UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO Y EVALUACIÓN DE UN IDE EN REALIDAD VIRTUAL

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

VÍCTOR STEFANO SEGURA CASTILLO

PROFESOR GUÍA:
ALEXANDRE BERGEL

MIEMBROS DE LA COMISIÓN:
JOSÉ A. PINO URTUBIA
DANIEL CALDERÓN SAAVEDRA

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: VÍCTOR STEFANO SEGURA CASTILLO
FECHA: 2021
PROF. GUÍA: ALEXANDRE BERGEL

DESARROLLO Y EVALUACIÓN DE UN IDE EN REALIDAD VIRTUAL

La realidad virtual consiste en la simulación de experiencias de la vida real, generalmente usando espacios en tres dimensiones generados por computadora. Aunque no sea usado de forma masiva en la actualidad, sí ha ganado terreno no sólo en el campo de los videojuegos, sino también en la ciencia y la investigación.

En ese sentido, nos preguntamos si es posible extender el concepto de realidad virtual al área de ingeniería de software. La popularización de la programación en vivo y las librerías de visualización ágil de código han otorgado una mejor experiencia al momento de generar y depurar código. Un entorno de programación en realidad virtual puede permitir tener estas herramientas a mano de forma inmediata, inmersiva y, por sobre todo, interactiva. Para lograr este objetivo, se hizo uso de Smalltalk Pharo como principal lenguaje de programación, capaz de recibir instrucciones mediante la librería ZincHTTP. Por otro lado, se utilizó el motor gráfico Unity3D, que contiene todas las herramientas necesarias para crear un entorno de realidad virtual.

Durante la implementación, se replicaron los principales elementos que componen un entorno de desarrollo en Pharo. Estos son: Playground, Browser, Inspector y Transcript. Usando primitivas básicas en 3D, fue posible traducir visualizaciones de la librería Roassal a modelos en tres dimensiones. Debido a las limitaciones que presentan las herramientas de realidad virtual y los controles, se optó por incluir una representación 3D de un teclado simplificado. Con esto, es posible añadir más opciones de interactividad dependiendo del tipo de usuario.

En general, los sujetos experimentales expresan mayor incomodidad al momento de escribir código, sumado a los errores que dependen del dispositivo que utilizan y su disponibilidad, así como los avances en tecnología y herramientas de software que interfieren con la compatibilidad. No obstante, fueron capaces de realizar las tareas de programación solicitadas, así como crear visualizaciones de las mismas.

Dadas las limitaciones mencionadas anteriormente, se concluye sí es posible crear un ambiente de Live Programming en realidad virtual. Por otro lado, se propone que en un trabajo futuro las limitaciones se vayan disipando y se pueda explorar nuevos usos de la tecnología para realizar tareas de programación.

Dedicado al Intelligent Software Construction Laboratory*, y a mi familia, por confiar en que llegaría hasta aquí.*

Agradecimientos

Agradezco al profesor Alexandre Bergel por permitirme formar parte de este ambicioso proyecto, así como proveerme del equipo necesario para lograrlo. También a los miembros del ISCLab: Milton Mamani, Leonel Merino, Ronie Salgado y Geoffrey Hecht por brindarme apoyo y ayudarme con todo lo que concierne a Pharo, sus librerías y sus conocimientos en el ámbito de la realidad virtual y la realidad aumentada.

Por último, agradecer a todas aquellas personas (desarrolladores, autores y/o investigadores) que han aportado su conocimiento al campo de la realidad virtual.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Con el surgimiento de la computación, la tecnología ha logrado un avance significativo. No sólo nos ha servido para resolver problemas o crear nuevas herramientas, sino también para imaginar y/o recrear escenarios fuera de nuestra realidad actual. En este contexto, la tarea del programador es utilizar esos elementos para construir estos mundos, usados generalmente en el campo de la investigación o el entretenimiento. Los avances de hardware han permitido no sólo observar estas simulaciones, sino también ser parte de ellas, estar inmersos en un espacio completamente diferente. Lo que antes era mera ciencia ficción ahora es realidad. Esta simulación de un ambiente del mundo real generado por computadora es lo que comúnmente se conoce como *Realidad Virtual* o *Virtual Reality (VR)*.

La rutina de un programador no deja de ser diferente: para crear estos elementos, debe sentarse frente a su computadora, escribir algo de código, compilarlo, depurarlo, corregirlo y, de vez en cuando, cambiar de vista para poder ver los resultados de su ejecución. Todo este flujo parece algo tedioso, sobretodo cuando la computadora puede mostrar sólo algunas cosas al mismo tiempo. Entonces, ¿Qué pasaría si tanto la programación como la ejecución del programa fuesen observables en el mismo espacio al mismo tiempo? ¿Qué tal si tuviésemos espacio infinito, con todas las herramientas a disposición y que esté bajo nuestro control? ¿Qué tal si pudiésemos ver y manipular nuestro programa a nuestro antojo? Estas preguntas son los que motivan este proyecto. El hecho de contar con todos los elementos anteriores nos lleva a una metodología en particular: la programación en realidad virtual.

El hecho de poder manipular un programa durante su ejecución, concepto conocido como *Live Programming* o *Programación en vivo*, no es algo nuevo. Por ejemplo, los navegadores de internet poseen herramientas de desarrollo para monitorizar el funcionamiento de una página web. Al abrir una consola, el usuario es capaz de ejecutar comandos en javascript y observar en tiempo real cómo es que los elementos de la aplicación web van cambiando. Del mismo modo, ventanas como el inspector de código fuente o el depurador van entregando retroalimentación de manera activa.

La idea es pensar esta misma situación, aprovechándose de las características que puede ofrecer la realidad virtual. Ya no sólo se trata de una pantalla bidimensional, que debe repartir su espacio entre diferentes ventanas y herramientas. Ahora disponemos de un espacio

tridimensional, donde nuestro enfoque visual está centrado únicamente en un espacio teóricamente infinito, donde no hay límites de las visualizaciones de nuestros programas. Es más, esto puede implicar que en un futuro, más de una persona pueda programar en el mismo espacio, sin importar la distancia global, ni lo que esté ocurriendo a nuestro alrededor. Es decir, los beneficios que puede traer un entorno de programación en realidad virtual abarcan no solo aspectos de carácter comunicacional y cognitivo, sino también en términos de eficiencia.

1.1. Situación Actual

Los intentos por llevar un entorno de programación de este estilo a realidad virtual no son nulos[1]. En el pasado, el profesor Alexandre Bergel había trabajado en proyectos de visualización de código usando objetos tridimensionales proyectados a imágenes del mundo real [9], concepto conocido como *Realidad Aumentada* o *Augmented Reality (AR)*. A través de lentes de realidad aumentada de Microsoft, conocidos como **HoloLens**, el usuario es capaz de ver visualizaciones de código en tres dimensiones, representando diferentes propiedades. Por ejemplo, la comparación de tamaños entre diferentes objetos se representa como un conjunto de estructuras cúbicas, similar a cómo se ve un grupo de edificios en una ciudad [2]. Otro uso consiste en el monitorización de la evolución de los objetos en el tiempo [8]. Las diferentes propiedades del objeto son representados según su forma, tamaño, color, etc.

En cuanto a realidad virtual se refiere, el año 2017, Craig Latta crea **Caffeine**, un entorno de desarrollo en tres dimensiones compatible con realidad virtual [6]. El software está desarrollado sobre javascript, y soportan tanto Smalltalk como Pharo como lenguajes de programación. Bajo el lema *"livecode the web!"* ("programa en vivo la red"), el sistema un puente bi-direccional que procesa mensajes de Smalltalk en objetos y promesas de Javascript. La interfaz presentada muestra una ventana similar a la que está presente en el entorno de desarrollo de escritorio en Smalltalk. Se muestran listas de paquetes, sus respectivas clases y los respectivos métodos (**Ver Figura 1.1**).

1.2. Objetivos

Si bien ambos proyectos representan un gran avance en cuanto a programación en VR, no es suficiente como para constituir un ambiente de desarrollo completo. Mientras que el primero se enfoca en el aspecto visual, el segundo prioriza los aspectos relativos a programación propiamente tal. Un entorno en realidad virtual con ambos elementos representaría la experiencia definitiva de programación inmersiva, brindando no sólo las herramientas para generar código, sino también para observarlo en acción.

Este proyecto busca materializar este objetivo, utilizando una lógica similar a Caffeine, pero bajo herramienta más dedicadas al desarrollo de escenarios en realidad virtual. En consecuencia, los principales elementos que componen este proyecto son dos. Por un lado, tenemos un ambiente back-end, compuesto por un servidor de Pharo que recibe y retorna instrucciones como mensajes HTTP. Esto es posible gracias a la librería Zinc HTTP, que

[1]Véase B. Peiris, 'RiftSketch' [En Línea], Disponible en `http://brianpeiris.github.io/RiftSketch/` [Visitado: 28-04-2020]

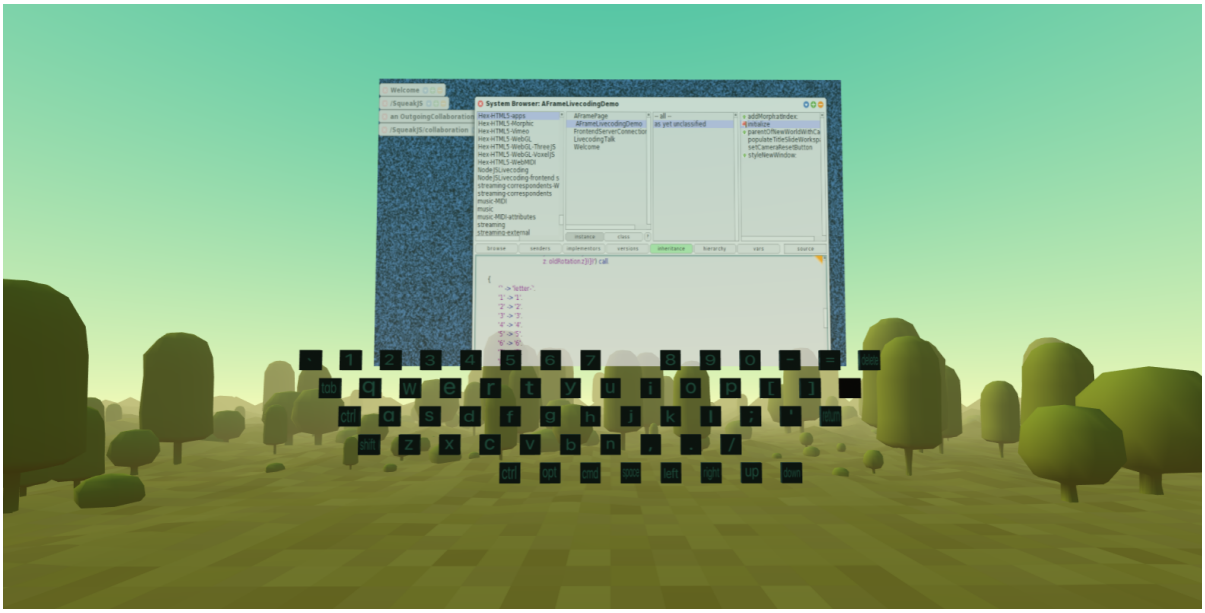

Figura 1.1: Demostración de CaffeineJS

permite especificar la dirección IP a la que se dirigirán los mensajes. Por otro lado, el front-end consiste en un escenario en tres dimensiones generado por el motor gráfico Unity, que contiene todos los plugins necesarios para crear una experiencia en realidad virtual. Entre algunas de sus ventajas está la facilidad y rapidez para crear el ambiente, sumado a su integración con múltiples dispositivos de realidad virtual como, por ejemplo, Oculus o HTC.

En resumen, los principales objetivos de esta memoria son:

- Crear un entorno de desarrollo que se desenvuelva completamente en realidad virtual.
- Generar visualizaciones interactivas en tres dimensiones del código producido.
- Explorar diferentes formas de programación e interacción con elementos de interfaz de usuario dentro de un ambiente de realidad virtual.
- Monitorear la carga cognitiva, evaluada a través de la participación de usuarios que realicen diferentes tareas de programación.

1.3. Pregunta de Investigación e Hipótesis

Dados los antecedentes en cuanto al uso de realidad virtual y realidad aumentada en programación, así como la disponibilidad de tecnologías para lograrlo, nuestra hipótesis es que sí es posible desarrollar un ambiente de desarrollo que esté basado completamente en realidad virtual, y que además permita realizar tareas de programación de una manera realista. Por otro lado, creemos que también es posible traducir el concepto de Live Programming a un entorno de este estilo.

En ese sentido, la pregunta que surgen al respecto es: ¿Cuáles son las limitaciones actuales que enfrentará un desarrollador al momento de programar una aplicación usando un entorno VR? Dado que la realidad virtual es una tecnología aún en crecimiento, es de esperar que

existan diferentes barreras. Es muy probable que tanto la curva de aprendizaje como el tiempo invertido en desarrollar un programa sea mucho mayor que de la forma tradicional. Esto, debido a que la interacción con un entorno VR podría no tener la misma precisión y comodidad que usar una pantalla y un teclado. Se espera que, con el tiempo, estén muchas más herramientas disponibles, que puedan hacer una integración más sencilla de estos elementos con el mundo virtual.

1.4. Metodología de trabajo

El flujo de trabajo a realizar durante la memoria, como se observa en la **Figura 1.2**, se puede resumir en las siguientes etapas:

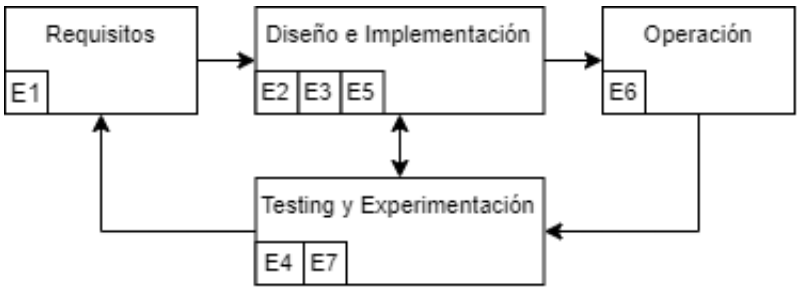

Figura 1.2: Proceso de desarrollo de Software

- **Etapa 1: Definir los requisitos**: Se definen cuáles son los requisitos mínimos que se necesitan para obtener un prototipo funcional.
- **Etapa 2: Diseño de la solución**: Se plantea el diseño que tendrá el entorno de desarrollo en VR que mejor satisfaga tanto los requerimientos antes mencionados como la experiencia de usuario. Por ejemplo, algunos de los aspectos a discutir fueron la arquitectura de software, el framework que se usará para desarrollar en VR, recursos y plugins a integrar, formas en que el usuario podrá escribir código, entre otros.
- **Etapa 3: Implementación**: Se realiza la implementación del software usando las diferentes herramientas de programación a disposición.
- **Etapa 4: Pilotos**: Durante y después del desarrollo, se realizan tests para verificar su funcionamiento y realizar experimentos. Tras esto, se hace una reevaluación de los requisitos y/o se corrigen bugs y funcionalidades.
- **Etapa 5: Diseño de los experimentos**: Una vez que el proyecto alcanza un estado de mínimo viable (esto es, que cumpla con los mínimos requisitos), se hace una planificación de las tareas a realizar y los aspectos a evaluar durante la fase experimental.
- **Etapa 6: Operación y Convocatoria**: Habiendo decidido cómo se realizará el experimento, el proyecto es dado a conocer públicamente y se difunde una encuesta que permita reunir a programadores que tengan acceso a un dispositivo VR para llevar a cabo las pruebas.
- **Etapa 7: Experimentación**: Tras coordinar una fecha y horario con los inscritos en el formulario, se procede a hacer la fase de experimentos controlados, donde la sesión es supervisada por el memorista. Al final del proceso, se obtiene la información de la sesión mediante un archivo log y, finalmente, se le pide al usuario responder una encuesta para medir la usabilidad del proyecto.

