Juho Ruuskanen

Design and Implementation of a Multi-Language Hand Gesture Typing System in Unity

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor's Thesis

17 May 2024

Abstract

Author:	Juho Ruuskanen
Title:	Design and Implementation of a Multi-Language Hand Gesture Typing System in Unity
Number of Pages:	28 pages + 1 appendix
Date:	17 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Electronics
Professional Major:	Electronics
Supervisors:	Janne Mäntykoski, Senior Lecturer

This thesis introduces and demonstrates a new approach to text input methods, utilizing hand gestures within the Unity engine. The thesis project aimed to develop a multilingual hand gesture typing system that enables text input in augmented and virtual reality environments.

For optimal user-computer interaction, this system in this project uses the Unity engine and integrates Ultraleap's pose detection algorithms with the Leap Motion Controller. In addition, the system uses 28 hand gestures mapped to either an alphanumeric character or a text editing command, providing users with a functional text input system.

The project is constructed in Unity by incorporating the hand gesture scripts into the pose detection algorithm and the pose event handler script, which handles the letter input to the text field. As a result, the pose detection algorithm binds these scripts together, forming the core components of this software design.

This thesis demonstrates the potential of the hand gesture typing system through careful research and development. The thesis' main findings include the successful integration of the Leap Motion Controller with Unity, the possibility to edit character sets to suit multiple languages and the identification of areas for further research and improvement.

This thesis offers insights into the potential of gesture-based interaction techniques to enhance user experiences and extend the capabilities of virtual and augmented reality environments. Through the development of gesture-based interaction and user interface design, this thesis presents new opportunities for research and innovation in the human-computer interaction field.

Keywords: Leap Motion Controller, Ultraleap, Unity

Tiivistelmä

Tekijä:	Juho Ruuskanen
Otsikko:	Monikielisen käsieleiden kirjoitusjärjestelmän suunnittelu ja toteutus Unityssa
Sivumäärä:	28 sivua + 1 liite
Aika:	17.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Electronics
Ammatillinen pääaine:	Electronics
Ohjaajat:	Lehtori Janne Mäntykoski

Tämä insinöörityö esittää uuden tavan luoda tekstiä käyttäen käsimerkkejä. Tekstin luonti ja käsimerkkien tunnistus tapahtui Unity-ohjelmassa. Insinöörityön tavoitteena oli luoda monikielinen kirjoitusjärjestelmä hyödyntäen käsimerkkejä virtuaalitodellisuudessa. Tietokoneen ja käyttäjän välistä vuorovaikutusta varten projektissa käytettiin Unity-ohjelmaa, johon integroitiin Leap Motion Controller sekä Ultraleapin kädentunnistusalgoritmi. Järjestelmässä oli yhteensä 28 eri käsimerkkiä, jotka toimivat kirjaimina tai tekstinmuokkauskomentoina. Nämä seikat mahdollistivat käyttäjälle toimivan tekstinsyöttöjärjestelmän.

Projekti rakennettiin Unity-ohjelmassa yhdistämällä käsimerkkikoodit ja kirjainsyöttöä käsittelevän ohjelman koodin käsimerkkientunnistusalgoritmiin. Näin ollen käsimerkkientunnistusalgoritmi sitoo nämä skriptit yhteen muodostaen ohjelmiston ydinkomponentit.

Huolellisen kehittämisen ansiosta insinöörityö osoittaa käsimerkkeihin perustuvan järjestelmän potentiaalin. Tärkeimpiä tuloksia insinöörityössä olivat Leap Motion Controllerin onnistunut integrointi Unity-ohjelman kanssa, monikielisyyden mahdollistaminen sekä lisätutkimuksen ja parannuskohteiden tunnistaminen.

Tässä insinöörityössä esitetään uusia näkemyksiä elepohjaisille vuorovaikutustekniikoille parantamalla käyttäjäkokemusta ja laajentamalla virtuaalitodellisuuden mahdollisuuksia. Käsimerkkeihin perustuvan vuorovaikutuksen ja käyttöliittymäsuunnittelun kehittämisen kautta insinöörityö avaa uusia mahdollisuuksia tutkimuksille ja innovaatiolle ihmisen ja tietokoneen välisen vuorovaikutuksen alalla.

Avainsanat: Leap Motion Controller, Ultraleap, Unity

Contents

List of Abbreviations

3D: Three-dimensional.

API: Application Programming Interface.

DC: Direct Current.

GameObject: Fundamental object in Unity that represents entities in the game world.

LeapC: Leap Motion C API, which provides a native interface for accessing LMC functionality from C-based programming languages.

LeapSDK: The Leap Motion Software Development Kit, includes libraries, examples, and documentation.

LMC: Leap Motion Controller.

Ultraleap: Company that develops technology for hand tracking.

Unity: Game development platform.

UnReal: Game development platform.

USB: Universal Serial Bus.

WebSocket: Communication protocol.

1 Introduction

Currently, effective text input methods are crucial for interacting with machines and computers. As digital devices become smaller, standard input methods such as keyboards and touchscreens cannot follow the advances. Other input techniques are required to offer comparable interaction experiences on these smaller devices. To fill this demand, a hand gesture typing system was developed in the Unity engine, that supports several languages and has 28 hand gestures.

Gesture-based interaction is an essential part of computer interface communication. Users may input text, move through digital environments, and operate robots with hand gestures. This project proposes gesture-based interaction techniques to enhance text input in virtual and augmented reality where standard input methods are impractical.

This thesis project used the advancements of gesture-based interaction by designing and implementing a hand gesture typing system. The system utilizes the LMC combined with Ultraleap's advanced pose detection algorithms, integrating them into a working software environment in the Unity engine. The project aimed to demonstrate the potential of gesture-based interaction techniques in enhancing user experiences and expanding the capabilities of virtual and augmented reality environments.

The hand gesture typing system has many advantages, including multi-language support and allowing users to use their preferred language and customize the character sets. Furthermore, the system has 28 hand gestures that offer users an extensive range of input choices.

The structure of the thesis targets all aspects of the project design and implementation. Chapter 2 provides background information on gesture-based interaction and the fundamentals of the project. Chapter 3 goes into the design and implementation of the hand gesture typing system. It showcases the system

architecture, integration with Unity, gesture recognition and mapping gestures to alphanumeric characters. Chapter 4 explores the future possibilities and the evaluation of the system. Chapter 5 concludes the thesis by summarizing its key components.

2 Theoretical Background

This chapter will explain the theoretical aspects of designing and developing the hand gesture typing system. The theories needed for understanding the principles of the system are gesture-based interaction, virtual and augmented reality, Unity as a development platform, and the fundamentals of the LMC.

2.1 Principles of Gesture-Based Interaction

Human interaction comes in many formats: speech, gesture, facial and body expressions. The best advantage of gesture-based interaction is its contactless human-computer interaction. Hand gestures come in dynamic and stationary; stationary implies the shape of the hand, and dynamic gestures are more about the movement of the hand. [1,1.]

Gesture-based interaction with computers comes in handy when keyboard and mouse interaction is not enough. Even though the invention of the keyboard and mouse has been great, communication between humans and computers has been tried to make as natural as possible, and the natural way of doing it is via speech or hand gestures. Hand gestures carry a lot of information, for example, pointing a person to move or do something. Thus, controlling computers with human gestures is an ideal option. [2,1-2.]

