R3-A4. Producción de un itinerario de formación adaptativo e inmersivo en realidad virtual (RV).

Erasmus+

INTRODUCCIÓN

El avance hacia una formación más inclusiva y adaptativa en el sector de la piedra natural toma forma con la creación de una ruta de aprendizaje en realidad virtual (RV) sumamente inmersiva. Este proyecto es un componente vital para integrar tecnologías avanzadas en la capacitación, facilitando así un entorno de aprendizaje personalizado y efectivo.

Diseñar este camino educativo requiere de un detallado proceso de desarrollo que abarca desde la identificación de escenarios clave hasta su implementación en un entorno de RV. Se parte de una fase de investigación y análisis, donde se seleccionan cuidadosamente las situaciones y tareas más representativas del sector. Luego, se procede a un diseño instruccional que toma en cuenta tanto la funcionalidad como la accesibilidad, asegurando que cada etapa de la formación sea relevante y alcanzable para todos los usuarios.

A través de este enfoque, se busca que los escenarios de RV no solo simulen con precisión las condiciones laborales, sino que también estén ajustados para satisfacer las necesidades educativas específicas de los usuarios. La experiencia inmersiva resultante tiene como meta no solo instruir, sino también empoderar a los usuarios, permitiéndoles desarrollar habilidades prácticas en un entorno seguro y controlado, listos para su aplicación en el mundo real.

Al final, esta herramienta educativa no solo se convierte en un puente hacia la competencia laboral, sino también en un medio para fomentar una mayor comprensión y aceptación de la diversidad en el lugar de trabajo.

Este informe y toda la información sobre el proyecto están disponibles en la web del InclusiveStone: https://inclusivestone.eu/

Contenido

1. DESARROLLO DE LA HERRAMIENTA

El núcleo de este proyecto se fundamenta en la capacidad de inmersión que ofrece la Realidad Virtual. En la actualidad, el estándar para la creación de aplicaciones en esta esfera se asienta en el uso de motores gráficos originarios del mundo del videojuego. Por ello, el enfoque adoptado para el desarrollo de nuestra herramienta educativa sigue una metodología similar a la creación de un videojuego.

Esta metodología se estructura en torno a tres pilares fundamentales que se desglosarán en detalle en las secciones subsiguientes:

- **Diseño de las escenas**
- **Desarrollo de los entornos 3D.**
- **Desarrollo de la funcionalidad y experiencia inmersiva.**

Para llevar a cabo el desarrollo de forma eficiente y eficaz, cabe destacar que se han utilizado diferentes tecnologías:

- **Blender:** Una herramienta de software libre y multiplataforma, utilizada para una amplia gama de funciones de gráficos 3D como modelado, animación y renderizado. Su naturaleza de código abierto y la existencia de una comunidad extensa facilitan el aprendizaje mediante tutoriales y documentación extensiva.
- **Unity:** Este motor gráfico, también de código abierto, es ampliamente utilizado en el desarrollo de videojuegos. Destaca por su escalabilidad y la posibilidad de personalización a través de scripts, apoyado por una comunidad robusta y una documentación completa.
- **Oculus Quest:** Un dispositivo de Realidad Virtual de fácil acceso y manejo, que se ha convertido en uno de los soportes esenciales de la visión de futuro de Meta. Ofrece la ventaja de ser inalámbrico y posee una integración fluida con Unity, potenciando así la accesibilidad y la versatilidad en el desarrollo de aplicaciones de RV.

2. DISEÑO DE ESCENAS

Este procedimiento implica la planificación detallada de la secuencia de pasos que el usuario realizará desde el inicio de su interacción con la herramienta hasta el momento en que alcanza una experiencia inmersiva completa. En resumen, se trata de la creación del guion que guiará el desarrollo integral de la aplicación.

Inicialmente, se seleccionaron 6 escenarios laborales que representan adecuadamente los diversos roles identificados en el paquete de trabajo R1, debido a su capacidad de adaptación y al hecho de que no representan un peligro para personas con distintas discapacidades. Las situaciones escogidas son las siguientes:

- **Operario de carretilla elevadora - Transporte de mercancías:** En el contexto de esta tarea, el objetivo consistirá en efectuar la carga y descarga de tres palés de losas de mármol, trasladándolos desde la planta de procesamiento hasta el área de almacenamiento designada. Esta actividad implicará la realización de maniobras en zonas con espacio limitado y el apilamiento de materiales de significativo peso.
- **Operario de carretilla elevadora - Carga de camiones:** En este escenario específico, se establece como meta la carga de un vehículo de transporte con tres palés de losas de mármol, partiendo desde la zona destinada para el almacenaje. La operación demandará la ejecución de maniobras en áreas de espacio confinado y la estiba de cargamentos de elevado peso.
- **Operario de puente grúa - Manipulación de tablas:** En esta tarea, el usuario tendrá como objetivo trasladar 2 tablas de mármol hasta la zona iluminada de almacenamiento, realizando las maniobras necesarias y cumpliendo con las normas de seguridad.
- **Operario de puente grúa - Manipulación de bloques:** En el desarrollo de esta actividad, la meta del usuario será transportar un bloque de mármol desde el camión hasta la zona de carga del telar, ejecutando las maniobras que sean requeridas y observando rigurosamente las normas de seguridad vigentes.
- **Operario de limpieza:** El propósito de esta actividad consiste en identificar y seleccionar los Equipos de Protección Individual (EPIs) adecuados para la limpieza y proceder a la higienización de la planta de trabajo, eliminando escombros y desechos. Dicha tarea deberá llevarse a cabo de manera responsable y con el máximo respeto por el medio ambiente.
- **Gestión de residuos:** En el desarrollo de esta tarea, recae sobre el usuario la responsabilidad de administrar los productos químicos y residuos. Esto comprende la selección adecuada de técnicas y herramientas para el reciclaje y la disposición correcta de los desechos, así como el seguimiento de directrices destinadas a garantizar una gestión de residuos responsable y sostenible con el medio ambiente.

En este contexto, se optó por diseñar un modelo de escenario compuesto por etapas secuenciales o misiones, al estilo de los videojuegos; o sea, solo se puede avanzar al siguiente paso una vez completado el anterior, y de esta manera progresivamente. Así, se impone al usuario la necesidad de finalizar el proceso completo de manera segura desde el inicio hasta el final, y cualquier incapacidad de hacerlo se califica como un fracaso en el escenario.

Dentro de este conjunto de etapas, la inicial consistirá en un tutorial detallado sobre la máquina o la tarea específica. De esta forma, se facilitará al usuario el aprendizaje práctico de los controles correspondientes a cada situación, permitiendo una mayor fluidez en la operación sin la necesidad de verificarlos frecuentemente. Para continuar con las tareas específicas, que dependerán de cada situación.