1.5. Dispositivo de realidad virtual

El dispositivo a utilizar tanto para el desarrollo de la aplicación como para los experimentos es un HTC Vive Cosmos. Se caracteriza por poseer correa ajustable, auriculares incorporados, 6 cámaras frontales y dos mandos, cada uno con una palanca y botones sensibles al tacto. Entre sus características más notables están su diseño ergonómico resolución de 2880 x 1700 píxeles, y la capacidad de detección de bordes y proximidad de objetos cercanos. Los controles poseen vibración que se activa al interactuar con objetos tridimensionales.

Figura 1.3: Fotografía de HTC Vive Cosmos (Fuente: Amazon)

1.6. Principales desafíos

A pesar de lo prometedor que pueda parecer el proyecto, la realidad virtual en la actualidad sigue sujeta a varias limitaciones. Esto ocurre principalmente en el ámbito de la programación, pues no sólo el VR es una tecnología novedosa, sino que suele enfocarse únicamente en la industria del entretenimiento electrónico, concretamente en los videojuegos. Hacer de la realidad virtual una tecnología mucho más versátil requiere de mayor exploración, y esto implica ciertas barreras de entrada como se detallan a continuación:

1.6.1. Disponibilidad de dispositivos VR y compatibilidad

Dado que los equipos de realidad virtual son bastante costosos, es de esperar que sean pocas las personas que tengan acceso a uno. Consideremos además de que, al igual que el mercado de las consolas, existe más de un fabricante y, en consecuencia, más de un modelo de visores. Esto también implica un mayor uso de un modelo por sobre otro. Por ejemplo, de acuerdo a las estadísticas de usuario de Steam, Oculus es actualmente la marca más popular de dispositivos VR en el mundo, colocándola por sobre dispositivos de otros fabricantes, como HTC VIVE o Valve Index (Ver **Figura 1.4**).

Es por este motivo que se debe asegurar su compatibilidad con la mayor cantidad de dispositivos que sea posible para aumentar las probabilidades de encontrar personas que puedan utilizar el software. En un futuro, esto ayudaría a disminuir las barreras de acceso para nuevos usuarios que quiera usar el software, pues lo esperable es que los dispositivos puedan ser más accesibles a un público mucho más amplio.

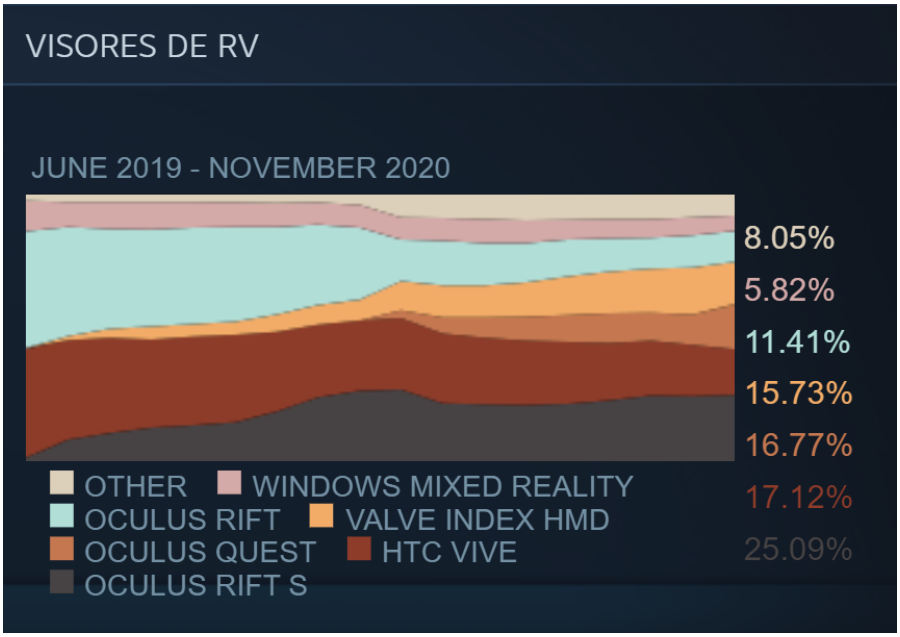

Figura 1.4: Estadísticas de Steam sobre uso de dispositivos VR

1.6.2. Plataformas con soporte para VR

No son muchas las plataformas que son capaces de soportar VR. Si bien existen computadoras personales de alta capacidad y rendimiento, así como visores VR autocontenidos (esto es, que pueden reproducir contenido sin necesidad de un PC), las librerías de desarrollo en VR suelen ser soportadas únicamente en determinadas plataformas.

De acuerdo a *statcounter GlobalStats*, Windows 10 y Android son las plataformas más populares por sobre Mac OS X o sistemas operativos basados en Linux a nivel global [11], como se puede ver en la **Figura 1.5**. Además, de acuerdo a las estadísticas de usuario de Steam, Windows 10 es el sistema operativo más usado en software 3D, incluyendo experiencias en VR [3].

En consecuencia, el uso de este tipo de software suele estar limitado a esas plataformas, dejando fuera a aquellos programadores que utilizan otro sistema operativo, como Mac OS o Linux. Al igual que en el caso de la disponibilidad de dispositivos, reducir las barreras de entrada implica también asegurarse que aquellos usuarios que dispongan de un dispositivo de realidad virtual tenga también la capacidad de hacerlo funcionar correctamente. Para esto, se busca que el proyecto pueda ser ejecutado en la mayor cantidad de plataformas posibles.

1.6.3. Experiencia de usuario: uso de pantalla y teclado

Los programadores están acostumbrados a programar en un escritorio, teniendo una pantalla y un teclado físico. Esta rutina es relativamente sencilla y ágil, pues el programador es capaz de escribir líneas de código a velocidad constante gracias a la motricidad de sus dedos. Por supuesto, también ayuda el hecho de que el teclado está cerca de la pantalla, por lo que nuestra visión puede saltar desde un objeto a otro sin problemas.

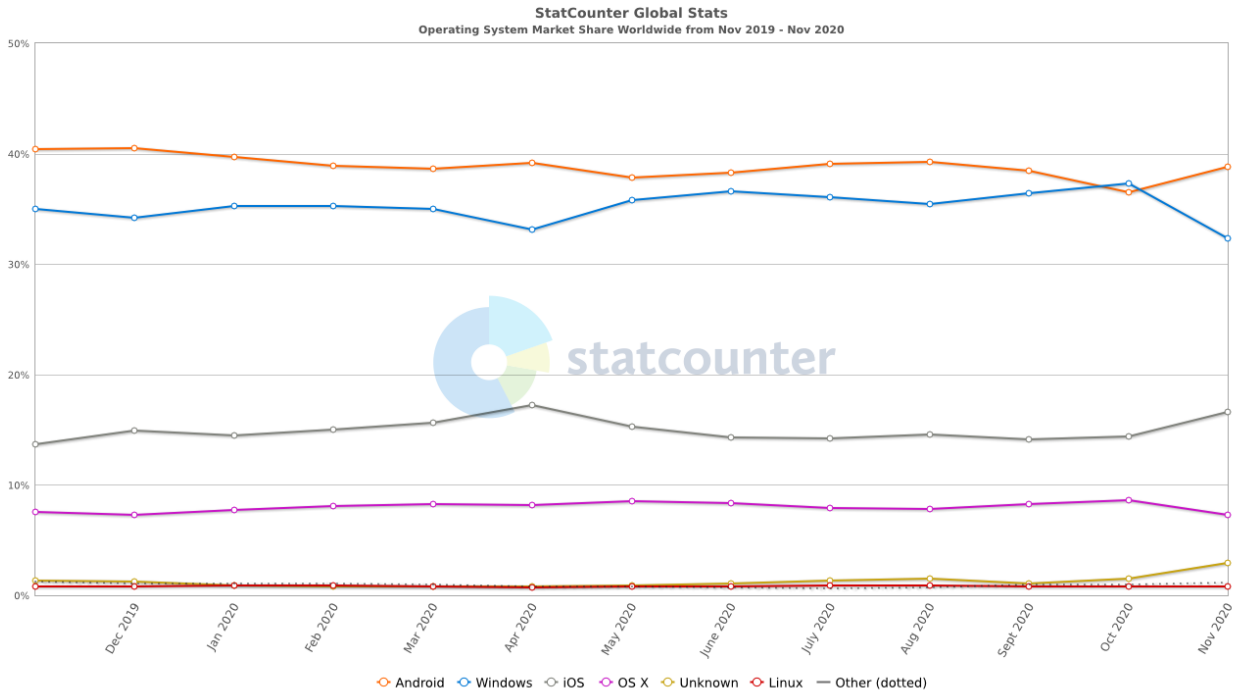

Figura 1.5: Estadísticas globales de uso de Sistema Operativo

En un ambiente de realidad virtual, la situación es muy diferente. Dado que los visores suelen venir acompañados de controles, no disponemos de una representación exacta de nuestro entorno físico, tanto de nuestro escritorio como de nuestras manos. Incluso ver nuestro mundo real a través de las cámaras externas de un visor puede ser difícil debido a la falta de detalle de la imagen. Debido a estas limitaciones, la comodidad y agilidad pueden verse drásticamente reducidas. En consecuencia, se deben idear formas de que el usuario pueda programar de manera fluida, sin necesariamente sacrificar conceptos con los que el programador pueda estar familiarizado, como lo son el uso de ventanas en conjunto con un método de escritura.

1.7. Resultados

Como principales resultados, se tiene que los usuarios han sido capaces de realizar tareas de programación sencillas y de visualización de código. Esto apoyaría la hipótesis de que sí es posible desarrollar un ambiente de programación en realidad virtual. A pesar de esto, se mantienen diversas limitaciones al rededor de esta tecnología, y es necesario tomar medidas para mejorar su usabilidad y reducir la carga cognitiva.

1.8. Estructura de la memoria

Este documento abarcará detalles de la metodología de trabajo través de cinco capítulos principales. Estos capítulos son:

1. **Decisiones de diseño**: Muestra el proceso de diseño de la solución, y las decisiones tomadas en cuanto a arquitectura, requisitos y cómo debe verse el entorno.
2. **Implementación**: Se explica de forma detallada la solución al diseño planteado en términos de programación de software.
3. **Testing**: Habla sobre las diferentes formas de testing empleadas para verificar el funcionamiento del software.
4. **Evaluación**: Explica la metodología experimental, métricas utilizadas y principales resultados.
5. **Discusión**: Se realiza un análisis tanto cuantitativo como cualitativo de los resultados obtenidos en la evaluación.

Capítulo 2

Decisiones de diseño

Como se habló en el capítulo anterior con respecto a los desafíos, el diseño del software no es sencillo. Este debe responder a las habilidades de programación del usuario, a la vez que se cambian algunas cosas para generar en flujo armonioso con los controles que utiliza un equipo de realidad virtual. Por lo tanto, no basta con replicar un IDE convencional dentro de un entorno 3D, sino que debe responder a estas limitaciones.

Lo anterior viene, por supuesto, desde una perspectiva de experiencia de usuario. Sin embargo, eso no excluye lo más importante: que el software sea capaz de generar código. El cómo y qué clase de código se discutirá en las secciones presentadas a continuación, donde el foco del diseño está sobre las herramientas a utilizar, así como la arquitectura y funcionalidades que debe cumplir.

2.1. Herramientas a utilizar

2.1.1. Lenguaje de programación Pharo

Una de las herramientas que más énfasis ha tenido durante este trabajo es el lenguaje que el usuario usará para programar. Pharo es un lenguaje de programación orientado a objetos, dinámicamente tipado y basado en Smalltalk. A diferencia de otros lenguajes, Pharo no es uno que almacene sus objetos en diferentes archivos, sino que toda la información es centralizada y administrada en un sólo archivo llamado *imagen*.

Su entorno de desarrollo provee de diferentes herramientas, entre ellas, un navegador o *Browser* que organizar los diferentes paquetes, clases y métodos. Uno de sus aspectos de suma importancia para este proyecto es que permite programar bajo un régimen de Live Programming. Al igual que una consola de javascript en un navegador web, Pharo posee una herramienta llamada Playground, que permite ejecutar líneas de código de forma inmediata, así como alterar los elementos de un programa en ejecución.

La sintáxis de Pharo es bastante minimalista, lo que permite que programas de alta complejidad puedan escribirse en tan sólo unas pocas líneas de código. Esto es bastante beneficioso para un ambiente de realidad virtual, pues la reducción en velocidad de escritura es

compensada con la poca cantidad de caracteres que se necesita escribir.

En cuanto a experiencia de usuario se refiere, el entorno de desarrollo permite el uso de múltiples ventanas a través del espacio que este provee, similar a como un editor de código permite observar más de un archivo, a la vez que se muestran otras herramientas como la consola de ejecución o el explorador de archivos. La diferencia radica en que cada herramienta es una ventana, y el usuario decidirá qué lugar en el espacio ocupar. Dado que un programador en Pharo está acostumbrado a este espacio, le resultará más provechoso un espacio aún mayor, disponible en todas direcciones.

2.1.2. Motor gráfico Unity3D

Gracias al avance que ha presentado la realidad virtual en la industria del entretenimiento (esto es, videojuegos y otros medios audiovisuales), existen muchos medios disponibles en la red para aquellos que deseen ser desarrolladores de este tipo de ambiente. El 29 de Julio de 2019, *Khronos Group*[1] lanza **OpenXR**, un estándar de código abierto que provee acceso de alto rendimiento a plataformas y dispositivos de realidad virtual y realidad aumentada. Este es distribuido en general como un kit de desarrollo dentro de productos de software orientados a desarrollar en VR o AR. Numerosas compañías soportan este proyecto, entre ellas Google, Microsoft, HTC y Facebook [5].

