

Proyecto Final de Realidad Virtual

“Simulador de ingreso a domicilios para entrenamiento policial”

Alumnos:

Juan Pablo Sibecas - 12224

Gabriel Vitelli - 13001

Matías Gaviño - 13019

Introducción

En el mundo actual, la formación y el entrenamiento de las fuerzas de seguridad son fundamentales para garantizar la seguridad pública y la eficacia en situaciones críticas. La Realidad Virtual (RV) puede convertirse en una herramienta revolucionaria para el entrenamiento policial, ya que permite la creación de entornos realistas y seguros para simular situaciones de riesgo, como entradas a domicilios en operativos policiales.

Este proyecto tiene como objetivo presentar el desarrollo de un simulador de entradas a domicilio para el entrenamiento policial, diseñado en el motor gráfico Unity para la creación de escenas y la física del juego. El simulador se enfoca en la inmersión del usuario y su interacción con situaciones de alto riesgo, utilizando lentes de Realidad Virtual, específicamente los Meta Quest 2. Además, se desarrolló un chaleco con motores de vibración que actúan como realimentadores hapticos para simular la sensación de recibir disparos, ofreciendo una experiencia más realista y envolvente.

Herramientas utilizadas

Meta Quest 2

En los últimos años ha habido un gran avance tecnológico en cuanto a lentes de realidad virtual, como el Meta Quest.

Posee su propio sistema operativo basado en Android, con una interfaz de usuario que permite navegar fácilmente para iniciar aplicaciones, activar o desactivar componentes, etc.

Tanto los lentes como sus controladores Meta Touch incorporan sistemas precisos de *pose estimation*. Esto permite el seguimiento de las manos del usuario con un nivel de precisión excelente, y sin necesidad de programarlo en cada aplicación. Diseñados para proporcionar interactividad precisa y envolvente, estos dispositivos están equipados con botones, joysticks y sensores de movimiento, ofreciendo a los usuarios un rango completo de opciones para participar activamente en sus experiencias virtuales.

Una de las pioneras técnicas distintivas del Meta Quest 2 reside en su capacidad para rastrear con precisión los movimientos de los usuarios en términos de rotación y traslación. Este dispositivo emplea un sistema de sensores internos, incluidos acelerómetros y giroscopios de alta precisión, que trabajan en conjunto para registrar cada movimiento del usuario en tiempo real. El cambio de orientación en el espacio tridimensional, se captura de manera fluida, permitiendo a los usuarios experimentar la sensación de mirar alrededor con naturalidad en su entorno virtual. Además, la traslación, o el desplazamiento físico en el espacio, se mide con una precisión excepcional, brindando una experiencia de caminar o desplazarse de manera natural y coherente en el mundo virtual.

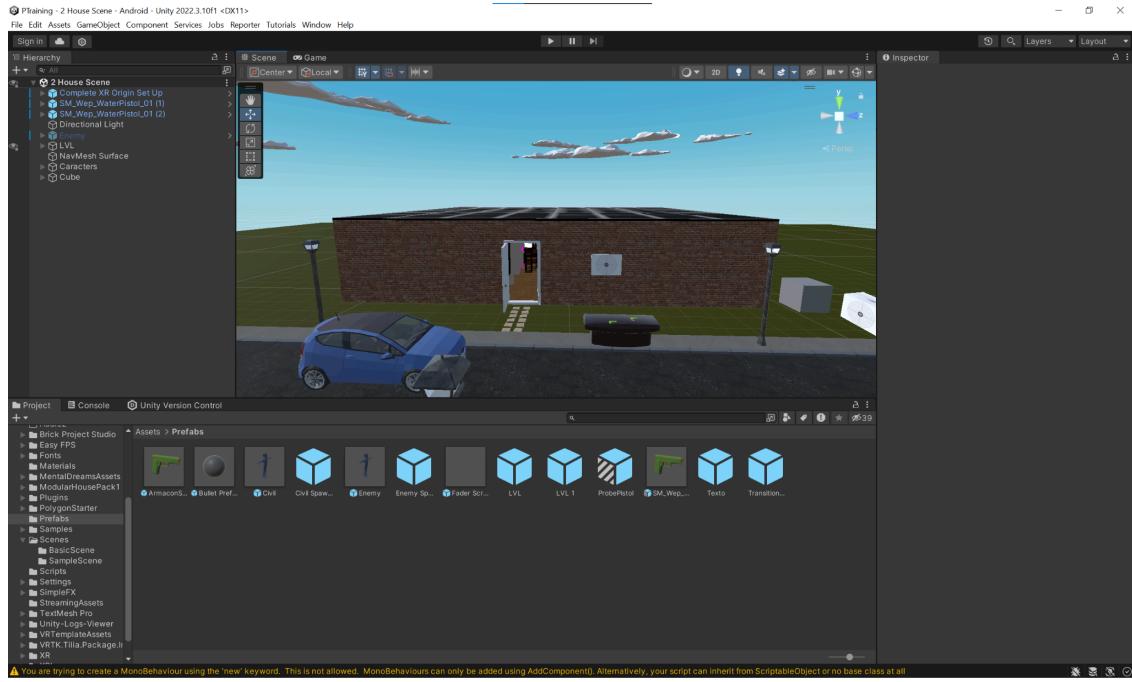
Todas estas características permiten desarrollar aplicaciones de realidad virtual con mucha mayor facilidad, ya que todos estos problemas iniciales están solucionados por el hardware.