2.1. Tutorial

En el transcurso de la fase de diseño de la app VR, hemos implementado un elemento fundamental para la orientación y preparación del jugador: el tutorial. Este componente es esencial, ya que proporciona una introducción estructurada y metódica al entorno de juego, las herramientas disponibles y las mecánicas de la misión.

El panel tutorial fue meticulosamente desarrollado para asegurar una experiencia de aprendizaje integral. Se inició con la creación de un guion detallado que descompone cada paso crítico que el jugador debe comprender antes de embarcarse en la misión.

La primera sección del tutorial abarca el manejo y las funcionalidades de las máquinas que los jugadores operarán durante el juego. Posteriormente, el tutorial introduce la herramienta de teletransporte, una mecánica crucial que amplía las posibilidades estratégicas y de movilidad dentro del juego. Se han elaborado ejercicios específicos que permiten a los jugadores experimentar con esta herramienta en un entorno controlado, asegurando su dominio antes de enfrentarse a situaciones en tiempo real.

Además, el panel incluye módulos informativos sobre el uso de cualquier funcionalidad adicional que sea relevante para la misión. Cada elemento ha sido cuidadosamente explicado y acompañado de ejemplos prácticos para garantizar una comprensión holística de su uso y aplicación.

Figura 1: Panel tutorial

La realización de este panel tutorial ha sido una de las piedras angulares del desarrollo de nuestro videojuego, garantizando que cada usuario esté adecuadamente equipado con el conocimiento y la experiencia necesarios para disfrutar plenamente del juego y tener éxito en sus misiones, y, por ende, en el trabajo real.

2.2. Misiones

En el desarrollo de nuestro proyecto, hemos implementado un sistema de misiones que es f undamental para la progresión del usuario en el aprendizaje del puesto de trabajo. Este sist ema está meticulosamente diseñado para adecuarse a los distintos roles y trabajos.

Las misiones se clasifican en dos categorías principales que contribuyen a la diversidad y riq ueza de la herramienta:

- **Misiones de Acción**: Estas misiones son el núcleo de la interactividad dentro del ju ego. Requieren que el jugador se involucre de manera activa, completando tareas q ue pueden incluir desde el manejo de maquinaria hasta el uso de las manos para re cogida y control de limpieza. Estas misiones están diseñadas para poner a prueba la s habilidades motoras y estratégicas del usuario.
- **Misiones de Pregunta y Respuesta**: En estas misiones, el jugador se enfrenta a situ aciones que requieren reflexión y toma de decisiones basadas en la información reci bida en los cursos que el usuario ha de tener para que esta app sea realmente útil.

A través de paneles de pregunta y respuesta, estas misiones evalúan la comprensió n y el conocimiento adquirido por el usuario.

Figura 2:Misión de transporte de bloques

Cada misión, independientemente de su categoría, está cuidadosamente entrelazada con la siguiente, asegurando que la experiencia de cada jugador sea coherente y profundamente i nmersiva. Además, este diseño de misiones promueve una curva de aprendizaje natural, do nde los jugadores pueden mejorar sus habilidades a medida que avanzan en el juego.

En resumen, nuestro sistema de misiones no solo busca entretener, sino también involucrar al jugador en un proceso de aprendizaje continuo y descubrimiento, lo cual es esencial para la experiencia integral que deseamos ofrecer.

3. DESARROLLO DEL ENTORNO 3D

La creación de herramientas dentro de los motores gráficos de videojuegos implica frecuent emente el manejo de objetos. Por consiguiente, la segunda tarea principal es la producción de los distintos objetos o assets, los cuales se especifican tanto de manera directa como ind irecta en los guiones técnicos elaborados previamente. Este procedimiento se segmenta en tres fases esenciales:

- **Modelado**: Se construye una representación matemática de un objeto en tres dime nsiones a través de un software especializado. Para este proyecto, se ha optado por el uso de Blender.
- **Rigging**: En esta etapa, se descomponen las partes constituyentes de un objeto par a formar un esqueleto digital controlable, lo que facilita la realización de animacion es fluidas y precisas. Este paso es crucial para los objetos que requerirán animación y también se ha llevado a cabo utilizando Blender.
- **Texturizado**: Durante el texturizado, se aplican colores y detalles a los modelos 3D para otorgarles un aspecto final más realista y detallado, empleando una vez más el software Blender.

La gama de objetos elaborados para este proyecto abarca un espectro amplio y variado, qu e va desde elementos de simpleza extrema hasta maquinarias complejas y detalladas. Los o bjetos producidos se han organizado en distintas categorías, de acuerdo con su complejida d y función dentro del juego. Podemos encontrar:

- Contenedores, edificios, estanterías y elementos de decoración.

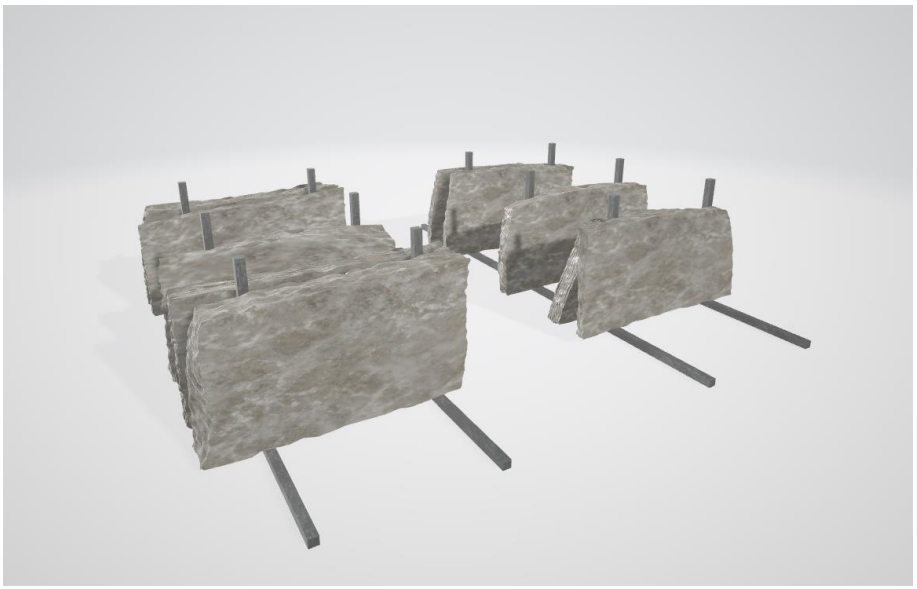


Figura 3: Láminas de piedra.

- Equipo personal y objetos útiles, con cierta funcionalidad.

Figura 4: Otros elementos.

- Máquinas específicas para las distintas situaciones.

