UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO
CENTRO UNIVERSITARIO UAEM ECATEPEC

Ingeniería en Computación

Análisis de rendimiento en aplicaciones de VR y AR desarrolladas con diferentes técnicas.

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

Presenta:

C. SERGIO GALICIA DEL ANGEL

Asesor:

M. EN C.C. JESÚS BARRÓN VIDALES

Revisores:

DRA. EN I. DE S. TERESA IVONNE CONTRERAS TROYA

M. EN C.C. ENRIQUE JOSÉ TINAJERO PÉREZ

ECATEPEC DE MORELOS, ESTADO DE MÉXICO, MARZO DE 2020.

CARTA DE CESIÓN DE DERECHOS DE AUTOR

El (la) que suscribe ***SERGIO GALICIA DEL ANGEL*** Autor del trabajo escrito de evaluación profesional en la opción de **TESIS** con el título **"ANÁLISIS DE RENDIMIENTO EN APLICACIONES DE VR Y AR DESARROLLADAS CON DIFERENTES TÉCNICAS"** por medio de la presente con fundamento en lo dispuesto en los artículos 5, 18, 24, 25, 27, 30, 32 y 148 de la Ley Federal de Derechos de Autor, así como los artículos 35 y 36 fracción II de la Ley de la Universidad Autónoma del Estado de México; manifiesto mi autoría y originalidad de la obra mencionada que se presentó en el Centro Universitario UAEM Ecatepec para ser evaluada con el fin de obtener el Título Profesional de **INGENIERÍA EN COMPUTACIÓN.**

Así mismo expreso mi conformidad de ceder los derechos de reproducción, difusión y circulación de esta obra, en forma NO EXCLUSIVA, a la Universidad Autónoma del Estado de México; se podrá realizar a nivel nacional e internacional, de manera parcial o total a través de cualquier medio de información que sea susceptible para ello, en una o varias ocasiones, así como en cualquier soporte documental, todo ello siempre y cuando sus fines sean académicos, humanísticos, tecnológicos, históricos, artísticos, sociales, científicos u otra manifestación de la cultura.

Entendiendo que dicha cesión no genera obligación alguna para la Universidad Autónoma del Estado de México y que podrá o no ejercer los derechos cedidos.

Por lo que el autor da su consentimiento para la publicación de su trabajo escrito de evaluación profesional.

a) Texto completo ■
b) Por capítulo ☐
c) Solamente portada y tabla de contenido ☐

Se firma presente en la ciudad de Ecatepec de Morelos, Estado de México, a los 20 días del mes de Marzo de 2020.

SERGIO GALICIA DEL ANGEL

ÍNDICE

RESUMEN 1
INTRODUCCIÓN 2
PLANTEAMIENTO DEL PROBLEMA 4
HIPÓTESIS 4
OBJETIVOS 5
JUSTIFICACIÓN 6
MARCO TEÓRICO 7
MARCO HISTÓRICO Y REFERENCIAL 13
CAPÍTULO 1: ANÁLISIS Y METODOLOGÍA.
1.1. PROPUESTA DE PROTOTIPO 19
1.2. IDENTIFICACIÓN DE REQUERIMIENTOS 19
1.3. ESPECIFICACIÓN DE REQUERIMIENTOS 20
1.4. ANÁLISIS DE RIESGOS 21
1.5. DESARROLLO EVOLUTIVO 22
1.5.1. FASE DE ANÁLISIS 23
1.5.2. FASE DE EVALUACIÓN 23
1.5.3. FASE DE CIERRE 23
1.6. GENERAL GAME PRODUCTION PIPELINE 23
1.6.1. FASE CONCEPTUAL (CONCEPT PHASE) 25
1.6.2. CREACIÓN DE CONTENIDO 3D (3D CONTENT CREATION PIPELINE) 25
1.6.3. ARTE CONCEPTUAL (CONCEPT ART) 25
1.6.4. MODELADO DE BAJO POLIGONAJE (LOW POLYGON MODELING) 25
1.6.5. DISEÑO UV (UV LAYOUT) 26
1.6.6. MODELADO DE ALTO POLIGONAJE (HIGH POLYGON MODELING) 27
1.6.7. TEXTURIZADO (TEXTURING) 27
1.6.8. RIGGING Y ANIMACIÓN (RIGGING AND ANIMATION) 27
1.6.9. DISEÑO DE NIVEL (LEVEL DESIGN) 28
1.6.10. ILUMINACIÓN (LIGHTING) 28
1.6.11. IMPLEMENTACIÓN (IMPLEMENTATION) 29
1.6.12. LANZAMIENTO (RELEASE) 29
CAPÍTULO 2: TÉCNICAS DE OPTIMIZACIÓN.
2.1. MODELADO 3D 30
2.1.1. Low Poly 31
2.2. CREACIÓN DE TEXTURAS 34
2.2.1. UV Layout 34

2.3. CREACIÓN DE ASSETS Y DISEÑO DEL NIVEL EN UNITY 35
2.3.1. Procesamiento por lotes (Batching) 36
2.3.2. Sistemas de partículas 38
2.3.3. Colisionadores (Colliders) 39
2.3.4. Cuadro y barra de desplazamiento (Scroll Rect and Scroll Bar) 40
2.3.5. Botón 40
2.4. ILUMINACIÓN 41
2.4.1. Baked lighting y lightmapping como técnicas de optimización en VR 42
2.5. BUENAS PRÁCTICAS DE PROGRAMACIÓN COMO OPTIMIZACIÓN EN UNITY 44
2.5.1. Eliminación de declaraciones innecesarias 44
2.5.2. La función Awake 46
2.5.3. Uso de Break y Return 48
CAPÍTULO 3: DESARROLLO DE PROTOTIPOS.
3.1. CONSIDERACIONES DE DISEÑO 50
3.2. DISEÑO DE INTERFAZ 57
3.3. SECCIÓN PRINCIPAL: “EL ÁTOMO” 61
3.4. SECCIÓN SECUNDARIA: “EL VISUALIZADOR” 63
3.5. SECCIÓN: “CARACTERÍSTICAS” 64
3.6. SECCIÓN: “INFORMACIÓN” 65
3.7. SECCIÓN: “TABLA PERIÓDICA Y CARACTERÍSTICAS” 69
3.8. SECCIÓN: “AR” 70
3.9. PROGRAMACIÓN DE INTERACCIONES 71
3.9.1. Programación y uso del raycast en VR 72
3.9.2. Programación de efectos visuales 74
3.8.3. Activación y desactivación de objetos 75
CAPÍTULO 4: EVALUACIÓN DE PROTOTIPOS.
4.1. FUNCIONALIDAD 77
4.2. PRUEBAS DE RENDIMIENTO 81
4.2.1. Unity Profiler 81
4.2.2. Obtención de datos 82
4.2.3. Comparación de datos 85
4.3. EXPERIENCIA DE USUARIO 99
4.3.1. Comparación de datos 102
CONCLUSIONES 113
Trabajos futuros 114
REFERENCIAS BIBLIOGRÁFICAS 116

ÍNDICE DE FIGURAS

Figura 1. Geometría que forma al target o marcador. 9
Figura 2. Modelado 3D de elefante creado con polígonos de tres lados (triángulos) 11
Figura 3. Metodología de desarrollo evolutivo de escenarios tridimensionales. 22
Figura 4. Fases del Game production pipeline 24
Figura 5. UV Layout del modelado 3D de una casa 26
Figura 6. Estructura ósea para la elaboración de animaciones....28
Figura 7. Implementación de *assets* para la elaboración de un nivel en Unity.....29
Figura 8. Modelo 3D low poly....31
Figura 9. Resultado de la aplicación booleana Unión.....32
Figura 10. Eliminación de los polígonos resultantes de la operación booleana y sustitución de éstos 32
Figura 11. El punto amarillo representa lo que parece ser, la unión de dos líneas rectas....33
Figura 12. Una toma más cercana muestra que, lo que parecía ser un vértice, en realidad son dos 33
Figura 13. UV Layout de modelos 3D que conforman la cama en la escena de realidad virtual. 34
Figura 14. Optimización de texturas por medio del inspector de Unity.....35
Figura 15. Ventana Statistics, la cual permite conocer métricas de rendimiento 36
Figura 16. Marcando a un GameObject como estático para el uso de Static Batching. 38
Figura 19. Componente Button donde se aprecia la clase OnClick. 41
Figura 20. Ventana Lighting, la cual muestra los modos de iluminación y lightmapping. 43
Figura 21. Capas en la interfaz gráfica de Adobe Photoshop. 50
Figura 22. Ícono del Oro dentro de la tabla periódica....51
Figura 23. Boceto para la elaboración del ícono del átomo dentro del nivel de VR. 51
Figura 24. Elaboración de íconos por medio de capas....52
Figura 25. Creación de imágenes vectoriales en Adobe Photoshop para su uso en la UI.....52
Figura 26. Referencia del espectro electromagnético e imagen resultante.....53
Figura 27. Conjunto de modelos 3D para su uso en el nivel de VR del presente caso práctico....53
Figura 28. Eliminación de polígonos no visibles por el usuario (polígonos inferiores y traseros). 54
Figura 29. Combinación de objetos como técnica de optimización....54
Figura 30. Implementación de la textura resultante con UV´s sobre la superficie del modelo 3D. 55
Figura 31. Captura de pantalla de la venta “proyecto” perteneciente a la interfaz gráfica de Unity. 55
Figura 32. Primer nivel, en el que el usuario determina el uso de VR o AR....56
Figura 33. Indicador y cuadricula que permiten el desplazamiento dentro de la habitación. 57
Figura 34. Boceto de la interfaz del nivel de AR que permite la navegación e interacción. 58
Figura 35. Plano de pantalla de la interfaz del nivel de AR cuya función es de guía visual.....59
Figura 36. Plano de pantalla de la interfaz dentro de la sección “el átomo” del nivel de VR.....59
Figura 37. Plano de pantalla de la comunicación entre la sección “información” y “el visualizador”.....60

Figura 38. Planos de pantalla de las secciones “características” y “el visualizador” del nivel de VR 60
Figura 39. Uno de los ciento dieciocho átomos hechos de sistemas de partículas. 62
Figura 40. Colliders representados como objetos que encapsulan a su objeto contenedor 63
Figura 41. Imágenes de los ciento dieciocho elementos químicos que conforman la tabla periódica. 64
Figura 42. Sección que muestra las características del elemento químico. 65
Figura 43. Información sobre el átomo mostrada en “el visualizador”. 66
Figura 44. Ventana Inspector, la cual muestra cómo declarar a un gameobject como estático. 67
Figura 45. Configuración del tipo de luz point light en donde se especifica el modo como baked. 67
Figura 46. Conjunto de lightmaps y de los datos de iluminación dentro del proyecto en Unity. 68
Figura 47. Antes y después de baked lighting en el nivel de VR. 69
Figura 48. Tabla periódica dentro de un Scroll Rect para poder visualizar dentro del área blanca. 69
Figura 49. Vista del átomo en AR. 71
Figura 50. Sistemas de partículas rodeando el centro de un gameobject padre 75
Figura 51. Ventana Profiler, la cual muestra los datos de la medición en milisegundos. 83
Figura 52. Comparación de resultados de la pregunta 1 del nivel de AR. 103
Figura 53. Comparación de resultados de la pregunta 2 del nivel de AR. 103
Figura 54. Comparación de resultados de la pregunta 3 del nivel de AR. 104
Figura 55. Comparación de resultados de la pregunta 4 del nivel de AR. 104
Figura 56. Comparación de resultados de la pregunta 5 del nivel de AR. 105
Figura 57. Comparación de resultados de la pregunta 6 del nivel de AR. 105
Figura 58. Comparación de resultados de la pregunta 7 del nivel de AR. 106
Figura 59. Comparación de resultados de la pregunta 8 del nivel de AR. 106
Figura 60. Comparación de resultados de la pregunta 1 del nivel de VR. 107
Figura 61. Comparación de resultados de la pregunta 2 del nivel de VR. 107
Figura 62. Comparación de resultados de la pregunta 3 del nivel de VR. 108
Figura 63. Comparación de resultados de la pregunta 4 del nivel de VR. 108
Figura 64. Comparación de resultados de la pregunta 5 del nivel de VR. 109
Figura 65. Comparación de resultados de la pregunta 6 del nivel de VR. 109
Figura 66. Comparación de resultados de la pregunta 7 del nivel de VR. 110
Figura 67. Comparación de resultados de la pregunta 8 del nivel de VR. 110
Figura 68. Comparación de resultados de la pregunta 9 del nivel de VR. 111
Figura 69. Comparación de resultados de la pregunta 10 del nivel de VR. 111
Figura 70. Comparación general de resultados. 112

ÍNDICE DE TABLAS

Tabla 1. Requerimientos funcionales y no funcionales. 20
Tabla 2. Análisis de riesgos. 21
Tabla 3. Pruebas de funcionalidad de las distintas secciones de la aplicación. 77
Tabla 4. Interacciones y acciones medidas en tiempo de ejecución. 83
Tabla 5. Momentos en los que se obtuvieron los valores máximos y mínimos. 85
Tabla 6. Comparación de resultados obtenidos del componente CPU. 86
Tabla 7. Comparación de resultados obtenidos del componente *Rendering*. 89
Tabla 8. Comparación de resultados obtenidos del componente Memory. 93
Tabla 9. Comparación de resultados obtenidos del componente Global Illumination. 96
Tabla 10. Comparación de resultados obtenidos del componente Physics. 97

RESUMEN

En este estudio se comprueba la viabilidad del uso de técnicas de optimización en cada una de las fases que conforman el desarrollo de una aplicación móvil que implementa realidad virtual y realidad aumentada, por medio de la medición de métricas de rendimiento y un instrumento aplicado a 30 estudiantes de la licenciatura en ingeniería en computación del Centro Universitario UAEM Ecatepec para medir la experiencia de usuario de éstos. Los datos obtenidos comprueban que una aplicación para dispositivos móviles que implementa realidad virtual y realidad aumentada desarrollada con dichas técnicas, presenta una disminución en la demanda de procesamiento sin afectar significativamente o de manera perceptible la experiencia de usuario.

PALABRAS CLAVE: Técnicas de optimización; aplicación móvil; realidad virtual; realidad aumentada; experiencia de usuario.

INTRODUCCIÓN

El presente trabajo consiste en el desarrollo y comparación de dos versiones de una aplicación para dispositivos móviles que implementa Realidad Aumentada y Realidad Virtual (AR y VR, por sus siglas en inglés). La diferencia entre las versiones radica en las prácticas y técnicas de optimización implementadas en el desarrollo de una de ellas, lo cual se hizo con el objetivo de demostrar que la implementación de dichas prácticas y técnicas mejoran el desempeño de la aplicación y disminuyen la demanda de recursos de hardware sin afectar de manera significativa la experiencia de usuario en los aspectos visual, auditivo, o en la interacción.

Debido al impacto que ha tenido la implementación de VR y AR en áreas como la medicina, la electrónica, la mecánica, la astronomía, la arquitectura, la mercadotecnia, el ámbito educativo y el entretenimiento, el caso práctico consiste en una aplicación capaz de mostrar la estructura básica e información del átomo de cada uno de los elementos químicos que conforman la tabla periódica en una escena *inmersiva* de VR y otra de AR.

Se tiene como objetivo generar en tiempo de ejecución una demanda de recursos de hardware significante para su desarrollo, pues el uso de VR requiere de iluminación, sombreado, animaciones e interacciones por parte del usuario dentro de un nivel *inmersivo*, y el uso de AR requiere de la cámara del dispositivo, de iluminación, sombreado, animaciones e interacciones. Los requerimientos anteriores tienen como propósito generar un buen nivel de *inmersión* y a su vez una buena experiencia de usuario.

Para lograr un buen desempeño en la ejecución de cualquier aplicación móvil y un buen nivel de *inmersión*, es necesaria la representación y reproducción de elementos digitales óptimos, por lo que en cada una de las fases de desarrollo de una de las versiones se implementaron prácticas y técnicas de optimización.

Ambas versiones de la aplicación fueron expuestas a pruebas de rendimiento (hardware) y a pruebas de usabilidad con estudiantes de ingeniería en computación del Centro Universitario UAEM Ecatepec, con la finalidad de comprobar la viabilidad de desarrollo y la mejora en rendimiento.

Los alcances o límites del presente trabajo radican en la experiencia de usuario deseada, pues ésta determina el estilo visual de todos los elementos digitales y las técnicas de optimización a implementar en cada fase del desarrollo. La delimitación de la optimización implica una experiencia de usuario atractiva pero funcional para un dispositivo móvil y la cantidad de elementos digitales a mostrar o reproducir en el tiempo de ejecución, y aunque exista un gran número de técnicas de optimización o buenas prácticas de desarrollo es necesario considerar los siguientes puntos que establecen un límite:

- La experiencia de usuario deseada.
- El estilo visual de la aplicación.
- La cantidad de contenido multimedia.
- El hardware del dispositivo objetivo.

PLANTEAMIENTO DEL PROBLEMA

La VR y la AR son tecnologías ejecutadas en su gran mayoría en dispositivos móviles, cuyo nivel de procesamiento gráfico no es tan eficiente como el de una computadora, aun cuando la mayoría de éstos cuentan ya con una unidad de procesamiento gráfico.

La problemática que se tiene a menudo en la implementación de dichas tecnologías son las causas de la demanda de procesamiento, las cuales pueden ser: la correcta representación de objetos tridimensionales, un alto nivel de *inmersión*, buena calidad de iluminación y sombreado, texturas de alta resolución, animaciones y audio. Cada elemento del conjunto anterior requiere de un proceso de desarrollo y contribuye a la demanda final de procesamiento en la ejecución de la aplicación. Dicha demanda significa un desafío en el desarrollo de aplicaciones para plataformas móviles e inclusive con el uso de técnicas de optimización en la fase de modelado tridimensional existe la posibilidad de un desempeño no deseado en un porcentaje significativo de dispositivos.

La implementación de técnicas de optimización y/o buenas prácticas durante el desarrollo es fundamental para reducir la demanda de procesamiento, sin embargo, el uso excesivo de técnicas de optimización o la optimización innecesaria puede afectar la calidad visual, auditiva e interactiva que forman parte de la experiencia de usuario.

HIPÓTESIS

Una aplicación de VR y AR para dispositivos móviles que se desarrolla con métodos de optimización y buenas prácticas presenta en sus métricas de desempeño una menor demanda de recursos de hardware y software sin afectar negativamente la experiencia de usuario de manera visual, auditiva o interactiva.

OBJETIVOS

Objetivo general

Comprobar la viabilidad de desarrollo de una aplicación móvil con realidad virtual y realidad aumentada utilizando técnicas de optimización para reducir la demanda de recursos de procesamiento sin que se vea afectada significativamente la experiencia de usuario.

