VR Software To Test and Train Cognitive and Physical Abilities & To Detect Brain Concussion

Author
Andy Chen Xia (1458480)

Supervisor
Dr. Paul Blain Levy

Co-supervisor
Dr. Sang-Hoon Yeo

BSc. in Computer Science
School of Computer Science, University of Birmingham

April 6, 2017

Contents

1 Abstract

This project goes through the development process of making a software for HTC Vive (Virtual Reality device), which can be used to train and test different aspects of the users, mainly their cognitive and physical abilities. Hence, it can be utilized for health recovery too; moreover, the software is also designed for detecting brain concussion primarily on the pitch. Besides, to achieve the goals mentioned above; it contains different activities allowing the users to take tests depending on their purpose. Instant feedback is displayed to the players at the end of the exercises to make them know about their performance, and .csv files with more detailed information about the tests are exported, which can be used for more thorough analysis. Furthermore, to make sure the exercises do function without errors and generate correct and reliable data, several tests and evaluations were done.

2 Acknowledgment

I would like to thanks, my supervisor and co-supervisor for all their support and patience.

I would also like to thanks all the participants who have taken part in my experiment or participated in the software evaluation.

Lastly, I would like to thanks my family for their love and support.

3 Introduction

The inspiration to make this software is from the use of Wii in recovery centers for strengthening upper limbs and interactive gyms, which are gyms with interactive floors and walls making cardio activities more entertaining than traditional cycling and running.

The aim of this project is to create a software that can be used to help the users to stay fit, and to test and improve their physical and cognitive abilities. Furthermore, the software is also designed for detecting brain concussion on the pitch, which is possible thanks to the portability of the HTC Vive.

The software needs to be visually intuitive so the players or the experimenters(e.g. neurologist, physician, etc.) can navigate easily through the software. However, because of the nature of the VR, the players are required to explore the *'world'* to have a better understanding of the software, this aspect makes the HCI with VR very different to traditional HCI.

Furthermore, the software has a few categories, each category has its set of exercises, and for each exercise, the user can decide the settings of it, such as the duration or number of rounds of the exercises. At the end of the activities; feedback is given to the players, and it will generate a .csv file containing detailed information about the performance, which can be used for further studies. Additionally, the software can be used to collect useful data with Near-infrared spectroscopy (NIRS) machine to observe how blood flows to the brain in different exercises.

Moreover, the software needs to be entertaining so it can encourage the players to continue and not to quit.

4 Further background

4.1 HTC Vive

The HTC Vive is a Virtual Reality headset developed by HTC and Valve Corporation; it consists of 1 headset for displaying the 3D environment, 2 wireless controllers that are used for interacting with the Virtual world and 2 'LightHouse' base stations that keep track of the headset and wireless controllers.[1]

Figure 1: HTC Vive [2]

4.2 Unity

Unity is a game engine made by Unity Technology; it uses C# or JavaScript to develop games for PC, mobile devices, and websites[4]. Furthermore, it is the environment the software has built on, and what executes the software, below are some things that readers need to know.

Transform, it contains information about the object's location, rotation, and scale; besides, it is responsible for the object's motion, this can be *world transform* relative to the world or *local transform* relative to the parent.

Rigid body, it is in charge of the physics such as gravity and simple forces like pushes.

Collider, responsible for collisions and detecting if 2 or more different objects are touching; additionally, this can be set to *trigger*, which makes the objects to ignore collisions.

Development scene, where the developers work on the game.

Game view scene, it displays what the player is going to see when the game is running.

Assets, this contains all the materials, files, and coding used to make the software.

Collision layer, this is used to make objects collide with other specific objects that are tied up to specific layers. Here we are using the *default layer* that detects all the objects of any layer; *fruit layer*, this does not identify objects of another *fruit layer*, and *fruit piece layer*, this does not collide with objects of *fruit layer*, but it does collide with other objects of *fruit piece layer*.

Update, this is the most commonly used function to implement any game behavior, it is called in every frame, but the frame rate of it is not constant.

Tag, this is a label for identifying objects.

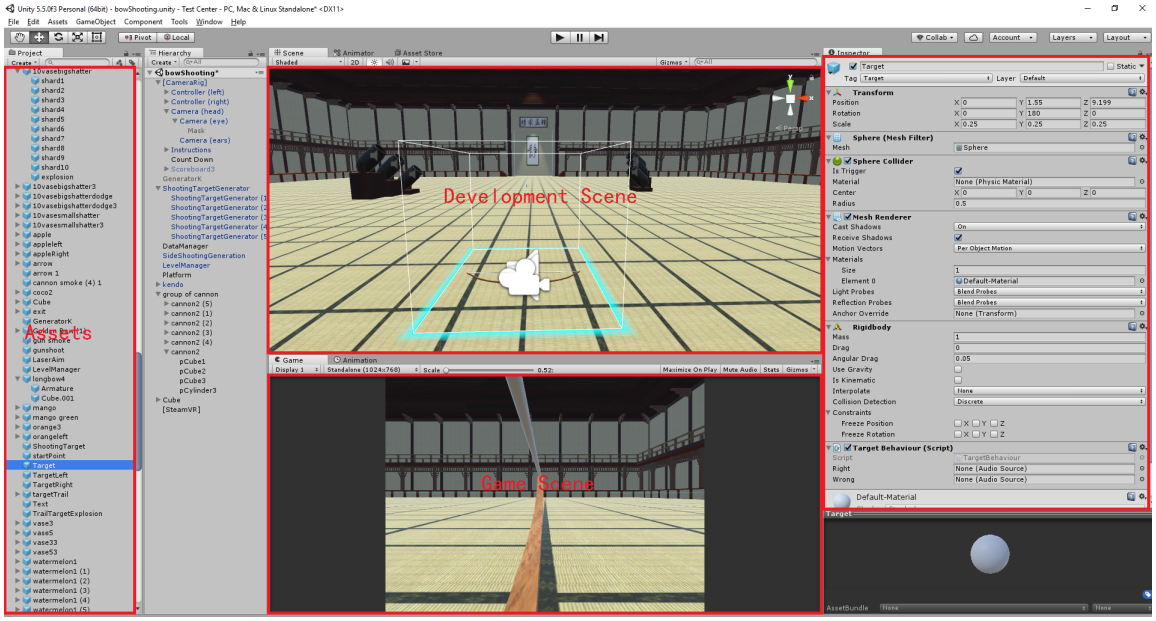


Figure 2: Unity Developer Window

4.3 Maya

Maya is a 3D Modeling software owned by Autodesk[3]; this is used to make all the graphics of my project. Moreover, it provides simple polygons, and by shaping and sculpting these simple polygons, we can get more complex forms like the apple in Figure 3. Additionally,

Maya supports UV Mapping[1], which converts a 3D object into a 2D Map, allowing me to put texture on the objects and make them look more realistic.

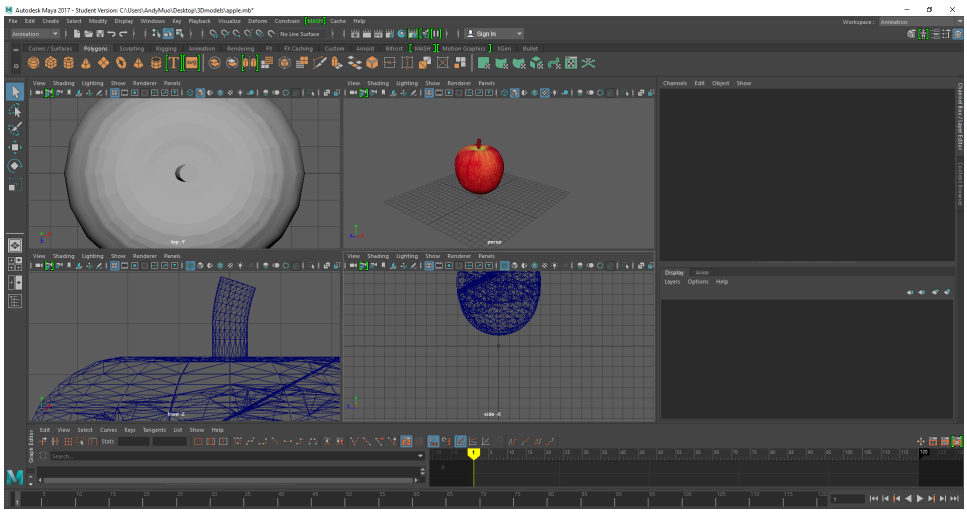


Figure 3: Making an apple using Maya

5 General Specification & Design

5.1 Functional Requirements

General requirements the software needs to achieve:

1. The software should allow the players to put their name before starting the exercise.
   (a) The software should tell the players if the name is valid.
   (b) The players cannot start if the name is invalid.
2. The software should allow the players to put a path for storing the data.
   (a) The software should tell the players if the path is valid.
   (b) The players cannot start if the path is invalid.
3. The software should allow the players to change and switch between exercises freely.
4. The software should allow the players to modify some aspects of the activities freely like the duration of it.
5. The software should display feedback about the performance of the players once the exercise is accomplished.
6. The software should generate a .csv data file of the activities under the path entered.

[1] `https://en.wikipedia.org/wiki/UV_mapping`

5.2 Non-Functional Requirements

5.2.1 Usability

The users should not have much difficulty to use the software, this might be a bit harder for people who do not have much experience playing games or using VR, but they should be able to guide themselves with the help provided in the software.

5.2.2 Performance

For a VR software responsiveness is key because this will determine the users' experience quality; hence, the game should not experience any latency.

5.2.3 Extensibility

The software should allow extensions, new categories and exercises should not be hard to add, and modification to current activities should be manageable with the comments provided in the code.

5.2.4 Entertainment

The users should find the software entertaining and joyful because entertainment is a significant factor for the users to use this software and to keep using it.

5.2.5 Reliability

The software needs to have some methods to recover from crashes if they ever occur, if the players were doing an exercise and a crash happens they can quit the exercise and restart if they want. Moreover, the data generated from different players and activities have to be easy to identify.

5.2.6 Portability

The software should be able to be executed on any operating system; however, the PC that is going to run the software needs to have at least a NVIDIA GeForce GTX 970/AMD 290 or equivalent and a HTC Vive.

5.3 Use Case

The diagram below shows a possible use case:

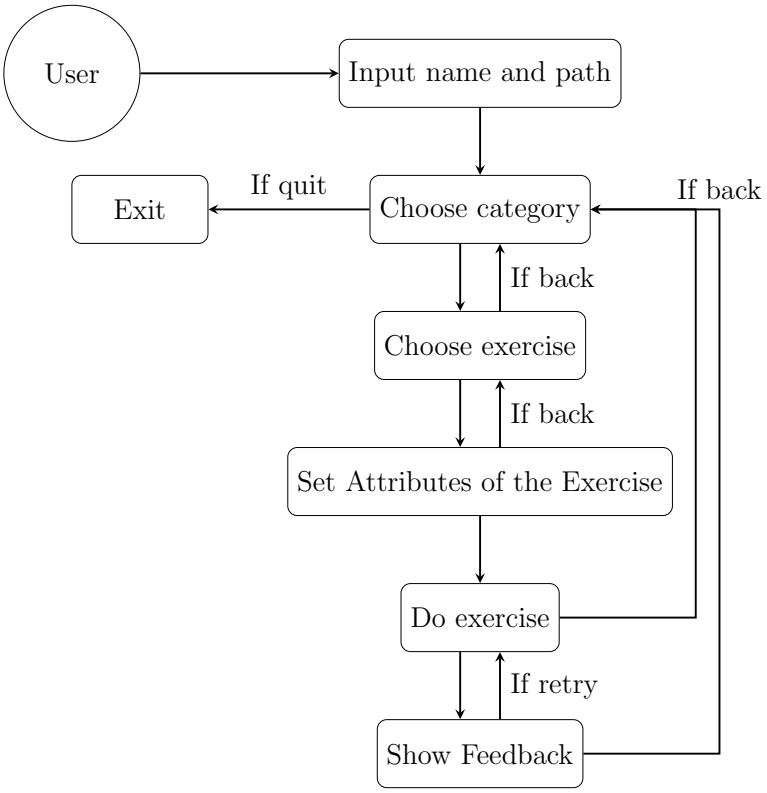


Figure 4: Use case diagram

5.4 Design

5.4.1 Structure

A bottom-up approach was used to design the software. The exercises of the software were made first, then they were grouped into different categories and finally they were linked through the GUI to get the final structure of the software. Figure 5 shows a general structure of the software; details will be cover in Section 6.

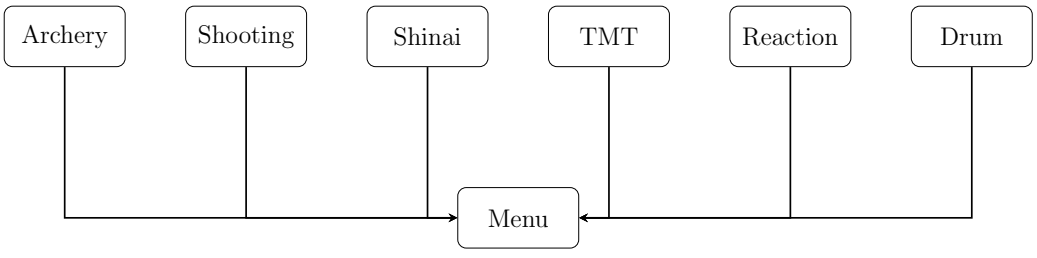


Figure 5: Bottom-up approach of the general structure of the software

5.4.2 Graphics

The software needs an environment where all the exercises can be performed, and the environment needs to be one that matches the exercises. Therefore, it is going to have an oriental feeling because the Japanese anime and game culture have been very popular among the young population, and another reason is that many exercises such as archery, shinai, and drum are widely practiced in their culture. Hence, the environment of the game is going to be a dojo (Japanese gym).

