Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSITY OF VALLADOLID

SCHOOL OF INDUSTRIAL ENGINEERING

Final Project International Semester

Development of a Hand Gesture Detection System Based on MediaPipe for Neuromotor and Cognitive Rehabilitation Using Augmented Reality on the M3Display Platform

Author:

Herasymova, Vladyslava

Tutors:

Fuente Lopez, Eusebio De La
Departamento de Ingeniera de Sistemas y Automatica;

Rica, Ana Cisnal De La
Departamento de Ingeniera de Sistemas y Automatica

Valladolid, 05 2025

ABSTRACT

This Final Project focuses on the design and development of a serious augmented reality-based game for rehabilitation of upper-limb mobility and fine motor functions. The game-based approach to rehabilitation aims to make the recovery process more engaging and enjoyable for the patient, and the usage of augmented reality aims to facilitate this effect by making the game more immersive.

The game is developed using Unity game engine and C# programming language. As its base, the game uses M3Display - an augmented reality-based system for the rehabilitation of upper limbs functionality and mobility, together with the real-time hand segmentation solution previously implemented for this system and based on the MediaPipe neural network.

KEYWORDS

Augmented reality, hand gesture detection, M3Display, rehabilitation, serious game.

MAIN INDEX

1. INTRODUCTION AND OBJECTIVES 9
1.1. Introduction 9
1.2. Objectives 11
2. LITERATURE REVIEW 13
3. TECHNOLOGIES USED 23
3.1. Unity 23
3.1.1. Introduction and overview 23
3.1.2. Main principles 24
3.1.2.1. Scenes 24
3.1.2.2. Game Objects 25
3.1.2.3. Components 26
3.2. M3Display 26
3.2.1. Introduction and history 26
3.2.2. Hardware 27
3.2.3. Software 28
3.2.3.1. Serious games 28
3.2.3.2. Hand detection algorithm 29
4. GAME STRUCTURE 33
4.1. General concept 33
4.2. Game windows 33
4.2.1. Main Menu 33
4.2.2. Image Gallery 34
4.2.3. Single Gallery Image 35
4.2.4. User Identification 36
4.2.5. Duration Selection 36
4.2.6. Hands Mode Selection 37
4.2.7. Landmarks Initial Calibration 38
4.2.8. Table Size Selection 39
4.2.9. Difficulty Initial Selection 40
4.2.10. Image Selection 41
4.2.11. Gameplay 42
4.2.12. Settings Scene 44
4.2.13. Landmarks Settings Calibration 45
4.2.14. Game End 46
4.2.15. Loading 46
4.3. Important additional elements 47

4.3.1. Persistent Data 47
4.3.2. Scene loader 48
4.3.3. Database 48
4.3.4. CSV data 49
4.5. Python scripts 49
4.6. Resulting data 50
4.7. Historical advancements and past versions 50
5. CONCLUSIONS 53
5.1. Degree of implementation 53
5.2. Possible future improvements 53
BIBLIOGRAPHY 55
ANNEX I – LandmarksGameManager Code 57
ANNEX II – Objects Code 61
ANNEX III – VideoFeed Code 65
ANNEX IV – FistTemplate Code 69
ANNEX V – PuzzleAssemblySpace Code 71
ANNEX VI – PuzzlePiece Code 73
ANNEX VII – ImageManager Code 75
ANNEX VIII – ImagesPython Code 79
ANNEX IX – Database Code 81

INDEX OF FIGURES

Figure 1. Rehabilitation of upper limbs mobility with AR with (right) and without (left) handheld roller-ball device..... 14
Figure 2. 'Whac-A-Mole' AR game for upper limb rehabilitation..... 15
Figure 3. Ball manipulation AR game for upper limb rehabilitation..... 15
Figure 4. Schematics of (a) PC display-based and (b) projector-based AR versions of the Fruit Ninja game..... 16
Figure 5. Serious games for upper limb mobility rehabilitation: (a) sweeping the crumbs from the table, (b) carrot grating, (c) knocking on doors, (d) cooking, (e) squeezing a sponge, (f) dialing a number, (h) playing piano, (h) buying items..... 17
Figure 6. Hardware for the cellphone augmented reality rehabilitation system: iPhone XR, iOS 13 (left), phone case with built-in fingerless glove (center); demonstration of the hardware usage (right)..... 18
Figure 7. Three augmented reality–based serious games for rehabilitation of upper limb motor function and cognitive function: Pyramid Reach (left), Add VS Sub (center), Stroop Game (right)..... 18
Figure 8. VR serious games for upper-limb mobility rehabilitation: (a) item selection, (b) knocking down structures with spheres, (c) picking apples from a tree, (d) putting a ball in a box, (e) picking cabbages in the garden, (f) pouring a cup of tea..... 19
Figure 9. Shared AR environment from the field of view of the (a) therapist's and (b) patient's headsets..... 20
Figure 10. Framework setup for a shared between patient and therapist AR-based system for upper limb mobility rehabilitation..... 20
Figure 11. The section of the Build Settings window with the list of added scenes..... 24
Figure 12. Game Object hierarchy..... 25
Figure 13. M3Display hardware setup: (a) full system; (b) mirror configuration; (c) final video feed with hand continuity on the screen..... 28
Figure 14. M3Display user interface..... 29
Figure 15. Some of the serious games developed for the M3Display: (a) Space, (b) Ball interaction, (c) Mouse, (d) Clock..... 29
Figure 16. Stages of hand segmentation..... 30
Figure 17. Random samples from the evaluation sequences of Ego2Hands..... 31
Figure 18. Main Menu scene..... 34
Figure 19. Image Gallery scene..... 34
Figure 20. Single Gallery Image scene..... 35

Figure 21. Single Gallery Image scene: delete image menu..... 35
Figure 22. User Identification scene..... 36
Figure 23. Duration Selection scene..... 37
Figure 24. Hand Mode Selection scene..... 37
Figure 25. MediaPipe hand landmarks..... 38
Figure 26. Landmarks Initial Calibration scene before the fist capture.... 39
Figure 27. Landmarks Initial Calibration scene after the fist capture..... 39
Figure 28. Table Size Selection scene..... 40
Figure 29. Difficulty Initial Selection scene..... 41
Figure 30. Image Selection scene..... 41
Figure 31. Gameplay scene..... 42
Figure 32. Gameplay scene: image detailed view menu..... 43
Figure 33. Gameplay scene: pause menu..... 44
Figure 34. Settings scene..... 45
Figure 35. Landmarks Settings Calibration scene..... 45
Figure 36. Game End scene..... 46
Figure 37. Loading scene..... 47
Figure 38. Example of a generated image with the hand movement trajectory..... 49

1. INTRODUCTION AND OBJECTIVES

1.1. Introduction

According to World Stroke Organization (WHO), 12 million people have a stroke each year, and about 1 in 4 people will suffer a stroke in their lifetime [1]. A stroke occurs when the blood supply to a part of the brain is either interrupted or reduced, which can be caused by the lack of blood flow (caused by the blockage in an artery supplying blood to the brain) or due to the internal bleeding in the brain (blood vessel in the brain ruptures). This may lead to a variety of complications depending on the part of the brain that is affected and the extent of damage to the brain tissue. Those complications may include [2]:

1. Physical (stiff or rigid muscles, fatigue, incontinence, issues related with muscle coordination and balance, numbness, chronic pain, decreased range of motion, weakness in or inability to move one side of the body, insomnia, seizures, disorder of fine motor skills, etc.);
2. Communicational (jumbled or fragmented speech, weakened control over pronunciation and loudness of the voice, reading difficulties, etc.);
3. Cognitive (problems with concentration and attention, confusion, forgetfulness, etc.);
4. Emotional and behavioral (depression, personality changes and mood swings, etc.);
5. Vision and hearing related (blind spots in the visual space, doubled vision and loss of depth perception, auditory overload, etc.).

Because of the drastic nature of the effects strokes can have on a person's life, rehabilitation is one of the most important things a stroke survivor may go through. It may include many processes depending on how exactly stroke affected the body and its main goal is to help the patient to regain and relearn the skills of everyday life. Such rehabilitation tactics may include [3]:

1. Physical therapy: focuses on improving mobility, strength and balance of a patient;
2. Occupational therapy: focuses on relearning daily activities and adaptation to new physical limitations;
3. Cognitive rehabilitation: focuses on cognitive functions and daily tasks, has a goal-oriented approach;
4. Speech and language therapy: focuses on difficulties with speaking, reading, writing and understanding languages;
5. Orthoses: focuses on providing support in patient's body's stability and alignment, as well as pain relief during physical therapy;
6. Usage of other assistive devices: canes, wheelchairs, walkers, etc.;

7. Alternative therapies: involves acupuncture, music and dance therapy, performing exercises under water, mirror therapy and so on.

Among the complications caused by a stroke, impairments in upper-limb mobility — affecting the arms, hands, and shoulders — are particularly common. Damage to the brain areas controlling these limbs can result in muscle weakness, stiffness, or even paralysis, as well as a loss of fine motor skills and chronic pain caused by weakened muscles. These challenges significantly impact patient's ability to perform essential every-day activities and lead a normal life, making upper-limbs rehabilitation a critical aspect of one's recovery. Tactics used in upper-limbs rehabilitation include physical and occupational therapy, as well as usage of orthoses and other assisting devices.

Technological advancement in recent years allowed for the emergence of a new alternative therapy tactic highly applicable for upper-limbs rehabilitation – robotic-based therapy. It uses robotic devices to assist, guide, and monitor movement in patients with motor impairments, such as those recovering from stroke, as well as spinal cord injuries or neurodegenerative diseases. Introduction of robotic-based therapy gives hope to healthcare and social assistance systems which are suffering from the ever-growing pressure due to the ageing population and the reduction in the therapist-to-patient ratio it causes. Other important benefits of robotic-assisted therapy include its adaptability, the resistance robotic elements can provide for muscles' strength training, as well as their ability to measure the patient's speed, force and range of motion.

Other technologies which were introduced in recent years and are currently being integrated into rehabilitation processes are virtual reality and augmented reality. Virtual reality (VR) is a technology that generates a digital environment and immerses the user into that environment using additional hardware (usually - headsets). Augmented reality (AR) is a technology that overlays digital elements onto the real world in real time and, as much as VR, requires additional hardware to use (usually mobile phones, tablets or glasses). Both VR and AR are used in robotic-assisted therapy, mostly in the form of games, to make the sessions more engaging and rewarding.

Despite all of its many advantages, robotic-assisted therapy has a pretty significate drawback – the robotic elements used for such therapy are usually rather pricy, which limits the widespread of their usage even in hospitals, let alone at home. To preserve as many benefits as possible while significantly cutting the cost, an attempt can be made to redistribute the job previously done by the robotic components to the software already used for VR and AR generation. That will require a way to monitor where and in which position the patient's limbs are in real time, which can be done using a camera in combination with object detection algorithms. This approach will preserve most

of the core benefits of robotic-assisted therapy, including lowered need for supervision from a therapist, monitoring of main characteristics of a patient's limb movements, possible adaptability and engaging nature provided by game-based therapy sessions, as well as a significant cost-result improvement granted by the absence of necessity for a highly specialized hardware. Aside from the aforementioned benefits, this approach also has its drawbacks, including lower level of accuracy in determining the hand position and movements, as well as worse overall performance caused by higher computation cost.

Another issue that should be taken into account is the discomfort and sickness some users experience while immersing in VR. Given that a person's brain is already damaged by the stroke, such side effects are undesirable and dangerous, since they can slow down the recovery process and cause other complications. This makes AR-based solutions more suitable, however they still require adaptation and should be used carefully, especially in cases when the damage to the other parts of the brain is either uncertain or confirmed to be present.

Augmented reality is still a new idea in the rehabilitation field, especially in relation to upper-limbs mobility, so the amount of such initiatives is very limited. One of such initiatives is M3Display – an AR system specifically designed for upper limb rehabilitation therapy and developed by the Institute of Advanced Production Technologies Medical Robotics Research Group. M3Display uses game-based approach to therapy, making sessions both engaging and more interesting for the patient. The real-time hand segmentation solution used in M3Display was developed in ITAP Medical Robotics Research Group [4], [5]. As its core, it uses MediaPipe – a highly optimized neural network developed by Google for hands landmark detection [5], [6].

1.2. Objectives

The main objective of this final project is to develop an AR game leveraging the aforementioned technologies and solutions to facilitate the fine motor skills recovery in a patient's upper limbs after suffering a stroke. The skills that will be included are the overall hand mobility, as well as fingers' flexion and extension (bending and straightening the fingers to either close or open the fist). The game developed will target those skills by presenting the player with pieces of a puzzle that need to be assembled to form a final picture. Those pieces must be grasped, moved and placed to their correct position by the patient. The game must provide the right amount of challenge to keep the user engaged but not so much so that playing it becomes discouraging for the patient. Considering that users playing the game may have different levels of disability, the difficulty

of the game must be adjustable, which may include, for example, the size of puzzle pieces and the precision of grasp. The strength at which a player can flex his or her fingers must be also taken into consideration in the form of tuning the system before the game is started.

To reach the goal of this research and develop the game described above, the following specific objectives must be achieved:

1. Conducting a literature review on the topic in general and the technologies used, including M3Display and Unity;
2. Implement the algorithm for gesture recognition;
3. Implement and test the beta-version of the game with reduced functionality;
4. Include grasp strength calibration, difficulty levels and gameplay area size selection;
5. Include additional game elements, such as score, puzzle images, therapy session duration;
6. Include gathering of patient's information and further graph representation of his or her progress over a selected span of time.

2. LITERATURE REVIEW

Although augmented reality (AR) remains a relatively novel technology in the field of physical rehabilitation, recent years have witnessed a growing interest in its usage to support motor recovery and improve therapeutic outcomes. Researchers have begun exploring AR-based systems for a variety of neurological and musculoskeletal conditions, including stroke, which is the main topic of current research, as well as Parkinson's disease, limb loss, balance impairments, tremors, and other motor pathologies. Current research in AR-assisted rehabilitation tends to adopt a broader perspective, often focusing on general upper-limb or hand motor function rather than having a specific disability as its core [7].

Most AR systems aim to support motor skill acquisition by integrating motion tracking and repetitive interactive tasks into user-centered experiences. The sensor technologies used in these systems are diverse and include SLAM (Simultaneous Localization and Mapping) systems, RGB-D and conventional cameras, as well as biosensors such as electromyography (EMG), force platforms, and inertial measurement units (IMUs) [7].

In terms of hardware used to display the digital components to the user, the most commonly used solution in the industry would be head-mounted displays (HMDs). With continuous technological advancements their popularity keeps rising. At the same time, it is important to remember that the usage of such headsets may cause sickness and discomfort in some patients, which is especially dangerous with strokes due to the already present brain damage. Such headsets can also be confusing for people of age. Monitor-based solutions remain a secondary choice and are followed by projector-based solutions and solutions that use more than one display type [7].

To put into perspective everything mentioned above, it is helpful to examine specific AR-based rehabilitation systems that have been developed and studied in recent years. Taking into consideration the main objectives of current research stated earlier, the main objects of examination would be AR and VR-based serious games for upper-limb rehabilitation. These systems are specifically designed to support motor recovery by combining task-oriented therapy with engaging, interactive game elements. While they vary in terms of design, target disabilities, modes of input gathering and ways of displaying the resulting digitally modified image, they all share a common objective of encouraging consistent and repetitive movement practice in an immersive and motivating environment. The researches in the list below are organized in a chronological order to paint a picture of how AR game-based systems evolved with time in addition to the overview of technologies they use and motor functionality they focus on.

One of the earliest instances of usage of augmented reality in physical rehabilitation is a serious game developed in 2009 by M. King et al. [8]. It used an overhead mounted web camera and a computer vision algorithm AR Toolkit to track the position of a marker attached to the user's hand or wrist and required the user to move his or her hand to catch the digitally generated butterflies with a net. Aside from the webcam used to capture the live video, the only other hardware required is the PC on which the program runs and displays the images.

Another early example of AR usage in rehabilitation is the software developed by N. I. De Leon et al., from the Sydney University of Technology [9]. It includes two games and was developed in 2014 using Adobe Flash CS5 and Adobe Flash builder 4.7 using coding language Actionscript 3.0 and augmented reality framework FLARManager. The developed system does not require any additional hardware – it uses a simple PC webcam to capture the video in real time and displays the resulting images with digitally added objects to the display of the aforementioned PC (Fig. 1, left). There is an option, however, to use some additional hardware, including a handheld roller-ball device and additional constructions to make the camera face downwards instead of sideways. This will drastically change the way the game is played since the movements performed with the roller-ball device on top of the table surface require other muscles in comparison to the free arm movement without it (Fig. 1, right).

Figure 1. Rehabilitation of upper limbs mobility with AR with (right) and without (left) handheld roller-ball device [9].

The first game included in the software was modeled after a famous arcade game 'Whac-A-Mole' (Fig. 2). When combined with the roller-ball device, this game focuses on elbow extension and flexion. Without the device, the focus switches to shoulder abduction and forward flexion. The game also tries to maximize the movement that the player needs to perform by placing the targets that the player needs to hit in 4 different corners of the screen.

