



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

Curso Académico 2018/2019

Trabajo Fin de Grado

INTERACCION GESTUAL PARA EL APRENDIZAJE DE POO
CON REALIDAD VIRTUAL MEDIANTE LEAP-MOTION

Autor: Pablo Rodríguez Vicente

Director: Maximiliano Paredes Velasco

A todos mis amigos por hacer de estos años los mejores.

A mis padres por su apoyo.

A mi tutor por darme la oportunidad de este proyecto.

Gracias.

Resumen

En esta memoria se explicará cómo se ha desarrollado una aplicación para el aprendizaje de conceptos de programación orientada a objetos haciendo uso de la tecnología gestual LeapMotion.

La aplicación tiene como objetivo mejorar el aprendizaje de los conceptos en los primeros cursos donde se imparte POO(Programación Orientada a Objetos). Se apoya en interacciones con objetos virtuales y se aleja del planteamiento clásico que se usa en las aulas.

En los primeros cursos de la universidad se trata el concepto de la programación orientada a objetos de manera muy teórica. Esto resulta en muchas ocasiones en problemas por parte del alumno para entender cómo funciona. La aplicación usa representaciones virtuales para transmitir estos conceptos a la hora de ser explicados, de tal manera que el alumno se familiarice mejor con los conceptos al interaccionar directamente con ellos y no ser un mero oyente.

Los conceptos que se practican en las asignaturas de programación orientada a objetos suelen ser objetos, variables, la asignación de objetos a variables y qué es una referencia frente a una variable tipo primitivo.

De esta manera se ha creado una aplicación en la que el usuario no tiene que escribir código para crear las clases ni trabajar con ellas y, se incluyen las líneas de código más importantes para cada acción. Así se consigue que el usuario practique mediante gestos e interacciones naturales lo que más adelante tendrá que hacer en un editor de código.

El desarrollo de la aplicación ha consistido en 2 grande partes. Por un lado tenemos una parte de desarrollo de scripts usando C# en Unity para crear toda la funcionalidad necesaria para cumplir los objetivos de la aplicación.

Por otro lado tenemos una parte de diseño de interfaz. Para esta parte ha hecho falta modelar algunos elementos, para lo que se ha usado Blender. También ha hecho falta crear algunos scripts en Unity para construir la interfaz.

Índice

Capítulo 1: Introducción.....	11
1.1 Motivación.....	11
1.2 Objetivos.....	11
1.3 POOLeapMotion	12
Capítulo 2: Fundamentos	13
2.1 Tecnologías.....	13
2.1.1 Unity.....	13
2.1.2 LeapMotion	15
2.1.3 Lenguaje C#	17
2.1.4 Blender	18
2.1.5 Visual Studio Code.....	19
2.1.6 GitHub.....	21
Capítulo 3: Descripción de la Aplicación	23
3.1 Metodología de trabajo	23
3.2 Inicios del Proyecto	24
3.3 Desarrollo y construcción.....	24
3.3.1 Manager.....	25
3.3.2 Interacción.....	27
3.3.3 Objetos, Variable, Métodos y Atributos.....	33
3.3.4 Sistema de menús	35
3.3.5 Transiciones	35
3.3.6 Diagrama UML	36
3.3.7 Problemas en el Desarrollo de los Scripts	37
3.3.8 Diseño de Interfaz	37
3.3.9 Optimización de Renderizado	39
3.3.10 Accesibilidad	40
3.3.11 Multilenguaje	41
3.3.12 Sonido.....	42
3.3.13 Modelado.....	43
3.3.14 Diagrama de navegación	47
3.3.15 Problemas en el Desarrollo de la Interfaz	47
3.4 Descripción de la herramienta	48
3.4.1 Creación y modificación de clases	48
3.4.2 Creación de Objetos y Variables	56
3.4.3 Inspección de Variables y Objetos	63

3.4.5 Ejecución de Métodos	65
Capítulo 4: Pruebas	69
4.1 Pruebas de caja blanca.....	69
4.2 Pruebas de caja negra	70
4.3 Pruebas de usabilidad	71
Capítulo 5: Conclusiones	73
5.1 Conclusión final.....	74
5.2 Trabajos Futuros	75
Bibliografía.....	77
Apéndices	79
Apéndice A: Guía de uso.....	79
Apéndice B: Guía de Instalación	79

Tabla de Ilustraciones

Ilustración 1: GameObject vacío.....	14
Ilustración 2: GameObject con componentes.....	14
Ilustración 3: LeapMotion en Unity.	16
Ilustración 4: Manos con rig.....	17
Ilustración 5: Blender.	19
Ilustración 6: Visual Studio Code.	20
Ilustración 7: Extensiones en Visual Studio Code.	20
Ilustración 8: Git Extension.....	21
Ilustración 9: Representación de un objeto y Script de interacción.	28
Ilustración 10: Anclajes en el menú de representación (en morado).	29
Ilustración 11: Botón y Script de interacción con botones.....	30
Ilustración 12: Mano cogiendo un objeto.....	30
Ilustración 13: Fotografía de la mano detectada por el LeapMotion cogiendo un objeto.....	31
Ilustración 14: Cartel informativo.	32
Ilustración 15: Fotografía de la mano detectada por el LeapMotion cerca de un objeto.	32
Ilustración 16: Objetos y variables creados en la aplicación.	33
Ilustración 17: Atributos y métodos de un objeto.	34
Ilustración 18: Asset Bundles de la aplicación.....	34
Ilustración 19: Script para las interpolaciones.	36
Ilustración 20: Diagrama UML.	36

Ilustración 21: Movimiento de las manos.	38
Ilustración 22: Calidad de la detección.	39
Ilustración 23: Occlusion Culling desactivado.....	40
Ilustración 24: Occlusion Culling activado.	40
Ilustración 25: Polyglot Tool lista de idiomas.	41
Ilustración 26: Polyglot Tool lista de traducciones.	41
Ilustración 27: Polyglot Tool script de traducción.	41
Ilustración 28: Polyglot Tool script de cambio de idioma.	42
Ilustración 29: Componente AudioSource de Unity.	42
Ilustración 30: Botón Cuadrado.	43
Ilustración 31: Botón Flecha.	44
Ilustración 32: Botón Cruz.	44
Ilustración 33: Botón Tic.....	45
Ilustración 34: Botón modular.....	45
Ilustración 35: Modelo del marco.	46
Ilustración 36: Modelo de la papelera.	46
Ilustración 37: Diagrama de navegación.....	47
Ilustración 38: Menú de creación de atributos.	49
Ilustración 39: Menú de creación de métodos.....	50
Ilustración 40: Menú de creación de clases.....	51
Ilustración 41: Métodos y atributos creados.	51
Ilustración 42: Información textual a la izquierda.	52
Ilustración 43: Introducción del nombre de la clase.	53
Ilustración 44: Creación de un atributo.	53
Ilustración 45: Introducción del nombre del atributo.....	54
Ilustración 46: Confirmación de la creación del atributo.....	54
Ilustración 47: Creación de un método.	55
Ilustración 48: Confirmación del método elegido.....	55
Ilustración 49: Confirmación de creación de la clase.	56
Ilustración 50: Menú de representación de las clases.	57
Ilustración 51: Menú para crear variables a partir de clases.	57
Ilustración 52: Menú con representaciones de variables y objetos creados.	58
Ilustración 53: Creación de un objeto de la clase de la primera fila.	59
Ilustración 54: Creación de una variable de la clase de la primera fila.....	59

Ilustración 55: Selección del nombre de la variable.	60
Ilustración 56: Confirmación de la creación de la variable.....	60
Ilustración 57: Observar los objetos y variables que se han creado.....	61
Ilustración 58: Asignación del objeto a la variable.	61
Ilustración 59: Asignación completada.	62
Ilustración 60: Eliminación de la variable y el objeto.....	62
Ilustración 61: Inspección de un objeto sin estar asignado a una variable.....	63
Ilustración 62: Inspección de una variable sin un objeto asignado.	64
Ilustración 63: Inspección de una variable con un objeto asignado.	64
Ilustración 64: Ejemplo de Submenú.	65
Ilustración 65: Inspección del método para su ejecución.....	66
Ilustración 66: Selección de parámetros de entrada.	66
Ilustración 67: Ejecución y resultado.	67

Capítulo 1: Introducción

1.1 Motivación

El problema del que nace la aplicación es la dificultad de los alumnos para aprender unos conceptos que muchas veces están dados de manera muy teórica. Estos conceptos son difíciles de entender al principio, lo que resulta en un problema a la hora de trabajar en conceptos más difíciles.

Los conceptos que más cuesta aprender son sobre todo, qué es una referencia, que es una asignación y la diferencia entre una variable y un objeto. Muchas veces los alumnos se ven desmotivados por la manera en que se imparten estos conceptos y proporcionando una interacción natural se puede mejorar esta motivación.

La aplicación busca paliar este problema apoyando los conceptos teóricos con interactividad, como si fuera un juego. De esta manera los alumnos no tendrán que memorizar los conceptos si no que trabajaran con ellos de una manera interactiva y divertida.

1.2 Objetivos

Se ha dividido la aplicación en varios objetivos más pequeños para conseguir el objetivo principal de transmitir los conceptos de programación orientada a objetos de manera correcta al alumno. Los objetivos se definen en base a las siguientes funcionalidades de la aplicación desarrollada:

- Crear clases con atributos y métodos, estas clases tienen que ser creadas dentro de la aplicación para facilitar el uso de la aplicación por parte del usuario.
- Modificar las clases, atributos y métodos ya creados.
- Ver el código java que compone la clase que ha creado.
- Crear objetos y variables a partir de una de las clases creadas.
- Coger y mover con las manos las representaciones físicas de objetos, variables, métodos y atributos y presionar con las manos todos los botones en la escena.
- Realizar asignaciones de objetos a variables.
- Inspeccionar el contenido de un objeto o variable creados a partir de una clase.
- Ejecutar los métodos de un objeto.

Se han valorado estos objetivos como los más esenciales de cara a conciliar los conceptos de programación orientada a objetos con la interacción que nos ofrece LeapMotion.

1.3 POOLeapMotion

Finalmente la solución que se propone pasa por la elaboración en Unity de una aplicación que, sin hacer uso de código y de manera interactiva, permita al usuario trabajar con la programación orientada a objetos.

Para la implementación de la interactividad se ha elegido LeapMotion que nos aporta por un precio inferior a los 100€ una tecnología de detección de manos de muy buena calidad y un software muy útil.

Para la implementación se ha elegido Unity, pues LeapMotion nos ofrece todo el software que necesitamos para hacer funcionar el dispositivo y añadir la interacción a la aplicación de manera sencilla y efectiva.

Otras tecnologías que se han usado para el proyecto han sido Visual Studio Code, para la edición de código, GitExtensions junto con GitHub, para el control de versiones, y Blender, para realizar el modelado necesario.

Antes de llegar a la solución arriba descrita se han valorado otras soluciones. La primera consistió en usar java pero se descartó rápidamente pues el soporte de LeapMotion no era suficiente y la creación de la interfaz se complicaba enormemente.

Otra solución que se valoró fue un punto intermedio entre el uso de código y la interacción. Esta opción se abandonó finalmente por la imposibilidad para comunicar Unity con la JVM y por las dificultades que supone para el usuario la introducción del código en la aplicación.

Capítulo 2: Fundamentos

2.1 Tecnologías

En el siguiente apartado se discutirán las tecnologías que han sido usadas para el proyecto y porque se han elegido.

2.1.1 Unity

Unity es un motor de renderizado de gráficos en tiempo real que aparece en 2005 creado por la empresa Unity Technologies. Su uso se ha extendido mucho en los últimos años. Aunque su principal enfoque eran los videojuegos ha demostrado que sirve para la creación de otro tipo de proyectos como por ejemplo animaciones. Unity ofrece el motor de manera gratuita para todas aquellas aplicaciones que no superen uno beneficios de 100000€ anuales con la aplicación realizada. Si se superan se debe comprar una licencia que cuestan 125€/mes, esto lo convierte en un motor ideal para realizar proyectos académicos.

En las versiones más antiguas de Unity no estaba disponible toda la funcionalidad de manera gratuita pero esto cambio en 2016 con la versión 5.6 del motor convirtiéndolo así en un motor gratuito muy potente. Unity se complementa muy bien con otras aplicaciones como por ejemplo Blender facilitando el desarrollo de toda la interfaz 3D. También cabe destacar que Unity posee una serie de paquetes que permiten añadir funcionalidades extra como por ejemplo textos 3D para la interfaz o postprocesos para la iluminación.

Una de las principales ventajas de Unity es que permite crear proyectos para más de 25 plataformas y el motor puede ser usado tanto en Windows como en Linux y Mac. Esto lo vuelve un motor muy versátil a la hora de trabajar con él. Además de varias plataformas Unity también es compatible con varios motores gráficos como OpenGL, Direct3D e interfaces propietarias como por ejemplo la Wii. El lenguaje de programación de Shaders es ShaderLab.

Unity está basado en scripting, que viene dado mediante Mono. Mono es una implementación de código abierto de .NET Framework. El lenguaje ofrecido para la realización de los scripts es C#.

Todos los objetos que se encuentran en una escena se llaman GameObjects. Un GameObject es el objeto más básico que se puede crear en Unity y posee un componente de transformación que almacena la matriz de transformación del objeto y ofrece métodos para trabajar con su posición escala y rotación. En la siguiente imagen se muestra un GameObject básico.

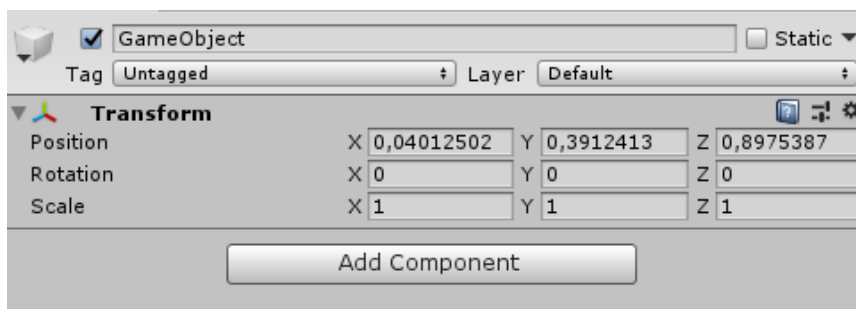


Ilustración 1: GameObject vacío.

A su vez a estos GameObjects se les puede añadir funcionalidad mediante componentes. Unity contiene muchos componentes básicos como cajas de colisión o cuerpos rígidos para el manejo de físicas, pero también contiene otros componentes más complejos como un controlador de personaje. Al mismo tiempo el usuario puede crear sus propios componentes mediante scripts que se añaden al objeto como cualquier otro componente. En la siguiente imagen se muestra un GameObject con componentes de Unity y creados por scripts.

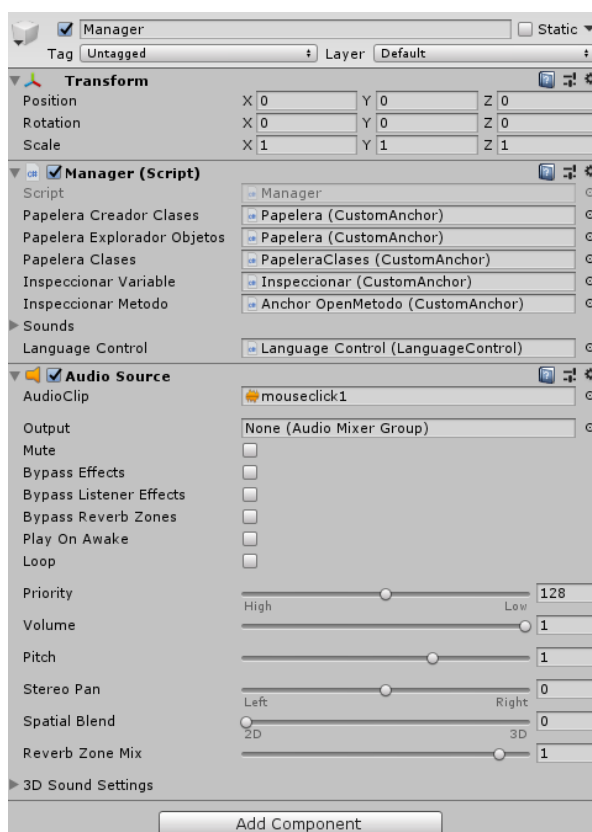


Ilustración 2: GameObject con componentes.

Se ha elegido Unity frente a su principal competidor Unreal porque este último no ofrece funcionalidad como la que ofrece Unity para la creación de aplicaciones y se centra más en el desarrollo de videojuegos. La programación en Unreal se realiza principalmente mediante su sistema de Blueprints, que es un editor por nodos para la creación de comportamientos en Unreal.

La versión de Unity usada para el proyecto es la 2018.3, esta versión incluye cambios en el sistema de prefabs. Un prefab es una copia de un objeto de la escena que es guardado como un archivo que Unity es capaz de instanciar en tiempo de ejecución.

Este nuevo sistema incluye un nuevo flujo de trabajo y la posibilidad de anidar prefabs facilitando así la creación de elementos duplicados, esta característica ha sido de mucha importancia para la interfaz permitiendo crear los botones de manera sencilla y pudiendo modificarlos todos a la vez.

Una funcionalidad muy destacable de Unity que lo hace muy útil a la hora de desarrollar este tipo de proyectos 3D es su sistema de escenas. Una escena es un archivo que contiene toda la información de todos los objetos que se encuentran en la jerarquía y que Unity puede leer y cargar en tiempo de ejecución o de edición de la aplicación.

Este sistema nos permite hacer pruebas sin tener que cambiar cosas en las escenas principales volviéndolo muy útil para probar funcionalidades antes de implementarlas.

2.1.2 LeapMotion

LeapMotion es un sensor de seguimiento de manos desarrollado por la empresa con el mismo nombre. Este dispositivo es la base de este trabajo por el potencial que tiene para el aprendizaje.

Es un dispositivo de pequeño tamaño que se conecta mediante USB. Para realizar el seguimiento hace uso de 2 cámaras infrarrojas monocromáticas y 3 leds infrarrojos y observa un área parecida a una semiesfera con alrededor de 1m de radio.

LeapMotion está pensado principalmente para la RV por las interacciones naturales que permite con los objetos virtuales frente al uso de controladores tradicionales. En su web indican que los mejores usos para el LeapMotion son la rehabilitación y la educación.

Además del hardware la empresa LeapMotion provee a los desarrolladores con una API escrita en C, cuyo uso no se recomienda directamente, y SDK para los principales motores de videojuegos, Unreal y Unity.

Además del SDK que permite el uso dentro del motor, LeapMotion también provee un modulo de interacción para la creación de interfaces 3D propias de la Realidad Virtual. Este motor hace uso del motor de físicas para la detección de colisiones entre las manos y los objetos interactivos.

Se ha elegido esta tecnología para el trabajo por el potencial que posee para el desarrollo de aplicaciones de aprendizaje y por los retos que plantea al ser una tecnología con la que no se ha trabajado antes.

La conexión entre Unity y LeapMotion es muy sencilla basta con bajarse el paquete de Unity de LeapMotion de su página web. Un paquete es un archivo que contiene los assets y puede ser descomprimido e importado por Unity. Una vez descargado el paquete basta con crear un objeto en la escena y añadirle los Scripts encargados de detectar las manos. A partir de ese momento, si el LeapMotion está conectado al ordenador al ejecutar la aplicación, ésta mostrara unas manos.

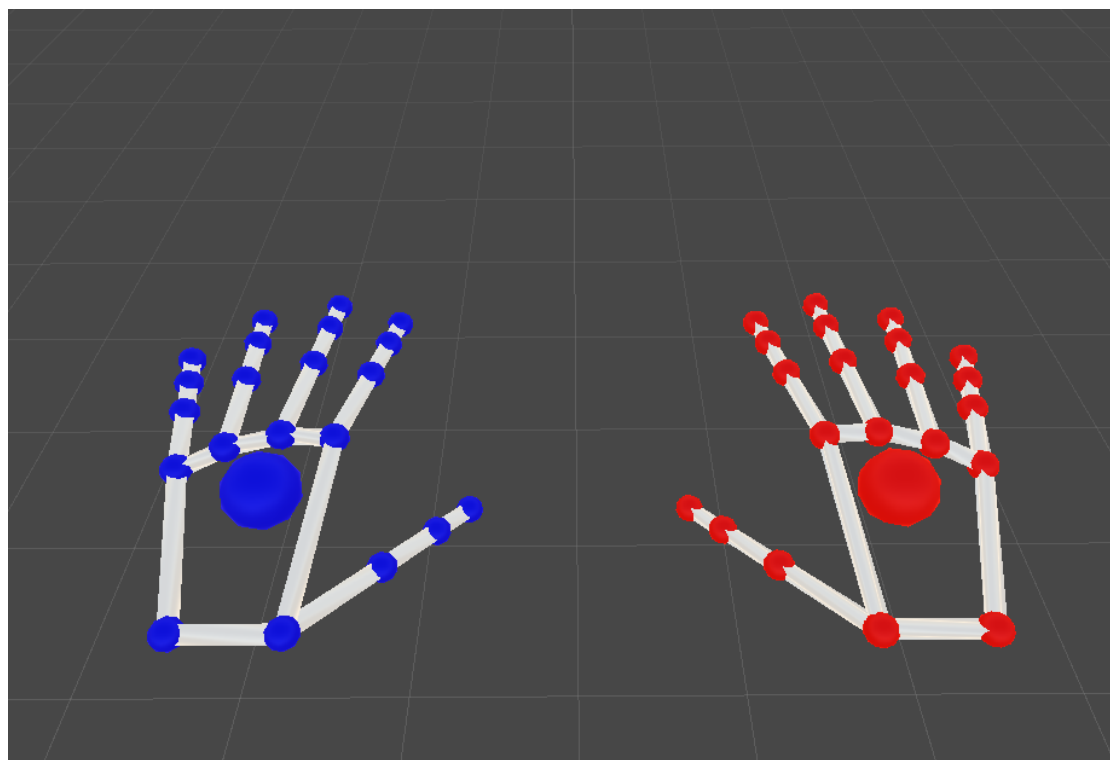


Ilustración 3: LeapMotion en Unity.

Además de esta funcionalidad básica, LeapMotion para Unity ofrece un conjunto de Scripts para posiciones de la mano llamados detectores que lanzaran un evento al colocarse la mano en una posición determinada. Estos detectores no han sido utilizados por su incompatibilidad con la interacción que se busca realizar con los objetos. Se busca una interacción en la que los objetos puedan cogerse y moverse de sitio y estos detectores no nos la aportaban.

También existe un paquete que ofrece interacciones con los objetos de la escena simplemente añadiéndoles el script del comportamiento que queremos.

Otra característica adicional que se obtiene a través de otro paquete es el poder crear manos personalizadas mediante una herramienta de autorig que nos permitirá usar modelos propios si es necesario. En la siguiente ilustración podemos ver unas manos rigeadas que vienen con el paquete de LeapMotion.

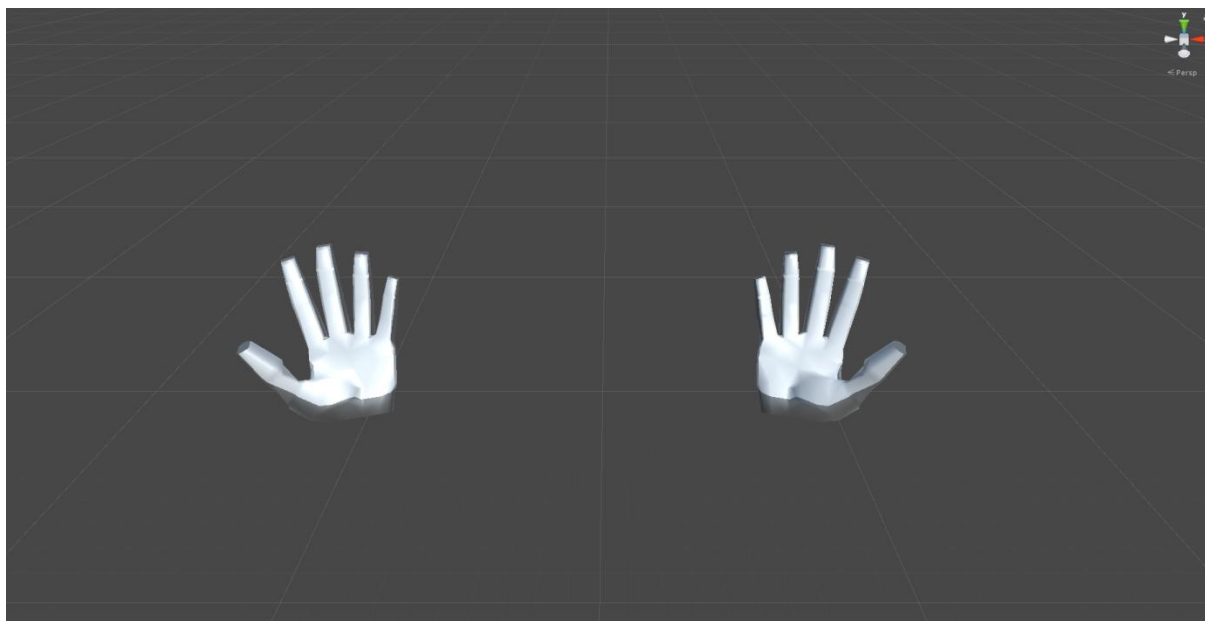


Ilustración 4: Manos con rig.