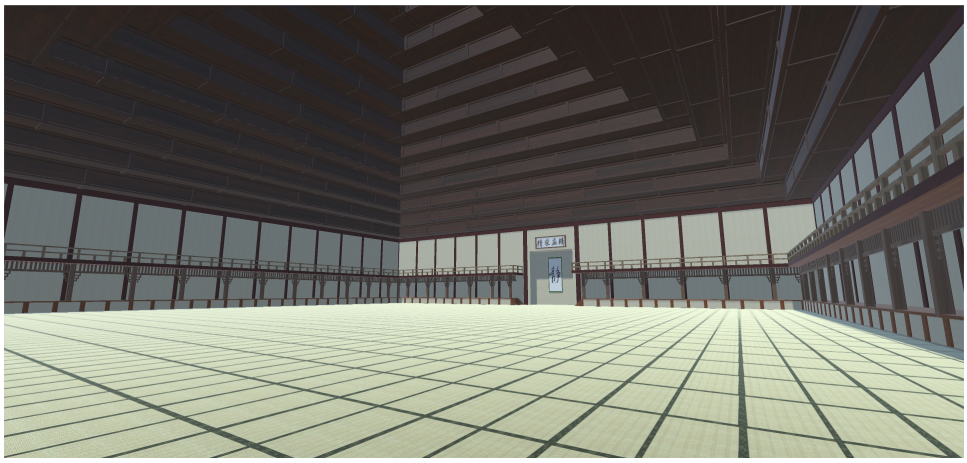


Figure 6: Dojo

6 Categories

This section will go through the details of each exercise as mentioned in Section 5.4.1

6.1 Archery & Shooting

The exercises of archery and shooting are the same; the only difference is that one category uses a bow and arrows another category uses a gun. The exercises in these categories are for testing the players' time and space prediction ability, but archery is also testing for their ability to accomplish tasks requiring 2 hand coordination. Furthermore, unlike the gun, which is instant shoot and hit, archery is asking the players to make a bigger effort in prediction since there is a period that the arrow is traveling before hitting the target. Therefore, we could say that archery is an upgraded version of the shooting exercises. Additionally, the archery's activities are designed to improve the users' upper limbs and back strength through the pull and shoot motion.

6.1.1 Design

6.1.1.1 Functional Requirements

1. The bow should spawn the arrow in the correct place and capable of doing pull and shoot.
2. The gun/bow should be accurate, aiming to where the player means to aim.
3. The bullet/arrow should recognize if the thing it is touching is a target or something else.
4. The number of targets hit, target missed and the number of shots missed need to be registered correctly.
5. Feedback needs to be displayed at the right moment.
6. Prefabs[2] need to be destroyed to improve PC performance.

6.1.1.2 Graphics

The basic graphic models made for this category are bow, arrow, gun, and targets. The bow has a skeleton allowing its string to be pulled and the targets are made from fragments, each target has 10 fragments so it can have the shattering effect when it is shot. Furthermore, in the actual exercise, the gun generates a laser for aiming.

Figure 7: Bow,gun,arrow and shelf model.

[2] https://docs.unity3d.com/410/Documentation/Manual/Prefabs.html

6.1.1.3 Implementation

These are the basic implementations that are used for all the exercises in these categories.

First of all, to make the bow interactive so it can detect the arrow and be pulled, it has a *trigger* in its center so when it detects that the user is putting an arrow on it, it will destroy the arrow on the player's hand and create a new one attached to the center of the bow, this is done this way due to the hierarchy change of the arrow, once the arrow is attached to the bow we can shoot and when the arrow is released sound effect is played.

The string of the bow can be pulled between a minimum and maximum value of its *local coordinates*, and the pull is done by calculating the distance between the 2 controllers; besides, it will determine the speed of the arrow. However, to make sure that the pull is realistic, the distance between the controllers is transformed into the scale of the *local coordinates* of the string since the controllers have the world as parent, the string has the skeleton as parent, and the skeleton is child of the bow. Therefore, this transformation was essential. Moreover, it is important to reset the parent of the attached arrow to the world. Otherwise, if it has the bow as the parent when the arrow is released it will move in accordance with the bow's motion like being 'remote controlled'.

When trying the bow, some problems were observed, sometimes when the arrow is released from the bow, the arrow was still on the string, and it was because occasionally the *trigger* of the bow goes back to the original point earlier due to the frame rate since the *update* frame rate is not constant. Hence, to solve this problem a boolean variable was added to the arrow; the *trigger* checks the boolean variable if it is false it will 'catch' the arrow, when the players release the arrow the boolean variable becomes true.

```
{
    stringDistance = (((Mathf.Abs(bow1.transform.localPosition.y) - 0.011f) / 0.05f));
    bow2.transform.SetParent(null);
    bow2.gameObject.GetComponent<Rigidbody>().AddForce(bow2.up * 50 * stringDistance, ForceMode.VelocityChange);
    bow2.gameObject.GetComponent<Rigidbody>().useGravity = true;
    bow1.gameObject.GetComponent<AudioSource>().Play();
    empty = true;
    bow2.GetComponent<ArrowContact>().shot = true;
    bow1.GetComponent<bow>().attached = false;
}
```

Figure 8: Code fragment for shooting arrow

The gun is the only 3D model that is not made by myself since it is provided in the *Unity Asset Store*, an online store where developers can download free or paid models from the Unity community. Furthermore, the gun has a laser, which is a red cube created dynamically, to visualize the ray cast and help the users to aim; the *raycast*[3] of the gun is used as the bullet since it can recognize if the object hit is a *target* or not. When the trigger is pressed

[3] https://en.wikipedia.org/wiki/Ray_casting

the gun shoots, plays a sound effect and creates smoke, and the smoke is created using *particle system*[4].

The targets, each of them is made from 10 fragments, and these fragments together make a single game object, the entire target, which is the parent of them making management much complicated. Furthermore, a simple *box collider* is created for every piece, but to make sure that these colliders do not interact with each other and waste PC performance their collision layer was set to *fruit layer*, when the target is shot it calls the class *Explode* which creates an explosion in the middle of the target. However, to make the explosion effect, the fragments are detached from the parent object and their *rigid body* is activated before the blast.

Additionally, *boundaries* are created, these are mesh-less cubes with tag *limit* and *box collider*, they are used to destroy the missed prefabs that contact them to improve PC performance.

```
if (other.CompareTag("Target") && other.transform.parent.GetComponent<ShootingTargetBehaviour>().missed == false
    && other.transform.parent.GetComponent<ShootingTargetBehaviour>().shot == false)
{
    other.transform.parent.GetComponent<ShootingTargetBehaviour>().missed = true;
    DataManager.sTragets.Add(other.transform.parent.gameObject.GetComponent<ShootingTargetBehaviour>().trajectory);
    Destroy(other.transform.parent.gameObject);
    DataManager.MissedTarget++;

}
```

Figure 9: Code fragment of boundary

Moreover, The exercises are divided into rounds, and each round can spawn a certain number of targets, and these are generated with a spawn rate, all these can be decided by the users before starting the exercise, and everything that happens during the exercise is recorded by the *DataManager*.

[4]https://en.wikipedia.org/wiki/Particle_system

```
if (hit.collider.CompareTag("Target")
    && hit.collider.gameObject.transform.parent.gameObject.GetComponent<ShootingTargetBehaviour>().shot == false)
{
    hit.collider.gameObject.transform.parent.gameObject.GetComponent<ShootingTargetBehaviour>().shot = true;;
    Destroy(hit.collider.gameObject.transform.parent.gameObject.GetComponent<Rigidbody>());
    childR = hit.collider.gameObject.transform.parent.gameObject.GetComponentsInChildren<BoxCollider>();

    foreach (Component childs in childR)
    {
        childs.gameObject.AddComponent<Rigidbody>();
        childs.gameObject.GetComponent<Rigidbody>().useGravity = true;
    }
    hit.collider.gameObject.transform.parent.gameObject.GetComponent<ShootingTargetBehaviour>().playAudio();
    hit.collider.gameObject.transform.parent.gameObject.GetComponentInChildren<explosion>().enabled = true;
    Destroy(hit.collider.gameObject.transform.parent.gameObject, 0.5f);

    hit.collider.gameObject.transform.parent.gameObject.GetComponent<ShootingTargetBehaviour>().stg.counter--;
    DataManager.ShootingHit++;
```

Figure 10: Code fragment of gun

6.1.2 Explanation : Shelf Exercise

In this exercises targets are generated from the sides of the shelf, this is the simplest exercise in these 2 categories because the targets are only moving in the X coordinate, which implies the trajectory is just a line.

6.1.2.1 Class Structure

The class diagram of this exercise is shown in appendix A.

6.1.2.2 Graphics

In this exercise, a shelf model, which can be observed in Figure 7 and 11, has been made as the platform for generating the targets. Other graphics that are used in this exercise are mentioned in Section 6.1.1.2.

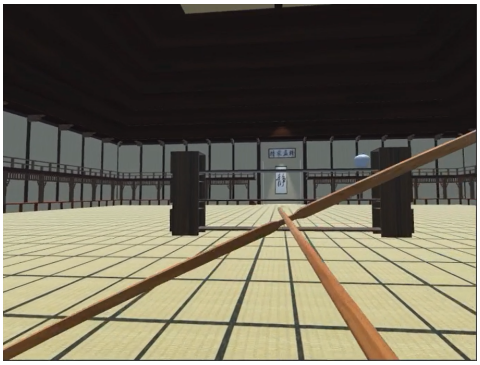


Figure 11: Player doing the Shelf Exercise

6.1.2.3 Implementation

The basic implementation has been introduced in Section 6.1.1.3, and for this exercise 6 generators responsible for spawning the targets have been created at the top, middle and bottom level of the right and left side of the shelf. The targets are spawned randomly from any of the 6 generators, but each level can only have one generator spawning targets; thus, they are grouped into 3 sets representing the 3 levels, and each set has 2 generators from the same level, and the generators will have a boolean variable *sender* that helps us to know which generator is spawning at the current level and which one is receiving, if *sender* is true the generator is sending targets, if it is false the generator is receiving, when *sender* of both generators are false one of them is randomly chosen to spawn the targets.

Moreover, to know when the targets spawned by one generator have all been received or destroyed, a *counter* variable for each generator is created, the *counter* is 2 times the number of fragments of the targets generated, for example, if the generator generates 2 targets it will have counter 40, when the counter of the generator becomes 0 it means that its *sender* becomes false; the counter is subtracted by 2 times the target's number of pieces when it exits the receiver's *trigger* or when the target is shot.

```
void OnTriggerExit(Collider other)
{
    if (LevelManager.scene == 3 || LevelManager.scene == 8)
    {
        destroyCounter++;
        if (destroyCounter == child)
        {
            stg = other.gameObject.GetComponent<ShootingTargetGenerator>();
        }
        else if (destroyCounter == child * 2 && shot == false
            && this.CompareTag("Target") && missed == false)
        {
            Destroy(this.gameObject);
            stg.counter--;
            DataManager.MissedTarget++;
            missed = true;
        }
```

Figure 12: Code fragment of generators

6.1.3 Explanation : Side & Front Cannon Exercise

These 2 exercises generate targets from cannons, their trajectories are more complex. The side cannon exercise makes a trajectory along the X and Y axis, while in the front cannon exercise the targets move in the Y and Z axis, both are making a parabola, but the second one is relatively easier than the first one since the targets are going toward the position of the player, unlike the side cannons which is going from right to left and vice versa.

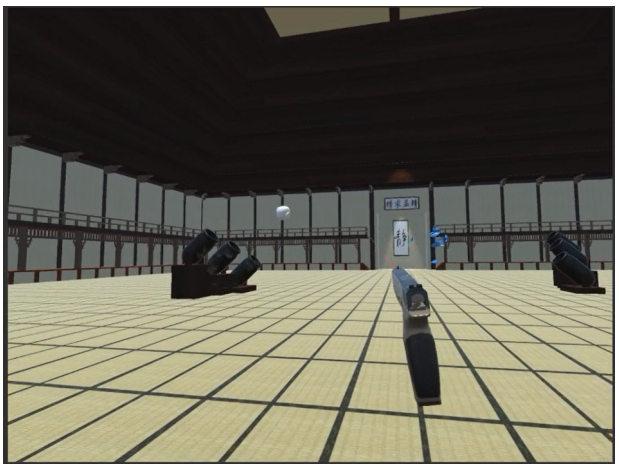


Figure 13: Side Cannon Exercise

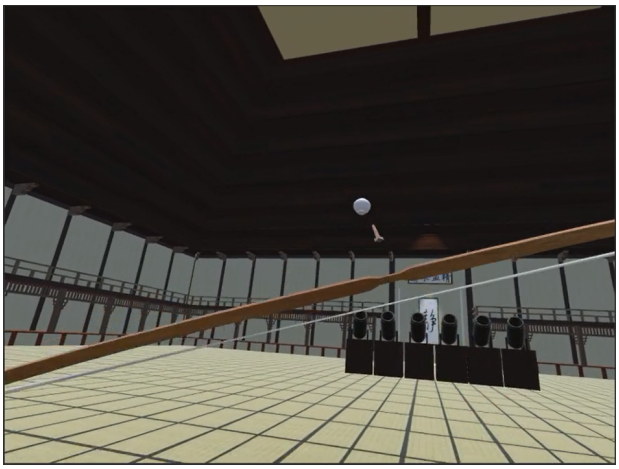


Figure 14: Front Cannon Exercise

6.1.3.1 Class Structure

These exercises have the same class structure shown in Section 6.1.2.1.

6.1.3.2 Graphics

These exercises contain the basic graphics shown in Section 6.1.1.2; however, 6 cannons and cannons' smoke ,which is created with *particle system*, have been added in this exercise.

Figure 15: Cannon with smoke

6.1.3.3 Implementation

The fundamental implementation has been introduced in Section 6.1.1.3, in these two exercises the targets are spawned randomly from any of the 6 cannons. Moreover, the targets apart from the force it receives to be ejected it will also have a rotation, which is a random Euler angle, and the targets will be destroyed when they touch the boundary or are shot.

