scripts pertaining to the different tasks in the game contain some variation of the lines of code seen until now, adding a variable amount of points to the total for each task that is correctly executed by the patient. Point calculation has been made so that in each level the player can obtain from a minimum of 0 to a maximum of 50 points. In order to give users a clear indication of how well they have performed, these numerical values have been converted into a 5-star system: every 5 points obtained in a certain level are awarded with a star, going from a minimum of 0 to a maximum of 5 stars for each level. With the upper limit for points being set to 50, reaching a 5 star score is made easier: this choice was made in order to lower patients' frustrations and to keep them engaged, giving them a greater sense of accomplishment. The star icon is shown under each level's name in the level selection menu and in the victory screen after the player completes a level, to let them know how well they have done: it is an empty GameObject with an Image component and the "StarsHandler" script (Listing 4.28). This script has three variables that need to be assigned in the inspector: the star icon's Image component (`img`), all of the different star icons (`starSprites`) and an integer representing the current level or, in the case of the menu, which level the star icon is underneath (0 for the kitchen level, 1 for the coffee level, 2 for the living room level). In order to always have the latest values displayed, the script checks at every frame (in the `Update` method) the current value of the relative level's points through a switch case statement, then it converts those points into an index with the `GetStarIndex` method, and lastly it uses said index to load the correct sprite (line 11). Using `Mathf.Clamp` was not strictly necessary in this context, since `GetStarIndex` can only return a value between 0 and 5, but it is an additional security measure to prevent any possible bug or mistake that could lead to an index outside of the allowed range.

```
1 public class StarsHandler : MonoBehaviour
2 {
3     public Sprite[] starSprites;
4     public Image img;
5     public int lvl;
6
7     void Update()
```

```
8       {
9           int points = GetPointsForLevel(lvl);
10          int starIndex = GetStarIndex(points);
11          img.sprite = starSprites[Mathf.Clamp(starIndex, 0, starSprites.Length - 1)
        ];
12      }
13
14      int GetPointsForLevel(int level)
15      {
16          switch (level)
17          {
18              case 0:
19                  return PointsManager.Instance.Points;
20              case 1:
21                  return PointsManager.Instance.SecondLevelPoints;
22              case 2:
23                  return PointsManager.Instance.ThirdLevelPoints;
24              default:
25                  return 0;
26          }
27      }
28
29      int GetStarIndex(int points)
30      {
31          if (points < 5) return 0;
32          else if (points < 10) return 1;
33          else if (points < 15) return 2;
34          else if (points < 20) return 3;
35          else if (points < 25) return 4;
36          else return 5;
37      }
38  }
```

Listing 4.28. The “StarsHandler” script

4.3 Hints

In every game, hints are fundamental to help guide the player and to hopefully eliminate any possible source of frustration; when dealing with people who have suffered a stroke, hints become even more important since common side effects include cognitive impairments, which can make it more difficult for them to understand goals and objectives. This is why in ReFlex VR there are both visual (such as arrows and glowing pulsating silhouettes around objectives, Fig. 4.4) and audio cues (in the form of audio clips explaining to the player what to do, admonishing them for using their healthy arm to complete objectives, and “ping” sounds to signal the successful completion of each task).

In order to make the application useful to a broader group of people, the choice between two different languages (Italian and English) has been given, which applies to every text in the application as well as to the helpers’ audio clips; the logic pertaining to the language selection has been handled through Unity’s Localization package, by creating a string table collection for the text and an asset table collection for the audio clips and assigning the “Localize Audio Clip Event” (courtesy of Unity’s Localization package), “Localized Audio Changer”, and “Play Localized Audio” scripts to the appropriate GameObjects (two empty GameObjects called “ManAudio” and “WomanAudio” to handle male helpers’ and female helpers’ audio clips respectively).

“Localized Audio Changer” (Listing 4.29) sets the appropriate audio clips based on the

Figure 4.4. Some examples of visual cues present in ReFlex VR

language that has been chosen in the menu scene through the `ChangeAudio` method and plays a single audio clip with the `PlayClip` method, while "Play Localized Audio" (Listing 4.30) contains two main methods: `IsClipPlaying` is a helper method that returns a boolean indicating whether any audio clip is currently playing, and `PlayLocalizedClip` is the wrapper method that is actually called in other scripts, receiving an index, checking whether it is within the allowed range, and if so calling "LocalizedAudioChanger"'s `PlayLocalizedClip` method with the received index. Many examples of this method being called can be found in this chapter's first section, such as in the `PlayAudioWithDelay` method from the "Slicing Script" script (Listing 4.2).

```
1  void Start()
2  {
3      localizeAudioClipEvent.OnUpdateAsset.AddListener(PlayClip);
4      ChangeAudio(clips[currentClip]);
5  }
6
7  void ChangeAudio(LocalizedAudioClip clip)
8  {
9      localizeAudioClipEvent.AssetReference = clip;
10 }
11
12 public void PlayLocalizedClip(int clipIndex)
13 {
14     if (clipIndex < 0 || clipIndex >= clips.Length)
```

```
15          return;
16      ChangeAudio(clips[clipIndex]);
17  }
18
19  private void PlayClip(AudioClip clip)
20  {
21      if (clip != null)
22      {
23          audioSource.clip = clip;
24          audioSource.Play();
25      }
26  }
```

Listing 4.29. A snippet of the "LocalizedAudioChanger" script: the Start, ChangeAudio, PlayLocalizedClip and PlayClip methods

```
1   void Start()
2   {
3       localizedAudioChanger = FindAnyObjectByType<LocalizedAudioChanger>();
4       if (localizedAudioChanger != null)
5       {
6           audioSource = localizedAudioChanger.GetComponent<AudioSource>();
7       }
8   }
9
10  public void PlayLocalizedClip(int clipIndex)
11  {
12      if (localizedAudioChanger != null && clipIndex >= 0)
13      {
14          if (audioSource.isPlaying)
15          {
16              audioSource.Stop();
17          }
18          localizedAudioChanger.PlayLocalizedClip(clipIndex);
19      }
20  }
21
22  public bool IsClipPlaying()
23  {
24      return audioSource != null && audioSource.isPlaying;
25  }
```

Listing 4.30. A snippet of the "PlayLocalizedAudio" script: the Start, PlayLocalizedClip and IsClipPlaying methods

As previously mentioned, hints can be a useful tool and even necessary at times; however, in order to add another dimension of difficulty for the players that have already completed all three levels at least once and want to further challenge themselves a "no helper" mode has been added. To play in this mode, the user will have to go to their account's settings in the menu scene and change the helper to "None". This way, no helper will spawn in the levels and no audio cues will be given, and no clipboard showcasing all the different tasks will be provided either, meaning that the player will have to independently remember the different steps required to complete the level in the correct order, further putting their cognitive abilities to the test.

As for the visual cues, they progressively appear and disappear throughout each level to guide the player towards every objective in the correct order: this is what the "Arrow" GameObject references in many of the scripts that have been explored in previous sections are for. For example, going back to the "SprayCleanerHandler" script (Listing 4.10), among the various variables there is one called `spongeArrow`, which has been assigned

through the inspector. This variable contains a GameObject representing a light blue, translucent arrow pointing to the sponge in the scene (which has been animated through an Animator and Unity's "Animation" window to continuously move up and down, to further catch the user's attention), which at the start of the level is inactive (meaning it is not visible), since the player is not supposed to pick up the sponge until after they have sprayed all of the spots on the window. When this condition is met (Listing 4.10, line 27), the `SprayCleaningCompleted` method (Listing 4.31), containing the end-of-task logic, is called. The same mechanism can be found in many other scripts, such as in "LeverRotationDetection" (Listing 4.8, line 18) and "SnapEventsHandler" (Listing 4.9, line 40).

```
1  void SprayCleaningCompleted()
2  {
3      victorySound.Play();
4      checkbox.sprite = checkedBox;
5      spongeArrow.SetActive(true);
6      sprayText.SetActive(false);
7      wipeText.SetActive(true);
8      StartCoroutine(PlayAudioWithDelay(1f));
9      hasPlayed = true;
10     spongeHGI.enabled = true;
11     if (sprayBottleSHD.hasUsedWrongHand)
12         PointsManager.Instance.ThirdLevelPoints += 1;
13     else
14         PointsManager.Instance.ThirdLevelPoints += 2;
15 }
```

Listing 4.31. A snippet of the "SprayCleanerHandler" script: the SprayCleaningCompleted method

In other cases, logic related to arrows appearing and disappearing is tied to certain GameObjects being grabbed by the player. For these instances the "ArrowHandler" script (Listing 4.32) has been created, specifically for the wooden spoon in the kitchen level and for the coffee capsule, cup and saucer in the coffee level. In the `Start` method it saves a reference to the HandGrabInteractable that is attached to the same GameObject that "ArrowHandler" is attached to, and in the `Update` method it checks whether the GameObject is currently being grabbed (line 10): if so, the arrow that was pointing to that GameObject (which is assigned in the Inspector) is rendered inactive, and in specific cases another arrow, pointing to the next target, is activated.