En el caso particular de software orientado al entretenimiento audiovisual, las compañías más reconocidas son Epic Games (Unreal Engine) y Unity3D. Este último es caracterizado por ser uno de los motores gráficos más populares del mercado hasta la fecha, debido a su simplicidad, soporte, documentación, compatibilidad y curva de aprendizaje. Además, diversos fabricantes de dispositivos VR, como Oculus y HTC, han aportado con sus propios plugins y herramientas para que la comunidad pueda desarrollar contenido dedicado a sus dispositivos.

El principal lenguaje de programación sobre el que funciona Unity3D es C# mediante el framework .NET (DotNet), cuyo símil a lenguajes como Java permite una programación orientada a objetos. Todos estos factores, sumado al conocimiento previo que se tenía con respecto al motor gráfico, fueron determinantes al momento de decidir qué herramientas utilizar.

Mediante los diferentes repositorios, tanto desde la comunidad de desarrolladores como de las grandes empresas de tecnología, se han recopilado las siguientes herramientas que serán utilizadas para el desarrollo del software:

- **Unity3D 2020.X**: La versión más reciente del motor gráfico, y que permite la integración con dispositivos VR.
- **OpenVR y SteamVR Plugin por Valve**: Librerías esenciales que son las que permiten la conexión del hardware con el motor gráfico. Además, asegura su funcionamiento en dispositivos HTC y Valve Index.
- **Oculus Integration y Oculus XR Plugin por Oculus, Facebook**: Permiten que el software funcione con dispositivos Oculus.

[1] `https://www.khronos.org`

- **HTC VIVE Plugin para Unity3D, por HTC**: Herramienta creada por HTC, en la que vienen objetos prefabricados, con toda la implementación de código incluída y lista para usar. Soporta una gran variedad de dispositivos.
- **Dispositivos VR VIVE Cosmos y Oculus Quest 2**: Visores de realidad virtual con los que se probará el software a desarrollar.

2.2. Arquitectura

El proyecto debe considerar dos cosas. En primer lugar, el tiempo disponible para realizar el proyecto necesitaba de una solución ágil para determinar cómo es que se interpretará código en Pharo. En segundo lugar, se debe asegurar que este elemento funcione correctamente en las diferentes plataformas. Por ejemplo, unos lentes Oculus no podrán interpretar instrucciones en Pharo a menos que se interfiera el sistema, lo que implica un poco más de trabajo. En este contexto, un acercamiento posible es modularizar ambos factores. Es decir, relegar las renderizaciones de elementos gráficos a los dispositivos, y encargar las tareas de programación a una máquina que establezca una conexión con los mismos.

En consecuencia, se optó por una arquitectura sencilla de cliente-servidor, en que el cliente corresponda al usuario de un dispositivo de realidad virtual, mientras que el servidor es una máquina separada que en sí misma tenga todas las capacidades para procesar información en Pharo. Cada vez que el usuario realice una acción que requiera leer o escribir código, se activará un evento asincrónico que enviará peticiones al servidor, quien responde con la información actualizada. De esta forma, la carga de procesamiento es distribuida, y los datos entre ambos estarán siempre en sincronía (**Ver Figura 2.1**).

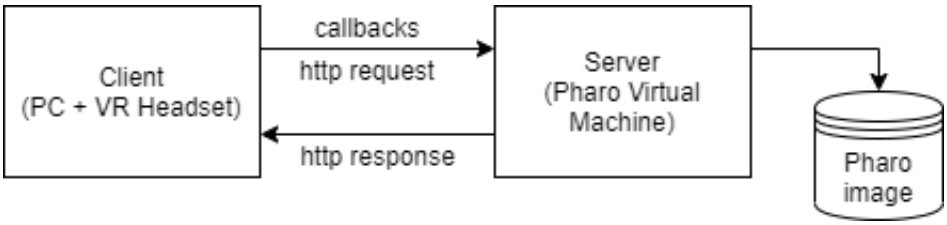

Figura 2.1: Arquitectura de software

2.3. Requisitos

Dada la complejidad que presenta un entorno de desarrollo en Pharo, se deben definir aquellas funcionalidades mínimas que debe poseer un prototipo inicial del proyecto. La meta principal es alcanzar un entorno inmersivo de Live Programming. En consecuencia, se definieron los siguientes requisitos:

- El usuario debe estar dentro de un ambiente de realidad virtual. Es decir, debe existir una inmersión total, en la que la única pantalla que exista sea aquella que se ve a través de los visores, y que el único entorno de trabajo sea aquel generado por el software. De esta manera, se cumple el objetivo de tener un espacio ilimitado en que quepan todas las herramientas de programación.

- El usuario debe poder interactuar con una interfaz de programación en Pharo, lo que implica que debe proveer herramientas que permitan crear programas mediante programación orientada a objetos. Estas herramientas suelen ser el Playground, Browser e Inspector. Otras utilidades abarcan un sistema de log como lo es el Transcript y el Debugger.
- El usuario debe ser capaz de escribir y ejecutar código en Pharo. En consecuencia, se debe proveer al usuario de formas para escribir y ejecutar código dentro del entorno. Esto puede lograrse mediante formas convencionales, como lo es el uso de un teclado, o explorar otras herramientas que se adecuen al carácter inmersivo del software.
- El usuario debe poder generar visualizaciones gráficas de código e interactuar con ellas. Tratándose de un espacio en tres dimensiones, se debe aprovechar la oportunidad para generar visualizaciones no sólo ocupen toda el área disponible, sino que también puedan aprovechar el uso de cuerpos tridimensionales.

2.4. Diseño del entorno

El bosquejo inicial del software debe ser tal que pueda replicar un entorno de escritorio, sin que se vea comprometida la usabilidad debido a las limitaciones que presentan los controles VR. Por esta razón, el diseño del prototipo busca presentan un entorno de desarrollo más simple que presente los siguientes elementos:

- Un ***Code Browser*** que tenga listas para agrupar paquetes, clases y métodos, y un campo de texto donde se pueda escribir el código fuente.
- Un ***Playground*** hecho a base de un campo de texto donde se puedan escribir y ejecutar líneas de código.
- Un ***Inspector*** con un par de listas que muestre las variables de un objeto y sus respectivos valores, más un campo de texto que pueda mostrar código fuente.
- Un ***Log*** que pueda imprimir mensajes de ejecución y/o de error.
- Un método de entrada para los campos de texto. En un principio, se planteó como solución usar simplemente el teclado físico, aunque en el futuro se explorarían otras alternativas.

La **Figura 2.2** muestra los primeros mockups planteados para el software. Con el paso del tiempo, se realizarían algunos cambios y añadirían otros elementos para enriquecer las herramientas y mejorar la experiencia de programación.

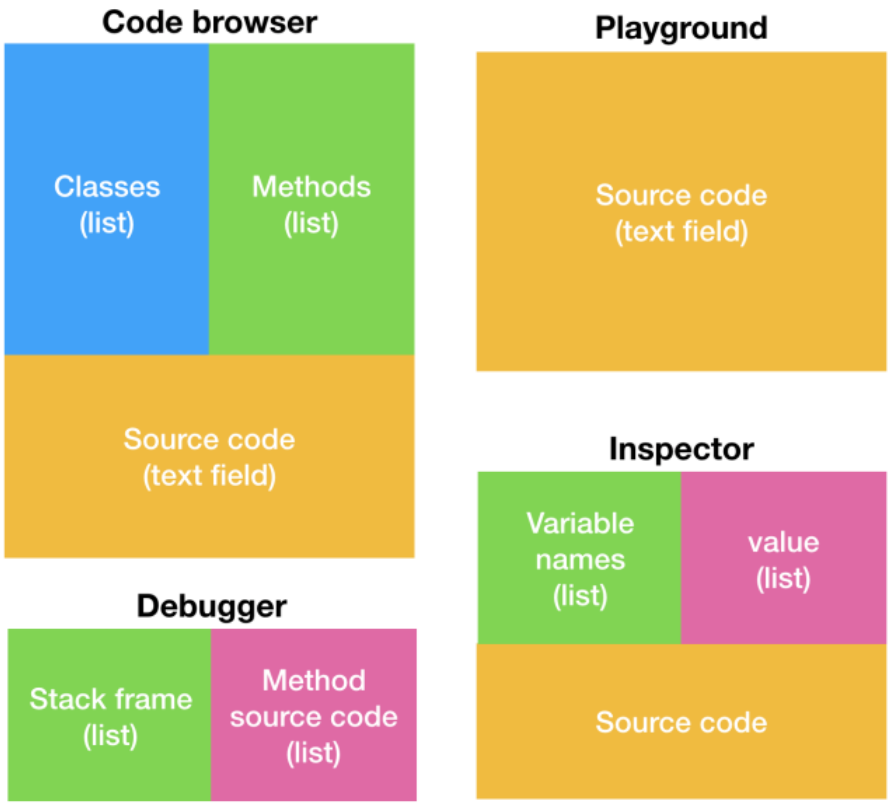

Figura 2.2: Mockups iniciales del proyecto

Capítulo 3

Implementación

Habiendo definido la arquitectura, podemos dar los primeros pasos hacia la implementación. Como se mencionó en el capítulo anterior, el software se dividirá en dos módulos que en conjunto forman un software de programación en realidad virtual. La lógica detrás del código generado será delegada al back-end, donde una imagen de Pharo se encargará de recibir aquellas peticiones del usuario que contengan líneas de código. El front-end, implementado en Unity3D, se encargará de atrapar las acciones del usuario, darles el formato correcto, y enviarlas como peticiones al servidor.

3.1. Pharo como Back-End

El primer desafío fue idear una forma de ejecutar instrucciones de Pharo dentro del ambiente en VR. Dado el tiempo disponible para el proyecto, representaba un trabajo extra tener que implementar un compilador dentro del entorno. La solución debía ser mucho más pronta y sencilla, sin sacrificar necesariamente la calidad del software. Se barajó la posibilidad de utilizar un proceso en segundo plano que enviara mensajes a una imagen de Pharo mediante línea de comandos. Esto se lograría mediante la instrucción:

```
./pharo Pharo.image eval "Some code"
```

Este método, sin embargo, tiene un tiempo de respuesta más extenso, pues Pharo debe ejecutarse en cada llamada. Un acercamiento más eficiente requeriría que una imagen esté siempre activa, recibiendo instrucciones constantemente. En este contexto, la librería ZincHTTP, creada por Sven Van Caekenberghe, permite ejecutar un servidor de Pharo que recibe y entrega mensajes HTTP. Las instrucciones son enviadas al servidor como petición POST. Por ejemplo, supongamos que queremos ejecutar "42 factorial". El código se envía como string al servicio *repl*, encargado de procesarlo como un comando de Pharo. El resultado es retornado como un nuevo string de datos.

```
ZnClient new
    url: 'http://localhost:1701/repl';
    contents: '42 factorial';
    post.
```

Una gran ventaja de esto es la inmediatez con que se pueden ejecutar líneas de código, pues toda la carga de procesamiento es asumida por el servidor. Sin embargo, esto implica que se necesitará de una conexión estable a internet, pues la velocidad a la que se envían los mensajes estará sujeta a la latencia que presente la red.

El servidor también debe contar con las librerías necesarias para poder visualizar código. La principal librería para esto es Roassal, que abarca una variedad de representaciones gráficas, además de exportadores para almacenarlas en archivos. Uno de ellos es **A-Frame**, un framework web de código abierto usado para crear entornos de realidad virtual. Gracias a esto, es posible traducir sus componentes a objetos en tres dimensiones.

Finalmente, el flujo de trabajo se reduce a una arquitectura de cliente-servidor, en la que el usuario, desde el entorno en realidad virtual, se comunica con un servidor que procesa instrucciones en Pharo, y devuelve el resultado en forma de texto. Luego, lo único que queda por resolver es el cómo se envían los mensajes. La lógica detrás de esta conexión es la que se verá en el front-end.

3.2. Unity3D como Front-End

El siguiente reto es resolver cómo ejecutar código escrito en Pharo desde una aplicación escrita en C#. Anteriormente se mencionó la posibilidad de usar una librería que permitía hacer llamadas de Pharo mediante mensajes HTTP. El framework DotNet posee una clase especial para estos fines llamada **HttpClient**. Con ella, podemos establecer conexión con una determinada dirección IP y enviar mensajes GET y POST, como se puede ver en el siguiente código de ejemplo.

```
using System.Net.Http;

public class PharoRequests {
    private static readonly HttpClient client = new HttpClient();

    async void PharoCall() {
        var content = new StringContent("42 factorial", Encoding.UTF8);
        var response = await client.PostAsync("http://localhost:1701/repl",
                content);
        var responseString = await response.Content.ReadAsStringAsync();
    }
}
```

Bastaría entonces enviar un string que contenga código en Pharo y esperar una respuesta. Sin embargo, esto no es siempre así. Las excepciones y notificaciones son desplegadas en el servidor, por lo que, ante una eventual falla, la interfaz se quedará esperando indefinidamente por la respuesta hasta que se acabe el tiempo de espera. Se necesita entonces una forma de que responda no sólo con el resultado de la ejecución, sino también con los mensajes de errores. Esto se soluciona con una directiva Try and Catch:

```
public static async Task<string> Execute(string code)
```

```
{
    var request = await client.PostAsync
    (
        IP, new StringContent
        (
            "[" + code + "]\n" +
                "\ton: Exception\n" +
                "\tdo: [:e | e traceCr].",
            Encoding.UTF8
        )
    );
    return await request.Content.ReadAsStringAsync();
}
```

Lo siguiente es crear una interfaz en la que el usuario pueda ingresar estos inputs. Utilizando los elementos de interfaz gráfica incorporados en Unity3D, se recrean las principales ventanas del entorno, cuyos diseños y acciones son detallados a continuación:

- **Browser**: Muestra el listado de paquetes disponibles (**Ver Figura 3.1**). Seleccionar uno de ellos despliega las clases que contiene y seleccionar una clase mostrará su código fuente y sus métodos. El código aparecerá en un editor de texto, que es donde se definen todos los elementos anteriores. Los métodos se distribuyen en dos listados:
  - **Instance Side**: Son accesibles desde una instancia de una determinada clase.
  - **Class Side**: Siguen una lógica similar a las variables y métodos estáticos.