Humans have an extensive range of universally known gestures. Gestures used with objects can be pointing, moving, changing, and handling objects. Gestures can be grouped into three types according to their function, which are semiotic, ergotic, and epistemic. Semiotics are used to communicate information, ergotics are used to manipulate the physical world, and epistemics are used to learn from

the surroundings. Communicating with computers only includes empty-handed communication, which is semiotic gestures. Semiotic gestures can be categorised into symbolic, deictic, and iconic gestures. Symbolic gestures have a single meaning, such as the “OK” gesture in Figure 1 meaning that everything is okay. Deictic gestures are pointing and directing to do something, for example in Figure 1 by pointing something such as pen. Iconic gestures give information about an object's size or shape, in Figure 1 the iconic gesture tells a size of something. Design in this project is concentrated on symbolic and deictic gestures because these gestures are used the most in human-computer interactions. [3,2-3.]

Figure 1. Symbolic, deictic and iconic gestures.

There are a variety of advantages to using symbolic gesture interaction: natural interaction, terse and powerful interaction, and direct interaction. Natural interaction means that gestures are easy to use and natural. Terse and powerful suggests that a single gesture can be used as a command and its parameters. Direct interaction means that using the hand as a device exiles the use of another device. [3,2-3.]

There are also problems with symbolic gesture interfaces. The user needs to memorise the hand gestures because the hand signs do not reveal their meaning. There is also a segmentation problem, meaning all the user's gestures are recognised. This becomes an issue because hand movements come so naturally in everyday life that accidentally, the user may make an unintended hand movement, which will throw off the interaction with the computer. [3,9.]

2.2 Gesture Based Typing

Input techniques for computers also change as they get more advanced. Therefore, new typing methods have been developed. Speech recognition systems, keyboards, touchscreens, and gesture recognition systems exist. Technology is going its way to make electronic devices smaller, making inputting systems such as touch screens and keyboards way too big for use. Gesture recognition would solve this problem, but it is less accurate than keyboards or touchscreens. [4,1.]

Wearable devices, cameras, and sensors have been created for gesture-based typing. For user experience, hand gesture typing methods are natural but slow and tiresome compared to touch screens and keyboards. [5,35.]

2.3 Virtual and Augmented Reality

Augmented reality is an overlay which shares digital information with the real world. Augmented reality blends virtual elements with the user's physical environment. Augmented reality enhances real-world experience with content such as images, videos, and text. This can be experienced through various smart devices. According to its definition, augmented reality is any system that mixes virtual and real-world elements, allows for real-time interaction, and is three-dimensionally registered. [5,5.]

Virtual reality creates an entirely artificial environment for the user, where the user is completely cut off from the real world. The real world is placed in a computer-generated environment. The main difference between augmented and virtual reality is that virtual reality creates its surroundings, while augmented reality keeps the real-life surroundings but adds overlays onto the environment. [5,5.]

2.4 Introduction to Unity

Unity is a game engine, a software tool that makes it possible to produce interactive digital content in real-time. It also has a code structure that allows operating on many platforms, such as computers, smartphones, and consoles. Game engines enable the developers to focus on designing and creating. Meanwhile, the game engine does all the hardware tasks, such as rendering. Game engines make game design much easier because of their toolbars, interfaces, and customisable techniques. With Unity, it is possible to create video games, applications, models, simulators, and software. What makes Unity an excellent game engine is its low cost, noob friendliness, efficiency, and standardised tools. [6,9-12.]

Unity is not the only game engine capable of using in this hand gesture typing system. Ultraleap also supports native LeapC and Unreal engine for LMC related projects. LeapC is a library that communicates with the developers application and the Hand Tracking Service. Unreal engine is also a game engine software tool as Unity but for the reasons stated earlier, Unity as a software tool was chosen for the project. [7,1.]

2.4.1 Unity Editor

The Unity editor has several interactive windows, also called panels, each with its purpose and need. As seen in Figure 2, all key panels are highlighted with different colours. Highlighted in blue is a Hierarchy panel, highlighted in yellow are the Scene and Game panels, highlighted in green are the Assets and Console panels, and lastly, highlighted in red is the Inspector panel. All of the highlighted panels will be explained in this chapter.

Figure 2. Unity editor.

The Hierarchy panel in Figure 3 lists all GameObjects on the Scene, but in a hierarchical order. The GameObjects contain a so-called “parent-child” relationship, which enables sub-objects. For example, the thesis project is called Hand Sign Typing System, and it has a GameObject called Canvas; the Canvas is a parent object, and under it is an InputField, a child object. The InputField can also contain more child objects. All the GameObjects seen in Figure 3 are part of this project, and each corresponds to something on the Scene. [8,19-20.]

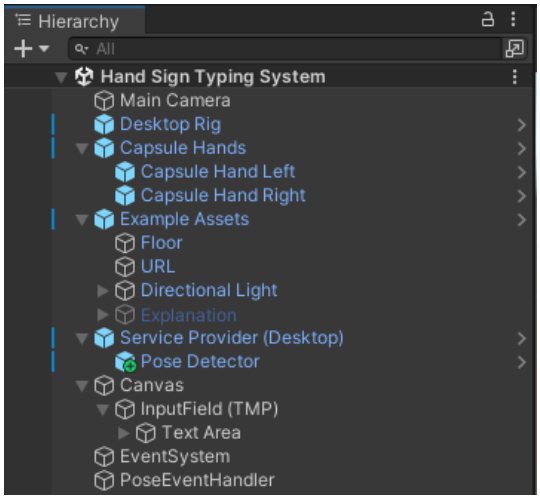


Figure 3. Hierarchy panel.

The Inspector panel seen in Figure 4 is the actual editor of the project. The Inspector panel is used to change the GameObjects parameters and properties. The Inspector panel looks different with every GameObject because GameObjects have different purposes and modifiability. For example, in Figure 4, there is a GameObject called Pose Detector opened, in which the script handles the gesture recognition; the script creates its parameters and properties, but they are adjustable inside the Unity editor so that the script does not need to be changed separately. [8,21-22.]

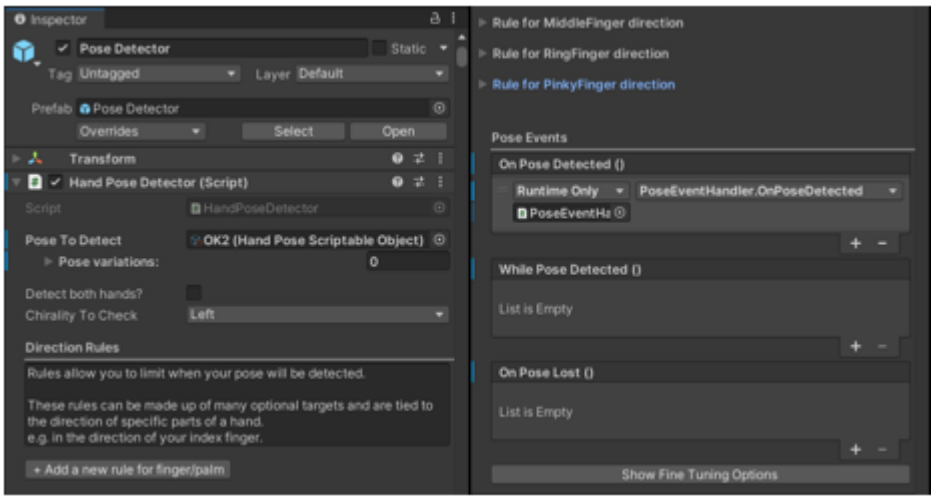


Figure 4. Inspector panel.

