UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Ampliación y mejora de un laboratorio virtual de neumática en el metaverso para la docencia

MATERIAL DIDÁCTICO

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Emilio Morales Luna

ASESOR DE MATERIAL DIDÁCTICO

M.F. Gabriel Hurtado Chong

Ciudad Universitaria, Cd. Mx., 2024

AGRADECIMIENTOS

A mis padres Víctor Hugo Morales Ramírez y Sandra Luna Sandoval, que han inculcado en mí un interés en la docencia, además he contado con su apoyo en mis aspiraciones académicas, personales y profesionales desde el día en que nací.

A la UNAM por haberme brindado educación desde el nivel medio superior hasta la licenciatura, donde me he desarrollado como individuo y visto en mí mis capacidades y aptitudes que irán dictando mi desarrollo profesional a lo largo de mi vida.

A mis asesores Yair Bautista y a Gabriel Hurtado Chong que guiaron mi proceso para el servicio social, el cual puse a prueba conocimientos y habilidades para organizarme, liderar y resolución de problemas.

También a los proyectos UNAM-DGAPA-PAPIME PE104922 “Neumática educativa en la realidad virtual (NERV)” y PE108323 “Implementación de gemelos digitales para la enseñanza en automatización industrial”, por el apoyo brindado para la realización de este material didáctico.

Agradezco la participación de los sinodales seleccionados que han sido parte de mi formación y quienes respeto inmensamente por sus habilidades de docencia, dedicación y conocimientos en su área.

CONTENIDO

1 INTRODUCCIÓN

La Facultad de Ingeniería tiene sus carreras organizadas en divisiones encargadas de administrar sus asignaturas correspondientes, una de ellas es la División de Ingeniería Mecánica e Industrial (DIMEI), que prepara a futuros ingenieros de las carreras de Ingeniería Mecánica, Industrial, Mecatrónica y en Sistemas Biomédicos. La DIMEI imparte la asignatura de Automatización Industrial, que consiste en técnicas para el manejo de procesos de manufactura con poca o nula intervención humana, utilizando principalmente sistemas mecánicos, eléctricos y computacionales, para tener su control y llevar a cabo operaciones de la producción [1].

El alumno, a lo largo del curso, conocerá los fundamentos de la automatización, identificará los sensores y actuadores que existen en la industria, su principio de funcionamiento, y diseñará procesos industriales automatizados. Los procesos que estudiará el alumno están divididos en tres clases:

1) Electrónica : Se introducen los controladores lógicos programables o PLC por sus siglas en inglés, "programmable logic controller", que son aparatos electrónicos operados digitalmente, usa una memoria programable para el almacenamiento interno de instrucciones. Se implementan funciones específicas, lógica, secuenciación, registro, control de tiempos, conteo y operaciones aritméticas para controlar procesos por medio de módulos de entrada y salida.

2) Neumática : Consiste en la aplicación de múltiples principios y leyes de los gases para aprovechar el aire comprimido y realizar algún trabajo. Se dan a conocer los elementos que conforman un circuito neumático, tales como la unidad de mantenimiento, redes neumáticas, actuadores neumáticos como los pistones, distintos tipos de válvulas, etc.

3) Mixta: Aquí, la energía eléctrica sustituye a la energía neumática como elemento principal para la generación y transmisión de señales de control. Los elementos a usar son aquellos que sirven para la manipulación y acondicionamiento de señales de voltaje y corriente que deberán ser aplicados para dispositivos de conversión de energía eléctrica a neumática y activar los actuadores.

La automatización industrial es una asignatura teórica-práctica, lo cual implica que el alumno además de aprender en un aula, también asistirá a un laboratorio que contiene equipo electrónico y neumático. El alumno realizará, a lo largo del semestre, catorce prácticas en total; las primeras cuatro son actividades de procesos automatizados con el empleo de PLC; las siguientes siete abarcan prácticas enfocadas a la neumática y las últimas tres son de electroneumática. Las prácticas se realizan dentro de la Facultad en el edificio **O** del Anexo de Ingeniería (ver figura 1).

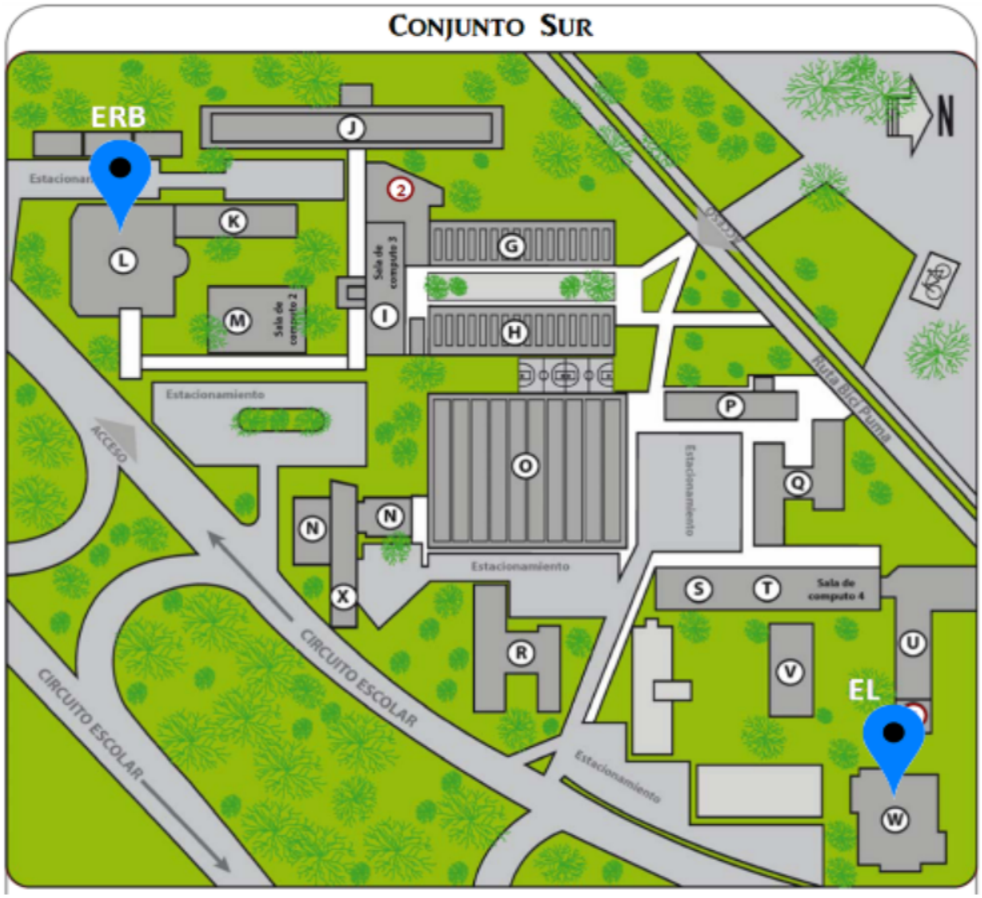


Figura 1 Mapa del Anexo de Ingeniería UNAM.

El laboratorio está conformado por mesas y sillas en el centro, mientras que en los márgenes del laboratorio se encuentran las mesas de trabajo que están equipadas con los elementos neumáticos necesarios. Durante una sesión del laboratorio, el docente inicialmente explicará los objetivos y actividades de la práctica a realizar, después se formarán brigadas de pares de alumnos, se dirigirán a su mesa de trabajo y recibirán un maletín que contiene los demás elementos para realizar sus actividades.

Este documento reporta una contribución plasmada, para la asignatura de automatización industrial que ha estado en desarrollo durante varios años, y consiste en un laboratorio virtual de neumática donde el alumno diseñará circuitos neumáticos para controlar procesos industriales. A lo largo del escrito se describirán las herramientas utilizadas detalladamente y se introducirán los conceptos que se emplearon para este fin.

2 ANTECEDENTES DEL PROYECTO

2.1 Emergencia sanitaria

Debido a la emergencia sanitaria global del 2019, múltiples establecimientos e instituciones cerraron y pararon actividades como medida preventiva para evitar contagios. Las instituciones educativas de todos los niveles buscaron distintos métodos y herramientas para reanudar actividades. Las clases en línea se volvieron un estándar, esto creó varios obstáculos, debido a que la mayoría de las carreras que se imparten en la Facultad de Ingeniería tienen planes de estudio que se enfocan en el aprendizaje práctico por medio de laboratorios y prácticas de campo donde se promueve el trabajo en equipo. Los docentes, con su esfuerzo, han propuesto soluciones usando distintas herramientas para impartir la materia y colaborar en la formación de los futuros profesionales de la ingeniería [2].

2.2 Entornos virtuales

Las tecnologías de la información y las comunicaciones, *TIC*, son herramientas que enfatizan el papel de integrar las telecomunicaciones y sus tecnologías, así como el software necesario, los agentes de intercambio de información de aplicaciones, o middleware y producción audiovisual, los cuales permiten a los usuarios almacenar, transmitir, manipular y acceder a información [3].

Los entornos virtuales son una de las *TIC* más interactivas, capaces de mostrar modelos, ambientes y escenas tan realistas como es posible, además, existe una mayor libertad para manipular sus características y funciones. Para que haya interacción directa con múltiples usuarios, se utiliza la interconectividad, con la finalidad de poder realizar actividades en conjunto y en tiempo real.

En la actualidad, existen proyectos ejemplares que utilizan entornos virtuales para la educación. Por ejemplo, la compañía VictoryXR fundada en 2016, se ha encargado del desarrollo de tecnologías de realidad virtual para la educación y ha creado entornos virtuales educativos para las siguientes plataformas:

1) HTC Vive

2) HTC Vive Focus

3) Windows Mixed Reality

4) Oculus Rift

5) Oculus Go

6) Oculus Quest

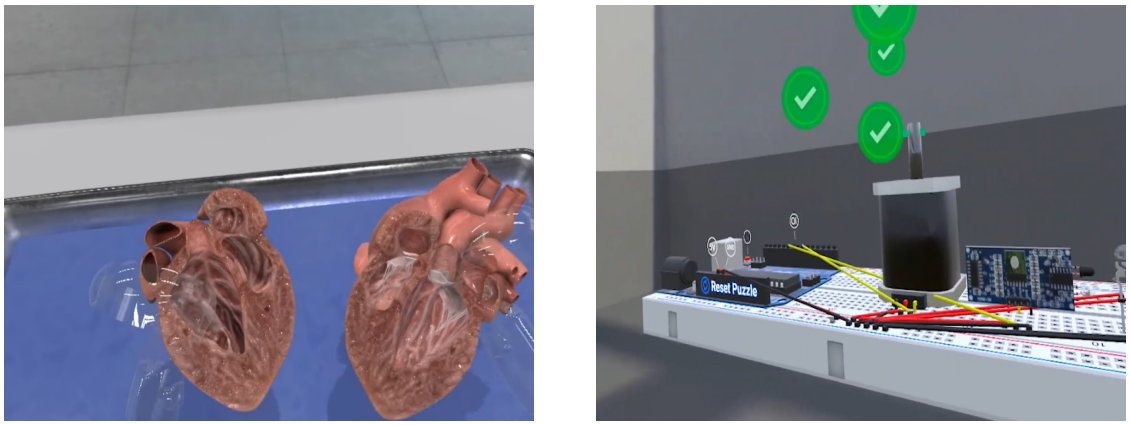
7) Pico

8) Lenovo Mirage

9) Google Cardboard.

VictoryXR ofrece distintos servicios para estudiantes de todos los niveles, en específico el servicio llamado Academy consiste en clases en el metaverso para estudiantes de educación media, media superior y superior, cuenta con un campus y cursos que abordan temas como robótica, biología, hablar en público, anatomía, etc. Cada curso brinda ambientes educativos virtuales, clases en vivo pregrabadas y más de nueve mil modelos funcionales que apoyan la enseñanza.

En la figura 2 *a* se muestra que en el curso de anatomía se tiene un modelo de un corazón donde el estudiante puede realizar incisiones y ver de forma realista su estructura. En la figura 2 *b* se aprecia una práctica del curso de robótica cuyo objetivo es diseñar un robot y el alumno realizará un circuito por medio de un modelo de "protoboard" con los elementos electrónicos correspondientes como motores y microprocesadores.



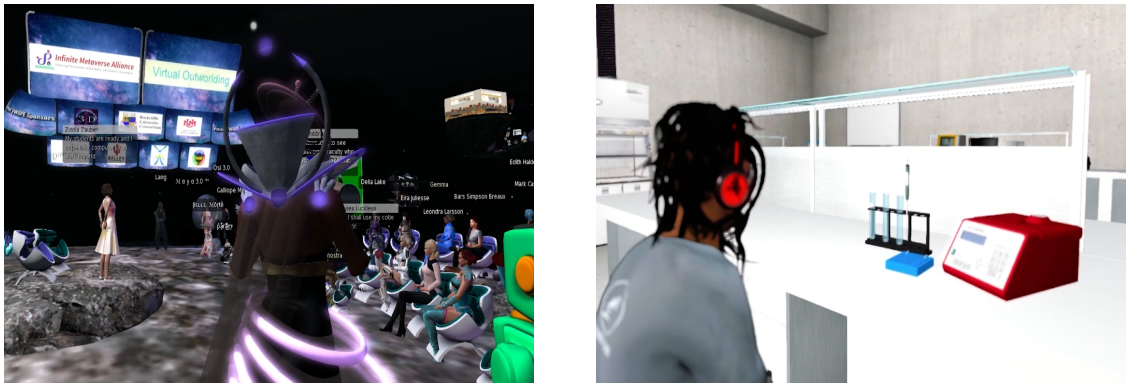
(a) (b)

Figura 2 Modelos tridimensionales en VictoryXR; a) modelo de un corazón para curso de anatomía; b) modelo de una "protoboard" para curso de robótica

Además, el alumno puede emplear la plataforma VXRLabs que contiene simuladores interactivos enfocados en temas específicos, diseñado para ser usado con múltiples usuarios. Los laboratorios con características de interconectividad brindan la oportunidad a los estudiantes de colaborar e interactuar con sus compañeros de formas más atractivas sin importar la distancia [4] [5].

Otro ejemplo notorio es Second Life, lanzado en 2003 por Linden Labs, donde los usuarios, también llamados residentes, gratuitamente pueden explorar el mundo virtual, interactuar con otros residentes, establecer relaciones sociales y participar en diversas actividades tanto individuales como en grupo.

Decenas de universidades y escuelas han dado clases y conferencias en Second Life, incluyendo varios campuses de la Universidad de Princeton, de Carolina del Norte, Colegio Bryn Mawr, entre otros, en Estados Unidos. En la figura 3 se pueden observar algunos usos en el ámbito de la enseñanza, *a* donde se muestra una conferencia y *b* se muestra un residente haciendo uso de un laboratorio de química forense.



(a) (b)

Figura 3 Second Life; a)Conferencia en Second Life; b)Laboratorio de química en Second Life.

También esta plataforma ha sido usada para entrenamiento corporativo. Compañías como IBM, Dell e Intel han transferido sus programas de entrenamiento al mundo virtual de Second Life, esto le da la posibilidad a los diseñadores instruccionales la capacidad de crear modelos que permiten a los aprendices interactuar por medio de sus avatares. Los tutores podrán ilustrar conceptos técnicos en formas novedosas y significativas al poder manipular la escala y perspectiva de los modelos. Por ejemplo, en un curso de capacitación para ser técnico de computadores se crearon modelos que representan microprocesadores de una computadora, los cuales tienen ciertas características como la capacidad de modificar su tamaño relativo al tamaño del avatar, permitiendo al aprendiz un mejor entendimiento del modelo y así, mejorando su aprendizaje [6].

Con estos ejemplos, se puede tener una visión de los objetivos que se quieren alcanzar al diseñar el ambiente virtual del laboratorio de neumática. Se busca crear e implementar un laboratorio virtual donde se puedan impartir clases y realizar las prácticas de neumática a distancia. La interactividad directa con los usuarios es un aspecto esencial para la colaboración en equipo; finalmente se requiere que las acciones dentro del laboratorio sean visibles para todos los participantes en tiempo real. En la siguiente sección se introducen las herramientas y software que se utilizaron para desarrollar el proyecto.

2.3 Software y herramientas

2.3.1 Unity

Unity es un motor de videojuegos y entornos virtuales multiplataforma creado por Unity Technologies en 2004. Este motor ayudará a administrar el proyecto, con el que se puede crear, editar y exportar varios modelos que formarán parte del entorno. Para el proyecto, cada modelo fue programado con diferentes características, dependiendo de la funcionalidad requerida, como el modelo del laboratorio

el cual debe de ser lo suficientemente grande para que los usuarios exploren los cuartos que ofrece. También se diseñaron modelos que representan los múltiples elementos neumáticos que se encuentran en el laboratorio de neumática de la Facultad de Ingeniería, los cuales los estudiantes manipularán para construir circuitos neumáticos.