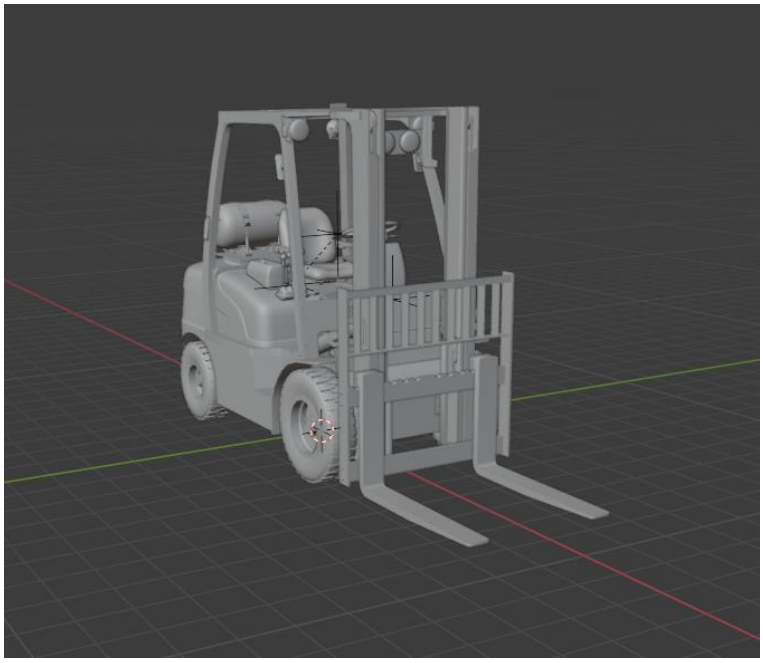


Figura 5: Modelo de carretilla elevadora

Una vez que se han diseñado y creado los objetos que serán empleados en el juego, el siguiente paso es la preparación de los escenarios. Este proceso es crucial para que el desarrollo de la funcionalidad del juego refleje con precisión el entorno en el que se desarrollará la acción final. Es importante destacar que este proceso es iterativo y sujeto a ajustes, ya que las pruebas subsiguientes pueden revelar la necesidad de realizar modificaciones. Para la creación del

escenario, se ha utilizado el motor gráfico Unity, que será el mismo que se empleará para la producción de la aplicación final.

Ya que los puestos laborales que se escogieron en tareas anteriores son puestos que se realizan en fábrica y no en cantera, solo se ha desarrollado un escenario. El escenario diseñado para este proyecto es el siguiente:

- **Fábrica de Piedra Natural**: El escenario es una representación virtual de una fábric a real, con áreas claramente diferenciadas que incluyen la sala de máquinas, el alma cén de productos químicos y la sección de extracción de residuos. Este diseño detall ado ayuda a los usuarios a familiarizarse con un entorno laboral específico y las disti ntas tareas que se llevan a cabo en la industria de la piedra natural.

Figura 6: Escenario fábrica

4. DESARROLLO DE LA FUNCIONALIDAD Y EXPERIENCIA INMERSIVA

El desarrollo la funcionalidad es el paso en el que los conceptos y la información recolectada previamente cobran vida. El propósito de esta fase es convertir la herramienta teórica en una aplicación práctica y operativa. Para llevar a cabo esta tarea, hemos seleccionado el motor gráfico Unity debido a su facilidad para incorporar elementos de Realidad Virtual, así como por el extenso soporte que ofrece su comunidad de usuarios.

En Unity, los proyectos se estructuran en Escenas, las cuales pueden estar conectadas a través de una serie de eventos. Cada Escena se compone de varios objetos, y a estos se les asignan componentes específicos que definen sus atributos y comportamientos dentro del juego. El proceso de desarrollo de funcionalidades es extenso y se puede desglosar en varias etapas clave, que incluyen el **Diseño y Arquitectura de la herramienta**, el **Modelado de Datos** para estructurar la información, la **Integración de SDK** para expandir las capacidades del motor y la **Gestión de Eventos** para un flujo de juego interactivo y coherente.

4.1. Diseño y arquitectura

El Diseño y Arquitectura implican la elaboración detallada del guion previamente desarrollado, con el objetivo de definir meticulosamente el trayecto que el usuario deberá seguir y las decisiones que tendrá que tomar en cada escena o pantalla del juego. Este proceso es fundamental, ya que establece la estructura base sobre la que se construirá la experiencia del usuario. Durante esta etapa, se toman en cuenta la lógica de navegación, la disposición de los elementos interactivos y la secuencia de eventos que guiarán al usuario a lo largo del juego. Se planifica minuciosamente la interacción con cada objeto y personaje dentro del entorno virtual, así como las transiciones entre escenas, para asegurar una experiencia intuitiva y envolvente que mantenga al usuario comprometido con la narrativa y los objetivos del juego.

Dentro del proyecto, los usuarios se encontrarán con dos tipos de Escenas principales diseñadas en Unity: la Escena de Menú y la Escena específica para cada puesto de trabajo.

4.1.1. Menú principal

La pantalla de Menú se presenta como una introducción intuitiva y accesible al entorno virtual, donde los usuarios dan sus primeros pasos en la herramienta. Al iniciar, se enfrentan a una elección fundamental: la configuración del idioma, asegurando que la experiencia esté personalizada y sea comprensible para ellos. Una vez establecida esta preferencia básica, la pantalla guía suavemente al usuario hacia la comprensión del propósito principal de la escena, que es la elección de una situación laboral específica para explorar. Aunque la interactividad en esta etapa se mantiene deliberadamente restringida para mantener el enfoque en el objetivo de aprendizaje, la experiencia está diseñada para ser acogedora y orientadora, preparando al usuario para la transición a la siguiente etapa, donde la selección realizada en el menú determinará la Escena específica de trabajo a la que se accederá a continuación.

4.1.2. *Escena situación laboral*

Esta fase representa el núcleo de la herramienta, donde cada situación laboral propuesta cobra vida en su propia Escena individual. Siguiendo el esquema de misiones previamente establecido, los usuarios se sumergen en una serie de tareas diseñadas para emular las responsabilidades y desafíos de un puesto de trabajo específico. La aventura comienza con un tutorial detallado, ajustado meticulosamente a la tarea en cuestión, proporcionando una base sólida para el aprendizaje y la ejecución de las misiones subsiguientes.

A medida que el usuario avanza y supera cada misión, se desbloquea la siguiente, permitiendo un progreso lineal y una sensación de logro constante. Al enfrentarse y completar con éxito la última misión, se le presenta la opción de reiniciar esa Escena y perfeccionar sus habilidades, o bien, regresar al menú principal para explorar y aprender sobre otro puesto de trabajo, facilitando así un proceso de aprendizaje continuo y una exploración más amplia de las diversas facetas de la industria de la piedra natural.

4.2. Modelado de datos

La meta de esta fase es crear un marco general para el almacenamiento y recuperación de datos. Se ha optado por emplear PlayerPrefs, un módulo de Unity que facilita la conservación de la información mediante un sistema de claves asociadas al nombre del proyecto. En aplicaciones para Oculus, estas variables se guardan en forma de un archivo XML, ubicado en el directorio /data/data/pkg-name/shared_prefs/pkg-name.v2.playerprefs.xml, donde "pkg-name" corresponde al nombre asignado a la aplicación. Unity hace que el acceso a estos datos sea extremadamente fácil a través de las funciones de la clase PlayerPrefs.