Además de la posibilidad de crear nuestras propias manos, LeapMotion incluye unas manos autogeneradas con capsulas y esferas y unas manos físicas para la creación de nuestras propias físicas fuera de las interacciones del paquete que LeapMotion provee.

2.1.3 Lenguaje C#

C# es un lenguaje de alto nivel orientado a objetos creado por Microsoft para su plataforma .NET. Es el lenguaje de scripting usado en Unity y posee muchas similitudes con Java pero es más cercano a C++ es su diseño. Con el tiempo se ha ido diferenciando aun más de Java.

Fue creado por Anders Hejlsberg en el año 2000 como un nuevo lenguaje de programación orientada a objetos, se basa en tipos y métodos, al principio se iba a llamar Cool(C-like Object Oriented Language), pero por razones de registro tuvieron que cambiarlo.

En su versión 3.0 C# incluye la extensión Linq. Esta librería usa sintaxis muy parecida a la de SQL para extraer elementos de bases de datos pero también se puede usar para buscar en colecciones. También incluye expresiones lambdas muy útiles a la hora de trabajar con eventos en Unity.

2.1.4 Blender

Blender es una herramienta de código abierto para la creación de contenido 3D desarrollada por Blender Foundation. Ofrece pues, toda la funcionalidad de un programa de modelado como puede ser 3dsMax pero de manera gratuita. Al igual que Unity es multiplataforma lo que lo convierte en una herramienta ideal para trabajar en paralelo a Unity.

Blender aparece en 1998 por parte de la compañía NaN, creada por Ton Roosendaal, que buscaba crear una herramienta profesional de contenido 3D pero de manera gratuita para el usuario frente a las aplicaciones del momento que costaban miles de dólares. La aplicación fue un éxito, pero económicamente no aportaba lo suficiente a los inversores que decidieron cerrar NaN. A pesar de esto la comunidad y Ton no dejaron que Blender cayera en el olvido, para ello Ton creó la organización no lucrativa Blender Foundation que se encargó junto a los usuarios de promocionar y desarrollar Blender.

Actualmente Blender ofrece a parte de herramientas de desarrollo 3D infinidad de funcionalidades creadas por los usuarios. En la actualidad Blender se ha convertido en una opción muy importante a la hora de considerar que software de modelado usar en un proyecto por su potencia y sobre todo por su coste.

Se ha elegido para este proyecto la versión 2.8 de Blender, esta versión se encuentra en beta pero ofrece muchas mejoras en la interfaz que se han visto necesarias para agilizar el proyecto en su fase de modelado.

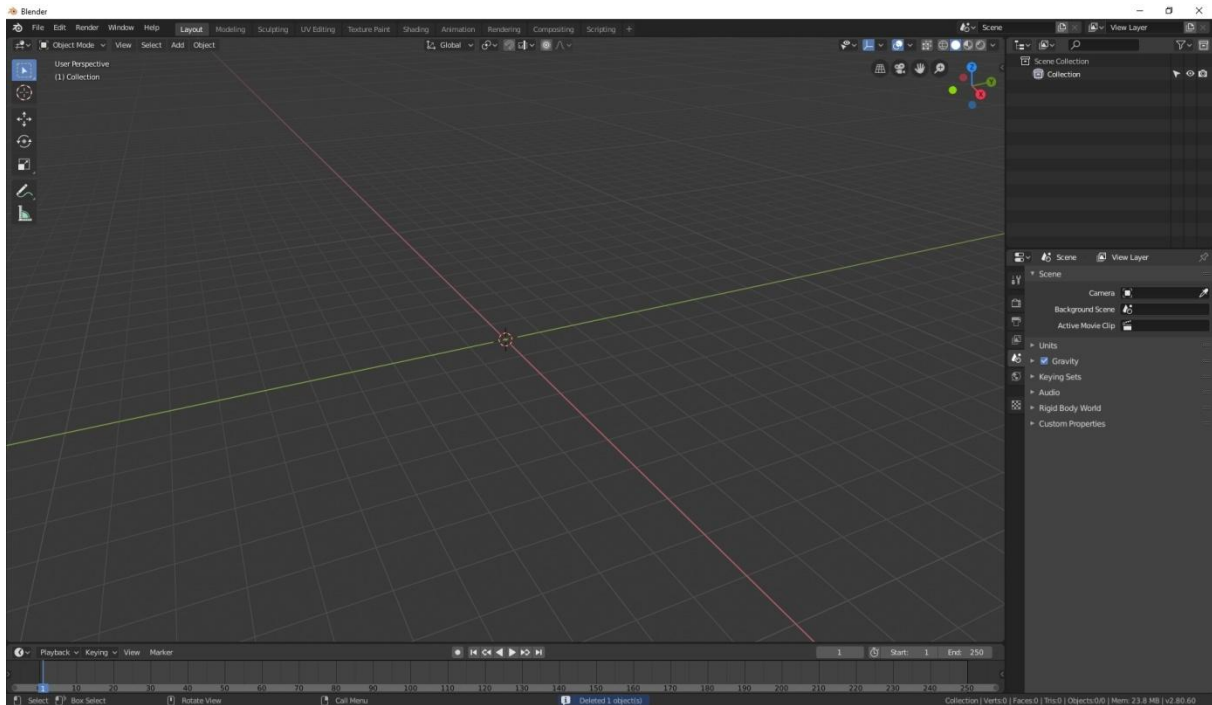


Ilustración 5: Blender.

2.1.5 Visual Studio Code

Visual Studio Code es un editor de código multiplataforma de código abierto desarrollado por Microsoft que aparece en 2015 bajo una licencia MIT. Está basado en Electron, un framework para la creación de aplicaciones y que utiliza Javascript como lenguaje. Es compatible con multitud de lenguaje gracias a una de sus principales características, las extensiones. Estas extensiones son desarrolladas por la comunidad y permiten mejorar el editor para que se adapte a las necesidades de cada momento.

En las últimas versiones de Unity se ha incluido compatibilidad total con Visual Studio Code mediante un par de extensiones, una para la detección de clases de Unity y otra para depurar el código a través de Visual Studio Code.

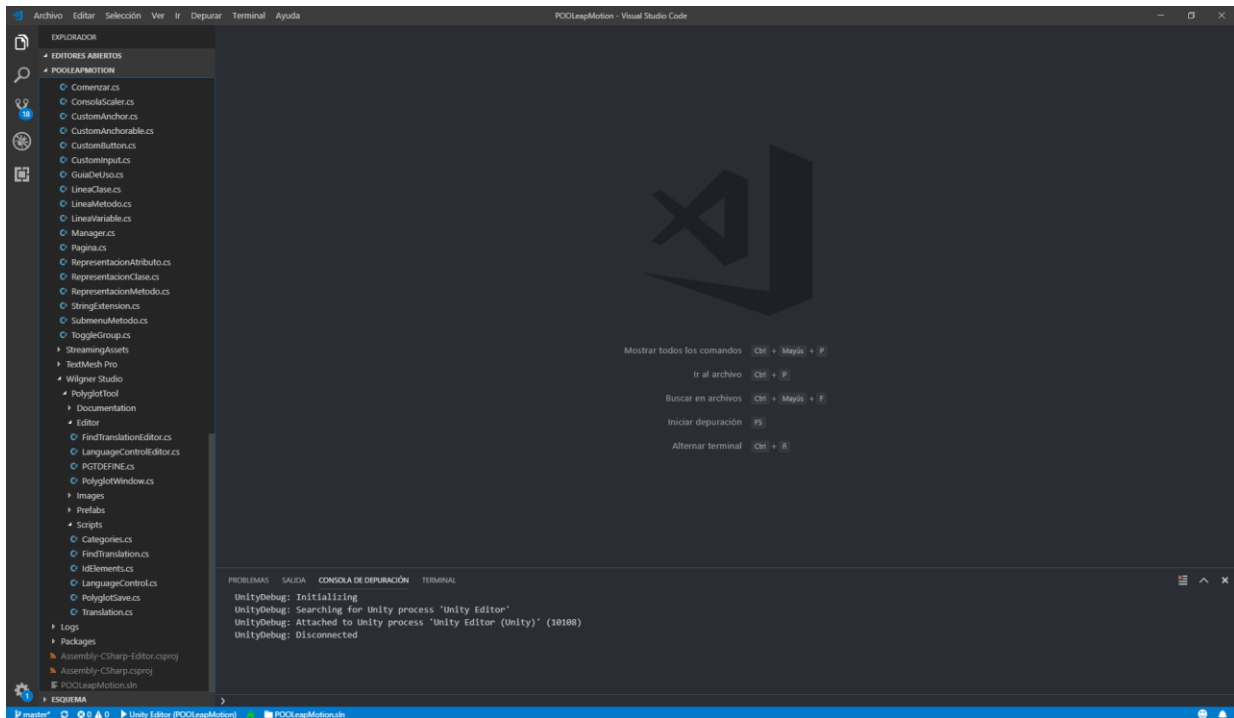


Ilustración 6: Visual Studio Code.

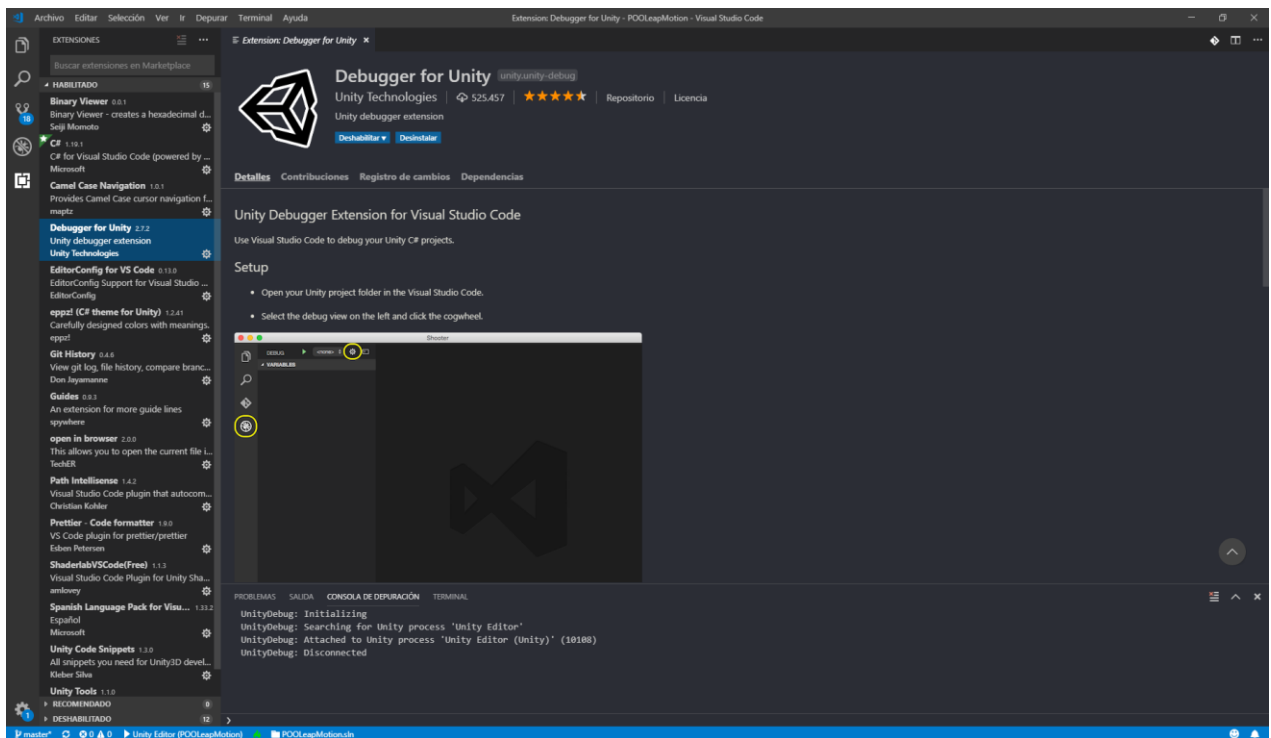


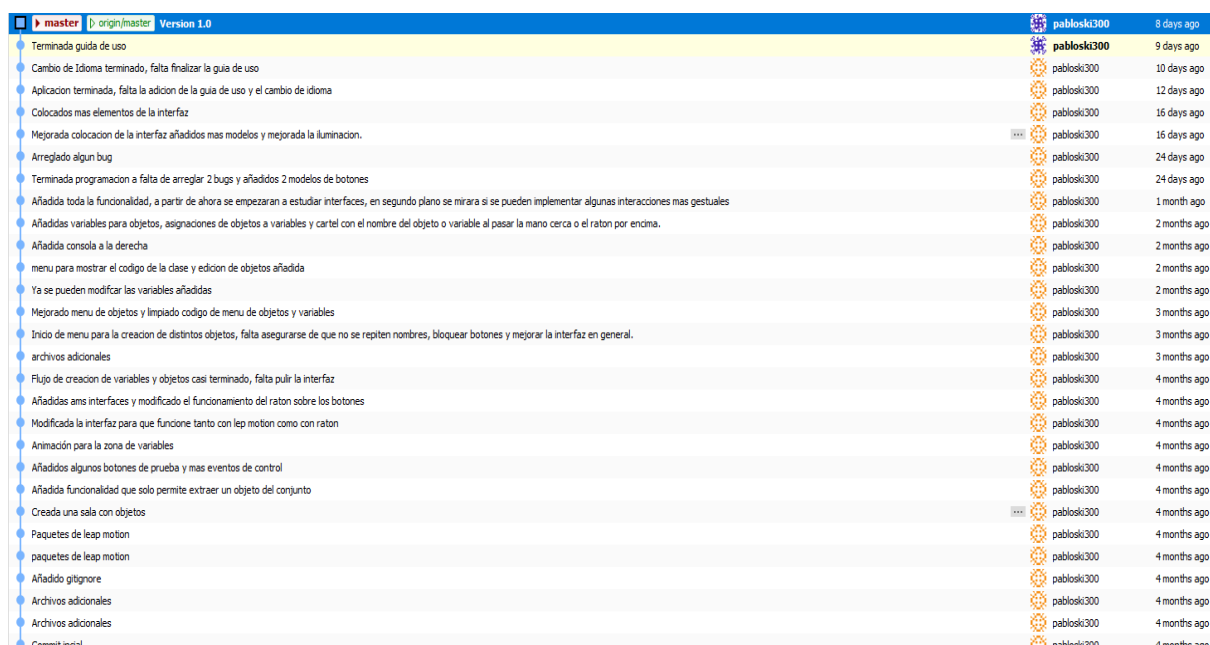
Ilustración 7: Extensiones en Visual Studio Code.

2.1.6 GitHub

GitHub es una plataforma para alojar proyectos software que hace uso del sistema de control de versiones Git. GitHub aparece el 8 de Febrero de 2008 por parte de la compañía GitHub Inc y el 4 de junio de 2018 es comprado por Microsoft.

Las herramientas de control de versiones son muy importantes en proyectos software, para tener siempre una copia del código en distintas versiones del proyecto para poder enmendar errores que aparezcan durante el desarrollo al añadir nuevas funcionalidades. En este caso se ha elegido GitHub por la sencillez para crear un repositorio de manera gratuita.

Para el uso del repositorio se ha descargado la herramienta GitExtension, esta herramienta ofrece una interfaz intuitiva para trabajar con el repositorio de todas las maneras necesarias.



master origin/master Version 1.0		pabloski300	8 days ago
Terminada guía de uso		pabloski300	9 days ago
Cambio de Idioma terminado, falta finalizar la guía de uso		pabloski300	10 days ago
Aplicacion terminada, falta la adición de la guía de uso y el cambio de idioma		pabloski300	12 days ago
Colocados mas elementos de la interfaz		pabloski300	16 days ago
Mejorada colocacion de la interfaz añadidos mas modelos y mejorada la iluminación.		pabloski300	16 days ago
Arreglado algun bug		pabloski300	24 days ago
Terminada programacion a falta de arreglar 2 bugs y añadidos 2 modelos de botones		pabloski300	24 days ago
Añadida toda la funcionalidad, a partir de ahora se empezaran a estudiar interfaces, en segundo plano se mirara si se pueden implementar algunas interacciones mas gestuales		pabloski300	1 month ago
Añadidas variables para objetos, asignaciones de objetos a variables y cartel con el nombre del objeto o variable al pasar la mano cerca o el raton por encima.		pabloski300	2 months ago
Añadida consola a la derecha		pabloski300	2 months ago
menu para mostrar el código de la clase y edicion de objetos añadida		pabloski300	2 months ago
Ya se pueden modificar las variables añadidas		pabloski300	2 months ago
Mejorado menu de objetos y limpiado código de menu de objetos y variables		pabloski300	3 months ago
Inicio de menu para la creacion de distintos objetos, falta asegurarse de que no se repiten nombres, bloquear botones y mejorar la interfaz en general.		pabloski300	3 months ago
archivos adicionales		pabloski300	3 months ago
Flujo de creacion de variables y objetos casi terminado, falta pulir la interfaz		pabloski300	4 months ago
Añadidas mas interfaces y modificado el funcionamiento del raton sobre los botones		pabloski300	4 months ago
Modificada la interfaz para que funcione tanto con lep motion como con raton		pabloski300	4 months ago
Animación para la zona de variables		pabloski300	4 months ago
Añadidos algunos botones de prueba y mas eventos de control		pabloski300	4 months ago
Añadida funcionalidad que solo permite extraer un objeto del conjunto		pabloski300	4 months ago
Creada una sala con objetos		pabloski300	4 months ago
Paquetes de leap motion		pabloski300	4 months ago
paquetes de leap motion		pabloski300	4 months ago
Añadido gitignore		pabloski300	4 months ago
Archivos adicionales		pabloski300	4 months ago
Archivos adicionales		pabloski300	4 months ago
Commit inicial		pabloski300	4 months ago

Ilustración 8: Git Extension.

Capítulo 3: Descripción de la Aplicación

En este apartado se discutirá como se ha desarrollado el proyecto. Se hablara de la metodología seguida y del trabajo realizado para llegar a la aplicación final.

3.1 Metodología de trabajo

El desarrollo de la aplicación ha sido dividido en 2 grandes partes, por un lado el desarrollo de la funcionalidad, es decir, los scripts necesarios para que todo el sistema de botones, menús e interacciones funcionara. Por el otro lado está el diseño de la interfaz que se ha centrado principalmente en buscar accesibilidad para daltónicos, la reduciendo el uso de textos en botones al mínimo y organizarse de manera que sea fácil de usar con la interacción gestual.

Para este proyecto se ha decidido implementar una metodología iterativa e incremental, en este caso se ha elegido el modelo de espiral. Esta metodología consiste en un desarrollo cíclico en el que en cada ciclo se añade un poco de funcionalidad a la aplicación. Esta se prueba y se mejora con el feedback recibido. Acto seguido se vuelve a repetir el ciclo añadiendo nueva funcionalidad.

Los ciclos han consistido en la repetición de 3 fases. La primera es la prueba de la funcionalidad que se desea implementar en una escena aparte. Se explora cual es la mejor manera para implementarla con todas las posibilidades que nos ofrecen tanto Unity como LeapMotion.

La segunda fase consiste en la implementación de la funcionalidad junto con todo el resto de funcionalidades corrigiendo los posibles errores que surjan.

La última fase consiste en la validación de las funcionalidades añadidas. Para esta fase se han realizado reuniones con el tutor para que diera el visto bueno o indicase que cambios habría que realizar.

Este proceso se ha seguido de manera iterativa, aumentando en cada iteración las funcionalidades de la aplicación y mejorando las funcionalidades ya existentes.

Esta metodología se ha apoyado sobre un repositorio, herramienta muy útil para este tipo de desarrollo, permitiéndonos llevar un control sobre los cambios de cada iteración.

3.2 Inicios del Proyecto

En los inicios del desarrollo se valoró la posibilidad de permitir al usuario introducir sus propias clases Java externas a la aplicación, esta característica se abandono tras un tiempo de investigación en el que se encontraron excesivas dificultades a la hora de juntar esta funcionalidad con la funcionalidad de ejecutar métodos. La principal dificultad pasaba por la imposibilidad de ejecutar el código java desde Unity.

Otra dificultad que se encontró con este planteamiento fue las dificultades por las que pasaría el usuario a la hora de añadir sus clases a la aplicación para que esta las leyera. Finalmente se decidió optar por dotar a la aplicación con su propio creador de clases que aporta las opciones necesarias para las explicaciones de los conceptos que se tratan.

El resto de objetivos no cambiaron a lo largo del desarrollo y fueron desarrollados sin problemas mayores.

3.3 Desarrollo y construcción.

En este apartado se discutirán tanto la creación de las funcionalidades por script como la creación de la interfaz 3D. Se comenzará hablando del desarrollo de los scripts para la funcionalidad.

El desarrollo se divide en varios Scripts. Algunos de ellos creados desde cero usando la clase base de los GameObjects, llamada MonoBehaviour, y otros heredando de los Scripts de LeapMotion. También se han creado clases auxiliares para la persistencia que se ha desarrollado mediante un archivo binario que guarda la información.

El desarrollo se ha dividido a su vez en tres partes. Una parte ha consistido en la creación de los menús de creación de clases. Estos menús eran necesarios ya que el usuario no puede escribir código como haría en una aplicación de aprendizaje al uso. Por lo tanto se ha decidido dotar a la aplicación con un creador de clases que da al usuario las opciones necesarias para la creación de clases que cumplan con las explicaciones de los conceptos de Programación Orientada a Objetos. El usuario puede crear atributos de 3 tipos (int, boolean, float) y 6 métodos que comprenden las operaciones básicas y operaciones de entrada y salida.

La segunda parte consiste en todo el sistema de creación y representación de objetos, variables, atributos y métodos en la escena. Esta parte se ha solventado usando mallas simples con distintos colores que las identifiquen, permitiendo interactuar con todos ellos haciendo

uso de los scripts de interacción de LeapMotion o creando clases que hereden de ellos. También se ha creado un menú que centraliza las clases. En este menú se dan varias opciones sobre las clases que van desde inspeccionar el código java que se corresponde a la clase hasta modificar clases ya creadas.

La tercera y última parte consiste en la inspección de objetos y variables para ver su contenido y la posibilidad de poder ejecutar los métodos que contengan los objetos. Esta parte guarda relación con la segunda pero tiene funcionalidades totalmente distintas que requerían la creación de scripts nuevos.

Por último ha habido una parte que ha afectado a todas las partes del desarrollo y del diseño de la interfaz y es la interacción gestual basada en LeapMotion. Ha afectado a todo el desarrollo pues el enfoque es muy distinto de una aplicación tradicional como puede ser BlueJ. Se ha enfocado la aplicación a una pseudo realidad virtual en la que toda la interfaz funciona por físicas y es tridimensional pero la cámara esta fija en una zona como una aplicación tradicional. Esto ha generado una cantidad muy grande comportamientos no deseados en los inicios del desarrollo que han sido solventados a medida que progresaba el proyecto.

Cabe destacar que la aplicación se ha basado en su mayoría en eventos producidos por interacciones, intentando así reducir el consumo por la ejecución de scripts. El único uso que se le ha dado a una función ejecutada periódicamente, en este caso el Update de Unity, ha sido para el movimiento de los objetos con el ratón.