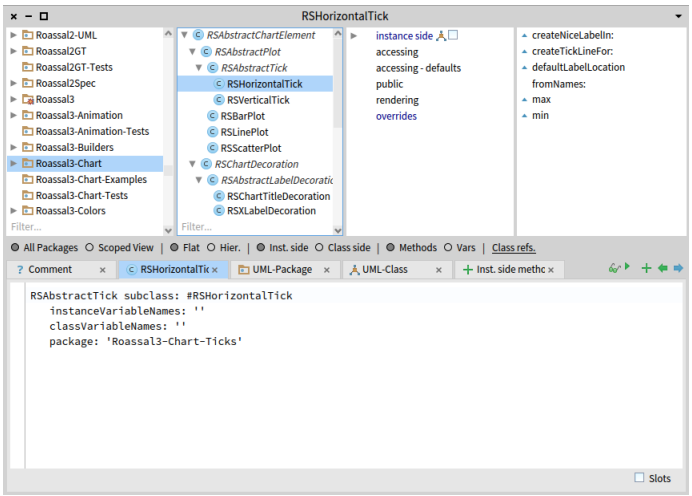

Figura 3.1: Captura de un Browser dentro del IDE Pharo Launcher

- **Playground**: Permite la ejecución de código de forma inmediata.También permite inspeccionar la estructura de un determinado objeto en el código (**Ver Figura 3.2**).
- **Inspector**: Muestra los elementos que componen al objeto que se está inspeccionando, cómo su tipo, valor intrínseco, variables, entre otros (**Ver Figura 3.3**).
- **Transcript**: Ventana que permite imprimir strings (por ejemplo, a partir de la ejecución de código) mediante la clase Transcript (**Ver Figura 3.4**).

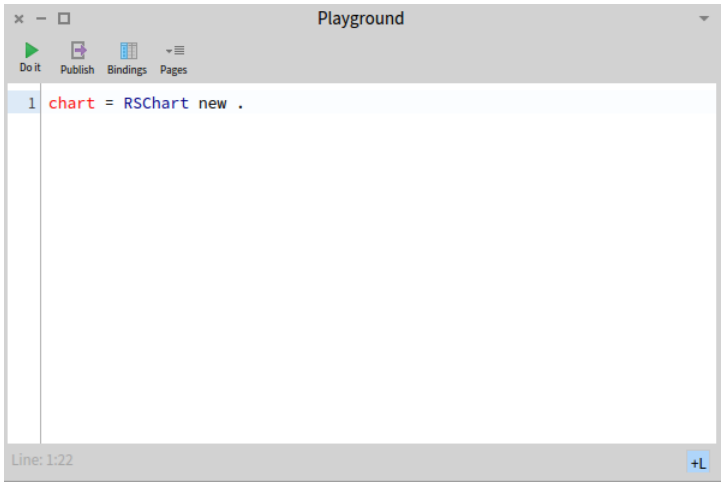

Figura 3.2: Captura de un Playground dentro del IDE Pharo Launcher

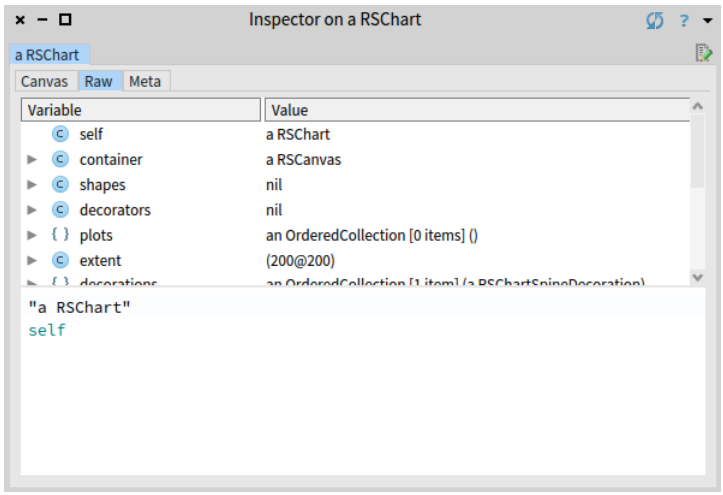

Figura 3.3: Captura de un Inspector dentro del IDE Pharo Launcher

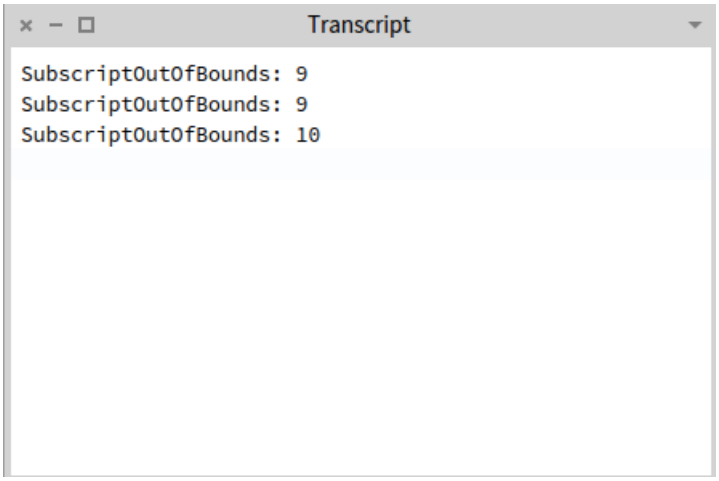

Figura 3.4: Captura de un Transcript dentro del IDE Pharo Launcher

La idea es poder representar estas herramientas en un entorno tridimensional. El motor Unity3D posee elementos de interfaz gráfica de usuario que fácilmente pueden adaptarse a lo que se parece a un IDE de Pharo. Así, por ejemplo, objetos como el Browser y el Playground pueden recrearse mediante ventanas desplazables, campos de texto, botones, casillas de verificación, entre otros. Dado que estos elementos se usan con frecuencia, sin alterar su composición, es posible almacenarlos como recursos prefabricados o *Prefabs*, que son invocados para generar una nueva copia cada vez que se necesiten (**Ver Figura 3.5**).

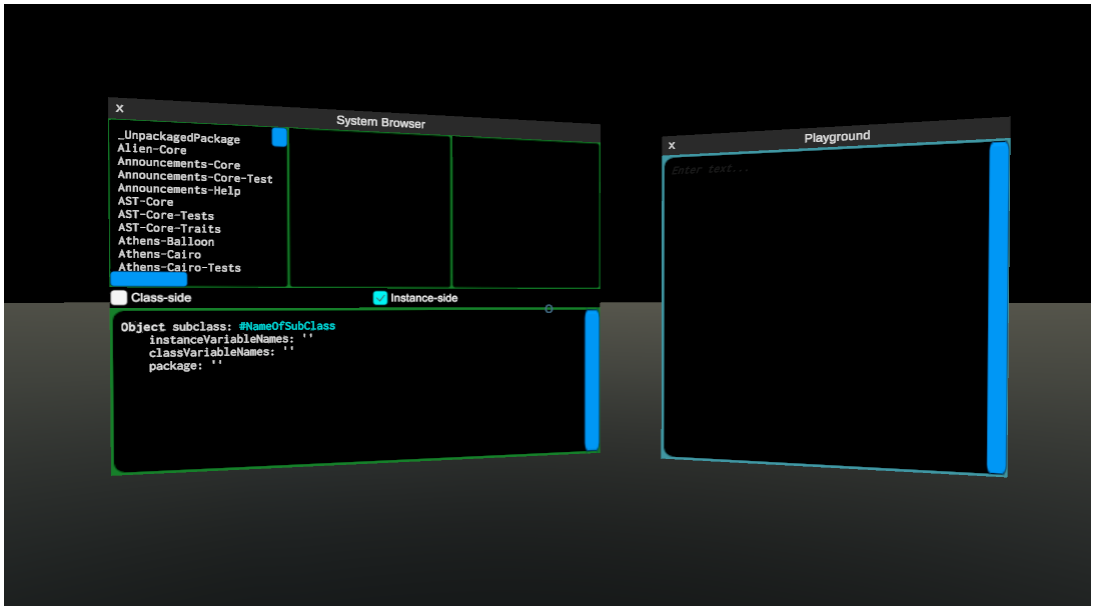

Figura 3.5: Ejemplos de Browser y Playground dentro del ambiente PharoVRIDE

Los elementos dinámicos, como paquetes, clases y métodos, son generados bajo un régimen *on-demand*, es decir, construidos a partir de consultas asincrónicas a la base de datos del servidor, lo que es logrado mediante el siguiente código:

```
// Obtener lista con los nombres de los paquetes
await Task.Run(async () => {
    string code = "RPackageOrganizer packageOrganizer packageNames .";
    string res = await Pharo.Execute(code);
    res = Regex.Replace(res, @"a SortedCollection\(#|#\(#|'|\)|\n", "");
    contents = res.Split(new string[] { " #" }, StringSplitOptions.None);
});

// Obtener clases a partir de un paquete
await Task.Run(async () => {
    string code = "(RPackageOrganizer packageOrganizer packageNamed: '"
        + key + "') classes asString .";
    string res = await Pharo.Execute(code);
    res = Regex.Replace(res, @"(a Set\()|\)|'|#|\n", "");
    contents = res.Split(' ');
});
```

```
// Obtener metodos a partir de una clase
await Task.Run(async () => {
    string code = theBrowser.classSideToggle.isOn ?
        "(" + key + " class) methodDict keys asString ." :
        key + " methodDict keys asString .";
    string res = await Pharo.Execute(code);
    contents = Regex.Replace(res, @"'|\(|\)|#|\n", "").Split(' ');
});
```

Puesto que las aplicaciones en 3D se ejecutan cuadro por cuadro, esto evita que el programa se congele esperando por la información, a la vez que ayuda a mejorar el rendimiento. Una representación gráfica de esta rutina de ejecución puede observarse en la **Figura 3.6**.

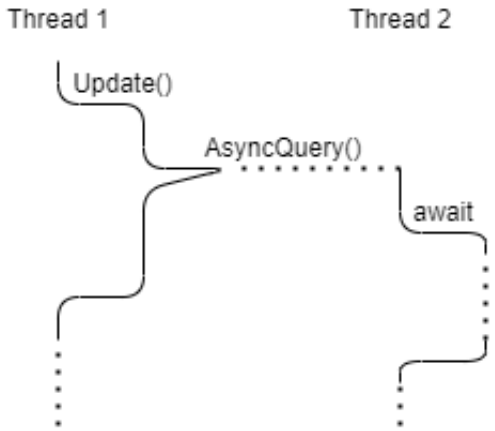

Figura 3.6: Diagrama de threads durante consulta asíncrona

Lo siguiente es lograr que el usuario pueda efectivamente escribir código e interactuar con las interfaces. Tanto las librerías de SteamVR como Oculus proveen recursos para usar de forma inmediata. Por simplicidad, se hizo uso del recurso Vive Input Utility provisto por HTC, que es compatible con una amplia gama de dispositivos, y que tiene los controles configurados de antemano. Así, por ejemplo, es posible desplazarse en el espacio, usar punteros o presionar botones.

No obstante, el principal desafío es hacer que el usuario sea capaz de escribir código. La mayoría de los dispositivos VR incluyen por defecto una vista del mundo real. De esta forma, el usuario puede cambiar desde la vista virtual a la vista real para utilizar su teclado y escribir código. De acuerdo a las pruebas, que serán mencionadas más adelante, esto resulta ser algo incómodo, pues se debe estar constantemente cambiando de vista, y las teclas no son del todo visibles. Se intentó obtener la imagen de la cámara frontal para utilizarla en el mundo virtual. Sin embargo, debido a recientes actualizaciones de OpenXR, no ha sido posible utilizar este método.

Una solución a este problema fue incorporar un teclado virtual, provisto por su autor Jonathan Ravasz[1]. El teclado en cuestión está compuesto por teclas en 3D que reaccionan ante colisiones, emulando el comportamiento de un teclado real (**Ver Figura 3.7**).

[1] `https://github.com/rjth/Punchkeyboard`

Figura 3.7: Teclados virtuales en conjunto con las ventanas

La última funcionalidad a implementar es la visualización de código a partir de A-Frames, que consiste en un framework web para crear entornos de realidad virtual mediante documentos HTML. Por ejemplo, una primitiva en A-Frame posee la siguiente estructura:

```
<a-entity
    position="0.03 0.47 0"
    geometry=
        "primitive: box;
        width: 0.06 ;
        height: 0.06;
        depth: 0.06;"
    material="
        color: #E0ECF4;
        roughness: 1.0;
        metalness: 0.2;">
</a-entity>
```

Dado que no puede abrirse directamente un A-Frame dentro de Unity3D, esto puede lograrse obteniendo su información desde el servidor de Pharo mediante los exportadores de Roassal3, tal como puede verse en el ejemplo de la **Figura 3.8**. El A-Frame es convertido a código HTML y enviado como string al software a través de una respuesta HTTP. A partir de esta información, podemos extraer toda información que sea relevante como *position*, *width*, *height*, *depth*, *color*, *roughness* y *metalness*. Basta entonces con generar la primitiva correspondiente al objeto en 3D y asignarle sus respectivas propiedades.

El método usado se basa en expresiones regulares, identificando las diferentes primitivas de acuerdo al patrón con que se definen. El código que genera el objeto en 3D es como sigue:

```
string tag =
    Regex.Match(aframe, "<a-entity(.*)primitive: box(.*)>").Value;

GameObject ob = GameObject.CreatePrimitive(PrimitiveType.Cube);
Match posMatch = Regex.Match(tag, @"position=""([0-9-.\s]+)""");
Match rotMatch = Regex.Match(tag, @"rotation=""([0-9-.\s]+)""");
Match widthMatch = Regex.Match(tag, @"width: ([0-9.]+) ;");
Match heightMatch = Regex.Match(tag, @"height: ([0-9.]+);");
Match depthMatch = Regex.Match(tag, @"depth: ([0-9.]+);");

if (posMatch.Success)
{
    float[] coords = Array.ConvertAll(
        posMatch.Groups[1].Value.Split(' '),
        i => float.Parse(i, CultureInfo.InvariantCulture));

    ob.transform.localPosition =
        new Vector3(coords[0], coords[1], coords[2]) - pivot;
}

if (rotMatch.Success)
{
    float[] coords = Array.ConvertAll(
        rotMatch.Groups[1].Value.Split(' '),
        i => float.Parse(i, CultureInfo.InvariantCulture));

    ob.transform.localRotation =
        Quaternion.Euler(coords[0], coords[1], coords[2]);
}

if (widthMatch.Success && heightMatch.Success && depthMatch.Success)
{
    Vector3 scale = new Vector3(
        float.Parse(
            widthMatch.Groups[1].Value, CultureInfo.InvariantCulture),
        float.Parse(
            heightMatch.Groups[1].Value, CultureInfo.InvariantCulture),
        float.Parse(
            depthMatch.Groups[1].Value, CultureInfo.InvariantCulture));

    ob.transform.localScale = scale;
}
```

El resultado es un conjunto de primitivas de Unity3D, ordenados de forma similar a su contraparte en A-Frame (Ver **Figura 3.9**). Además de sus características elementales, como color, tamaño o posición, es posible añadirle nuevas propiedades, como el poder ser tomados por el usuario, o desplegar información al ser seleccionados.