The Scene window in Figure 5 showcases the whole Unity project. The Scene is a stationary background that visually contains all Game Objects. The Scene and the GameObjects inside it can be moved. The Scene also has some perspective and lighting settings that are irrelevant to this project. [8,19.]

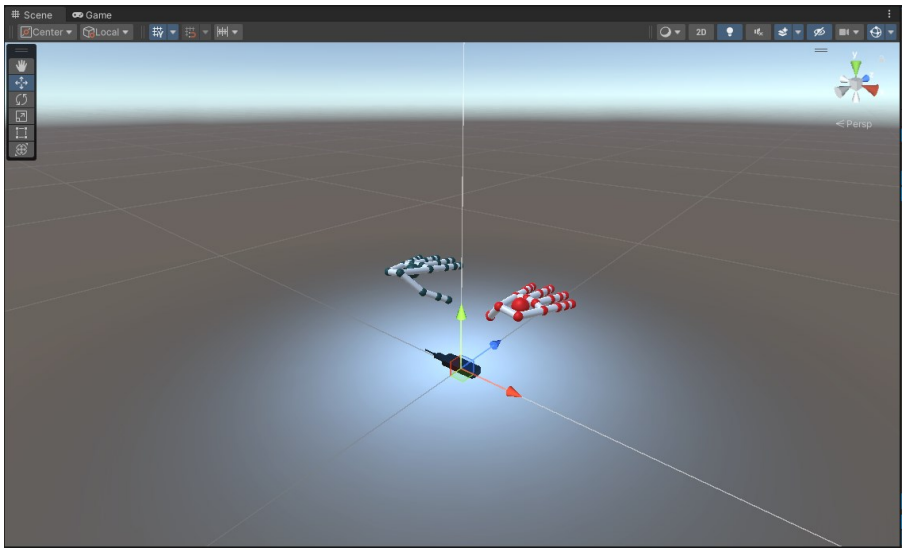


Figure 5. Scene window.

The Game window in Figure 6 is the same as the Scene window, but this is the actual game, meaning the design looks like this when the project is started. The game is shown from the camera's point of view. The Game and Scene window is vital to the Unity editor because it showcases the result. In the projects design, the Scene and Game window are not as crucial as in making a game because the projects design concentrates on the typing mechanism and text form. [8,19.]

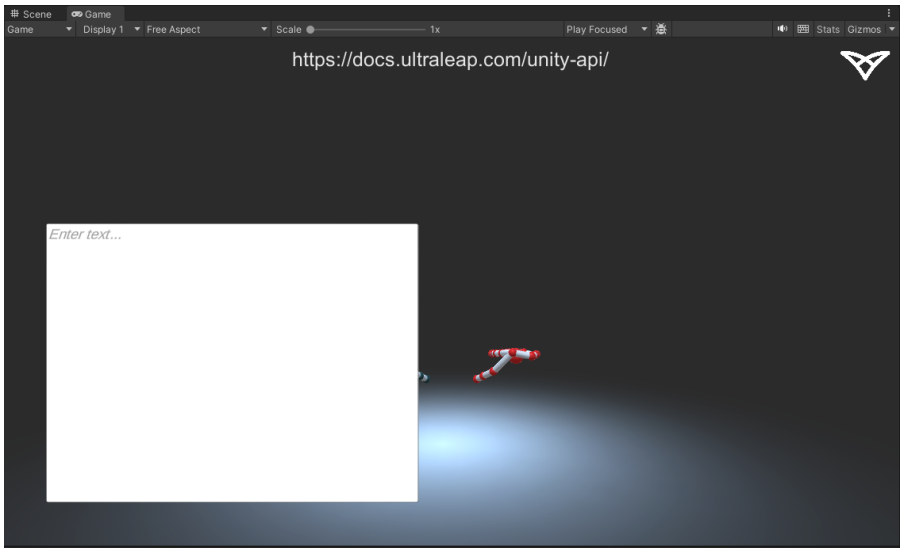


Figure 6. Game window.

The Project panel, as seen in Figure 7, contains assets. Assets are folders that contain scripts, pictures, files, and scenes. They are not part of the project but can be dragged to it and modified in the Inspector panel. The Project panel works the same as in Computer File Explorer. [8,20-21.]

Figure 7. Project panel.

As seen in Figure 8, the Console panel is next to the Project tab. It shows outputs from the design, including errors and warnings. The Console panel helps in debugging the project. [8,21.]

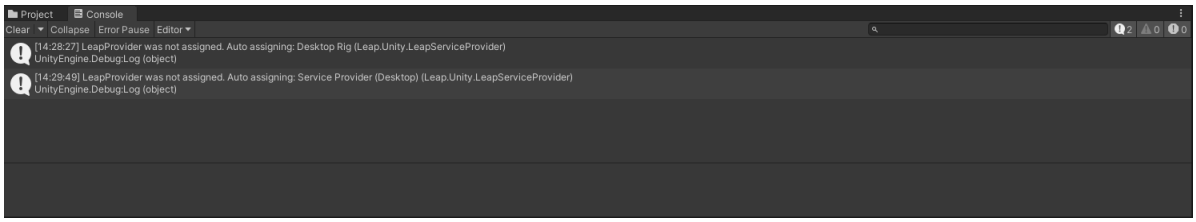


Figure 8. Console panel.

2.5 Introduction to the LMC

The LMC is a small device that tracks hand and finger movements in three-dimensional space. The LMC sits on a desk and uses its large field of view to detect hand positions in high precision. The LMC comes with a USB for sending the captured data to the computer. The data is used by Ultraleap's advanced algorithms and it creates a virtual space where the joints and bones of the hand can be seen. In the following sections, the capabilities and operation of the LMC will be thoroughly explained.

2.5.1 LMC Parameters

A visual hand-tracking device, the LMC by Ultraleap, records users' hand and finger movements to enable natural interaction with technology. The controller has three infrared light-emitting diodes and two infrared light cameras (Figure 9). The infrared light-emitting diodes are seen in red, and the two cameras are between the middle and outer ones. The controller dimensions are 80x30x11.30 millimetres (x, y, z) and weight of 32 grams, which makes this device small and compatible. The device is also fast with its two infrared cameras that operate at 120Hz and can capture movements in 1/2000th of a second. The device gets its 5V DC power from a USB connection. [9,1-2.]

Figure 9. A picture of the LMC

The device interacts in a 3D zone with a field of view 140x120° degrees, and the 3D zone reaches up to 80cm from the device. Environmental conditions do not make a difference when operating this device. The device separates a human hand into 27 parts: bones and joints. All 27 joints and bones are observed, even though other hand elements block the view. As seen in Figure 10 all 27 joints are visible and indicated with these orange dots. The device can be used in various applications such as virtual reality, Windows, and displays. [9,1-2.]

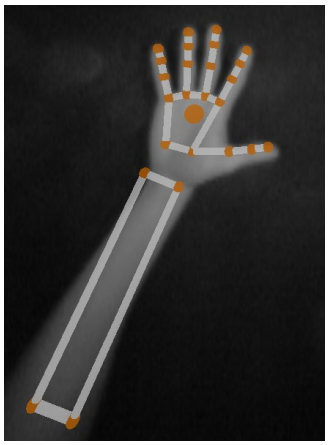


Figure 10. A picture from LMC point of view with hands 27 joints and bones visible.

The LMC is as accurate as a human hand, averaging 0.4mm, and the LMC is averaging 0.7mm. The accuracy measurement means that the LMC chooses correct position in the 3D space compered to real life. Also, the repeatability with the device is averaging below 0.17mm. Repeatability means determining the accurate position multiple times in the 3D space. According to these parameters and specs, the LMC was an excellent choice for the projects design because of its robustness, accuracy, and repeatability. [10,12.]