6.1.4 Explanation : Stroboscopic Training

This exercise was made because of lack of cognitively demanding activities in this category, this is similar to the Front Cannon Exercise, but the difference is that the targets are alternating between visible and invisible; mocking Michael Jordan's strobe glasses behavior [5]. Besides, strobe lights blocking visual information for a short time has been shown to improve people's visual-motor abilities, their time and space prediction capacity and their visual memory [6, 7]; hence, helping them to create a high-quality internal model of the world.

6.1.4.1 Class Structure

These exercises have the same class structure shown in Section 6.1.2.1.

6.1.4.2 Graphics

The graphics of this exercise are the same graphics in Section 6.1.3.2.

6.1.4.3 Implementation

The implementation is like Section 6.1.3.3, but stroboscopic behavior has been added to the targets, the visible and invisible duration can be decided by the users, and these 2 will be alternating till the target gets destroyed.

```
IEnumerator SST()
{
    while (true)
    {
        for (int i = 0; i < mesh.Length; i++)
        {
            mesh[i].enabled = false;
        }

        yield return new WaitForSeconds(PlayerPrefs.GetFloat("invisibleSST"));
        for (int i = 0; i < mesh.Length; i++)
        {
            mesh[i].enabled = true;
        }
        yield return new WaitForSeconds(PlayerPrefs.GetFloat("visibleSST"));
    }
}
```

Figure 16: Stroboscopic behavior

6.1.5 Data Collected

The performance data collected from this category are the number of targets hit, targets missed, shots missed, the trajectory of the arrow, and the trajectory of the targets.

6.2 Shinai

This category has 3 exercises using a shinai to interact with the objects, and its main purpose is to train the strength of players' upper limbs, unlike the archery & shooting category where the users could interact with the target at a far distance, this requires the users to do an actual reaching movement to the objects and can only hit them when they are close enough.

6.2.1 Design

6.2.1.1 Functional Requirements

1. The shinai should interact with the targets.
   (a) The targets will give feedback when the players do a slice action.
   (b) The targets will not give feedback if the players do not slice them.
2. The shinai should always be in the hands of the players, even if a collision occurs.
3. The data of each exercise needs to be recorded correctly.
4. The fruit targets should have a smashed effect when they are hit by the shinai.
5. The unnecessary prefabs need to be destroyed to improve PC performance.
6. In the Dummy Exercise, the dummy should move around the player with random directions.

6.2.1.2 Graphics

The basic model of this category is the shinai, which is used in every exercise and what interacts with the targets. Furthermore, the targets are mainly fruits which are made from fragments, and each fruit is made of 6 pieces.

Figure 17: Shinai

6.2.1.3 Implementation

The shinai should always be fixed to the players' hand like a glove, when the shinai hits the target it receives a collision force, and this will make the shinai to move away from the players' hand; thus, every time the shinai enters or exits a collision its position is reset to the players' hand position, the rotation and speed are nullified too.

The fruits are made of 6 *colliders* from the 6 pieces, and do not collide with other fruits when they are as a whole since their collision layer is set to *fruit layer*. Nonetheless, when the fruit is hit, the collision layer of the pieces are changed to *fruit piece layer*, this is done by finding the fragments of the fruit and change them 1 by 1 since the number is not large a *for loop* is used.

Furthermore, the speed of the shinai is calculated in every frame to know if the players are slicing or not; moreover, this speed is used by the targets to decide the feedback they are going to produce.

```
public void FixedUpdate()
{
    velocity = new Vector3
        (this.transform.position.x - lastpos.x,
        this.transform.position.y - lastpos.y,
        this.transform.position.z - lastpos.z);
    velocity = velocity / Time.deltaTime;
}
```

Figure 18: Calculating speed of the shinai

6.2.2 Explanation : Watermelon Exercise

This exercise is created to improve the users' upper limbs strength and their dynamic visual acuity to differentiate moving objects. In this activity 3 watermelons will be generated in each round, the 3 of them will have different size(small, medium and big) and the players are asked to slice the one with medium size.

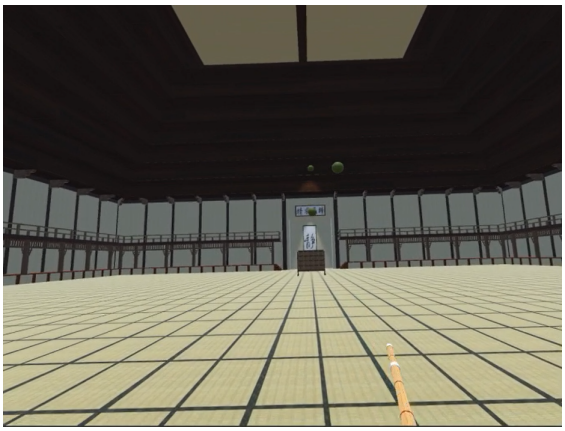


Figure 19: Player doing the Watermelon Exercise

6.2.2.1 Class Structure

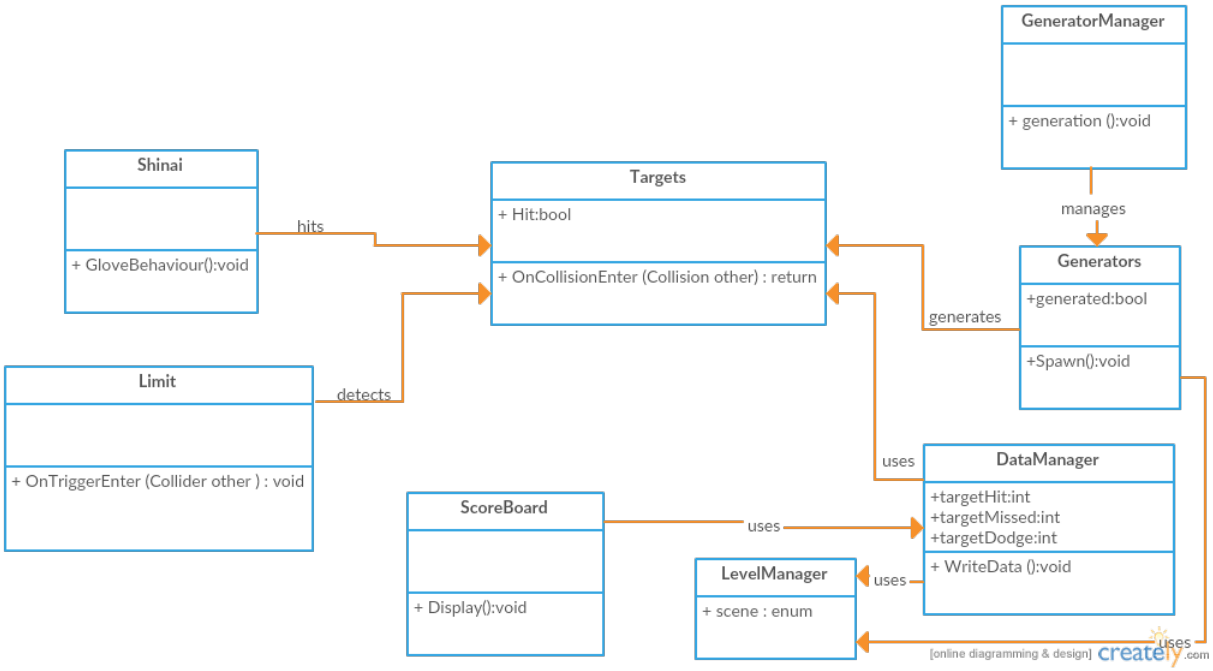


Figure 20: Class diagram for the Watermelon Exercise

6.2.2.2 Graphics

The 3D Models used are 6 watermelons with different size and a shelf as the visual object to let the users know where the targets are going to be generated.

Figure 21: Watermelons and shelf

6.2.2.3 Implementation

The watermelons will be spawned from 6 different generators, in each round 3 watermelons with different size are randomly chosen, the watermelon with medium size will be labeled as *target* and the other two as *dodge*. Furthermore, to know which watermelon should be labeled as *target* their *scales* are compared, and these will be spawned by 3 different randomly chosen generators. When the players slice a watermelon, the shinai checks if it is a *target* or *dodge* and based on that different sound feedback is displayed. Moreover, this has mesh-less cubes with tag *limit* to get rid of prefabs.

Additionally, the exercise is divided into rounds, each round generates a number of targets, and the spawn speed is determined by the spawn rate. The users can decide these before doing the exercise.

6.2.3 Explanation : Fruit Slice Exercise

This exercise focuses more on the physical aspect rather than cognitive; this is mainly testing and training visual-motor[5] abilities by asking the user to slice fruits that are spawned following some specific patterns.

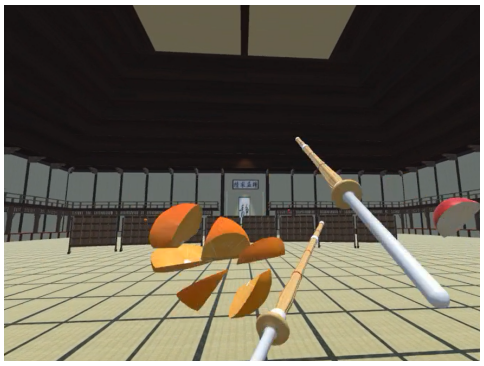


Figure 22: Player doing Fruit Slice Exercise

6.2.3.1 Class Structure

The class structure of this exercise is very similar to Section 6.2.2.1; the difference is that this does not generate targets with label *dodge*.

6.2.3.2 Graphics

The Graphics used for this exercise are many different kinds of fruits that are made from fragments and many shelves as the visual objects to tell the players where the targets are spawned.

[5] `https://en.wikipedia.org/wiki/Fine_motor_skill`

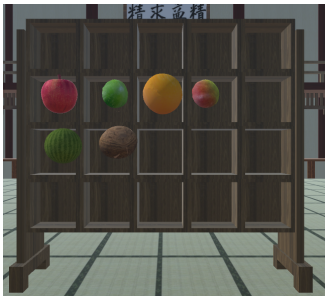


Figure 23: Fruits and shelf

6.2.3.3 Implementation

The fruits can be spawned from 48 different generators, 24 at the top 24 at the bottom, 9 basic patterns have been designed and shown in Figure 24. Furthermore, the spawn rate of fruits varies along the exercise between 0.5 seconds and 2 seconds so the players cannot predict, and to do that two different randoms were tried, the *Random* class provided by C# and the *log probability*, after comparing both the *log probability* is implemented since it provides less likelihood. Moreover, to guarantee that the pattern does not get any unexpected shapes, each pattern has a specific range of generators it can use.

Additionally, the duration of the exercise, which is controlled in seconds, is decided by the users before starting the exercise.

```
switch (Random.Range(0, 9))
{
    case 0: // one on the top of another
        spawn2TUD();
        break;
    case 1:// 2targets, one next to another
        Spawn2T();
        break;
    case 2: // 3 targets, one next to another
        Spawn3T();
        break;
    case 3: //4  targets, one next to another
        Spawn4T();
        break;
    case 4:// generate one randomly
        generators[Random.Range(0, generators.Length)].Shoot(target[Random.Range(0, target.Length)], 1);
        break;
    case 5:// 2 on the top of other 2
        spawn4TD();
        break;
    case 6: // 3 targets making a triangle
        spawnTriangular3();
        break;
    case 7: // 2 random targets
        generators[Random.Range(0, generators.Length)].Shoot(target[Random.Range(0, target.Length)], 1);
        generators[Random.Range(0, generators.Length)].Shoot(target[Random.Range(0, target.Length)], 1);
        break;
    case 8: // 4 targets making a triangle
        spawnTriangular4();
        break;
```

Figure 24: Code fragment showing the patterns for spawning fruits

6.2.4 Explanation : Dummy Exercise

This exercise is for training shoulders and arms, the players will be asked to do a kendo[6] training; they need to put the shinai over their head and land hits to specific parts of the dummy.

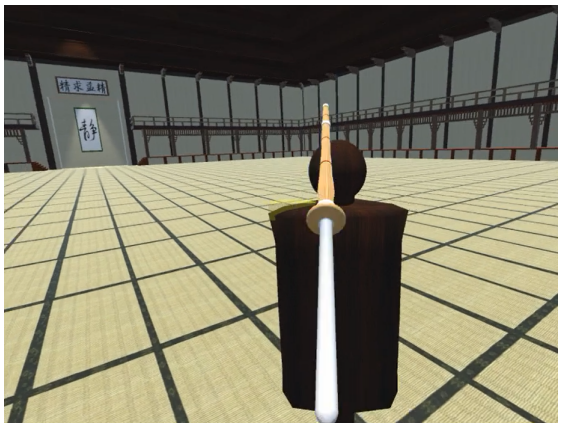


Figure 25: Player doing the Dummy Exercise

6.2.4.1 Class Structure

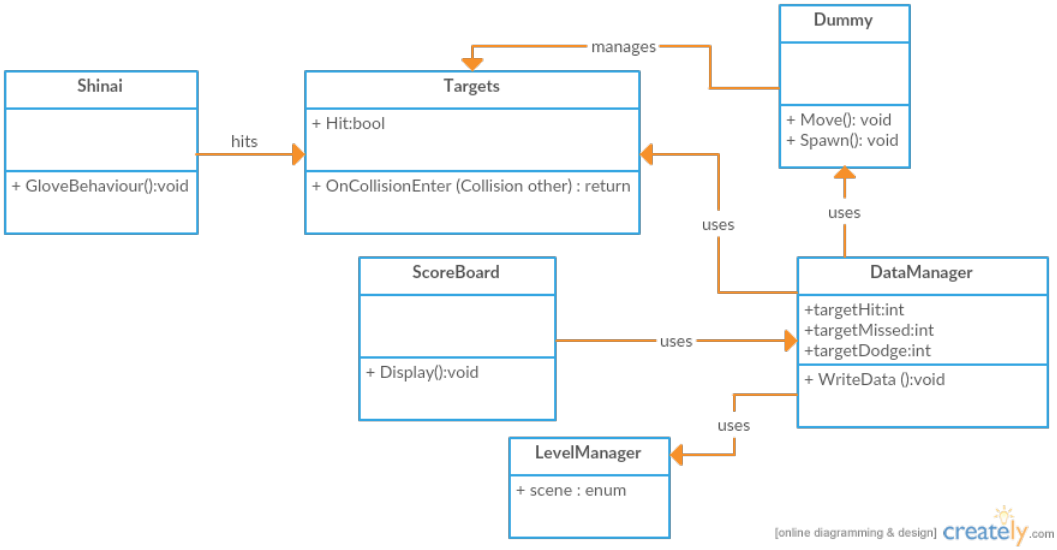


Figure 26: Class diagram of the Dummy Exercise

6.2.4.2 Graphics

The basic graphics have been covered in Section 6.2.1.2 and a dummy has been created for this exercise, and it is used as the visual object for the player to land hits on it.

