INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Proyecto Terminal

"Framework Cooperativo de Realidad Virtual para Modelado en 3D"

Que para obtener el título de

"INGENIERO EN TELEMÁTICA"

Presentan:

Leopoldo Andrés Hernández Vázquez

Alejandro Ávila Gómez

Ricardo Muñoz Guerrero

Asesores:

M. en C. Bella Citlali Martínez Seis

M. en C. Noé Sierra Romero

Ciudad de México, Junio, 2017

upiita·ipn

INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Proyecto Terminal

"Framework Cooperativo de Realidad Virtual para Modelado en 3D"

Que para obtener el título de:

"Ingeniero en Telemática"

P r e s e n t a n:

Alejandro Ávila Gómez

Ricardo Muñoz Guerrero

Leopoldo Andrés Hernández Vázquez

Asesores:

M. en C. Bella Citlali Martinez Seis

M. en C. Noé Sierra Romero

Presidente del Jurado:

M. en C. Blanca Alicia Rico Jiménez

Secretario del Jurado:

M. en C. Cyntia Eugenia Enriquez Ortiz

Agradecimientos

A mi papá por siempre haberme apoyado y animado cuando más lo necesitaba, nunca hubiera hecho este trabajo de no ser por su apoyo incondicional. A mi mamá por siempre acompañarme y creer en mí en los momentos difíciles. A mis asesores por su apoyo para la elaboración de este proyecto. A mi amigo de viajes Armando y mi amiga Sahiry por siempre darme ánimos y apoyarme en los momentos difíciles. A todos los profesores que he tenido porque de todos aprendí algo. Finalmente, agradezco a mi institución UPIITA, por haber tenido la oportunidad de estudiar ahí y darme las mejores experiencias de mi vida.

LAHV

Índice General

Índice de Figuras

Índice de Diagramas

Índice de Código

Índice de Tablas

Resumen

En este proyecto se presenta una aplicación de Realidad Virtual en la que múltiples usuarios son capaces de manipular figuras básicas en 3D, de manera que con ellas se pueden formar modelos más complejos. Se espera que este sistema sea la base para un entorno de desarrollo y modelado a nivel industrial.

Todos los usuarios que participan en este entorno virtual son capaces de realizar todas las modificaciones que éste ofrece, dichas modificaciones son perceptibles para el resto de los participantes en el momento en el que se realizan.

Para la elaboración de este proyecto se utilizaron los dispositivos de Realidad Virtual “Oculus Rift”, ya que éstos nos permiten utilizar sus características y aplicarlas a la visualización de prototipos que se diseñan por medio de realidad virtual. Además, se utilizó el Controlador “Leap Motion” cuya principal función es interpretar los movimientos de las manos que se traducen en sentencias de control sobre los modelos.

Palabras clave: *Realidad Virtual, Oculus Rift, Leap Motion, Framework, Software Colaborativo.*

Abstract

The main purpose of this project is to develop a virtual reality application. Multiple users can manipulate different 3D basic figures to create more complex ones. It is expected that this system could be the basis for an industrial development environment.

All users involved in this environment can make all the offered modifications, and these are visible for the rest of the participants in the moment they are made.

To develop this project, the virtual reality device "Oculus Rift" is used. It allows us to use its characteristics and to apply them to visualize prototypes, which are designed using virtual reality. In addition, Leap Motion sensor is used to interpret the movements of the hands, which were translated in control statements over the models.

Key Words: *Virtual Reality, Oculus Rift, Leap Motion, Framework, Collaborative Software.*

Reseña de Capítulos

Este documento se conforma de siete capítulos que recuperan la esencia del proyecto “Framework Cooperativo de Realidad Virtual para Modelado en 3D”. A continuación, se describe de manera general cada uno de ellos.

Capítulo 1: Introducción

Se dan las causas que originaron el desarrollo del proyecto, su justificación, la propuesta de solución y los objetivos tanto generales como particulares.

Capítulo 2: Marco Teórico

Se muestra la información que se investigó para la realización de este proyecto terminal, tanto referente a los dispositivos, así como temas relacionados al sistema que fueron utilizados para su implementación.

Capítulo 3: Estado del Arte

Tanto en el IPN como en otras instituciones, se han desarrollado diversos proyectos que se relacionan con este trabajo, en este capítulo se abordan las similitudes, pero sobre todo los aspectos que lo distinguen de sus predecesores.

Capítulo 4: Análisis de Requerimientos

Al desarrollar un software es vital seguir una especificación que permita tener en claro las capacidades que tendrá y las restricciones respecto a su funcionalidad.

Capítulo 5: Diseño del Sistema

El diagramado UML utilizado para la realización de este Proyecto Terminal.

Capítulo 6: Implementación

En este capítulo se documenta el proceso que materializó lo estipulado en el apartado anterior, se incluye la implementación del software y dispositivos, fragmentos de código y, la programación de los algoritmos diseñados.

Capítulo 7: Pruebas y Resultados

Se reportan las pruebas a las que fue sometido el Sistema para validar su funcionamiento, cuantificar su rendimiento y verificar el cumplimiento del objetivo general del proyecto.

Capítulo 1: **Introducción**

En los últimos años, los sistemas de Realidad Virtual se han ido popularizando. Hoy en día existe un sinfín de aplicaciones de Realidad Virtual en el mercado, las cuales están enfocadas a diversas áreas, tales como el entretenimiento, la publicidad o el diseño. Estas aplicaciones pueden ser usadas en diversas plataformas, como teléfonos inteligentes, tabletas electrónicas o computadoras personales.

Sin embargo, el principal enfoque que se le ha dado a esta tecnología es el entretenimiento; es muy fácil encontrar aplicaciones como juegos, videos y hasta en proyectos de enseñanza, pero, sin duda, uno de los campos menos desarrollados es el campo de modelado para diseño industrial.

1.1 Planteamiento del Problema

El diseño industrial es una tarea complicada debido a la dificultad para modelar y mostrar un diseño en 3D. El principal método de diseño es utilizando un programa de computadora, por ejemplo, AutoCAD, pero en este caso el problema radica cuando se quiere visualizar el diseño, ya que un ordenador solo puede simular un efecto en 3D. Otro camino es la realización de maquetas, pero requieren una gran cantidad de tiempo y recursos, y el cometer un error en la construcción puede ser incorregible. Un inconveniente más al hacer este tipo de diseños es que ocasionalmente se requiere un gran número de personas para completar un proyecto y hacer un diseño simultáneo en un lugar físico puede resultar complicado.

Actualmente existen aplicaciones de desarrollo industrial virtuales como Daqri[1], pero éstas solo funcionan con una sola persona y eso conlleva a que la realización de manera virtual sea muy lenta.

1.2 Justificación

En los últimos años la revolución tecnológica ha avanzado vertiginosamente y la utilización de la Realidad Virtual (RV) es más común.

Actualmente, han salido distintos dispositivos sencillos que nos permiten visualizar RV tales como Cardboards o lentes que se sobreponen en un Smartphone para poder visualizar distintos objetos en 3D. De igual forma, existen aparatos más complejos que nos permiten RV más compleja y con un procesamiento mayor como el Oculus Rift.

Igualmente, la utilización de aplicaciones cooperativas en donde distintos usuarios colaboren en un mismo proyecto encontrándose en distintos lugares es algo muy útil hoy en día, ya que nos ahorra la dificultad de trasladarnos a otras partes y trabajar eficientemente en un único fin.

Bajo esta perspectiva, se propone desarrollar un sistema para crear modelos usando la RV y donde múltiples usuarios puedan colaborar en un único proyecto colaborando de forma concurrente por medio de turnos y bloqueos.

Un proyecto con estas características pertenece al área de Telemática puesto que, se usará lenguaje de programación, así como intercambio de información por medio de protocolos de comunicación para lograr la sincronización de los usuarios usando distintas instancias del mismo proyecto.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una aplicación de Realidad Virtual cooperativa para diseño de escenarios en 3D partiendo de algunas formas básicas, donde los distintos usuarios sean capaces de modificar y observar los cambios en el diseño.

1.3.2 Objetivos Particulares

- Desarrollar un programa en el Oculus para que varios usuarios sean capaces de observar un escenario virtual en tres dimensiones.
- Desarrollar un programa que detecte los movimientos de manos sobre las imágenes virtuales utilizando el Leap Motion.
- Desarrollar un programa que, en base a los movimientos de manos detectados, genere cambios sobre el Oculus.
- Lograr la comunicación entre el servidor central y todos los usuarios.

1.4 Propuesta de Solución

Una aplicación de Realidad Virtual cooperativa hará más eficiente cualquier diseño que se quiera hacer, además de que personas de distintas áreas y en distintos lugares pueden estar involucradas en el diseño de éstos.

El sistema consiste en un ambiente virtual en donde, al menos tres usuarios, visualizan el mismo ambiente compuesto de diferentes formas básicas, las cuales, pueden ser modificadas. Cabe resaltar que las modificaciones que realice un usuario en alguna figura del ambiente son visibles para los demás usuarios.

Las formas básicas o figuras que se pueden manipular son las siguientes:

- Cilindro
- Cono
- Cubo
- Cápsula

Estas figuras son mostradas en el Oculus Rift.

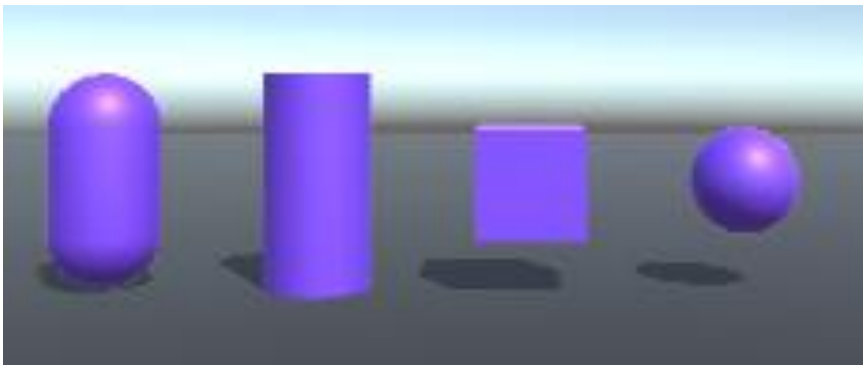


Figura 1. Formas básicas que podrán ser modeladas

Además de estas figuras básicas, el sistema está preparado para agregar figuras más complejas y realizar las mismas manipulaciones que tienen las figuras básicas.

Al mismo tiempo, los usuarios pueden realizar distintos cambios en el ambiente virtual mediante movimientos de sus manos, los cuales son detectados por el Leap Motion. Las modificaciones que se pueden realizar son:

- Insertar nuevas formas básicas al ambiente
- Cambiar el color de una figura
- Cambiar el tamaño de una figura
- Rotar una figura
- Cambiar la posición de una figura

Figura 2. Ejemplo de funcionamiento de Leap Motion

El sistema entonces se divide en dos módulos:

1. Un programa de control para cada uno de los Usuarios. Este programa cumple lo siguiente:
    a. Generar la imagen virtual del diseño a modelar y mostrarla en los Oculus Rift.
    b. Captar los datos del movimiento de las manos que interactúan sobre la imagen virtual, a través del LeapMotion.
    c. Enviar los datos al servidor central.
2. Un programa de escritorio que cumple los siguientes requisitos para el Servidor:
    a. Procesar las peticiones de los Usuarios, así como llevar a cabo la administración de éstos.
    b. Procesar las modificaciones realizadas al modelo.
    c. Enviar la información necesaria del modelo para ser visualizado por los usuarios.
    d. Mantener la sincronización de los Usuarios.

1.5 Metodología

Se decidió utilizar el modelo en espiral como técnica de desarrollo del software propuesto en este documento. A continuación, se enlistan y describen las actividades generales que se llevaron a cabo para realizar este proyecto terminal:

- Recopilar documentación relacionada con la Realidad Virtual, Leap Motion y Software Cooperativo con sus características mínimas.
- Investigación bibliográfica referente a las tecnologías de generación de gráficos para Realidad Virtual y detección de movimientos para el Leap Motion.
- Investigación para la sincronización, bloqueo y comunicación entre aplicaciones gráficas.
- Generar un análisis de requerimientos para satisfacer las necesidades de un software cooperativo.
- Diseñar el sistema de basándonos en la realización de tres módulos: detección de movimientos, generación de gráficos y sincronización entre usuarios.
- Integración final de los tres módulos y su adaptación.
- Realización de múltiples pruebas, búsqueda de errores y su corrección.

1.6 Alcances

Al finalizar este proyecto, se obtuvo un prototipo de Framework donde múltiples usuarios son capaces de ver y modificar un diseño virtual de manera simultánea. El sistema es capaz de soportar hasta cinco usuarios en una misma sesión de manera concurrente por medio de turnos encolados a través de la red. Sin embargo, es importante aclarar que las pruebas finales solo se hicieron con tres equipos conectados al mismo servidor central, el primero tuvo conectado el Oculus Rift y el Leap Motion, el segundo solamente el sensor de movimiento y el último solo se usó para mostrar la funcionalidad de petición de turnos y visualización de imágenes. De igual forma, para mostrar el proyecto es necesario que todos los usuarios tengan una conexión a Internet y la sincronización depende de la velocidad de ésta.

En lo que respecta con la precisión con la que se detectan las manos por medio del Leap Motion, ésta se puede ver afectada por la iluminación y la velocidad con la que se mueven.

Finalmente, es importante aclarar que la resolución es un poco baja debido a que lo equipos que se utilizaron en las pruebas no poseen capacidades gráficas muy altas, sin embargo, la aplicación funciona correctamente.

Capítulo 2: **Marco Teórico**

Para la visualización y manipulación de objetos en 3D se deben considerar conceptos relacionados con la Realidad Virtual, así como dispositivos que nos ofrezcan características aplicables a esta tecnología, como visores en 3D y sensores de movimientos, que se explicará a lo largo de este capítulo. De igual forma, se abordó el tema de Software Colaborativo el cual es la base de la interconectividad de este proyecto, así como concurrencias y bloqueos.

2.1 Realidad Virtual

La Realidad Virtual es un entorno de escenas u objetos de apariencia real generado mediante tecnología informática, que crea en el usuario la sensación de estar inmerso en él. Dicho entorno es contemplado por el usuario a través normalmente de un dispositivo conocido como gafas o casco de realidad virtual.

Este puede ir acompañado de otros dispositivos, como guantes o trajes especiales, que permiten una mayor interacción con el entorno, así como la percepción de diferentes estímulos que intensifican la sensación de realidad.

La aplicación de la realidad virtual, aunque centrada inicialmente en el terreno del entretenimiento y de los videojuegos, se ha extendido a otros muchos campos, como la medicina, la arqueología, la creación artística, el entrenamiento militar o las simulaciones de vuelo.

La Realidad Virtual puede ser de dos tipos: inmersiva y no inmersiva:

- Los métodos inmersivos de realidad virtual con frecuencia se ligan a un ambiente tridimensional creado por un ordenador, el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano.
- La realidad virtual no inmersiva también utiliza el ordenador y se vale de medios como el que actualmente nos ofrece Internet, en el cual podemos interactuar en tiempo real con diferentes personas en espacios y ambientes que en realidad no existen sin la necesidad de dispositivos adicionales al ordenador[2].

2.2 Software Colaborativo

Los softwares colaborativos son sistemas de cómputo que permiten a usuarios trabajar en entornos comunes y de forma virtual, en los cuales comparten información y documentos entre sí de manera ordenada y controlada. Así mismo permiten a empresas, personal de éstas, y en general cualquier persona que tenga acceso estas herramientas, crear entornos de trabajo virtuales, en donde usuarios en diferentes lugares de puedan trabajar de manera conjunta a través de Internet. [3]

Existen muchos beneficios entre los cuales, se deben nombrar los que han sido de mayor relevancia a nivel general:

- Ahorro en tiempo, ya que no tiene que desplazarse físicamente una persona hasta una oficina en un lugar específico.
- Comunicación más estrecha entre los miembros del equipo de trabajo.
- Mejor control de las actividades de cada miembro del equipo, ya que se tienen evidencias por escrito de las actividades reportes, comentarios, etcétera.
- Los reportes que ha tenido cada una de las personas participantes, sin necesidad de desplazarse físicamente para reportar lo hecho.

Los softwares colaborativos son también llamados groupware, el término "groupware" (en español, conjunto de programas informáticos colaborativos) se refiere al uso de métodos y herramientas de software que permiten que los usuarios realizar trabajos colectivos a través de las redes. Por lo tanto, groupware hace referencia a las diversas y variadas aplicaciones que contribuyen a una única y misma meta: permitir que usuarios separados geográficamente trabajar en equipo, como se había mencionado con anterioridad.

Figura 3. El software colaborativo permite realizar trabajos colectivos a través de la red.

Las características más importantes del software colaborativo o groupware son:

- Proveer de un ambiente de colaboración en el que realmente se perciba que el trabajo en grupo se lleva a cabo.
- Mantener la información en un solo sitio común para todos los miembros.
- Interactuar con otros usuarios de forma escrita, con voz y/o vídeo.

Los softwares colaborativos se pueden clasificar en base al tiempo, los cuales pueden ser sincrónicos y asincrónicos, también están clasificados en base al espacio, los cuales pueden estar en un mismo lugar o estar de forma distribuida. Las aplicaciones típicas de los sincrónicos (los cuales soportan aplicaciones en tiempo real) son: pizarrones compartidos, tele-conferencias, chat y sistemas de toma de decisiones. Algunos ejemplos de aplicaciones típicas de los softwares colaborativos asincrónicos son: e-mail, SharePoint, calendarios y sistemas de escritura de colaboración.

Los softwares colaborativos se están volviendo más populares dentro de las empresas y grupos de trabajo de estudio y toma de decisiones, ya que resulta mejor instalar esta herramienta y comprar o implementar un sistema de colaboración, a diferencia que a estar transportando el personal de un lugar a otro.

Los softwares colaborativos deben proporcionar tres funciones esenciales dentro de un grupo, llamadas las 3 C's:

- Comunicación
- Colaboración
- Coordinación

2.2.1. Comunicación

Es la función más importante, ya que es el medio en que la información es compartida. Esto se consigue mediante una serie de reglas para iniciar la emisión y recepción de información entre distintos usuarios.

2.2.1. Colaboración

Es utilizada para unir la cooperación y resolver problemas de negocios de alguna actividad empresarial o a nivel de estudios. Proporciona la ventaja de resolver problemas de las reuniones físicas de trabajo entre varias personas para la realización de la misma y tener la disponibilidad de la información.

2.2.1. Coordinación

Es la acción de asegurar que el equipo está trabajando eficientemente y en conjunto para alcanzar una meta. Esto incluye la distribución de tareas y revisión de su ejecución.

2.3 Oculus Rift

Figura 4. Oculus Rift DK

Oculus Rift es un casco de realidad virtual que está siendo desarrollado por Oculus VR. Durante su periodo como compañía independiente, Oculus VR ha invertido 91 millones de dólares para el desarrollo de Oculus Rift. La versión para el consumidor fue liberada en el 2016 y aunque es más sencillo conseguir la versión para desarrollador. [4] Para la realización de este proyecto terminal, se utilizó el Oculus Rift Development Kit 1, la primera versión para desarrollo liberada por Oculus.

Características:

- Pantalla de 7 pulgadas LCD de 24 bits por pixel.
- Campo de visión es de más de 90 grados horizontales.
- Está destinado a cubrir casi todo el campo visual del usuario de vista, para crear un fuerte sentido de la inmersión.
- La resolución es de 1280 × 800.
- Rastreador de movimiento de cabeza a 1000 Hz. (Permite detectar de forma muy fiable las imágenes mostradas al movernos).
- Peso: 379 g.
- Posee entradas DVI y HDMI.

- Utiliza una interfaz USB la cual se utiliza para dar datos de seguimiento al servidor.
- Incluye un kit de desarrollo de software.
- Costo: 350 USD.

2.4 Controlador Leap Motion

Figura 5. Controlador Leap Motion

Es un dispositivo que utiliza un sensor de movimiento que anula el uso de un ratón de ordenador. Los sensores detectan los movimientos de las manos y lo convierte en datos que pueden ser recuperados y mandados a un script con los cuales se puede trabajar. [5]

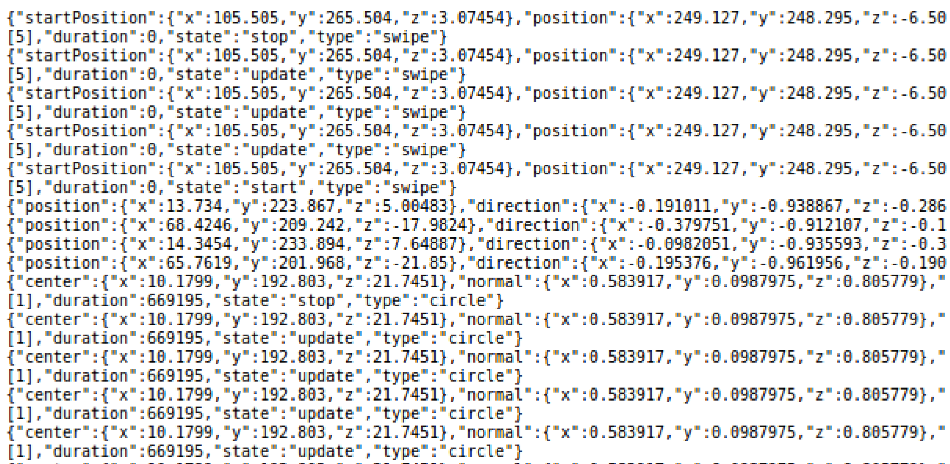
Características:

- El controlador del Leap Motion es un pequeño dispositivo periférico USB que está diseñado para ser colocado en un escritorio físico mirando hacia arriba.
- Usa dos cámaras de infrarrojos monocromáticos y tres infrarrojos LEDs, el dispositivo observa en un área aproximadamente semiesférica a una distancia de aproximadamente 1 metro.
- Los LEDs generan patrones de luz y las cámaras generan cerca de 300 cuadros por segundo de datos reflejados, que se envía a través de un cable USB al ordenador central, donde es analizada por el software del controlador de Leap Motion usando "matemáticas complejas" de una manera que no ha sido divulgada por la empresa. En el ordenador central es posible recuperar los datos del movimiento y a partir de esto empezar a desarrollar

2.4.1. Comunicación por Web Socket

El software Leap Motion instalado en cualquier computadora con su controlador proporciona datos de seguimiento a través de un servidor web. El servidor puede ser escuchado a través del puerto 6347 como localhost (http://127.0.0.1:6437). Cualquier cliente, incluidos los clientes web, pueden realizar una conexión y pueden acceder a los datos de seguimiento del Leap Motion en forma de mensajes con formato JSON. El servidor Web es proporcionado por el proceso del Leap Motion, que se ejecuta como un servicio en Windows y un demonio en OS X y Linux.[6]

JSON (JavaScript Object Notation) es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en JavaScript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.



{"startPosition":{"x":105.505,"y":265.504,"z":3.07454},"position":{"x":249.127,"y":248.295,"z":-6.50
[5],"duration":0,"state":"stop","type":"swipe"}
{"startPosition":{"x":105.505,"y":265.504,"z":3.07454},"position":{"x":249.127,"y":248.295,"z":-6.50
[5],"duration":0,"state":"update","type":"swipe"}
{"startPosition":{"x":105.505,"y":265.504,"z":3.07454},"position":{"x":249.127,"y":248.295,"z":-6.50
[5],"duration":0,"state":"update","type":"swipe"}
{"startPosition":{"x":105.505,"y":265.504,"z":3.07454},"position":{"x":249.127,"y":248.295,"z":-6.50
[5],"duration":0,"state":"update","type":"swipe"}
{"startPosition":{"x":105.505,"y":265.504,"z":3.07454},"position":{"x":249.127,"y":248.295,"z":-6.50
[5],"duration":0,"state":"start","type":"swipe"}
{"position":{"x":13.734,"y":223.867,"z":5.00483},"direction":{"x":-0.191011,"y":-0.938867,"z":-0.286
{"position":{"x":68.4246,"y":209.242,"z":-17.9824},"direction":{"x":-0.379751,"y":-0.912107,"z":-0.1
{"position":{"x":14.3454,"y":233.894,"z":7.64887},"direction":{"x":-0.0982051,"y":-0.935593,"z":-0.3
{"position":{"x":65.7619,"y":201.968,"z":-21.85},"direction":{"x":-0.195376,"y":-0.961956,"z":-0.190
{"center":{"x":10.1799,"y":192.803,"z":21.7451},"normal":{"x":0.583917,"y":0.0987975,"z":0.805779},"
[1],"duration":669195,"state":"stop","type":"circle"}
{"center":{"x":10.1799,"y":192.803,"z":21.7451},"normal":{"x":0.583917,"y":0.0987975,"z":0.805779},"
[1],"duration":669195,"state":"update","type":"circle"}
{"center":{"x":10.1799,"y":192.803,"z":21.7451},"normal":{"x":0.583917,"y":0.0987975,"z":0.805779},"
[1],"duration":669195,"state":"update","type":"circle"}
{"center":{"x":10.1799,"y":192.803,"z":21.7451},"normal":{"x":0.583917,"y":0.0987975,"z":0.805779},"
[1],"duration":669195,"state":"update","type":"circle"}
{"center":{"x":10.1799,"y":192.803,"z":21.7451},"normal":{"x":0.583917,"y":0.0987975,"z":0.805779},"
[1],"duration":669195,"state":"update","type":"circle"}

Figura 6. JSON generado por el controlador Leap Motion que describe sus movimientos

2.5 Concurrencia

En ciencias de la computación, la concurrencia es una propiedad de los sistemas en la cual los procesos de un cómputo se hacen simultáneamente, y pueden interactuar entre ellos.[7]

Como los procesos en un sistema concurrente pueden interactuar entre ellos mismos mientras se están ejecutando, el número de posibles rutas de ejecución en el sistema puede ser extremadamente grande, y el resultado final puede ser indeterminado. El concurrente uso de recursos compartidos puede ser una fuente de indeterminación dirigida a asuntos como Bloqueo mutuo, y *Starvation*.

El diseño de sistemas concurrentes a menudo implica la búsqueda de técnicas confiables para coordinar su ejecución, el intercambio de información, asignación de memoria, y una ejecución programada para minimizar la respuesta de tiempo y maximizar el rendimiento.

2.2.1. Bloqueo Mutuo

El bloqueo mutuo (también conocido como deadlock o abrazo mortal) es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema

2.2.2. Exclusión Mutua

La exclusión mutua es usada en programación concurrente para evitar el uso simultáneo de recursos comunes, como variables globales, por fragmentos de código conocidos como secciones críticas.

La mayor parte de estos recursos son las señales, contadores, colas y otros datos que se emplean en la comunicación entre el código que se ejecuta cuando se da servicio a una interrupción y el código que se ejecuta el resto del tiempo. Se trata de un problema de vital importancia porque, si no se toman las precauciones debidas, una interrupción puede ocurrir entre dos instrucciones cualesquiera del código normal y esto puede provocar graves fallos.

La técnica que se emplea por lo común para conseguir la exclusión mutua es inhabilitar las interrupciones durante el conjunto de instrucciones más pequeño que impedirá la corrupción de la estructura compartida (la sección crítica). Esto impide que el código de la interrupción se ejecute en mitad de la sección crítica.

2.2.3. Sección Crítica

Se denomina sección crítica, en programación concurrente, a la porción de código de un programa de ordenador en la que se accede a un recurso compartido (estructura de datos o dispositivo) que no debe ser accedido por más de un proceso o hilo en ejecución. La sección crítica por lo general termina en un tiempo determinado y el hilo, proceso o tarea sólo tendrá que esperar un período determinado de tiempo para entrar. Se necesita un mecanismo de sincronización en la entrada y salida de la sección crítica para asegurar la utilización en exclusiva del recurso, por ejemplo, un semáforo.

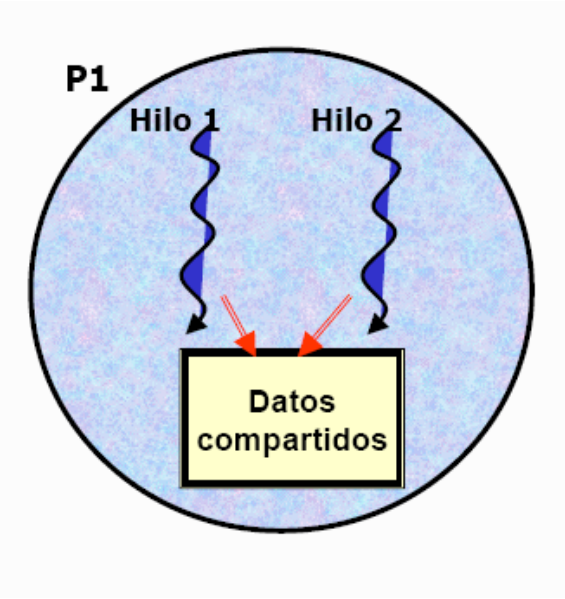


Figura 7. Concurrencia, Sección Crítica, Exclusión Mutua

Capítulo 3: **Estado del Arte**

El estado del arte es una modalidad de la investigación documental que permite el estudio del conocimiento acumulado dentro de un área específica. Para la realización de este Proyecto Terminal se revisaron sistemas que existen actualmente en el mercado, además de sistemas dentro del Instituto Politécnico Nacional.

3.1 Sistemas Similares en el Mercado

Actualmente hay una aplicación de Realidad Aumentada compartida en desarrollo en un laboratorio de Microsoft dirigido por Jaron Lanier, uno de los pioneros de la realidad virtual desde la década de 1980 a través de su compañía VPL Investigación. El proyecto, llamado Comradre, permite a varios usuarios compartir experiencias virtuales y de realidad aumentada. Permiten un campo de visión de 60 grados y cada equipo se comunica entre sí por medio de una red LAN. Es muy importante mencionar que este avance fue conseguido por pasantes de maestría y doctorado en dos meses[8].

Otro organismo que usa la Realidad Virtual para la industria es la empresa IC.IDO, la cual que genera ambientes industriales por medio de Realidad Virtual. Estos ambientes pueden ser ambientes de fábrica y farmacéuticos.

3.2 Sistemas Similares en el IPN

Investigadores del Instituto Politécnico Nacional desarrollaron una simulación de realidad virtual del sistema solar utilizando sensores de movimiento Kinect y los lentes de realidad virtual, Oculus Rift. Se diseñó una interfaz sencilla y práctica, en donde el usuario intuitivamente puede interactuar con el espacio creado, acercarse a los planetas y leer la información de cada uno, girar la cabeza y observar otros astros, entre otras cosas. Para la elaboración del software se utilizó el motor gráfico Unity 3D, que es una plataforma para crear videojuegos y experiencias en 2D Y 3D; también se utilizó el programa Maya y algoritmos de programación que hicieron posible la compatibilidad entre el Kinect y los Oculus Rift. [9]

3.3 Sistemas Similares en UPIITA

Se encontraron dos trabajos terminales para obtener el título en Ingeniero en Telemática que se basaron en el uso de Realidad Virtual, aunque ninguno uso realidad virtual compartida:

- “Sistema didáctico de interacción con mundo de realidad virtual: Caso de prueba de sistema solar”. Presentado por Delgado Resendiz Elizabeth. Se hizo un modelado del sistema solar por medio de VRLM, Flash y HTML del sistema solar. Este modelado ocupaba realidad virtual no inmersiva y funcionaba sobre un ordenador.
- “Ambiente de realidad virtual para la exploración del planeta marte desarrollado con Kinect (Digimars 3d)”. Presentado por Casablanca Cruz Claudia América y Durán Rincón Mariana Montserrat.

Como se puede mostrar, a pesar que hay una buena cantidad de proyectos de realidad virtual, estos no han estados dirigidos a ver modelos para la industria. De igual forma, todos estos proyectos han sido dirigidos hacia un solo usuario y no a múltiples, además ningún proyecto ha usado la realidad virtual inmersiva ni modificar el ambiente de realidad virtual. Estas características hacen a este proyecto un pionero en UPIITA en ese campo.

Capítulo 4: **Análisis de Requerimientos**

Para que el proyecto se pudiera desarrollar de manera correcta se especificaron primero los requerimientos, tanto funcionales como no funcionales, para tomarlos como base y poder realizar el diseño del Sistema.

También se especificaron los dispositivos o herramientas que se utilizaron, realizando una comparación entre los productos existentes con el fin de seleccionar los que mejor se adaptaran a las necesidades y requerimientos del Sistema.

Así mismo se especificó la Arquitectura del Sistema, como se muestra en la Figura 8, la cual consta de los siguientes componentes:

- Un servidor en la nube el cual se encarga de reproducir los cambios a todos los usuarios.
- Las distintas computadoras las cuales tienen instalada la aplicación.
- Un Oculus Rift y un Leap Motion conectados a cada equipo.

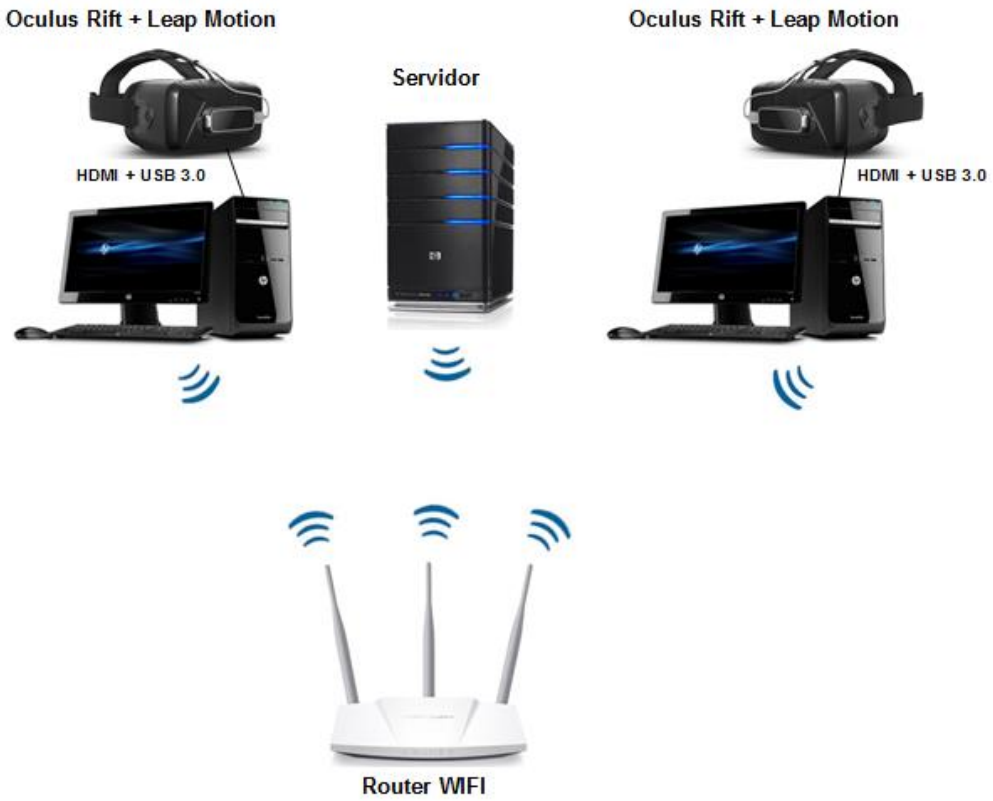


Figura 8. Arquitectura general del Sistema.