Yet another implementation for the arrow logic can be found both for the coffee machine buttons and the button on the vacuum cleaner, which both contain a Poke Interactable and have an arrow pointing to them: in these cases, the task of deactivating the respective arrows has been assigned to the "Interactable Unity Event Wrapper" component assigned to both of these buttons, which sets the arrows as inactive in the "When Select ()" event.

```
1  void Start()
2  {
3      _interactable = targetObject.GetComponent<HandGrabInteractable>();
4      localizedAudio = FindAnyObjectByType<PlayLocalizedAudio>();
5  }
6
7  void Update()
8  {
9      var hand = _interactable.State;
```

```
10      if(hand == InteractableState.Select)
11      {
12          if (gameObject.CompareTag("Spoon") || gameObject.CompareTag("CoffeeCapsule
        ") || gameObject.CompareTag("Plate") || gameObject.CompareTag("Cup"))
13              spoonOutline?.SetActive(false);
14          arrow.SetActive(false);
15          if(!firstObjectiveDone)
16              otherArrow.SetActive(true);
17          firstObjectiveDone = true;
18          if(gameObject.CompareTag("CoffeeCapsule") && !hasPlayed)
19          {
20              localizedAudio?.PlayLocalizedClip(2);
21              takeCapsuleText.SetActive(false);
22              putCapsuleText.SetActive(true);
23              hasPlayed = true;
24          }
25      }
26  }
```

Listing 4.32. A snippet of the "ArrowHandler" script: the Start and Update methods

In the "ArrowHandler" snippet above, besides the arrow activating and deactivating there is an example of how the other major visual cue present in the game is handled, which is the outline (line 13). Outlines are pulsating light blue hues that surround certain objectives: in order to create them, another GameObject has been added as a child to each GameObject that needed an outline, and the child GameObject has been assigned the same mesh as the parent, in order to obtain an outline that perfectly follows the original object's edges. The child GameObject has then been assigned a material (called "OutlineMat") obtained from a custom-made Shade Graph that makes the material translucent, makes only the parts of the outline that don't overlap with the parent mesh visible to the user (effectively giving the outline effect) and exposing three variables: "Outline Thickness" (a float with a slider ranging from 0 to 0.5), "Fade Distance" (another float with no fixed maximum or minimum value), and "Outline Color" (a color variable with a color picker coming with it).

These variables are then accessed through the "OutlinePulse" script (Listing 4.33), assigned to the child GameObject itself. The Renderer variable is a reference to the object's renderer and it's necessary in order to be able to modify its material; the MaterialPropertyBlock variable allows to override the particular instance of the material's properties without having to create a new material each time; `minThickness` and `maxThickness` allow to choose in the Inspector the thickness values between which the outline should oscillate, and `speed` determines how fast this oscillation will occur. At every frame, this script uses the sine function to generate a value between 0 and 1 that continuously oscillates over time, then uses that generated value to smoothly move between the defined minimum and maximum values; after that, `GetPropertyBlock` retrieves the current property values from the renderer and saves them into `propBlock`, which then modifies the "Outline Thickness" property with the value that has just been calculated, and lastly `SetPropertyBlock` pushes the updated property block back to the renderer.

```
1   public class OutlinePulse : MonoBehaviour
2   {
3       private Renderer rend;
4       private MaterialPropertyBlock propBlock;
5
6       [SerializeField] private float minThickness = 0.02f;
```

```
7       [SerializeField] private float maxThickness = 0.045f;
8       [SerializeField] private float speed = 2f; // Controls how fast it pulses
9
10      void Awake()
11      {
12          rend = GetComponent<Renderer>();
13          propBlock = new MaterialPropertyBlock();
14      }
15
16      void Update()
17      {
18          float t = (Mathf.Sin(Time.time * speed) + 1f) / 2f;
19
20          float thickness = Mathf.Lerp(minThickness, maxThickness, t);
21
22          rend.GetPropertyBlock(propBlock);
23          propBlock.SetFloat("_Outline_Thickness", thickness);
24          rend.SetPropertyBlock(propBlock);
25      }
26  }
```

Listing 4.33. A snippet of the "OutlinePulse" script

As previously said, these "outline" GameObjects are assigned as children to many different objectives that the player is supposed to either grab or press, often paired with an arrow; this is why in most scripts that deal with the enabling/disabling of arrows there is also some logic related to the enabling or disabling of outline GameObjects, some of which can be found in the scripts that have been discussed so far.

Going back to the arrow hints, a particular case worth mentioning is the one regarding the first two macro-tasks in the living room level, namely the window cleaning and the floor vacuuming tasks: each dirt spot on the window and dust ball on the floor has its own arrow pointing to it, that gets disabled respectively when the dirt spot comes in contact with the sponge and when the dust ball is reached by the vacuum cleaner. However, in order to remove one dirt spot from the window the user needs to make it collide with the sponge four times (Listing 4.12, line 27), and to remove one dust ball from the ground it needs to collide with the vacuum cleaner at least three times, as can be seen from the "VacuumCleaningHandler" script attached to each dust ball (Listing 4.34, line 27). Furthermore, both the dirt spots and the dust balls shrink proportionally each time they collide with the sponge and the vacuum cleaner, respectively, until they disappear completely, which means that the player might have difficulty seeing them when their scale gets reduced.

```
1   public class VacuumCleaningHandler : MonoBehaviour
2   {
3       private VacuumCleanerHandler vch;
4       private int hitCount = 0;
5       private Vector3 originalScale;
6       public GameObject arrow;
7       private float timer = 0f;
8       private bool arrowStillActive = true;
9
10      void Start()
11      {
12          vch = FindAnyObjectByType<VacuumCleanerHandler>();
13          originalScale = transform.localScale;
14      }
15
```

```
16      void Update()
17      {
18          if (!arrowStillActive)
19          {
20              timer += Time.deltaTime;
21              if (timer >= 10f)
22              {
23                  arrow.SetActive(true);
24                  timer = 0f;
25              }
26          }
27          if (hitCount >= 3)
28          {
29              gameObject.SetActive(false);
30              vch.UpdateDustRemovedVariable();
31          }
32      }
33
34      private void OnTriggerEnter(Collider other)
35      {
36          if (other.gameObject.CompareTag("Vacuum") && vch.VacuumIsOn())
37          {
38              timer = 0f;
39              arrow.SetActive(false);
40              arrowStillActive = false;
41              hitCount++;
42              float shrinkFactor = Mathf.Clamp01(1f - (hitCount / 3f));
43              transform.localScale = originalScale * shrinkFactor;
44          }
45      }
46  }
```

Listing 4.34. A snippet of the "VacuumCleaningHandler" script

This is why for these particular tasks a timer mechanism has been implemented in order to make the arrows reappear periodically after every successful collision over dirt spots or dust balls that are yet to be completely removed, so that the user will know not only that the task has not been completed yet, but also where to exactly direct their efforts towards. In the "DirtOnGlassHandler" script (Listing 4.12), a float variable called `timer` has been defined and it is set to zero when the dirt spots collide with the spray particles, also disabling the arrow related to the specific dirt spot and setting the `arrowStillActive` boolean variable to false: this last step is necessary to be able to access the else-if statement in the `Update` method, which can only be reached when the spot has been sprayed, its arrow is currently disabled and all of the spots have been sprayed. If that is the case, it means that the spraying task is completed and the player has to now scrub the spots with the sponge: from this point on, the timer variable allows to create a 10 second time buffer, after which if the specific spot is yet to be completely eliminated, its arrow will be enabled again (line 22). It will then be disabled after every collision with the sponge and reenabled after 10 more seconds if the player does not rub it again in the meantime, ensuring that they will not miss a single spot. An identical logic has been adopted in the "VacuumCleaningHandler" script, making the arrows related to each dust ball periodically reappear as long as the dust ball itself has not been completely removed.