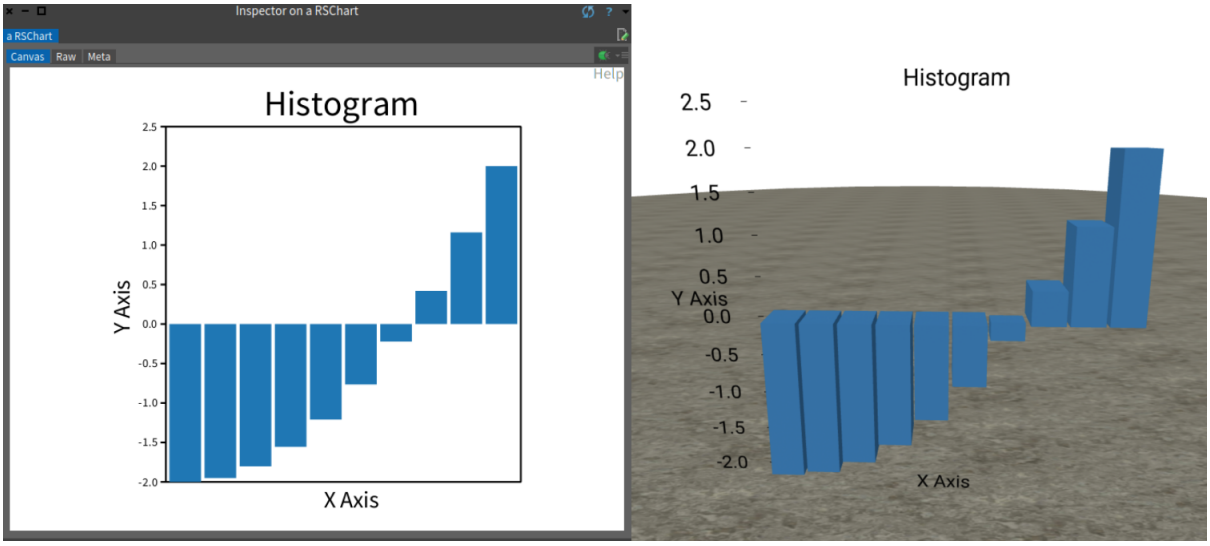

Figura 3.8: Comparación entre una visualización 2D de Roassal con su versión exportada en A-Frame

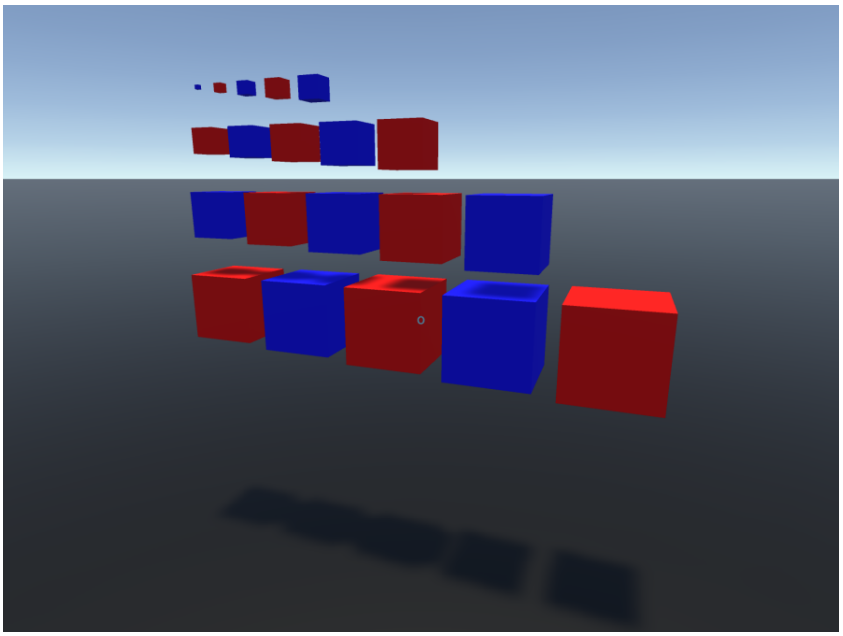

Figura 3.9: Ejemplo de A-Frame exportado a Unity3D

Otros detalles han sido añadidos para mejorar la usabilidad, tales como:

- Posibilidad de arrastrar las ventanas y los cuerpos sólidos de las visualizaciones.
- Botones para alterar la dimensión y escala de las ventanas y visualizaciones.
- Interfaces sencillas para recibir entradas de texto y para ejecutar comandos especiales de Pharo.

Capítulo 4

Testing

Uno de los aspectos más complejos al momento de desarrollar un software en realidad virtual es el testing, pues no sólo se debe realizar sobre el código fuente del programa, sino también en la ejecución del mismo espacio tridimensional. El primer caso es un tanto más sencillo, pues consiste simplemente en utilizar herramientas de testing y depuración que ya son conocidas, como tests unitarios y logging. El segundo, sin embargo, es mucho más complicado, pues dependerá del estado del hardware, esto es, la computadora y el visor de realidad virtual. En este caso, la solución es realizar pruebas con usuarios mediante casos de uso, o bien simular un flujo de trabajo dentro del motor gráfico.

Durante el desarrollo se han aplicado diversos tipos de testing. Mientras que algunas dependen del uso de frameworks y otras extensiones de software, otras han requerido de la participación activa de usuarios para poder encontrar bugs durante sesiones de prueba. El detalle de estas se encuentra a continuación.

4.1. Unit Testing

Cuando queremos comprobar que un bloque de código funciona correctamente, ya sea que retorne el valor correcto o que se haya inicializado correctamente, hablamos de un test unitario o **Unit Testing** [10]. En el caso del proyecto, es posible realizar test unitarios al código del programa sobre el que se construye el front-end.

Afortunadamente, el motor gráfico Unity3D posee un framework especializado para realizar test unitarios. La librería utilizada se llama *"NUnit"*, y permite crear clases que contengan métodos definidos bajo la anotación *Test*. Una vez hecho esto, el motor reconoce estas clases como “clases de test”, y permite que sus métodos se ejecuten de forma secuencial, vigilando que se hayan cumplido todos los *asserts*.

Esto es bastante útil, no sólo sobre el código en sí mismo, sino sobre el software en ejecución. El framework NUnit posee también una rutina de ejecución en "Modo de juego", nombre que refiere a la renderización en tiempo real de la aplicación 3D propiamente tal. En este contexto, es posible simular ciertas acciones del usuario y poner a prueba elementos del espacio sin necesidad de un usuario real. Aunque estas pruebas se limitan a propiedades de

los objetos como posición o dirección, también sirve para evaluar ciertos tipos de eventos e interacciones.

4.2. Smoke Testing

Cuando por cada versión del software hacemos tests para asegurarnos de que es posible volver a realizar tests sobre una versión más actualizada, hablamos de **Smoke Testing** [10]. Un uso común de este tipo de test es ver si un módulo funciona correctamente, para ver si el siguiente módulo también lo hace.

En el caso de este proyecto, es posible aplicar smoke testing sobre aquellos objetos que estén sujetos a jerarquías de clase. Por ejemplo, la **Figura 4.1** muestra un diagrama UML de las diferentes ventanas del entorno virtual (Playground, Transcript, Browser e Inspector). Estas ventanas comparten una clase padre, pudiendo así heredar métodos en común. Los métodos más usados son, por ejemplo, el arrastre y cierre de ventanas, o los callbacks de interacción con elementos UI. Una vez que se ha comprobado que cada objeto muestra un comportamiento básico de ventana, podemos implementar y, por tanto, evaluar funcionalidades más específicas, como definir clases o ejecutar líneas de código.

Si bien smoke testing funciona bien para objetos, no es el único elemento en que puede aplicarse. El diseño de la arquitectura de software también ha seguido una suerte de smoke testing. Por ejemplo, antes de crear las ventanas de interacción, fue necesario comprobar que podían enviarse instrucciones de Pharo desde el código fuente en C#. A su vez, antes de implementar las llamadas desde el front-end, fue necesario comprobar que el servidor estuviera correctamente configurado y que estuviera respondiendo peticiones dentro de la red local.

4.3. Alpha Testing y Beta Testing

Como se mencionó anteriormente, no todos los casos de uso admiten smoke tests o unit tests. Se necesita también probar el software en funcionamiento, lo que se logra mediante el uso de dispositivos de realidad virtual. Durante el desarrollo se han ido llevando a cabo pilotajes con un grupo reducido de gente, generalmente con aquellos que son parte del equipo supervisor del proyecto, para poder determinar que se cumplan los requisitos funcionales mínimos y detectar errores.

En general, las pruebas se realizaron usando una computadora portátil y equipos de realidad virtual HTC Vive Cosmos, Oculus Quest 2 y Valve Index, esto debido a que uno de los principales desafíos es abarcar la mayor cantidad de dispositivos posible. Algunas de las tareas que han sido solicitadas durante los tests son interacción con elementos de interfaz de usuario, como abrir, cerrar y desplazar ventanas, escribir código en los campos de texto, ejecutar código Pharo y comprobar sus resultados, entre otros. Estas pruebas fueron realizadas cada vez que se actualizaba la versión del software. Estas instancias estaban marcadas por la incorporación de una nueva funcionalidad.

A raíz de la crisis sanitaria ocurrida el año 2020, fue imposible realizar testing de forma

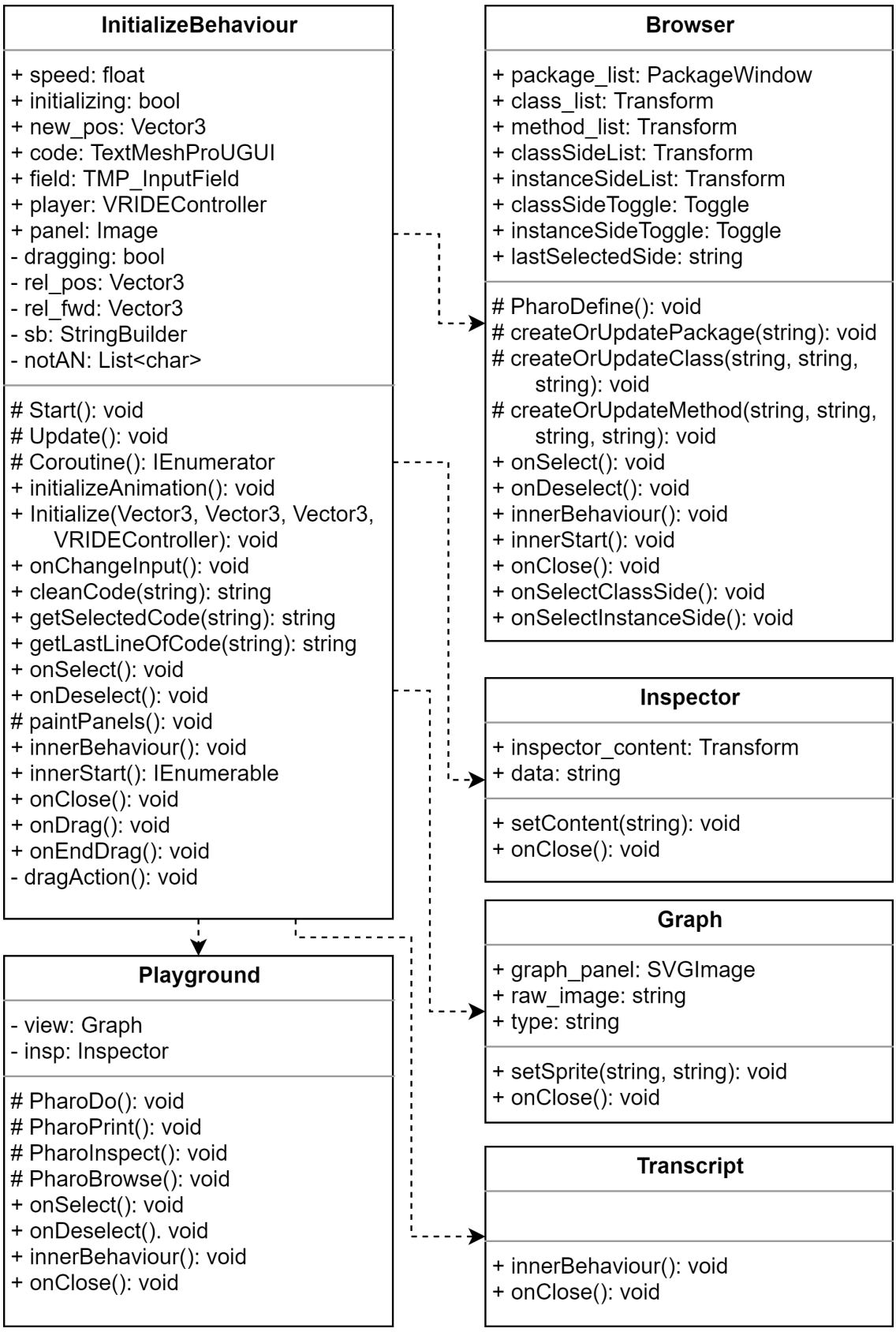

Figura 4.1: Diagrama de dependencia de clases

presencial, por lo que la totalidad de las pruebas ha sido realizada de manera remota. Esto añade otros obstáculos, como la dificultad para reproducir errores o los posibles problemas de comunicación. En la **Figura 4.2** se puede apreciar una versión temprana del software durante un periodo de prueba.