Lentes de realidad virtual Meta Quest 2 y sus controladores Meta Touch

Unity

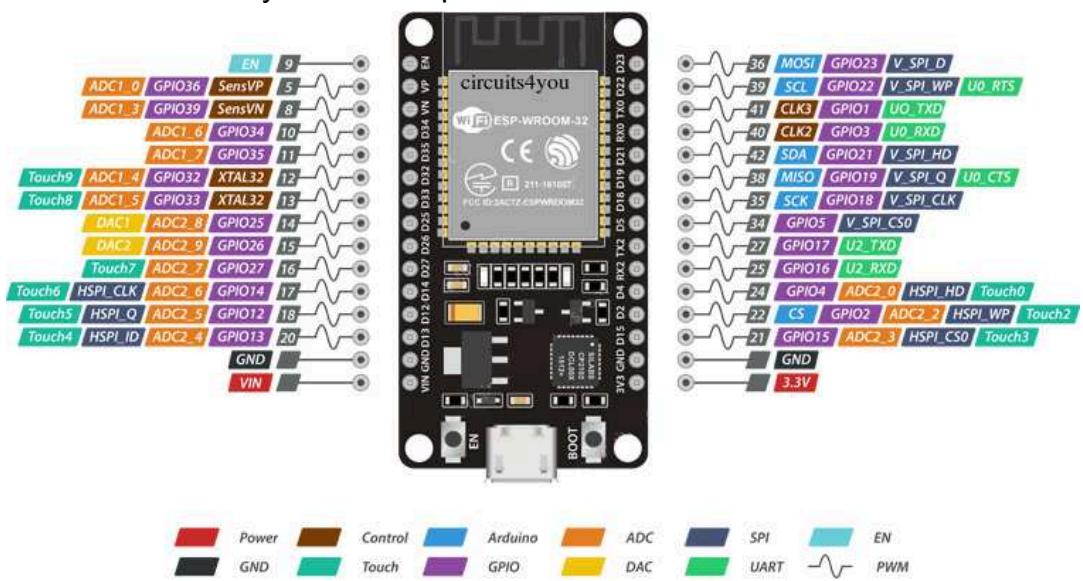
Unity se presenta como una herramienta integral y eficaz para el desarrollo de aplicaciones dirigidas a Meta Quest 2, basado en la plataforma Android. Unity simplifica el proceso de diseño y desarrollo al proporcionar un entorno intuitivo que permite a los creadores visualizar y perfeccionar sus creaciones en tiempo real. Además, la compatibilidad nativa de Unity con Android facilita la adaptación de aplicaciones para la plataforma Meta Quest 2, garantizando un rendimiento óptimo y una experiencia de usuario sin problemas. La extensa comunidad de desarrolladores respalda a Unity, ofreciendo un amplio conjunto de recursos y tutoriales específicos para VR, agilizando el proceso de aprendizaje y mejorando la eficiencia en el desarrollo.



Interfaz gráfica de programación Unity

ESP-32

La placa de desarrollo ESP32 posee una gran cantidad de funcionalidades a un precio muy bajo. Su capacidad para soportar conexiones Wi-Fi y Bluetooth simultáneas proporciona a los desarrolladores una amplia flexibilidad para la interconexión de dispositivos en entornos IoT. Además, la ESP32 incorpora características avanzadas, como Bluetooth de baja energía (BLE), lo que la convierte en una elección ideal para este caso, donde se necesita una comunicación ocasional entre lentes y chaleco hástico.