A continuación comentare los scripts que se han desarrollado y las fases por las que han pasado hasta conseguir el resultado que se buscaba.

3.3.1 Manager

Este Script merece un punto aparte pues no se parece a ningún otro de la aplicación. Es el encargado de contener todos los datos de interés para la aplicación. Se ha implementado usando un patrón singleton para mantener una única instancia en la escena.

El Manager almacena distintos tipos de datos:

- Instancias de las representaciones de objetos, variables, atributos y métodos que luego aparecerán en la escena. De esta manera se reduce el número de instanciaciones de objetos desde memoria que se hace.

- Un diccionario con todos los menús de la aplicación, de tal manera que el acceso y búsqueda de estos se puede realizar con una clave que será el nombre del menú. De esta manera los accesos a estos menús serán muy rápidos.
- Un diccionario con los colores generados aleatoriamente, la razón de esto es tener una comprobación rápida a la hora de comprobar si un color ya ha sido creado con anterioridad
- Una lista con todos los objetos creados, otra con todas las variables creadas y una lista con todas las clases creadas, estas listas se han colocado en este scripts pues es el que se encarga de guardarlas para la persistencia.

Además de los datos el Manager cuenta con unas funciones auxiliares para generar colores y guardar y cargar partida. El guardado se ha realizado usando el sistema de archivos binarios de Unity que permite crear archivos binarios serializando clases con tipos primitivos.

La función de guardado es llamada cada vez que se vuelve al menú de inicio y la de carga cuando se pulsa el botón de carga. Como se ha mencionado antes se han creado 4 clases para realizar este guardado y carga, las clases son serializadas por un BinaryFormatter que viene incluido en el lenguaje C#.

Para el guardado se extraen los datos necesarios de la aplicación que se guardan en un objeto de la clase principal de guardado y esta se serializa en un archivo binario. Para la carga se realiza el proceso inverso, creando un objeto de la clase principal y creando de nuevo todos los objetos, variables, atributos y métodos a partir de los datos guardados.

Las clases creadas para el sistema de persistencia son las siguientes:

- Atributo: esta clase contiene los datos más relevantes de un atributo para poder ser creado de nuevo al cargar. Esos datos son el nombre y el nivel de acceso, ambos strings.
- Método: al igual que el atributo contiene los datos que identifican el método. En este caso es simplemente el nombre del método que es un string.
- Clase: para identificar una clase es necesario guardar dos listas una para métodos y otra para atributos y además el nombre de la clase que es un string.
- Variable: para identificar la variable se guardan el nombre de esta como string y el nombre de la clase de la variable.

- **Save:** Este es el archivo principal. Guarda una lista de variables y de clases además de una lista de enteros indicando que variables estaban referenciando que objetos. Tambienn guarda una lista indicando cuantos objetos hay y de que clase son usando un entero y una lista de strings que guarda la información de la consola. Esta es la clase que se serializa en un archivo binario.

3.3.2 Interacción

Este es el apartado con más importancia en el desarrollo de la aplicación pues es la principal diferencia que ofrece con respecto a otras aplicaciones clásicas de aprendizaje sobre la programación.

Para este apartado se ha hecho uso de los Scripts de Interacción que ofrece LeapMotion para Unity. Se han creando scripts que heredan o hacen uso de la funcionalidad que se brinda al desarrollador. Se puede dividir la interacción en dos partes, la interacción con objetos y la interacción con la interfaz.

La interacción con objetos, representados por cubos en la escena, se ha implementado haciendo uso de 2 scripts de interacción. El primero se llama InteractionBehaviour y permite a un objeto sea tocado y cogido con las manos. Este script nos permite lanzar un evento para cada tipo de interacción además de cambiar diversas configuraciones o bloquear algunas de esas interacciones.

En la siguiente imagen se muestra un objeto interactivo y el script encargado.

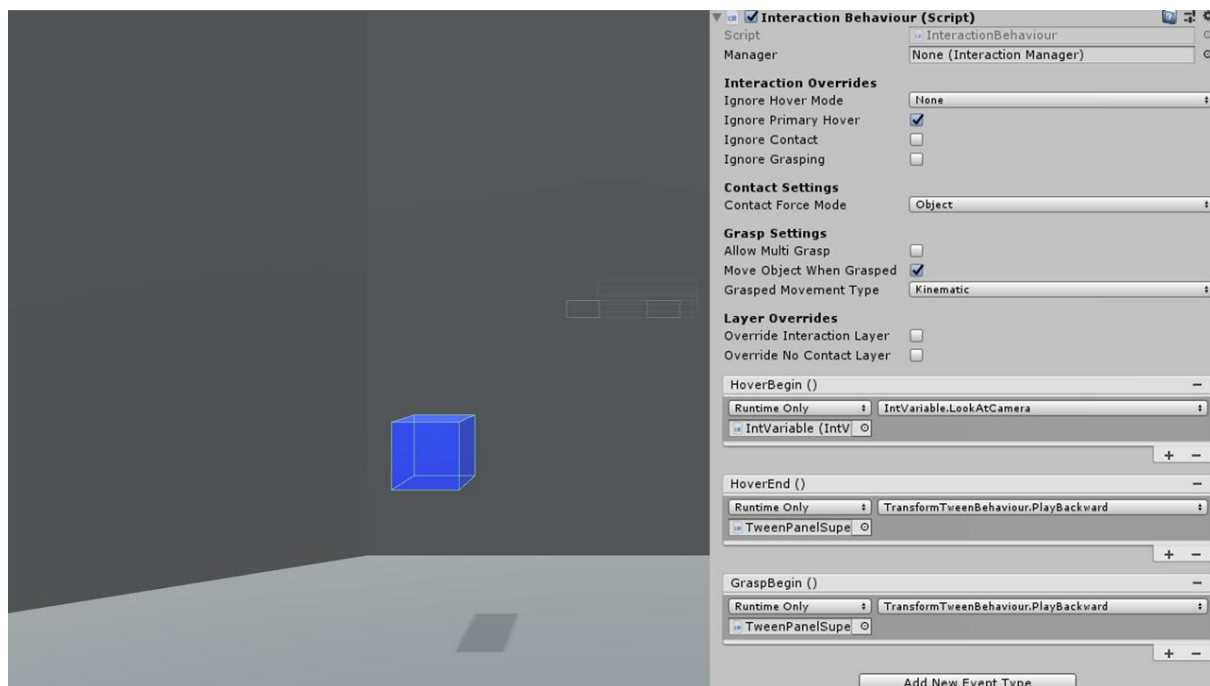


Ilustración 9: Representación de un objeto y Script de interacción.

El segundo script que se ha usado se llama `AnchorableBehaviour` y es el encargado que anclar los objetos a puntos en la escena. De esta manera cuando un objeto no ha sido cogido por el usuario no queda flotando en la escena o perdido en algún punto no accesible. Para hacer uso de este comportamiento se han colocado en los distintos menús objetos con un script de `LeapMotion` que se encarga de convertir el objeto en puntos de anclaje, permitiendo anclar a ellos los objetos. Todos estos anclajes son gestionados por un script que poseen todos los objetos interactivos.

Agrupando los 2 anteriores scripts se ha creado un script propio que es el encargado de gestionar la interactividad y eventos con todos los objetos que no son botones. Este script se llama `CustomAnchorable`. El script se encarga de devolver los objetos a su posición de origen cuando el usuario no los está cogiendo o de colocarlos donde es necesario dependiendo del menú.

Este script se encarga de recoger los anclajes y scripts de interacción del objeto y realizar operaciones sobre ellos para que el objeto este anclado o no cuando le corresponde.

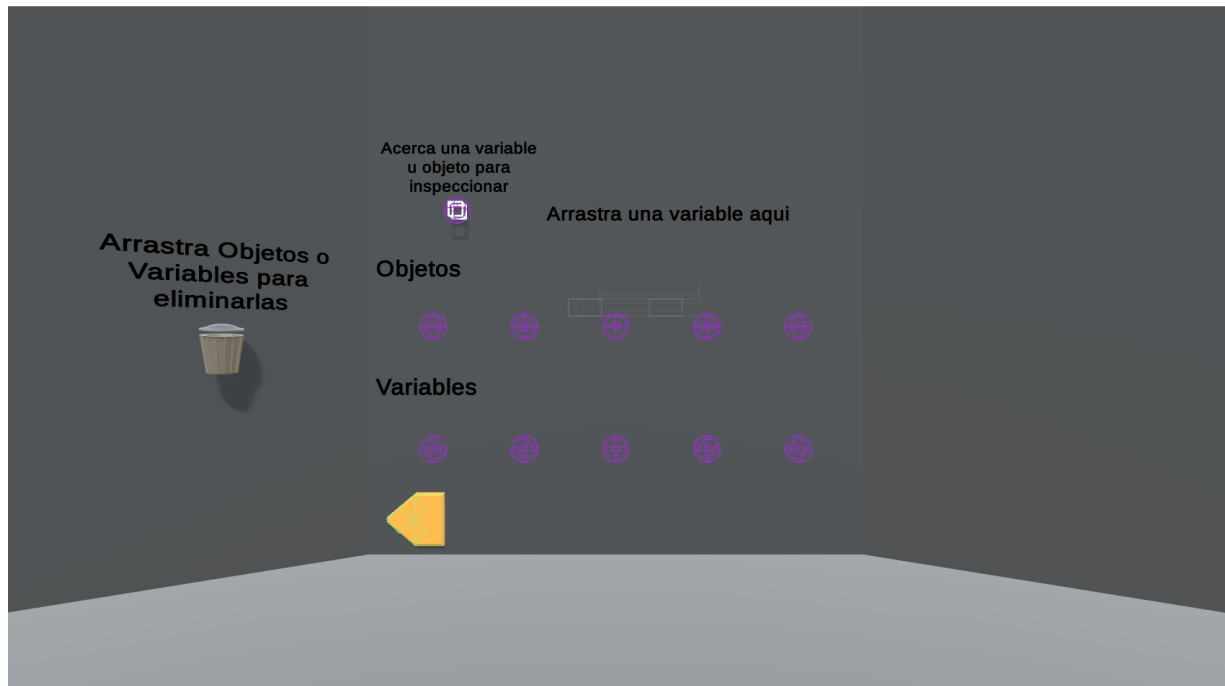


Ilustración 10: Anclajes en el menú de representación (en morado).

La interacción con los botones hace uso de un script propio, llamado CustomButton, que hereda del script de LeapMotion InteractionButton y añade métodos para poder bloquear los botones y evitar así interacciones no deseadas después de una pulsación o durante las transiciones de menús.

Para la interacción con las manos LeapMotion hace uso de las colisiones del motor de físicas de Unity. Esto ha vuelto necesaria la creación de un script propio que heredara y gestionara cuando el botón puede interaccionar y cuando no.

En la siguiente imagen podemos ver un botón en la escena y su script.

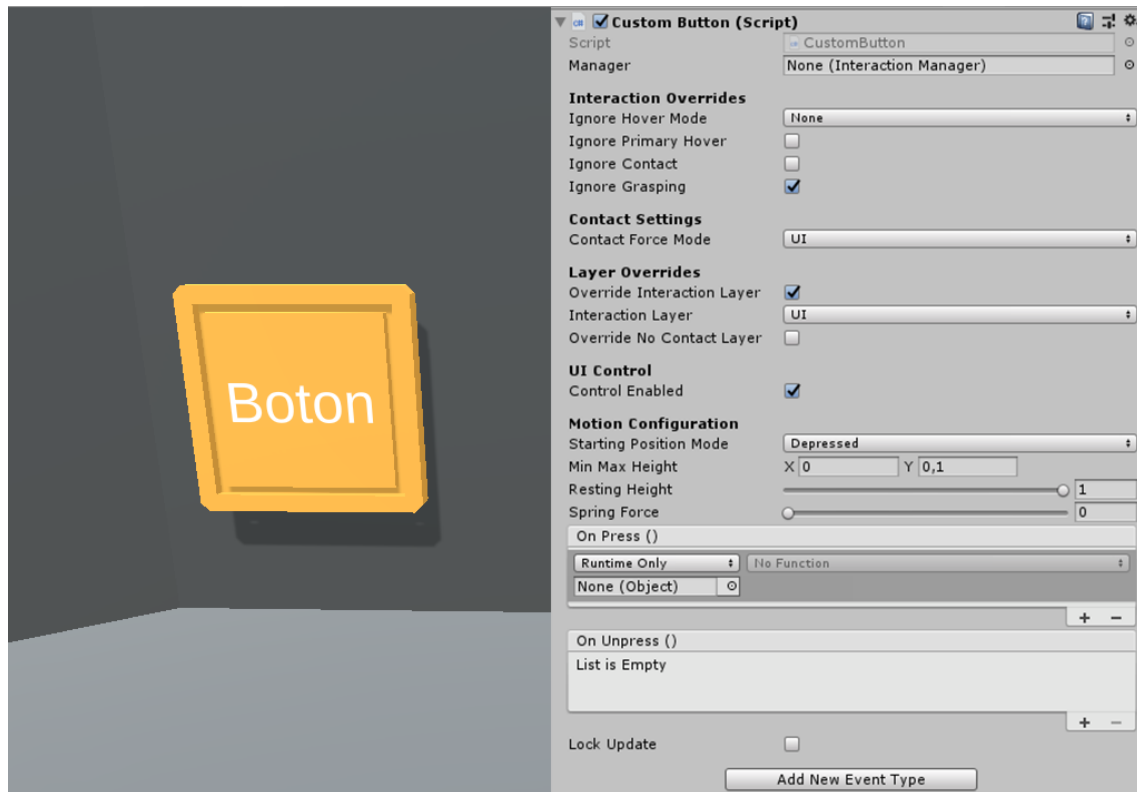


Ilustración 11: Botón y Script de interacción con botones.

Las interacciones se realizan sobre los objetos y botones de manera natural. Los botones pueden ser presionados usando las manos y los objetos cogidos acercando la mano a estos y cerrándola.

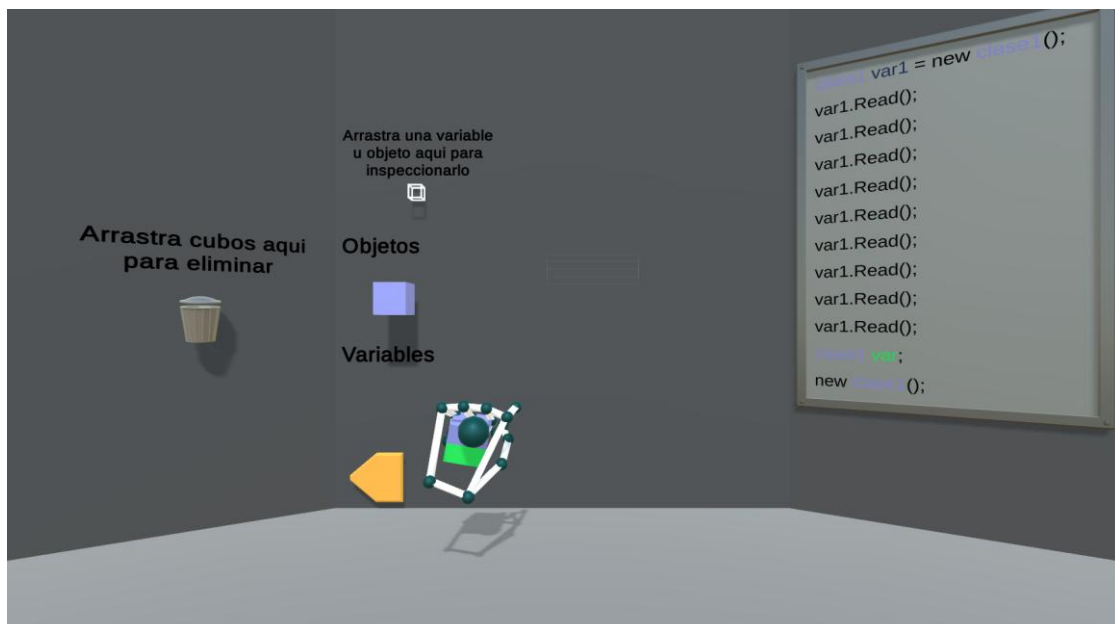


Ilustración 12: Mano cogiendo un objeto.

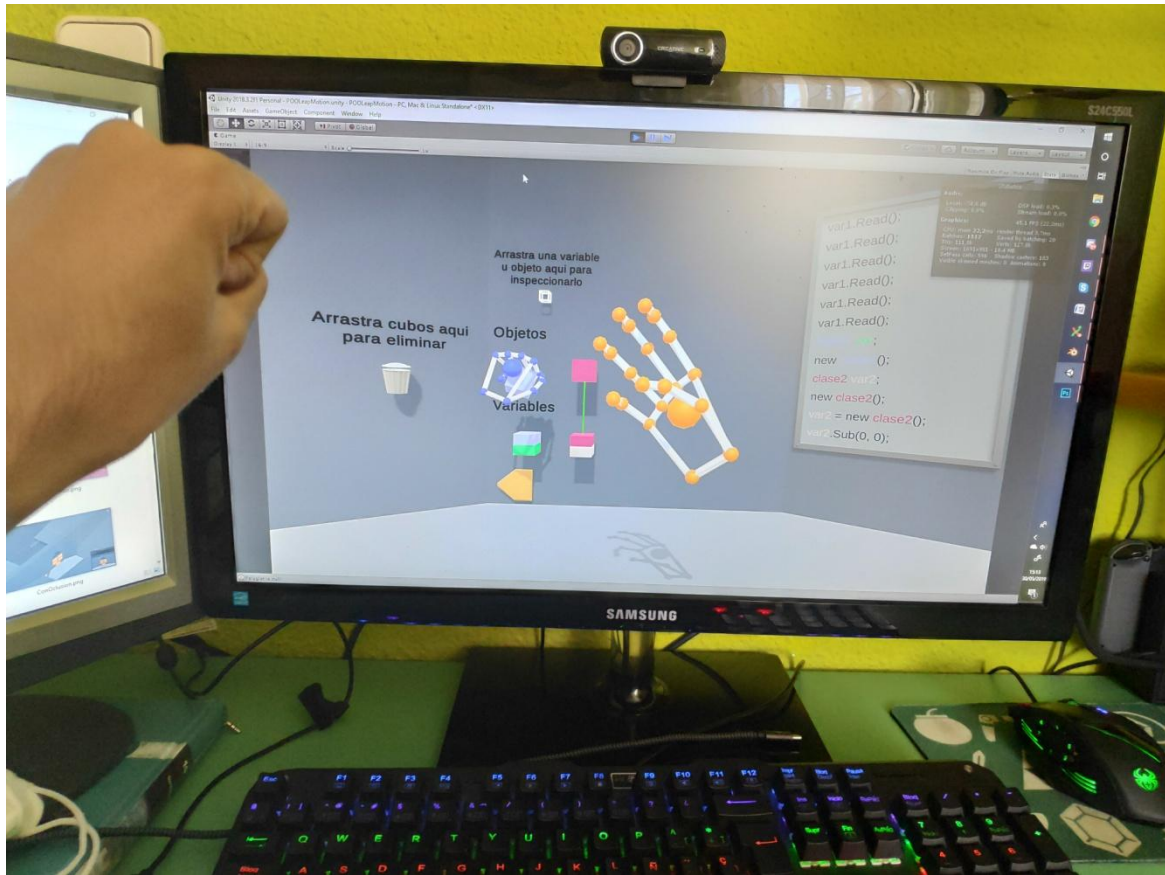


Ilustración 13: Fotografía de la mano detectada por el LeapMotion cogiendo un objeto.

Además, aprovechando los eventos lanzados por las interacciones, se ha añadido un pequeño cartel a todos los objetos interactivos que indica que representa ese objeto. De esta manera se ha podido ahorrar espacio y se aprovecha al máximo la interacción.



Ilustración 14: Cartel informativo.

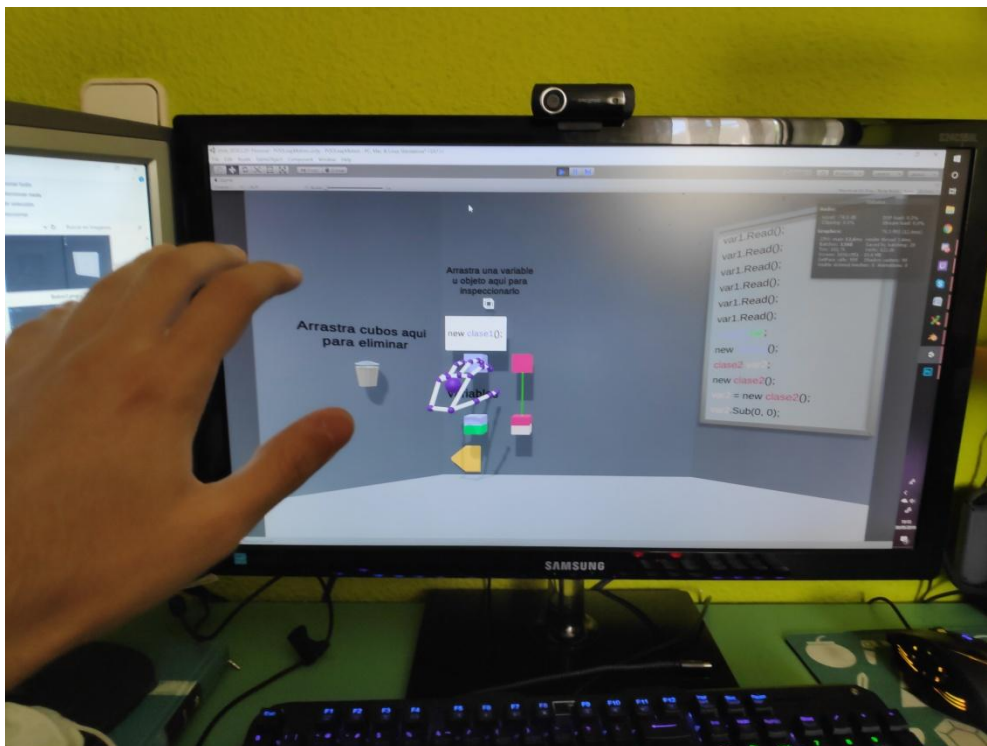


Ilustración 15: Fotografía de la mano detectada por el LeapMotion cerca de un objeto.

Ambos scripts, el que se ha creado para los objetos y el que se ha creado para los botones, añaden funcionalidad para el ratón, de tal manera que el usuario pueda realizar las mismas acciones con un dispositivo LeapMotion que con un ratón.

3.3.3 Objetos, Variable, Métodos y Atributos.

Estos objetos constituyen las unidades básicas con las que interaccionara el usuario. Todos ellos están representados por un cubo en la escena y tienen un script propio que se encarga de la funcionalidad.

Todos los scripts propios heredan del script anteriormente descrito CustomAnchorable que es el que añade la interactividad.

El script del objeto se encarga de almacenar en listas todos los métodos y atributos que contiene además del nombre y el color. Además de estos datos tiene un método que se encarga de detectar cuando el objeto está anclado a un anclaje especial para inspeccionar el objeto.

El script de la variable almacena el nombre y el color de la variable además del nombre y el color de la clase. Al igual que el objeto posee un método para detectar cuando e ha anclado a un anclaje especial. En la siguiente imagen se muestran varias variables y objetos creados.