Debido a cómo está estructurada esta aplicación, habrá una variable común que necesitaremos almacenar: la que determina el idioma. Usaremos las siguientes funciones de PlayerPrefs para gestionarla:

- **SetString (*nombre, valor*):** Con esta función asignaremos un valor de texto, que será el código ISO del idioma elegido, a la variable identificada por "nombre".
- **GetString (*nombre*):** Esta función nos permite recuperar el valor asignado a la variable "nombre". La utilizaremos en cada escena para identificar el idioma seleccionado.

4.3. Integración SDK

La creación de aplicaciones de Realidad Virtual se basa en tres pilares fundamentales: un dispositivo compatible que pueda conectarse a un ordenador, un motor gráfico de videojuegos y un Kit de Desarrollo de Software (SDK). Al comienzo de este capítulo se detallaron los dos primeros elementos, siendo Oculus el dispositivo y Unity el motor gráfico escogidos. En cuanto al SDK, que es el conjunto de herramientas que asiste a los desarrolladores de software en el proceso de creación, hay varias opciones disponibles para adaptarse a Unity. A pesar de que existe uno específico para Oculus, se ha seleccionado el XR Interaction Toolkit, un paquete desarrollado por Unity que facilita la adaptación del proyecto a distintos tipos de dispositivos. Esto permite que, aunque la herramienta se desarrolle inicialmente para Oculus Quest, pueda ser instalada en cualquier dispositivo que sea compatible con Unity, ampliando así su accesibilidad y alcance.

Para integrar el SDK y habilitar la interacción en Unity, se deben seguir los pasos a continuación:

1. Al iniciar Unity, se crea un nuevo proyecto utilizando la plantilla Universal Render Pipeline, la cual es clave para obtener gráficos optimizados, un aspecto crucial para una experiencia de Realidad Virtual fluida y satisfactoria.
2. Se instala el SDK, en este caso, el paquete XR Interaction Toolkit, a través de la ventana Package Manager que se encuentra en la opción de menú Window.
3. Es necesario especificar el tipo de dispositivo en el que se desarrollará el proyecto. Para ello, se activa la casilla VR supported dentro de Project Settings/Player.

Con estos pasos, el proyecto de Unity queda preparado para el desarrollo en Realidad Virtual. Sin embargo, para comenzar la interacción del dispositivo con el entorno gráfico, es esencial seguir las instrucciones del apartado siguiente.

4.4. Gestión de eventos

La administración de eventos es posiblemente la parte más amplia en la creación de la funcionalidad de nuestra aplicación. Se basa en la utilización de los componentes que Unity ofrece, junto con la creación de scripts concisos que posibilitan la interacción deseada entre el

usuario y el sistema. Dado que cada escena está repleta de detalles únicos que afectan la manera en que se interactúa con ella, esta sección podría extenderse considerablemente. No obstante, en este documento nos centraremos únicamente en destacar aquellos aspectos comunes y cruciales que son esenciales para alcanzar nuestros objetivos de interactividad.

4.4.1. Preparar la escena para la interacción VR

Tras tener listo el proyecto de Unity y los escenarios diseñados gráficamente, el paso subsiguiente consiste en preparar la escena para que el dispositivo de interacción se integre con una cámara, facilitando la realización de pruebas durante el desarrollo. Este procedimiento, descrito en las etapas siguientes, se deberá ejecutar para cada una de las escenas del proyecto:

1. Elimina la cámara predeterminada que aparece por defecto en la escena.
2. Crea un objeto vacío y añádele un componente XR Rig, que funcionará como el núcleo de la interacción entre el usuario y la máquina. Nombraremos a este objeto como VR-Rig.
3. Dentro del VR-Rig, crea un objeto hijo vacío que servirá como punto de referencia inicial para la interacción del usuario. Lo llamaremos Camera Offset.
4. Añade una cámara como hija del objeto Camera Offset e incorpora un componente Tracked Pose Driver a esta cámara. Este elemento actuará como nuestros ojos virtuales y el componente permitirá la rotación de la vista.
5. Configura las variables del componente XR Rig (padre, VR-Rig) con la posición de referencia y la cámara que acabas de crear. Establece el valor de 'floor' en el atributo Tracking Origin Mode para que la cámara se ajuste automáticamente a la altura del usuario.
6. Para habilitar la movilidad y la visibilidad de los controladores, crea dos objetos adicionales como hijos del Camera Offset y dótalos con el atributo XR Controller.
7. Asigna el modelo que deseas utilizar como controladores en el atributo Model Prefab de cada XR Controller.
8. Finalmente, incorpora el componente XR Direct Interactor a estos dos objetos, lo cual proporcionará la capacidad de interactuar con otros objetos en la escena.

4.4.2. Entrada del controlador

Este proceso se refiere a la captura de los valores emitidos al interactuar con los botones de los controles. Estos valores son cruciales para realizar una variedad de acciones dentro de la aplicación, como agarrar objetos o seleccionar diferentes opciones. Antes de proceder a cualquier operación con estos datos, es esencial comprender el tipo de información que proporciona cada entrada de los mandos. Esta información se puede consultar en la ventana del Depurador de Interacción XR, accesible a través de Ventana/Análisis.

Una vez que se tiene esta comprensión, el paso siguiente es elaborar un script que permita recoger estos valores. La clase creada para este fin se denomina `HandController` y se adjunta

como componente a cada uno de los modelos 3D especificados en el atributo Model Prefab de los controles. Los aspectos más sobresalientes del código incluyen:

```
public class HandController : MonoBehaviour
{
    //Input
    private InputDevice targetDevice;

    //Controller model options
    public bool showController = false;
    public List<GameObject> controllers;
    private GameObject spawnedController;

    //Device characteristics
    public InputDeviceCharacteristics controllerChar;

    //Hand model
    public GameObject handModel;
    private GameObject spawnedHandModel;

    private Animator handAnimator;

    //Get if is Right or left
    public string isRight;

    //Input variables
    public float trigVal;
    public bool primaryButton;
    public float gripValue;
    public Vector2 axisVal;
```