ESP32 Dev. Board / Pinout

Placa de desarrollo ESP32 - Pinout

Desarrollo

Experiencia virtual

Se desarrolló un único nivel de juego que corresponde a un allanamiento de domicilio con civiles y delincuentes. Mediante la interfaz visual de Unity se ubicaron los distintos recursos de la escena para lograr una experiencia parecida a una situación real. Los Assets utilizados se obtuvieron de la Unity Store, donde se encuentran varios Assets preparados todo en un mismo lugar para facilitar el desarrollo de distintos proyectos. También se utilizaron de referencia otros proyectos introductorios al desarrollo en VR.

Escenas

Menú Principal

Para configurar el menú, se utilizó un canvas con botones programados a través de un script de Unity. Este script únicamente detecta si se hace click sobre uno de los botones y activa las funciones correspondientes a cada uno. Lo mismo sucede para los menús de reinicio cuando el usuario termina el nivel (gane o pierda).

Se tiene también un tutorial en video y explicaciones sobre controles y puntuación.



Menú principal.

Objetivo del escenario y metodo de evaluacion:.



Siguiente...

Video Tutorial.

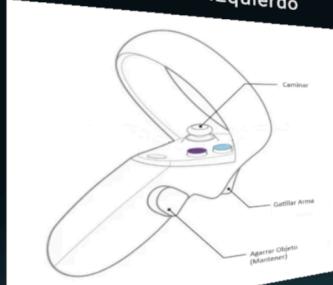
Evaluacion y restado de puntuajes:

Acción	Penalización
Abatir a un civil	-2 puntos por cada civil abatido
No neutralizar a un malviviente	-1 punto por cada malviviente
Usar un cartucho extra al necesario	-0.25 puntos por cada cartucho extra
Olvidar colocar el seguro del arma reglamentaria	-1 punto

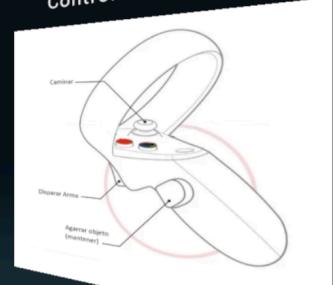
Siguiente...

Tabla de puntuaciones.

Controlador Izquierdo



Controlador Derecho



Inicio de Partida y Controles:
Gesto de recarga Gesto del seguro

Practicar

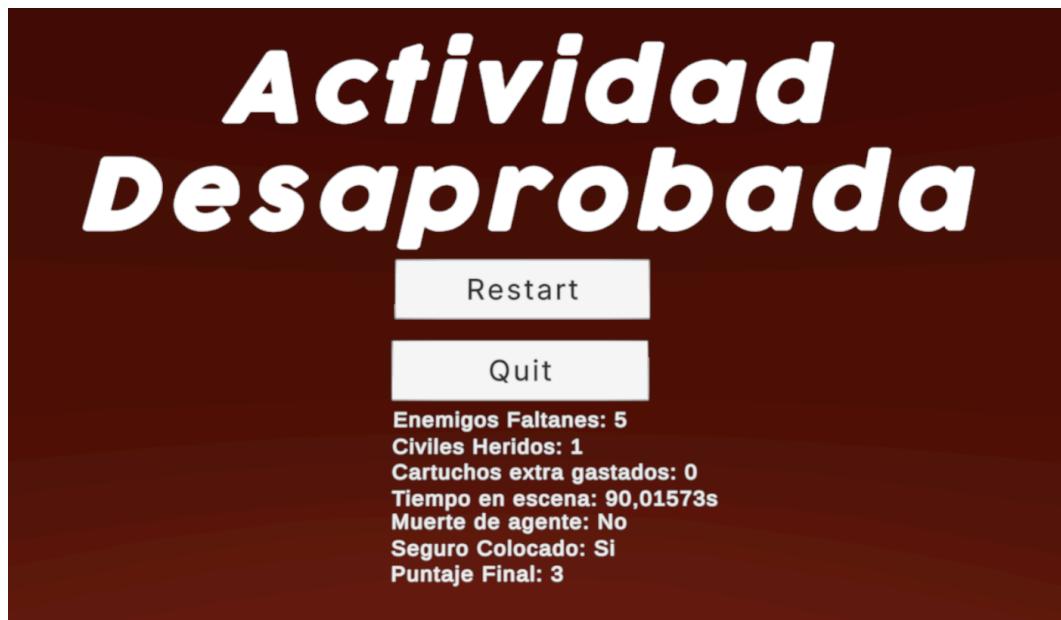
Evaluar

Controles y gestos básicos.