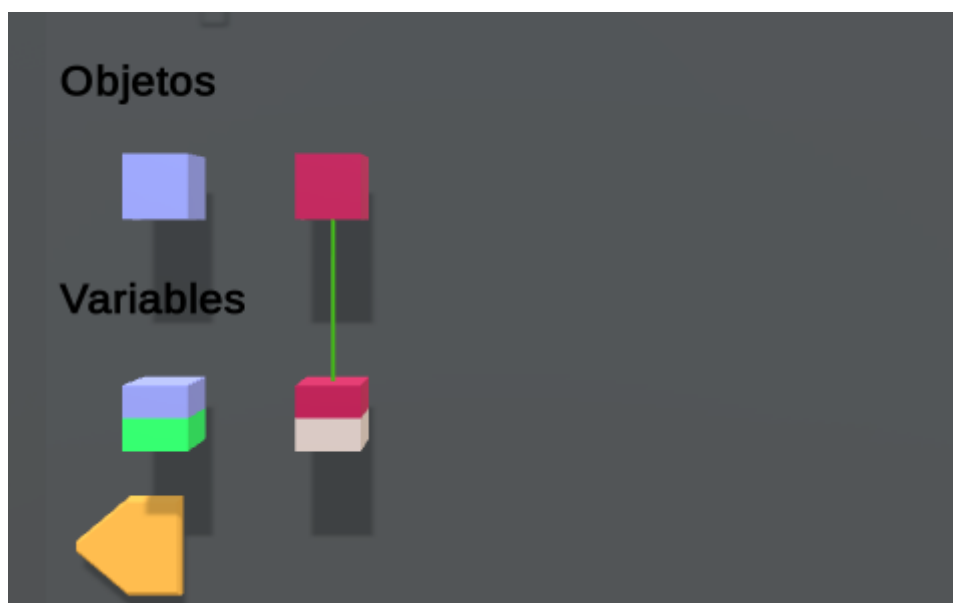


Ilustración 16: Objetos y variables creados en la aplicación.

Para los atributos se han creado un script que contiene el nombre y la protección y 3 scripts que heredan de éste y se encargan de guardar el valor dependiendo de si es un entero un booleano o un float.

Finalmente, para los métodos se ha creado un script con una clase abstracta y seis scripts que heredan de esta clase abstracta. Los scripts individuales se encargan de definir la cabecera del

método y el método que se ejecuta. En la siguiente imagen podemos ver los atributos y métodos de un objeto dentro de la aplicación.



Ilustración 17: Atributos y métodos de un objeto.

Todos los objetos se han creado previamente como prefabs y se cargan en la aplicación al iniciarla. Más adelante se instancian copias de estos prefabs y se modifican los datos de las copias para crear las variables objetos y métodos que el usuario desee.

Para facilitar el acceso a los prefabs en tiempo de ejecución se ha hecho uso de una funcionalidad de Unity llamada Assets Bundles. Esta funcionalidad permite guardar datos de manera comprimida. Estos datos pueden cargados en la aplicación cuando sea necesario para extraerlos usarlos y eliminarlos una vez se han terminado de usar.

Esta característica se ha usado para hacer un Asset Bundle con cada conjunto de objetos. Uno para objetos, otro para variables, otro para atributos y otro para métodos. En la siguiente imagen se muestran los Asset Bundles.

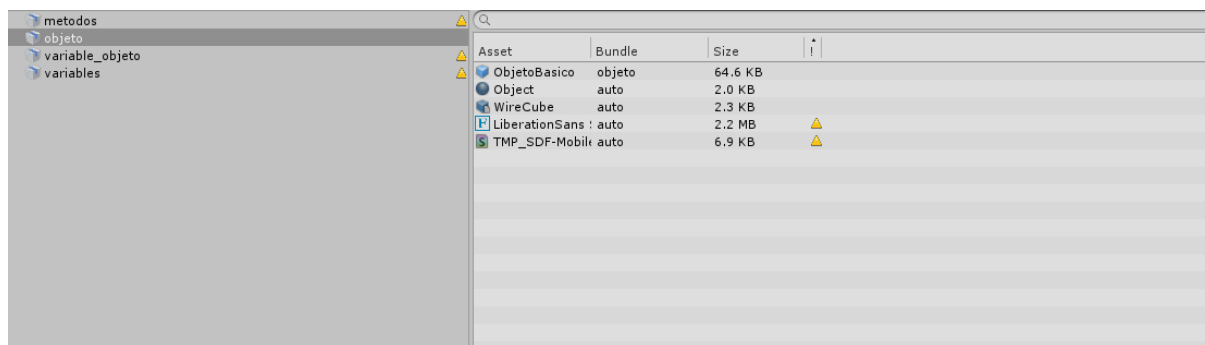


Ilustración 18: Asset Bundles de la aplicación.

3.3.4 Sistema de menús

Para la creación de los distintos menús se ha creado un script padre que se encarga de la funcionalidad compartida por todos los menús, esta funcionalidad consiste en la activación, desactivación y almacenamiento de los botones. Esta funcionalidad es muy necesaria para que no se produzcan pulsaciones no deseadas con los botones al realizar las transiciones entre menús.

Para el almacenamiento se han guardado los botones en un diccionario de tal manera que cualquier menú pueda acceder a sus botones con el nombre de estos de manera muy rápida.

Heredando de este primer script llamado CustomMenu se han creado el resto de scripts que dan la funcionalidad propia a cada menú.

3.3.5 Transiciones

El sistema de menús se apoya en un sistema de transiciones entre estos. Para la creación de este sistema se ha hecho uso de un script de LeapMotion que permite, dándole 3 objetos, hacer la transición de posición, rotación y escala de uno de ellos entre la posición, rotación y escala de los otros 2.

Este Script tiene otra peculiaridad muy importante, el script lanza un evento siempre que se llega o abandona la posición inicial o final de la interpolación. Estos eventos han sido muy importantes a la hora de solucionar problemas con interacciones no deseadas.

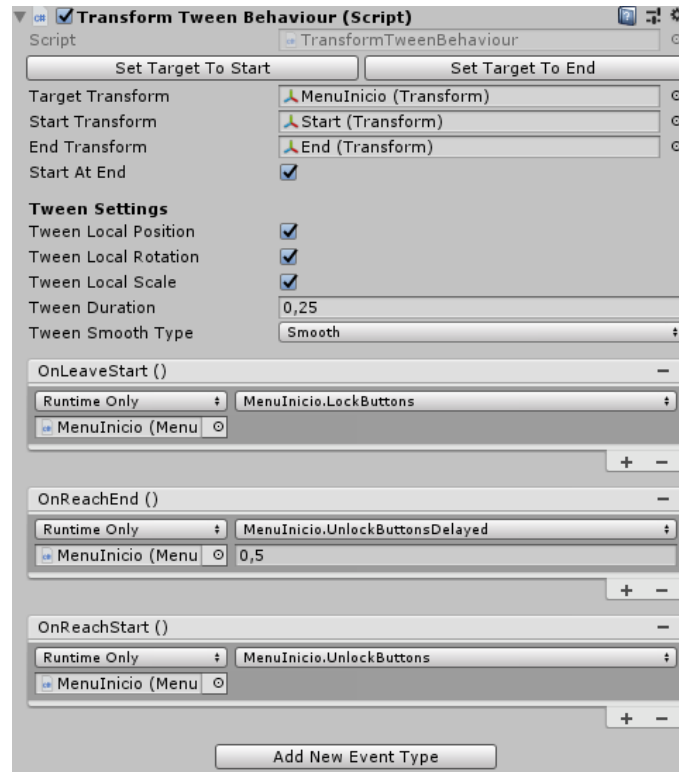


Ilustración 19: Script para las interpolaciones.

3.3.6 Diagrama UML

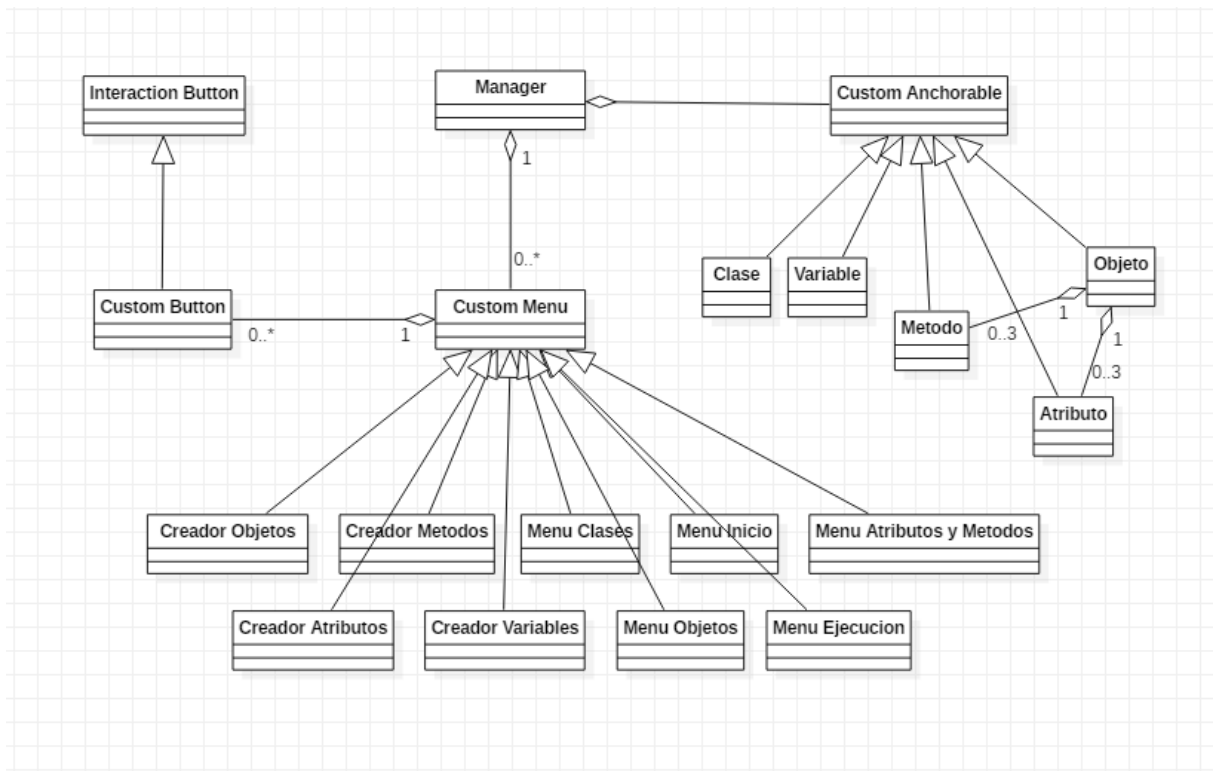


Ilustración 20: Diagrama UML.

3.3.7 Problemas en el Desarrollo de los Scripts

En el proceso para desarrollar los scripts han surgido diversidad de problemas sobre todo al añadir nuevas funcionalidades. Estos problemas no estaban fuera de los cálculos del proyecto y no han supuesto un retraso considerable. Sin embargo, hay un problema que se ha llevado a lo largo de todo el proyecto hasta su completa solución. Este problema tiene que ver con la interacción con los botones.

Si bien esta interacción funcionaba de forma correcta, es decir, los botones respondían a todo contacto, esta no era la interacción que se buscaba siempre. En ocasiones al cambiar de menú o pulsar sobre un botón se producían dobles pulsaciones o se pulsaba un botón de otro menú nada más aparecer en pantalla. Esto era poco intuitivo para el usuario y producía problemas.

La solución a la que se ha llegado consiste en la creación de varios métodos en el script de todos los botones. Al funcionar estos con físicas en primera instancia se pensó en bloquear la función que se encargaba de actualizar todo el sistema de detección y movimiento del botón, esta aproximación solucionaba algunos problemas pero se generaban otros.

Otra aproximación que se valoró fue desactivar las físicas del botón unos segundos tras cambiar de menú o pulsar un botón, esta solución funcionaba muy bien al pulsar botones pero no tenía ningún efecto al cambiar de menú. Finalmente se llegó a una solución híbrida que bloquea la función de actualización en los cambios de menú, cuando no da problemas con las interacciones, y desactiva las físicas cuando se pulsa un botón. De esta forma se ha conseguido que la interfaz responda de manera correcta e intuitiva.

3.3.8 Diseño de Interfaz

El diseño de la interfaz ha ocupado una gran parte del desarrollo y ha seguido el mismo proceso iterativo que el resto de elementos de la aplicación. El principal elemento a tener en cuenta para la creación de la interfaz ha sido la limitación de espacio. La aplicación se localiza en un espacio limitado al no poder moverse la cámara pero tiene que seguir un paradigma de RV donde todos los objetos deben tener cierto tamaño para ser interactivos.

Se comenzó prototipando una primera interfaz con cubos y adaptándola. Al final se tomó la decisión de organizar la interfaz sobre tres paredes en frente del usuario una central totalmente paralela a la cámara y otras 2 a los lados giradas 30° cada una. Esta decisión se ha tomado por la forma en que el ser humano mueve sus brazos. Éstos forman un arco alrededor

del usuario por lo tanto la interfaz no puede ser totalmente plana y debe estar colocada en un arco.

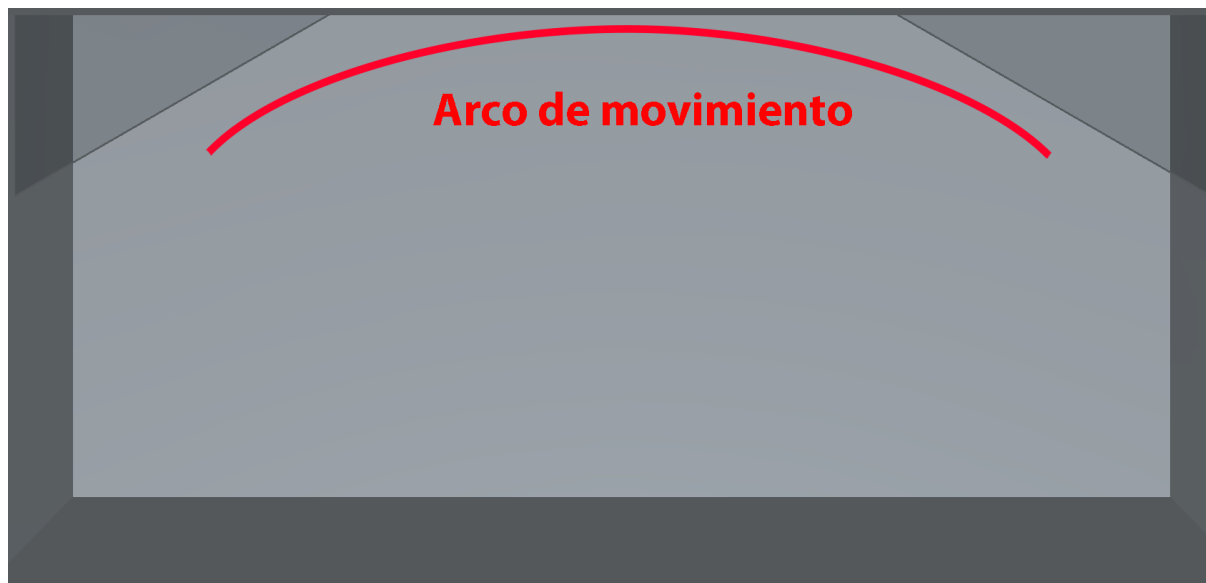


Ilustración 21: Movimiento de las manos.

Otra característica de la interfaz es que la mayoría de elementos se centran en el centro y a una altura intermedia. Esta decisión se ha tomado por las limitaciones del dispositivo LeapMotion. Este dispositivo aunque detecta las manos en un área cercana al de una semiesfera de un radio del alrededor de 1m, pierde mucha calidad en la detección cuando las manos están muy abajo o muy cerca de los extremos. Este problema aparece por el posicionamiento del dispositivo sobre una mesa mirando hacia arriba. Al ser un dispositivo para realidad virtual su mejor posicionamiento es delante de los ojos. Esto se debe a que el dispositivo está pensado para su uso junto con un casco de realidad virtual.

En la siguiente ilustración podemos ver cómo va empeorando la detección según nos alejamos del centro o acercamos mucho las manos al dispositivo.

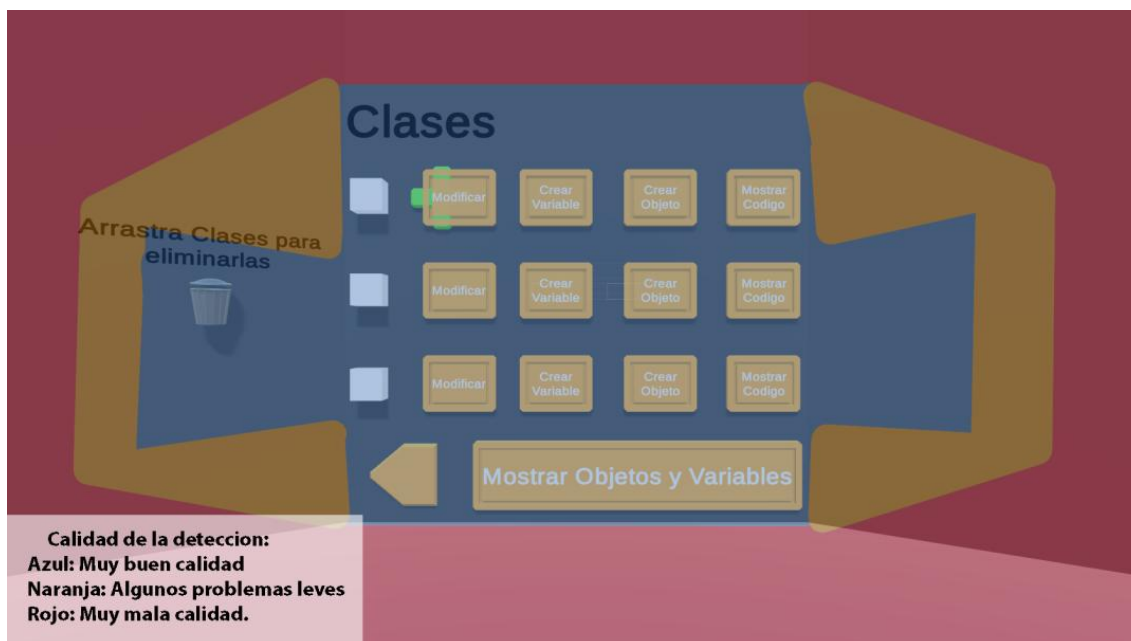


Ilustración 22: Calidad de la detección.

3.3.9 Optimización de Renderizado

En cuanto a las transiciones de los menús, antes de llegar a la solución arriba descrita se realizaron pruebas desactivando los menús para reducir el consumo de la tarjeta grafica al pintar todos los botones. Esta solución no funcionó como se esperaba pues daba grandes problemas con el comportamiento de los botones al reactivar los menús. Finalmente se llegó a la solución de las transiciones por movimiento arriba descrita y para reducir el consumo del pintado de las mallas se ha hecho uso de una herramienta de Unity que se llama Occlusion Culling.

El Occlusion Culling consiste en la división de la escena en cuadrantes de tamaño definido por el desarrollador. Usando estos cuadrantes el motor se encarga de renderizar todas aquellas mallas que estén en los cuadrantes que ve la cámara y en los cuadrantes adyacentes al cuadrante en el que se encuentra la cámara. De esta manera se ahorra mucho cómputo en renderizar mallas que no son visibles.



Ilustración 23: Occlusion Culling desactivado.

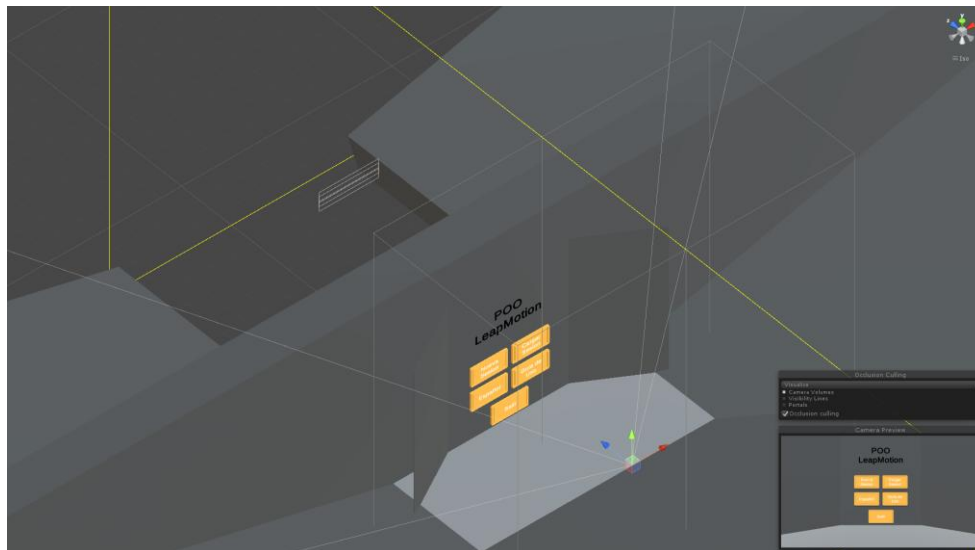


Ilustración 24: Occlusion Culling activado.

3.3.10 Accesibilidad

La última consideración que se ha tenido con la interfaz es la accesibilidad. Se ha elaborado la interfaz de tal manera que sea accesible para personas con deficiencia visual de colores. Para esta tarea se ha dado otro canal para transmitir la información a través de formas o textos en los botones y carteles informativos sobre todos los objetos interactivos.

3.3.11 Multilenguaje

También se ha dotado a la aplicación de dos idiomas. Para ello se ha hecho uso de un asset gratuito descargado de la Asset Store de Unity, el asset se llama Polyglot Tool. Este asset nos permite definir una lista de idiomas y para todos estos idiomas una lista de claves y su correspondiente valor. Este valor será distinto para cada idioma y será la traducción de un elemento para cada idioma.

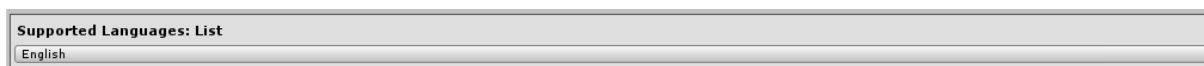


Ilustración 25: Polyglot Tool lista de idiomas.



Ilustración 26: Polyglot Tool lista de traducciones.

Para poder cambiar el idioma en ejecución tiene que existir un script con la clave en cada texto que se desee cambiar.

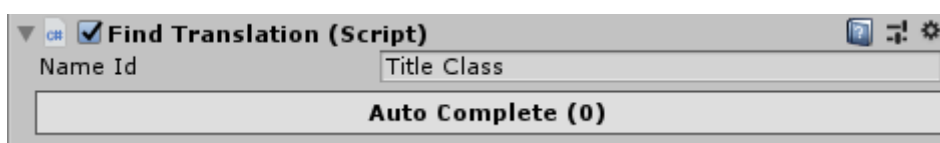


Ilustración 27: Polyglot Tool script de traducción.

También tiene que existir en la escena un objeto con un script que será el encargado de cambiar el idioma de todos los textos cuando se le ordene. Se guarda una referencia a este script en el Manager.

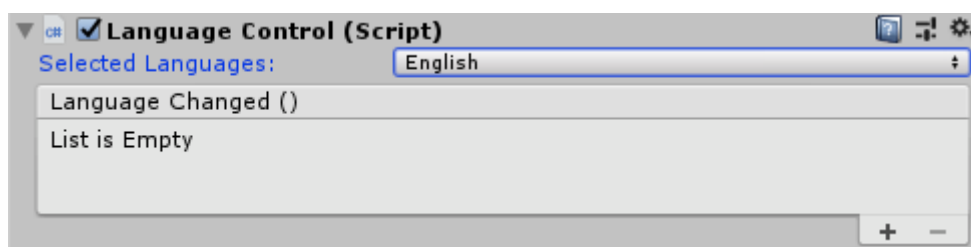


Ilustración 28: Polyglot Tool script de cambio de idioma.

Esta solución no se ha podido aplicar a los textos que se escriben en ejecución dependiendo de las acciones del usuario, para estos textos se ha hecho uso de un booleano guardado en el Manager que nos indica en qué idioma estamos.

3.3.12 Sonido

Además de los estímulos visuales la aplicación también hace uso de estímulos sonoros para transmitir la información de que se ha pulsado un botón. Los sonidos que se han usado en el proyecto han sido descargados de una página web llamada Kenney. Esta página ofrece una variedad de assets gratuitos y de uso libre que van desde modelos 3D hasta iconos 2D.

Para este proyecto se ha hecho uso de 2 sonidos que simulan el clic. Estos sonidos se reproducen usando el componente AudioSource de Unity. Para optimizar el consumo por parte del sonido solo existe un AudioSource en la escena que pertenece al Manager y este se encarga de reproducir el sonido cuando los botones lo indiquen.