Objetivos específicos

- Aplicar buenas prácticas y técnicas de optimización durante el desarrollo de una aplicación de VR y AR.
- Usar la optimización de *modelos 3D* como una técnica de diseño.
- Utilizar técnicas de optimización de renderizado para un buen desempeño visual en dispositivos móviles.
- Analizar la relación optimización-experiencia de usuario.
- Delimitar la optimización con base en la experiencia de usuario deseada.
- Diseñar un instrumento para evaluación de la percepción de usuario.

JUSTIFICACIÓN

El tema de investigación del presente trabajo consiste en el desarrollo de una aplicación móvil y en la implementación de técnicas de optimización para reducir la demanda de recursos de procesamiento. El desarrollo de dicha aplicación se relaciona con la planeación, el diseño, la programación, buenas prácticas y el análisis de un proyecto de desarrollo de sistemas computacionales, como los ejercidos durante la licenciatura en Ingeniería en Computación de la Universidad Autónoma del Estado de México. La importancia del estudio está en comprobar la viabilidad del desarrollo de una aplicación de realidad virtual y aumentada utilizando técnicas de optimización sin afectar significativamente la experiencia de usuario, teniendo como beneficio mejorar el rendimiento en dispositivos móviles para permitir una óptima ejecución y una experiencia de usuario satisfactoria.

Las aportaciones prácticas que genera la presente investigación se relacionan con la optimización y el análisis de la experiencia de usuario, debido al uso de técnicas de optimización y a los instrumentos que permiten la obtención de métricas para un análisis de desempeño. Las aportaciones prácticas son:

- La generación de una experiencia de usuario óptima en aplicaciones de VR y/o AR para dispositivos móviles por medio de buenas prácticas y técnicas de optimización.
- Técnicas y buenas prácticas para implementar en las fases de desarrollo de una aplicación de VR y/o AR para dispositivos móviles.
- Una metodología de desarrollo de entornos virtuales basada en buenas prácticas de modelado 3D.
- Métodos para identificar y mejorar problemas de rendimiento en tiempo de ejecución de aplicaciones de VR y/o AR.
- Técnicas de iluminación para la mejora visual y de desempeño.

MARCO TEÓRICO

Realidad virtual

La VR carece de adiciones de objetos digitales al mundo real, sin embargo, contiene también un alto nivel de *inmersión*, pues en la VR se crea un entorno virtual para generar al usuario la sensación de encontrarse en dicho entorno y mejorar la experiencia de éste mediante interacciones con objetos o elementos multimedia. Pérez (2011), declaró que la segunda revolución tecnológica la ha propiciado la Realidad Virtual, la percepción en 3D de entornos simulados que permiten trasladar al usuario a mundos de ensueño y le posibilitan viajar a través del tiempo al pasado y al futuro.

Al contrario que la AR, la VR no usa técnicas con *marcadores* o *targets* que ayuden a posicionar un objeto tridimensional. La VR se apoya de la implementación de una visión estereoscópica, de un giroscopio y de un visor con lentes biconvexos o un dispositivo Head-Mounted Display (HMD) para generar un alto nivel de *inmersión*. En la actualidad la mayoría del hardware para implementar VR de alta calidad o de alto rendimiento es alámbrico y requiere de un equipo de cómputo con características específicas para su uso. Sin embargo, existe una alternativa gracias a los dispositivos móviles y a las características de sus componentes, los cuales permiten la implementación de aplicaciones de VR sin la necesidad de dispositivos alámbricos o de un equipo de cómputo, por eso Steed y Julier (2013), creen que el poder creciente de las CPU y GPU móviles es cada vez más fácil integrar todos los componentes de un sistema de realidad virtual interactivo e *inmersivo* en un dispositivo móvil.

Realidad aumentada

Cawood y Fiala (2007), mencionaron que liberar a los jugadores de sus salas, es sólo una aplicación de la floreciente tecnología llamada realidad aumentada (AR).
Según Cawood, el primer sistema de VR y AR que se desarrolló y que es considerado el origen de la AR, fue el trabajo de Ivan Sutherland, quien creó un prototipo basado en *wire-frames* que consta de visualizar la estructura de un objeto de manera plana o bidimensional. El resultado de las primeras implementaciones de AR fueron dos técnicas llamadas Magic Mirror y Magic Lens, la primera consiste en colocar una pantalla o monitor frente al usuario, realizar la lectura del entorno y los *marcadores* con la ayuda de la cámara, desplegar la grabación sobre el monitor y colocar los objetos virtuales sobre ésta.

La segunda consiste en realizar la lectura de los *marcadores* sobre el entorno y desplegar por medio del monitor del dispositivo los objetos virtuales. Ambas técnicas permiten la adición de geometría virtual al mundo real para dar la sensación de que dicha geometría existe en éste.

Una forma simple de AR surgió en las cámaras fotográficas, ya que éstas cuentan con un viewfinder el cual agrega por medio del visor una serie de líneas y curvas que ayudan a enfocar y tomar la fotografía. En la actualidad todos los dispositivos móviles inteligentes cuentan con este tipo de AR en la aplicación de su cámara fotográfica, el cual permite a los usuarios navegar y cambiar la configuración al tomar fotografías.

“Realidad Aumentada (AR) en su más amplia y simple definición es la tecnología que permite la adición de contenido virtual al mundo real”. (Cushnan & EL Habbak, 2013, p.6).

Para poder indicar a la aplicación de AR en dónde se quiere realizar el despliegue de objetos móviles se recomienda el uso de *marcadores* (AR *markers* o *image targets*), los cuales están elaborados de cualquier patrón o geometría que sea reconocible para la cámara del dispositivo. Para esto, el código de la aplicación debe contener al patrón o

geometría a reconocer. Cushnan, y EL Habbak, (2013, p.10), mencionan que simplemente agregando el contenido a este componente y configurando la imagen que necesita rastrear, el contenido AR aparecerá en relación con la imagen rastreable en el mundo real. La figura 1 muestra un ejemplo de *image target* para su uso con AR.

Figura 1. Geometría que forma al target o marcador. Fuente: Elaboración propia.

Gráficos 3D

Un videojuego de cualquier plataforma e incluso una aplicación de AR tiene una jerarquía de elementos llamados *assets*, que son todos aquellos elementos que se necesitan para construir un nivel, un mundo, un escenario o simplemente una escena para el despliegue en AR de un objeto 3D. Watkins (2011), declaró que los *assets* pueden ser muchas cosas: Elementos 2D como interfaces gráficas de usuario y diseños de interfaces, texturas, *modelos 3D*, archivos de audio, clips de animación, también cosas como scripts y otros mecanismos que maneja el juego.

Los modelos u objetos 3D cuentan con una metodología de desarrollo, también conocida como pipeline, la cual va desde plasmar el diseño del *modelo 3D* en papel (boceto) a la aplicación de una textura y la animación de éste. En el desarrollo de este *modelo 3D* se debe hacer uso de buenas prácticas de modelado, texturizado, *rigging* y animado, todo con el fin de obtener un modelo optimizado, capaz de cumplir con todas las características y requisitos para formar parte de la jerarquía de *assets*. La parte más importante de la

optimización se encuentra en el modelado del objeto, pues es aquí donde se determina la cantidad de polígonos a usar para la correcta visualización de éste.

Esta cantidad de polígonos en el objeto determina el tiempo de procesamiento gráfico que el dispositivo tardará en visualizar a dicho objeto.

Watkins (2011), llamó *rendering* al dibujo asociado con los polígonos, las texturas y la iluminación. El *rendering* es el proceso encargado del despliegue visual de cualquier objeto 3D y/o 2D y existen dos tipos de render:

- *Software Rendering*: Este tipo de render es usado en la televisión y el cine para la elaboración de efectos visuales o especiales. La escena es construida dentro de una aplicación 3D, se incluyen las texturas y las luces necesarias para la correcta iluminación de dicha escena. Después, la CPU se encarga del procesamiento de la interacción del objeto con sus colores y luces. Ya que los resultados no son mostrados en tiempo real, no importa si el tiempo de renderizado toma un par de minutos u horas.
- *Hardware Rendering*: En este tipo de render se encuentran los videojuegos y aplicaciones de cualquier tipo, ya que en este caso el *rendering* es en tiempo real, la tarjeta de video y la GPU del dispositivo se encargan de renderizar los polígonos dentro del espacio digital para la representación de un espacio tridimensional. Por lo tanto, el hardware realiza este proceso muchas veces por segundo.

Tomando en cuenta ambos tipos de *rendering*, la optimización en el *modelo 3D* es fundamental para determinar un límite en el uso de recursos de hardware y software.

El proceso de *rendering* es el trabajo principal de un motor de renderizado, para el cual es más fácil realizar los cálculos necesarios si el modelado está construido únicamente con polígonos de tres lados (triángulos), para esto es necesaria la triangulación en la malla del modelado. La figura 2 muestra un *modelo 3D* estilo *low poly* para facilitar el proceso de *rendering*.

Figura 2. Modelado 3D de elefante creado con polígonos de tres lados (triángulos).
Fuente: Elaboración propia.

Experiencia de usuario

La experiencia de usuario se origina a partir de la interacción humano-computadora (HCI, por sus siglas en inglés). Hassenzahl & Tractinsky (2006) mencionan que el término 'experiencia de usuario' se encuentra asociado a una gran variedad de significados, relacionándose con la usabilidad tradicional del software hasta aspectos de belleza, afectivos o experienciales al uso de la tecnología. Se puede creer que la experiencia de usuario tiene como objetivo la usabilidad o productividad del programa o software, sin embargo, su razón de ser es simplemente generar una experiencia en el usuario en el momento que se desee comunicar o transmitir algo. Si bien el objetivo no es la usabilidad, ésta forma parte del desarrollo de la experiencia de usuario, al igual que las interacciones, la accesibilidad, el diseño y todos los principios de éste. Hassan (2017).

Optimización de software

En la informática, cuando se habla de optimización normalmente se piensa en código y en los procesos que se generan dentro del procesador al ejecutar cada línea o sentencia, y este tipo de optimización se aplica mediante algoritmos específicos, funciones

matemáticas, eliminación de código muerto o redundante para simplificar o mejorar el algoritmo usado anteriormente por el programador.
Sin embargo, en la actualidad la optimización de software no está involucrada únicamente con la programación, también se encuentra dentro del Diseño Asistido por Computadora (CAD, por sus siglas en inglés) y funciona similar a su implementación en la programación, pues consiste en la aplicación de técnicas que mejoren el funcionamiento del programa en su ejecución.

García Carrasco (1992), menciona que el objetivo de las técnicas de optimización es mejorar el programa objeto para que nos dé un rendimiento mayor, y dicho objetivo abarca cada una de las fases de desarrollo involucradas en cualquier tipo de software a desarrollar, pues lo que busca el desarrollador es un buen funcionamiento sin afectar los objetivos del software; como la experiencia del usuario antes mencionada, y el usuario busca de manera inconsciente que el programa o aplicación sea fácil de manejar y que su ejecución sea lo más rápida posible. El CAD no ha generado únicamente técnicas específicas para su área, también ha generado el uso de métricas que indiquen un nivel máximo o mínimo de optimización, pues al optimizar intuitivamente se pueden obtener:

- Complicaciones y tiempos largos en el desarrollo del programa al usar técnicas de optimización innecesarias.
- Cuellos de botella dentro del código.
- Afectar negativamente la experiencia del usuario.

Dickinson (2015), concluyó que la evaluación de rendimiento para la mayoría de los productos de software es un proceso muy científico: determinar las métricas máximas de rendimiento admitidas (número de usuarios concurrentes, uso máximo de memoria permitido, uso de CPU, etc.).

La determinación de dichas métricas en parte está relacionada al público que se tiene en mente como objetivo para el videojuego, aplicación o programa, pues así se realiza una

determinación inicial de las limitantes que se tendrán en el desarrollo, de la plataforma, del dispositivo objetivo y de la experiencia de usuario a generar.

Dickinson (2015), menciona que el hecho de que el desarrollo de un videojuego (programa que contiene en su mayoría elementos multimedia como imágenes, clips de audio, clips de vídeo y objetos tridimensionales) sea un proceso muy artístico no significa que no se deba tratar de manera igualmente objetiva y científica.

MARCO HISTÓRICO Y REFERENCIAL

Realidad virtual

Fue gracias a más de cincuenta años de investigaciones, desarrollos y pruebas que en la actualidad se puede hablar de la VR como una de las tecnologías de mayor demanda en distintas áreas de la industria.

El primer caso registrado de VR con un nivel de *inmersión* fue el *sensorama*, una máquina creada por Mort Heilig en 1958 cuyo objetivo era el de permitir al usuario visualizar una de cuatro películas por medio de una pantalla y generar *inmersión* a través de un sistema de sonido estéreo, un sistema de energía eólica y de distintos aromas. Dichos sistemas eran activados durante la pelicula para generar al usuario la sensación de estar en el lugar y tiempo donde ocurrían los hechos.

Pocos años después de la presentación del *sensorama*, Ivan Sutherland desarrolló el primer HMD funcional, el cual impulsó a la VR y a la AR pues éste primer HMD implementaba una pantalla estereoscópica (técnica que crea la ilusión de profundidad al visualizar objetos tridimensionales digitales) y un sistema *head-tracking* (seguimiento de cabeza) mecánico, dichas implementaciones permitían al usuario un mayor grado de *inmersión* y la capacidad de cambiar su *point of view* (punto de vista) para observar todo el entorno virtual.

Puesto que el uso de HMD´s era un tanto incómodo para el usuario por el tamaño, el peso y los grandes brazos metálicos diseñados para sostenerlos, en la década de los 90´s Carolina Cruz-Neira diseñó un sistema de VR similar llamado Cave Automatic Virtual Environment (CAVE) que consiste en un cuarto vacío para la proyección de imágenes de alta resolución por medio de proyectores ubicados detrás de las paredes. CAVE también cuenta con un sistema *head-tracking* que es implementado gracias al uso de unas gafas que permiten la visión estereoscópica al usuario y que cuentan con el *head-tracking* acoplado a ellas para mostrar en tiempo real las imágenes proyectadas en las paredes de acuerdo con el *point of view* del usuario.

Además del sistema *head-tracking* usado en los HMD´s y en el CAVE, se creó un sistema de interacciones por medio de hardware y software. Dicho sistema surgió después de la reducción en tamaño y peso de los HMD´s, el uso de controles con componentes como *joysticks*, *giroscopios*, *touchpads* y *botones* permitieron que el usuario fuera capaz de seleccionar, recoger y mover objetos dentro del entorno virtual. Los controles y el *head-tracking* permitieron también el surgimiento de la *virtual navigation* (navegación virtual) que permite al usuario desplazarse a través del entorno virtual.

Considerando que el CAVE es usado en su mayoría en laboratorios de investigación para la simulación de procesos y/o tareas determinadas, y que los HMD’s son económicamente costosos y requieren de un equipo de cómputo con características específicas, en el año 2014 Google presentó una plataforma de VR para dispositivos móviles llamada *Google Cardboard*.

Cardboard aprovecha los componentes de los smartphones, principalmente el giroscopio para crear un sistema *head-tracking*, sin embargo, se requiere también de un par de lentes biconvexos sujetos a un visor de cartón en el que es colocado el smartphone para la generación de la vista estereoscópica y a su vez, de un nivel de *inmersión*.

Fue ese uso de nuevas tecnologías de procesamiento gráfico y la implementación de hardware especializado lo que permitió la fabricación de dispositivos móviles capaces de reproducir entornos virtuales mediante VR.

Realidad aumentada

No fue mucho el tiempo que pasó para que los dispositivos móviles fueran aprovechados también para el uso de AR y el desarrollo de proyectos como lo fue el proyecto *Tango* de Google, que constaba en la implementación de una plataforma diseñada para su uso con aplicaciones de AR.

“Realidad Aumentada (AR), considerada como una de las tecnologías más perturbadoras de Google, crea un compuesto geométricamente preciso de lo físico y lo virtual, superponiendo el mundo real con información digital e imágenes virtuales”. (Balakrishna, 2013).

Anteriormente la AR solía presentar un problema grave en la interacción del usuario con los objetos tridimensionales, ya que la representación de objetos era limitada por el campo de visión de la cámara. Para controlar dicho problema se generaron *marcadores* (secuencias de gráficos o imágenes) que, al ser captados por la cámara, el dispositivo realizaba la representación de objetos sobre éstos. Una ventaja de *Tango* fue que permitía que los objetos siguieran en su origen aun cuando la cámara no los visualizaba.

“Con la correcta combinación de hardware y software, Tango puede dar a nuestros dispositivos este sentido 3D de capacidades de detección de movimiento, así como la capacidad de reconocer lugares en los que ha estado antes”. (Johnny Lee, Google I/O '17), esta combinación generó el Servicio de Posicionamiento Visual (VPS por sus siglas en inglés) el cual estaba enfocado en el uso de la cámara principal del móvil y en los sensores internos de éste, los cuales fueron el giroscopio, el acelerómetro y principalmente el sensor de profundidad (*Tango* estaba únicamente integrado en el

dispositivo móvil Phab2 Pro de Lenovo y en el ZenFone AR de Asus). El objetivo de *Tango* era tener una mayor *inmersión* con la ayuda de un espacio tridimensional en un área geográfica pequeña, el cual era creado al rotar la cámara en cualquier dirección, dicho espacio podía ser explorado con la ayuda del acelerómetro y el giroscopio que permitían el reconocimiento del desplazamiento del dispositivo (*3D Tracking*).

A principios del año 2018, *Tango* cede en su intento de crear la mejor plataforma de AR gracias al lanzamiento de *ARKit* de Apple, cuya tecnología permite que millones de dispositivos móviles de la marca ejecuten aplicaciones de AR de alta calidad. Como respuesta a *ARKit*, Google lanza el 1 de marzo del mismo año *ARCore*, cuyo propósito es el mismo que el de *Tango*, sin embargo, *ARCore* es capaz de ser ejecutado en una mayor cantidad de dispositivos móviles de distintas marcas, tales cómo; Samsung, Huawei, Motorola, Xiaomi, LG, Sony, entre otras. Aunque actualmente *ARCore* se encuentra en la versión 1.2 y compatible con pocos modelos de las marcas antes mencionadas, ya existen varias aplicaciones y videojuegos que lo implementan.

Análisis de rendimiento (Profiling)

La optimización de software se origina por medio del análisis de rendimiento, pues dicho análisis permite que los ingenieros de software evalúen qué tan bien se desempeña un programa en nuevas arquitecturas o sistemas, también permiten que los programadores analicen sus programas e identifiquen partes críticas de código o que un usuario promedio conozca el desempeño de un sistema.

