Technion, Israel Institute of Technology
234329 – A Project in Image Processing

Blackjack**VR**

Final Report

By **Ofek Gutman**, **Eduardo Abramoff**
and **Guy Lecker**

Supervised by **Yaron Honen** and **Boaz Sternfeld**
Augmented Virtual Reality Laboratory
Powered by CGGC and GIP Labs

Table of Contents

1. Introduction and Abstract

Virtual reality has taken the world by a storm, bringing technological capabilities that open virtual worlds in the grasp of our hands, enabling us experiences that once we could only dream of. We wished to join the VR revolution uniquely, enabling a full virtual experience that transfers your mind into the virtual world.

This is when the **BlackjackVR** idea was born. With our love for the 250 years old game, we wanted to bring it into the virtual world, with a unique experience that was never done before. We took the Oculus Rift virtual reality headset and the Leap Motion Controller for bringing the game from the casino right onto your living room desk. So, **BlackjackVR** at its core is the same good old card game that is common in every casino, but it is not just about the game, it is about the experience you can live from the comfort of your home.

With time, we truly understand we have something unique in our hands as sitting in a casino became a rare luxury these days. The coronavirus pandemic hit us in every aspect of our lives. People could not meet with their families, businesses were shut down, and many of our free time activities were forbidden as well. Restaurants, movie theaters, and casinos were all closed, and we think virtual reality became an opportunity to fill the void.

We built **BlackjackVR** with gameplay and user experience to be both the highest priority in our minds. As such, we made sure the player's hand movements were smoothly recognized and we designed the casino room as close to reality as we could. With meshes of casino tables, chairs, and roulettes we have enriched the player's experience so once he put on the Oculus Rift and connects the Leap Motion Controller, the atmosphere of a casino will be truly vivid in his mind.

The game itself was written in C# and implemented in Unity. We used both Oculus Rift and Leap Motion unity packages to interact with the devices.

Our hand movement recognition algorithm was first researched in C++ and was built one step at a time until we got the satisfying results we will later show. It is a numeric algorithm, recognizing movements using recordings matrices with L2 distance and DTW. Once we got the final algorithm, we implemented it into the game in Unity.

The final product lived up to our standards and expectations, and we think it can easily be developed into a multiplayer game so one can maximize the casino experience from the comfort of his home.

We believe the hand recognition can be upgraded to a deep learning neural network so perhaps it can be more generic and less prone to the leap motion controller usage limitations we will discuss in this report.

2. System Description

*In short, **BlackjackVR** is a PC game written in C# and implemented in Unity, our chosen graphic engine. It is using an Oculus Rift headset for the virtual reality aspect of the game and a Leap Motion Controller to elevate the gameplay experience with hand movement recognition. We will further explain here about these components.*

2.1. Unity

Unity is a cross-platform graphics engine developed by Unity Technologies and used to develop video games for computers, consoles, smartphones, and websites.

Unity uses C# as the programming language of the application programming interface.

We used Unity as the engine and soul of our game. The main reason was due to both the Oculus Rift headset and the Leap Motion camera have unity-supported packages. Moreover, Unity's large community and creators provide a variety of tutorials we used to learn the software and meshes to download from the asset store.

The real-time 3D objects and scripts were simple to create and write while enabling rich performance and experience to us both as developers and as players.

2.1.1. Unity Logo

2.2. Oculus Rift Headset

Oculus Rift is a line of virtual reality headsets developed and manufactured by Oculus VR. With its highly respected market impact, oculus rift became a name in the industry regarding everything VR-related.

We gladly received Oculus Rift from the lab and with the Unity supported SDK we implemented it into the **BlackjackVR** game. With it, we achieved the virtual world filling – being inside a virtual casino.

2.2.1. Oculus Rift Headset

2.3. Leap Motion Controller

Leap Motion Controller is a computer hardware sensor device that supports hand and finger motions as input, analogous to a mouse, but requires no hand contact or touching. It is developed by Ultraleap (Ultraleap acquired Leap Motion in 2019).

The controller we were provided by the lab was designed for hand tracking in virtual reality, a perfect fit for our **BlackjackVR** game. We used the Leap Motion Controller for researching and developing our hand movements tracking algorithm and of course, for the gameplay of **BlackjackVR**.

2.3.1. Leap Motion Controller

3. Description of Execution

We will describe our development journey step by step, dividing it into three major categories. First will be researching and developing our hand movement recognition algorithm. Secondly, we will discuss the implementation and development of the game in Unity. And Lastly, the connection of the Leap Motion Controller, the algorithm, and the Oculus Rift headset to the game.

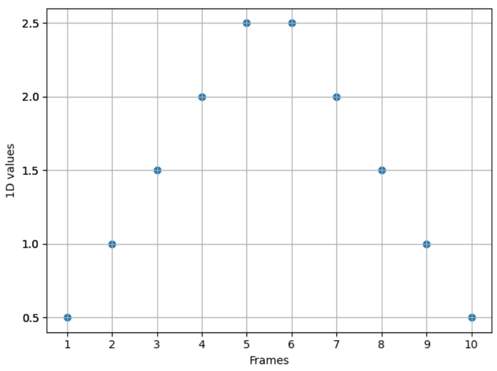
3.1. Researching the Hand Movement Recognition Algorithm

3.1.1. The Algorithm Base

The algorithm's basic idea is straightforward, and we will demonstrate our thought process thinking of it.

We wished to recognize hand movements and we knew we had the coordinates of the hand in the world as part of the Leap Motion SDK. We thought of a movement as a series of coordinates values changing in time – frames captured by the Leap Motion Controller. So, the first idea that came into our minds and the base of this entire algorithm is to compare the coordinates values captured in real-time to a pattern we know the movement should look like – later will be called the movement matrix.

Let us think of the hand as a simple one-dimension point. If this point from its current location was to move right, we know that for each frame we capture the point's value should increase (assuming right is the plus side of the axis) and if it would have moved left, the values would decrease. In the example below we can see in each frame that our hand is moving right, holds for a little, and goes back to the left where it came from:

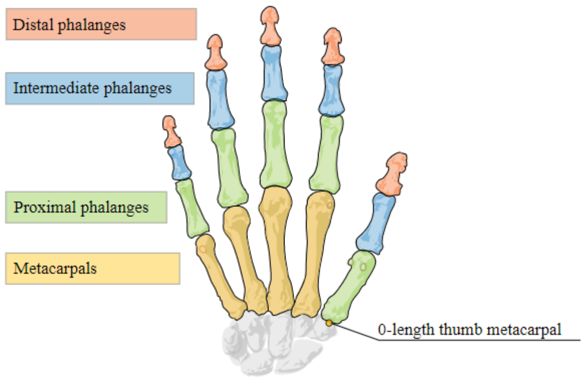


3.1.1.1. Simplification of a movement tracking

If we were to recognize this movement, we would have compared a group of captured live stream values from the leap motion camera to this pattern of values. This pattern is our movement vector, a vector where each element is the point value at a certain frame.

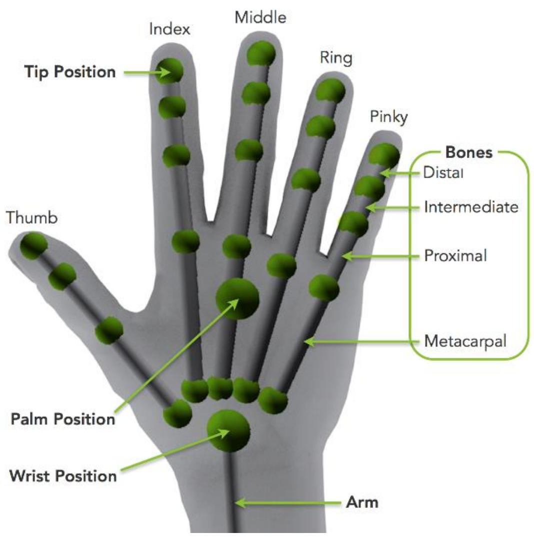
To recognize the movement, we need to measure how close the movement that was captured in the live stream to our movement vector. We calculate the distance of each point in the stream to the corresponding point in the movement vector at a specific frame. We used a simple L2 metric (without calculating the root for making the calculation faster) and some threshold value for this assessment. If the movement captured is close enough to the movement vector, we assume the movement is recognized.

Now, let us go back to reality, our hand is not a single point but a set of points that are being tracked by the Leap Motion Controller. Each point in the hand is represented as a 3D vector for their respected axis values (x, y, z). The available points of the hand object being tracked by the Leap Motion Controller are split by the bones structure of the human hand:



3.1.1.2. Leap Motion Controller Hand Structure

Between each bone, there is a point being tracked alongside the wrist and palm:



3.1.1.3. Leap Motion Controller Hand Hierarchy

We will declare two sets:

- $Fingers$ set: Thumb (T), Index (I), Middle (M), Ring (R) and Pinky (P).
- $Bones$ set: Distal (D), Intermediate (I), Proximal (P) and Metacarpal (M).

For our movements, we track the Palm (Pm) and all the fingers in all their bone's points ($Fingers \times Bones$). Overall, we track in our recognition algorithm 20 points, each point is a vector of 3:

$$\underbrace{\left(\left(\underbrace{\underbrace{5}_{|\{T,I,M,R,P\}|}}_{Fingers} \cdot \underbrace{\underbrace{4}_{|\{D,I,P,M\}|}}_{Bones}\right) + \underbrace{1}_{|\{Palm\}|}\right)}_{Tracked\ Points} \cdot \underbrace{\underbrace{3}_{|(x,y,z)|}}_{3D\ Sapce} = 63$$

So, 63 values in total are being recorded for each frame.