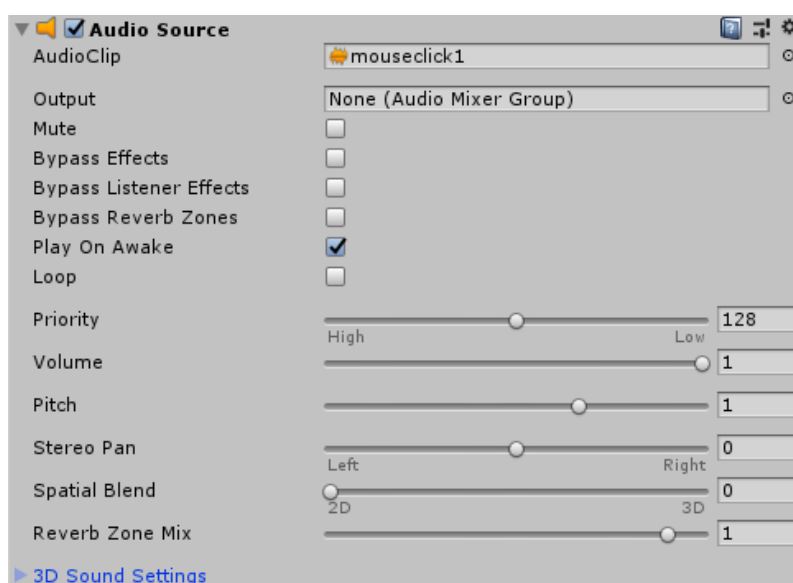


Ilustración 29: Componente AudioSource de Unity.

3.3.13 Modelado

Al ser una interfaz para realidad virtual ha hecho falta crear los elementos de 0 en un programa de modelado. En este caso se ha elegido Blender. Para este proyecto se han modelado varios elementos que se pueden dividir en botones y otros elementos.

En el modelado se ha buscado reducir al mínimo el número de polígonos pues van a existir varios elementos por menú y esto dispararía el consumo al pintar las mallas. Esta reducción a su vez se apoya en las herramientas del motor para reducir el número de llamadas de pintado al compartir los botones de un mismo tipo malla y material.

Botones

El objetivo de los botones es producir una reacción a las acciones del usuario, se ha intentado usar formas que se diferencien y cuando no era posible se han usado textos, de esta manera los colores no son el único estímulo que recibe el usuario consiguiendo así accesibilidad para usuarios con deficiencia visual de colores.

En las siguientes imágenes se muestran los modelos de los botones en Blender.

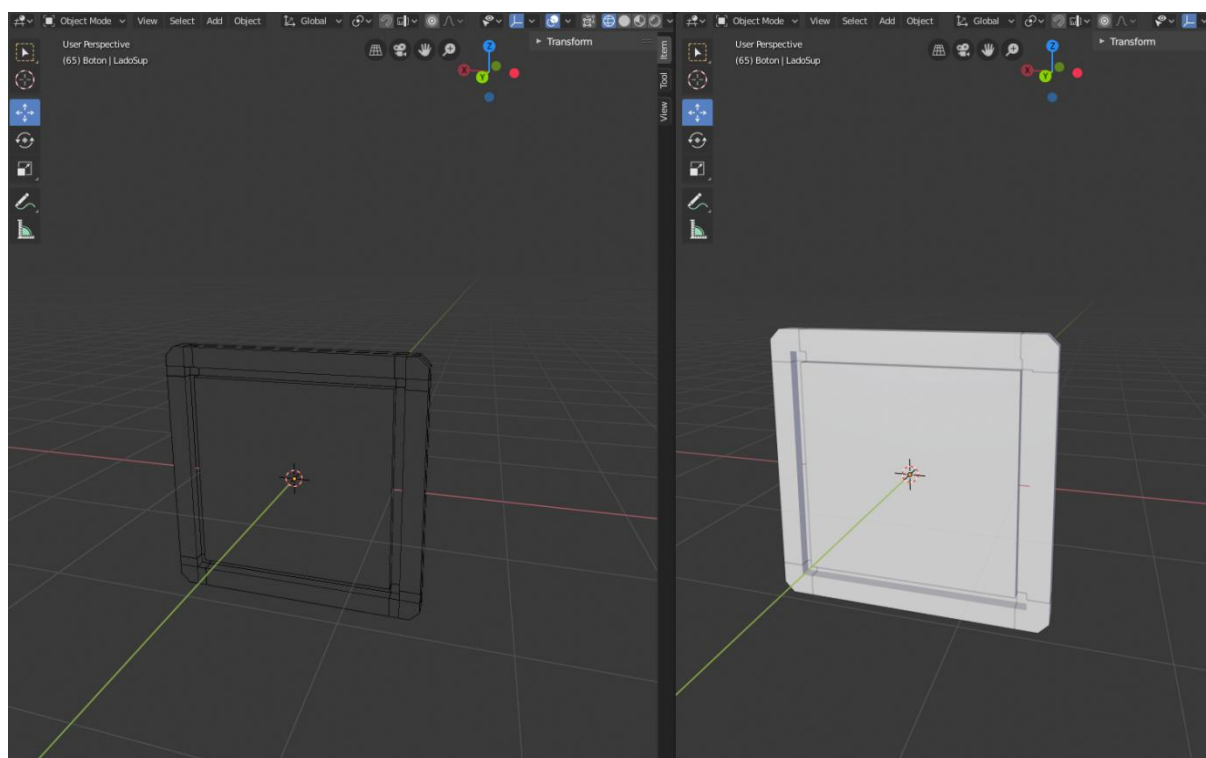


Ilustración 30: Botón Cuadrado.

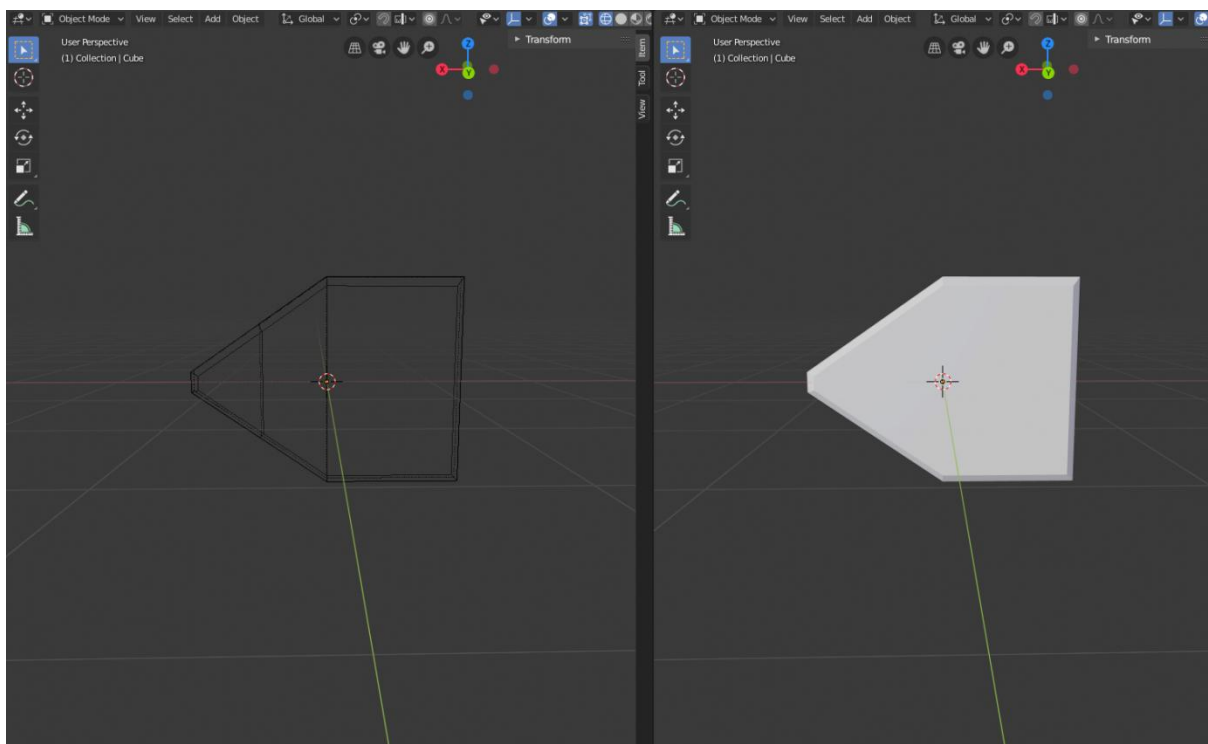


Ilustración 31: Botón Flecha.

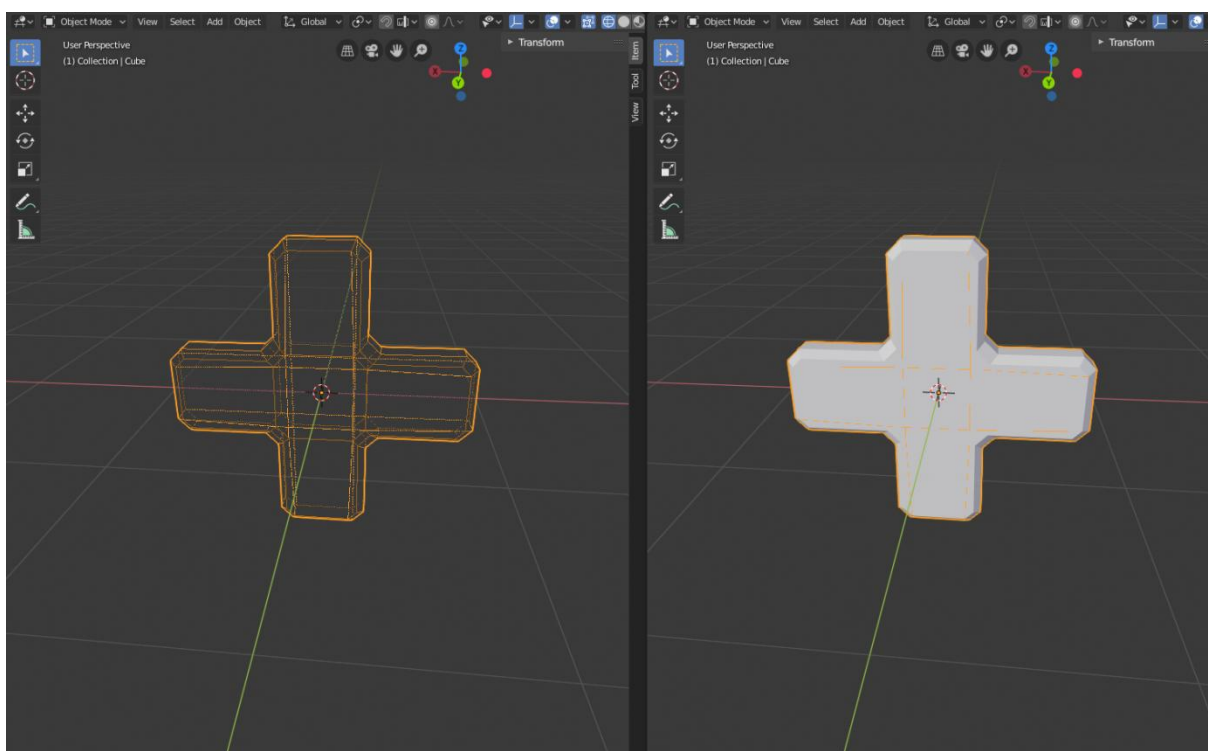


Ilustración 32: Botón Cruz.

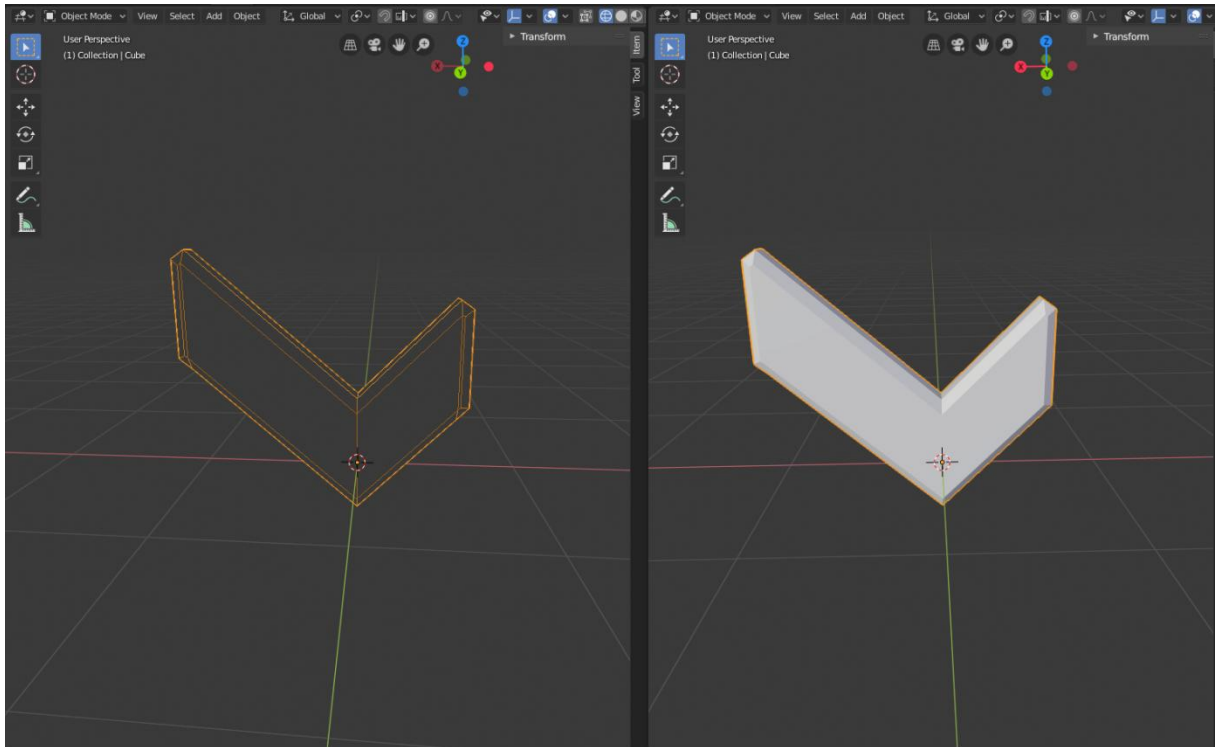


Ilustración 33: Botón Tic.

Siguiendo un diseño parecido a los botones bidimensionales se han elaborado modelos modulares para algunos botones, de tal forma que no se deformen los laterales y esquinas al escalarlos. Para agilizar la colocación de las piezas de estos botones modulares en Unity se han elaborado algunos scripts que se encargan de colocar todos los elementos de un botón en las posiciones que le corresponden además del redimensionar el texto que contiene el botón

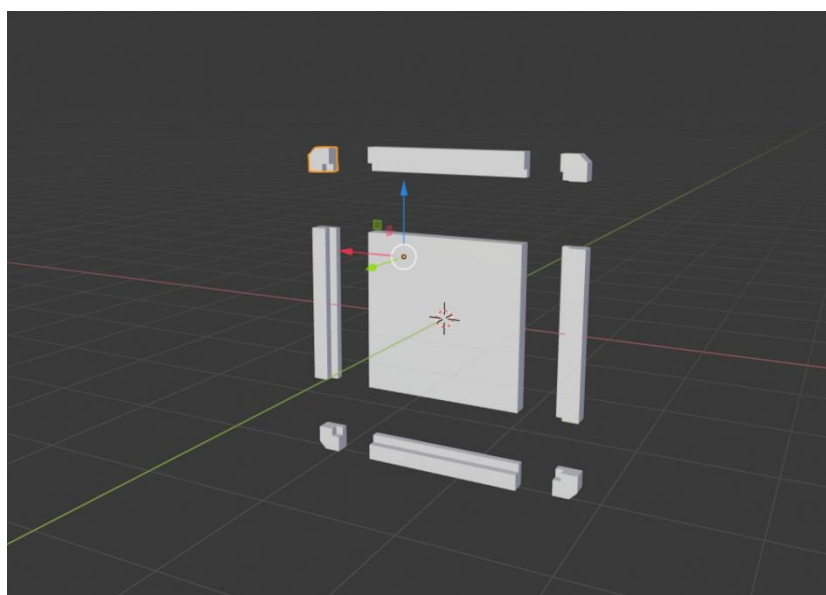


Ilustración 34: Botón modular.

Otros modelos

A parte de los botones se han creado otros modelos. El primero consiste en un fondo con un marco. Este objeto se ha usado en el menú que representa la consola, en el menú que muestra el código java y en el menú de la guía de uso. Este marco se ha modelado modularmente al igual que los botones, de manera que los marcos mantienen su grosor aprovechando así el máximo de espacio. Para su uso en Unity se ha elaborado un script muy parecido al que se usa para el botón.

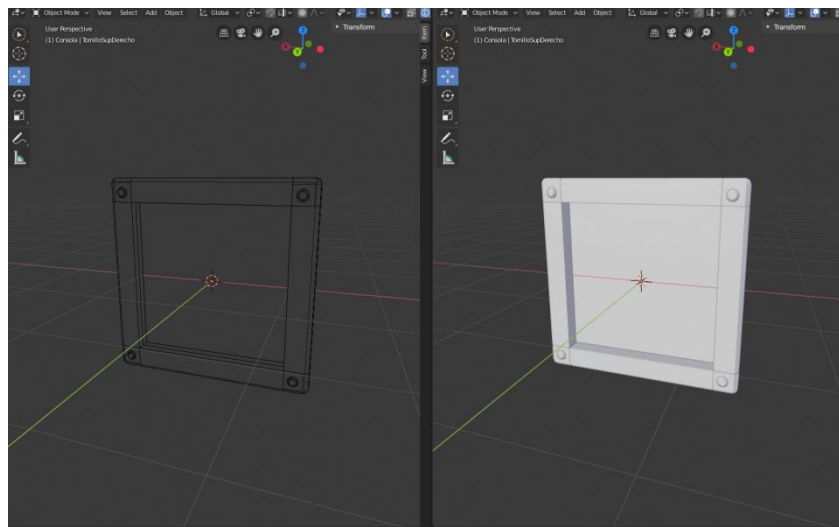


Ilustración 35: Modelo del marco.

El segundo modelo que se ha creado ha sido una papelera. Esta papelera se usa para indicar la eliminación de objetos y va siempre acompañada de un texto superior como apoyo.

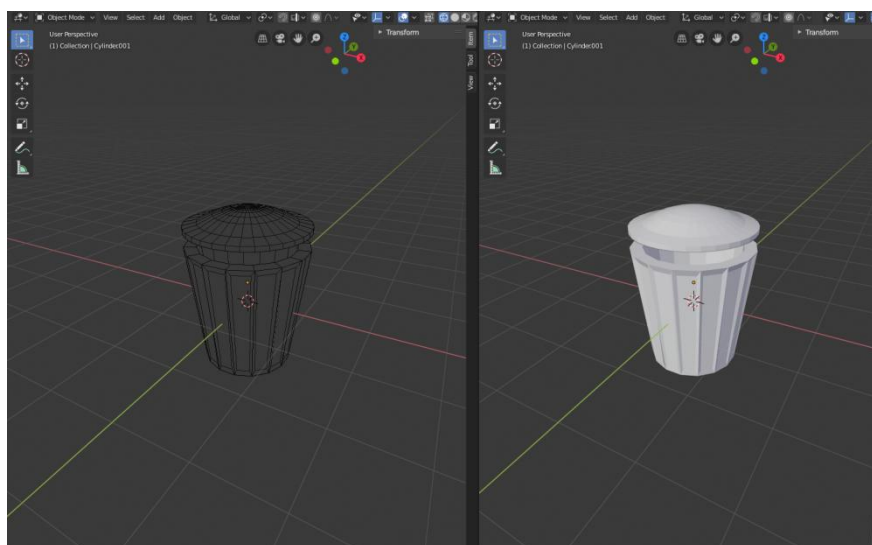


Ilustración 36: Modelo de la papelera.

Implementación en Unity.

Finalmente para acabar con la parte de modelado se han exportado los modelos en FBX. Este formato compatible con la mayoría de motores y en este caso Unity. Una vez exportados basta con introducir los archivos en cualquier carpeta del proyecto y Unity será capaz de reconocerlos y renderizar la malla correspondiente sin necesidad de realizar ningún cambio o trabajo extra.

3.3.14 Diagrama de navegación

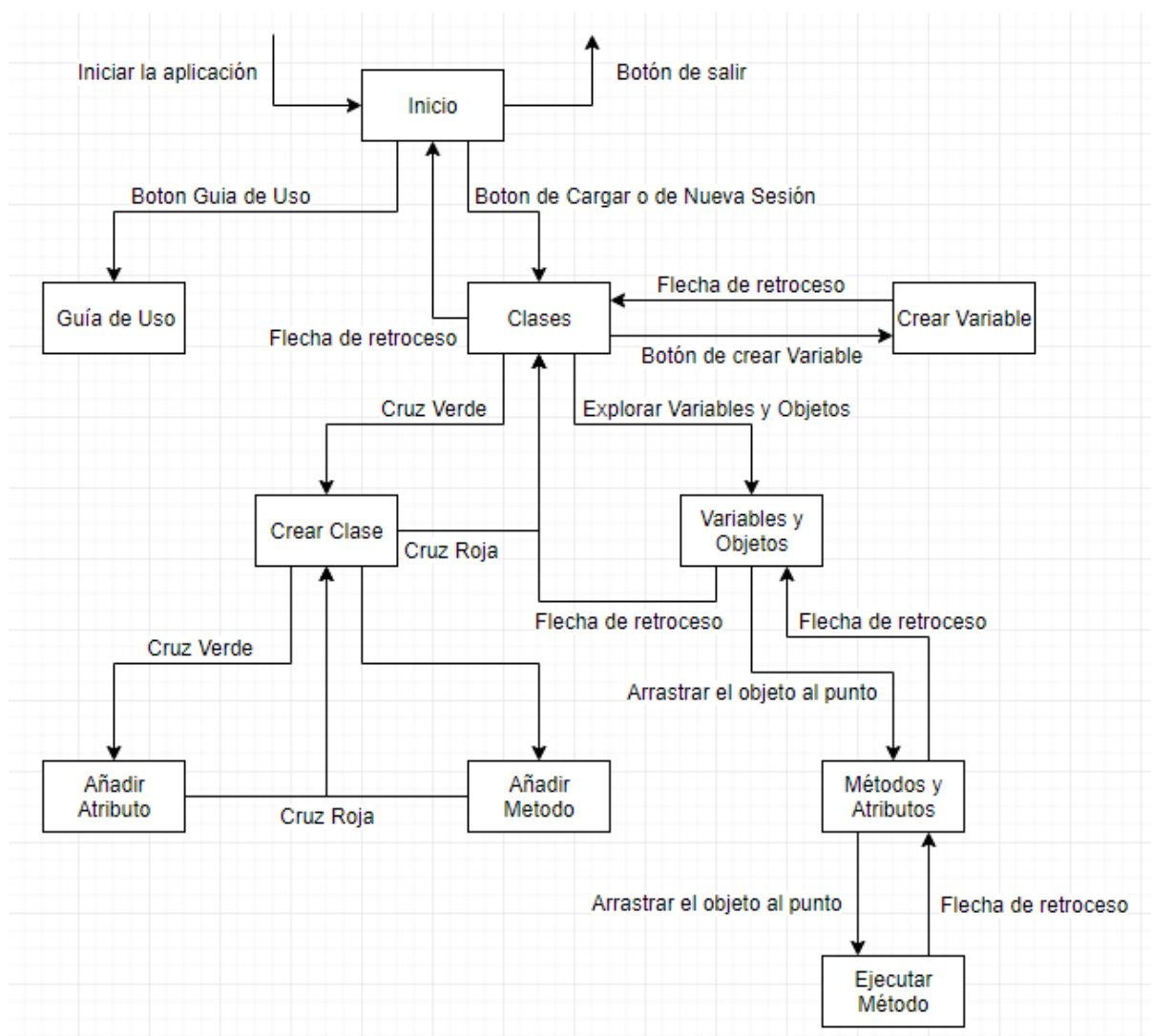


Ilustración 37: Diagrama de navegación

3.3.15 Problemas en el Desarrollo de la Interfaz

El principal problema que se ha encontrado al desarrollar la interfaz ha consistido en el espacio limitado que se tenía para la colocación de todos los elementos. Esto ha supuesto la

necesidad de separar la interfaz en varios menús. Se ha conseguido encontrar un equilibrio entre el tamaño de los elementos, la cantidad de elementos y el número de menús de manera que se pueda acceder a todos ellos de manera sencilla sin tener que pasar por una cantidad excesiva de menús y sin que los elementos estén muy pegados.