Chapter 5

Usability Testing

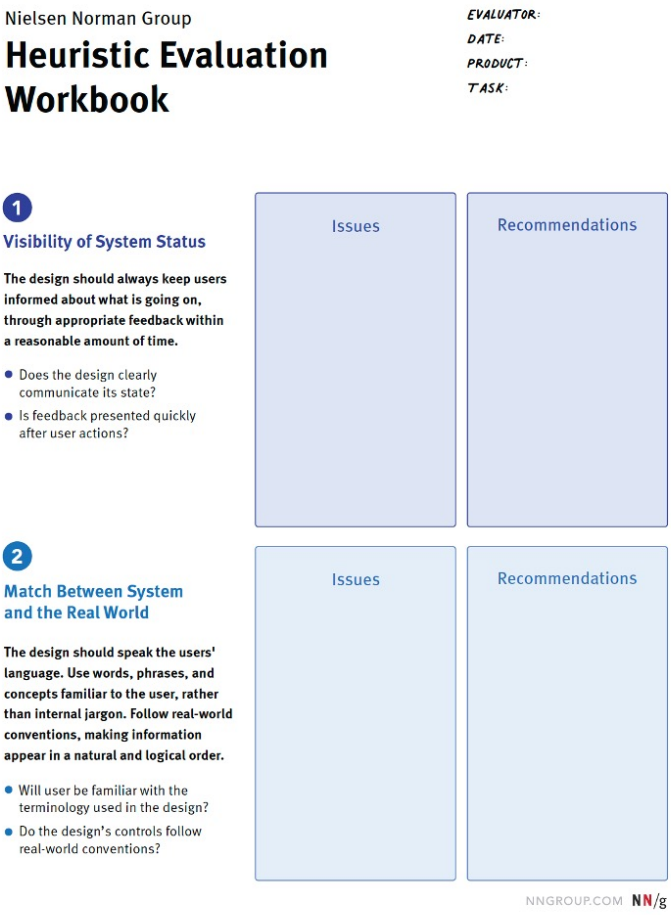
ReFlex VR has been developed with the express purpose of being employed in an experiment involving a group of patients from Centro Puzzle in Turin aimed at exploring the efficacy of such technologies in helping people that suffered a stroke regain upper limb mobility and independence in their day-to-day life. However, before doing that, it was important to submit the application to a usability expert, who would be able to use a set of standard heuristics to rate its usability and find any potential issues that should be fixed before patients get to try it.

In order to evaluate the application, the usability expert has employed Nielsen Norman Group's Heuristic Evaluation Workbook (Fig. 5.1), which is based on Jakob Nielsen's 10 usability heuristics, a group of high-level requirements that have been designed based on how human beings tend to react to and process information. The workbook, which the expert has digitally filled out and then shared via email, contains the 10 guidelines in a numbered list with a small description for each one and two boxes next to it, one for the issues that the application presents related to that specific guideline and the other for recommendations to fix said issues. To make the evaluation process more robust, a group of experts should have been involved, each independently giving their own inputs, however for the scope of this thesis a singular expert has been deemed to be more than enough.

The following list contains each usability heuristic, its explanation, and then the potential concerns that have been raised by the expert regarding that heuristic, the relative fixes, and a score out of five that has been established by them for each point to make the application's strengths and weaknesses clearer at a glance:

1. **Visibility of System Status**: this heuristic is associated with how easy it is for the user to know what is going on and to have access to all relevant information. The system should clearly communicate to the user the current state they're in, so that they can make an informed and confident decision on what to do next, but it should also give immediate feedback, in order to give them a sense of agency and let them know that whatever they are doing is working (or is not working).

   The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, acknowledging how the player is properly supported and informed on what is happening every step


Nielsen Norman Group

Heuristic Evaluation Workbook

EVALUATOR:
DATE:
PRODUCT:
TASK:

1 Visibility of System Status

The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.

- Does the design clearly communicate its state?
- Is feedback presented quickly after user actions?

Issues | Recommendations

2 Match Between System and the Real World

The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.

- Will user be familiar with the terminology used in the design?
- Do the design's controls follow real-world conventions?

Issues | Recommendations

NNGROUP.COM NN/g

Figure 5.1. The first page of Nielsen Norman Group's Heuristic Evaluation Workbook

of the way, both with visual and with audio cues. The application always performs in a predictable and reliable manner, never making the user feel disoriented.

2. **Match between the System and the Real World**: this principle is based on the fact that human beings feel the most comfortable in familiar environments, therefore the application should strive to strictly contain language that is easily understandable by end users, so as to not induce them to go look for definitions somewhere else (which is undesirable in general but even more so in the context of a VR experience, since it would completely break immersion and flow). Nielsen's second heuristic also evaluates how much the interactions present in the application mimic real world activities, since a naturalistic approach leverages users' existing knowledge making it that much easier to learn how to navigate the virtual experience without a steep learning curve.

The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, due to the lack of obscure and overly technical terminology throughout the entire application and the ease they experienced while navigating the menu and even more so while completing the levels. This result has been made possible by it being a Virtual Reality application and by the design choices that have been made, which were aimed at making the different levels feel as life-like and realistic as possible, interactions included.

3. **User Control and Freedom**: this section specifically relates to how easy it is for the user to go back or to easily fix any mistake they could possibly make; the more effortless it is, the greater their sense of control and agency will be. The "Undo" or "Exit" options must be present in the application and also clearly labeled and easily discoverable, so as not to frustrate the user. The user must never feel stuck in an undesirable state and unsure on what to do.

   The usability expert gave ReFlex VR a 3 out of 5 for this heuristic, since at the time of the evaluation they found the menu to be easily navigable but there was no way of exiting the levels without completing them if not by closing the application altogether. There were already some fail-safes in place, such as the "RespawnOnDrop" script attached to every grabbable object, ensuring that they would respawn in a reachable position in the eventuality of the player dropping them on the ground, and an invisible wall on the far end of the kitchen that would prevent the carrot from rolling too far away from the user, but for any other possible bug, or even just in case the user wanted to take a break, there was no exit option. This has been fixed afterwards by adding to every level an exit button (Fig. 3.2) which has been assigned a bright red color to make it pop and an "exit" icon to make its purpose clear, giving users the option to either go back to the menu scene or to quit playing whenever they feel like it.

4. **Consistency and Standards**: according to this heuristic, any application should strive to maintain visual and semantic consistency; users should never wonder whether different words mean the same thing or whether different actions lead to the same outcome. Designers are encouraged to follow already established industry standards to further enhance the users' sense of familiarity and to lower their cognitive load and learning curve when it comes to UI elements, so that they can fully concentrate on the application's actual content. Consistency must be both internal (the entire application must adopt the same "language") and external (the application's UI must meet the industry standards and conform to well-known correlations, such as an "x" icon meaning "close").

   The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, praising the clean, straightforward and consistent UI solutions that have been adopted (Meta's own guidelines for VR UI elements have been followed to this end). Everything works as one would expect, from drop-down menus to buttons; all of the icons present in the UI elements have easily inferrable meanings, from the question mark icon to get more information to the arrow buttons to go back in the menu. From

a visual standpoint, consistency has been achieved in the form of all of the UI elements having the same look and feel throughout the entire application.