Menús Finales

Reportan si el nivel fue aprobado o desaprobado y aporta el detalle de las penalizaciones obtenidas por mal desempeño. Estas son:

- Civiles heridos
- Enemigos faltantes
- Cartuchos gastados
- Tiempo de finalización
- Muerte del usuario
- Colocación del seguro al finalizar actividad
- Puntaje obtenido



Menús secundarios (nivel aprobado o desaprobado)

Escenario de juego



Posición inicial.

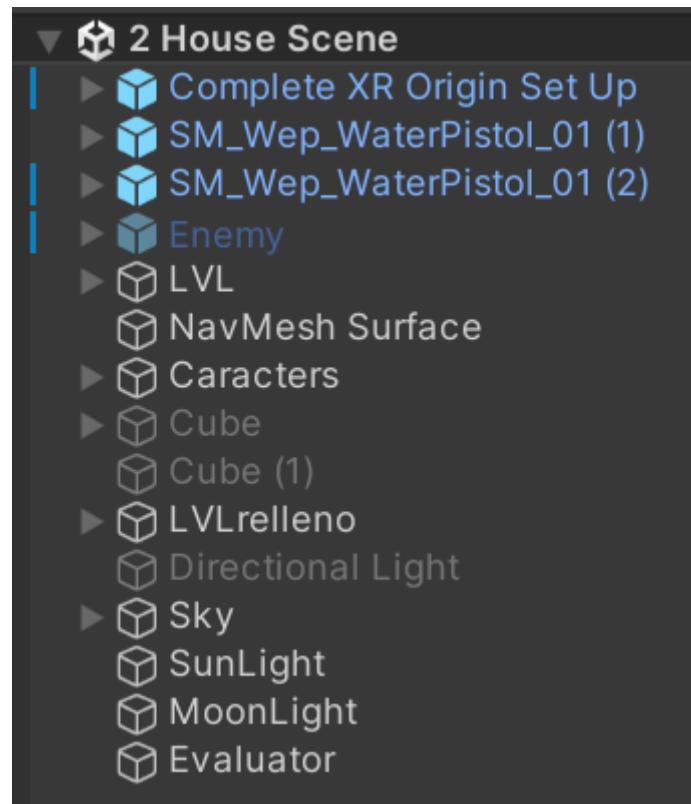


Sala principal del domicilio.



Otras habitaciones del domicilio.

Componentes



Vista de componentes de Unity del escenario domicilio.

En la figura se muestran los componentes utilizados para definir el escenario. Se puede observar:

- El controlador de realidad virtual, que hace de interfaz entre las acciones del usuario y sus efectos dentro de la experiencia virtual.
- Las pistolas que puede elegir el usuario.
- Assets del escenario (casa principal y relleno)
- Agentes que participan en la experiencia (civiles y delincuentes)
- Ambiente
- Evaluador, que realiza el seguimiento de los objetivos del escenario y otorga una puntuación acorde

Scripts

Pistolas

Cada pistola posee un script propio que determina las funciones para disparar, que se activan mediante el controlador de realidad virtual. Se implementaron funciones de recarga, habilitación de seguro del arma y disparo. Se especificó una cantidad de 12 municiones que corresponden al cargador reglamentario.

```
0 references
private void OnSelectAction()
{
    ToggleSafety();
}

0 references
private void OnActivateAction()
{
    Reload();
}
1 reference
protected override void Reload()
{
    base.Reload();

    currentBullets = maxBullets;
}

1 reference
protected override void ToggleSafety()
{
    base.ToggleSafety();

    isSafetyOn = !isSafetyOn;
}
```

Script de pistolas. Funciones de recarga y seguro.

