
Helwan University Faculty of engineering Computer
department

Physical therapy
using video games

Presented by:

- Marwa Abd-Elbasit Ismail Marzook
- Mostafa Tarek Abd-ElBar Seif
- Hatem Mohamed Salah Abd-Elmaqsod
- Mostafa Ali Abd-Elhamid
- Mostafa Osama Bakry

Supervised by:

Prof.Dr. Samir Gaber

*The best physical therapy session is in
the form of a game like exercise*

Abstract

Physical therapy (PT), also known as physiotherapy, is in demand nowadays. Someone in the United States has a stroke every 40 seconds. According to the World Health Organization (WHO), 15 million people suffer stroke worldwide each year. Of these, 5 million die and another 5 million are left permanently disabled. The need of tele-physical therapy, even in the presence of corona virus is in serious demand. We therefore intend to provide a solution by developing and designing a tele-physical therapy platform that has the capability to provide people having strokes and in need for rehabilitation with exercises through playing videogames that have on-screen assessments and exercise instruction. Also, the doctors will be able to see if the patients are getting better through statistics and reports on the web application. This system containing a collection of videogames ensures the stroke patients rehabilitation while staying safe at home.

Approximately 20% of PT patients drop out of treatment within the first three visits, and 70% fail to complete their full course of care. So, we created a more fun, engaging, and motivating way of doing physical therapy exercises by playing games that help the patient perform the sufficient exercises for his/her recovery.

Keywords: Physical therapy, physiotherapy, Stroke rehabilitation, Videogames

Acknowledgment

First and foremost, praises and thanks to Allah, the almighty, for his showers of blessings throughout our whole journey of finding a real beneficial idea, and working as a great team on the project.

We would like to express our deep and sincere gratitude to our project supervisor, Prof.Dr.Samir Gaber for providing invaluable guidance throughout this project. He has always supported, motivated, and lift our spirits to give our maximum effort for this project and for its noble purpose. It was a great privilege and honor to work and study under his guidance. We are extremely grateful for what he has offered us. We truly hope that our outcome can make him proud of us.

Besides our supervisor. We would like to thank the physical therapist in our team Dr.Hagar Abd-Elbasit Ismail for providing invaluable guidance with the physical therapy part of the project. She supported us, motivated us, and inspired us with so many ideas. She helped us understand so many aspects of physical therapy, how the treatment program should be, and what exercises can be performed by games. A lot couldn't be possible without her guidance, she truly made a lot of things easier for us.

Last but not least, we members of this team would like to thank each other for always supporting and motivating each other. Always being there for each other. Understanding and considerate with each other. Appreciating each and every effort by each member. And always learning from each other.

Table of Contents

Abstract	2
Acknowledgment.....	3
Table of Contents.....	4
Table of Figures	8
Chapter (1): Introduction	11
1.1 Introduction	12
1.2 Literature Review	12
1.3 Motivation	13
1.4 Problem statement	13
1.5 Project Diagram.....	14
1.6 Project phases	15
Chapter (2): Hand Detection	16
2.1 Computer vision – MediaPipe.....	17
2.1.1 What is MediaPipe	17
2.1.2 MediaPipe pipeline	17
2.1.3 Palm detection model	18
2.1.4 Hand landmark detection model	19
2.2 Deep learning – YOLO.....	20
2.2.1 Introduction to YOLO Algorithm	20
2.2.2 Introduction to object detection	20
2.2.3 What is YOLO?.....	20
2.2.4 Why the YOLO algorithm is important	21
2.2.5 How the YOLO algorithm works.....	21
2.2.6 Disadvantages of YOLO.....	24
2.2.7 YOLO Versions.....	24
2.3 Leap motion.....	26
2.3.1 What is Leap Motion	26
2.3.2 Leap Motion breakdown.....	27

Chapter (3): Web Application	29
3.1 Back-end	30
3.1.1 Clients	30
3.1.2 Server	31
3.1.3 Application Programming Interface (API)s	31
3.1.4 Payload validation	36
3.1.5 Database	38
3.1.6 Returning response	39
3.2 Front-end	40
3.2.1 Languages, framework, and tools	40
3.2.2 Setting up the Router	43
3.2.3 Views	44
Chapter (4): Game Development – Unity	51
4.1 Game Engines.....	52
4.1.1 Purpose of Game Engines.....	53
4.2 Unity Game Engine	54
4.2.1 Overview	54
4.2.2 Supported platforms	55
4.3 Developing Games for Physical Therapy	56
4.3.1 CURRENT TECHNOLOGIES IN REHABILITATION: WHERE DO VIDEO GAMES FIT IN?.....	56
4.3.2 WHAT CAN GAMES DO?	57
4.3.3 Our Aim	58
4.3.4 Feedback	58
4.3.5 Clear Goals and Mechanics	59
4.4 In-Game Hand Detection and Tracking	59
4.4.1 Mediapipe Hand Tracking	60
4.4.2 Leap Motion Hand Tracking.....	62
4.4.3 Games	63
4.5 Cubes Game	64

4.5.1	Target injuries	64
4.5.2	Level design	65
4.5.3	Scripting	67
4.6	Musical Fingers	71
4.6.1	Target injuries	71
4.6.2	Level design	72
4.6.3	Scripting	73
4.6.4	UI Design	74
4.6.5	Sound Design	78
4.6.6	In game sound Consistency	79
4.6.7	In game sound management	80
Chapter (5): Unity Interface with Other Environments.....		81
5.1	Media pipe Interface with Unity	82
5.1.1	Socket Programing	82
5.1.2	Leap Motion Interface with Unity.....	83
5.1.3	Integration between Unity and Server.....	84
Chapter (6): Design specifications		86
6.1	System requirements.....	87
6.1.1	Functional requirements	87
6.1.2	Non-functional requirements	88
6.1.3	Functional requirements specification	88
*	Stakeholders	88
6.1.4	Use cases Diagram:	90
6.1.5	System Sequence Diagrams.....	91
6.1.6	Class Diagram	92
6.2	Tools and technologies:	93
6.2.1	Tools:	93
	Technologies	93
Chapter (7): Conclusion		94
7.1.	conclusion.....	95

7.2. Future Work.....	96
7.3 References	97

Table of Figures

Figure 1 Mediapipe logo	17
Figure 2 Mediapipe hand coordinates	19
Figure 3 Mediapipe hand detection examples	19
Figure 4 Gridding an image	22
Figure 5 Y-matrix of CNN	22
Figure 6 IOU example	23
Figure 7 YOLO versions	24
Figure 8 Precision accuracy	25
Figure 9 Predicted vs actual	25
Figure 10 Tiny YOLO	25
Figure 11 YOLOv3-320	25
Figure 12 Leap motion usage	26
Figure 13 Leap Motion from inside	27
Figure 14 What Leap Motion sees	28
Figure 15 Network overview	30
Figure 16 API process	31
Figure 17 Injury creation API	31
Figure 18 Injury POST request's url	32
Figure 19 Injury POST request's body	32
Figure 20 Injury request's response's body	33
Figure 21 Creating collection	33
Figure 22 Creating environment	34
Figure 23 Creating requests	34
Figure 24 Fetching data from response's body	34
Figure 25 Using environment variables	35
Figure 26 Run button	35
Figure 27 Response of creating a patient	36
Figure 28 Payload validation	37
Figure 29 Create patient code snippet	38
Figure 30 Find patient by name code snippet	38
Figure 31 Creating custom game DTO	39
Figure 32 Using icons in Vue	42
Figure 33 Router main file	43
Figure 34 Using router instance	43
Figure 35 Using router-view tag	44

Figure 36 Navigation bar	44
Figure 37 Abstract section	44
Figure 38 Services panel	45
Figure 39 Games section.....	45
Figure 40 Suggestions section.....	46
Figure 41 Posting suggestion to server	46
Figure 42 Footer	46
Figure 43 Home page.....	47
Figure 44 Login form.....	48
Figure 45 Login code snippet.....	48
Figure 46 Add patient form.....	49
Figure 47 Add patient code snippet.....	49
Figure 48 Get all patients code snippet	49
Figure 49 Displaying patients	50
Figure 50 Integrating gameplay into therapy can increase time spent in therapy by increasing the likelihood of starting a therapy session (point of engagement), amount of time engaged in therapy (engagement and disengagement), and chances of going back to therapy	57
Figure 51 A schematic representation of the process of developing gaming principles for rehabilitation.	58
Figure 52 Mapping function using C#.....	60
Figure 53 Detecting hand gestures using Mediapipe	61
Figure 54-Euclidean Distance Using python	61
Figure 55 Hand Gestures Flowchart.....	61
Figure 56 Leap XR Package	62
Figure 57 Shoulder subluxation rehabilitation exercises	64
Figure 58 Level 1 Unity	65
Figure 59 Level 2 Unity	66
Figure 60 Level 3 Unity	66
Figure 61 Level 4 Unity	67
Figure 62 Distance_Check function	68
Figure 63 Initialization of scoring data.....	69
Figure 64 Collision detection function	69
Figure 65 Time since the level is loaded	70
Figure 66 Catching the events of clicking pause and resume button.....	70
Figure 67 Calculating the pausing period of the game.....	70
Figure 68 Converting time format from seconds to hours format	71
Figure 69 Hand opposition exercises.....	71

Figure 70 Hand opposition exercises 2.....	72
Figure 71 Opposition 2D hand design	72
Figure 72 Musical Fingers Game on Unity	73
Figure 73 Generating random hand moves	73
Figure 74 Check hand move and calculate score	74
Figure 75 Start menu in Unity	74
Figure 76 Options menu in Unity	75
Figure 77 Cubes' game levels.....	75
Figure 78 Difficulty menu in Unity	76
Figure 79Difficulty menu in Unity 2.....	76
Figure 80 In-game menu	77
Figure 81 Pause menu in Unity	77
Figure 82 Level complete menu in Unity	78
Figure 83 Audio Manager	80
Figure 84 Sound Class Diagram.....	80
Figure 85 State diagram for server and client model	82
Figure 86 Python to Unity communication.....	82
Figure 87 Get Request code in unity	84
Figure 88 Post Request code in unity	85
Figure 89 Posting feedback data	85
Figure 90 Use case diagram	90
Figure 91 System sequence diagram	91
Figure 92 Class diagram	92

Chapter (1): Introduction

1.1 Introduction

According to the World Health Organization, a Stroke is defined as an accident to the brain with rapidly developing clinical signs of focal or global disturbance to cerebral function, with symptoms lasting 24 hours or longer, or leading to death. The better and meaningful term for a stroke is "brain attack", which is similar to "heart attack". Thanks to treatment advances, about 83 percent of Americans who suffer a stroke will survive but will be in need to rehabilitation which mainly involves physical therapy. The aim of physical therapy is to have the stroke patient relearn simple motor activities such as walking, sitting, standing, lying down, and the process of switching from one type of movement to another. Our project is about simulating these motor activities using video games therapy. Therefore, the patient can exercise at home and under the supervision of a doctor remotely. The project is a platform with a collection of games, each game is simulating some activities like grasping, standing, walking and moving different parts of the body. With the aid of computer vision, Artificial intelligence and deep learning we are able to develop games that simulates the activities mentioned above. Patients will only have to stand in front of the camera and follow the on screen assessments to complete the levels and do the exercises successfully. Each patient will have an online profile containing his/her progress through statistics and graphs. Also, the doctor will be able to check and follow the progress of his patients through a web app.

1.2 Literature Review

Researchers began looking into virtual reality systems as a form of rehabilitation in the early 2000s, and the research has expanded since. Now, common gaming consoles such as the Wii and Kinect allow researchers to use cheaper, more readily available systems in their labs, as well, opening up new possibilities for games and rehabilitation options. But that would require proper installation, configuration and hardware. Some studies (Keith PhD; Shirzad, Nicola PhD; Van der Loos, journal of neurologic physical therapy 2013) found that patients are not meeting the sufficient "dosage" of movements required to induce neuroplastic adaptations underlying behavioral improvement. With our rehabilitation platform, the patients will have the right dosage as they can do the exercises from home. Also, it will not need extensive installation or hardware as we have minimized the needed hardware to computer and camera only.

1.3 Motivation

After knowing how many people around the world drop out and don't complete their physical therapy session and knowing that stroke is the most common cause of disability among adults. We were motivated to make it psychologically more comfortable for patients as playing a video game for therapy can be a way of distraction from the insecurity of being a disable person, and also a way of distraction from the physical pain. The patient will feel that he is a normal person who is able to play games like any other, and a good time during therap. The patient will be much more engaged and motivated throughout the rehabilitation journey.

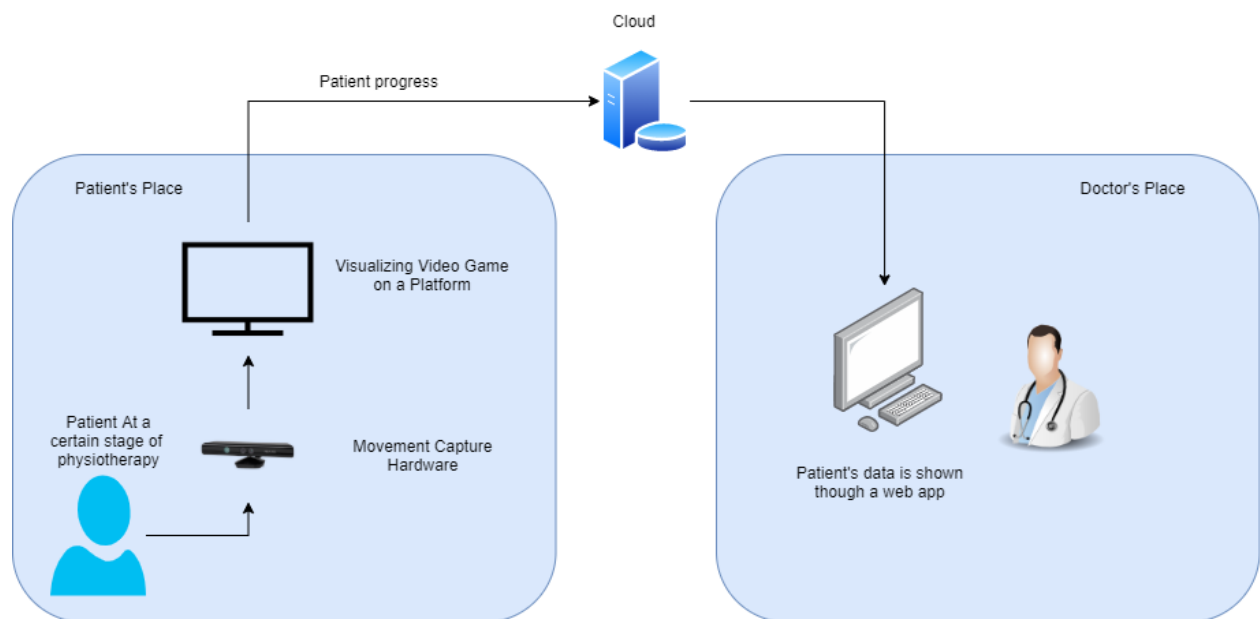
1.4 Problem statement

Stroke is the most common cause of long-term disability among adults in industrialized nations. About 80% of people who survive a stroke experience motor disability, such as hemiparesis (a partial paralysis of one side of the body), that severely constrain their lives. Especially the upper limb (hand and arm) which can remain weak in up to 66% of stroke survivors. Researchers suggest that hundreds of daily repetitions of therapeutic motions can help in recovering motor control following a stroke. Yet, during typical outpatient therapy, clients and therapists meet weekly for an hour or less. During these meeting, clients typically perform only tens of motions. Research suggests that motion-based video games can provide a motivating context in which players will perform the number of repetitions necessary for recovery. Providing suitable feedback during gameplay. Adapt to Changing abilities.

In physical therapy the priority is to target disabilities that make patients can't help themselves with the very basic everyday tasks, for example grabbing and grasping a cub of water and drink it by themselves, or washing their faces, button a shirt or wearing clothes, brushing their hair, opening a door, and many more. So, it was clear that we needed to target upper limb functions, and specifically and ordering discerningly by vitality:

1. Hand opposition function: The followed protocol in physical therapy is to make the patient able to move his thumb then gradually by practicing more hand opposition the patient will be able to move all of his fingers which is building up for the next function.
2. Grasp function
3. Shoulder function
4. Elbow function

1.5 Project Diagram



1.6 Project phases

1. Planning, searching and having a clear vision on how can we perform physical therapy using video games with maximum efficiency and benefits for the patient and doctor, also searching for a suitable approach for hand detection.
2. Implementing the first game which is cubes game.
3. Using media pipe tool with the game.
4. Implementing second game which is musical fingers game.
5. Studying yolo deep learning tool.
6. Using leap motion in hand detection as a final solution.
7. Implementing the web application backend and frontend in parallel.
8. Integrating web application with unity.

Chapter (2): Hand Detection

Hand localization and detection is a very important step in this project, since the whole game depends on hands' position and gestures, and it highly affects many aspects of the project; thus we gave this subject a lot of efforts and tried many approaches to achieve a highly accurate, fast and reliable hand detection.

2.1 Computer vision – MediaPipe

2.1.1 What is MediaPipe

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, this method achieves real-time performance on a mobile phone, and even scales to multiple hands.



Figure 1 Mediapipe logo

2.1.2 MediaPipe pipeline

MediaPipe Hands utilizes a machine-learning pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand key points.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g., rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in this pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalise the hand.

2.1.3 Palm detection model

To detect initial hand locations, a single-shot detector (SSD) model optimized for mobile real-time is used. Since that detecting hands is a decidedly complex task, this model has to work across a variety of hand sizes with a large-scale span ($\sim 20\times$) relative to the image frame and be able to detect occluded and self-occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

This method addresses the above challenges using different strategies. First, a trained palm detector is used instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, MediaPipe minimizes the focal loss during training to support a large number of anchors resulting from the high scale variance.

With the above techniques, MediaPipe achieves an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

2.1.4 Hand landmark detection model

After the palm detection over the whole image the hand landmark model performs precise key point localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, which is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, the model is trained on manually annotated ~30K real-world images with 21 3D coordinates, as shown below. To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, a high-quality synthetic hand model is rendered over various backgrounds and mapped to the corresponding 3D coordinates.