As such, our movement vector from the last example is now a movement matrix, with 63 columns for every value we track. The rows will be the recorded frames:

	Tracked Points																	
	Thumb Distal			Thump Intermediate			Thumb Proximal			Thumb Metacarpal			…			Palm		
Frame	T_{D_x}	T_{D_y}	T_{D_z}	T_{I_x}	T_{I_y}	T_{I_z}	T_{P_x}	T_{P_y}	T_{P_z}	T_{M_x}	T_{M_y}	T_{M_z}	…	…	…	Pm_x	Pm_y	Pm_z
1	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
3	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

3.1.1.4. A Movement Matrix

We should mention that for each of the cameras we used (will soon be discussed) the available points and hand object in general were different. We chose the cover the Leap Motion Controller Hand object as this is what we used eventually. With this said, we can say although the pointes were different, the idea was the always same.

3.1.2. Intel RealSense D435

With the algorithm's idea in mind, we wanted to test it out using the camera we got from the lab – the Intel RealSense D435 depth camera.

We downloaded the camera's SDK and with Visual Studio we tried to open some of the examples to run. It was easy to interface with the camera and visual studio, but we discovered the Hand Tracking Module is not included in the free SDK, and for that reason, we returned the camera to get a new one.

3.1.2.1. Intel RealSense D435

3.1.3. Intel RealSense SR300

The camera we got was a test unit of the Intel RealSense SR300. Before everything else, we checked that this camera's SDK includes the Hand Tracking Module, and it did.

Interfacing with this camera was challenging, taking us a lot of attempts but eventually, we successfully interacted with the camera in visual studio and C++.

We wrote the algorithm we planned and wanted to test it out. To test the algorithm performance, we wrote an entire system with the following modes:

- Recording – for recording movements into movements matrices.
- Optimizing – Implement optimizations on the movement matrices we recorded. Will be discussed later.
- Testing – A live test mode to see the accuracy of our algorithm in action.

3.1.3.1. Intel RealSense SR300

3.1.4. Origin Setup

After recording some basic movements to test like moving to the right, we noticed a big problem: each time we turn on the camera, its place in the world was changed and with it the values of the coordinates for every movement matrix being captured. Moreover, we noticed that if we were to move the hand to the right, but from a different starting place, the values although representing the same movement, will be far away L2 metric-wise, something we do not wish to consider when recognizing movements.

The solution for both problems was simple, for every movement matrix, we treated the first frame vector of points as a relativity vector. We subtract all the frames by the relative vector values so that he will become the origin point of the movement – a vector of zeros $(0,0,\dots,0)$, and all the next frames start the movement from it.

This way, no matter the place of the camera in the world, or the position from which we start to perform the movement, it will all be relative to the first frame.

Let us demonstrate using a simple example, tracking only the palm point. We will take the same movement from section 3.1.1 – moving the hand to the right, hold for a little and go back to the left. After recording the movement, the movement matrix will look something like this:

	Palm		
Frame	Pm_x	Pm_y	Pm_z
1	5	12	-20
2	5.5	12	-20
3	6	12	-20
4	6.5	12	-20
5	7	12	-20
6	7	12	-20
7	6.5	12	-20
8	6	12	-20
9	5.5	12	-20
10	5	12	-20

3.1.4.1. Recorded Movement Matrix

The camera's location in the world is unknown (can be located anywhere by the player for all we know) and the wrist point first frame location was at $(5,12,-20)$.

Now, let us try to recognize the movement. We performed the movement and got this matrix from the live stream (for the example we performed the movement with extreme precision so the L2 distance from the matrix to the recorded movement matrix should be 0):

	Palm		
Frame	Pm_x	Pm_y	Pm_z
1	20	3	14
2	20.5	3	14
3	21	3	14
4	21.5	3	14
5	22	3	14
6	22	3	14
7	21.5	3	14
8	21	3	14
9	20.5	3	14
10	20	3	14

3.1.4.2. Captured Movement from the Live Stream

Let us look what we got: even if the x axis was the same, still the y axis and z axis values that do not even participate in the movement would inflate our L2 distance value. It is easy to see the matrices are not close.

Now, let us apply our Origin Setup solution, the relativity vector of our matrix is the first frame: $(5,12,-20)$:

Frame	Palm		
	Pm_x	Pm_y	Pm_z
1	5	12	-20
2	5.5	12	-20
3	6	12	-20
4	6.5	12	-20
5	7	12	-20
6	7	12	-20
7	6.5	12	-20
8	6	12	-20
9	5.5	12	-20
10	5	12	-20

$$\xrightarrow{-(5,12,-20)}$$

Frame	Palm		
	Pm_x	Pm_y	Pm_z
1	0	0	0
2	0.5	0	0
3	1	0	0
4	1.5	0	0
5	2	0	0
6	2	0	0
7	1.5	0	0
8	1	0	0
9	0.5	0	0
10	0	0	0

3.1.4.3. Applying Origin Setup to the Recorded Movement Matrix

Let us use the same matrix from the live stream as before, we apply for it the Origin Setup solution as well and then we will calculate the L2 distance:

Frame	Palm		
	Pm_x	Pm_y	Pm_z
1	20	3	14
2	20.5	3	14
3	21	3	14
4	21.5	3	14
5	22	3	14
6	22	3	14
7	21.5	3	14
8	21	3	14
9	20.5	3	14
10	20	3	14

$$\xrightarrow{-(20,3,14)}$$

Frame	Palm		
	Pm_x	Pm_y	Pm_z
1	0	0	0
2	0.5	0	0
3	1	0	0
4	1.5	0	0
5	2	0	0
6	2	0	0
7	1.5	0	0
8	1	0	0
9	0.5	0	0
10	0	0	0

3.1.4.4. Applying Origin Setup to the Captured Movement from the Live Stream

Now, we can see that the L2 distance is 0 as both matrices are the same – the movement is recognized!

Of course, in real life situations getting 0 distance (meaning performing the exact movement) is hard, for this we have our threshold value.

3.1.5. Optimizations

When testing, we noticed that even by trying to do the same movement repeatedly, we got random and unstable distances. We took a close look and found out our RealSense SR300 camera is having a lot of noise in its hand's points recognition. The noise was even worse when trying to record small and delicate movements like splitting in Blackjack.

We did not want to give up, so we performed some optimizations to try and overcome the noisiness of the camera:

- First, we tried averaging the movement matrix. We recorded the same movement several times (5-10), hence producing multiple movement matrices. Then, we went frame by frame and calculated the average of all the matrices into a single movement matrix. This way we thought we will cover errors captured falsely by the camera that may be repeating. It did improve the recognition, but it was far from being usable.
- Secondly, we thought of limiting the noise influence by applying weights to the calculation of the distance. If we go back to the simple example of moving the hand on the x axis, both y and z axes can be completely ignored. So, we could multiply the matrices by $(1,0,0)$. So, if the movement matrix is A, the currently captured frames are B and the weights vector is W, the distance would be calculated as an L2 distance from each row $a_i \in A$ to the corresponding $b_i \in B$ multiplied element-wise by W:

  $$distance = \sum_{i}^{n} (a_i - b_i) \cdot W$$

  This optimization also made an improvement, but ever so slightly and was still not in our standards.

As the noise and instability of the SR300 were too much, we decided we should schedule a meeting with Yaron to show him our results so far. We showed the demo and the poor hand recognition by the camera, both visually and metrically. It is important to add that we showed the visualizing demo included in the SDK for the hand recognition of the SR300, and even when we held our hand still in a well-lighted room with contrasting background, the hand visualization would jump and twist all over, further proving the noises of this camera. We all agreed that we should replace the camera again.

3.1.6. Leap Motion Controller

The replaced camera was the Leap Motion Controller. Again, we needed to interface with a new device. The Leap Motion SDK was challenging to install, with many versions and updates, it took us time to land on the desired version that fits our game the best.

After successfully interfacing with the Leap Motion Controller, we checked the quality of the hand recognition. We were happy to see it was performing extraordinary well. The numbers were stable and far better than the previous SR300 performance. When visualizing the difference in quality was even more noticeable, the hands were moving much smoother and there were almost no jumps at all between frames.

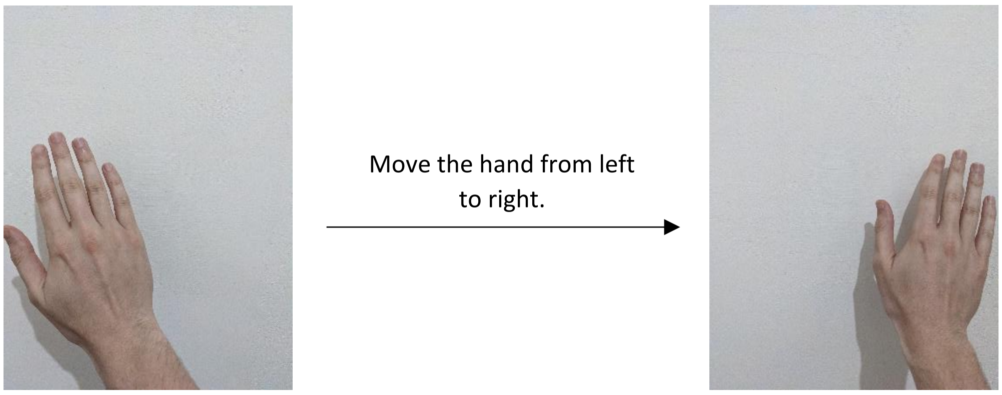
We wrote our recording and testing system again for the Leap Motion Camera to continue developing and testing our algorithm.