Para establecer las características, funcionalidad e interactividad a un modelo, Unity ofrece múltiples herramientas que se pueden usar para construir un entorno virtual. La programación de las funciones deseadas en los modelos, se implementa a través de código utilizando el lenguaje de programación C#, junto con los archivos de biblioteca de Unity. Además, se implementaron animaciones que indican activación o desactivación para algunos elementos, como llaves de válvulas, los vástagos de los pistones, etc. En la figura 4 se muestra el entorno de trabajo de Unity; en el documento se explorarán las herramientas con base en las actividades desarrolladas de forma cronológica.

Figura 4 Entorno de Unity.

2.3.2 VRChat

Es una plataforma en línea de realidad virtual que permite a sus usuarios interactuar con otros avatares y mundos creados por la misma comunidad. Los usuarios que deseen desarrollar contenido para la VRChat, necesitan usar un kit de desarrollo de software, o SDK, por sus siglas en inglés. Desde Unity se usarán los archivos de biblioteca del kit que dará acceso a distintas funciones y métodos, los cuales servirán para desarrollar el laboratorio; una vez que el entorno esté en condiciones satisfactorias, se subirá al portal de VRChat [7].

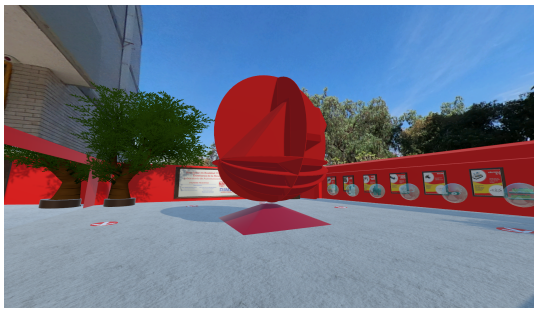
Una manera para obtener el SDK es descargando el software de asistencia llamado *Creator Companion*, que es un programa que descarga los SDK's más recientes; además de gestionar los archivos, uno puede crear copias de respaldo y actualizar el proyecto. El kit contiene un compilador llamado Udon, que tiene archivos de biblioteca para programar en C# llamadas: "VRC.SDKBase", "VRC.Udon" y "VRC.Udon.Common". Los archivos de biblioteca permitirán crear código compatible para VRChat, entonces el entorno tendrá interconectividad y se podrán aplicar las características que ofrece VRChat a los objetos del laboratorio. La figura 5 presenta el software y el kit de desarrollo.

Figura 5 Herramientas y plataformas para desarrollo del proyecto.

Además de programar en C# usando *Udon*, también se puede crear código usando *UdonGraph*, que es un compilador gráfico a base de nodos. El compilador utiliza bloques que representan variables, funciones y declaraciones que son conectados lógicamente por líneas o nodos. Cumple la misma función de cualquier otro compilador tradicional, pero es una alternativa para programadores que encuentran lo visual más sencillo [8] [9].

2.4 NERV

Este proyecto PAPIME se ha desarrollado a través de varios semestres por múltiples colaboradores; antes de esta contribución, ya se tenía un laboratorio funcional que fue utilizado durante la crisis sanitaria. El entorno cuenta con un espacio amplio, donde el usuario verá en medio la escultura del Anexo de la Facultad la *Leonardita*, presentada en figura 6 a. Los usuarios se reunirán en esta sala de espera, que también cuenta con una sección que expone los modelos tridimensionales de los elementos neumáticos que podrán usar en el laboratorio; cada modelo tiene un cartel con una imagen que contiene información relevante y un diagrama que lo representa, así como se muestra en la figura 6 b.



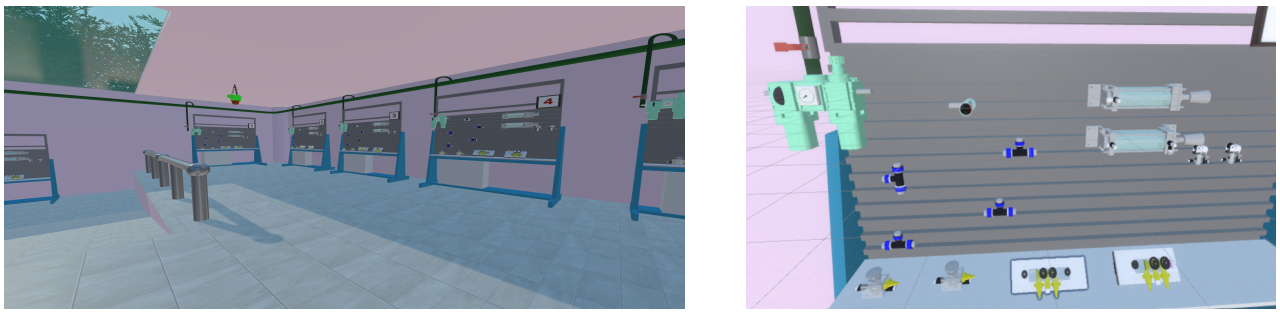

(a) (b)

Figura 6 Cuarto de inicio; a) La *Leonardita*; b) Museo

El laboratorio se accede por unas escaleras y cuenta con diez mesas de trabajo distribuidas ordenadamente en sus paredes laterales, hay una mesa más bajando una rampa donde el docente realizará demostraciones, como se aprecia en la figura 7 a. En la figura 7 b se muestra una mesa de trabajo con

sus elementos neumáticos; estos elementos interactúan entre sí cuando se diseñe un circuito neumático. Cada mesa contiene los siguientes elementos neumáticos que el alumno manipulará:

1) Pistón de simple efecto
2) Pistón de doble efecto
3) Finales de carrera
4) Dos botones de válvula 3/2
5) Unidad de mantenimiento o FRL cuyas siglas indican las funciones de filtración, regulación y lubricación
6) Piloto
7) Válvula 5/2 monoestable
8) Válvula 5/2 biestable.



(a) (b)

Figura 7 Laboratorio virtual; a) Mesas de trabajo; b) Elementos neumáticos

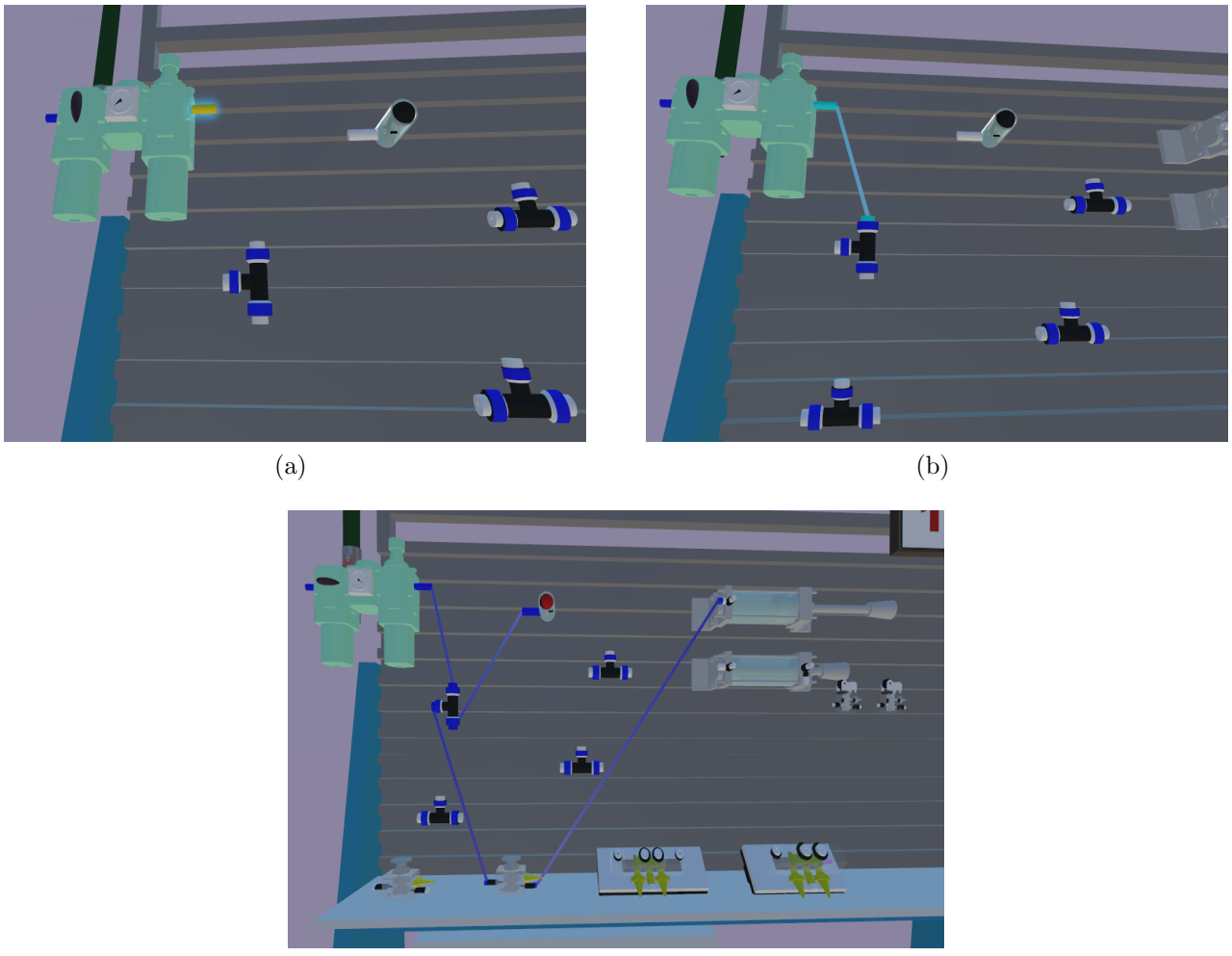
Los usuarios utilizarán conectores tipo 'T' para realizar conexiones con los elementos mencionados anteriormente; el alumno podrá tomar, manipular y posicionar el conector a su gusto. Si un usuario desea llevarse algun conector a otra mesa que no le corresponde, no podrá interactuar con los demás elementos neumáticos, esto es porque cada mesa y su conjunto de elementos comparten un guión de código, también llamado script, de nombre *Man_Valv*, el cual funciona como un panel de control que registra los elementos neumáticos presentes de una mesa, posteriormente gestiona sus funciones y estados de cada elemento, entonces al querer usar otro elemento ajeno a su mesa no se identificará correctamente.

Además, varios elementos de las mesas realizan animaciones, algunos necesitan interacción directa del usuario para cambiar de estado, como los botones o válvulas de paso de aire y de la unidad de mantenimiento, mientras que otros elementos reproducen una animación cuando interactúan con otros elementos, como los finales de carrera que son empujados por el pistón, así como el mismo pistón que, al ser alimentado, avanza o retrocede su vástago y actúan las partes que conforman su mecanismo.

Cuando se desee realizar una práctica después de conocer la teoría, los objetivos e identificar los elementos que se usarán para las actividades, los alumnos y el profesor ingresarán al entorno virtual. Una vez dentro, se crearán las brigadas y elegirán una mesa para realizar las actividades, diseñarán los circuitos creando

las conexiones adecuadas, hasta que esté listo para probar. Las mesas cuentan con una tubería tipo *cuello de cisne*, la cual se encuentra conectada a una red neumática de anillo y, al activarse, entregará aire a la unidad de mantenimiento FRL, que se encarga de filtrar, lubricar y regular la presión para que la presión de aire sea constante, por lo general 6 bares, para todo el circuito neumático diseñado.

El usuario será capaz de identificar fácilmente con cuáles elementos podrá interactuar, ya que al posicionar el cursor sobre un objeto, se iluminará y aparecerá una imagen que indica el botón que se debe apretar para usarlo. Todos los elementos de la mesa de trabajo tienen unos modelos en forma de cilindros pequeños de color blanco, que representan los ductos de entrada y salida de cada elemento. Al seleccionar un ducto con el botón de interacción, éste cambiará a amarillo, indicando que será el origen de una conexión que se creará, posteriormente se da clic a otro ducto blanco y creará una línea recta la cual representa una manguera de color cian uniendo ambos ductos. El color cian de la manguera indica que no hay aire circulando en ese momento, hasta que el FRL sea activado después de encender la válvula de paso de aire, lo que alimentará a todo el circuito cambiando el color de las mangueras a un azul marino indicando flujo de aire. En la figura 8 se puede observar un conjunto de imágenes ordenadas que describen el proceso para realizar conexiones neumáticas.



(a) (b)

(c)

Figura 8 Pasos para crear un circuito neumático; a) La salida del FRL será el origen del circuito; b) Ya que se tiene un origen falta seleccionar un destino para que aparezca la manguera; c) Se hacen las conexiones necesarias hasta tener el circuito esperado

Es importante saber cómo funcionan los elementos para hacer las conexiones correctamente; en caso que haya un error, se presentará una fuga y el ducto de salida que presente la fuga será de color rojo, seguido de un sonido de aire escapando; en ese estado ya no se puede hacer nada más que desactivar el FRL y corregir el circuito. Con las características y funciones que tiene el entorno desarrollado, el usuario será capaz de entrar al laboratorio y realizar las primeras prácticas de neumática, que consisten en usar los pistones de simple y doble efecto. En la siguiente sección se hablará del trabajo realizado para expandir y mejorar las cualidades presentes del entorno; una vez cumplidas esas metas se pretende subir el entorno a la plataforma de VRChat.

3 TRABAJO DESARROLLADO

3.1 Integración del temporizador neumático

El desarrollo del temporizador neumático positivo y negativo fue la primera contribución al proyecto. Los temporizadores se trabajaron en un archivo personal de Unity, y una vez que estos fueron finalizados, se integraron al laboratorio existente. Primero, se diseñaron tres objetos virtuales que simulan el comportamiento del temporizador neumático. En Unity, cualquier objeto, accesorio o personaje creado se le llama *GameObject*, el cual se le pueden añadir características como animaciones, texturas, scripts, entre otros, para darles distintas funcionalidades en el entorno. Los tres GameObject creados representan tres señales que son: la alimentación, que es por donde ingresa el aire, el control, que realiza una cuenta regresiva y la salida de aire que, dependiendo del tipo de temporizador, al terminar el tiempo transcurrido se enciende cuando es positivo y se apaga cuando es negativo.

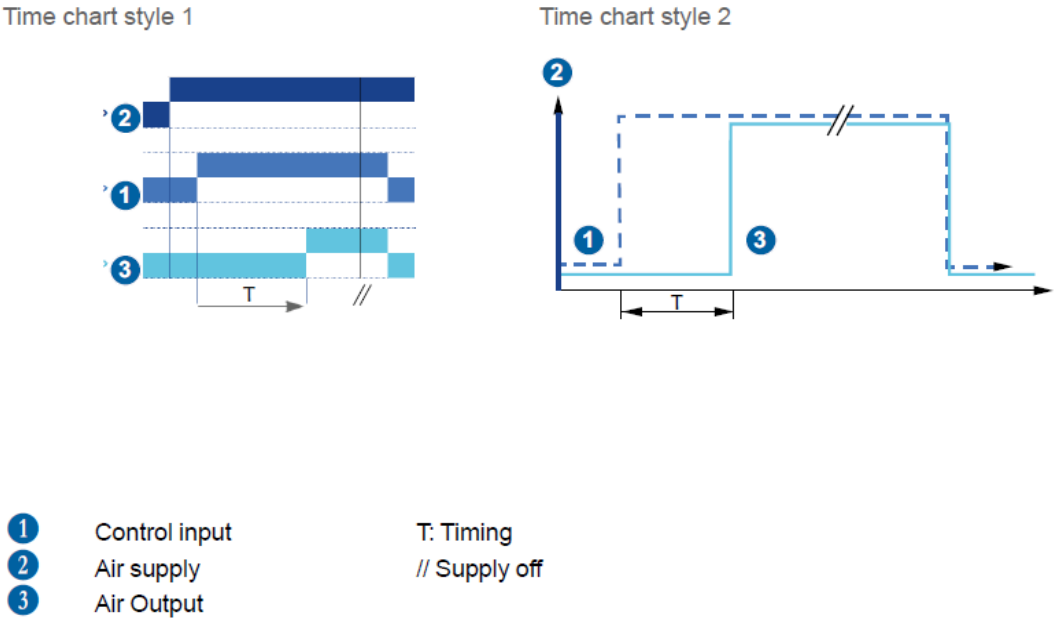


Figura 9 Gráfica del Temporizador Positivo [10].

Lo anterior se apoya en una representación gráfica del funcionamiento de un temporizador positivo (ver figura 9), en la parte inferior de la imagen se aprecian las tres señales (1 Señal de control, 2 Suministro

de aire y 3 Salida de aire). Primeramente, es necesario encender la señal dos antes de las demás señales; después, cuando la señal uno se haya encendido, se espera la variable 'T' que representa el tiempo transcurrido en segundos. Al terminar el tiempo transcurrido 'T', la señal tres se encenderá, y para desactivarla, se debe apagar la señal uno.

En la figura 10 se muestra la representación gráfica del temporizador negativo. Al activarse la señal dos, automáticamente se activará la señal tres, entonces, cuando se activa la señal uno y termina el tiempo transcurrido 'T', se apagará la señal tres.