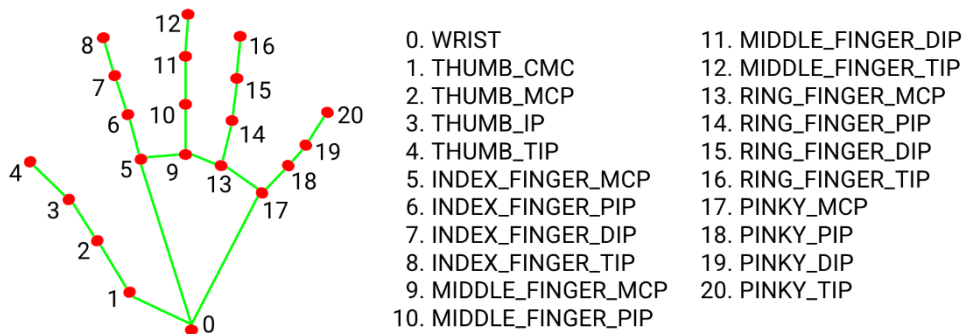


Figure 2 Mediapipe hand coordinates

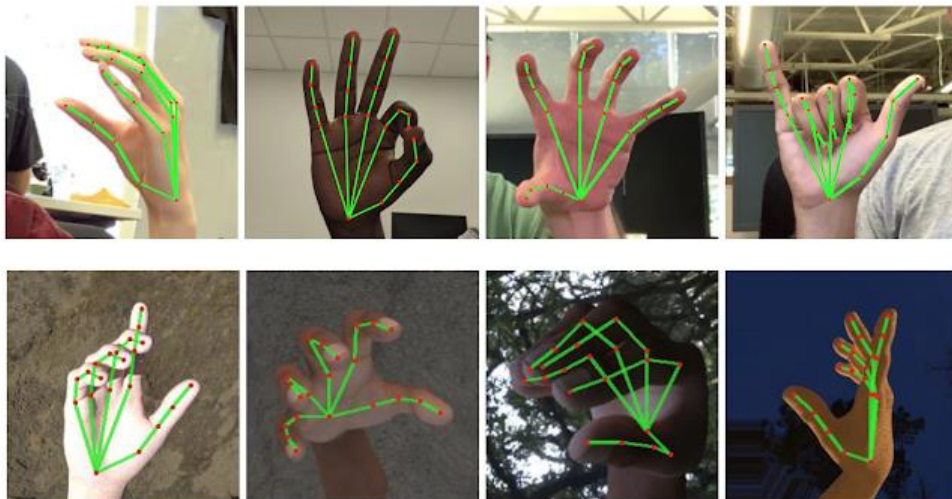


Figure 3 Mediapipe hand detection examples

2.2 Deep learning – YOLO

2.2.1 Introduction to YOLO Algorithm

YOLO is an algorithm that uses neural networks to provide real-time object detection. This algorithm is popular because of its speed and accuracy. It has been used in various applications to detect traffic signals, people, parking meters, and animals.

2.2.2 Introduction to object detection

Object detection consists of various approaches such as fast R-CNN, Retina-Net, and Single-Shot MultiBox Detector (SSD). Although these approaches have solved the challenges of data limitation and modeling in object detection, they are not able to detect objects in a single algorithm run. YOLO algorithm has gained popularity because of its superior performance over the aforementioned object detection techniques.

2.2.3 What is YOLO?

YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

2.2.4 Why the YOLO algorithm is important

- Speed: This algorithm improves the speed of detection because it can predict objects in real-time.
- High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.
- Learning capabilities: The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

2.2.5 How the YOLO algorithm works

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

– Residual blocks

First, the image is divided into various grids. Each grid has a dimension of $S \times S$

– Bounding box regression

Every bounding box in the image consists of the following attributes:

- Width (B_w)
- Height (B_h)
- Bounding box center (B_x, B_y)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter. ($C_1 - C_n$).
- Object detected (P_c).

After image passes through the CNN algorithm each pixel will have a Y matrix which represents information about the object that is represented in that pixel.

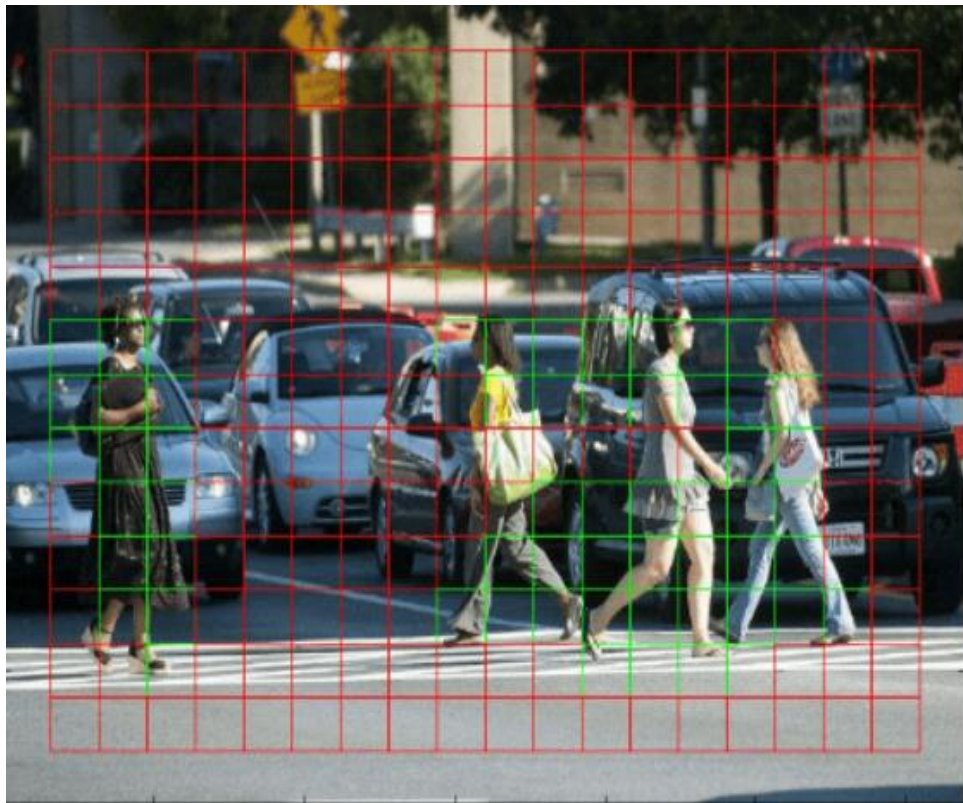


Figure 4 Gridding an image

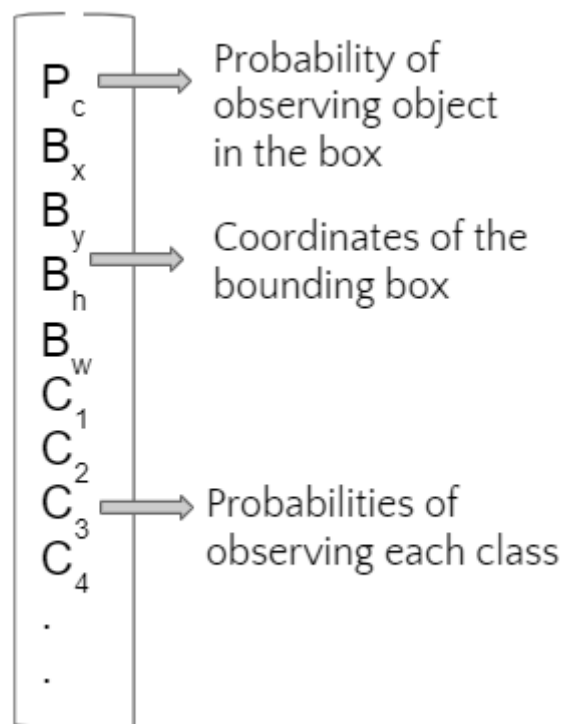


Figure 5 Y-matrix of CNN

Pc: 1

Only if the object in the pixel is detected as one of the classes from.

Pc: 0

Only if there is no object in the pixel that could be detected.

- Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

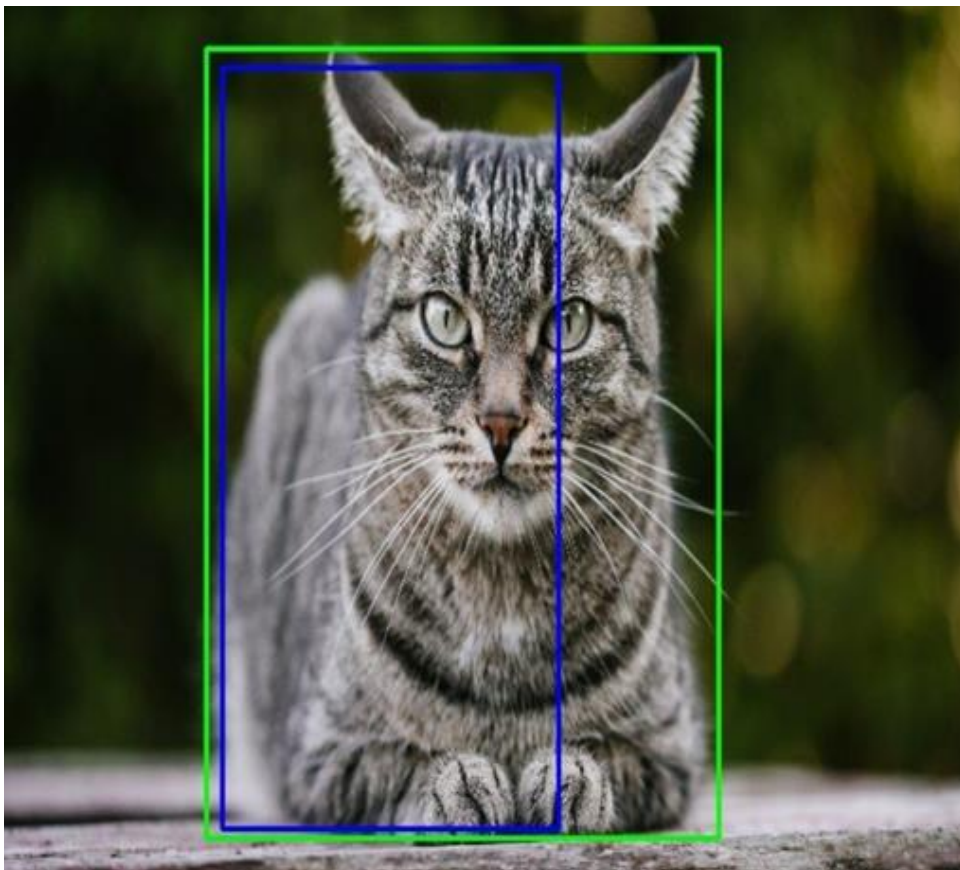


Figure 6 IOU example

In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

2.2.6 Disadvantages of YOLO

The most important point that any deep learning Algorithm must be given a High GPU to run on it as it needs high processing power.

So, to train your own module and your own data on a deep learning algorithm it will take a long time, for learning and also for detecting.

2.2.7 YOLO Versions

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Figure 7 YOLO versions

– mAP (mean Average Precision)

Is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, and CNN.

Precision: measures how accurate is your predictions. I.e. the percentage of your predictions are correct.

$$Precision = \frac{TP}{TP + FP}$$

TP = True positive

TN = True negative

FP = False positive

Figure 8 Precision accuracy

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 9 Predicted vs actual

– FPS

FPS (Frame per Second) defines how fast your object detection model process your video and generate the desired output.

Which version of YOLO did we use?

Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
-----------	---------------	----------	------	---------	-----	-----	---------

Figure 10 Tiny YOLO

YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
------------	---------------	----------	------	----------	----	-----	---------

Figure 11 YOLOv3-320

Conclusion after using both versions

As we mentioned before, deep-Learning Algorithms must run on GPUs to give its full power of detection, which is not available in most of our computers, so we decided to run those Algorithms on our CPU to give a non-real-life application, as our application depends on live Stream which must be in real-life time, this technique for hand detection was to suitable.

2.3 Leap motion

2.3.1 What is Leap Motion

The Leap Motion Controller developed by Ultraleap is an optical hand tracking module that captures the movement of users' hands and fingers so they can interact naturally with digital content. Small, fast, and accurate, the Leap Motion Controller can be used for productivity applications with Windows computers, integrated into enterprise-grade hardware solutions or displays, or attached to virtual/augmented reality headsets for Augmented, Virtual or Mixed Reality prototyping, research, and development.

The Leap Motion Controller is capable of tracking hands within a 3D zone that extends up to 60cm or more, extending from the device in a 140x120° typical field of view. Leap Motion's software is able to discern 27 distinct hand elements, including bones and joints, and track them even when they are obscured by other parts of the hand.

Leap Motion has many applications in many fields, such as entertainment, robotics, personnel training, and simulators, but most importantly it is used in therapy and healthcare, where it can be used in, stroke rehabilitation, lazy eye treatment, anatomic visualizations and many more.

Leap Motion is built to be integrated easily with customer applications and can be retro-fitted to existing concepts or hardware, plugins are built for Unity and Unreal game engines to enable developers to integrate hand-tracking with their established workflow.



Figure 12 Leap motion usage

2.3.2 Leap Motion breakdown

The Lap Motion is 6.2mm thick, 25mm wide and 75mm long. At one end you'll find the data port- a USB 3.0 micro-B connector. While it can be used as a 3.0 device.

There are three infrared LEDs inside, and that they're positioned to provide a nice, wide coverage area. Leap Motion has an important feature, dynamic LED driving- as you move your hand closer to the sensor, the device will automatically dim the LEDs to prevent the imagers from becoming saturated and to keep the data quality high.

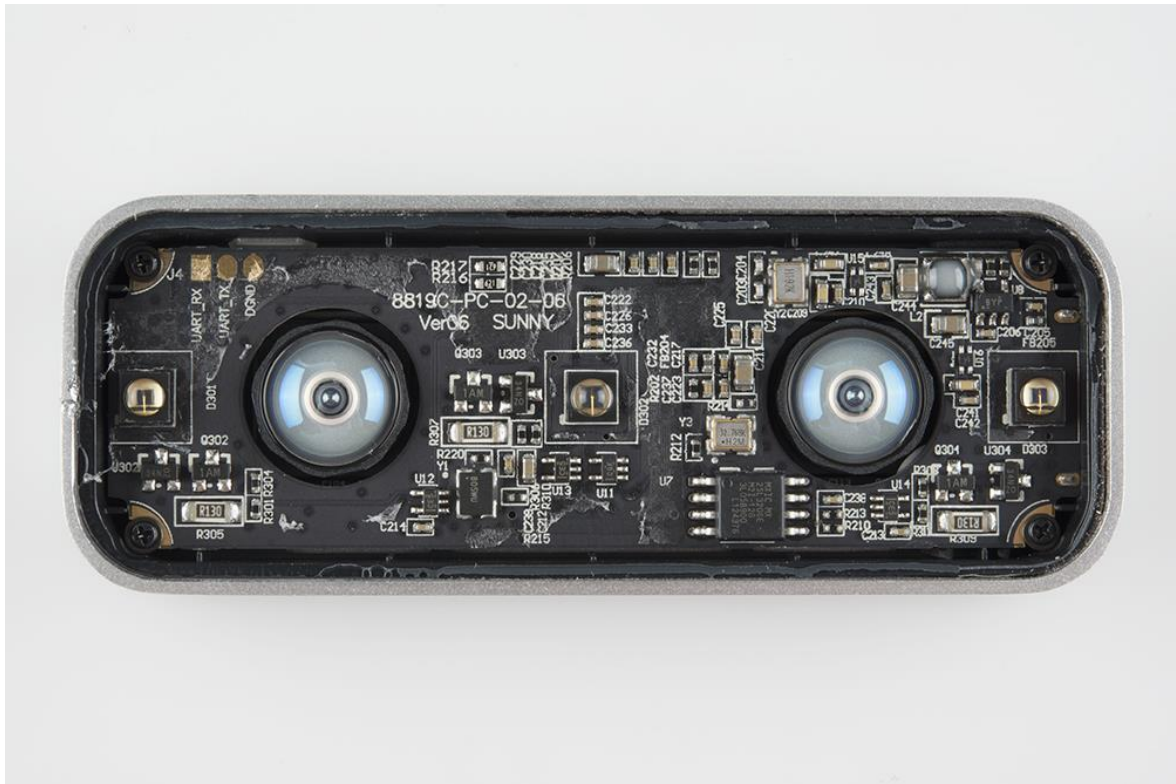


Figure 13 Leap Motion from inside

- **Infra-Red leds**

There are three high-power IR leds, one on each side and one on the middle of the board, they are used to emit high intensity infra-red light that eliminates any objects from the background and only focuses on the objects near the camera.



Figure 14 What Leap Motion sees

As it can be seen from the above image, that the objects near the camera (hands) have high values (white), while other objects that are far away from the camera are nearly black. From this we could conclude one of the main reasons that gives Leap Motion low latency and high detection accuracy, as it only focuses on a portion of the image and not the whole image.

- **Cameras**

Stereo camera is used in Leap Motion, this structure allows the device to simulate human binocular vision thus, giving the ability to determine depth, also giving Leap Motion the ability to determine the order of objects in terms of z-depth.

Chapter (3): Web Application

3.1 Back-end

In the back-end side of the project we used Nodejs to program our server in addition to Express package to create/expose the needed by client. We used MySQL database with Object-Relational Mapping (ORM) technique using Sequelize package.

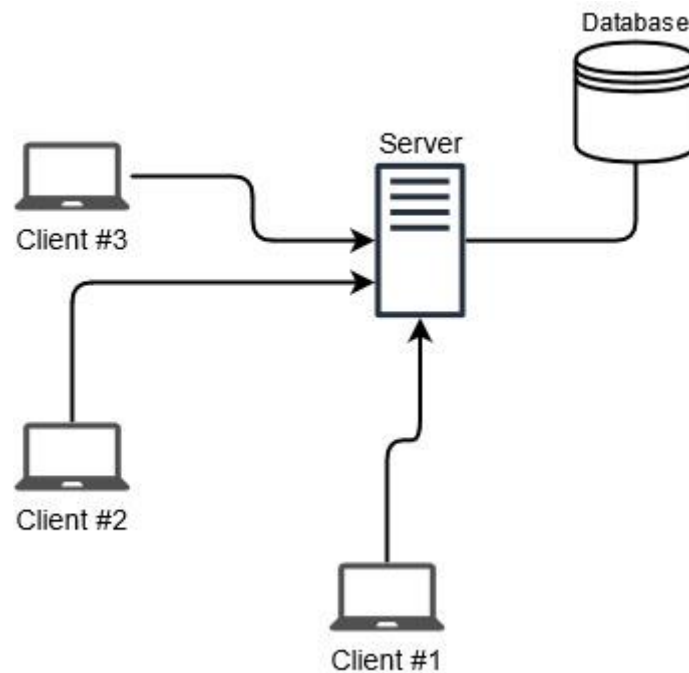


Figure 15 Network overview

The above image shows an abstract representation of the system as a whole. Clients communicate with the server through exposed APIs, clients could be a web application requesting/sending information or a game sending patient's progress in a certain level. The server then validates these requests and payloads then accesses the database to either write or read information, then it reformats data to the suitable form expected by the client, then it sends it to the client.