La primera herramienta de análisis de rendimiento o profiler corresponde a la década de los 70´s, era una herramienta básica llamada *Prof*, la cual enlistaba las funciones y mostraba la cantidad de tiempo de ejecución que usaba un programa. Fue implementado en el sistema operativo Unix. Para la década de los 80´s, se presenta a *Gprof* como una versión mejorada de *Prof* por parte de un grupo del departamento de Ingeniería Eléctrica y Ciencias de la Computación de la Universidad de California, Berkeley, liderado por

Susan L. Graham. *Gprof* tenía como propósito ayudar al usuario a evaluar implementaciones alternativas de abstracciones al mostrar la información de un análisis de rendimiento de una manera más conveniente y eficiente, pues los programas complejos se componen de varias rutinas pequeñas que implementan abstracciones o aislamientos para las rutinas por las que son llamadas, por lo tanto, se creyó que un profiler debería contener el tiempo de ejecución de una manera significativa para la estructura lógica del programa.

Gprof cuenta de cada rutina el tiempo de ejecución usado por esta gracias a las invocaciones de otras rutinas que realiza, el conteo es realizado por el ensamblado de un método conocido como gráfico de llamadas o *call graph* y las rutinas del programa.

Entre 1980 y 1990 se desarrollaron tres clases distintas de herramientas de análisis; la primera clase refiere a herramientas como *Pixie*, *Epoxie* y *QPT* que eran herramientas de conteo básico de bloques, la segunda clase refiere a herramientas como *ATUM* y *MPTRACE* que eran herramientas de seguimiento de direcciones que generaban datos y rastros de instrucción, y la última clase refiere a simuladores como *PROTEUS*, *Tango Lite* y *g88* de Motorola. En 1994, se presentó *ATOM* (Analysis Tools with OM), una plataforma capaz de insertar código en cualquier programa para que éste al ser ejecutado pudiese implementar su propia herramienta de análisis de rendimiento y producir salidas de datos. De esta manera el programa objetivo era capaz de analizarse a sí mismo y mostrar los datos al usuario, dicha acción es conocida como *instrumentación*. La instrumentación permitió que *ATOM* fuera capaz de dar a elegir al usuario qué partes o secciones del programa objetivo quería analizar por medio de las rutinas y así crear un programa personalizado y a su vez un listado de salidas de análisis de rendimiento. *ATOM* usa tecnología de tiempo de enlace para poder organizar a las rutinas de análisis encargadas de la recolección de datos y al programa, de cierta manera que éste no se vea afectado por el análisis de rendimiento mientras se ejecuta. Las rutinas de análisis realizan la transferencia de información desde el programa por medio de llamadas de procedimiento y no de comunicación entre procesos.

El desarrollo de herramientas de análisis como *Gprof* y *ATOM* impulsaron la implementación de herramientas de este tipo en sistemas operativos y en la mayoría de software de desarrollo. Dos ejemplos de herramientas de análisis de rendimiento o profilers en la actualidad son el administrador de tareas en Windows de Microsoft y el monitor de actividad del S.O. de Apple.

Capítulo 1. Análisis y metodología

1.1. PROPUESTA DE PROTOTIPO

El objetivo general del presente trabajo es comprobar la viabilidad de desarrollo de una aplicación móvil con realidad virtual y realidad aumentada utilizando técnicas de optimización para reducir la demanda de recursos de procesamiento sin que se vea afectada significativamente la experiencia de usuario.

Para lograr dicho objetivo se propuso desarrollar dos versiones de una aplicación móvil capaz de mostrar la estructura básica e información del átomo de cada uno de los elementos químicos que conforman la tabla periódica en una escena *inmersiva* de VR y otra de AR. La diferencia entre las versiones radica en que una de éstas implementa técnicas de optimización y buenas prácticas en cada fase de su desarrollo.

Para comprobar la viabilidad de desarrollo se medirá la demanda de procesamiento de cada versión y se aplicará una prueba de usabilidad a estudiantes del Centro Universitario UAEM Ecatepec.

1.2. IDENTIFICACIÓN DE REQUERIMIENTOS

La comparación de los modelos de desarrollo requiere de la correcta medición de las causas de demanda de procesamiento, las cuales pueden ser; la representación de objetos tridimensionales, la iluminación y sombreado, texturas, animaciones y audio.

Las fases relacionadas a las causas mencionadas anteriormente implementan un análisis y un desarrollo independiente, por ejemplo; se implementa un software distinto para la creación de objetos tridimensionales y para la creación de texturas. Ambos modelos de desarrollo requieren de un flujo de trabajo que implemente el análisis, el diseño y el

desarrollo, sin embargo, uno de ellos requiere cubrir necesidades relacionadas a técnicas de optimización y buenas prácticas; tales como los límites de optimización para no afectar la experiencia de usuario y la generación de código óptimo que permita el mejoramiento de las interacciones sin afectar de manera negativa su ejecución.

La comparación también exige de la evaluación por parte del usuario, por lo tanto, para la generación de dicha comparación se requiere de un instrumento que apoye a medir la experiencia del usuario.

1.3. ESPECIFICACIÓN DE REQUERIMIENTOS

Se definió una lista de requerimientos del presente trabajo y se determinó si éstos pueden contribuir al buen desarrollo de las versiones del caso práctico. Dichos requerimientos están relacionados a los objetivos específicos del presente trabajo.

Tabla 1. Requerimientos funcionales y no funcionales.

Objetivo:	Comprobar la viabilidad de desarrollo de una aplicación móvil con realidad virtual y realidad aumentada utilizando técnicas de optimización para reducir la demanda de recursos de procesamiento sin que se vea afectada significativamente la experiencia de usuario.

Requerimientos

No.	Descripción	Funcional	No funcional
1	Desarrollar elementos multimedia óptimos para su despliegue en dispositivos móviles.	x	
2	Crear *modelos 3D* estilizados con una cantidad baja de polígonos.	x	
3	Medir de la demanda de procesamiento de ambas versiones de la aplicación por medio de un software de análisis de rendimiento especializado.	x	
4	Delimitar niveles de optimización en cada fase de uno de los modelos de desarrollo.	x	
5	Software de pago para el desarrollo de *modelos 3D*, texturas, imágenes y la creación de las aplicaciones.	x	
6	Generar un instrumento para medir la percepción del usuario al usar ambas versiones de la aplicación.	x	
7	Implementar el instrumento a estudiantes de ingeniería en computación e informática administrativa del C.U. UAEM Ecatepec.	x	

Fuente: Elaboración propia.

1.4. ANÁLISIS DE RIESGOS

Tabla 2. Análisis de riesgos.

Riesgo	Descripción	Tipo	Plan de respuesta
Medición errónea de recursos de hardware.	Medir o leer erróneamente la demanda de recursos de hardware por parte de la aplicación en su tiempo de ejecución.	Proyecto.	Comprobar los datos medidos varias veces durante el tiempo de ejecución de la aplicación.
Implementación del instrumento para medir la experiencia de usuario después del tiempo establecido.	Debido a que dicha actividad requiere de una gran cantidad de tiempo y será llevada a cabo con estudiantes del C.U. UAEM Ecatepec, se toma en cuenta la estadía de éstos en el plantel durante el semestre.	Proyecto.	Ajustar el flujo de trabajo, haciendo de este uno más estricto en cuestión de tiempo.
Mal uso y generación de daños a las gafas de realidad virtual durante el uso de la aplicación.	Para la aplicación del presente caso práctico se cuenta con gafas de realidad virtual Cardboard, las cuales están fabricadas en su mayoría con cartón.	Proyecto.	Capacitar al usuario antes del uso de la aplicación para el correcto manejo de las gafas y reparar el daño, o si es el caso, comprar otras gafas de realidad virtual.
Generación de errores en el análisis estadístico por parte del usuario.	El usuario respondió de manera errónea el instrumento para medir la experiencia de usuario, generando errores en la medición o la obtención de una medición coherente.	Proyecto.	Volver a implementar el instrumento, asegurándose de que la estructura de las preguntas guiará al usuario a generar un buen análisis estadístico.
Actualizaciones de software que afecten el flujo de desarrollo de la aplicación.	El desarrollo se ve afectado negativamente por cambios en el software utilizado.	Técnico.	Instalar versiones anteriores del software o deshacer los cambios en éste.
Presencia de errores de programación o diseño en una o ambas versiones de la aplicación.	Durante el uso de la aplicación, resaltan interacciones mal programadas, problemas en el diseño de la interfaz gráfica o en el diseño de las escenas de VR y AR.	Técnico.	Probar ambas versiones de la aplicación en cada fase

Fuente: Elaboración propia.

1.5. DESARROLLO EVOLUTIVO

La metodología de desarrollo evolutivo de escenarios tridimensionales (Morales, et al, 2015) mostrada en la figura 3, permite el desarrollo de productos intermedios completamente operacionales, los cuales se pueden proponer en consideración del cliente o usuario final con el fin de mejorar en cada iteración. Dicha metodología consta de cinco fases; Análisis, Diseño, Desarrollo, Evaluación y Cierre.

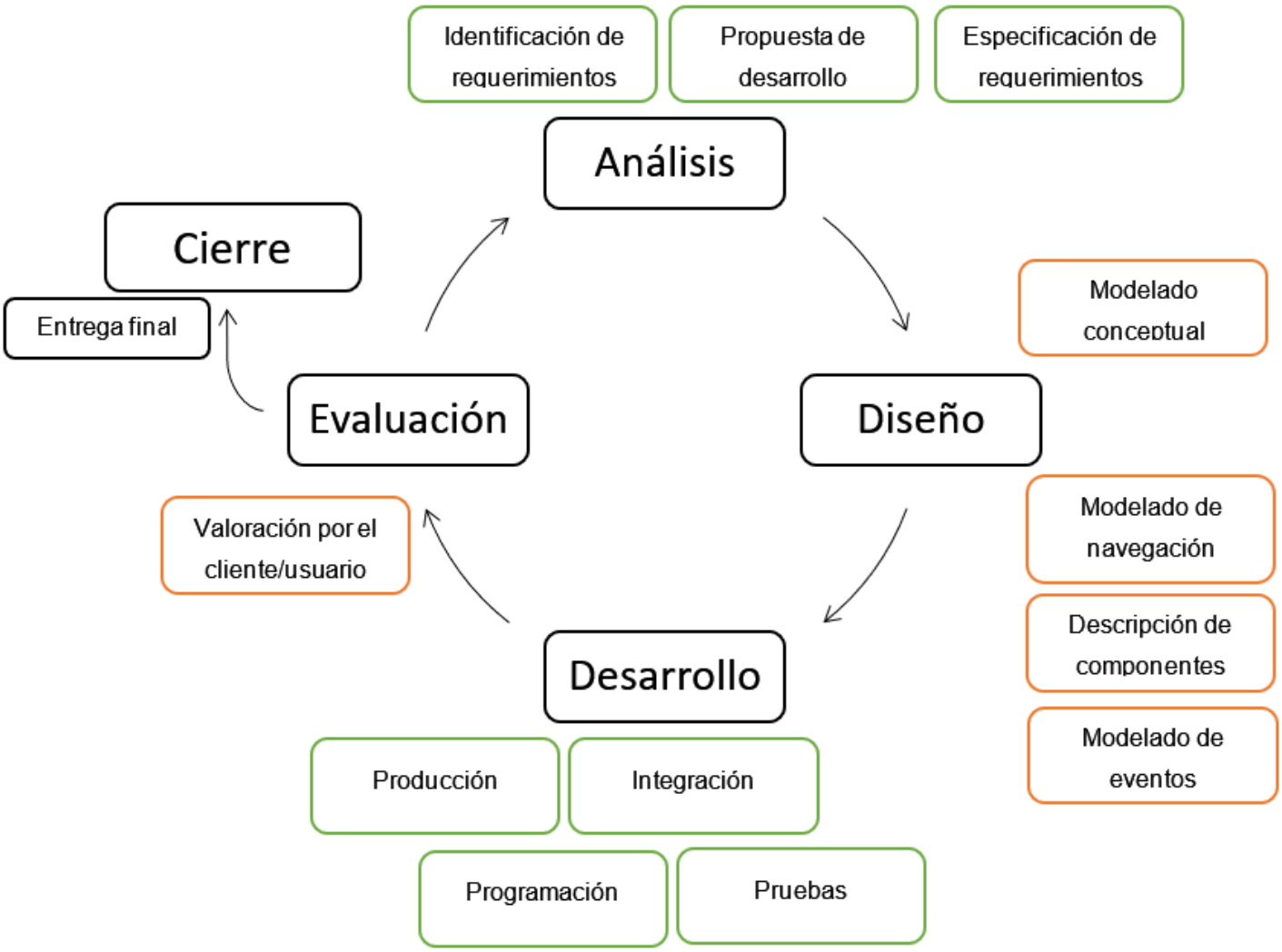


Figura 3. Metodología de desarrollo evolutivo de escenarios tridimensionales. Fuente: Elaboración propia.

1.5.1. FASE DE ANÁLISIS

Dentro de esta fase se definen las necesidades del cliente por medio de tres actividades; identificación de requerimientos para saber qué necesidades deberán ser cubiertas en el desarrollo, una propuesta de desarrollo para proponer un plan de desarrollo y cubrir las necesidades establecidas, y una especificación de requerimientos que consiste en crear la documentación sobre la información obtenida.

1.5.2. FASE DE EVALUACIÓN

En esta fase el cliente o usuario final se encarga de valorar la aplicación con base en los requerimientos establecidos en la fase de análisis. La versión por valorar es entregada una vez se considera que todos los componentes desarrollados han sido integrados de manera adecuada en la aplicación.

1.5.3. FASE DE CIERRE

Se entrega la versión final de la aplicación y ésta se encuentra lista para ser implementada en un ambiente con condiciones reales y usuarios.

1.6. GENERAL GAME PRODUCTION PIPELINE

El presente caso práctico fue considerado como un proyecto único, o no estándar, debido a su objetivo y a las técnicas a aplicar durante todo el proceso de desarrollo, por lo tanto, se reemplazaron las fases de diseño y desarrollo de la metodología antes mencionada con el fin de tener un mejor flujo de trabajo en cada fase.

Las fases de diseño y desarrollo fueron remplazadas con la metodología conocida como *General Game Production Pipeline*, la cual básicamente es un concepto de flujo de trabajo para su uso en el proceso de desarrollo de un videojuego o aplicación con alto

contenido multimedia. Las tareas dentro de ésta necesitan ser completadas hasta el lanzamiento. La figura 4 muestra que dicha metodología consta de seis fases; Concept phase, 3D content creation pipeline, Level design, Lighting, Implementation y Release.

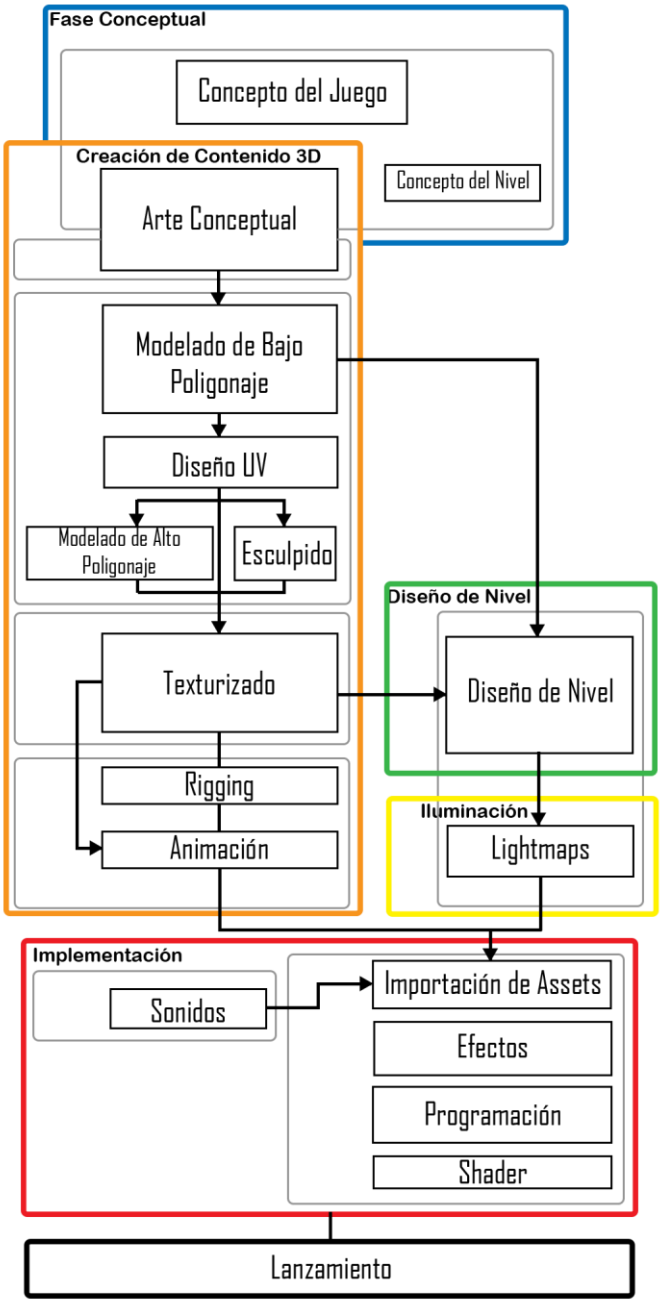


Figura 4. Fases del Game production pipeline. Fuente:(Labschütz, et al, 2011).

Algunas de las tareas requieren ser elaboradas de manera secuencial y otras como Level design y 3D content creation pipeline de manera paralela, ya que los equipos de desarrollo de cada fase necesitan de elementos y/o características para desarrollar el trabajo.

1.6.1. FASE CONCEPTUAL (CONCEPT PHASE)

Durante esta fase, un equipo de ilustradores dibuja los elementos y/o personajes, los escenarios, los mecanismos del juego y el mundo en el que se ambientara al usuario. En esta fase también se proponen las posibles texturas y colores de los elementos que conformaran las escenas.

1.6.2. CREACIÓN DE CONTENIDO 3D (3D CONTENT CREATION PIPELINE)

Esta fase contiene las etapas por las que cada modelado 3D tiene que pasar, y van desde el concept art hasta el modelado final.

1.6.3. ARTE CONCEPTUAL (CONCEPT ART)

Concept Art es la fase principal de un modelado 3D, es aquí donde se especifican las características físicas del objeto o personaje a modelar, éste es dibujado desde varios ángulos o vistas, las cuales son: frontal, lateral derecha, lateral izquierda, trasera y por encima. Después de los bocetos iniciales, los artistas importan el boceto final a un editor gráfico para obtener un archivo digital de éste. En el archivo digital se plasman los colores sugeridos para el objeto o personaje en cada una de las vistas.

1.6.4. MODELADO DE BAJO POLIGONAJE (LOW POLYGON MODELING)

Para crear un modelado 3D basado en el concept art, primero es recomendado realizar un modelado con una cantidad alta de polígonos y después definir un nivel de detalle

para proceder a la optimización del objeto o personaje sin perder calidad. Dado que los motores de videojuegos pueden trabajar con triángulos y cuadrados no existe alguna restricción para usar una técnica de modelado en particular.

Para los objetos con un alto nivel de detalle es necesario determinar una cantidad máxima de polígonos, esto con el fin de optimizar cada modelado 3D y de mantener el nivel de detalle deseado.