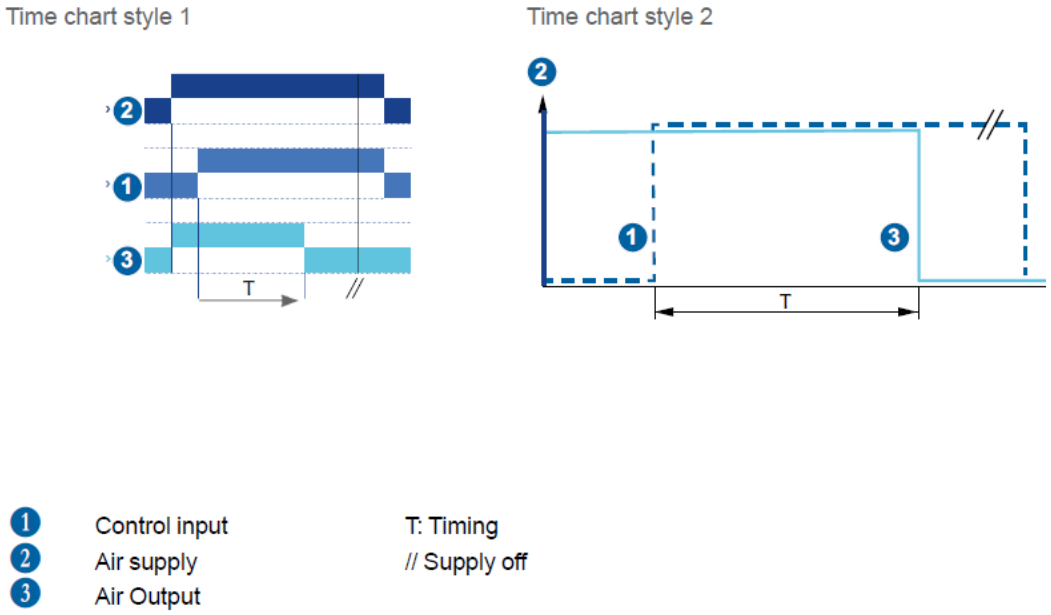


Figura 10 Lógica del Temporizador Negativo [10].

Se creó un código en UdonGraph que contiene tres variables del tipo booleano, las cuales cuentan con dos valores posibles: verdadero (true) o falso (false). Ya que las señales de alimentación y control se activan por el usuario, se añadieron en Unity unos modelos cúbicos a la escena del proyecto y actuarán como botones a partir de un script, el cual es un componente del tipo *Behaviour* que proviene de un código y se le aplica a algún GameObject.

En el software para modelado 3D, los modelos tridimensionales se les llaman *Mesh* o *malla*, los cuales son una colección de vértices, ejes y caras que en conjunto forman un objeto tridimensional. En Unity las mallas pertenecen a la clase de objetos GameObjects; se puede modificar sus características y crear funciones gracias a los múltiples componentes disponibles que, por defecto, cuentan con cuatro componentes esenciales al añadir un Gameobject a una escena:

- *Transform*: determina la posición, rotación, y escala de cada objeto en la escena. Cada GameObject tiene un Transform.

- *Mesh Filter*: toma una malla de los activos del proyecto y los pasa al Mesh Renderer para que sea generada una imagen realista, en adelante renderizada en la pantalla.

- *Mesh Renderer*: toma la geometría del Mesh Filter y lo renderiza en la posición descrita por el componente Transform. Si el Mesh Renderer no está presente, la malla todavía existirá en la escena así

como en la memoria de computador, pero no será renderizado. Dentro de este componente se pueden configurar características como su comportamiento con la luz, sombras, texturas, etc.

- *Box Collider*: crea un cubo para representar la colisión de objetos; se puede habilitar un modo llamado *isTrigger* con el cual se pueden programar eventos del mismo nombre.

Además de estos componentes, se añadió el componente de UdonBehaviour donde residirá el script o guión, hecho en UdonGraph o UdonSharp y le otorgará todas la características necesarias para que el botón funcione. Se muestra en la figura 11 la ventana *Inspector* de un GameObject de nombre Bent22, el cual permite observar y editar los componentes y propiedades que contienen los GameObjects, materiales, activos y configuraciones del editor [11].

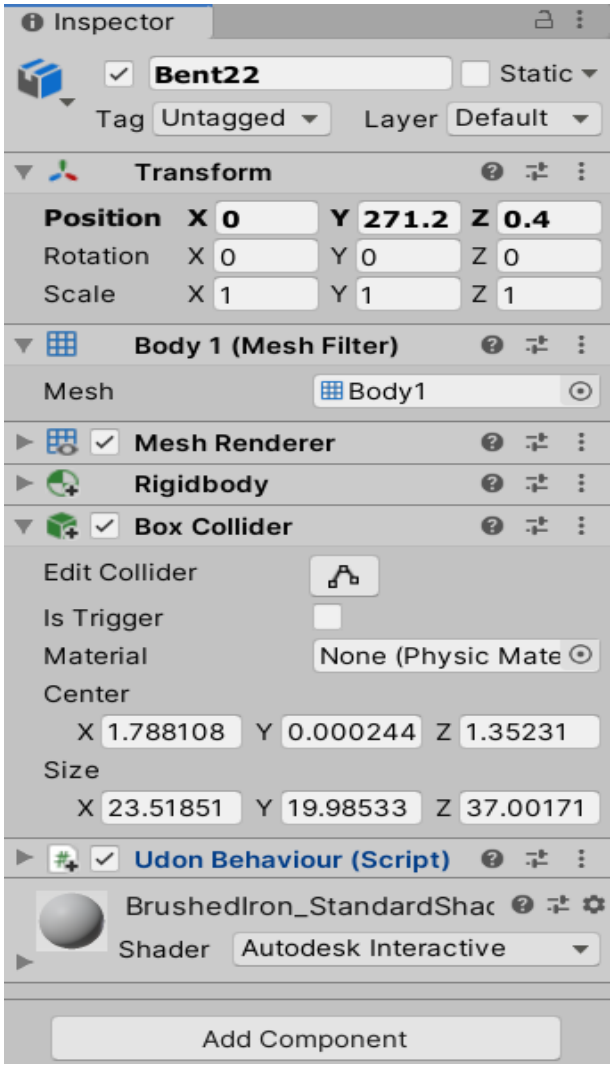


Figura 11 Inspector de Unity.

Una vez que se tiene la malla con sus componentes necesarios, se realizó el código el cual simula un botón que controla la señal de alimentación. Primero, se asignó una variable booleana de nombre 'Es'. Para que cualquier GameObject sea interactivo en UdonGraph, se usa un bloque llamado *Interact*; la función que se realiza después de que el botón virtual sea oprimido es una condición que pregunta el estado actual de la variable 'Es' y cambia su valor usando el bloque *Set*.

Es necesario representar visualmente el estado actual del botón cuando se le da clic, así que se asignaron al guión colores que representan sus estados, el rojo es cuando está apagado/false y verde cuando está encendido/true. Se crearon dos texturas que en Unity son llamadas *materials*; entonces para realizar un cambio de material en una malla, se agregó en el script una variable llamada 'Es_M' de tipo MeshRenderer que representa la malla del botón, se enlaza con el bloque *Set Material* y se une con una variable del tipo material. El cambio de material se agregó a la par del cambio de valor a 'Es' (ver figura 12). Con el script finalizado, se agregó al componente UdonBehaviour del cubo, con lo que se tiene como resultado un botón para la señal de alimentación.

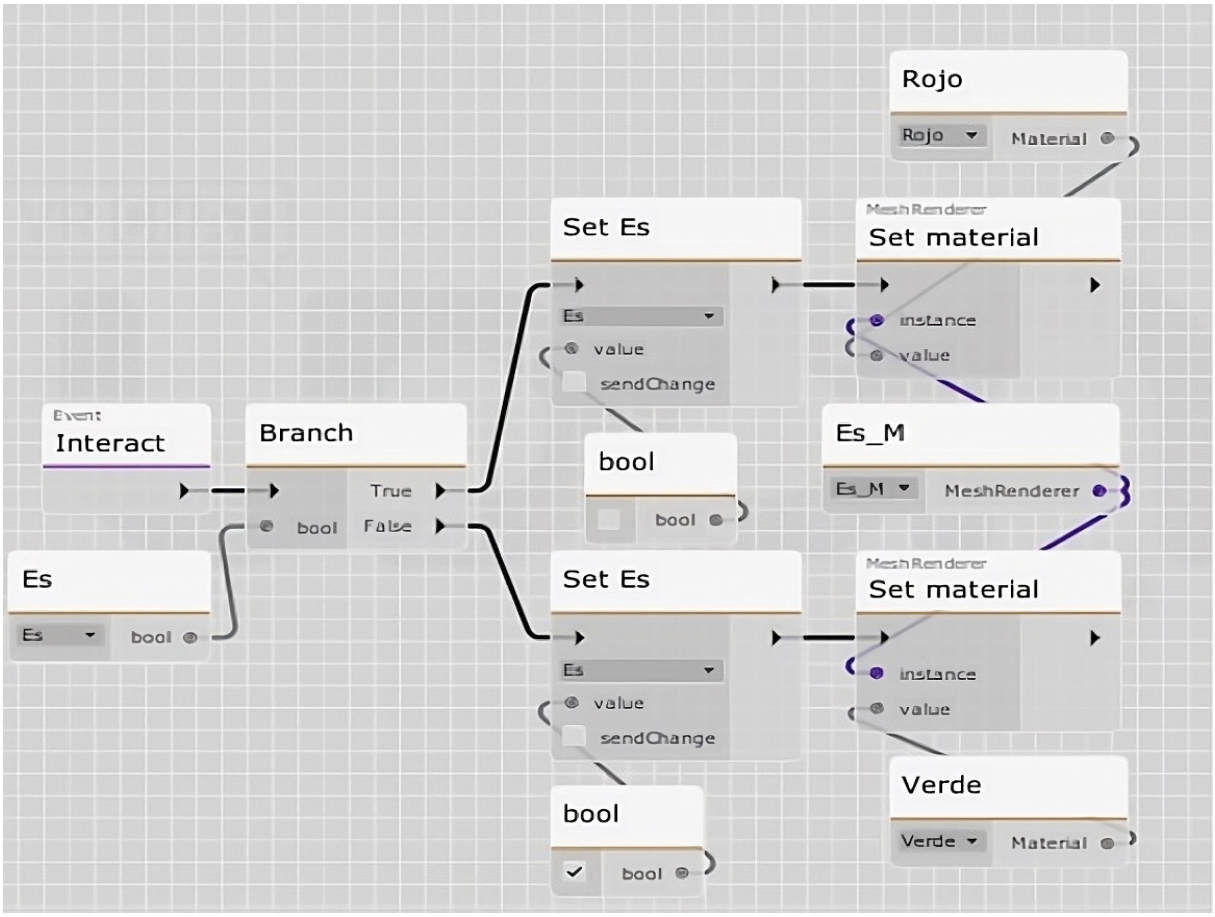


Figura 12 Código UdonGraph botón y señal.

Posteriormente, se implementó otro botón virtual con una nueva malla que opera la señal de control; se empleó la misma lógica que el primer botón al ser una señal que se enciende o apaga al ser oprimido por el usuario. Ahora que se tienen las dos señales, se necesita una condición tal que cuando las señales de control y alimentación sean encendidas, transcurra el tiempo 'T' y la señal de salida se vuelva true cambiando a verde automáticamente. Para que un evento comience después de 'T' en segundos, se usa el bloque llamado *Send Custom Event Delay Seconds*, que se encarga de realizar un evento despúes de un tiempo trasncurrido; para este código, al acabar el tiempo la salida se activará en el caso del temporizador positivo.

En la figura 13 se muestra un diagrama de flujo del funcionamiento de un temporizador positivo; se necesita de un operador del tipo AND que condiciona a la señal de salida y la de control: el operador requiere que las dos señales sean True para que la condición también sea True. Una vez con la condición

cumplida, transcurre el tiempo establecido y se enciende la alimentación. Si alguna de las señales de control o de alimentación son False, la salida será False.

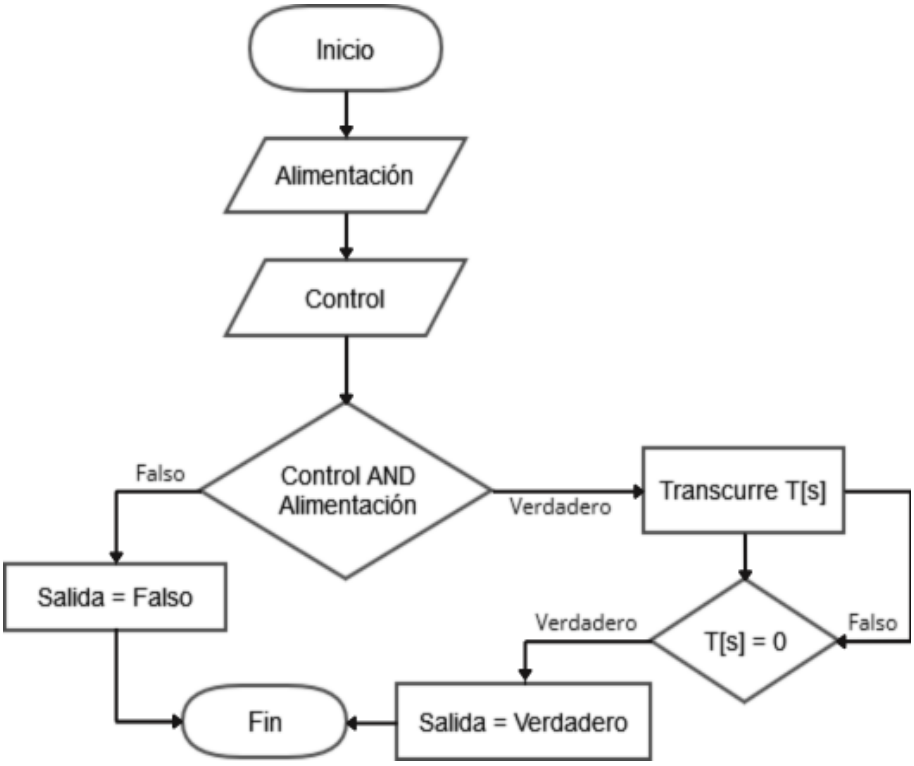


Figura 13 Diagrama de flujo para representar la lógica de señales.

Una vez que se tiene el funcionamiento básico de los temporizadores positivo y negativo se tuvo acceso a un modelo tridimensional del temporizador hecho por el colaborador Roberto Pineda para integrarlo a Unity como una malla y añadir las características del guión anterior. Es importante conocer la forma en que los modelos tridimensionales son hechos, para aprender a manipularlos y agregarle los componentes que se necesitan. Un modelo es construido en cualquier programa de modelado o animación que, por medio de múltiples figuras simples, son manipulados para darle forma a cada elemento hasta tener un resultado deseado.

En la figura 14 se muestra la ventana de jerarquía de la malla del temporizador negativo **TimerN**, donde se despliegan múltiples objetos que están ordenados. TimerN es el objeto 'padre', el cual tiene un ícono de flecha que despliega otros objetos como el dial, el cuerpo, cuerpo_1 y cylinder.004, que son llamados objetos 'hijos' y conforman el modelo completo del temporizador. En la figura también se pueden observar los objetos de entrada, control y salida, que también son hijos y, además, tienen otros objetos hijos del mismo nombre. La forma en que la jerarquía funciona en cualquier GameObject es cuando un objeto padre es modificado, sus elementos hijos heredan las nuevas características, mientras que si se modifica un elemento hijo, el elemento padre y sus hermanos no heredan los cambios.

Figura 14 Jerarquía.

Se añadió el componente UdonBehaviour con el guión hecho al GameObject padre del temporizador; al revisar el guión en el inspector, se pueden verificar las variables públicas que cuenta, las cuales se pueden modificar desde el editor; las variables tipo GameObject desde el inspector requieren su dirección de la escena desde la jerarquía. Entonces, las variables que representan las señales fueron los GameObjects hijos entrada, control y salida. Con esto se obtuvo un modelo interactivo en Unity que simula los temporizadores positivo y negativo.

Se requiere que el tiempo no sea una constante sino una variable que el usuario controle en tiempo real. Unity ofrece GameObjects del tipo UI; por las siglas en inglés Interfaz de Usuario y se elige el elemento 'Slider', un deslizador cuya posición de la perilla determina un valor numérico. Es necesario agregar un GameObject llamado *canvas* que crea un plano ubicado en la escena y uno puede agregar elementos UI como texto, botones, imágenes y deslizadores. El *canvas* se vuelve un elemento padre, mientras que los demás elementos son GameObjects hijos. La función del slider es que el usuario tome la perilla, cuya posición inicial al extremo izquierdo es cero y el extremo derecho es treinta. Cuando se suelta la perilla, el número permanece y es programado a la variable de espera T del temporizador. Como son dos temporizadores, se necesitan dos sliders con sus propios guiones que controlan sus respectivas variables.