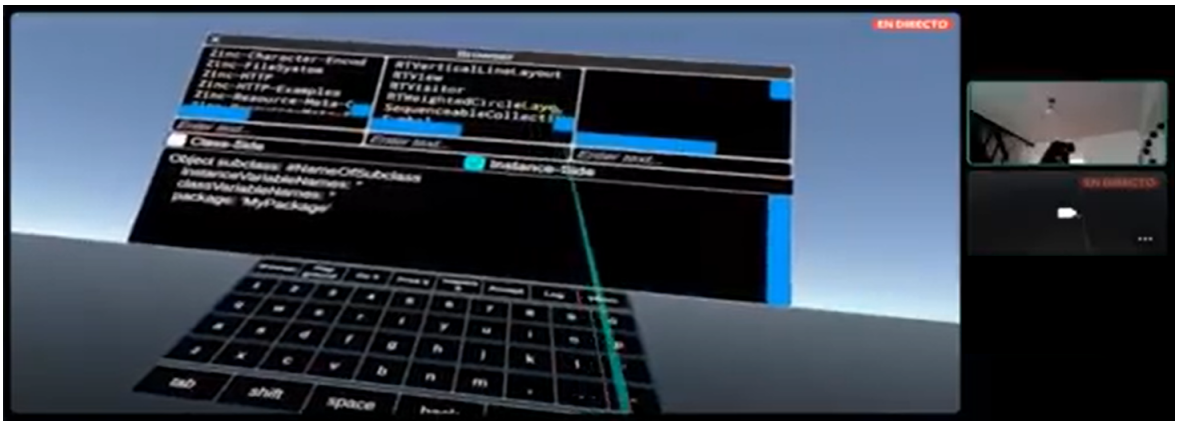

Figura 4.2: Captura de test de una versión temprana del software

Capítulo 5

Evaluación

Una vez finalizado un prototipo que sea viable para la generación de código, el proyecto avanza a la fase de experimentación. Dicho procedimiento involucra una serie de actividades, las que se resumen en una convocatoria abierta para atraer usuarios, observación de los usuarios mientras realizan tareas de programación y análisis de las métricas obtenidas a partir de las sesiones. Los detalles del procedimiento se mencionarán más adelante.

Esta fase es de suma importancia para el proyecto, pues es lo que nos permite ver si se valida o no nuestra hipótesis. Gracias a los experimentos, será posible observar cómo es que un programador se desenvuelve en un ambiente de realidad virtual, cuál es el resultado final de su trabajo, y qué impresiones logra tener en un ambiente de este tipo.

5.1. Método Experimental

El primer paso fue a personas para realizar pruebas y experimentos controlados. Debido a la pandemia de COVID-19, esta tuvo que realizarse mediante un formulario web enviado a diferentes programadores. El formulario contiene preguntas sobre el nivel de conocimiento en áreas como la realidad virtual, programación en Pharo y librerías de visualización.

Posteriormente, se les asignarían tareas de programación sencillas a los postulantes para medir la usabilidad y carga cognitiva al momento de probar el software. Los usuarios pueden realizar las actividades de la forma en que más se sientan cómodos en cuanto a interacción (usando el teclado físico o virtual) y/o la solución a implementar. Las tareas a realizar consisten en:

1. Escribir una clase Librería que pueda contener diferentes ítems, como por ejemplo, Libros, Revistas, etc.
2. Crear métodos que permitan realizar consultas sobre dicha librería. Esto puede hacerse mediante definir métodos sobre la misma clase, o utilizar algún patrón de diseño, como lo es el Visitor Pattern.
3. Crear una visualización del código generado utilizando la librería Roassal. Si lo desea, el usuario puede basarse en alguno de los ejemplos incluidos de antemano en el entorno.

Las diferentes acciones realizadas por el usuario durante el experimento son registradas en un archivo de log. Cada registro es identificado por la fecha y hora en el que fue realizado, y son separados en diferentes categorías como se muestran a continuación:

- **Code Typing**: El usuario empieza a escribir código. El tiempo es tomado desde el momento en que se enfoca un campo de texto.
- **Physical Keyboard**: Registra el momento en que el usuario comienza a teclear en un teclado físico. Este es el modo de escritura por defecto.
- **Virtual Keyboard**: Registra el momento en que el usuario comienza a teclear en un teclado virtual.
- **Playground**: Instante en que se abrió un Playground.
- **Browser**: Instante en que se abrió un Browser.
- **Inspector**: Instante en que se abrió un Inspector.
- **Transcript**: Instante en que se abrió un Transcript.

Al finalizar el experimento, se le solicita al usuario que responda dos cuestionarios. El primero de ellos corresponde a una **escala de usabilidad del sistema** o *System Usability Scale (SUS)*[1], que contiene 10 preguntas, evaluadas en una escala de 1 a 5 (desde *Muy en desacuerdo* hasta *Muy de acuerdo*) [1]. El puntaje final se obtiene a partir de la ecuación:

$$S = ((X - a) + (b - Y)) * 2{,}5 \tag{5.1}$$

donde X es la suma de los puntajes de preguntas de orden impar, Y la suma de puntajes de preguntas de orden par, y a y b factores de conversión, que se les asignará un valor de 5 y 25 respectivamente. Así, el resultado estará en un rango de 0 a 100 [7]. De acuerdo con [1], investigaciones muestran que, en promedio, los sistemas alcanzan un puntaje de 68.

El segundo método corresponde a **NASA Task Load Index** (NASA-TLX). Este se divide en dos secciones: la primera consiste en determinar el peso de una determinada carga de trabajo, mediante escoger el más relevante de entre diferentes pares de carga, mientras que la segunda consiste en asignarle a cada carga un determinado nivel de esfuerzo en una escala de 1 a 20. La carga de trabajo del sistema será igual a la suma de estos niveles ponderados por su peso correspondiente [4]. El formulario es aplicado de manera remota, mediante una implementación web disponible en el Instituto de Seguridad Laboral del Gobierno de Chile[2].

5.2. Resultados

5.2.1. Encuesta preliminar

En total fueron 11 personas quienes respondieron la encuesta de convocatoria. Los resultados pueden observarse en las **Figuras 5.1, 5.2 y 5.3**. La primera muestra estadísticas de uso de sistemas operativos, la segunda sobre el uso de dispositivos de realidad virtual, y la

[1] `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`
[2] `https://ergomedia.isl.gob.cl/app_ergo/nasatlx/`

tercera recopila información sobre el nivel de conocimiento en el lenguaje de programación Pharo y sus librerías de visualización. Los puntos a destacar son:

- Un 64 % usa Mac OS como sistema operativo.
- Un 27 % nunca a usado un dispositivo de realidad virtual.
- La mayoría usa dispositivos Oculus Quest o Quest 2.
- Un 82 % afirma tener alto conocimiento en programación orientada a objetos.
- Un 46 % afirma tener alto conocimiento en Pharo.
- Sólo un 9 % dice tener alto conocimiento en la librería de visualización Roassal.

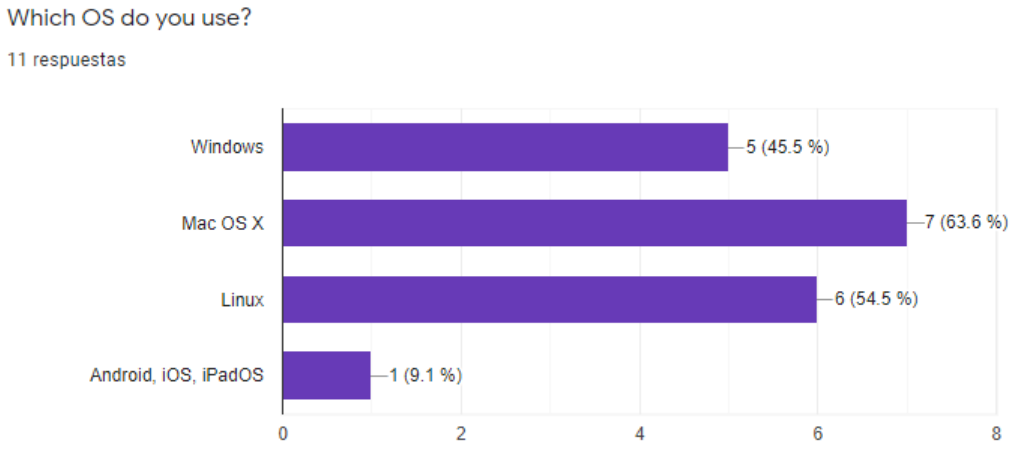

Figura 5.1: Resultados encuesta (SO)

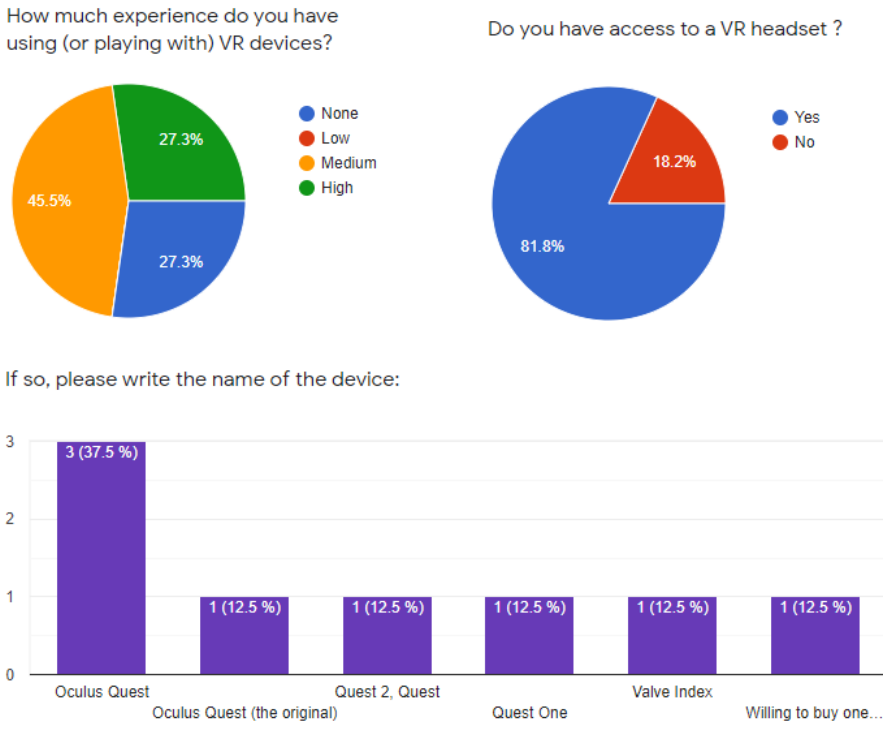

Figura 5.2: Resultados encuesta (Dispositivos VR)

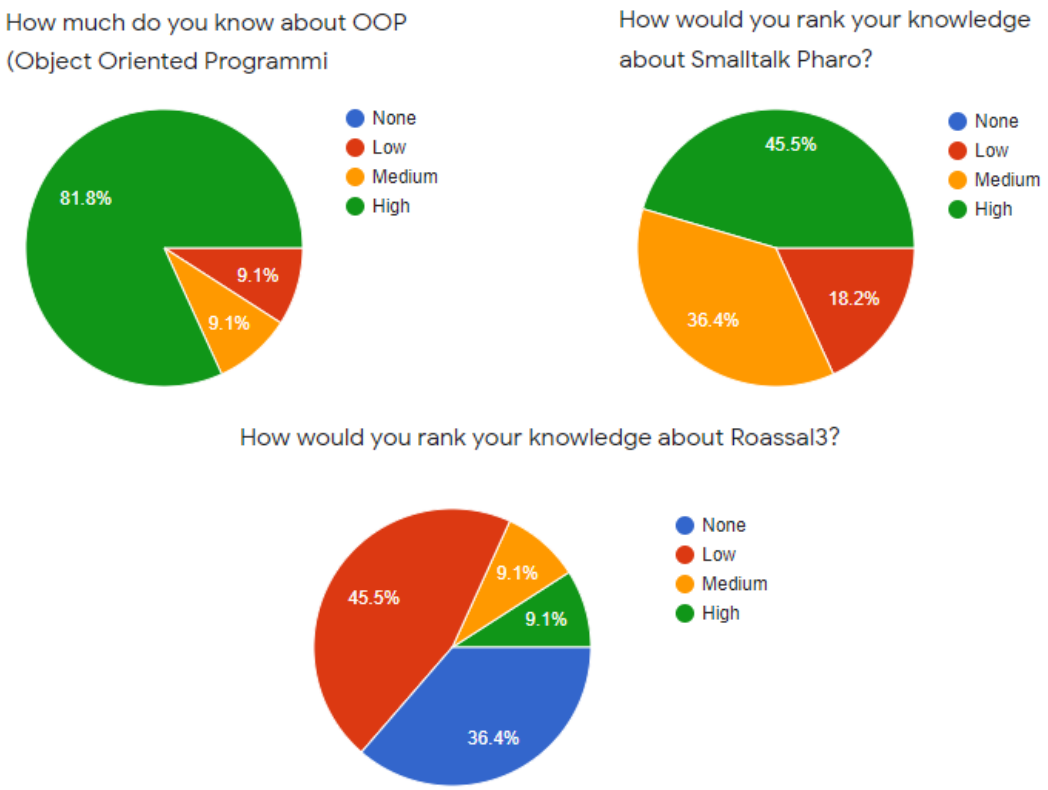

Figura 5.3: Resultados encuesta (Conocimientos en Pharo)

5.2.2. Experimentos con usuarios

Lamentablemente, factores como la disponibilidad de los usuarios, ubicación geográfica o dificultades durante el desarrollo del software, impidieron que las personas que han respondido la encuesta de convocatoria pudieran realizar las pruebas. En su lugar, el experimento se realizó con contactos cercanos, residentes en la ciudad de Santiago. El equipo utilizado en los experimentos fue un visor HTC Vive Cosmos en conjunto con una computadora portátil con sistema operativo Windows 10 de 8 GB de RAM, procesador Intel Core i7-9750h y tarjeta gráfica Nvidia RTX 2060. El tiempo que tardó cada prueba fue aproximadamente entre 60 y 120 minutos para realizar las tareas de programación solicitadas. Las sesiones fueron grabadas bajo el consentimiento de los participantes siempre bajo la supervisión de un observador, con quien tenían permitido resolver dudas y realizar comentarios en voz alta.

Los resultados que se mostrarán a continuación consisten en tres métricas, la primera correspondiente a la escala SUS, la segunda a la encuesta NASA TLX, y la tercera a una visualización de las instancias de interacción del usuario con el sistema. De izquierda a derecha, los puntos representan a una determinada acción realizada en un cierto instante en el tiempo. El color gris corresponde a escritura de código y uso de teclado, verde es a la apertura de ventanas, rojo a la definición de clases y métodos, amarillo a ejecución y azul a visualización. El tamaño de cada punto representa el tamaño de la tarea, medido como el tamaño de la respuesta en caracteres entregada en el registro. Esto también incluye el código que haya escrito el usuario, así como la información recibida desde el servidor.

- **Experimento N°1**: El usuario corresponde a uno de los desarrolladores de la librería Roassal, por lo que posee un amplio conocimiento del lenguaje Pharo y de visualización de código.

  Los resultados de la encuesta SUS arrojan un puntaje de 37,5, situando al sistema debajo de la media. La encuesta NASA TLX (Ver **Tabla 5.1**) alcanzó un puntaje de 850, lo que significa un nivel intermedio de carga cognitiva. Los puntos de mayor exigencia son exigencias mentales, esfuerzo y frustración.