IA

Se programó a los bots (delincuentes y civiles) con rutinas simples (fijas) de movimiento para esta primera versión. En el caso de los civiles, se colocaron *spawners* que los inicializan en determinadas posiciones según el siguiente script:

```

0 references
private void Start()
{
    timeSinceLastSpawn = spawnInterval;
}

0 references
private void Update()
{
    timeSinceLastSpawn += Time.deltaTime;
    if (timeSinceLastSpawn > spawnInterval)
    {
        timeSinceLastSpawn = 0f;
        if (spawnedCivil.Count < maxCivilNumber)
        {
            SpawnCivil();
        }
    }
}

1 reference
private void SpawnCivil()
{
    // Selecciona un prefab de civil aleatoriamente
    CivilAI civilPrefab = civilPrefabs[UnityEngine.Random.Range(0, civilPrefabs.Length)];
    CivilAI civil = Instantiate(civilPrefab, transform.position, transform.rotation);
    int spawnPointIndex = spawnedCivil.Count % spawnPoints.Length;
    civil.Init(player, spawnPoints[spawnPointIndex]);
    spawnedCivil.Add(civil);
}

```

Script de *spawners* civiles

La función `SpawnCivil()` inicializa al civil con una posición de cobertura que toma la IA.

La IA posee funciones para cubrirse, contabilizar disparos recibidos, etc.

En el caso de los delincuentes, la inicialización es similar, con *spawners* ubicados en distintas posiciones del mapa. La mayor diferencia radica en la complejidad de la IA:

```

0 references
private void Update()
{
    if (agent.isStopped == false && (transform.position - occupiedCoverSpot.position).sqrMagnitude <= 0.1f)
    {
        agent.isStopped = true;
        //detenemos el la reproducción de sonido por unica vez
        controlAudio.Stop();
        StartCoroutine(InitializeShootingCO());
    }
    if (isShooting)
    {
        RotateTowardsPlayer();
    }
}

2 references
private IEnumerator InitializeShootingCO()
{
    HideBehindCover();
    yield return new WaitForSeconds(UnityEngine.Random.Range(minTimeUnderCover, maxTimeUnderCover));
    StartShooting();
}

```

Script de IA delincuente. Rutina de disparo. El agente toma cobertura, espera una cantidad determinada de tiempo y empieza a disparar.

La función Shoot() de la IA controla el sistema de disparo de los delincuentes, y posee varios atributos configurables como la precisión de disparo, etc.

A futuro se debería implementar una IA más compleja que pueda tomar decisiones teniendo en cuenta factores del entorno, por ejemplo utilizando *behavior trees*. Esto mejoraría ampliamente la calidad del producto final, ya que proporcionaría una gran cantidad de escenarios de práctica adicionales.



Asset elegido para representar a los agentes enemigos

Evaluador

Se encarga de contabilizar la puntuación final del escenario. Incluye penalidades por diferentes acciones que se deberían evitar, como por ejemplo, exceso de disparos, dispararle a un civil, o por el uso del seguro del arma. Posee puntajes fijos de cada

una de estas acciones que se restan a una puntuación inicial. Cada objeto del escenario reporta estas acciones al evaluador, que determina la puntuación.

Efectos gráficos y de sonido

Para obtener un producto más inmersivo se agregaron diferentes efectos gráficos y de sonido. Para resaltar los efectos de disparar con una pistola se utilizaron efectos de sangre cuando las balas impactan en una persona y efectos de rotura en paredes u otros objetos. Además se agregaron efectos de sonido, como ruidos de disparo, pasos de cada agente, para que el usuario perciba lo que está sucediendo en la experiencia virtual de una forma más real.