En la figura 15 se muestra una sección del guión desarrollado, donde se añade la espera en segundos por el deslizador agregado. La función comienza en el bloque *Custom Event* con su nombre 'Timer', que conecta a otro bloque donde se declara la variable S_time, cuyo valor será tomado del guión del deslizador. Para extraer el valor del deslizador, se usa el bloque *Get Program Variable*, el cual necesita de dos argumentos, el primero es una variable del tipo UdonBehaviour que es el guión del deslizador y el segundo es una cadena llamada *Slider* que se refiere al nombre de la variable dentro del código que representa el valor de la perilla. Una vez con el valor de la variable del deslizador, se transforma con *Get Value* para que sea un valor numérico flotante que alimentará al bloque *Send Custom Event Delay Seconds*, el cual es el tiempo de retardo del evento, cuando acaba el tiempo, la malla que representa la salida se torna verde.

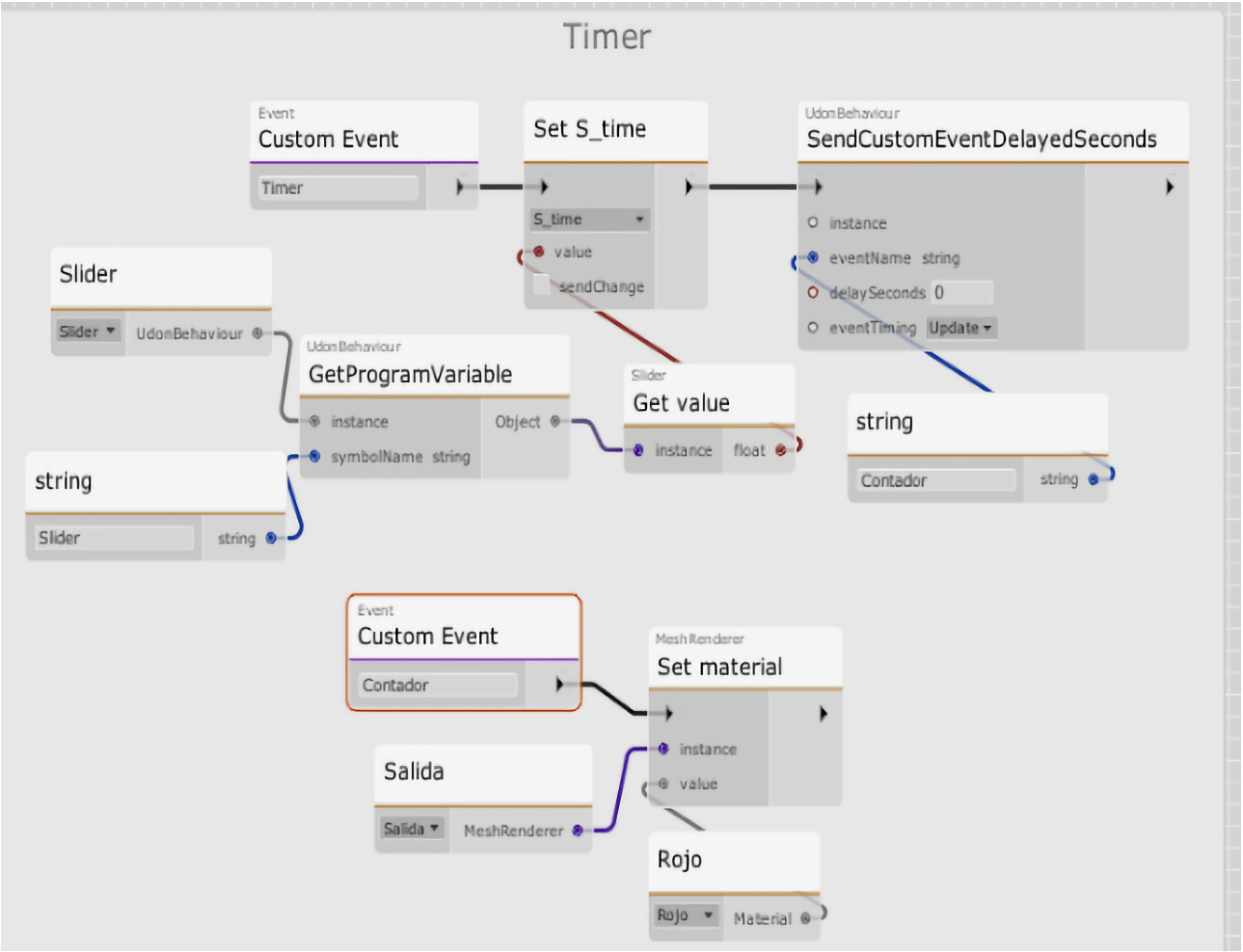


Figura 15 Funcion Timer en UdonGraph.

Ya con los temporizadores acabados, se recibió el entorno trabajado y la siguiente tarea consistió en exportar lo desarrollado a NERV. Se analizó su contenido en las carpetas y los códigos que se hicieron con UdonSharp, que es un compilador más tradicional, esto no permite exportar los códigos hechos en UdonGraph por lo que se tuvieron que estudiar los códigos para adaptar mis guiones a UdonSharp. El código que se encarga del sistema de las mesas de trabajo, los elementos neumáticos y su interacción es uno de nombre *Man_Valv*. En el código se encuentran declaradas múltiples listas que contienen variables booleanas, que se encargan de responder preguntas como: ¿está oprimido? para botones o válvulas. Para las conexiones neumáticas, se tiene la lista booleana con la pregunta ¿es el origen? y ¿es el objetivo? para conocer el ducto de origen y el de destino respectivamente; también hay una lista de GameObjects llamados *Line Renderer*, la cual es una variable que toma dos puntos en un espacio tridimensional y crea una línea recta de un color programado entre los puntos, que actúan como mangueras.

Así como fue descrito en la figura 8 del capítulo anterior, el usuario a través de los Line Renderer puede crear conexiones. Cuando termine de realizar el circuito neumático, se puede activar el paso de aire y posteriormente el FRL; entonces todos los elementos neumáticos están en un estado de condición con la pregunta ¿hay fugas de aire?, si no hay fugas de aire, todos de los elementos neumáticos actúan e interactúan entre sí, realizando la función del circuito.

Más a detalle, el código *Man_Valv* consiste en un panel de control, el cual inicializa las variables de los

elementos neumáticos de una mesa de trabajo, gestiona los estados de los ductos en tiempo real durante el diseño de un circuito neumático y, cuando se ponga a prueba el circuito, pueda simular las funciones de los elementos neumáticos que participan. Además, *Man_Valv* tiene otra lista de GameObjects de interés de nombre *Botones* y contiene todos los objetos que representan los ductos de cada elemento neumático de una mesa. En la figura 16 se puede observar la lista Botones que se encuentra en el componente UdonBehaviour del código.

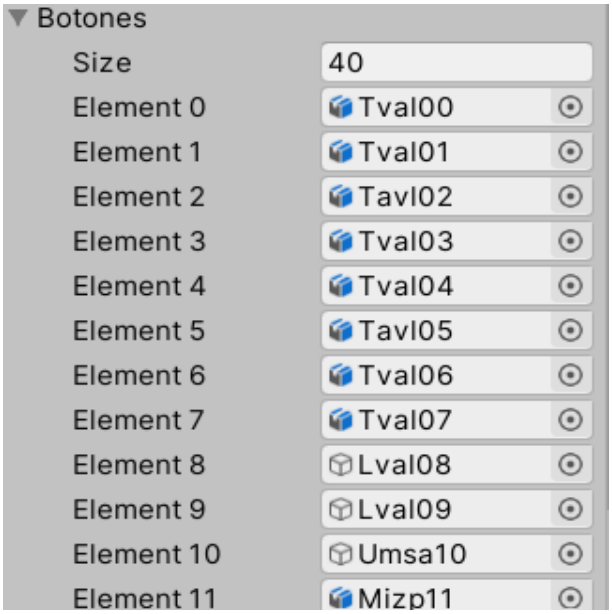


Figura 16 Lista de elementos del tipo GameObject en el componente UdonBehaviour desde el inspector.

Al revisar el componente UdonBehaviour de *Man_Valv* en el inspector, se pueden ver todas las variables públicas del código; de estos elementos se encuentran en la lista de botones y al darle clic a algún GameObject de esa lista, Unity indica su dirección en la jerarquía. En la figura 17 se muestra resaltado desde la jerarquía el *Tval03* que es un ducto el cual pertenece a la lista Botones; se observa que el objeto es hijo de un GameObject llamado *Conector0* que también es hijo del objeto padre de nombre T2 que representa una conexión tipo 'T'.

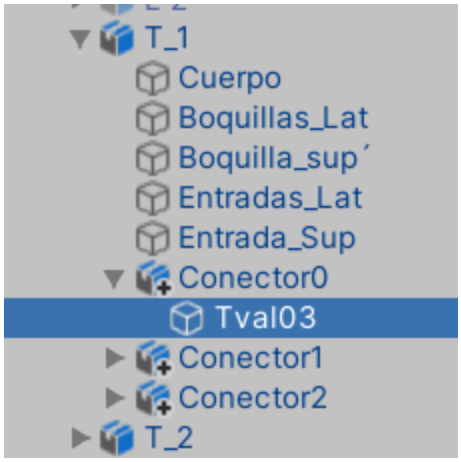


Figura 17 Ubicación de *Tval03* en la jerarquía que es hijo del objeto Conector0.

Todos los ductos de la lista botones son GameObjects hijos que necesitan compartir ciertas características para poder ser parte de la lista. El ducto *Tval03* posee un componente Mesh Filter con un material llamado *Body1*; este material está presente en los ductos de todos los elementos neumáticos, también tiene un componente UdonBehaviour con un guión de nombre *btnconexion*, que se relaciona con el panel *Man_Valv*.

El panel se encarga de darle a los ductos un comportamiento por medio de un código de colores, cuando un usuario le da clic al elemento se vuelve amarillo indicando el origen de la manguera, y una vez que se le dé clic a otro ducto, se forma una conexión; los ductos son color cian y la manguera es del mismo color. Cuando se activa el paso de aire y el FRL, las mangueras y ductos toman un color azul indicando que hay paso de aire, y si el circuito tiene fugas el ducto es de color rojo seguido de un sonido de aire escapando. De esta forma, el panel detecta los ductos activos y los declara como argumentos para las funciones de los elementos neumáticos que trabajan en conjunto para realizar la función del circuito. En la figura 18 se observan los componentes que tiene el GameObject Tval03 y que comparten los demás objetos dentro de la lista botones.

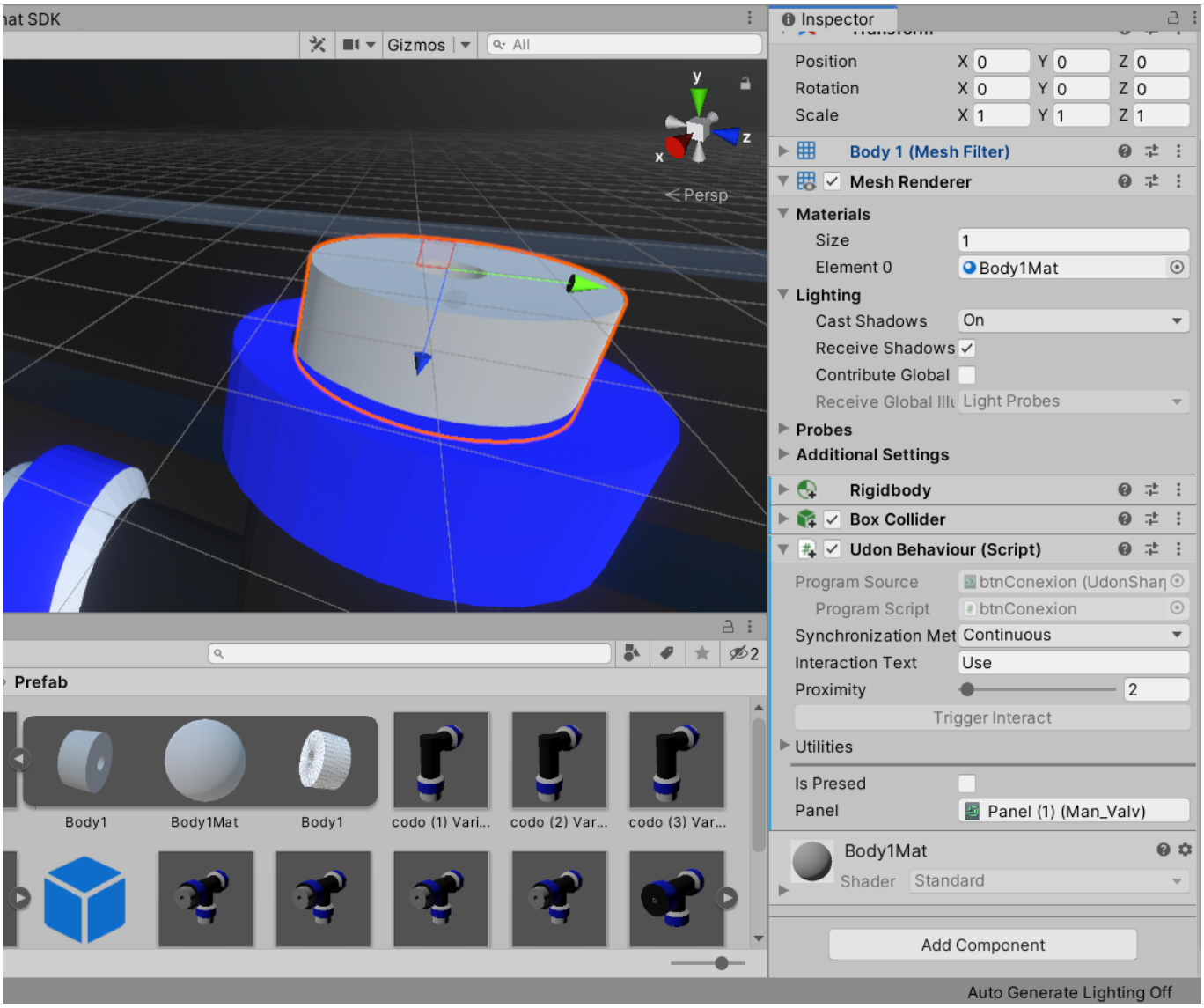


Figura 18 Material *Body1Mat* en *Tval03*.

Para comenzar a integrar los temporizadores, se modificó el código de *Man_Valv* para aumentar la longitud de la lista botones para seis nuevos ductos de los dos temporizadores. Se crearon nuevas mallas que forman parte de la jerarquía de los temporizadores como hijos de los componentes entrada, control y salida. Estos nuevos objetos tienen los componentes que se vieron en el ducto Tval03; primero se agregó el componente *Mesh Renderer* con el material *Body1*.

Se analizó el guión *btnconexion* del componente UdonBehaviour de los ductos; el código realiza la operación de registrar los ductos a *Man_Valv* y de sincronizarlos. Primero se declaran dos variables, una booleana llamada isPressed y el guión de *Man_Valv* como una variable del tipo UdonBehaviour llamada 'Panel'; cuando un ducto es presionado se activa *Send Custom Network Event*, cuya funcionalidad se verá más a detalle en la sección 3.2; posteriormente, activa la función Toggle Target que inicia con el resultado negado de isPressed y se declara una nueva variable del tipo entero llamada 'num'. La variable 'num' está compuesta de dos dígitos que son extraídos del nombre del ducto presionado, se salta cuatro caracteres tomando el quinto y sexto que son números únicos para cada elemento. Finalmente se llama a la variable 'Panel', la cual registra el ducto activado al código con su número correspondiente y su nuevo valor de isPressed. En la figura 19 se muestra un diagrama de flujo para complementar visualmente lo explicado anteriormente.