2.5.2 LMC Fundamentals

The LMC is designed for hand gesture and finger positioning software applications. The controller works by tracking the infrared light at 850 nanometers wavelength, which is invisible to human eyes. The light-emitting diode pulse and the two cameras framerate sync, creating the interaction zone. This manoeuvre increases the controller's intensity and lowers its power use. In detail, the light-emitting diodes send light to users' hands, and the two cameras spot the difference in light compared to the environment and send this captured data via USB to the Ultraleap's hand-tracking software. In Figure 11, the data is seen in the Ultraleap Control Panel, which shows the raw footage of the LMC. The footage is grey because of its infrared light-emitting diodes. [11,1.]

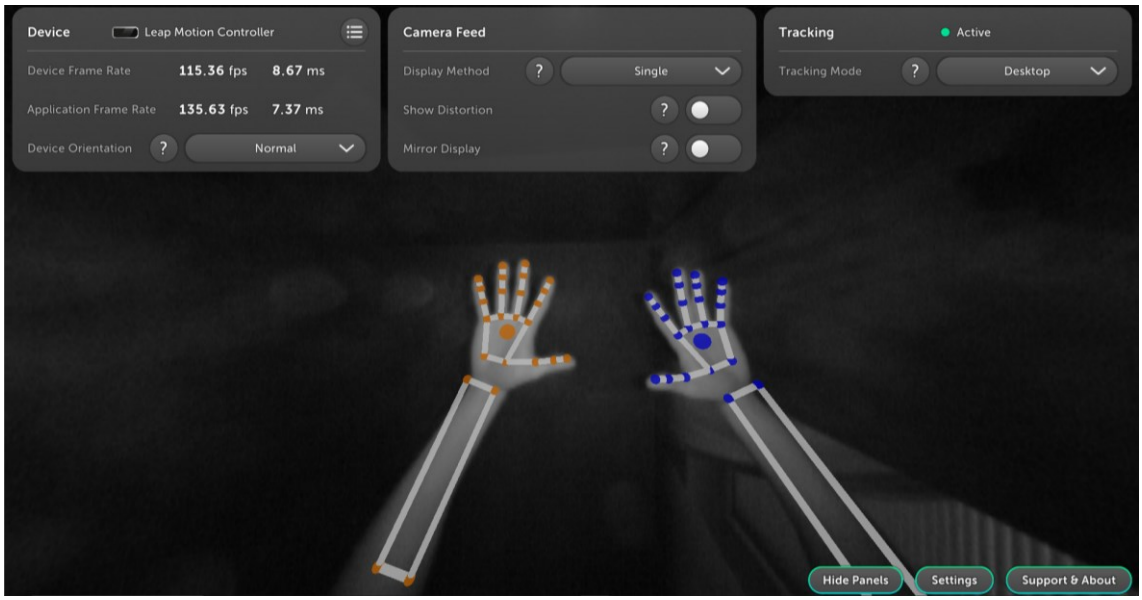


Figure 11. Ultraleap Control Panel.

The tracked hands in Figure 11 are tracked into virtual hands with a skeleton model. The skeleton model shows the positions of single bones, such as the metacarpal, proximal phalanx, intermediate phalanx, and distal phalanx. These names are for the joints in the finger from closest to the palm to the farthest. The distal phalanx is the tip joint of the hand. In Figure 11, other bones and joints are also seen, such as the arm bones, but they are not relevant to this project because the concentration is on hand and finger movements. [12,12-14.]

When the picture data reaches the computer, then the data is handled by an advanced proprietary algorithm. The algorithm uses filtering techniques to filter objects such as fingers from the data. Subsequently, the algorithm sends the results into a protocol as frames and snapshots. The protocol enables a service via transmission control protocol or WebSocket connection and communicates with the Leap Motion Control Panel and native and web client libraries. The Leap Motion Control Panel is the software that contains the picture data from the controller and can adjust the tracking settings (Figure 11). The client library supplies helper functions and classes, handles frame history, and arranges the data into an object-oriented API structure. Consequently, a motion-controlled interactive experience is made possible by the application logic ties into the Leap Motion input. [12,12-14.]

3 System Design and Implementation

This chapter will showcase the design and implementation of the hand gesture typing system in the Unity engine. This chapter builds on the theoretical framework of the previous chapter. It covers the architecture of the system, the integration of the LMC into Unity, the gesture recognition technique, and the translation of hand movements into alphanumeric characters. By addressing the design and implementation aspects of the hand gesture typing system, this chapter aims to provide insights into the development process and technical considerations.

3.1 Overview of the Hand Gesture Typing System

The hand gesture typing system comprises the LMC, computer, LeapSDK, and Unity software. The LMC captures hand movements. The computer handles the system process so the LeapSDK and Unity can run without issues. The LeapSDK handles the data from the LMC and sends it to the Unity software. Unity then has various scripts that use the corresponding data to the application. All four parts of the system have a significant role in the project.

The hand gesture typing system aims to provide a natural and engaging way to input text within virtual reality environments. Using hand gestures, users can type and interact with virtual keyboards without using actual keyboards or controllers. Removing the obstacle of conventional text input techniques improves the user experience in virtual reality applications.

The systems' functionalities are real-time gesture recognition, virtual keyboard interface, text inputting, language support, and integration with virtual reality applications. The system uses advanced gesture recognition algorithms to detect and interpret hand movements for real-time gesture recognition accurately. The virtual keyboard interface is presented within the virtual reality environment, allowing users to see and interact with alphanumeric characters using hand gestures visually. Users can input text by performing gestures corresponding to

individual characters. The system supports multiple languages and character sets for language support, enabling users to input text in their preferred language. Integrating with virtual reality applications, the systems support other LMC applications, allowing developers to utilise this virtual hand gesture typing system.

This virtual hand gesture system is intended for virtual reality prototyping. Developers can employ the hand gesture typing system in Unity to prototype virtual reality applications, allowing them to input text during the development process. Additionally, the system supports virtual games based on Unity using LMC. The hand gesture typing system enables users to interact with in-game interfaces, traverse menus, and enter text-based commands. This method can also improve in-game communication by allowing the players to converse with one another.

3.1.1 Architecture of the System

The hand gesture typing system consists of four critical scripts: pose detector, pose event handler, hand pose scripts, and the text field. All these scripts interact with each other in the project. The pose detector uses the hand poses to recognise hand poses, the pose event handler script creates an action when the hand pose is detected, and the characters are inputted into the text field onto the scene. Figure 12 shows the system's architecture: It has layered all the scripts and game objects onto the tab.

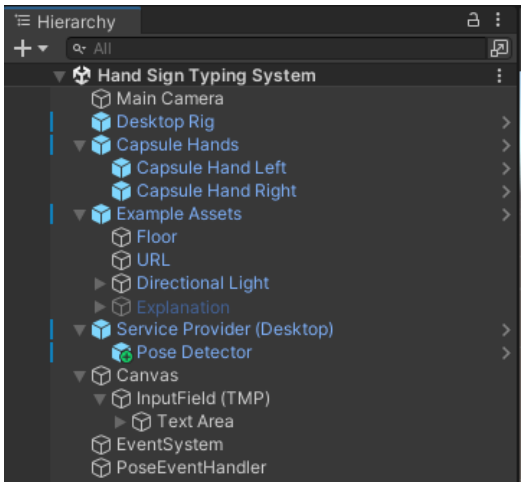


Figure 12. Project hierarchy.