Figura 7: Clase HandController

1. Al comienzo del script, se observa la declaración de las variables. Aquellas que son públicas se pueden modificar y actuarán como atributos del componente.
2. Al crear un nuevo script, Unity genera automáticamente dos funciones: `Start`, que se ejecuta una sola vez al comienzo cuando se instancia el componente, y `Update`, que se lleva a cabo repetidamente durante el ciclo de vida del componente.
3. La primera función que se activa dentro de este componente es `TryInitialize`, que identifica el dispositivo VR por sus características específicas, como si el controlador es derecho o izquierdo, mediante la función `GetDevicesWithCharacteristics` del módulo `InputDevices`.

```
//Try get input device and attach the selected model to it
void TryInitialize()
{
    //Get VR devices by characteristics
    List<InputDevice> devices = new List<InputDevice>();
    InputDevices.GetDevicesWithCharacteristics(controllerChar, devices);


    if (devices.Count > 0)
    {
        //Get target device and get a controller model
        targetDevice = devices[0];
        GameObject prefab = controllers.Find(con => con.name == targetDevice.name);

        if (prefab)
        {
            spawnedController = Instantiate(prefab, transform);
        }
        else
        {
            Debug.LogError("Did not find corresponding controller model");
            spawnedController = Instantiate(controllers[0], transform);
        }

        //Instantiate Hand Model and get Animator component
        spawnedHandModel = Instantiate(handModel, transform);
        handAnimator = spawnedHandModel.GetComponent<Animator>();
    }
}
```

Figura 8: Función TryInitialize

4. La función `Update` se ejecuta a continuación, invocando a su vez las funciones `UpdateEvents` y `UpdateAnimations`.
5. `UpdateEvents` se encarga de recopilar los valores emitidos por los controles utilizando la función `TryGetFeatureValue` y los almacena en las variables públicas declaradas previamente.

6. `UpdateAnimations` es una función diseñada para activar animaciones correspondient es a cada botón, en el caso de disponer de un modelo de mano con rigging. Aunqu e esta función no es crucial en la tarea de adquisición de valores, contribuye a la int eractividad y realismo de las acciones.

```
//Update hand events
void  UpdateEvents()
{
    //Trigger event
    if(targetDevice.TryGetFeatureValue(CommonUsages.trigger, out float triggerVal))
    {
        trigVal = triggerVal;
    }

    //PrimaryButton event
    if (targetDevice.TryGetFeatureValue(CommonUsages.primaryButton, out bool primary))
    {
        primaryButton = primary;
    }

    //Grip event
    if (targetDevice.TryGetFeatureValue(CommonUsages.grip, out float gripVal))
    {
        gripValue = gripVal;
    }

    //Joystick event
    if(targetDevice.TryGetFeatureValue(CommonUsages.primary2DAxis, out Vector2 axis))
    {
        axisVal = axis;
    }
}
```

Figura 9: Función UpdateEvents

4.4.3. Control de la carretilla elevadora

Para lograr una experiencia de realidad virtual convincente y funcional en el manejo de una carretilla elevadora, es esencial contar con una serie de scripts detallados y especializados. Estos scripts son los encargados de proporcionar la interactividad y el control precisos que se requieren para simular de manera efectiva la operación de este tipo de maquinaria. Dentro de la herramienta de VR, se necesitará un script primordial dedicado a la conducción, el cual permitirá al usuario maniobrar la carretilla elevadora dentro del entorno virtual. Además, se requerirán varios scripts adicionales enfocados en el control de las palancas y otros mecanismos de la carretilla, asegurando que todas las acciones posibles en la operación real puedan ser replicadas virtualmente.

A continuación, se procederá a explicar detalladamente cada uno de estos scripts, explorando su funcionamiento, la interacción con los elementos de la carretilla elevadora y la forma en que contribuyen a una experiencia de usuario inmersiva y realista en el manejo de esta maquinaria en el contexto de la realidad virtual.

- El script `CarController` proporciona una experiencia de conducción interactiva en realidad virtual, donde el usuario puede influir directamente en la aceleración del

vehículo a través del gatillo de los controles, lo que se refleja en la variable `accelVal`. Asimismo, el script ajusta la dirección del vehículo modificando el ángulo de las ruedas en función de la entrada del usuario, lo cual se maneja con la variable `currentSteerAngle`. Los frenos se activan mediante un botón, lo que impacta en la variable `isBreaking`, y el script gestiona las físicas de colisión para simular un rebote cuando es necesario. En resumen, este código traduce las interacciones del usuario en comportamientos mecánicos del vehículo dentro del entorno virtual.

```
// Update is called once per frame
void FixedUpdate()
{
    GetValues();

    if (!isBouncing)
    {
        foreach (WheelCollider wheel in wheels)
        {
            wheel.motorTorque = strength * Time.deltaTime * accelVal;

            wheel.wheelDampingRate = dampening;

            wheel.brakeTorque = isBreaking ? brakeStrength : 0;
        }

        foreach (var wheel in wheelsSteer)
        {
            WheelCollider collider = wheel.GetComponent<WheelCollider>();
            Transform trans = wheel.transform;

            collider.steerAngle = currentSteerAngle;
            collider.wheelDampingRate = dampening;
            //UpdateWheelVisual(collider, trans);
        }
    }


}
```

Figura 10: Función FixedUpdate

- El script `RotateElement` permite al usuario controlar la rotación del elevador en el entorno de realidad virtual mediante el uso de controles manuales. A través de la interacción con el eje de entrada vertical del controlador, la variable `upForce` se ajusta y determina la intensidad y dirección de la rotación del objeto `toRotate`. Con restricciones incorporadas que detienen la rotación al alcanzar límites predefinidos, este script asegura una operación fluida y controlada del mecanismo rotacional del elevador.

```
void FixedUpdate()
{
    UpdateForce();

    if (upForce > 0 && rotateUp)
    {
        float rotationAmount = 20f * Time.deltaTime * upForce;
        toRotate.localRotation *= Quaternion.AngleAxis(rotationAmount, Vector3.up);
        rotateDown = true;
    }
    else if (upForce < 0 && rotateDown)
    {
        float rotationAmount = 20f * Time.deltaTime * upForce;
        toRotate.localRotation *= Quaternion.AngleAxis(rotationAmount, Vector3.up);
        rotateUp = true;
    }

    }

void OnTriggerEnter(Collider col)
{
    if(col.gameObject.tag == "RotationLimit" && upForce > 0)
    {
        rotateUp = false;
    }
    else if(col.gameObject.tag == "RotationLimit" && upForce < 0)
    {
        rotateDown = false;
    }
}

void UpdateForce()
{
    if (inputs.Length == 0)
    {
        inputs = GameObject.FindGameObjectsWithTag("GameController");
    }
    else
    {
        foreach (var element in inputs)
        {
            HandController hand = element.GetComponent<HandController>();

            if (hand.isRight == "right" && grab.isGrabbing)
            {
                upForce = hand.axisVal.y * 0.5f;
            }
            else if(!grab.isGrabbing)
            {
                upForce = 0;
            }

        }
    }
}
```