4.1 Requerimientos del Sistema

Se requiere un Software que permita al usuario visualizar en 3D los modelos que se desarrollen gracias a los gestos o movimientos que éste haga con las manos, por lo que se necesitan dispositivos específicos, tanto para la visualización de objetos en 3D, como para la detección de gestos manuales.

Este Sistema permite que varios usuarios participen en el modelado de objetos virtuales. Cada usuario que desee participar en el proyecto debe tener un nombre o nickname con el que se le identifica durante su colaboración.

Todos los colaboradores tienen su vista compartida del proyecto en sus respectivos dispositivos de Realidad Virtual. También muestra otros detalles como una cola de turnos donde se especifica que es su turno y que no muestre su propio nickname.

El primer usuario en acceder al sistema se le otorga la categoría de "Administrador", y él es capaz de iniciar o finalizar una sesión compartida. Al iniciar el Sistema, éste le ofrece las opciones de "Crear un Nuevo Proyecto" y "Abrir Proyecto".

Cuando uno de los colaboradores desee realizar alguna modificación al modelo actual debe solicitar un turno mediante un gesto con la mano. El sistema guarda las solicitudes en una cola y sólo inicia el turno del siguiente usuario hasta que la persona que realiza las modificaciones lo indique mediante otro gesto. Es importante mencionar que un usuario que se encuentra ya en la cola de turnos no puede solicitar de nuevo un turno.

Las modificaciones que están definidas para este Software, son:

- Inserción de una nueva figura.
- Cambio de tamaño.
- Cambio de posición.
- Cambio de color.
- Rotación de una figura.
- Eliminación de una figura.

Mientras un usuario realiza modificaciones, los demás no pueden hacer ningún cambio y sólo están en modo de espectador, la única acción que pueden realizar es solicitar un turno, esto no interrumpe al colaborador que está modelando, sólo se agrega a la cola de turnos.

Las modificaciones realizadas en el proyecto se almacenan en una pila para que sea posible deshacer cambios. Un usuario solo puede deshacer cambios cuando el haya sido el autor de éstos. Al ceder el turno, la pila se limpiará y las acciones ya no podrán ser revertidas.

El proyecto se almacena en la aplicación en la que trabaja el Administrador, esto pensando en el hecho de que él debe volver a participar en el proyecto, aunque ninguno de los otros colaboradores lo vuelva a hacer.

4.2 Dispositivo de Gestos Manuales

Existen diversos dispositivos que nos permiten reconocer o identificar movimientos corporales, pero son muy pocos los que están enfocados a los gestos manuales. Se encontraron dos dispositivos, cuyas características nos permiten identificar este tipo de movimientos, el Controlador Leap Motion y los guantes GloveOne. Éstos últimos tienen la característica de que, además de hacer la detección del movimiento de las manos, poder hacer sentir texturas tal como si estuviéramos tocando algo en la vida real gracias a unos actuadores que están colocados a la altura de las huellas dactilares, tiene además sensores detectan movimientos. La desventaja de estos dispositivos es que, si se desea una mayor precisión en la detección de los movimientos, se debe utilizar con otro sensor, como el Leap Motion.[10]

El Controlador Leap Motion, por otro lado, da seguimiento de las manos y los dedos, informa de posición, velocidad y orientación con un rápido tiempo de respuesta y buena precisión. Leap Motion puede ser usado de dos formas: sobre una mesa o montado en un dispositivo de realidad virtual. Por lo que lo consideramos como el mejor dispositivo para la realización de este proyecto.

4.3 Dispositivos de Realidad Virtual

Hay una gran variedad de dispositivos de Realidad Virtual en el Mercado, sin embargo, no todos nos ofrecen las mismas características.

Existen unos cascos de Realidad Virtual de PlayStation que ofrecen características de vídeo y sonido avanzadas, sin embargo, están aún en desarrollo y al ser dispositivos nuevos no existe gran cantidad de documentación disponible[11].

Otra alternativa son las carcasas o Cardboards, que disminuyen considerablemente el precio, pero no cuentan con una pantalla incluida, empleando así la de un teléfono inteligente[12].

Los Oculus Rift, por su parte, cuentan con una amplia cantidad de documentación y foros, donde se puede encontrar información de ayuda al implementar alguna aplicación para estos. Además, es importante mencionar que los desarrolladores del Controlador Leap Motion, desarrollaron una API especialmente para la interacción de estos dos dispositivos llamada ORION[13]. Esta nueva API ofrece las siguientes características:

- Mejoras en la detección de movimientos en el Leap Motion.
- Incluye nuevas superficies para mejorar la experiencia visual.
- Permite que el Controlador Leap Motion expanda su rango de detección.
- Hace una detección de movimientos más rápida.
- Incluye nuevas funciones que permiten la interacción con objetos virtuales.

4.4 Motor Gráfico

Los dos principales motores gráficos que se usan en Oculus son Unity y Unreal Engine[14]. Ambos funcionan con el asset de Leap Motion para la conectividad, así que decidimos crear una tabla comparativa para saber que motor nos convendría más en nuestro proyecto.

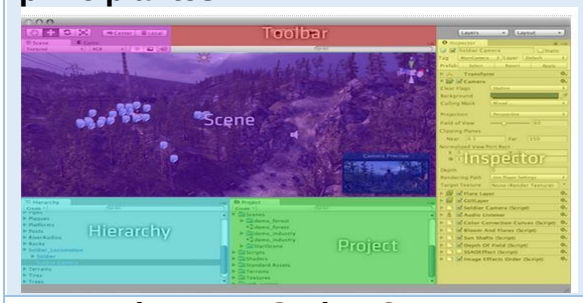
Unity 5	Unreal Engine 4
Usado por 47% de los desarrolladores.	Usado por 13% de los desarrolladores.
Unity se ofrece en dos ediciones: • **"Personal Edition": ofrecido gratuitamente.** • **"Professional Edition": precio único de $1,500 dólares o $75 dólares al mes.**	Ofrecido de forma gratuita.
Documentación muy bien escrita con excelente explicación y con capturas de pantalla.	Gran cantidad de documentos, pero aun con falta de documentación.
Plataformas: Windows PC, Mac OS X, iOS, Android, VR, Linux, Steamos, HTML5, Xbox One, and PS4.	Plataformas: Windows PC, Mac OS X, Linux, Web Player, WebGL, VR (incluyendo HoloLens), SteamOS, iOS, Android, Windows Phone 8, Tizen, Android TV and Samsung SMART TV, as well as Xbox One & 360, PS4, PlayStation Vita, y Wii U.
Editor de uso fácil para principiantes 	Interfaz más complicada de usar para usuarios principiantes. 
Lenguajes: JavaScript, C# y Boo.	Lenguajes: Blueprints Visual Scripting y C++.
Servicios: Ads, multijugador, grabador, compartir contenido y soporte.	Financiamiento para agregar los mismos servicios de Unity a futuro.

Tabla 1. Tabla comparativa entre Unity 5 y Unreal Engine 4.

El motor de gráficos Unity nos ofrece mejores características para el desarrollo de nuestro Proyecto, tales como el soporte multijugador y los Ads, por lo cual, lo seleccionamos como el motor de gráficos a utilizar.

4.5 Lenguaje de Programación

Como se especificó en el Apartado anterior el motor gráfico que se utiliza es Unity. Los tres principales lenguajes soportados por Unity son: JavaScript, C#, y un dialecto de Python llamada Boo[15].

Unity cuenta con soporte completo para .NET 2.0 para desplegar una aplicación de escritorio y un subconjunto de .NET 2.0 cuando se despliega como aplicación web o de dispositivo móvil.

Figura 9. Lenguajes Soportados por Unity 5

- Boo: Es un lenguaje de programación orientado a objetos, de tipos estáticos para la *Common Language Infrastructure* con una sintaxis inspirada en Python y un énfasis en la extensibilidad del lenguaje y su compilador. Sus características incluyen la inferencia, los generadores, multimétodos, duck typing opcional, macros, clausuras, currificación y funciones de primera clase. Boo es software de código abierto[16].

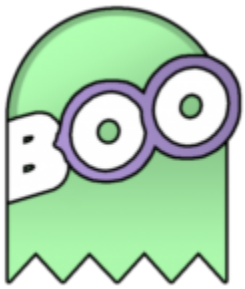


Figura 10. Lenguaje de Programación Boo

- C#: es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java[17].

Figura 11. Lenguaje de Programación C#

- JavaScript: JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas[18].

Figura 12. JavaScript

Boo es el lenguaje que menos soporte tiene en Unity, de igual forma no existen muchos ejemplos por lo cual, no existen muchas aplicaciones. Por lo tanto, se decidió que este lenguaje no se ocuparía ni seria tomado en cuenta.

De igual forma, el JavaScript que soporta Unity es un lenguaje similar a este conocido como UnityScrip con cambios muy significativos en la que muchas de las características dinámicas de JavaScript han sido eliminadas. En conclusión, aun cuando se esté familiarizado con JavaScript, aun será necesario aprender nuevas cosas.

Por otro lado, C# al ser un lenguaje muy usado, existe gran cantidad de soporte con características muy avanzadas y genéricas, por lo tanto, puedes encontrar ayuda en tus problemas fuera de la comunidad de Unity. También, como editor podemos usar Visual Studio con herramientas como autocompletar o soporte que serán de gran ayuda para agilizar un proyecto grande como este.

Finalmente, se decidió utilizar el lenguaje C#, ya que este tiene un mejor soporte, da más control al programador de la aplicación. Además, es más fácil de lograr una familiarización con el editor de C#.

4.6 IDE a Utilizar

Unity cuenta con su propio compilador y editor de código, sin embargo, durante el flujo de instalación de Unity es posible descargar Visual Studio 2015 y crear herramientas de integración entre ambos para desarrollar el código en Visual Studio[19].

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django[20].

La integración con VisualStudio de Unity permite crear y mantener archivos de proyecto de VisualStudio automáticamente. Además, cuando haga doble click en un script o en un mensaje de error en la consola de Unity, se abrirá VisualStudio. De igual forma, VisualStudio cuenta con mucha documentación y diferentes librearías.

Finalmente, se decidió usar Visual Studio debido a la amplia documentación que existe y la integración por defecto que existe con Unity.

4.7 Definición de gestos

Leap Motion Controller es un pequeño dispositivo periférico que está diseñado para ser colocado sobre un escritorio físico, idealmente mirando hacia arriba. Posee dos cámaras y tres LEDs infrarrojos y está básicamente orientado a permitir la detección de manos, dedos, herramientas de ciertas características y gestos realizados con estas de forma muy precisa. El alcance espacial es de aproximadamente 0,01 mm, con un FOV (field of view) de 115° centrado en el dispositivo, el cual forma una pirámide invertida.

Leap Motion Controller maneja un sistema de coordenadas cartesiano centrado en el dispositivo y los valores reportados por el mismo son en milímetros. Los ejes se pueden ver en la Figura 9, donde el eje X y Z conforman un plano horizontal siendo el eje X paralelo al lado más largo del dispositivo. Por otro lado, el eje Y es perpendicular al dispositivo considerando los valores positivos como aquellos por encima del mismo, mientras que los valores en Z positivos son aquellos por delante del dispositivo.

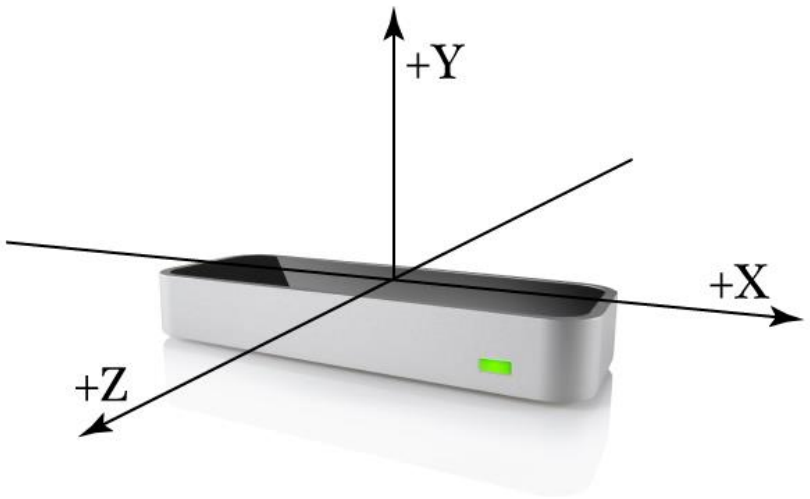


Figura 13. Sistema de coordenadas de Leap Motion Controller

4.7.1 Insertar una Nueva Figura

Para insertar una figura el usuario deberá realizar el siguiente gesto mostrado en la Figura 14, el cual consiste en extender la palma de la mano hacia arriba.

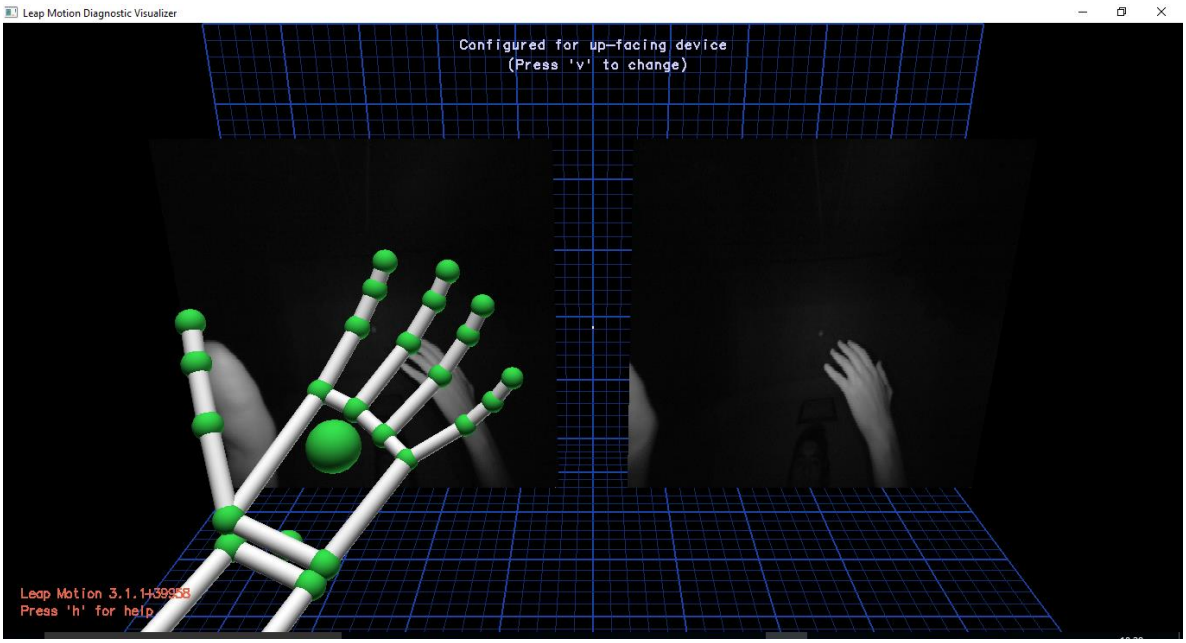


Figura 14.Gesto para la inserción de una Nueva Figura.

Manteniendo la mano de esta forma, una paleta de figuras básicas es desplegada como se ven a continuación:

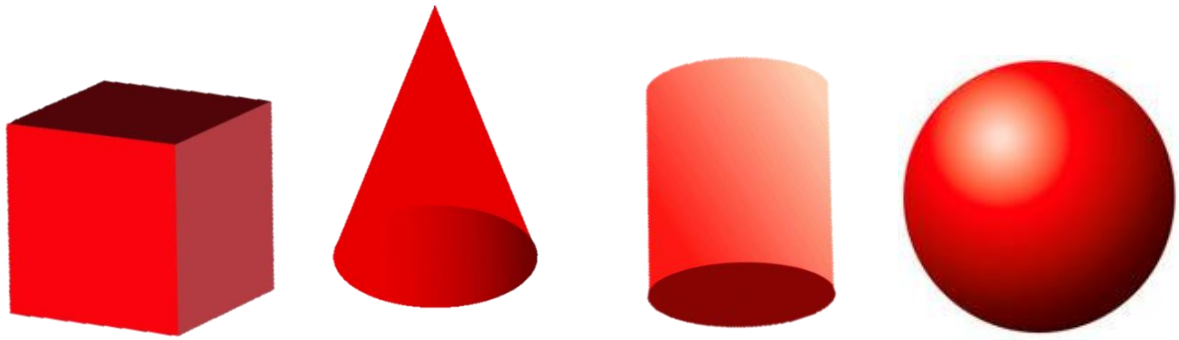


Figura 15. Paleta con las Figuras que se pueden insertar en el modelo.

A continuación, el usuario puede elegir la figura que desea señalándola con el dedo índice como se observa en la figura 16.

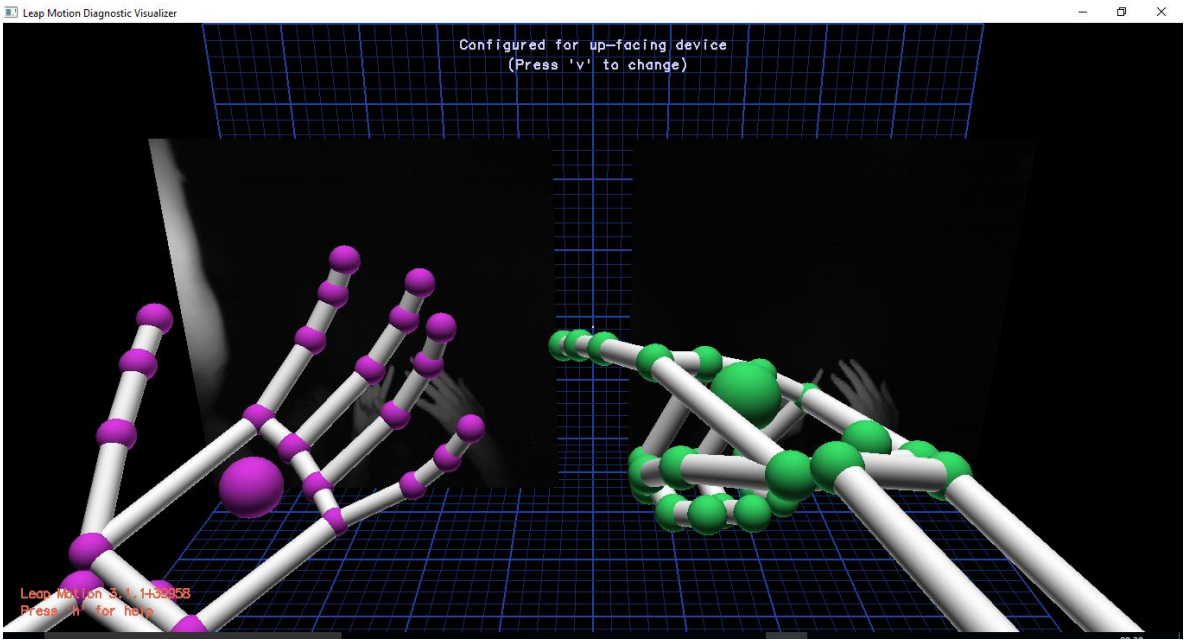


Figura 16. Gesto para la selección de una Nueva Figura.

En el escenario aparecerá la figura que se tocó con el dedo índice como se observa en la figura 17. Una vez que el usuario termino de hacer inserciones, es posible ocultar la paleta de figuras volteando la palma de la mano hacia abajo.

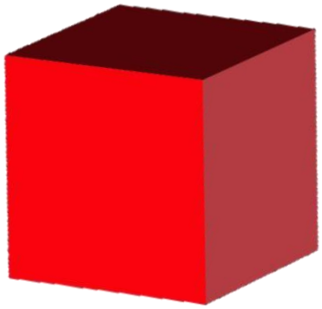


Figura 17. Ejemplo de inserción de un cubo.

Después de la creación de una figura es posible modificar sus atributos inmediatamente, es decir, seleccionar la posición, rotación, color y tamaño como se muestra en las siguientes secciones.

4.7.2 Cambiar Atributos

El primer paso es realizar un cambio a una figura ya existente que se encuentre seleccionada, para esto se utiliza el gesto de selección extendiendo el dedo índice hacia el frente y generando movimientos para realizar cambios, ya sea moviendo barras de desplazamiento o usando otras figuras.

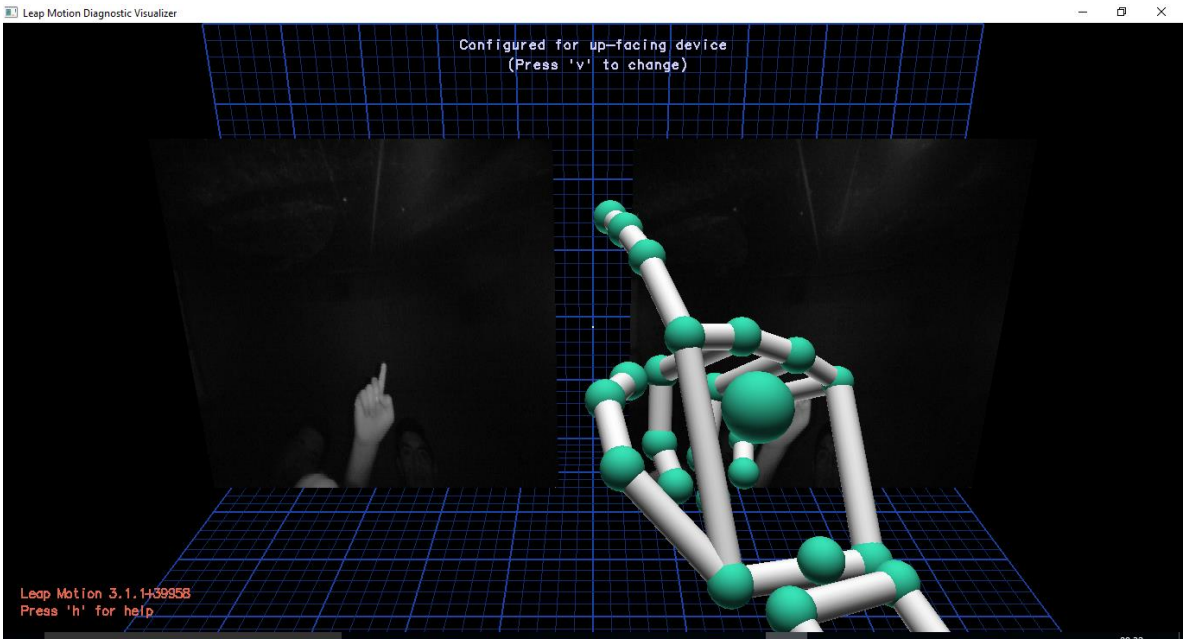


Figura 18. Gesto para la selección de una Figura

También es posible cambiar el color. Esto es mediante el uso de una paleta de colores como se muestra en la Figura 19:

Figura 19. Paleta de Colores

A continuación, solo es necesario elegir un color de la paleta y al tocarlo como se describió en la Figura 18, la figura básica que se encuentre seleccionada cambiará de color automáticamente como se muestra en la Figura 20:

.

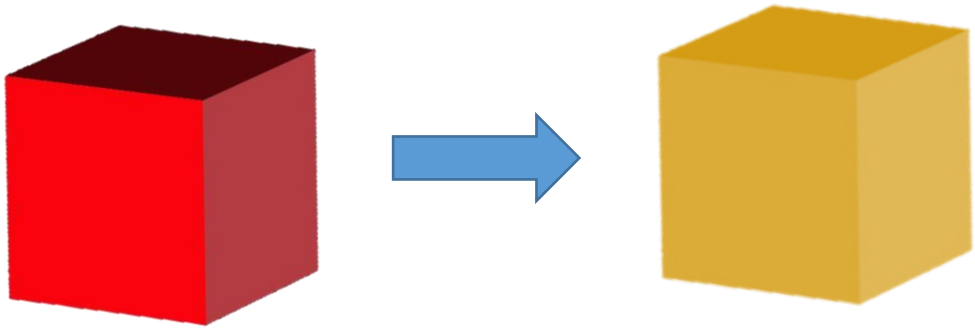


Figura 20. Resultado de cambiar el color de una figura.

4.7.3 Cambiar Posición

Ya una vez teniendo figuras dentro de nuestro escenario virtual, es posible cambiar su posición del plano por medio del gesto en la Figura 21, que consiste en la simulación de tomar un objeto juntando todos los dedos en un punto común simulando una “pinza”.

Figura 21. Gesto para Cambiar la Posición

Con este gesto se “sujeta” la imagen virtual y por medio de desplazar o rotar la mano, la figura modificara su posición o rotación respectivamente. Para liberar una figura solo es necesario separar los dedos y está quedará en la posición soltada.

4.8 Protocolo de Comunicaciones

Como se mencionó anteriormente, uno de los motivos por el cual se eligió Unity como motor gráfico para este proyecto fue debido a las herramientas de interconexión de red con las cuales cuenta ya que nos brindan la oportunidad de conectarnos a servidores virtuales a los cuales podemos acceder a través de Internet. Por medio de estas herramientas se puede tener distintos usuarios conectados a una aplicación y el intercambio de información necesario para cumplir los objetivos de este sistema.

En el sistema de red de Unity, las aplicaciones cuentan con un servidor y múltiples clientes. Cuando no se tiene un servidor dedicado, uno de los clientes juega el rol de servidor y es conocido como el “host”[21], como se ve en la Figura 22.

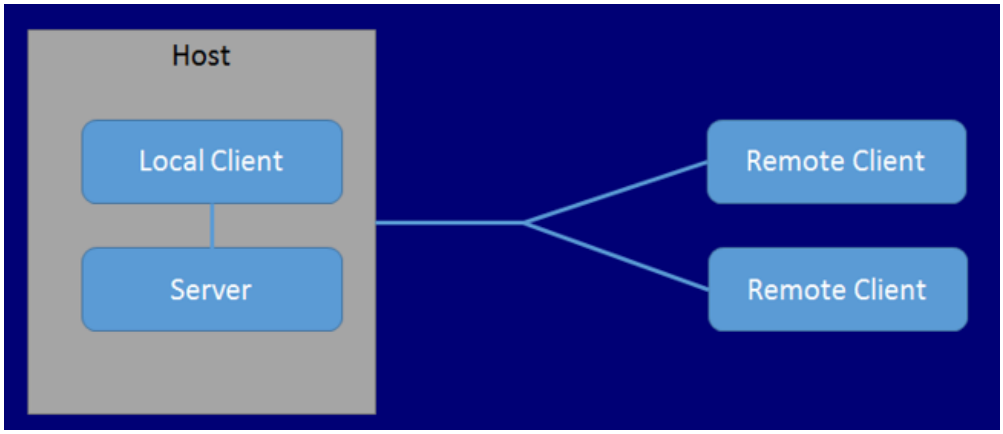


Figura 22. El host es servidor y cliente en el mismo proceso y los demás clientes se conectan a él como clientes locales.

El host es servidor y cliente en el mismo proceso, y éste usa un tipo especial de cliente conocido como Local Client, mientras los demás clientes son conocidos como Remote Client.

Cuando manejamos objetos en red en Unity, los objetos deben de generarse en el servidor y a partir de eso, éstos se generan en los clientes conectados. Estos objetos deben tener un ciclo de vida y principios de sincronización[22].

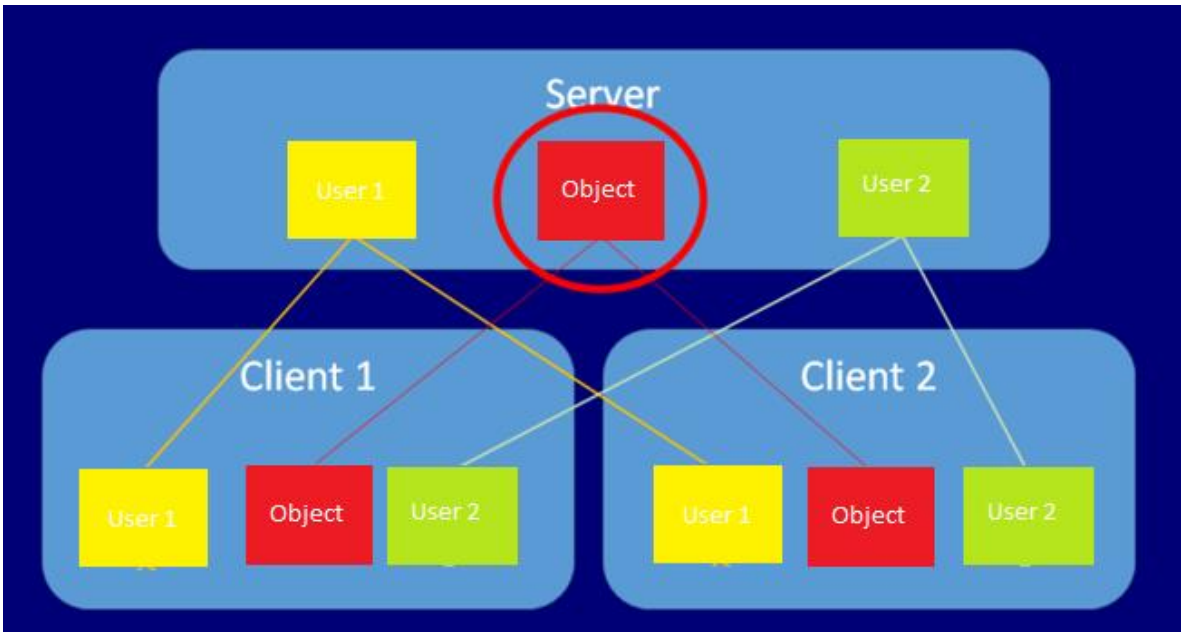


Figura 23. Un objeto generado en el servido se transmite simultáneamente a todos los clientes conectados.

Como se mencionó en los requerimientos del sistema, sólo un usuario puede modelar un objeto a la vez, cediendo el turno al siguiente cuando ha terminado. Esto se logra con la autorización de clientes para objetos no jugables. Esto permite que el dueño del objeto creado pase el manejo al siguiente usuario.

4.8.1 Photon Unity Networking

En el caso de aplicaciones de RV, se usó Photon Unity Networking, un paquete para multijugador, el cual ofrece opciones de autenticación entre los usuarios. Este *Asset*[1] se ofrece en dos distribuciones PUN y PUN+[23].

Figura 24. Asset para conexión de red, Photon

[1] Los Assets son distintas librerías que Unity nos ofrece desde su Tienda de aplicaciones con Scripts y Objetos prefabricados para implementarlo en nuestras aplicaciones.

PUN es una versión gratuita y nos ofrece la posibilidad de hasta tener hasta 20 usuarios concurrentes y un host, mientras que PUN+ ofrece hasta 100 usuarios concurrentes, un host y compatibilidad para usarse con iOS y Android. Por las características del proyecto solo se necesitó la versión gratuita.

PUN está disponible en la tienda de *assets* de Unity, al solo descargarlo tendremos acceso a sus librerías y a un servidor el cual tendrá que ser configurado para que nuestra aplicación pueda ser usada.

Como se muestra en la Figura 25, PUN nos permite utilizar un servidor en la nube y que se tiene que configurar para que pueda ser utilizado. Una aplicación que se conecte a PUN debe de tener todas las librerías, las claves de acceso, y un identificador.

La concurrencia en Photon se lleva a cabo por medio de *Rooms*, que es una conexión reservada en el servidor para que los usuarios que ingresen a ésta puedan compartir contenido entre ellos. PUN nos permite tener múltiples *Rooms* corriendo a la vez, por lo que se pueden tener distintos grupos separados trabajando a la vez.

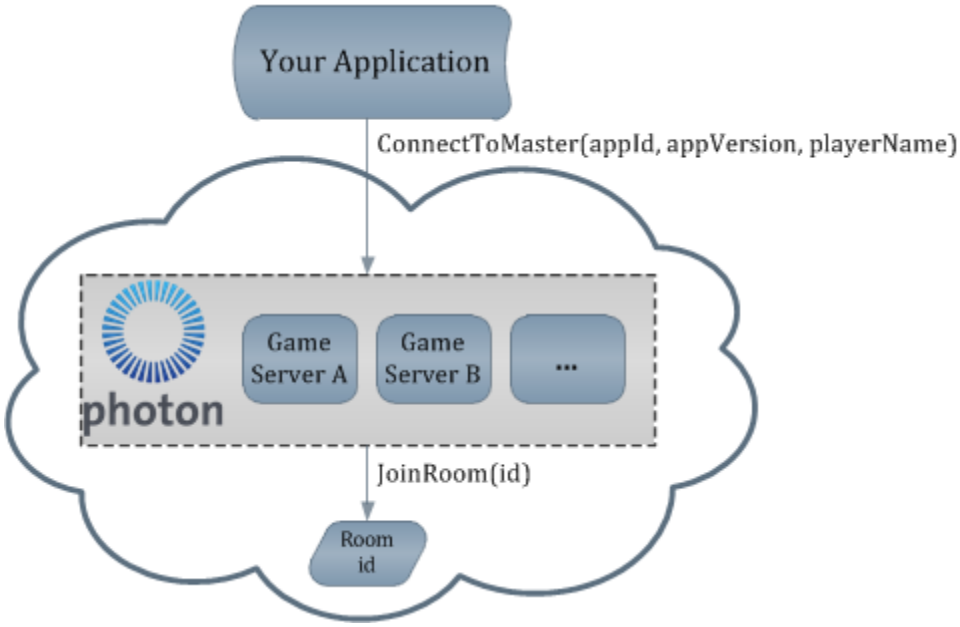


Figura 25. La aplicación local se conecta a Photon y de aquí puede crear y unirse a habitaciones.