The hierarchy starts at the top from the “Hand Sign Typing System”, which is the project's name. Under the project folder in Figure 12, there are eight key GameObjects: Main Camera, Desktop Rig, Capsule Hands, Example Assets, Service Provider, Pose Detector, Canvas, and Pose Event Handler. All of these GameObjects will be explained below.

The Main Camera is the user's point of view of the project; it shows everything on the scene. Desktop Rig and Service Provider are prefabs that provide hand-tracking data to the Pose Detector software. Prefab is a pre-configured GameObject that can be customised in the Unity editor; it allows creating a template for GameObjects with specific components, properties, and behaviours, and then copies of that template are created in the scene. The Pose Detector is a script for detecting hand signs made by the user. Capsule Hands is a visual look at the hands in the scene. They can be customised, but I kept them as default because it made visually inspecting the hand signs easier. Example Assets is a GameObject that contains the light feature and the floor design onto the scene. The Canvas overlay includes the input field where the characters are input from users' corresponding hand gestures. Lastly, the Pose Event Handler script handles the algorithm inputting the characters into the text field.

3.1.2 Integration of the LMC with Unity

The integration of the LMC with Unity started by downloading Unity's newest version and Ultraleap Gemini Hand Tracking Software. After that, the Ultraleap package was added to the Unity Package manager and added example content from the Package manager to the Unity editor. At this point, the project had two hands on the scene, and when play was pressed on the game, it tracked hand movements from the LMC. The Ultraleap Package Manager has many examples of tracking applications, hand designs, Unity scene interfaces, and other assets like hand poses. [13,1.]

Consequently, the 3D project was created in Unity with the packages and samples set up. A service provider Desktop version needed to be added for the project because the LMC is the desktop version. This service provider allows the Pose Detector script to be added to the project and lets the script handle the Service provider's tracking data.

3.2 Gesture Recognition

There were two plugin features for capturing hand gestures; the first was hand pose recording, where every pose used on the project could be recorded and saved. The second feature chosen was pose detection, which enabled Ultraleap to be chosen, as it already made hand gestures. Six hand gesture scripts were already made: fist, horns, pinch, open palm, point, and thumbs up. Only fist, pinch, point, and thumbs up were used in the project. These four hand gestures helped in creating twenty-six gestures by modifying each hand gesture script. The Pose Detector script needed to be changed for each hand gesture made. Overall, twenty-six different scripts were made.

3.2.1 Pose Detector

The Pose Detector was helpful for this project because it enabled the capturing of hand gestures, which were chosen. The Pose Detector is a C# coding

language script that can be modified in the Unity editor. The script can be altered in various ways in the Inspector panel, as depicted in Figure 13.

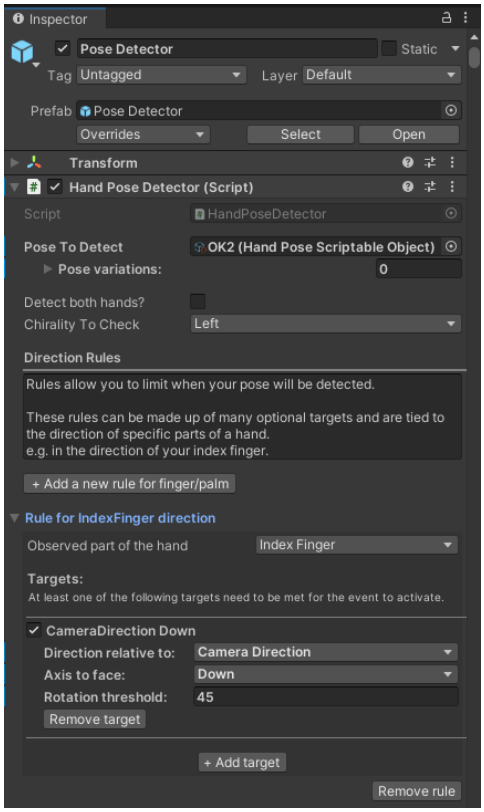


Figure 13. Pose Detector script in an Inspector panel.

For the Pose Detector script to work, some essential modifications must be made. Firstly, the “Pose to Detect” setting chooses which pose the script detects. In Figure 13, the “OK2” hand gesture is chosen, which is the pinch touching the index and thumb together, as seen in the Appendix 1 letter “e”. Then there is a box for choosing which hand to detect called “Chirality to Check”; the left hand was chosen for this instance. The scripts need to be made one by one for both hands.

The direction rules separate each hand gesture from the others by choosing where each finger or palm should be directed. As seen in Figure 13, a rule is selected for the index finger. A target for the index finger is also chosen, with settings such as direction relative to, axis to face, and rotation threshold. The direction where the target is relative is chosen to be the camera direction,

meaning that the LMC is on the table, and the camera on the LMC is looking up. Axis to face setting is down, meaning the index finger should point to the LMC. The rotation threshold determines how much rotation is required. The rotation threshold setting helps in filtering out unintended movements that might occur. The rules and targets are made to distinguish between hand gestures so they do not overlap. Some hand gestures are similar, and the only thing that separates them is a single-finger position. Therefore, it is a must to change the rules accordingly.

In the Inspector panel are so-called “Pose Events”, as seen in Figure 14. Pose Events is a developer tool that allows the easy modification of the Pose Detector script. There can be chosen when, how and how long the event occurs. On Pose Detected means that when the pose is detected, the event occurs, while Pose Detected implies that when the pose is detected, the event runs until it is lost, and On Pose Lost means what happens after the pose is lost. These events have some modifications available, such as whether the event occurs only in runtime, which script it runs, and what command it runs. In this instance, the goal was to make the hand sign print out a character. Therefore, when the pinch gesture is detected at runtime, it only runs the Pose Event Handler script.

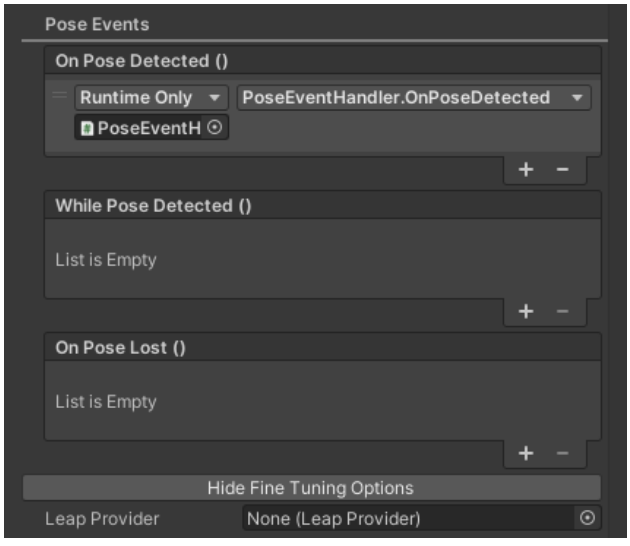


Figure 14. Pose Events

3.2.2 Recognizing Hand Gestures

For the LMC, hand gestures were easy to detect but difficult to separate. Therefore, specific rules and detection patterns were created for the hand gestures so they do not overlap. For making hand gestures in the Unity editor, there are possible modifications such as Fingers to Detect, Finger Joint Rotation Threshold, and each finger's own Joint Threshold (Figure 15). The Fingers to Detect setting means which finger the script concentrates on. For example, if the Finger to Detect is just the index finger and thumb, the script follows them and does not care about the other fingers. The Finger Joint Rotation Threshold is the minimum amount of rotation required to register a change in the orientation of a finger joint. The lower the value, the easier it is to detect the rotation of finger joints, and if the value is high, then the tiny rotations of finger joints are not detected. The last parameter allows the change of individual specs of these joint thresholds in each finger. [14,64-66.]