3.1.7. Dynamic Time Warping

We ran the basic tests – trying to recognize big movements, and the results with the Leap Motion Controller were very impressive. We did not need the optimizations we added for it to recognize the basic big movements like for example wave from the right to the left.

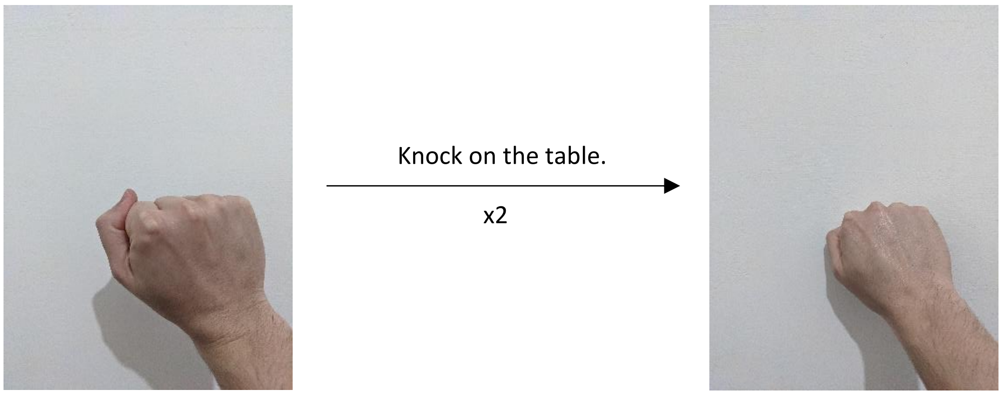
Happy and excited we went on to record and test the blackjack movements:

- Stand – Move the hand from left to right.



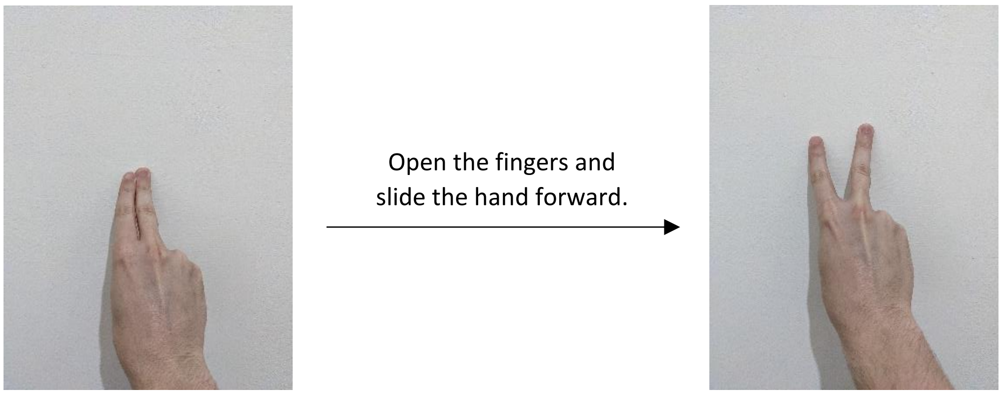
3.1.7.1. Fold Movement

- Hit – With close hand, knock on the table twice.



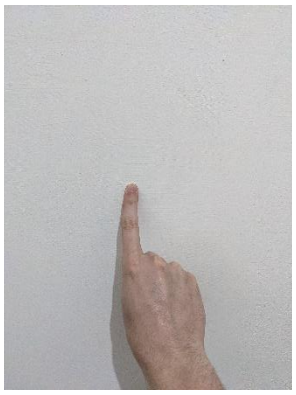
3.1.7.2. Hit Movement

- Split – With close hand and both the index and middle fingers pointing forward and close to each other, open both and slide the hand forward at the same time.



3.1.7.3. Split Movement

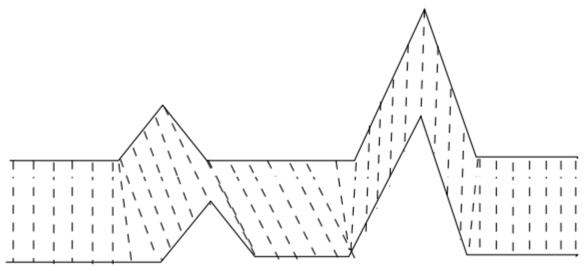
- Double down – With close hand, point the index finger forward and hold still for a moment.


3.1.7.4. Double Down Movement

These are the movements of the Blackjack game and so these are the movements we want to be in our game.

Now, although the movements were being able to be recognized, when testing out some of our attempts were missed by the algorithm. We sure can accept small errors but it was too much for our standard of gameplay. After debugging the performance, we noticed that If we were to repeat the movement at the same pace, it was recognized better, but if we were to do the movement faster or slower in an acceptable percentage, it was missed.

After researching the internet, we found the solution to our problem: DTW – Dynamic Time Warping. DTW is an algorithm for measuring similarity between two temporal sequences, which may vary in speed. It is commonly used for automatic speech recognition, to cope with different speaking speeds.



3.1.7.5. Dynamic Time Warping Demonstration

DTW sounded perfect for our needs. We implemented it into our algorithm when comparing two-movement matrices and indeed, the performance was far better. In the results chapter section 4.2 we show the impact DTW had on the accuracy of our recognition algorithm.

But, even with DTW we still had a problem. Our first vision of the algorithm was to recognize the movement globally, not in a specific time frame, meaning that when it is the player's turn, the algorithm will try to recognize each serial of frames in a chosen gap to all our movement matrices by our threshold value until some movement is recognized. This scenario happens to work only in good testing grounds as when debugging we found out we cannot rely on the threshold as our accuracy measurement.

Although the recognition and values were better thanks to the Leap Motion Controller and DTW, still we faced a major issue with the threshold value. Because if the camera moved accidentally, or perhaps the player moved his chair so he is now further or closer to the camera, or maybe the angle of either the camera or the player has changed, the threshold value should change accordingly and dynamically, something we wanted to avoid. And, if the threshold was to be changed, how will we know when the user is not doing any movement? Perhaps he did a movement wrong so it should not be recognized. These scenarios are not farfetched and as the distance results may vary, we could not count on our threshold value to work live in-game. We have further dry tested this and came to the same conclusion (the results are showcased in the results section 4.3). So, we took another approach on when and how to run the algorithm.

3.1.8. Correctness Factor

The new approach was that when it is the player's turn, he will have a time window to perform his move, and when it is done, we assume he did a movement and so the closest movement would be chosen. This way we did not have to face a problem when the player is not performing any movement and thus the threshold value was not needed anymore.

Now, with the algorithm working in our hands, we lastly thought on edge cases regarding errors that might still occur. The issue that came to our minds was a situation of misrecognizing a movement, causing the game to perform a false move the player did not want to. In our tests, we saw that if we did a movement of the 4 above, the algorithm recognized it without any problems whatsoever. The issue was that if a player performed a hand movement wrong, for example, stopping mid-way, it could be recognized as two movements and the wrong one could be chosen.

To fix the issue, we implemented something we called the Correctness Factor. Each player when starting up the game is recording each of the four movements several times. All of the recordings will be located in the movements directory so later in-game the algorithm will calculate the L2DTW distance from all of them.

The Correctness Factor is now fully replacing the threshold and the top closest L2DTW value. The algorithm will consider the top 3 smallest distances – meaning a Correctness Factor of 3. If the majority of the 3 are of the same movement (2/3 or 3/3), we can be sure this is the movement the player did. If each one of the top 3 is of different movements, we can be sure the player did the movement wrong, and he should repeat it.

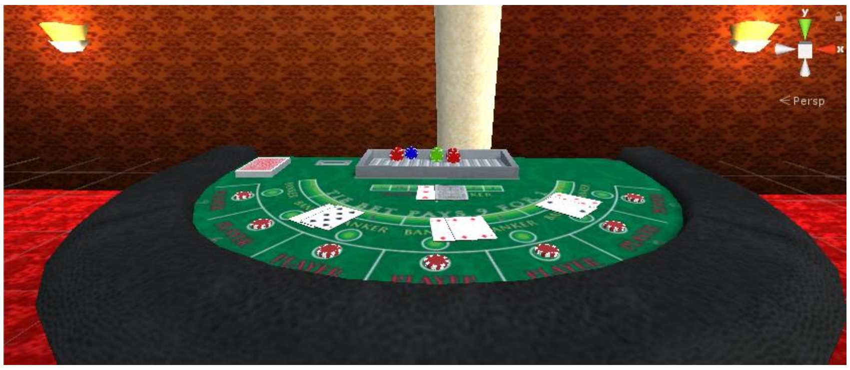
This solution played well in-game and was fast and easy to implement. It overcame the issue in what we think is an elegant way. And, more importantly, it made the recognition within the time window works very, very good!