Física

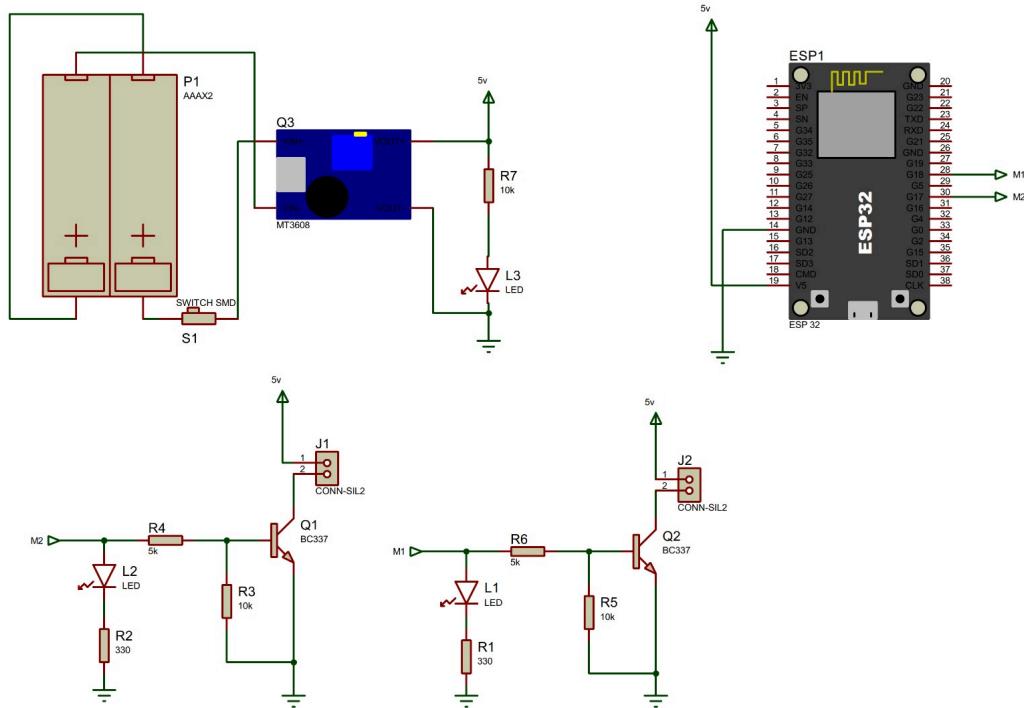
Los scripts PhysicsDamage y PhysicsProjectiles se encargan de la física de los proyectiles y determinan la colisión o no de estos con algún objetivo de la escena.

Chaleco haptico

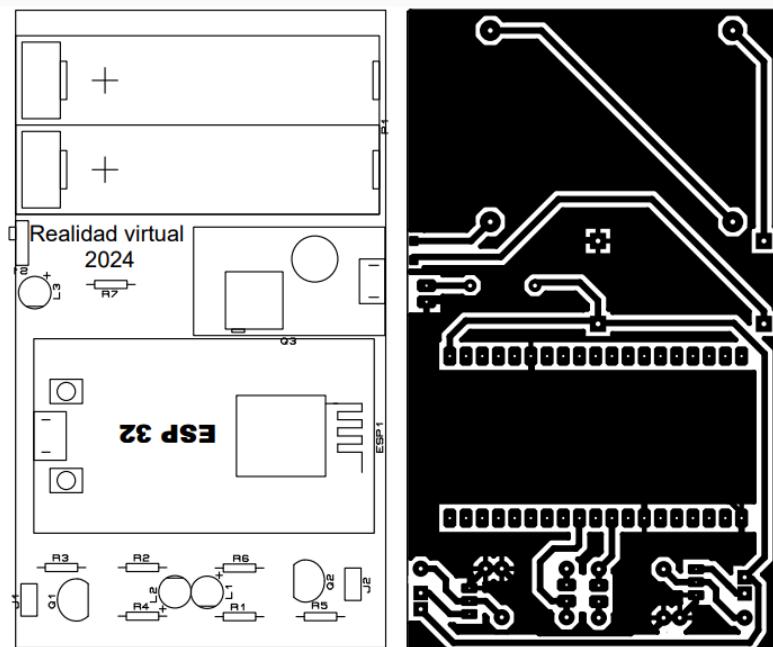
El dispositivo haptico diseñado y actualmente implementado se compone de dos vibradores ubicados en el pecho del usuario, con el propósito específico de simular la colisión de disparos de bala.

Este dispositivo refuerza la sensación de estar involucrado directamente en situaciones de entrenamiento realistas, lo que lleva a una experiencia más inmersiva.

El chaleco se comunica mediante Bluetooth, eliminando la necesidad de cables que podrían interferir con la libertad de movimiento. La alimentación del dispositivo se gestiona mediante baterías incorporadas, lo que no solo mejora la portabilidad del sistema, sino que también garantiza una operación sin interrupciones durante las sesiones de entrenamiento. Para esto se diseñó el siguiente esquema electrónico, a partir del cual se desarrolló una placa dedicada.



Esquema de conexiones y electrónica del dispositivo háptico



Placa dedicada

La implementación del dispositivo se basa en la utilización del microcontrolador ESP32, seleccionado por sus características de conectividad. Este microcontrolador no solo proporciona una plataforma eficiente para el manejo de datos y señales, sino que también permite una integración fluida con otros dispositivos y sistemas, potenciando así la funcionalidad general del dispositivo háptico.