5. **Error Prevention**: this section is based on the principle of avoiding situations in which errors occur as much as possible through appropriate design choices. Errors may arise from two different user behaviors: slips and mistakes. Slips are unconscious wrong actions that may occur when the user is distracted, whereas mistakes are made consciously when the user has a mental scheme of the application that does not match with the application itself, leading to confusion and errors. Slips can be prevented by gently guiding the user towards the right path, either by adding helpful constraints, offering suggestions and designing good defaults that can be used when all else fails; mistakes are avoided by following design conventions, clearly communicating which elements can be interacted with, offering a preview of the results of a certain action, and supporting undo.

   The usability expert gave ReFlex VR a 4 out of 5 for this heuristic, since most possible errors are accounted for through the fail-safes in place (such as the previously mentioned "RespawnOnDrop" script) and the various hints that help guide the player towards all of the different goals; however, the usability expert expressed some concern for the carrot slicing task, seeing as the way in which it was implemented can lead to bugs and unexpected behaviors, especially when the slices are particularly thin. They suggested that instead of programmatically creating two new meshes and deleting the previous one based on raycasting, the carrot slicing could be achieved by having a pre-sliced carrot that initially looks intact and progressively detaching the various meshes as the user makes the knife collide with the carrot, losing the sense of realism in favor of a more reliable application.

6. **Recognition Rather than Recall**: these two terms represent two different ways to trigger one's memory, with the first one consisting of remembering based on many received cues related to that specific memory, making this process less mentally vexing, and the second one meaning having to solely go off of one's recollection with little to no external cues to help them. This heuristic states that users should have to actively remember as little as possible when it comes to the application's interface and commands, relying on the options presented to them instead, which should be sufficiently visible and clear at a glance. Contexts that rely on recognition more than recall reduce the users' memory load and thus their cognitive effort, making in turn their experience with the application more pleasant and intuitive. Furthermore, all of the relevant information should be given in context based on the application's current state as the user progresses through their tasks, rather than supplying the user with a general tutorial and then expecting them to remember everything by themselves.

   The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, recognizing the care put into the application's UI in order to lower the users' cognitive effort outside of the tasks themselves as much as possible. They found the menu to be very clear and easy to use, down to the info buttons (the small round buttons with a question mark icon next to each option in the user creation menu, Fig. 3.6B)

that give detailed explanations for each option so that the user can make a truly informed decision. Provided that they chose to have a helper, the user will also rely on recognition rather than recall in all of the levels, on account of the many visual and audio cues present throughout the game, each one appearing only when the player actually needs it.

7. **Flexibility and Efficiency of Use**: this heuristic relates to the application's predisposition to be informative enough for novice users, guiding them step-by-step in order to lower frustration and confusion, but also to have customization features, shortcuts and so-called accelerators (faster methods to complete frequent sets of actions) for experienced users. This type of flexibility allows for a more efficient use of the application for repeat users while not making it arduous to navigate for whoever might interact with it for the first time.

   The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, even though there are no shortcuts to speak of in the application but that is by design, since patients are meant to complete each level step by step without skipping anything in order to train their upper limbs. There is, however, a way to personalize the experience and make it less guided for those who are already familiar with all of the different tasks and desire to be challenged further, and that is by choosing to have no helper, thus skipping all of the hints and just playing.

8. **Aesthetic and Minimalist Design**: Nielsen's eight heuristic urges designers to adopt a minimalist approach to their work, keeping only relevant information visible to the user, while also maintaining an eye for aesthetics, since first impressions are extremely important even when it comes to applications and a strong aesthetic imprint can lead to higher brand recognition. There must not be any purely decorative elements that take attention away from the actual useful information and may even hide it altogether, but the elements that are present have to be visually pleasing and well designed; furthermore, designers must keep a balance between having no visual clutter but also not removing any useful elements, so as not to make the application less navigable and clear.

   The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, acknowledging how visual clutter is kept to a minimum in all of the different UI elements, while showing all relevant information at any given time. The aesthetics are pleasant and consistent throughout the entire application; the UI elements are nice to look at, with rounded edges and soft colors, and all of the text has high contrast in order to make it easily readable. All of the icons are high-resolution, signifiers are clear and helper text is present and written in plain, understandable language. Aside from the UI, visual clutter is kept to a minimum even in the levels themselves, with very little objects that could distract the player from their main objective; the visuals can be customized to be even more minimalist by choosing the "Essential" option for the environment in the main menu.

9. **Help Users Recognize, Diagnose, and Recover from Errors**: the title of this heuristic is self-explanatory; in order to achieve it some guidelines have been

defined, which include displaying the error message near the error's source to avoid confusion, using noticeable and well-known indicators (such as a red outline and high-contrast text), describing the issue at hand to not leave the user wondering while using plain and familiar language so that they can understand what went wrong, offering constructive advice to fix the problem without ever using blaming and condescending tones towards the user, putting safeguards in place against common mistakes, and preserving the user's input in the case of a failure, so that they will not have to go through the hassle of repeating what they have already done.

The usability expert gave ReFlex VR a 4 out of 5 for this heuristic; the application does not contain any error messages per se, since most of the possible errors have been dealt with through safeguards and protective measures. When the patient uses their unaffected arm to complete the different tasks, an audio clip will gently remind them to use their other arm in order to obtain more points, while still praising them for a job well done. The only concern raised by the expert on this area was about the carrot slicing task, since neither the audio clip explaining the task nor the clipboard with the list of objectives mentioned exactly how many carrot pieces were expected before considering the task to be completed, which could lead to confusion and to the player thinking they have finished and they can move on to the next assignment prematurely.

10. **Help and Documentation**: the last section goes over the importance of providing users with important documentation whenever necessary, so that they never have to wonder how to proceed. Help documentation must be concise but comprehensive, easy to search, and it should supply a list of concrete steps that the user should follow in order to reach their goal. Help can be divided into proactive and reactive: proactive help is provided a priori, before any issue has occurred, to help familiarize users with the application, and it can either come as push reveleations (generic help that is not relevant to the user's current task) and pull revelations (contextual tips that only appear when relevant, for example when the user has started the corresponding task); reactive help is sought by users after they have encountered a problem that needs troubleshooting or by users that intend to become experts. Proactive help usually comes in the form of tutorials, templates, instructional overlays, and tooltips, whereas reactive help is usually provided as a FAQ (Frequently Asked Questions) page, technical documentation, training modules, or sometimes even graphics and videos. When proactive help is provided, pull revelations should be the preferred solution, since push revelations tend to be ignored by the user as they are not immediately applicable.

    The usability expert gave ReFlex VR a 5 out of 5 for this heuristic, praising how thorough the application is in helping the user understand it at every step, from the user creation with the tooltips next to each option to give concise, easily understandable explanations for the various options but only if the user wants to read them (making them an example of reactive help) to the clipboard that is clearly visible to the player at all times and contains a list of all the various steps they

must follow to reach the end of the level (which might be considered a push revelation) to the audio clips and the visual hints that guide inexperienced players as they go (which are pull revelations). How to complete the tasks themselves is quite self-explanatory, since the naturalistic approach taken expects users to act as they would in the real world (for example placing the pan on the stove is done by performing a grabbing motion on the pan's handle and moving it on top of the stove), but in order to be even more exhaustive an informational video has been shot, showing a person performing the various tasks with their arms and hands clearly visible and then pointing to what those gestures produce in the game, and explaining everything with a voiceover, which is meant to be shown to the patients before they try the application for the first time and is then available for them to consult whenever they might need to.

5.1 Findings, fixes, and possible future changes

To summarize, the points given for each heuristic were:

- Visibility of System Status: 5/5
- Match between the System and the Real World: 5/5
- User Control and Freedom: 3/5
- Consistency and Standards: 5/5
- Error Prevention: 4/5
- Recognition Rather than Recall: 5/5
- Flexibility and Efficiency of Use: 5/5
- Aesthetic and Minimalist Design: 5/5
- Help Users Recognize, Diagnose, and Recover from Errors: 4/5
- Help and Documentation: 5/5

The mean value is 4.6, with a standard deviation of 0.66.