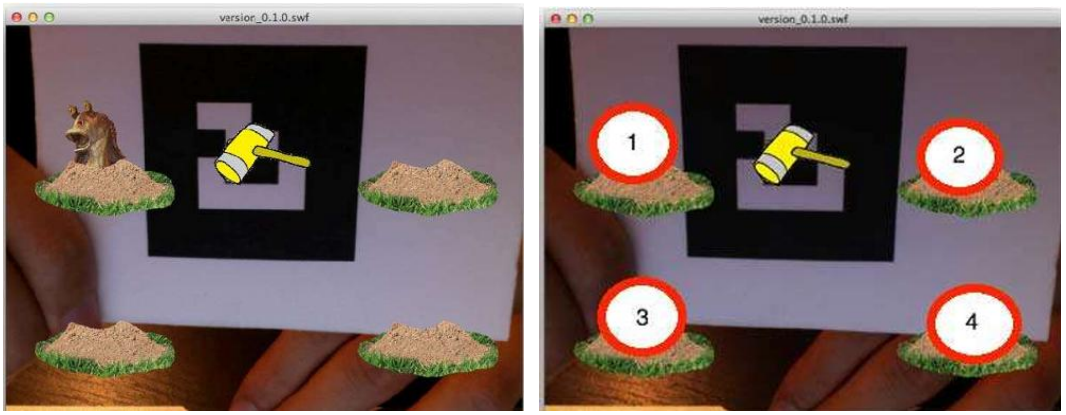

Figure 2. 'Whac-A-Mole' AR game for upper limb rehabilitation [9].

The second game requires the user to hit a generated ball towards the wall with the same color using a paddle (Fig. 3). When using the roller-ball device, the main focus of the game is on hand muscles through radial deviation, as well as on shoulder medial and lateral rotation. Without the device, the focus is on the forearm muscles through supination & pronation. Some shoulder abduction and flexion are involved regardless of the usage of the device.

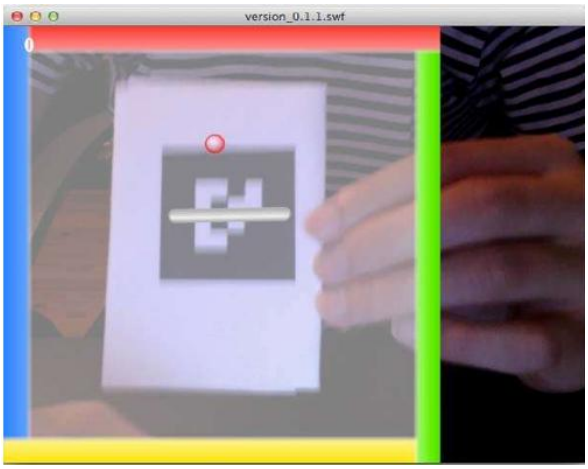

Figure 3. Ball manipulation AR game for upper limb rehabilitation [9].

A year later, in 2015, research was conducted by H. M. Hondori et al. on the topic of the optimal interaction mode for stroke rehabilitation [10]. It compared two ways of playing an AR version of Fruit Ninja game – one with a PC display, and another with an augmented reality and a projector (Fig. 4).

The research demonstrated a significant difference in performance with AR solution providing a much better experience. It is worth noting that in a PC

version of the game, the hand movement was not visible to the subject directly – you could only see the projection of the arm-controlled element on the screen, which makes its operation much more difficult as it requires additional cognitive effort to couple the movement of the element on the screen with the movement of one's hand on the table 45 degrees away from the player's direction of gaze.

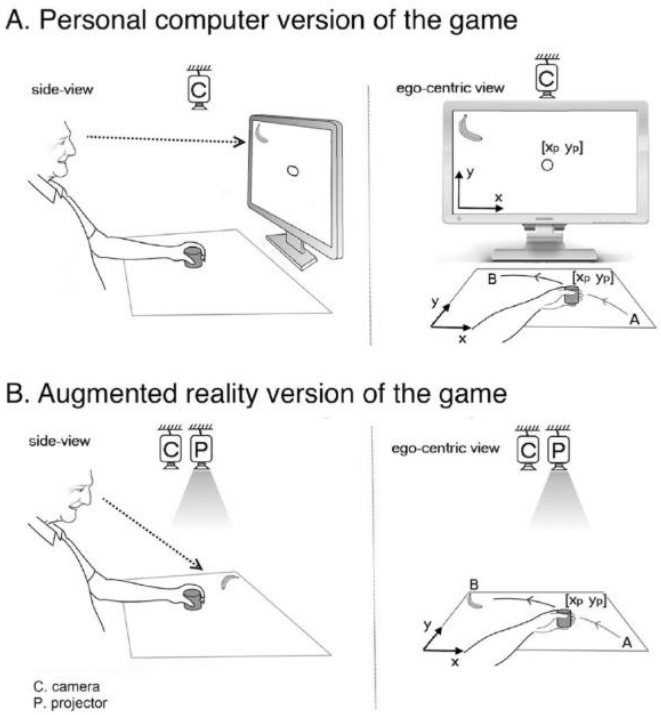

Figure 4. Schematics of (a) PC display-based and (b) projector-based AR versions of the Fruit Ninja game [10].

A wide range of serious games was developed by C. Colomer et al. [11]. The hardware they used included a depth sensor and a projector, as well as a table surface and a computer on which the game itself runs. The projector displayed the game elements on the table surface that the player could later interact with (Fig. 5). In some games, users were interacting with a system using an additional tangible object. The system was focused on the flexion and extension of the elbow, the wrist, and the metacarpophalangeal joint, and presented the players with exercises focused on daily tasks.

The exercises had to be performed as many times as possible and would be considered finished if a number of repetitions were performed accurately enough within a time interval. The system included difficulty adjustments in the form of choosing the required speed, number of repetitions, and accuracy of the movements. The system also automatically increased or decreased the level of difficulty if a specific success rate was hit. The system also provided the

player with positive or negative audiovisual reinforcement depending on whether the task was achieved or not.

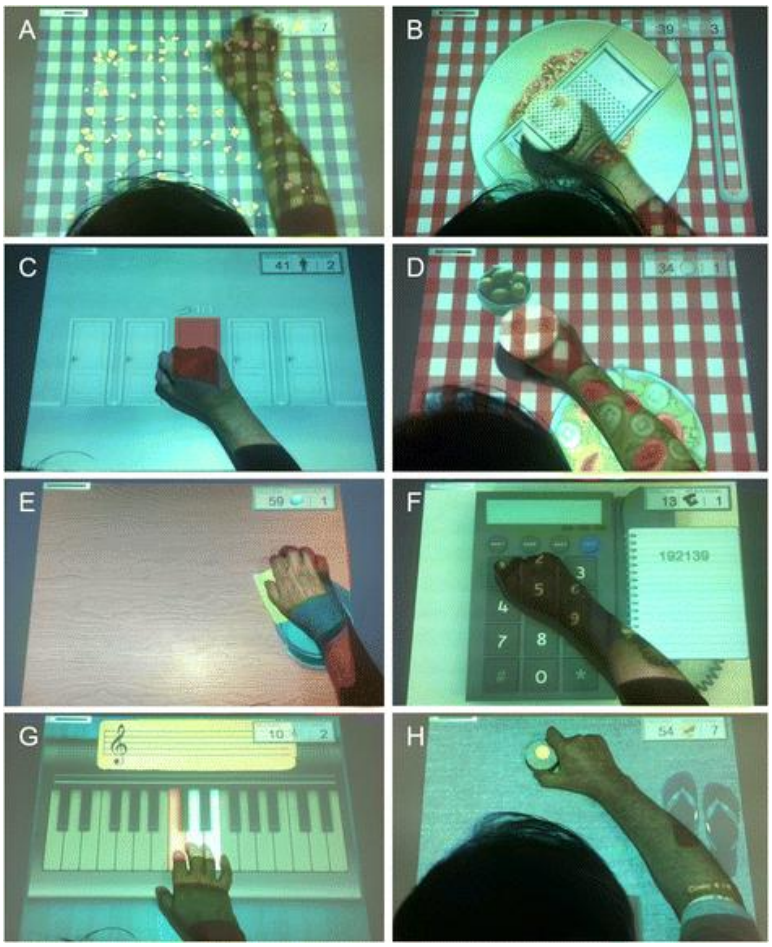

Figure 5. Serious games for upper limb mobility rehabilitation: (a) sweeping the crumbs from the table, (b) carrot grating, (c) knocking on doors, (d) cooking, (e) squeezing a sponge, (f) dialing a number, (h) playing piano, (h) buying items [11].

Moving on to some more recent studies and new technologies employed, research done in 2021 explored the usage of a smartphone as a portable medium for a serious game-based augmented reality system for rehabilitation of stroke survivors [12]. This cellphone augmented reality rehabilitation system (CARS) was based on the ARKit toolbox run on an iPhone XR with an iOS 13 operating system. During the research the fact that stroke survivors may not be able to pick up and hold their phone for long enough period of time was taken into consideration and addressed by means of designing a special phone case with a built-in fingerless glove (Fig. 6). Patients could move their affected hand to use the cellphone and interact with 3D virtual targets generated on the

cellphone screen (Fig. 7).

Figure 6. Hardware for the cellphone augmented reality rehabilitation system: iPhone XR, iOS 13 (left), phone case with built-in fingerless glove (center); demonstration of the hardware usage (right) [12].

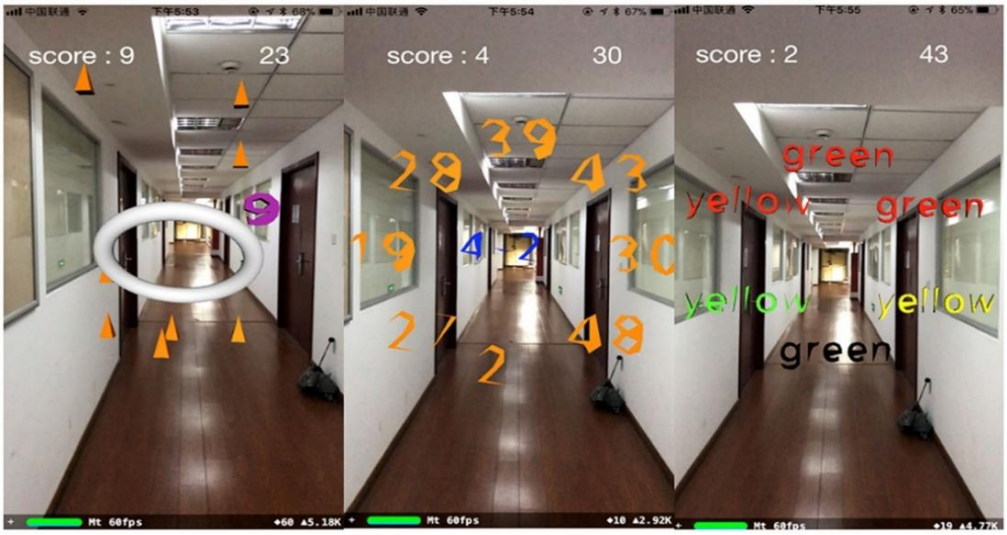

Figure 7. Three augmented reality–based serious games for rehabilitation of upper limb motor function and cognitive function: Pyramid Reach (left), Add VS Sub (center), Stroop Game (right) [12].

CARS included three serious games. The first game was focused on enhancing upper limb functionality and concentration and required the patients to move their phone to touch generated targets (Fig. 7, left). The second game was focused on a patient's cognitive ability as much as on their physical recovery – the patient needed to calculate the result of a formula generated in the center of the screen and touch the correct answer in the provided set of numbers (Fig. 7, center). The third game was also focused on the recovery of both cognitive and motor functions – it showed the player a list of color names and required them to choose the one which coloring and text correspond to each other (Fig. 7, right).

Another research in gamification of upper limb rehabilitation in mixed-reality environment was published in 2022 by A. Pillai et al. [13]. It used a pair of HoloLens 2 - fully immersive MR smart glasses released by Microsoft in 2019, as a way to display the digitally generated environment for the user. Hololens 2 has a range of built-in technologies that make the game development process much easier, including light cameras for hand and eye tracking, IMU, depth sensors, etc. The serious game used for rehabilitation was designed using Unity game engine in combination with the Mixed Reality ToolKit provided by Microsoft for HoloLens 2 (Fig. 8).

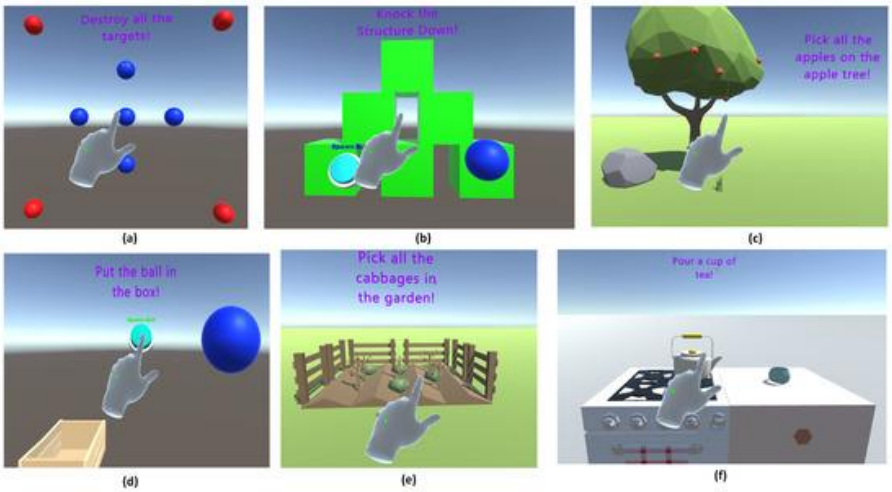

Figure 8. VR serious games for upper-limb mobility rehabilitation: (a) item selection, (b) knocking down structures with spheres, (c) picking apples from a tree, (d) putting a ball in a box, (e) picking cabbages in the garden, (f) pouring a cup of tea [13].

Since the HoloLens 2's camera was incapable to recognize all of the movement that the games targeted, a new algorithm had to be developed and implemented. This algorithm could estimate the positions of the shoulder and elbow joints as well as the ROM for all three degrees of freedom of the shoulder, and the flexion and extension of the elbow. The algorithm relied on such personal information as upper arm and forearm lengths, shoulder width and distance between the eye and the clavicle. That information could be entered by user at the start of each level. The system included many games and focused on a wide range of upper limb motions. It also gave the player an opportunity to choose the difficulty level to provide him or her with the correct level of necessary movements based on their stage of rehabilitation.

HoloLens 2 was used in a number of other recent projects. One of them was a rather unusual and interesting research of creating a shared augmented reality

between a patient and a clinical professional published in 2023 [14]. The aim of doing so was to ensure that the therapist is capable of conducting accurate assessment of the patient's progress in real time and intervene in the rehabilitation process if necessary. This issue arises with the usage of headsets and smart glasses, as the image is displayed personally to the person wearing them, and not on a screen that can be visible for several people at the same time. In the research both the therapist and the patient are wearing Microsoft HoloLens 2 and are engaging in a serious game of setting up the table for eating (Fig. 9). The system built for this research consists of multiple elements and is rather complex (Fig. 10).

Figure 9. Shared AR environment from the field of view of the (a) therapist's and (b) patient's headsets [14].

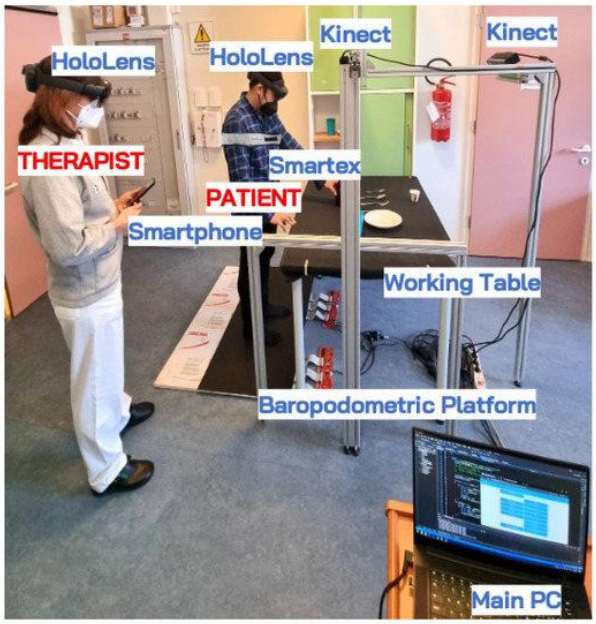

Figure 10. Framework setup for a shared between patient and therapist AR-based system for upper limb mobility rehabilitation [14].

It can be concluded that despite the increase in both quantity and quality of AR game-based solutions for rehabilitation of motor function in upper limbs, this area still remains underdeveloped due to its relative novelty. The rapid transition to head-mounted displays in recent years, while being an understandable process caused by technological advancements and aimed to make the gaming process more immersive, may cause a problem of excluding some people from receiving help in rehabilitation due to dizziness or sickness while using such technologies, or even due to an issue as easy as not having a perfect eyesight. This makes the development of a more inclusive rehabilitation system that would still incorporate all of the advantages of modern technologies that much more important.

3. TECHNOLOGIES USED

3.1. Unity

3.1.1. Introduction and overview

Unity is a powerful, cross-platform game engine developed by Unity Technologies. Initially launched in 2005 as a Mac-only game engine [15], Unity has since grown into one of the most popular tools for creating both 2D and 3D interactive content, including video games, simulations, architectural visualizations, virtual reality (VR), and augmented reality (AR) experiences. It supports over 25 deployment platforms, including Windows, macOS, Android, iOS, WebGL, Linux, PlayStation, Xbox, and more.