3.2. Developing the Game

3.2.1. 3D Graphics

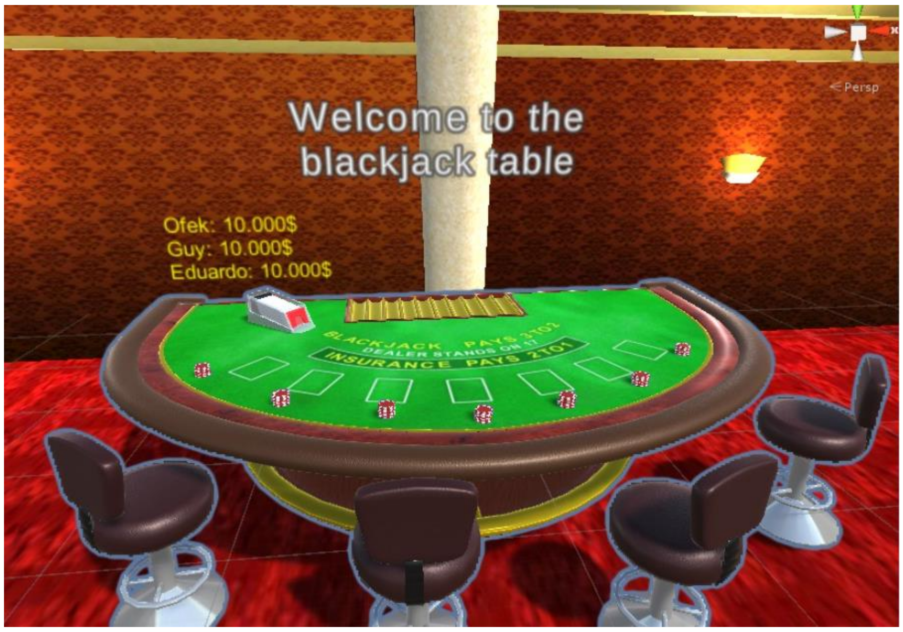
The first thing we did coming to develop the game was to design and create the game environment. It was very important for us to give the player an out-of-the-ordinary experience and the realism aspect of our virtual casino is a major component to make the atmosphere vivid and alive. We will cover the major 3D components and their implementation in the game.

- Static Objects: The static objects as their name suggests are for the scene and looks. We took aspiration from looking at different casino photos and so we filled our scene with different objects we found capturing the casino atmosphere. We of course used a Blackjack table, but also, we added a Roulette table, a Slot machine, wall and ceiling patterns, lights, and a casino-style rug to elevate the casino vibe. We first used free assets from the Unity Asset Store and end up with the following scene:



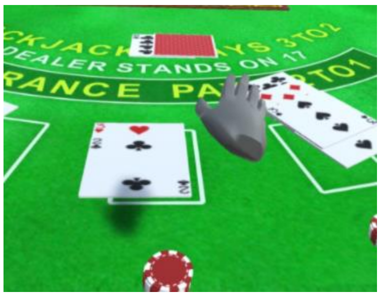
3.2.1.1. First Version of the Virtual Casino

Later the development process, after the interim meeting, the GIP lab was kind enough to supply us with high-resolution assets bought from the Unity Asset Store and the result is now:



3.2.1.2. Final Version of the Virtual Casino

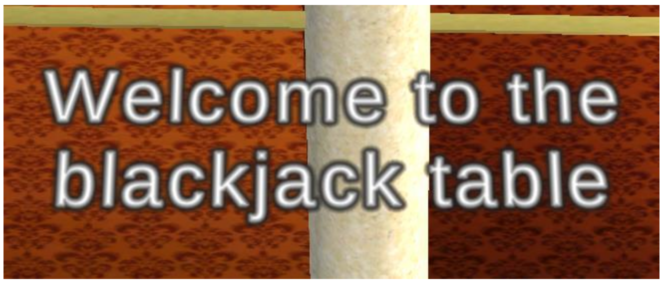
- Playing Cards: The playing cards have a preset location close to the center of the Blackjack table to be at the dealer's location. During the game, the script 'CardHandler.cs' (we will talk later in this chapter) will create the cards in their pre-set location and with each player's orientation will deal the cards. The scripts are the only ones who control the cards from this point on.
- Game's Camera: The game's camera is fixed to the player's direction of looking with the Oculus Rift.
- Hands: The hands are being rendered by the Leap Motion Controller hands library. In favor of synchronization, we configured the origin point of the camera to be at the beginning of the Blackjack table so the hand's location will be in a convenient position during the game.
- Lighting: To further enrich the feeling and sense of realism, we decided to add directional light that is adding shadow to the player's hand and cards.


3.2.1.3. Player's Hand Over His Cards with the Hand Shadow

3.2.2. 2D Canvases

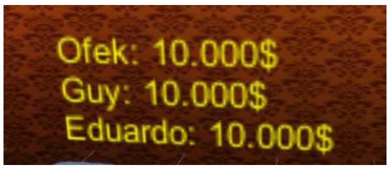
After designing and creating our virtual casino we thought about how the game should be played from the user perspective. If we were to play, we thought how will it flow nicely, in a fun and interactive way? We decided to add 2D canvases for the user interface of our game. Scores, timers, and instructions are needed for the game to be more user-friendly and easier to play. The major 2D canvases are:

- TextPro – A text box of Unity enabling communication with the player, navigating it through the game.



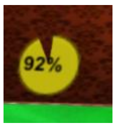
3.2.2.1. TextPro Example

- Scores: Texts for the player scores, how much money they earned or lost.



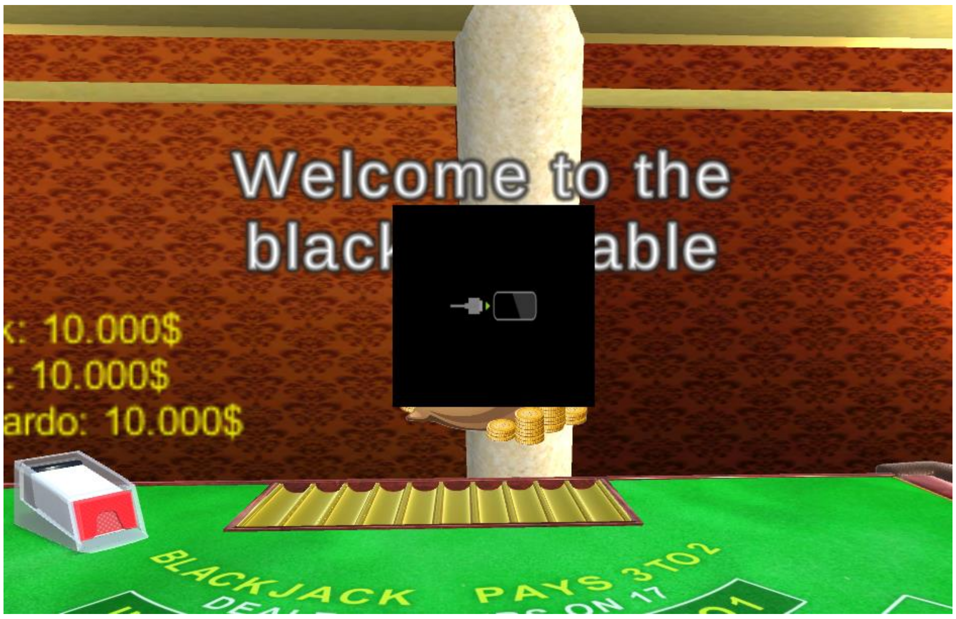
3.2.2.2. Player Scores Text

- Move Timer: To make it clear for the user when it is his turn and when he should interact with the Leap Motion Controller, we added a timer to when he should perform his move. The timer is a circle that is filling up from 0% to 100%.



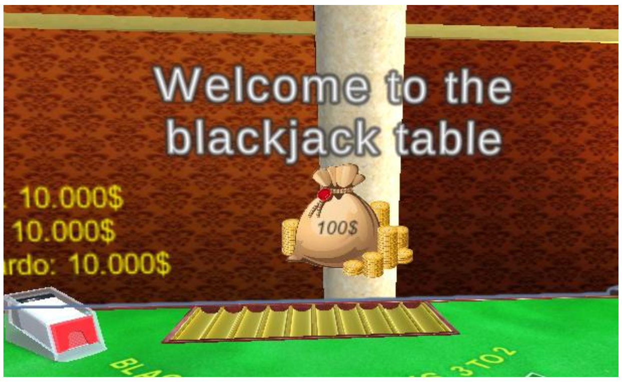
3.2.2.3. Move Timer

- Leap Motion Controller Notifier: An image presented when the Leap Motion Controller is not recognized. It was highly needed as our Leap Motion Controller cable was unstable and kept disconnecting.



3.2.2.4. Leap Motion Controller Notifier Showing the Controller is Unplugged

- Betting Image: An image presenting the current height of the player's bet. We took a simple image with white background and used Flood Fill on the alpha channel of the image to remove its background.



3.2.2.5. Betting Image

3.2.3. C# Scripts

After finishing with the environment and the UI, we started developing the core code of our game. The game is written in its entirety in the following scripts:

- GameManager.cs: Controls most of the game's logic. Within this script, the Coroutines of the game are running. These are the loops that run the game flow such as dealing cards, betting questions, player movements, and so on. The script is importing 'Leap.Unity' from which we were able to get the hand coordinates recognized by the Leap Motion Controller API – LeapServiceProvider. Other than that, it is initializing the Oculus Rift and updates the 2D canvases according to the game's current state, like instructions and scores. The connection of the Leap Motion Controller and Oculus Rift will both be discussed more in chapter 3.3.