In conclusion, the usability expert has expressed overall positive opinions on the application, pointing to some issues that in their opinion needed to be fixed before the experiment with the patients and also adding some changes that they would consider as improvements but that were not strictly necessary. The problems that have been fixed immediately after receiving the expert's feedback include the inability to leave a level before finishing it (the "exit" button has been added) and the lack of a clear number for the slices that the carrot should be cut into (the audio clip related to that task has been modified to include the number, and the text on the clipboard has been adjusted as well), whereas as for the slicing of the carrot itself it has been decided to keep it based on the raycasting logic and the creation of new meshes instead of having a pre-sliced

carrot, since the realism of having the slices reflect the user's actions has been deemed more important than preventing possible bugs arising from physics issues.

The expert has raised one more concern in the event of ReFlex VR becoming commercially available, and that is a privacy concern: when a patient creates a new user they are expected to insert their own name, and when a returning player wants to select their own user to keep playing they are presented with a list of names, thus not protecting any of the users' privacy. The expert has suggested forgoing the name system altogether and identifying different users through numbers instead, or perhaps even better creating a username-password system and presenting players with a screen where they are expected to insert their name and the password that they chose, instead of making them select their name from a list. This suggestion has been considered to be outside of this thesis' scope, since the application was meant to be used only in a controlled environment where the patients all knew each other and privacy was not much of a concern in this sense, but that was very useful advice that will surely be taken into consideration for any future developments of ReFlex VR.

Chapter 6

Results & analysis

6.1 Setup and Pilot

For the experiment thirteen patients were recruited from the Centro Puzzle rehabilitation center in Turin; they were people of both genders between ages 18 and 65. The inclusion criteria were:

- confirmed stroke (ischemic or hemorragic) or traumatic brain injury (TBI) on computerized tomography (CT) or magnetic resonance imaging (MRI);
- upper limb motor impairment (Fugl-Meyer score - upper limb subscale);
- willingness and ability to give informed consent;
- normal or corrected-to-normal vision.

The exclusion criteria were:

- cognitive impairment or severe receptive aphasia precluding ability to understand tasks and provide informed consent;
- contraindications to VR (nausea, dizziness, etc.);
- history of other neurological or phychiatric disorders;
- presence of unilateral spatial neglect.

Before proceeding with the actual experiment, twelve of the thirteen participants were made to try the application once to see how they would fare and to gather their first impressions and any possible suggestion to improve the game from a user-centric perspective. Every patient was made to sit on a chair in a quiet room and watch a video (Fig. 6.1) that was created as a tutorial of sorts, to show first time users how to wear the headset, how to make their hands clearly detectable by the headset's sensors by wearing short-sleeve tops or by rolling up their sleeves, how to use controllers and how to navigate the menu and the levels.

Figure 6.1. A frame of the ReFlex VR presentation video

After watching the video, the patients were asked if they had any question, and if not they were instructed to wear the headset (which was adjusted on their head by external help to make them feel comfortable), open the application, create a new user by using the controller, and finally play the first level (the kitchen level). When they were done, they were asked a series of questions to gather their thoughts on the application and on the technology employed. The following list contains all of the question asked with all of the possible answers and a number representing how many participants chose that specific answer:

- How easy was it to learn how to use the VR system?
  - Very easy: 3
  - Easy: 6
  - Neutral: 2
  - Difficult: 1
  - Very difficult: 0
- Did you feel comfortable while using the VR headset?
  - Very comfortable: 4
  - Comfortable: 4
  - Neutral: 2
  - Uncomfortable: 2
  - Very uncomfortable: 0
- Did you experience any of the following? (Check all that apply)
  - Dizziness: 2

  - Nausea: 1
  - Headache: 0
  - Eye strain: 0
  - Fatigue: 0
  - None: 10

- Were the instructions in the game clear and easy to follow?
  - Yes: 7
  - Somewhat: 4
  - No: 1

- How physically tiring was the session for you?
  - Not tiring: 9
  - Slightly tiring: 2
  - Moderately tiring: 1
  - Very tiring: 0

- Did you enjoy using the VR game for rehabilitation?
  - Yes, very much: 8
  - Yes: 4
  - Neutral: 0
  - Not really: 0
  - Not at all: 0

- Did the game make you feel motivated to move and try harder?
  - Yes, definitely: 5
  - Somewhat: 4
  - Not really: 3
  - No: 0

- Would you be willing to use this game again in future sessions?
  - Yes: 10
  - Maybe: 1
  - No: 1

Participants were also asked how likely would they be to recommend ReFlex VR to others for rehabilitation on a scale from 1 to 10: two of them said 7, four said 8, five said 10 and one patient said 11. Finally, all participants were asked to freely express any suggestions or comments about their experience, and their remarks were recorded: they mostly gave suggestions for possible new levels to add to ReFlex VR, among which there were household chores (cleaning plates, floors, folding clothes), gardening, hobby modeling/furniture assembling, classifying books, shopping at a supermarket, changing a tire or otherwise using work tools, painting, and doing sports (skiing, e-bike using hands instead of feet, playing table football).

From a technical standpoint, the overall setup time before the patients could play the game was of about 1 minute for three patients, 2 minutes for six of them, 5 minutes for two of them and 10 minutes for one patient. Only a minimal amount of technical assistance was needed for all twelve participants. The following are observations on physical and cognitive performance, engagement, and emotional response made by a therapist overlooking the patients playing the game, with a list of possible terms describing what has been observed and a number next to them indicating the amount of patients said term relates to:

- Motor engagement level (limb use, range of motion):
  - Full: 11
  - Partial: 1
  - Minimal: 0
- Fatigue or physical strain observed:
  - None: 10
  - Mild: 2
  - Moderate: 0
  - Severe: 0
- Cognitive load (e.g., confusion, trouble understanding a task):
  - None: 8
  - Mild: 4
  - Moderate: 0
  - High: 0
- Did the participant require physical assistance?
  - Yes: 1
  - No: 11

  If yes, please describe: the physiotherapist pushed the patient's torso forward to pick up the spoon in the kitchen level.

- Participant appeared:
  - Engaged: 11
  - Neutral: 1
  - Bored: 0
  - Frustrated: 0
  - Anxious: 0

Lastly, the therapist was asked to give their comments on the application: they found the game to be suitable for upper limb motor rehabilitation and they gave some suggestions to make the game more customizable in terms of difficulty and pacing.

Overall, the pilot was considered to be a success in terms of high user satisfaction, low adverse effects, and low setup time overhead.

6.2 VR Rehabilitation

Participants were divided in two randomly composed groups: seven of them would keep following their traditional rehabilitation as usual (control group), whereas the remaining six would combine the traditional therapy with two sessions of ReFlex VR per week (VR group). The training period lasted for five consecutive weeks; some tests were conducted on all of the participants to keep track of their motor skills both right before and right after the training period, so that the results could be compared and some conclusions could be drawn. The chosen tests were: Fugl-Meyer [43], Motor Activity Log-28 [45], Action Research Arm Test [28], Naturalistic Action Test [42] and Stroke Impact Scale 3.0 [46].

During each VR session, patients would be greeted by the application's creator and would be asked to sit on a chair in the middle of the room with no obstacles in front or around them; they would then have a couple of minutes of small talk to put them at ease and after that they would wear the headset and play one, two, or all three available levels, depending on the week. The following is a week-by-week breakdown of the levels that were played:

- Week 1: kitchen level
- Week 2: coffee level
- Week 3: kitchen level and coffee level
- Week 4: living room level
- Week 5: kitchen level, coffee level and living room level

Each VR session would conclude with the patient expressing whether they experienced any discomfort or pain or whether they had any additional comment. Some small issues with the levels were discovered during the training period and were then fixed in between sessions, such as the highest dirt spots and the farthest dust balls in the living room

level being too hard to reach for wheelchair-bound patients. One patient struggled with understanding where they were supposed to place the coffee capsule in the coffee level, blaming it on the fact that they have not used a coffee machine in over three years, therefore in order to make them figure it out the application's creator played the level in front of them before one of their sessions to show them where the capsule is meant to go. One of the patients frequently mentioned feeling fatigued and experiencing tiredness in their paretic limb after their VR sessions, but they said that the fatigue was absolutely bearable and not a cause of concern. Lastly, one of the patients said that in their day-to-day life they tend to always use their unaffected limb as it is easier for them, therefore they remarked that being somewhat forced to use their paretic limb to complete these tasks, albeit virtual, felt beneficial in a concrete way to them.