3.1.1 Clients

In our project, we expect the client to either be a web application or a game running on patient's laptop. In case the client is a web application, it can send login credentials, request webpages/images/assets, etc... If the client is a game running on the patient's laptop, then it can send games' progress to the server or request all games to display it to the patient.

3.1.2 Server

We used Express package to create our server and Application Programming Interface (API)s. Express is a package that allows you to easily create, configure and run a server, it also gives you the ability to use middleware which can become really useful when validating payloads or filtering requests. Middleware is anything you put in the middle of one layer of the software and another, so for example we used a payload validator as a middleware between receiving a request and accessing database.

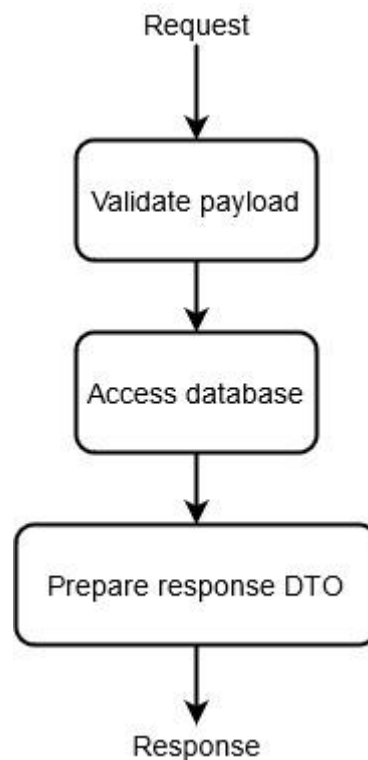


Figure 16 API process

3.1.3 Application Programming Interface (API)s

In our server we exposed many APIs for the clients to consume, we used Request For Comments (RFC) standards in creating the APIs. Each entity (patient, doctor, game, ...) has CRUD endpoints.

```
158 app.post('/injury', async (req, res)=>{  
159 |   res.status(201).send(await Injury.create(req.body));  
160 | })
```

Figure 17 Injury creation API

Here in line 158, we expose an endpoint “/injury”, this endpoint uses a POST method. In line 159 a couple of actions are taken, firstly we get the request body (req.body), then we create a new injury of that body and wait until the injury is created in the database successfully, lastly, we set the response code to 201 to indicate that the injury is created then we send the return of the injury creation to the client as the response.

– Testing APIs

We used Postman in testing our APIs, Postman is a collaboration platform for API development. Postman's features simplify each step of building an API, so it is very suitable for our project.

For example, to create an injury, we firstly select the method type as POST and write the endpoint as localhost: 3000/injury, as shown below.

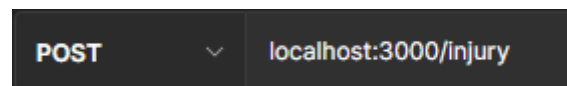
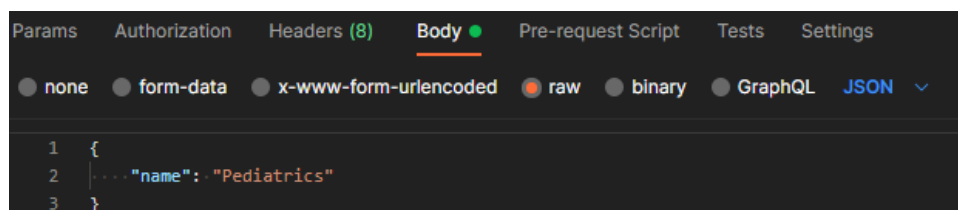


Figure 18 Injury POST request's url

Then we go into Body tab, select raw and JSON from the dropdown menu on the right. In this step we tell postman that the method's body is a json object. After that we write down the expected body to be received by this endpoint, in this case it only expects a name for the injury.



If we send the request, we'll get this response. The response code is 201 Created which means that the record was created and stored in the database without any problems, and the response body has the injury model/record saved in the database with id and name.

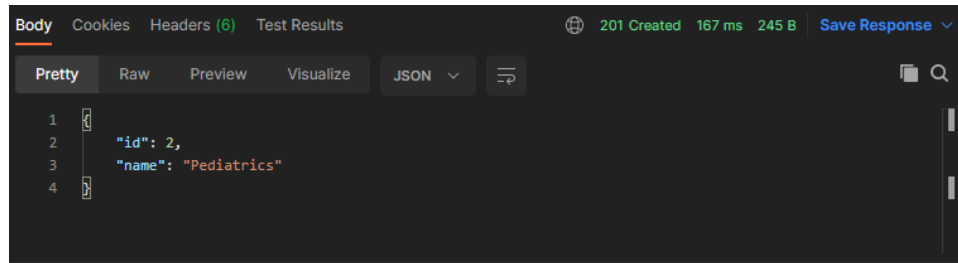


Figure 20 Injury request's response's body

– Chaining APIs

If we want to create patient A with injury X and the patient is assigned to doctor B, we'll have to do several steps, first we need to create injury X and get its id then we'll create doctor B and get its id as well, lastly, we will send a POST request with injury X's id and doctor B's id and patient A's information to create a record for patient A.

As you can see this is exhausting to do several times in a row when we are testing our APIs since that injury X's id and doctor B's id will change every time, we send a POST request and we have to update patient A's request's body with these ids, for that we used the technique of chaining method calls and using the response as an environment variable (global variable).

First, we create a collection to hold our requests.

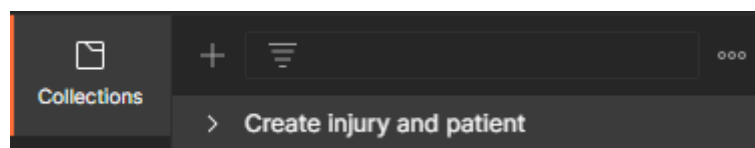


Figure 21 Creating collection

Then, we create an environment to hold our variables. Make sure to set it as active by clicking on the check sign.

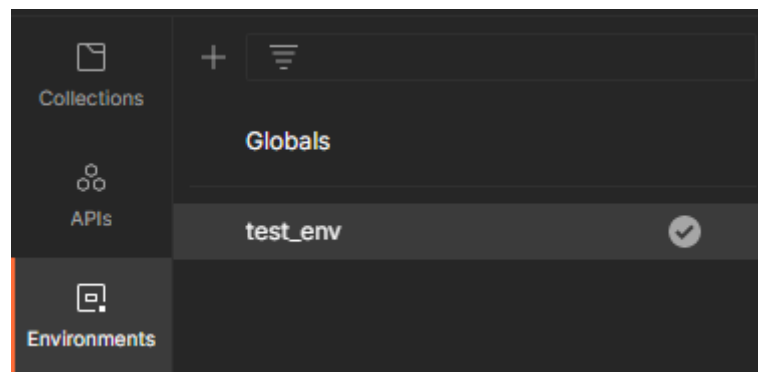


Figure 22 Creating environment

Next, we created three POST request, one for creating a doctor, one for creating an injury and the last one for creating a patient.

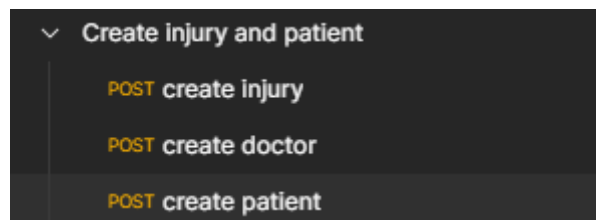


Figure 23 Creating requests

After we send the POST request to create an injury, the response is a JSON object that has the injury's id, we need to save that id in a variable to use it later. Inside "create injury" request, in the Tests tab we can write these two lines of code to get the id from the response JSON object and save it in a variable named "injury_id". The same is done while creating a doctor, we save the doctor's id in a variable called "doctor_id".

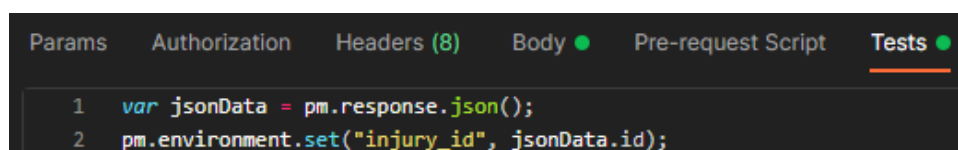


Figure 24 Fetching data from response's body

Now we have the doctor's and injury's ids, so we can create the patient, in the "create patient" request, write its body as shown below. Here we want to use the environment variables that we declared earlier, to do that write the variable name between doubled-curly brackets.

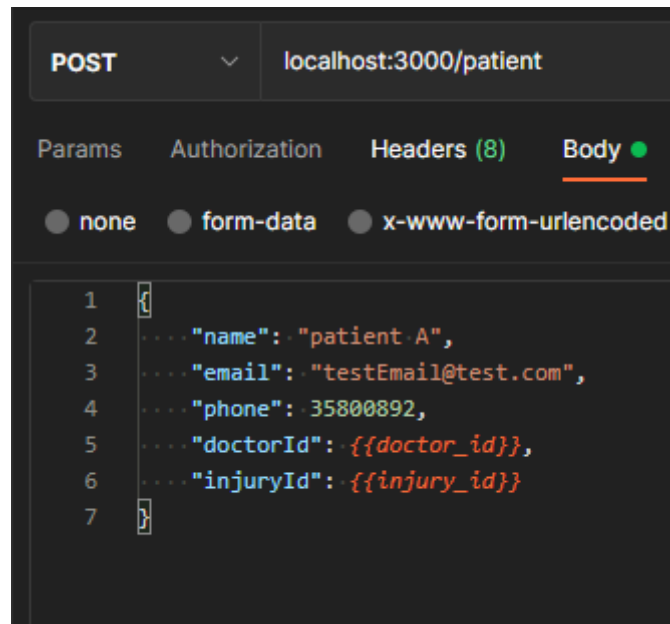


Figure 25 Using environment variables

Finally, to call the requests we have two options, either we send them manually one by one, or make a runner for this collection of requests, for that we will select the collection from the left panel then click on the Run button on the right, this will run the requests in order.

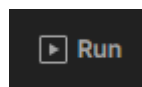


Figure 26 Run button

After running the collection, we can observe the response body of the “create patient” request. As we can see, “doctorId” is set to 8, and the patient has an injury with id of 9.

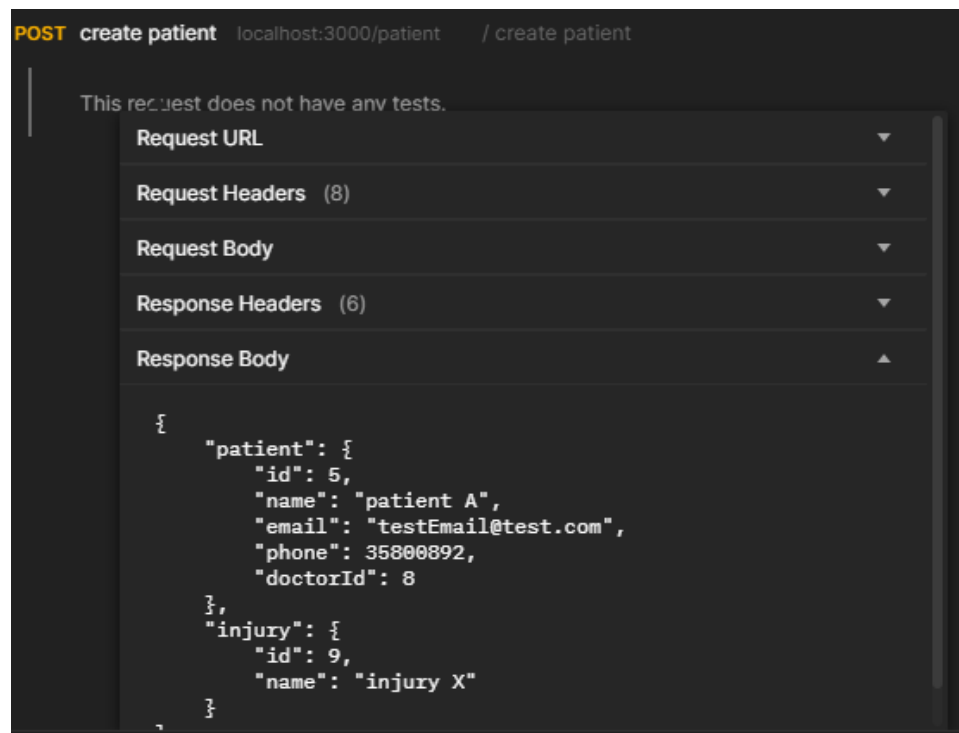


Figure 27 Response of creating a patient

We have successfully created an injury, doctor, and patient without using the traditional way of sending requests individually.

3.1.4 Payload validation

Before accessing the database and take any action, we firstly validate the payloads to make sure that everything required by the database is sent from the client. We used express validator package to validate payloads, it is used as a middleware as shown below, inside “checkSchema” method, we identify the shape of the received payload. In the following example, we specify that “name” attribute must be a string, also we made a custom error message, the same goes for the rest of the attributes.

In line 101, we call the built-in “validationResult” method and pass it the request object, this method returns errors that were created after we checked the schema of the request’s body. In line 102 and 103, we check if there are any errors, if yes, then send code 400 Bad Request to the client, if no errors are found then send code 201 Created to the client.

```
76 app.post('/patient',
77   checkSchema({
78     name: {
79       isString: true,
80       errorMessage: "name is invalid",
81     },
82     email: {
83       isEmail: true,
84       errorMessage: "email is invalid",
85     },
86     phone: {
87       isNumeric: true,
88       errorMessage: "phone number is invalid",
89     },
90     doctorId: [
91       isNumeric: true,
92       errorMessage: "doctor id is invalid",
93     ],
94     injuryId: {
95       optional: true,
96       isNumeric: true,
97       errorMessage: "injury id is invalid",
98     },
99   }),
100   (req, res) => {
101     const errors = validationResult(req);
102     if (!errors.isEmpty()) {
103       return res.status(400).json({ errors: errors.array() });
104     }
105     res.status(201).send();
106   })
```

Figure 28 Payload validation

3.1.5 Database

Having a database is essential in any project that requires saving large data, for that we chose MySQL as our database. We worked with Object-Relational Mapping (ORM) to interact with the database and entities, ORM is a technique used to interact with the database in an Object-Oriented paradigm, so instead of the traditional way of writing native queries in the code to select/insert/modify records from the database, we can use ORM to interact with classes and objects and still manipulate our database efficiently. ORM is basically a library made to facilitate the interaction with databases, the library translates methods to queries, for example, if we write `Doctor.findAll()` it will be translated into a “SELECT * FROM ...” implicitly.

The code snippet below will create a new patient with a name, email, phone and doctorId, then it waits for the database response then it will save that response in a constant for later use.

```
const patient = await this.create({
  name    : patientI.name ,
  email   : patientI.email,
  phone   : patientI.phone,
  doctorId: patientI.doctorId
});
```

Figure 29 Create patient code snippet

For finding/selecting a certain patient record we can write something like the following

```
const patient = await Patient.findOne({
  where: {
    name: 'Patient X'
  }
})
```

Figure 30 Find patient by name code snippet

3.1.6 Returning response

When a client requests an entity (a game for example), we do not send all the attributes of the game to the client (name, id, url, ...), instead we send certain attributes that the client needs, id and name in this case. Also, we would like to send levels' ids only, in other words, we ignored other attributes like difficulty, maxScore, ...

As shown below, we select all the games, each game should have an id and name only, each game will include the ids of its levels.

```
124     var gamesDTO = await Game.findAll({
125         include: {
126             model: Level,
127             attributes: ['id']
128         },
129         attributes: ['id', 'name']
130     })
```

Figure 31 Creating custom game DTO

3.2 Front-end

3.2.1 Languages, framework, and tools

HTML

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and last major HTML version that is a World Wide Web Consortium (W3C) recommendation. The current specification is known as the HTML Living Standard. It is maintained by the Web Hypertext Application Technology Working Group (WHATWG), a consortium of the major browser vendors (Apple, Google, Mozilla, and Microsoft).

CSS

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

JavaScript

A lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g., functional programming) styles.

Vuejs

A progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries (opens new window).

Installation and project setup

Vue.js is built by design to be incrementally adoptable. This means that it can be integrated into a project multiple ways depending on the requirements.

There are four primary ways of adding Vue.js to a project:

1. Import it as a CDN package on the page
2. Download the JavaScript files and host them yourself
3. Install it using npm
4. Use the official CLI to scaffold a project, which provides batteries-included build setups for a modern frontend workflow (e.g., hot-reload, lint-on-save, and much more)

Using npm method:

Type in the cmd the following command to install the latest version of vue:

```
$ npm install vue@next
```

To create a new project choose the project directory and navigate to it in the cmd, then run this command:

```
$ vue create <project name>
```

To run the local server run this command:

```
$ npm run serve
```

Font awesome

Is a font and icon toolkit based on CSS and Less. As of 2020, Font Awesome was used by 38% of sites that use third-party font scripts, placing Font Awesome in second place after Google Fonts.