- Player.cs: Saves the cards, points, and bets of each player in every round.
- CardsManager.cs: Draws a card randomly and sets its orientation according to the player's seat at the table. In case it is the dealer's second card, it also flips it.
- Cards.cs: Controls the game object of every card on the table. The function 'Update' in it is declared that all cards are advancing in a uniform direction (interpolation) from the dealer to the player until a threshold distance from the dealer is achieved. If "Split" was performed, the script also moves the card accordingly.
- HandModelManager.cs: A script by Leap Motion that updates the hand coordinates in each frame. In it, we inserted our hand recognition algorithm.

3.3. Connecting the Algorithm and Devices

3.3.1. Connecting the Leap Motion Controller

Coming to connect the devices and algorithm we first decided to connect the Leap Motion Controller. The main reason was due to the fact our algorithm depends on it and we thought it is best to leave the Oculus Rift to the end as we saw it more as a final touch.

To connect the Leap Motion Controller to our game we installed the addon of the controller dedicated to the Unity Engine. With it came a 'GameComponent' of type Leap that on it the hands are being rendered. The script 'HandModelManager.cs' that comes with the Leap package allows us to sample the hand's coordinates values in every frame. As we already discussed in 3.2.1 regarding the hand 3D object, we configured the origin point of the Leap Motion Controller to be at the beginning of the Blackjack table to mimic the player's position.

3.3.2. Implementing the Hand Movement Recognition Algorithm

Once we finished the connection of the Leap Motion Controller, implementing our hand movement recognition algorithm in the game was easy. We used an existing script named 'HandModelManager.cs' to do so. In this script, each time the hand's coordinates are being updated, we updated the relevant details to our algorithm as well using a dedicated API we wrote for that.

The API supports both recording movements and recognizing them. The main functions in this API are:

- RecordGesture(): Record a movement.
- IsRecording(): Return true if a recording is currently being performed.
- StartRecognizing(): Begin the hand movement recognition process.
- GetIsNoHandsDetected(): Return true if at the end of the recognition process the Leap Motion Controller did not detect any hands.
- GetLastMove(): Return the last recognized movement.

During the development of this API, we tried using Leap's API for "Pinch" gesture recognition. We wanted to use "Pinch" as an input from the player to approve certain actions in our game. Sadly, Leap's detection was not accurate enough, so we gave up on this idea. Instead, we used the Oculus Rift's controller for input from the player, more of it in the next section.

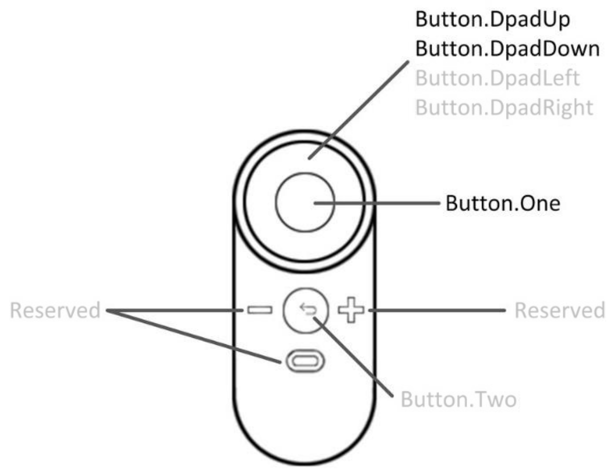
3.3.3. Connecting the Oculus Rift Headset

Having both Leap Motion Controller and our hand movement recognition algorithm working in our game, we lastly wanted to connect the Oculus Rift Headset. At first, we had trouble handling the Oculus Rift as it required a dedicated GPU, and all of us had only integrated GPU on our computers. Luckily, we found a computer with the right hardware to support the Oculus and it made our life much easier (we were not able to come to the lab due to the closure in the hard Coronavirus times).

After finding the computer, we installed the Oculus Rift's Unity addon and got its API. We configure the game's camera to aim at the direction of the Oculus Rift direction and we got the desired result: a virtual casino in virtual reality!

It was a cool experience, but we had a problem with our 2D canvas. We first made the canvas stick to the player's screen, meaning it will always be visible no matter the direction the player is looking. It made a lot of sense to the scores and instructions text be this way, but Unity's engine had problems making the text on the canvas vibrate a lot which made our eyes hurt and an overall unpleasant experience, to say the least from playing. To fix this issue, we simply moved the 2D canvasses to a static location in world space at the end of the Blackjack table.

Then, as we explained at the end of section 3.3.2, we wanted to replace the "Pinch" recognition mechanism as it performed poorly. We installed another package named OVR which enables the usage of the Oculus Rift's controller. With it, we configured the main button – 'Button.One' to be our OK input and we further used the up and down arrows ('Button.DpadUp' and 'Button.DpadDown') to increase or decrease the money in the betting stage.



3.3.3.1. Oculus Rift Controller

4. Results

Our way of work on researching the algorithm was to perform dry and wet tests. First, we implemented and ran our ideas in Python on recorded movements and visualize the results – the dry tests. Only then, when we saw good results, we would take the idea to test in our testing system to see actual live results – the wet tests. If the wet tests performed as expected, we kept our dry test results to showcase here.

4.1. Intel RealSense SR300 Noises

Sadly, we did not save any of our logs. The noisy data by the SR300 camera was presented and shown live to Yaron and we all agreed we should exchange the SR300 in favor of the Leap Motion Controller. We felt it is important to mention it in the report as presenting the noisy data would have been nice.

We ask the reader to believe in our tests and work, keeping in mind that replacing the device cost in rewriting the algorithm and testing system from scratch for the new Leap Motion Controller.

4.2. L2DTW Impact

We implemented L2DTW and L2 in our Python lab script. For each one of the movement matrices, we generated 25 additional matrices of different lengths and values using uniform distribution to randomly increase or decrease by given percentage.

Let us mark the original matrix as $M_{n\times 63}$. Our inflation configuration to generate a matrix $M'_{n'\times 63}$ from it was:

- Frames inflation $f = 0.5$, is effecting the generated movement's length – the number of frames. So, the generated matrix length n' is equal to:
  $$n' = U\big(n \cdot (1-f), n \cdot (1+f)\big)$$
  With $f = 0.5$ we mean that for us the movement should still be recognized if its frames length is between $0.5 - 1.5$ of the original length.
- Values inflation $v = 4$, is affecting the generated movements values. For each value $m_{i,j}$ in the movement matrix, the generated value $m'_{i',j'}$ for our generated matrix would be equal to:

$$m'_{i',j'} = U\left(m_{i,j}\cdot\frac{1}{v}, m_{i,j}\cdot v\right)$$

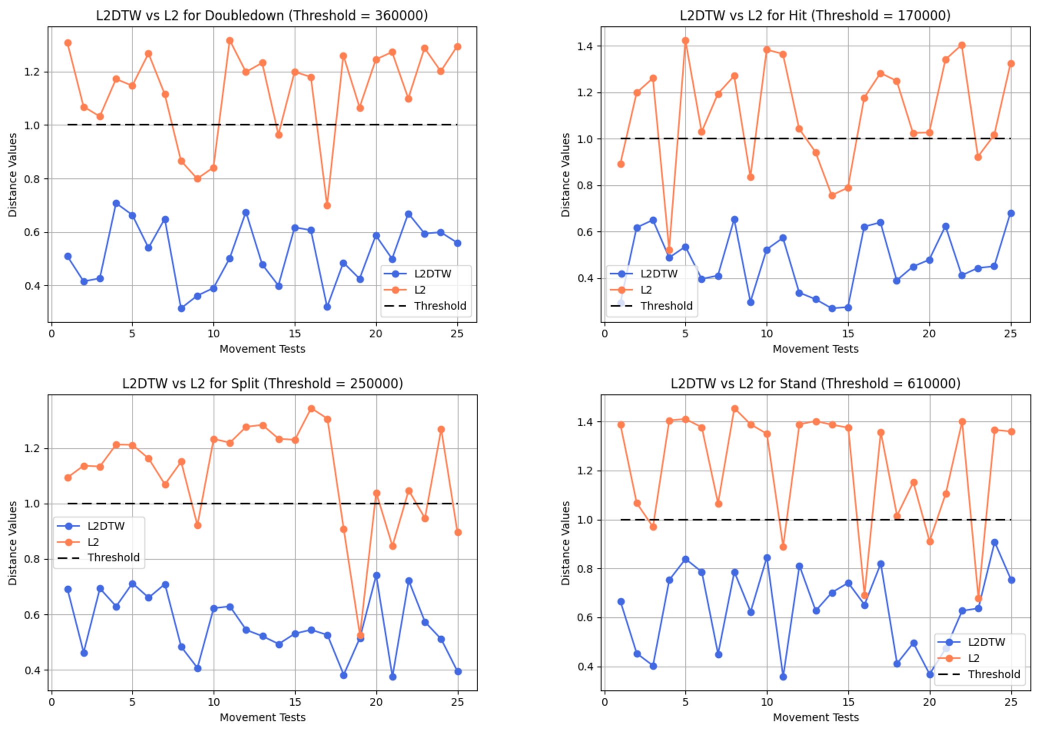
Both i', j' are relative to i, j according to the generated new length n':

$$i' = \left\lfloor\frac{n}{n'}\cdot i\right\rfloor$$

$$j' = \left\lfloor\frac{n}{n'}\cdot j\right\rfloor$$

With $v = 4$ we mean that from our experience with our live testing system, the values can be between $0.25 - 4$ times of their original value and still make sense to be the same movement.