6.3 Tests and Comparisons

The points scored and the time spent for the VR training section of each session with the patients were manually recorded as well. Fig. 6.2 shows three graphs representing the progress of the kitchen level points, the coffee level points, and the living room level points across all of the sessions and color-coded based on each patient in the VR group. The x axis spans from the minimum possible score obtainable (0) to the maximum score that has been obtained by any patient for that level at least once. The moving average has also been computed, which is represented by the thinner lines going from the higher part to the bottom of each graph, each color illustrating the moving average for each patient. The moving average is a type of convolution that is often used in statistics to highlight long-term trends in data sets: in this case, it can be easily seen that there is a growing trend in the points for most participants, meaning that they got progressively better at completing the levels as the sessions went on. However, the gap between the first session and the last one is not wide enough to be considered statistically significant, for none of the levels.

The same can be said for the tests that have been conducted by professionals in the Centro Puzzle on the patients of both groups immediately before and immediately after the training period: the data has been analyzed by a collaborator from the University of Essex, coming to the conclusion that there has been a certain level of improvement in motor ability and social participation for all of the patients involved, and furthermore the results from the VR group have been slightly better than those from the control group, however, again, the difference is too modest to be statistically significant in any way.

These results have been attributed to many different causes, which have been summarized in the following list.

- All of the patients involved in the study happened to have reached a chronic stage in their motor impairments; this fact has not been by design but rather due to circumstances (the only patients available to participate at the time were selected). When motor impairments become chronic, improvements typically reach a plateau and it becomes much harder to gain back any amount of mobility.

- Each session should have probably lasted much longer: the VR training segment

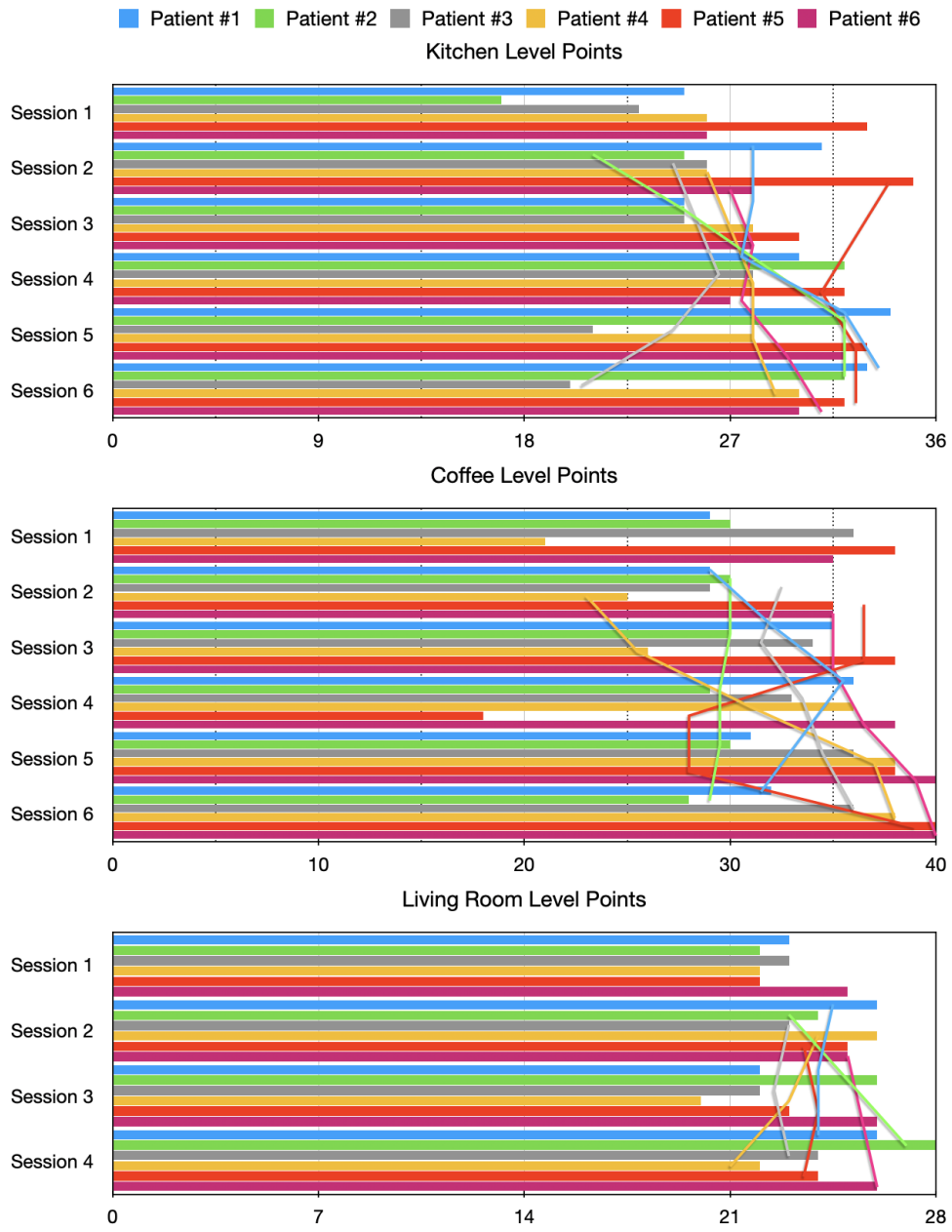


Figure 6.2. Three graphs showing the progress of, respectively, the kitchen level points, the coffee level points, and the living room level points

of each session lasted from a minimum of 5 minutes to a maximum of 35 minutes, with a mean value of about 10 minutes across all recorded sessions. This has been due to the fact that the approach taken to organize the sessions has been level-based rather than time-based, as highlighted by the week-by-week breakdown

in Section 6.2, however each level can be completed relatively quickly once the instructions have been figured out. In all of the literature on similar studies that has been analyzed, VR sessions lasted on average 45 minutes, the same amount of time spent for traditional physiotherapy sessions.

- On the same vein of the previous point, only two sessions of VR training per week might have been too scarce to allow for a significant impact.
- Lastly, the training period lasted only for five weeks, which is a shorter period than those reported in most literature on this topic. Even when discussing how to proceed with a group of healthcare professionals in the Centro Puzzle, they advised to make the training period last for at least eight consecutive weeks, however it had to be cut short due to time constraints both on the patients' side and on the thesis student's side.

Therefore, what has been done could be considered a feasibility and usability study, and in those regards it could also be deemed as a success; in the future another study could be conducted with ReFlex VR to try to assess this type of technology's potential in aiding stroke survivors in their rehabilitation journey. Said study, however, should try to include people that are in the early stages of their recovery (to avoid participants that have reached a chronic stage), it should prioritize having longer VR sessions by making patients re-do levels multiple times in the same session if necessary, and it should maximise the amount of VR sessions both on a weekly basis and with respect to the length of the training period itself.

Chapter 7

Conclusions and Future Works

ReFlex VR has been developed with the express purpose of making upper limb motor rehabilitation for stroke survivors more affordable, practical, and engaging, so as to entice them to keep practicing even when back in their own homes to improve their quality of life and lessen the burden on their loved ones. As previously stated, further studies need to be done to establish the application's effectiveness in conjunction with traditional physiotherapy or even on its own, however it has already been proven to be well accepted and liked by patients and to not cause any side effects of note.

Aside from performing more in-depth studies, the application itself could be further expanded by adding more levels, perhaps taking inspiration from the suggestions that patients have come up with during the pilot of this thesis' study: for example, a garage environment could be created, in which users could exercise their upper limbs by changing a motorcycle's tires, or they could practice their fine motor skills by building some furniture, trying to secure different pieces together with screws with as much precision as possible. Another environment that is yet to be explored is the bedroom, where patients could re-learn real world daily activities by making the bed or folding clothes, or even doing the laundry by placing clothes in the machine, carefully filling the laundry soap container without spilling, and choosing the correct program by pressing some buttons. Additionally, even some tasks outside of the home could be devised, by making patients tend to their gardens with many different tools or mowing the lawn.