Installation and setup

You can install it using npm by running those lines:

Install the core package and icon content.

```
$ npm i --save @fortawesome/fontawesome-svg-core
```

```
$ npm i --save @fortawesome/free-solid-svg-icons
```

```
$ npm i --save @fortawesome/free-brands-svg-icons
```

Using Vue 3.x

```
$ npm i --save @fortawesome/vue-fontawesome@prerelease
```

Then you will have to import those packages in main.js file and define fontawsome component:

```
import {FontAwesomeIcon} from '@fortawesome/vue-fontawesome';
import {library} from '@fortawesome/fontawesome-svg-core';
import {fas} from '@fortawesome/free-solid-svg-icons';
import {fab} from '@fortawesome/free-brands-svg-icons';

library.add(fas);
library.add(fab);

createApp(App)
  .component('fa', FontAwesomeIcon)
  .mount("#app");
```

Figure 32 Using icons in Vue

3.2.2 Setting up the Router

1. Install the vue 3 router from command line:

```
$ npm i vue-router@next
```

2. Add a routing directory & configuration file (/src/router/index.js)

```
import { createWebHistory, createRouter } from "vue-router";
import Home from "@views/Home.vue";

const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
  },
]

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

Figure 33 Router main file

3 Importing our Routes & using Vue Router:

Next, we'll need to edit our `main.js` file to use our router in our application.

/src/main.js

```
import router from './router';

createApp(App)
  .use(router)
  .mount("#app");
```

Figure 34 Using router instance

Using `<router-view/>` and `<router-link>` App.vue:

```
<template>
  <div id="app">
    <router-view />
  </div>
</template>
```

Figure 35 Using router-view tag

3.2.3 Views

– Home

Navbar: Contain a logo and four navigation items that is when clicked scrolls down to the section of the page which it refers to

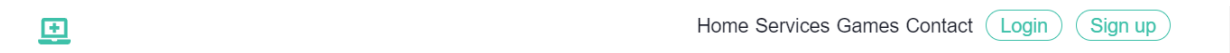


Figure 36 Navigation bar

Abstract: A section that contain a brief description about the project

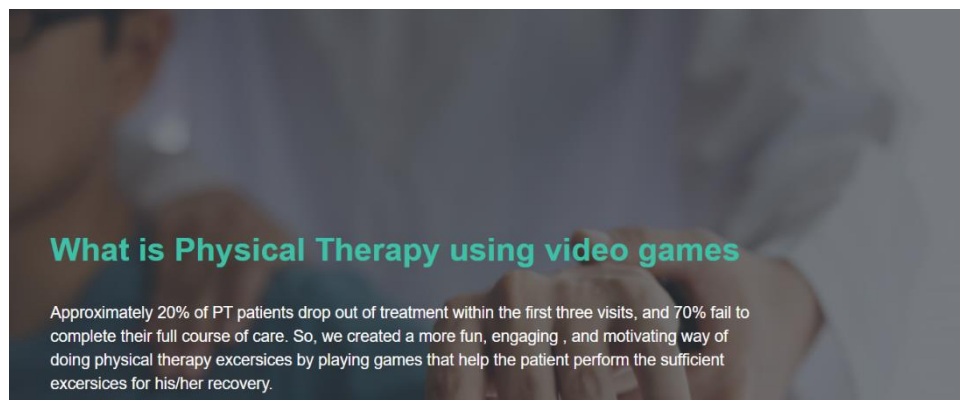


Figure 37 Abstract section

Services: This section contains a list brief description about the different services the project can provide

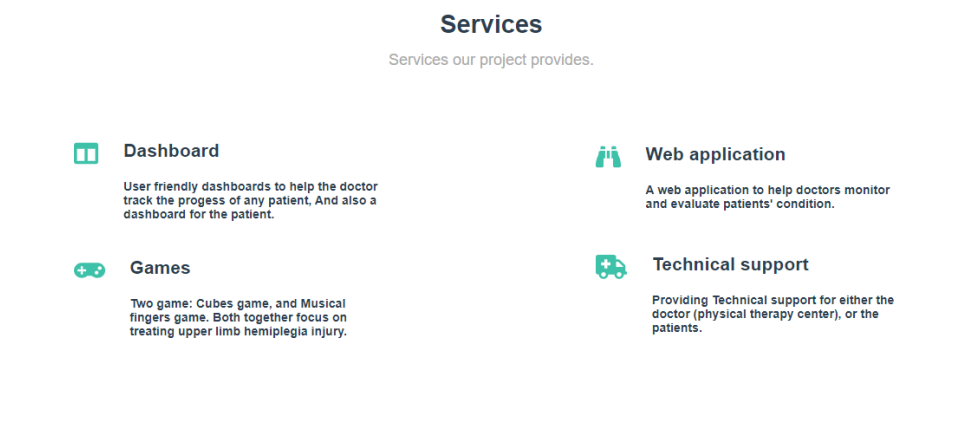


Figure 38 Services panel

Games: Contain a brief description about our games and two clickable cards that direct to a separate page for each game containing more information about the game, demonstrating video, and a download link.

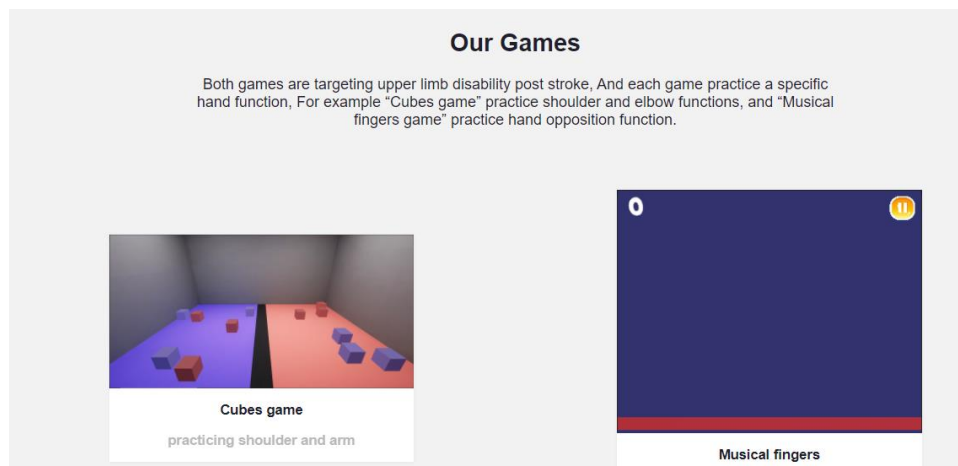
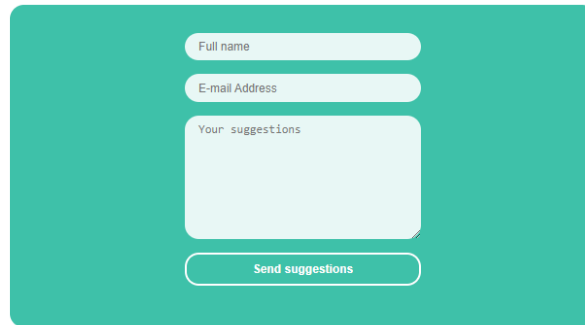


Figure 39 Games section

Suggestion: If anyone would like to send any suggestion to help improve the project more

Suggestions ?

Feel free to leave any suggestion of any kind that you believe is gonna help us improve any thing in our project.



Full name

E-mail Address

Your suggestions

Send suggestions

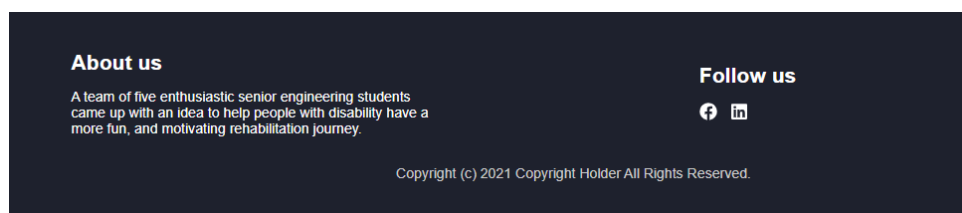
Figure 40 Suggestions section

Post request communicating with the API:

```
async suggest() {  
  const resp = await axios.post('http://f9b588909b24.ngrok.io/suggestion', this.suggestForm);  
  console.log(resp);  
},
```

Figure 41 Posting suggestion to server

Footer: Containing a brief about us description and social media links



About us

A team of five enthusiastic senior engineering students came up with an idea to help people with disability have a more fun, and motivating rehabilitation journey.

Follow us

[f](#) [in](#)

Copyright (c) 2021 Copyright Holder All Rights Reserved.

Figure 42 Footer

Home page as a whole

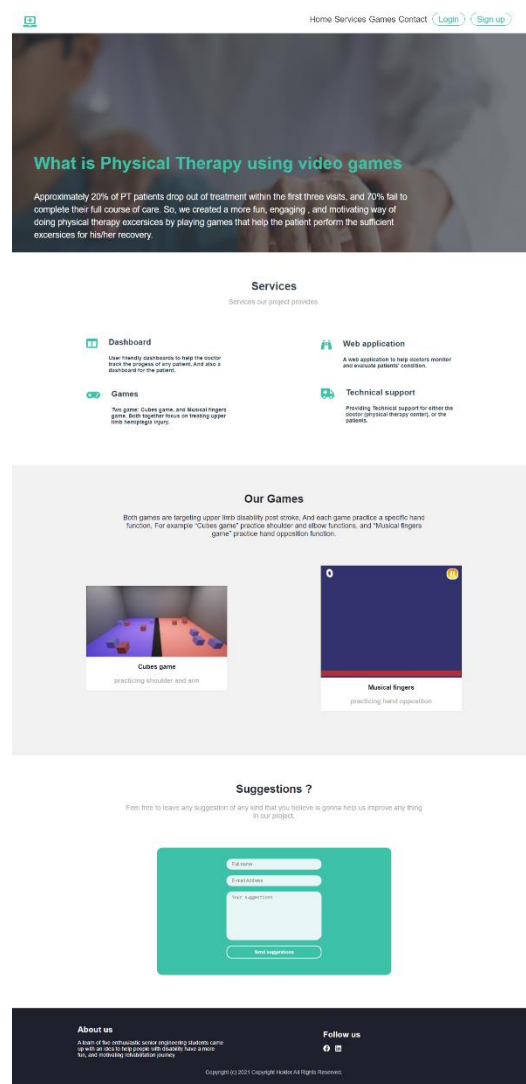
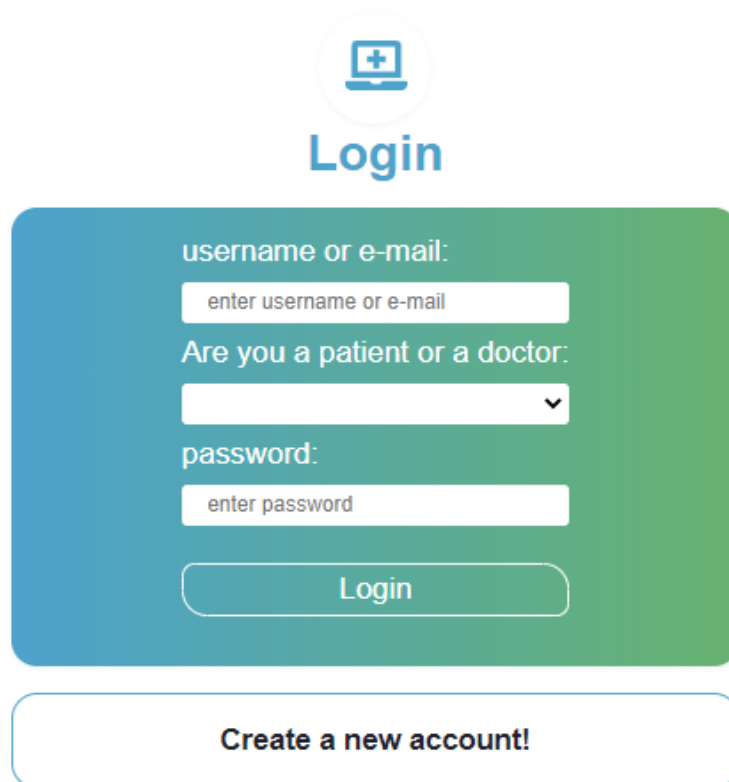


Figure 43 Home page

Login

The user enters his email or username, next chooses whether he/she is a doctor or a patient, because based on that choice he will be directed to Doctor dashboard or patient dashboard, lastly enter his/her password. If the user doesn't have an account, he/she can create a new one by clicking on "create a new account down below" which directs to sign up page.

The image shows a login form with a blue and green gradient background. At the top, there is a circular icon containing a blue laptop with a white plus sign. Below the icon, the word "Login" is written in a large, bold, blue font. The form itself is a rounded rectangle with a blue-to-green gradient. It contains the following elements: a label "username or e-mail:" followed by a white input field with the placeholder text "enter username or e-mail"; a label "Are you a patient or a doctor:" followed by a white dropdown menu with a downward arrow; a label "password:" followed by a white input field with the placeholder text "enter password"; a rounded "Login" button; and a separate rounded button at the bottom labeled "Create a new account!".

username or e-mail:

enter username or e-mail

Are you a patient or a doctor:

password:

enter password

Login

Create a new account!

Figure 44 Login form

Login get request method

First, check what the user chose from the select box, if doctor send get request to API with username of the user, if the user exists then redirect to doctor dashboard page, else if patient send get request to API with username of the user, if the user exists then redirect to patient dashboard page.

```
methods: {
  async login() {
    console.log(this.loginForm.select);
    if (this.loginForm.select === 'doctor') {
      await axios.get(`http://f9b588909b24.ngrok.io/doctor/${this.loginForm.username}`)
        .then(() => {
          this.$router.push({name: 'DoctorDashboard'});
        });
    }
    else if (this.loginForm.select === 'patient') {
      await axios.get(`http://f9b588909b24.ngrok.io/patient/${this.loginForm.username}`)
        .then(() => {
          this.$router.push({name: 'PatientDashboard'});
        });
    }
  },
}
```

Figure 45 Login code snippet

Doctor Dashboard

The doctor can use his dashboard to:

- Add a patient to the system
- Browse through all patients he supervises and monitor their progress

Physical therapy using video games

Welcome Dr.

Home

Add a patient

Show all patients

Evaluate a patient

patient's name:

patient's e-mail:

patient's password:

patient's phone number:

Choose gender:

supervising doctor ID:

type of injury:

Figure 46 Add patient form

Add a patient post request

```
methods: {
  async addPatient() {
    console.log(typeof this.patientForm);
    const jsonForm = JSON.stringify(this.patientForm);
    console.log(typeof jsonForm);
    const resp = await axios.post('http://f9b588909b24.ngrok.io/patient', this.patientForm);
    console.log(resp);
  },
}
```

Figure 47 Add patient code snippet

Show all patients get request