For each of the 25 generated movements, we calculated the distance from the original one using L2 and L2DTW. The thresholds for each movement were chosen using our testing system live with the Leap Motion Controller to see what works well with each movement. To simplify the graph, we divided the distance (L2 and L2DTW) by the threshold so each threshold will be equal to 1. The results were strongly in favor of DTW without leaving a doubt for all 4 movements:

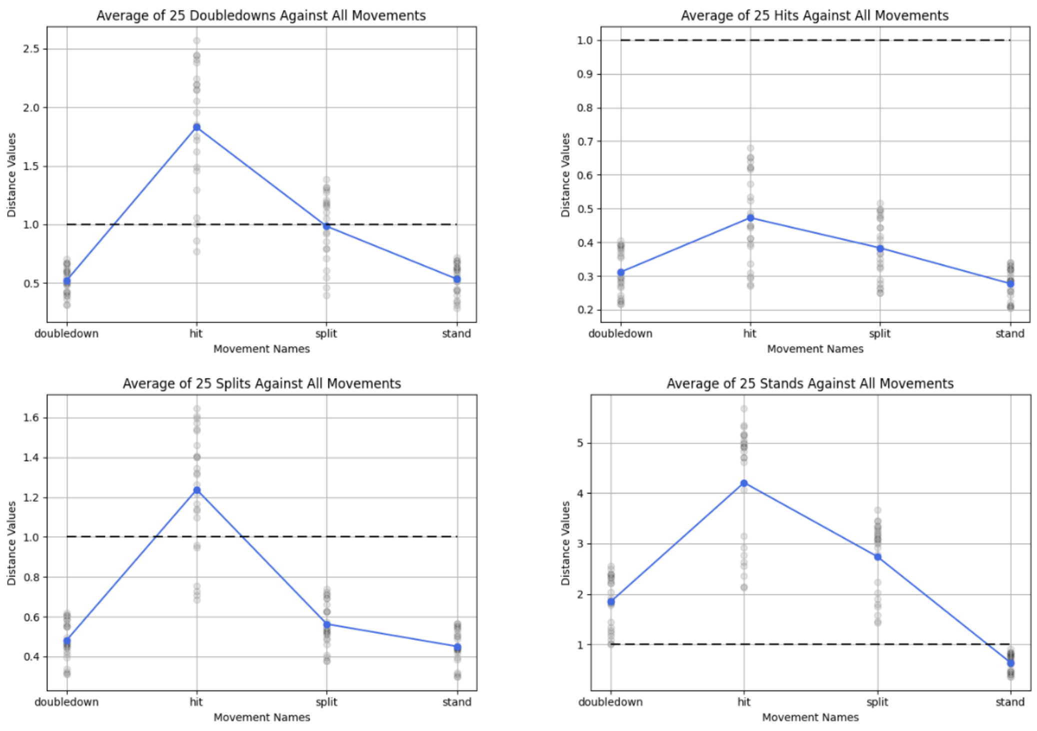


4.2.1. L2DTW Against L2 Tests Results

After seeing the impressive results, we implemented the algorithm in our testing system and saw the improvement in real time using the Leap Motion Controller as was told in section 3.1.7.

4.3. Correctness Factor Against Threshold

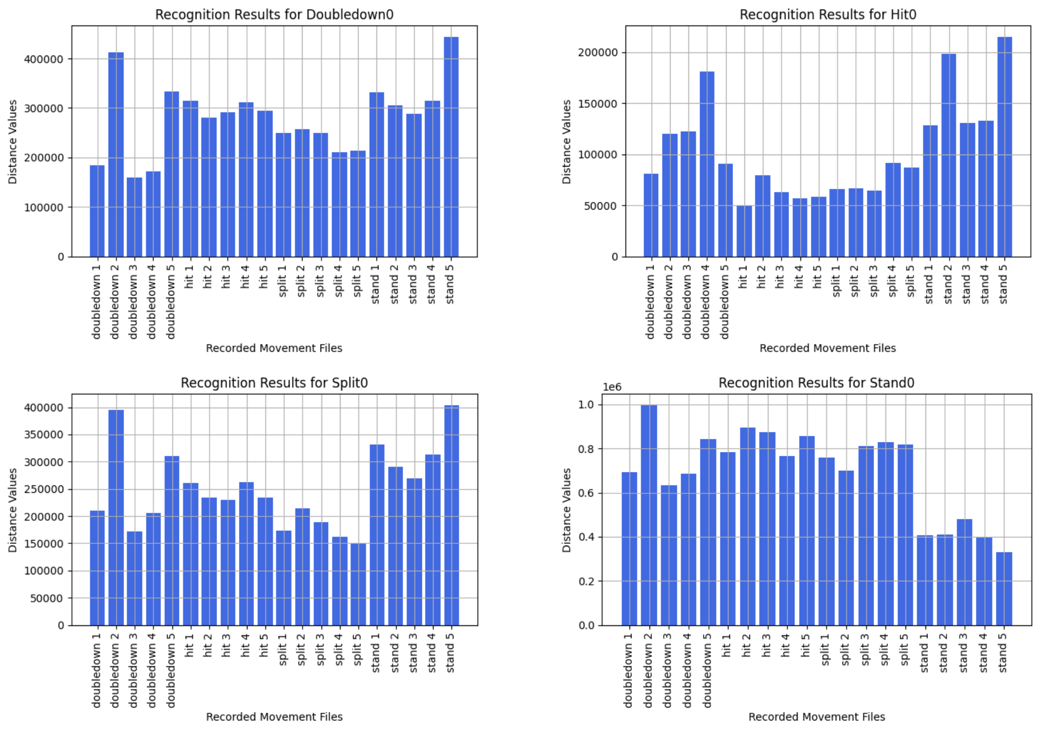
We will first show dry tests we ran in our Python lab script showcasing why the threshold approach cannot be trusted. In this test, we averaged the L2DTW distance of all the 25 generated movements to each of our recorded movements:



4.3.1. Average of 25 Generated Movements Against All Movements Tests Results

The grey circles are distance results, the darker the circle seems means more calculations yielded the same distance. The blue color is the average of all the 25 distance calculations. We remind you that these 25 movements were generated by us and they are not a representation of the real-life scenarios. And, even with these highly accurate generated movements, apart from "Stand", all the movements were not alone under their threshold value, meaning there is a high chance of miss-recognizing a movement. You can argue that the threshold should be changed and fine-tuned for each movement more but changing the threshold for one will harm the other and for more reasons explained in section 3.1.7, it is easy to see why the threshold approach is problematic. Moreover, in our testing system, the same problem remains, and it is much more noticeable testing live. It was too random to put the finger on a threshold value that was not hitting more than one movement. Too tight will not recognize a lot, too far will miss-recognize a lot.

So, as said in section 3.1.8 we decided to go with the Correctness Factor approach. We ran a few recognitions in our Python lab and seeing the good behavior from the algorithm we decided to go right away to test it in our testing system. We will present an example of a recognition graph here:



4.3.2. Example for Recognition Results

All the distance values are presented as bars, the top 3 lowest bars are the closest distances. So, a Correctness Factor of 3 is enough to fix our issue. You can see for example in the "Split" bars graph, the third highest is not a "Split" but "Double Down", yet still, we have a major of 2 so we would have recognized it well.

In real-time when testing it live with the Leap Motion Controller, this approach worked even better, and hence we used it in our game.

5. Conclusions

After a lot of attempts and failures along the way, our vision for BlackjackVR came to life and we are very proud and happy with the results. The game is fun, enjoyable and we learned a lot from the process of developing it. From the research of our hand movement recognition algorithm to the development of the game in Unity, it was an adventure we will not forget.

5.1. Hand Movement Recognition Algorithm

- Our naïve numerical approach for the algorithm works. With DTW and a different approach from a threshold of recognition, we were able to make the best out of it without compromising on performance.
- Although it did work, we will not recommend this algorithm for more than 8-10 movements, as this algorithm is not time efficient.
- With that said, the algorithm does have its positives: it is very easy to understand and the implementation of it in any system working with a Leap Motion Controller is quick and simple.

5.2. BlackjackVR Game

- Unity was perfect for our needs. It is very user-friendly, and the large and professional community of it helped our vision to become a reality – a virtual reality.
- Both Oculus Rift and Leap Motion Controller have packages dedicated to Unity and the connection between them was nice and easy.
- We will recommend Unity to every developer looking to develop a virtual reality product.

6. Recommendations

Our recommendations are more of a future work regarding our project. We believe it can be pushed further and if it does, we will be happy to know.