An aspect that has yet to be explored in ReFlex VR, and that could be expanded upon by future students, is the integration of AI (Artificial Intelligence) into the level design: for example, in the hypothesis of patients using the application in their own homes, the Meta Quest 3's advanced sensors could be employed to scan the user's real life environment and then procedurally create a one-to-one virtual reproduction, in order to further leverage the patients' sense of familiarity. Alternatively, an AR (Augmented Reality) version of the application could be devised, still by scanning the user's rooms, then still showing them through the headset's built-in cameras but adding the virtual objects that are required to complete the levels' tasks. By integrating the player's actual, tangible environments into the application, not only will they truly feel like they are in their own homes but they might gain some cognitive benefits: for example, for the kitchen level the counter top would not be simply a virtual illusion anymore, but rather

a material object that they can actually touch and rest their arms on top of to do the required tasks.

As mentioned in Chapter 6, for the experiment the amount of time spent by patients to complete each level has been recorded manually to then be analyzed; another rather simple and quick fix to the application could be to record in the JSON file relative to each user the time as well as the points, and this new information could also be shown to them in the victory UI, in order to give them further information on their progress and add a new dimension to the challenge of the game by inducing them to try to beat their latest time score while still performing the different tasks as accurately as possible to gather more points.

Lastly, some advancements or modifications could also be made from a gamification perspective. In his famous article from 1996, Bartle [37] conducted a study on people that played multi-user dungeons (MUD) and derived four macro-categories of players, each with their own objectives, interests and motivations:

- "achievers" are interested more in the virtual environment they find themselves in rather than other players, and as such they typically prefer to play by themselves. They tend to put time and effort towards reaching the goals set by the game or even by themselves, and they enjoy perfecting their in-game capabilities, becoming experts;
- "explorers" are still interested more in the virtual world than any perspective other players, but they do enjoy interacting with them. What drives these types of players is curiosity: they revel in exploring, as the name suggests, whether it be new areas of the map, new levels or new events within the game;
- "socializers" are more interested in other players rather than the virtual environment. They tend to get along with people easily and see games as an opportunity to interact with new people and make friends; they do not care for games that prioritize competition or in-game achievements. They go for games with a strong social component;
- "killers" are interested in other players more than in the virtual environment much like the socializers, however unlike them it is not to get along and make friends but rather to compete and win against them. They prefer acting alone rather than collaborating and enjoy challenges and high-adrenaline games.

Each type of player prefers certain kinds of games and avoids others. Because of how it was implemented, ReFlex VR is very much achiever-oriented (it is a single-player game where there is not much exploration to speak of and the emphasis is put on completing levels and getting as many stars as possible), but obviously not every stroke survivor is going to find such a game completely to their taste. Therefore, in order to make this type of technology useful for the widest range of people, different kinds of games could be developed to appeal to every player type: for example, a game that involves multiple patients collaborating to reach a certain goal could be a better fit for socializers, whereas a game where patients directly compete against each other through some kind of gamified upper limb exercise could further motivate killers. In any case, adding a social component

to that effect would make the perspective game less feasible in a home setting (unless the multiplayer aspect is implemented through a cloud system), but it could further benefit patients from a socializing and cognitive standpoint.

The possibilities in this field are many and there is still a lot of unexplored territory, mainly due to the fact that this type of technology is relatively new, but it is becoming increasingly affordable and commercially available: hopefully researchers and neuroscience experts will be able to definitively prove that Virtual Reality is a viable addendum or even an alternative to at-home traditional rehabilitation programs, so that people that have suffered a stroke or any other kind of ABI can have fun playing an engaging game while gradually regaining their motor skills and improving their quality of life.

Chapter 8

Glossary

- **ABI** Acquired Brain Injury
- **ADL** Activities of Daily Living
- **AF** Augmented Feedback
- **AI** Artificial Intelligence
- **HMD** Head-Mounted Display
- **IVR** Immersive Virtual Reality
- **NIVR** Non-Immersive Virtual Reality
- **OVR** Oculus Virtual Reality
- **ROM** Range Of Movement
- **SDK** Software Development Kit
- **UE** Upper Extremity / Upper Extremities
- **UI** User Interface
- **VR** Virtual Reality

Bibliography

[1] Pollock A., Farmer S.E., and Brady M.C. et al. Interventions for improving upper limb function after stroke. *Cochrane Database of Systematic Reviews*, 2014.

[2] Roby-Brami A., Jarrasse N., and Parry R. Impairment and compensation in dexterous upper-limb function after stroke. from the direct consequences of pyramidal tract lesions to behavioral involvement of both upper-limbs in daily activities. *Frontiers in Human Neuroscience*, 2021.

[3] Rojo A., Santos-Paz J.A., Sanchez-Picot A., Raya R., and Garcia-Carmona R. Farmday: A gamified virtual reality neurorehabilitation application for upper limb based on activities of daily living. *Applied Sciences*, 2022.

[4] Tessari A., Mengotti P., Faccioli L., Tuozzi G., Boscarato S., and Taricco M. et al. Effect of body-part specificity and meaning in gesture imitation in left hemisphere stroke patients. *Neuropsychologia*, 2021.

[5] Todhunter-Brown A., Baer G., Campbell P., Choo P.L., Forster A., Morris J., Pomeroy V.M., and Langhorne P. Physical rehabilitation approaches for the recovery of function and mobility following stroke. *Cochrane DatabaseSyst Rev.*, 2014.

[6] Sousa A.S.P., Moreira J., Silva C., Mesquita I., Macedo R., Silva A., and Santos R. Usability of functional electrical stimulation in upper limb rehabilitation in post-stroke patients: A narrative review. *Sensors*, 2022.

[7] Frasca D., Tomaszczyk J., McFayden B.J., and Green R.E. Traumatic brain injury and post-acute decline: what role does environmental enrichment play? a scoping review. *Front Hum Neurosci*, 2013.

[8] Mekbib D.B., Debeli D.K., Zhang L., Fang S., Shao Y., Yang W., Han J., Jiang H., Zhu J., and Zhao Z. et al. A novel fully immersive virtual reality environment for upper extremity rehabilitation in patients with stroke. *Annals of the New York Academy of Sciences*, pages 75–89, 2021.

[9] Gibson E., Koh C.L., Earnes S., Bennett S., Scott A.M., and Hoffman T.C. Occupational therapy for cognitive impairment in stroke patients. *Cochrane Database Syst Rev*, 2022.

[10] Dimbwadyo-Terrer I. et al. Effectiveness of the virtual reality system toyra on upper limb function in people with tetraplegia: a pilot randomized clinical trial. *BioMed Res. Int.*, 2016.

[11] Rose F., Attree E., and Johnson D. Virtual reality: an assistive technology in neurological rehabilitation. *Current Opinion in Neurology*, pages 461–468, 1996.

[12] Institute for Health Metrics and Evaluation (IHME). Gbd compare data visualization. *IHME Website*, 2016.

[13] Fernandes Cyrino G. *Harpy Game: um jogo serio customizavel com interface multimodal*. PhD thesis, 2019.

[14] Goncalves G., Coelho H., Monteiro P., Melo M., and Bessa M. Systematic review of comparative studies of the impact of realism in immersive virtual experiences. *ACM Comput Surv.*, 2022.

[15] Kwakkel G., Veerbeek J.M., van Wegen E.E., and Wolf S.L. Constraint-induced movement therapy after stroke. *Lancet Neurol.*, 2015.

[16] Ase H., Honaga K., Tani M., Takakura T., Wada F., Murakami Y., Isayama R., Tanuma A., and Fujiwara T. Effects of home-based virtual reality upper extremity rehabilitation in persons with chronic stroke: a randomized controlled trial. *Journal of NeuroEngineering and Rehabilitation*, 2025.

[17] Thielen H., Tuts N., Welkenhuyzen L., Huenges Wajer I.M.C., Lafosse C., and Gillebert C.R. Sensory sensitivity after acquired brain injury: a systematic review. *J Neuropsychol.*, 2023.