Figura 11: Clase RotateElement

- Los scripts `MoveUp` y `MoveRetractil` trabajan conjuntamente para gestionar el movimiento vertical de un objeto en un entorno de realidad virtual. Mientras que `MoveUp` se encarga del desplazamiento vertical directo del

objeto, `MoveRetractil` controla el movimiento vertical de un componente relacionado, como puede ser un mástil retráctil. Ambos utilizan la entrada de fuerza vertical `upForce`, que es influenciada por la interacción del usuario con el controlador, para determinar la dirección y la intensidad del movimiento. La coordinación entre estos dos scripts asegura un movimiento vertical fluido y restringido, donde `MoveRetractil` también considera las colisiones y limitaciones espaciales para prevenir movimientos no deseados, como alcanzar el tope superior o inferior del recorrido permitido.

```
// Update is called once per frame
void FixedUpdate()
{
    UpdateForce();

    if (goUp && upForce > 0)
    {
        transform.Translate(new Vector3(0, 0, 1f) * upForce * constForce);
        goDown = true;
    }
    else if (goDown && upForce < 0)
    {
        transform.Translate(new Vector3(0, 0, 1f) * upForce * constForce);
        goUp = true;
    }

}

void UpdateForce()
{
    if (inputs.Length == 0)
    {
        inputs = GameObject.FindGameObjectsWithTag("GameController");
    }
    else
    {
        foreach (var element in inputs)
        {
            HandController hand = element.GetComponent<HandController>();

            if (hand.isRight == "right" && grab.isGrabbing)
            {
                upForce = hand.axisVal.y * 0.5f;
            }
            else if (!grab.isGrabbing)
            {
                upForce = 0;
            }

        }
    }
}
```

Figura 12: Clase MoveUp

4.4.4. Control del puente grúa

El control del puente grúa en el entorno de realidad virtual es una parte integral de la experiencia interactiva, permitiendo a los usuarios manipular objetos dentro de un espacio tridimensional

con precisión y realismo. Para el movimiento del puente y del gancho, se utilizan dos scripts esenciales: `CraneController` y `GanchoController`. El primero gestiona las operaciones del puente grúa, como la traslación en diferentes direcciones, mientras que el segundo se ocupa de los movimientos específicos del gancho, facilitando el descenso, ascenso y posicionamiento preciso del mismo. Adicionalmente, para el agarre y manejo de las tablas y otros objetos, se emplea el script `AttachTarget`, que permite un acoplamiento y desacoplamiento efectivo, asegurando que los objetos permanezcan fijos durante el transporte y sean liberados con suavidad en el destino deseado. Juntos, estos scripts crean un sistema de control cohesivo y eficiente para la operación del puente grúa dentro de la simulación.

- El script `CraneController` es el núcleo del sistema de control del puente grúa, permitiendo el movimiento lateral, vertical y frontal del motor, la cabina y las cuerdas del puente grúa. Utiliza la variable `power` para determinar la intensidad y dirección del movimiento, mientras que `state` indica el tipo de movimiento que se está ejecutando. Las variables booleanas como `isBack`, `isFront`, `isLeft`, `isRight`, `isDown` y `isUp` actúan como interruptores de límite para evitar que la grúa se mueva más allá de sus límites físicos. Por otro lado, el `GanchoController` trabaja en conjunto con el `CraneController`, específicamente monitoreando las colisiones con etiquetas específicas para gestionar y restringir el movimiento vertical del gancho, evitando que se mueva más allá de los límites superior e inferior. Ambos scripts aseguran una operación fluida y segura del puente grúa, permitiendo al usuario manipular cargas con precisión dentro del entorno virtual.
- El script `AttachTarget` se encarga de la mecánica de agarre y sujeción de objetos, como tablas, en un entorno de realidad virtual. Utiliza un `HingeJoint` para simular un punto de conexión flexible entre el objeto y un punto de amarre, permitiendo un movimiento pendular controlado. La detección de la proximidad del objeto a un objetivo y la entrada del usuario son cruciales para activar el agarre. Cuando el usuario presiona el botón de agarre y el objeto está en la posición correcta, el script activa la textura de amarrado y configura las propiedades del `HingeJoint`, estableciendo límites y velocidad para simular un movimiento realista. Si el usuario suelta el botón de agarre o el objeto se aleja demasiado, el script desactiva la conexión, lo que permite liberar el objeto de manera controlada. Además, el script interactúa con un panel de instrucciones para guiar al usuario en el proceso de manejo de los objetos.

```
// Update is called once per frame
void FixedUpdate()
{
    //Check stop state and power value
    if (power != 0 && state != "Block")
    {
        //Lateral movement
        if (state == "left_right")
        {
            if (power > 0 && !isLeft)
            {
                motor.transform.Translate(new Vector3(1f, 0, 0) * power * constForce);
                isRight = false;
            }
            else if (power < 0 && !isRight)
            {
                motor.transform.Translate(new Vector3(1f, 0, 0) * power * constForce);
                isLeft = false;
            }
        }

        //Vertical movement
        if (state == "up_down")
        {
            if(power > 0 && !isUp)
            {
                ropes.transform.localScale = ropes.transform.localScale + new Vector3(0, 0, 1f) * (-power * constForce * scaleValue);

                foreach (var ele in gancho)
                {
                    ele.transform.Translate(new Vector3(0, 0, 1f) * power * constForce);
                }

                isDown = false;
            }
            else if (power < 0 && !isDown)
            {
                ropes.transform.localScale = ropes.transform.localScale + new Vector3(0, 0, 1f) * (-power * constForce * scaleValue);

                foreach (var ele in gancho)
                {
                    ele.transform.Translate(new Vector3(0, 0, 1f) * power * constForce);
                }

                isUp = false;
            }
        }

        //Frontal movement
        if (state == "back_front")
        {
            if (power > 0 && !isBack)
            {
                overhead.transform.Translate(new Vector3(0, 1f, 0) * power * constForce);
                isFront = false;
            }
            else if (power < 0 && !isFront)
            {
                overhead.transform.Translate(new Vector3(0, 1f, 0) * power * constForce);
                isBack = false;
            }
        }
    }
}
```

Figura 13: Función FixedUpdate de la clase CraneController

4.4.5. Gestor de cuestionarios

Con el objetivo de optimizar y simplificar el proceso de creación de cuestionarios, se diseñó una clase denominada `QuizManager` que estandariza la construcción de paneles de preguntas. El funcionamiento de esta clase se desglosa en los siguientes pasos:

1. Se definen variables para almacenar todas las preguntas, la pregunta en curso, la respuesta acertada y los elementos de la interfaz de usuario (UI).
2. Se implementa la función `InitQuiz`, cuyo propósito es inicializar todas las variables necesarias y posteriormente invocar a `SetNextQuestion` para dar comienzo al cuestionario.

3. Las funciones `SetNextQuestion` y `SetAnswerOptions` se encargan de actualizar las variables y componentes del panel, como textos y botones, asignando los datos de la nueva pregunta y desechando la anterior si fuera necesario.
4. La función `SelectButton(Button but)` se activa cuando se presiona un botón de respuesta. Para preguntas de respuesta única, esta función conducirá a la ejecución de `CheckResponse`.
5. La función `CheckResponse` contiene la lógica necesaria para verificar si la respuesta seleccionada es correcta. En caso afirmativo, el usuario avanzará al siguiente panel; de lo contrario, tendrá la oportunidad de intentar responder nuevamente.