Unity is designed to support the entire development process, from asset importing and scene design to scripting, real-time testing, optimization, and final deployment. Its popularity stems from its balance of ease of use, flexibility, and powerful features, making it suitable for both hobbyists and large studios.

Unity uses C# as its primary scripting language via Mono/.NET runtime. While the engine itself does not provide a code editor to comfortably manage the code files within Unity itself, it does support integration with other IDEs, including Visual Studio, which was used during the development of the current project.

Aside from coding, Unity provides its users with visual tools to manage scenes, lighting and rendering, assets, animation, UI layout, etc. Unity also supports multiple rendering pipelines and has a built-in 2D and 3D physical engines that support ray casting, which was used in the development of this game, as well as rigid body dynamics, collision detection, joints, etc.

Some of the other important Unity features that made this engine a popular choice for many but were not used in the development of this project include:

- Unity Asset Store – a marketplace where developers can buy or sell 3D models, scripts and plugins, AI systems and even full project templates;
- Monetization – Unity provides its developers with built in Ads and In-App purchasing functionality;
- XR (VR/AR) support – Unity supports platforms such as Oculus, HTC Vive, Magic Leap, HoloLens, ARKit, ARCore. Since the hardware used in this project is custom, the built in Unity XR features weren't used in the development process.

Main advantages of Unity that were not yet mentioned include its free plan for individuals and small companies, huge community and extensive

documentation, friendly to beginners learning curve, live editing and debugging, as well as Unity Cloud Services ready to use.

Unity does have its drawbacks, including proprietary ecosystem that may tie developers to Unity, uncontrollable memory management due to self-managed garbage collector, huge resulting build sizes, etc.

3.1.2. Main principles

3.1.2.1. Scenes

A Scene in Unity is a container or workspace that holds all the game objects and environment data needed to represent a part of a game or application — such as a level, menu, cutscene, or even a UI overlay.

Scenes are saved as Unity files in the project and are created and edited using the Unity Editor.

Scenes in Unity serve several purposes. Mainly, scenes represent a level or a stage in the game. Scenes also usually represent different types of content or functionality: while gameplay itself can be separated into different scenes-levels, menus and cutscenes are usually also separated in different scenes. Additionally, scenes can be used to optimize performance and memory usage by separating the game components into relatively sustainable parts.

Scenes are usually loaded and managed in code using the SceneManager class. To load scenes at runtime, they must be added to File > Build Settings > Scenes in Build (Fig. 11). After this, each scene is assigned a build index that can be used to identify and load that scene. Scenes not listed there can only be loaded using asset bundles or addressables.

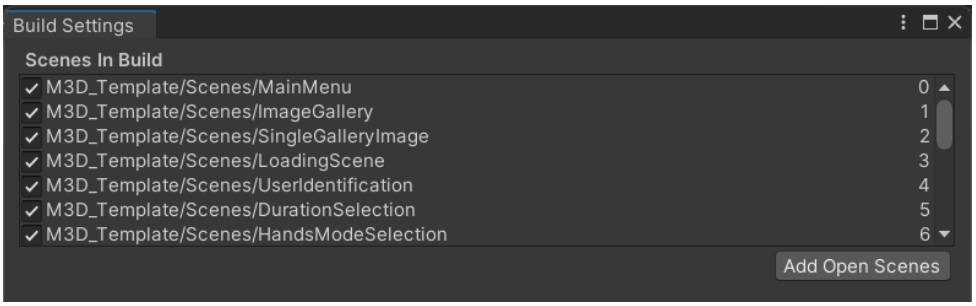

Figure 11. The section of the Build Settings window with the list of added scenes.

3.1.2.2. Game Objects

A Game Object is the basic entity in a Unity scene. It represents anything that exists in the game world — a player character, enemy, wall, light, UI element, camera, etc.

By itself, a Game Object is empty and has no behavior or appearance. Its functionality comes from the Components attached to it. The one component that is always attached to a Game Object is Transform Component. It holds the position, rotation, and scale of the object in the scene and defines the object's location in the world or relative to a parent.

One of the main concepts of Unity is the Game Object hierarchy (Fig. 12), which defines how Game Objects are organized and related to each other within a Scene. The Game Object hierarchy is a tree-like structure in which Game Objects can be nested under other Game Objects. This relationship is known as a parent-child hierarchy. Every Scene has a root level (top-level) of Game Objects – those game objects are the direct children on the Scene. Each Game Object can have zero, one or more children, and each child Game Object can itself be a parent to other Game Objects, creating a nested structure. In the Unity Editor, this is visualized in the Hierarchy window.

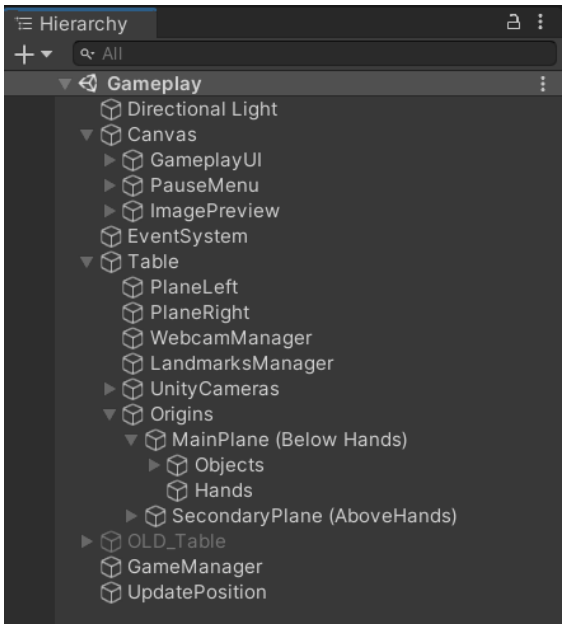

Figure 12. Game Object hierarchy.

Unity presents developers with a wide selection of precomposed game objects that include several Components and sometimes even child game objects. The components used the most often during the development of the project

included cameras and UI components, such as Canvas – the parent object for all UI components, Text, Button and others.

3.1.2.3. Components

A Component is a modular piece of behavior or functionality that can be attached to a Game Object. Each component adds specific capabilities, like rendering, physics, audio, input, or custom logic.

The main purpose of components is to make Unity modular and flexible. Instead of creating many specialized classes, Unity lets you build functionality by composing small behaviors together on Game Objects.

For example, to make a moving character:

- You add a Rigid Body for physics movement.
- A Collider component to detect collisions.
- A custom Script component to define movement behavior.

As shown in the examples above, Unity supports some built in components, as well as an opportunity to create custom components with the behavior determined by the developer. Built-in components are the components provided by Unity. They are used to handle rendering, physics, audio, lighting, navigation, and more. Those components usually inherit either Component class, or Behavior class, which is a class derived from Component class.

User-defined Components are C# scripts that inherit MonoBehaviour class and can be attached to Game Objects to define custom behavior. MonoBehaviour class inherits the Behavior class mentioned earlier and allows the developer to use and redefine a long list of methods and properties, including the Unity lifecycles methods. Those methods are, perhaps, ones of the most important ones and are invoked when the enabled script instance is being loaded (MonoBehaviour.Awake()), on the first frame when a script is enabled (MonoBehaviour.Start()) or on every frame (MonoBehaviour.Update()).

3.2. M3Display

3.2.1. Introduction and history

As mentioned earlier in the subsection 1.1 Introduction, M3Display is an AR system for upper limb rehabilitation developed by the Institute of Advanced Production Technologies Medical Robotics Research Group.

Institute of Advanced Production Technologies (ITAP) is a research institute of the University of Valladolid, Spain [16]. It was formed by more than 28 research professors from different areas of knowledge of the Faculty of Engineering and the Faculty of Medicine of the University of Valladolid and includes several research groups studying thermal engineering, robotics and vision, advanced industrial technologies, analysis and diagnosis of electrical networks, etc. ITAP Medical Robotics Research Group is one of the subdivisions of ITAP and focuses its research on medical robotics, more specifically - rehabilitation robotics and collaborative and autonomous surgery [4]. This research group is composed of professors, PhD and Master students of the Department of System Engineering and Automation of the Faculty of Engineering of the University of Valladolid.

The M3Display is still in the development stage – its goals for the future are to expand the list of serious games, have it undertaken a series of clinical trials and conduct some usability and risk analysis [17].

3.2.2. Hardware

The M3Display consists of several physical components [5]:

- Logitech HD Pro C920 camera – to capture the hand movement;
- Computer with Intel Core i5 11400 CPU processor – to process the image captured by the camera and run the game;
- 27" Philips 273V7QDSB display – to display the game to the user;
- 380 × 550 mm mirror – to make the visual alignment between the patient's perspective hand and the captured image more natural and seamless;

The overall structure of the M3Display is shown on figure 13. The screen is placed in a position between the patient's view and his hands perpendicular to the patient's vision.

To guarantee a distortion-free representation of the hand on the screen, a precise position of both the camera and the mirror are required. The camera is placed at a 45-degree angle in regard to the mirror and is directed towards its center. That way an overlap between the patient's line of sight and the image captured by the camera is achieved, leading to an illusion that the patient's arm extends directly into the screen.

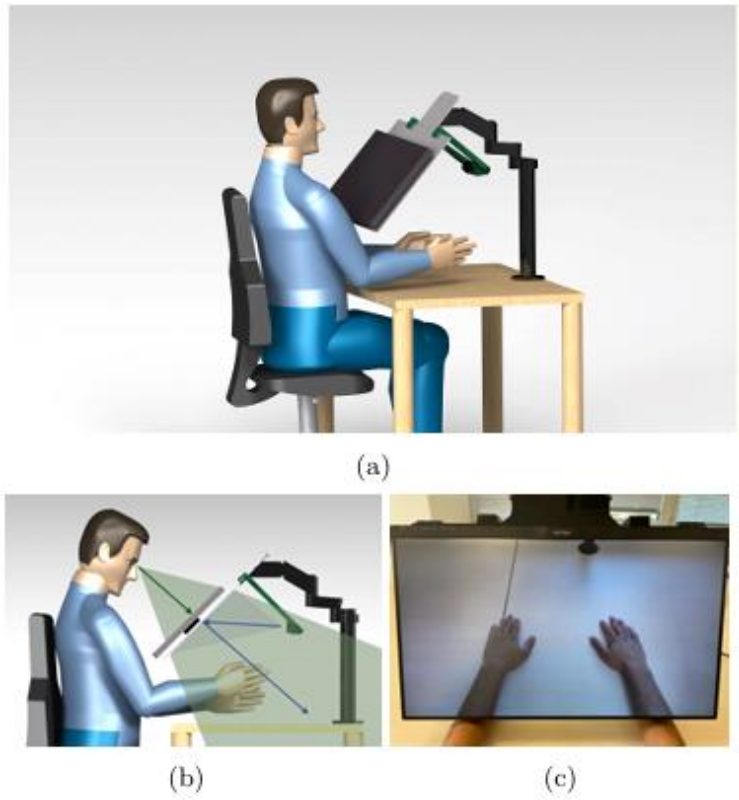

Figure 13. M3Display hardware setup: (a) full system; (b) mirror configuration; (c) final video feed with hand continuity on the screen [5].

3.2.3. Software

3.2.3.1. Serious games

The serious games used during rehabilitation are developed by ITAP Medical Robotics Research Group with Unity game engine and facilitate upper limbs' recovery by allowing the player to interact with the computer-generated virtual components of the game using his or her hands. They can be launched through a unified user interface shown on figure 14.

These serious games require a different range of motion from the player, including moving the entirety of the upper limb, rotating the wrist with the hand folded into a fist, etc. (Fig. 15).

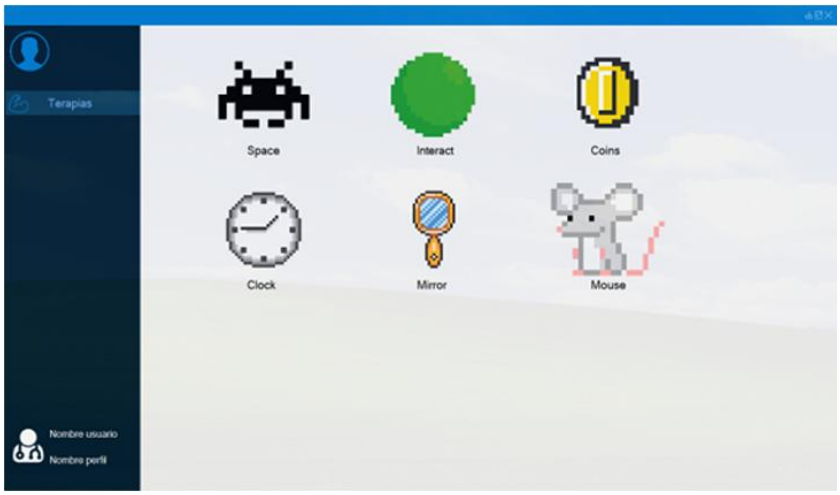

Figure 14. M3Display user interface [5].

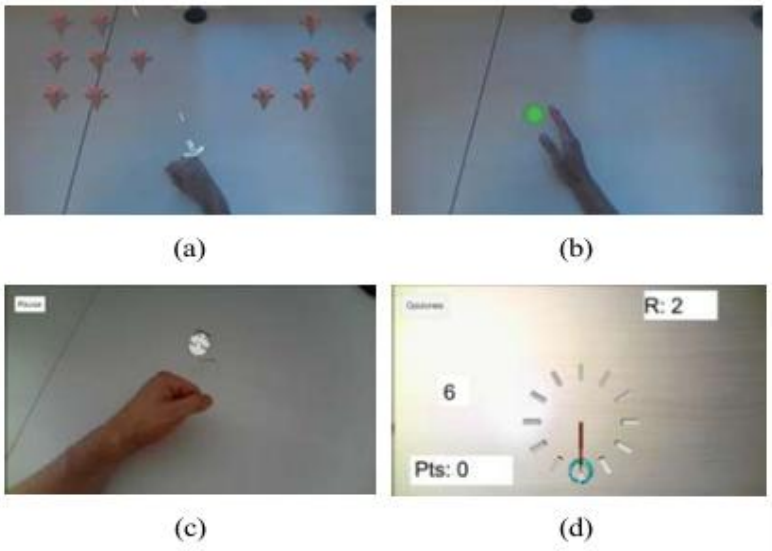

Figure 15. Some of the serious games developed for the M3Display: (a) Space, (b) Ball interaction, (c) Mouse, (d) Clock [5].

3.2.3.2. Hand detection algorithm

As previously mentioned, the games developed for M3Display use the real-time hand segmentation solution developed by the ITAP Medical Robotics Research Group [6]. As its core, it uses MediaPipe, a highly optimized neural network developed by Google for hands landmark detection [18], [19].

The algorithm extracts the characteristic hand landmarks and the mask of the user's hands with the following algorithm:

1. The video feed image is processed by MediaPipe to get the hand

landmarks and draw a skeleton of the hand;

2. Based on the skeleton, a mask of the area where the hand boundaries may be present is formed. To do so, at first the mask is created from the skeleton itself (M1). Then this mask is dilated and increased in size using 21x21 pixels kernel to create a second mask (M2). This size for the kernel was obtained through trials as the best-performing one;
3. The original image is converted to the CIELab color space and is combined with the skeleton mask M1 to obtain the pallet of probable skin color – this operation only leaves us with those pixels from the converted image that were present on the skeleton;
4. The colors obtained in the previous step are used to randomly sample 1500 values that correspond to the hands in both the α and β channels. These values are then sorted and used to get the first and third quartiles that pose as the boundaries to a region in the CIELab color space where the majority of the pixels corresponding to hands in the same frame are expected to be located;
5. The boundaries obtained in the previous step are used to create two additional masks, one in α and one in β channel respectively (Ma and Mb), to eliminate the values outside of the boundaries. Those masks are then combined to only obtain the values that fall into the determined boundaries in both channels (Mab);
6. The mask gained in the previous step (Mab) is combined with the M1 mask obtained in the second step; The union is then subject to morphological operations of opening and closing to smooth out the edges, fill possible holes and reduce the noise.
7. The resulting mask is combined with the M2 mask obtained in the second step to eliminate the values that correspond to the range of skin colors but are present outside of the predetermined hand possible position boundaries.

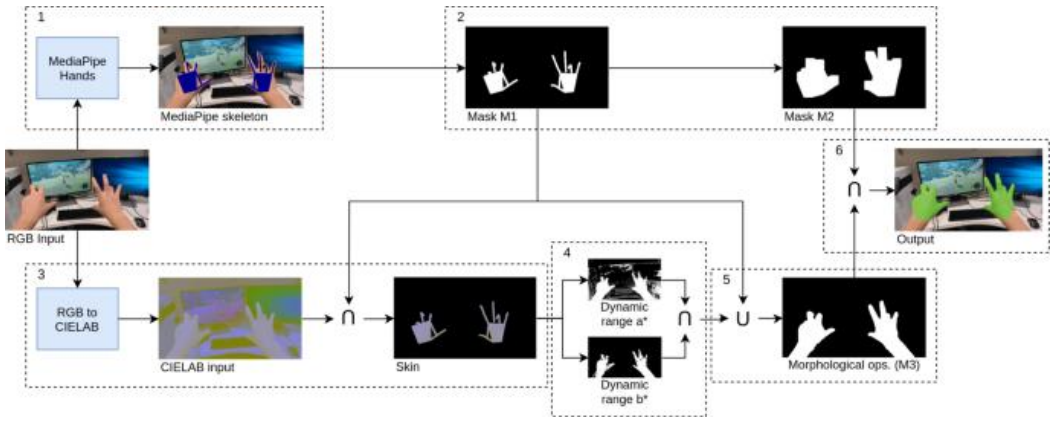

Figure 16. Stages of hand segmentation [6].