1.6.5. DISEÑO UV (UV LAYOUT)

En algunos modelados 3D se requieren distintas texturas o colores, estos pueden presentar una mayor cantidad de almacenamiento en el dispositivo al generar un archivo para cada color o textura. Una manera de optimizar el proceso de modelado por medio de texturas es con el uso de *UV Layouts*, los cuales consisten en el uso de una única textura que es capaz de contener a varias de éstas. Un *UV Layout* sirve como plantilla para la adición de colores y/o texturas, en la figura 5 se observa el *UV Layout* resultante de un *modelo 3D*.

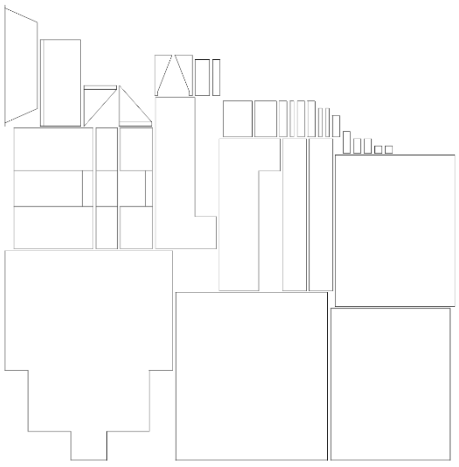


Figura 5. UV Layout del modelado 3D de una casa. Fuente: Elaboración propia.

1.6.6. MODELADO DE ALTO POLIGONAJE (HIGH POLYGON MODELING)

Después de obtener *UV Layouts*, algunos modelados 3D con cantidad baja de polígonos son mejorados o aumentados a alta resolución, esto se logra al agregar subdivisiones de polígonos para mejorar la calidad del modelado. Este proceso es realizado normalmente para la realización de un render o la obtención de un normal map, el cual ayuda a agregar detalles al modelado sin agregar más polígonos.

1.6.7. TEXTURIZADO (TEXTURING)

El proceso de texturizado consiste en la aplicación de colores y texturas al *modelo 3D* con base en el arte conceptual de éste, dicho proceso puede ser logrado con la implementación de *UV Layout*. Para la elaboración de las texturas los artistas suelen contar con una tabla que contiene distintas combinaciones de colores y texturas como madera, superficies rugosas o determinados patrones.

1.6.8. RIGGING Y ANIMACIÓN (RIGGING AND ANIMATION)

Para crear animaciones que luzcan naturales, una técnica llamada *rigging* crea una estructura ósea virtual para cada modelado 3D. Esta estructura puede ser usada para controlar los movimientos y obtener una animación natural. La estructura ósea es un conjunto de uniones (joints) que simulan las articulaciones. Para obtener una serie de movimientos naturales es necesario un buen radio de interacción de parte de las uniones con la geometría del modelado, esto se conoce como ‘Skin Weights’ (figura 6).

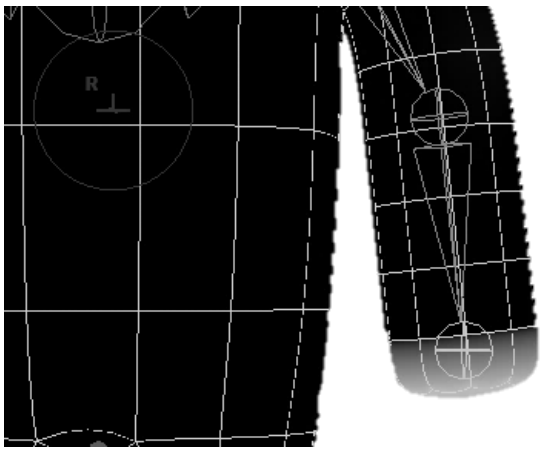


Figura 6. Estructura ósea para la elaboración de animaciones. Fuente: Elaboración propia.

1.6.9. DISEÑO DE NIVEL (LEVEL DESIGN)

Esta fase consta del diseño de cada uno de los niveles del videojuego o aplicación, para esto es necesario contar con todos los elementos conocidos como *assets* para la creación de niveles. En esta fase se especifica el cómo y por dónde un usuario realizará una tarea, también en qué momentos se ejecutarán ciertas mecánicas del videojuego o aplicación.

1.6.10. ILUMINACIÓN (LIGHTING)

Algo que da realismo a los elementos dentro de un motor de videojuegos son las luces, estas determinan el ambiente del nivel y ambientan al usuario dentro de la aplicación.
Sin embargo, el uso de demasiadas luces y el cambio de ángulo al desplazar el personaje o al usuario requieren que el software haga el cálculo para renderizar la escena varias veces por segundo, lo cual puede optimizarse al definir una cantidad máxima de luces en la escena y el uso de técnicas como ‘Baking light’ para crear texturas que contengan luces y sombras.

1.6.11. IMPLEMENTACIÓN (IMPLEMENTATION)

Esta es la fase final en la elaboración del videojuego o aplicación, aquí se importan todos los *assets* a un motor de videojuegos y se elabora la escena. Para este proceso, es importante crear una jerarquía de *assets* y organizar a éstos por tipo de archivo. En la elaboración de la escena se colocan los modelados 3D, las luces, los efectos visuales, los clips de animación, los archivos de audio y se programan las mecánicas del videojuego o aplicación, así es creado el proceso de producción.

La figura 7 muestra la implementación de *assets* dentro de una escena en Unity, la cual consiste en un conjunto de *assets* de árboles.

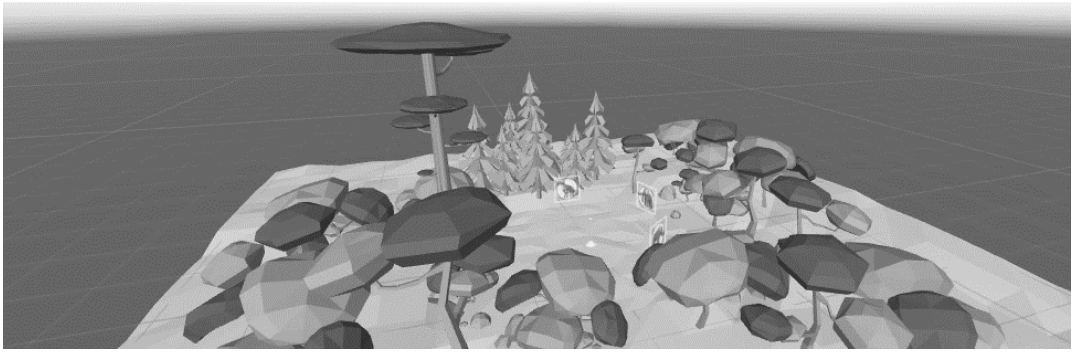


Figura 7. Implementación de *assets* para la elaboración de un nivel en Unity. Fuente: Elaboración propia.

1.6.12. LANZAMIENTO (RELEASE)

El resultado final del proceso de producción es el prototipo final del videojuego o aplicación y éste es lanzado para su uso.

Capítulo 2. Técnicas de optimización

Para la comprobación de la hipótesis mencionada anteriormente se escogió como presente caso práctico el desarrollo de una aplicación capaz de mostrar un entorno de realidad virtual *inmersivo* e implementar realidad aumentada para mostrar al usuario los átomos de los distintos elementos químicos conocidos, tomando en cuenta la calidad visual y la programación que una aplicación de este tipo requiere en su desarrollo. A continuación, se describen las técnicas consideradas óptimas en el desarrollo del presente caso práctico, y los componentes clave para generar la experiencia de usuario deseada.

2.1. MODELADO 3D

El uso de *modelos 3D* en dispositivos móviles se ha limitado a cantidades de polígonos relativamente bajas, ya que la mayoría de éstos no cuenta con chips de procesamiento gráfico capaces de renderizar altas cantidades sin disminuir gravemente el desempeño. Existen distintas técnicas y prácticas de modelado 3D para tener una cantidad óptima de polígonos sin perder calidad visual, éstas van desde determinar una cantidad mínima y máxima de polígonos por cada modelado hasta la creación de curvas con pocas líneas rectas. Comúnmente en el proceso de modelado se realizan *modelos 3D* con una cantidad alta de polígonos (*high polygon modeling*) y posterior a esto se realiza una optimización en los modelos sin perder calidad visual (*low polygon modeling*), sin embargo, existen casos donde el proceso es el inverso, se empieza por *modelos 3D low poly* y después de otros procesos se realiza una aumentación de polígonos en la que se busca mejorar el aspecto del modelo.

2.1.1. Low Poly

Low poly, puede referirse tanto a la técnica de optimización como a un estilo de modelado que se relaciona más con el aspecto del modelo, ambas contienen una cantidad relativamente baja de polígonos, tomando en cuenta el objeto que se está modelando (figura 8).

Figura 8. Modelo 3D low poly. Fuente: Elaboración propia.

En el proceso de modelado suelen aplicarse operaciones booleanas de *unión*, *diferencia* o *intersección* entre objetos tridimensionales, dependiendo de cómo se hayan colocado los objetos a los que se aplicó la operación, el resultado puede generar una cantidad mayor de polígonos (figura 9).

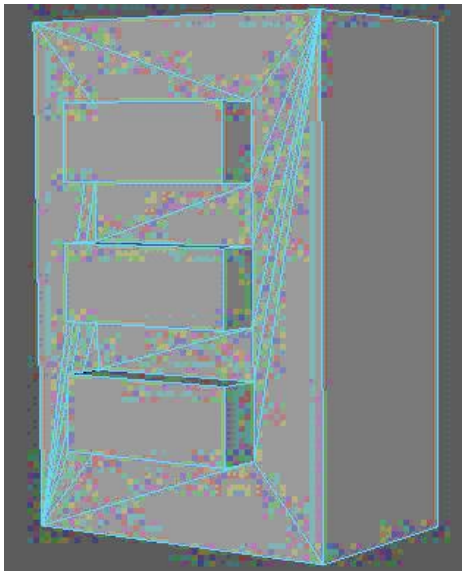


Figura 9. Resultado de la aplicación booleana Unión. Fuente: Elaboración propia.

Hay otra técnica que permite disminuir la cantidad de polígonos, pero a diferencia de la anterior, trabaja sobre polígonos que serán visibles por el usuario, consiste en seleccionar varios polígonos (generados por posibles operaciones *booleanas*) y eliminarlos, para la creación de uno o dos polígonos que ocupen el área de los anteriores sin interactuar de manera directa con otros polígonos que estén sobre ésta (figura 10).

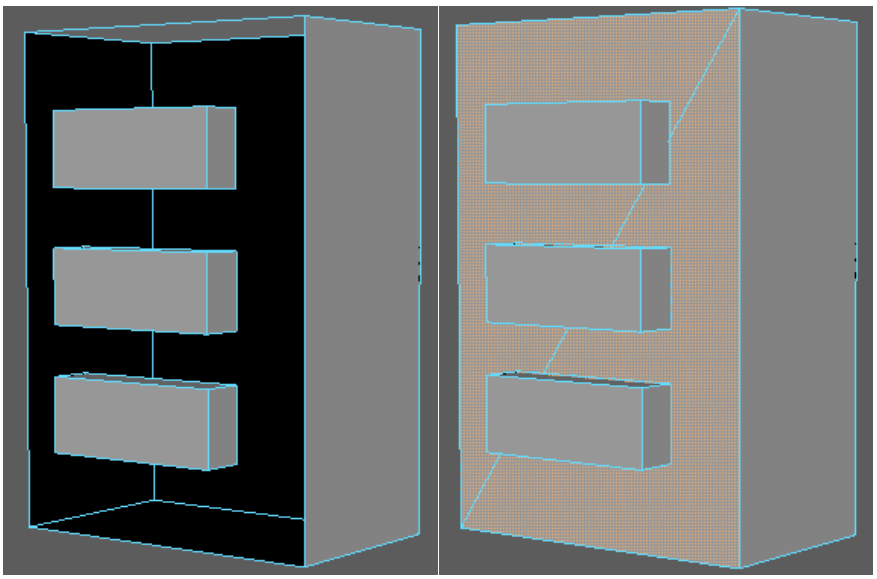


Figura 10. Eliminación de los polígonos resultantes de la operación booleana y sustitución de éstos. Fuente: Elaboración propia.

A veces al aplicar *booleanos* se generan irregularidades en los polígonos que conforman al objeto, si no se es cuidadoso en el proceso de optimización, se pueden tener polígonos formados por *vértices* tan cercanos que parecen a simple vista ser uno solo, la figura 11 muestra lo que parece ser la intersección de dos líneas rectas formando un ángulo de 90°, sin embargo, la figura 12 muestra de un punto más cercano la misma área, y se aprecia que existen dos pares de líneas perpendiculares unidas por un par de *vértices*.

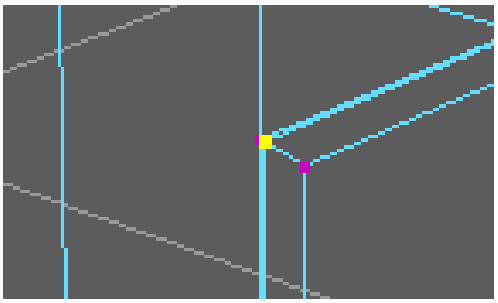


Figura 11. El punto amarillo representa lo que parece ser, la unión de dos líneas rectas. Fuente: Elaboración propia.

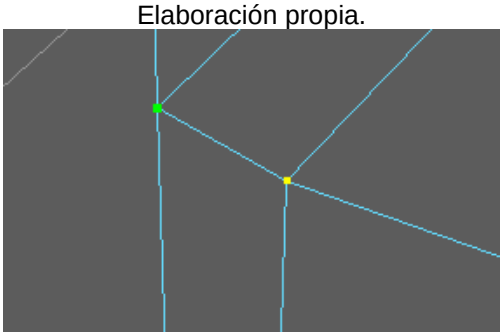


Figura 12. Una toma más cercana muestra que, lo que parecía ser un vértice, en realidad son dos. Fuente: Elaboración propia.

La problemática anterior tiene dos soluciones (en ambos casos se busca eliminar el o los polígonos sobrantes generados por los *vértices*), la primera es seleccionar las líneas rectas o *vértices* sobrantes y eliminarlas adecuadamente, y la segunda es combinar ambos *vértices* para tener un solo conjunto de líneas rectas y *vértices*, lo último se puede lograr haciendo uso de herramientas como *Target Weld* o *Merge* de Autodesk Maya.

2.2. CREACIÓN DE TEXTURAS

Las texturas son parte fundamental del *modelo 3D*, ya que son las encargadas de dar mayor realismo con el uso de colores y técnicas de mapeado que se enfocan en detallar las superficies de éstos con iluminación y relieves. También se encargan del atractivo visual de la aplicación o videojuego, una buena implementación de luces y texturas hacen atractivo y realista a cualquier *modelo 3D*. Existen distintas técnicas para texturizar un *modelo 3D*, sin embargo, no todas son óptimas para su implementación en dispositivos con bajos recursos de hardware o con poco almacenamiento.

2.2.1. UV Layout

UV Layout es un sistema coordenado de dos dimensiones que facilita el uso de texturas en las superficies de los *modelos 3D*. El resultado de la implementación de esta técnica es una imagen (figura 13) que contiene todas las caras del *modelo 3D* (Como si éste fuera un objeto bidimensional).

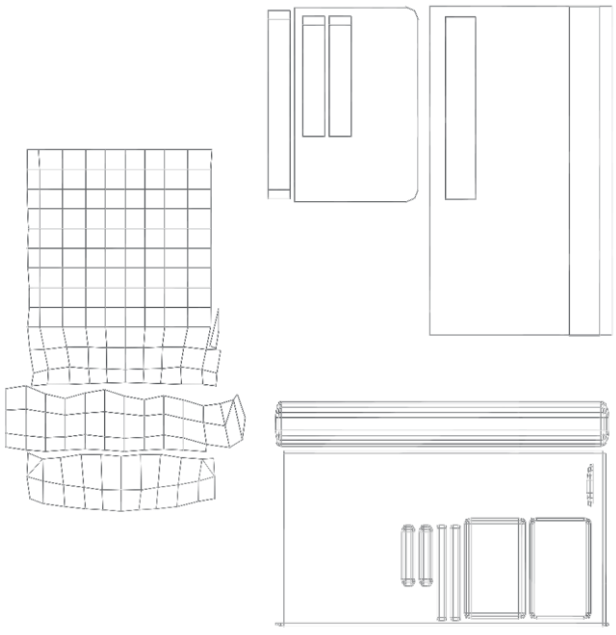
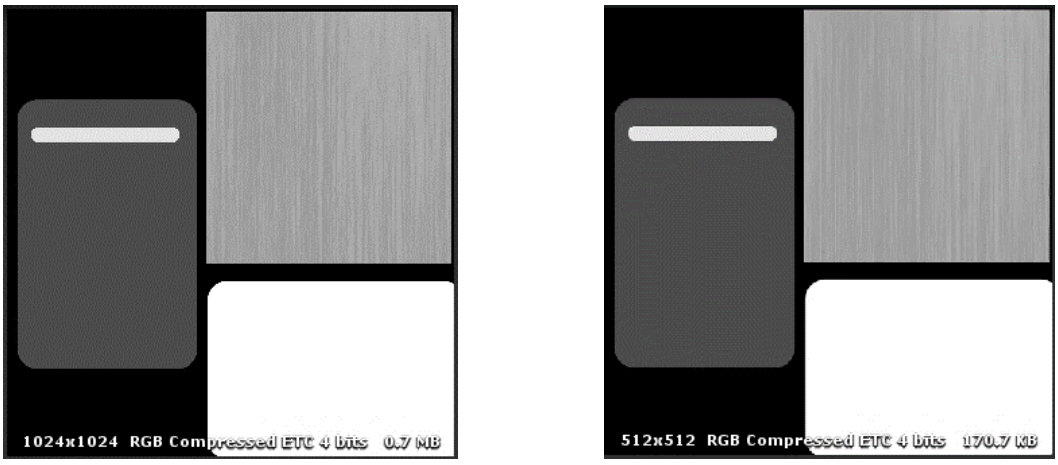


Figura 13. UV Layout de modelos 3D que conforman la cama en la escena de realidad virtual. Fuente: Elaboración propia.

Cada *modelo 3D* cuenta con su propio *UV Layout*, el cual es utilizado para facilitar la creación de la textura a implementar sobre la superficie de éstos. Sin embargo, la realización de varias texturas, por más simples que sean, generan un mayor almacenamiento por parte de la aplicación una vez instalada en el dispositivo. Para reducir la cantidad de almacenamiento, se puede combinar a varios objetos tridimensionales para la elaboración de una menor cantidad de texturas y, así implementar un tipo de *batching*. Por ejemplo: si se tienen dos objetos, al combinarlos, éstos podrían manipularse como uno solo y el *UV Layout* contendría las caras de ambos.