Para eliminar elementos excesivos muchas veces se ha buscado una solución que se acercara a la interacción con los objetos en vez del uso de botones. Este es el caso de la acción de eliminar que en toda la aplicación se ejecuta arrastrando el objeto a una papelera.

De esta manera se ha conseguido llegar a una interfaz que aprovecha al máximo el LeapMotion sin usar un dispositivo de realidad virtual.

3.4 Descripción de la herramienta

En esta sección se mostrara el resultado obtenido y como haría uso un usuario de cada una de las partes de la aplicación. Se ha dividido este apartado en las partes por las que pasa el usuario a la hora de usar la aplicación.

3.4.1 Creación y modificación de clases

Para este objetivo se han creado 4 scripts de menor a mayor nivel. Los dos primeros sirven para crear métodos y atributos.

Para la creación de atributos, el script guarda un string con el nombre del atributo y otros dos strings para el tipo y el nivel de acceso. El string del nombre es recogido del objeto que contiene el campo de entrada en la escena.

Además de estas variables, el script contiene un método para abrir el menú. Este método se encarga de borrar toda la información del anterior atributo que se creó y poner unos valores por defecto en los strings de tipo y nivel de acceso.

Para mostrar la información al usuario el menú hace uso de textos en la escena que tienen el mismo valor que el que guarda el script.

Por último el script posee dos métodos, uno para crear un atributo y otro para modificarlo, ambos métodos son muy similares la principal diferencia es que el método de modificación destruye el atributo que se va a modificar antes de crear uno nuevo con los nuevos valores.

Para la creación el método utiliza los strings que contienen los datos e instancia en la escena un objeto del tipo atributo que contiene toda la funcionalidad de un atributo. Acto seguido le

asigna los valores de nombre tipo y nivel de acceso y finalmente lo almacena en el script que se encargara de crear la clase.

En la siguiente imagen podemos observar el menú que gestiona el script.



Ilustración 38: Menú de creación de atributos.

Para la creación de métodos se usa un sistema similar al de atributos. La principal diferencia se encuentra en que el script que gestiona la creación de métodos solo usa un string para identificar el método elegido.

Para mostrarle la información al usuario se usan textos en la escena al igual que con la creación de atributos.

De igual manera el script tiene un método para abrir el menú que se encarga de cambiar el método elegido a uno por defecto y dos métodos para la creación y modificación. Para la creación se instancia en la escena una copia del objeto que representa al método que se ha elegido y para la modificación, antes de crear el método nuevo borra el que se está modificando. Después de instanciar la copia esta se guarda en un diccionario en el script que creara la clase.

En la siguiente imagen podemos ver el menú que gestiona el script.

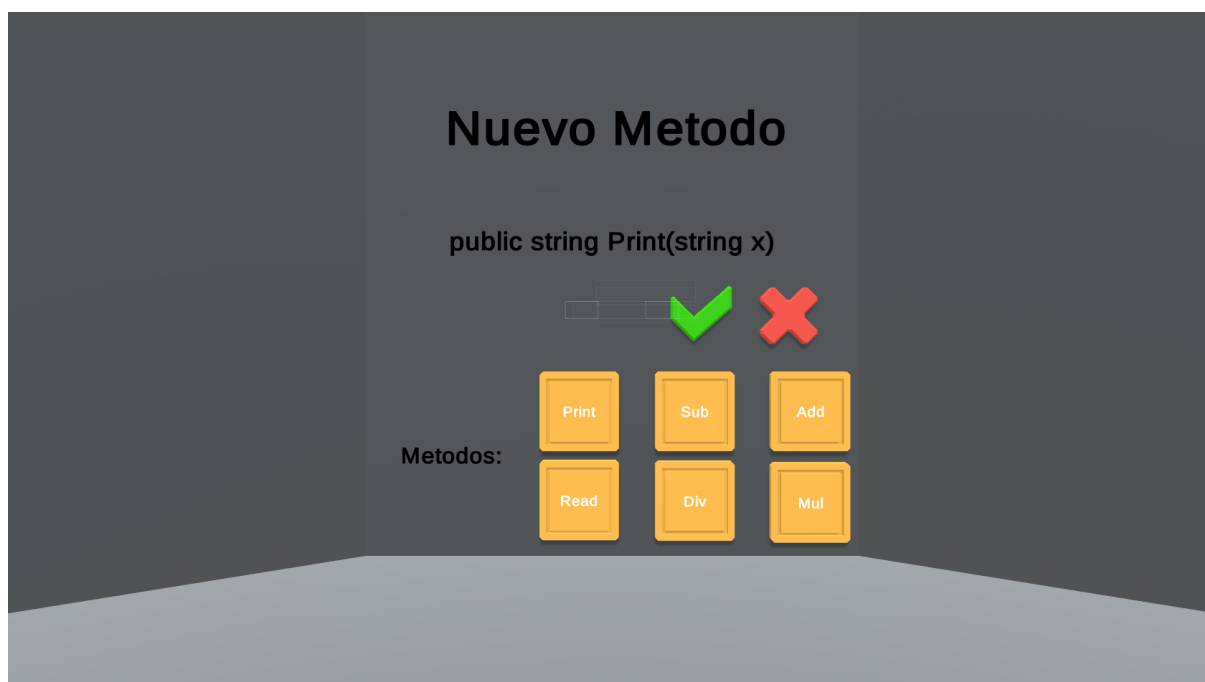


Ilustración 39: Menú de creación de métodos.

El siguiente script es el que se encarga de gestionar el menú de creación de clases. Este script contiene una lista para los atributos y otra para los métodos que se han creado.

Al igual que el script de atributos guarda un string con el nombre para luego asignárselo a la instancia que crea. También tiene un método que se encarga de abrir el menú y vaciar las listas y el nombre.

El método más importante del script es el encargado de crear la clase. Para esta tarea el método se encarga de recorrer la lista de atributos y la lista de métodos colocando las instancias de los métodos y atributos dentro de la instancia de la clase que crea.

Acto seguido le asigna un color a la clase y genera un string con el código Java que corresponde a esa clase. Finalmente esta instancia se almacena en una lista dentro de otro script para la creación de objetos de esta clase.

En la siguiente imagen podemos ver el menú que gestiona el script.

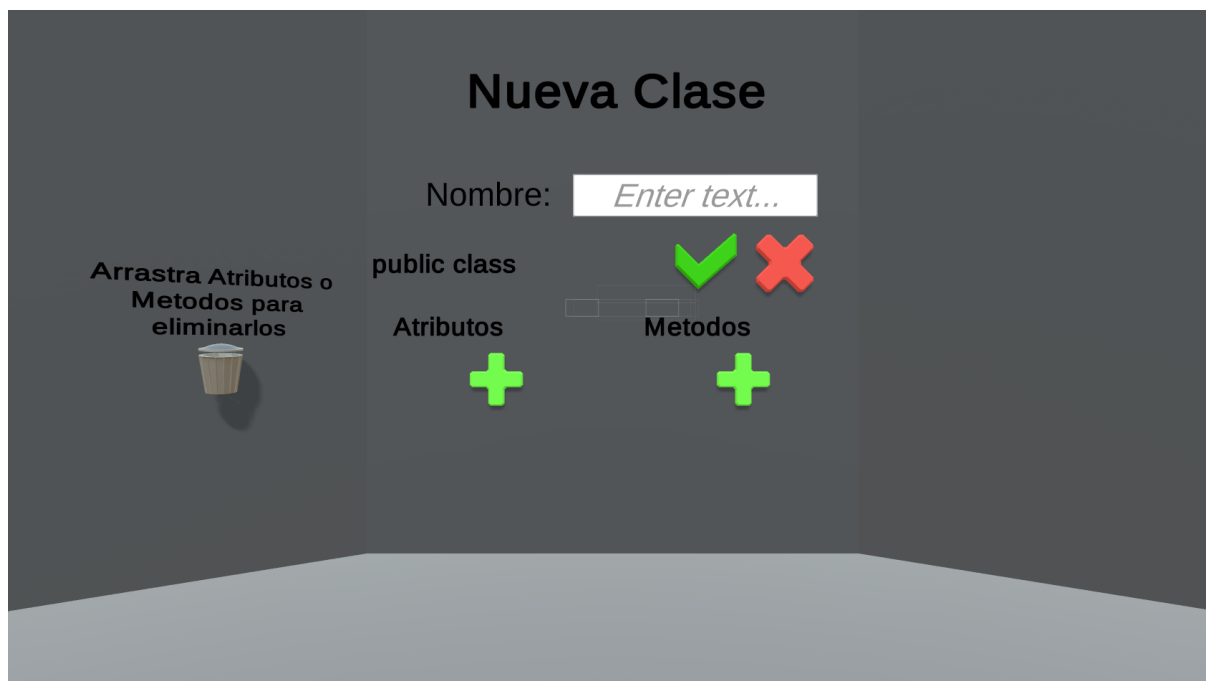


Ilustración 40: Menú de creación de clases.

El último script dentro de este objetivo es un script que se encarga de mostrar al usuario que atributos y métodos lleva creados, sus nombres y el nombre de la clase. Para esta tarea el script usa varios objetos que se encargan de guardar el atributo o método del que presentan la información además de un método para abrir el menú que se encarga de modificarlo. En esta imagen podemos apreciar los atributos y métodos junto a un botón que permite abrir la modificación de estos.



Ilustración 41: Métodos y atributos creados.

Además, este script usa textos en la escena para mostrar la información cuando se está creando un método o un atributo. Como podemos ver en la siguiente imagen.



Ilustración 42: Información textual a la izquierda.

El resultado final se muestra en las siguientes imágenes donde el usuario está creando una clase haciendo uso de los campos de entrada para introducir nombres y los botones para confirmar las acciones.

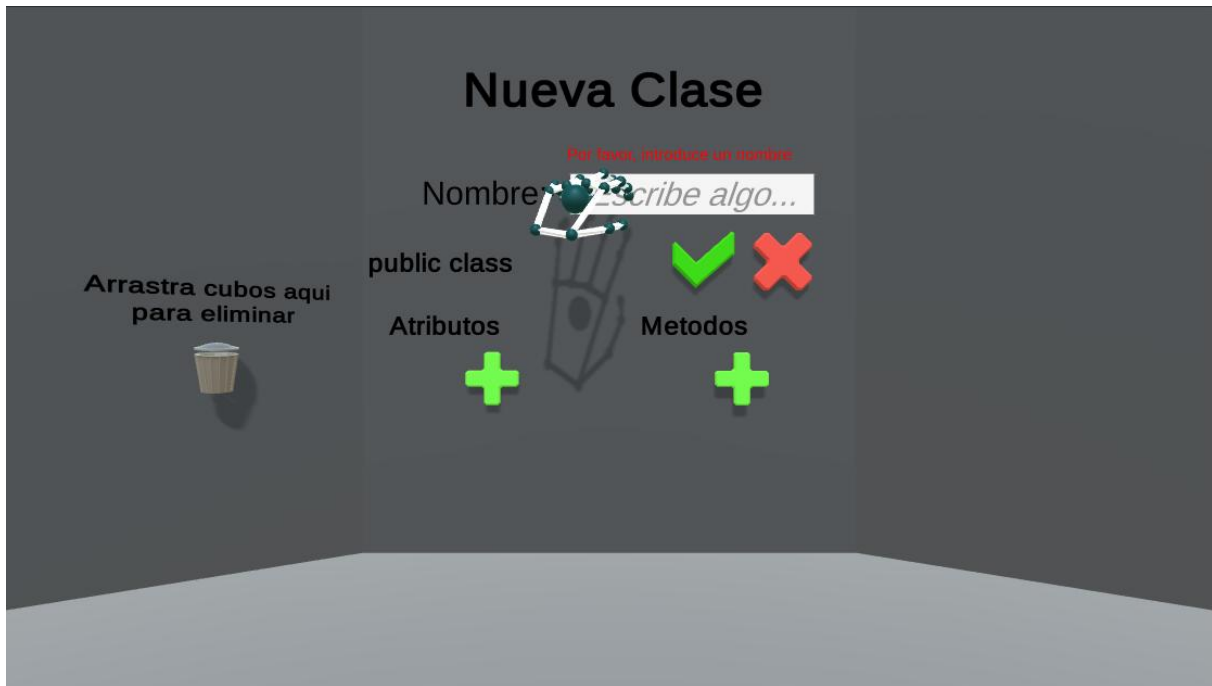


Ilustración 43: Introducción del nombre de la clase.



Ilustración 44: Creación de un atributo.



Ilustración 45: Introducción del nombre del atributo.



Ilustración 46: Confirmación de la creación del atributo.

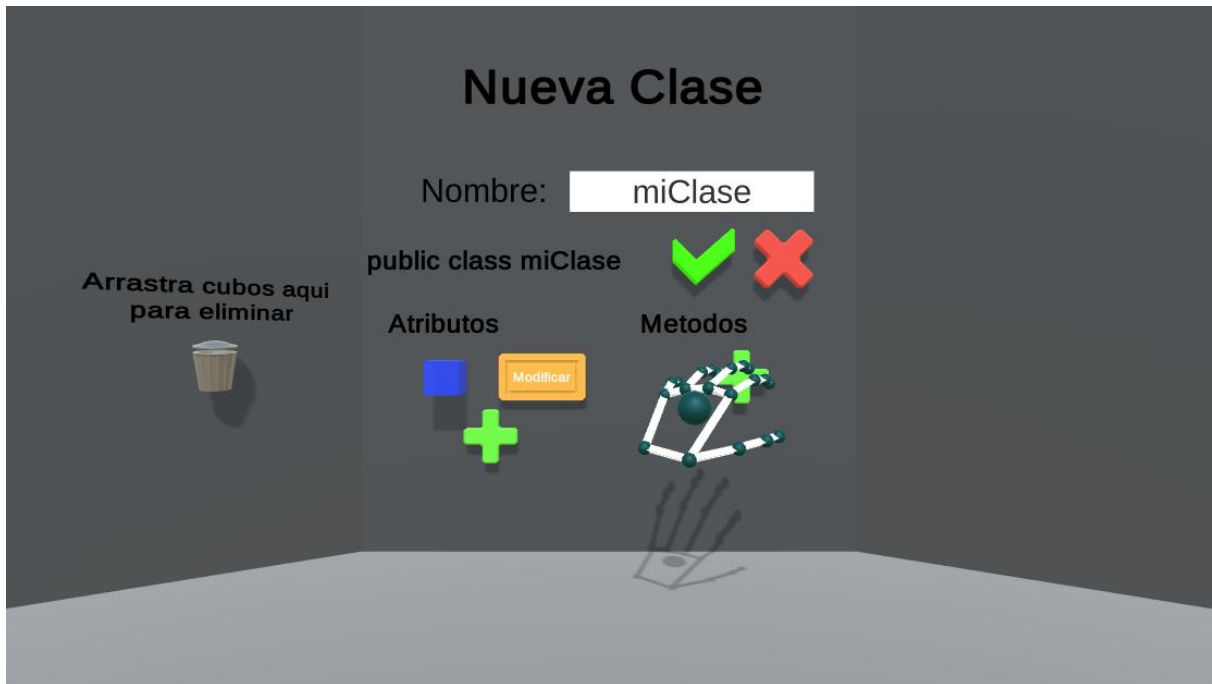


Ilustración 47: Creación de un método.



Ilustración 48: Confirmación del método elegido.

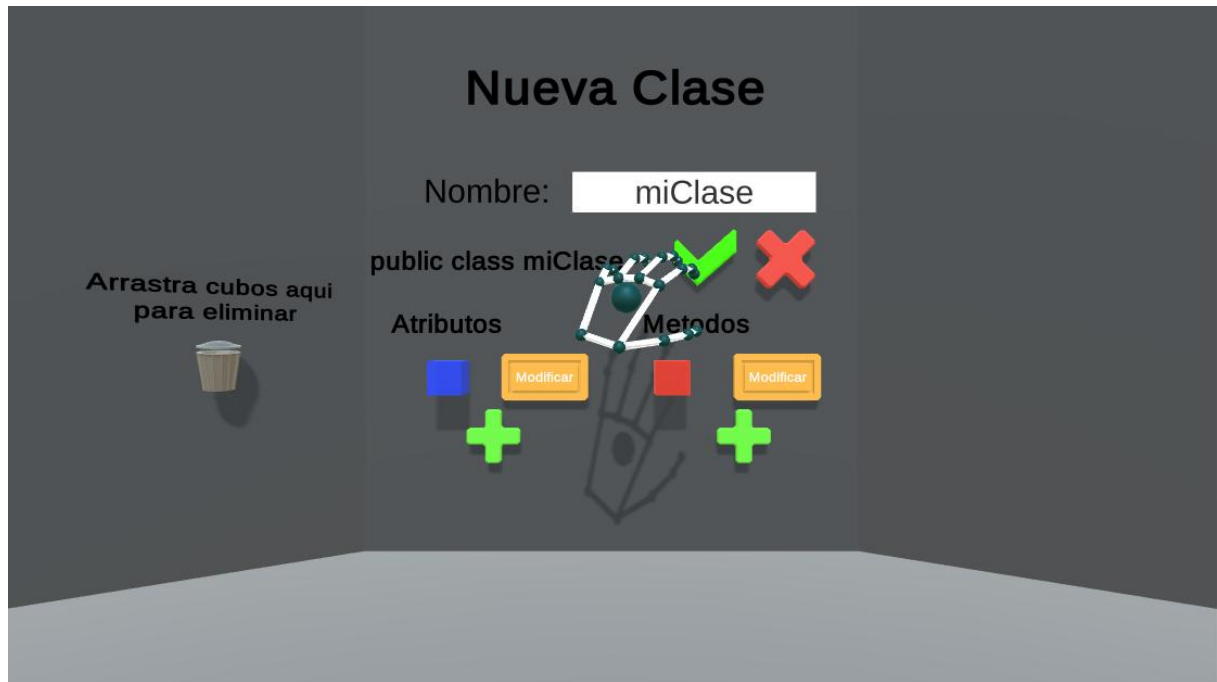


Ilustración 49: Confirmación de creación de la clase.

3.4.2 Creación de Objetos y Variables

Para esta tarea se han creado 3 scripts. El primero de ellos no se encarga expresamente de crear ni los objetos ni las variables pero es el encargado de mostrarnos las clases creadas y permitirnos realizar todas las acciones necesarias con ellas.

El script contiene 3 objetos que contienen otro script. Este script es el encargado de contener los métodos para abrir los menús que necesitamos para cada acción o de llamar a los métodos necesarios. Es simplemente un menú que contiene las clases para su visualización e interacción.

En la siguiente imagen podemos ver el menú que gestiona este script.

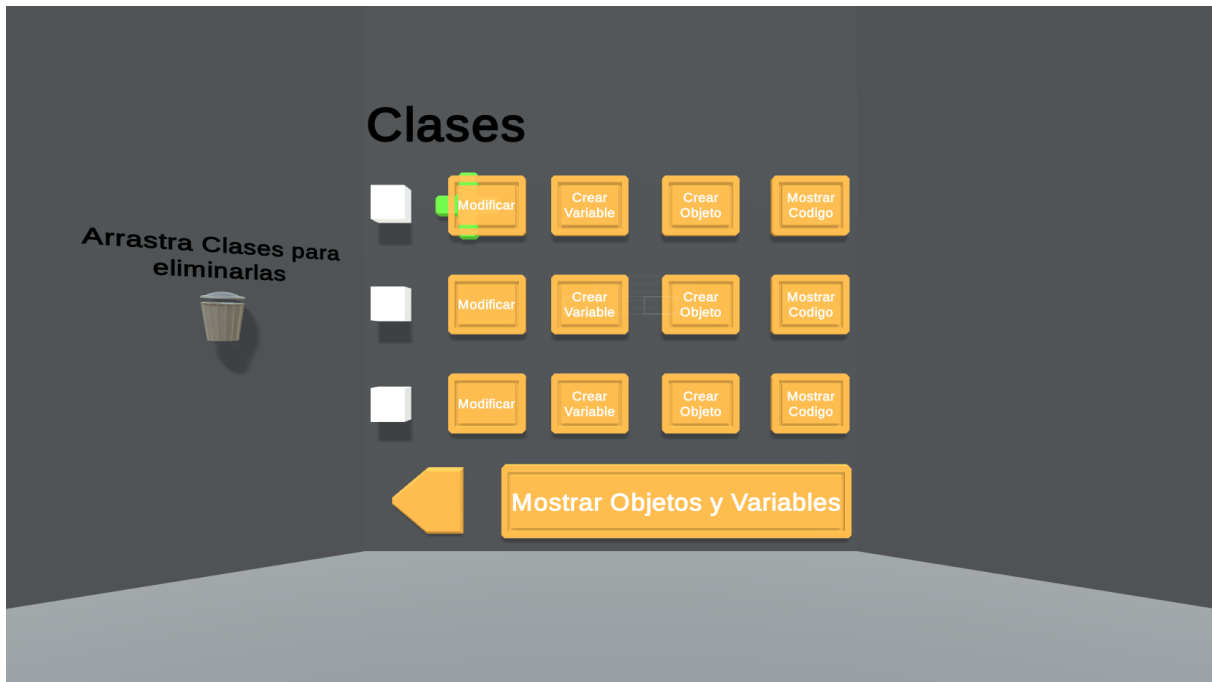


Ilustración 50: Menú de representación de las clases.

Otro script que se usa en el proceso de creación de objetos y variables es un script que gestiona un menú con un campo de entrada. Este script almacena en un string el nombre y es el encargado de llamar al método que creará la variable.

En la siguiente imagen aparece el menú que gestiona este script.



Ilustración 51: Menú para crear variables a partir de clases.

Finalmente se ha creado el script que se encarga de crear los objetos y las variables y almacenarlos en listas.

El método para crear una variable recibe el nombre de la clase, el nombre de la variable, el color de la clase y un color aleatorio. Con estos valores se encarga de instanciar en la escena una copia del objeto que representa una variable con los colores que se pasaron como parámetros.

Después de instanciarla la coloca en la escena dentro de otro menú para que el usuario pueda interactuar con ella cuando desee.

La creación del objeto sigue un proceso similar pero el método solo recibe la posición de la clase, dentro de la lista de clases, de la que se quiere instanciar el objeto. Una vez creado el objeto se coloca al igual que la variable.

En la siguiente imagen se muestra el menú que gestiona este script con instancias de variables y objetos.

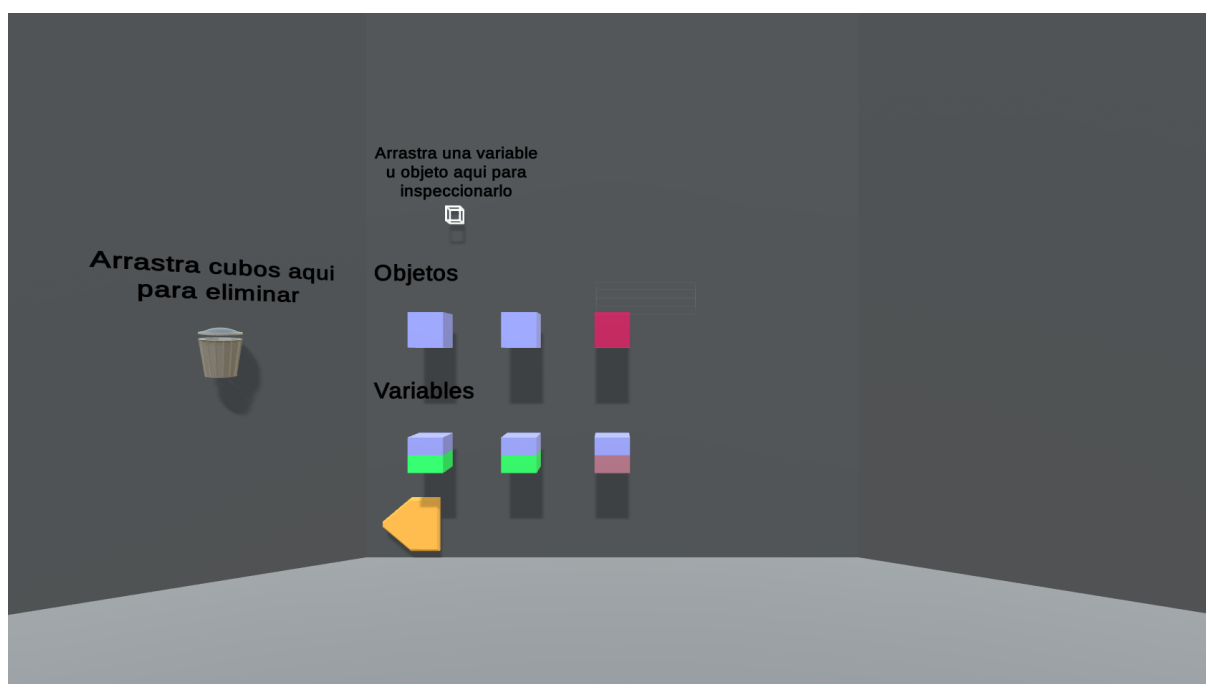


Ilustración 52: Menú con representaciones de variables y objetos creados.