```
methods: {
  async getPatients() {
    const resp = await axios.get('http://f9b588909b24.ngrok.io/allpatients');
    console.log(resp);
    this.patients = resp.data;
    this.injuries = resp.data.injuries;
  },
}
```

Figure 48 Get all patients code snippet

Patient Dashboard:

The doctor can use his dashboard to:

- View his/her profile info
- View his/her progress in each game



Figure 49 Displaying patients

Chapter (4): Game Development – Unity

Video game development is the process of developing a video game. The effort is undertaken by a developer, ranging from a single person to an international team dispersed across the globe. Development of traditional commercial PC and console games is normally funded by a publisher, and can take several years to reach completion. Indie games usually take less time and money and can be produced by individuals and smaller developers. The independent game industry has been on the rise, facilitated by the growth of accessible game development software such as Unity platform and Unreal Engine. The first video games, developed in the 1960s, were not usually commercialized. They required mainframe computers to run and were not available to the general public. Commercial game development began in the '70s with the advent of first-generation video game consoles and early home computers like the Apple I. At that time, owing to low costs and low capabilities of computers, a lone programmer could develop a full and complete game. However, in the late '80s and '90s, ever-increasing computer processing power and heightened expectations from gamers made it difficult for a single person to produce a mainstream console or PC game. The average cost of producing a triple-A video game slowly rose, from US\$1–4 million in 2000, to over \$5 million in 2006, then to over \$20 million by 2010. Mainstream commercial PC and console games are generally developed in phases: first, in pre-production, pitches, prototypes, and game design documents are written; if the idea is approved and the developer receives funding, then full-scale development begins. The development of a complete game usually involves a team of 20–100 individuals with various responsibilities, including designers, artists, programmers, and testers.

4.1 Game Engines

A game engine is the core software component of a game. This terminology is similar to the term "software engine" used in the software industry. Game engine can also refer to the development software utilizing this core component, typically offering a suite of tools and features for developing games. Developers can use game engines to construct games for video game consoles and other types of computers. The core functionality typically provided by a game engine may include a rendering engine ("renderer") for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading,

localization support, scene graph, and video support for cinematics. Game engine implementers often economize on the process of game development by reusing/adapting, in large part, the same game engine to produce different games or to aid in porting games to multiple platforms.

4.1.1 Purpose of Game Engines

In many cases, game engines provide a suite of visual development tools in addition to reusable software components. These tools are generally provided in an integrated development environment to enable simplified, rapid development of games in a data-driven manner. Game-engine developers often attempt to preempt implementer needs by developing robust software suites which include many elements a game developer may need to build a game. Most game-engine suites provide facilities that ease development, such as graphics, sound, physics and artificial-intelligence (AI) functions. These game engines are sometimes called "middleware" because, as with the business sense of the term, they provide a flexible and reusable software platform which provides all the core functionality needed, right out of the box, to develop a game application while reducing costs, complexities, and time-to-market — all critical factors in the highly competitive video-game industry.

Like other types of middleware, game engines usually provide platform abstraction, allowing the same game to run on various platforms (including game consoles and personal computers) with few, if any, changes made to the game source-code. Often, programmers design game engines with a component-based architecture that allows specific systems in the engine to be replaced or extended with more specialized (and often more expensive) game-middleware components. Some game engines comprise a series of loosely connected game middleware components that can be selectively combined to create a custom engine, instead of the more common approach of extending or customizing a flexible integrated product. However achieved, extensibility remains a high priority for game engines due to the wide variety of uses for which they are applied. Despite the specificity of the name "game engine", end-users often repurpose game engines for other kinds of interactive applications with real-time graphical requirements - such as marketing demos, architectural visualizations, training simulations, and modeling environments.

As technology ages, the components of an engine may become outdated or insufficient for the requirements of a given project. Since the complexity of programming an entirely new engine may result in unwanted delays (or necessitate that a project restart from the beginning), an engine-development team may elect to update their existing engine with newer functionality or components.

4.2 Unity Game Engine

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. It is particularly popular for iOS and Android mobile game development. It is cited to be easy to use for beginner developers and is popular for Indie game development. The engine can be used to create three-dimensional (3D) and two-dimensional (2D) games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering and construction.

4.2.1 Overview

Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality. Prior to C# being the primary programming language used for the engine, it previously supported Boo, which was removed with the release of Unity 5, and a version of JavaScript called UnityScript, which was deprecated in August 2017, after the release of Unity 2017.1, in favor of C#.

Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

4.2.2 Supported platforms

Unity is a cross-platform engine. The Unity editor is supported on Windows, macOS, and the Linux platform, while the engine itself currently supports building games for more than 19 different platforms, including mobile, desktop, consoles, and virtual reality. Officially supported platforms as of Unity 2020 LTS are:

- Mobile platforms iOS, Android (Android TV), tvOS;
- Desktop platforms Windows (Universal Windows Platform), Mac, Linux;
- Web platform WebGL;
- Console platforms PlayStation (PS4,PS5), Xbox (Xbox One, Xbox Series X/S), Nintendo Switch, Stadia;
- Virtual/Extended reality platforms Oculus, PlayStation VR, Google's ARCore, Apple's ARKit, Windows Mixed Reality (HoloLens), Magic Leap, and via Unity XR SDK Steam VR, Google Cardboard.

As of 2018, Unity had been used to create approximately half of the mobile games on the market and 60 percent of augmented reality and virtual reality content, including approximately 90 percent on emerging augmented reality platforms, such as Microsoft HoloLens, and 90 percent of Samsung Gear VR content. Unity technology is the basis for most virtual reality and augmented reality experiences, and Fortune said Unity "dominates the virtual reality business". Unity Machine Learning Agents is open-source software whereby the Unity platform connects to machine learning programs, including Google's TensorFlow. Using trial and error in Unity Machine Learning Agents, virtual characters use reinforcement learning to build creative strategies in lifelike virtual landscapes. The software is used to develop robots and self-driving cars.

Unity formerly supported other platforms including its own Unity Web Player, a Web browser plugin. However, it was deprecated in favor of WebGL. Since version 5, Unity has been offering its WebGL bundle compiled to JavaScript using a 2-stage language translator (C# to C++ and finally to JavaScript).

Unity was the default software development kit (SDK) used for Nintendo's Wii U video game console, with a free copy included by Nintendo with each Wii U developer license. Unity Technologies called this bundling of a third-party SDK an "industry first".

4.3 Developing Games for Physical Therapy

There is a major concern in rehabilitation that patients are not meeting the sufficient “dosage” of movements required to induce neuroplastic adaptations underlying behavioral improvement. We argue that motion-controlled video games are an appealing avenue of research to augment therapy, because using rehabilitation-relevant movements in the context of an engaging and motivational game can potentially augment the dosage of therapy. In this article, we review experimental studies of video game training in basic science and rehabilitation. These data suggest that gameplay can induce desirable behavioral and physiological changes. We also review interdisciplinary evidence to suggest that specific game mechanics/design principles increase motivation and engagement, increasing the amount of time players (patients) are willing to spend in the game (as a therapy supplement). On this evidence, we present a framework for motivational properties of games to justify games as a therapeutic tool.

4.3.1 CURRENT TECHNOLOGIES IN REHABILITATION: WHERE DO VIDEO GAMES FIT IN?

In the first 6 months poststroke, there is a significant amount of spontaneous recovery, and a physical therapist takes advantage of this increased neuroplasticity immediately following stroke. Following this period, lengthy and rigorous physical therapy using various techniques can continue to improve motor function, with increased time in therapy the primary predictor for increased functional recovery.

In spite of the many techniques available to facilitate therapy, motivation and access are major obstacles to patients in achieving the necessary dosage of movements needed for recovery. Motivation, accessibility, and cost all contribute to attrition/abatement in the amount of time spent in therapy following an injury/illness, and current data suggest that the dosage of therapy, in terms of the number of repetitions of an exercise, is already far less than optimal for recovery. Thus, by adding a gaming supplement to therapy, patient adherence can be improved (shown abstractly in Figure 1). Motion controllers in games allow therapy-relevant movements to be practiced in the context of a game, and many of the benefits of virtual environments and augmented feedback can be realized in video game play. Borrowing on a model of usability from human computer interaction, we posit that the reinforcing factors of gameplay could increase the likelihood of initiating a gameplay session (the point of engagement), augmenting

engagement, delaying disengagement, and increasing the likelihood of reengagement (see Figure 50). Increasing patient engagement through gameplay has the potential to increase the dosage of therapy-relevant movements by integrating these movements into an interactive environment.

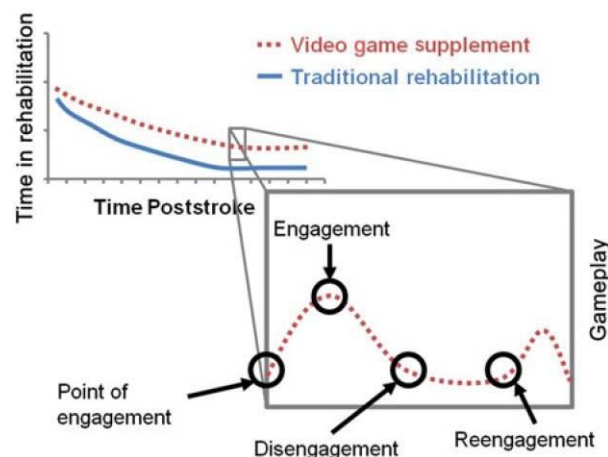


Figure 50 Integrating gameplay into therapy can increase time spent in therapy by increasing the likelihood of starting a therapy session (point of engagement), amount of time engaged in therapy (engagement and disengagement), and chances of going back to therapy

4.3.2 WHAT CAN GAMES DO?

Playing video games has been shown to have many positive behavioral and physiological effects. In rehabilitation settings, video games have been shown to have positive impacts on cognitive performance, motor performance, and affect. In quasi-experimental studies comparing video game experts with video game novices, experts show improved attentional capacity, expanded useful field of vision, and improved temporal resolution of attention relative to novices. Surgeons who played video games made 37% fewer errors, were 27% faster with laparoscopic drills, and were 33% better at suturing tasks than nongamer surgeons. Although supportive of video game-training benefits, the quasi-experimental nature of these studies prevents any causal judgments. Thus, in this Special Interest article, we focus exclusively on experimental studies of video game training (by searching electronic databases and the bibliographies of relevant research). This Special Interest article is inclusive in the types of games used, duration of the intervention, and sample population studied. Positive results and null findings found in published studies are presented hereafter. While we have attempted to be comprehensive in our review of the evidence, the interpretations are qualitative rather than quantitative. Considerable research still remains to be done in this area before the effectiveness and efficacy of video games in rehabilitation can be conclusively determined.

4.3.3 Our Aim

We are trying to maximize the likelihood that our product will be commercially successful. This success is largely defined by the number of hours that patients are willing to invest in a single title. By extracting factors of good game design, we can gain insight into factors that should increase patient engagement with rehabilitation and hence suggest criteria by which games might be well-suited for rehabilitation. These criteria are not solely based on game design principles from industry. We took an interdisciplinary approach (illustrated in Figure 51) to find areas of overlap between what we know about game design, the neuroscience of motivation, and principles of motor learning that have been shown to improve long-term retention and transfer in the general population.

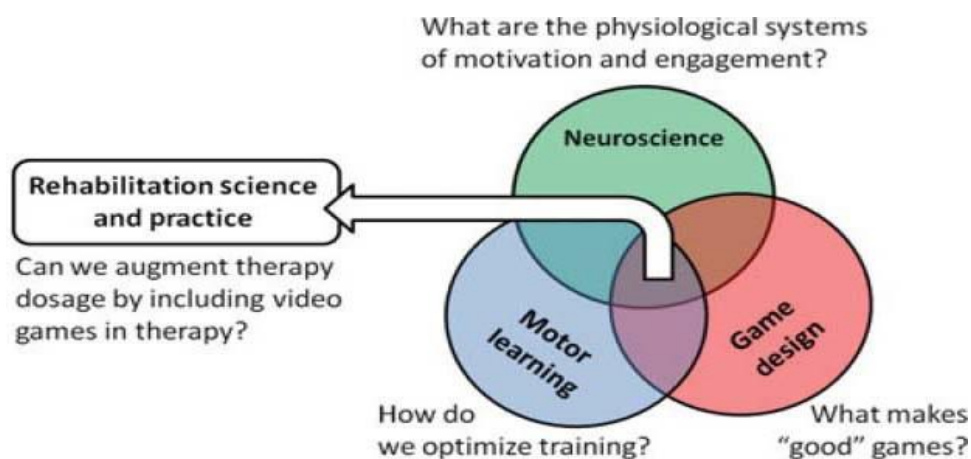


Figure 51 A schematic representation of the process of developing gaming principles for rehabilitation.

4.3.4 Feedback

Feedback is any information about how a skill was performed and/or the effectiveness with which the skill was performed. Feedback can be a natural consequence of an action, referred to as intrinsic feedback, which includes vision, proprioception, and equilibrioception. Importantly for engagement and learning, additional feedback is often provided from external sources, termed augmented feedback, which includes verbal feedback from coaches/therapists, video feedback, points, badges, and other performance-contingent rewards. Interpretable feedback leads to more efficient learning, but feedback should also be prescriptive rather than simply descriptive. Giving learners prescriptive feedback about what to do following an error reduces the cognitive load, which might be helpful early in practice or at a point of impasse.

Feedback serves an informational function for improving motor learning, but it also has a motivational function. Data from laboratory studies suggest that providing participants with feedback after good trials leads to better long-term retention of the skill than giving participants equal amounts of feedback following poor trials. This suggests that people like to receive affirmation of competency rather than errors or failure. In support, if participants feel that they are doing well relative to other people, this also increases their motivation to practice, their level of performance, and their retention of a skill. In game design, feedback can similarly be used to augment other factors, such as difficulty or socialization. For instance, clear and immediate feedback on differences between players could increase the competitive/adversarial aspects of a game, even if the game is designed to be cooperative. Therapists should be careful in evaluating games for the type of feedback they provide, choosing games that provide positive feedback more frequently.

4.3.5 Clear Goals and Mechanics

Gameplay offers achievement-based satisfaction, because completing the goals makes the patients feel satisfied with their progress. Clear goals and instructions for completing a task make task execution efficient. To reach goals, games must include clear instructions, which can be explicit tutorials or implicit cues. Explicit tutorials can either be dynamically included into gameplay or be separate documents. Exploration of the game and its mechanics can serve as implicit cues. Learning the mechanics of the game doesn't require a long time as it simulates the acts of real life which we do every day.

4.4 In-Game Hand Detection and Tracking

User interaction is an essential feature in the design of an interactive game especially in our game which simulates physical therapy exercises. In-Game hand tracking allows you to interact without needing any physical controllers. Sensors capture data on the position, orientation, and velocity of your hands. Hand tracking software then uses this data to create a real-time virtual embodiment of them. These virtual hands are integrated into applications, allowing you to see and use your hands naturally. But while the end-user experience of hand tracking feels intuitive, in reality this relies on layers of sophisticated technology.

4.4.1 Mediapipe Hand Tracking

As mentioned in Chapter 2, Mediapipe provides 21 3D coordinates. We use this coordinates in order to move the hand in game world and be able to detect gestures such as grasp and O position. The Coordinates is sent from python program to unity (will be discussed in chapter 5).

Hand Movement Mapping

The hand coordinates is mapped in order to fit the world's coordinate in the game. Using the following simple equation we were able to map these coordinate.

$$\frac{\mu - a_1}{b_1 - a_1}(b_2 - a_2) + a_2$$

Where:

μ is the value to be mapped

a_1 is the starting limit of the input (in real world)

b_1 is the ending limiting of the input (in real world)

a_2 is the starting limit of the output (in unity world)

b_2 is the ending limit of the output (in unity world)

The Equation is converted to the following function

```
public float Remap_Trans(float value, float in_start, float in_end, float Out_start,
float out_end) {
    return (value - in_start) / (in_end - in_start) * (out_end - Out_start) + Out_start;
}
```

Figure 52 Mapping function using C#

Hand Gestures Detection

To determine the position of the hand and detect a certain gesture we made an algorithm that calculates the distances between certain coordinates of the hand using Euclidean distance in order to detect the gestures.

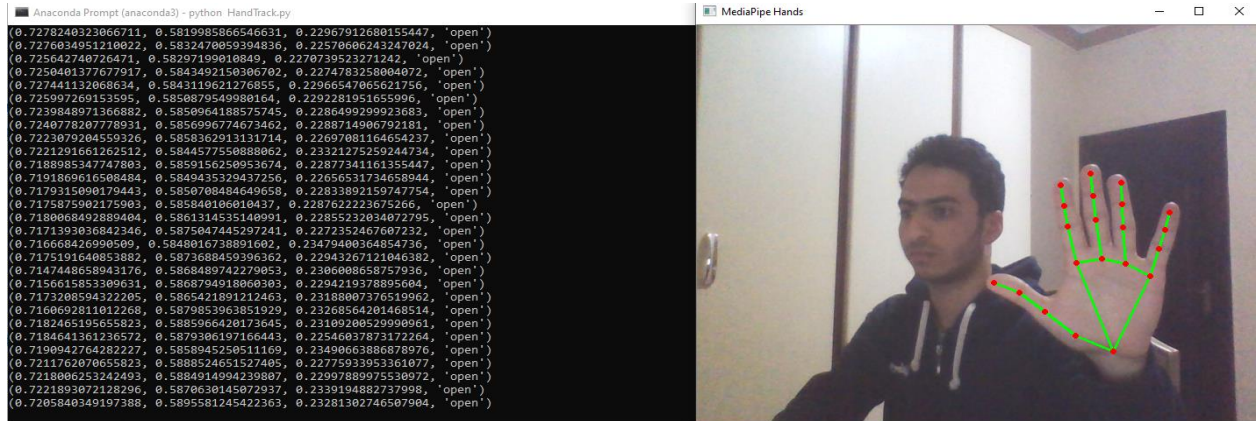


Figure 53 Detecting hand gestures using Mediapipe