2.3. CREACIÓN DE ASSETS Y DISEÑO DEL NIVEL EN UNITY

Unity es una plataforma de desarrollo de videojuegos en 2D, 3D, VR y AR, es el software de creación de videojuegos más usado en el mundo y tiene soporte para plataformas como Android, iOS, Xbox One, Play Station, Oculus, entre otras. Unity permite optimizar elementos multimedia reduciendo su cantidad de almacenamiento, en el caso de las imágenes, el *inspector* de Unity permite modificar el tipo de imagen y escoger el tamaño máximo de ésta, de esta manera se pueden optimizar imágenes que no necesitan de un alto nivel de detalle, ya que en algunos casos no se perciben a simple vista detalles dentro de las imágenes (figura 14).



0.7 MB 170.7 KB

Figura 14. Optimización de texturas por medio del inspector de Unity. Fuente: Elaboración propia.

2.3.1. Procesamiento por lotes (Batching)

Batching es un término que describe la agrupación de una alta cantidad de datos para poder procesarlos como uno solo. El objetivo del *batching* es reducir el tiempo de computación por parte del programa o aplicación, logrando lo anterior con la reducción del número de *draw calls* (llamadas de dibujo) requeridas para renderizar los objetos.

En cada *frame* que es visualizado por el usuario se generan *draw calls* o llamadas de dibujo, éstas almacenan los datos de cada objeto que será renderizado por la cámara y generan peticiones desde la CPU a la GPU para dibujar o renderizar el objeto. La CPU genera paquetes de datos llamados *batches* por cada *draw call*, cada *batch* puede o no contener los datos de las *draw calls*. Por lo tanto, se conoce que una cantidad elevada de *batches* suponen un alto procesamiento por parte de la CPU y podrían significar también un bajo desempeño de la aplicación, sin embargo, la cantidad de *batches* que produce un conjunto de elementos al ser renderizados por la cámara nos da a entender también que pueden actuar como una métrica para determinar un nivel de optimización.

La ventana *Statistics* en Unity permite conocer la cantidad de batches y otras métricas de rendimiento (figura 15).

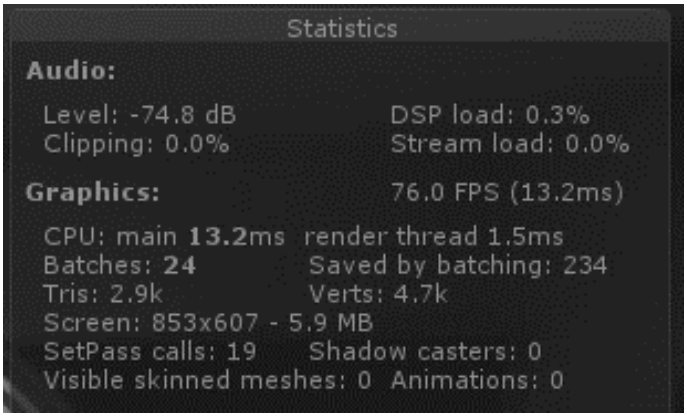


Figura 15. Ventana Statistics, la cual permite conocer métricas de rendimiento. Fuente: Captura de pantalla de Unity.

Unity cuenta con dos mecanismos para la reducción de *draw calls* como una herramienta de optimización para la aplicación que se desarrolle: *Dynamic Batching* (procesamiento por lotes dinámico) y *Static Batching* (procesamiento por lotes estático).

Dynamic Batching tiene como objetivo agrupar grandes cantidades de mallas de distintos *modelos 3D* y procesarlos como si fueran uno solo, sin embargo, este mecanismo funciona únicamente para los objetos que son visibles para la cámara, lo que hace que el agrupamiento de las mallas varíe en cada *frame* si se renderizan *modelos 3D* distintos o se miran desde otro ángulo. Es por lo anterior que se le conoce como procesamiento por lotes dinámico y, por lo tanto, dicho procesamiento se realiza en el tiempo de ejecución de la aplicación.

El objetivo del *Static Batching* es el mismo que el de su similar dinámico, la diferencia con el procesamiento por lotes dinámico es que el mecanismo estático realiza el proceso con mallas de *modelos 3D* no idénticas o de cualquier tamaño (lo que para el procesamiento dinámico es un criterio) y el procesamiento es pre-calculado (el procesamiento se realiza en la inicialización de la aplicación y no en el tiempo de ejecución). El uso de *Static Batching* es en la mayoría de los casos más eficiente que *Dynamic Batching* por reducir aún más las *draw calls*, sin embargo, usa más memoria. La cantidad de memoria demandada por el mecanismo varía con respecto a las características de las mallas, pues todas las mallas declaradas como estáticas (figura 16) se combinan para formar una malla grande y pasar al sistema de renderizado como una única *draw call*.

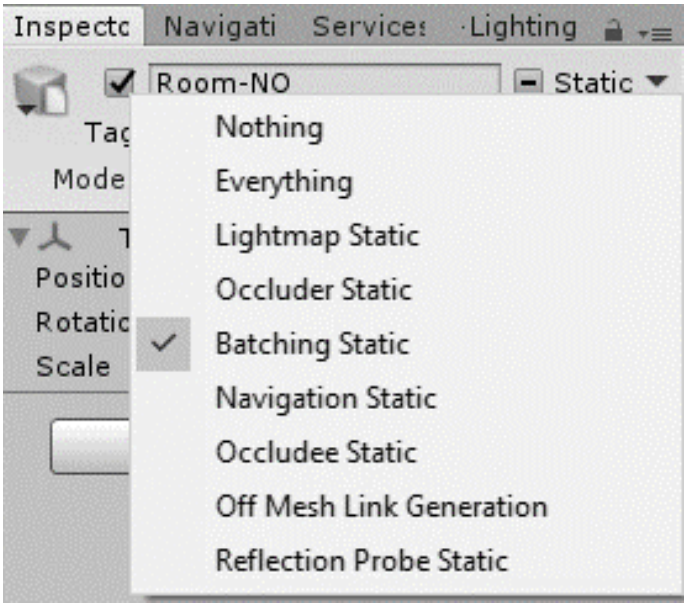


Figura 16. Marcando a un GameObject como estático para el uso de Static Batching. Fuente: Captura de pantalla de Unity.

2.3.2. Sistemas de partículas

Los sistemas de partículas de Unity son componentes que ayudan a representar líquidos, nubes, llamas, humo (sustancias o elementos no sólidos), también permiten la elaboración de efectos visuales específicos que generan la sensación de objetos luminosos y/o fluidos por medio de imágenes que siempre apuntan su cara visible hacia la cámara, creando la sensación de que es un objeto tridimensional.

Es importante conocer cómo trabaja el sistema de partículas, pues cada elemento que lo constituye colabora a la demanda de recursos de hardware en el tiempo de ejecución. Mencionado eso, se debe tener en mente que no importa demasiado la cantidad de sistemas de partículas usados o ejecutados en una escena, si no, la cantidad de elementos o partículas que cada uno de éstos emiten, el tiempo de emisión, el tiempo de vida de la partícula y si su emisión se encuentra dentro de un *bucle* o *loop* (figura 17).

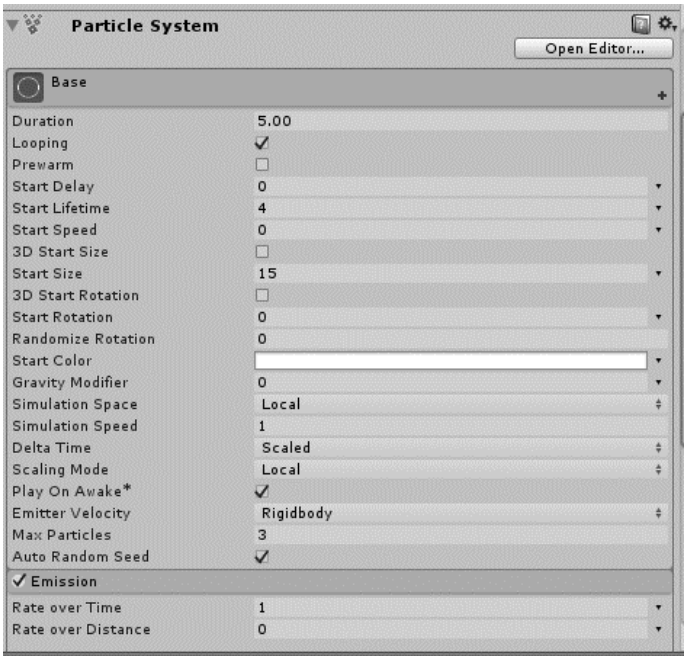


Figura 17. Configuración del sistema de partículas de Unity. Fuente: Captura de pantalla de Unity.

2.3.3. Colisionadores (Colliders)

Los *colliders*, son componentes que permiten a un objeto ser sólido y generar una colisión al interactuar de manera física. Sin embargo, los *colliders* pueden no comportarse como objetos sólidos y tener la capacidad de detectar cuando entran o salen del espacio de otro sin crear una colisión, es en este momento en el que son nombrados *triggers*. Cuando un *trigger* detecta la entrada o salida de un *collider* ejecuta una tarea previamente programada.

Existe una forma de detectar a un *collider* sin necesidad de provocar una colisión o la entrada y salida de éste al espacio de otro, se trata del uso de *rays* o rayos generados por la cámara que ocupan un espacio dentro de la escena y su uso más común es la generación de *raycast*. El *raycast* envía un rayo con el mismo origen del generado desde la cámara y termina al detectar o “golpear” un *collider* dentro de la escena, la detección genera un paquete de datos que permite conocer a qué *collider* se ha golpeado, así el desarrollador puede diferenciar entre *colliders* y asignar tareas específicas a cada grupo de éstos.

2.3.4. Cuadro y barra de desplazamiento (Scroll Rect and Scroll Bar)

El componente *Scroll Rect* (figura 18) es capaz de mostrar contenido que ocupa demasiado espacio en un área pequeña y *Scroll Bar* permite desplazar ese contenido por el área para verlo completamente.

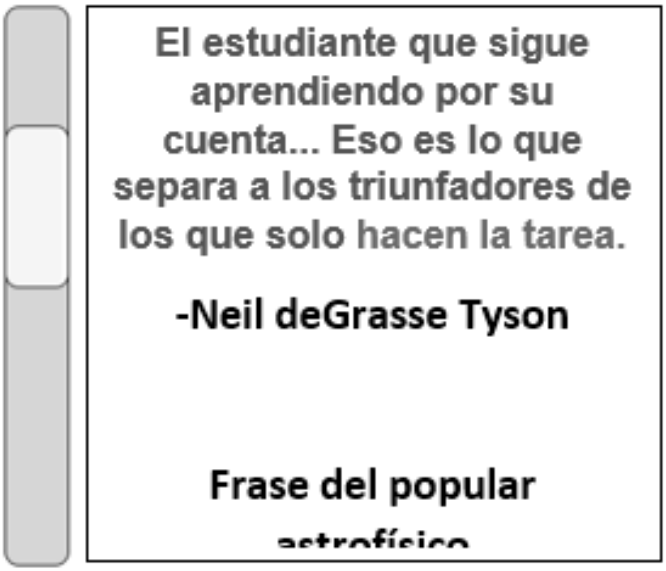


Figura 18. Ejemplo del uso del *Scroll Rect* (derecha) y del uso del *scroll bar* (izquierda). Fuente: Elaboración propia.

2.3.5. Botón

Button (figura 19) permite que una imagen sea interactiva y que inicie o confirme un evento al detectar un toque sobre el área de la imagen por parte del usuario. Los eventos son funciones o partes de código destinados a realizar una acción sobre un objeto dentro de la escena y se gestionan gracias al componente *EventSystem* o sistema de eventos, que es agregado por Unity al crear un *canvas*. Los eventos son declarados dentro del componente *Button* por la clase *OnClick* y es necesario únicamente dar a conocer el script contenedor de la función o acción a ejecutar.

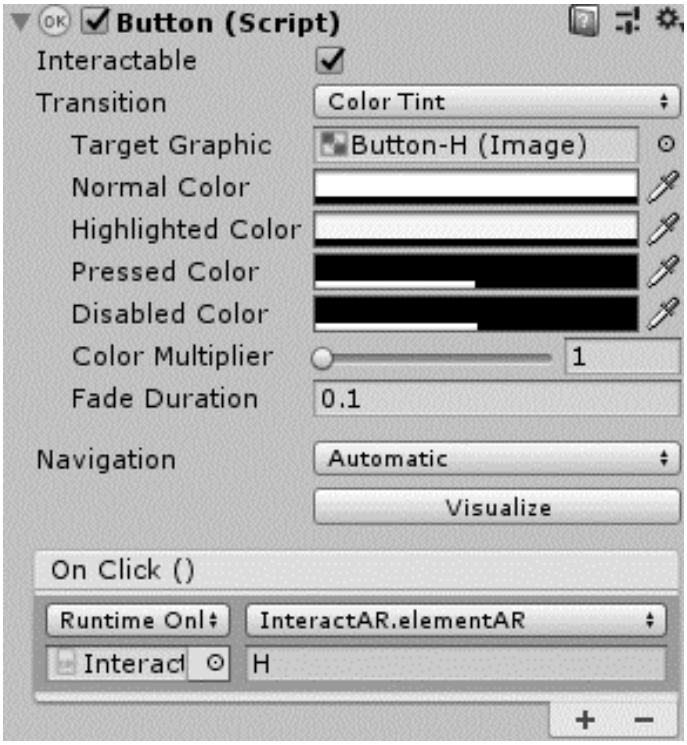


Figura 19. Componente Button donde se aprecia la clase OnClick. Fuente: Captura de pantalla de Unity.

2.4. ILUMINACIÓN

Si bien el uso de luces dentro de una escena involucra en su mayoría un sombreado, la iluminación global calculada en tiempo de ejecución puede afectar el desempeño de todas las partes gráficas dentro de la aplicación. La iluminación es quizás una de las partes más importantes del arte de la aplicación, su diseño o su atractivo visual, y hacen que una escena tome realismo o un aspecto colorido y bello.

La iluminación global (GI, por sus siglas en inglés), es un sistema dentro de Unity que permite una luz indirecta al desplazar la luz de una superficie a otra. GI genera un nivel de realismo distinto en una escena con un renderizado de luces dinámico, sin embargo, su uso requiere de cálculos en tiempo real. Para evitar la demanda de recursos por parte de la GI, Unity ofrece una técnica llamada *baked lighting* (horneado de luces) a través de *lightmaps*. *Lightmapping* es un proceso que forma parte del motor de renderizado de Unity y su función es pre-calcular el brillo sobre superficies determinadas dentro de una escena.

Dicho proceso genera *lightmaps*, los cuales pueden definirse como un conjunto de texturas que almacenan la información de las luces reflejadas y las sombras proyectadas sobre las superficies. Los *lightmaps* contienen también y en gran parte la información de los materiales, las texturas y las mallas de los *modelos 3D* poseedores de las superficies que interactúan con las luces.

El uso de *baked lighting* significa una disminución importante en el procesamiento de iluminación y lo que ésta involucra en el tiempo de ejecución de la aplicación, pues las luces y sombras son parte de las texturas y no procesadas al abrir la aplicación, escena o sección. La gran ventaja es la disminución en la demanda de recursos, sin embargo, el tamaño de memoria ocupado por contenido multimedia dentro de la aplicación y del dispositivo en el que ésta fue o será instalada aumenta en relación con el tamaño y resolución de los *lightmaps*.

2.4.1. Baked lighting y lightmapping como técnicas de optimización en VR

Para generar los *lightmaps* mediante *lightmapping* es necesario realizar algunos cambios en la configuración de iluminación, en las luces y en los *modelos 3D* que conforman la escena.

La ventana *Lighting* (figura 20) permite personalizar la configuración de la iluminación de la escena actual, y así establecer si se desea o no una iluminación en tiempo real. Es necesario mencionar que la configuración que se muestra en la ventana dependerá del modo de iluminación seleccionado, ya sea *Realtime Global Illumination* o *Baked Global Illumination*. El presente trabajo se enfoca en los parámetros más importantes del modo de iluminación conocido como *Baked Global Illumination* para la generación de un mejor desempeño en el tiempo de ejecución.

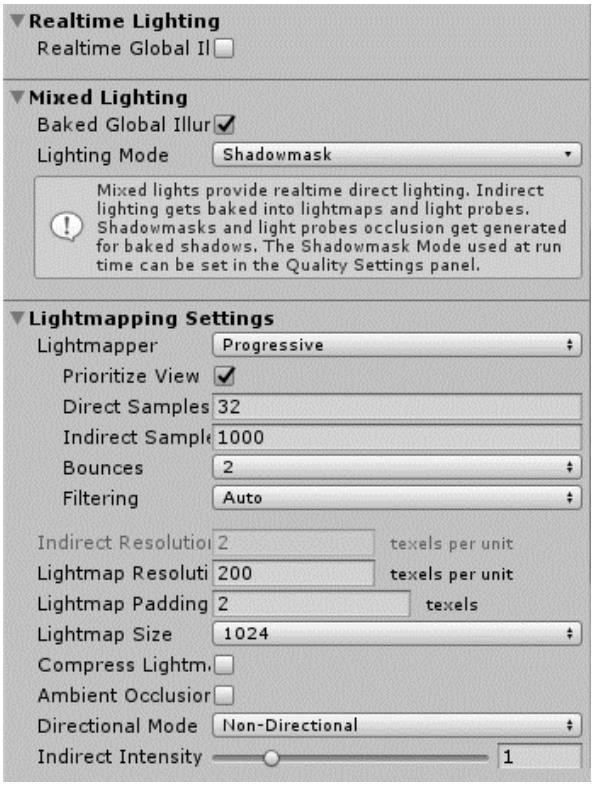


Figura 20. Ventana Lighting, la cual muestra los modos de iluminación y lightmapping. Fuente: Captura de pantalla de Unity.

Los parámetros que es necesario considerar son:

- *Lightmapper*: Refiere al software de cálculo de iluminación a usar para la generación de los *lightmaps*, sea *Progressive* o *Enlighten* (ambos con parámetros distintos que permiten al desarrollador obtener varios efectos de iluminación).
- *Texel* (Texture pixel): Entiéndase como la unidad mínima dentro de una textura o *lightmap*.
- *Lightmap Resolution*: La cantidad de texels por unidad a usar en los *lightmaps*.
- *Lightmap Size*: El tamaño en píxeles del *lightmap*.
- *Samples*: Refiere a la cantidad de muestras o de valores de color que genera el *lightmapper* tomando en cuenta la configuración de la luz o luces dentro de la escena para reducir o eliminar el "ruido" dentro del *lightmap*. Incrementar este valor resulta en una mejor calidad del *lightmap*, pero también en mayor tiempo de procesamiento en el *baked lighting* dentro de Unity.