4.4.6. Traductor

Este proceso actúa sobre cada una de las escenas en segundo plano para proporcionar a cada elemento de la interfaz de usuario de la herramienta los textos traducidos a la lengua seleccionada.

El primer paso ha consistido en traducir los textos a los idiomas de los socios participantes. Estas traducciones se estructuran en un formato en el que cada palabra o frase está asociada a una clave única, lo que facilita la recuperación del texto en el idioma deseado. Para gestionar este sistema, se han desarrollado dos clases específicas:

1. `MainTranslate`: Esta clase tiene la función de enlazar los scripts de traducción con el entorno operativo, o sea, el código. Cada vez que se necesite localizar una clave específica generada en el script mencionado, `MainTranslate` proporcionará la palabra o frase correspondiente en el idioma correcto.
2. `SceneTranslate`: La responsabilidad de esta clase es comunicar a `MainTranslate`, al inicio de cada escena, cuál es el idioma que se ha seleccionado en el menú de opciones.

Con estos procesos activos, el único requerimiento para el usuario es reemplazar cada una de las líneas de texto que desee insertar por la siguiente línea de código:

```
text = MainTranslate.Fields[textKey];
```

Figura 14: Línea para traducir.

4.4.7. Interacción con rayos

Esta funcionalidad habilita al usuario para que pueda interactuar con objetos o paneles que estén localizados a una distancia determinada. Se logra mediante la creación de un rayo virtual que parte de la mano del usuario, permitiéndole realizar acciones específicas sobre ciertos objetos. En el contexto de este proyecto, es crucial que el usuario pueda responder a las preguntas planteadas utilizando esta técnica. Para implementar este tipo de interacción, se deben seguir los pasos descritos a continuación:

1. Generar un objeto Ray Interactor para cada mano, accesible a través del menú GameObject/XR.
2. Elegir los objetos con los que se va a interactuar, ajustando el parámetro Raycast Mask del componente XR Ray Interactor. Este ajuste permite seleccionar la Capa en la que actuará el rayo, por lo tanto, los objetos destinados a interactuar con el rayo deben estar asignados a la Capa correspondiente.
3. En este escenario, se ha optado por seleccionar la capa UI para los paneles proporcionados por Unity.
4. Para evitar que el rayo esté visible constantemente, es necesario modificar el atributo Invalid Color Gradient del componente XR Interactor Line Visual a un valor transparente. De esta forma, el rayo solo se hará visible cuando la mano esté apuntando a un panel.
5. Ambos objetos se incorporarán como hijos del objeto Camera Offset, el cual fue creado previamente en la fase de preparación de la escena, asegurando que formen parte del objeto vinculado al dispositivo VR.

Con la implementación de estos pasos, se logra la funcionalidad requerida para generar e interactuar con los distintos paneles descritos anteriormente.

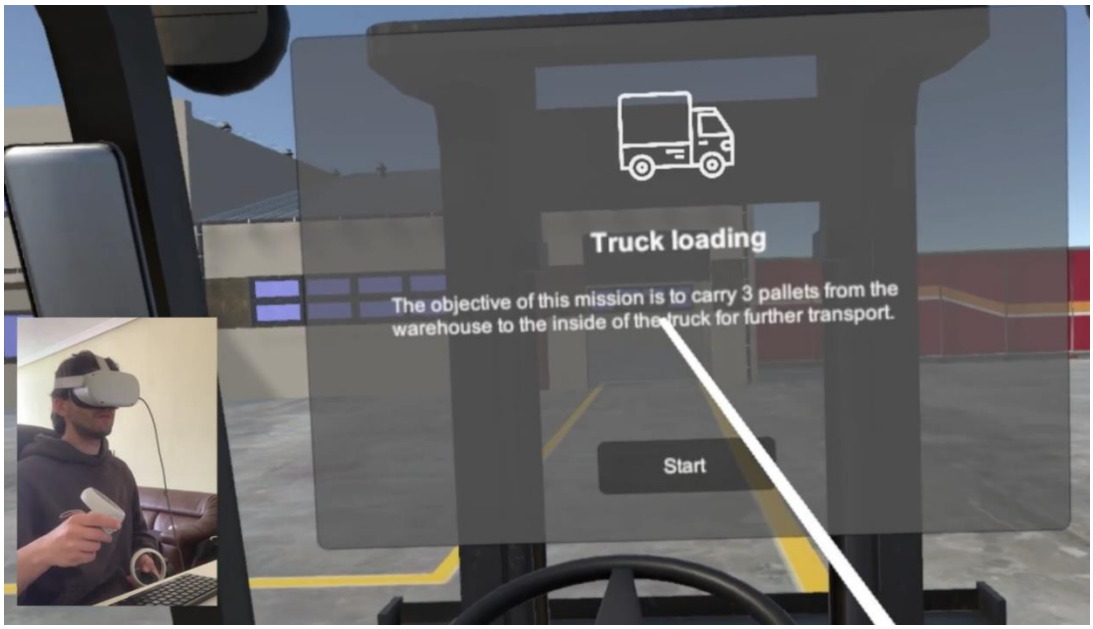


Figura 15: Interacción con Rayos

5. VÍDEOS DE YOUTUBE

Para complementar la documentación técnica y ofrecer una perspectiva más dinámica de nuestra herramienta VR, hemos seleccionado una serie de videos de YouTube que demuestran

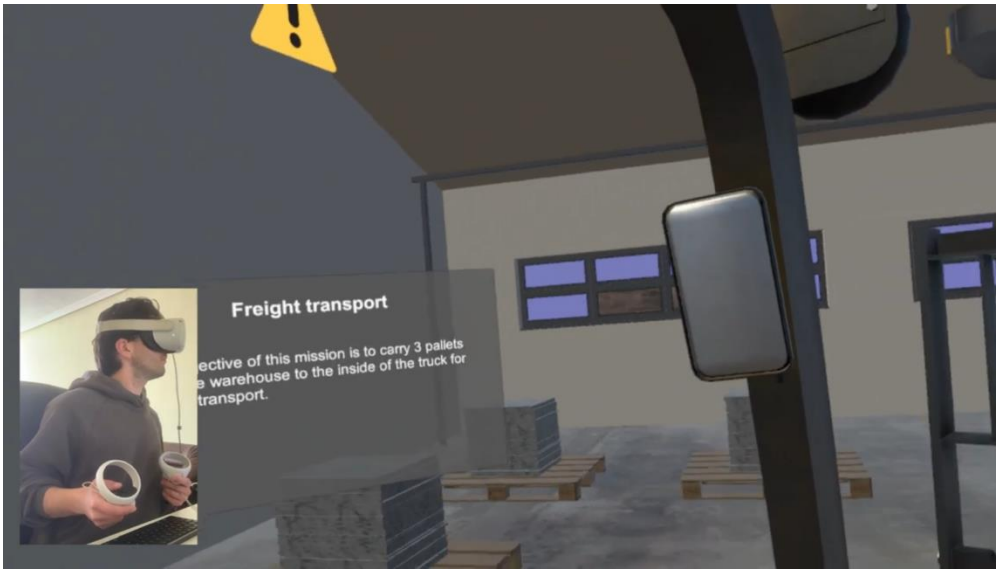
su funcionalidad. A continuación, presentamos estos materiales audiovisuales que brindan una muestra clara y directa del rendimiento y las posibilidades que nuestra solución VR ofrece.