Además de estos dos métodos para la creación el script contiene varios métodos para la eliminación. Un método para eliminar una variable u objeto en concreto, pasándole la variable

u objeto, un método para la eliminación de todas las variables u objetos y un método para la eliminación de todos los objetos y variables que pertenecen a una clase.

En las siguientes imágenes podemos ver las acciones arriba descritas en funcionamiento. Se muestra la creación de un objeto, de una variable y la eliminación tal y como las hace el usuario. También se muestra como el usuario asigna una variable a un objeto.

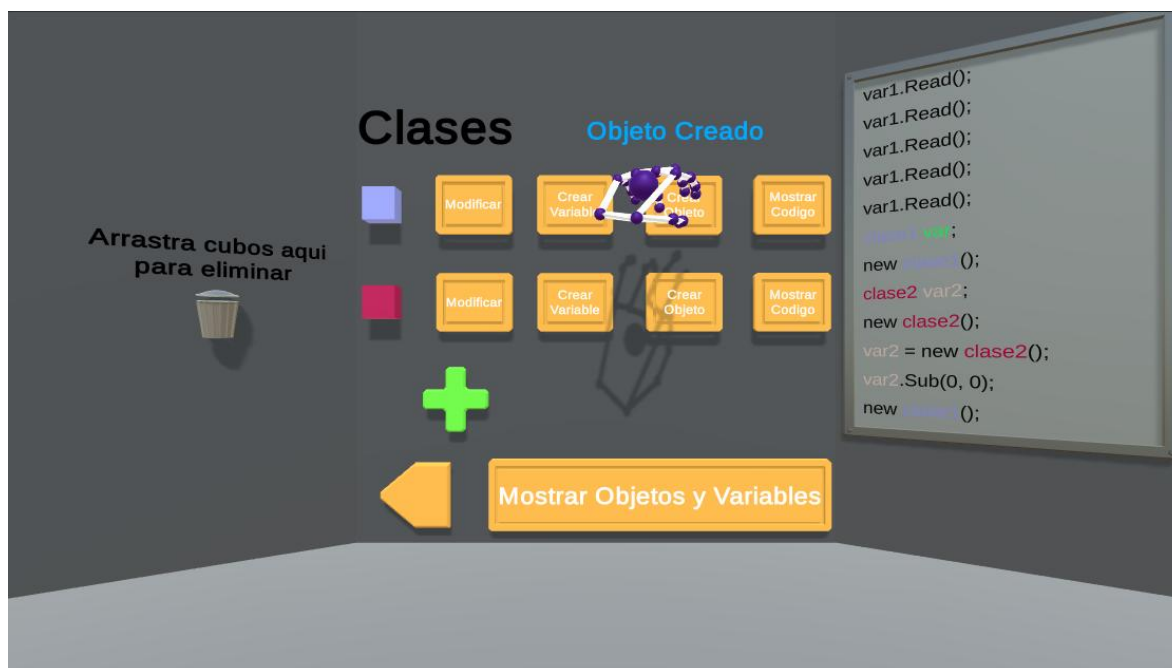


Ilustración 53: Creación de un objeto de la clase de la primera fila.

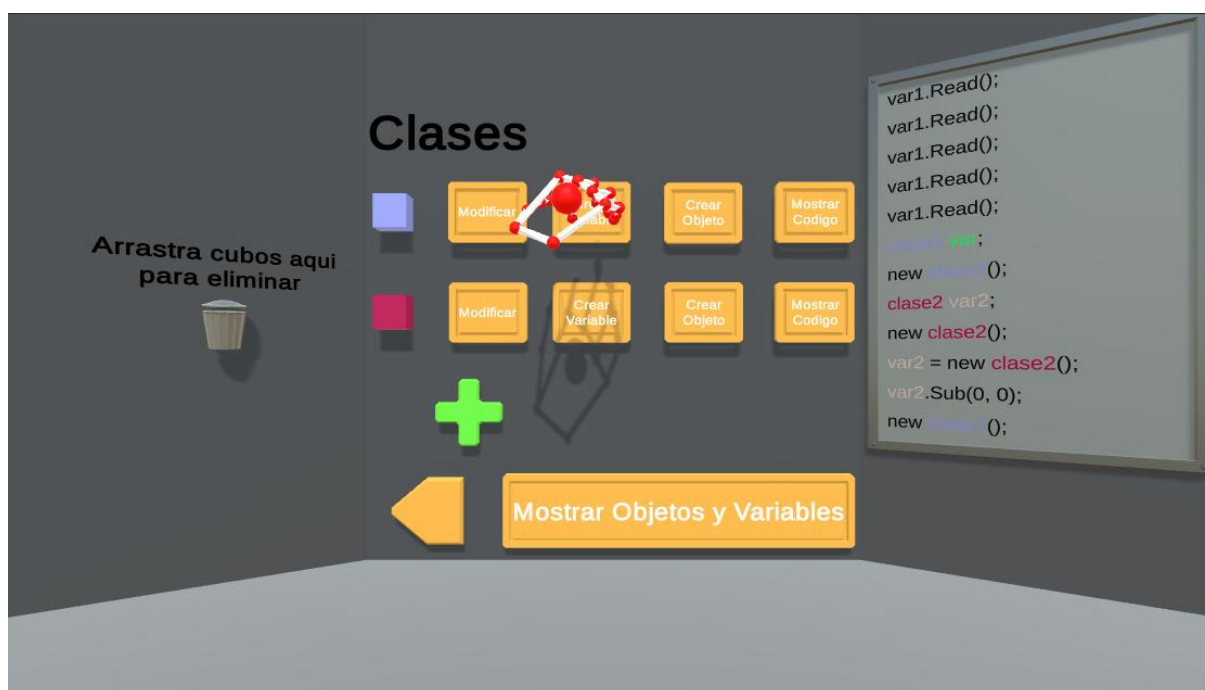


Ilustración 54: Creación de una variable de la clase de la primera fila.



Ilustración 55: Selección del nombre de la variable.

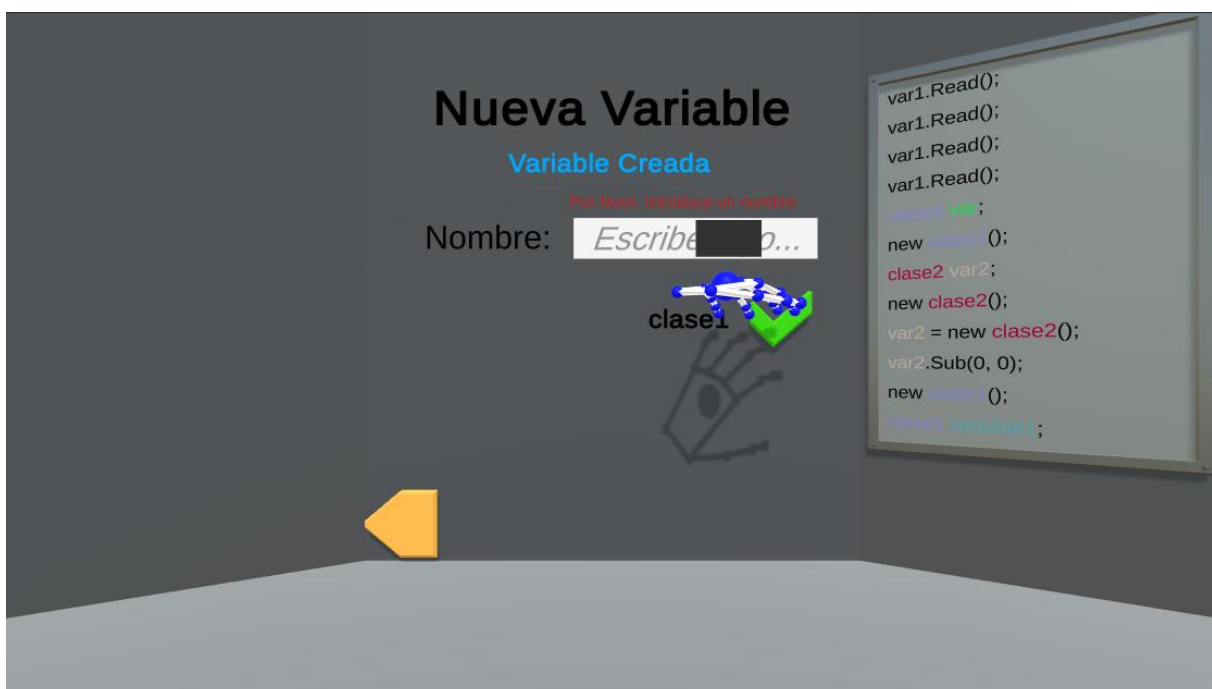


Ilustración 56: Confirmación de la creación de la variable.

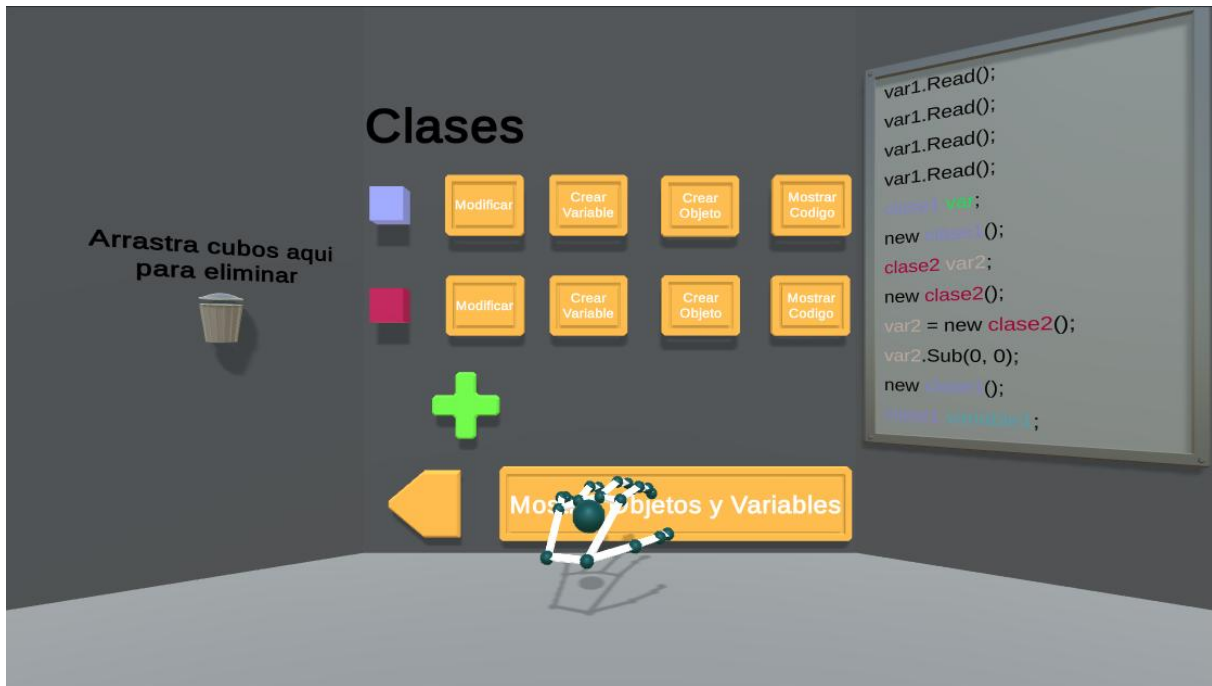


Ilustración 57: Observar los objetos y variables que se han creado.

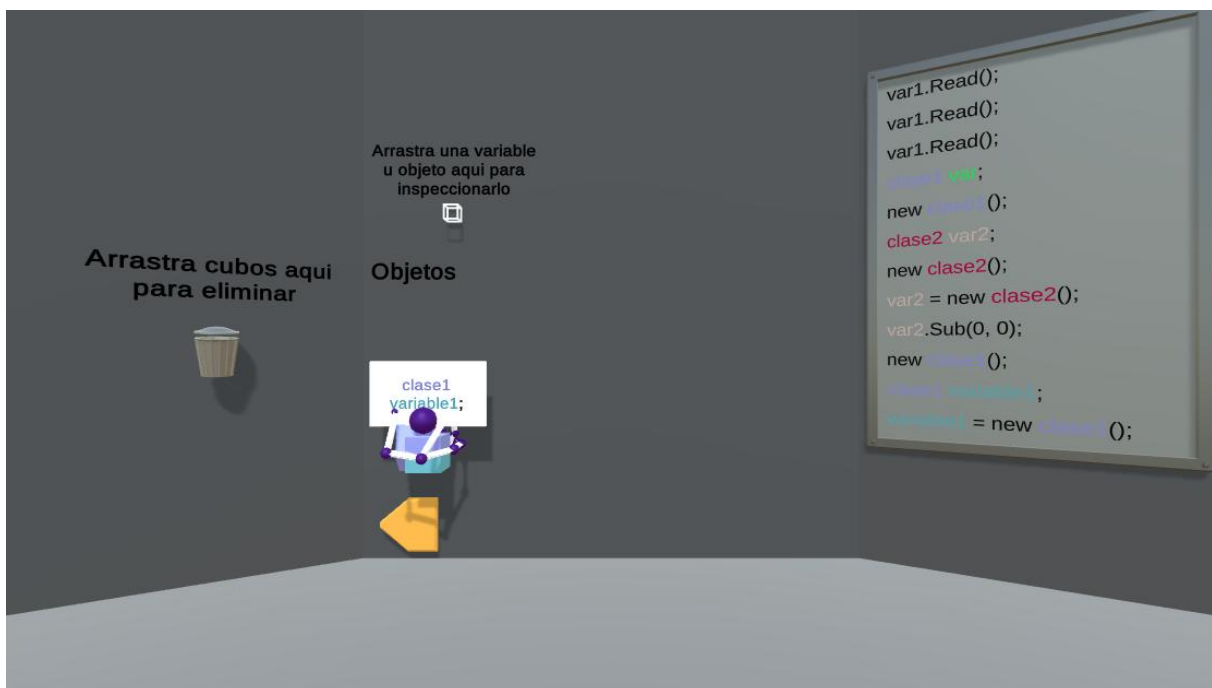


Ilustración 58: Asignación del objeto a la variable.

La asignación se realiza cogiendo un objeto o una variable y tocando con ese objeto o variable un objeto o variable de la misma clase.

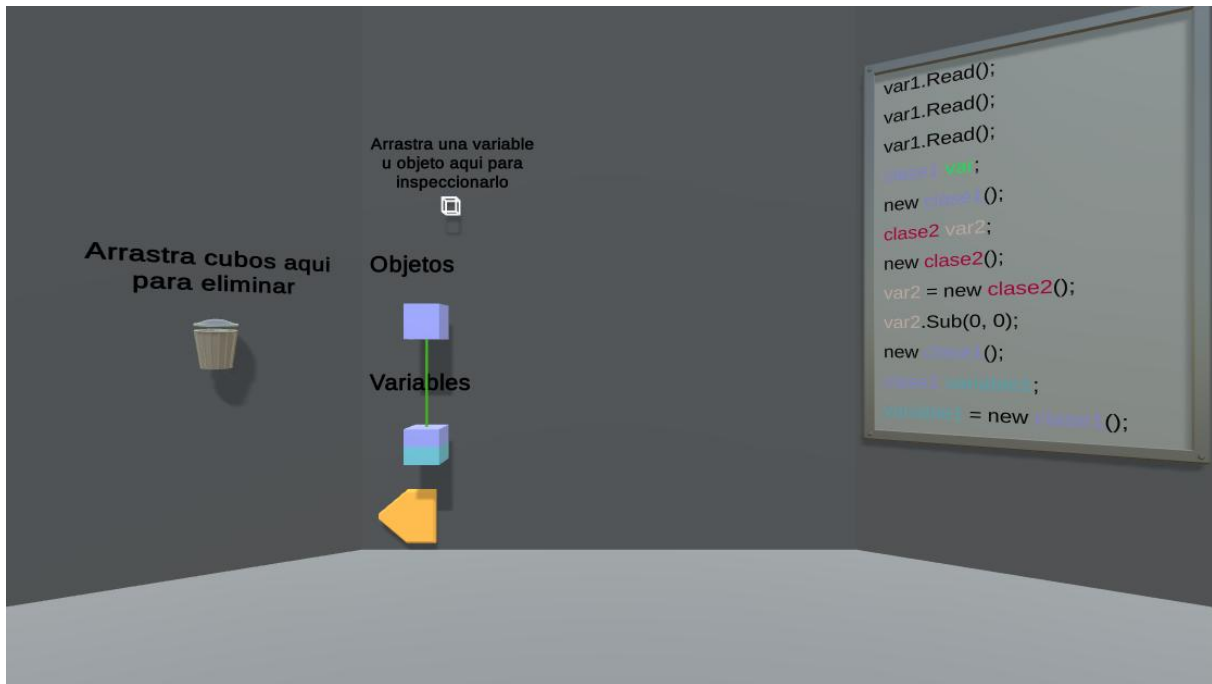


Ilustración 59: Asignación completada.

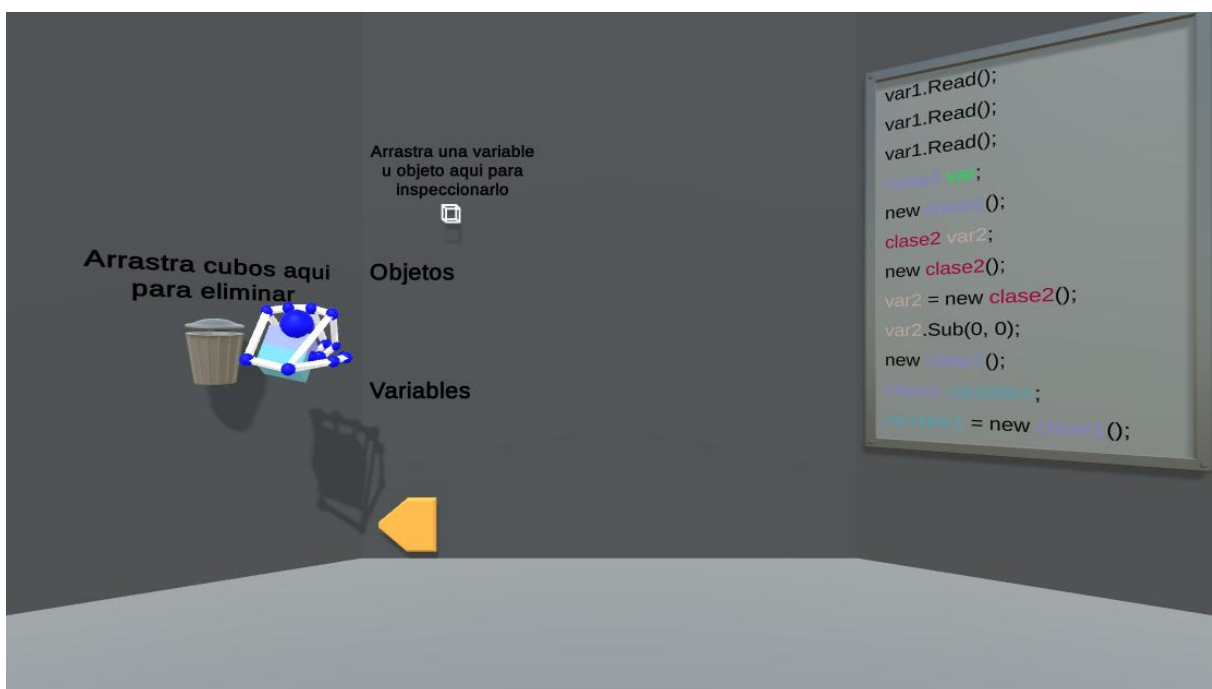


Ilustración 60: Eliminación de la variable y el objeto.

La eliminación se producirá al soltar el objeto o variable cerca de la papelera.

Como podemos apreciar en las imágenes tenemos en todo momento una consola que nos indica textualmente las operaciones que se corresponden con las acciones que realizamos.

3.4.3 Inspección de Variables y Objetos

Para la inspección de variables y objetos se ha creado un script. Este script es el encargado de recibir el objeto o variable que se quiere inspeccionar y hacer una de las siguientes cosas.

Si el menú recibe una variable que no tiene ningún objeto asignado no abre el menú y muestra al usuario un mensaje.

Si el script recibe un objeto, lo guarda, abre el menú y llama a la función de expansión del objeto que muestra los métodos y atributos que tiene dentro.

Si el script recibe una variable con un objeto asignado, guarda la variable y el objeto, abre el menú, llama al método de expansión del objeto y permite la ejecución de métodos.

Para abrir el menú que gestiona este objeto se hace uso de un anclaje al que pueden anclarse tanto variables como objetos. Una vez anclados se intenta abrir el menú con el método que recibe una variable o un objeto arriba descrito.

En las siguientes imágenes podemos ver el menú en los 3 estados. El primero con un objeto pero sin poder ejecutar métodos, el segundo el menú no se ha abierto y estamos en el menú previo con un mensaje y el último con una variable y un objeto.



Ilustración 61: Inspección de un objeto sin estar asignado a una variable.

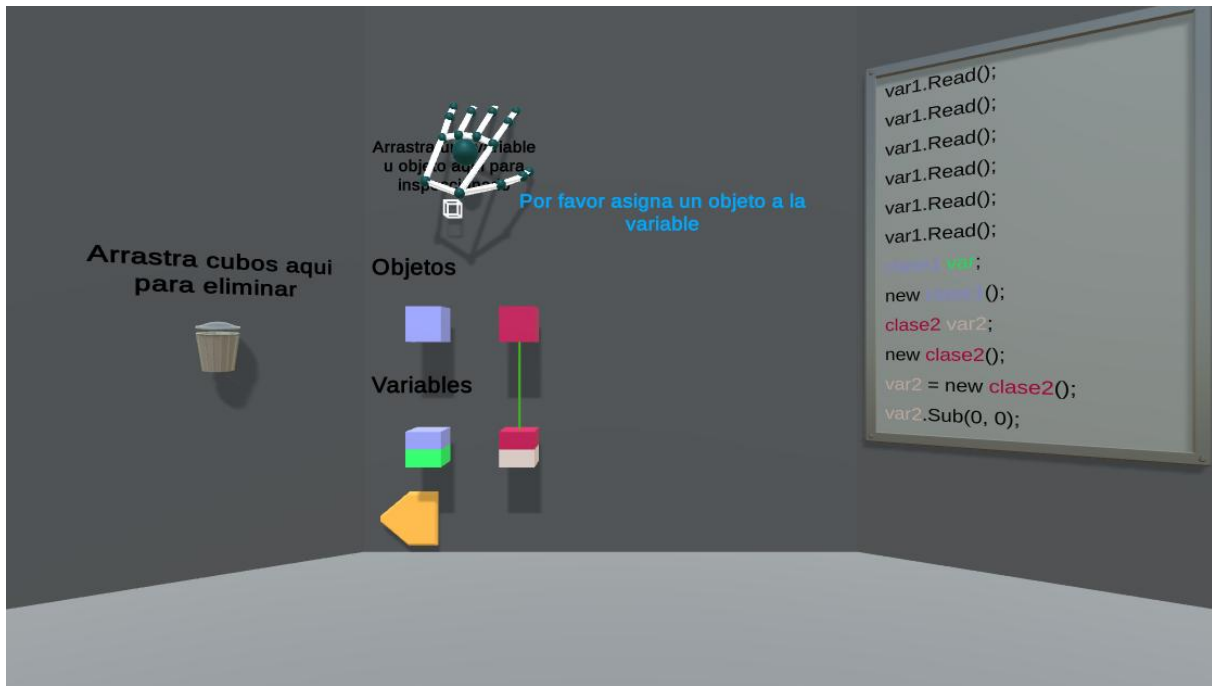


Ilustración 62: Inspección de una variable sin un objeto asignado.