  El gráfico de interacción (Ver **Figura 5.4**) muestra una sesión ágil, esto debido a que las instancias en que necesita abrir ventanas para programar son pocas. La cantidad de clases y métodos utilizados no fueron muchos, y representan poca cantidad de código en comparación a las instancias de ejecución. Las líneas verticales son pocas y en general separadas una de la otra. Se observa que el código de la visualización ha ido aumentado cada vez que este es llamado.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	5	11	55	275
Exigencias Físicas	2	5	25	50
Exigencias Temporales	1	2	10	10
Rendimiento	1	15	75	75
Esfuerzo	4	17	85	340
Frustración	2	17	50	100
TOTAL	15			850

Tabla 5.1: NASA TLX Experimento 1

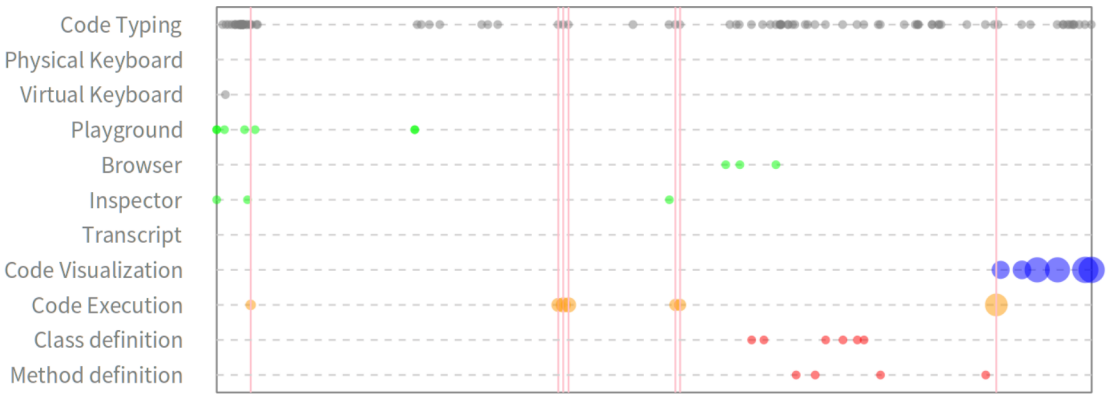

Figura 5.4: Sesión de Experimento 1

- **Experimento N°2**: El usuario evaluado afirma poseer conocimientos sobre Pharo. aunque admite no tener experiencia con librerías de visualización, en particular con Roassal3.

  Los resultados de la encuesta SUS arrojan un puntaje de 70, lo que sitúa al sistema más cerca de la media. Los resultados de NASA TLX (Ver **Tabla 5.2**) muestran un puntaje de 1165, lo que significa un Alto nivel de carga cognitiva, por lo que se deben tomar medidas inmediatas para reducir la carga de trabajo.

  La sesión del usuario (Ver **Figura 5.5**) muestra una mayor densidad de puntos en el gráfico, lo que se traduce en muchas más instancias de escritura de código, apertura de ventanas y escritura de clases y métodos. A diferencia del caso anterior, no parece observarse mayor tamaño de ejecución, aunque sí se aprecia una mayor cantidad de instancias dado que hay más líneas verticales. Sin embargo, la cantidad de visualizaciones realizadas es notablemente menor a las realizadas por el primer usuario.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	4	19	95	380
Exigencias Físicas	1	14	70	70
Exigencias Temporales	1	5	25	25
Rendimiento	3	18	90	270
Esfuerzo	3	15	75	225
Frustración	3	15	65	195
TOTAL	15			1165

Tabla 5.2: NASA TLX Experimento 2

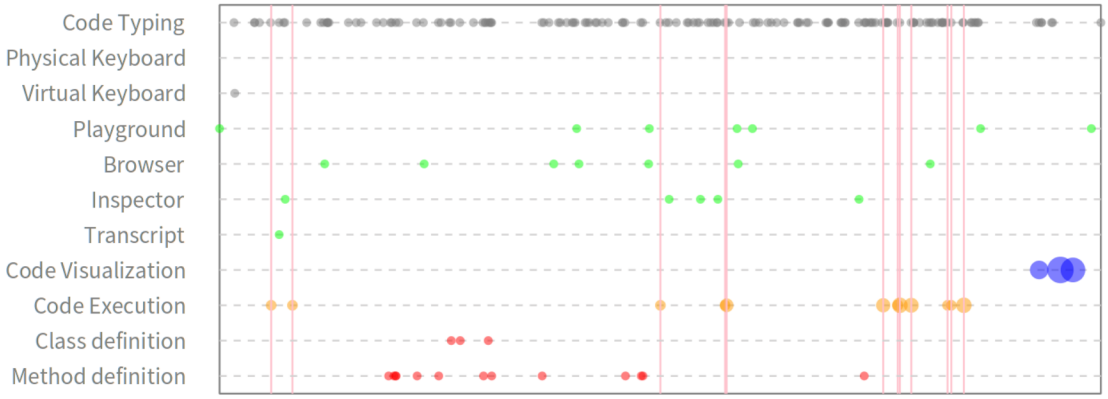

Figura 5.5: Sesión de Experimento 2

- **Experimento N°3**: El usuario afirma tener conocimiento en Pharo y experiencia con la librería Roassal.

  Las respuestas de la escala SUS entregan un puntaje de 57.5. En cuanto a los resultados de NASA TXL (Ver **Tabla 5.3**), se obtuvo un total ponderado de 865, situando al sistema en niveles intermedios de carga cognitiva. Aquellos aspectos que obtuvieron mayor ponderación son exigencias físicas, rendimiento, y esfuerzo.

  La **Figura 5.6** muestra la sesión experimental del tercer usuario. Se puede observar que durante el desarrollo no se han abierto muchas ventanas, lo que se refleja en una poca cantidad de puntos. La cantidad de definiciones y/o actualizaciones de métodos es considerablemente mayor a la creación de clases. Se observan dos instancias de visualización: la primera corresponde a la apertura de un ejemplo, mientras que el segundo aparece tras varias instancias de escritura y edición de código. Se observa una mayor cantidad de líneas verticales, donde se agrupan en dos instancias principales.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	0	7	35	0
Exigencias Físicas	5	13	65	325
Exigencias Temporales	2	9	45	90
Rendimiento	2	11	55	110
Esfuerzo	4	12	60	240
Frustración	2	12	50	100
TOTAL	15			865

Tabla 5.3: NASA TLX Experimento 3

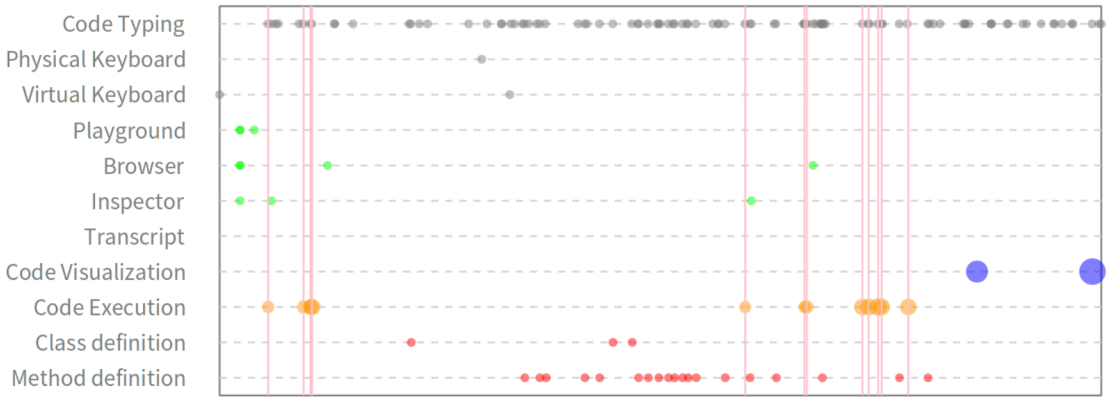

Figura 5.6: Sesión de Experimento 3

- **Experimento N°4**: El usuario afirma tener conocimiento avanzado en Pharo, aunque ha pasado tiempo desde la última vez que ha usado Roassal.

  El puntaje de la escala SUS resultó ser 32.5. En la **Tabla 5.4** se puede observar que el puntaje total ponderado de la encuesta NASA TLX es de 1195 (niveles altos de carga cognitiva). Los aspectos que más destacan son esfuerzo y frustración.

  En la **Figura 5.7** se observa que el periodo de reconocimiento inicial está marcado por la apertura de ventanas y ejemplos de visualización. El usuario cambia constantemente entre el uso del teclado virtual y el uso de teclado físico, así como también invierte más tiempo escribiendo código. Esto puede verse mediante la densidad de puntos durante la definición de métodos, de ejecución, visible a través de una alta densidad de líneas verticales, y de escritura. También se muestra la apertura de muchas ventanas. Esta sesión se diferencia de las demás al mostrar una mayor cantidad de instancias de ejecución al final de la definición de métodos.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	2	13	65	130
Exigencias Físicas	2	11	55	110
Exigencias Temporales	1	9	45	45
Rendimiento	1	18	90	90
Esfuerzo	4	16	80	320
Frustración	5	16	100	500
TOTAL	15			1195

Tabla 5.4: NASA TLX Experimento 4

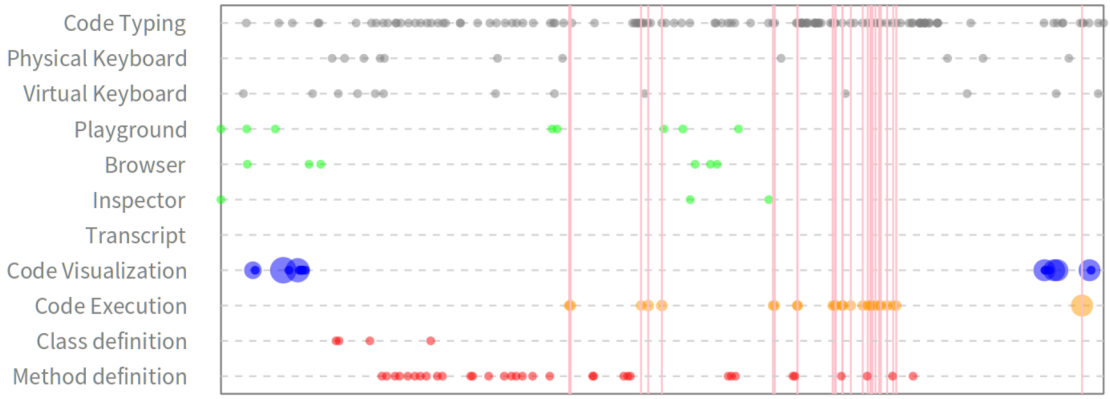

Figura 5.7: Sesión de Experimento 4

- **Experimento N°5**: El usuario de esta sesión dice tener conocimientos tanto en Pharo como en Roassal.

  Tras responder la escala SUS, el puntaje obtenido es de 50, estando dieciocho puntos bajo el promedio. En la **Tabla 5.5** se puede observar que el puntaje total de la encuesta NASA TLX es 1020, situando al sistema en altos niveles de carga cognitiva. Los aspectos que obtuvieron mayor puntaje fueron exigencias físicas, esfuerzo y frustración. No parece haber exigencias temporales.

  Al inicio de la sesión el usuario no necesitó de muchas ventanas, pues son pocos los puntos que se han registrado en el tiempo. Comparado a los anteriores usuarios, presenta una menor cantidad de ejecuciones de código, pues sólo se aprecian un par de líneas verticales. Por otro lado, se ve una mayor cantidad de clases definidas, aunque la concentración de puntos podría indicar modificaciones. En cuanto a la definición de métodos, se ve que hay pares de puntos que están a una distancia relativamente uniforme, lo que podría indicar que hubo corrección de código.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	2	8	40	80
Exigencias Físicas	3	16	80	240
Exigencias Temporales	0	1	5	0
Rendimiento	1	17	85	85
Esfuerzo	4	17	85	340
Frustración	5	17	55	275
TOTAL	15			1020

Tabla 5.5: NASA TLX Experimento 5

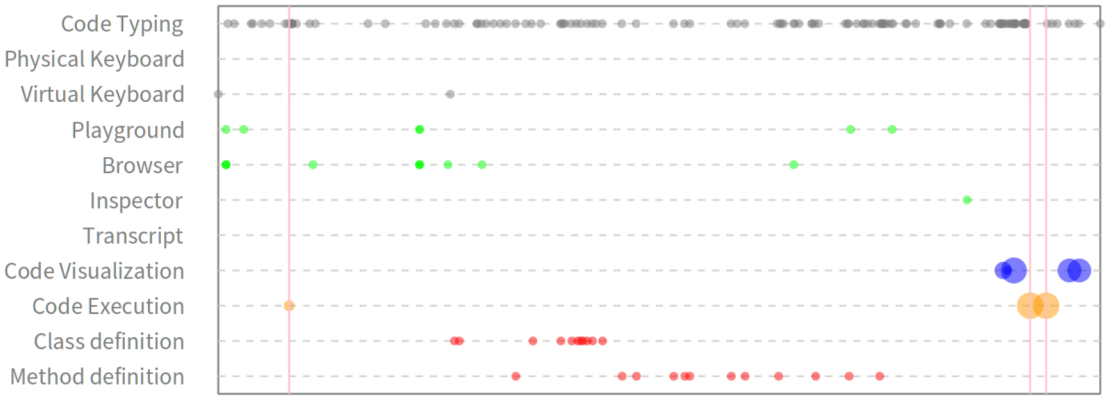

Figura 5.8: Sesión de Experimento 5

- **Experimento N°6**: El usuario posee un conocimiento avanzado del lenguaje de programación Pharo y de librerías de visualización como Roassal.

  El puntaje obtenido tras responder el cuestionario SUS es de 87,5, lo que sitúa al sistema por sobre la media. La **Tabla 5.6** muestra los resultados de NASA TLX, donde la calificación obtenida es de 975 puntos, ubicándose en niveles intermedios de carga cognitiva.

  La sesión presentada en la **Figura 5.9** muestra patrones más definidos durante la escritura de código. Estos se destacan por la definición y/o modificación de métodos y posteriormente una ejecución de código, lo que se ve delimitado por las líneas verticales. La programación ha sido realizada en su mayoría utilizando teclado físico, y destaca por haber empleado patrones de diseño. Al final de la sesión, se puede notar una alta concentración de visualizaciones que van aumentando su cantidad de código paulatinamente.