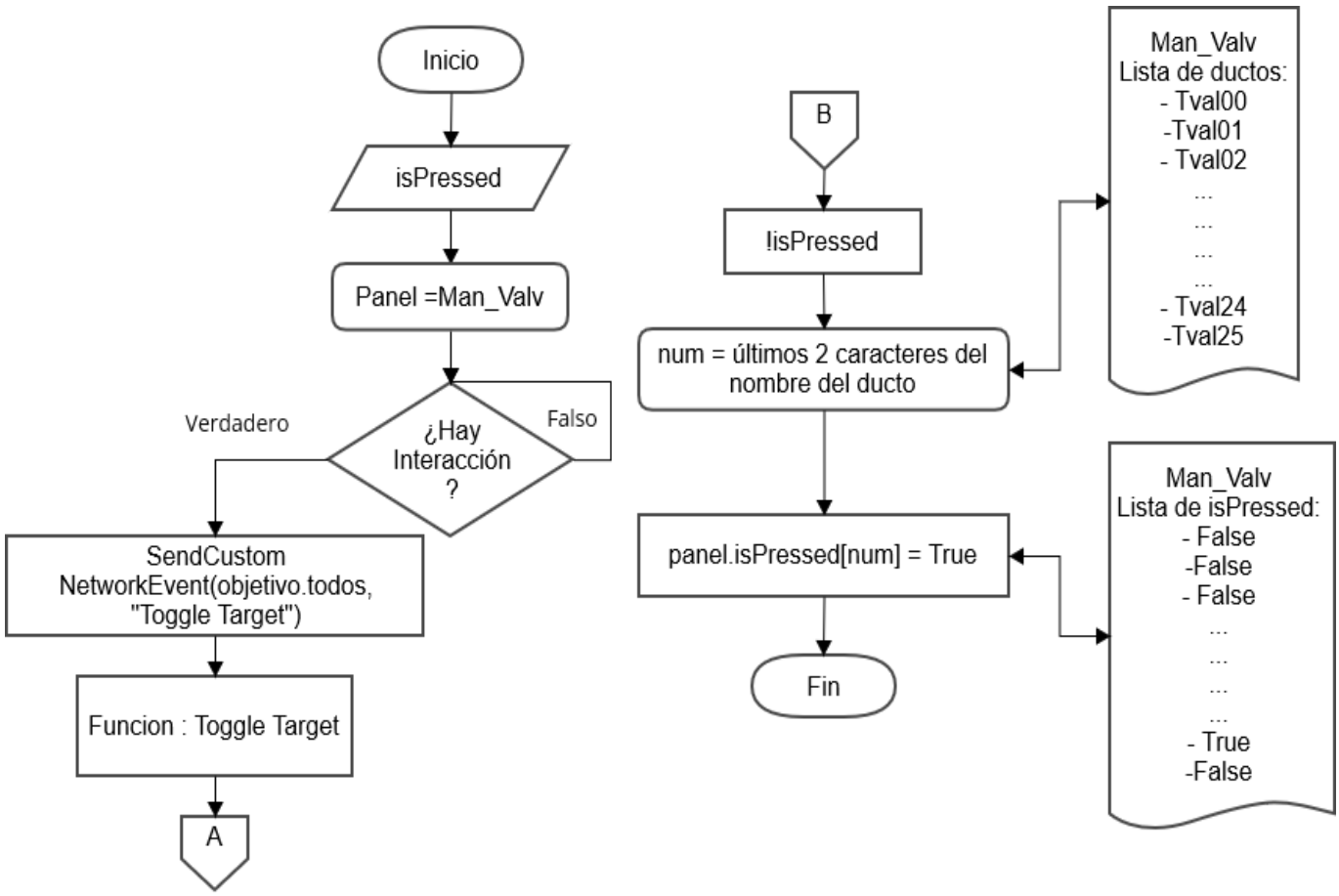


Figura 19 Diagrama del código de *btnConexión*.

Tomando en cuenta la lógica dada por el código *btnConexión*, los nuevos seis ductos se nombraron de la siguiente forma: Eval##, Cval## y Sval##. La numeración fue siguiendo el orden de los ductos anteriormente disponibles.

Hasta el momento ya se tienen registrados los ductos de ambos temporizadores en el panel *Man_Valv* y en *btnConexion*. Como se había mencionado, una vez que el FRL esté encendido, las funciones de los elementos del circuito diseñado se activan e interactúan entre sí para realizar un trabajo. Las funciones que controlan los elementos neumáticos en el panel cuentan con múltiples argumentos. Los argumentos son variables de entrada que pueden ser de distintos tipos como enteros, booleanos, GameObject o de animación que realizan una tarea específica.

Todas las funciones en *Man_Valv* necesitan como argumento la lista *botones* donde se obtienen los ductos requeridos del elemento neumático. Para indicar los ductos correspondientes, se agregan como argumentos unas variables del tipo entero, tomando el número registrado por la variable num de *btnconexion*; en el caso del temporizador positivo, los números de los tres ductos son 37 (entrada), 38 (control) y 39 (salida). Entonces, lo que se hace en la función es mandar a llamar la lista botones y se indica el ducto ingresando el número registrado en el índice de la lista, así se toma el GameObject necesario para luego ser manipulado.

Otro argumento general es una lista de variables booleanas de nombre *isActive*, el cual indica lógicamente cuándo un ducto presenta un flujo de aire en tiempo real. La lista *isActive* utiliza de igual manera el número registrado de *btnconexion* para obtener el ducto de la lista botones. Finalmente, se tiene la lista de enteros *Num* que forma parte de un índice para la lista de mangueras *LineRenderer* y también forma parte de otra función presente llamada *ConQuien*, la cual indica el elemento neumático a ser conectado después de haber seleccionado el origen; de esta forma se realizan las conexiones y se comunican los elementos neumáticos entre sí.

En las funciones de algunos elementos neumáticos se tienen unas variables llamadas *fuga#*, las cuales son variables booleanas para elementos neumáticos con más de dos ductos, que indican si algún ducto no es conectado con otro elemento al ser activado el circuito. Estas variables pertenecen a la válvula monoestable *fuga* y la válvula biestable *fuga2*. Los temporizadores también necesitan de las variables booleanas de fuga; se crearon *fuga3* para el temporizador negativo y *fuga4* para el temporizador positivo.

Se rehicieron en UdonSharp los programas que representan el funcionamiento de los deslizadores, uno es para la lógica negativa y el otro de lógica positiva; estos códigos son llamados en *Man_Valv* al ser declarados como variables del tipo *UdonBehaviour*, para obtener el valor del deslizador y usarlo en la función de los temporizadores con el método *Get Program Variable*.

En la figura 20 se muestra un diagrama de flujo que representa la función creada para *Man_Valv* del temporizador negativo. En la figura se toman en cuenta las variables de los GameObjects, así como su código de colores para crear condiciones.

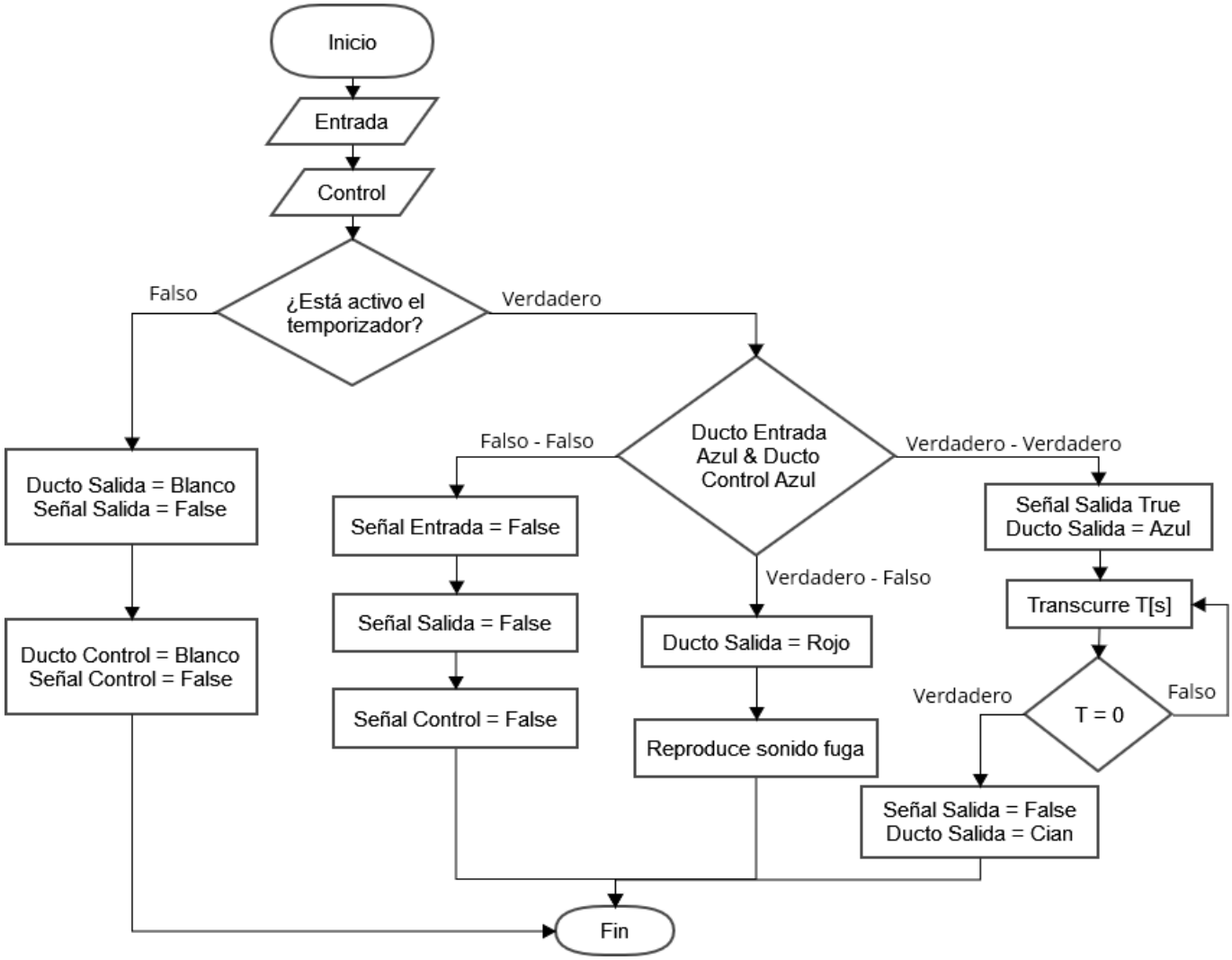


Figura 20 Diagrama de flujo del Temporizador Negativo.

3.2 Creación de redes de contactos

Uno de los objetivos del entorno es la interactividad entre usuarios durante sus prácticas; ocupando el kit de desarrollo (SDK) de VRchat se puede conectar el laboratorio al servidor para que un usuario pueda ingresar, conectarse y realizar las actividades en tiempo real con otros usuarios conectados. El SDK proporciona varios métodos para otorgarle interconectividad a las funciones que por defecto se les llaman objetos locales; este concepto proviene del *Ownership* o pertenencia. Cuando una variable u objeto tiene su pertenencia como local, significa que cualquier cambio que realice un usuario en su sesión a un GameObject sólo lo podrá ver el usuario que está haciendo la acción; en la sesión de cualquier otro usuario no se podrán ver los cambios, ya que no están sincronizados los objetos.

En el código *Man_Valv*, la mayoría de las variables declaradas usan un método llamado *UdonSynced*, que permite que las variables dentro del código se encuentren sincronizadas en todas las instancias del entorno. Además de las variables tipo booleano, gameObject y UdonBehaviour, tambien se encuentran sincronizadas variables del tipo animación que afectan a algunos elementos neumáticos como los pistones, la manecilla del FRL y el piloto, que se habla con más detalle en la sección 3.4.

Por último, se tiene en el proyecto múltiples códigos en UdonSharp que se encargan de darle interactividad e interconectividad con otros elementos como los botones, llave del FRL, llave de la toma de aire, los fines de carrera y los ductos. Las llaves y botones requieren de una interacción directa por los usuarios, mientras que el fin de carrera se activa cuando el vástago de un pistón doble toca su accionador. Entonces, en los códigos de estos elementos neumáticos se declara una variable booleana y a través de una condición cambiará dicha variable; ante cada transición de estado de la variable se produce una animación. *Man_Valv* se encarga de registrar el estado de las variables, y que se encuentren sincronizadas para todas las sesiones conectadas; para esto es necesario un método llamado *Send Custom Network Event* el cual se encarga de sincronizar los eventos de los objetos interactivos. En la figura 21 se observan dos sesiones abiertas del entorno para probar la interconectividad de un deslizador para los temporizadores.

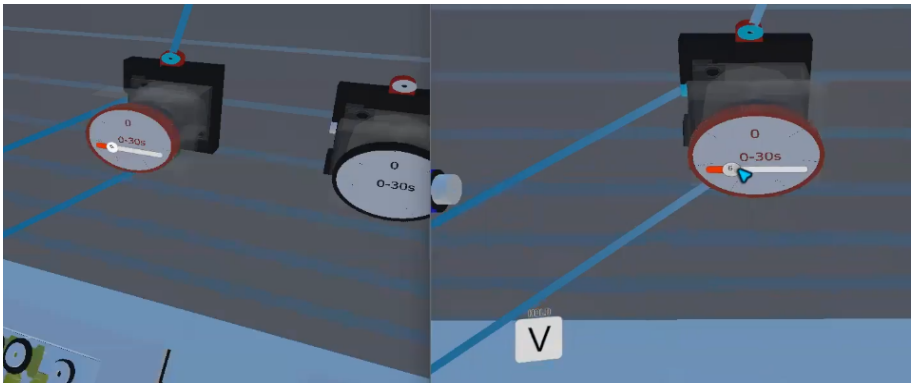


Figura 21 Prueba de sincronización de deslizadores.

Send Custom Network Event necesita de dos argumentos, primero se establece que todos los usuarios en el entorno estén sincronizados al evento y el segundo argumento se refiere al nombre de la función o evento. En el apartado anterior se trató el código de los ductos *btnConexión*, que usa el método mencionado para que la interacción con los ductos esté sincronizada con los usuarios conectados y envíe información en tiempo real al panel.

Con esto se puede corroborar que los ductos de los temporizadores están sincronizados correctamente para crear conexiones, sin embargo, hace falta sincronizar los deslizadores de ambos temporizadores para que las mesas de trabajo y sus elementos sean completamente funcionales. Los códigos que le dan la funcionalidad a los deslizadores se modificaron, de manera que cuando un alumno mueva la perilla del deslizador, éste se inhabilite para los demás alumnos hasta que sea soltada, y sólo un usuario a la vez pueda interactuar con ella.

En la figura 22 se muestra un diagrama que describe el funcionamiento y sincronización de los deslizadores; primero se modifica la pertenencia de un GameObject cuando es activado usando el método de SetOwner; fijar propietario, con sus dos argumentos: la relación con el usuario, el cual se especifica como Localplayer para que el deslizador sólo sea movido por el usuario actual, y el otro argumento se refiere al gameObject que se desea que tenga este método que es el de la perilla del deslizador.

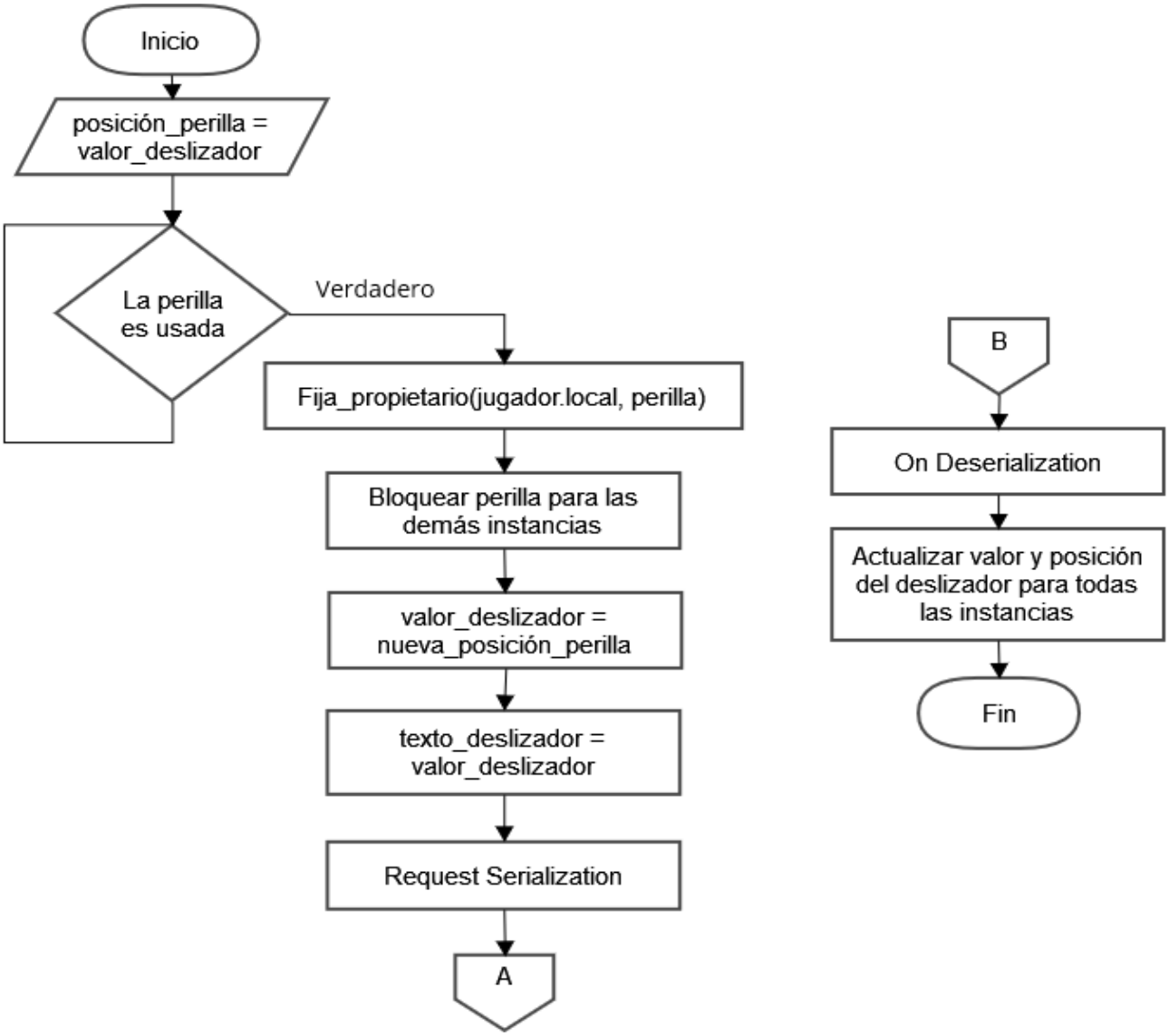


Figura 22 Diagrama para la sincronización del slider.