```
def Calc_Distance(x1,y1,x2,y2):  
    dist = math.sqrt((x2-x1)**2 + (y2-y1)**2)  
    return dist
```

Figure 54-Euclidean Distance Using python

Gesture Detection Algorithm

By calculating the coordinates of finger positions relative to each other we can detect which gesture the hand is performing.

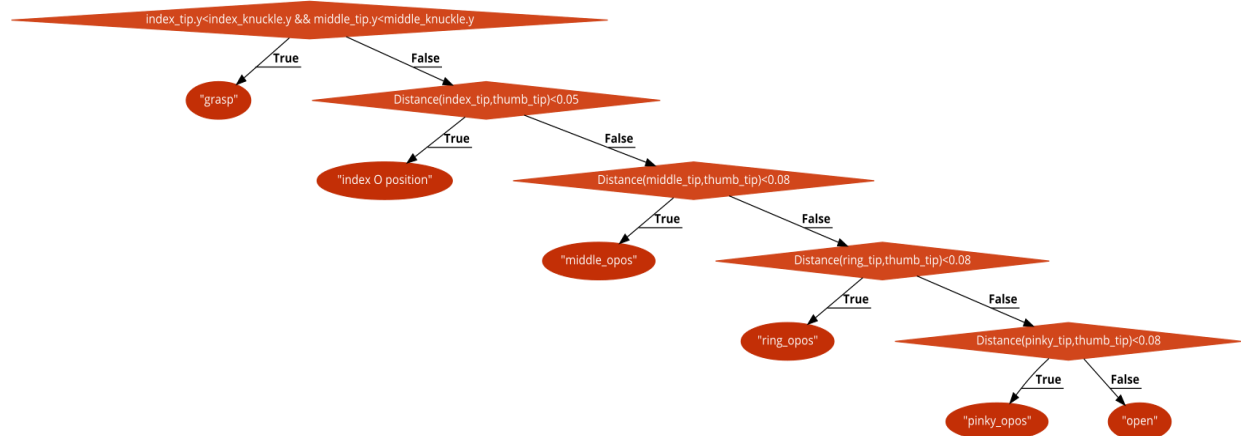


Figure 55 Hand Gestures Flowchart

4.4.2 Leap Motion Hand Tracking

Leap Motion provides unity developer package which contains all the tools the developer needs in order to design in game hand tracking. It contains the following modules:

- Core module

It's the dependency for all other modules. Leap Motion's Core Assets provide the foundation for VR applications with a minimal interface between Unity and the Leap Motion Controller and a collection of garbage less C# utilities. With Core, you can render a basic set of Leap hands, attach objects to hand joints, and find generally-useful utilities like a non-allocating LINQ implementation and a Tween library.

- The Interaction Engine module

It provides physics representations of hands and VR controllers fine-tuned with interaction heuristics to provide a fully-featured interaction API: grasping, throwing, stable 'soft' collision feedback, and proximity. It also comes with a suite of examples and prefabs to power reliable, stable 3D user interfaces as well as any physics-critical experiences akin to Leap Motion's Blocks demo.

- The Hands module

Provides the tools you need to rig your 3D hand assets to work with Leap hands, including the powerful hands-auto-rigging-tool.

Using the previous modules we can use our hands in game, interact with objects and detect gestures by only importing the Leap rig package into unity scene as in figure 56.

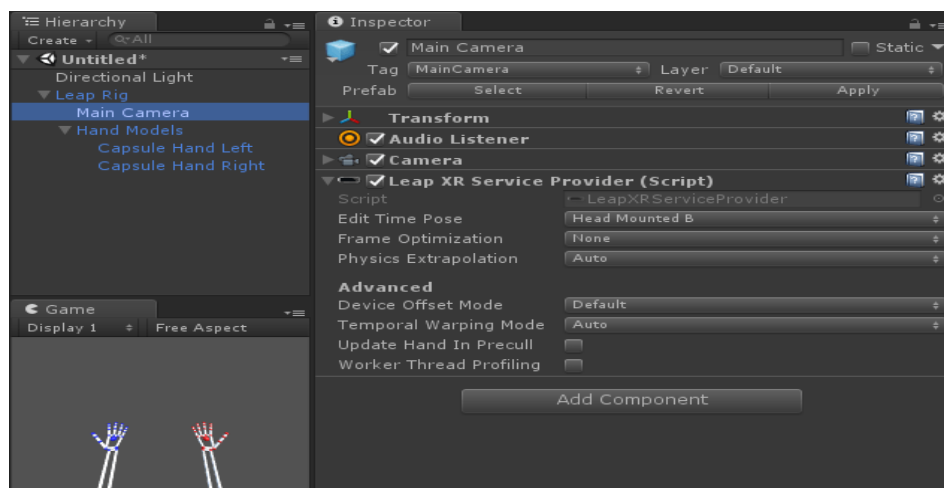


Figure 56 Leap XR Package

4.4.3 Games

Overview on games

We completed exactly two games which are “Cubes game” and “Musical fingers game”. Both games are targeting upper limb disability post stroke, and each game practice a specific hand function, for example “Cubes game” practice shoulder and elbow functions, and “Musical fingers game” practice hand opposition function (Note: Target injuries of each game will be discussed at length in the next subtitles.).

Cubes Game

Basic idea: It is basically a simulation of a room or a box that contain a ground divide into two or four regions depending on the level. A varying number of colored cubes - depending on the level of difficulty suitable for the patient – are generated randomly on the divided ground, and the patient is asked to grab each cube and put it in the region with the same color of the cube itself, so, it is basically like a sorting game but actually the patient in practicing his grasp, shoulder and elbow which help him/her restore normal functions in those specific parts of his/her arm, which are vital in his/her everyday activity.

Graduation of difficulty: has four levels of different challenges and each level has three gradual difficulties of easy, medium and hard, the difficulty is directly proportional with the number of cubes in each level.

Musical fingers Game

Basic idea: As we mentioned above this game practice hand opposition function which may help in improving the patient ability of moving his/her fingers which is vital in many of the everyday activities. So, the game is so much like piano tiles, instead of moving black blocks, pictures demonstrating hand opposition exercises are moving down and the patient earns a point by simulating the specific move appearing in front of him/her before it reaches the end of the screen.

Graduation of difficulty: This game contains only one level with varying difficulty between easy, medium and hard, the more the difficult the game is the faster it goes down.

4.5 Cubes Game

4.5.1 Target injuries

As we mentioned before cubes game target exercising the following functions, forming a multi-exercise program which help the patient exercise as many functions as possible more than a usual physical therapy session:

1. Grasp function: the patient will have to clench his fist to grab a cube
2. Shoulder function: as the patient will have to move his/her arm using shoulder to be able to lift the cubes from one region to another
3. Elbow function: the patient will have to move his elbow to reach out for the cubes

The following figure shows a variety of shoulder function exercises which are essential for restoring the ability to perform everyday tasks:

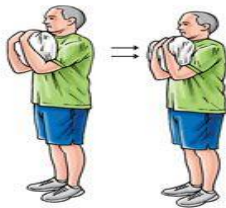
Shoulder Subluxation Rehabilitation Exercises



Isometric shoulder external rotation



Isometric shoulder internal rotation



Isometric shoulder adduction



Isometric shoulder flexion



Isometric shoulder extension



Isometric shoulder abduction

Figure 57 Shoulder subluxation rehabilitation exercises

4.5.2 Level design

We wished to make game environment interactive and helpful to patient also it should be as simple as possible, so in the **Cubes Game**, we designed a simple 3D design to the room and the cubes for each level in the game.

As we said earlier levels are graduated in difficulty as we tried to make the ordering of the levels starts with patient from the easiest shoulder and hand moves to the most difficult one.

Level One

Level One starts with two regions and the number of cubes in the regions depends on the difficulty selected:

Difficulty	Number of Cubes (in each region)
Easy	3
Medium	5
Hard	7



Figure 58 Level 1 Unity

Level Two

Level Two starts with four regions different in color, and the number of cubes generated in each region depends on the difficulty as **Level One** (refer to the table in **Level One** description).



Figure 59 Level 2 Unity

In this level we targeting the patient moves the cubes more at Z-axis, so he can train to stretch his arm.

Level Three

Level Three has the same design as **Level One**, but it has cubes on shelves with different heights, we are targeting that the patient catches the cubes that are on the shelves, so he can train to move his shoulder.

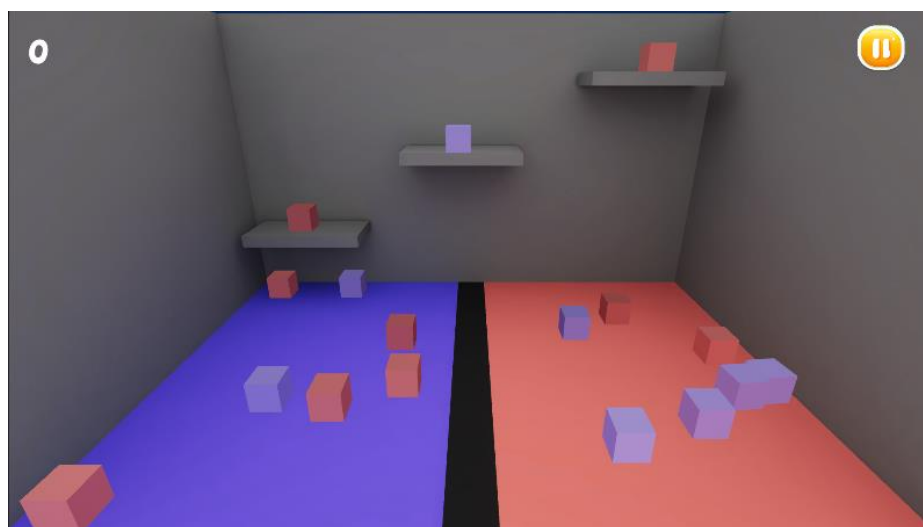


Figure 60 Level 3 Unity

Level Four

Level Four has the same design as **Level One** also, but it has a barrier between the two regions, the height of the barrier increases with the difficulty, we are targeting that the patient moves the cubes in Y-axis as high as possible, so he can train to move his shoulder up.

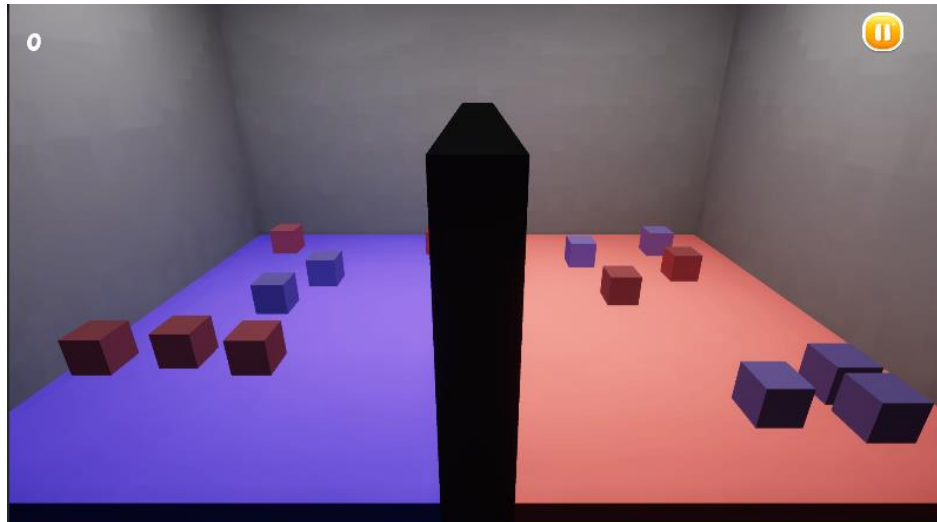


Figure 61 Level 4 Unity

4.5.3 Scripting

Here we are going to talk about the functionality in **Cubes Game**, like how we generate the level and how we calculate score and time and other things.

Level Generation

We developed our algorithm to generate the levels of the game, the idea of the algorithm that we want to generate n-cubes at a random position of a specific area such that two cubes can't have the same position.

Here is our algorithm that we make sure that each cube is not generated in the same position of the other cubes using the **Distance_Check** function, this function takes as arguments: the number of cubes we want to generate, the region (area) we want to generate the cube in it and an array that have all positions of all the generated cubes before this cube. Then we calculate the distance between the new cube and all other cubes in Z-axis then we check if this distance is enough or not. If they are far enough from each other the cube is generated successfully, if not we check the distance between the new cube and other cubes in X-axis, if

they are far enough the cube is generated, if not we add or subtract a value to the x-position of the cube to make the cube far enough from the other cubes.

```
private void Distance_Check(int cube_num, Area area, Vector3[] Cubes) {
    float distanceX, distanceZ;
    string flag;

    if (cube_num > 0) {
        for (int cnt = 1; cnt < (cube_num + 1); cnt++) {
            if (Cubes[cube_num].z >= Cubes[cube_num - cnt].z) {
                distanceZ = Cubes[cube_num].z - Cubes[cube_num - cnt].z;
            }
            else {
                distanceZ = Cubes[cube_num - cnt].z - Cubes[cube_num].z;
            }
            if (distanceZ < limit) {
                if (Cubes[cube_num].x >= Cubes[cube_num - cnt].x) {
                    distanceX = Cubes[cube_num].x - Cubes[cube_num - cnt].x;
                    flag = "greater_than";
                }
                else {
                    distanceX = Cubes[cube_num - cnt].x - Cubes[cube_num].x;
                    flag = "less_than";
                }
                if (distanceX < limit) {
                    if (flag == "greater_than") {
                        if (Cubes[cube_num].x + (limit - distanceX) < area.Xmax)
                            Cubes[cube_num].x += (limit - distanceX);
                        else Cubes[cube_num].x -= (limit + distanceX);
                    }
                    else {
                        if (Cubes[cube_num].x - (limit - distanceX) > area.Xmin)
                            Cubes[cube_num].x -= (limit - distanceX);
                        else Cubes[cube_num].x += (limit + distanceX);
                    }
                }
            }
        }
    }
}
```

Figure 62 Distance_Check function

Score Calculation

Here we are going to talk about how we calculate the score in the game, the main idea in calculating the score that we give a **Game Object Tag** to the cubes and the different regions, such that if a red cube is transferred from the blue region to the red region, we catch this event and increases the score by one.

We can this in the **Cube Score** script at the **Start** function we get the position of the cube and its tag and according to the tag we get the boundaries of the region of the same cube color, for example if the cube tag is red cube, then we get the boundaries of the red region.

```

private void Start()
{
    First_Position = this.GetComponent<Rigidbody>().position;
    Object_Tag = this.gameObject.tag;

    switch (Object_Tag)
    {
        case "Blue_Cube":
            Collider_Tag = "Blue_Side";
            boundryMin = Level_1_Generation.BlueSideX_Min;
            boundryMax = Level_1_Generation.BlueSideX_Max;

            break;
        case "Red_Cube":
            Collider_Tag = "Red_Side";
            boundryMin = Level_1_Generation.RedSideX_Min;
            boundryMax = Level_1_Generation.RedSideX_Max;
            break;
    }
}

```

Figure 63 Initialization of scoring data

Then we have the **OnCollisonEnter** event, which is called when the cube collides with something, we first check if the cube moved from his initial position or not, then we check the tag of the GameObject that the cube collided with it, if it is the required region or if it is another cube that lies in the required region one point is added to the score.

```

private void OnCollisionEnter(Collision collision)
{
    if (Cube_In_Score())
    {
        if (First_Position.x != Current_Position.x || First_Position.z !=
Current_Position.z) moved = true;
        if (Scored == false && moved == true)
        {
            if (collision.gameObject.tag == Collider_Tag)
            {
                One_Point();
                Scored = true;
            }
            if (collision.gameObject.tag == "Blue_Cube" ||
collision.gameObject.tag == "Red_Cube")
            {
                if (collision.gameObject.GetComponent<Transform>().position.x <=
boundryMax && collision.gameObject.GetComponent<Transform>().position.x >= boundryMin)
                {
                    One_Point();
                    Scored = true;
                }
            }
        }
    }
}

```

Figure 64 Collision detection function

Note: **Cube_In_Score** is method that specifies whether the cube is calculated to the score or not because as we said before we generated some cube to make the level more difficult, but they are not purposed to increase the score.

Time Calculation

We used a built method in unity to calculate the time, this function is **timeSinceLevelLoad()**, it calculates the time in seconds that passes after the unity scene is loaded.

```
Clock = (int)Time.timeSinceLevelLoad;
```

Figure 65 Time since the level is loaded

Then we have two events related to the time calculation which are **Pause Button** even and **Resume Button** event, we want to calculate the time that the game is paused to subtract it from the time since the level is loaded to get the actual playing time, so for this purpose we added lines of code to the Resume and Pause Button functions. If the pause button is clicked then we get the time at this moment, then when the resume button is clicked, we subtract the time since level is loaded from the moment at which the pause button is clicked, so now we have the time that kept paused, every time the paused the time is added, so at the end we have the total time that the game kept paused.

```
public void OnPauseButton_Click()
{
    InGame_Menu.gameObject.SetActive(false);
    Pause_Menu.gameObject.SetActive(true);
    pause_moment = Clock ;
}
public void OnResumeButton_Click()
{
    InGame_Menu.gameObject.SetActive(true);
    Pause_Menu.gameObject.SetActive(false);
    pause_time = Clock - pause_moment;
    resume_time += pause_time;
}
```

Figure 66 Catching the events of clicking pause and resume button

We calculate the actual time by subtracting the whole time from the time the game kept paused.

```
Stop_Watch = (int)Time.timeSinceLevelLoad - resume_time;
```

Figure 67 Calculating the pausing period of the game

As a final touch we convert the time from seconds to the format of **HH:MM:SS**, for this purpose we used a built-in data type in C# which is **TimeSpan**:

```
public string Calculate_Time()
{
    int hour, minute, second;
    hour = (int)Stop_Watch / hour_second;
    minute = (int)Stop_Watch / minute_second;
    second = Stop_Watch;
    minute -= hour * 60;
    second -= minute * 60;
    TimeSpan t = new TimeSpan(hour, minute, second);
    return t.ToString();
}
```

Figure 68 Converting time format from seconds to hours format

4.6 Musical Fingers

4.6.1 Target injuries

As we mentioned before Musical fingers game target exercising the following hand exercises which altogether help the patient in improving his/her grasping skill and that will help a lot in everyday tasks as we mentioned that the target is to help the patient get back to help him/her self and feel normal again:

1. Abduction
2. Adduction
3. Extension
4. Opposition
5. Reposition

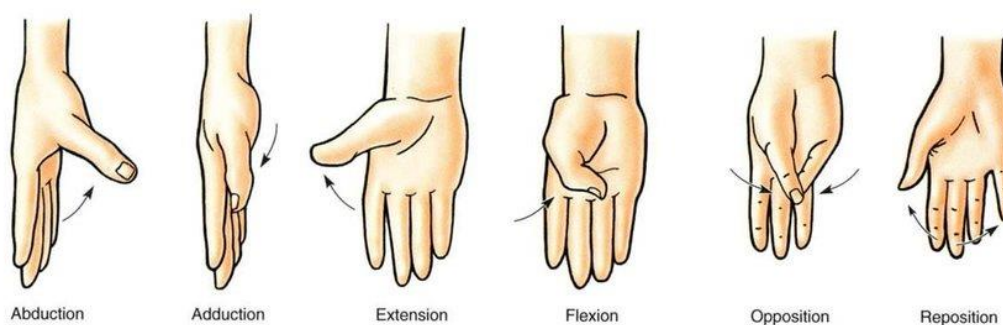


Figure 69 Hand opposition exercises

The following figure illustrates hand opposition exercises our game focuses on:



Figure 70 Hand opposition exercises 2

4.6.2 Level design

Our target in this game different, so we thought that we don't need a 3D design we just need a simple 2D design that does the required task.

We draw the hand moves on blender to use it in the game:



Figure 71 Opposition 2D hand design

The design of the game is so simple, it only consists of the hand moves and the end limiter, the game has one level only and the speed of falling of the hand moves increases with the difficulty.

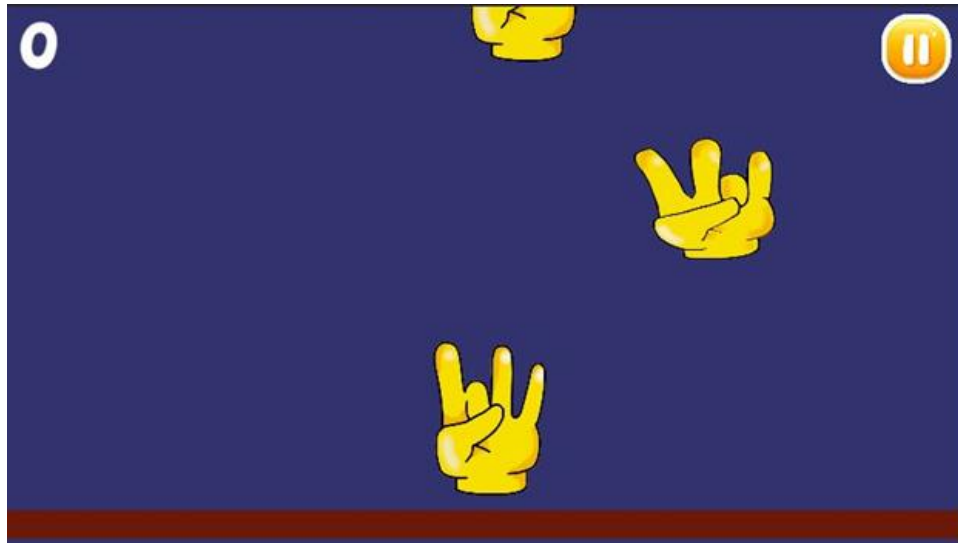


Figure 72 Musical Fingers Game on Unity

4.6.3 Scripting

Level Generation

The level generation is passed on generating random moves at random position in X-axis within the area of the game screen, the idea that we put the images of the moves on game objects, then we make an array with these game objects and the code picks a random element of the array and generates it.

```
void Generate_Move()
{
    random_num = Random.Range(0, Moves.Length);
    while (random_num == temp)
    {
        random_num = Random.Range(0, Moves.Length);
    }
    GameObject Spwaned_Move = GameObject.Instantiate(Moves[random_num],
        new Vector3(Random.Range(-5.6f, 5.6f), 5.6f, 0), new Quaternion(0, 0, 0,
1));
    temp = random_num;
}
```

Figure 73 Generating random hand moves

Score Calculation

To calculate score, we check the variable sent by the hand detection algorithm (refer to the topic of hand detection to read more about how this variable is sent), this variable is a **string** that have the same names of the tags that

are added to the moves game objects, so when this string is sent from the hand detection, we make some validation that it is not null or another hand position that we don't want, we destroy the game object that have this tag and add one point to the score.