- We think our numerical algorithm can be replaced with a Deep Learning model. Just like speech recognition is now commonly recognized by DL, we think movement recognition can be done as well. The coordinates of the hand provided by the Leap Motion Controller could be used as an input to a Sequence-to-Sequence type model so the recognition can be more accurate, fast, and able to support much more movements, removing the Oculus Rift's controller for the bidding stage.
- Perhaps, Leap Motion Controller can be dropped in favor of the Oculus Quest which has a new hand recognition as well. For a player, this will make the game cheaper and more comfortable to use.
- Game-wise, we would like Unity to resolve the text on screen vibration issue so the player will not need to look for a specific place for the scores and other information.
- Moreover, we think the game can be made into a multiplayer game, enriching the experience of the player, making the casino come to life.
- Last but by no means least, graphics-wise, we know that for today's standard our game is not as sharp looking. So of course, this can be improved as well.

7. Literary Sources

7.1. General Information

Unity
https://en.wikipedia.org/wiki/Unity_(game_engine)

Oculus Rift
https://en.wikipedia.org/wiki/Oculus_Rift

Leap Motion
https://en.wikipedia.org/wiki/Leap_Motion

Dynamic Time Warping
https://en.wikipedia.org/wiki/Dynamic_time_warping

7.2. Image Links

2.1.1. Unity Logo
https://www.unity.com

2.2.1. Oculus Rift
https://www.amazon.com/Oculus-Rift-Virtual-Reality-Headset-pc/dp/B00VF0IXEY

2.3.1. Leap Motion Controller
https://www.robotshop.com/en/leap-motion-controller.html

3.1.1.2. Leap Motion Controller Hand Structure
https://developer-archive.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html

3.1.1.3. Leap Motion Controller Hand Hierarchy
https://blog.leapmotion.com/getting-started-leap-motion-sdk/hand-hierarchy/

3.1.2.1. Intel RealSense D435
https://www.bhphotovideo.com/c/product/1432415-REG/intel_82635awgdvkprq_realsense_d435_webcam.html

3.1.3.1. Intel RealSense RS300
https://he.aliexpress.com/i/4000073134324.html

3.1.7.5. Dynamic Time Warping Demonstration
https://en.wikipedia.org/wiki/Dynamic_time_warping

3.3.3.1. Oculus Rift Controller
https://docs.unity3d.com/550/Documentation/Manual/OculusControllers.html

8. Appendices

We decided to join our Python Lab script. Each of the plots seen in this report was produced fully by the functions in this script. It is highly documented so we will not add additional explanations here.