Por medio de PUN podremos crear *Rooms* en donde los diferentes usuarios podrán acceder para interactuar. Estas interacciones funcionan por medio de enviar y recibir eventos específicos y de esa forma funciona toda la lógica de la aplicación. A todos los usuarios *Photon* los denomina *Players*, y el primero en

accesar será el encargado de crear una *Room*. Como se ve en la figura 26, el primer Player en acceder es el Player X, este es el encargado de crear la *Room* "1" donde posteriormente los siguientes usuarios, Y y Z accederán. Los tres usuarios podrán trabajar de manera concurrente en esta *Room*.

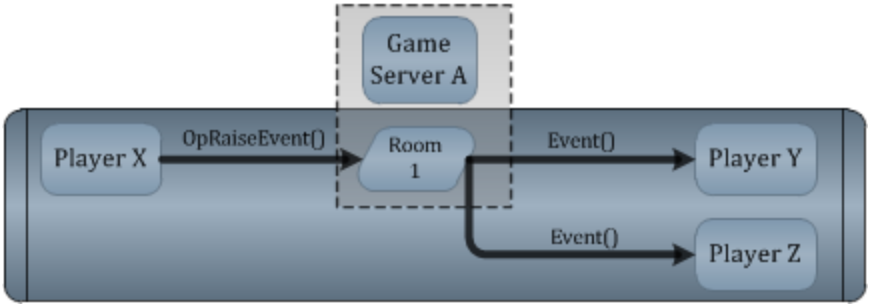


Figura 26. Para que los clientes interactúen unos con otros se usa un sistema de eventos.

Para establecer una conexión con el servidor de *Photon* es necesario configurar los siguientes parámetros:

- Tipo de Host. Se escoge si se usará el servidor de *Photon*, un servidor dedicado o sin conexión.
- Región: La región geográfica más adecuada de acuerdo a tu localización. Para este proyecto se usará la región US.
- AppId. Identificador para poder acceder a nuestro servidor *Photon* que se encuentra en la nube.
- Protocolo. Se escoge entre dos protocolos, TCP y UDP.

4.8.4 Coordinación entre Usuarios

Como se mencionó en el Marco Teórico, un Software Colaborativo tiene que incluir las tres C's (Comunicación, Cooperación y Coordinación). La primera se logrará mediante la instalación y configuración de las librerías de Photon. Para la segunda y la tercera será necesario la creación de un algoritmo para saber cuando bloquear y liberar a un usuario. Esto se logrará mediante la utilización del comando *[Init Message]* para enviar mensajes personalizados usando cadenas de texto. Estos mensajes serán recibidos por todos, pero solo habrá respuesta para quien este dirigido y se almacenará en una cola para saber el orden de los turnos y los bloqueos.

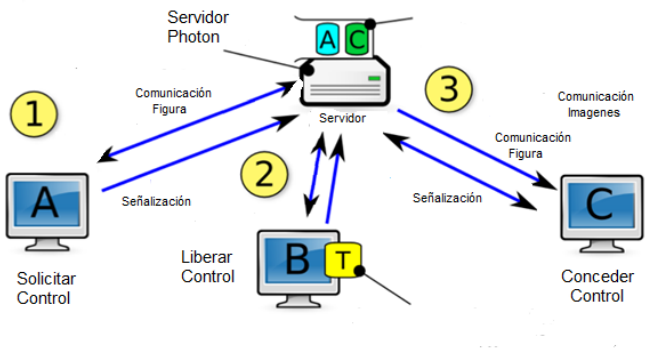
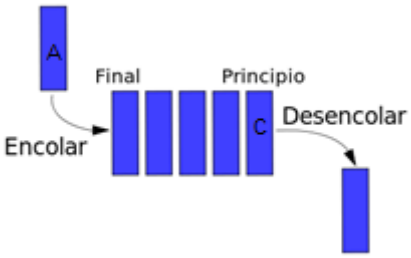


Figura 27. Algoritmo de Coordinación y Cooperación

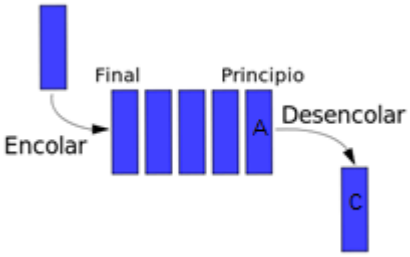
El primer usuario en ingresar a la aplicación tendrá el rol de administrador y automáticamente tendrá el rol de Modelador por lo cual obtendrá también el control del ambiente virtual. Los siguientes que accedan al programa serán sólo observadores, por lo cual estarán bloqueados para utilizar el ambiente.

Para que un usuario pida un turno se deberá seguir un algoritmo para la liberación y petición de turnos. Cada instancia tendrá su propia cola y almacenará los turnos en el orden los vaya recibiendo.

1) B tiene el control del ambiente. A está bloqueada. A solicita el control del ambiente virtual. A es encolada.



2) B libera el ambiente. Desencolamos. B se bloquea.



3) C obtiene el control. C tiene el control del ambiente.

Como se dijo previamente, los mensajes serán recibidos por todos, pero solo se bloqueará y liberará dependiendo del contenido del mensaje y a quien va dirigido.

4.9 Protocolo de transporte

Por último, Photon nos permite elegir el protocolo de transporte a utilizar, por lo cual, se describirá de forma general cada uno:

- TCP (Transmission-Control-Protocol, en español Protocolo de Control de Transmisión) es de los protocolos fundamentales en Internet. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. TCP da soporte a muchas de las aplicaciones más populares de Internet, incluidas HTTP, SMTP, SSH y FTP. TCP es un protocolo de comunicación orientado a conexión y fiable del nivel de transporte.
- User Datagram Protocol (UDP) es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Su uso principal es para protocolos como DHCP, BOOTP, DNS y demás protocolos en los que el intercambio de paquetes de la conexión/desconexión son mayores, o no son rentables con respecto a la información transmitida, así como para la transmisión de audio y vídeo en tiempo real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

El protocolo UDP nos ofrece mejores características para este proyecto ya que buscamos que al modelar figuras, los cambios sean perceptibles por los demás con el menor retardo posible. De igual forma, la información manejada puede llegar en el orden que sea, usar TCP ocasionaría cierto lag en nuestra aplicación.

De igual forma, para el envio de sentencia de control se utilizó la comunicación por multidifusión, siendo el que mejor se adaptaba para esto el protocolo UDP

4.8.3 Protocolo Binario PUN

El servidor *Photon* y los clientes conectados a él usan un protocolo binario altamente optimizado para comunicarse que es usado por la aplicación a través de sus librerías paras que solo sea necesario llamar los distintos métodos para crear la lógica de la comunicación. Las capas del protocolo se muestran en la Figura 27:

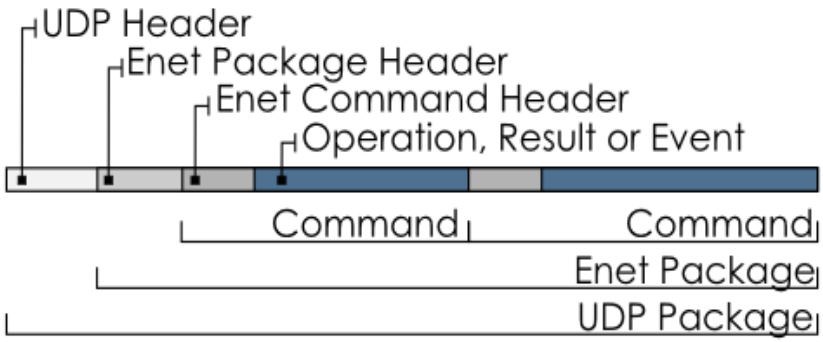


Figura 28. Capas del Protocolo Binario PUN

El protocolo binario de *Photon* se organizan en varias capas. En su nivel más bajo, UDP se utiliza para transportar los mensajes que se desea enviar. Dentro de éste se encuentran distintos encabezamientos que reflejan las propiedades que se esperan de *Photon*: confiabilidad, secuenciación, agregación de mensajes, sincronización de tiempo, etc. Cualquier paquete UDP contiene una cabecera Enet y al menos un comando Enet. Cada comando Enet lleva nuestros mensajes: una operación, resultado o evento. Éstos a su vez consisten en el encabezado de la operación y los datos que proporcionó.

La Tabla 2 muestra los diferentes comandos Enet:

Nombre	Enviado Por	Descripción
Conectar [Connect]	Cliente	Se envía al establecer la conexión
Verificar Conexión [Verify Connect]	Servidor	Se envía en respuesta a la conexión establecida
Iniciar Mensaje [Init Message]	Cliente	Se envía a petición del cliente durante la conexión
Iniciar Respuesta [Init Response]	Servidor	Se envía en respuesta al mensaje enviado por el cliente
Desconectar [Disconnect]	Ambos	Se envía al finalizar la conexión

Tabla 2. Comandos Enet para el envio de mensajes

Estos comandos se utilizaron para establecer conexiones y el envío de mensajes de control entre los usuarios y el servidor.

Capítulo 5: **Diseño del Sistema**

El Sistema se programó utilizando el paradigma orientado a objetos, por lo que su diseño se presenta mediante un diagramado UML, mismos que permiten visualizar y documentar los módulos que lo conforman.

5.1 Diagrama de Casos de Uso

Los casos de uso son estructuras que ayudan a los analistas a determinar la forma en que se utilizará el Sistema. Los usuarios reflejados en estos diagramas no son únicamente personas, si no también dispositivos y sistemas externos interactuando. Dado que el comportamiento del Sistema depende del tipo de datos que recibe como entrada, se elaboró un diagrama en el que se encuentran reflejados los casos de uso de datos ingresados.

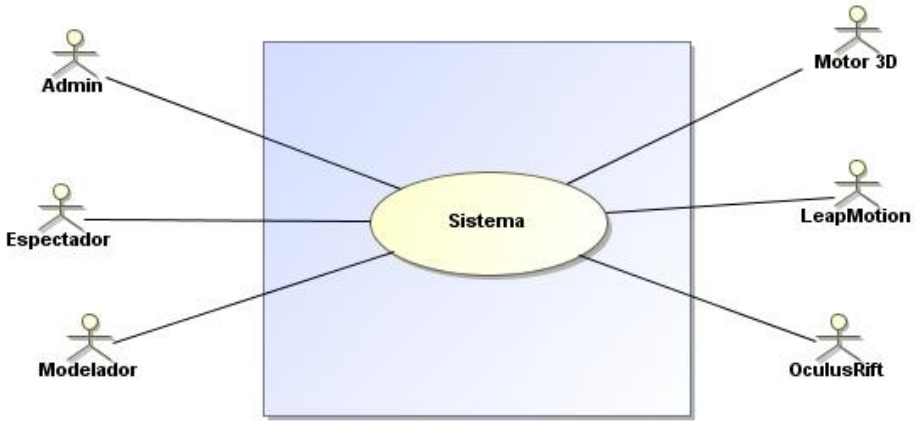


Diagrama 1. Actores en el Sistema

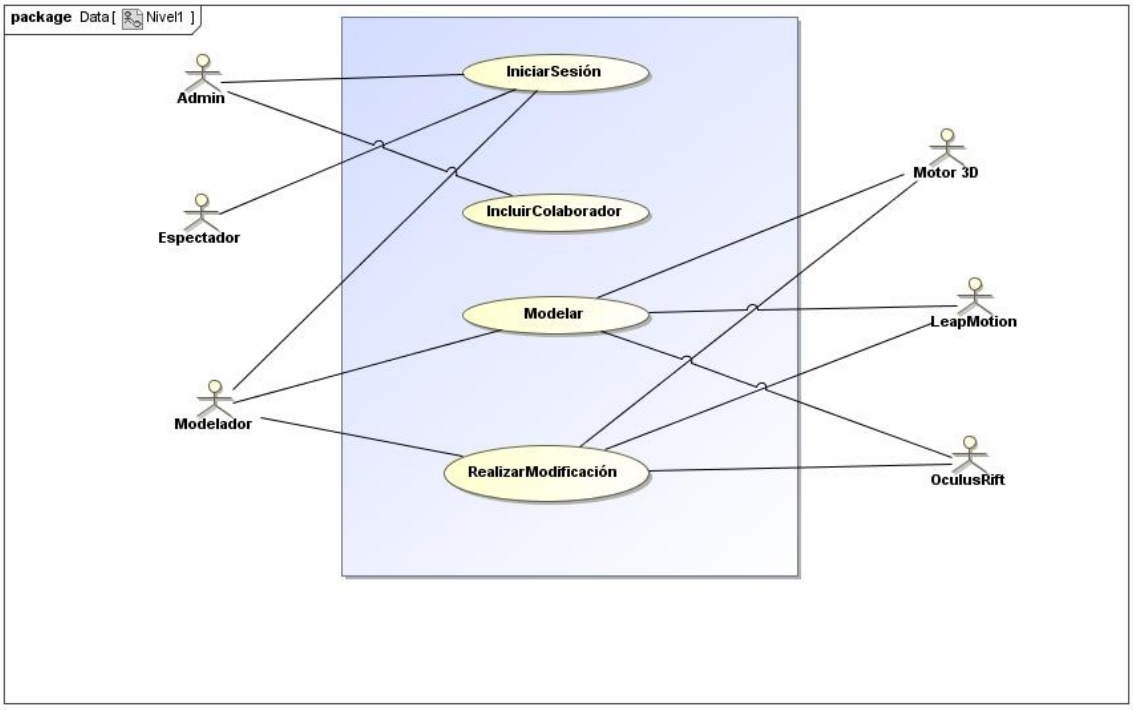


Diagrama 2. Interacción entre los actores y los casos de uso

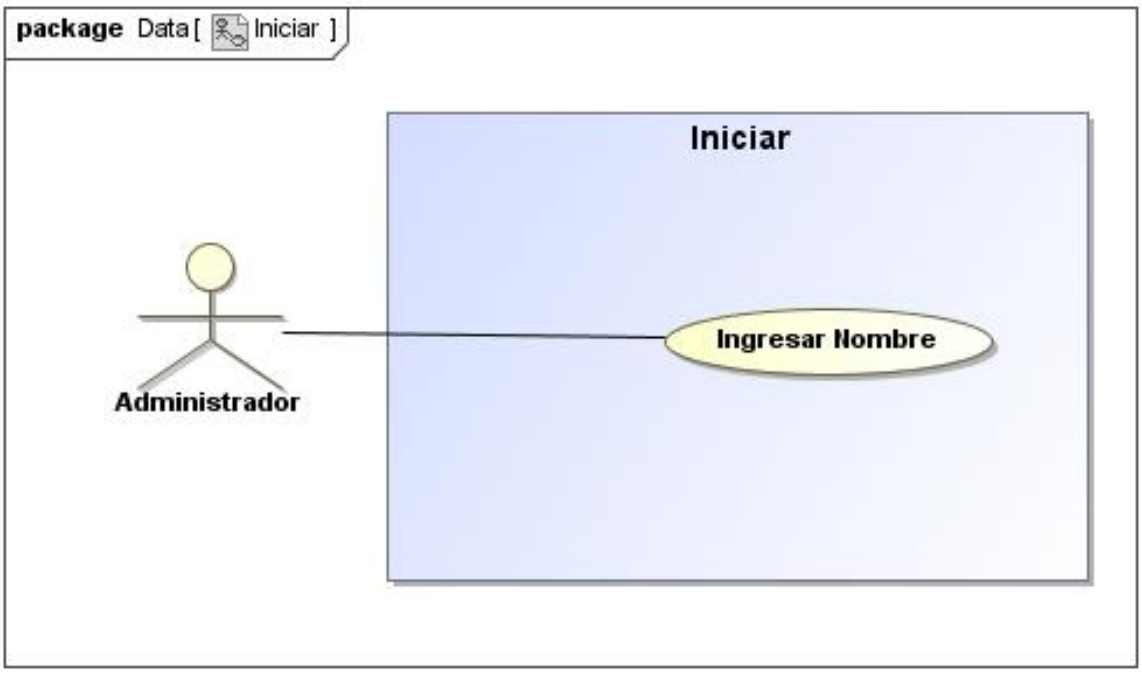


Diagrama 3. Caso de uso ingresar nombre

Caso de uso	Iniciar
Descripción	Un usuario que funge como administrador inicia al framework mediante el ingreso de su nombre.
Actores	Administrador.
Precondiciones	Se muestra pantalla de bienvenida en la interfaz. No puede haber ningún otro usuario en el sistema.
Flujo normal	1. El administrador abre la aplicación de escritorio. 2. El administrador ingresa su nombre. 3. El sistema da ingreso al administrador.
Flujo alternativo	El administrador no puede ingresar al sistema debido a que ya hay un administrador que ha ingresado previamente. 1. Se muestra una pantalla de error.
Postcondiciones	El framework entra al entorno de desarrollo en espera de acciones del administrador. El usuario entra en modo modelador.

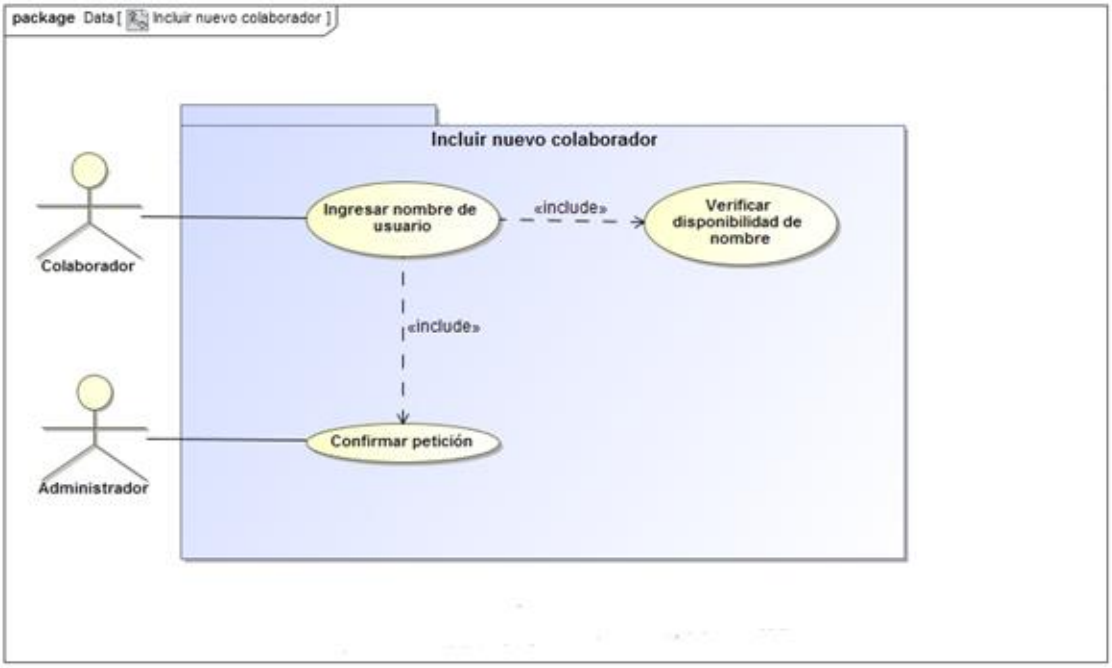


Diagrama 4. Caso de uso incluir nuevo colaborador

Caso de uso	Incluir nuevo colaborador
Descripción	Un usuario que funge como colaborador accesa al framework mediante el ingreso de su nombre, se checa la disponibilidad del nombre y el administrador confirma la petición.
Actores	Colaborador, Administrador.
Precondiciones	Se muestra pantalla de bienvenida en la interfaz. Un administrador ya debe haber accesado al framework.
Flujo normal	1. El colaborador abre la aplicación de escritorio. 2. El colaborador ingresa su nombre. 3. El sistema revisa que no haya en el sistema un colaborador con el mismo nombre. 4. El administrador recibe una notificación de que un nuevo colaborador intenta accesar al framework. 5. El administrador acepta al nuevo colaborador.
Flujo alternativo	El colaborador no puede ingresar al sistema debido a que el nombre de usuario que ingreso ya está ocupado. 1. Se muestra una pantalla de error. El colaborador no puede ingresar al sistema debido a que el administrador negó su petición de acceso. 2. Se muestra una pantalla de acceso negado.

Postcondiciones	El framework entra al entorno de desarrollo en espera de acciones de los usuarios. El colaborador entra en modo espectador.

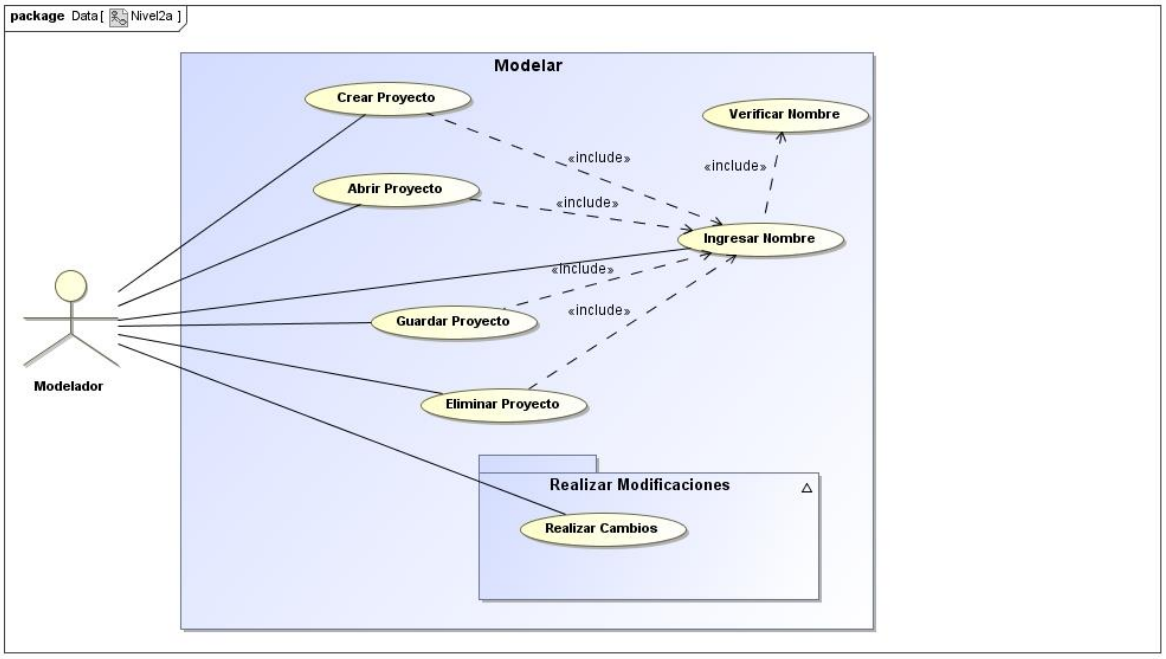


Diagrama 5. Caso de uso modelar

Caso de uso	Modelar
Descripción	Un usuario que este como colaborador en el framework abre o crea un nuevo proyecto, en este es capaz de realizar modificaciones, después de terminar puede guardar el proyecto. De igual forma puede eliminar proyectos ya existentes.
Actores	Colaborador.
Precondiciones	Un colaborador debe haber accesado al framework. Se muestra pantalla de entorno de desarrollo en la interfaz.
Flujo normal	1. El colaborador abre un nuevo proyecto. a. El colaborador ingresa el nombre del proyecto. b. El sistema verifica que no haya otro proyecto con el mismo nombre. c. El colaborador selecciona la ubicación es

	donde se guardara el proyecto. d. El sistema verifica la ubicación este libre. 2. El colaborador abre un proyecto existente. a. El colaborador selecciona un proyecto existente. 3. El colaborador ingresa a la opción de hacer modificaciones sobre el proyecto. a. El framework verifica el status del colaborador para saber que flujo seguir. 4. El colaborador termina de realizar modificaciones sobre el proyecto y activa la opción de guardar el proyecto. 5. El colaborador decide eliminar un proyecto ya abierto.
Flujo alternativo	El colaborador al abrir un nuevo proyecto ingresa un proyecto ya existente. 1. Se abre la opción de sobrescribir proyecto. El colaborador selecciona una opción no válida para guardar el proyecto. 1. Se abre una ventana de error. El colaborador no puede ingresar al sistema debido a que el administrador negó su petición de acceso. 2. Se muestra una pantalla de acceso negado.
Postcondiciones	El entorno de desarrollo queda en espera de acciones de los usuarios.

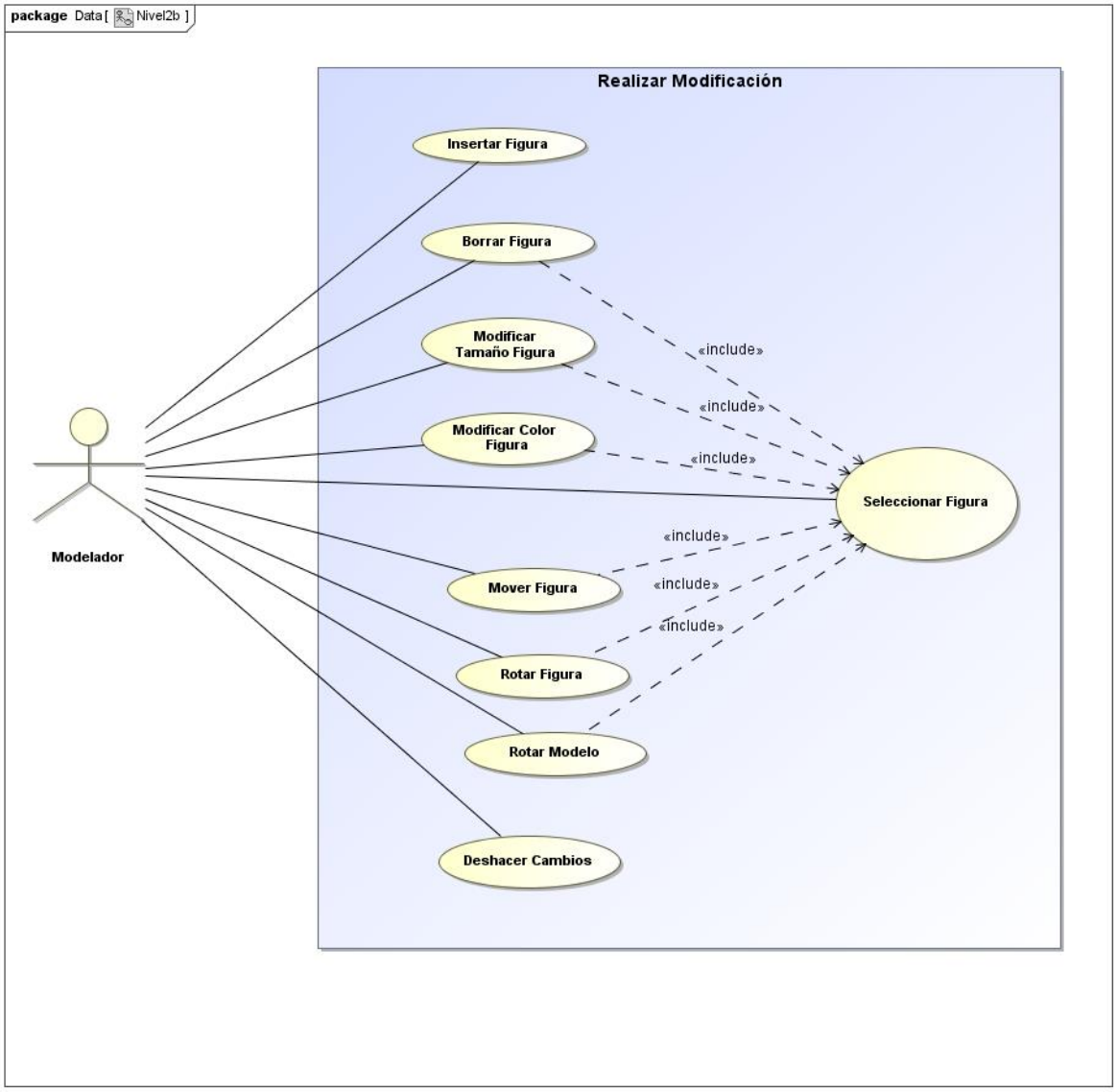


Diagrama 6. Caso de uso realizar modificaciones

Caso de uso	Realizar Modificación
Descripción	Un usuario que funge como modelador realiza modificaciones a un proyecto abierto.
Actores	Modelador.
Precondiciones	Un modelador se encuentra en el entorno de desarrollo e ingresa la opción de realizar modificaciones sobre el proyecto.
Flujo normal	1. El modelador ingresa a la opción de hacer modificaciones sobre el proyecto. 2. El modelador inserta una figura al entorno.

	3. El modelador puede realizar las siguientes acciones sobre la figura ya insertada: a. Borrar la figura. b. Modificar el tamaño de la figura. c. Modificar el color de la figura. d. Cambiar la posición de la figura. e. Rotar la figura. 4. El modelar inserta una nueva figura al entorno. 5. El entorno combina las figuras. 6. El modelador realiza acciones sobre el nuevo modelo combinado. 7. El modelador realiza acciones sobre cada figura por individual. 8. El modelador revierte un cambio realizado. 9. El entorne revierte el cambio y genera la figura previa. 10. El modelador elimina una figura por individual. 11. El modelador elimina todo el modelo.
Flujo alternativo	Al modelador no se le permite insertar figuras. 1. Se abre una ventana diciendo que debe pedir turno. Al modelador no se le permite realizar cambio sobre la(s) figura(s) en el entorno. 1. Se abre una ventana diciendo que debe pedir turno. Al modelador no se le permite deshacer cambios. 1. Se abre una ventana diciendo que debe pedir turno. 2. Se abre una ventana diciendo que el administrador no permitió deshacer el cambio. Al modelador no se le permite eliminar figura(s). 1. Se abre una ventana diciendo que debe pedir turno. 2. Se abre una ventana diciendo que el administrador no permitió eliminar la(s) figura(s).
Postcondiciones	El entorno de desarrollo queda en espera de acciones de los usuarios.

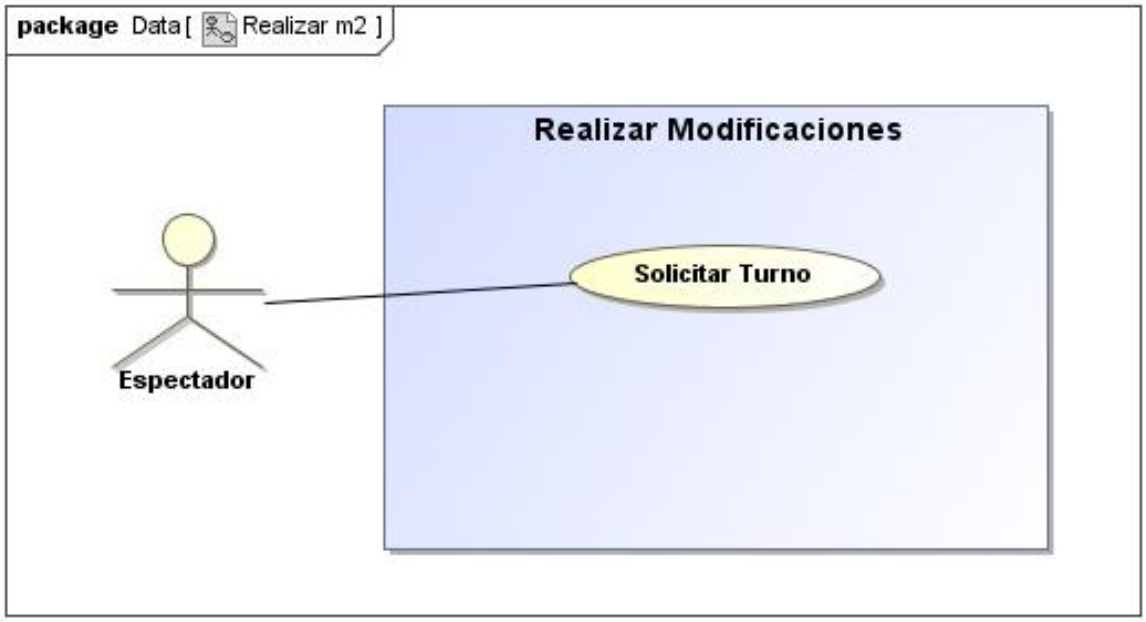


Diagrama 7. Caso de uso solicitar turno

Caso de uso	Solicitar Turno
Descripción	Un usuario que funge como espectador solicita turno para realizar modificaciones.
Actores	Espectador.
Precondiciones	Un modelador se encuentra en el entorno de desarrollo e ingresa la opción de realizar modificaciones sobre el proyecto.
Flujo normal	1. El espectador ingresa a la opción de hacer modificaciones sobre el proyecto. 2. Se abre una ventana diciendo que debe pedir turno. 3. El espectador pide turno. 4. El modelador acepta la petición.
Flujo alternativo	El modelador niega el acceso al espectador. 1. Se abre una ventana diciendo que el modelador negó el acceso.
Postcondiciones	El usuario entra en modo modelador. El Framework entra al entorno de desarrollo en espera de acciones del nuevo modelador.

5.2 Diagrama de Clases

Los diagramas de clases son diagramas de estructura estática que muestran las clases del sistema y sus interrelaciones (incluyendo herencia, agregación, asociación, etc.). Los diagramas de clase son el pilar básico del modelado con UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar cómo puede ser construido (diseño).