```
private void Track_Move(string move)
{
    if (!string.IsNullOrEmpty(move) && !move.Equals("open") &&
        !move.Equals("grasp"))
    {
        Destroy(GameObject.FindGameObjectsWithTag(move)[0]);
        Game_Leader2.points++;
    }
}
```

Figure 74 Check hand move and calculate score

4.6.4 UI Design

Start Menu

First UI that appears to the user is the start menu from which the user can choose between the two games “Cubes game” on the right and “Musical fingers game” on the left.

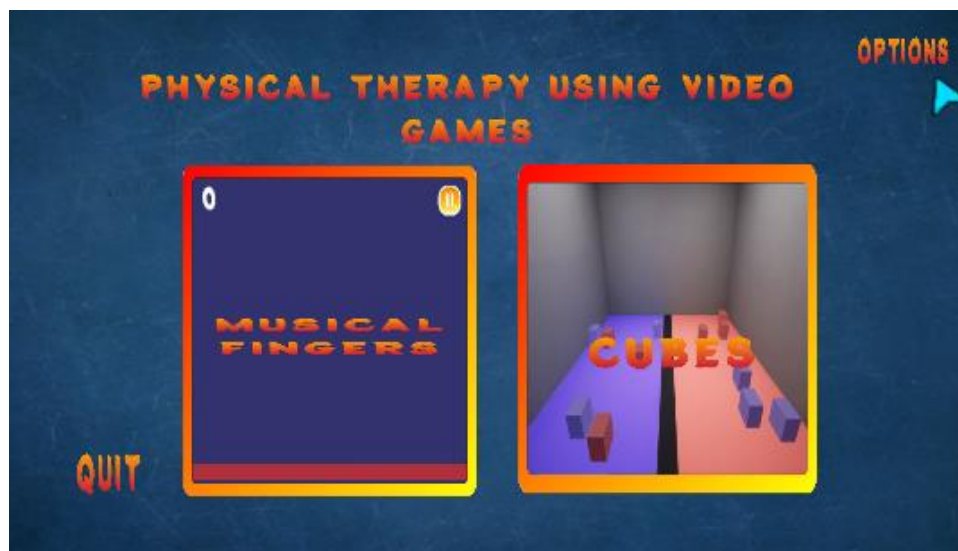


Figure 75 Start menu in Unity

Options Menu

Also, from that start menu the user can access the options menu from which the user can control the volume level of the games or may be turn off music.

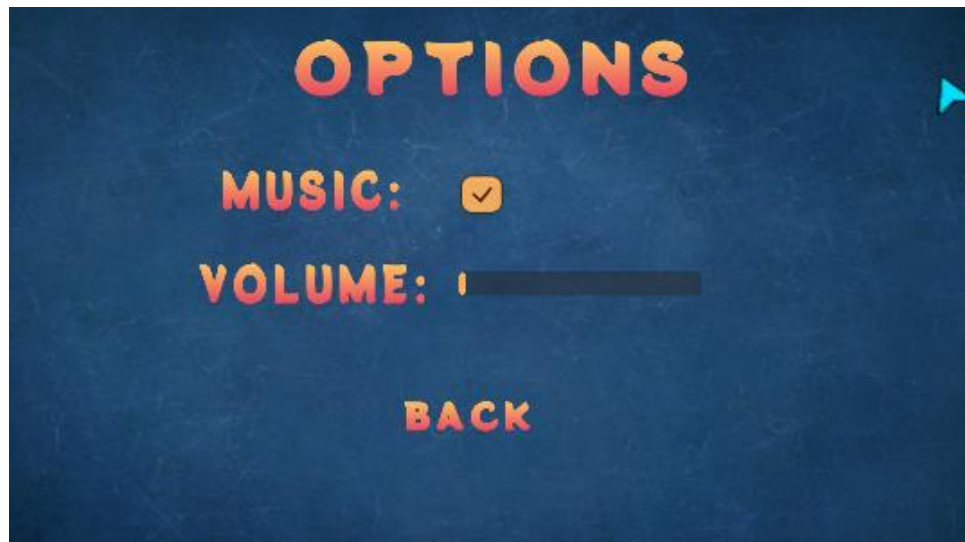


Figure 76 Options menu in Unity

Cubes game levels menu

If the user chooses to play cubes game, he/she will be directed to another level's menu from which he/she can choose a specific level to play.



Figure 77 Cubes' game levels

Difficulty menu

When the user decides which level to play in cubes level menu and click on it, A small menu of three levels Easy, Medium, and Hard appears so that he/she can choose difficulty of the level.



Figure 78 Difficulty menu in Unity

Also, if the user chooses to play musical fingers game, when he/she click on it in the start menu game a difficulty menu will appear so that he/she can choose level of difficulty from it.

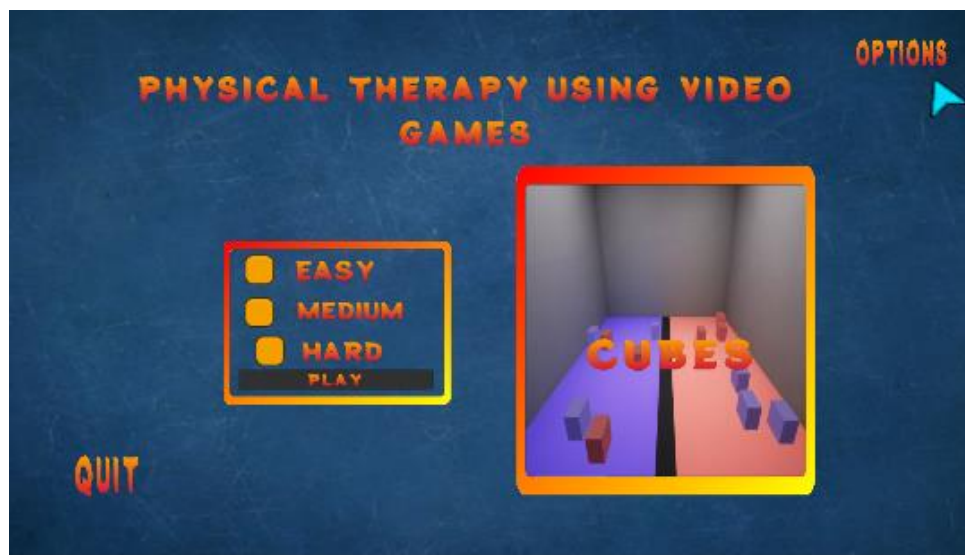


Figure 79Difficulty menu in Unity 2

In-game Menu

A very simple menu appears during game play including a counter at the top left side of the screen to increment every time the patient succeeds at putting one cube in its same color region, Also, includes a button on the top right side of the screen the user clicks when he/she wants to pause the game.



Figure 80 In-game menu

Pause Menu

If the user clicks the pause button on the top right side of the in-game menu this pause menu would appear containing sound controls and three button:

1. Restart Game button to restart the level the user was playing
2. Back to main menu button to get back to start menu from which the user chooses between games
3. Back to game button to resume level.

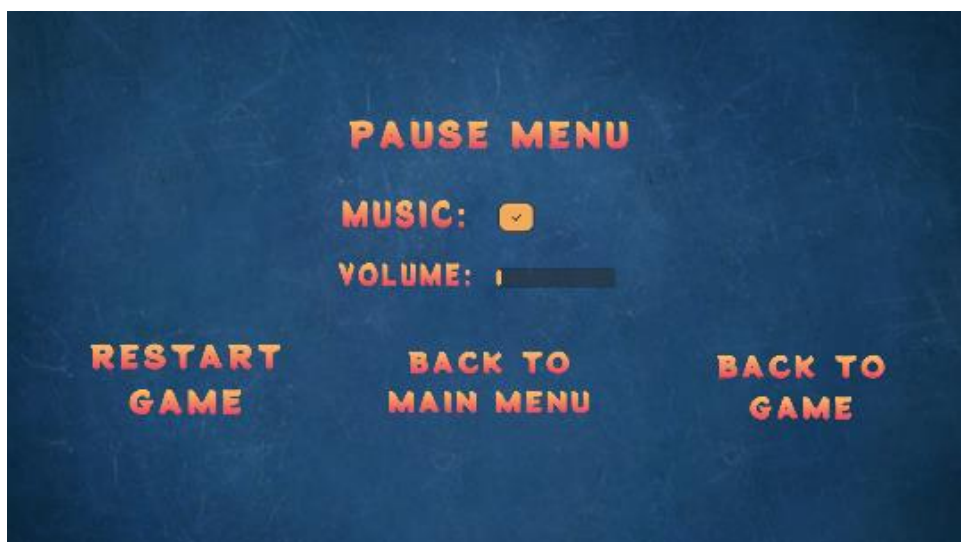


Figure 81 Pause menu in Unity

Level Complete Menu

This menu appears when the user succeeds at completing the level. This menu shows how much time spent to complete the level and the score. Also, it includes three buttons:

1. Main Menu button to get back to the start menu
2. Retry button to restart playing the level
3. Next level button to move to the next level with the same difficulty



Figure 82 Level complete menu in Unity

4.6.5 Sound Design

Video game sound design is the art of creating and adding audio elements to a video game. This involves creating entire libraries of custom sound effects to give the game a sense of realism and uniqueness. The sound effects must then be implemented properly to the images that will be seen by the player.

Biofeedback

Definition

Biofeedback therapy is an instrument-based learning process that is based on “operant conditioning” techniques. The governing principle is that any behavior—be it a complex maneuver such as eating or a simple task such as muscle contraction—when reinforced its likelihood of being repeated and perfected increases several fold.

Physical therapists use biofeedback to help refine a movement sequence or activation pattern to assist patients to achieve a goal. This technique involves using visual, physical and/or auditory feedback to guide the patient to give their optimal performance.

The ultimate purpose is that the patient gets to know his own body signs and that he can control them consciously in first place using biofeedback equipment, afterwards even without.

Mechanism of action

The autonomic nervous system regulates the functioning of the organs and functions of the body like breathing and heart beating. It isn't dominated by our will, but it reacts to our mood. There are two major components of the autonomic nervous system, the sympathetic and the parasympathetic systems. The parasympathetic works in particular at rest and recuperation while the sympathetic works at efforts. Due to chronic stress the autonomic nervous system can function worse, consequently the regulation of the body may get disrupted and an imbalance between the two systems may arise.

Role of Biofeedback in our project

We applied the concept of Biofeedback mainly using sound. For example, when the patient puts a cube in its right place a unique sound is triggered as feedback to his nervous system. Also, when the patient moves his hand to click a button in the UI this is also considered an exercise so a unique sound would be triggered as feedback. And throughout the game we used a relaxing background music to help the patient's nervous system to relax and be more focused.

4.6.6 In game sound Consistency

Sound in game consists of several parts which are

- MX = Music - any non-diegetic music.

Relaxing background music will help the patient to focus on the task and do it correctly. It also motivates the patient brain and helps him to recover faster.

- SFX = Sound Effects (Hard Effects) - any sound from a real-life object.

SFX Acts as a feedback to the patient helping him to know if he made the right move, clicked a button or gained a score point.

- BG = Backgrounds (ambience) - noise from the environment.

Background sound gives the scene realism and makes the game feels real.

4.6.7 In game sound management

To control in game sound we created a class called Audio Manager. It inherits MonoBehaviour, so it can be assigned as a Component to Game object called AudioManager. This class controls the attributes of the sound and its functions can play, stop or loop sounds.

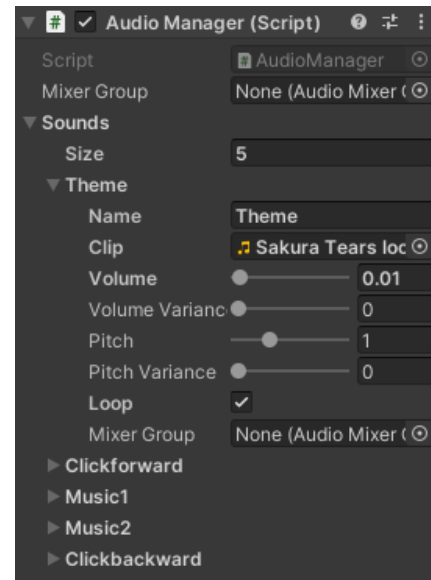


Figure 83 Audio Manager

Audio Manager Class Diagram

The AudioManager inherits from Sound class and it contains the functions to play or stop any sounds. Audio_Functions Class is a collection of all in game sound events that happens during the game.

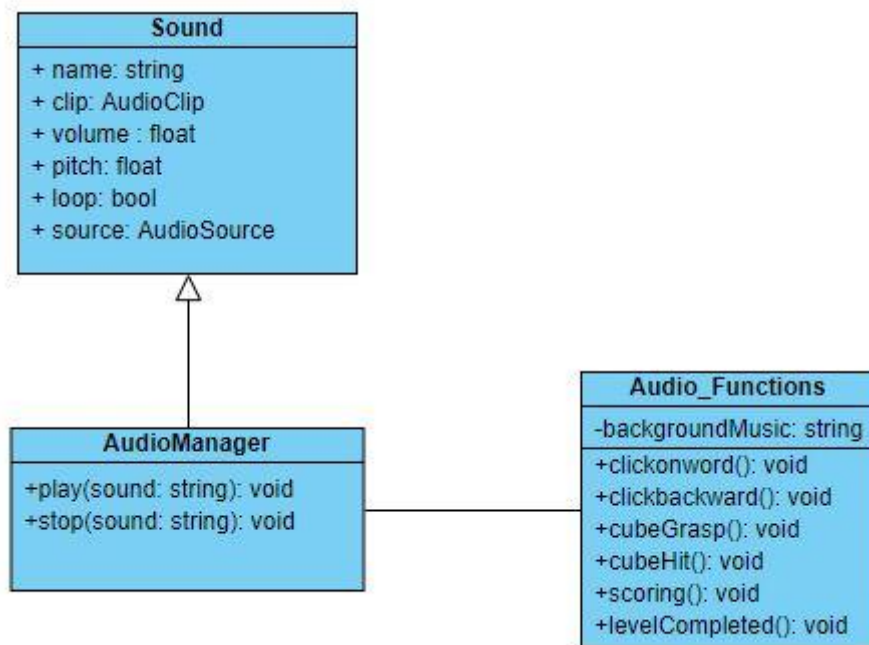


Figure 84 Sound Class Diagram

Chapter (5): Unity Interface with Other Environments

5.1 Media pipe Interface with Unity

Media Pipe Hand tracking is processed with python. Meanwhile, Unity uses C# as scripting language. In order to send data from python to C# and vice versa we used **Socket Programing**.

5.1.1 Socket Programing

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. In our case we use the local host in order to send and receive data from python to unity.

Advantages of using Socket Programing

- Flexible and sufficient
- Based on programming
- Can be easily implemented
- Low network traffic
- Send raw data between applications

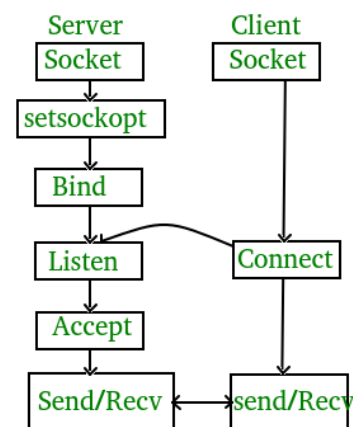


Figure 85 State diagram for server and client model

How data is sent from python to unity

In order to send a message using socket programming we construct a message with string data type. That message contains all the information we need on the other side separated by comas. On the other side we split the message to get each element as in figure 9.

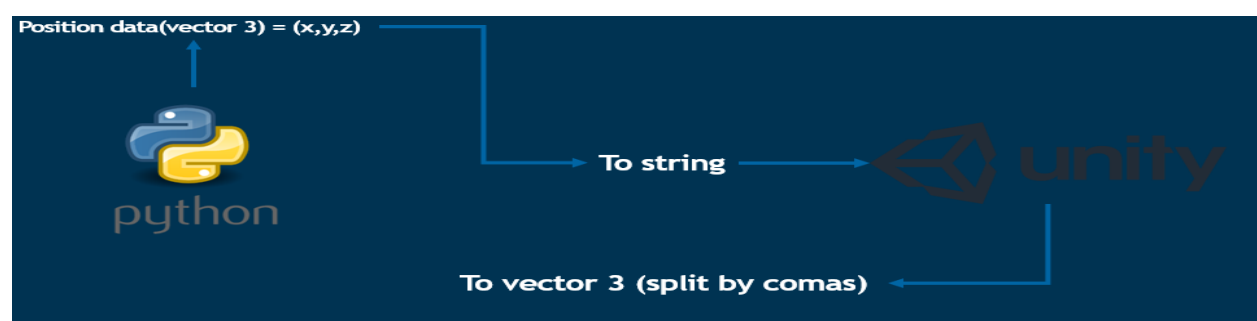


Figure 86 Python to Unity communication

5.1.2 Leap Motion Interface with Unity

As we mentioned earlier in chapter 2 that leap motion is just a monochrome camera which makes the hand detection and tracking easier for the CPU. The Leap Motion SDK is responsible for the hand detection and tracking. In order to send these data to unity we use the ultraleap plugin, which is developed by leap motion developers to work with unity.

Leap Motion SDK

The Leap Motion SDK contains two basic libraries that define the API to the Leap Motion tracking data. One library is written in C++, the second is written in C. Wrapper classes for these libraries define language bindings for C# and Objective-C. Language bindings for Java and Python use SWIG, an open source tool. The SWIG-generated bindings translate calls written in the bound programming language to calls into the base C++ Leap Motion library. Each SWIG binding uses two additional libraries. For JavaScript and web application development, Leap Motion provides a WebSocket server and a client-side JavaScript library.

Leap Motion Unity Modules Package

Unity Modules Package unlocks tools that we can use to design the hand tracking into our game. As we mentioned in the last chapter, the unity modules package provides 3 modules that helps us to view our hands in game , interact with objects and perform gestures.

5.1.3 Integration between Unity and Server

For designing a feedback system that allows the doctor to always supervise his patient progress in the treatment course, we needed to get the data of patient progress in the games such as the score he gets in the games, the time he takes to finish the level and etc..

For this purpose, it was required to send these data from the game to the server to handle the data and add the feedback data in the database.

Unity talks to the APIs through GET and POST requests to get info of patient and games from the database then publishing the feedback data to it.

```
IEnumerator GetRequest(string URL)
{
    using (UnityWebRequest R = UnityWebRequest.Get(URL))
    {
        yield return R.SendWebRequest();
        if (R.isHttpError)
        {
            Debug.Log(R.error);
        }
        else if (R.isDone)
        {
            Debug.Log(R.downloadHandler.text);
        }
    }
}
```

Figure 87 Get Request code in unity

```

IEnumerator Post_Handler(string [] key , string [] value , string URL)
{
    WWWForm form = new WWWForm();
    for ( int i= 0; i< key.Length; i++ )
    {
        form.AddField(key[i], value[i]);
    }
    using (UnityWebRequest R = UnityWebRequest.Post(URL, form))
    {

        yield return R.SendWebRequest();

        if (R.isHttpError)
        {
            Debug.Log(R.error);
        }
        else
        {
            Debug.Log("Form upload complete!");
            Debug.Log(R.downloadHandler.text);
        }
    }
}

```

Figure 88 Post Request code in unity

In the game, first it is required to sign with the patient username and password, we sent a post request with these data to the server to verify it, if the login was successful the server replies with patient id.

Then a GET request is sent to the server to get the games id and the levels id which is needed to be sent with feedback data.

When the patient tends to exit the game or after finishing a level in the game, a POST request is sent to the server with all the feedback data required.