En términos de potencia, el dispositivo cuenta con una etapa específica dedicada a alimentar los motores encargados de generar la vibración. Esta etapa de potencia garantiza un suministro estable y eficiente de energía.

Para la generación de vibración en el dispositivo, se ha incorporado un motor con una masa descentrada en el eje. En cuanto al diseño físico, el dispositivo está equipado con una carcasa diseñada para evitar obstrucciones con la ropa del usuario. Además, se ha incorporado un sujetador que asegura una posición estable y adecuada del dispositivo, manteniéndolo en su lugar durante las sesiones de entrenamiento.

Se implementó el siguiente código en el IDE de Arduino para la ESP32. En este se observa la inicialización de la comunicación Bluetooth, lectura de mensajes, y una máquina de estados para clasificar las acciones según los mensajes recibidos.

```
void setup() {
    pinMode(motor1, OUTPUT);
    pinMode(motor2, OUTPUT);

    SerialBT.begin("ESP32_BT");
}

void loop() {
    if (SerialBT.available()) {
        char command = SerialBT.read();
        switch (command) {
            case 'F':
                digitalWrite(motor1, HIGH);
                motor1StartTime = millis();
                break;
            case 'B':
                digitalWrite(motor2, HIGH);
                motor2StartTime = millis();
                break;
        }
    }
}
```

Código ESP32

Conexión bluetooth con el chaleco haptico (BlueUnity)

Para el funcionamiento del chaleco haptico es necesario en primer lugar lograr la comunicación bluetooth de los lentes de RV con la placa ESP32. Para esto, se utilizó la biblioteca de código abierto BlueUnity, que nos permite inicializar el componente de bluetooth con su configuración correspondiente. Además, la biblioteca posee funciones de comunicación que permiten enviar o recibir mensajes. En este proyecto únicamente se enviarán mensajes desde el Meta Quest 2 a la ESP32. Si bien esto no es lo óptimo para un desarrollo robusto, funciona para esta versión inicial. Más adelante se podrían implementar funciones para verificar que los

mensajes sean enviados y recibidos correctamente, haciendo un *handshake* con el receptor.

A continuación, se muestran extractos del código fuente que realiza la comunicación:

```
// creating an instance of the bluetooth class from the plugin |
0 references
public static void CreateBluetoothObject()
{
    if (Application.platform == RuntimePlatform.Android)
    {
        unityPlayer = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
        activity = unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
        context = activity.Call<AndroidJavaObject>("getApplicationContext");
        unity3dbluetoothplugin = new AndroidJavaClass("com.example.unity3dbluetoothplugin.BluetoothConnector");
        BluetoothConnector = unity3dbluetoothplugin.CallStatic<AndroidJavaObject>("getInstance");
    }
}
```

Función para crear el objeto bluetooth.

```
// starting bluetooth connection with device named "DeviceName"
// print the status on the screen using native android Toast
0 references
public static bool StartBluetoothConnection(string DeviceName)
{
    if (Application.platform == RuntimePlatform.Android)
    {
        try
        {
            string connectionStatus = BluetoothConnector.Call<string>("StartBluetoothConnection", DeviceName);
            Toast("Start connection status: " + connectionStatus);
            if (connectionStatus == "Connected")
                return true;
        }
        catch (Exception e)
        {
            Toast("Start connection error");
        }
    }

    return false;
}
```

Función para iniciar la comunicación.

```
// write data as a string to the bluetooth device
0 references
public static void WritetoBluetooth(string data)
{
    if (Application.platform == RuntimePlatform.Android)
    {
        try
        {
            BluetoothConnector.Call("WriteData", data);
        }
        catch (Exception e)
        {
            Toast("Write data error");
        }
    }
}
```

.Función para enviar un mensaje.

```
//read data from the bluetooth device
// if there is an error or there is no data coming, this method will return "" as an output
0 references
public static string ReadFromBluetooth()
{
    if (Application.platform == RuntimePlatform.Android)
    {
        try
        {
            return BluetoothConnector.Call<string>("ReadData");
        }
        catch (Exception e)
        {
            BluetoothConnector.Call("PrintOnScreen", context, "Read data error");
        }
    }
    return "";
}
```

Función para recibir un mensaje.