Lo que se tiene que sincronizar es la variable numérica que representa el tiempo en segundos usando UdonSynced al declarar la variable flotante y también se sincroniza la posición de la perilla, así como el texto que indica su valor actual. Se ocupa la función *Request Serialization* la cual sincroniza la perilla y su valor numérico cada vez que se manipula, ya que no es necesario enviar información en cada fotograma [12]. Al usar la función anterior, es necesario complementarlo con otra función llamada *OnDeserialization*, que es el estado cuando deja de ser sincronizado. Entonces, cuando se suelta la perilla, el estado actual del deslizador es actualizado con su valor numérico y posición. La figura 23 muestra los deslizadores que permiten modificar los elementos sincronizados, los cuales son el texto que indica el tiempo en segundos y la perilla.

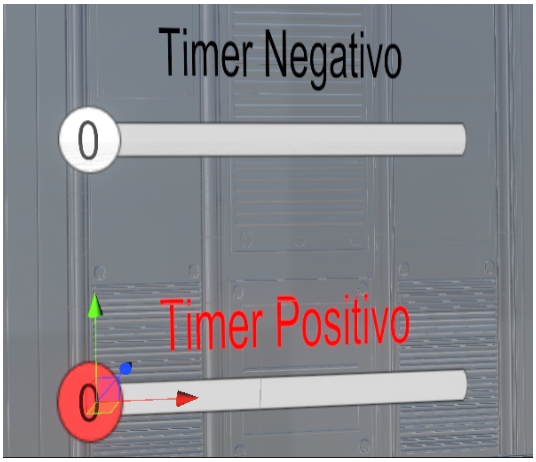


Figura 23 Deslizador.

3.3 Modificaciones al mundo virtual

3.3.1 Pruebas del entorno

Una vez con el temporizador neumático integrado, se hicieron múltiples pruebas para verificar su correcto funcionamiento. Se detectó un error que se encontró al realizar ciertas conexiones con los fines de carrera y los botones. El circuito que presentó el error consistió en un botón pulsador que a través de una válvula biestable activa un pistón de doble efecto e interactúa con los finales de carrera; al avanzar el vástago, toca el accionador del fin de carrera para mandar una señal de vuelta a la válvula biestable y hace retroceder el vástago realizando contacto con el otro fin de carrera, el cual nuevamente envía una señal para que el vástago vuelva a avanzar creando un ciclo.

En la figura 24 se observa el circuito descrito en ejecución, aunque el pistón está en avance y el fin de carrera correspondiente es accionado, las señales no realizan ningún cambio, dejando el circuito bloqueado y se tiene como única opción apagar el FRL para reiniciar el circuito.

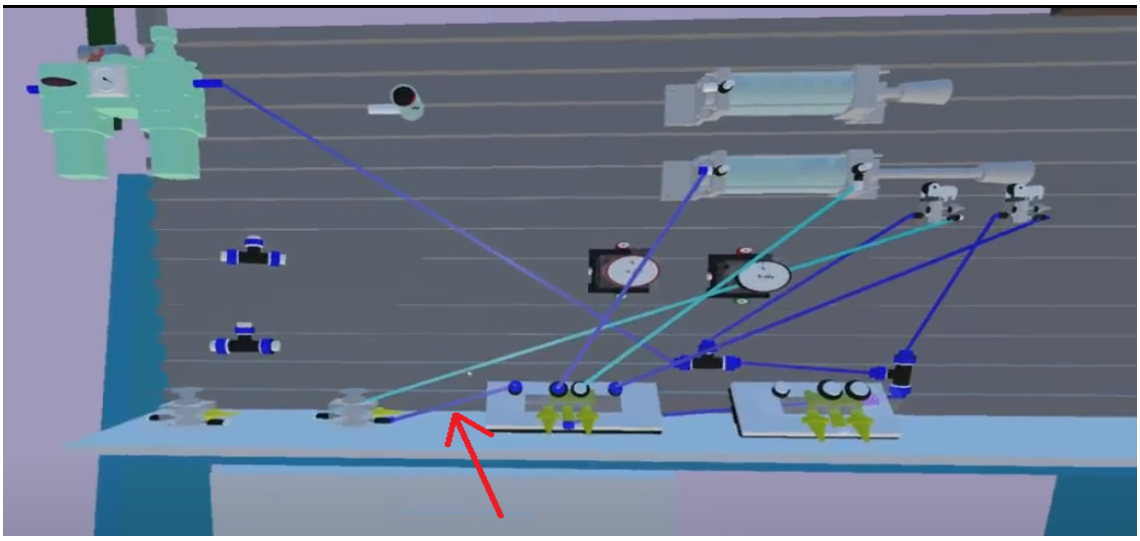


Figura 24 Circuito con error.

Al analizar más a detalle la figura, se observa que en el primer fin de carrera su salida está desactivada, sin paso de aire,ya que el vástago no está haciendo contacto con éste; además, su salida está conectada a un botón cuya salida es señalada con una flecha roja y se encuentra activada, hay paso de aire, a pesar de no recibir aire del fin de carrera provocando un error lógico. Dado que el botón fue el primer elemento en mostrar inconsistencias en su funcionamiento, se revisó su función dentro del código *Man_Valv*.

En la figura 25 se ve un diagrama de la función del botón dentro del panel. La lista de variables booleanas isActive se usan para saber el estado de cualquier ducto; se indica el ducto en específico con el número registrado por el código *btnConexion* como fue mostrado en el apartado anterior. Una vez con los ductos de salida y entrada del botón registrados, la función inicia con una condición que pregunta si el ducto de entrada es azul, lo cual quiere decir que hay paso de aire, indicando que el botón puede ser usado; después espera poruna interacción del usuario. Cuando el botón es presionado, la variable isActive del ducto de alimentación toma el valor verdadero y pasa por una última condición que pregunta si el ducto de salida está conectado a algún otro elemento; en caso de estarlo, cambia su valor isActive como verdadero y el color de la salida se vuelve azul; si no está conectada, la salida se vuelve roja y se reproduce el sonido de fuga. Por último, se tiene la condición cuando el botón es desactivado, la variable de isActive para el ducto de salida se vuelve falso y cambia el color del ducto a blanco indicando ausencia de aire.

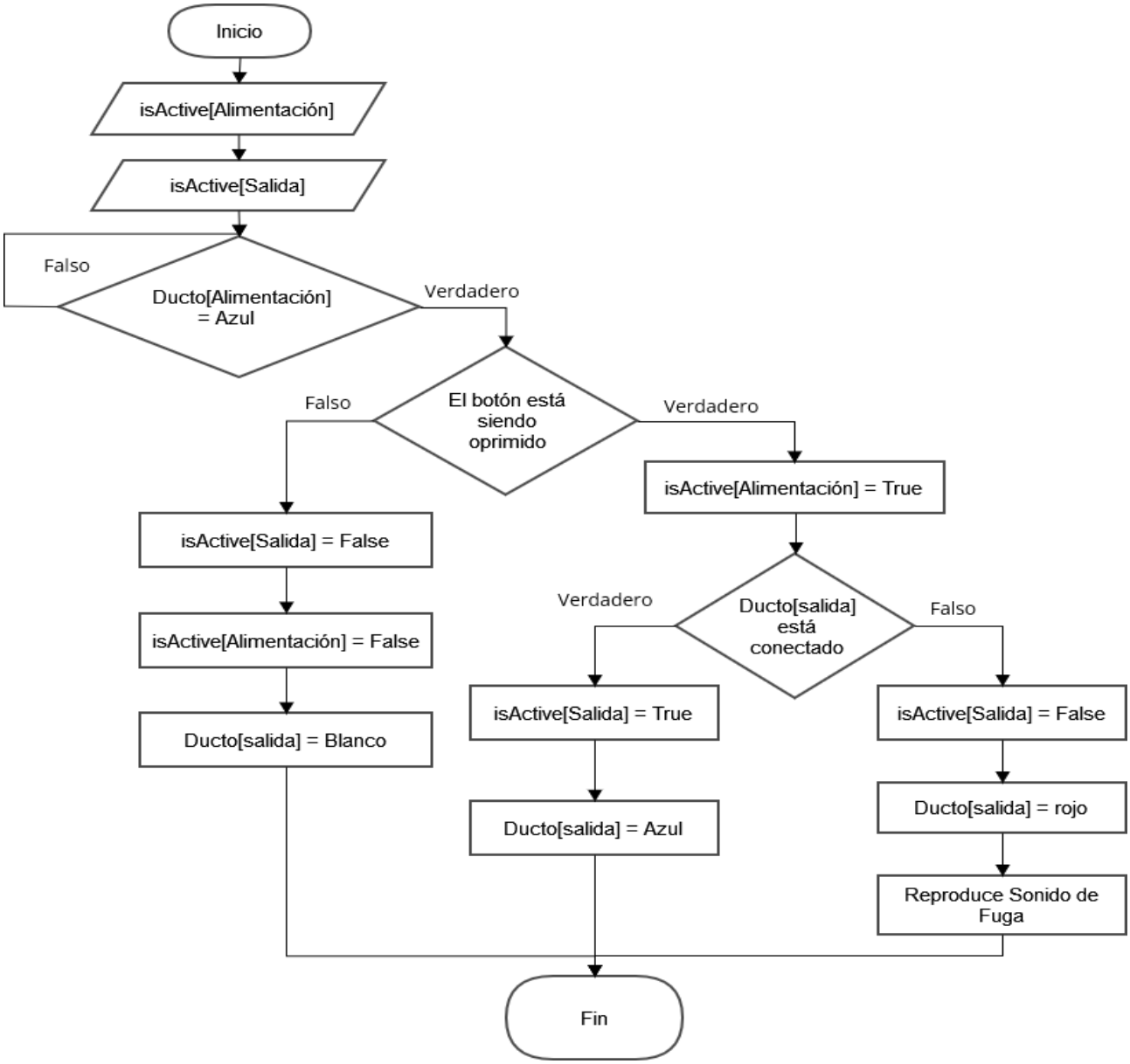


Figura 25 Diagrama de flujo de la lógica del botón (antes).

Entonces, se modifcó la función del botón; del diagrama anterior se puede ver que el botón sólo puede cambiar de estado cuando un usuario interactúa con él. Por eso, aunque la alimentación se encuentre desactivada por no haber flujo de aire debido a otro elemento como el final de carrera, la salida del botón sigue suministrando aire. Además, al intentar desactivar el botón de forma manual, no responde ya que la alimentación se encuentra en un estado que no cumple con ninguna condición del anterior diagrama, dejando al elemento bloqueado.

En la figura 26 se observa el diagrama de la función actualizada; se expandió la primera condición cuando el ducto de alimentación no se esté alimentando, que se indica con el color blanco. Se agregó otra condición preguntando si la variable isActive del ducto de alimentación estaba activa anteriormente; si cumple la condición los valores isActive de ambos ductos del botón se volverán falsos y el ducto de la salida será blanco, indicando falta de flujo del aire, de esta forma se soluciona el error.

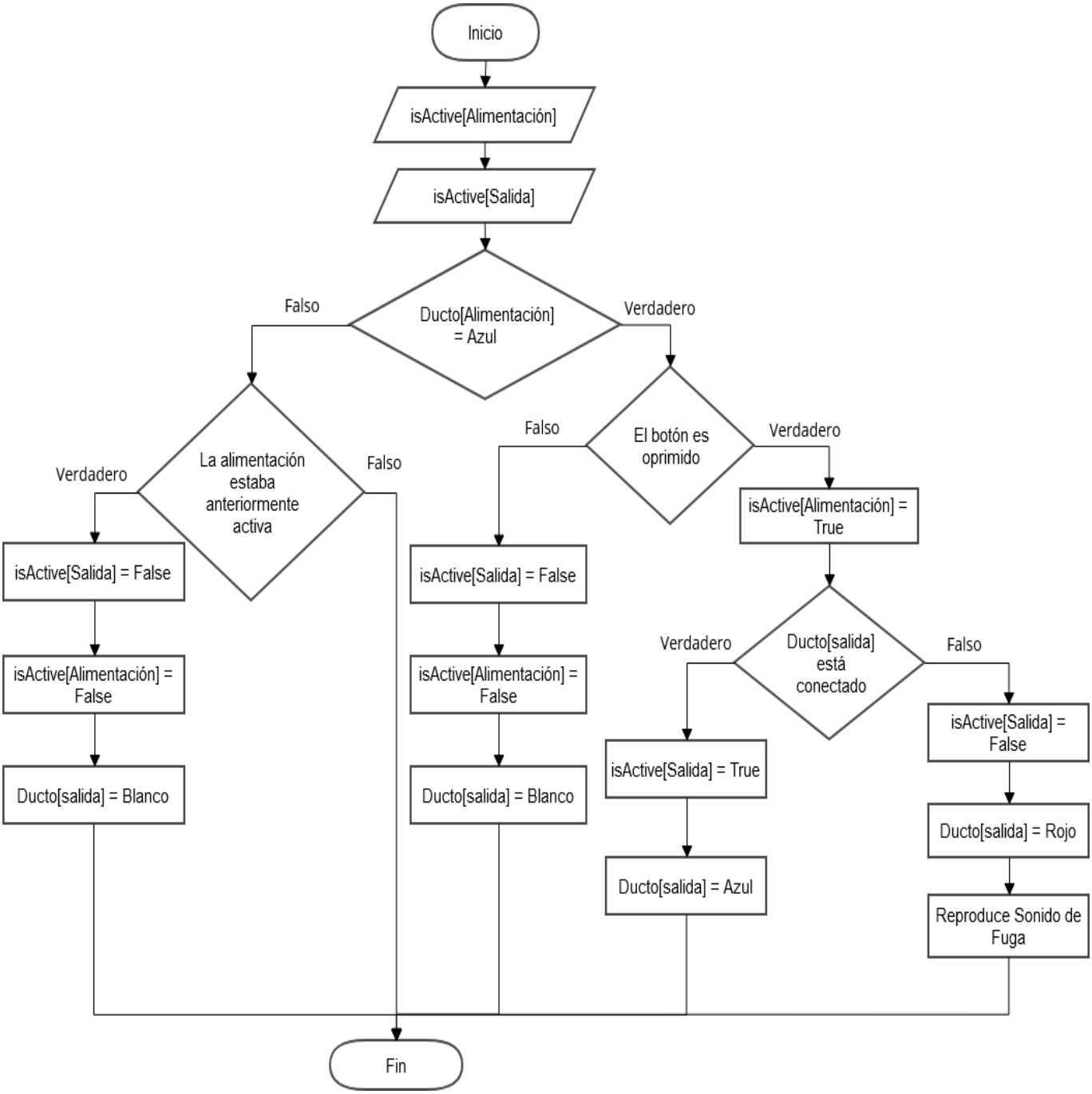


Figura 26 Diagrama de flujo de la función actualizada.