[6] `https://en.wikipedia.org/wiki/Kendo`

Figure 27: Dummy model

6.2.4.3 Implementation

The users can land hits on the dummy's head, shoulders and dorsal; the part that the users need to hit with the shinai will become yellow. However, before telling the players which body part they have to hit; they need to do the 'ready pose', which is putting the shinai over their head, after that, an area will become yellow and interactive. Besides, to know if the players are in the 'ready position' or not we look for the *world coordinate* of the shinai and compare it to the *world coordinate* of the users' camera (this is like the users' head). Additionally, after each hit, the next body part is chosen randomly.

Moreover, the dummy moves in circle around the users, and it is always facing them; this is done by getting the direction of the players, which is obtained by subtracting the coordinates of the dummy with the players' coordinate, after that, this direction is normalized. Furthermore, to know how much the dummy needs to rotate in its Y axis to have its Z axis facing the players; we do the dot product of the dummy's forward direction (normalized Z axis of the dummy) with the normalized direction obtained previously. Then, we do the inverse cosine for the result of the dot product to get the angle and implement it as an Euler angle to the dummy, and the code is shown in Figure 28.

Furthermore, the movement is done to the right and left randomly for a period between 0.5 seconds and 5 seconds using the *log probability*, because if we make the dummy to move always in the same direction, the users will get dizzy since they need to follow it. Moreover, the move is very slow 0.3m/s in the X axis of the dummy.

Besides, the duration of this exercise is decided by the number of hits that players want to perform, which can be decided before starting.

```
if (right)
{
    if (tt < 0.03)
    {
        tt = tt + Time.fixedDeltaTime;
        this.transform.position += this.transform.right * (-0.3f) * Time.fixedDeltaTime;
    }
    else
    {
        transform.eulerAngles = new Vector3(0, (angle) + this.transform.eulerAngles.y, 0);
        this.transform.position += this.transform.right * (-0.3f) * Time.fixedDeltaTime;
        direction = (target.position - this.transform.position);
        normalizedDirection = direction.normalized;
        angle = Mathf.Acos(Vector3.Dot(this.transform.forward, normalizedDirection)) * Mathf.Rad2Deg;
    }
}
else if (left)
{
    if (tt < 0.03)
    {
        tt = tt + Time.fixedDeltaTime;
        this.transform.position += this.transform.right * (0.3f) * Time.fixedDeltaTime;
    }
    else
    {
        transform.eulerAngles = new Vector3(0, (-angle) + this.transform.eulerAngles.y, 0);
        this.transform.position += this.transform.right * (0.3f) * Time.fixedDeltaTime;
        direction = (target.position - this.transform.position);
        normalizedDirection = direction.normalized;
        angle = Mathf.Acos(Vector3.Dot(this.transform.forward, normalizedDirection)) * Mathf.Rad2Deg;
    }
```

Figure 28: Code fragment of the dummy's rotation and move

6.2.5 Data Collected

The performance the data that can be collected from the Watermelon Exercise are the numbers of the correct targets hit and the number of wrong targets hit. The Fruit Slice exercise generates information about the number of fruits sliced and missed. Finally, from the Dummy Exercise we can get the number of hits to be done, and the time it took to accomplish it.

6.3 Sphere Reaction

This category is created to test the players cognitive ability, mainly their reaction, and to test brain concussion; 3 different tests are used pro, anti and pro/anti-saccade, these can test the functionality of the player's frontal lobe[8]. Therefore, if the players have suffered damage on this part of the brain, they would make many errors.

Anti-saccade is asking the users to do a saccade[7] away from the stimulus. Pro-saccade asks the user to do a saccade to the stimulus. However, the saccade in these exercises instead of eye movement is going to be a touch to the touchpad.

[7] https://en.wikipedia.org/wiki/Saccade

6.3.1 Design

6.3.1.1 Functional Requirements

1. The way to select the sphere to be anti needs to be random.
   (a) The anti sphere should be red.
2. The way to select the sphere to be pro needs to be random.
   (a) The pro sphere should be green.
3. The touchpads should respond to a simple touch, not press to avoid movement.
4. The touchpads should be interacting with the spheres.
5. Sound feedback should be given based on the users' answer.
6. Reaction time and the number of correct and wrong answers should be recorded.

6.3.1.2 Graphics

The graphics made for this category are the two spheres in the figure below, the characters that are on the spheres mean wind(风) and thunder(雷); the inspiration came from the Japanese culture Fūjin (God of the wind) and Raijin (God of the thunder) because these two gods always appear together.

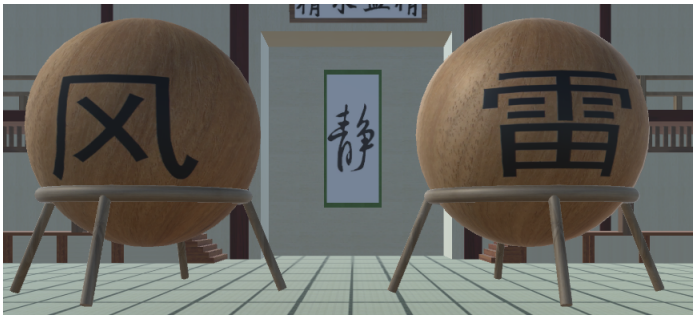


Figure 29: Wind sphere and thunder sphere

6.3.2 Explanation : Pro, Anti & Pro/Anti-Saccade

The exercises consist of 2 spherical objects, in the pro-saccade test the sphere will turn green, and when that happens the players need to touch the touchpad of the controller on the same side; therefore, if the right sphere becomes green we need to touch the touch-pad of the right controller. In the anti-saccade exercise the sphere will turn red, and when the sphere is red the users need to use the touchpad of the opposite side; hence, if the right sphere is red

we use the touchpad of the left controller. Finally, the pro/anti-saccade exercise is a mixture of the previous 2.

Figure 30: Anti-saccade

Figure 31: Pro-saccade

6.3.2.1 Class Structure

This is a class diagram covering the main features of the 3 exercises.

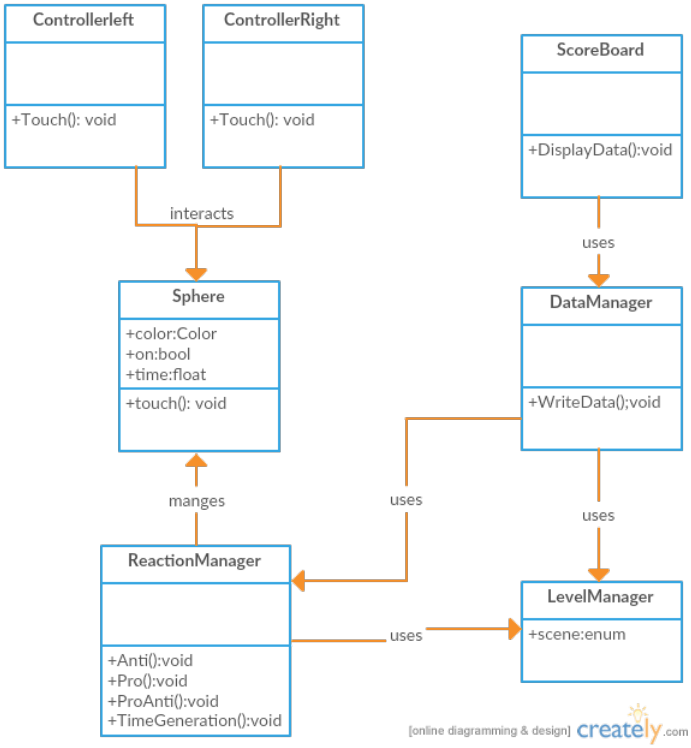


Figure 32: Class diagram of the Sphere Reaction Exercises

6.3.2.2 Graphics

Graphics for these three exercises have been covered in the Section 6.3.1.2.

6.3.2.3 Implementation

The implementation of the 3 exercises are very similar, all of them use a random variable to decide a sphere to be anti or pro, in the case of pro/anti-saccade, an additional random number is generated to choose a sphere and then decide if it is going to be anti or pro. Red stands for the sphere being anti and green means the sphere is pro; the spheres have a boolean variable *on*, which is used to mark which sphere needs to receive the saccade. Therefore, when the players touch the touchpad it will check if the corresponding sphere's *on* is true or not, if it is true the saccade is good if it not it is bad.

Furthermore, the time between each test in the exercise cannot be constant. Otherwise, the user might learn the pattern and cheat; therefore, the *log probability* is used making the tests harder to predict as Section 6.2.3.3. However, a time limit of 3 seconds is set, if the random generation generates 3 seconds or more, it regenerates a new time.

Moreover, the exercise is divided into rounds, each round asks the user to do a saccade, and the number of rounds can be decided before doing the exercise.

```
// pro/anti
for (int i = 0; i < rounds; i++)
{
    start = true;
    //nonageing random time between test
    TimerGeneration();
    yield return new WaitForSeconds(timer);
    //pro/anti targetchoice
    TargetChoice();
    yield return new WaitUntil(() => left.GetComponent<ReactionObject>().on == false
    && right.GetComponent<ReactionObject>().on == false);

    left.GetComponent<MeshRenderer>().material.color = new Color(Color.white.r, Color.white.g, Color.white.b, 0);
    right.GetComponent<MeshRenderer>().material.color = new Color(Color.white.r, Color.white.g, Color.white.b, 0);
    DataManager.recorded = false;
    start = false;
}
```

Figure 33: Code fragment of Pro/Anti Exercise

6.3.3 Data Collected

The data that can be collected from the 2 exercises are the number of good and bad saccade done, and the time taken for each saccade.

6.4 Drum

The exercises of this category are divided into two kinds, one for training and testing the users' reaction and another one consists of aerobic exercises for maintenance, fat burn or cardio training. The first one involves pro, anti and pro/anti-saccade. The second one has 6 drums that players need to hit with their drumstick following a sequence, and like that we try to achieve some beneficial fitness effect on the players.

6.4.1 Design

6.4.1.1 Functional Requirements

1. The drumstick should interact with the drums.
   (a) Feed back is given to the players when the drumstick hits the drum.
2. The drumstick should always be in the hands of the players even if a collision occurs.
3. The drums should show different color with different meanings.
4. The drums should not be affected by forces.
5. The indicators will show the same color as their corresponding drums.
6. The data of the exercises should be properly recorded.

6.4.1.2 Graphics

The graphics used for these exercises are drums as targets and drumsticks as the objects to interact with the drums.

Figure 34: Drum and drumstick

6.4.1.3 Implementation

The drumstick should always be fixed to the players' hands like in Section 6.2.1.3; they are the same the only difference between them is that one is shinai and another one is drumstick.

The drums detect collisions and they are fixed to some specific location, and cannot be moved when collisions are detected, so having a similar behavior like drumsticks.

```
void OnCollisionExit(Collision collision)
{
    drumstick.transform.localEulerAngles = originRotation;
    drumstick.transform.localPosition = origin;
    drumstick.gameObject.GetComponent<Rigidbody>().velocity = Vector3.zero;
    drumstick.gameObject.GetComponent<Rigidbody>().angularVelocity = Vector3.zero;

}
```

Figure 35: Code fragment of drumstick behavior

6.4.2 Explanation : Pro, Anti & Pro/Anti Saccade

The exercise is very similar to Section 6.3.2, but this time involving motor skills[8] like that we try to test and train the reaction for identifying cues, the response time [9], and to detect possible brain injury. Additionally, it has two drums changing positions every time, but here in order to interact with the drums we need to hit them with the drumstick; hence, if the drum is green we hit it but if the drum is red we hit the other one.