[18] Hsu H.Y. and Kuo L.C. et al. Effects of a virtual reality-based mirror therapy program on improving sensorimotor function of hands in chronic stroke patients: a randomized controlled trial. *Neurorehabilitation and Neural Repair*, pages 335–345, 2022.

[19] Alves Marques I., Marques Alves C., Rastrelo Rezende A., Cardoso Mendes L., Sa de Paiva T., Fernandes Cyrino G., Tannus de Souza J., Maia Silva M.A., Pascucci Sande de Souza L.A., and Martins Naves E.L. Virtual reality and serious game therapy for post-stroke individuals: A preliminary study with humanized rehabilitation approach protocol. *Complementary Therapies in Clinical Practice*, 2022.

[20] Robsbergen I.C.M., Grimley R.S., Hayward K.S., and Brauer S.G. The impact of environmental enrichment in an acute stroke unit on how and when patients undertake activities. *Clin Rehabil.*, 2019.

[21] Rogers J.M., Duckworth J., Middleton S., Steenbergen B., and Wilson H.P. Elements virtual rehabilitation improves motor, cognitive, and functional outcomes in adult stroke: evidence from a randomized controlled pilot study. *Journal of Neuro-Engenineering and Rehabilitation*, 2019.

[22] Goldman L., Siddiqui E.M., Khan A., Jahan S., Rehman M.U., Mehan S., Sharma R., Budkin S., Kumar S.N., Sahu A., Kumar M., and Vaibhav K. Understanding acquired brain injury: a review. *Biomedicines*, 2022.

[23] Huang L.L. and Chen M.H. Effectiveness of the immersive virtual reality in upper extremity rehabilitation. In *Proceedings of the Cross-Cultural Design, Applications in Health, Learning, Communication, and Creativity: 12th International Conference*, pages 89–98, 2020.

[24] Harrison M., Ryan T., Gardiner C., and Jones A. Psychological and emotional needs, assessment, and support post-stroke: a multi-perspective qualitative study. *Top Stroke Rehabil.*, pages 119–125, 2017.

[25] Hinwood M., Ilicic M., Gyawali P., Kluge M.G., Coupland K., Smith A., Nilson M., and Walker F.R. Exploration of stress management interventions to address psychological stress in stroke survivors: a protocol for a scoping review. *BMJ Open*, 2020.

[26] Maier M., Ballester B.R., and Verschure P. Principles of neurorehabilitation after stroke based on motor learning and brain plasticity mechanisms. *Front Syst Neurosci*, 2019.

[27] Rodriguez-Hernandez M., Polonio-Lopez B., Corregidor-Sanchez A., Martin-Conty J.L., Mohedano-Moriano A., and Criado-Alvarez J. Can specific virtual reality combined with conventional rehabilitation improve poststroke hand motor function? a randomized clinical trial. *Journal of NeuroEngineering and Rehabilitation*, 2023.

[28] Michelle McDonnell. Action research arm test. *Australian Journal of Physiotherapy*, 54(3):220, 2008.

[29] Ogun M.N., Kurul R., Yasar M.F., Turkoglu S.A., Avci S., and Yildiz N. Effect of leap motion-based 3d immersive virtual reality usage on upper extremity function in ischemic stroke patients. *Arq. Neuropsiquiatr.*, pages 681–688, 2019.

[30] Einstad M.S., Saltvedt I., Lydersen S., Ursin M.H., Munthe-Kaas R., Ihle-Hansen H., Knapskog A.B., Askim T., Beyer M.K., Naess H., Seljeseth Y.M., Ellekjaer H., and Thingstad P. Associations between post-stroke motor and cognitive function: a cross-sectional study. *BMC Geriatr.*, 2021.

[31] Townsend N., Kazakiewicz D., and Lucy Wright F. et al. Epidemiology of cardiovascular disease in europe. *Nature Reviews Cardiology*, pages 133–143, 2022.

[32] Wenk N., Buetler K.A., Penalver-Andres J., Muri R.M., and Marchal-Crespo L. Naturalistic visualization of reaching movements using head-mounted displays improves movement quality compared to conventional computer screens and proves high usability. *Journal of NeuroEngenineering and Rehabilitation*, 2022.

[33] Nahid Norouzi-Gheidari, Alejandro Hernandez, Philippe S. Archambault, Johanne Higgins, Lise Poissant, and Dahlia Kairy. Feasibility, safety and efficacy of a virtual reality exergame system to supplement upper extremity rehabilitation post-stroke: A pilot randomized clinical trial and proof of principle. *International Journal of Environmental Research and Public Health*, 17(1), 2020.

[34] Langhorne P., Bernhardt J., and Kwakkel G. Stroke rehabilitation. *The Lancet*, 2011.

[35] Shah-Basak P., Boukrina O., Li X.R., Jebahi F., and Kielar A. Targeted neurorehabilitation strategies in post-stroke aphasia. *Restor Neurol Neurosci*, pages 129–191, 2023.

[36] Qiu Q., Cronce A., Patel J., Fluet G.G., Mont A.J., Merians A.S., and Adamovich S.V. Development of the home based virtual rehabilitation system (hovrs) to remotely deliver an intense and customized upper extremity training. *Journal of NeuroEngineering and Rehabilitation*, 2020.

[37] Bartle R. Hearts, clubs, diamonds, spades: Players who suit muds. *J. MUD Res.*, 1996.

[38] Gammeri R., Iacono C., Ricci R., and Salatino A. Unilateral spatial neglect after stroke: current insights. *Neuropsichiatr Dis Treat*, pages 131–152, 2020.

[39] Psenicnik Sluga S. and Kozinc Z. Sensorimotor and proprioceptive exercise programs to improve balance in older adults: a systematic review with meta-analysis. *Eur J Transl Myol*, 2024.

[40] Ventura S., Tessari A., Castaldini S., Magni E., Turolla A., Banos R., and Lullini G.

Effectiveness of a virtual reality rehabilitation in stroke patients with sensory-motor and proprioception upper limb deficit: A study protocol. *PLoS ONE*, 2024.

[41] Wallwork S.B., Bellan V., Catley M.J., and Moseley G.L. Neural representations and the cortical body matrix: implications for sports medicine and future directions. *British Journal of Sports Medicine*, 2016.

[42] Myrna F. Schwartz, Mary Segal, Tracy Veramonti, Mary Ferraro, and Laurel J. Buxbaum. The naturalistic action test: A standardised assessment for everyday action impairment. *Neuropsychological Rehabilitation*, 12(4):311–339, 2002.

[43] Barbara Singer and Jimena Garcia-Vega. The fugl-meyer upper extremity scale. *Journal of Physiotherapy*, 63(1):53, 2017.

[44] Cucinella S.L., De Winter J.C.F., Grauwmeijer E., Evers M., and Marchal-Crespo L. Towards personalized immersive virtual reality neurorehabilitation: a human-centered design. *Journal of NeuroEngenineering and Rehabilitation*, 2025.

[45] G. Uswatte, E. Taub, D. Morris, K. Light, and P. A. Thompson. The motor activity log-28. *Neurology*, 67(7):1189–1194, 2006.

[46] Ercole Vellone, Serenella Savini, Roberta Fida, Victoria Vaughan Dickson, Gail DâEramo Melkus, Francisco Javier Carod-Artal, Gennaro Rocco, and Rosaria Alvaro. Psychometric evaluation of the stroke impact scale 3.0. *Journal of Cardiovascular Nursing*, 30(3):229–241, 2015.

[47] Feigin V.L., Brainin M., Norrving B., Martins S., Sacco R.L., Hacke W., Fisher M., Pandian J., and Lindsay P. World stroke organization (wso): global stroke fact sheet 2022. *Int J Stroke*, pages 18–29, 2022.

[48] Parker V.M., Wade D.T., and Langton H.R. Loss of arm function after stroke: measurement, frequency, and recovery. *Int Rehabil Med*, pages 69–73, 1986.

[49] Zhang Y., Xing Y., Li C., Hua Y., Hu J., Wang Y., Ya R., Meng Q., and Bai Y. Mirror therapy for unilateral neglect after stroke: A systematic review. *European Journal of Neurology*, 29(1):358–371, 2022.