We ran the script using Python 3.7 interpreter and PyCharm. The requirements are 'numpy' and 'matplotlib'. The script is using recorded matrices expected to be in a directory named 'recorded_movements'.

```python
"""
                                            BlackjackVR's Python Lab
                                A movement recognition algorithm research script.


Instructions:
    To run the script, make sure the recorded movements are located in a directory named 'recorded_movements' and
    that all 4 movements are there. Comment and uncomment the calls to the tests from the 'main' function.

Requirements:
    numpy>=1.20.1
    matplotlib>=3.3.4

Authors:
    Ofek Gutman
    Eduardo Abramoff
    Guy Lecker
"""
from typing import List, Tuple, Dict
import os
import random
import numpy as np
import matplotlib.pyplot as plt

# Paths:
SCRIPT_PATH = os.path.join(os.path.dirname(os.path.realpath(__file__)))  # Script path is set automatically.
OUTPUT_PATH = SCRIPT_PATH  # To edit the script's output, defaulted to the script's location.

# Amount of matrices to generate, meaning amount of tests to run:
GENERATED_MOVEMENTS_AMOUNT = 25

# Used main and secondary colors in the plots:
COLOR_1 = "royalblue"
COLOR_2 = "coral"


def insure_reproducible_results(seed: int):
    """
    To get reproducible results to match our report this method seeds random and numpy.random.
    :param seed: The seed's value.
    """
    random.seed(seed)
    np.random.seed(seed)


class Directories:
    """
    Output directories names to use.
    """
    PLOTS = "plots"  # Output directory for the plots.
    RECORDED_MOVEMENTS = "recorded_movements"  # Original recorded matrices expected to be located here.
    GENERATED_MOVEMENTS = "generated_movements"  # Output directory for generated movement matrices.

    @staticmethod
    def create_directory(path: str):
        """
        Create the directories involved in the given path.
        :param path: The path to create the directories.
        """
        os.makedirs(path,
                    exist_ok=True)
```

```python
class Movements:
    """
    Tested movements names and their thresholds value of recognition.
    """
    DOUBLEDOWN = ("doubledown", 360000)
    HIT = ("hit", 170000)
    SPLIT = ("split", 250000)
    STAND = ("stand", 610000)

    @staticmethod
    def to_list() -> List[Tuple[str, int]]:
        """
        Get all movements tuples as a list.
        :return: The movements list.
        """
        return [
            attribute_value
            for attribute_name, attribute_value in Movements.__dict__.items()
            if (not (attribute_name.startswith('__') and attribute_name.endswith('__'))
                and isinstance(attribute_value, tuple))
        ]


class MovementRecognitionFactory:
    """
    Functions library for movements recognition methods.
    """

    @staticmethod
    def generate_movement_matrix(original_matrix: np.ndarray, frames_inflation: float,
                                 values_inflation: float) -> np.ndarray:
        """
        Generate a movement matrix from a given original matrix. The frames and values inflation can be adjusted as
        how accurate the generated copy will be.
        :param original_matrix: The matrix to generate a new matrix from.
        :param frames_inflation: Percentage of how much the frames amount can be larger or smaller than the
                                 original.
        :param values_inflation: Multiplication of how much the values can be larger or smaller than the original.
        :return: The new generated movement matrix.
        """
        # Initialize the randomize matrix:
        randomize_matrix_shape = (int(np.random.uniform(
            low=original_matrix.shape[0] * (1 - frames_inflation),
            high=original_matrix.shape[0] * (1 + frames_inflation),
            size=(1,))), 63)
        randomize_matrix = np.zeros(shape=randomize_matrix_shape)

        # Prepare for running:
        original_matrix_index = 0
        step = original_matrix.shape[0] / randomize_matrix.shape[0]
        randomize_matrix = randomize_matrix.flatten()
        original_matrix = original_matrix.flatten()

        # Start filling the randomize matrix:
        for random_matrix_index in range(randomize_matrix_shape[0] * 63):
            randomize_matrix[random_matrix_index] = float(np.random.uniform(
                low=original_matrix[int(original_matrix_index)] * (1 / values_inflation),
                high=original_matrix[int(original_matrix_index)] * values_inflation,
                size=(1,)
            ))
            original_matrix_index += step

        # Return the randomized matrix:
        randomize_matrix.resize(randomize_matrix_shape)
        return randomize_matrix

    @staticmethod
    def l2(a: float, b: float):
        """
        Perform L2 distance with the square root.
        :param a: Number a.
        :param b: Number b.
        :return: (a - b) ^ 2
        """
        return (a - b) ** 2

    @staticmethod
    def l2_distance(recorded_movement: np.ndarray, movement: np.ndarray) -> float:
        """
        Calculate the L2 distance between the movement matrices given to the shortest frame length.
        :param recorded_movement: The recorded original matrix - A.
        :param movement: The generated matrix - B.
        :return: sum_i((a_i - b_i) ^ 2) where i = len(A) > len(B) ? len(B) : len(A)
        """
        # Prepare for calculating the distance:
        recorded_movement = recorded_movement.flatten()
        movement = movement.flatten()
        limit_index = min(recorded_movement.shape[0], movement.shape[0])
        distance = 0

        # Calculate the distance:
        for i in range(limit_index):
            distance += MovementRecognitionFactory.l2(recorded_movement[i], movement[i])
        return distance
    @staticmethod
    def l2dtw_distance(recorded_movement: np.ndarray, movement: np.ndarray, w: int) -> float:
        """
        Calculate the L2 distance between the movement matrices given using the DTW algorithm.
        :param recorded_movement: The recorded original matrix - A.
        :param movement: The generated matrix - B.
        :param w: Window size. Defaulted to 10 as in our game.
        :return: The L2DTW distance between A and B.
        """
        # Prepare the DTW matrix:
        recorded_movement = recorded_movement.flatten()
        movement = movement.flatten()
        n = recorded_movement.shape[0]
        m = movement.shape[0]
        w = max(w, abs(n - m))
        dtw_matrix = np.ones(shape=(n, m)) * np.inf
        dtw_matrix[0][0] = 0
        for i in range(1, n):
            for j in range(max(1, i - w), min(m, i + w + 1)):
                dtw_matrix[i][j] = 0

        # Start the DTW algorithm:
        for i in range(1, n):
            for j in range(max(1, i - w), min(m, i + w + 1)):
                cost = MovementRecognitionFactory.l2(recorded_movement[i], movement[j])
                dtw_matrix[i][j] = cost + min(
                    dtw_matrix[i - 1][j],
                    dtw_matrix[i][j - 1],
                    dtw_matrix[i - 1][j - 1]
                )
        return dtw_matrix[n - 1][m - 1]


class Tests:
    """
    Functions library for dry tests to run from the main function.
    """

    @staticmethod
    def generate_movement_matrices(frames_inflation: float, values_inflation: float):
        """
        Generate movement matrices as much as the global parameter 'GENERATED_MOVEMENTS_AMOUNT' equal to. The
        generated movements will be located at OUTPUT_PATH/Directories.GENERATED_MOVEMENTS
        :param frames_inflation: Percentage of how much can the frame number increase or decrease.
        :param values_inflation: Percentage of how much can the values increase or decrease.
        """
        # Prepare to generate:
        insure_reproducible_results(seed=100)
        Directories.create_directory(path=os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS))
        file_suffix = ".1.csv"
        movements_list = Movements.to_list()

        # For each original movement matrix, generate 'GENERATED_MOVEMENTS_AMOUNT' matrices:
        for movement_name, _ in movements_list:
            # Load the original matrix:
            original_matrix = np.genfromtxt(os.path.join(SCRIPT_PATH, Directories.RECORDED_MOVEMENTS,
                                                         movement_name + file_suffix),
                                            delimiter=",")
            # Generate:
            for i in range(GENERATED_MOVEMENTS_AMOUNT):
                matrix = MovementRecognitionFactory.generate_movement_matrix(original_matrix=original_matrix,
                                                                             frames_inflation=frames_inflation,
                                                                             values_inflation=values_inflation)
                np.savetxt(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS,
                                        movement_name + str(i) + ".csv"),
                           matrix,
                           delimiter=",")

    @staticmethod
    def l2_vs_l2dtw():
        """
        Run the test comparing the recognition of L2 against L2DTW.
        """
        print("L2 vs L2DTW")
        Directories.create_directory(path=os.path.join(OUTPUT_PATH, Directories.PLOTS))
        file_suffix = ".1.csv"
        for movement_name, threshold in Movements.to_list():
            # Load original matrix:
            original_matrix = np.genfromtxt(os.path.join(SCRIPT_PATH, Directories.RECORDED_MOVEMENTS,
                                                         movement_name + file_suffix),
                                            delimiter=",")
            # Load generated matrices:
            matrices = []  # type: List[np.ndarray]
            for matrix_file in os.listdir(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS)):
                if movement_name in matrix_file:
                    matrices.append(np.genfromtxt(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS,
                                                               matrix_file),
                                                  delimiter=","))
            # Calculate distances:
            l2_distances = []  # type: List[float]
            l2dtw_distances = []  # type: List[float]
            for matrix in matrices:
                l2_distances.append(MovementRecognitionFactory.l2_distance(recorded_movement=original_matrix,
                                                                           movement=matrix) / threshold)
                l2dtw_distances.append(MovementRecognitionFactory.l2dtw_distance(recorded_movement=original_matrix,
                                                                                 movement=matrix,
                                                                                 w=10) / threshold)
            # Plot the results:
            figure, axes = plt.subplots()  # type: plt.Figure, plt.Axes
            axes.grid()
            axes.plot(np.arange(1, len(matrices) + 1), l2dtw_distances,
                      marker='o',
                      color=COLOR_1,
                      label="L2DTW")
            axes.plot(np.arange(1, len(matrices) + 1), l2_distances,
                      marker='o',
                      color=COLOR_2,
                      label="L2")
            axes.plot(np.arange(1, len(matrices) + 1), np.ones(shape=(len(matrices), )),
                      dashes=[6, 3],
                      color="black",
                      label="Threshold")
            axes.legend()
            axes.set_title("L2DTW vs L2 for {} (Threshold = {})".format(movement_name.capitalize(), threshold))
            axes.set_xlabel("Movement Tests")
            axes.set_ylabel("Distance Values")
            plt.show()
            figure.savefig(os.path.join(OUTPUT_PATH, Directories.PLOTS,
                                        "l2dtw_vs_l2_for_{}.png".format(movement_name)))
            # Log the results:
            print(movement_name)
            print("l2: {}".format(l2_distances))
            print("l2dtw: {}".format(l2dtw_distances))
            print()

    @staticmethod
    def l2dtw_vs_all_movements():
        """
        Run the test comparing the recognition of L2DTW of a movement on all recorded movements to see the average
        of results among all of them.
        """
        print("L2DTW vs All Movements")

        # Prepare to run:
        Directories.create_directory(path=os.path.join(OUTPUT_PATH, Directories.PLOTS))
        file_suffix = ".1.csv"
        movements = Movements.to_list()

        # Load the original movements:
        original_movements = {}  # type: Dict[Tuple[str, int], np.ndarray]
        for movement in movements:
            # Load original matrix:
            original_movements[movement] = np.genfromtxt(os.path.join(SCRIPT_PATH, Directories.RECORDED_MOVEMENTS,
                                                                      movement[0] + file_suffix),
                                                         delimiter=",")

        # For each movement, load the generated movement and compare distances to all 4 original movements:
        for generated_movement_name, _ in movements:
            # Initialize the distances dictionary to hold the results:
            l2dtw_distances = {}  # type: Dict[str, List[float]]
            # Go though the generated movements directory:
            for generated_movement_file in os.listdir(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS)):
                # Check its a matrix of the current movement being checked:
                if generated_movement_name not in generated_movement_file:
                    continue
                # Load the generated movement:
                generated_sample = np.genfromtxt(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS,
                                                              generated_movement_file),
                                                 delimiter=",")
                for ((movement_name, threshold), original_matrix) in original_movements.items():
                    # Calculate distance:
                    if movement_name not in l2dtw_distances:
                        l2dtw_distances[movement_name] = []
                    l2dtw_distances[movement_name].append(
                        MovementRecognitionFactory.l2dtw_distance(recorded_movement=generated_sample,
                                                                  movement=original_matrix,
                                                                  w=10) / threshold
                    )
            # Plot the results:
            figure, axes = plt.subplots()  # type: plt.Figure, plt.Axes
            axes.grid()
            for movement_name, distances in l2dtw_distances.items():
                axes.scatter(x=[movement_name] * len(distances),
                             y=distances,
                             color="black",
                             alpha=0.1)
            axes.plot(l2dtw_distances.keys(), [sum(distances) / len(distances)
                                               for _, distances in l2dtw_distances.items()],
                      marker='o',
                      color=COLOR_1)
            axes.plot(np.arange(len(movements)), np.ones(shape=(len(movements), )),
                      dashes=[6, 3],
                      color="black",
                      label="Threshold")
            axes.set_title("Average of {} {}s Against All "
                           "Movements".format(len(l2dtw_distances[generated_movement_name]),
                                              generated_movement_name.capitalize()))
            axes.set_xlabel("Movement Names")
            axes.set_ylabel("Distance Values")
            plt.show()
            figure.savefig(os.path.join(OUTPUT_PATH, Directories.PLOTS,
                                        "{}_average_of_{}.png".format(generated_movement_name,
                                                                      len(l2dtw_distances[generated_movement_name])
                           )))
            # Log the results:
            print("{} against:".format(generated_movement_name))
            print(l2dtw_distances)
            print()
        @staticmethod
        def recognize(matrix_file: str):
            """
            Run the recognition algorithm on the given generated sample from the given directory (noisy or not).
            :param matrix_file: The movement matrix file name including the '.csv'.
            """
            # Prepare to run:
            Directories.create_directory(path=os.path.join(OUTPUT_PATH, Directories.PLOTS))
            movements = Movements.to_list()
            l2dtw_distances = {}  # type: Dict[str, float]

            # Load the original movements:
            original_movements = {}  # type: Dict[Tuple[str, int], np.ndarray]
            for movement in movements:
                i = 0
                for movement_file in os.listdir(os.path.join(SCRIPT_PATH, Directories.RECORDED_MOVEMENTS)):
                    if movement[0] not in movement_file:
                        continue
                    i += 1
                    original_movements[(movement[0] + " " + str(i), movement[1])] = np.genfromtxt(
                        os.path.join(SCRIPT_PATH, Directories.RECORDED_MOVEMENTS,
                                     movement_file),
                        delimiter=","
                    )

            # Load the matrix to recognize:
            unrecognized_matrix = np.genfromtxt(os.path.join(OUTPUT_PATH, Directories.GENERATED_MOVEMENTS,
                                                             matrix_file),
                                                delimiter=",")

            # Calculate distances against all recorded movements:
            for ((movement_name, threshold), original_matrix) in original_movements.items():
                l2dtw_distances[movement_name] = MovementRecognitionFactory.l2dtw_distance(
                    recorded_movement=original_matrix,
                    movement=unrecognized_matrix,
                    w=10
                )

            # Plot the results:
            figure, axes = plt.subplots()  # type: plt.Figure, plt.Axes
            axes.grid()
            axes.bar(l2dtw_distances.keys(), l2dtw_distances.values(),
                     color=COLOR_1)
            plt.xticks(np.arange(len(l2dtw_distances)), l2dtw_distances.keys(),
                       rotation='vertical')
            axes.set_title("Recognition Results for {}".format(matrix_file.split('.')[0].capitalize()))
            axes.set_xlabel("Recorded Movement Files")
            axes.set_ylabel("Distance Values")
            plt.show()
            figure.savefig(os.path.join(OUTPUT_PATH, Directories.PLOTS,
                                        "recognition_of_{}.png".format(matrix_file.split('.')[0])))


    def main():
        """
        The main function the script is running. Comment and uncomment to run tests and re-generate the matrices.
        """
        # Generate the movements - notice it is seeding for reproducible results (to make it random, comment line 205).
        Tests.generate_movement_matrices(frames_inflation=0.5,
                                         values_inflation=4)

        # Call the tests:
        Tests.l2_vs_l2dtw()
        Tests.l2dtw_vs_all_movements()
        for matrix_file in ["doubledown0.csv", "hit0.csv", "split0.csv", "stand0.csv"]:
            Tests.recognize(matrix_file=matrix_file)


    if __name__ == '__main__':
        main()
```