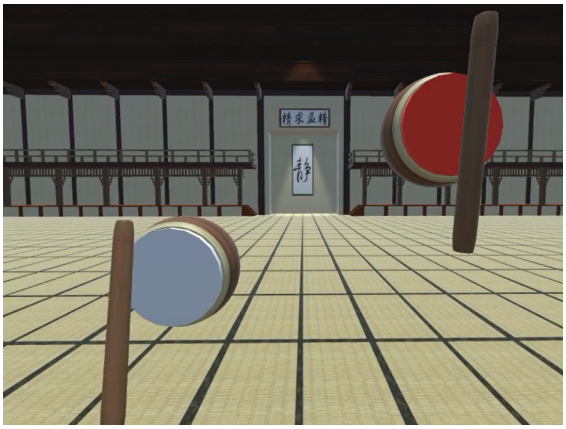


Figure 36: Player doing the Drum Anti-saccade Exercise

[8] https://en.wikipedia.org/wiki/Motor_skill

6.4.2.1 Class Structure

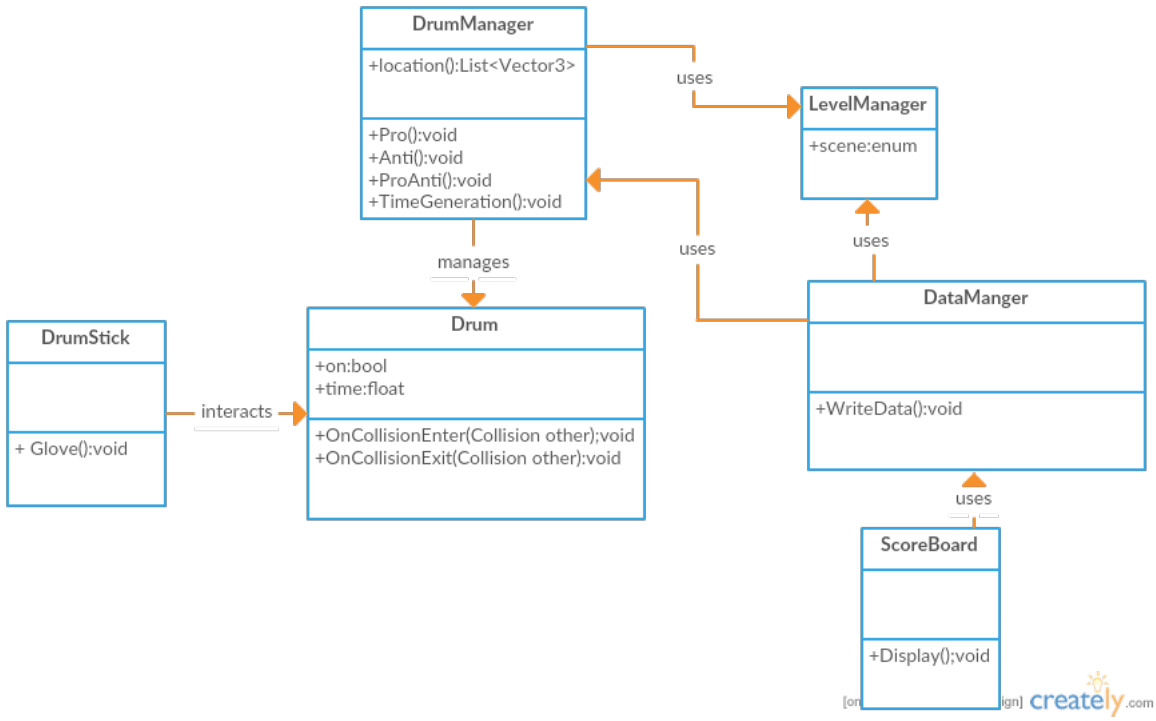


Figure 37: Class diagram of Pro,Anti and Pro/Anti Exercises

6.4.2.2 Gaphics

The graphics have been covered in section 6.4.1.2.

6.4.2.3 implementation

The implementation of this section is the same as Section 6.3.2.3; everything is the same except that the interaction instead of touching the touchpad of the controllers, it is hitting the drums with the drumsticks, and we are not checking if the drumstick that hits the target is the left or right one. Moreover, the locations of the drums are chosen randomly from a list containing 6 location.

This exercise is divided in the same way like Section 6.3.2.3.

6.4.3 Explanation : Simple & Hard Fitness Exercise

These two exercises are designed to keep the user in a better fit by achieving some beneficial aerobic exercise effect. The players will see 6 drums in different locations and at the center of them there are 6 indicators (non-interactive) representing the 6 drums to guide the players since the drums are big and close making them hard to see. The drums have colors, and the users need to evaluate them and do a sequence of actions.

The Simple Fitness Exercise has 2 colors, yellow indicating that the drum needs be hit by a single drumstick, and blue indicating where the next yellow is going to appear. The exercise at the beginning had only one color yellow, but this made the exercise too simple since the user did not need to think much. Hence, by giving an advance indication with the blue color the users are receiving an extra stimulus allowing them to be faster but also confusing them at the same time.

The Hard Fitness Exercise instead of 2 colors is having 4, it still has yellow and blue both keeping the same meaning; magenta and red have been added. Magenta means the drum needs to be hit with two drumsticks, and red indicates where the next magenta is going to appear. Moreover, adding 2 new colors have increased the possible combinations from 6^2 to 6^4 making it more entertaining and cognitively demanding, but since reaching drums with two drumsticks have been added; it is also increasing the physical demand asking the user to use more muscle in a single action.

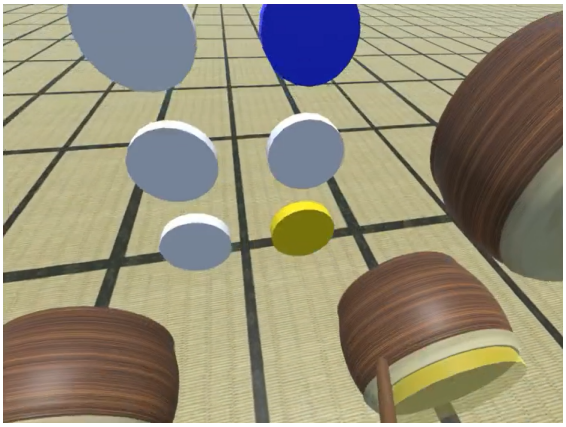


Figure 38: Drum Simple Fitness Exercise

6.4.3.1 Class Structure

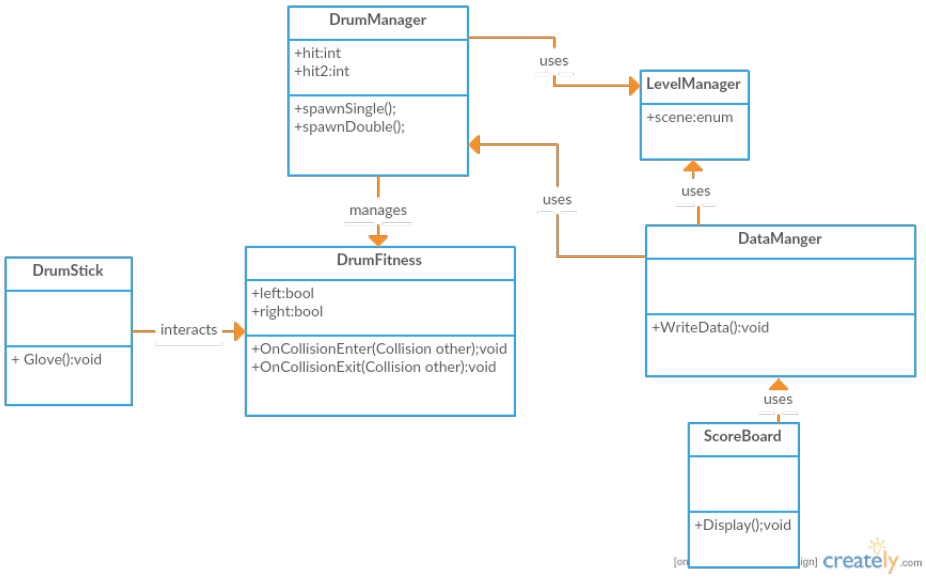


Figure 39: Class diagram for Fitness Exercise

6.4.3.2 Graphics

The graphics used in these exercises are the 6 drums, the drumsticks and the 6 indicators shown in Figure 34 and Figure 40.

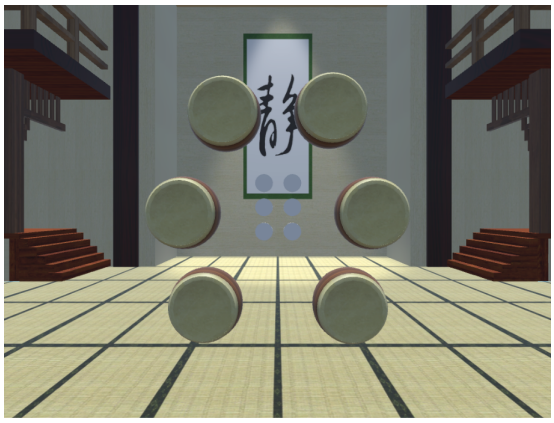


Figure 40: Drums and Indicators

6.4.3.3 Implementation

The center of the drums is adjust depending on the users' position by reading their position before starting; furthermore, in the simple version at the beginning 2 drums are chosen randomly, the first one is yellow and the second one is blue when the yellow one is hit the blue becomes yellow, and a new drum is randomly chosen to be blue. The hard version is similar to the simple version, but having 2 more colors red and magenta; moreover, it generates an extra random variable to decide if the drum is going be hit by single or double drumstick before choosing the drum.

These exercises are not done in rounds but controlled by time, and the user can choose the duration of it before starting.

```
if (Random.Range(0, 4) == 1 && LevelManager.scene != 14)
{
    newN = Random.Range(0, 6);
    roller();
    targets[newN].GetComponent<PunchingSideBehaviour>().bothHand = true;
    targets[newN].SetActive(true);
    targets[newN].GetComponent<MeshRenderer>().material.color = new Color(Color.red.r, Color.red.g, Color.red.b, 0.49f);
    targets[newN].GetComponent<PunchingSideBehaviour>().wait = true;
    targets[newN].GetComponent<PunchingSideBehaviour>().next = true;
    punched = false;
}
else
{
    newN = Random.Range(0, 6);
    roller();
    targets[newN].SetActive(true);
    targets[newN].GetComponent<MeshRenderer>().material.color = new Color(Color.blue.r, Color.blue.g, Color.blue.b, 0.49f);
    targets[newN].GetComponent<PunchingSideBehaviour>().wait = true;
    targets[newN].GetComponent<PunchingSideBehaviour>().next = true;
    punched = false;
}
```

Figure 41: Code fragment of Hard Fitness Exercise

6.4.4 Data Collected

From the saccade exercise we can collect the number of good and bad saccades and the response time of each saccade. The fitness activities generate the number of hits done with single and double hands during the exercises.

6.5 Trail Making Test

This category contains two exercises Trail Making Test A and B referred as simple and hard in this project, these are not only going to test players' cognitive and motor skills but also to detect any brain damage such as brain concussion.

6.5.1 Design

6.5.1.1 Functional Requirements

1. The first targets need to be shown in green.
2. Feedback based on the users' answer needs to be given.
3. The targets should disappear when the time limit is reached.
4. The center of the targets will read the position of the users' controller at the beginning for calibration.
   (a) The center of the targets will be represented by a yellow sphere.
5. The test should always start with the users' controller at the targets' center.
6. The data of the exercise should be properly recorded.

6.5.1.2 Graphics

The graphics used in these exercises are six spheres with different text meshes representing their value, a small sphere representing the controller, and a yellow sphere as the starting point.

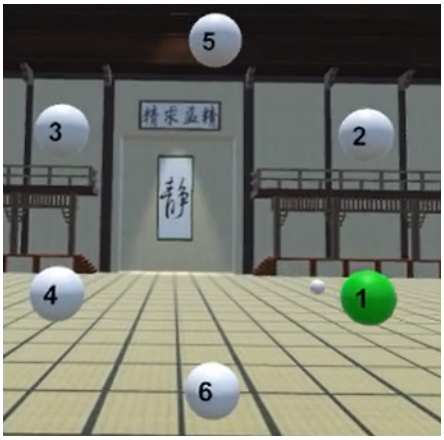


Figure 42: Trail Making Test Simple

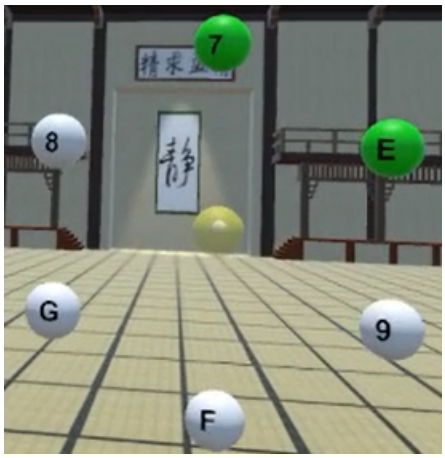


Figure 43: Trail Making Test Hard

6.5.2 Explanation : Trail Making Test Simple & Hard

These are two neuropsychological tests are used to verify the users visual search speed, memory, processing speed, and mental flexibility[10]. Furthermore, these are used to diagnose if the subject has suffered any brain damage[11], these tests can also be used for testing player's motor ability, but normally these are done with pen and paper, which is 2D, now we take them into a 3D environment using VR.

6.5.2.1 Class Structure

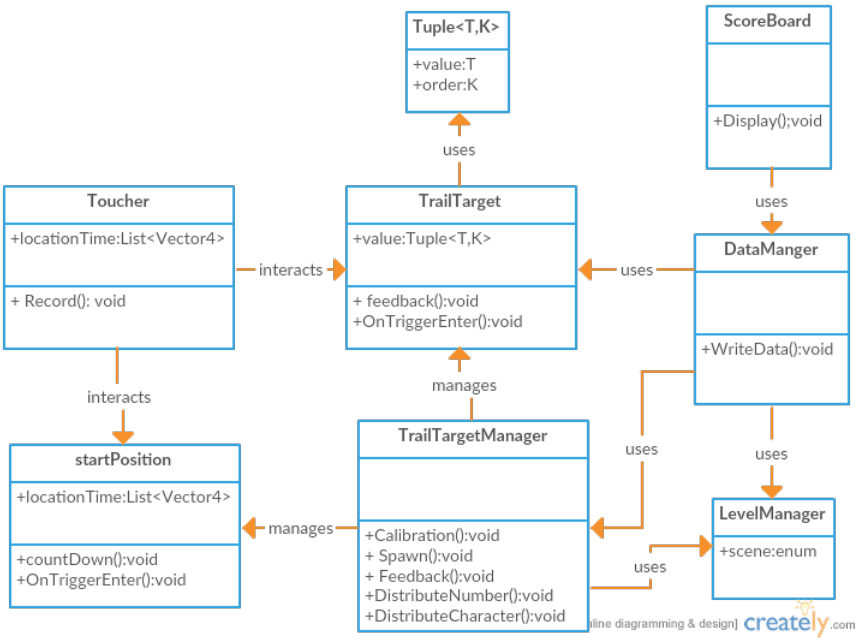


Figure 44: Class diagram of the TMT Exercise

6.5.2.2 Graphics

The graphics have been covered in Section 6.5.1.2

6.5.2.3 Implementation

The users will be asked to do a calibration at the beginning so they can perform the test in a comfortable position, after calibration the test begins. At the beginning of every test, the players are asked to put their controller in the center of the start point, which is a yellow sphere, for 3 seconds to make sure that the users always start from the center, after 3 seconds, 6 targets labeled with 6 different values appear. In the simple case these are 6 numbers, but in the hard case, these are 3 numbers and 3 characters. When the players do the simple version, they just need to reach the targets in ascending order starting from the smallest one which is always the green sphere. In the hard one the players need to do the same thing but alternating between numbers and characters(e.g., 1,A,2,B,3,C, etc.) the first number and character are shown in green.