2.5. BUENAS PRÁCTICAS DE PROGRAMACIÓN COMO OPTIMIZACIÓN EN UNITY

Si bien el arte y/o el diseño son parte importante de una aplicación en cualquier plataforma, es el código lo que permite la comunicación entre procesos o secciones de la aplicación y así, generar una correcta interacción por parte del usuario. En aplicaciones cuyo enfoque es el entretenimiento o la educación (tal es el caso de la mayoría de las aplicaciones de VR y AR), la programación podría representar la mayor parte de la experiencia del usuario. La programación dentro de Unity puede realizarse por medio de dos lenguajes; C# o JavaScript, y depende del desarrollador o programador el lenguaje a usar para el desarrollo del proyecto. El presente caso práctico desarrolla varios tipos de interacciones entre la aplicación y el usuario por medio del lenguaje C# y presenta las prácticas que se consideran importantes durante la programación para obtener un mejor desempeño.

2.5.1. Eliminación de declaraciones innecesarias

Si bien cada método, función o variable innecesaria dentro del código representa una demanda pequeña de recursos, la suma de estas declaraciones puede representar una disminución considerable de desempeño en el tiempo de ejecución de la aplicación.

Al crear un script dentro de Unity, éste contiene dos funciones base; *Start* y *Update*, la primera es una función que únicamente es llamada una vez al iniciar la escena (si el *gameobject* al que está sujeto se encuentra dentro), o también puede ser llamada al instanciar un *gameobject* en tiempo de ejecución. La función *Update* a diferencia de *Start* es llamada cada vez que la escena es renderizada o en cada *frame*, por ejemplo; si la escena es renderizada cincuenta veces por segundo, se harán cincuenta llamadas a la función y todo el código dentro de ésta será compilado dicha cantidad de veces.

Dependiendo de lo que se desee hacer mediante programación, los scripts pueden o no hacer uso de las funciones mencionadas, y es óptimo eliminarlas en caso de no ser útiles, pues si al cargar la escena existen demasiados *gameobjects* (o son instanciados en tiempo de ejecución), y cada uno de éstos contiene un script con funciones vacías, se asegura un decremento de desempeño en la CPU.

```
using UnityEngine;
using System.Collections;
public class Ejemplo : MonoBehaviour
{
    void Start ()
    {
    }
    void Update ()
    {
    }
}
```

Es recomendable saber identificar cuántas variables, métodos y funciones usar para realizar una determinada tarea, y así, reducir las líneas de código y evitar un decremento de desempeño al tener scripts con declaraciones innecesarias. Dichas declaraciones con frecuencia se ven relacionadas con el uso excesivo de condicionales como *if* o del uso descontrolado de *bucles* o iteraciones como *for*. Un ejemplo de declaraciones innecesarias podría ser el siguiente segmento de código, en el que se aprecian métodos y variables innecesarias y el uso excesivo de condicionales para activar un conjunto de *gameobjects* mediante el método *SetActive* de la clase *GameObject*.

```
    void Start () {
    }
    void Update (){
        Mask = LayerMask.GetMask ("Ejemplo");
        ray.origin = transform.position;
        ray.direction = transform.forward;
        if (Physics.Raycast (ray, out hit, 10f,Mask)) {
            activo = true;
            valor = activo;
        }
        if(activo==true){
            activo = false;
```

```
            for(int i=0;i<itera.Length;i++){
                itera [i].SetActive (valor);
        } }
    }
```

En el segmento de código anterior se puede identificar que la variable "*activo"* de tipo bool es usada únicamente para conocer si un rayo de la cámara ha golpeado un *collider*, y usar el valor de dicha variable en una segunda declaración condicional cuando la primera declaración *if* es capaz de devolver los valores de "true" o "false" y proceder a realizar una determinada acción.

Se aprecia también el uso de la variable "*valor*" para activar a los *gameobjects* del arreglo "*itera*", lo cual puede reemplazarse de manera fácil con un "true" como argumento de *SetActive*. El resultado de la eliminación de las variables "*activo*" y "*valor*" podría verse de la siguiente manera;

```
void Update (){
    Mask = LayerMask.GetMask ("Ejemplo");
    ray.origin = transform.position;
    ray.direction = transform.forward;
        if (Physics.Raycast (ray, out hit, 10f,Mask)) {
        for(int i=0;i<itera.Length;i++){
            itera [i].SetActive (true);}
        }
  }
```

2.5.2. La función Awake

Existe una función similar a *Start* llamada *Awake* y al igual que la primera, *Awake* es llamada sólo una vez en todo el tiempo de vida del *gameobject* al que está asociada. La diferencia de la última es que el código dentro de ésta es ejecutado sin importar que el script sea o no un componente activo dentro del *gameobject* y todas las funciones *Awake* siempre son llamadas antes que cualquier función.

El uso correcto de *Awake* supone una mejora menor pero importante en el desempeño de la aplicación, por ejemplo; si se desea realizar una determinada acción dentro de la función *Update* después de activar un conjunto de scripts en tiempo de ejecución pertenecientes a un arreglo de *gameobjects*, la mejor manera de realizar la acción y evitar una mayor demanda de recursos al instanciar variables en el momento de activar *n* cantidad de scripts, sería usar la función *Awake* en el script asociado con cada *gameobject* del arreglo de la siguiente manera;

```
void Awake () {
    gameobjectA = GameObject.Find ("GameObject-Ejemplo");
    level = gameobjectA.GetComponent<Level1> ();
    Mask = LayerMask.GetMask ("Estudiante");
}
void Update () {
    if (Physics.Raycast (ray, out hit, 10f, Mask)) {
        level.kills (1);
        Debug.Log ("Muerto...");
    }
}
```

En este ejemplo, la función *Awake* de cada script asociado a cada uno de los *gameobjects* del arreglo, busca un *gameobject* llamado “GameObject-Ejemplo”, obtiene su componente “Level 1” (script principal de “GameObject-Ejemplo”) y una vez esté activo el conjunto de scripts envía un valor a la función “kills” mediante *Update* si el rayo de la cámara golpea un *collider* llamado “Estudiante”.

Una manera fácil pero poco óptima cuando de activar demasiados scripts en un caso similar al presentado se trata, sería usar la función *Start* en lugar de *Awake*, sin embargo, al activar el conjunto de scripts asociados a cada *gameobject* del arreglo, el código dentro de *Start* sería ejecutado junto a las demás líneas del script e incrementaría la demanda de recursos de hardware con relación a la cantidad de scripts activados.

Por lo tanto, se considera una buena práctica instanciar variables al cargar una escena y no obtener los valores cada vez que se requieran u obtenerlos una vez, pero justo antes

de requerirlos, estén o no activos todos los scripts a usar, y hacerlo mediante el uso de *Awake* en casos similares al presentado.

2.5.3. Uso de Break y Return

Se pueden realizar repeticiones de acciones mediante *bucles* (For, While, Do-While, For Each) y es muy común el uso de dichos *bucles* para realizar acciones como volver transparente u opaca a una imagen, la obtención de un arreglo de componentes pertenecientes a un grupo de *gameobjects* o la obtención de un dato entre un conjunto de éstos.

Dependiendo de la acción para la que esté destinada un *bucle*, éste puede o no hacer uso de las declaraciones *break* y *return*. La primera permite salir o "romper" un *bucle* y continuar con el resto de las líneas de código dentro de la función, y la segunda declaración permite regresar o salir de una función sin continuar con la compilación de las siguientes líneas de código dentro de ésta.

El siguiente ejemplo muestra el uso del *bucle for* y de cómo se podría omitir el uso de las declaraciones mencionadas en algunos casos. El objetivo en el ejemplo es realizar el efecto conocido como *fade out* y generar la ilusión de desvanecimiento en una imagen por medio de la estructura *Color* y la modificación de los canales RGBA; donde R es rojo, G es verde, B es azul y A es el canal de transparencia.

```
for(float i = 1; i >= 0; i -= Time.deltaTime){
  ArB.GetComponent<Image> ().color = new Color(1, 1, 1, i);
}
```

El ejemplo anterior carece del uso de *break*, pues no es necesario que el *bucle* sea interrumpido y así, reducir el valor del canal A hasta cero para lograr el objetivo del desvanecimiento.

Un caso común en el que suele omitirse el uso de *break* es cuando se hace uso de algún *bucle* para obtener o conocer un determinado valor, dato o *gameobject*. El siguiente ejemplo muestra cómo se podría usar *break* para interrumpir un *bucle* después de encontrar o conocer un valor y no seguir con la demanda de procesamiento que involucra la iteración entre decenas, centenas o miles de elementos.

```
for(int i=0;i<Elements.Length;i++){
      if(Elements [i].activeSelf) {
            elementActive = Elements [i];
            atomActive = Atoms [i];
            break;}
}
```

El código anterior fue usado para conocer cuál de los ciento dieciocho átomos elaborados con sistemas de partículas se encuentra activo, y así proceder a desactivarlo en el momento que el usuario seleccione un elemento químico distinto.

La implementación de la declaración *return* en las funciones es similar al ejemplo anterior, sólo es necesario especificar en qué momento se desea salir de la función mediante condicionales u otros métodos, según sea el caso. Se considera una buena práctica de programación el uso de ambas declaraciones, ya que, si en el tiempo de ejecución de la aplicación se realiza un uso constante de iteraciones o *bucles* e incluso de llamadas a la función *Update* se apoya al incremento de recursos de hardware, y el uso de *break* y *return* podrían reducir la demanda en segmentos que podrían ser críticos para la CPU o la memoria del dispositivo.

Cada programador implementa distintas prácticas para mejorar el desarrollo e implementación de código o el tiempo de desarrollo de la aplicación.

Capítulo 3. Desarrollo de prototipos

3.1. CONSIDERACIONES DE DISEÑO

Las texturas, los íconos de los elementos, las imágenes de los espectros, las imágenes usadas en la UI y las usadas en el sistema de partículas fueron desarrolladas en Adobe Photoshop Cs6, el cual permite el uso de capas para facilitar la elaboración de éstas y realizar modificaciones en un futuro sin tener que repetir todo el proceso (figura 21).

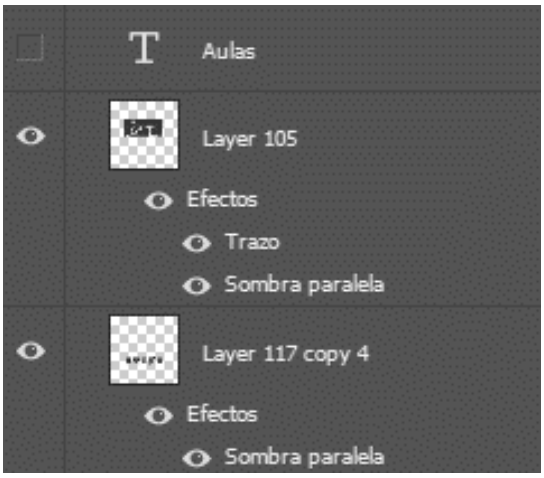


Figura 21. Capas en la interfaz gráfica de Adobe Photoshop. Fuente: Captura de pantalla de Adobe Photoshop.

Cada uno de los íconos de los elementos que conforman la tabla periódica fueron elaborados con base en el diseño establecido que ordena a los elementos por su número atómico, períodos y grupos. Para respetar el orden de los elementos y sus grupos se escogieron distintos colores, los cuales otorgan también un atractivo visual a la aplicación (figura 22).

Figura 22. Ícono del Oro dentro de la tabla periódica. Fuente: Elaboración propia.

Los íconos para la UI (figuras 23 y 24), tanto en el nivel de VR como en el de AR, fueron elaborados con base en los bocetos y con el uso de capas para un fácil desarrollo. Se determinaron colores en su mayoría claros como fondo y para los elementos de cada ícono, se agregaron sombras para simular distancias entre las capas que lo conforman y el fondo a ser usado dentro de la UI como un atractivo visual.

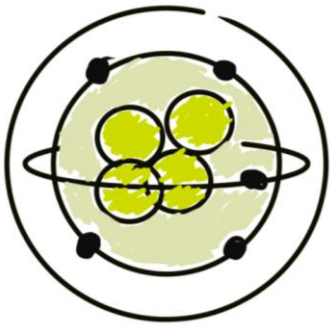


Figura 23. Boceto para la elaboración del ícono del átomo dentro del nivel de VR. Fuente: Elaboración propia.

La mayoría del contenido multimedia fue elaborado de manera vectorial (figura 25) y con base en bocetos previamente hechos, es decir, el contenido fue creado dentro de Adobe Photoshop Cs6 y no editado de una imagen ya existente. La mayoría de los editores de imágenes contienen una herramienta para la elaboración de polígonos o formas vectoriales (normalmente conocida como *pluma*), consiste en el uso de curvas de Bézier que ayudan al diseño asistido por computadora facilitando la creación de formas.

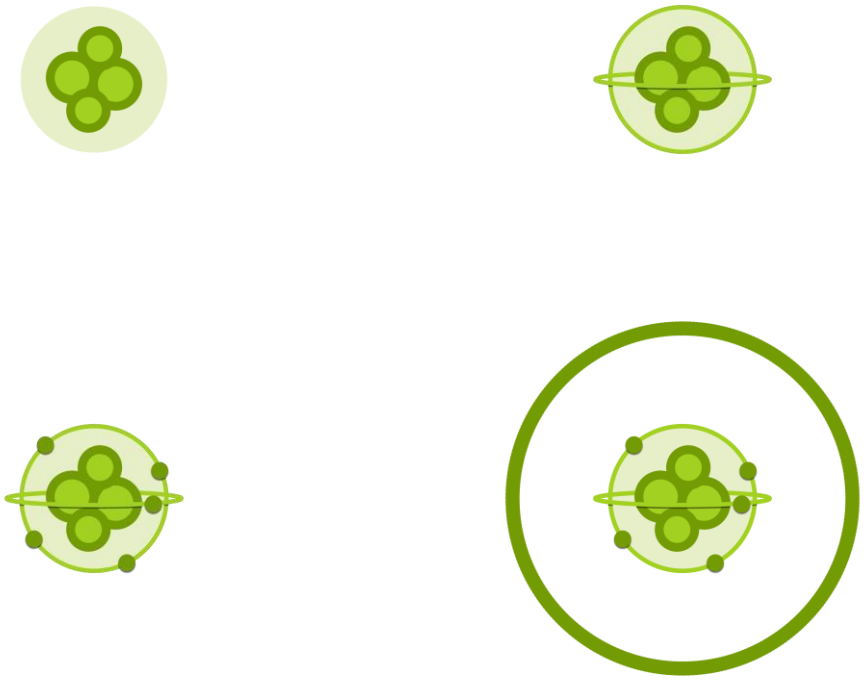


Figura 24. Elaboración de íconos por medio de capas. Fuente: Elaboración propia.

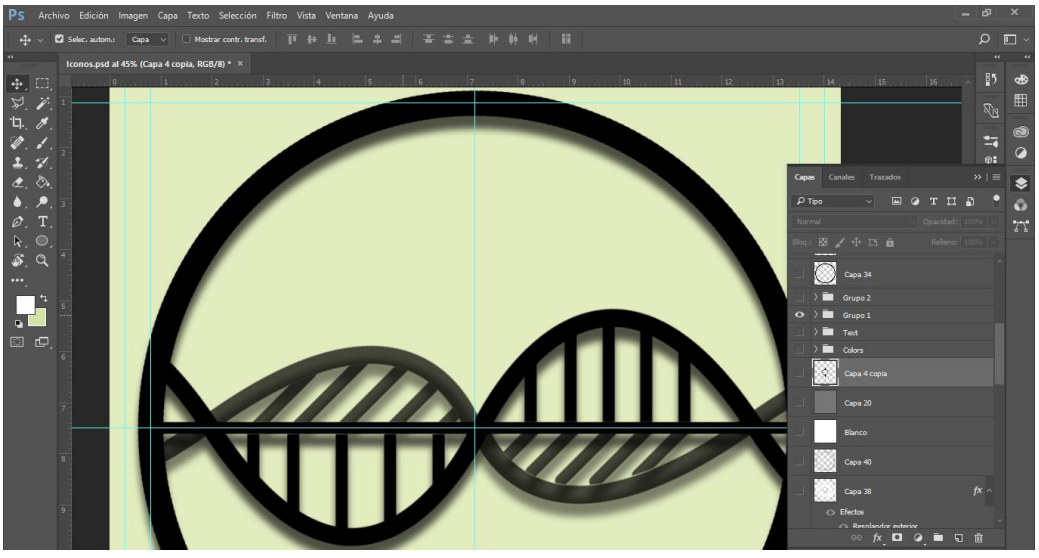


Figura 25. Creación de imágenes vectoriales en Adobe Photoshop para su uso en la UI. Fuente: Captura de pantalla de Adobe Photoshop.

La elaboración de los espectros electromagnéticos de cada uno de los elementos químicos fue con base en referencias que muestran los espectros de emisión (figura 26). Dichas referencias ayudaron a generar imágenes con mejor calidad visual de los espectros electromagnéticos para su uso en el presente caso práctico.

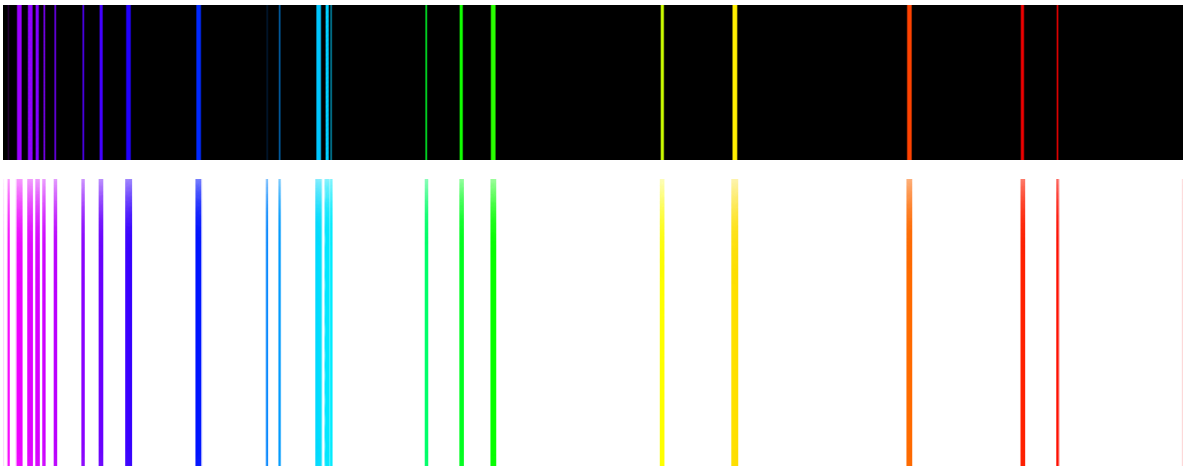


Figura 26. Referencia del espectro electromagnético e imagen resultante. Fuente: Elaboración propia.