As could have been noticed already, the hand positions that are used in the game process were extracted on step 1. The remaining 6 steps were needed to get the entirety of the hand itself to later transpose it on top of the generated game objects to create an illusion of player's direct contact with the game elements, thus enhancing the user's immersion and playing experience.

Figure 17. Random samples from the evaluation sequences of Ego2Hands [6]

4. GAME STRUCTURE

4.1. General concept

The serious game presented in this paper aims to facilitate the fine motor skills recovery in a patient's upper limbs. Its gameplay requires the patient to move their hand across the scrin, grasp the scattered puzzle pieces and move them onto the assembly space, where they should be put into their proper places.

The game allows the player to choose which hand will be used, adjust the gameplay area based on the distances the patient can comfortably reach, select the game difficulty based on the gameplay area and the advancements the patient has made in their recovery. The player can also calibrate at what point the hang will be recognized as a fist thus allowing the user to pick up the pieces underneath it based on how well the patient's fine motor skills currently are.

The medical personnel can choose the duration of a therapy session and load additional images into the system to keep the game engaging.

Later, the information about the therapy session can be accessed through another application developed by the ITAP Medical Robotics Research Group. This information is outlined in detail in section 4.6 Resulting Data.

4.2. Game windows

4.2.1. Main Menu

Main menu is the starting point of the game and the first screen the player sees upon the game launch.

It has 3 buttons, "Play", "Gallery" and "Exit", that allow the player to start the game, open the gallery with uploaded images, or quit the game.

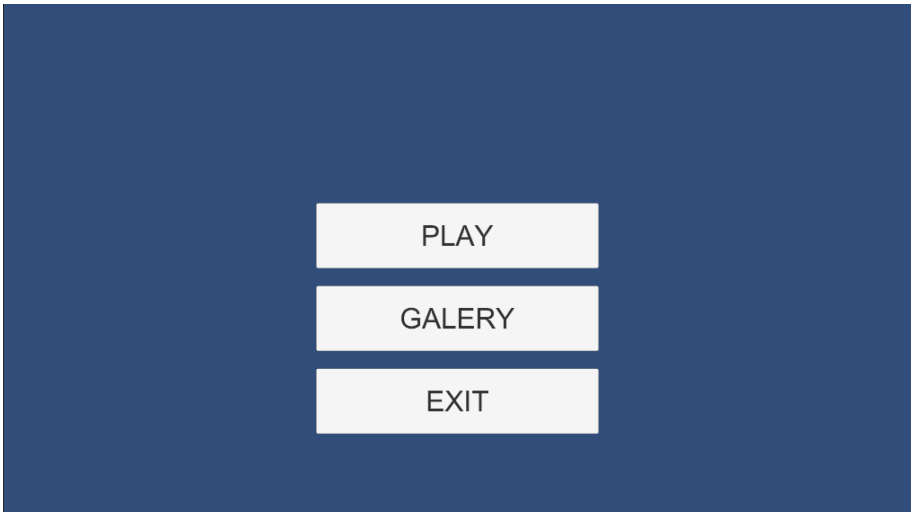

Figure 18. Main Menu scene.

4.2.2. Image Gallery

In the Image Gallery the player can see all the images previously added to the game. They can also click on one of the images to open the single image view to see the image in more detail or to delete it.

The scene's user interface has 2 buttons: "Return" and "Add Image". By pressing the "Return" button the player can navigate back to the main menu. Upon pressing the "Add Image" button the player can select an image from the memory of their machine. This image will be added to the game and will be displayed with the others in the Image Gallery.

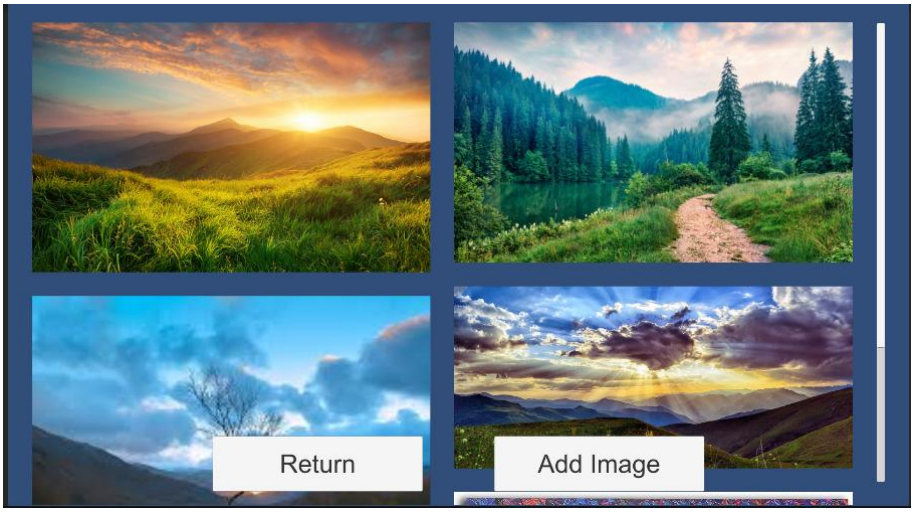

Figure 19. Image Gallery scene.

4.2.3. Single Gallery Image

Single Gallery Image scene is opened when one of the images in the Image Gallery is pressed. As previously mentioned, here, the player can see the image in more details and delete it, if necessary, by pressing the “Delete Image” button. The player can also go back to the Image Gallery by pressing the “Return” button.

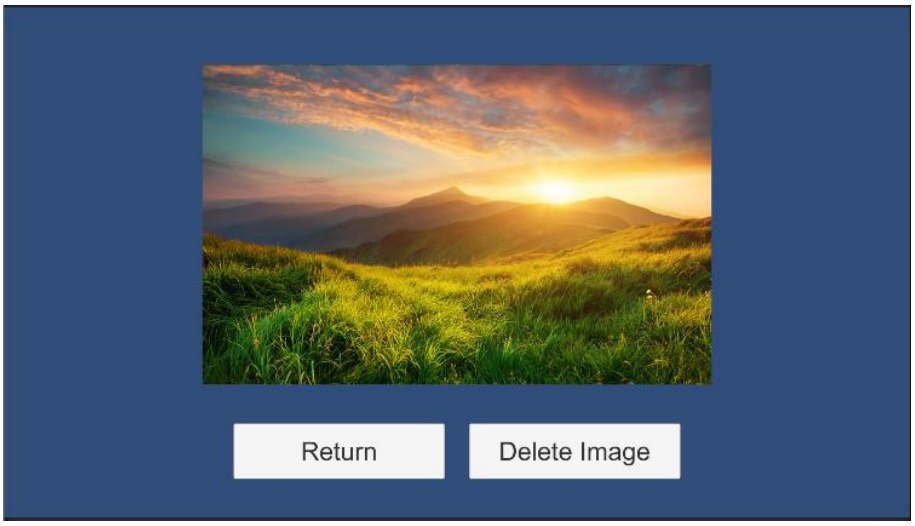

Figure 20. Single Gallery Image scene.

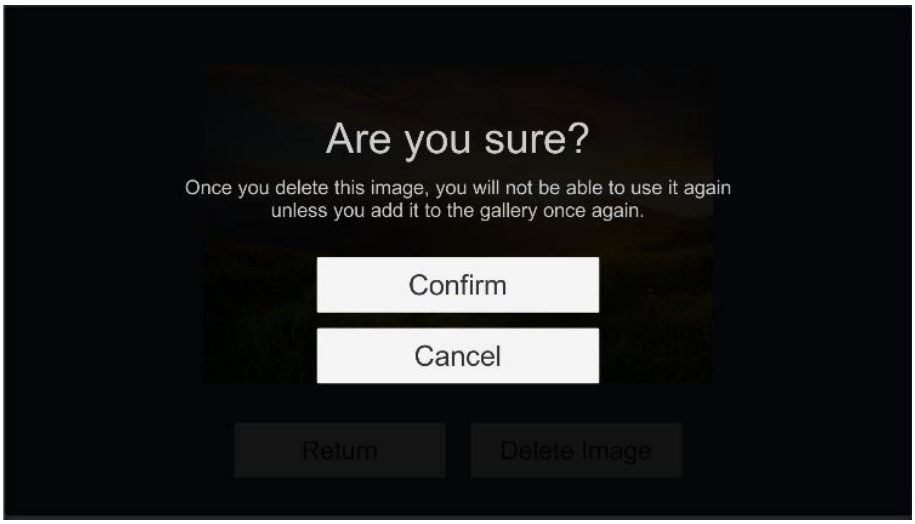

Figure 21. Single Gallery Image scene: delete image menu.

4.2.4. User Identification

The User Identification scene is displayed after the user presses “Play” button in the Main Menu. It has two dropdown selectors – one for the therapist and one for the patient. The unique identifiers of both therapists and patients are loaded from the database to which they were added previously from a different program.

When both choices are made, the player can go to the next scene (Duration Selection) by pressing the “Continue” button.

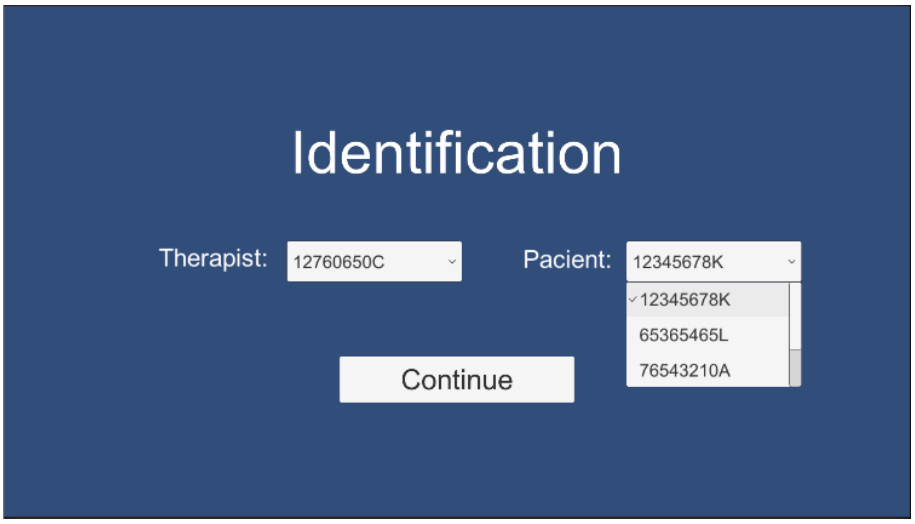

Figure 22. User Identification scene.

4.2.5. Duration Selection

On the Duration Selection scene, the player can choose the duration of the therapy session. They can do so by dragging the slider left and right to select the desirable value. Maximum and minimum value of the session can be adjusted in the Unity editor and for now are set to 600 and 0 seconds accordingly.

After choosing the necessary value the player can continue with the game by pressing the “Confirm” button.

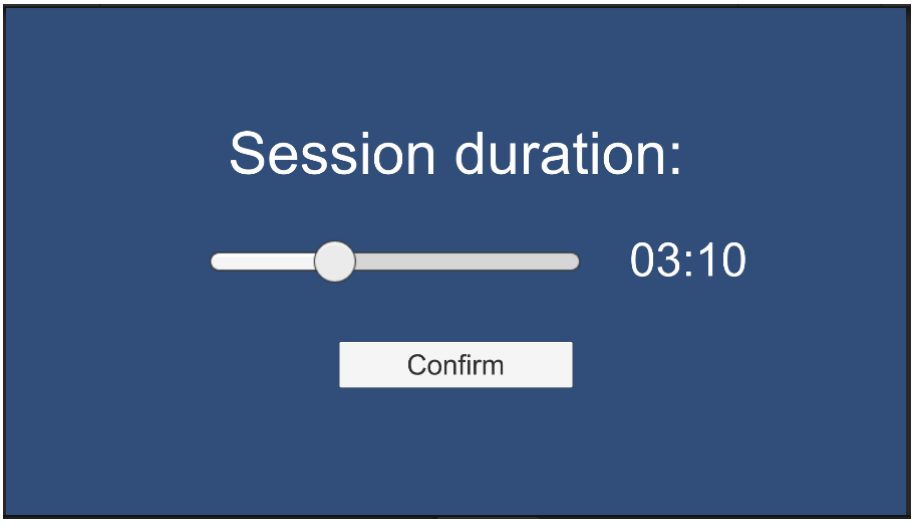

Figure 23. Duration Selection scene.

4.2.6. Hands Mode Selection

The Hands Mode Selection scene comes after the Duration Selection scene and presents the player with the choice which hand to use during the game. After the choice is made and the “Confirm” button is pressed the player is transferred to the Landmarks Initial Calibration scene.

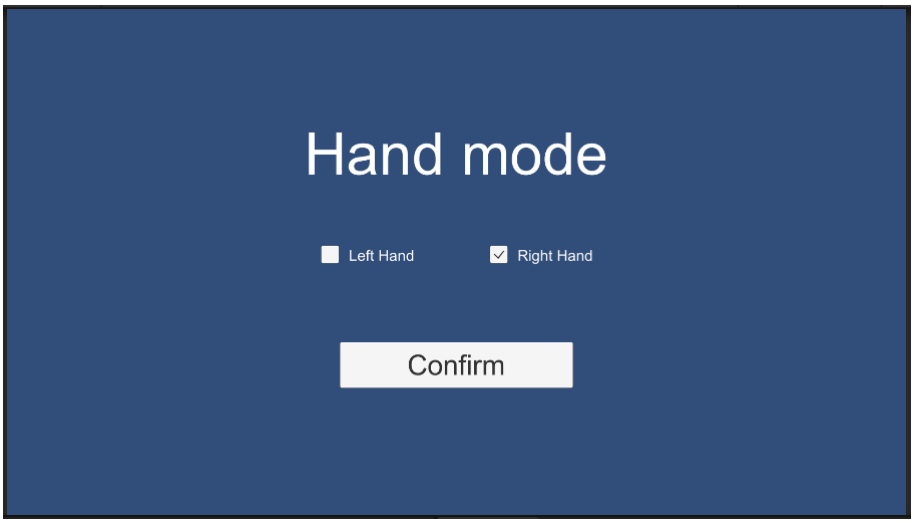

Figure 24. Hand Mode Selection scene.

4.2.7. Landmarks Initial Calibration

On the Landmarks Initial Calibration scene, the player is asked to place the hand selected on the previous scene in front of the camera. Once the hand is detected the player is asked to form a fist with that hand. Once the patient curled his hingers as far as they can, they need to press the “Confirm” button. After the information is successfully recorded, the “Continue” button, previously disabled, becomes enabled. After pressing it the player can go to the next scene.

Upon pressing the “Cancel” button the player can go back to the Hands Mode Selection scene to change the selected hand, if necessary.

When the initial calibration is underway, a special game object called Webcam Manager captures the video feed from the camera on every frame. The captured image is processed by the python script which was already addressed in section 3.2.3.2. This script returns data with the position of each of the hand landmarks pointed out on Fig. 25. This data is passed into another game object called Landmarks Initial Calibration Manager, which calculate the distance from the start on the hand to the start of the finger and from the start of the hand to the end of the finger for each finger (for example – from point 0 to point 5 and from point 0 to point 8 for index finger). Then the distance to the end of the finger is divided by the distance to the start of the finger. This distance is later multiplied by 1.1 to account for possible inaccuracies in the algorithm and is used as the minimum threshold for considering the hand grasped. This value for each finger is saved in the Landmarks Settings to be accessed during the gameplay.

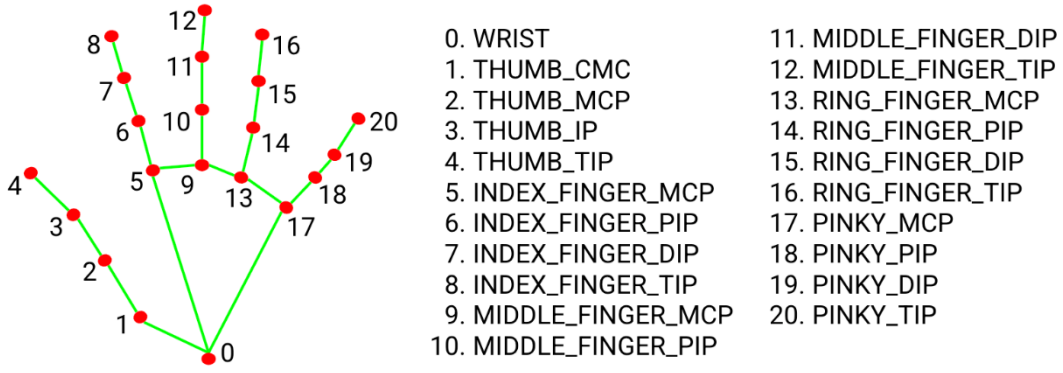

Figure 25. MediaPipe hand landmarks [18].