Furthermore, to know which sphere is the first one in the generation process tuples are used. The tuples contain 2 values one is the order another one is the value, every time the controller touches a target it checks its order to know if it is the correct one. The reason for using tuples is that they match better to my logical understanding and it is more reusable

since they use generic types; hence, they can contain 2 integers or an integer and character or other combinations, and this adds flexibility for future extensions.

These exercises are divided into rounds each round has a time limit for completion, the number of rounds and the time limit can be decided by the users.

```
for (int i = 0; i < rounds; i++)
{
    GameObject start1 = Instantiate(startPoint, vectors[6].position, vectors[6].transform.rotation) as GameObject;
    start1.transform.parent = this.transform;
    start1.GetComponent<startPointBehaviour>().text = text;

    yield return new WaitUntil(() => startPointBehaviour.trialReady);
    InitialGeneratorSimple();
    trialEnd = false;

    yield return new WaitUntil(() => tt.allDone == true);
    tt.doneTime = tt.timer;

    yield return new WaitForSeconds(0.5f);
    tt.timer = 0;
    if (tt.correct == 6)
    {
        text.text = "Great Job!";
    }
    else
    {
        text.text = "Not Good Enough";
        tt.wrong.Play();
    }
    TrailToucher.order = 0;
    startPointBehaviour.trialReady = false;
    tt.once = false;
    indexEmpty();
    trialEnd = true;
    DataManager.recorded = false;
    yield return new WaitForSeconds(0.5f);
    Destroy(start1);
```

Figure 45: Code fragment of TrailTargetManager

6.5.3 Data Collected

The data that are collected from these exercises are the numbers of correct targets hit, the number of mistakes made, the position of the targets, the time taken to accomplish the round and the trajectory of the players' controller.

7 HCI

The HCI with VR is a bit different from the traditional HCI since the users are not interacting in 2D, but they are actually in the software surrounded by it. The GUI is designed in a way that can be used and understood by users who do not have background knowledge about it. Moreover, HCI with VR requires the users to explore the environment like in the real world to have a better understanding of it.

The GUI is designed into four parts. The first part is the main page showing six options representing the categories explained in Section 6. The second part contains the exercise of each category and will be shown to the users based on their choice from the first part. The third part is the configuration section, where the players can decide the setting of the chosen exercise from the second part. The fourth part is the actual exercise the users are going to do and a scoreboard is displayed at the end of it.

In the first part, which is the main page, the GUI interacts with the VR, but functionalities like typing names and paths for storing data are done with mouse and keyboard because of efficiency. Figure 46 shows the canvas for introducing the name and path, the software checks for validity of both, if any of them is not valid, the users cannot make any selection with the VR and will see a warning with the goggles. Furthermore, the headset does not display the canvas because it is not a 3D object.

Figure 46: Canvas for name and path

Once the name and path are valid, they will see their name at the top and can select the category they want to do from the options given by pointing it with the laser of the controller. Additionally, to help users know which option is being pointed the color of the pointed object becomes yellow, but for the option of leaving or stepping back; the lantern lights up, and a sound effect is displayed with every selection to tell the users that they are making a choice. Moreover, to confirm the choice the users need to press the trigger of the controller that has the laser. Furthermore, if the users need any help, instructions can always be found at the left-back through the entire software with some models of the controllers to help the users to identify the different parts of it.

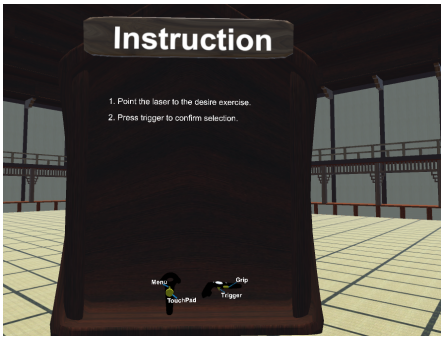


Figure 47: Main page instructions

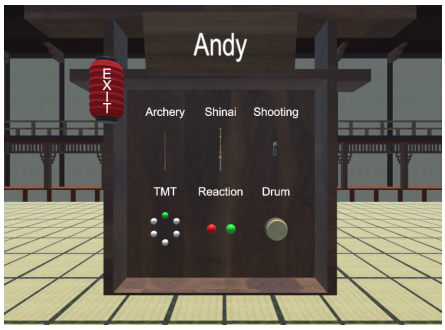


Figure 48: Main page category options

Once the users have selected the category they want to do, they are sent to the second part, which is the section containing the exercises of the selected category, they can choose the exercise they want to do by repeating the process above or by checking the instructions.

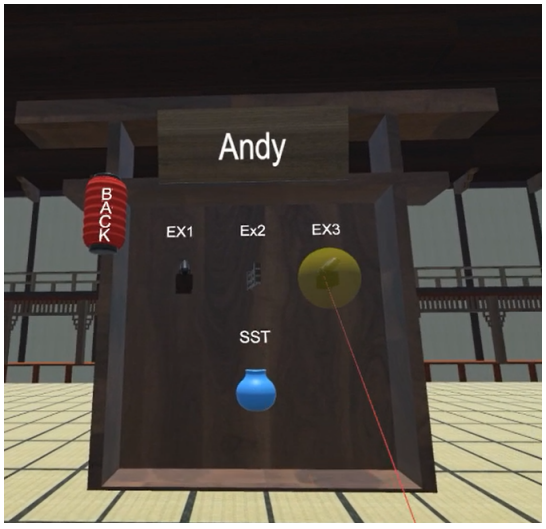


Figure 49: Exercise of the second part

After picking the exercise, they go to the third part, containing the settings for it. Here users can modify attributes of the exercises' setting by interacting with the '+' and '-' button, and using different buttons of the controller to add or subtract different values. The details of it are in the instructions and when the button pointed by the laser is gray; it means the button is not interactive because it has reached a minimum or maximum, but if it is yellow, it is interactive. When users are done with the setting they can start the exercise by pressing the button 'Go!' and a sound effect of 'Yooooo' is displayed. The reason of using this sound effect is because it is typically used in kabuki[9] to tell the audience that the show is going to start. Hence, trying to transmit a message of the exercise is starting and to give more oriental feeling.

[9] `https://en.wikipedia.org/wiki/Kabukil`

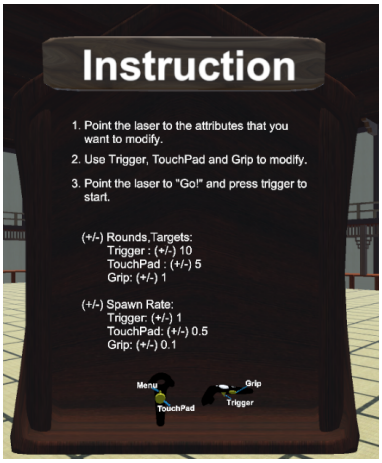


Figure 50: Setting's instructions

Figure 51: Setting's options

In the fourth part, which is the actual exercise, the users can check the instructions to know how to do the exercise, when the players finish the exercise a scoreboard is shown providing information about their performance and giving them the option to do it again or leave.

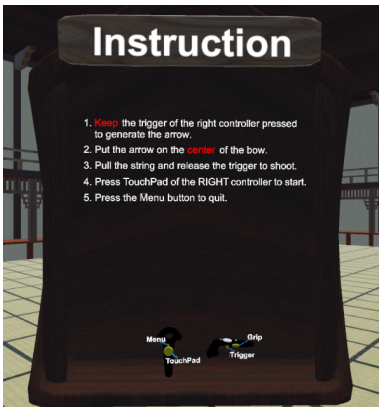


Figure 52: Exercise's instructions

Figure 53: Score board

Additionally, between the transition of different parts or exercises the *game view* becomes dark slowly and then becomes clear slowly, this is for avoiding sudden changes that might cause possible discomfort to the players. Moreover, Japanese background music is played during the whole software from part 1 to part 3, and each exercise has its Japanese music.

8 Testing

Black box testing was done before finishing the product; volunteers tried the software and tested its functionality. The software in general was working as expected but some problems were found.

First of all, volunteers reported that dynamic objects like the shinai, the drumstick, and the targets were blurry and had latency when they were moving. After some debugging, the problem was found, and the lights were causing the problem since they were a burden for the GPU. Therefore, some minor lights were removed; furthermore, the lighting type and its computation were changed from real time (lighting calculated while running the software) to mix(part of the lighting calculated beforehand and part of it while running).

Secondly, the software sometimes got stuck during the transition between different scenes crashing the GUI. It turns out that it was because the players were changing scenes too quickly and tried to access another scene while the current one was still loading. Hence, to solve this problem the players could not access to another scene or interact with the current one till the current scene is completely loaded, and this is done by adding a boolean variable that becomes true when the scene is fully loaded.

Lastly, some volunteers during the archery exercise experienced the problem of the arrow disappearing, but the bow was still acting as the arrow was still on it, for this problem no bug could be found in the code or in Unity. After some thinking a common point between the participants who experienced this problem was found, it was that they were not as tall as the participants who did not have that issue; thus, to solve this problem, calibration of the 'LightHouse' stations were done for each player and the problem did not happen again.

9 Project Management

To manage the progress, there were weekly tasks that had to be accomplished; moreover, meetings with my supervisors were arranged at least once every 2 weeks to keep him informed of the progress and discuss ideas. Besides, I was also meeting my co-supervisor 2-3 times every week to discuss about the ideas, implementation of the software, and its functionalities.

Week	Task
1-4	Learning Unity and working on the basic functionality of the shinai, shooting and Trail Making Test.
5	Learning Maya, working on graphics, and making the Dojo.
6	Learning Maya, still working on the dojo, and started to work on fruits and shooting targets.

7	Learning how to create a skeleton, and started to work on creating the bow model and bow exercise code.
8	Making the Drum exercise, and started to make the GUI and ran experiments for data collection.
9	More experiments and users' evaluations were done, and added sound effect and background music to the exercises.
10	More evaluations were done, finished debugging, and prepared demo.
11	Demo presentation and discussed the future use of the software.

Figure 54: Weekly tasks

As mentioned, the project used a bottom-up approach because this method is more flexible, even if all the exercises planned at the beginning could not be done in time, the software would still have a nice structure.

This approach allowed me to finish the core functionality first to ensure that the software got something to show at the end of the project. Once the basic functionalities were done, I started to learn Maya and made the graphics, which is an important part too since this is the visual stimulus of the software having a significant contribution to entertainment, user experience, and making it more realistic as Virtual Reality suggests. Finally, the GUI, this is still an important piece since one of the the goals is to create a software that can be used by anyone. However, I chose to work on it at the end because the GUI is not as important as the previous two, since the software can still be used without it, but it will be very hard for people who do not have knowledge about Unity and programming.

10 Result and Evaluation

The primary purpose of the software is to test and train the users' performance and to detect brain concussion. Nevertheless, because of ethical problems tests on concussed patients could not be done, but based on the studies of the methods implemented and discussions with my co-supervisor the software should be able to provide useful data for detecting brain concussion.

Another feature is that the software needs be user-friendly and entertaining while providing useful data.

10.1 Performance Data Analysis

Volunteers were recruited and were asked to perform the Archery Exercise with and without strobe, the Watermelon Exercise, the Trail Making Test Simple and Hard; Pro, Anti, Pro/Anti-saccade Exercise of the Sphere Reaction and Drum, and the Simple Fitness Exercise. The volunteers were asked to do the exercises twice, the first time in their first visit and the second time after 24h, performance data and NIRS data were collected from them.

10.1.1 Archery

The experiments had 8 rounds each round generating 10 targets. In the *hit* graph, the Y axis represents the number of shots hitting the targets and the X axis represents rounds. In the *missed* graph, the Y axis represents the number of shots missed and the X axis represents the rounds. Besides, the bar graph represents the total number of shots missed and hit.

Improvement is observed from the data collected from the Archery Exercise with and without strobe, the players hit more targets and missed less on the second day compared to the first day.

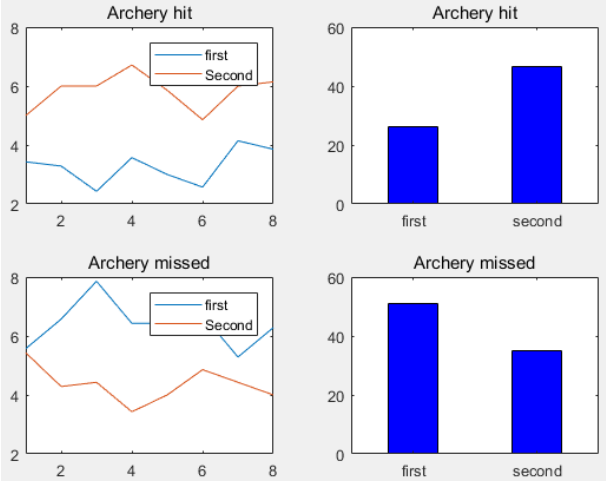


Figure 55: Graph of Archery without strobe

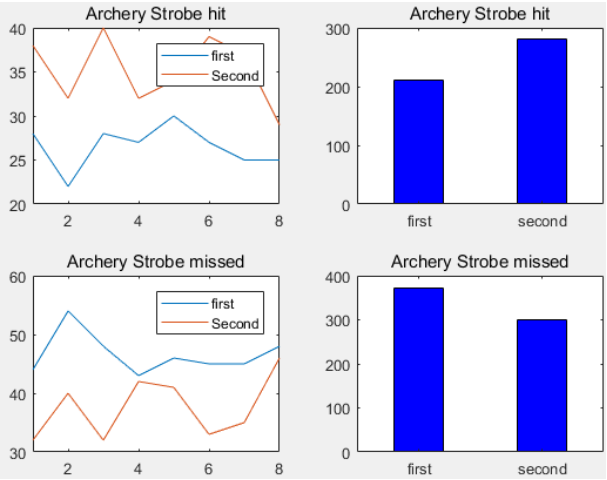


Figure 56: Graph of Archery with strobe

Furthermore, the locations of where the targets got hit were analyzed too, and an interesting thing was found; most of the player have the tendency to hit the targets when they are at 5-7 meters high. Although we do not know why the users have this tendency, we think the reason is the speed of the arrow.