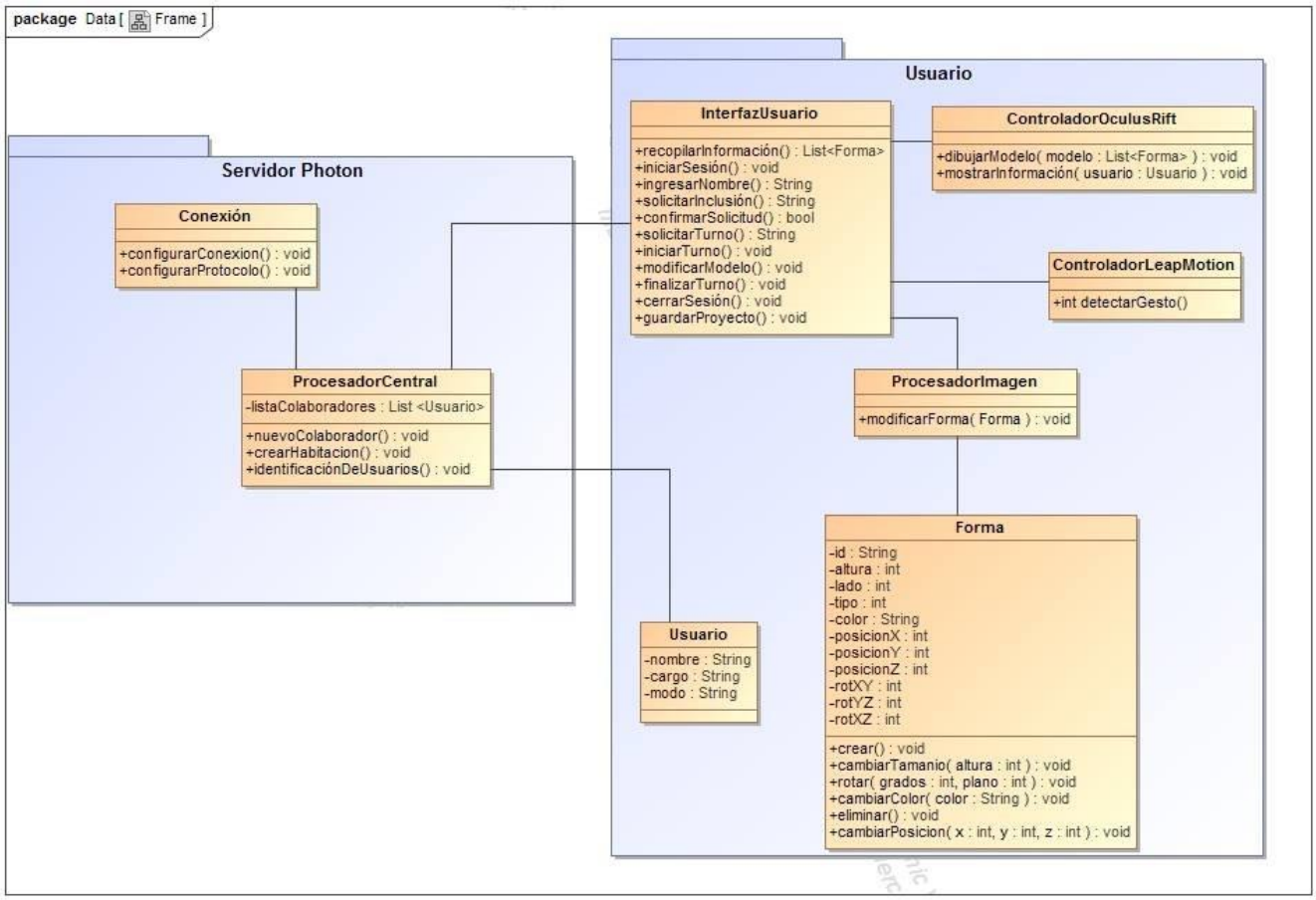


Diagrama 8. Diagrama de Clases

5.3 Diagramas de Secuencias

Un diagrama de secuencias muestra la interacción de un conjunto de objetos de una aplicación a través del tiempo, en el cual se indicarán los módulos o clases que formaran parte del programa y las llamadas que se hacen cada uno de ellos para realizar una tarea determinada, por esta razón permite observar la perspectiva cronológica de las interacciones. Es importante recordar que el diagrama de secuencias se realiza a partir de la descripción de un caso de uso.

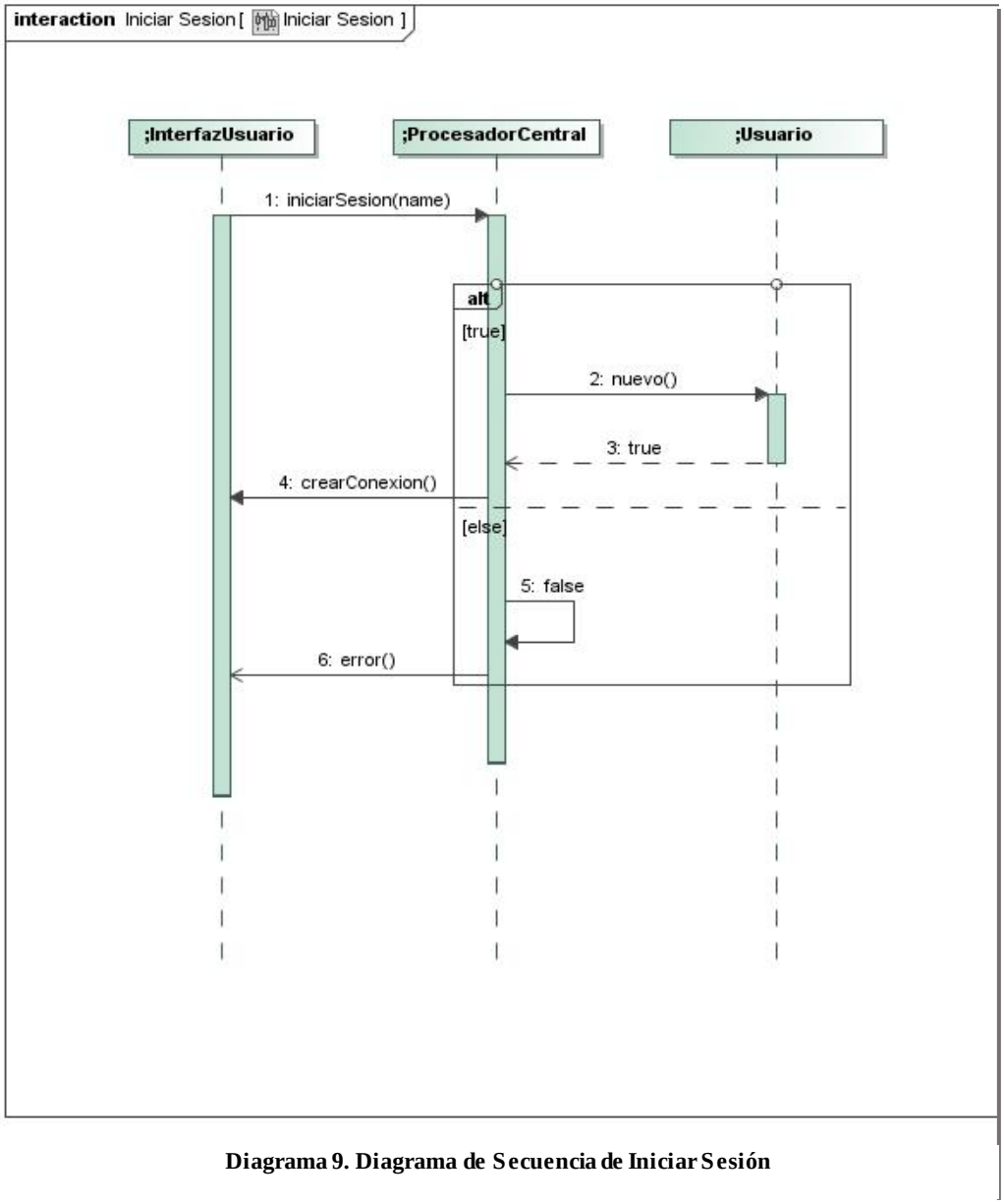


Diagrama 9. Diagrama de Secuencia de Iniciar Sesión

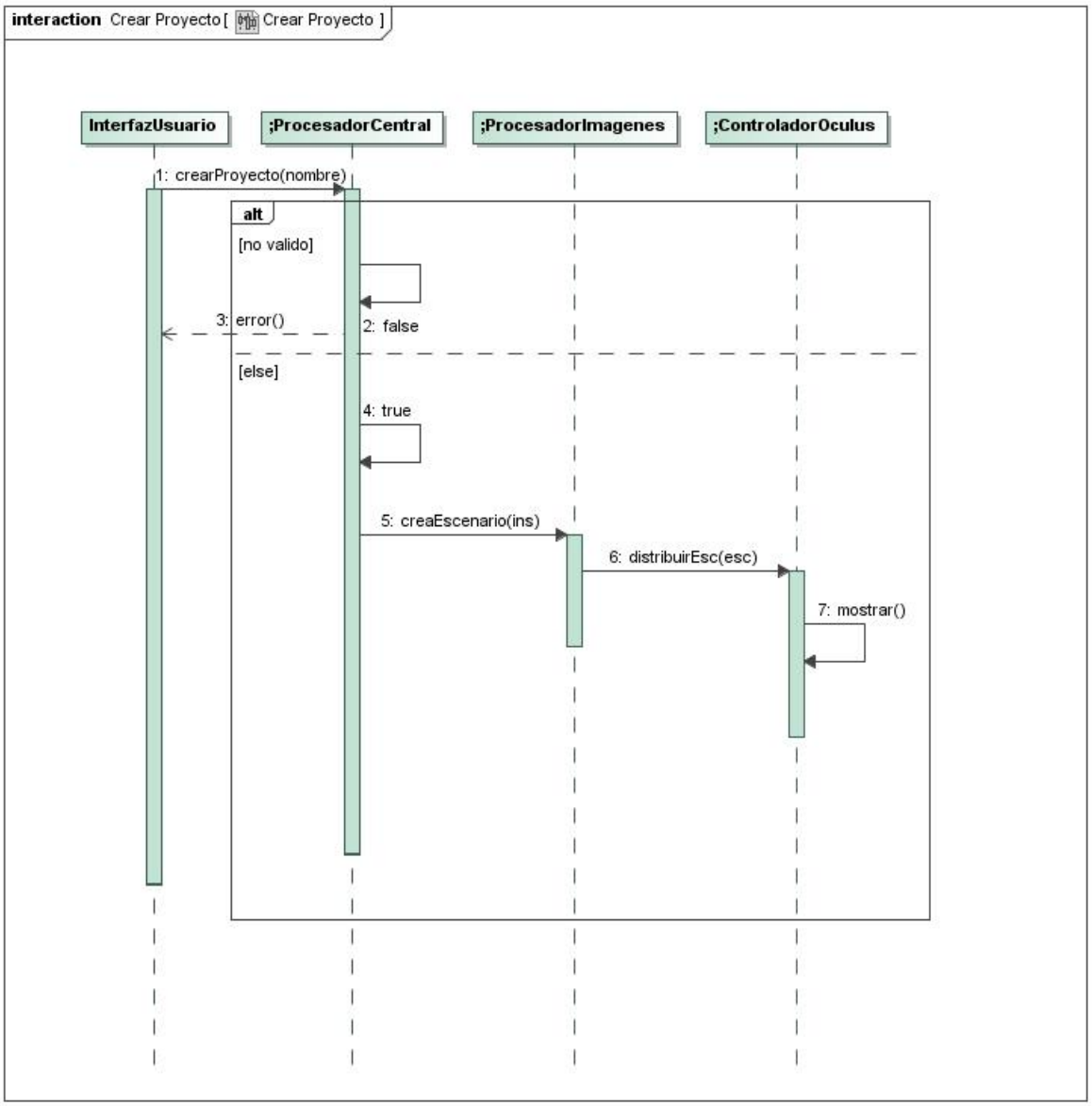


Diagrama 10. Diagrama de Secuencia de Crear Proyecto

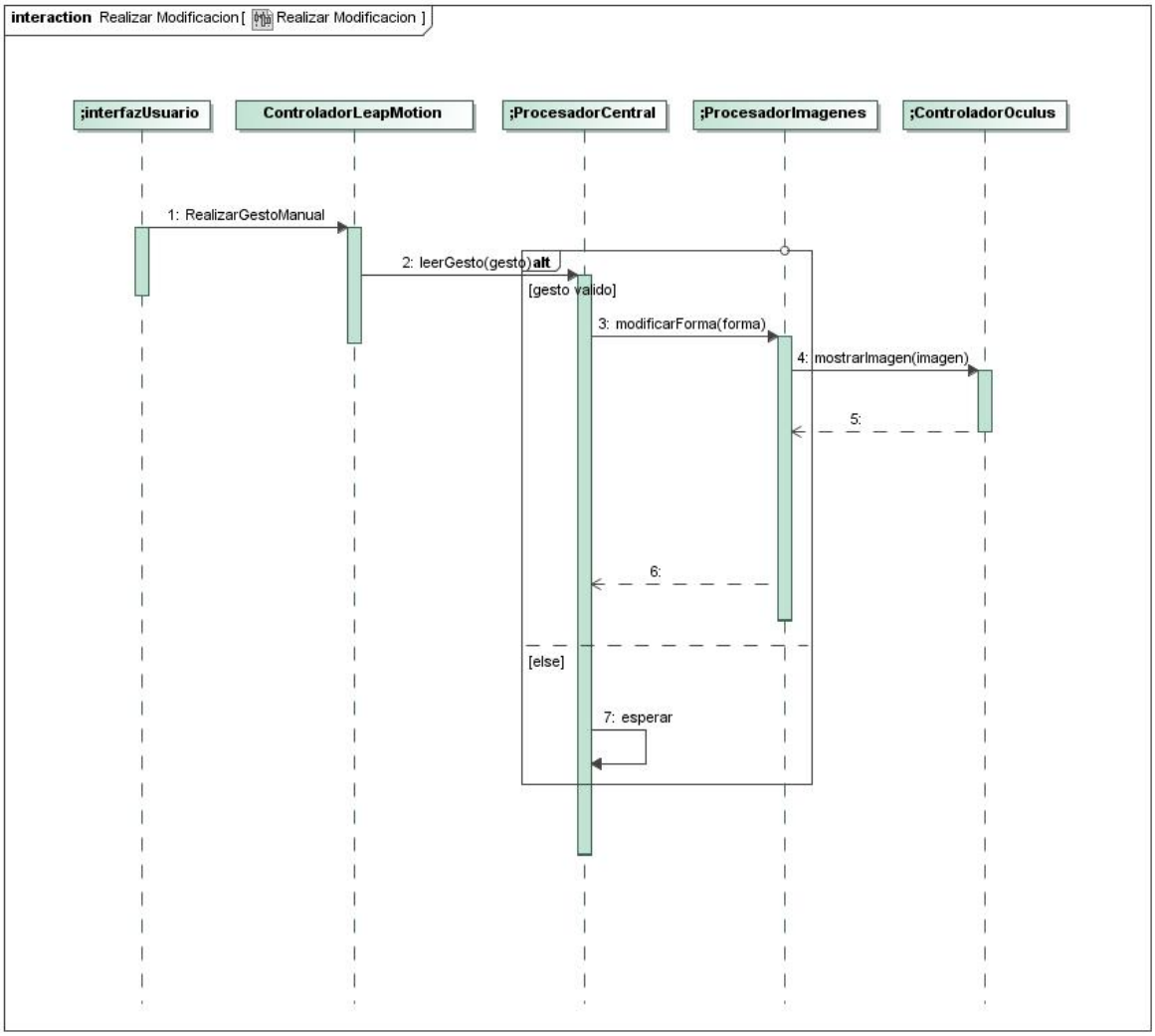


Diagrama 11. Diagrama de Secuencia de Realizar Modificaciones

5.4 Diagramas de Actividades

Un diagrama de Actividad demuestra la serie de actividades que deben ser realizadas en un uso-caso, así como las distintas rutas que pueden irse desencadenando en el uso-caso.

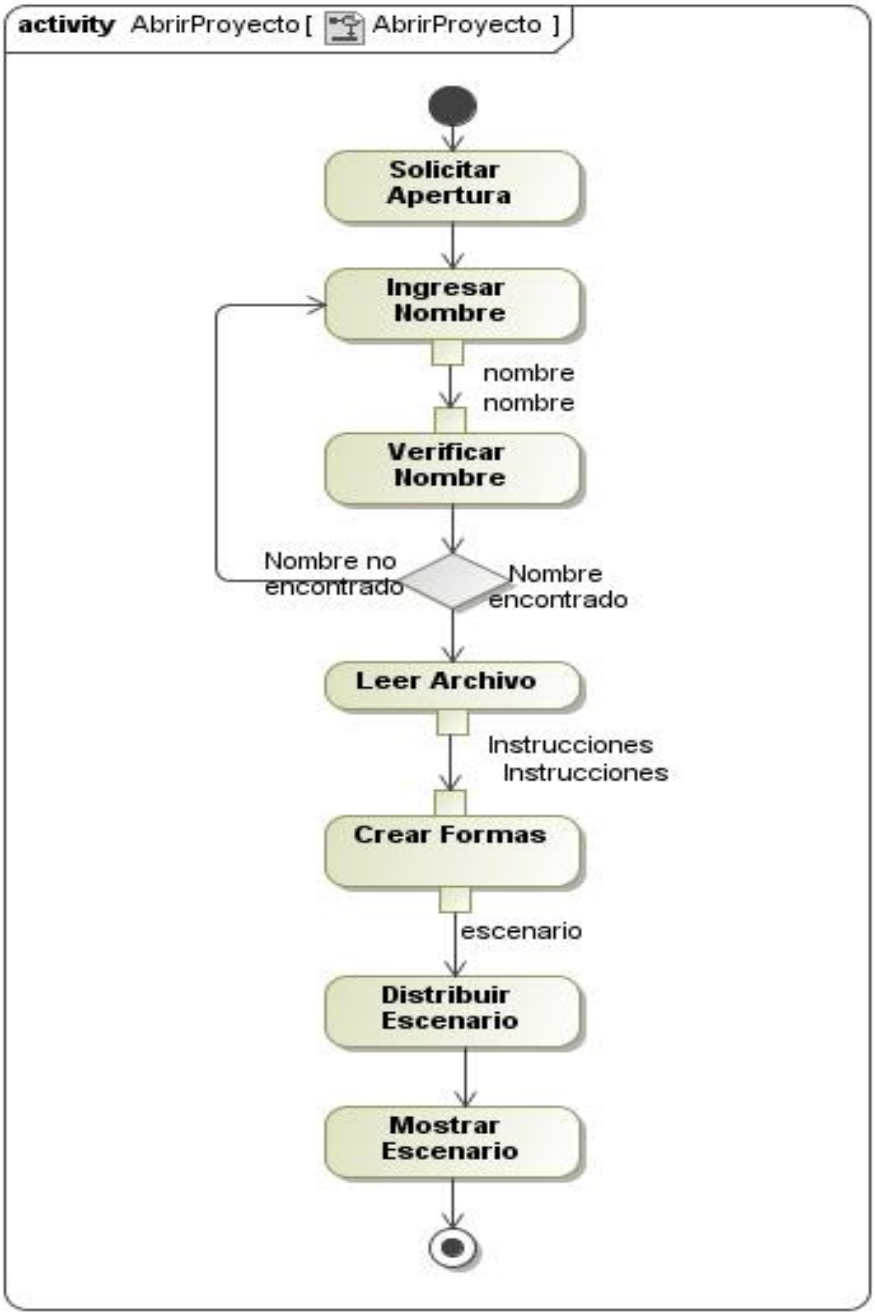


Diagrama 12. Diagrama de Actividades de Abrir Proyecto

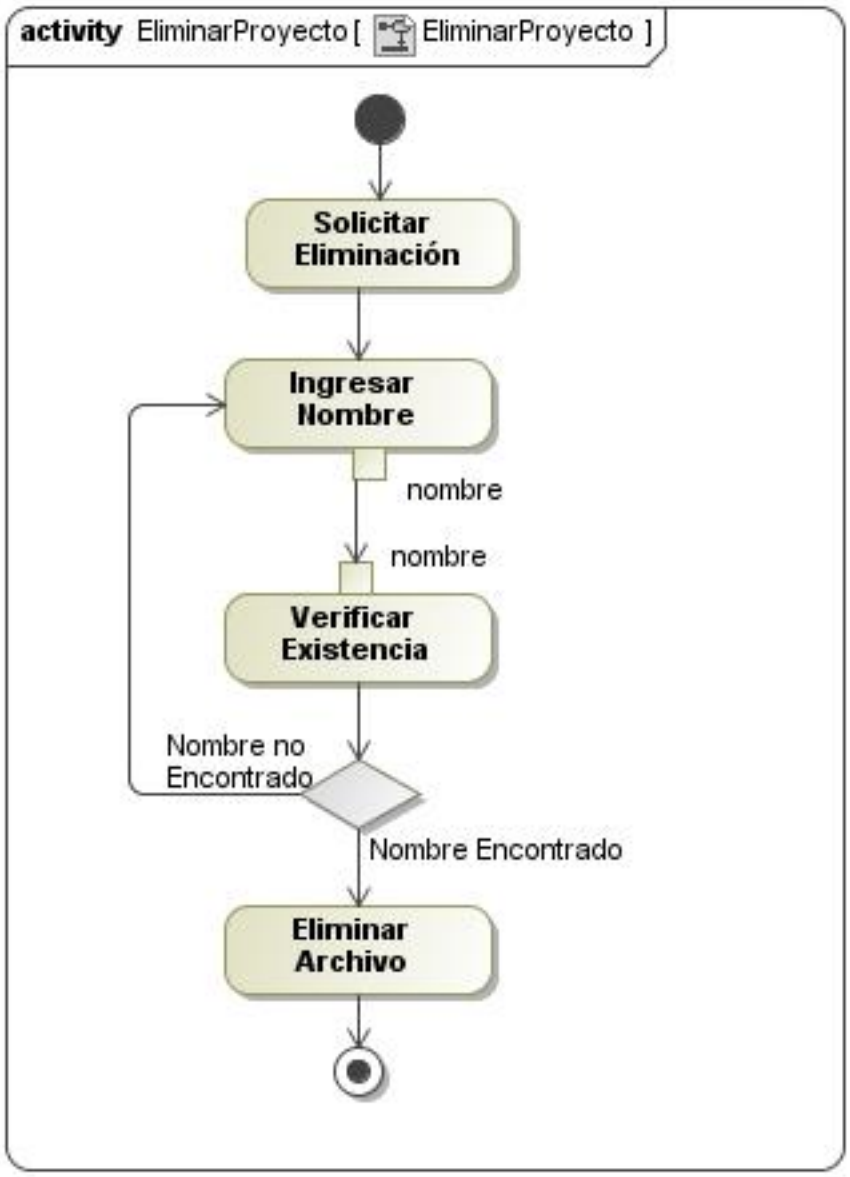


Diagrama 13. Diagrama de Actividades de Eliminar Proyecto

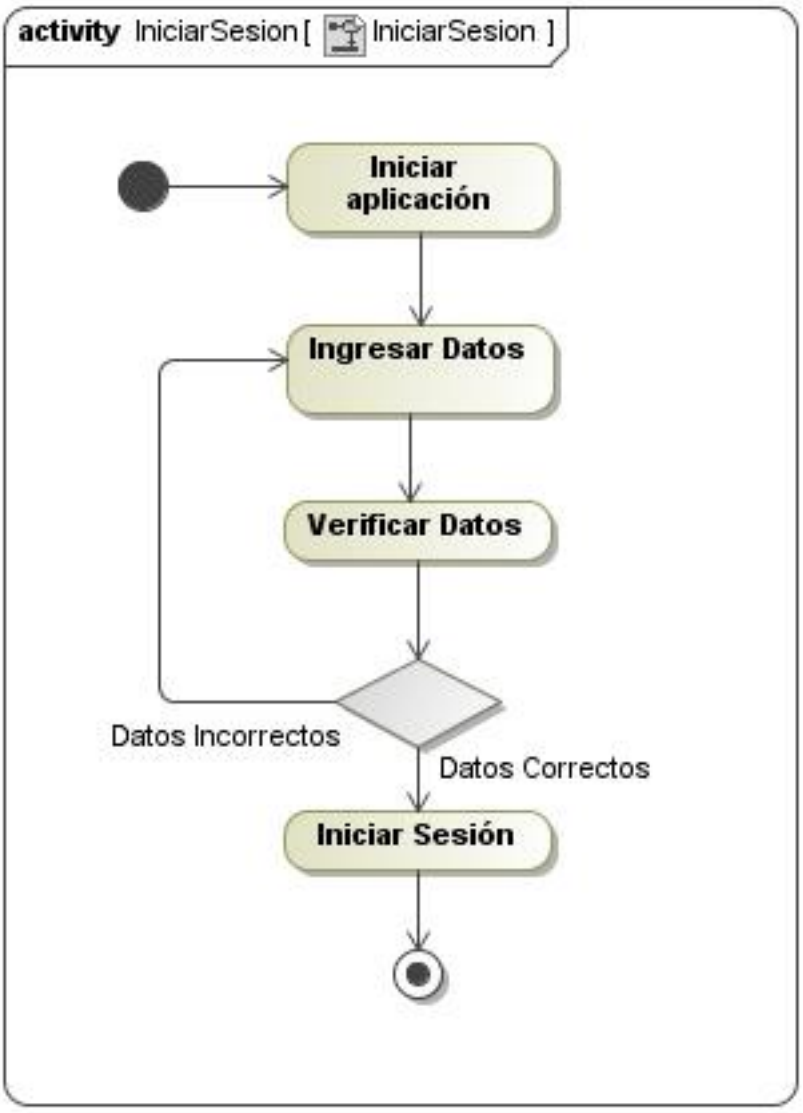


Diagrama 14. Diagrama de Actividades Iniciar Sesión

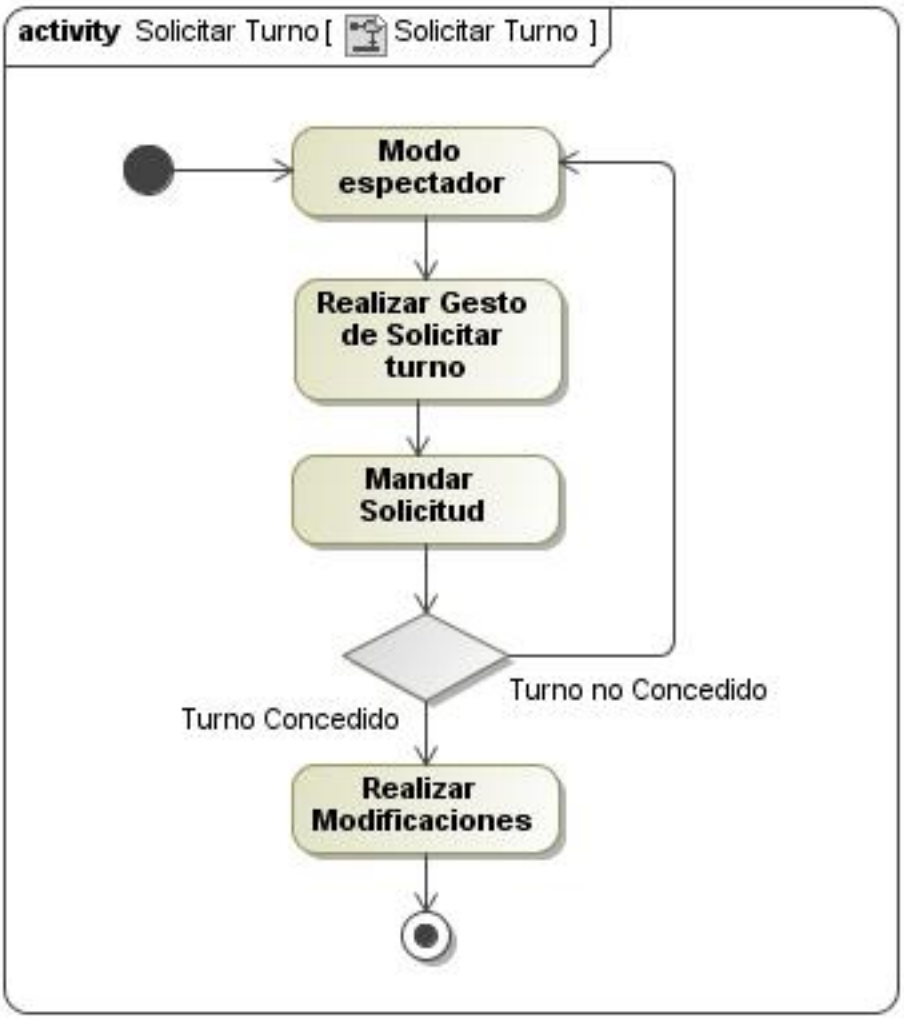


Diagrama 15. Diagrama de Actividades de Solicitar Turno

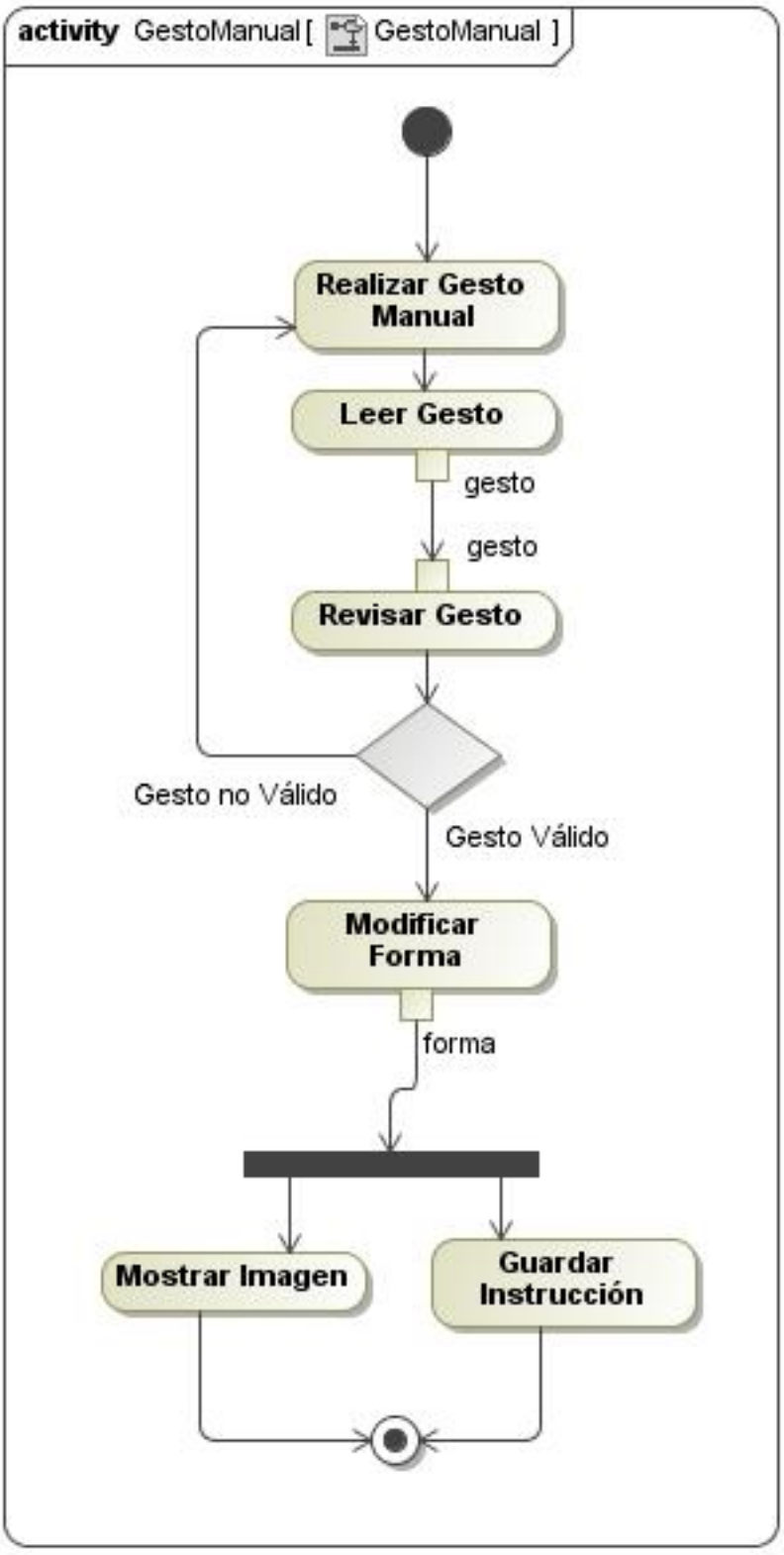


Diagrama 16. Diagrama de Actividades de Realizar Modificaciones

Capítulo 6: **Implementación**

En este capítulo se describe la implementación del desarrollo gráfico en Unity, la visualización en el dispositivo de Realidad Virtual Oculus Rift, el desarrollo de gestos en el sensor Leap Motion, la integración de estos tres componentes, así como la comunicación entre las entidades a través de Photon, y el sistema de turnos y de Framework dado por la lógica del negocio.

6.1 Desarrollo Gráfico

6.1.1 Instalación Unity

Como se mencionó en el apartado de Análisis de Requerimientos, el motor gráfico que se decidió utilizar fue Unity. Parte de la implementación fue descargar el software de desarrollo y aprender a utilizarlo.

Primeramente, ingresaremos a la página oficial y elegimos que edición de Unity descargar, a lo cual seleccionaremos la “Personal Edition” la cual no tiene ningún costo.

Figura 29. Selección de plan Unity

Iniciamos la instalación del motor gráfico de Unity después de descargarlo de la página oficial.

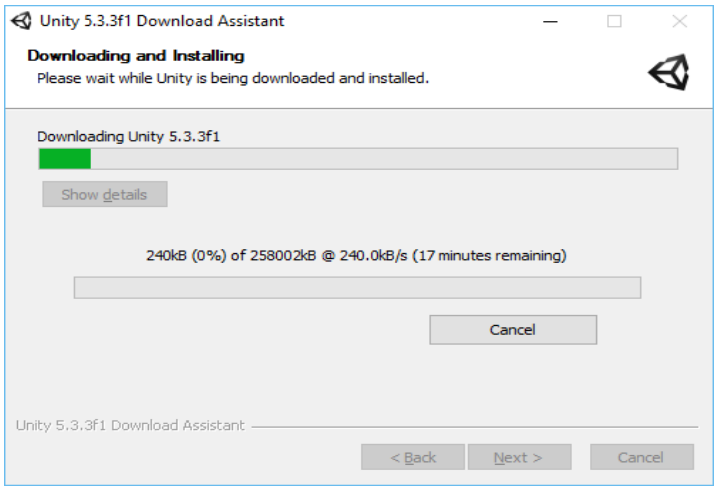


Figura 30. Flujo de instalación Unity

Después de haber instalado Unity en nuestro equipo, es necesario la creación de una cuenta para poder usar el motor gráfico y completar la instalación.