Ilustración 63: Inspección de una variable con un objeto asignado.

3.4.5 Ejecución de Métodos

Esta parte de la aplicación está compuesta de unos menús especiales que no heredan de la clase principal de los menús y han sido llamados Submenús. Todos los Submenús están almacenados en una lista dentro de un script más grande que se encarga de recibir el método a ejecutar y hacer visible el submenú correspondiente.

Estos submenús contienen las entradas de texto necesarias para cada método además de la cabecera del método. También se encargan de mostrar el resultado de la ejecución del método.

El Submenú guarda estas entradas en una lista y cuando se llama a la función de ejecutar envía como string la información guardada al script del método para que lo ejecute.

Una vez se ha terminado la ejecución el método le envía un string al submenú con el resultado para que lo muestre.

La siguiente imagen es un ejemplo de submenú.

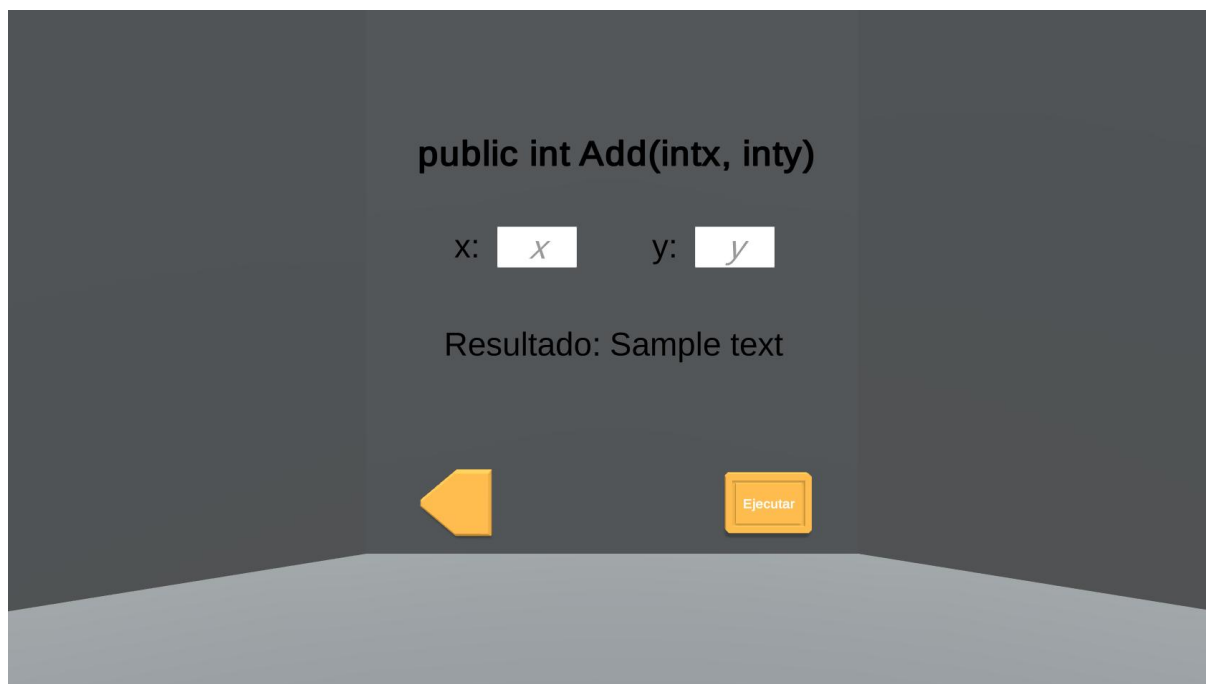


Ilustración 64: Ejemplo de Submenú.

En las siguientes imágenes se muestra la ejecución de un método por el usuario.



Ilustración 65: Inspección del método para su ejecución.

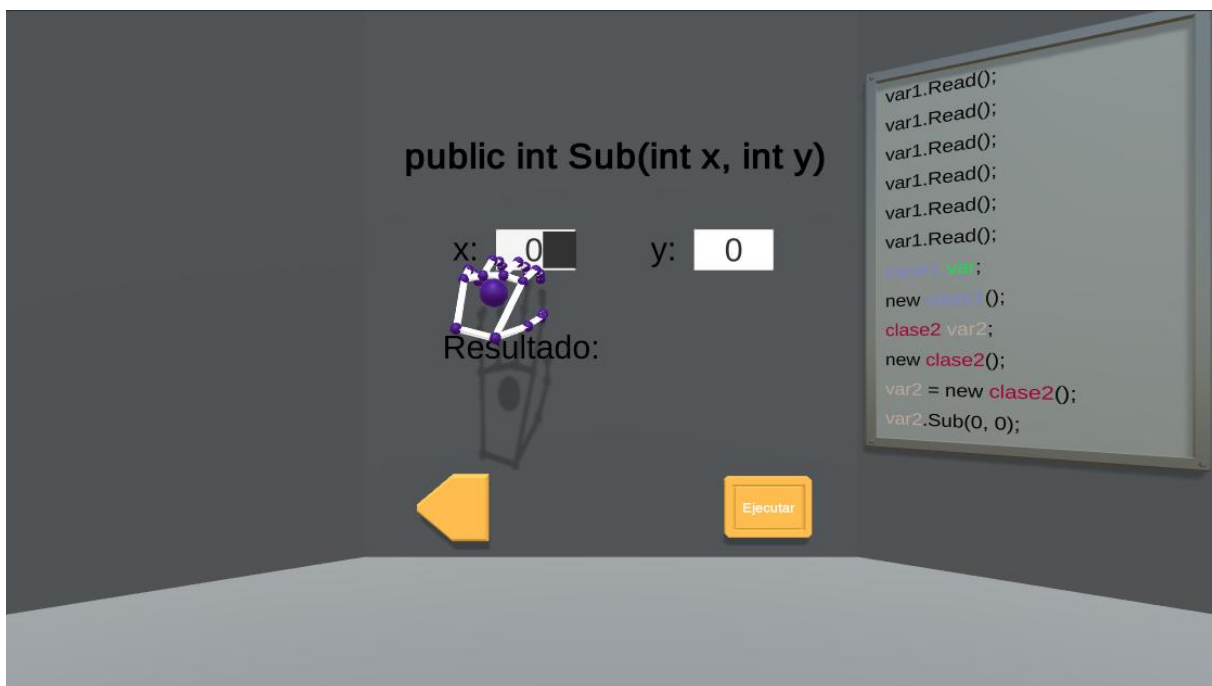


Ilustración 66: Selección de parámetros de entrada.

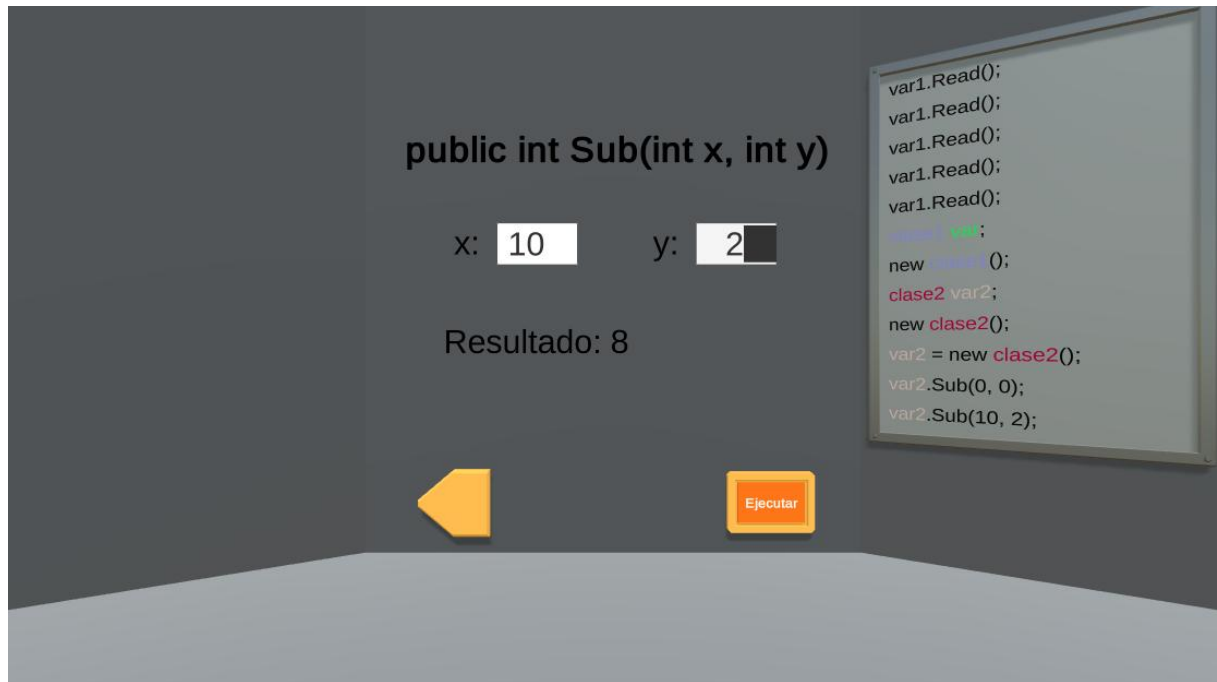


Ilustración 67: Ejecución y resultado.

Capítulo 4: Pruebas

En esta sección se detallaran las pruebas realizadas con la aplicación. Se han realizado pruebas de caja blanca y caja negra y pruebas de usabilidad haciendo uso de las heurísticas de Nielsen.

4.1 Pruebas de caja blanca

En estas pruebas se ha comprobado el correcto funcionamiento de los scripts de interacción de LeapMotion. En concreto se han probado 4 scripts.

El primer script que se ha probado ha sido el encargado de la detección de las manos. Para ello se ha creado un objeto en la escena con el script como componente, se ha configurado el script y se ha ejecutado la aplicación con el dispositivo LeapMotion enchufado.

La detección es correcta y fluida. Las limitaciones de rango del dispositivo son correctas. La detección de movimiento de los dedos también es correcta y fluida.

El siguiente script que se ha probado ha sido el encargado de la interacción con los objetos. Para esta prueba se han creado varios objetos en la escena y se les ha asociado el script de interacción.

Los objetos responden correctamente al contacto con la mano. Los objetos pueden ser agarrados con ambas manos. Los objetos respetan las colisiones y fuerzas del motor de físicas.

El tercer script con el que se han hecho pruebas ha sido el encargado de las interacciones con los botones. Para esta prueba se han creado objetos en la escena que simulan botones y se les ha asociado el script encargado de la interacción.

Los botones responden correctamente al contacto con la mano. Los botones lanzan correctamente el evento de pulsación cuando cumplen las condiciones configurables del script.

El último script que se ha probado ha sido el encargado de las transiciones de los menús. Para esta prueba se han creado 3 objetos en escena. Uno realizará la transición y los otros 2 marcan el inicio y el fin de esta.

El primer objeto hace una transición correcta entre la posición rotación y escala de los otros dos en ambos sentidos. Se ha comprobado que los eventos producidos al iniciar y terminar la transición se lanzan correctamente.

4.2 Pruebas de caja negra

En estas pruebas se ha comprobado que los sistemas que ofrece la aplicación funcionan de manera correcta.

En primer lugar se ha probado el sistema de creación de clases. Para ellos se han creado clases con distinto número de atributos y métodos. También se ha probado a modificar las clases.

Las clases se crean correctamente con toda la información elegida por el usuario. Las clases, atributos y métodos se modifican correctamente y muestran en los menús la información correcta.

El segundo sistema que se ha probado ha sido el de creación de objetos y variables. Para esta prueba se han creado objetos y variables de distintas clases y se han eliminado. Al mismo tiempo que este sistema se probado el sistema para realizar asignaciones de objetos y variables haciendo uso de los objetos y variables creados.

Los objetos y variables se crean correctamente con la información elegida por el usuario. La asignación se realiza correctamente entre objetos y variables de la misma clase. Los objetos y variables se eliminan correctamente.

El siguiente sistema que se ha probado ha sido el encargado de la inspección de objetos y variables. Para esta prueba se han usado los objetos y variables creados para la prueba anterior.

Los objetos muestran correctamente los atributos y métodos que contienen. Se muestra el contenido del objeto correcto al inspeccionar una variable con un objeto asociado.

Finalmente se ha probado el sistema de ejecución de métodos. Para probar este sistema se han usado los objetos y variables de la anterior prueba.

Los métodos se ejecutan correctamente mostrando el resultado al usuario. Solo se ejecutan los métodos cuando se cumplen las condiciones establecidas por la aplicación.

4.3 Pruebas de usabilidad

Para estas pruebas se ha hecho uso de las heurísticas de Nielsen. Son 10 heurísticas que debe cumplir nuestra aplicación para ser usable. Se analizarán todos los puntos y se concluirá si la aplicación los cumple. A cada apartado se le dará una puntuación de 1 a 5. 5 significa que la heurística se cumple y 1 que no se cumple.

- 1. Visibilidad:** La aplicación hace uso de un sistema de transiciones y mensajes para informar al usuario de todos los cambios tanto externos como internos de la aplicación. A este apartado se le da un 5.
- 2. Relación con la realidad:** La aplicación usa palabras y frases cortas con un lenguaje correspondiente al nivel del usuario de la aplicación de tal manera que lo entienda siempre. A este apartado se le da un 3.
- 3. Control y libertad:** La aplicación permite hacer y deshacer cambios mediante sus sistemas de creación, eliminación y modificación de clases variables y objetos. A este apartado se le da un 4.
- 4. Consistencias y estándares:** La aplicación mantiene unas convenciones. Por ejemplo la forma del botón de aceptar siempre será un tic, la de volver atrás una flecha o la de cancelar una cruz roja. A este apartado se le da un 5.
- 5. Prevención de errores:** La aplicación indica en cada momento que acción se ha realizado y el efecto que tiene es visible para evitar que el usuario cometa errores en la utilización. A este apartado se le da un 3.
- 6. Reconocimiento:** Toda la información es visible al usuario cuando la necesita. Por ejemplo, las clases solo mostrarán su nombre cuando el usuario desee verlas acercando la mano. Mientras tanto esa información no ocupará espacio en la interfaz. A este apartado se le da un 4.
- 7. Flexibilidad:** El uso de la aplicación mejora en fluidez con el uso frecuente de ésta al adaptarse a los controles gestuales. A este apartado se le da un 4.
- 8. Estética y minimalismo:** La información innecesaria se oculta al usuario tras interacciones sencillas que le permitirán ver esa información cuando lo desee. A este apartado se le da un 4.
- 9. Recuperarse de los errores:** Se le indica al usuario en todo momento que acciones necesita hacer para continuar usando la aplicación. También se incluye una lista con los errores más comunes y sus soluciones. A este apartado se le da un 5.

10. Ayuda y documentación: Se incluye dentro de la aplicación una guía de uso con consejos y errores comunes para facilitar los primeros usos de la aplicación al usuario. A este apartado se le da un 4.

Podemos concluir que la aplicación es usable pues cumple todas las heurísticas de Nielsen. Este ha sido siempre un objetivo a la hora de desarrollar la interfaz para el usuario de manera que fuera cómoda y usable.

Capítulo 5: Conclusiones

El desarrollo de la aplicación ha sido un proceso enriquecedor por las tecnologías usadas, principalmente el LeapMotion. Comenzare hablando de los puntos más personales.

Esta tecnología era desconocida para mí antes de comenzar el desarrollo de este proyecto y ha conllevado un proceso de investigación previo a la implementación bastante extenso. Una vez superada esta barrera la sensación de añadir elementos que funcionaran con el LeapMotion era gratificante.

Además del aprendizaje de esta nueva tecnología el trabajo también ha conllevado el aprendizaje del desarrollo de una interfaz completa y desde 0. Cosa con la que nunca antes no se había trabajado. Además esta interfaz sigue un paradigma de realidad virtual que es aun más raro de encontrar en aplicaciones convencionales.

En cuanto a los objetivos, se han cumplido todos satisfactoriamente:

- **Crear clases con atributos y métodos, estas clases tienen que ser creadas dentro de la aplicación para facilitar el uso de la aplicación por parte del usuario.** Para este objetivo la aplicación posee 3 scripts que se encargan de permitir al usuario la creación y modificación de clases con métodos y atributos, permitiéndole editar el nombre y contenido de la clase como desee.
- **Modificar las clases, atributos y métodos ya creados.** Este objetivo va de la mano del de arriba y se ha implementado justo después de la creación de clases haciendo modificaciones en los mismos 3 scripts que se encargan de la creación.
- **Ver el código java que compone la clase que ha creado.** Para este objetivo se ha creado un script que se encarga de mostrar el código de la clase que ha sido almacenado en un string al crearla.
- **Crear objetos y variables a partir de una de las clases creadas.** Este objetivo se ha cumplido con la creación de 1 script que se encarga de crear y eliminar los objetos y las variables y mantenerlas almacenadas para su uso.
- **Coger y mover con las manos las representaciones físicas de objetos, variables métodos y atributos y presionar con las manos todos los botones en la escena.** Este objetivo se ha cumplido haciendo uso de los scripts que provee LeapMotion y de otros scripts que heredan de los anteriores. Se ha conseguido llegar a un control satisfactorio y se han reducido al mínimo problemas derivados del uso del motor de físicas.

- **Realizar asignaciones de objetos a variables.** Para este objetivo se ha implementado funcionalidad en las variables que permite detectar cuando el usuario toca una variable con un objeto y viceversa.
- **Inspeccionar el contenido de un objeto o variable creados a partir de una clase.** Este objetivo se ha cumplido haciendo uso de un script que se encarga de mostrar los objetos físicos que se encuentran dentro del objeto, de tal manera que el usuario pueda interactuar con ellos como lo hace con el resto de objetos en la aplicación.
- **Ejecutar los métodos de un objeto.** Para este objetivo se han creado varios scripts que se encargan de pasar la información que necesita el método para ejecutarse y recibir el resultado de la ejecución y mostrarlo.

En cuanto a la interacción esta principalmente basada en botones para los menús. Esta decisión fue tomada por las características de la interfaz y el mundo virtual y por las limitaciones del LeapMotion. Se considera que se cumple este objetivo de manera satisfactoria para el usuario.

La segunda parte de la interacción consistía en los objetos que representabas las variables, atributos, métodos y objetos. Esta interacción se ha conseguido realizar de una manera natural pudiendo coger los objetos, mirarlos, explorarlos y eliminarlos solo con las manos.

5.1 Conclusión final

LeapMotion ha sido una herramienta muy útil que le aporta mucho valor a la aplicación. Sin embargo, no está pensado totalmente para un uso fuera de la realidad virtual lo que ha generado problemas. Aun así estos problemas han podido ser solventados y se ha obtenido una aplicación completa y satisfactoria.

Por su parte, Unity ha resultado de gran utilidad a la hora de trabajar con LeapMotion por todo el código ya desarrollado para este motor. Si bien está pensado para el desarrollo de videojuegos, es un código muy útil que sirve para otra infinidad de usos. Tras haber finalizado la aplicación se considera Unity como una herramienta ideal para trabajar con LeapMotion.

Personalmente considero que la aplicación es útil a la hora de transmitir los conocimientos de POO que de manera más teórica al usuario le resultan más difíciles de aprender. Todo esto es gracias a la potencia de la interacción que permite ver los conceptos y no tener que simplemente imaginarlos.

5.2 Trabajos Futuros

Finalmente me gustaría comentar algunos trabajos futuros que se han valorado para esta aplicación:

- **Realidad virtual:** Un cambio que resultaría increíblemente positivo para este tipo de aplicación sería el uso de unos cascos de realidad virtual. Esta mejora tendría consecuencias muy grandes en el desarrollo pues habría que cambiar toda la interfaz pero supondría una mejoría en interacción del usuario con el mundo y potenciaría así ese factor de interacción en el que la aplicación se apoya para conseguir sus objetivos.
- **Mas gestos:** Por las limitaciones del hardware se ha recurrido a una interfaz basada en botones. Si la aplicación pasase a una realidad virtual completa, la colocación del LeapMotion sobre el casco de realidad virtual mejoraría enormemente la detección y permitiría la implementación de mas gestos que podrían sustituir a botones.
- **Mejor interfaz:** Como se ha mencionado antes, la interfaz tendría que ser rehecha si se cambiara a una realidad virtual completa usando un casco de realidad virtual. Se apuesta por la creación de un mundo virtual donde el usuario pueda moverse y en el que se usen objetos cotidianos para la explicación de los conceptos. Un ejemplo sería un aula con una mesa en la que estén los objetos y puedas interactuar con ellos totalmente. Si hubiera necesidad de algún menú, éstos podrían estar a tu alrededor y no fijos en una pared.

Bibliografía

- .NetFramework*. (2019). Obtenido de <https://dotnet.microsoft.com/>
- BaymaxDreams*. (s.f.). Obtenido de <https://unity.com/es/madewith/baymax-dreams>
- Blender*. (s.f.). Obtenido de <https://www.blender.org/>
- Documentación de LeapMotion para Unity*. (s.f.). Obtenido de <https://leapmotion.github.io/UnityModules/interaction-engine.html>
- Documentación de TextMeshPro*. (s.f.). Obtenido de <http://digitalnativestudios.com/textmeshpro/docs/>
- Documentación de Unity*. (s.f.). Obtenido de <https://docs.unity3d.com/2018.4/Documentation/Manual/>
- GitExtension*. (s.f.). Obtenido de <http://gitextensions.github.io/>
- Github*. (s.f.). Obtenido de <https://github.com/>
- Heurísticas de Nielsen*. (s.f.). Obtenido de <https://www.beeva.com/beeva-view/disenio-y-ux/logica-de-la-usabilidad/>
- Kenney*. (s.f.). Obtenido de <https://www.kenney.nl/>
- LeapMotion*. (s.f.). Obtenido de <https://www.leapmotion.com/>
- PolyglotTool*. (s.f.). Obtenido de <https://github.com/WilgnerFSDev/PolyglotTool-Unity>
- Prefab Workflow Unity 2018.3*. (s.f.). Obtenido de <https://unity.com/es/prefabs>
- Unity*. (s.f.). Obtenido de <https://unity.com/>
- Visual Studio Code*. (s.f.). Obtenido de <https://code.visualstudio.com/>

Apéndices

Apéndice A: Guía de uso

A continuación se detallan una serie de consejos para usar la aplicación de forma correcta y posibles errores y como solucionarlos.

Consejos

- Colocar las manos sobre el dispositivo LeapMotion a una altura de 15-18cm.
- Mantener las manos lejos de los botones si no se van a pulsar
- Alejar las manos de los botones tras pulsar un botón.
- La aplicación guarda su estado al volver al menú de inicio, no se recomienda cerrarla forzosamente a menos que no haya otra opción.
- Para conocer más información sobre un objeto pasa la mano cerca del objeto.
- Para coger un objeto acerca la mano al objeto y ciérrala.
- Para destruir objeto cógelo y muévelo a la papelera.

Problemas

- Si un botón no reacciona a las pulsaciones, aleja las manos del dispositivo LeapMotion y vuélgelas a colocar sobre él.
- Si el botón sigue sin responder pulsa otros botones y vuelve a probar a pulsar el botón.
- Si el botón sigue sin responder sal de la aplicación y vuélvela a inicial.
- Si las manos funcionan incorrectamente, retira las manos del dispositivo y vuélgelas a colocar sobre él.

Apéndice B: Guía de Instalación

La aplicación no requiere ninguna instalación previa, basta con descargar el Zip con la carpeta que contiene el ejecutable del siguiente link de github:
<https://github.com/pabloski300/POOLeapMotion/releases>

Una vez descargado descomprime el Zip en cualquier carpeta y ábrela, dentro encontraras un UnityCrashHandler que no hace falta tocar y el ejecutable, basta con hacer doble clic y la aplicación comenzara a funcionar.

Para usar el dispositivo LeapMotion basta con enchufarlo al ordenador a través de un puerto USB antes o durante la ejecución de la aplicación.