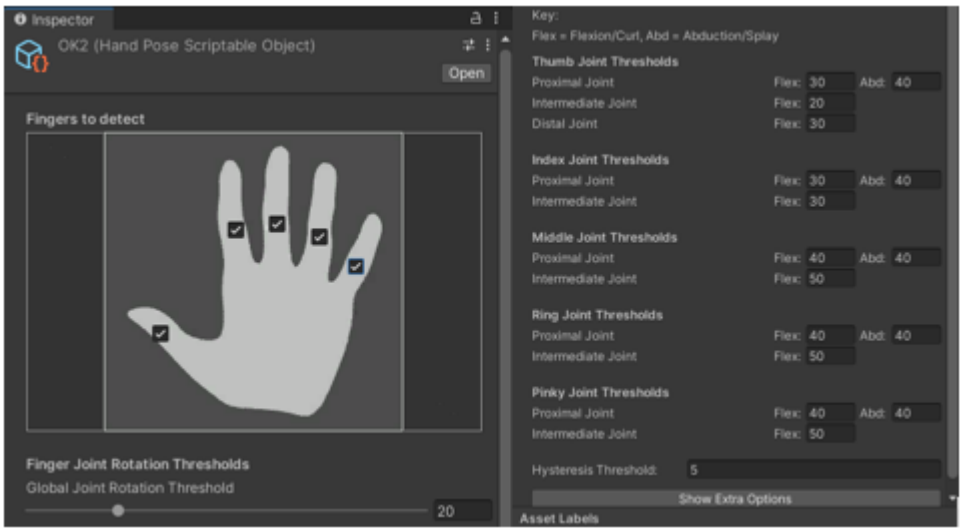


Figure 15. Hand pose script in Inspector panel

In Figure 15 these joints' “flex” and “abd” values can be changed, which refers to the finger's flexibility degree. The higher value in flexing the joint indicates more significant bending. The “abd” means finger abduction, which is spreading fingers apart. The higher the value is, the farther the spreading is. Additionally, there are named joints in each finger's parameters. These are proximal, intermediate, and

distal joints. The proximal joint refers to the joint closest to the palm, the intermediate joint is the middle one, and the distal joint is the farthest in the fingers seen in the Figure 16. [14,65.]

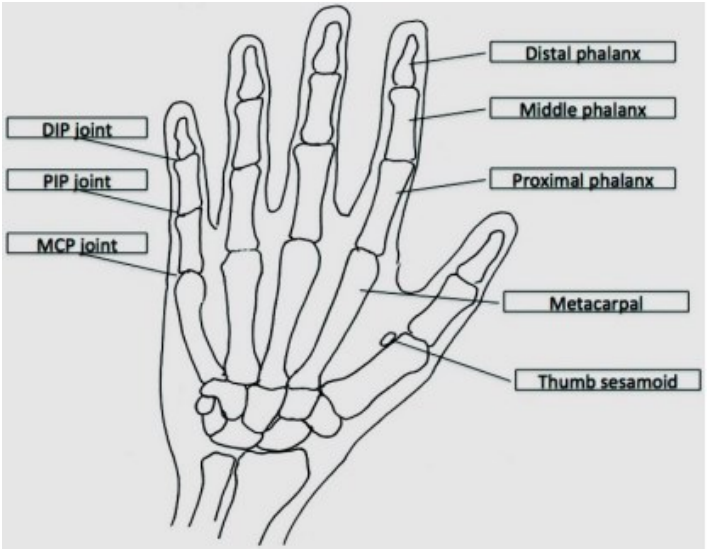


Figure 16. Human hands joints named. Reprinted from the study [14,64].

3.2.3 The Hand Gestures

The first hand gesture created was the pinch gesture, as seen in Figure 17. This hand gesture was chosen because it allowed the creation of eight similar gestures. The pinch gesture involves connecting the index finger and thumb. It was created by modifying the preexisting "OK" hand gesture, while keeping the same threshold parameters in the Pose Detector settings. Changes were made to the detection rules so that the index finger needed to point down at the LMC, the other fingers should move forward from the user, and the palm should point downward at the LMC. These adjustments in the settings allowed this hand gesture to be distinguished from others, with the script only detecting it when the hand gesture was precisely executed. Figure 17 displays the pinch gesture, with all its hand joints visible.

Figure 17. A pinch gesture with the thumb and index finger connecting.

The other 28 hand gestures were created in the same way as the pinch gesture. The ready-made hand gesture asset was chosen and modified to work with the setup and design. The other gestures needed more rule changes than the pinch because they are more like each other. As seen in Appendix 1, five different types of hand gestures were created: the first section is the pinch gestures, the second section is palm down, the third section is palm back, the fourth section is palm up, and the last section is the extra ones with the palm facing right or left. These gestures were put into five sections because each section has its similarities. The first is just pinching with different fingers; the palm down section has hands in the same position, but the finger position changes; the palm back and palm up follow the same formula as the palm down section. Lastly, the extra ones have palms open-faced to the other right or the left of the axis to face.

3.3 Mapping Gestures to Alphanumeric Characters

In the project designing phase, many ways were considered for mapping the gestures to alphanumeric characters. The QWERTY system was a viable option, also known as the keyboard layout, but it was not a great fit for mapping the gestures. The decision was to map the gestures in a descending order based on the English alphabet's most used letters because this allowed to creation of a defined order for the gestures. [15,18.]

In Appendix 1, the first four sections of the hand gestures have English letters in descending order, starting with the most frequent letters in the English alphabet. The first section, pinch, has the eight most frequent letters, and the palm down

section has the nine to fifteen most used letters. The eight most frequent letters are e, a, t, i, o, n, s, and r, with the letter “e” having 12.15% frequency and the letter “r” having 6.62% frequency. The last section, the additional ones, does not print letters; they are different from the other hand gestures, so they have their section—these first hand gestures in section five print a space between the letters to separate words. The second hand gesture works the same as backspace on the keyboard, deleting a previous letter if the user misinterpreted one. [15,19-24.]

Appendix 1 is made to be used as a layout to learn the letters used by the design. The pinch hand gestures were put in the first section because they keep the hand and finger positions almost identical. The palm down gesture was put on the second section because the palm down and pinch gestures keep the palm down in both scenarios, which allows to not constantly rotate the hand. The palm back gestures are in the third section for the same reason as in the palm down because the hand rotation is insignificant. The palm up gestures were put on to the last section because the LMC takes time to recognise these hand gestures. It occasionally mixes the palm-up gestures with the palm-down gestures; for this reason, the palm-up gestures are put on with the least frequent letters. The most frequent letter in the palm-up section is the letter “v”, with a frequency of 1.06%, meaning that almost every hundred letters are the letter “v”, making it a rare letter in the English alphabet. The other letters in the palm-up section are even rarer, which means the palm-up hand gestures are not used as much as the other hand gestures. [15,19-24.]