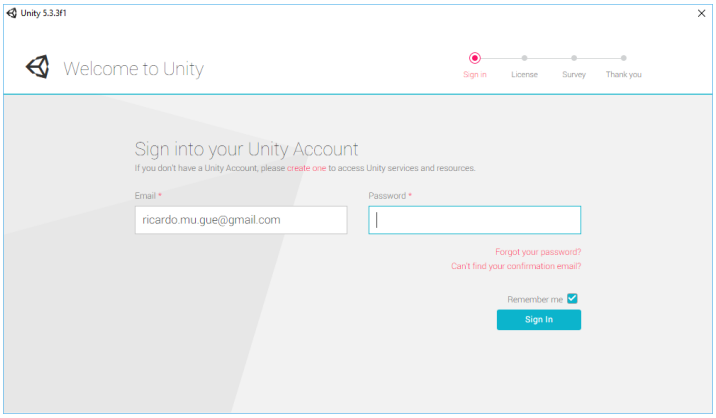


Figura 31. Tras ingresar la cuenta podremos usar Unity

Tras finalizar la instalación podremos iniciar la creación de proyectos para la implementación de este sistema. Es importante mencionar que se debe contar con la versión de Visual Studio 2015 con la última actualización, de lo contrario al finalizar la instalación de Unity se pedirá que reinstalemos toda la paquetería de Unity.

6.1.2 Desarrollo del Escenario

Ya después de la descarga y configuración de Unity, se prosiguió a la creación de nuestro primer escenario tridimensional usando las figuras que tiene el editor por defecto. En este escenario se implementaron los siguientes GameObjects que Unity tiene por defecto:

- Plano: El Piso
- Cubo: Paredes
- Luz: Para la iluminación
- Material: Para nuestra primera imagen
- Cámara: Para tener una vista del escenario

Unity por defecto te crea un escenario vacío en donde se pueden arrastrar y agregar los objetos o GameObjects y éstos pueden ser colocados basándonos en un sistema formado por tres rectas perpendiculares entre sí (X, Y, Z), que se cortan en el origen (0, 0, 0), cada punto del espacio puede nombrarse mediante tres números: (x, y, z), denominados coordenadas del punto, que son las distancias ortogonales a los tres planos principales.

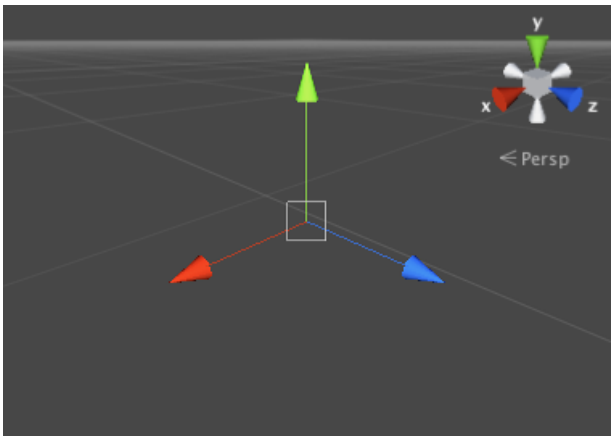


Figura 32. Coordenadas Espaciales de Unity

El escenario virtual que se ocupo toma como referencia que nuestra vista estará colocada en el origen mirando hacia el lado positivo Y, y las paredes estarán rodeando al usuario.

Una vez colocado el GameObject se pueden definir dos propiedades más: el tamaño y la rotación. La primera se denomina escala en Unity y se obtiene mediante la modificación del tamaño, ancho y largo, usando como referencia X, Y, Z. El editor tiene punteros que nos permite ajustar el tamaño de forma manual o podemos modificar los valores directamente en su configuración para mayor precisión.

Una vez que se comprendieron estos conceptos básicos, se generó el escenario que sirvió como base para esta aplicación.

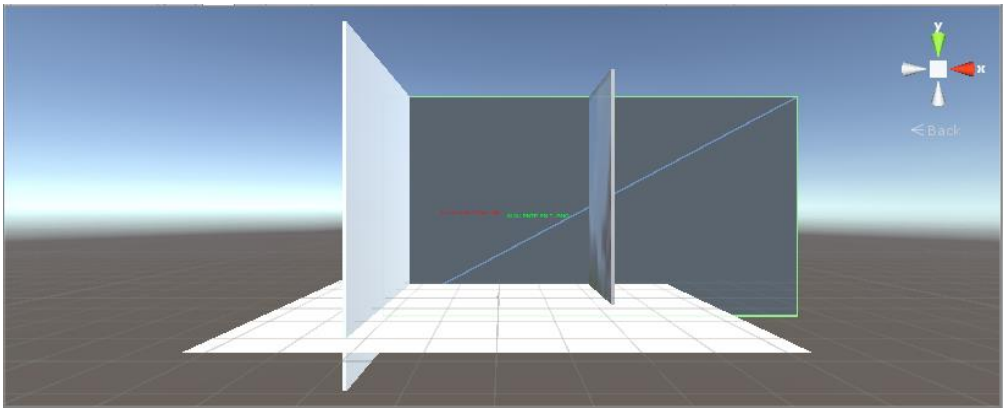


Figura 33. Escenario Virtual Visto desde el Editor de Unity

De igual forma, Unity usa un concepto llamado *Parenting*. Para hacer que cualquier GameObject sea hijo de otro, sólo será necesario arrastrar la figura al padre deseado en la jerarquía. Un hijo va a heredar el movimiento y rotación de su padre.

6.1.3 Desarrollo de Figuras Personalizados

El siguiente paso fue comenzar a crear las figuras básicas que se utilizarían en el escenario, para eso se usó un recurso de Unity llamado Prefab.

Un Prefab es un tipo de asset que permite almacenar un objeto GameObject completamente con componentes y propiedades. El prefab actúa como una plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena. Cualquier edición hecha a un Prefab asset será inmediatamente reflejado en todas las instancias producidas por él, pero, también se puede anular componentes y ajustes para cada instancia individualmente.

Los pasos para crear un Prefab son los siguientes:

1. Ir al menú Assets -> Create -> Prefab. Lo cual creará un prefab vacío.
2. Asignar nombre al prefab, en este caso lo llamaremos “prefab_cube”.
3. Crear un cubo GameObject -> 3D Object -> Cube.

4. Asignar al cubo los elementos que guardaremos en el prefab (scripts, material, textura, componentes, etc.).

5. Arrastrar el cubo creado al prefab “prefab_cube” para guardarlo.

Al seleccionar el prefab “prefab_cube” podemos ver todos sus componentes, los cuales, son los mismos que tendrán sus instancias.

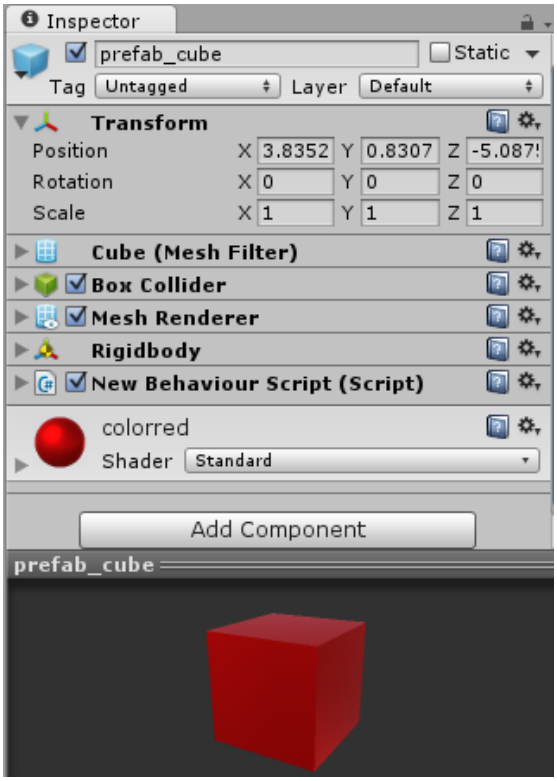


Figura 34.Componentess de un Prefab

Después de realizar estos pasos podremos crear instancias del Prefab de manera muy rápida.

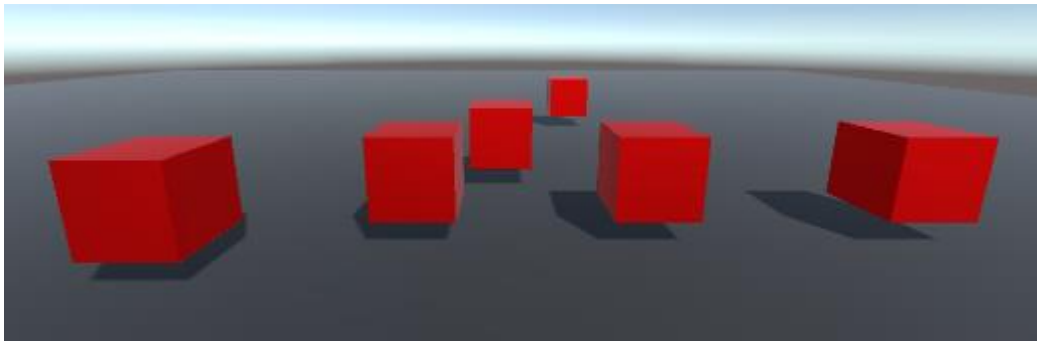


Figura 35. Instancias de un Prefab

6.1.4 Creación de Eventos Unity

En el capítulo 4 se explicó que para la realización de este proyecto se usaría C# como lenguaje de programación para los scripts de las figuras. Desde el editor de Unity tenemos la opción de abrir Visual Studio y se nos da un script en blanco con la librería de Unity: UnityEngine, además de dos eventos que se usaran mucho durante todo el evento:

- `Start ()`: Este método se llama por el sistema una vez al iniciar la aplicación. Se utiliza para llamar scripts que se ejecutan al iniciar el programa.
- `Update ()`: Este método se llama cada que la pantalla se actualiza, específicamente, si nuestra aplicación corre a 60 Hz, se llamara 60 veces por segundo. Se utiliza para recibir entradas (teclado o mouse) o para ajustar figuras.

El primer script que se realizó fue una función que cambiará el color de un Prefab cubo al presionar la tecla especificada en el código.

```
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour{

        // Usado para inicializar

        void Start () {

        }

        // Update es llamado cada vez que se refresca la pantalla
```

```
        void Update () {
        if (Input.GetKeyDown(KeyCode.R))
            gameObject.GetComponent<Renderer>().material.color = Color.red;
        if (Input.GetKeyDown(KeyCode.G))
            gameObject.GetComponent<Renderer>().material.color = Color.green;
        if (Input.GetKeyDown(KeyCode.B))
            gameObject.GetComponent<Renderer>().material.color = Color.blue;
    }
}
```

Código 1. Cambiar Color de una Figura

Las figuras en Unity tienen la opción de agregar scripts como componente para programar su comportamiento. Al agregar este script a un Prefab, éste cambiará de color al presionar la tecla, por ejemplo, al presionar 'B' nuestro objeto cambiará a color azul.

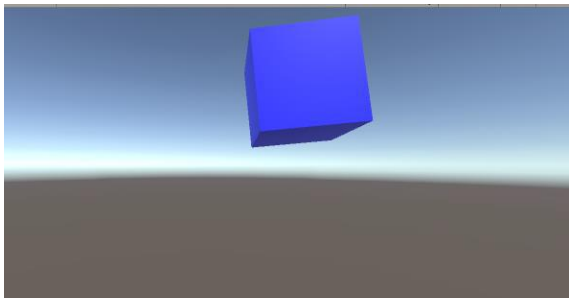


Figura 36. Cambio de Color en la Figura.

Tomando como base esta estructura de creación de scripts de Unity, se desarrollará el código que se usará en todo el proyecto. Igualmente, podemos generar nuestras funciones personalizadas que podremos llamar, por medio de eventos, ya sean los básicos o personalizados, o a través de botones.

6.2 Implementación del Leap Motion

6.2.1 Instalación del Controlador Leap Motion

Como se mencionó previamente, el dispositivo que se ocupará para la detección de movimientos será el controlador Leap Motion y de igual forma se deberá descargar todo el software necesario para que pueda ser utilizado en nuestro equipo entrando a la página oficial del sensor.

Figura 37 Descargar software de la página oficial

Posteriormente el asistente nos guiará por el proceso de instalación y podemos corroborar se instaló correctamente el controlador al tener el ícono en verde.

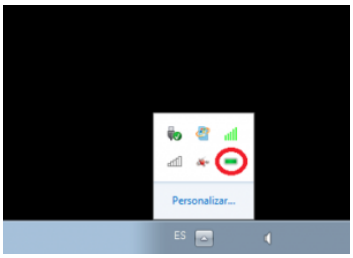


Figura 38. El ícono en verde significa que nuestra computadora reconoce el Leap Motion

6.2.2 Pruebas Iniciales Leap Motion

Para asegurarse de que el dispositivo está configurado correctamente y que funciona de manera óptima, se debe proceder a realizar la prueba incluida en el software de Leap Motion.

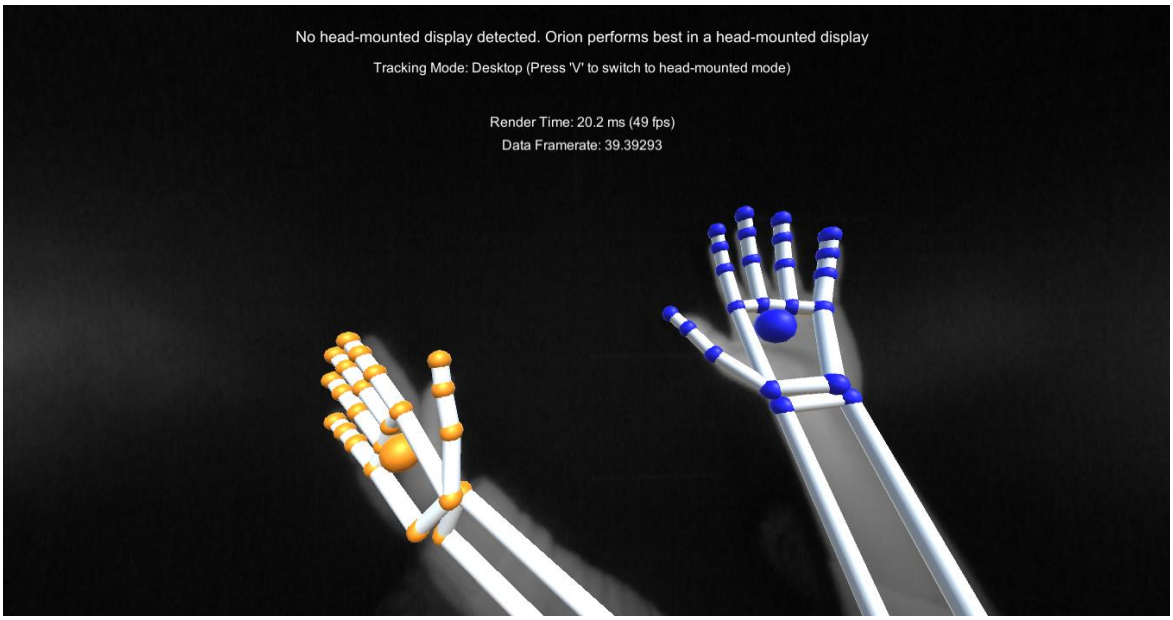


Figura 39 Visualizador de pruebas de Leap Motion

Con esta vista, podemos percibir si el dispositivo hace bien la detección de los elementos de la mano, además de que permite la resolución de algunos problemas de funcionamiento.

6.2.3 Conectividad Unity y Leap Motion

Leap Motion provee una API especial para realizar aplicaciones dirigidas a la Realidad Virtual, llamada Orion, la cual se puede descargar de manera gratuita desde el sitio oficial.

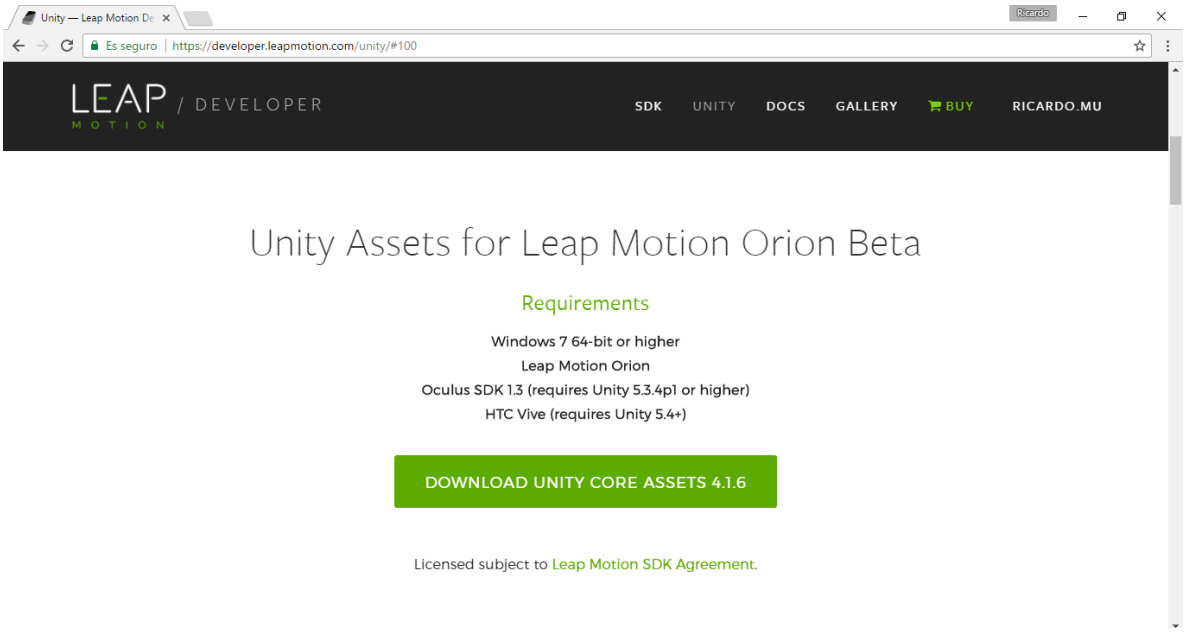


Figura 40 Sitio Oficial de Leap Motion

Orion incluye diversos módulos especializados para la creación de aplicaciones, sin embargo, el principal es el mostrado en la Figura 40, ya que incluye las funciones esenciales básicas para el funcionamiento del dispositivo.

Leap Motion requiere el empleo de diferentes modelos para las manos, ellos realizan funciones específicas para lograr visualización, interacción, y anclaje de objetos, todos estos modelos vienen incluidos en los paquetes que provee Orion.

Una vez descargado el módulo principal *Core Assets*, lo importaremos desde Unity, haciendo click en Assets → Import Package → Custom Package, y seleccionando el archivo que descargamos.

Después de confirmar la importación de los archivos, podremos empezar a trabajar, haciendo una pequeña prueba con la inclusión del Controlador y la visualización de los modelos de las manos en una escena hecha en Unity.

Para realizar esto, se debe insertar el elemento “LeapHandController” y los modelos para las manos CapsuleHand y RigidRoundHand, incluidos en la carpeta de Prefabs, en la escena. CapsuleHand es un modelo de las manos que ofrece la visualización de las mismas, mientras que RigidRoundHand es el modelo que permite la lógica detrás de su funcionamiento. Estos modelos se deben insertar como hijos de un objeto llamado “HandModels”. Este objeto contendrá más modelos que se utilizarán con posterioridad.

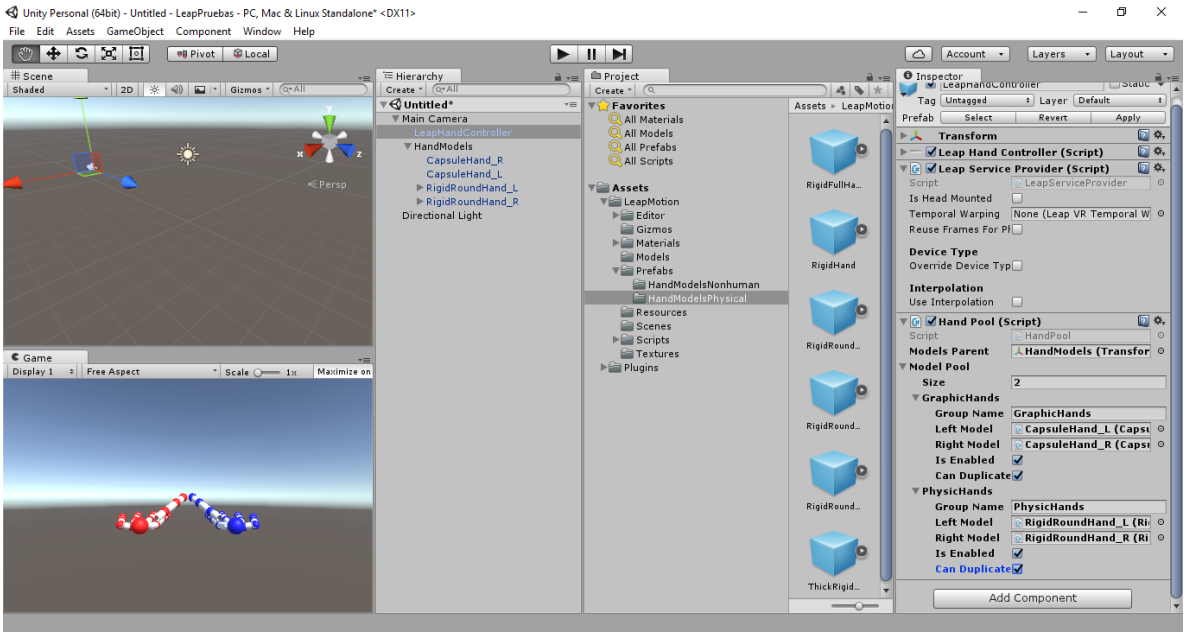


Figura 41 Escena en Unity incluyendo modelos de Leap Motion

Se deben incluir los modelos de las manos que se utilizarán en la escena en el elemento "Hand Pool" del LeapHandController, tal como se muestra en la Figura 41. Al ejecutar esta escena, podemos ver nuestras manos al colocarlas sobre el Leap Motion.

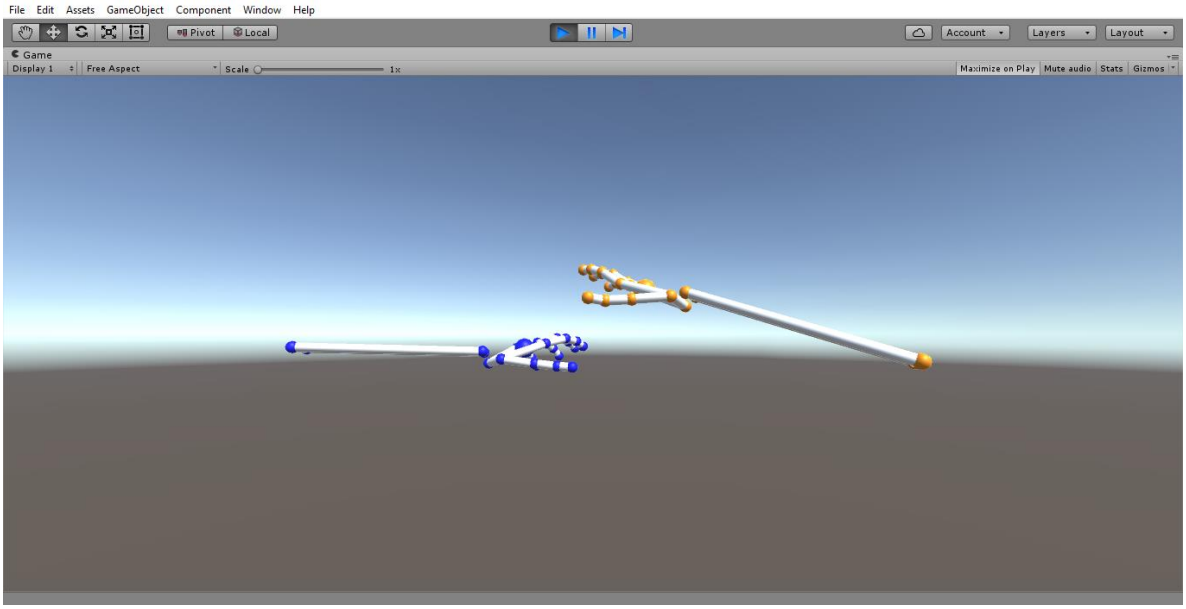


Figura 42 Escena en Unity utilizando el Controlador Leap Motion

6.2.3 Implementación de Gestos

La parte más esencial del manejo del Controlador Leap Motion, es la detección de gestos manuales. Orion incluye algunos Detectores por defecto, así como una Compuerta Lógica para poder utilizar más de un detector al mismo tiempo para desencadenar alguna función específica, pudiendo configurarla como AND u OR, además de poder hacer negaciones.

La mayoría de los detectores tienen un elemento llamado *Gizmos* que no es más que una opción de visualización de Debug, para saber cuándo se está cumpliendo una condición, y cuándo no.

6.2.3.1 Inserción de Figuras.

Para la inserción de figuras, se definió que se mostraría un menú con las figuras que se pueden insertar en el escenario, este menú sólo será visible dirigiendo la palma de la mano izquierda extendida hacia arriba y seleccionar una figura con sólo tocarla con el índice de la mano derecha.

Para lograr esto se utilizó el detector de dirección de la palma de la mano, el detector de extensión de los dedos, el detector de proximidad y el módulo de Orion para el anclaje del menú de figuras a la palma de la mano izquierda.

El primer paso fue utilizar los modelos "LeftHandAttachments" y "RightHandAttachments" que nos permiten anclar objetos a los elementos de las manos. Estos están ubicados en la carpeta LeapMotionModules/HandAttachments/Prefabs creada al importar el módulo "HandAttachment", y en la palma de la mano izquierda al incluir las figuras básicas y posicionarlas como lo deseemos.

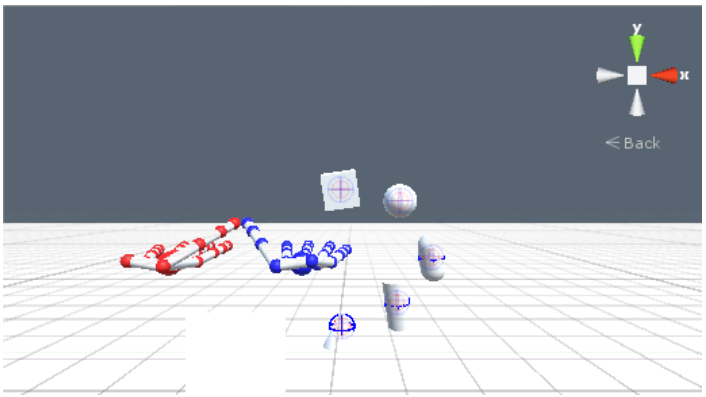


Figura 43 Menú de Figuras

El siguiente paso es definir el detector de extensión de los dedos. Este se debe agregar como un elemento del modelo “LeftHandAttachment” y se debe configurar para saber qué dedos deseamos que estén extendidos y cuáles no, tal como se muestra en la siguiente figura.

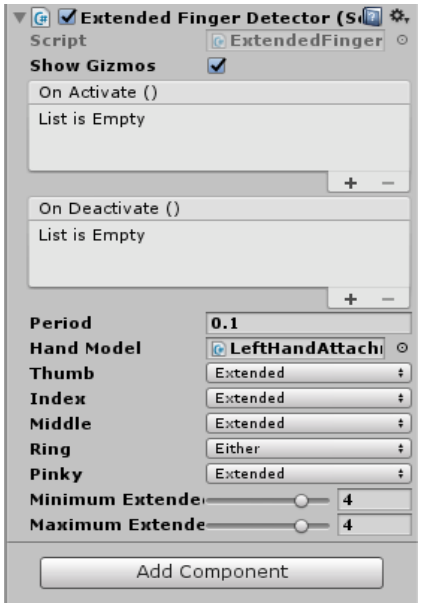


Figura 44 Configuración del detector de extensión de dedos

Una vez configurado, se debe configurar el detector de dirección de la palma de la mano. El cual se configura de la siguiente manera.

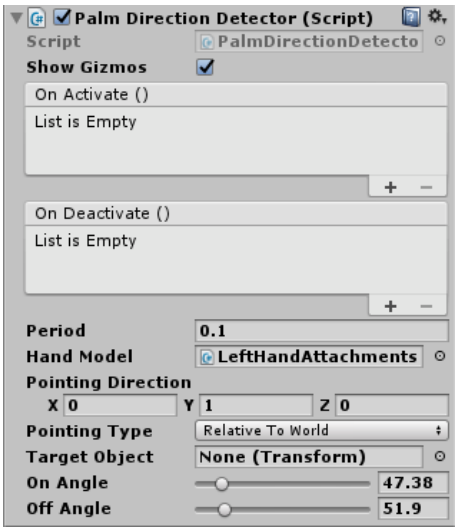


Figura 45 Configuración del detector de dirección de la palma de la mano

Se debe configurar la dirección en la cual deseamos que el detector dispare una acción en formas de las coordenadas X, Y y Z. Como deseamos que se active

cuando la palma de la mano esté apuntando hacia arriba, se debe definir que es en dirección del vector Y y el ángulo permitido para activarse y desactivarse (On Angle y Off Angle).

Para lograr que el menú sea visible cuando estos dos detectores se activen, necesitamos incluir una compuerta lógica (*Detector Logic Gate*).

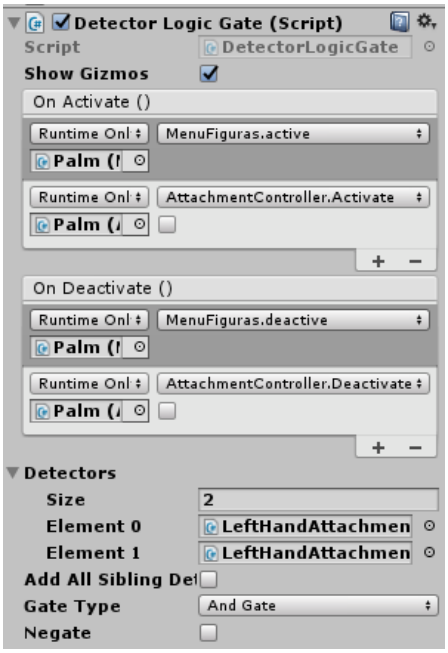


Figura 46 Configuración de compuerta lógica

Para configurar este elemento, se debe definir la acción que queremos que se realice cuando se activen los detectores; que en este caso es hacer visible el menú de las figuras, la acción a realizar cuando se desactiven los detectores; para nuestro proyecto es quitar la visibilidad del menú, los detectores que deseamos que funcionen juntos, y el tipo de compuerta lógica, que en este caso es AND ya que deseamos que se detecten los dos al mismo tiempo.

Hasta el momento, el menú es visible, pero aún no hay ningún tipo de interacción.

Para lograr que, al tocar una figura, ésta se inserte en el escenario, se debe incluir un detector de proximidad en las figuras del menú y configurarlo de la siguiente manera.

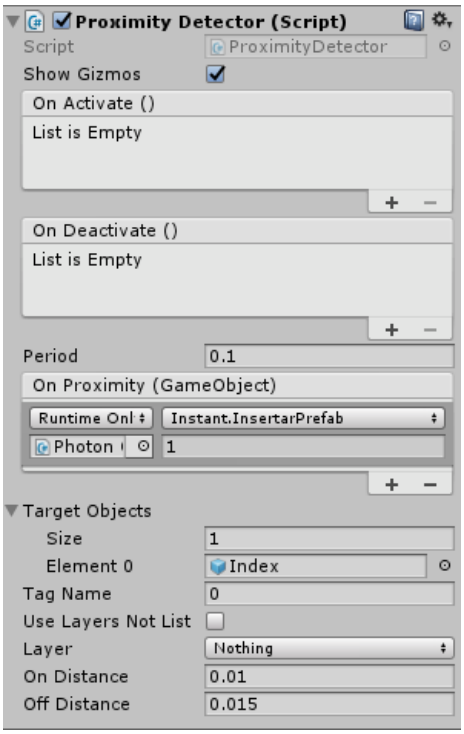


Figura 47 Configuración de detector de proximidad

Como se puede ver, se debe definir la acción a realizar cuando se activa el detector, en “Target Objects” se definen los objetos con los que se desea que se detecte la proximidad, que en este caso es el índice de la mano derecha y la distancia en la que queremos que se active y se desactive.

En la Figura 46, se puede ver que, cuando se activa el detector se manda a llamar una función llamada `InsertarPrefab` del script `Instant`, insertado en el objeto “Photon Objects”. Lo que esta función hace es insertar en el escenario una figura, mandando como parámetro un entero que hace referencia al tipo de figura.

```
//La siguiente función inserta un Prefab al escenario en la posición especificada de la figura.
public void InsertarPrefab(int i)
    {

        ChatGui myInstance = gameObject.GetComponent<ChatGui>();
        GameObject objeto = null;
        if (myInstance.esModelador == true)
        {
            float val = 0.3f;
            if (i == 1)
```

```
            {
                objeto = PhotonNetwork.Instantiate(Prefab.name, new Vector3(-0.3f, 0, 1),
Quaternion.identity, 0);
            }
            if (i == 2)
            {
                objeto = PhotonNetwork.Instantiate(Prefab2.name, new Vector3(-0.1f, 0, 1),
Quaternion.identity, 0);
            }
            if (i == 3)
            {
                objeto = PhotonNetwork.Instantiate(capsula.name, new Vector3(0.1f, 0, 1),
Quaternion.identity, 0);
            }
            if (i == 4)
            {
                objeto = PhotonNetwork.Instantiate(cilindro.name, new Vector3(0.3f, 0, 1),
Quaternion.identity, 0);
            }
            if (i == 5)
            {
                objeto = PhotonNetwork.Instantiate(cono.name, new Vector3(0, -1, 1.1f),
Quaternion.identity, 0);
            }
            if (objeto != null)
            {
                pilaGameObjects.Push(objeto);
                objeto.GetComponent<Renderer>().material.color = new Color(1, 1, 1);
            }
        }
    }
```

Código 2 Inserción de figura

El código anterior, hace referencia a un objeto llamado `PhotonNetwork`, este se explicará a detalle más adelante en el capítulo.

Otro gesto se desarrolló por medio de estas compuertas lógicas fue el gesto “Extender Dedo”. Este se activaba al cambiar el dedo de estar contraído a extendido.

6.2.3.2 Cambiar posición y rotación.

Una vez que las figuras se encuentran en el escenario, es posible cambiar su posición y su rotación. Esto es posible gracias al módulo de Interacciones de Orion.