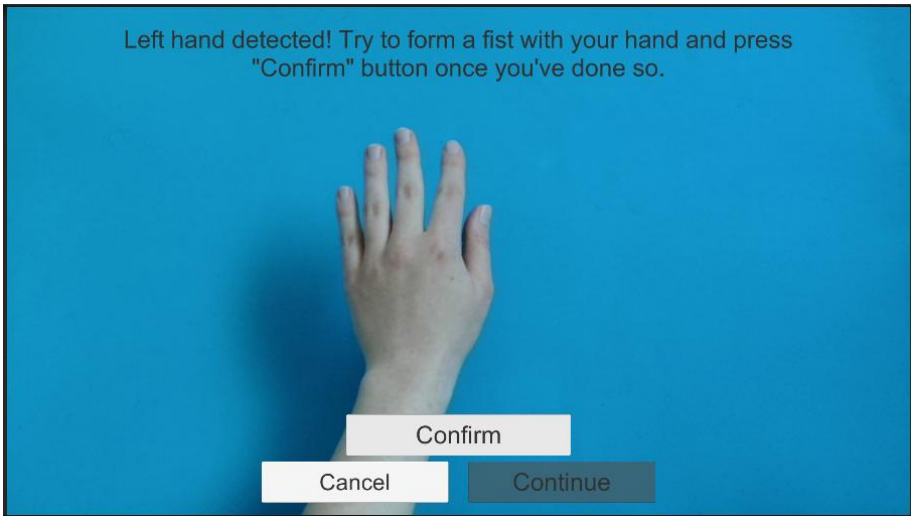

Figure 26. Landmarks Initial Calibration scene before the fist capture.

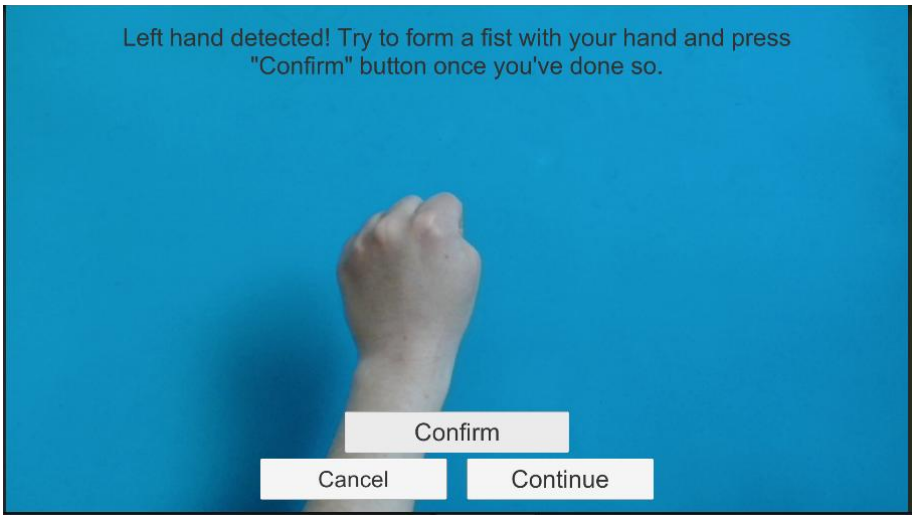

Figure 27. Landmarks Initial Calibration scene after the fist capture.

4.2.8. Table Size Selection

The Table Size Selection scene comes after the Landmarks Initial Calibration scene. Here by adjusting the value of the slider the user can control the size of the purple rectangle on the screen. This rectangle represents the area of the screen that the player can comfortably reach. The slider allows the player to choose the value between 0.7 and 1 representing the size in the range from 70 to 100 percents of the screen.

The aim of this adjustment is to make the game as accessible as possible even for the people whose upper-limb mobility is heavily limited.

To confirm the chosen area and move on to the next screen the user has to press the “Continue” button.

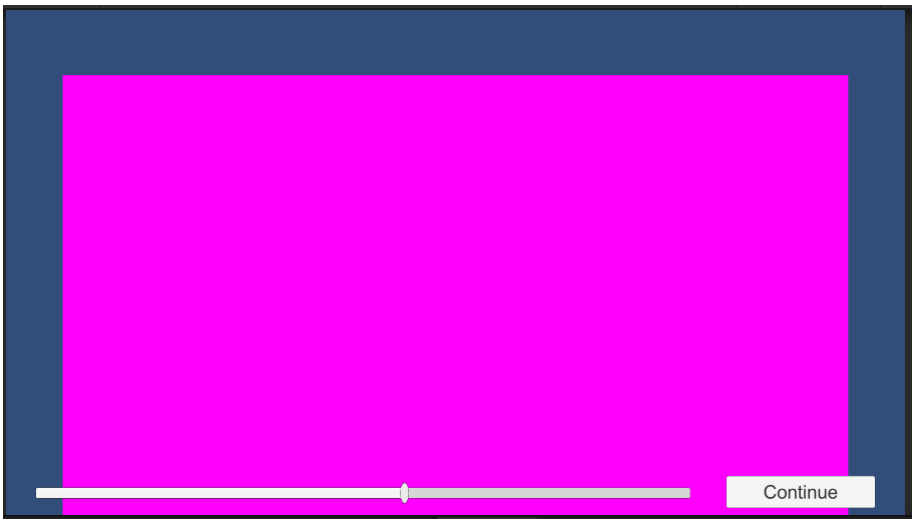

Figure 28. Table Size Selection scene.

4.2.9. Difficulty Initial Selection

The Difficulty Initial Selection scene comes after the Table Size Selection scene and allows the user to choose, in how many elements will the final puzzle be split. On the easy difficulty, the puzzle will be split into 2 pieces; on normal – into 4 pieces, and on hard – into 6.

This choice is added to compensate for the possible variations in playing area sizes as well as different degrees of possible upper-limb mobility for different patients.

When the player moves their mouse on top of one of the buttons, a text explaining the difficulty specifications appears below the “Choose Difficulty Level” text. Once the player clicks on one of the buttons they are transferred to the next scene.

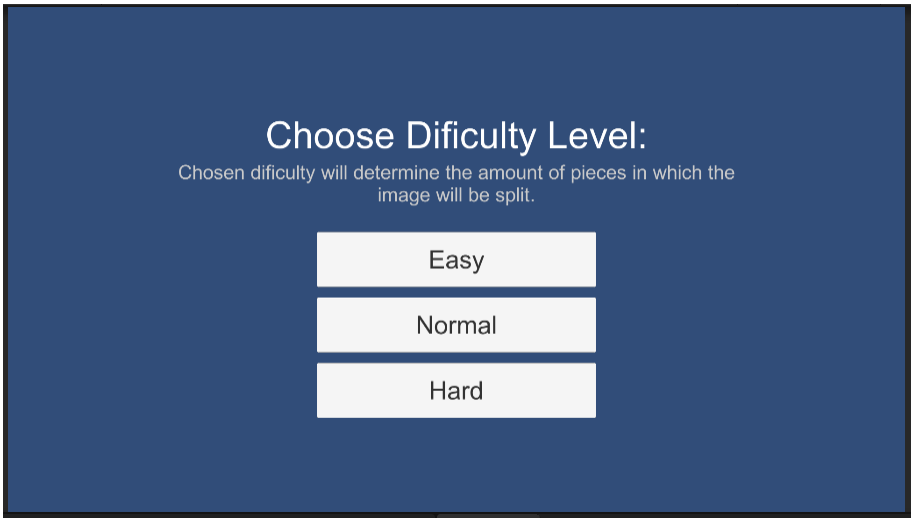

Figure 29. Difficulty Initial Selection scene.

4.2.10. Image Selection

Image Selection scene comes after the Difficulty Initial Selection scene and is also launched every time the player finishes the puzzle before the therapy period is finished. On this scene the player can scroll the images with “Next image” and “Previous image” buttons. Once the player is satisfied with their choice, they can click the “Confirm” button and start the game itself.

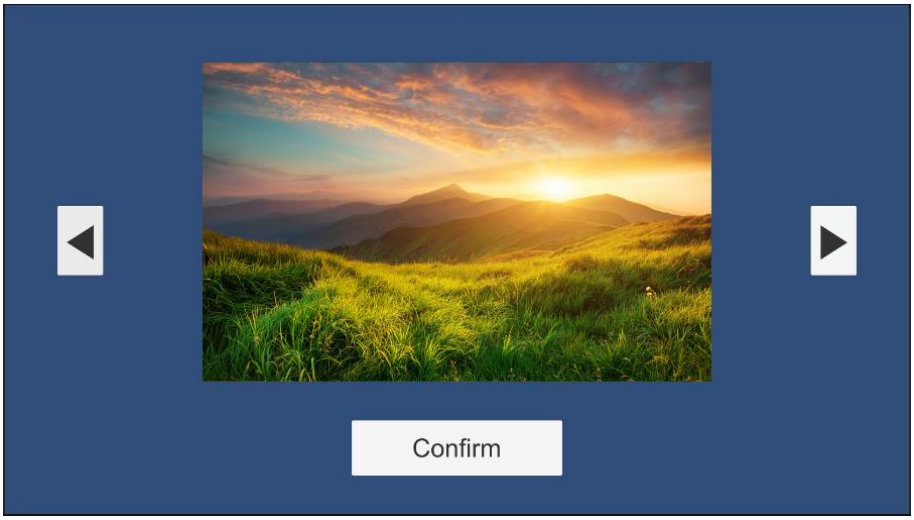

Figure 30. Image Selection scene.

4.2.11. Gameplay

The gameplay scene is the most important scene as it is the scene with the actual game mechanics.

When the scene is first loaded, the previously selected image is cut into the number of pieces determined by the chosen difficulty. Their order is randomized, and they are situated at the bottom of the screen. A light area located on the top of the screen is the assembly table where the pieces are supposed to be moved and assembled into an image. It has darker areas used as landmarks of where the pieces should be dropped to help the player.

The player can pick the pieces up when their hand is formed into a fist above the puzzle piece in question and release them by uncurling their fingers. The calculations performed during the initial calibration are performed again on each frame and the value received is compared to the value saved during the calibration, If the value is lower than the saved one the finger is considered curled. If this condition is fulfilled for each finger, the hand is considered to be in a fist. Otherwise, it is considered released.

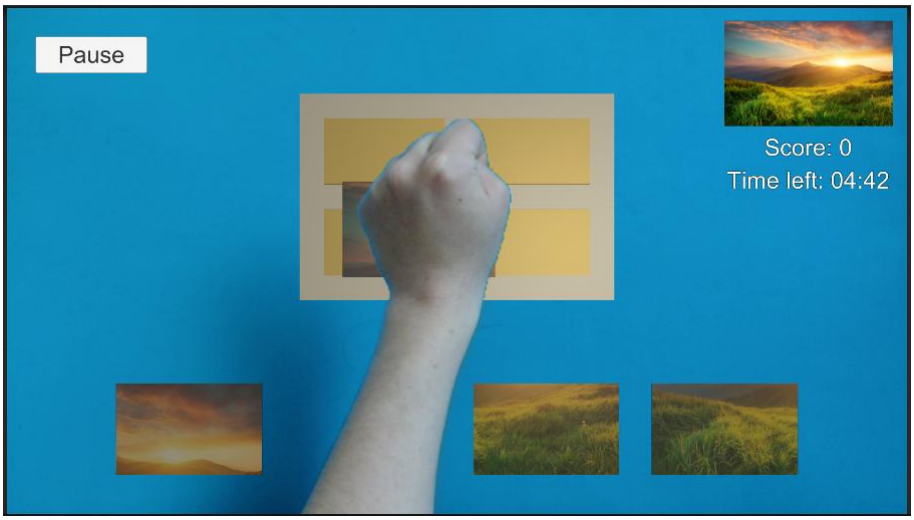

Figure 31. Gameplay scene.

The data about hand landmarks position on the Gameplay scene is accessed in the exact same manner as on the Landmarks Initial Calibration scene.

To display the player's hand on top of the objects the scene uses two planes and two cameras. The first set films the background video feed from the web camera and the gameplay itself. The second set processes the feed from the web camera using a python hand segregation script explained in section 3.2.3.2

and later displays and films only the player's hands. The feed from both cameras is combined in such a manner that the player's hands are displayed on top of the game elements. This creates the effect of immersion that the game aims to provide for the patient.

The user interface of the Gameplay scene includes a "Pause" button, a minimalistic image, as well as a "Score" and a "Time" texts.

The "Score" text updates every time the player scores or loses a point – when they drop the piece they are holding at the appropriate position or when they pick up a piece from its correct position accordingly. The score displayed is the score of the current playthrough, not of the entire therapy session.

The "Time" text updates every frame and displays how much time the player has left in the therapy session.

The minimalistic image can be pressed to display the image across the screen in case the player has difficulties figuring out the puzzle (Fig. 32). This preview can be closed by pressing the "Close" button. When the preview is open, the game is paused, meaning that the video feed is not updated and the time countdown is stopped.

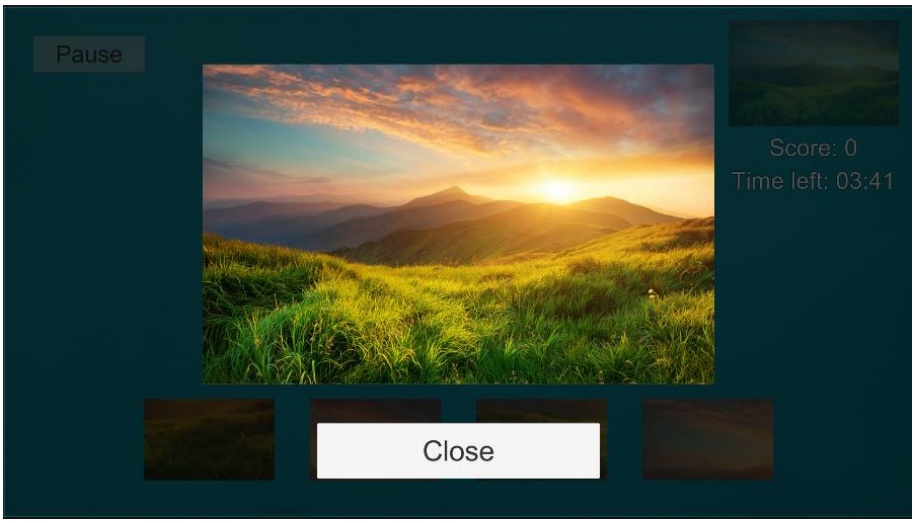

Figure 32. Gameplay scene: image detailed view menu.

When the "Pause" button is pressed, a pause menu is displayed (Fig. 33) and the game is paused. From the pause menu the player can resume the game, go to settings or exit the game.

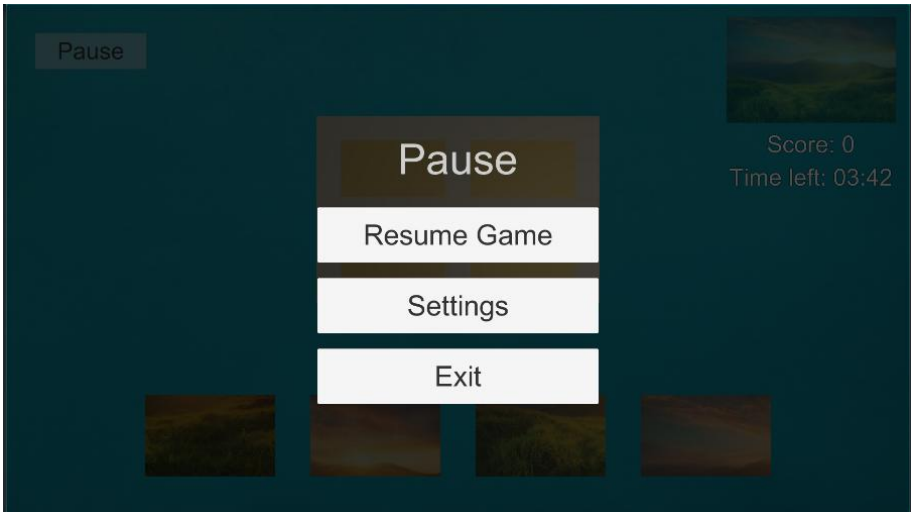

Figure 33. Gameplay scene: pause menu.

The end of the current playthrough is determined by the current score – once it is equal to the number of pieces, the game is considered finished. If the therapy session is not over yet and after a small delay, the player is transferred back to Image Selection scene to pick an image for the next gameplay.

If the game was previously paused and the Gameplay scene was left to go into Settings scene, upon the return to the Gameplay the scene will be regenerated. In this case, the puzzle pieces would not be generated in their default places. Instead, they will be generated in the positions in which they were left before the Gameplay scene was left. It is done with the use of a special object called Persistent Data (more information about it will be in section 4.3.1) – it stores all the information about the therapy session in general and the current playthrough, including the information about each puzzle piece (position, unique identifier and the section of the image, displayed on the piece).

4.2.12. Settings Scene

The Settings Scene can be accessed from the pause menu of the Gameplay scene. From here, the user can either go to the Landmarks Settings Calibration scene or go back to the gameplay by pressing “Calibrate” and “Return” buttons accordingly.

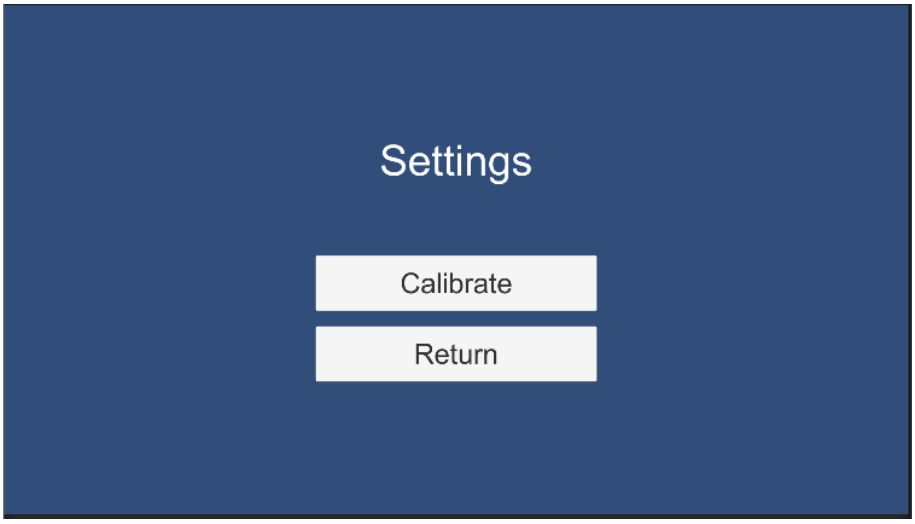

Figure 34. Settings scene.