En nuestra implementación, inicializamos el objeto bluetooth en el componente que corresponde al usuario mediante el siguiente código:

```

0 references
private void Start()
{
    #if UNITY_2020_2_OR_NEWER
        #if UNITY_ANDROID
            if (!Permission.HasUserAuthorizedPermission(Permission.CoarseLocation)
            || !Permission.HasUserAuthorizedPermission(Permission.FineLocation)
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_SCAN")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_ADVERTISE")
            || !Permission.HasUserAuthorizedPermission("android.permission.BLUETOOTH_CONNECT"))
                Permission.RequestUserPermissions(new string[] {
                    Permission.CoarseLocation,
                    Permission.FineLocation,
                    "android.permission.BLUETOOTH_SCAN",
                    "android.permission.BLUETOOTH_ADVERTISE",
                    "android.permission.BLUETOOTH_CONNECT"
                });
        #endif
    #endif

    IsConnected = false;

    BluetoothService.CreateBluetoothObject();

    if (!IsConnected)
    {
        IsConnected = BluetoothService.StartBluetoothConnection(deviceName);
    }
    if (!IsConnected)
    {
        IsConnected = BluetoothService.StartBluetoothConnection("ESP32_BT");
    }
    BluetoothService.WritetoBluetooth(motor1);
    BluetoothService.WritetoBluetooth("F");
}

```

Inicialización de la comunicación Bluetooth en el componente Player.

En esta figura se observa que se solicitan permisos a Android para utilizar la comunicación Bluetooth, se crea el objeto y se inicia la comunicación.

Finalmente, según corresponda, se envía “1” o “2” para que vibre el motor 1 o 2 respectivamente:

```

//probabilidad 50% de activar 1 motor o el otro
if( UnityEngine.Random.Range(0, 100) < 50)
{
    BluetoothService.WritetoBluetooth(motor1);
    BluetoothService.WritetoBluetooth("F");
}
else
{
    BluetoothService.WritetoBluetooth(motor2);
    BluetoothService.WritetoBluetooth("B");
}

```

Mensajes enviados al dispositivo haptico cuando el usuario recibe un disparo.

Conclusión

En el desarrollo de este proyecto se pudo realizar un primer prototipo de una aplicación de realidad virtual inmersiva con dispositivos hápticos de realimentación al usuario. Se lograron distintas escenas de un nivel gráfico bajo, pero que permite realizar demostraciones del futuro potencial de la aplicación. También se desarrolló un dispositivo háptico que realimenta señales clave para lograr una mejor inmersión del usuario en la experiencia virtual. Se integraron distintos sistemas típicos utilizados en este tipo de aplicaciones, tanto en hardware (Meta Quest 2, microcontrolador y placa dedicada para el dispositivo háptico) como en software (programación en entorno gráfico Unity, comunicación Bluetooth).

Si bien este desarrollo es utilizable para entrenamiento policial, cabe notar que es un prototipo inicial, y que se debe seguir desarrollando sobre el mismo para lograr un nivel profesional y comercial. Como mejoras a futuro, se encuentran:

- Mejora del nivel gráfico de la aplicación, personajes, escenas, assets, armas y efectos.
- Mayor cantidad de escenarios de entrenamiento.
- Mejora de la IA para lograr experiencias diferentes en los mismos escenarios.
- Mejora del dispositivo háptico para lograr una mayor inmersión.

Referencias

BlueUnity - <https://github.com/bentalebahmed/BlueUnity>

Assets - <https://assetstore.unity.com/>

Assets utilizados:

- Brick Project Studio - <https://assetstore.unity.com/publishers/32000>
- Easy FPS -
<https://assetstore.unity.com/packages/3d/characters/humanoids/sci-fi/easy-fps-73776>
- Modular House Pack -
<https://assetstore.unity.com/packages/3d/environments/urban/modular-house-pack-1-236466>
- Polygon Starter -
<https://assetstore.unity.com/packages/essentials/tutorial-projects/starter-pack-synty-polygon-stylized-low-poly-3d-art-156819>
- Simple FX -
<https://assetstore.unity.com/packages/vfx/particles/simple-fx-cartoon-particles-67834#content>
- Stylized Male Character -
<https://assetstore.unity.com/packages/3d/characters/humanoids/stylized-modular-character-male-191985>
- Stylized Female Character -
<https://assetstore.unity.com/packages/3d/characters/humanoids/stylized-modular-character-female-204813>
- XR Interaction Toolkit -
<https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html>
- Polaris -
<https://assetstore.unity.com/packages/tools/terrain/low-poly-terrain-polaris-2021-196648>

UnityDocs - <https://docs.unity.com/>