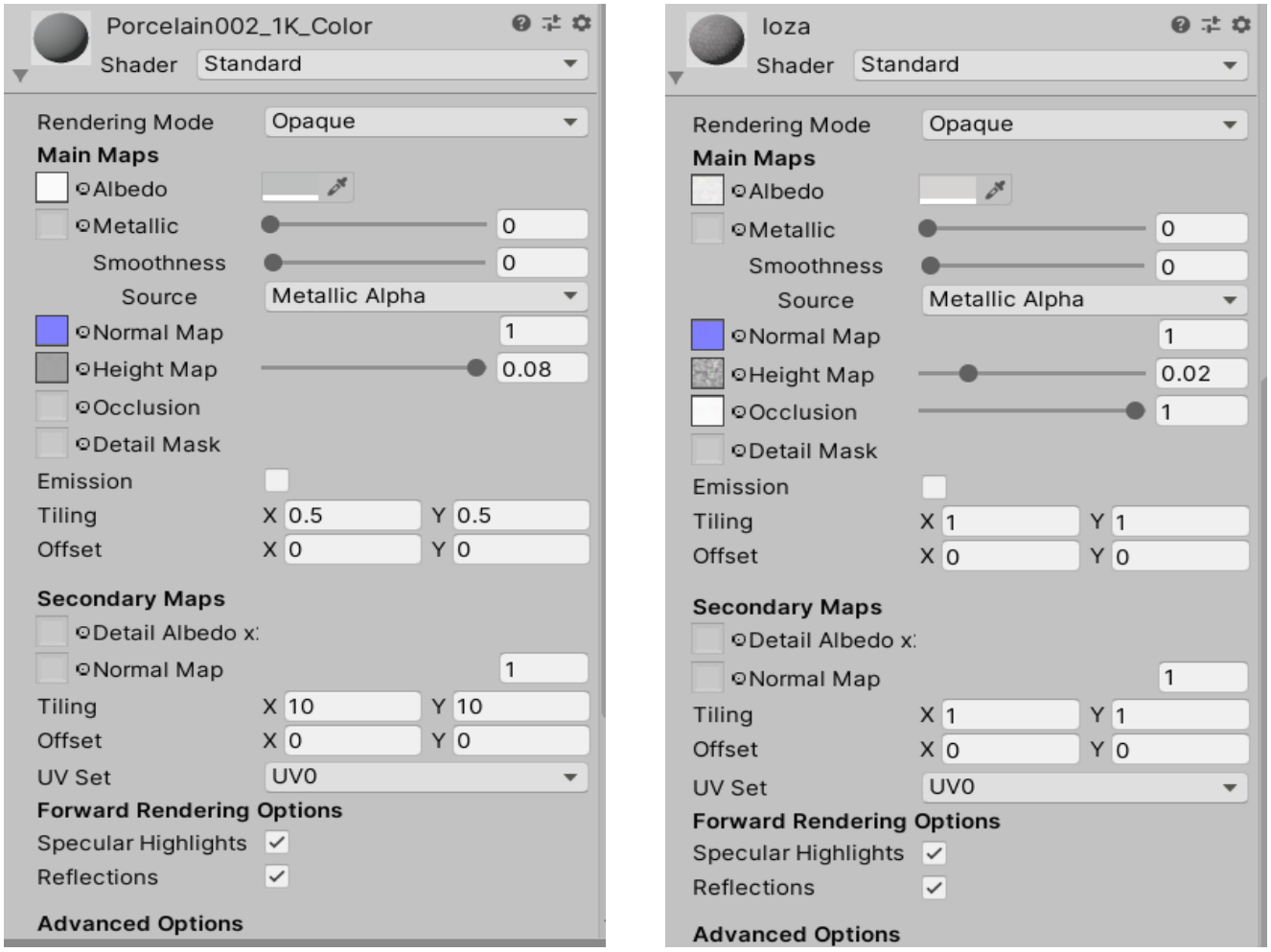
3.3.2 Actualización gráfica del entorno

Como siguiente paso en la mejora del proyecto, se propuso cambiar el entorno por uno más llamativo, con una arquitectura más compleja, amplia y moderna. Con ayuda del compañero Mauricio Villalba González, quien usó el software de modelaje tridimensional Blender, se creó el nuevo laboratorio que contiene múltiples características adicionales como texturas y modelos que complementan el aspecto visual del laboratorio virtual. Se exportó el modelo tridimensional a *Unity* y se ubicó en la carpeta del proyecto para después ubicarlo en la escena del proyecto.

El entorno entregado consiste en un GameObject seccionado en múltiples elementos hijos; dentro de estos elementos se encuentran objetos que brindan iluminación a los cuartos presentes; además, se tienen GameObjects como un área verde, árboles, una pared de piedra y engranes que decoran los alrededores del laboratorio, mientras que los demás elementos representan distintos componentes que conforman el edificio, como pisos, escaleras, paredes, techos y ventanas.

Después, se exploraron las texturas que cuenta el archivo del entorno; a cada GameObject que representa una sección del entorno como los pisos, paredes, ventanas, áreas verdes, entre otros, le corresponde una textura. Por ejemplo, para el piso se usó una imagen que contiene un patrón de figuras como si fuera losa, la imagen se exporta al inspector de Unity como un componente tipo material, desde el inspector se puede configurar y añadir múltiples características como los archivos del tipo Maps o Texture Maps, los cuales son imágenes que guardan información y proporcionan instrucciones a la computadora para renderizar la superficie de un modelo; a su vez se puede modificar la forma en la que el modelo interactúa con la luz, puede verse más brillante, opaco, metálico e incluso transparente.

Se observa en la figura 27 dos componentes en el inspector del tipo material del nuevo entorno, que son implementados con dos pisos distintos; para tener materiales simples se necesita de un *Albedo Map* que se encarga de mostrar un color plano o una imagen que en este caso son patrones simples. Para la profundidad del material, se agrega un *Normal Map*; lo que hace es que cuando la luz interactúa con la superficie, hace parecer que tiene detalles que realmente no tiene el modelo como hendiduras, relieve disparejo o rasguños; esto ayuda a optimizar el rendimiento gráfico ya que si se hiciera un modelo tomando en cuenta los detalles, sería uno muy pesado. Finalmente, se usó un *Height Map* que actúa de forma similar al anterior, sólo que esta vez sí afecta directamente a la geometría del modelo [13].



(a) (b)

Figura 27 Texturas de nuevo entorno;a) Material para la pared; b) Material para el piso del "Lobby".

Se integraron las texturas correspondientes a cada elemento del entorno y se agregaron componentes para activar las colisiones de la geometría dada por el modelo, para que el usuario pueda caminar por el entorno sin atravesar pisos o paredes. Se colocaron los modelos de los elementos neumáticos en el lobby con una descripción de su funcionamiento y una representación visual simplificada, para que los usuarios identifiquen los elementos a usar antes de ingresar al cuarto con las mesas de trabajo.

Para migrar las mesas de trabajo con todos sus elementos neumáticos y sus funciones al nuevo entorno, dado que cada mesa de trabajo en la jerarquía es un objeto padre y sus componentes neumáticos son los hijos, sólo se tuvo que posicionar de forma ordenada cada mesa en la nueva sala del laboratorio. Al colocar las mesas en la escena del proyecto se detectó un error; cuando se rotaba el modelo, las dimensiones del elemento y sus hijos cambiaban de forma no deseada; se solucionó modificando manualmente los modelos afectados para que tengan su escala apropiada. Finalmente, la red de anillo y las tuberías de cuello de cisne se modificaron para adecuarse a las nuevas dimensiones del cuarto. En la figura 28 se muestra el nuevo entorno, el cuarto blanco es el lobby donde se encuentra en medio *la Leonardita* y residen los modelos de los elementos neumáticos, y el cuarto más grande es el laboratorio con las mesas de trabajo y donde los usuarios realizan sus actividades.

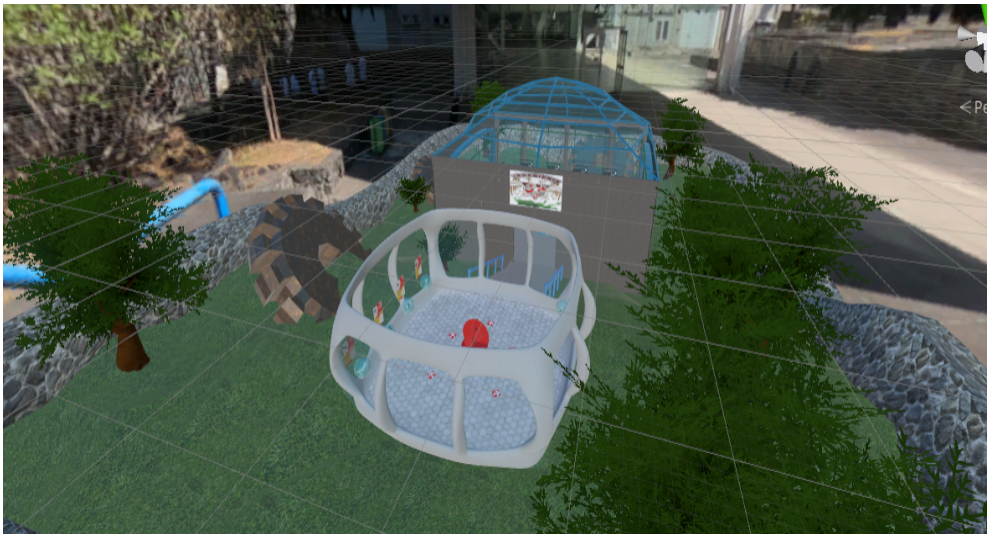


Figura 28 Nuevo entorno.

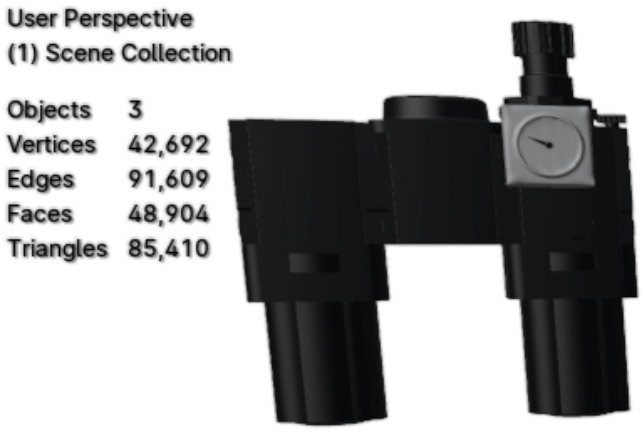
3.4 Modelado de nuevos componentes neumáticos virtuales

Con los cambios hechos en el entorno, se realizó una prueba presencial en el área de proyectos de ingeniería avanzada en el edificio de la DIMEI, con el uso de visores de realidad virtual se probó el entorno virtual. Tres integrantes ingresaron a los servidores VRChat y recibieron una invitación para entrar al laboratorio virtual; al acceder al lobby se observó que había un bajo rendimiento de fotogramas por segundo, con disminuciones de hasta cinco fotogramas y un máximo de veinticinco. La tasa de fotogramas es la frecuencia con la que pueden aparecer una serie consecutiva de ellas o imágenes en un panel de visualización. Esta frecuencia generalmente se mide en cuadros por segundo o fotogramas por segundo (fps); cuando se visualiza a 30 fps significa que aparecen treinta imágenes distintas en sucesión en un solo segundo. Si los fps son demasiado bajos, menor a treinta, el movimiento parecerá irregular y entrecortado. Pero también se puede tener problemas si los fps son demasiado altos debido a errores de sincronización, lo que podría sobrecargar el monitor y provocar un mal funcionamiento.

Un bajo rendimiento al correr una escena de un juego o entorno virtual se le puede atribuir a muchos factores, desde un código mal optimizado, múltiples instancias corriendo al mismo tiempo o la renderización de los gráficos. Se teorizó que los modelos causaban el bajo rendimiento, el compañero Mauricio Villalba González se dedicó a analizarlos, posteriormente se se percató que los elementos actuales tenían una cantidad de polígonos muy alta, por lo que cuando se ejecutaba la escena intentaba renderizar todos sus elementos; mientras más polígonos tenga que renderizar la computadora usará más potencia para cumplir esta tarea por lo que se hace más lento.

En los gráficos 3D por computadora, el modelado poligonal es un enfoque empleado para dibujar objetos representando o aproximando sus superficies usando mallas poligonales. Este proceso se realiza hasta obtener el modelo deseado; para crear modelos más complejos consecuentemente se necesitan más polígonos, sin embargo, es importante saber optimizar las figuras dependiendo de su función o nivel de detalle deseado. Como se puede notar, el modelo del FRL mostrado en la figura 29a contiene en su modelo un total de 42,692 vértices, 91,609 ejes, 48,904 caras y 85,410 triángulos, mientras que el pistón ilustrado en la figura 29b, tiene 14,312 vértices, 40,707 ejes, 26,378 caras y 28,426 triángulos. Nueve son los elementos que presentan una cantidad de polígonos anormalmente alta, que son el botón, final de

carrera, FRL, pistón de simple y doble efecto, conexión T, el temporizador neumático y las válvulas monoestable y biestable. Elementos de geometría similar pueden ser construidos con menos de 7,000 polígonos, tomando en cuenta la optimización, así como un énfasis menor en el realismo y detalle.

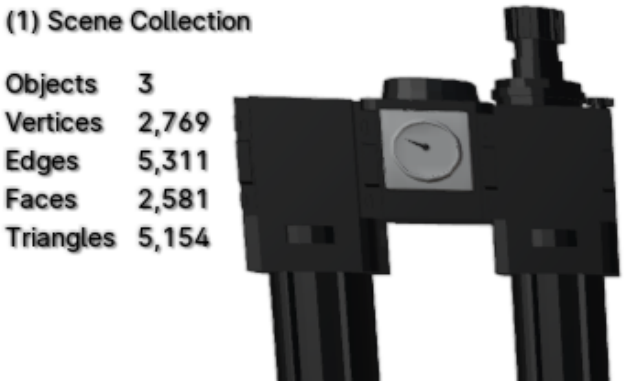


(a)

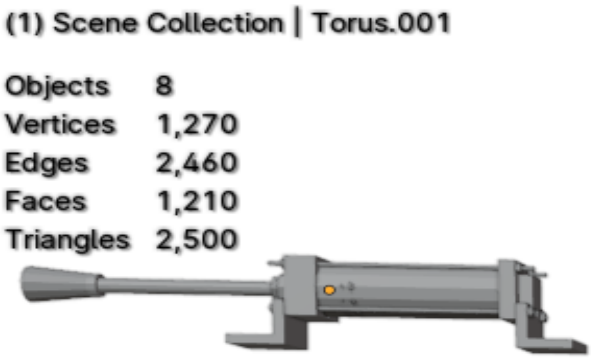
(b)

Figura 29 Análisis en Blender de los modelos anteriores del FRL y pistón.

Se entregaron los nuevos modelos y se analizaron para comparar la cantidad de polígonos con los anteriores; se verificó que el nuevo modelo del FRL cuenta ahora con 2,769 vértices, 5,311 ejes, 2,581 caras y 5,154 triángulos, como se muestra en la figura 30a, mientras que el piston tiene 1,270 vértices, 2,460 ejes, 1,210 caras y 2,500 triángulos, ver figura 30b. Las reducciones para cada aspecto de los modelos fueron considerables, los vértices presentaron reducciones del 93.51 %, los ejes de 94.20 % , las caras un 94.72 % y los triángulos un 93.96 %.



(a)



(b)

Figura 30 Análisis en Blender de los modelos actualizados del FRL y pistón.

Todos los modelos que fueron actualizados les faltan sus respectivos archivos de animación, que son esenciales para la inmersión e interactividad del entorno. Un clip de animación se puede hacer con un software más especializado en modelado y renderización, sin embargo, Unity ofrece en su motor un editor de animaciones. El editor se abre como una ventana que está ligada a los objetos de la jerarquía, la escena y el inspector permite crear clips de animación, al modificar las características en el espacio de los GameObjects, así como su apariencia.

Se realizaron dos clips de animación que indican la transición de un estado a otro. Para el final de carrera, una vez que el vástago del pistón realiza contacto con el GameObject que representa el accionador, éste es rotado y posicionado para dar la impresión que está presionado. La llave del paso de aire se encuentra en una posición de reposo, y para activarse, se debe mover la llave a 180°, con un clic del usuario, la llave rota hasta llegar a su nueva posición. El pistón de doble efecto se encuentra en estado de reposo cuando está retraído el vástago y, al recibir aire en el ducto correspondiente, el vástago avanzará; finalmente, cuando su otro ducto reciba una señal y se encuentre el vástago extendido, va a regresarse a su posición original

Se muestra en la figura 31 el editor de Unity con el archivo que contiene la animación del final de carrera; para empezar a crear una animación se elige una propiedad a modificar, puede ser el tamaño, posición, rotación, etc. En el caso del final de carrera, se toma el GameObject hijo que representa el accionador, que contiene otro elemento hijo siendo el soporte; ambos se encuentran en una posición inicial de reposo, y cuando se reproduce la animación el soporte y el accionador realizan una rotación de -15° en el eje Z.