4.2.13. Landmarks Settings Calibration

On the Landmarks Setting Calibration scene the player can readjust the setting previously recorded at the Landmarks Initial Calibration scene. It works in the exact same way as the Landmarks Initial Calibration; only the user interface elements are slightly different. The scene has only two buttons – “Confirm”, that mirrors the functionality of the button with the same name on the Landmarks Initial Calibration scene, and “Return”, which brings the user back to the Settings scene.

Figure 35. Landmarks Settings Calibration scene.

4.2.14. Game End

The Game End scene is displayed automatically once the time for the therapy session runs out. It only has one button – “Leave”. Once this button is pressed, the data gathered during the therapy session is recorded and the game is closed.

Figure 36. Game End scene.

4.2.15. Loading

Loading Scene is displayed automatically between scenes. Its purpose is to make the user experience slightly better and signal that the game is working on generating the next scene, which can take quite a bit of time, especially with heavy scenes, such as Landmarks Calibration or Gameplay.

Once this scene is displayed, the next scene, previously specified with the Scene Loader, is launched. Because of this, the Loading scene will be displayed for as much time as the next scene takes to load.

Figure 37. Loading scene.

4.3. Important additional elements

4.3.1. Persistent Data

As mentioned previously in section 4.2.11, the Persistent Data object is used to store all of the data regarding the current gameplay and the therapy session in general. It calls a special method called “DontDestroyOnLoad” upon its creation, which makes it so that this game object is not destroyed when its parent scene is destroyed and another one is built. Instead, it stays in the memory and thus can be accessed through every scene that comes after the one it is created in. Because of this, the Persistent Dasa object is first created and used in the very first scene – Main Menu scene.

Persistent Data class also implements the Singletone pattern, which guarantees that there will only ever be one instance of this class and makes the access to this instance simple and straightforward.

The Persistent Data object stores the following information about the therapy session and current gameplay:

- Patient’s unique identifier;
- Therapist’s unique identifier;
- The duration of the therapy session;
- How much time is left in the therapy session at the moment;
- User score (both in the therapy session in general and in the current playthrough);
- The name of the file that stores the selected image and the image itself

(as a Texture2D);
- The coordinates specifying the gameplay area chosen during Table Size Selection;
- Maximum and actual sizes of gameplay area (for convenience);
- Which hand is selected for the therapy session;
- The selected difficulty;
- The name of the file to which the coordinates of the hand recorded throughout the therapy session would be saved once the game is finished;
- The information about each generated puzzle piece mentioned in section 4.2.11;
- Whether the gameplay was stopped and left for the setting scene previously (to signal that the Gameplay scene should rebuild itself with the consideration of the information about puzzle pieces mentioned earlier);
- The coordinates of the hand with timestamps recorded throughout the therapy session.

The Persistent Data object also has methods and events dealing with score changes, the end of the current gameplay and the end of the therapy session.

4.3.2. Scene loader

Scene Loader is a special static class that deals with the transitions between scenes. It has an enum with the names of each game scene in it to minimize the number of possible mistakes caused by the fact that the scene loading in Unity happens with the usage of scenes' names as just plain text. It is also the class that launches the Loading scene before launching the scene it was actually supposed to launch.

4.3.3. Database

Upon the end of the therapy session, some of the data about it previously stored in the Persistent Data will be saved in an SQLite database for the future ease of use and seamless integration with the other components of the M3Display software. It is done by a special class called Database with the usage of Mono.Data.SqliteClient assembly previously added to the current Unity project.

The list of the data stored in the database is specified in section 4.6 Resulting data.

4.3.4. CSV data

As mentioned earlier, during the gameplay the data about the player's hand position is stored in the Persistent Data object. It is done with the usage of a special class that receives the data about the position of the center of the person's hand from the Landmarks Manager, responsible for decoding the data produced by the landmarks determination script, and stores those coordinates together with the timestamp from the time of the beginning of the game.

4.5. Python scripts

The game utilizes several python scripts to work properly. The first set of scripts deals with hand segmentation and landmarks determination and was already mentioned and discussed in details in section 3.2.3.2. Aside from MediaPipe, it utilizes such libraries as cv2, numpy, fpng_py and PIL to work with images. Those scripts are used in Gameplay and Landmarks Calibration scenes to access the position of hand landmarks to determine where the player's hand is at the moment and whether or not it is curled into a fist.

Another python script used in the game forms a graph of the patient's hand movements during the therapy session and relies on previously formed csv file with this data (see sections 4.3.4 and 4.6 for more details). This script relies on the matplotlib library, specifically the pyplot set of functions. It draws points according to the coordinates from the csv file and then connects them with lines to form a trajectory of the hand's movement (Fig. 38).

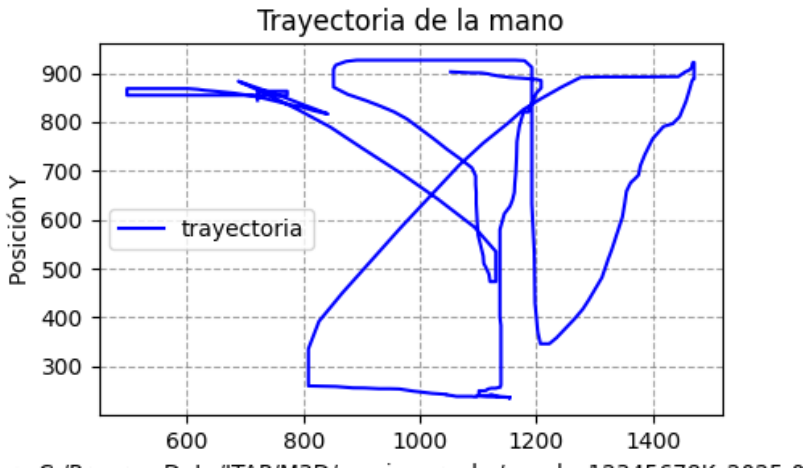

Figure 38. Example of a generated image with the hand movement trajectory.

4.6. Resulting data

As was mentioned in previous sections, after the therapy session is finished, the information about it is stored for future analysis.

The data is stored in three formats – database, csv and image.

In the database, the following data is stored (the names of the corresponding columns in the database are given in the brackets):

- Date of the record (fecha);
- Time of the record (hora);
- Patient's unique identifier (dni_paciente);
- Therapist's unique identifier (dni_terapeuta);
- Name of the game (juego);
- Selected difficulty (dificultad);
- Final score (puntuacion);
- Duration of the therapy session in seconds (duracion_seg);
- The name of the csv file with the hand coordinates (fich_datos);
- The coordinates of the screen edges (ventana).

The csv file, which name is saved in the database, stores the data about hand positions in different points of time that was previously collected in the Persistent Data object (see section 4.3.4). Each record stores a timestamp and the x and y coordinates of the hand's position on the screen. This data is later processed to make a graph of the user's hand movements during the gameplay (see section 4.5).

All data mentioned in this section can be accessed using separate special software developed by ITAP Medical Robotics Research Group or with the use of other apps (by opening the files directly from the file explorer).

4.7. Historical advancements and past versions

During its development, the game discussed in this paper has gone through several versions. Some of the solutions and decisions made for the previous versions of the game still remain in the current version of the game – for example the names of specific scenes or the way some sections of the game are organized.

An example of naming peculiarity caused by the manner in which the previous version of the game was constructed is the Difficulty Initial Selection scene. As it is the only place in the game in which the player can choose the difficulty, the scene could have been named just Difficulty Selection. The reason why it was

originally called differently is that in the first version of the game the difficulty could have been changed in the setting during the gameplay. This feature was later changed, but it was decided to keep the original name of the scene since it still communicates its purpose well. This is also the reason why the DificultyInitialSelectionUI class, instead of just implementing all of the necessary code, instead inherits the DificultySelectionUI class, which was a parent class for both DificultyInitialSelectionUI and DificultySettingsSelectionUI classes.

A feature of the game structure that could have arisen questions is the existence of a settings menu with only one feature aside from going back to the pause menu. Originally, such structural decision was caused by the possibility to change the difficulty of the game during gameplay, as was mentioned earlier. Four buttons seemed to be too much for a pause menu, so both the difficulty change feature and the recalibration were moved to a separate menu. In the current version the game this structure was kept for possible additional expansion of game settings.

5. CONCLUSIONS

5.1. Degree of implementation

During the production of this paper and the implementation of the game this paper is about, all the goals stated in the Objectives section were achieved. The literature review conducted to get a better grasp on the issue that the game aims to facilitate in solving, as well as on the existing solutions and the technologies that would be used in its development are present is Introduction, is summarized in the Literature review and Technologies used sections of this report (number 1.1, 2 and 3 accordingly).

The algorithm for gesture recognition was based on the script developed for hand segmentation in ITAP Medical Robotics Research Group and was explained in sections 3.2.3.2. Hand detection algorithm and 4.2.9. Landmarks Initial Calibration.

The beta-version of the game with reduced functionality, while not discussed in the paper, was developed first to test the implemented hand segmentation algorithm and the functionality related to puzzle assembly. It only consisted of the Gameplay scene and didn't have hand landmarks calibration, relying instead on hardcoded data that was adjusted though trial and error. The beta-version of the program also didn't have difficulty levels, gameplay area size selection, user score and remaining time countdown, as well as any user interface elements and scenes. It did not use an image and instead relied on differently colored squares to imitate a puzzle.

Starting from the beta-version of the program, additional functions stated in further goals were implemented, including grasp strength calibration, gameplay area size selection and difficulty levels, user score, puzzle images and therapy session duration with a countdown. The game also gathers the data about the patient, which was originally done in .json format and was later adapted to fit into the broader M3Display software framework.

5.2. Possible future improvements

While the game in and of itself presents a finished product, there is still a lot of space for future improvements and development.

Such potential areas of improvement may include:

- Aid in puzzle solving – currently, the game relies on the image preview

as the main help in puzzle solving for the player. As patients suffering from a stroke may have serious cognitive issues in addition to the lack of fine motor function, a new feature may be added that will include hints about where to place a puzzle piece that is currently in the player's hand after some time of relative inactivity to make playing the game more enjoyable and less challenging;

- Localization – the game was developed in English as the author of the current article speaks very little Spanish, but an option to choose between the two languages may be very useful as the M3Display is developed in Spain and will probably be used in Spanish hospitals and therapy centers;
- Music and sounds – currently, the only multimedia in the game are the puzzle images themselves. Adding some idle music and sounds corresponding, for example, to pieces being picked up and dropped, will facilitate the user experience and make the therapy process more enjoyable;
- Inclusivity – as of right now, the game works with the assumption that the player does not have any disabilities or disfigurements and is in possession of all 5 fingers on his or her hand. As it may not always be the case, some additional trials should be conducted to determine whether the algorithm is adaptable enough to account for unusual cases and if it isn't, adjustments should be made;
- Productivity – the game in its current form relies on MediaPipe Hands library, which was developed in 2020 and may become obsolete in the near future due to the rapid advancements in the artificial intelligence area in recent years. If this happens and a more optimized solution emerges, the system should be reworked to replace MediaPipe with the new technology.

BIBLIOGRAPHY

[1] "WSO," World Stroke Organization. Accessed: Mar. 12, 2025. [Online]. Available: https://www.world-stroke.org/

[2] "Effects of Stroke | American Stroke Association." Accessed: Mar. 12, 2025. [Online]. Available: https://www.stroke.org/en/about-stroke/effects-of-stroke

[3] "Stroke," *Wikipedia*. Mar. 03, 2025. Accessed: Mar. 12, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stroke&oldid=1278585458#Rehabilitation

[4] "ITAP Medical Robotics - Home - Robótica Médica." Accessed: Mar. 12, 2025. [Online]. Available: https://roboticamedica.itap.uva.es/

[5] A. Cisnal, G. Alonso-Linaje, M. Veganzones, J. P. Turiel, and J. C. Fraile, "M3Display: Sistema de realidad aumentada para la rehabilitacio´n de la funci´on motora del miembro superior," 2023.

[6] G. Sánchez-Brizuela, A. Cisnal, E. de la Fuente-López, J.-C. Fraile, and J. Pérez-Turiel, "Lightweight real-time hand segmentation leveraging MediaPipe landmark detection," *Virtual Real.*, vol. 27, no. 4, pp. 3125–3132, Dec. 2023, doi: 10.1007/s10055-023-00858-0.

[7] A. Farsi, G. L. Cerone, D. Falla, and M. Gazzoni, "Emerging Applications of Augmented and Mixed Reality Technologies in Motor Rehabilitation: A Scoping Review," *Sensors*, vol. 25, no. 7, Art. no. 7, Jan. 2025, doi: 10.3390/s25072042.

[8] M. King, L. Hale, A. Pekkari, and M. Persson, "An affordable, computerized, table-based exercise system for stroke survivors," in *Proceedings of the 3rd International Convention on Rehabilitation Engineering & Assistive Technology*, in i-CREATe '09. New York, NY, USA: Association for Computing Machinery, Apr. 2009, pp. 1–4. doi: 10.1145/1592700.1592717.

[9] N. I. D. Leon, S. K. Bhatt, and A. Al-Jumaily, "Augmented Reality Game Based Multi-Usage Rehabilitation Therapist For Stroke Patients," *Int. J. Smart Sens. Intell. Syst.*, vol. 7, no. 3, pp. 1044–1058, Sep. 2014, doi: 10.21307/ijssis-2017-693.

[10] H. Mousavi Hondori, M. Khademi, L. Dodakian, A. McKenzie, C. V. Lopes, and S. C. Cramer, "Choice of Human-Computer Interaction Mode in Stroke Rehabilitation," *Neurorehabil. Neural Repair*, vol. 30, no. 3, pp. 258–265, Mar. 2016, doi: 10.1177/1545968315593805.

[11] C. Colomer, R. Llorens, E. Noé, and M. Alcañiz, "Effect of a mixed reality-based intervention on arm, hand, and finger function on chronic stroke," *J. NeuroEngineering Rehabil.*, vol. 13, no. 1, p. 45, May 2016, doi: 10.1186/s12984-016-0153-6.

[12] C. Li *et al.*, "Long-term Effectiveness and Adoption of a Cellphone Augmented Reality System on Patients with Stroke: Randomized Controlled Trial," *JMIR Serious Games*, vol. 9, no. 4, p. e30184, Nov. 2021, doi: 10.2196/30184.

[13] A. Pillai, M. S. H. Sunny, M. T. Shahria, N. Banik, and M. H. Rahman,

"Gamification of Upper Limb Rehabilitation in Mixed-Reality Environment," *Appl. Sci.*, vol. 12, no. 23, Art. no. 23, Jan. 2022, doi: 10.3390/app122312260.
[14] M. De Cecco *et al.*, "Sharing Augmented Reality between a Patient and a Clinician for Assessment and Rehabilitation in Daily Living Activities," *Information*, vol. 14, no. 4, Art. no. 4, Apr. 2023, doi: 10.3390/info14040204.
[15] "Unity (game engine)," *Wikipedia*. May 18, 2025. Accessed: May 22, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1290940553
[16] "Instituto de las Tecnologías Avanzadas de la Producción." Accessed: Mar. 12, 2025. [Online]. Available: https://www.itap.uva.es/en/
[17] "Que es M3Display?," Robotica Medica.
[18] "Hand landmarks detection guide | Google AI Edge," Google AI for Developers. Accessed: Mar. 12, 2025. [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker
[19] F. Zhang *et al.*, "MediaPipe Hands: On-device Real-time Hand Tracking," Jun. 18, 2020, *arXiv*: arXiv:2006.10214. doi: 10.48550/arXiv.2006.10214.