Después de descargarlo del sitio oficial, y de importarlo en nuestro proyecto, debemos sustituir los modelos "RigidRoundHand" que ya teníamos importados, por los de BrushHands, incluidos en la carpeta LeapMotionModules/InteractionManager/Prefabs. Estos modelos permiten la interacción con objetos dentro de la escena.

Para ello se debe agregar el elemento "Interaction Behaviour" a los modelos que deseemos mover con nuestras manos.

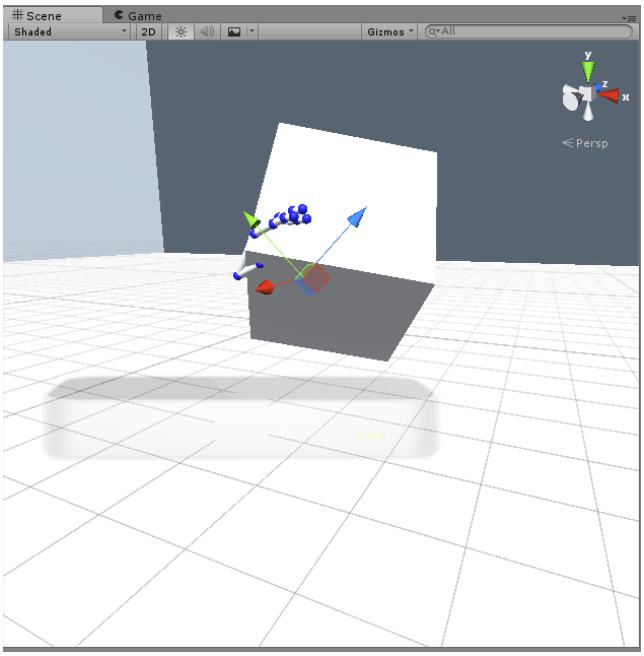
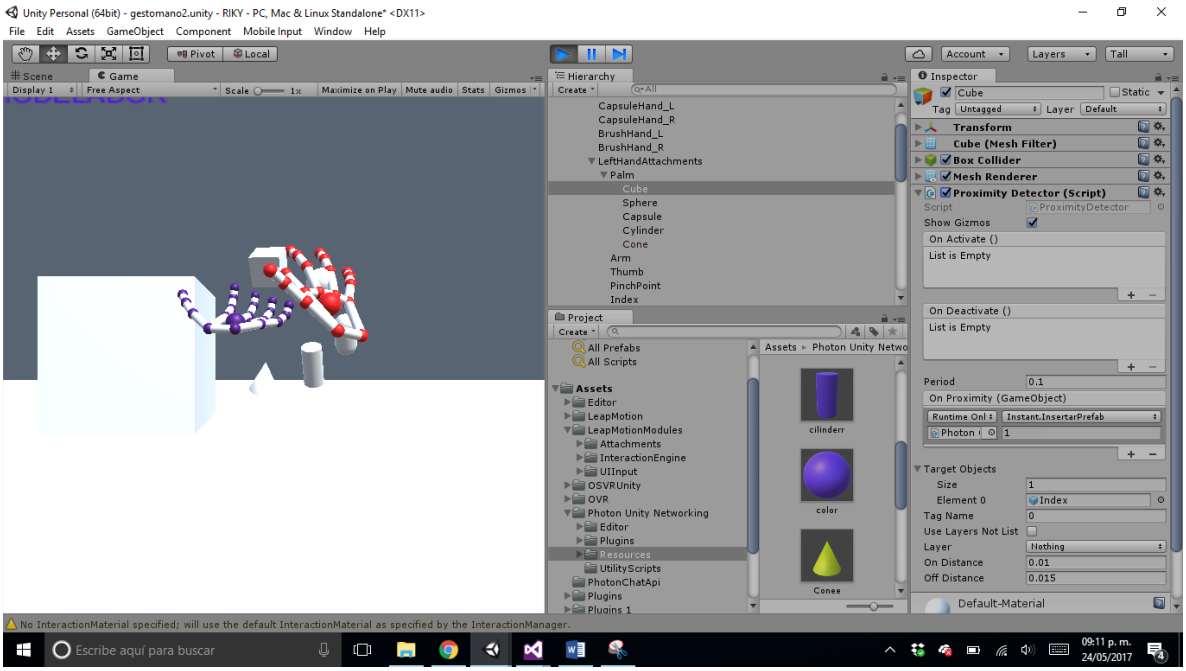


Figura 48 Interacción con una figura



Finalmente, teniendo la implementación de estos distintos gestos, tendremos la capacidad de agregar diferentes eventos como se verá en los capítulos siguientes.

6.3 Implementación de Oculus Rift

6.3.1 Instalación del Controlador Oculus Rift

Para la creación de este proyecto se decidió ocupar el Oculus Rift DK1, el cual es la primera versión para desarrollo del dispositivo. Esto fue debido a que es la versión más económica que se pudo adquirir. El software que existe actualmente en la página de desarrollo de Oculus no era compatible ni con los lentes de Realidad Virtual ni con las capacidades gráficas y de procesamiento del equipo que se utilizaría, por lo que se tuvo que buscar una versión más antigua del driver y desinstalar algunos componentes para limitar el procesamiento gráfico y que el ordenador que se ocupará fuera aceptado para la utilización del Oculus Rift DK1. El software se descargó de la página oficial de Oculus Rift y actualmente se encuentra descontinuado debido a la salida de las nuevas versiones tanto de hardware como de software del Oculus Rift.

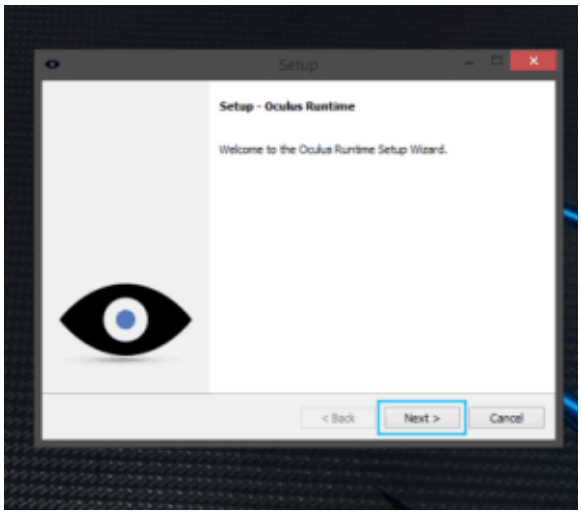


Figura 49. Flujo de Instalación Oculus Rift

Después de descargar el driver, se procedió a instalar todos los componentes y se pudo accesar a la configuración de Oculus desde la computadora.

La configuración de Oculus reconocerá el dispositivo al conectarlo y en caso de no hacerlo sólo será necesario reiniciar el servicio. Desde este Utility podremos revisar el software instalado y la versión del dispositivo. Para este proyecto ocuparemos la versión del SDK 0.4.4.

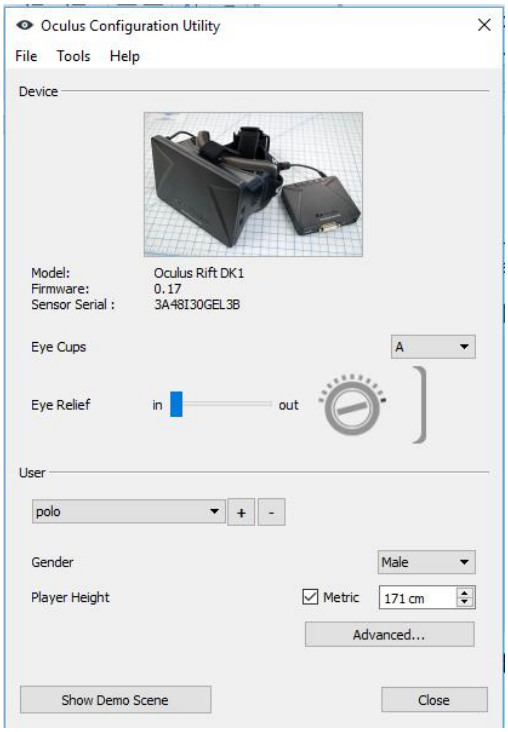


Figura 50. Configuración del Oculus

Una vez confirmado que la computadora reconoce el dispositivo, se puede lanzar una escena Demo para probar que funciona la detección de movimiento de la cabeza y se puede ver la imagen correctamente en nuestro visor.

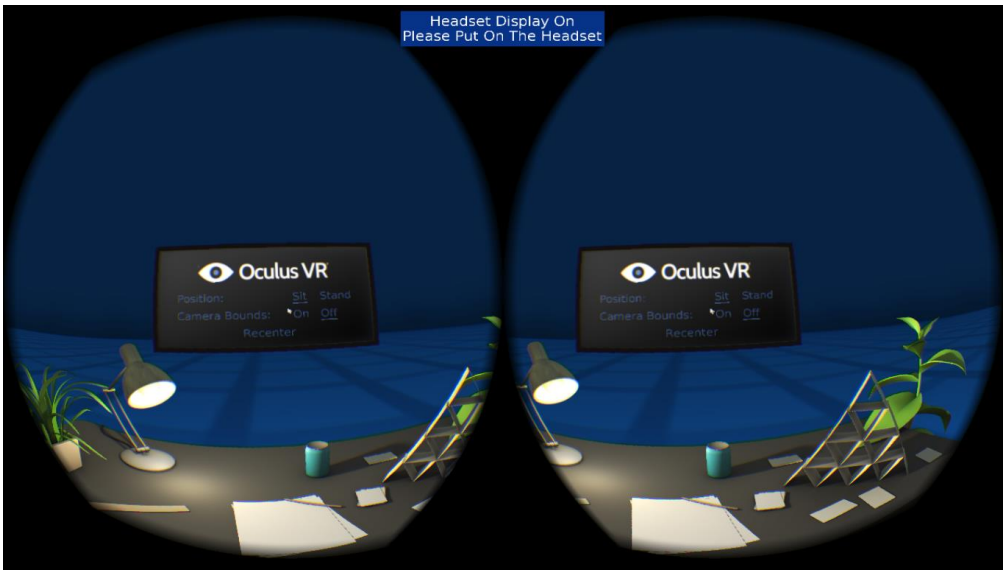


Figura 51. Escena de prueba Oculus VR

Una vez hecha esta prueba, se ha conectado correctamente el Oculus Rift DK1 al equipo para poder empezar a desarrollar en él.

6.3.2 Conectividad Unity y Oculus Rift DK1

Después de conectar correctamente el dispositivo Oculus, se procedió a conectar el motor gráfico Unity con el visor de Realidad Virtual. Como se mencionó en el capítulo anterior, debido a las características gráficas de nuestra computadora y la versión de los lentes de Realidad Virtual que se usaron, fue necesario usar un SDK anterior. Eso ocasiono que los Scripts existentes estuvieran diseñados para la versión 4 de Unity, sin embargo, este proyecto había sido diseñado para ser hecho en Unity 5, por lo que fue necesario actualizar muchos scripts a la nueva versión.

```
[ExecuteInEditMode]

public class OVRCameraRig : MonoBehaviour

{

    // Camara Izquierda.

    private Camera leftEyeCamera;

    // Camara Derecha.

    private Camera rightEyeCamera;
```

```
    // Camara Central.
    private Camera centralEyeCamera;


    // Siempre coincide con la posición del Ojo Izquierdo
    public Transform leftEyeAnchor { get; private set; }
    // Siempre coincide con la posición del Ojo Central
    public Transform centerEyeAnchor { get; private set; }
    // Siempre coincide con la posición del Ojo Derecho
    public Transform rightEyeAnchor { get; private set; }
```

Código 3. Declaración de la clase OVRCameraRig y sus atributos

El script `OVRCameraRig` es el que se agregó al GameObject para hacer una salida hacia los Oculus Rift. Como se ve en el script anterior, la clase OVRCameraRig tiene como atributos Camera, el cual es el objeto de Unity por defecto que sirve como una cámara para poder observar el entorno que se crea.

Como se puede ver en el Diagrama 17, cada ojo tendrá su objeto Camara, la cual se encargará de reproducir a la pantalla lo que tenga en su campo de visión. Estos al ser hijos del objeto `OVRCameraRig` se moverán al mover el objeto padre para mantener una sincronización de la rotación.

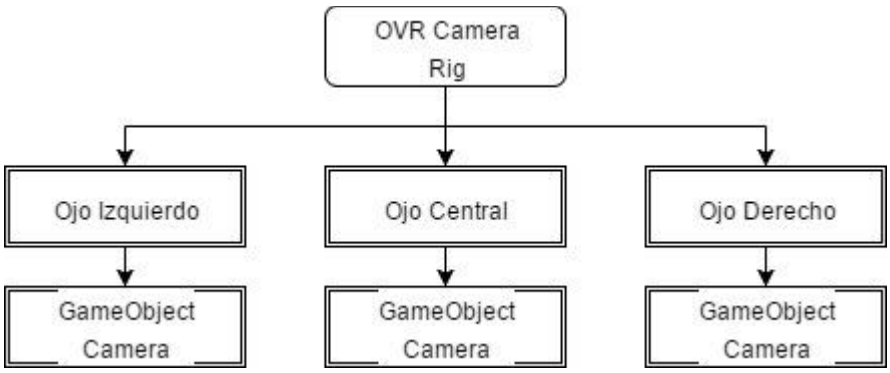


Diagrama 17. Creación de la Vista de Realidad Virtual

```
private Camera ConfigureCamera(OVREye eye)
        {
                Transform anchor = (eye == OVREye.Left) ? leftEyeAnchor : rightEyeAnchor;
```

```
                Camera cam = anchor.GetComponent<Camera>();


                OVRDisplay.EyeRenderDesc eyeDesc = OVRManager.display.GetEyeRenderDesc(eye);


                cam.fieldOfView = eyeDesc.fov.y;

                cam.aspect = eyeDesc.resolution.x / eyeDesc.resolution.y;

                cam.rect = new Rect(0f, 0f, OVRManager.instance.virtualTextureScale,
OVRManager.instance.virtualTextureScale);

                cam.targetTexture = OVRManager.display.GetEyeTexture(eye);


                if (cam.actualRenderingPath == RenderingPath.DeferredLighting)

                        QualitySettings.antiAliasing = 0;


                return cam;

        }
```

Código 4. Métodos de la clase OVRCameraRig de movimiento y actualización de imagen

El script anterior muestra los métodos para la actualización y movimiento de imagen que se muestra en los Oculus. Al mover la cabeza, todas las cámaras se moverán para ser capaces de ver el entorno.

Para que un GameObject pueda ser utilizado para visualizar objetos usando Oculus Rift se tendrá que agregar otro script llamado `OVRManager` en donde se puede elegir la textura, la profundidad, entre otras cosas.

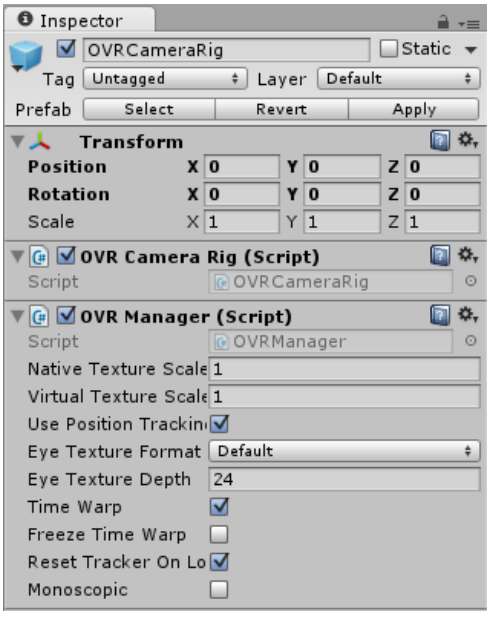


Figura 52. Definición de propiedades del objeto OVRCameraRig

Finalmente, este objeto se agregó en el origen del escenario (0,0,0), para visualizar figuras diseñadas en el proyecto de manera personalizada. Se detectará el movimiento de la cabeza y se podrá ver el escenario en todas direcciones.

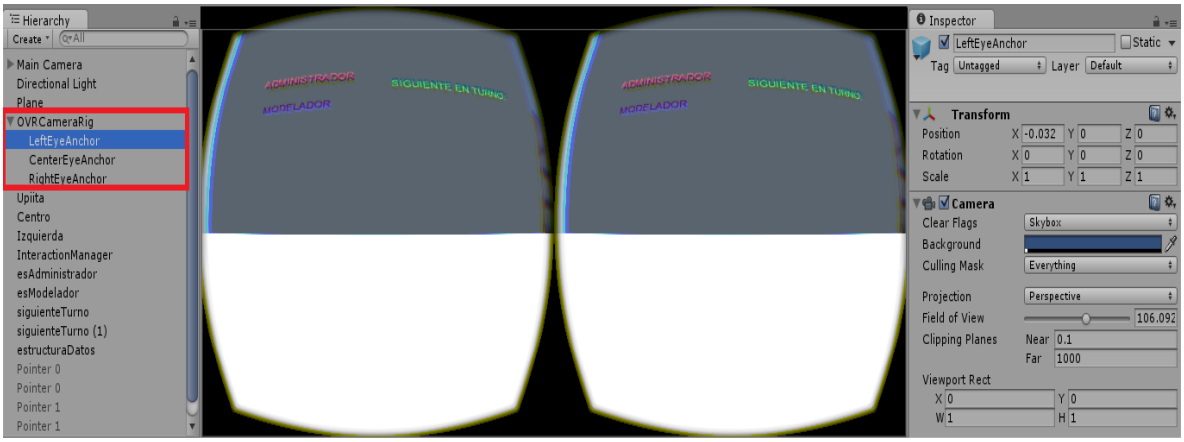


Figura 53. Conectividad Oculus con Unity

El objeto `OVRCameraRig` agrega por defecto tres objetos hijos los cuales son dos cámaras Unity para cada ojo y una central para el fondo negro que se ve en la figura. Al correr la aplicación, se muestra por defecto la vista como se ve en la Figura 52, y esto permitirá darnos el efecto de realidad virtual. Esto se logra repitiendo la imagen dos veces en cada ojo para que, al usar el visor, se tenga un efecto de inmersividad en el escenario de Realidad Virtual.

Por último, es importante aclarar que las imágenes que se mostrarán son capturas de pantalla en donde se ven las imágenes son las generadas para ambos ojos, sin embargo, al tener puestos los Oculus Rift, el escenario se verá como se muestra la Figura 53.

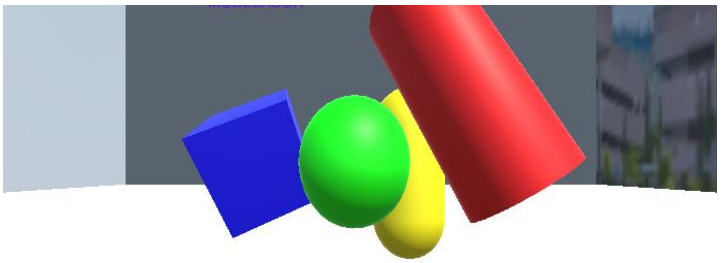


Figura 54. Vista Generada para Oculus Rift

6.4 Integración de Oculus Rift y Leap Motion

En este punto ya se tenía conectividad de Unity tanto con Oculus Rift como con Leap Motion, y el siguiente objetivo fue conectar ambos módulos para tener una aplicación que mostrará las manos generadas por Leap Motion en el visor de Realidad Virtual. Unity permite guardar distintas escenas en un único proyecto, y éstas pueden incluir distintos scripts y objetos. Lo que se realizo fue crear una escena de Oculus y otra de Leap Motion, sobreponer ambas escenas y simplemente jalar los GameObjects desde el editor, esto permitió conservar todos los scripts agregados a las figuras. Todos los GameObjects del módulo de Leap Motion se agregaron como hijos de la Cámara Principal para que éstos se cambiaran de posición al moverse el visor.

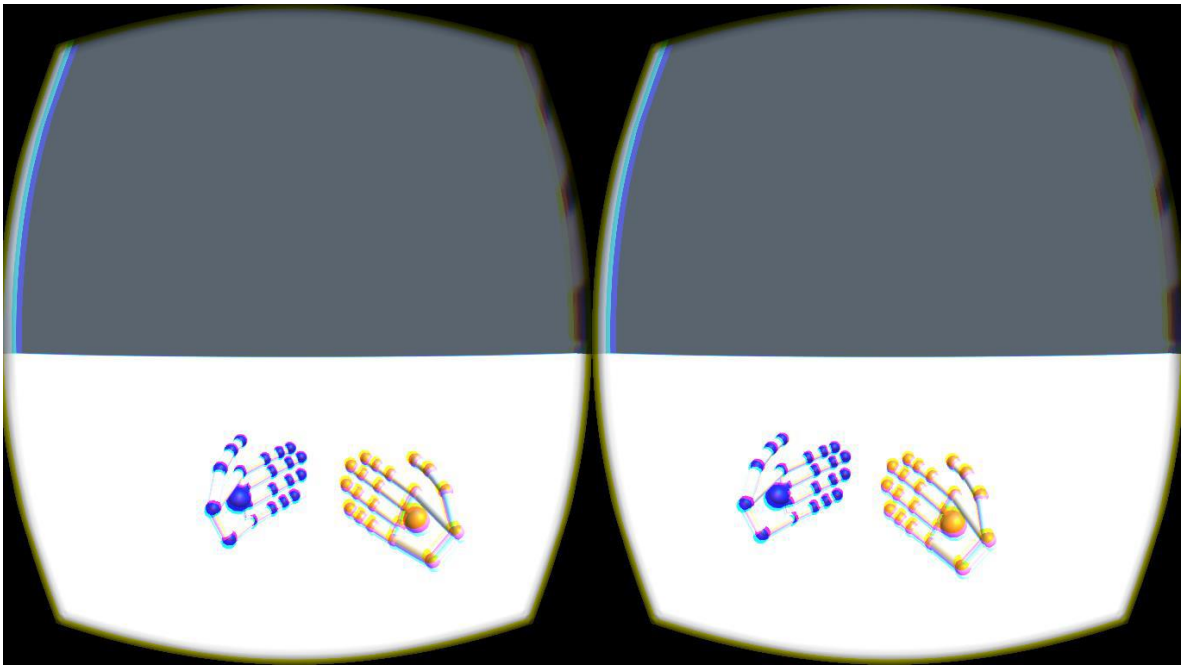


Figura 55. Manos del Leap Motion visualizadas a través del Oculus Rift

En la figura 54 se puede observar las manos generadas en Leap Motion visualizadas a través de la cámara de Oculus Rift. Se pueden detectar todos los movimientos que se realizan a través del sensor de movimiento, sin embargo, se tuvo que cambiar el diseño Canvas y Paneles usados anteriormente en el módulo de Leap Motion debido a que los visores no eran capaces de soportar ese tipo de texturas, por lo que en vez de usar sólo botones o sliders definidos por Unity, se sobrepusieron GameObjects, los cuales si son capaces de ser visualizados por los lentes de Realidad Virtual.

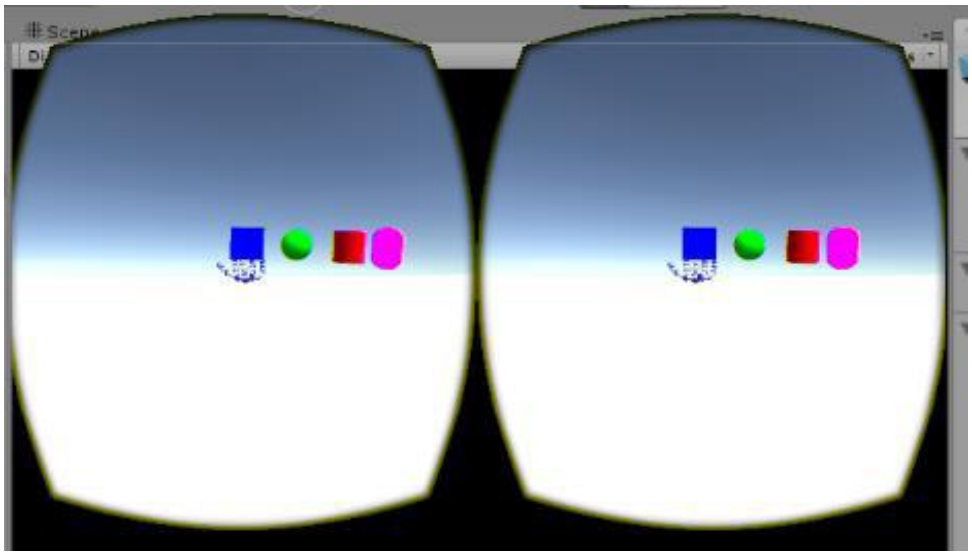


Figura 56. Vista del panel por medio de GameObjects

6.5 Implementación de Photon

En el capítulo 4 se habló de Photon y su funcionalidad en proyectos multijugador, a continuación, se explicarán los pasos para lograr su implementación.

El primer paso fue descargar de la "Asset Store" de Unity el asset de Photon.

Figura 57. Asset de Photon en la Asset Store

Fue necesario registrarnos en la página oficial de Photon e ingresar algunos datos de nuestra aplicación para obtener un Id y tener acceso a los servidores de Photon.

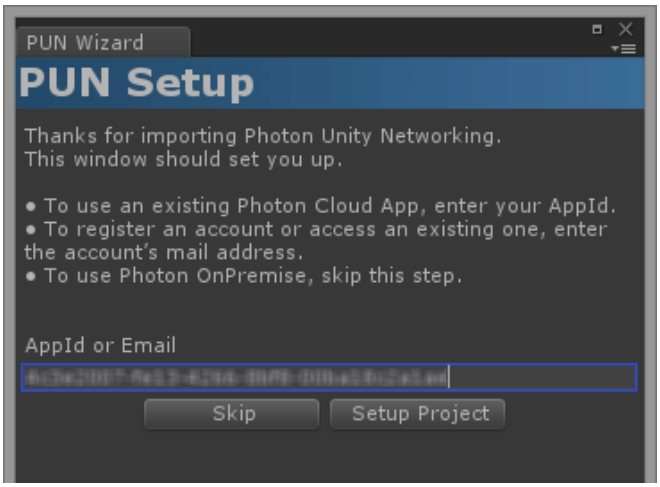


Figura 58. PUN Wizard

El siguiente paso fue importar el asset de Photon a nuestro proyecto e ingresar el Id de aplicación en el PUN Wizard. Una vez que se validó el appId pudimos a comenzar a utilizar las funciones de Photon.

6.5.1 Sincronización de Figuras

El primer paso que se hizo para la sincronización de figuras fue la selección del protocolo de transporte a utilizar.

```
public const string NameServerHost = "ns.exitgames.com";

private static readonly Dictionary<ConnectionProtocol, int> ProtocolToNameServerPort = new Dictionary<ConnectionProtocol, int>() { { ConnectionProtocol.Udp, 5058 }, { ConnectionProtocol.Tcp, 4533 } };
```

Código 5. Selección del Protocolo de Transporte

Como se mencionó en el capítulo 6, el protocolo que se eligió fue UDP debido a que se adecuaba más a un formato de Multidifusión y como se puede ver en el código, estos funcionan utilizando el puerto 5058 para UDP y 4533 para TCP. También se encuentra la dirección DNS del Servidor en la nube a conectar que es "ns.exitgames.com".

Una vez configurado esto, se utilizaron los mensajes Enet que se pueden ver en la Tabla 2 de este documento del capítulo 6. Estos pasos se llevan a cabo para establecer la conexión:

- Un cliente inicia la aplicación y automáticamente se intenta conectar por medio de un mensaje [Connect].
- El servidor Photon acepta la conexión y le responde con un mensaje [Verify Connect].

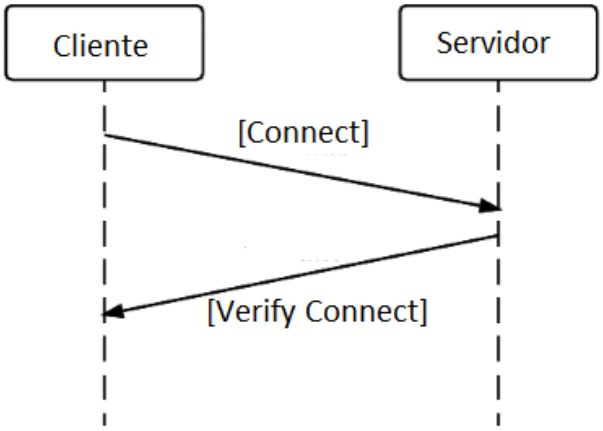


Diagrama 18. Establecimiento de la Conexión

Ya una vez establecida la conexión, es posible el envio de distintos eventos por medio de mensajes Multidifusión, logrando que múltiples usuarios puedan ver los cambios al momento que se realizan. Todos estos mensajes son almacenados en un buffer en el servidor, por lo que al conectarse nuevos usuarios recibirán todos estos mensajes en el orden en que se enviaron.

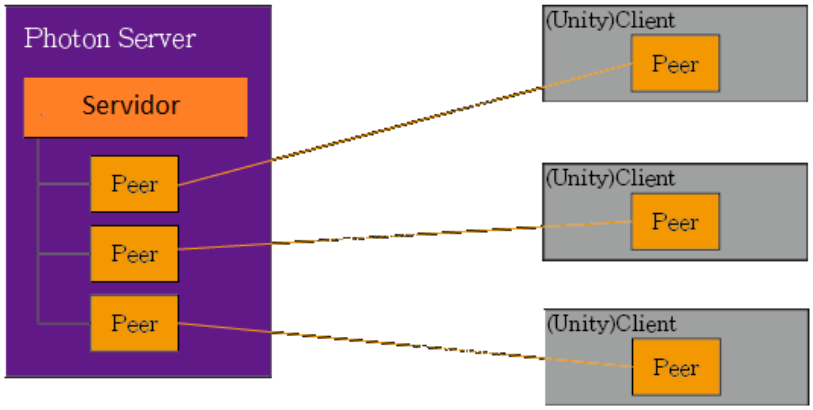


Diagrama 19. Conexión al servidor Photon

Un mensaje enviado por un cliente es almacenado en el servidor y enviado al instante a todos los clientes demás conectados, incluyendo al emisor. Todos estos mensajes dictarán el comportamiento de las distintas figuras que nosotros elijamos para sincronizar agregándole un script que Photon nos facilita llamado `PhotonView`.

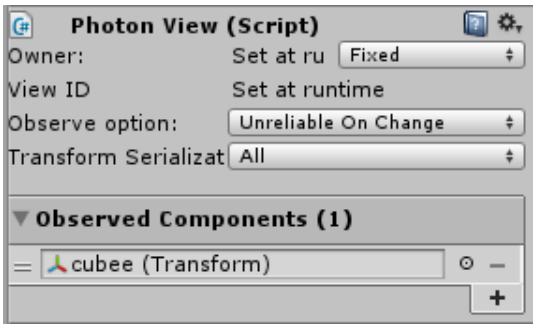


Figura 59. Script Photon View

Simplemente con la inclusión de este script será posible sincronizar las diferentes figuras que sean ingresadas por medio del método Instantiate de la clase estática PhotonNetwork como se puede ver en el Código 2. Las características que se sincronizan por defecto son:

- Posición (Position)
- Rotación (Rotation)
- Tamaño (Scale)

Igualmente, se implementó un servicio de envio de mensajes por multidifusión, esto sirvió como semáforo para la señalización de turnos que se implementó. Los mensajes son enviados a todos los clientes, pero solo responde el cliente a quien está dirigido el envio.

Después de esto, se creó una primera aplicación que creaba un simple cuadrado y este se reproducía en todas las instancias conectadas, además del envio de simples cadenas de texto que también eran recibidas por todos los clientes y contenía el nombre del usuario que la envio

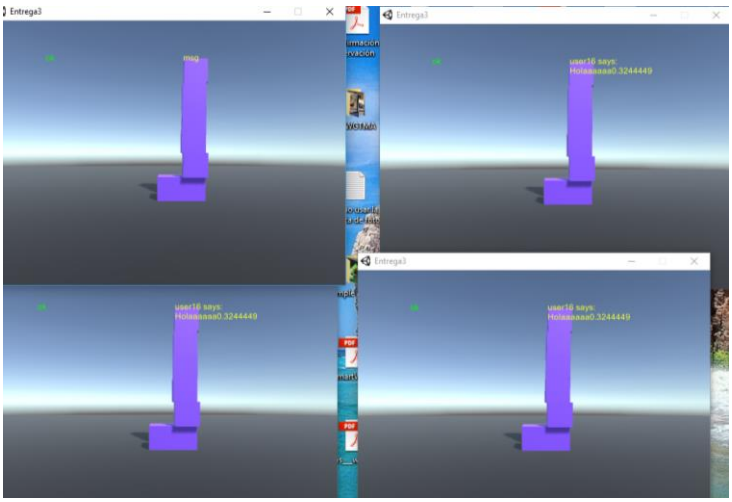


Figura 60. Primer Programa de Sincronización de Figuras

En la figura anterior podemos ver como se crearon distintos procesos de la misma aplicación, posteriormente se fueron agregando figuras y estas eran reproducidas en todas las instancias. Por último, la instancia que se encontraba en la parte superior izquierda envio un mensaje de hola que todos recibieron. El emisor no lo imprimió debido a que se creó una sentencia de control para distinguir quien envio el mensaje.

La creación de esta aplicación nos permitió tener las bases para la generación del programa final.

6.6 Implementación del Sistema de Turnos

La principal función del sistema de turnos es evitar problemas de concurrencia cuando hay dos o más usuarios trabajando en el mismo diseño.

El siguiente diagrama de flujo muestra la forma en que se implementó el sistema de turnos.

Diagrama 20. Solicitud de Turnos

Cuando un usuario está editando el modelo y otro usuario solicita el turno, el actual editor tiene la opción de ceder el turno o seguir trabajando, y automáticamente se guarda el nombre de la persona que pidió el turno en la lista de espera.

Cuando el editor cede el turno, el turno se otorga a la primera persona en la lista de espera y además se le envía la lista de espera restante para dar continuidad a los turnos que ya se habían solicitado. De esta forma siempre se tendrá solamente a un usuario editando lo cual evitará los problemas de concurrencia.