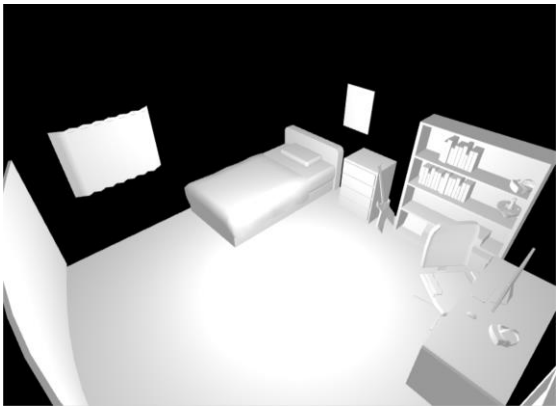


Figura 27. Conjunto de modelos 3D para su uso en el nivel de VR del presente caso práctico. Fuente: Elaboración propia.

En la creación de los *modelos 3D* de la escena de realidad virtual (figura 27), se hizo uso de herramientas que, en conjunto a técnicas de optimización, disminuyen la cantidad de polígonos significativamente, la técnica de optimización dentro del modelado 3D más usada en el desarrollo del presente trabajo consiste en la eliminación de polígonos no

visibles para el usuario, ya que el nivel de procesamiento es proporcional a la cantidad de polígonos con los que cuenta el objeto tridimensional, mientras menos polígonos tenga éste, mejor para el rendimiento del procesador. La figura 28 muestra desde otros ángulos al mismo conjunto de *modelos 3D* de la figura anterior, en esta se observa la escasez de polígonos con la que cuenta el conjunto, pues el usuario no es capaz de visualizar las partes inferiores de los objetos tridimensionales dentro de la aplicación. La combinación de objetos también puede ser usada como técnica de optimización (figura 29).

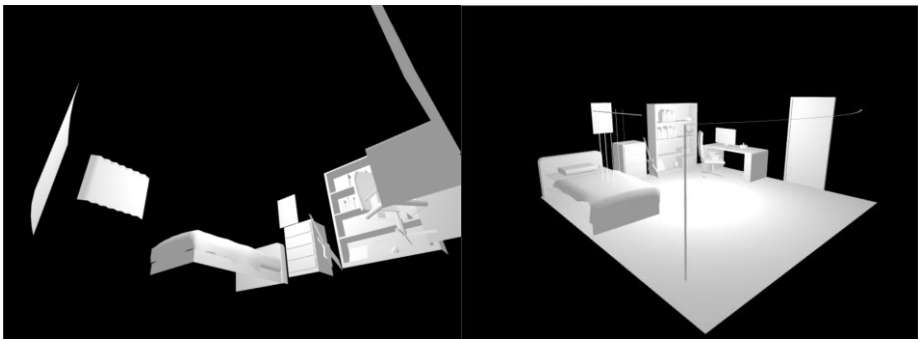


Figura 28. Eliminación de polígonos no visibles por el usuario (polígonos inferiores y traseros). Fuente: Elaboración propia.

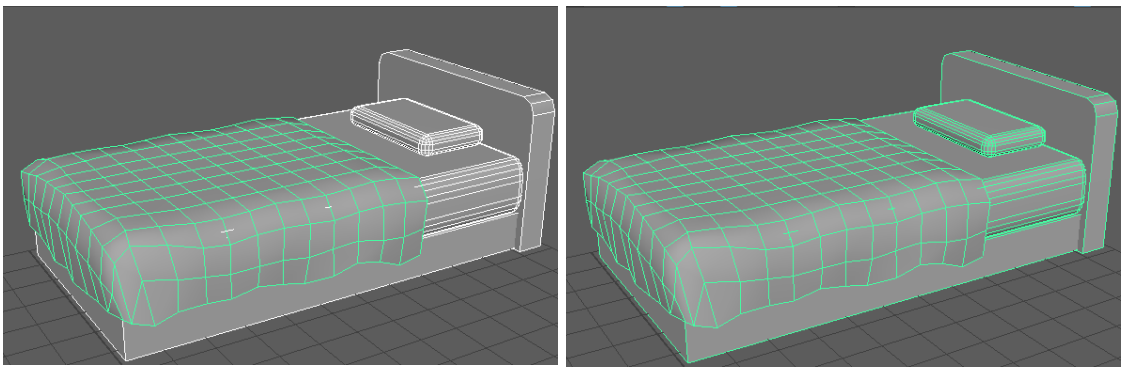


Figura 29. Combinación de objetos como técnica de optimización. Fuente: Elaboración propia.

La figura 13 muestra el *UV Layout* resultante de la combinación, éste fue usado para la elaboración de la textura de la figura 30, en la cual se usan colores sólidos y con detalles minimalistas para reducir la cantidad de bytes de la textura resultante.

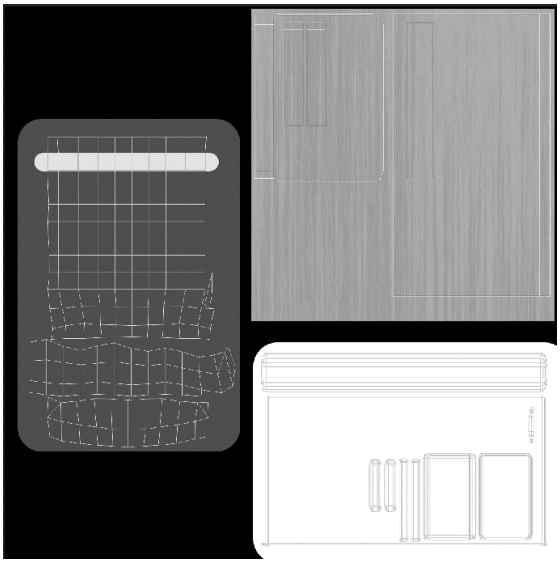


Figura 30. Implementación de la textura resultante con UV´s sobre la superficie del modelo 3D. Fuente: Elaboración propia.

Para el desarrollo de la aplicación fueron necesarios únicamente tres niveles, en éstos el usuario es capaz de interactuar con el contenido multimedia e interactuar con *modelos 3D* en VR y AR. La importación de assets dentro de Unity es sencilla y aunque existen distintas maneras de hacerlo, la más fácil consiste en arrastrar los elementos de su ubicación a la ventana *Project* dentro de la interfaz de Unity. Una vez importados los elementos de cualquier formato digital (png, jpeg, fbx, mp3, etc.), Unity los reconoce como *assets*. Dentro de Unity es posible la creación de más *assets*, tales como scripts (archivos de código) y *GameObjects* (objetos 3D como cubos, esferas y cilindros, objetos 2D, canvas para la implementación de interfaces gráficas, etc.) y más archivos reconocidos como *assets* por Unity (figura 31).

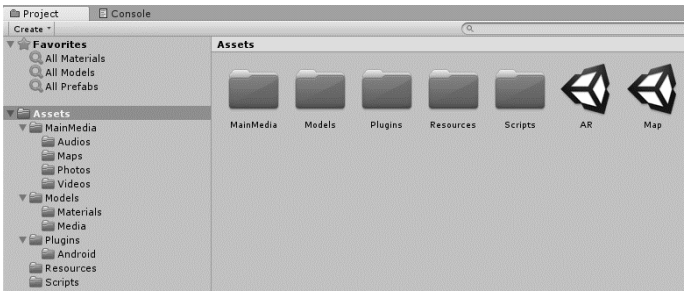


Figura 31. Captura de pantalla de la venta “proyecto” perteneciente a la interfaz gráfica de Unity. Fuente: Captura de pantalla de Unity.

El primer nivel de la aplicación permite al usuario escoger entre VR y AR para el acceso a los niveles relacionados con cada una de éstas. Para su elaboración sólo fueron necesarias dos imágenes y un botón para acceder a la escena seleccionada (figura 32).

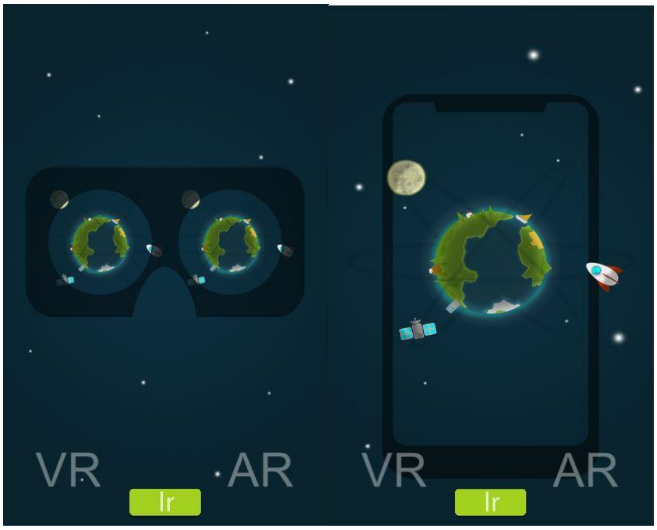


Figura 32. Primer nivel, en el que el usuario determina el uso de VR o AR. Fuente: Elaboración propia.

Para la elaboración del nivel de VR (segundo nivel) se hizo uso de *modelos 3D* para formar la habitación que conforma la escena, dentro de dicha habitación se permite al usuario desplazarse entre la mayor parte del área que la conforma con ayuda de un indicador y una cuadricula adherida al suelo (figura 33).

El desplazamiento por medio de la cuadricula genera una perspectiva de distancia y permite una navegación virtual al usuario sin la necesidad de desplazarse en el entorno físico. La navegación virtual permite visualizar e interactuar con cada una de las cuatro secciones; “el átomo”, “el visualizador”, “características” e “información”.

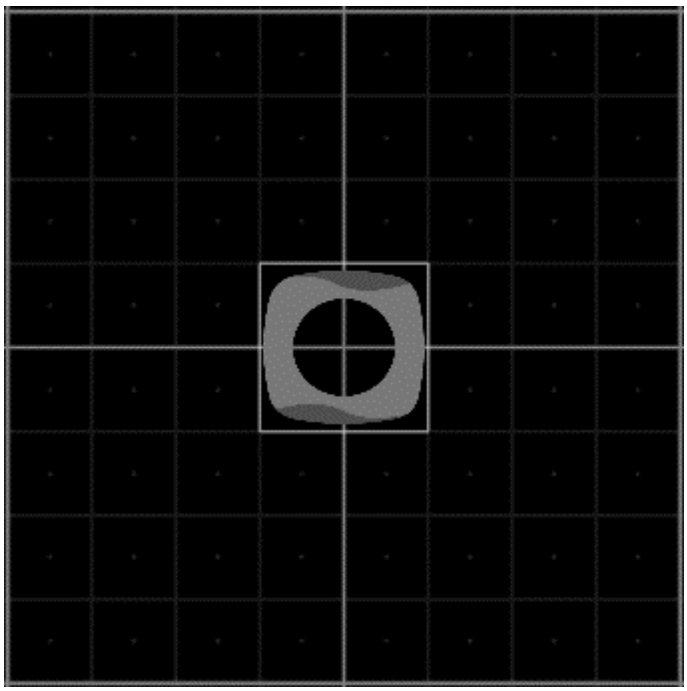


Figura 33. Indicador y cuadricula que permiten el desplazamiento dentro de la habitación. Fuente: Elaboración propia.

3.2. DISEÑO DE INTERFAZ

El diseño de la interfaz de usuario fue parte fundamental para permitir las interacciones con la aplicación en los niveles de VR y AR. Se realizaron bocetos en papel de varias propuestas, teniendo en mente una interfaz intuitiva y minimalista que permita al usuario interacciones sencillas y sin el uso de elementos multimedia o efectos visuales que comprometan la experiencia y/o el buen desempeño de la aplicación. Para el nivel de AR se diseñó una interfaz que en su mayoría está compuesta por la tabla periódica de los elementos químicos, pues se consideró como interacción principal la selección de un elemento y el mostrar su información básica. La figura 34 muestra el boceto realizado para permitir la navegación e interacciones en el nivel.

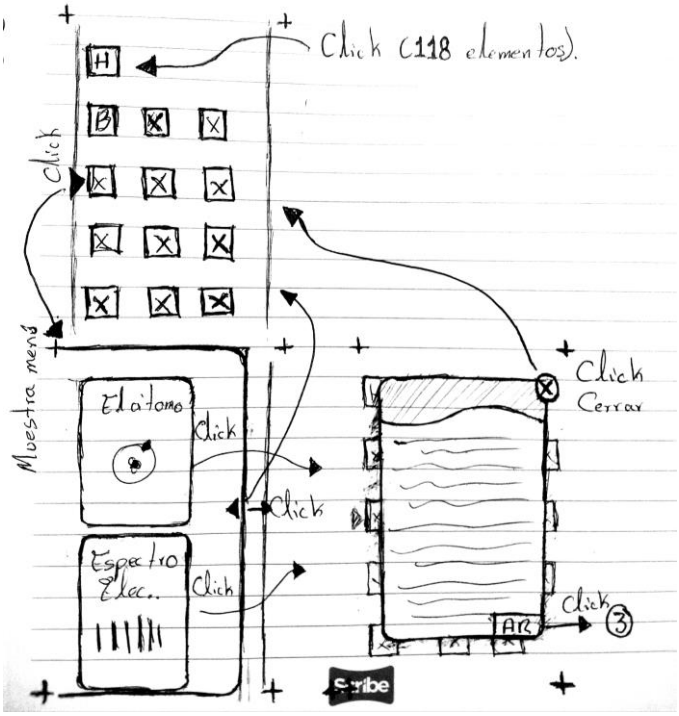


Figura 34. Boceto de la interfaz del nivel de AR que permite la navegación e interacción. Fuente: Elaboración propia.

Con base en el boceto anterior se realizó un plano de pantalla o *Wireframe* que permitió guiar visualmente el desarrollo del nivel de AR. La figura 35 muestra dicho plano, y en éste se aprecian distintas pantallas con la interfaz gráfica del usuario y cómo se relacionan los componentes entre sí para permitir la interacción por parte del usuario.

Para el nivel de VR el diseño consistió en la elaboración de varios planos de pantalla, pues la interfaz de este nivel se conforma de varias secciones que, si bien se comunican entre sí, las interacciones que contienen afectan únicamente a la sección a la que pertenecen (a excepción del visualizador de la tabla periódica y parte de la sección "información"). Para la sección "el átomo" se diseñó una interfaz que permita al usuario de una manera fácil interactuar con la estructura del átomo sin invadir su campo visual. Lo anterior fue la razón de que la interfaz de esta sección se encuentre en la parte inferior del nivel, así se permite al usuario apreciar en la mayor parte del campo visual la estructura del átomo de uno de los ciento dieciocho elementos químicos.

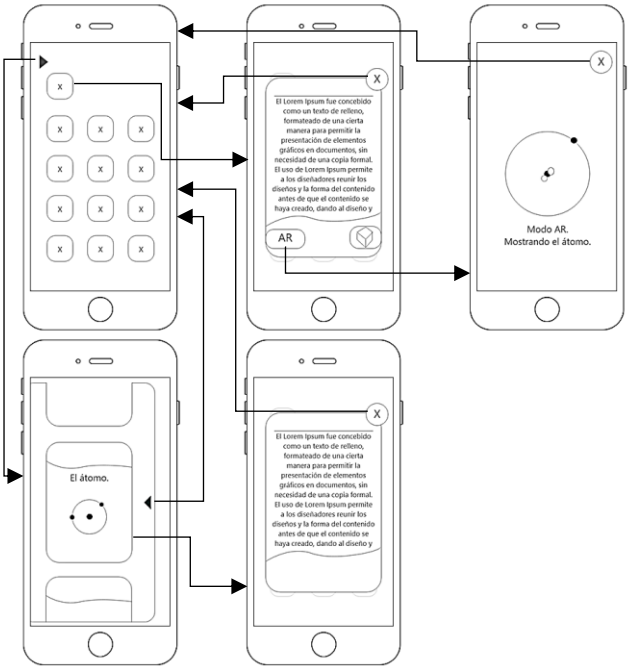


Figura 35. Plano de pantalla de la interfaz del nivel de AR cuya función es de guía visual. Fuente: Elaboración propia.

La siguiente figura muestra los planos de pantalla que fueron usados como guía visual en el desarrollo de la interfaz gráfica, al igual que las interacciones dentro de ésta.

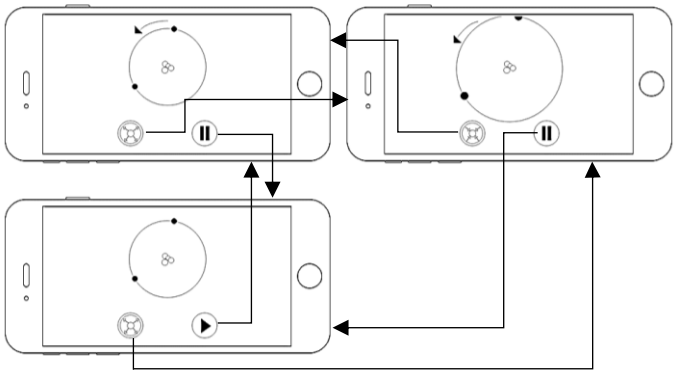


Figura 36. Plano de pantalla de la interfaz dentro de la sección “el átomo” del nivel de VR. Fuente: Elaboración propia.

La sección “información” afecta de manera directa al visualizador, pues es en éste donde se muestra la información sobre uno de los tres temas del nivel. La siguiente figura muestra el plano de pantalla que permitió el desarrollo de la comunicación entre secciones.

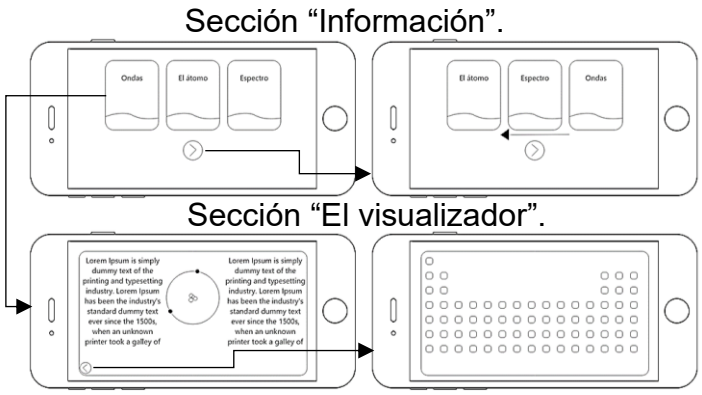


Figura 37. Plano de pantalla de la comunicación entre la sección “información” y “el visualizador”. Fuente: Elaboración propia.

Para las secciones “características” y “el visualizador” el diseño elaborado fue mucho más simple (figura 38), pues la primera tiene como objetivo mostrar únicamente las características del elemento químico seleccionado y la segunda mostrar la tabla periódica, permitir la selección del usuario, mostrar el espectro electromagnético de la selección e información sobre algún tema.