The graphs below show the position of the targets getting destroyed in the X and Y axis.

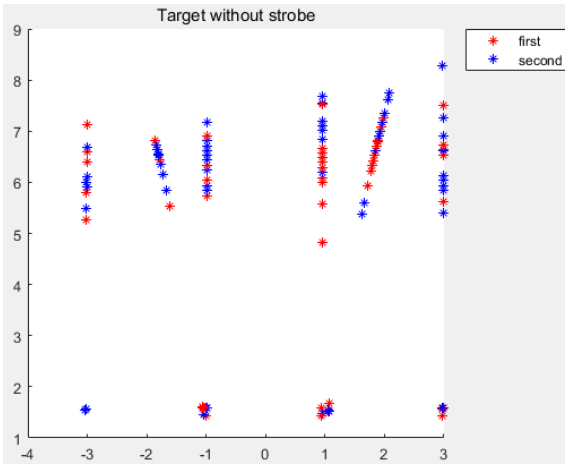


Figure 57: Graph of locations without strobe

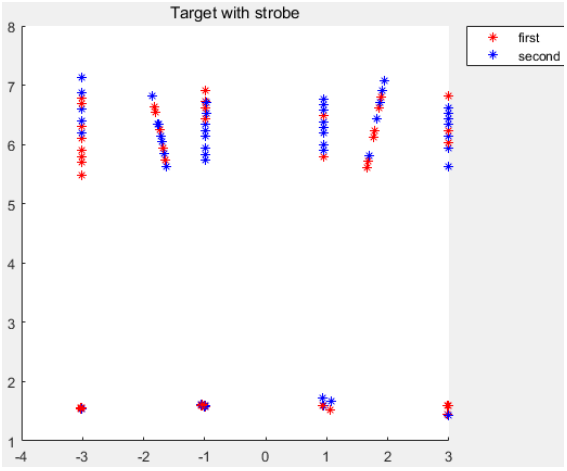


Figure 58: Graph of locations with strobe

10.1.2 Watermelon Exercise

The participants were asked to do 6 rounds, each round spawning 10 targets. The Y axis of the *correct hit* graph represents the number of *targets* hit, and in the *wrong hit* graph it represents the number of *dodge* hit. Besides, the X axis in both graphs represents the rounds, and the bar graphs represent the total number of *targets* and *dodge* hit.

Improvement in the watermelon exercise can be observed, the users did better in the second day compared to the first day.

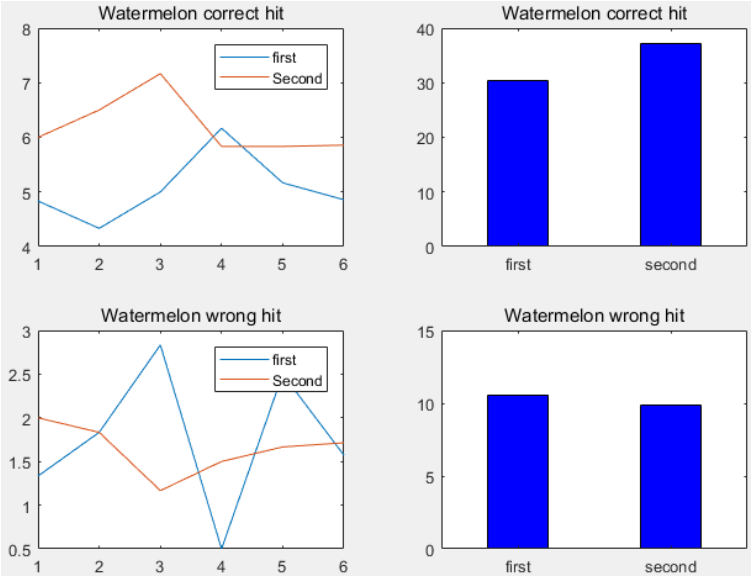


Figure 59: Graph of the Watermelon Exercise

10.1.3 Trail Making Test Simple & Hard

The participants were asked to do 54 rounds for both tests. The Y axis in the *correct* graph represents the number of correct answers; in the *mistake* graph it represents the number of mistakes, and in the *time* graph it represents the time in seconds taken to finish the round. The X axis in the 3 graphs represents the rounds. Furthermore, the bar graphs shows the total number of correct answers, mistakes and time taken in the first and second day.

The users reached more targets on the second day for both the simple and hard case, but at the same time they make more mistakes too, and it can also be observed that users became slightly faster.

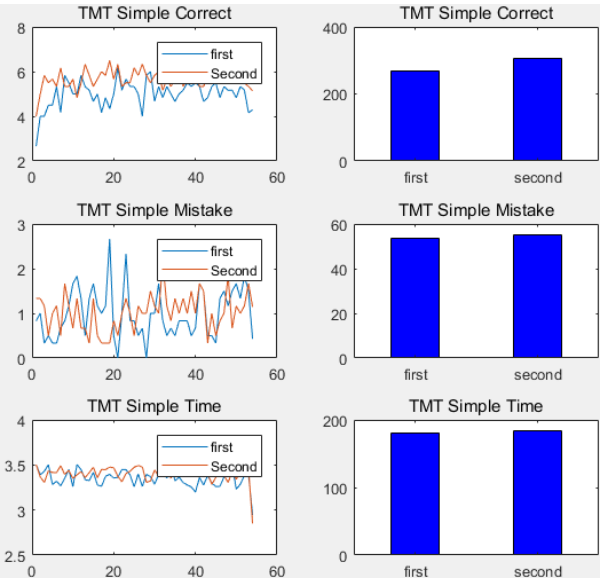


Figure 60: Graph of TMT Simple

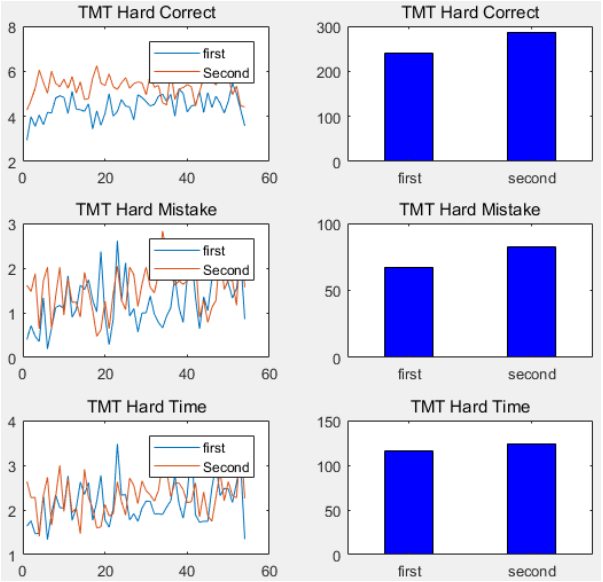


Figure 61: Graph of TMT Hard

10.1.4 Pro,Anti and Pro/Anti-Saccade of Drum & Sphere Reaction

The volunteers did 24 rounds for pro-saccade, 26 rounds for anti-saccade, and 46 rounds for pro/anti-saccade for both drum and sphere reaction. The Y axis of the *time* graph represents the time taken to do a saccade in seconds. In the *saccade* graph, the Y axis alternates between 2 values 0 and 1, 1 means that the saccade was good, and 0 means it was bad. Moreover, the X axis in both graphs represents the rounds. Additionally, the bar graphs represent the total reaction time and the number of correct saccades.

Data collected from the reaction and drum reaction exercise do not show improvement since the users were asked to do a saccade that does not involve complex cognitive or physical processing.

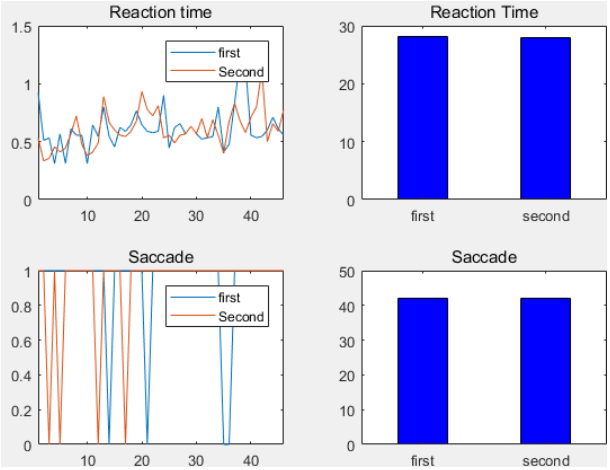


Figure 62: Graph of Pro/Anti Sphere

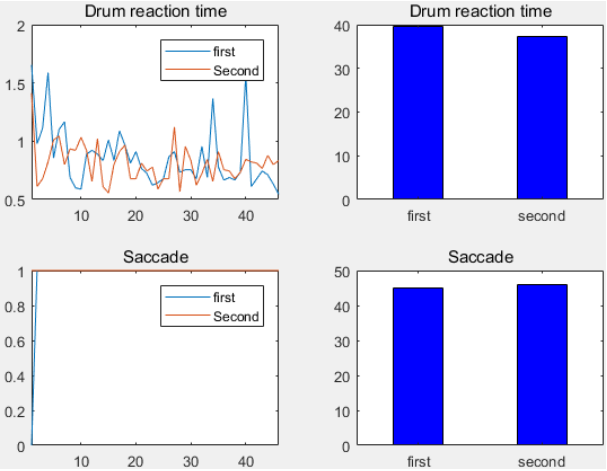


Figure 63: Graph of Pro/Anti Drum

10.1.5 Simple Fitness Exercise

This exercise, unlike the others, was only done on the first day since we want to observe if it increases the heart rate to achieve fitness goals. The volunteers were asked to rest for a minute before starting the exercise, then they were invited to do the activity for 3 minutes and then rest another minute. The heart rate was significantly increased during the exercise; hence, we can say that it does have some beneficial fitness effect on the players.

The Y axis of the graph below represents the heart rate and the X axis represents the time in minutes.

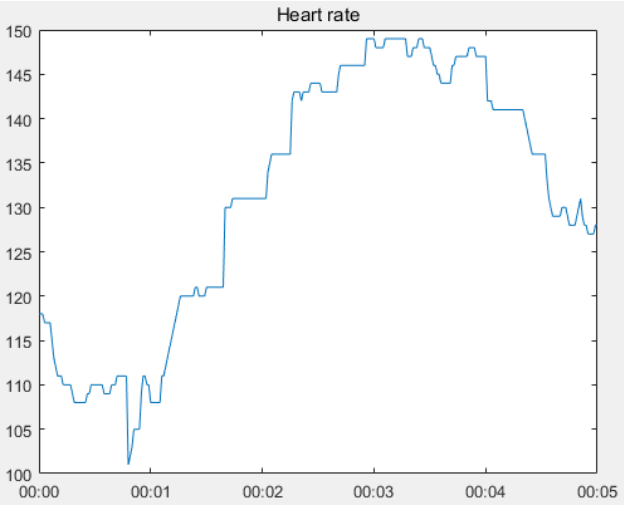


Figure 64: Graph of Simple Fitness Exercise

10.1.6 NIRS

Furthermore, the software can be used to collect NIRS data as Shown in Figure 65. However, I could only analyze the performance data since I could not understand this, but this can be given to a neurologist or physician for detailed analysis.

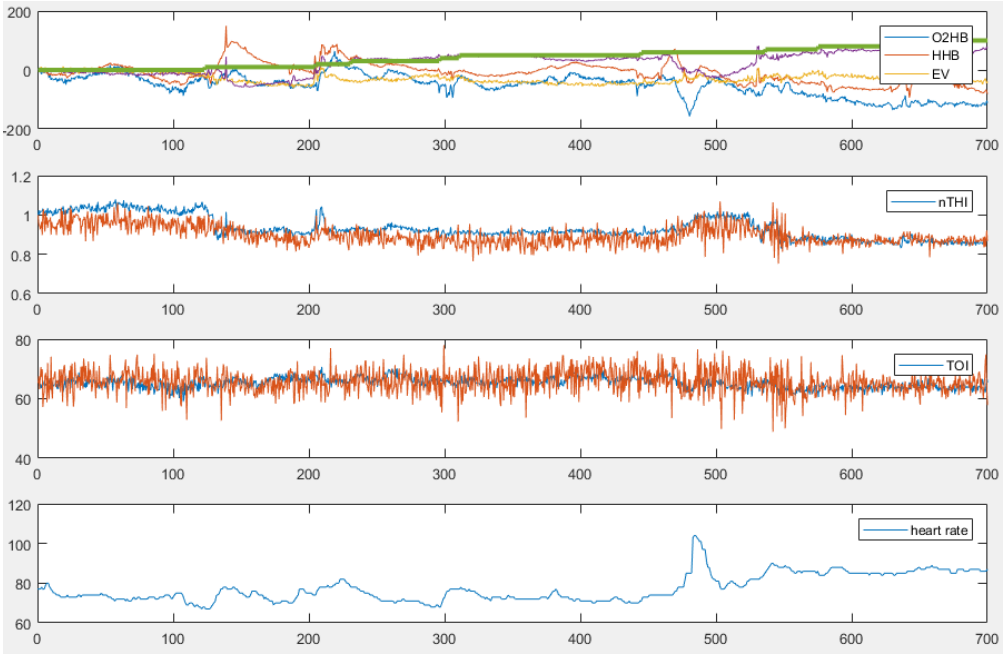


Figure 65: Graph of NIRS data

10.2 Evaluation

For evaluating the software volunteers were recruited to try the software, and after that, they filled a questionnaire shown in appendix B and each of them were interviewed.