Como se mencionó en el capítulo de Análisis de Requerimientos, después de configurar los protocolos de comunicación de Photon y las llaves de acceso, se procedió a la creación de la lógica de la petición y donación de turnos creando

scripts y métodos que responderán a eventos externos. De manera general, hay dos métodos principales, el primero es el envio de sentencias de control y el segundo es la recepción de esas sentencias. Todos estos mensajes funcionan por medio del envio de paquetes por multidifusión y estos son recibidos por todos los usuarios del proyecto. Sin embargo, la aplicación solo responderá si el mensaje está dirigido a ese usuario en particular.

```
public void pedirDarTurno()

    {

        //Dar turno

        if (esModelador)

        {

            if (colaDeTurnos.Count != 0)

            {

                string siguienteEnTurno = colaDeTurnos.Dequeue();

                chatClient.PublishMessage("channelNameHere", "1" + siguienteEnTurno);

                string texto = recuperarCola();

                //chattag.text = "CEDER TURNO A " + siguienteEnTurno;

                cola.text = texto;

                esModelador = false;

                GameObject.Find("Main
Camera/HandsModel/LeftHandAttachments").GetComponent<Leap.Unity.PalmDirectionDetector>().enabled =
false;

                InteractionManager gras =
GameObject.Find("InteractionManager").GetComponent<InteractionManager>();

                gras.GraspingEnabled = false;

                turnos.text = "PEDIR\nTURNO";

                chattag.text = "OBSERVADOR";

                Instant myInstance = gameObject.GetComponent<Instant>();

                myInstance.pilaGameObjects = new Stack<GameObject>();

            }

        }

        //SendChatMessage("msgmsg");

        //chatClient.PublishMessage("channelNameHere",PhotonNetwork.playerName+" Holaaaaaa" +
UnityEngine.Random.value);

        //Pedir turno

        else
```

```
        {
            cola.text = "TURNO PEDIDO POR MI:" + PhotonNetwork.playerName;
            turnos.text = "TURNO\nPEDIDO";
            chatClient.PublishMessage("channelNameHere", "2" + PhotonNetwork.playerName);
        }
        //chatClient.PublishMessage("channelNameHere", PhotonNetwork.playerName + " Holaaaaaa" +
UnityEngine.Random.value);


    }
```

Código 6. Solicitud o Donación de Turno

Como se muestra en el Código 6, el método *pedirDarTurno()* se encarga de solicitar o dar el turno por medio del envio de un mensaje a todos los usuarios. En los campos de éste se incluye el nombre del usuario a quien está destinado. Como se mencionó en los requerimientos del sistema, un usuario puede estar modelando una figura o estar solo observándola, por lo tanto, si el usuario esta en modo Modelador, el mensaje se encargará de dar el turno al usuario que está esperando en la cola de turnos cediéndole el rol de Modelador. Por el contrario, si el que usa el método esta en modo Espectador, este estará pidiendo la solicitud de turno para Modelar y estará en modo espera y será encolado a respuesta de que el turno le sea cedido por el Modelador en turno.

Este método puede ser activado en respuesta al presionar una tecla, tocar una figura o con un gesto. Para las pruebas de este módulo en particular, se agregó al método *Update()* al presionar la tecla “S”.

```
     public void OnGetMessages(string channelName, string[] senders, object[] messages)
    {
        if (channelName.Equals(this.selectedChannelName))
        {
            // update text
            ShowChannel(this.selectedChannelName);
        }
        //*************
        int msgCount = messages.Length;
        Debug.Log("Got " + msgCount + " new messages!");


        for (int i = 0; i < msgCount; i++)
```

```
        { //go through each received msg and display them


            string sender = senders[i];

            string msg = messages[i].ToString();

            //RECIBIR TURNO

            if (msg[0] == '1')

            {

                //chattag.text = msg;

                if (msg == ("1" + PhotonNetwork.playerName))

                {

                    esModelador = true;

                    GameObject.Find("Main
Camera/HandsModel/LeftHandAttachments").GetComponent<Leap.Unity.PalmDirectionDetector>().enabled =
true;

                    InteractionManager gras =
GameObject.Find("InteractionManager").GetComponent<InteractionManager>();

                    gras.GraspingEnabled = true;

                    chattag.text = "MODELADOR";

                    cola.text = recuperarCola();

                    if (colaDeTurnos.Count == 0)

                        turnos.text = "SIN\nCOLA";

                    else

                        turnos.text = "CEDER\nTURNO";




                }

            }

            //ACUMULAR TURNOS

            else if (msg[0] == '2')

            {

                if (esModelador)

                {

                    if (!colaDeTurnos.Contains(msg.Substring(1)))

                    {

                        //chattag.text = "ALMACENAR EN COLA " + msg.Substring(1);

                        colaDeTurnos.Enqueue(msg.Substring(1));

                        cola.text=recuperarCola();
```

```
                    turnos.text = "DAR\nTURNO";
                }
            }
        }

        else if (msg[0] == '3')
        {
            if (msg.Substring(1) != PhotonNetwork.playerName)
            {
                chattag.text = "NUEVO USUARIO " + msg.Substring(1);
            }
        }
    }
    //**********************************
}
```

Código 7. Recepción de Mensajes

El segundo método que fue muy importante para la realización de este proyecto fue el de `OnGetMessages()` y solo funciona cuando un mensaje es recibido por el usuario. Si el receptor del mensaje no es a quien está dirigido éste, el mensaje es desechado. Como el método explicado anteriormente, el comportamiento de `OnGetMessages()` depende del estado del usuario que utiliza el método, si se encuentra en modo Modelador, el mensaje significará que una persona está pidiendo turno y esta persona será encolada para que se le sea cedido el turno posteriormente. Si, por el contrario, el usuario está en modo Espectador y a la espera de obtener el turno, el mensaje lo liberara y podrá comenzar a modelar mientras que el emisor del mensaje se bloqueará para evitar problemas de concurrencia.

6.7 Lógica de Negocios

6.7.1 Integración con Photon

Como parte final del Proyecto Terminal, se procedió a integrar los módulos anteriores con lo desarrollado en Photon. Como base, se utilizó la unión que ya se

tenía entre el ambiente virtual, y la implementación entre Oculus Rift y Leap Motion. A estos módulos ya combinados se les procedió agregarle un objeto vacío que sería el encargado de agregar todos los scripts en C# relacionados al establecimiento de conexión, sincronización de figuras y el bloqueo y liberación de turnos.

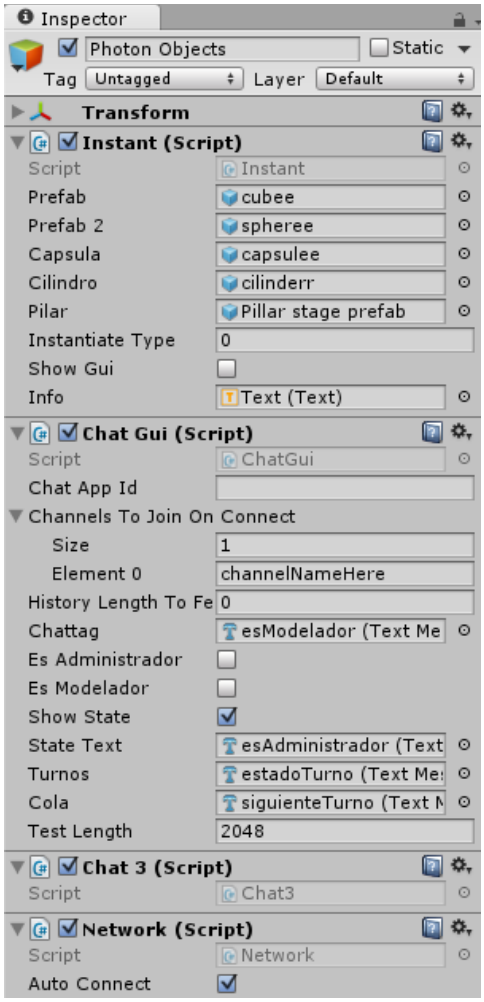


Figura 61. Scripts de Establecimiento de Conexión

Como se puede ver en la figura anterior, se creó el GameObject “Photon Objects” en donde le agregamos los cuatro Scripts encargados de conectividad: `Instant`, `ChatGui` y `Network`. Igualmente, en estos scripts le agregamos las figuras que se estarán sincronizando durante la realización del proyecto, también agregamos las salidas de texto que tendrán los scripts y finalmente distintas configuraciones que se utilizarán durante la utilización de la aplicación. Una vez integrado esto, se

procedió a crear las características que le darían el carácter de Framework o ambiente de trabajo al proyecto.

6.7.2 Pila de Trabajo

Como se mencionó en el análisis de requerimientos, para el manejo de las figuras insertadas se utilizó una pila en donde se almacena cada figura que es insertada al ambiente.

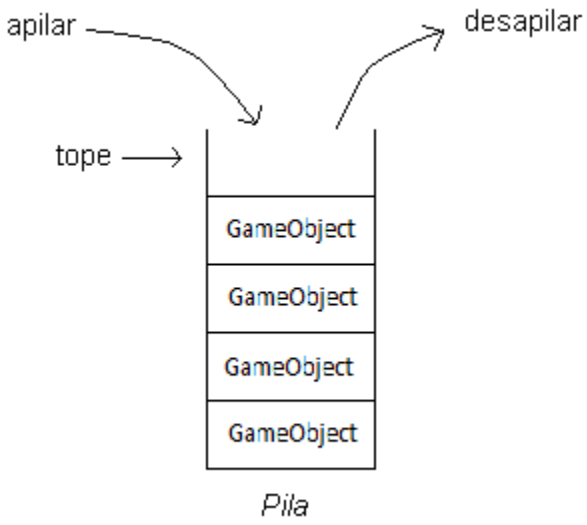


Figura 62. Pila de Trabajo

Todas las instancias de GameObjects de Unity pertenecen a la clase GameObject en C#, entonces, cada vez que se crea una figura en nuestro ambiente virtual, este objeto es almacenado en una pila.

```
public Stack<GameObject> pilaGameObjects = new Stack<GameObject>();


pilaGameObjects.Push(objeto);


PhotonNetwork.Destroy(pilaGameObjects.Pop());


PhotonView photonView = PhotonView.Get(pilaGameObjects.Peek());
```

Figura 63. Creación de Pila, Push y Pop

Utilizando éstos métodos básicos `Push` (apilar), `Pop` (desapilar) y `Peek` (tope) se puede realizar todas las acciones básicas del Framework. Push se utiliza al agregar una nueva figura; Pop cuando se usa el botón retroceso para eliminar la última figura insertada, es importante mencionar que figuras instanciadas por medio de Photon deben ser eliminadas tanto de manera local como en el servidor, esto para que haya una replicación en todos los usuarios. Por último, se ocupa el método `Peek` para tener siempre el último objeto que se agregó al ambiente para hacer distintas operaciones, tales como cambiar el tamaño y cambiar el color.

6.7.2 Funciones RPC

Los scripts que tiene Photon por defecto que se agregan a los Prefabs que tiene la aplicación ya incluye la sincronización tanto de tamaño, posición y rotación. Sin embargo, otras características fuera de las ya mencionadas sólo se pueden realizar por medio de la utilización de llamado a procedimientos remotos (RPC)[24]. Como su nombre lo indica, RPC es un programa que utiliza una computadora para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas.

Para mantener una sincronización de color en el Framework, es necesario la utilización de una función usando RPC, a continuación, se muestra la función que se utilizó para mantener el mismo color entre distintos usuarios:

```
public void cambiarColo(int x)
    {
        if (pilaGameObjects.Count != 0)
        {
            PhotonView photonView = PhotonView.Get(pilaGameObjects.Peek());
            photonView.RPC("cambiarColorPhoton", PhotonTargets.AllBuffered, x);
        }
    }
```

Código 8. Llamada a una función RPC

```
[PunRPC]
    public void cambiarColorPhoton(int x)
```

```
    {
        if (x == 0)
        {
            this.gameObject.GetComponent<Renderer>().material.color = Color.blue;
        }
        if (x == 1)
        {
            this.gameObject.GetComponent<Renderer>().material.color = Color.green;
        }
        if (x == 2)
        {
            this.gameObject.GetComponent<Renderer>().material.color = Color.red;
        }
        if (x == 3)
        {
            this.gameObject.GetComponent<Renderer>().material.color = Color.yellow;
        }
    }
```

Código 9. Función RPC para Cambiar Color

El primer código se encarga de la invocación de la función RPC. El usuario que se encargue de la ejecución tendrá una resolución de inmediato al igual que los usuarios que se encuentren dentro del ambiente virtual. Esto se logra debido a que estas llamadas a función son almacenadas en un buffer, y en cuanto un nuevo usuario ingrese al ambiente, este buffer de llamados a funciones RPC es resuelto y todas las funciones acumuladas se resuelven.

El segundo código es donde asignamos le asignamos a una función la modalidad de RPC con solo agregar la propiedad [PunRPC] al método y de esta forma, al ser llamado se ejecutará remotamente en todos los usuarios de la aplicación.

6.7.3 Modificación de Atributos en las Figuras

Durante el Análisis de Requerimientos se mencionó que uno de los motivos por los que se eligió C# fue porque era un lenguaje de programación orientado a objetos. Cada figura que se inserta en el ambiente es controlada desde el código por una instancia del objeto GameObject y cualquier cambio que se realice desde el

código hacia la figura que se muestra se puede realizar por medio de la modificación de los atributos.

```
public void cambiarTamano(float sliderposition)
    {
        if (pilaGameObjects.Count != 0)
        {
            Vector3 vectp = pilaGameObjects.Peek().transform.localScale;
            if (pilaGameObjects.Peek().name != "Pillar stage prefab(Clone)")
            {
                pilaGameObjects.Peek().transform.localScale = new Vector3(sliderposition, vectp.y,
vectp.z);
            }
            else
            {
                pilaGameObjects.Peek().transform.localScale = new Vector3(sliderposition / 10,
vectp.y, vectp.z);
            }

        }
    }
```

Código 10. Modificar Ancho de la Figura

Como se ve en el Código 10, el ancho de la figura se ve modificado por la llegada de un parámetro recibido. De la pila de trabajo extraemos la última figura insertada, y llamamos a uno de sus atributos, `transform`. Posteriormente, esta clase tiene como atributo `localScale`, y al modificar este parámetro se observará un cambio en el GameObject que se encuentra en la cima.

Usando este mismo método, se es posible modificar diferentes atributos de las figuras, tales como su posición, tamaños, color, etcétera. Por último, es importante mencionar que todos los atributos de la clase GameObject son públicos, por lo cual los se pueden modificar directamente del editor.

6.7.4 Borrar, Guardar y Cargar

Finalmente, las últimas características del Framework fueron las opciones de borrar todo, guardar y cargar. Cuando se habló de la pila de trabajo con la que funciona el Framework, se mencionó que cuando se desapila un GameObject esta figura es eliminada, entonces, si se quería eliminar todas las figuras teníamos que eliminar elemento por elemento de la pila, esto era muy incómodo al utilizar la aplicación, además de que solo se pueden eliminar los objetos que creo el usuario durante su turno, por lo que se creó un método en para escanear todas las figuras, y ser capaz de borrarlas. Un punto importante a aclarar, es que el único que puede borrar todas las figuras es el Administrador.

```
    public void borrarTodo()
    {
        ChatGui myInstance = gameObject.GetComponent<ChatGui>();
        if (myInstance.esAdministrador && myInstance.esModelador)
        {
            foreach (GameObject obj in GameObject.FindObjectsOfType(typeof(GameObject)))
            {
                if (obj.name == "cubee(Clone)" || obj.name == "spheree(Clone)" || obj.name ==
"cilinderr(Clone)" || obj.name == "capsulee(Clone)" || obj.name == "Pillar stage prefab(Clone)")
                {
                    PhotonNetwork.Destroy(obj);
                }
            }
            pilaGameObjects = new Stack<GameObject>();
        }
    }
```

Código 11. Borrar Todas las Figuras

En el código 11, podemos ver el método borrarTodo, el cual utiliza el método `GameObject.FindObjectsOfType(typeof(GameObject))`, que nos regresa una lista con todos los GameObjects. Posteriormente se itera cada objeto de esta lista y se destruyen aquellos objetos que fueron creados por medio de nuestra inserción de Photon.

De igual forma, para guardar todas las figuras creadas se realizó una función similar al código anterior mostrado, con la diferencia de que en lugar de usar la

función `PhotonNetwork.Destroy()` se guardaron todas las características necesarias para insertar la figura otra vez, como el tamaño, la posición, la rotación y el color. El código siguiente muestra cómo se recuperaron todos los atributos necesarios para reproducir nuevamente la figura y se almacenaron en listas pertenecientes a una instancia llamada infos de la clase ObjetosGuardar.

```
infos.namet.Add(obj.name.Substring(0, obj.name.Length - 7));


infos.color.Add(obj.GetComponent<Renderer>().material.GetColor("_Color").ToString());

                    Vector3 vectp = obj.transform.position;

                    infos.xp.Add(vectp.x);

                    infos.yp.Add(vectp.y);

                    infos.zp.Add(vectp.z);

                    Vector3 vecr = obj.transform.rotation.eulerAngles;

                    infos.xr.Add(vecr.x);

                    infos.yr.Add(vecr.y);

                    infos.zr.Add(vecr.z);

                    Vector3 vecs = obj.transform.localScale;

                    infos.xs.Add(vecs.x);

                    infos.ys.Add(vecs.y);

                    infos.zs.Add(vecs.z);
```

Código 12. Recuperación de Atributos del GameObject

Una vez recorrida toda la lista, se procedió a convertir la instancia de ObjetosGuardar a un JSON que se guarda en un archivo para poderse cargar después. A continuación, se muestra la clase `ObjetosGuardar` desarrollada.

```
public class objetosJuegoGuardar

    {

        public List<string> namet = new List<string>();

        public List<string> color = new List<string>();

        public List<float> xp = new List<float>();

        public List<float> yp = new List<float>();

        public List<float> zp = new List<float>();

        public List<float> xr = new List<float>();

        public List<float> yr = new List<float>();

        public List<float> zr = new List<float>();

        public List<float> xs = new List<float>();
```

```
        public List<float> ys = new List<float>();
        public List<float> zs = new List<float>();
    }
```

Código 13. Clase Pública objetosJuegoGuardar

El siguiente JSON nos muestra cómo se ve guardado un escenario donde ingresamos nuestras cuatro figuras básicas en el ambiente:


```
{
  "namet": [
    "capsulee"
  ],
  "color": [
    "RGBA(0.423, 0.179, 0.934, 1.000)"
  ],
  "xp": [
    0.10000000149011612
  ],
  "yp": [
    0
  ],
  "zp": [
    1
  ],
  "xr": [
    0
  ],
  "yr": [
    0
  ],
  "zr": [
    0
  ],
  "xs": [
    0.30000001192092896
```

```
  ],
  "ys": [
    0.30000001192092896
  ],
  "zs": [
    0.30000001192092896
  ]
}
```


Código 14. JSON generado de la instancia de ObjetosJuegoGuardar

6.7.5 Importación de Figuras 3D en Unity

Unity tiene muchas herramientas muy útiles a la hora de realizar diseños, sin embargo, no tiene el mismo potencial que otros programas enfocados totalmente al diseño en 3D como AutoCad. Es por esto que en Unity es posible importar diseños de otro software en formato FBX.

Para importar un diseño a un proyecto de Unity se deben seguir los siguientes pasos:

1. Ubicar el archivo FBX que se va a importar
2. Copiar el archivo dentro de la carpeta de nuestro proyecto
3. En el explorador de Unity seleccionar la opción Refresh
4. Ubicar la figura importada y agregarla a un Prefab

Después de realizar estos pasos se podrán crear instancias de esta figura en el entorno de Unity.

En el proyecto se importó el diseño de una viga, la cual se definió como figura básica y se puede usar junto con las demás figuras.

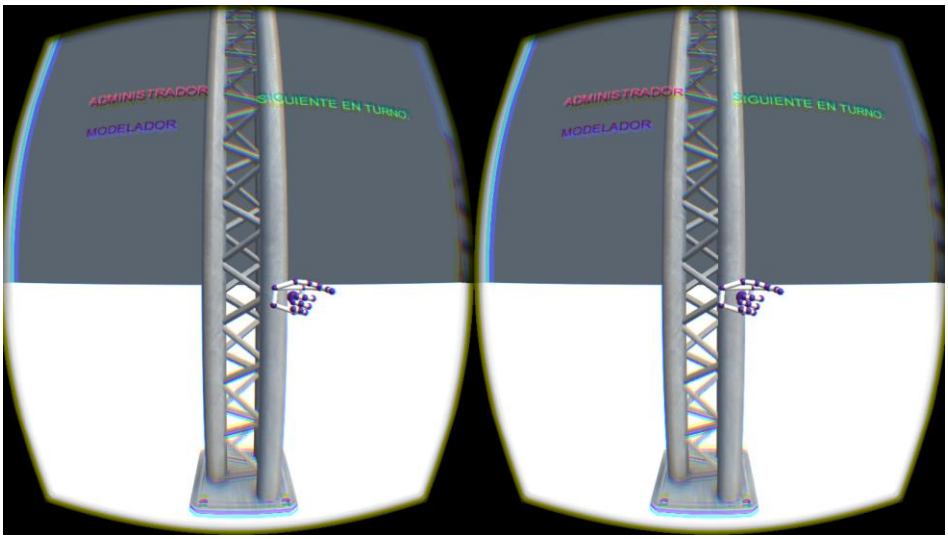


Figura 64. Figura Virtual Viga

6.8 Creación de la Aplicación

Ya una vez creados los Scripts necesarios para la aplicación, el siguiente paso fue la integración entre toda la parte gráfica con la lógica del Framework. Todos los scripts creados pueden ser activados presionando botones o por medio de gestos, si estos son agregados como componente de la figura.

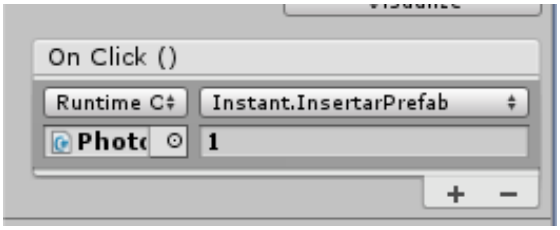


Figura 65. Método Agregado a un GameObject

Todos los scripts se agregaron en los diferentes GameObjects que se encuentran en todas las figuras. Es importante mencionar que solo se pueden agregar métodos públicos, además, es posible ingresarle los parámetros desde la interfaz, esto para no reproducir código innecesario.

Para toda la creación de nuestra interfaz, básicamente se crearon dos paneles en cada palma de las manos. En la mano derecha se activa al voltear el puño hacia arriba y en la otra al extender la palma completa.

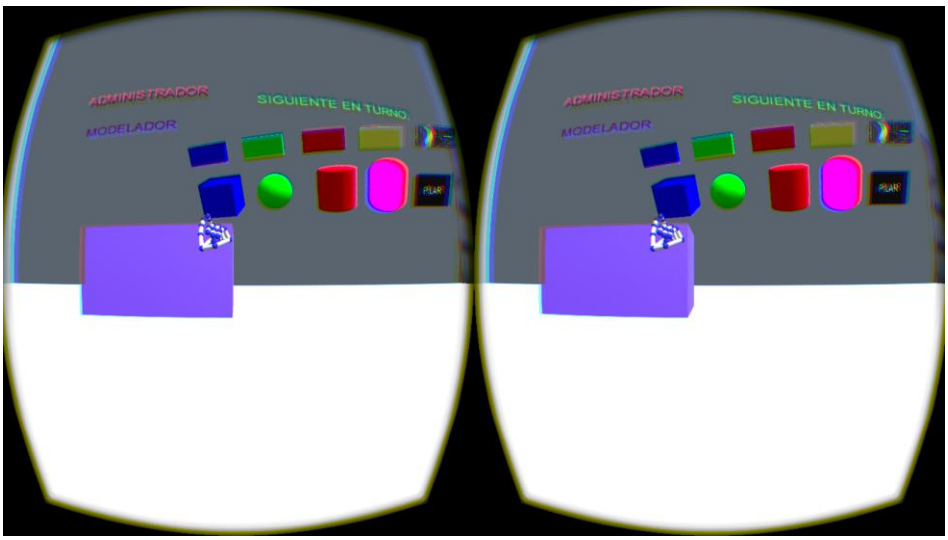


Figura 66. Panel Mano Izquierda

En esta mano se agregaron los siguientes métodos:

- Insertar figuras básicas
- Insertar un pilar (Figura extra)
- Borrar Todas las Figuras

Este panel solo será mostrado cuando el usuario este en modo Modelador. En caso de ser Observador, este panel será invisible.

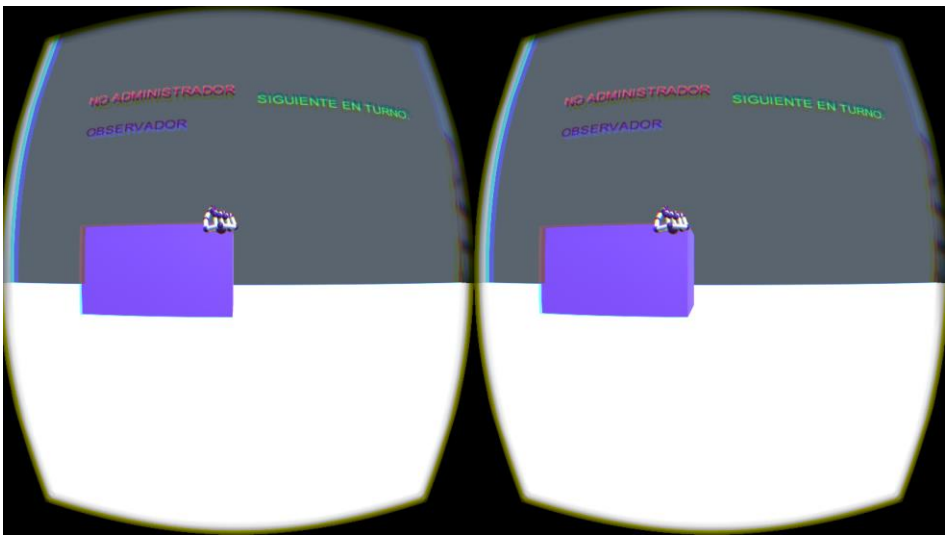


Figura 67. Mano Panel Izquierda en Modo Observador

En lo que respecta a la mano derecha, se creó el siguiente panel:

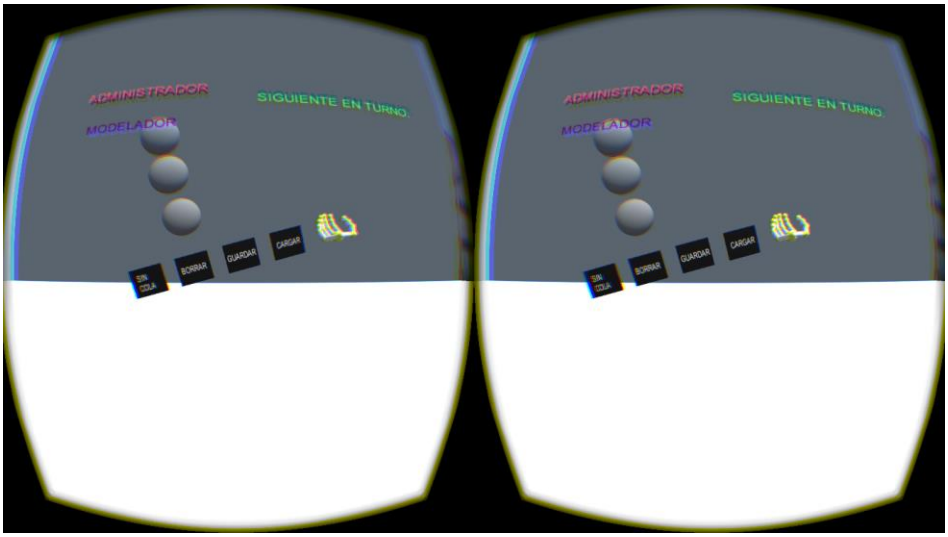


Figura 68. Panel Mano Derecha

A diferencia del panel anterior, este panel será mostrado en cualquier modo que este el usuario. Sin embargo, todos los objetos estarán inhabilitados de realizar cualquier acción con excepción del botón de solicitar turno/ceder turno. Los siguientes eventos se agregaron a las diferentes figuras:

- Cambiar ancho, largo, alto por medio de las barras superiores.
- Borrar última figura.
- Guardar o cargar figura (sólo está habilitado si el usuario es Administrador)
- Pedir/Ceder turno.

En la parte superior de la vista se puede observar el estado del usuario. Por defecto, el primero en ingresar toma el turno de administrador y empieza como modelador. Posteriormente, los distintos usuarios que pidan turno serán encolados y esta cola se muestra en la parte derecha de color verde.

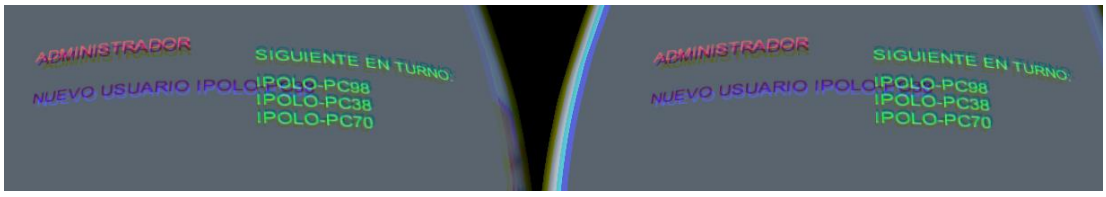


Figura 69. Estado del Usuario y Cola de Turnos de Forma Gráfica

Los siguientes usuarios tendrán el estado de Observadores y su vista será la siguiente.

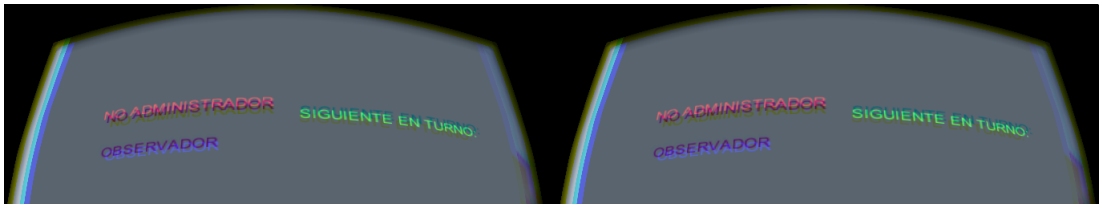


Figura 70. Estados de Control de un Nuevo Usuario

Finalmente, existe un botón para pedir o ceder turno y dependiendo del estado del usuario este tendrá los siguientes estados:

- Pedir Turno: El usuario esta en modo observador. Al ser presionado se pide turno al modelador.
- Turno Pedido: El botón está inhabilitado porque un usuario en modo observador ha pedido el turno.
- Ceder Turno: El usuario esta en modo modelador y existen usuarios en la cola de turnos en espera. Al ser presionado el usuario entra en modo observador y el usuario que es desencolado se convierte en modelador.

Unity tiene la capacidad de crear diferentes subproyectos en un mismo proyecto y son conocidos como escenas. Estas escenas pueden interactuar entre ellas para cambiar una a la otra, pasar valores u objetos. Como última actividad, creo un menú inicial que se activa la escena principal por medio de presionar un objeto.

Figura 71. Escena del Menú Principal

Una vez teniendo todas las clases y verificando que todos los scripts compilen, se procedió a crear el archivo ejecutable, para esto se ingresó a File -> Build Settings. En esta parte veremos todas las escenas que se tienen y seleccionaremos cuales se compilarán.

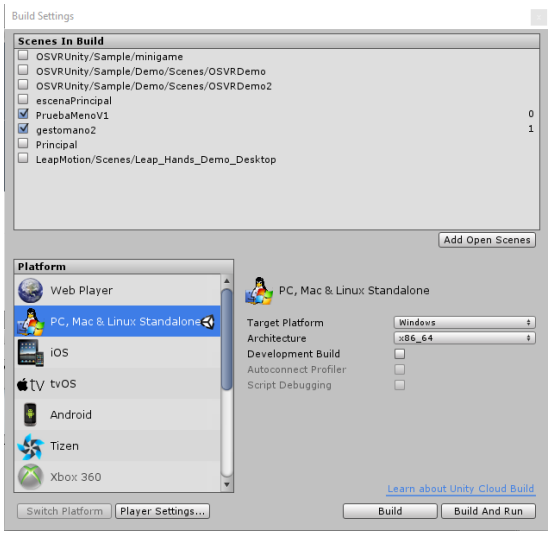


Figura 72. Compilación del Proyecto

Una vez compilado se nos generará un .exe junto con otro ejecutable para que la aplicación sea completamente funcional con Oculus Rift.

Capítulo 7: **Pruebas y Resultados**

En este capítulo se documentan las pruebas a las que fue sometido el Sistema para validar su funcionamiento, cuantificar su rendimiento y verificar el cumplimiento del objetivo general del proyecto. El sistema puede funcionar con un solo usuario como con múltiples, se harán pruebas con ambos para ver su funcionamiento.

7.1 Configuración Inicial

Antes de iniciar la aplicación de Framework Cooperativo de Realidad Virtual es importante estar colocado en la posición correcta como se puede observar en la figura siguiente:

Figura 73. Posición Inicial de la Aplicación

Tendremos que tener colocado el dispositivo Oculus en nuestros ojos, además de tener el controlador Leap Motion en una posición plana debajo de nuestras manos para que nuestras manos sean detectadas correctamente.

Posteriormente, se deberá tener conectados todos los dispositivos en la computadora en la que se iniciará la aplicación. Se harán las pruebas iniciales que

se mencionaron en el Capítulo 6 tanto para Leap Motion como para Oculus Rift para verificar que todos sus servicios funcionen correctamente y sea posible trabajar con ellos sin ningún problema. También es importante revisar que se tenga una conexión con el suficientemente ancho de banda para poderse conectar a nuestro servidor virtual, ya que sin este la aplicación no funcionará.

Ya una vez colocados en una posición correcta y con nuestros dispositivos funcionando correctamente, podremos comenzar nuestra aplicación.

7.2 Modo Local

La aplicación inicia con una pequeña escena en donde podemos ver el nombre del proyecto, los integrantes y los asesores. Para iniciar el escenario principal será necesario picar el botón de Unirse a Modelo.

Figura 74. Menú Incial

Ya dentro del escenario principal será posible ver alrededor de éste girando la cabeza a cualquier dirección, La aplicación responderá dándonos una visión de todo lo que nos rodea.