ANNEX I – LandmarksGameManager Code

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class LandmarksGameManager : LandmarksManager
{
    /// <summary>
    /// The distances from the 0 point (start of the hand) to the end
of fingers
    /// </summary>
    private readonly List<Vector2> _handVectors = new List<Vector2>();

    /// <summary>
    /// Prefab that will initiate the fist for each hand.
    /// </summary>
    [SerializeField]
    private FistTemplate _prefabFist;
    /// <summary>
    /// Transform of the gameobject that will be the parent of the
hands.
    /// </summary>
    [SerializeField]
    private Transform _fistsParent;


    [SerializeField]
    private CSVPosition _csvPosition;

    private FistTemplate _fist;

    /// <summary>
    /// Resets the position of every landmark object in a specified
hand (List of landmarks).
    /// The position of every landmark after resetting is -100, -100,
50
    /// </summary>
    /// <param name="hand">Initial letter of the hand which positions
should be reset ('L' or 'R'). </param>
    private void ResetHand()
    {
        foreach (var vect in _handVectors)
        {
            vect.Set(-100, -100);
        }

        _fist.TryReleasePuzzlePiece();
        _fist.IsClenched = false;
        _fist.transform.localPosition = new Vector3(-500, -500, 50);
        _csvPosition.WriteCoords(Vector3.zero);
    }

    /// <summary>
    /// Called before the first frame update.
    /// </summary>
    void Awake()
    {
        for (int i = 0; i < 21; i++)
```

```
        {
            _handVectors.Add(new Vector2());
        }

        _fist = Instantiate(_prefabFist, _fistsParent);
        _fist.name = "Fist";

        ResetHand();
    }

    /// <summary>
    /// Processes the message composed by the coordinates of each
landmark, changing the corresponding gameobject position.
    /// </summary>
    void Update()
    {
        if (_message == null)
        {
            ResetHand();
            _csvPosition.WriteCoords(Vector3.zero);
            return;
        }

        var separated = _message.Split(',');
        // First hand
        var hand = separated[0];
        if (hand != "-100")
        {
            for (int i = 2; i < (2 + 21 * 6 / 2); i += 3)
            {
                var index = Int32.Parse(separated[i]);
                // First hand
                if (index != -100)
                {
                    _handVectors[index] = new Vector2(
                        Int32.Parse(separated[i + 1]),
Int32.Parse(separated[i + 2]));
                }
            }

            if (hand == "R" && PersistentData.Instance.SelectedHand ==
Hand.Right
                || hand == "L" && PersistentData.Instance.SelectedHand
== Hand.Left)
            {
                var handPosition = new Vector3(
                        (_handVectors[0].x + _handVectors[9].x) / 2,
                        (_handVectors[0].y + _handVectors[9].y) / 2,
50);


_csvPosition.WriteCoords(transform.TransformPoint(handPosition));

                // checking if the right hand is in a fist
                var clenched = true;
                for (int i = 0; i <
LandmarksSettings.MaxHandDistances.Length; i++)
                {
                    var distanceStartFinger = Vector2.Distance
                        (_handVectors[0], _handVectors[i * 4 + 1]);
```

```
                    var distanceEndFinger = Vector2.Distance
                        (_handVectors[0], _handVectors[(i + 1) * 4]);

                    var fingersDistance = distanceEndFinger /
distanceStartFinger;

                    if (fingersDistance >
LandmarksSettings.MaxHandDistances[i])
                    {
                        clenched = false;
                        break;
                    }
                }

                // moving fist object and taking object in hand of
found
                if (clenched)
                {
                    _fist.transform.localPosition = handPosition;

                    if (!_fist.IsClenched)
                    {
                        _fist.IsClenched = true;
                        _fist.TryTakePuzzlePiece();
                    }
                }
                else
                {
                    _fist.TryReleasePuzzlePiece();
                    _fist.IsClenched = false;
                    _fist.transform.localPosition = new Vector3(-500, -
500, 50);
                }
            }
            else
            {
                ResetHand();
            }
        }
        else
        {
            ResetHand();
        }
    }
}
```

ANNEX II – Objects Code

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Objects : MonoBehaviour
{
    [SerializeField]
    private PuzzleAssemblySpace _puzzleAssemblySpacePrefab;

    [SerializeField]
    private PuzzlePiece _puzzlePiecePrefab;

    private List<PuzzlePiece> _puzzlePieces;

    private void Awake()
    {
        _puzzlePieces = new List<PuzzlePiece>();
        var piecesPlacementDetails = new
List<PuzzlePlacementDetails>();

        var puzzleSize = DificultySettings.PuzzleSizesDictionary
            [PersistentData.Instance.SelectedDificulty];

        var piecesAmount = puzzleSize.Item1 * puzzleSize.Item2;

        if (PersistentData.Instance.InitialLaunch)
        {
            var piecesDetails = new List<PuzzlePieceDetails>();
            var piecePositions =
GenerateRandomPiecesPlacement(piecesAmount);

            var parts =
ImageManager.SliceTexture(PersistentData.Instance.ChosenImage,
puzzleSize.Item2, puzzleSize.Item1);

            var spaceFromBorder = 100 +
(PersistentData.Instance.MaxDisplaySizeX -
PersistentData.Instance.DisplaySizeX) / 2;

            for (int i = 0; i < piecesAmount; i++)
            {
                // make a piece
                var piece = Instantiate(_puzzlePiecePrefab, transform);
                piece.SetId(null);
                piece.SetOriginalParent(transform);
                piece.SetSize();

                // give position
                piece.transform.localPosition = new Vector3(
                        spaceFromBorder +
(PersistentData.Instance.MaxDisplaySizeX - spaceFromBorder * 2) /
                        piecesAmount * (piecePositions[i] * 2 + 1) / 2,
                        PersistentData.Instance.MaxDisplaySizeY -
piece.transform.localScale.y / 2
```

```
                            - 100 *
(PersistentData.Instance.DisplaySizeY /
PersistentData.Instance.MaxDisplaySizeY) - 20,
                        10 + piecePositions[i]);

                Material mat = new Material(Shader.Find("Standard"));
                mat.mainTexture = parts[i];

                piece.GetComponent<Renderer>().material = mat;

                piece.name = $"Piece{i + 1}";

                _puzzlePieces.Add(piece);

                // generate a piece of info regarding where the piece
should be in the puzzle
                var piecePlacement = new PuzzlePlacementDetails(
                    piece.Id, new Vector2(i % puzzleSize.Item1, i /
puzzleSize.Item1));
                piecesPlacementDetails.Add(piecePlacement);

                piecesDetails.Add(new PuzzlePieceDetails(
                    piece.Id, piece.transform.localPosition,
                    piecePlacement.PuzzlePosition, mat));
            }

PersistentData.Instance.SetPuzzlePieceDetails(piecesDetails);
            //GameState.DisplayPiecesPositions();
        }
        else
        {
            PersistentData.Instance.ResetScore();

            var piecesDetails =
PersistentData.Instance.PuzzlePiecesDetails
                .OrderBy(p => p.CurrentPosition.z).ToList();
            print("Loading game state");
            //GameState.DisplayPiecesPositions();
            for (int i = 0; i <
PersistentData.Instance.PuzzlePiecesDetails.Count; i++)
            {
                // make a piece
                var piece = Instantiate(_puzzlePiecePrefab, transform);
                piece.SetId(piecesDetails[i].Id);
                piece.SetOriginalParent(transform);
                piece.SetSize();

                //piece.GetComponent<Renderer>().material.color =
piecesDetails[i].Color; // temp
                piece.GetComponent<Renderer>().material =
piecesDetails[i].Material;
                piece.transform.localPosition =
piecesDetails[i].CurrentPosition;
                piece.name = $"Piece{i + 1}";
                _puzzlePieces.Add(piece);

                // generate a piece of info regarding where the piece
should be in the puzzle
                var piecePlacement = new PuzzlePlacementDetails(
                    piece.Id, piecesDetails[i].PlacementPosition);
```

```
                piecesPlacementDetails.Add(piecePlacement);
            }
        }

        // create the space where the pieces should be assembled
        var assemblySpace = Instantiate(_puzzleAssemblySpacePrefab,
transform);
        assemblySpace.SetSize();
        assemblySpace.transform.localPosition = new
Vector3(PersistentData.Instance.MaxDisplaySizeX / 2,
            PersistentData.Instance.MaxDisplaySizeY -
PersistentData.Instance.DisplaySizeY +
PersistentData.Instance.PuzzleSizeY / 2 +
            200 * (PersistentData.Instance.DisplaySizeY /
PersistentData.Instance.MaxDisplaySizeY - 0.5f), 5);
        assemblySpace.GeneratePlacementSpaces(piecesPlacementDetails);
    }

    private void Start()
    {
        StartCoroutine(DelayedDrop());
    }

    private IEnumerator DelayedDrop()
    {
        yield return new WaitForSeconds(0.05f);
        foreach (var piece in _puzzlePieces)
        {
            piece.GetDropped();
        }
    }

    private int[] GenerateRandomPiecesPlacement(int piecesAmount)
    {
        var piecePositions = new int[piecesAmount];
        for (int i = 0; i < piecesAmount; i++)
        {
            piecePositions[i] = i;
        }

        int tempStorage;
        var rand = new System.Random();

        for (int i = 0; i < 100; i++)
        {
            var firstElement = rand.Next(piecesAmount);
            var secondElement = rand.Next(piecesAmount);

            tempStorage = piecePositions[firstElement];
            piecePositions[firstElement] =
piecePositions[secondElement];
            piecePositions[secondElement] = tempStorage;
        }


        return piecePositions;
    }
}
```

ANNEX III – VideoFeed Code

```
using UnityEngine;
using System;
using System.Diagnostics;

using ImgProcessing.PythonImplementation;

/// <summary>
/// Componente responsable del manejo de la webcam y del display de los
frames en una textura. Crea una textura con dos frames, esto se hace
/// para garantizar la funcionalidad de los juegos legacy de M3Display.
/// </summary>
public class VideoFeed : MonoBehaviour
{
    /*
     * Los renderers de los objetos sobre los que se va a aplicar la
textura con la cámara.
     */
    [SerializeField]
    private Renderer _tableRendererLeft;

    [SerializeField]
    private Renderer _tableRendererRight;

    /*
     * El objeto al que se mandará el mensaje con los landmarks de
MediaPipe.
     */
    [SerializeField]
    private LandmarksManager _landmarksManager;

    //La instancia del procesador de las texturas.
    private ImgProcessor _processor;
    //La textura en la que se guardan los frames de la cámara web.
    private WebCamTexture _webCam;

    //Distintas texturas temporales que se utilizan en el resto del
código.
    //Es más rápido crearlas globalmente y actualizarlas que crearlas
cada vez que se necesiten
    private Texture2D _originalFeedTexture;
    private Texture2D _tempUnprocessed;
    private Texture2D _tempProcessed;


    /// <summary>
    /// Se le llama una vez antes del primer frame.
    /// </summary>
    void Start()
    {
        //Obtenemos los dispositivos de cámara, elegimos uno, creamos
una WebCamTexture y empezamos a obtener frames
        WebCamDevice[] devices = WebCamTexture.devices;
        _webCam = new WebCamTexture(devices[0].name, 1280, 720, 30);
        _webCam.Play();

        //Creamos una Texture2D que servirá para guardar los datos de
la textura webcam, y la inicializamos
```

```
        _originalFeedTexture = new Texture2D(_webCam.width,
_webCam.height);
        _originalFeedTexture.SetPixels(_webCam.GetPixels());
        _originalFeedTexture.Apply();

        //Creamos un ImgProcessor e iniciamos su runtime
        _processor = new ImgProcessor();
        _processor.StartRuntime();

        //Inicializamos las texturas temporales
        _tempUnprocessed = new Texture2D(_webCam.width,
_webCam.height);
        _tempProcessed = new Texture2D(_webCam.width, _webCam.height);
    }

    /// <summary>
    /// Se le llama una vez por frame.
    /// </summary>
    void Update()
    {
        if (!PersistentData.Instance.IsPaused)
        {
            //Obtenemos el frame de la cámara
            Graphics.CopyTexture(_webCam, 0, 0, 0, 0, _webCam.width,
_webCam.height, _originalFeedTexture, 0, 0, 0, 0);
            _originalFeedTexture.Apply();

            //Redimensionamos el frame (si es necesario) y lo guardamos
en _tempUnprocessed
            //TextureUtils.ResizeAndMove(_originalFeedTexture,
_tempUnprocessed, 1280, 720);
            Graphics.CopyTexture(_originalFeedTexture, 0, 0, 0, 0,
_originalFeedTexture.width, _originalFeedTexture.height,
_tempUnprocessed, 0, 0, 0, 0);

            //Volteamos la textura sin procesar en el eje horizontal
(M3Display funciona con un espejo)
            //TextureUtils.Flip(_tempUnprocessed);

            //Convertimos la textura a un array de bytes codificado en
JPG
            Byte[] frameJPG =
ImageConversion.EncodeToJPG(_tempUnprocessed);

            //Obtenemos el mensaje con los landmarks de MediaPipe y se
lo enviamos a la clase responsable
            String message = _processor.ProcessMessage(frameJPG, 1920,
1080);
            _landmarksManager.UpdateMessage(message);

            //Obtenemos el frame procesado y lo guardamos en una
textura
            Byte[] processedByteArray =
_processor.ProcessBitmap(frameJPG, 1280, 720);
            _tempProcessed.LoadImage(processedByteArray);

            //Actualizamos la textura del renderer
            _tableRendererLeft.material.mainTexture = _tempUnprocessed;
            _tableRendererRight.material.mainTexture = _tempProcessed;
        }
```

```
    }

    private void OnDestroy()
    {
        //Cerramos la textura webcam y el runtime del ImgProcessor
        _webCam.Stop();
        _processor.StopRuntime();
    }
}
```

ANNEX IV – FistTemplate Code

```
using UnityEngine;

public class FistTemplate : MonoBehaviour
{
    /// <summary>
    /// Layer mask of the puzzle pieces.
    /// Needed to see if the place over which the fist is
    /// formed is a puzzle piece
    /// </summary>
    [SerializeField]
    private LayerMask _puzzlePieceLayerMask;

    /// <summary>
    /// Current puzzle piece
    /// </summary>
    private PuzzlePiece _puzzlePieceInHand;

    /// <summary>
    /// A bool value to (idealy) only pick up a puzzle piece
    /// when the fist is formed and not just hovering over it.
    /// </summary>
    public bool IsClenched = false;


    public void TryTakePuzzlePiece()
    {
        if (Physics.Raycast(transform.position, -transform.forward, out
RaycastHit hit, 100f, _puzzlePieceLayerMask))
        {
            if (hit.transform.TryGetComponent(out PuzzlePiece
puzzlePiece))
            {
                puzzlePiece.GetPickedUp(transform);
                _puzzlePieceInHand = puzzlePiece;
            }
        }
    }

    public void TryReleasePuzzlePiece()
    {
        if (_puzzlePieceInHand != null)
        {
            _puzzlePieceInHand.GetDropped();
            _puzzlePieceInHand = null;
        }
    }
}
```

ANNEX V – PuzzleAssemblySpace Code

```
using System.Collections.Generic;
using UnityEngine;

public class PuzzleAssemblySpace : MonoBehaviour
{
    [SerializeField]
    private Transform _puzzlePlacementPointParent;

    [SerializeField]
    private PuzzlePiecePlacementPoint _puzzlePlacementPointPrefab;

    [SerializeField]
    private Transform _objectCube;

    /// <summary>
    /// Generates spots to which puzzle pieces will be connected
    /// </summary>
    /// <param name="puzzlePlacementDetails"></param>
    public void GeneratePlacementSpaces(
        List<PuzzlePlacementDetails> puzzlePlacementDetails)
    {
        foreach (var details in puzzlePlacementDetails)
        {
            var puzzlePlacementPoint = Instantiate(
                _puzzlePlacementPointPrefab,
_puzzlePlacementPointParent);

            puzzlePlacementPoint.SetPuzzlePieceId(details.PuzzleId);

            var position = details.PuzzlePosition;

            var puzzleSize = DificultySettings

.PuzzleSizesDictionary[PersistentData.Instance.SelectedDificulty];

            var pieceSizeX = PersistentData.Instance.PuzzleSizeX /
puzzleSize.Item1;
            var pieceSizeY = PersistentData.Instance.PuzzleSizeY /
puzzleSize.Item2;

            puzzlePlacementPoint.transform.localPosition = new Vector3(
                pieceSizeX * (position.x * 2 + 1) / 2 -
puzzleSize.Item1 * pieceSizeX / 2 ,
                pieceSizeY * (position.y * 2 + 1) / 2 -
puzzleSize.Item2 * pieceSizeY / 2, 1);

            // set size from dificulty
            puzzlePlacementPoint.SetSize((int)pieceSizeX - 50,
(int)pieceSizeY - 50);
        }

        print("palacement spaces generated");
    }

    public void SetSize()
    {
        _objectCube.localScale = new Vector3(
```

```
                PersistentData.Instance.PuzzleSizeX + 50,
                PersistentData.Instance.PuzzleSizeY + 50, 1);
        }
    }
```

ANNEX VI – PuzzlePiece Code

```
using System;
using UnityEngine;

public class PuzzlePiece : MonoBehaviour
{
    /// <summary>
    /// Table from which the puzzle pieces are initially taken
    /// </summary>
    private Transform _originalParent;

    /// <summary>
    /// Layer mask of puzzle placement point.
    /// Needed so that pieces can see if the place they are
    /// being dropped at is a placement point
    /// </summary>
    [SerializeField]
    private LayerMask _puzzlePiecePlacementPointLayerMask;

    /// <summary>
    /// The spot that the puzzle piece will get glued to
    /// </summary>
    private Transform _puzzlePieceCarrier;

    /// <summary>
    /// Unique identifyer of a puzzle piece -
    /// needed to know where the piece must be placed
    /// </summary>
    public Guid Id { get; private set; }

    public void SetId(Guid? id)
    {
        Id = id ?? Guid.NewGuid();
    }

    public void SetOriginalParent(Transform originalParent)
    {
        _originalParent = originalParent;
    }

    public void GetPickedUp(Transform hand)
    {
        if (_puzzlePieceCarrier != null &&
            _puzzlePieceCarrier.TryGetComponent(
                out PuzzlePiecePlacementPoint placementPoint))
        {
            placementPoint.TryReleasePuzzlePiece();
        }
        _puzzlePieceCarrier = hand;

        transform.parent = hand;
        transform.localPosition = Vector3.zero;
    }

    public void GetDropped()
    {
        // get droped on the table
        transform.parent = _originalParent;
```

```
        _puzzlePieceCarrier = null;
        transform.localPosition = new Vector3(
            transform.localPosition.x, transform.localPosition.y, 10);

        PersistentData.Instance.UpdatePuzzlePiecePosition(Id,
transform.localPosition);

        // check for piece placement point
        if (Physics.Raycast(transform.position, new Vector3(0, -1, 0),
            out RaycastHit hit, 100f,
_puzzlePiecePlacementPointLayerMask))
        {
            if (hit.transform.parent != null &&
                hit.transform.parent.transform.
                TryGetComponent(out PuzzlePiecePlacementPoint
placementPoint)
                && !placementPoint.IsOcupied())
            {
                // get connected to a piece placement point
                placementPoint.SetPuzzlePiece(this);
                _puzzlePieceCarrier = placementPoint.transform;
                transform.parent = placementPoint.transform;
                transform.localPosition = Vector3.zero;
                transform.rotation = new Quaternion();
            }
        }
    }

    public void SetSize()
    {
        var puzzleSize = DificultySettings

.PuzzleSizesDictionary[PersistentData.Instance.SelectedDificulty];

        transform.localScale = new Vector3(
            PersistentData.Instance.PuzzleSizeX / puzzleSize.Item1,
            PersistentData.Instance.PuzzleSizeY / puzzleSize.Item2, 1);
    }
}
```