The majority of them gave very positive feedback to the software as shown in Figure 66, they did not have much difficulty to guide themselves through it, but some of them suggested to change the location of the instructions and move it a bit forward. Furthermore, some players found the Trail Making Test very frustrating since the time limit for each round was too short; hence, the time limit was modified to a changeable attribute that can be adjusted by the players.

Moreover, some players suggested that the drum fitness exercises could follow the rhythm of some music or song, this is a nice idea, but because of the deadline, this feature could not be added to the software. The players liked the sound effect and haptic feedback[10] from the different interactions. Additionally, they also liked the graphics and music, and they

[10] `https://en.wikipedia.org/wiki/Haptic_technology`

think that it helps to get themselves more immersed in the VR world, and the music also contributes to creating a pleasant atmosphere for the exercises.

Furthermore, the software matches the expectation of the users in the neurological and motor control field since the software contains many different activities that can provide interesting data for evaluating different cognitive and physical abilities. Besides, it can be used to collect data for analyzing the brain behavior using the NIRS machine while doing different exercises.

	Yes			No	
First time using VR?	10			1	
	Very easy	Easy	Neither easy nor hard	Hard	Very hard
Difficulty to understand the instructions	0	7	3	1	0
Difficulty to guide yourself through the software	1	8	1	0	0
Difficulty to understand and interact with the exercises	1	8	1	0	0
	Very satisfied	Satisfied	Neither satisfied nor dissatisfied	Dissatisfied	Very dissatisfied
Satisfaction with the responsiveness	6	5	0	0	0
Satisfaction with the game feedback	6	5	0	0	0
Satisfaction with the score feedback	0	9	2	0	0
	Very Entertaining	Entertaining	Neither Entertaining nor unentertaining	Unentertaining	Very unentertaining
Archery without strobe	8	3	0	0	0
Archery with strobe	8	3	0	0	0
Shooting without strobe	5	6	0	0	0
Shooting with strobe	5	6	0	0	0
Trail Making Test	0	2	9	0	0
Reaction sphere	0	0	3	8	0
Reaction drum	0	1	5	5	0
Drum fitness	3	8	0	0	0
Shinai	8	3	0	0	0
	Very satisfied	Satisfied	Neither satisfied nor dissatisfied	Dissatisfied	Very dissatisfied
Satisfaction with the software in general	9	2	0	0	0

Figure 66: Questionnaire result

11 Impact

After analyzing the results from the collected data and evaluating the volunteers' feedback, we could say the software has met and exceeded the expectations set at the beginning. Hence, it is going to be taken to the British Neuroscience Festival[11] for demonstrations. Furthermore, it is also going to be used by third-year undergraduate students of the School of Sport, Exercise and Rehabilitation Sciences to do experiments and write their dissertation.

12 Discussion

The software has all the exercises that were planned at the beginning, and it does generate interesting performance data. However, collecting NIRS data was not in my plan when designing the project, it was a suggestion made by my co-supervisor who found the software suitable for collecting NIRS data. The idea was to collect the data and show it to a neurologist, we did some analysis of the collected NIRS data at the beginning, and they seemed good, but due to time and our schedules, no further analysis was done.

Furthermore, if I had more time I would like to see more detailed analysis of the NIRS data collected from my software, and do more experiments on the archery exercise to understand why the players have the tendency to shoot the targets at 5-7m high. Additionally, I would like to make the fitness exercise to follow the rhythm of some music or song as suggested by one of my volunteers. Besides, the software only has exercises designed for upper limbs, it does not have activities that can interact or test the lower limbs due to hardware restrictions since this is a technology in a very early stage, and there are no device on the market that can be used to detect the movement or balance of the lower limbs with VR. Nevertheless, some corporations are working on it, and once those devices are on the market, it would be nice to combine them and make exercises involving the 4 limbs to perform more demanding tasks and collect data that cannot be collected with the current technology using the VR.

13 Conclusion

The software is capable of doing some exercises and can produce performance and NIRS data; it can train and test the users through some exercises, as observed from the data collected, the players did improve in some activities and in the fitness exercise an increase in heart rate was observed. Thus, the goal of testing and training the users have been achieved, so the possibility for the software to be used for health recovery is very high

[11] `https://www.bna.org.uk/meetings/bna2017/`

However, brain concussion detection could not be tested since we could not do experiments on concussed patients due to ethical problems. Even though I cannot proof that the software can detect brain concussion, but based on the results of what other researchers did with the Trail Making Test and Saccade Test the software should be capable of providing useful data about it. Additionally, the software should provide more accurate data about brain injury than the tests performed on the pitch.

Furthermore, the feedback received from the questionnaires and interviews shows that most of the users liked the software and found it entertaining which is one of the primary goals too.

Hence, the software is quite complete and fruitful since it has met the requirements and contains all the exercise that were thought at the beginning. Moreover, it is going to be taken to the British Neuroscience Festival and going to be used by third-year undergraduate students of the School of Sport, Exercise and Rehabilitation Sciences to do experiments and write their dissertation, which can be considered as a recognition of the achievements of this project.

References

[1] Vive.com. (2017) *VIVE^{TM} United Kingdom.* Available at: https://www.vive.com/uk/product/ (Accessed: 20 Mar 2017).

[2] Navajas, E.(2016) *HTC Vive*. Available at: https://voltaico.lavozdegalicia.es/2016/12/htc-vive-2-4k-sin-cables-no-ces-2017/ (Accessed: 20 mar 2017).

[3] Autodesk.co.uk. (2017) *Maya — Computer Animation & Modelling Software — Autodesk*. Available at: http://www.autodesk.co.uk/products/maya/overview (Accessed: 20 Mar 2017).

[4] Unity. (2017) *Unity - Game Engine*. Available at: https://unity3d.com/ (Accessed: 20 Mar 2017).

[5] Haberstroh,T. (2016) *Strobe-light training: From Michael Jordan to Kawhi Leonard*. Available at: http://www.espn.co.uk/nba/story/_/id/18002545/kawhi-leonard-strobe-light-training-nba (Accessed: 23 Mar 2017).

[6] Smith, Q.T., and Mitroff, S.R. (2012) 'Stroboscopic Training Enhances Anticipatory Timing', *International Journal of Exercise Science*, 5(4), pp.344-353. *PMCID: PMC4738880*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4738880/(Accessed: 24 Mar 2017).

[7] Appelbaum, L., Cain, M., Schroeder, J., Darling, E., and Mitroff, S. (2012) 'Stroboscopic visual training improves information encoding in short-term memory', *Attention, Perception, & Psychophysics*, 74(8), pp.1681-1691.

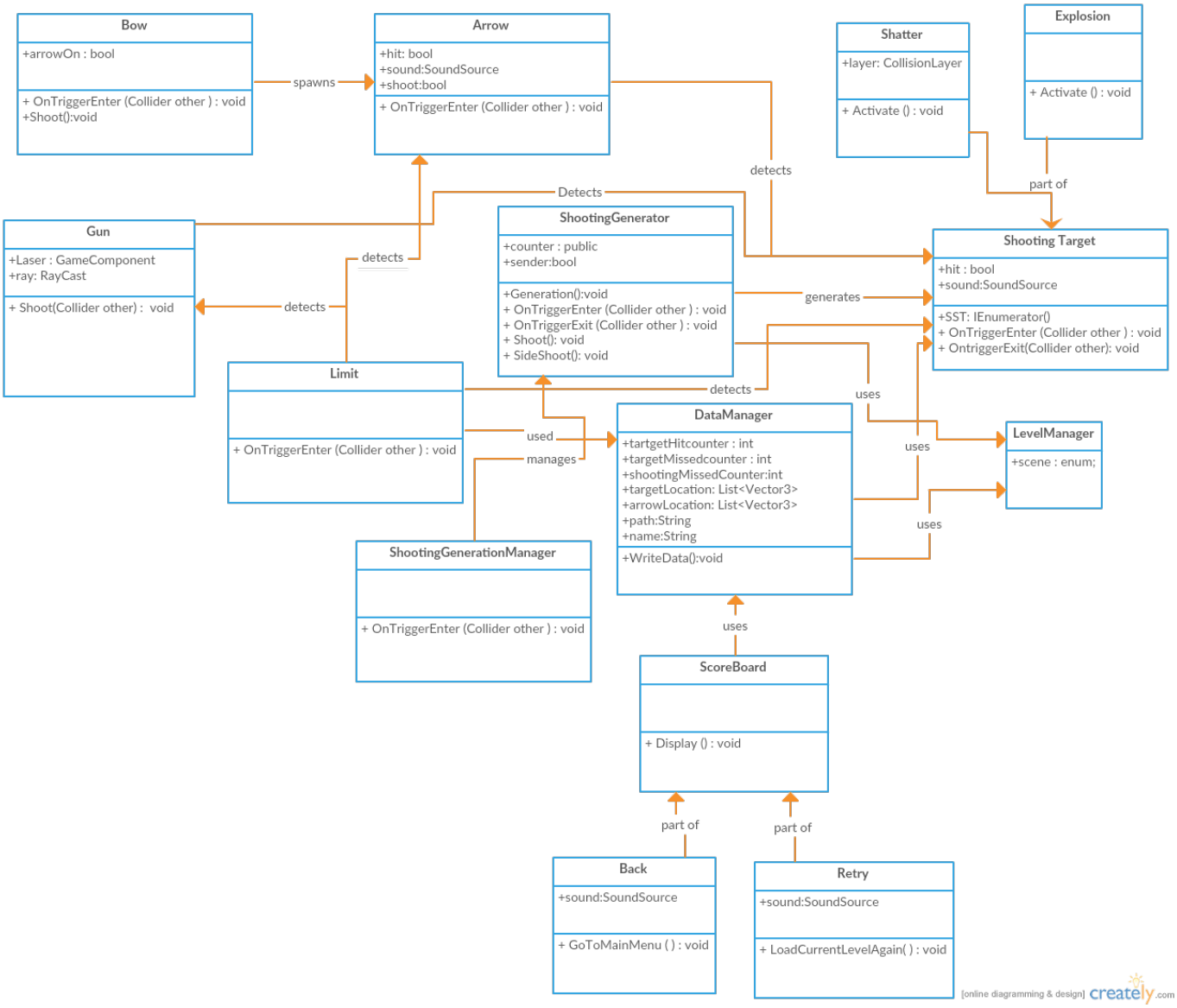
[8] Levy, D.L., Mendell, R.N., and Holzman, P.S. (2004) 'The antisaccade task and neuropsychological tests of prefrontalcortical integrity in schizophrenia: empirical findings and interpretativeconsiderations', *World Psychiatry*, 3(1), pp.32-40. *PMCID: PMC1414662*. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1414662/ (Accessed: 23 Mar 2017).

[9] Robertson, E. (2007) 'The Serial Reaction Time Task: Implicit Motor Skill Learning?', *Journal of Neuroscience*, 27(38), pp.10073-10075.

[10] Tombaugh, T. (2004) 'Trail Making Test A and B: Normative data stratified by age and education', *Archives of Clinical Neuropsychology*, 19(2), pp.203-214.

[11] Reitan, R. (1958) 'VALIDITY OF THE TRAIL MAKING TEST AS AN INDICATOR OF ORGANIC BRAIN DAMAGE.', *Perceptual and Motor Skills*, 8(7), p.271-276.

A Class Diagram of the Shelf Exercise



B Questionnaire

1. Is this your first time using a VR software?

- ◯ Yes
- ◯ No

2. How easy was for you to...

	Very Easy	Easy	Neither easy nor hard	Hard	Very hard
Understand the instructions	◯	◯	◯	◯	◯
Guide yourself through the software	◯	◯	◯	◯	◯
Understand and interact with the exercises	◯	◯	◯	◯	◯

3. How satisfied are you with...

	Very satisfied	Satisfied	Neither satisfied nor dissatisfied	Dissatified	Very dissatisfied
The responsiveness of the game(lags, delay)	◯	◯	◯	◯	◯
The game feedback (sound effect,vibration)	◯	◯	◯	◯	◯
The feedback of the Score board	◯	◯	◯	◯	◯

4. How Entertaining are the following exercises

	Very Entertaining	Entertaining	Neither Entertaining nor unentertaining	unentertaining	Very unentertaining
Archery without strobe	◯	◯	◯	◯	◯
Archery with strobe	◯	◯	◯	◯	◯
Shooting without strobe	◯	◯	◯	◯	◯
Shooting with strobe	◯	◯	◯	◯	◯
Trail Making Test	◯	◯	◯	◯	◯
Reaction sphere	◯	◯	◯	◯	◯
Reaction Drum	◯	◯	◯	◯	◯
Drum Fitness	◯	◯	◯	◯	◯
Shinai	◯	◯	◯	◯	◯

5. How satisfied are you with the software in general?

6. Is there anything else you would like to say about the software?

C Structure of the .zip file

The .zip file contains the things listed below and the software needs a PC with at least a NVIDIA GeForce GTX970/AMD 290 or equivalent, Unity 5.5.0f or above, and a HTC Vive:

- One electronic copy of the report (file/FinalYearReport.pdf).
- One electronic copy of the proposal written after talking to my co-supervisor and got permission to use the HTC Vive (file/Proposal.pdf).
- One .txt file containing the repository (file/Repository.txt).
- A video of a player using the software (file/PlayerUsingSoftware.mp4).
- A directory called *Test Center* containing the software(file/TestCenter).
- A directory called *Assets* containing all the materials, models and files used to make the software (file/TestCenter/Assets).
- A directory called *Scripts* containing all the codes (file/TestCenter/Assets/Scripts).
- A directory called *Models* containing all the models (file/TestCenter/Assets/Models).
- A directory called *Prefabs* containing all the prefabs (file/TestCenter/Assets/Prefabs).
- To run the software please execute *UIMain* (file/TestCenter/Assets/UIMain.unity).