Variable	(a) Peso	(b) Puntuación	(c) Convertida $(5 * b)$	(d) Ponderada $(c * a)$
Exigencias Mentales	1	3	15	15
Exigencias Físicas	3	18	90	270
Exigencias Temporales	0	3	15	0
Rendimiento	5	20	100	500
Esfuerzo	2	1	5	10
Frustración	4	1	45	180
TOTAL	15			975

Tabla 5.6: NASA TLX Experimento 5

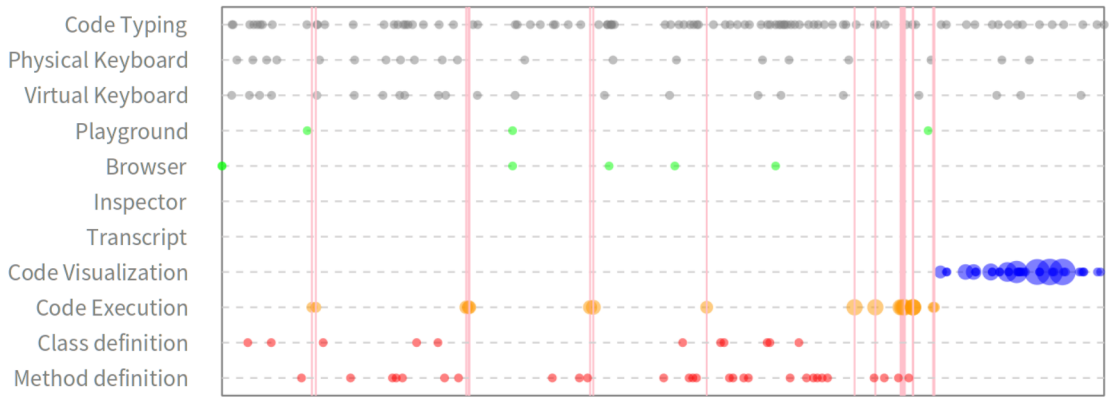

Figura 5.9: Sesión de Experimento 6

- **Resumen General**: A continuación se muestra un resumen de las calificaciones obtenidas en cada encuesta por cada usuario. Como se observó anteriormente, el resultado general es una usabilidad baja, pero a la vez cercana a la media, mientras que el nivel de carga cognitiva varía entre niveles intermedios y altos.

Participante	SUS	NASA TLX
N°1	37.5	850
N°2	70.0	1165
N°3	57.5	865
N°4	32.5	1195
N°5	50.0	1020
N°6	87.5	975
Promedio	55.8	1012

Tabla 5.7: Resumen de los experimentos

- **Notas adicionales**:
  - El usuario del experimento N°4 se sintió más cómodo utilizando el teclado físico en conjunto con el teclado virtual, lo que se ve reflejado en una mayor densidad de puntos de escritura de código y de definición de clases y métodos.
  - El usuario del experimento N°5 dijo sentirse demasiado cansado de usar el casco de realidad virtual durante los últimos minutos del experimento.
  - Los usuarios solían usar un sólo control en lugar de ambos, ocupando con mayor frecuencia la mano con la que mejor escribían cuando se trataba de interactuar con el entorno, ya sea para seleccionar elementos de interfaz, o para escribir en el teclado virtual.
  - Dado que se usó el mismo equipo, aquellos usuarios que llegaban después de que otra persona había realizado el experimento, se encontraba con aquellas ventanas y objetos que estaban abiertos previamente. La totalidad de los usuarios prefirió moverse a otro punto en el espacio y abrir sus propias ventanas, evitando así tener que manipular el espacio de trabajo de otra persona.

Capítulo 6

Discusión

El siguiente capítulo servirá de análisis en torno a los resultados de los experimentos descritos en el capítulo anterior. La discusión planteada tratará temas vistos en el Capítulo 1. En particular, queremos ver cómo se relacionan los resultados con la hipótesis planteada, y qué ventajas y limitaciones de un entorno en realidad virtual se pueden vislumbrar a partir de las sesiones con los usuarios.

6.1. Sobre las estadísticas de usuario

Como se puede observar en la **Figura 5.1** del capítulo anterior, la mayoría de los usuarios que fueron encuestados utilizan un sistema operativo Mac o alguna distribución de Linux, lo que es un factor clave en los problemas para realizar pruebas. Como se observó en las **Figuras 1.4 y 1.5**, el mercado actual es liderado por plataformas como Windows 10 y Android, lo que coincide con las dificultades de compatibilidad que se han tenido durante el desarrollo, pues se suele priorizar el soporte de las herramientas a dicha plataformas, como se puede observar en la **Figura 6.1**.

De acuerdo con la **Figura 5.2**, mucha gente afirma tener acceso a un dispositivo VR, concretamente de la línea Oculus, lo que también concuerda con lo mostrado en la **Figura 1.4**. Si bien no ha sido posible desarrollar el software para el modelo Quest 2, debido a que es un lanzamiento reciente, sí se ha logrado ejecutar en modelos anteriores.

A partir de esto, se puede desprender la idea de que existe una barrera de entrada para quienes estén interesados en la programación en realidad virtual debido a dos factores:

- Las preferencias de los programadores en cuanto a tecnología de hardware y software, pues Linux y OS X son las opciones más comunes.
- El enfoque de las experiencias VR hacia la industria del entretenimiento, siendo Windows quien lidera este sector.

Por otro lado, la totalidad de los encuestados demuestra tener conocimientos en Pharo. No obstante, son pocos los que están familiarizados con la librería de visualización Roassal. Esto motivó a que se precargaran ejemplos de esta librería en el software, pues es común que los

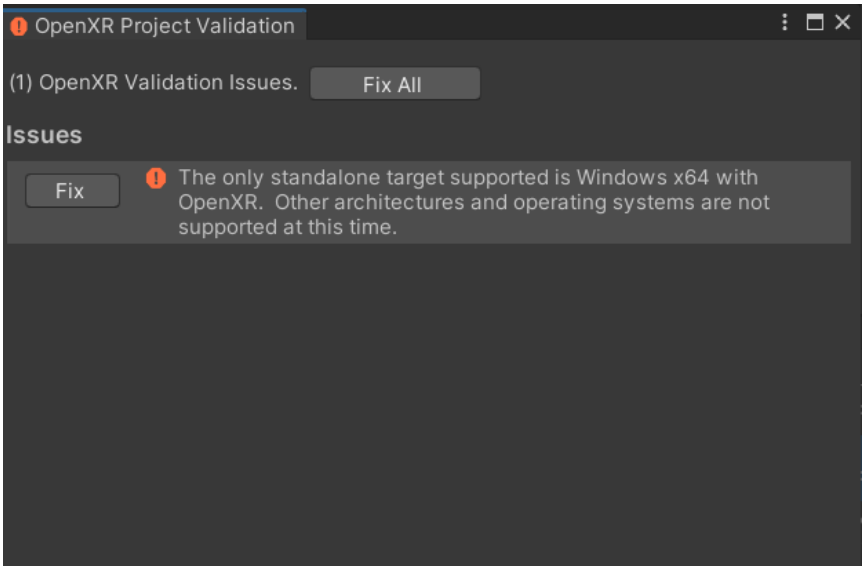

Figura 6.1: Demostración de log al momento de compilar para plataformas OS X y Linux

programadores estén más familiarizados con visualización de datos (por ejemplo, *Matplotlib* en Python o *JFreeChart* en Java) que con visualización de código. Dado este problema, podría ser mucho más útil para el usuario que las visualizaciones sean visibles siempre que se esté escribiendo código, liberando así una mayor carga cognitiva.

6.2. Sobre los resultados de la experimentación

De acuerdo los resultados presentados en la **Tabla 5.7** de la **Sección 5**, vemos que el puntaje promedio de carga cognitiva es 1012. Esto implica que el prototipo del sistema usado se encuentra actualmente bajo altos niveles de carga cognitiva. Aquellas variables que más han obtenido puntajes altos fueron esfuerzo y frustración. El promedio de los puntajes obtenidos a partir de la System Usability Scale forman un promedio de 55,8. En consecuencia, la escala de usabilidad del software se encuentra por debajo de la media.

Se puede decir que estos resultados se encuentran dentro de las dificultades esperadas, pues a pesar del esfuerzo por reducir la complejidad del entorno, los usuarios manifiestan mayor frustración que cuando lo harían en un entorno de programación convencional. No obstante, existe otra mirada que vale la pena destacar. Los puntajes de la escala SUS, por ejemplo, son más altos de lo que podrían esperarse, estando más cerca del promedio general que se postula en [1].

Pareciera que, si bien el software representa una alta carga cognitiva, sigue siendo un sistema usable. Esto es apoyado por los registros de sesiones, donde los usuarios han podido satisfactoriamente realizar tareas sencillas de programación y realizar visualizaciones simples de objetos y/o jerarquías de clase. La **Figura 5.6**, por ejemplo, muestra claramente que durante la fase de desarrollo y visualización, existen tanto instancias de definición y ejecución de líneas de código usando objetos, como la creación de una visualización a partir de un código de ejemplo. La **Figura 5.7** presenta la mayor cantidad de instancias de ejecución de código. Esto puede atribuirse a que, complementando los tipos de teclado, pueda realizar código

mucho más complejo y, por tanto, más susceptible a errores y tests.

Existen también otros factores más allá de la usabilidad del software, y se relacionan con las preferencias de cada usuario mencionadas en las notas adicionales del Capítulo 5, como lo son el uso de un teclado físico o virtual, y la forma en que son manipulados los controles. Estas tendencias responderían a las limitaciones de programar en un entorno en VR, y podrían explicar por qué los usuarios optan por un acercamiento más sencillo al problema. Esto se ve reflejado a la hora de utilizar la menor cantidad de clases y métodos posibles, pues así se ahorra mucho más tiempo y compensa los niveles de frustración. En consecuencia, los factores antes mencionados guardan relación con la necesidad del usuario de tener que adaptarse al funcionamiento de la realidad virtual.

Otro aspecto relevante de los experimentos es la capacidad que tuvieron los usuarios para utilizar el espacio disponible para abrir nuevas ventanas, tal como se mencionó en las notas adicionales. Esto confirma una de las ventajas de programación en VR de la que se habló en el Capítulo 1, que consiste en poder usar un espacio teóricamente infinito para trabajar, dada la tridimensionalidad del entorno.

Finalmente, fue posible observar comportamiento propio de un ambiente de Live Programming. Los usuarios que lograron crear visualizaciones solían ejecutar códigos de ejemplo, para poder así editarlos en el Playground, y luego ver cómo este variaba a medida que se cambiaban los objetos a representar.

Capítulo 7

Conclusión

Al principio de este proyecto, nuestra hipótesis era que es posible crear un entorno de programación completamente en realidad virtual a partir de la tecnología de la actualidad. Tras los resultados obtenidos durante la fase experimental, podemos concluir que, efectivamente, sí es posible realizar Live Programming en un ambiente de este tipo. Los usuarios sometidos a las tareas de programación en VR fueron capaces de escribir objetos en Pharo y usarlos para generar visualizaciones de manera satisfactoria. También se ha podido observar cómo los participantes interactúan con el entorno mediante el uso de controles VR, teclado físico y teclado virtual.

No obstante, esta metodología ha estado sujeta a diversos obstáculos y limitaciones. Para empezar, los resultados de la experimentación podrían no ser del todo contundentes, dada la baja cantidad de participantes debido a la reciente crisis sanitaria.

Por otro lado, el hecho de que la tecnología de realidad virtual esté mayormente enfocada al área del entretenimiento y no a tareas de programación propiamente tal, hace que el rango de acciones dentro de un entorno de desarrollo sea limitado. Es por esta razón que los usuarios puedan encontrar el proceso más lento y tedioso, pues no es lo mismo usar las articulaciones y la visión de forma directa frente a un teclado y una pantalla, que utilizar un casco y un par de controles con una cantidad limitada de botones y gestos.

Además, el alcance de este software dependerá de las capacidades de acceso que tenga un usuario, pues no todos tienen la oportunidad de usar un dispositivo, así como no todos usan el mismo dispositivo. De igual manera, el enfoque de las compañías proveedoras hacia un determinado tipo de dispositivo, software o sistema operativo, provoca la alienación de los usuarios que buscan en estas herramientas oportunidades más allá de simples representaciones gráficas.

En consecuencia, hubieron elementos que no fue posible concretar, como el enfoque de compatibilidad hacia las diferentes plataformas y dispositivos debido a problemas surgidos durante el desarrollo, así como otros métodos de interacción que no involucren necesariamente el uso de controladores.

Como resultado, es la propia comunidad de programadores la que ha inventado diversas herramientas y extensiones (como lo son los trabajos relacionados mencionados en la introducción y los recursos utilizados durante el desarrollo) para expandir el uso de la tecnología VR más allá de lo que está actualmente. Gracias a estas contribuciones es que este proyecto a sido posible, y es por esa razón que para el trabajo futuro, se buscará explorar otras estrategias que hagan de la programación virtual mucho más amena.

Bibliografía

[1] U.S. General Services Administration. *System Usability Scale (SUS).* [En Línea], Disponible en `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html` [Visitado: 04-01-2021].

[2] N. Capece, U. Erra, S. Romano, and G. Scanniello. *Visualising a Software System as a City Through Virtual Reality.* Universita degli Studi della Basilicata, Potenza, Italy, Junio 2017.

[3] Valve Corporation. *Encuesta sobre hardware y software de Steam: November 2020.* [En Línea], Disponible en `https://store.steampowered.com/hwsurvey` [Visitado: 29-12-2020].

[4] S. G. Hart. *NASA-Task Load Index (NASA-TLX); 20 Years Later.* NASA-Ames Research Center, Moffett Field, CA, 2006.

[5] Kronos Group Inc. *OpenXR Overview.* [En Línea], Disponible en `https://www.khronos.org/openxr/` [Visitado: 06-01-2021].

[6] C. Latta. *CaffeineJS: Livecode the Web!* [En Línea], Disponible en `https://caffeine.js.org` [Visitado: 28-04-2020].

[7] J. R. Lewis and J. Sauro. *Item Benchmarks for the System Usability Scale.* Journal of Usability Studies, Vol. 13, Issue 3, May 2018 pp. 158–167.

[8] L. Merino, A. Bergel, and O. Nierstrasz. *Overcoming Issues of 3D Software Visualization through Immersive Augmented Reality.* in IEEE Working Conference on Software Visualization (VISSOFT 2018), at Madrid, Spain, September 2018.

[9] L. Merino, M. Hess, A. Bergel, O. Nierstrasz, and D. Weiskopf. *PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness.* in Companion of the 2019 ACM/SPEC International Conference, March 2019.

[10] S. Pittet. *The different types of software testing.* [En Línea], Disponible en: `https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing`, [Visitado: 20-12-2020].

[11] StatCounter. *Operating System Market Share Worldwide.* [En Línea], Disponible en `https://gs.statcounter.com/os-market-share` [Visitado: 29-12-2020].