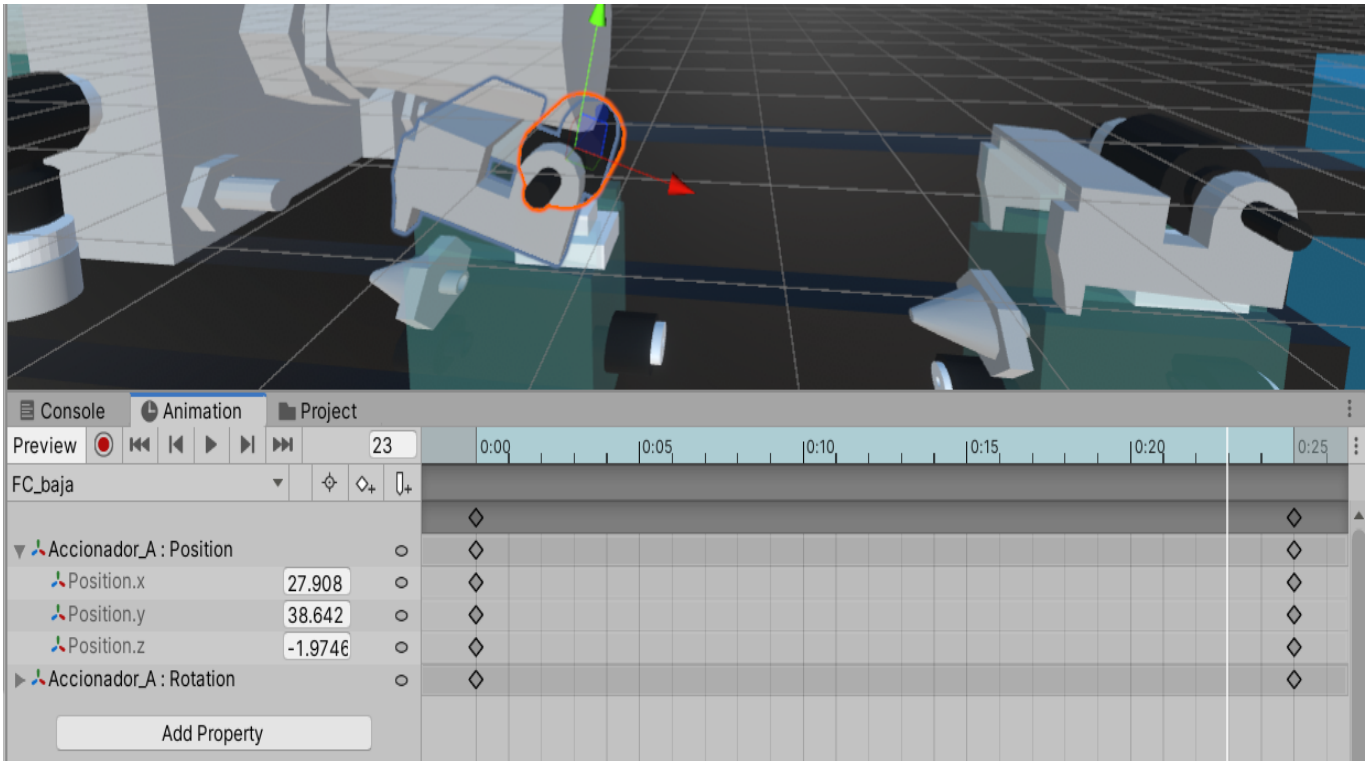


Figura 31 Editor de animación en Unity.

El FRL posee una llave que sigue el mismo principio que la llave de paso para activarse; además, se realizó otra animación para el FRL que simula el cambio de presión indicado por una manecilla del dial al ser activado. Por último, se tiene el pistón de simple efecto, con un vástago que avanza o retrocede, también cuenta con un resorte que se estira y contrae; esta animación se reproduce a la par que la del vástago.

Una vez que se cuenta con los clips de animación, se tienen que reproducir en el momento apropiado, para eso se necesita un controlador llamado *Animator Controller*, el cual permite ordenar y programar conjuntos de animaciones para un objeto o personaje. La interfaz del Animator Controller es similar a UdonGraph, lo que quiere decir que es a base de nodos. Dentro de esa interfaz, se tendrán los dos clips de animación como bloques que representan los estados de los elementos neumáticos, y se unen con nodos que tienen condiciones para controlar las transiciones de las animaciones de los nuevos modelos a través de variables booleanas. Finalmente, el controlador se implementa en el código *Man_Valv* como una variable del tipo Animator, que se usará como argumento para la función del elemento neumático correspondiente.

Se realizó un guión de nombre *FinCarreraScript*. En la figura 32 se puede observar cómo funciona el guión cuyo código usa la colisión del modelo del vástago para cambiar el valor de la variable booleana llamada animFinal, y su relacion con el Animator Controller para reproducir las animaciones. Cuando el vástago del pistón de doble efecto avance y choque su caja de colisión con la del final de carrera, manda una señal, lo que puede usarse para crear una condición: si el vástago está realizando contacto con el fin de carrera, animFinal se volverá True. El controlador recibe el estado actual de animFinal y reproduce el clip correspondiente, creando un ciclo.

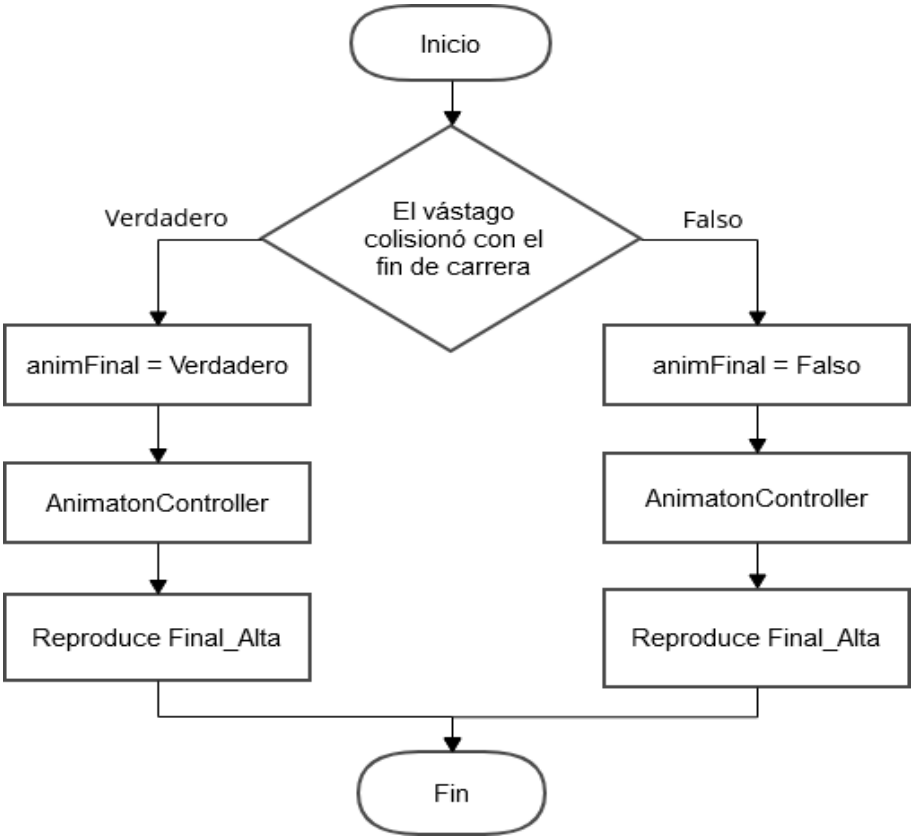


Figura 32 Diagramas del Animator Controller y el guión del final de carrera.

Para los elementos que requieren una interacción directa como la llave de paso de aire y el FRL, en sus códigos se tiene la condición de interacción con el usuario para reproducir sus animaciones, cambiando el valor de su respectiva variable booleana. Para otros elementos como los pistones, las condiciones dependen del estado de los ductos.

4 RESULTADOS

4.1 Pruebas con usuarios

Con el entorno terminado que presentó un rendimiento adecuado, se decidió solicitar a cuatro alumnos al área de proyectos de ingenieria avanzada del edificio de la DIMEI, para probar el laboratorio virtual y simular las prácticas cinco y seis del manual de prácticas del Laboratorio de Automatización Industrial, las cuales consisten en el uso de un pistón de simple y de doble efecto. Antes de que los usuarios ingresen al entorno virtual, se reprodujeron dos vídeos que se encargan de mostrar el entorno, cómo interactuar con los objetos así como la explicación de los objetivos y el principio de funcionamiento de los elementos neumáticos presentes en ambas prácticas; estos videos fueron realizados por el autor.

Posteriormente, los usuarios ingresaron al laboratorio y se familiarizaron con la navegación en el entorno virtual; lo primero que se notó es que el entorno no presentó reducciones de fotogramas por segundo. Después, se crearon dos equipos conformados por dos estudiantes mientras que un quinto usuario tomó el rol de revisar el progreso y resolver dudas; ambas prácticas fueron resueltas sin inconvenientes. Se realizaron más pruebas en distintas mesas de trabajo, hasta que un usuario en una mesa de su instancia se desvínculó de las demás; el usuario pudo manipular los objetos de la mesa y activar el circuito, pero los demás usuarios no fueron capaces de ver los cambios ni manipular los objetos de la mesa.

Despúes de estas pruebas se realizaron cambios finales referentes al uso del temporizador, específicamente el deslizador para controlar el tiempo, ya que era muy difícil de manipular para los usuarios con lentes de realidad virtual; se hizo un panel para cada mesa donde estará el *canvas* que resguardará el deslizador que es mucho más grande y así facilitar su uso. Finalmente, se solicitó subir el entorno actual a VRChat como primera versión, además de que se realizó otro laboratorio de neumática para un solo usuario, con un edificio más pequeño y con una sola mesa de trabajo que igualmente se subió al servidor.

4.2 Beneficios logrados

Con la finalización de este proyecto, se logró ofrecer un espacio virtual de aprendizaje a distancia con múltiples objetos interactivos, donde el alumno puede recibir clases y realizar actividades en tiempo real con otros compañeros. También se tiene un entorno donde un solo usuario puede entrar en cualquier momento para realizar actividades aún estando fuera de línea.

Para ambos entornos, el uso de visores de realidad virtual no es requerido; entonces, si se desea impartir clases a distancia con el uso de un computador personal, es posible y recomendable. Gracias a la

integración de los temporizadores negativo y positivo, se pueden realizar otras prácticas del manual. El manual de prácticas del Laboratorio de Automatización Industrial cuenta con 4 práctias de PLC, 7 de neumática y 3 de electroneumática, con un total de 14 prácticas; de las 7 prácticas de neumática, el laboratorio virtual se puede realizar 4.

Los modelos y sus componentes que forman el nuevo laboratorio, muestran una arquitectura atractiva con múltiples detalles para que los usuarios se encuentren más interesados al ingresar, interactuar y realizar las actividades en el entorno. Al tener las herramientas para construir ambientes sin límites de costos, espacio y tiempo, la calidad de los nuevos modelos mejoró el apartado visual general del laboratorio.

Se mejoró la experiencia de los usuarios ampliamente, al reemplazar múltiples objetos neumáticos con nuevos de un menor número de polígonos; el entorno ahora se renderiza sin caídas notables de fotogramas. Posteriormente se retocaron muchas animaciones e interacciones de los objetos neumáticos, para que la realización de los circuitos fuera más cómoda e intuitiva.

5 CONCLUSIONES Y TRABAJO FUTURO

5.1 Conclusiones

Los entornos virtuales para el aprendizaje se han convertido en una herramienta atractiva para empresas dedicadas a la tecnología, con el propósito de impartir cursos virtuales o entornos virtuales para conocer de una forma más llamativa e informativa sus productos; también han surgido comunidades o instituciones que usan estos medios para impartir educación y crear recursos didácticos. Sin embargo, despúes de la crisis sanitaria del COVID-19 estas herramientas han sido uno de los objetos de discusión de múltiples instituciones educativas. La Facultad de Ingeniería de la UNAM ha desarrollado múltiples entornos virtuales, además del Laboratorio de Neumática, tales como el Laboratorio de Materiales y entornos que simulan zonas naturales en México para prácticas de ingeniería civil, entre otros.

Con los beneficios logrados, se concluye que para la realización de un entorno virtual es necesario conocer el motor gráfico que se emplee y el software que se use. Para este proyecto, VRChat y Unity fueron herramientas que facilitaron ampliamente su desarrollo y expansión. VRChat y su kit de desarrollo contienen mucha documentación y una comunidad grande, en el que cualquier desarrollador puede expresar dudas para que sean resueltas por miembros mucho más experimentados; esto fue útil para entender varios conceptos de la interconectividad, el uso de los distintos métodos y funciones que ofrecen sus archivos de biblioteca en C#.

Además, las pruebas presenciales fueron de suma importancia para identificar los aspectos que afectan negativamente la experiencia del usuario. Con la retroalimentación de los usuarios ajenos al proyecto, se realizaron cambios respecto al diseño de los temporizadores, la optimización de los modelos de los elementos neumáticos y la detección de errores de sincronización para considerarlos en el trabajo futuro.

Es importante tener conciencia que, más que demostrar las capacidades de la tecnología, el objetivo de estas herramientas es impartir la educación de forma más innovadora e inclusiva, ya que la distancia entre los usuarios son factores que ya no son problema gracias a estos entornos. Finalmente, una realidad donde sea normalizado el uso de entornos virtuales para el aprendizaje, es algo que en este país aún requiere más tiempo de desarrollo y planeación.

5.2 Trabajo futuro

Posterior a esta contribución, el entorno requiere de algunos elementos neumáticos más para que el laboratorio se encuentre completo. Es necesario añadir las válvulas lógicas que representan las operaciones lógicas OR y AND, las cuales necesitan de su propio modelo con animaciones y su función dentro del código.

Además, despúes de encontrar el error de sincronización en una prueba, se consideró que es prioritaria su resolución para quienes desarrollarán las nuevas versiones de este laboratorio. Los códigos son otros aspectos que se recomienda optimizar para mejorar el rendimiento del proyecto; existen archivos que contienen códigos con funciones que pueden integrarse a *Man_Valv*, lo que esto ahorraría memoria al renderizar el entorno y sus funciones.

El proyecto en Unity que contiene toda la información, desde modelos tridimensionales, códigos en C# y demás, cuenta con una gran cantidad de archivos que ya no se utilizan o se encuentran incompletos. Se recomienda gestionar y organizar los contenidos del proyecto para disminuir su gran peso, además de que a los próximos desarrolladores se les facilitará su exploración.

Una vez que se realicen los cambios pertinentes para tener un laboratorio de neumática completo, se podrá contribuir ampliamente con esta herramienta de estudio y enseñanza para mejorar el aprendizaje de los alumnos. El docente podría mostrar simulaciones creadas en el entorno en clase como apoyo visual, además que puede ser una herramienta útil para los estudiantes que cursan la materia Automatización Industrial.

Los planes de estudio de cada carrera contienen muchas materias con laboratorio o con prácticas de campo en su temario. Se considera que tener un entorno virtual para las materias con un alto índice de reprobación en laboratorios y prácticas pueden ser herramientas de aprendizaje que mejoren el rendimiento de los estudiantes.

REFERENCIAS

[1] Facultad de Ingeniería, "Acerca de la División de Ingeniería Mecánica e Industrial," [Online]. Disponible: https://www.ingenieria.unam.mx/dimei/acerca.php, [Accedido en: 20-01-2024].

[2] Elvia Esthela Aispuro Felix, Jaime Suarez Villavicencio, Javier Aguilar Parra, Reyes Juárez-Ramírez, "The impact of the COVID-19 pandemic on higher education - A case of success of ICT educational programs," in *2022 10th International Conference in Software Engineering Research and Innovation (CONISOFT)*, Chiapas, México, 20 March 2023, pp. 49–57.

[3] María Luz Cacheiro González , *Educación y Tecnología: Estrategias Didácticas para la Integración de las TIC*. Madrid, España: Editorial UNED, 2018.

[4] "Home Virtual Reality VR Education Software & Augmented Reality Learning - VictoryXR," [Online]. Disponible: https://www.victoryxr.com/, [Accedido en: 05-11-2023].

[5] "VictoryXR Academy Virtual Reality VR Education Software & Augmented Reality Learning - VictoryXR," [Online]. Disponible: https://www.victoryxr.com/victoryxr-academy/, [Accedido en: 05-11-2023].

[6] A. Brown and W. Sugar, "Second life in education: The case of commercial online virtual reality applied to teaching and learning," in *Themes in Science and Technology education*, Carolina del Norte, Estados Unidos, 2010, pp. 107–115.

[7] VRChat Creator, "Getting Started," [Online]. Disponible: https://creators.vrchat.com/sdk/, [Accedido en: 12-08-2023].

[8] VRChat Creator, "Udon Node Graph," https://creators.vrchat.com/worlds/udon/graph/, [Accedido en: 12-08-2023].

[9] VRChat Creator, "UdonSharp," [Online]. Disponible: https://udonsharp.docs.vrchat.com/, [Accedido en: 12-08-2023].

[10] Crouzet Corporation, "Pneumatic Timers," [Online]. Disponible: https://media.crouzet.com/catalog/{_}datasheet/pdf/en/CO{_}PN{_}LO{_}Timers{_}EN.pdf, Oct 2022, [Accedido en: 18-08-2023].

[11] Unity Technologies, "Unity - Manual: Introducción a los componentes," [Online]. Disponible: https:

//docs.unity3d.com/es/530/Manual/Components.html, [Accedido en: 20-08-2023].

[12] Unity Technologies, "Unity - Manual: Serialización de script," [Online]. Disponible: https://docs.unity3d.com/es/530/Manual/script-Serialization.html, [Accedido en: 20-08-2023].

[13] V. Taylor, "Height Maps And Displacement In Unity HDRP," [Online]. Disponible: https://medium.com/geekculture/height-and-normal-maps-in-unity-hdrp-324fefb0d188, [Accedido en: 05-09-2023].