- Promotion: https://www.youtube.com/watch?v=Ogs2WzzCRe0&ab_channel=AEIPiedraNatural

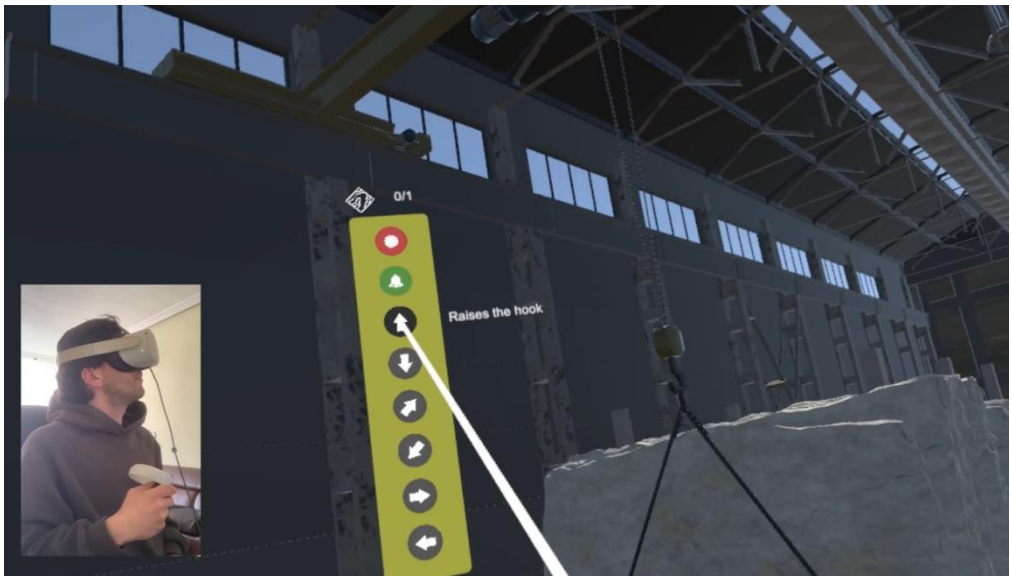
- Main Menu: https://www.youtube.com/watch?v=I7j9rTMeWmo&ab_channel=AEIPiedraNatural

- Forklift – Storage: https://www.youtube.com/watch?v=hCkCl9ihiLU&t=14s&ab_channel=AEIPiedraNatural

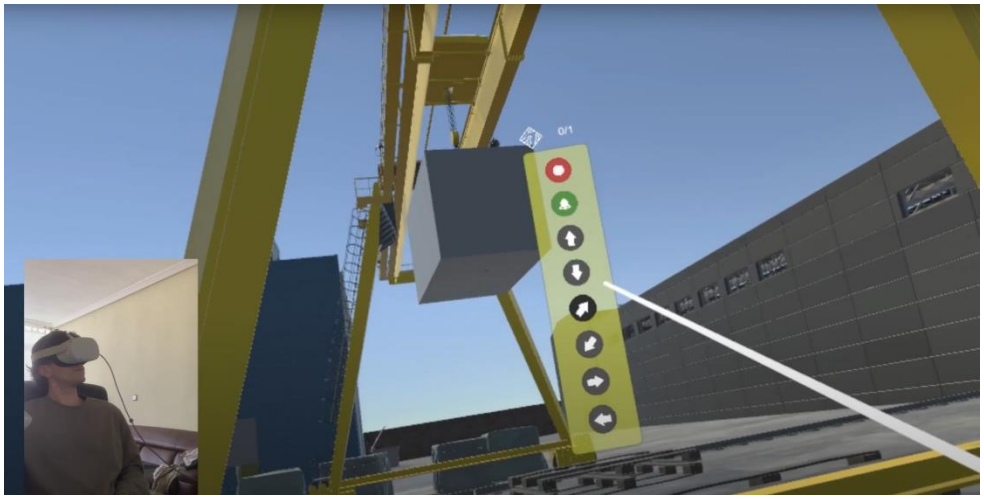
- Forklift – Truck loading: https://www.youtube.com/watch?v=NnpA44V6DMY&t=13s&ab_channel=AEIPiedraNatural



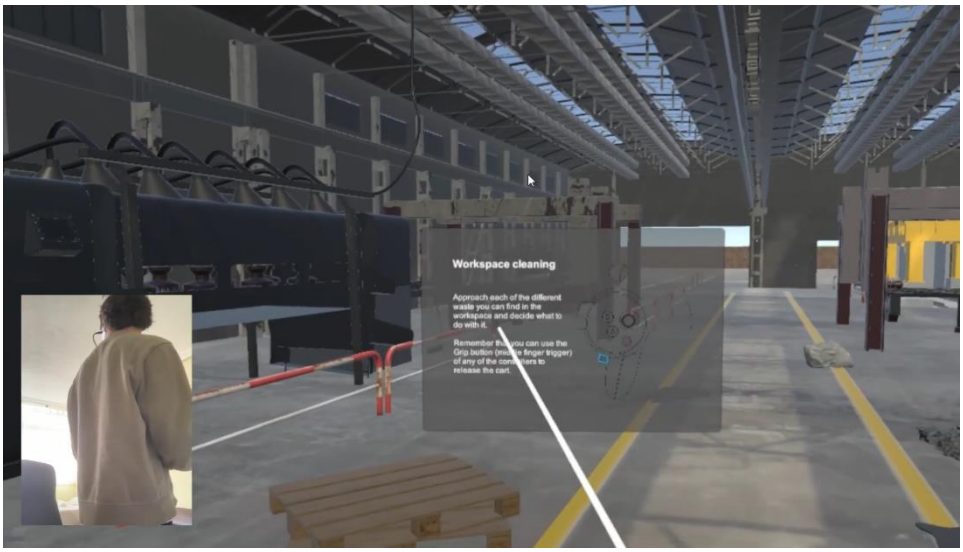
- Overhead Crane – Slabs: https://www.youtube.com/watch?v=LK9pSCPLMrw&ab_channel=AEIPiedraNatural



- Overhead Crane – Blocks: https://www.youtube.com/watch?v=tVyFFRJyQ4U&ab_channel=AEIPiedraNatural



- cleaning: hattops://www.youtube.com/watch?v=twiffarjyak4u&ab_channel=apedarentural



- Waste management: https://www.youtube.com/watch?v=mojMZ2G6Huc&ab_channel=AEIPiedraNatural