3.3.1 Pose Event Handler

For inputting and deleting characters from the text field, a Pose Event Handler script was created, which inputs and deletes characters from the text field, manages pose cooldown and hold time, and allows the user to choose whatever character to input into the text field. This script was created in Visual Studio 2019, a coding platform supported by Unity. The following text will go deeply through the functions and functionalities of the script to gain a deep understanding of the script (Listing 1).

```
using UnityEngine;
using TMPro;

public class PoseEventHandler: MonoBehaviour
{
    public TMP_InputField textField;
    public char characterToAppend = 'a';
    public float poseCooldown = 1.0f;
    public float poseHoldTime = 1.0f;
    private float lastPoseTime;
    private float poseStartTime;
    public void OnPoseDetected()
    {
        if (Time.time - lastPoseTime >= poseCooldown)
        {
            if (Time.time - poseStartTime >= poseHoldTime)
            {
                AppendCharacter(characterToAppend);
                poseStartTime = Time.time;
                lastPoseTime = Time.time;
            }
        }
    }
    public void AppendCharacter(char character)
    {
        textField.text += character;
    }
    public void Backspace()
    {
        if (textField.text.Length > 0)
        {
      textField.text = textField.text.Substring(0, textField.text.Length - 1);
        }
    }
}
```

Listing 1. Pose Event Handler script in C# coding language.

The first two lines of the script provide access to Unity's scripting API and TextMeshPro classes and functionalities. The following line creates a new script called PoseEvent Handler. It also includes a built-in class called MonoBehaviour that allows the script to be used in Unity (Listing 1). [16,1.]

The "public TMP_InputField textField;" line creates a space where the text box can be put in the projects scene. The following line decides which character should be added to the text field; by default, it is "a". The *poseCooldown* line decides how long to wait before a next-hand gesture can be detected; it is set to one second. The *poseHoldTime* indicates how long the hand pose should be kept to detect a pose. These lines are set to public because it allows me to change their variable in unity environment. The following two private lines create a secret

place to keep track of time; they are used for remembering when the last poses happened. [17,1.]

The following line, called “public void OnPoseDetected()”, creates an action that activates when a pose is detected. The following five lines of code are all part of this one action. The following lines are IF statements, which check that everything is accordingly to print out the chosen character. The first IF statement checks if enough time has passed since the last pose; this helps avoid detection errors so that hand signing is fluent. The second IF statement checks if the pose has been held enough time. This IF statement is necessary because many hand gestures are like each other, preventing them from being misinterpreted. After the IF statements are factual, the following line of code is called “AppendCharacter(characterToAppend);” it is called the *AppendCharacter* method and passes the *characterToAppend* as an argument. So when the line of code is executed, it triggers the *AppendCharacter* method, instructing it to append the character stored in the *characterToAppend* variable to the text field. The other two lines inside the IF statement update the time value of the start of the current pose and the time of the last pose detected.

The following line, called “public void AppendCharacter(char character)”, creates an action which allows the code inside the brackets to input the specified character into the text field.

The *public void Backspace()* line of code creates an action that handles the backspace functionality. The IF statement following it checks if the text field contains any characters. If the IF statement goes through, the following line removes the last character by setting the text into a substring that excludes the previous character.

The script is simple to modify because a feature was added that enables changing the script in the Unity editor (Figure 18). In Unity, it is possible to choose where to input character and which character to input. In the project the Input Field and character “a” and “b” was selected. The pose cooldown and hold time

can also be modified from the inspector panel. This feature was created in the script because it allowed to change the parameters in the Unity editor without changing the whole script for each hand gesture individually.

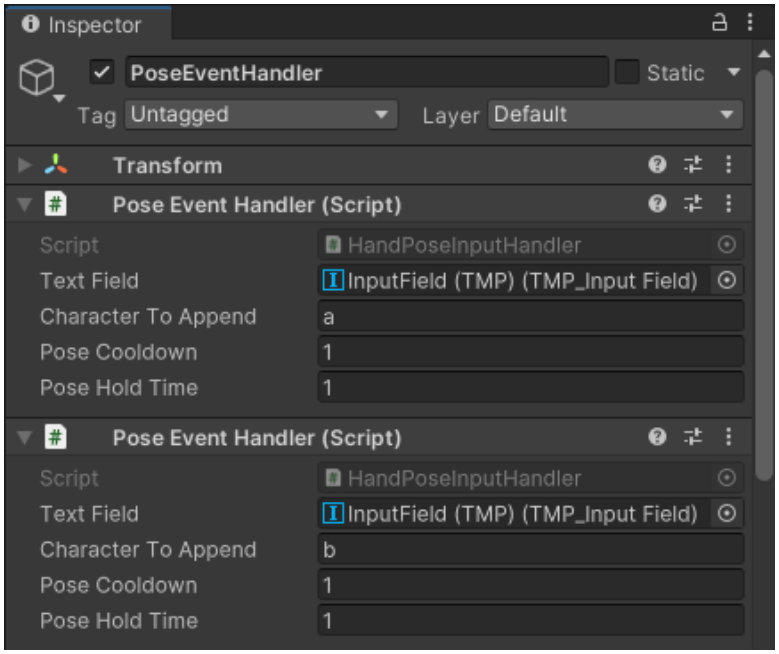


Figure 18. Pose Event Handler script in Inspector panel.

3.3.2 The Text Field

The design needed a text field where the letters were inputted, this was choosen from the UI toolbar in the Unity editor. As seen in Figure 19, the text field can be customised in many ways. In the project the point size, character limit, line type, text field size and coordinates in the text field were changed in the inspector panel. The point size is the character size of the letters; it was changed to 30 so that the text is readable to the user. A character limit was added to the text field so that the text does not get more extended than the text field. Then the line type was changed to Multi Line Submit so that the text is not on the same line and does not get over the text field. Lastly, the coordinates and size of the text field was changed so that the text field is visible on the Game panel and does not fill the whole screen because otherwise, the hand model could not be seen. The other settings and parameters in Figure 19 are irrelevant to this project.

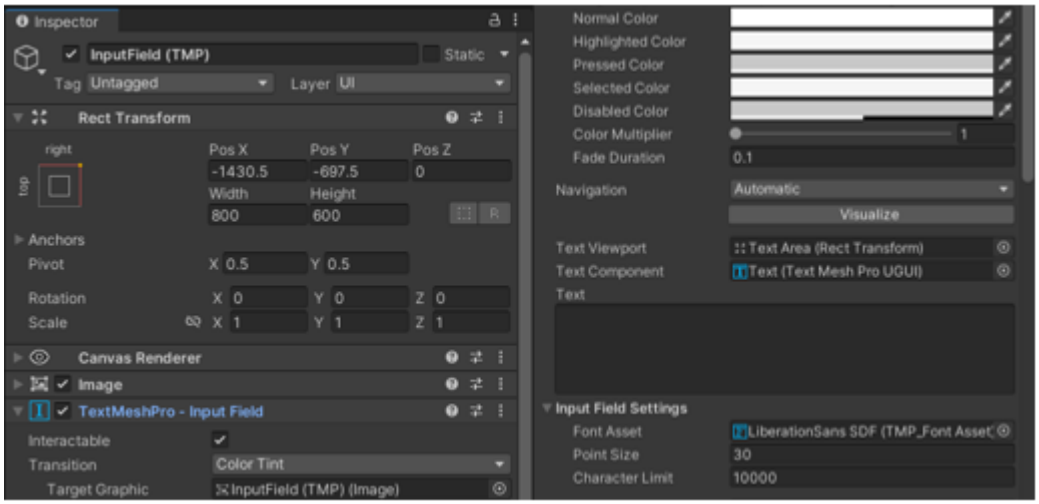


Figure 19. The text field in the Inspector panel.