```

public void Post_Stats()
{
    TimeTaken = Calculate_Time();
    string[] key = { "patientId", "gameId " , "levelId", "timeSpent",
                    "score", "MaxScore" , "Diffculty"};
    string[] value = { PatientID, GameID, LevelID,
                      TimeTaken, Score, MaxScore, Diffculty };
    GameObject.Find("Manager").GetComponent<API_Handler>().Post_Request(key,
                                value,Progress_URL);
}

```

Figure 89 Posting feedback data

API handles these feedback data to publish it in the database, for more information about how these data is handled refer to the API section in the **Web Development Chapter**.

Chapter (6): Design specifications

6.1 System requirements

6.1.1 Functional requirements

Requirements	Requirements brief description
REQ-1	Patient chooses game, level, difficulty using game UI
REQ-2	Goal of the cubes game is to sort all cubes according to their colors in the right color regions
REQ-3	Patient earns a point each time he/she puts a cube in its right color region
REQ-4	With each point scored a unique sound effect is triggered to notify the patient that he/she is doing the exercise in a right way (that approach is obeying biofeedback concept)
REQ-5	Simple UI to guide patient through the game
REQ-6	Cubes game contains four different levels each level has three difficulties (easy/ medium/ hard)
REQ-7	Progress of the patient is measured by two factors: <ul style="list-style-type: none">* How much he/ she spend to finish a level (long time is a bad sign, and less time is a good one).* How many times the patient plays the game in a day
REQ-8	Progress data is sent from the game engine to the database server
REQ-9	Web application is used to display the progress data of the patient, and facilitate communication between the physical therapist and the patient
REQ-10	A system admin can create/ modify/ delete the data in the system
REQ-11	A patient can create an account through the web application, modify, and delete his data
REQ-12	A physical therapist can create an account through the web application, modify, and delete his data
REQ-13	A patient can view his progression through the web application and contact his physical therapist
REQ-14	A physical therapist can follow up with his patient by reviewing his/her progress report via the web application
REQ-15	A physical therapist can set type of injury and therapy program for a patient via the web application
REQ-16	A physical therapist can specify how many patients he have

REQ-17	A physical therapist can evaluate his patients or send an evaluation report to his patients based on progression report
REQ-18	using hand tracking algorithm as a reflection of the patients hand to simulate his movements and control the game

6.1.2 Non-functional requirements

Requirements	Requirements brief description
REQ-19	Games provides exercises to help restore hand functions gradually
REQ-20	Cubes game focuses on shoulder functions and grasp function
REQ-21	Music tiles game focuses on fingers opposition functions
REQ-22	Progression measurement system
REQ-23	Games should adapt to changing abilities
REQ-24	These video games will help that patient to do exercises more easily as he just follow the game instructions and play it
REQ-25	Games will also motivate the patient as he/she earns score points, passes levels and gets motivating messages after every achievement.
REQ-26	These games also help the doctor so much as the doctor can explain the exercise to the patient easily by just explaining to him how to play the game.

6.1.3 Functional requirements specification

* Stakeholders:

- * Hospital.
- * Doctor.
- * Patient.
- * System Admin.

Actor and Goals

Participating:

Doctor: Deals with the system to see the progress that patient have done in the physical therapy, giving him evaluation for his work.

Patient: Uses the system to help in recovering from physical injury by playing video games that have been developed under the supervision of a specified doctor.

Initiating

System Admin: Monitors the system and can add or remove features from the system.

Camera: Captures live stream video for the patient during his physical session helping him practicing the video game.

Controller: Processes data and controls the system, it's the core of the System which is connected to the camera for processing the stream of camera and takes the right action according to the data.

* Use Cases:

* **Use cases description:**

* **Login:**

All users must login through the system to reach the system services.

* **Select Game:**

Patients select the suitable game according to his physical injury.

* **Play Game:**

Patient will overcome his injury by playing the game.

* **Read Patient Status:**

Doctor will have the ability to see the progress of his patients.

* **Evaluate Patient:**

Doctor could evaluate is patient according to the status of the patient.

* **Add Doctor:**

Admin will have the ability to add doctor and give him the ability to access patient's data.

* **Remove Doctor:**

Admin will have the ability to remove any doctor from the Database(system).

* **Add Patient:**

Admin will have the ability to add patient and give him the ability to play games.

* **Remove Patient:**

Admin will have the ability to remove any patient from the Database(system).

6.1.4 Use cases Diagram:

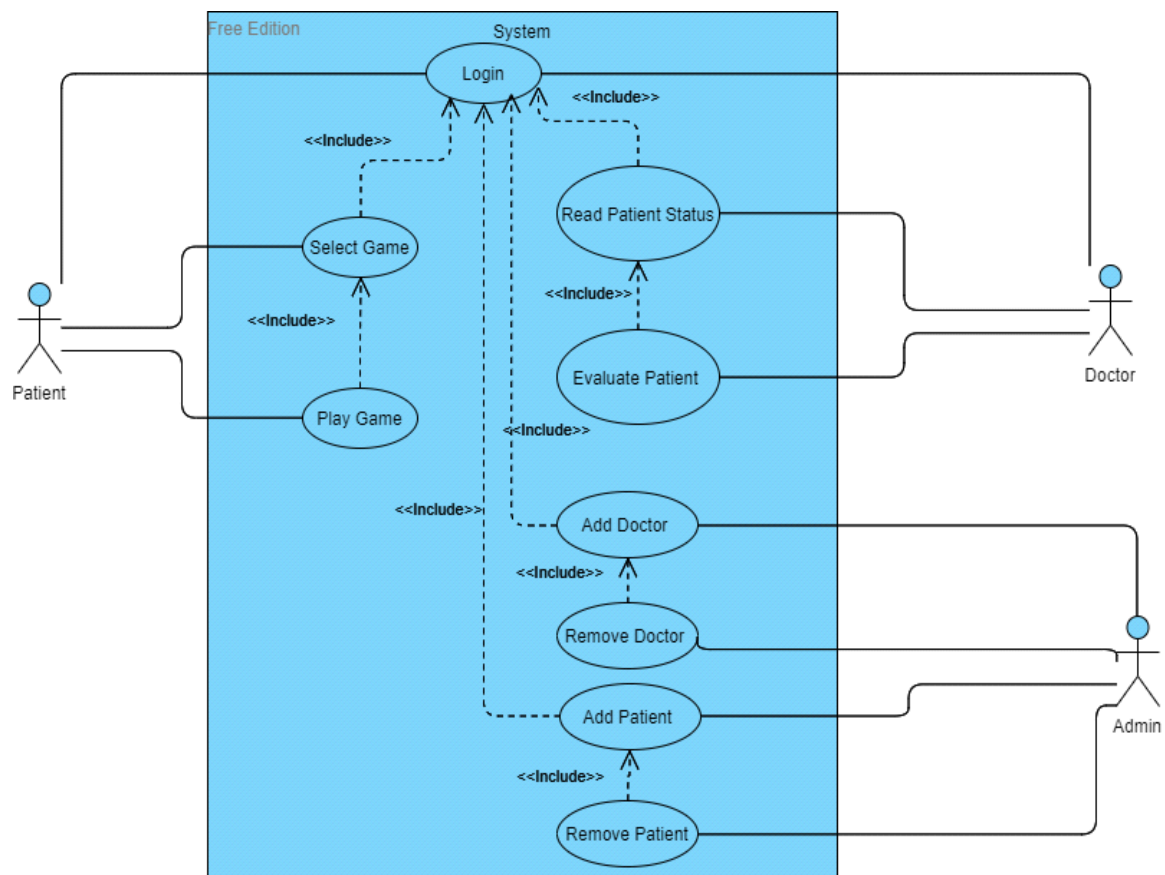


Figure 90 Use case diagram

6.1.5 System Sequence Diagrams

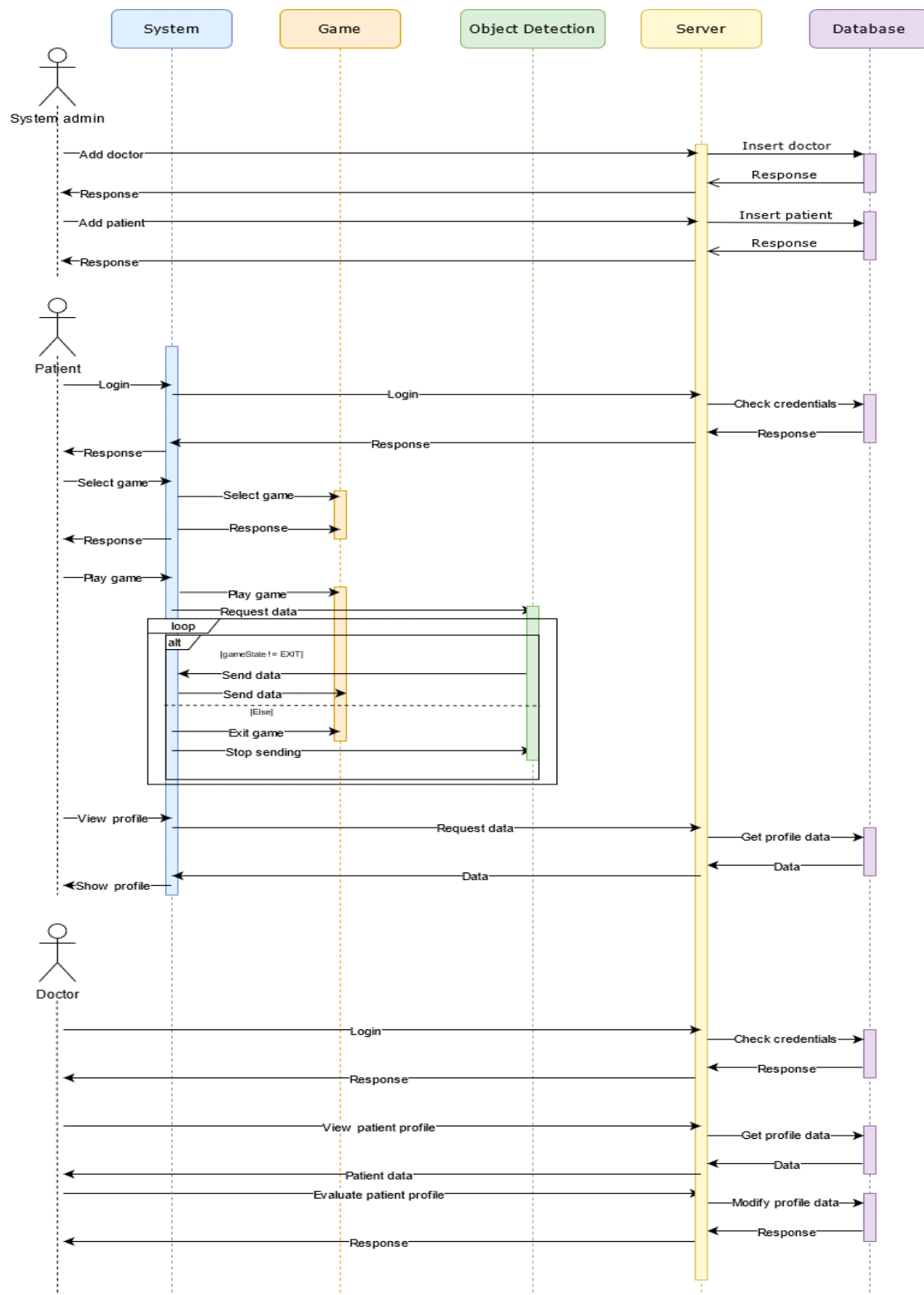


Figure 91 System sequence diagram

6.1.6 Class Diagram

Visual Paradigm Online Free Edition

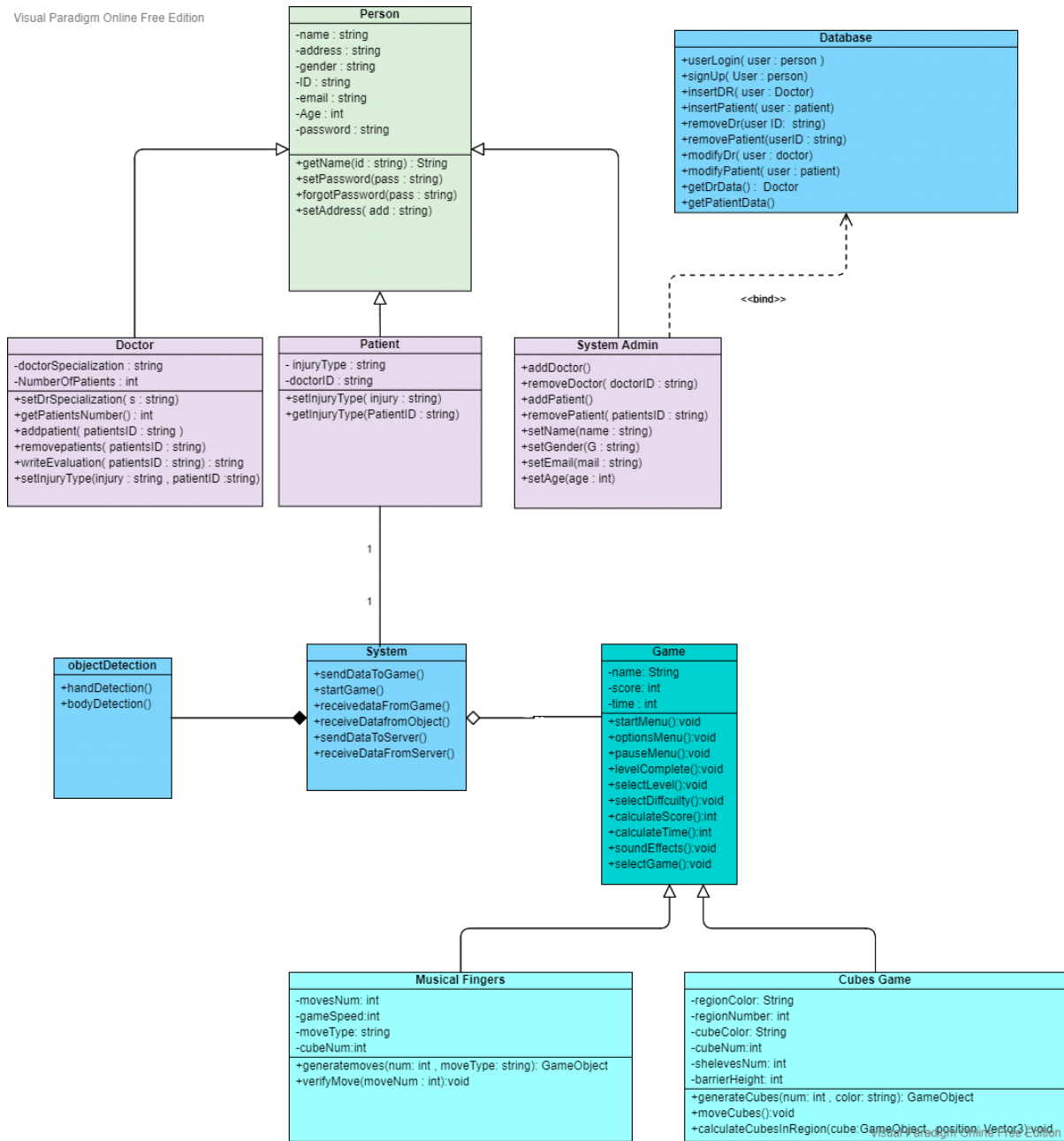


Figure 92 Class diagram

6.2 Tools and technologies:

6.2.1 Tools:

- * Unity game engine
- * Milanote (task management and documentation)
- * OpenCV : Used it in image processing to emphasize the features in the image, also tried various feature descriptors like HOG.
- * Tensorflow : Used it to create neural networks (deep learning model) with various types of layers like, CNN, max pooling, flattening and more.
- * Node.js
- * Vue.js

Technologies:

- * Deep learning
- * Computer vision

Chapter (7): Conclusion

7.1. conclusion

After all this hard work we succeeded at creating a complete treatment program for upper limb injury post stroke, by implementing two fun, easy, and truly help in the treatment process.

The first game (Cubes game) basically is a simulation of a room full of cubes and the ground is divided into two regions, a number of cubes specified according to the level difficulty are generated on the ground and the patient is asked to grab the cubes and put them in their region of color, this game contain four different levels each level help the patient exercise more shoulder, reaching out, and grasp function in a different way.

The second game (Musical fingers game) is basically like piano tiles ordinary game except instead of the moving black blocks pictures demonstrating the moves required from the patient are moving from up to down and the patient should make the move with his hands before the picture reaches the bottom of the screen.

We managed to detect hand movements using leap motion module as it provides great accuracy which our project count on so much.

In order to help the doctor, track the patient's progression we implemented a web application that help the doctor keep track of the patient's condition and also the patient can check put his progression.

7.2. Future Work

1. Plan for a case study: a person with no injuries tries the game and a patient try the game and compare between them.
2. Improving the web application to help in communicating between the doctor and the patient.
3. Implement more games covering more and different types of injuries so that it is not only for upper limb patients.
4. Perform hand detection using better or cheap tools with more accuracy.
5. Try our best to make our project fun, easy, and helpful for people with disabilities.

7.3 References

1. [Introduction to HTML \(w3schools.com\)](https://www.w3schools.com/html/html_intro.asp)
2. [HTML5 - Wikipedia](https://en.wikipedia.org/wiki/HTML5)
3. [CSS Introduction \(w3schools.com\)](https://www.w3schools.com/css/css_intro.asp)
4. [JavaScript | MDN \(mozilla.org\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript)
5. [Introduction | Vue.js \(vuejs.org\)](https://vuejs.org/guide/introduction.html)
6. [Installation | Vue.js \(vuejs.org\)](https://vuejs.org/guide/quick-start.html#installation)
7. [Font Awesome - Wikipedia](https://fontawesome.com/)
8. [Biofeedback - Physiopedia \(physio-pedia.com\)](https://physio-pedia.com/biofeedback/)
9. [Biofeedback - Physiopedia \(physio-pedia.com\)](https://physio-pedia.com/biofeedback/)
10. Lohse, Keith PhD; Shirzad, Navid MASc; Verster, Alida; Hodges, Nicola PhD; Van der Loos, H. F. Machiel PhD Video Games and Rehabilitation, Journal of Neurologic Physical Therapy: December 2013 - Volume 37 - Issue 4 - p 166-175 doi: 10.1097/NPT.0000000000000017
11. A. T. S. Chan, H. V. Leong and S. H. Kong, "Real-time tracking of hand gestures for interactive game design," 2009 IEEE International Symposium on Industrial Electronics, 2009, pp. 98-103, doi: 10.1109/ISIE.2009.5219910.

ⁱ [Installation | Vue.js \(vuejs.org\)](https://vuejs.org/guide/quick-start.html#installation)