ANNEX VII – ImageManager Code

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using UnityEngine;

public static class ImageManager
{
    private const string galleryFolder = "Gallery";
    private const string galleryDefaultsFolder = "GalleryDefaults";
    private static bool initialized = false;

    private static void Init()
    {
        if (initialized)
            return;
        initialized = true;

        string galleryPath =
Path.Combine(Application.persistentDataPath, galleryFolder);

        if (Directory.Exists(galleryPath))
            return;

        Directory.CreateDirectory(galleryPath);
        string defaultPath =
Path.Combine(Application.streamingAssetsPath, galleryDefaultsFolder);

        if (!Directory.Exists(defaultPath))
            return;

        foreach (var filePath in Directory.GetFiles(defaultPath))
        {
            string fileName = Path.GetFileName(filePath);
            string destPath = Path.Combine(defaultPath, fileName);
            File.Copy(filePath, destPath, true);
        }
    }

    public static string SaveImage(string sourcePath)
    {
        byte[] fileData = File.ReadAllBytes(sourcePath);

        // Ensure directory exists
        string galleryDir =
Path.Combine(Application.persistentDataPath, galleryFolder);
        Directory.CreateDirectory(galleryDir);

        // Save the image in gallery
        string fileName = Path.GetFileName(sourcePath);
        string destinationPath = Path.Combine(galleryDir, fileName);
        File.WriteAllBytes(destinationPath, fileData);

        return fileName;
    }

    public static void DeleteImage(string fileName)
    {
```

```
        string galleryDir =
Path.Combine(Application.persistentDataPath, galleryFolder);
        string filePath = Path.Combine(galleryDir, fileName);

        if (File.Exists(filePath))
        {
            File.Delete(filePath);
        }
        else
        {
            Debug.LogWarning("File not found: " + filePath);
        }
    }

    public static Texture2D LoadImage(string fileName)
    {
        Init();
        string filePath = Path.Combine(Application.persistentDataPath,
galleryFolder, fileName);

        if (File.Exists(filePath))
        {
            byte[] fileBytes = File.ReadAllBytes(filePath);
            Texture2D texture = new Texture2D(2, 2);
            texture.LoadImage(fileBytes);

            return texture;
        }
        else
        {
            Debug.LogError("File not found: " + filePath);
            return null;
        }
    }

    public static List<(string, Texture2D)> LoadAllImagesWithNames()
    {
        Init();
        var allImages = new List<(string, Texture2D)>();

        string galleryPath =
Path.Combine(Application.persistentDataPath, galleryFolder);
        if (Directory.Exists(galleryPath))
        {
            string[] imageFiles = Directory.GetFiles(galleryPath,
"*.*")
                .Where(f => f.EndsWith(".png") || f.EndsWith(".jpg") ||
f.EndsWith(".jpeg"))
                .OrderBy(path => new
FileInfo(path).LastWriteTime).ToArray();

            foreach (string filePath in imageFiles)
            {
                byte[] bytes = File.ReadAllBytes(filePath);
                Texture2D tex = new Texture2D(2, 2);
                tex.LoadImage(bytes);
                allImages.Add((Path.GetFileName(filePath), tex));
            }
        }
```

```
        return allImages;
    }

    public static List<Texture2D> LoadAllImages()
    {
        return LoadAllImagesWithNames()
            .Select(pair => pair.Item2).ToList();
    }

    public static Texture2D[] SliceTexture(Texture2D image, int rows,
int cols)
    {
        var originalWidth = (float)image.width;
        var originalHeight = (float)image.height;
        if (originalWidth / originalHeight >
PersistentData.Instance.MaxPuzzleSizeX /
PersistentData.Instance.MaxPuzzleSizeY)
        {
            PersistentData.Instance.PuzzleSizeX =
PersistentData.Instance.MaxPuzzleSizeX;
            PersistentData.Instance.PuzzleSizeY =
PersistentData.Instance.MaxPuzzleSizeX * originalHeight /
originalWidth;
        }
        else
        {
            PersistentData.Instance.PuzzleSizeY =
PersistentData.Instance.MaxPuzzleSizeY;
            PersistentData.Instance.PuzzleSizeX =
PersistentData.Instance.MaxPuzzleSizeY * originalWidth /
originalHeight;
        }

        var resized = ResizeTexture(image,
(int)PersistentData.Instance.PuzzleSizeX,
            (int)PersistentData.Instance.PuzzleSizeY);
        var flipped = FlipTextureVertically(resized);
        flipped = FlipTextureHorizontally(flipped);

        int pieceWidth = flipped.width / cols;
        int pieceHeight = flipped.height / rows;
        var pieces = new Texture2D[rows * cols];

        for (int y = 0; y < rows; y++)
        {
            for (int x = 0; x < cols; x++)
            {
                Texture2D piece = new Texture2D(pieceWidth,
pieceHeight);
                piece.SetPixels(flipped.GetPixels(flipped.width - (x +
1) * pieceWidth, y * pieceHeight, pieceWidth, pieceHeight));
                piece.Apply();
                pieces[y * cols + x] = piece;
            }
        }

        return pieces;
    }
```

```
    public static Texture2D ResizeTexture(Texture2D source, int
targetWidth, int targetHeight)
    {
        RenderTexture rt = RenderTexture.GetTemporary(targetWidth,
targetHeight);
        RenderTexture.active = rt;

        Graphics.Blit(source, rt);

        Texture2D result = new Texture2D(targetWidth, targetHeight,
TextureFormat.RGBA32, false);
        result.ReadPixels(new Rect(0, 0, targetWidth, targetHeight), 0,
0);
        result.Apply();

        RenderTexture.active = null;
        RenderTexture.ReleaseTemporary(rt);
        return result;
    }

    public static Texture2D FlipTextureVertically(Texture2D original)
    {
        int width = original.width;
        int height = original.height;

        Texture2D flipped = new Texture2D(width, height,
original.format, false);

        for (int y = 0; y < height; y++)
        {
            flipped.SetPixels(0, y, width, 1, original.GetPixels(0,
height - 1 - y, width, 1));
        }

        flipped.Apply();
        return flipped;
    }

    public static Texture2D FlipTextureHorizontally(Texture2D original)
    {
        int width = original.width;
        int height = original.height;

        Texture2D flipped = new Texture2D(width, height,
original.format, false);

        for (int y = 0; y < height; y++)
        {
            Color[] row = original.GetPixels(0, y, width, 1);
            System.Array.Reverse(row);
            flipped.SetPixels(0, y, width, 1, row);
        }

        flipped.Apply();
        return flipped;
    }
}
```

ANNEX VIII – ImagesPython Code

```
using Python.Runtime;
using UnityEngine;

public class ImagesPython : MonoBehaviour
{
    private readonly string csvPath = "C:/ProgramData/ITAP/M3D/csv/";
    private readonly string imgPath =
"C:/ProgramData/ITAP/M3D/sessiongraphs/";

    private string pythonDll =
"C:\\ProgramData\\M3DPythonService\\Python_DLL\\dll\\python311.dll";
    private string filePath =
"C:\\ProgramData\\M3DPythonService\\Python_DLL\\";

    public void StartRuntime()
    {
        Runtime.PythonDLL = pythonDll;
        PythonEngine.Initialize();
        using (Py.GIL())
        {
            PyObject pyObject = Py.Import("os");
            PyObject pyObject2 = Py.Import("sys");
            pyObject2.GetAttr("path").InvokeMethod("append",
filePath.ToPython());
        }
    }

    public void StopRuntime()
    {
        PythonEngine.Shutdown();
    }

    public void CreatePythonImage()
    {
        //string pythonDll =
@"C:\ProgramData\M3DPythonService\Python_DLL\dll\python311.dll";
        //Runtime.PythonDLL = pythonDll;
        //PythonEngine.Initialize();
        StartRuntime();

        using (Py.GIL())
        {
            PyObject fromFile = Py.Import("funcionGrafica");
            string filePath = csvPath +
PersistentData.Instance.FileName;
            Debug.Log(PersistentData.Instance.CSVPrint.Count);
            Debug.Log(filePath);
            PyObject csvFilePath = filePath.ToPython();
            PyObject fileName =
PersistentData.Instance.FileName.ToPython();
            PyObject pngPath = imgPath.ToPython();

            Debug.Log(csvFilePath);
            Debug.Log(fileName);
            Debug.Log(pngPath);
```

```
            PyObject result = fromFile.InvokeMethod("createImage",
csvFilePath, fileName, pngPath);
        }

        StopRuntime();
    }
}
```

ANNEX IX – Database Code

```
using System.Collections.Generic;
using System;
using UnityEngine;
using CsvHelper;
using System.Globalization;
using System.IO;
using CsvHelper.Configuration;
using System.Text;
using Mono.Data.Sqlite;

public class Database : MonoBehaviour
{
    private readonly string juegoString = "puzzle";

    private readonly string dbPath =
"C:/ProgramData/ITAP/M3D/database/";
    private readonly string csvPath = "C:/ProgramData/ITAP/M3D/csv/";

    public class Sesiones
    {
        string dbPath = "C:/ProgramData/ITAP/M3D/database/";
        public Sesiones(DateTime _fec, string _dniPac, string _dniTer,
string _juego, string _dif, int _punt, int _dur_seg, string _fich_dat,
string _ventana)
        {
            fechahora = _fec;
            dni_paciente = _dniPac;
            dni_terapeuta = _dniTer;
            juego = _juego;
            dificultad = _dif;
            puntuacion = _punt;
            duracion_seg = _dur_seg;
            fich_datos = _fich_dat;
            ventana = _ventana;
        }
        public DateTime fechahora{ get; set; }
        public string dni_paciente { get; set; }
        public string dni_terapeuta { get; set; }
        public string juego { get; set; }
        public string dificultad { get; set; }
        public int puntuacion { get; set; }
        public int duracion_seg { get; set; }
        public string fich_datos { get; set; }
        public string ventana { get; set; }
        public void saveToDB()
        {
            using (SqliteConnection conn = new SqliteConnection("Data
Source=" + dbPath + "M3Display.db;Version=3;"))
            {
                conn.Open();
                string insert = "INSERT INTO sesiones
(fecha,hora,dni_paciente,dni_terapeuta,juego,dificultad,puntuacion,dura
cion_seg,fich_datos,ventana) VALUES
(@fecha,@hora,@dni_paciente,@dni_terapeuta,@juego,@dificultad,@puntuaci
on,@duracion_seg,@fich_datos,@ventana)";
                SqliteCommand command = new SqliteCommand(null, conn);
                command.CommandText = insert;
```

```
                command.Parameters.AddWithValue("@fecha",
fechahora.Date.ToString("yyyy-MM-dd"));
                command.Parameters.AddWithValue("@hora",
fechahora.TimeOfDay.ToString("hh\\:mm\\:ss"));
                command.Parameters.AddWithValue("@dni_paciente",
dni_paciente);
                command.Parameters.AddWithValue("@dni_terapeuta",
dni_terapeuta);
                command.Parameters.AddWithValue("@juego", juego);
                command.Parameters.AddWithValue("@dificultad",
dificultad);
                command.Parameters.AddWithValue("@puntuacion",
puntuacion);
                command.Parameters.AddWithValue("@duracion_seg",
duracion_seg);
                command.Parameters.AddWithValue("@fich_datos",
fich_datos);
                command.Parameters.AddWithValue("@ventana", ventana);
                command.ExecuteNonQuery();
                conn.Close();
            }
        }
    }

    public List<string> readDNIPacientes()
    {
        List<string> listaDNI = new List<string>();
        using (SqliteConnection conn = new SqliteConnection("Data
Source="+ dbPath + "M3Display.db;Version=3;"))
        {
            conn.Open();
            string query = "SELECT dni FROM pacientes";
            SqliteCommand command = new SqliteCommand(query, conn);
            SqliteDataReader reader = command.ExecuteReader();
            while (reader.Read())
            {
                listaDNI.Add((string)reader["dni"]);
            }
            conn.Close();
        }
        return listaDNI;
    }
    public List<string> readDNITerapeutas()
    {
        List<string> listaDNI = new List<string>();
        using (SqliteConnection conn = new SqliteConnection("Data
Source=" + dbPath + "M3Display.db;Version=3;"))
        {
            conn.Open();
            string query = "SELECT dni FROM terapeutas";
            SqliteCommand command = new SqliteCommand(query, conn);
            SqliteDataReader reader = command.ExecuteReader();
            while (reader.Read())
            {
                listaDNI.Add((string)reader["dni"]);
            }
            conn.Close();
        }
        return listaDNI;
    }
```

```
    public void saveSession()
    {
        string dniPaciente = PersistentData.Instance.PatientId;
        string dniTerapeuta = PersistentData.Instance.TherapistId;
        int puntuacion = PersistentData.Instance.UserScoreCombined;
        int durSegundos = PersistentData.Instance.TimeSecs;
        DateTime dateTimeToUse = DateTime.Now;
        string dificultad =
PersistentData.Instance.SelectedDificulty.ToString();
        string fichero = juegoString + "_" + dniPaciente + "_"
            + dateTimeToUse.ToString("yyyy-MM-dd_THH-mm-ss");
        PersistentData.Instance.FileName = fichero;
        string ventanaString = PersistentData.Instance.MinX + "," +
PersistentData.Instance.MaxX+","

+PersistentData.Instance.MinY+","+PersistentData.Instance.MaxY;
        Sesiones sesion = new Sesiones(dateTimeToUse, dniPaciente,
dniTerapeuta,
            juegoString, dificultad, puntuacion, durSegundos, fichero,
ventanaString);
        sesion.saveToDB();
        var config = new CsvConfiguration(new CultureInfo("es-ES",
false)) { Delimiter = ";", Encoding = Encoding.UTF8 };
        FileStream stream = new FileStream(csvPath + fichero + ".csv",
FileMode.CreateNew);
        using (var writer = new StreamWriter(stream))
        using (var csv = new CsvWriter(writer, config))
        {
            csv.WriteRecords(PersistentData.Instance.CSVPrint);
        }
    }

    public void leaveWithoutCSV()
    {
        string dniPaciente = PersistentData.Instance.PatientId;
        string dniTerapeuta = PersistentData.Instance.TherapistId;
        DateTime dateTimeToUse = DateTime.Now;
        string dificultad =
PersistentData.Instance.SelectedDificulty.ToString();
        string fichero = juegoString + "_" + dniPaciente + "_"
            + dateTimeToUse.ToString("yyyy-MM-dd_THH-mm-ss");
        PersistentData.Instance.FileName = fichero;
        string ventanaString = PersistentData.Instance.MinX + "," +
PersistentData.Instance.MaxX + ","
            + PersistentData.Instance.MinY + "," +
PersistentData.Instance.MaxY;
        Sesiones sesion = new Sesiones(dateTimeToUse, dniPaciente,
dniTerapeuta,
            juegoString, dificultad, -1, -1, "-1", ventanaString);
        sesion.saveToDB();
    }

    public void leaveWithoutPoints()
    {
        string dniPaciente = PersistentData.Instance.PatientId;
        string dniTerapeuta = PersistentData.Instance.TherapistId;
        DateTime dateTimeToUse = DateTime.Now;
        string fichero = juegoString + "_" + dniPaciente + "_" +
dateTimeToUse.ToString("yyyy-MM-dd_THH-mm-ss");
```

```
        PersistentData.Instance.FileName = fichero;
        string dificultad =
PersistentData.Instance.SelectedDificulty.ToString();
        string ventanaString = PersistentData.Instance.MinX + "," +
PersistentData.Instance.MaxX + ","
            + PersistentData.Instance.MinY + "," +
PersistentData.Instance.MaxY;
        Sesiones sesion = new Sesiones(dateTimeToUse, dniPaciente,
dniTerapeuta, juegoString, dificultad, -1, -1, fichero, ventanaString);
        sesion.saveToDB();
        var config = new CsvConfiguration(new CultureInfo("es-ES",
false)) { Delimiter = ";", Encoding = Encoding.UTF8 };
        FileStream stream = new FileStream(csvPath + fichero + ".csv",
FileMode.CreateNew);
        using (var writer = new StreamWriter(stream))
        using (var csv = new CsvWriter(writer, config))
        {
            csv.WriteRecords(PersistentData.Instance.CSVPrint);
        }
    }
}
```