4 Future Work

This chapter will discuss the potential enhancements for the hand gesture typing system developed in the Unity engine. Despite the system's functionality, the current setup must address certain limitations. These include issues with gesture recognition accuracy, restricted language support, system usage limited to the Unity engine, and the lack of sophisticated text editing tools. To better meet users' diverse needs, the system may also require enhancements in usability and customisation possibilities.

The accuracy of the LMC could be improved with a better gesture recognition device because the LMC has difficulties tracking finger joints. In the last section of the hand gestures in Appendix 1, where the palm is faced up, the LMC has difficulties to recognise the hand gestures because the fingers are obscured by the palm. This issue negatively affects the virtual typing system, slowing the typing speed and accuracy. To improve this issue, Ultraleap has released a new, better hand-tracking device, called the Leap Motion Controller 2, with better advancements, such as a larger interactive space and better hand-tracking capabilities. [18,1.]

In the hand gesture typing system, there are 26 hand gestures corresponding to different letters, but this is not enough for all languages. This issue could be improved by creating more hand gestures. This improvement could enhance the language support of the system.

The project design can type small letters, add spaces, and delete characters. With more hand gestures added to the system, it could have more text editing tools, such as Shift, for typing large letters and other characters available on a keyboard.

Currently, the project design can only be used with the Unity engine. This could be improved by making it available with other software and operating systems like Windows and adding more capabilities.

Lastly, the system's usability and customizability could be enhanced by creating options to personalise users' typing experience, such as gesture mapping preferences, keyboard layout customisation, and accessibility features for users with specific needs. The projects current design has options to personalise the typing experience, with the ability to customise 28 different hand gestures, but this is not enough for advanced typing systems.

In conclusion, the potential enhancements outlined in this chapter offer promising opportunities for improving the capabilities of the hand gesture typing system. Implementing the proposed enhancements this system can be further improved by the functionality, usability, accessibility, and ultimately improving the user experience in virtual reality environments.

5 Conclusion

In this thesis, a working typing method for Unity was successfully developed, which supports multiple languages and has 28 hand gestures. In the implementation phase of the project, several key findings and contributions have emerged.

This thesis project contributes to the field of gesture-based interaction and user interface design, by demonstrating the potential of a hand gesture typing system as a natural input method for virtual and augmented reality environments. The creation of this multilingual hand gesture typing system opens possibilities for interaction within virtual reality environments and increases the potential of virtual interfaces.

The project's research and implementation efforts have resulted in some notable findings. With the Unity engine, a working system was able to be designed by integrating the LMC and utilizing Ultraleap's pose detection algorithms. This integration created a solid gesture recognition foundation for the text input system. In addition, resolving technical difficulties, improved user experience by allowing character sets to be customized and highlighted the potential for further research and development. Collectively, these important discoveries demonstrate that the multi-language hand gesture typing system in the Unity engine could be implemented and highlight areas that require more research and development.

During the implementation phase of the project, certain technical restrictions were encountered, such as the LMC's inaccuracy in gesture recognition and the need for better user interface and experience design. Although these difficulties were overcome during the development phase, opportunities remain to enhance and maximize the system's potential.

In conclusion, this thesis project demonstrates the potential of a multilingual hand gesture typing system in virtual and augmented reality environments. By designing and implementing a hand gesture typing system that supports multiple languages and includes 28 hand gestures for typing, the project contributes to the fields of gesture-based interaction and creates new possibilities for future research and development.

References

1 MDPI (2020) Hand Gesture Recognition Based on Computer Vision: A Review of Techniques, (July 23) [WWW document] https://www.mdpi.com/2313-433X/6/8/73 (Accessed March 15, 2024)

2 ScienceDirect (2015) Human Computer Interaction using Hand Gesture, (August 21) [WWW document] https://www.sciencedirect.com/science/article/pii/S187705091501409X (Accessed March 17, 2024)

3 BillBuxton (2018) Gesture Based Interaction, (May 18) [WWW document] https://www.billbuxton.com/input14.Gesture.pdf (Accessed March 19, 2024)

4 ACM Digital Library (2015) TypingRing: A Wearable Ring Platform for Text Input, (May 18) [WWW document] https://dl.acm.org/doi/10.1145/2742647.2742665 (Accessed March 21, 2024)

5 Diva Portal (2017) Interacting with Hand Gestures in Augmented Reality: a Typing Study, (June 9) [WWW document] https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1138555&dswid=-5130 (Accessed March 22, 2024)

6 Benjamin Nicoll & Brendan Keogh (2019) The Unity Game Engine and the Circuits of Cultural Software. Palgrave Pivot Cham

7 Ultraleap (2024) Ultraleap documentation, [WWW document] https://docs.ultraleap.com/xr-and-tabletop/tabletop/index.html (Accessed March 27, 2024)

8 Jared Halpern (2019) Developing 2D Games with Unity. California: Apress Berkeley

9 Ultraleap (2024) Leap Motion Controller overview, [WWW document] https://www.ultraleap.com/datasheets/Leap_Motion_Controller_Datasheet.pdf (Accessed March 30, 2024)

10 National Library of Medicine (2013) Analysis of the Accuracy and Robustness of the Leap Motion Controller, (May 14) [WWW document] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3690061/ (Accessed April 4, 2024)

11 Ultraleap (2020) How Hand Tracking Works, (September 3) [WWW document] https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/ (Accessed April 6, 2024)

12 MDPI (2018) Review of Three-Dimensional Human-Computer Interaction with Focus on the Leap Motion Controller, (July 7) [WWW document] https://www.mdpi.com/1424-8220/18/7/2194 (Accessed April 10, 2024)

13 Ultraleap (2024) Ultraleap documentation, [WWW document] https://docs.ultraleap.com/xr-and-tabletop/tabletop/unity/getting-started/index.html (Accessed April 15, 2024)

14 IEEE Xplore (2018) Comparison of hand gesture inputs of leap motion controller & data glove in to a soft finger, (January 11) [WWW document] https://ieeexplore.ieee.org/document/8250099 (Accessed April 20, 2024)

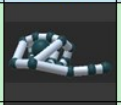
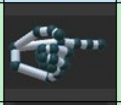
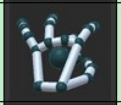
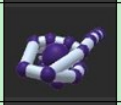
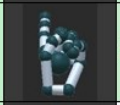
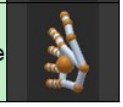
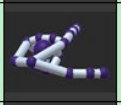
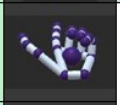
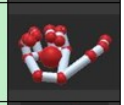
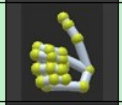
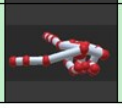
15 Ideas Spread (2018) Letter Frequency Analysis of Languages Using Latin Alphabet, (March 26) [WWW document] https://j.ideasspread.org/index.php/ilr/article/view/14 (Accessed April 22, 2024)

16 Unity Documentation (2024) Creating and Using Scripts, [WWW document] https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html (Accessed February 20, 2024)

17 Unity Documentation (2024) Variables and the Inspector, [WWW document] https://docs.unity3d.com/Manual/VariablesAndTheInspector.html (Accessed February 20, 2024)

18 Ultraleap (2024) Leap Motion Controller 2, [WWW document] https://leap-2.ultraleap.com/ (Accessed April 27, 2024)

Hand Gesture Table

Pinch		Palm Down		Palm Back		Palm Up		Palm Right/Left	
Character	Gesture	Character	Gesture	Character	Gesture	Character	Gesture	Command	Gesture
e		h		f		v		Space	
a		l		p		k		Backspace	
t		d		g		x			
i		c		w		j			
o		u		y		z			
n		m		b		q			
s									
r