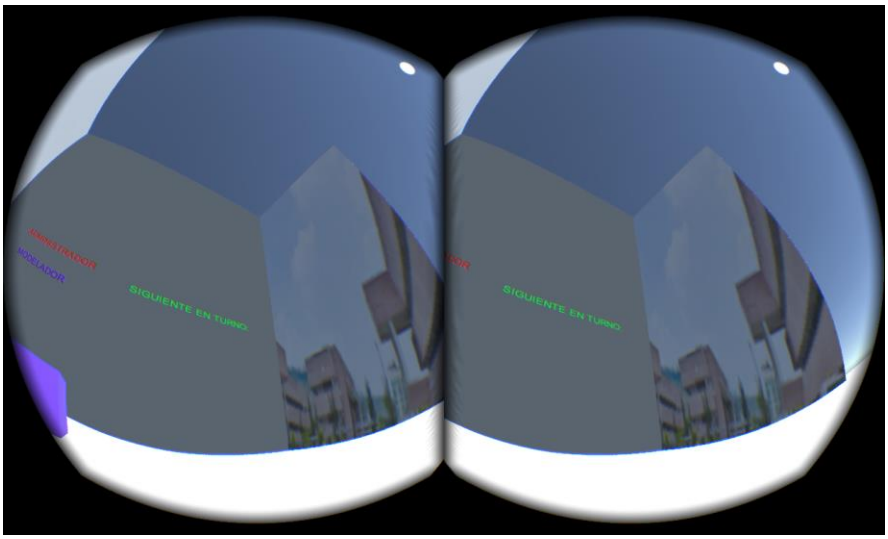


Figura 75. Visión del Escenario Principal

Ya dentro del escenario virtual ingresaremos nuestra primera figura mediante el ingreso de este gesto:

Figura 76. Insertar Cuadrado

En el escenario virtual se podrá observar de la siguiente forma además de poder ver nuestro primer cuadrado ingresado:

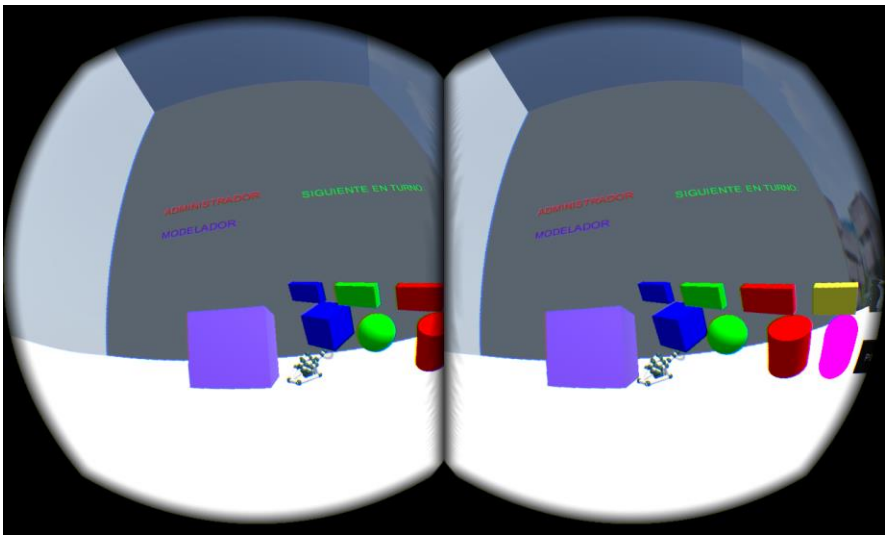


Figura 77. Ingreso de un Cuadrado

Igualmente podremos ingresar las distintas figuras por medio de señalarlas con un toque de esta forma:

Figura 78. Ingresar Figura de la Paleta de Figuras

Y a continuación podremos ingresar todas las figuras dentro del escenario:

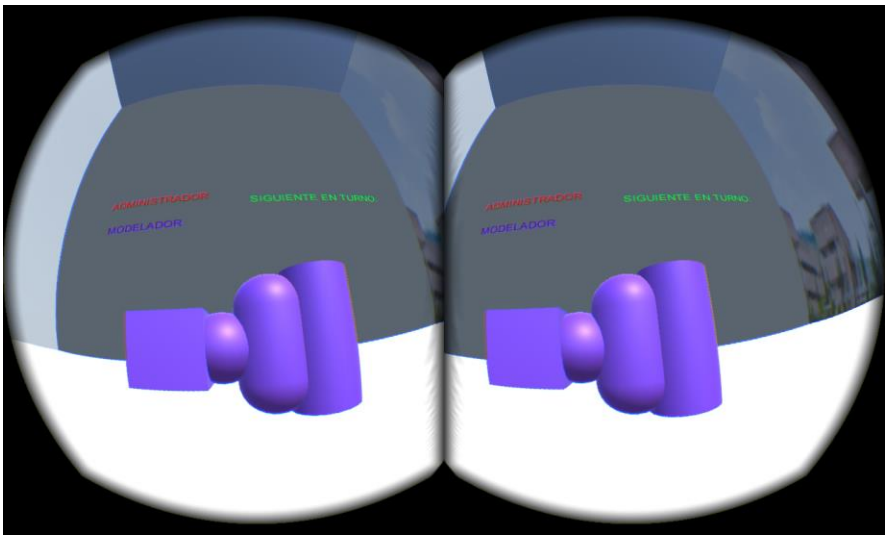


Figura 79. Cuatro Figuras Básicas Ingresadas

Como extra, se creó la figura Pilar la cual se puede ingresar usando el botón “Pilar”, y a ésta se le podrán realizar las mismas modificaciones de las figuras básicas.

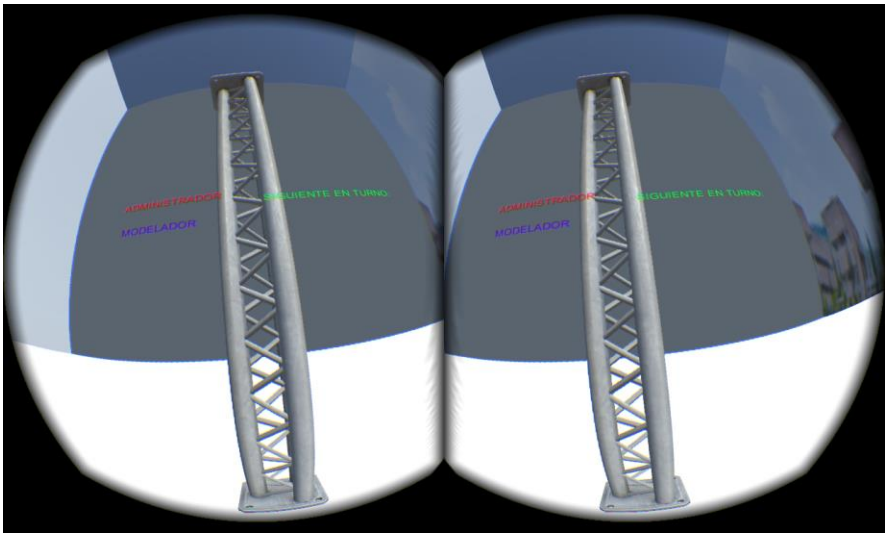


Figura 80. Pilar Ingresado al Ambiente

Todas estas figuras se van agregando a una Pila como ya se mencionó anteriormente y la figura que este en la punta será la única a la que se le podrá modificar sus características. A continuación, se muestra como se cambió el color de un Cubo ingresado a verde:

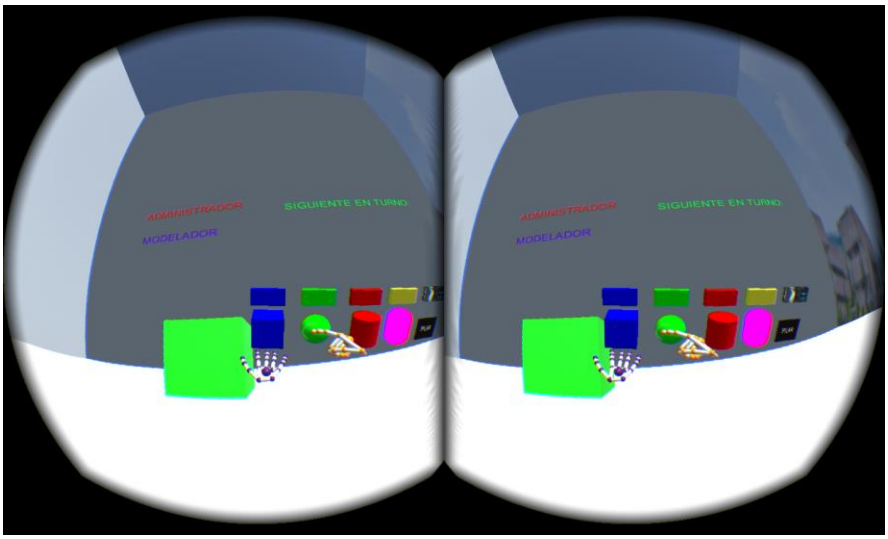


Figura 81. Cubo a Color Verde

Todas las figuras pueden cambiar de color eligiendo alguno de los de la paleta:

- Azul
- Verde
- Rojo
- Amarillo

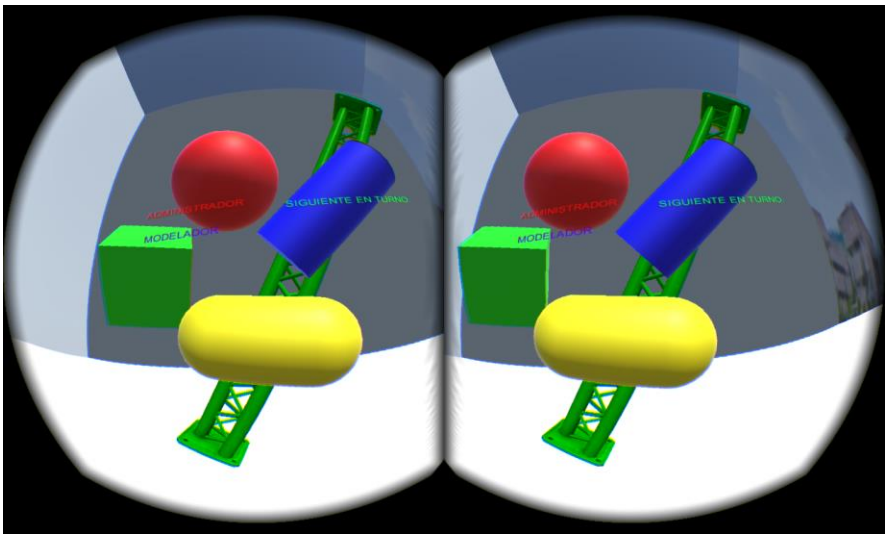


Figura 82. Multiples Figuras con Distintos Colores en el Escenario

Finalmente, es posible eliminar todas las figuras mediante el cuadro negro a lado de la paleta de figuras.

Una figura ingresada será puesta en una posición por defecto por configuración, sin embargo, un usuario es capaz de modificar la posición de este objeto manualmente usando cualquiera de las dos manos, izquierda o derecha, mediante el siguiente gesto:

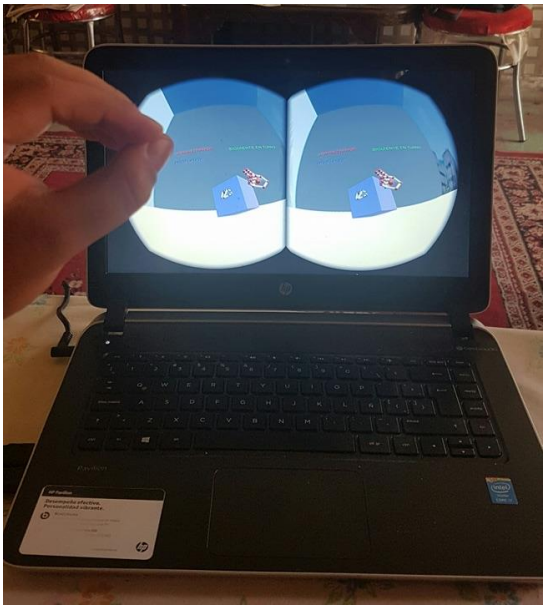


Figura 83. Gesto de Movimiento

Mediante gesto podremos rotar la figura y cambiar su posición en el escenario. Si al tener una figura "sujetada" mediante este gesto y la mano es quitada de la visión del sensor, esta figura desaparecerá, por lo que será importante quitar la mano y volverla a poner para que esta vuelva a aparecer. De igual forma, este gesto está muy sujeto a la cantidad de luz que hay a nuestro alrededor y el campo de visión del Leap Motion para que pueda funcionar correctamente.

Como se mostró en la figura 74, mediante la extensión de la palma izquierda se es posible ver una paleta de figuras y sus colores. Esto aplica usando la palma derecha en done tendremos otro menú para hacer distintas acciones:

- Guardar Modelo: Se podrá guardar un modelo que se encuentre en vista, y reemplazará al anterior que se haya guardado. Este será almacenado en un archivo temporal en la carpeta de la aplicación en un formato JSON. Solo el primer usuario que acceda a la aplicación podrá guardar los archivos.
- Cargar Modelo: Se cargará un modelo guardado que se haya almacenado en un archivo JSON. Antes de insertar las nuevas imágenes se eliminarán todas las figuras que se encuentren en el escenario. Posteriormente se cargarán las nuevas imágenes. Es posible archivos JSON manualmente

para generar imágenes de forma más exacta y poderlas visualizar con la aplicación. Al igual que con la opción de Guardar un Modelo, sólo el Administrador podrá activar este método.

- Borrar: El usuario en turno podrá borrar la última figura que ingreso durante su turno mediante la paleta de figuras. En caso de estar en modo observador o no haber ingresado ninguna figura en su actual turno, el botón no hará nada.
- Sin cola: Cuando la aplicación se use con solo un usuario, el botón permanecerá sin cola y no hará ninguna acción.

Figura 84. Acciones Mano Derecha

De igual forma, es posible modificar el ancho, alto y largo de la última figura ingresada mediante la utilización de las distintas barras de trabajo que se encuentran en la mano derecha. Esto con el fin de crear otras figuras basándonos en nuestras figuras básicas:

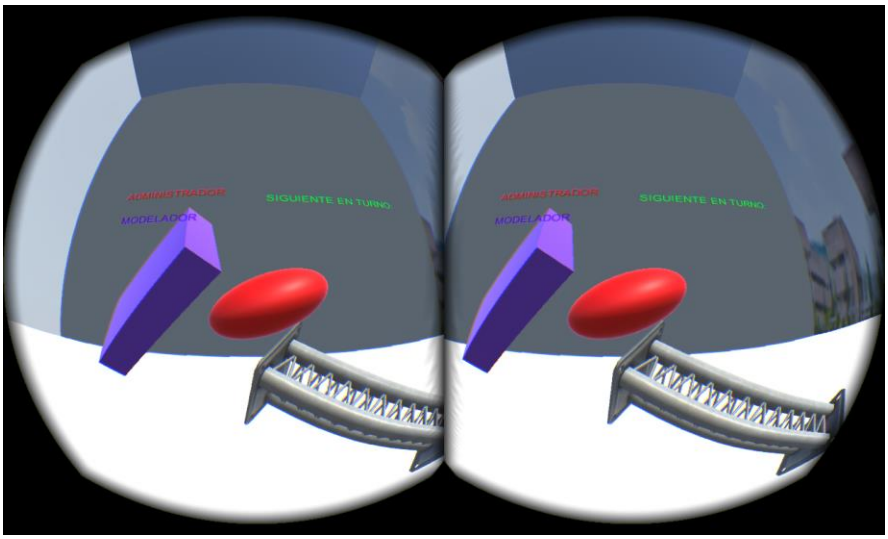


Figura 85. Modificaciones en Ancho, Largo y Alto

7.3 Cooperatividad

Después de haber revisado las funcionalidades de la aplicación con un solo usuario, se revisarán las acciones al tener múltiples clientes conectados. Primeramente, se ingresará un nuevo usuario a la aplicación, del lado del Administrador nos llegará el aviso.

Figura 86. Aviso Nuevo Usuario

La aplicación toma como usuario el nombre que usa la computadora más un número aleatorio. De esta forma la aplicación distingue a los distintos usuarios dentro de la aplicación.

Desde la vista del Nuevo Usuario, será posible ver sus características:

- No Administrador: Esto nos indica que no es el primer usuario en accesar al Framework, por lo tanto, no puede guardar ni cargar figuras.

- Observador: Hasta que no se le de turno, el usuario no puede insertar figuras.

Figura 87. Vista desde el Nuevo Usuario

Este nuevo usuario tendrá la posibilidad de pedir turno mediante la utilización del botón pedir turno o presionando la tecla 'S'.

Figura 88. Encolamiento de Turnos

Posteriormente, el usuario en modo Modelador podrá ceder el turno decolando la Cola de Turnos para que el nuevo usuario pueda empezar a crear figuras. El sistema es capaz de soportar hasta cinco usuarios conectados. En caso de excederse en ese número, la aplicación quedará bloqueada y no será utilizable. En la siguiente imagen se muestra múltiples usuarios esperando turno para empezar a modelar:

Figura 89. Múltiples Usuario Encolados

En caso de que el que se encuentre en modo Modelador salga abruptamente sin ceder el turno, tendremos un Deadlock, donde múltiples usuarios esperarán de un de un recurso que nunca se liberará.

Todas las figuras para modelar serán visualizadas en los distintos usuarios que estén dentro de la aplicación, las cuales son:

- Cubo
- Esfera
- Cilindro
- Capsula
- Pilar

Las demás figuras tales como menús, las manos y paredes solo funcionan de manera local y como se observa en la siguiente figura, todo el entorno esta sincronizado, con excepción de las manos y el texto de control.

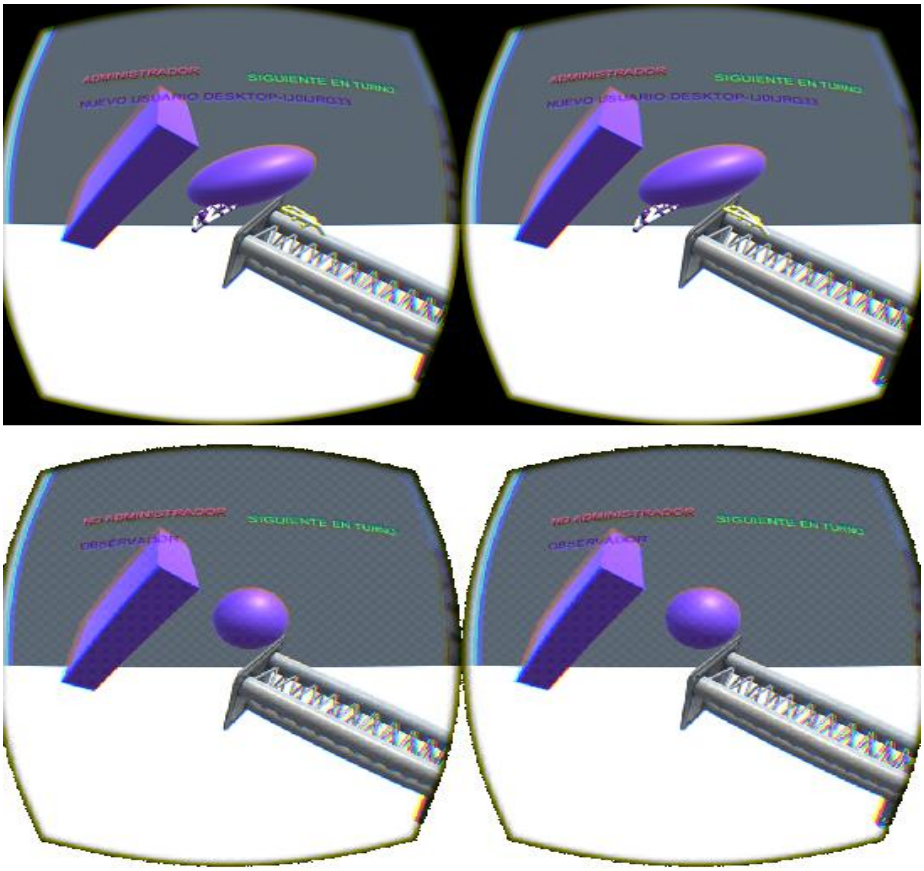


Figura 90. Distintos Usuarios Dentro del Mismo Entorno

Es importante mencionar que cuando ingrese un nuevo usuario en un ambiente con figuras ya creado, su ambiente será sincronizado y se le ingresarán todas las figuras que se habían estado haciendo previamente junto con todas sus características. Este principio aplica cuando el administrador carga una figura y este cambio es reproducido en todos los usuarios. Algunas ocasiones por caídas en el Internet es posible tener fallos cuando ingresa un usuario, en caso de tener una aplicación bloqueada y que no haya avisado a los distintos usuarios de que hay un nuevo integrante, será necesario solo cerrar la aplicación y volverla a abrir.

Todos los cambios son visualizados correctamente, aunque existe un tiempo de latencia que podemos depreciar.

7.4 Fiabilidad de los Gestos

La única forma de ingresar datos a la aplicación es por medio de la generación de gestos que ya se definieron en capítulos anteriores, por lo tanto, se hicieron pruebas por gesto para ver que tan confiables son:

- Palma hacia arriba: Este gesto es muy confiable ya que el controlador Leap Motion lo puede detectar sin ningún problema. Es importante mantener cierta distancia entre las manos para que no se generen figuras o cambios que no se requieren en este momento. También es importante que al voltear la palma izquierda hacia arriba, esta la tengamos cerrada, ya que si no entra en conflicto con el gesto extender dedo y nos generará cubos indeseados.
- Extender Dedo Izquierdo: Para aumentar la fiabilidad de este gesto, es importante tener bien cerrada la palma hacia arriba, posteriormente hay que extender con firmeza el dedo índice hasta que suene el sonido de ingreso de una figura y regresarlo a su posición original. Este gesto puede fallar si no se extiende correctamente el dedo índice, además de que debe haber suficiente iluminación para que el Leap Motion lo lea.
- Tocar Objeto: Es un gesto muy confiable, además que al fallar es posible hacer un reintento de forma muy sencilla. En algunas ocasiones se pueden insertar figuras dobles por lo que será necesario apoyarnos en el botón de borrar para eliminar figuras indeseadas.
- Agarrar Objeto: Es el gesto menos confiable de todos ya que depende completamente de la iluminación. Además, al controlador le cuesta mucho ver cuando se suelta una figura por la posición del Leap Motion, por lo que, en la mayoría de los casos, será muy difícil dejar el objeto en las posiciones

exactas deseadas. Este gesto aumenta de fiabilidad si se usa el Leap Motion montado sobre el Oculus Rift, aunque en este proyecto no se implementó de esa manera.

- Modificar Tamaño: Para la correcta detección de este gesto, es importante ayudarnos mucho de los círculos que se ven flotando sobre la palma, ya que el dedo debe tocarlos correctamente. Se mejora el desempeño haciendo movimientos suaves y apoyándonos en los sonidos generados por el Leap Motion.

Por último, es importante aclarar que el Leap Motion debe estar correctamente colocado sobre una superficie plana mirando hacia arriba y checar que ningún objeto obstruya su campo de visión, esto para aumentar la fiabilidad de los gestos.

7.5 Uso de Recursos

Para utilizar esta aplicación, es necesario tener los suficientes recursos disponibles para que esta funcione correctamente ya que básicamente ocupa tres servicios:

- Gráficos
- Servicios Leap Motion
- Controlador Oculus Rift

Se recomienda cerrar aplicaciones que ocupen mucha memoria RAM, ya que la aplicación Framework Cooperativo de Realidad Virtual para Modelado en 3D requiere gran cantidad de recursos. En caso que la computadora no tenga la suficiente memoria o no tenga la capacidad, se experimentaran imágenes pausadas y la detección de movimientos será muy lenta, por lo que la aplicación no se podrá utilizar. Todas las máquinas usadas en las pruebas tenían Windows 10, al menos 8 Gb de RAM y una tarjeta de video Intel básica, además, en la que se corrieron los tres servicios simultáneamente tenía un procesador Intel i7 con 2.6 GHz. Esto nos muestra que no es necesario tener computadoras muy poderosas, pero si tener algunas capacidades mínimas.

Finalmente, podemos decir que los siguientes recursos son necesarios para utilizar la aplicación:

- Una computadora con 2 Puertos USB 2.0 y una entrada HDMI
- Conexión eléctrica para los Oculus Rift
- Tener instalado los drivers para Oculus Rift y Leap Motion

7.6 Validación

En base a las pruebas realizadas en la finalización de este proyecto, podemos demostrar a continuación como se cumplió con cada uno de los objetivos, tanto particulares como generales:

- Se desarrolló una aplicación de Realidad Virtual cooperativa para diseño de escenarios en 3D partiendo de algunas formas básicas, donde los distintos usuarios son capaces de modificar y observar los cambios en el diseño.
- Se desarrolló un programa en el Oculus para que varios usuarios sean capaces de observar un escenario virtual en tres dimensiones por medio del diseño y creación de figuras en el motor gráfico Unity, además del desarrollo de scripts de sincronización con los visores de Realidad Virtual
- Se desarrolló un programa que detecta los movimientos de las manos sobre las imágenes virtuales utilizando el Leap Motion por medio de Scripts en C# a través del API Orion para Unity. Se implementaron gestos para crear las sentencias de control sobre las figuras.
- Se logró la comunicación entre el servidor central y todos los usuarios usando un servicio de Unity llamado Photon en donde todos los modelos creados eran sincronizados entre todos por medio de un sistema de turnos desarrollado en C# a través de mensajes de multidifusión.

Además de cumplir con los objetivos particulares y generales, se creó distintos extras que mejoraron esta aplicación:

- Se creó una figura extra llamada Pilar, con esto se demostró que la aplicación funciona no solo con las figuras básicas, si no con figuras más complejas.
- El sistema soporta hasta cinco usuarios concurrentes en Red en distintas ubicaciones geográficas.
- Es posible guardar y cargar figuras.
- Es posible generar escenarios manualmente por medio de la creación de Scripts JSON.

Conclusión y Aportaciones

La Realidad Virtual con Múltiples Usuarios en la red sigue siendo un reto a superar; sin embargo, en los últimos años la investigación en esta área ha acaparado la atención de la comunidad científica que se dedica al desarrollo de sistemas de comunicación y de software. Se ha demostrado que el uso de esta tecnología tiene potencial para muchas áreas más allá de sólo el entretenimiento.

Derivado de lo que se presenta en este documento se presentan las siguientes conclusiones:

- La creación de aplicaciones de Realidad Virtual por medio de Motores Gráficos es una tendencia hoy en día ya que actualmente existen distintos dispositivos de RV de diferentes rangos de precios. Además de que es posible generar aplicaciones de no tan alto costo por medio de la generación de librerías y drivers.
- El desarrollo de la Realidad Virtual Inmersiva es aún muy costosa, sin embargo, nos ofrece características inigualables de visualización de imágenes.
- Los sensores de movimiento son herramientas que tienen grandes características que aún no son explotadas. Los desarrollos de herramientas por medio de éstas las hacen más inmersivas y a la vez, más fáciles de utilizar.
- La utilización de servicios Serverless son una tendencia hoy en día ya que ahorra a los desarrolladores la creación de servidores dedicados para solo configurar lo necesario para conectar la aplicación. Esto ayuda a reducir costos al desarrollar aplicaciones en la red y facilita su implementación.

De igual forma, a continuación, se presentan las siguientes aportaciones del proyecto de Framework Cooperativo de Realidad Virtual para Modelado en 3D:

- Se desarrollaron scripts que son utilizables hoy en día para conectar Unity 5 con los Oculus Rift DK1 tomando como base scripts diseñados para al versión anterior de Unity.
- Se desarrolló una aplicación que es posible ser utilizada sin tener las grandes características que pide Oculus Rift para un equipo de cómputo.
- Las aplicaciones de Realidad Virtual Cooperativa aún están en una etapa muy joven, por lo que esta aplicación puede ser una gran aportación.
- Se desarrolló la primera aplicación de Realidad Virtual Inmersiva en la carrera de Telemática en UPIITA.
- Se desarrolló la primera aplicación de Realidad Virtual Cooperativa en UPIITA.

Es importante mencionar que para la realización de este Proyecto Terminal se tuvieron distintos impedimentos, tanto de software como hardware, que finalmente fueron superados. Resulto ser una aplicación un tanto costosa de realizar, sin embargo, la aportación que se realiza es muy buena y tiene un gran campo para ser trabajada a futuro.

Finalmente, para la realización de este Proyecto Terminal fueron necesarios los conocimientos de distintas Unidades de Aprendizaje que se enlistan a continuación:

- Algebra Lineal
- Programación Estructurada
- Programación
- Estructura de Datos
- Programación Avanzada
- Teoría de los Circuitos
- Protocolos de Internet
- Aplicaciones Distribuidas
- Bases de Datos Distribuidas
- Redes Inteligentes

Este al ser un proyecto multidisciplinario, es aceptable como un Proyecto de Titulación como Ingeniero Telemático.

Trabajo a Futuro

En el desarrollo de este proyecto se lograron grandes avances, sin embargo, aún hay mucho por desarrollar, así como, mejoras que incrementarían las funcionalidades del proyecto. Se pueden incluir las siguientes actividades como trabajo a futuro:

- Diseñar e incluir más formas relacionadas al diseño industrial, por ejemplo, engranes y tornillos, para crear modelos más enfocados a la industria.
- Migración de los módulos del Kit de Desarrollo de Oculus a la versión comercial. Es importante que esto sólo es realizable teniendo equipos de cómputo con las características necesarias para soportar el software, además del Oculus Rift en su versión final.
- Trabajar con la versión de paga de PUN, la cual entre otras ventajas nos permitiría trabajar con más usuarios simultáneos.

- Desarrollar un algoritmo de compresión para disminuir la cantidad de datos enviados y recibidos, para poder trabajar de forma correcta aún con conexiones a internet de baja velocidad.
- Usar un formato de texto estandarizado para el envio de sentencias de control.
- Implementar nuevas opciones de edición como copiar y pegar figuras.
- Agregar la opción de colocar texturas a las figuras para lograr una visión más realista del modelo.
- Guardar los datos JSON en un repositorio en la nube para que cualquier usuario pueda accesar a él.
- Analizar los datos ingresados por medio de las manos.

La correcta implementación de los incrementos antes mencionados, fortalecerían al “Framework cooperativo de realidad virtual para modelado en 3D” haciéndolo aún más funcional y robusto para tener la capacidad de soportar diseños más complejos.

Referencias

[1] «Daqri» [En línea]. Available: http://daqri.com. Fecha de consulta: 20/10/2015.

[2] Gálvez Mozo, A. Posicionamientos y puestas en pantalla. Un análisis de la producción de sociabilidad en los entornos virtuales. Barcelona: UAB. 2004.

[3] Mistrick, Ivan. Collaborative Software Engineering. Ed, Springer.

[4] «Oculus Rift Development Kit 2» [En línea]. Available: https://www.oculus.com/en-us/dk2/. Fecha de consulta: 20/10/2015.

[5] «Leap Motion Developers» [En línea]. Available: https://developer.leapmotion.com/. Fecha de consulta: 20/10/2015.

[6] «WebSocket Communication» [En línea]. Available: https://developer.leapmotion.com/documentation/javascript/supplements/Leap_JSON.html. Fecha de consulta: 10/04/2017.

[7] Tanenbaum, Andrew S.; Van Steen, Maarten (2002). Distributed Systems: Principles and Paradigms. Prentice Hall.

[8] «Microsoft lab working on multiperson augmented reality» [En línea]. Available: http://www.cnet.com/news/microsoft-lab-working-on-multi-person-augmented-reality. Fecha de consulta: 10/10/2015.

[9] «Con un Kinect y unos Oculus Rift, el IPN te hace viajar por el sistema solar» [En línea] Available: http://www.elfinanciero.com.mx/universidades/con-un-kinect-y-unos-oculus-rift-ipn-te-hace-viajar-por-el-sistema-solar.html Fecha de consulta: 10/12/2016.

[10] «User Guide - GloveOne» [En línea]. Available: https://www.gloveonevr.com/dev/doc/user-guide. Fecha de consulta: 10/05/2016.

[11] «PlayStationVR PlayStation» [En Línea] Available: https://www.playstation.com/en-au/explore/playstation-vr/. Fecha de consulta: 10/05/2016.

[12] «Google Cardboard» [En Línea] Available: https://www.google.com/get/cardboard/. Fecha de consulta: 10/05/2016.

[13] «Orion. Leap Motion Developers» [En línea]. Available: https://developer.leapmotion.com/orion. Fecha de consulta: 14/03/2016.

[14] «Unity 5 vs UnrealEngine 4» [En Línea] Available: https://create3dgames.wordpress.com/2015/09/07/unity-5-vs-unreal-engine-4/. Fecha de consulta: 11/04/2016.

[15] «Scripting in Unity» [En línea]. Available: http://www.3dgep.com/scripting-in-unity-3-5/. Fecha de consulta: 11/04/2016.

[16] «Sitio oficial de Boo»: [En línea]. Available: http://boo-lang.org/. Fecha de consulta: 11/04/2016.

[17] «C# Programming Site»: [En línea]. Available: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx. Fecha de consulta: 11/04/2016.

[18] «Sitio oficial de JavaScript»: [En línea]. Available: https://www.javascript.com/. Fecha de consulta: 11/04/2016.

[19] «Integración en Visual Studio C#» [En línea] Available: http://docs.unity3d.com/es/current/Manual/VisualStudioIntegration.html. Fecha de consulta: 11/05/2016

[20] «Sito oficial de Visual Studio» [En línea] Available: https://www.visualstudio.com/. Fecha de consulta: 11/05/2016

[21] «Network System Concepts» [En línea] Available: http://docs.unity3d.com/Manual/UNetConcepts.html. Fecha de consulta: 10/05/2016

[22] «Object Spawning» [En línea] Available: http://docs.unity3d.com/Manual/UNetSpawning.html. Fecha de consulta: 10/05/2016.

[23] «Photon Unity Networking Intro» [En línea] Available: http://doc.photonengine.com/en/pun/current/getting-started/pun-intro. Fecha de consulta: 10/05/2016

[24] «RPCs And RaiseEvent» [En línea] Available: https://doc.photonengine.com/en-us/pun/current/manuals-and-demos/rpcsandraiseevent. Fecha de consulta: 01/06/2017