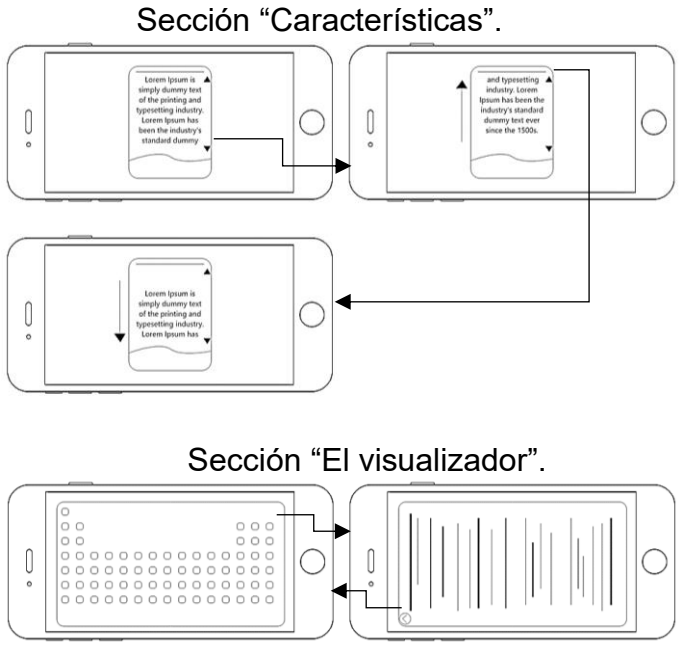


Figura 38. Planos de pantalla de las secciones “características” y “el visualizador” del nivel de VR. Fuente: Elaboración propia.

Es importante mencionar que, aunque se elaboró un diseño simple o minimalista con una cantidad relativamente baja de elementos multimedia, el objetivo fue evitar altas o innecesarias demandas de hardware, pero considerando la programación de interacciones que atribuyan a generar una buena experiencia de usuario. Por lo tanto, se puede mencionar que las interacciones óptimas en cuestión de desempeño dentro de los niveles tuvieron su origen en el diseño de la interfaz, pero que su correcto y óptimo funcionamiento depende de cómo sean programadas.

Considerando lo anterior, el diseño de interfaz asumió como objetivo principal guiar visualmente la elaboración y el diseño del nivel, pero tuvo también un papel importante en la etapa de programación, en la que se busca respetar el diseño de la interfaz y del nivel construyendo las interacciones y mecánicas de la manera más eficiente posible.

3.3. SECCIÓN PRINCIPAL: “EL ÁTOMO”

El objetivo de esta sección es mostrar una representación de la estructura del átomo, lo que involucra la visualización del núcleo y los electrones de los ciento dieciocho elementos químicos reconocidos de la tabla periódica, se encuentra en el centro de la escena y permite visualizar la estructura del átomo seleccionado por el usuario. Debido a que el átomo es el atractivo visual principal de la aplicación, se usaron sistemas de partículas para su elaboración, por lo tanto, los átomos carecen de modelos tridimensionales (lo que reduce la cantidad de polígonos a renderizar en la escena).

Cada electrón de los átomos es un sistema de partículas, sin embargo, cada sistema emite una única partícula que tiene un tiempo de vida largo, de esta manera se cree que se trata de un solo objeto sólido y se mantiene un desempeño óptimo en la visualización de todo el átomo al no renderizar varias partículas en tiempos cortos.

Sin embargo, cada átomo tiene una cantidad de electrones distinta y el número máximo de éstos puede llegar a ser de ciento dieciocho, también, el número de protones y neutrones que conforman el núcleo aumenta con relación a la cantidad de electrones, por lo tanto, en determinado momento la aplicación renderiza un átomo con poco más de ciento dieciocho sistemas de partículas ejecutándose al mismo tiempo.

La mayoría de los 24 *batches* del átomo del Oganesón son producidos por los sistemas de partículas pertenecientes a sus electrones y por un solo sistema de partículas del núcleo.

Los núcleos de los ciento dieciocho átomos fueron elaborados por medio de un único sistema de partículas (figura 39) que emite hasta cien partículas (para átomos con una cantidad de protones mayor a cien), pero contiene un tiempo de vida largo, lo que significa que las partículas que lo conforman son emitidas y renderizadas una vez al seleccionar el elemento a mostrar y no son emitidas de nuevo hasta quince minutos después (si el usuario no ha seleccionado otro elemento químico).

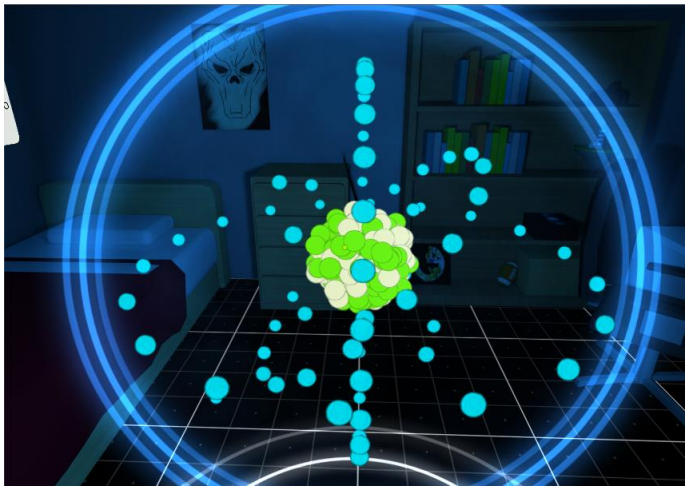


Figura 39. Uno de los ciento dieciocho átomos hechos de sistemas de partículas. Fuente: Elaboración propia.

3.4. SECCIÓN SECUNDARIA: “EL VISUALIZADOR”

“El visualizador” es un lugar determinado a la lectura, interacción y visualización de la información relacionada al átomo, al espectro electromagnético y a las ondas electromagnéticas. Actúa también como tabla periódica, en la que el usuario es capaz de visualizar y seleccionar uno de los ciento dieciocho elementos químicos por medio de *colliders* y *rays*. Se hizo uso de *raycast* para todas las interacciones permitidas al usuario, considerando como buena práctica el uso de un sólo rayo para la detección de todos los objetos y el uso de *colliders* primitivos (*colliders* básicos como cubos, esferas y cilindros que son sencillos de procesar) (figura 40).

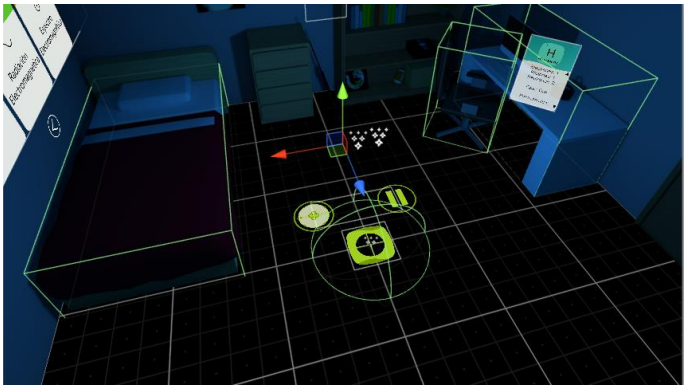


Figura 40. Colliders representados como objetos que encapsulan a su objeto contenedor. Fuente: Elaboración propia.

Cada imagen representante de un elemento químico conforma la tabla periódica (figura 41) y contiene un *box collider* (colisionador con forma de cubo) que permite al ser golpeado por el rayo un desplazamiento hacia el usuario, la acción anterior tiene únicamente como objetivo el indicar al usuario qué elemento se ha seleccionado.

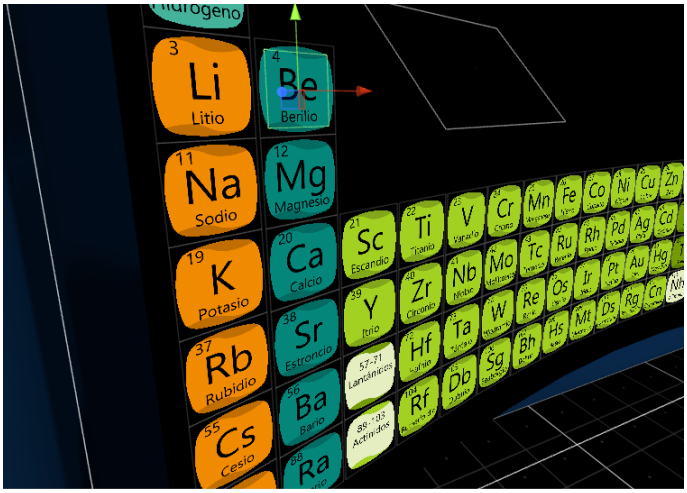


Figura 41. Imágenes de los ciento dieciocho elementos químicos que conforman la tabla periódica.
Fuente: Elaboración propia.

3.5. SECCIÓN: “CARACTERÍSTICAS”

Una vez seleccionado el elemento por medio de *raycast*, las características de éste aparecen en un *canvas* (espacio dentro de la escena para la elaboración de una UI), que muestra la cantidad de neutrones, electrones y protones que contiene, al igual que su peso atómico, su distribución de electrones y otros datos considerados relevantes para el usuario.

Las características están ordenadas de manera vertical sobre el *canvas* para una fácil lectura por parte del usuario (figura 42), sin embargo, dada la cantidad de texto que ocupan fue imposible integrar a éste en el espacio resumido destinado al *canvas*. Para tener un orden y permitir una mejor lectura, se usaron los componentes *Scroll Rect* y *Scroll Bar*. Desafortunadamente el componente *Scroll Bar* requiere de una entrada táctil o del uso de un dispositivo apuntador para su funcionamiento común, por lo que para el presente caso práctico se implementa una manera distinta para hacer uso de dicho componente y desplazar el texto perteneciente a las características del elemento químico seleccionado.

Se trata de desplazar el contenido por medio del uso de *raycast* y modificando así los valores del *Scroll Bar* mediante código, por lo tanto, se requirió colocar dos imágenes con *colliders* que indican y permiten al usuario visualizar completamente el contenido.

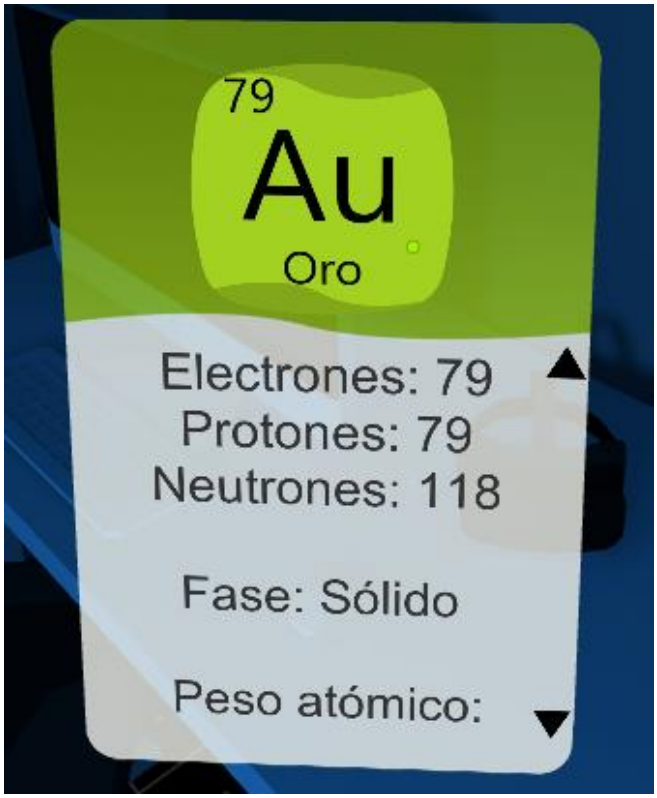


Figura 42. Sección que muestra las características del elemento químico. Fuente: Elaboración propia.

3.6. SECCIÓN: “INFORMACIÓN”

La última sección tiene el objetivo de permitir al usuario escoger uno de los tres temas a leer; El átomo, Radiación electromagnética y Espectro electromagnético. Al seleccionar un tema, la información es mostrada en “el visualizador” y para mejorar la lectura del texto el color de éste cambia a un gris claro mediante código (figura 43).

Para apoyar de manera gráfica a la lectura se crearon varias imágenes que se muestran en la sección “el visualizador” junto con la información del tema seleccionado.

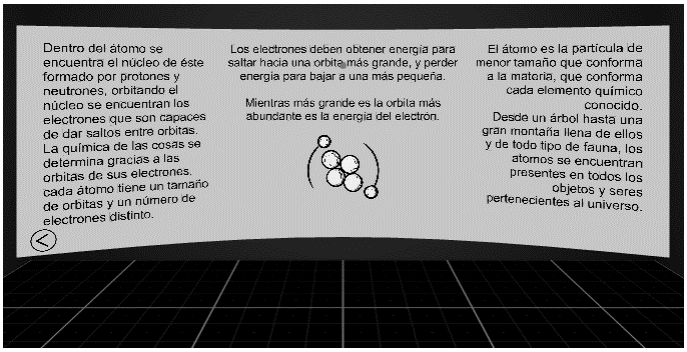


Figura 43. Información sobre el átomo mostrada en “el visualizador”. Fuente: Elaboración propia.

La selección, como todas las interacciones en este nivel implementa los componentes *raycast* y *colliders*.

Debido a la iluminación deseada para el nivel de VR y la alta demanda de recursos de hardware que ésta genera al ser calculada en tiempo real, para dicho nivel que se conforma en su mayoría por *modelos 3D*, texturas, y efectos de iluminación y sombreado, se generaron *lightmaps* mediante el uso de *baked lighting*.

Una vez que se especificó el uso de *Baked Global Illumination*, el *lightmapper* a usar, la resolución y el tamaño de los *lightmaps* en las configuraciones, fue necesario especificar qué *modelos 3D* dentro de la escena se verían afectados.

Es necesario que todos aquellos *modelos 3D* afectados por el *baked lighting* sean estáticos, por lo tanto, que carezcan de animaciones o de desplazamientos y/o rotaciones por medio de programación o animación. Se declaró a cada *modelo 3D* como estático al seleccionar dentro del *inspector* la opción *Lightmap Static* (figura 44).

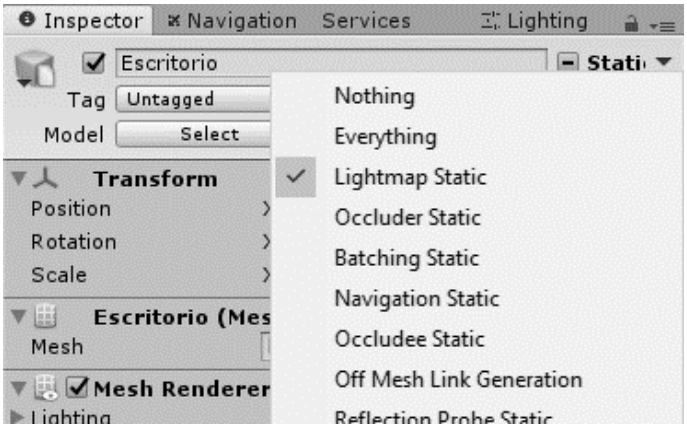


Figura 44. Ventana Inspector, la cual muestra cómo declarar a un gameobject como estático. Fuente: Captura de pantalla de Unity.

Por último, para la generación de los *lightmaps* se cambió el modo de cada luz a *baked*, y los demás parámetros que ofrece la configuración de la luz quedaron a consideración con base en el efecto de iluminación deseado. Para el nivel se tomó en cuenta el color de los sistemas de partículas que conforman el átomo, y la ubicación de éste dentro de la escena, por lo tanto, el tipo de luz usado fue *point light*, que es un punto dentro del espacio en la escena que emite luz hacia todas direcciones, formando algo similar a una esfera y provocando la ilusión de que es el átomo ubicado en el centro el emisor de luz (figura 45).

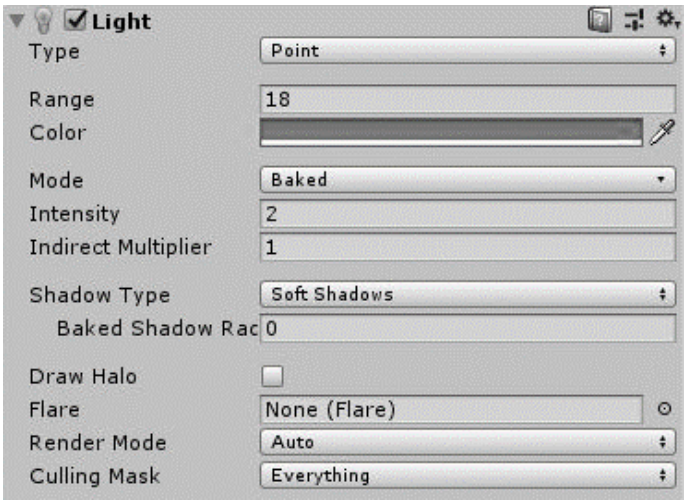


Figura 45. Configuración del tipo de luz point light en donde se especifica el modo como baked. Fuente: Captura de pantalla de Unity.

Una vez realizados los pasos anteriores, se generaron los *lightmaps* en la ventana *Lighting*, los cuales fueron almacenados dentro de una carpeta creada por el proceso con el nombre de la escena actual y ubicada dentro de la carpeta principal de *assets*. La parte inferior de la ventana permite generar de manera automática o manual la iluminación especificada, sin embargo, para que los *lightmaps* fueran tomados como *assets* (figura 46), se almacenaran dentro de la aplicación y se pudiese crear un archivo contenedor de los datos de iluminación, fue necesario desactivar la generación automática y realizar el proceso de *lightmapping* de manera manual.

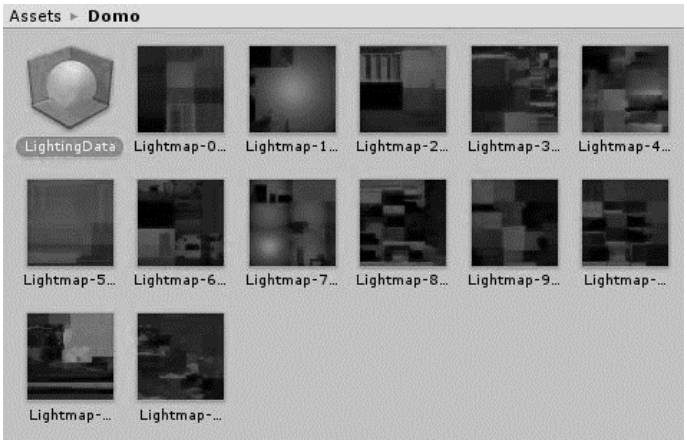


Figura 46. Conjunto de lightmaps y de los datos de iluminación dentro del proyecto en Unity. Fuente: Captura de pantalla de Unity.

Es recomendable tener una visión de lo que se quiere lograr con la iluminación, pues dependiendo del hardware del dispositivo considerado, de la arquitectura de la escena, del objetivo o del nivel de *inmersión* que se desea, la iluminación puede o no ser *baked*. También podrían existir casos o situaciones en las que dentro de una misma escena existan objetos estáticos y no estáticos, y se pueda optar por una iluminación mixta (superficies que requieran del cálculo de iluminación en tiempo real y otras que simplemente contengan precálculos de ésta). La figura 47 muestra el antes y el después del proceso de *baked lighting*.

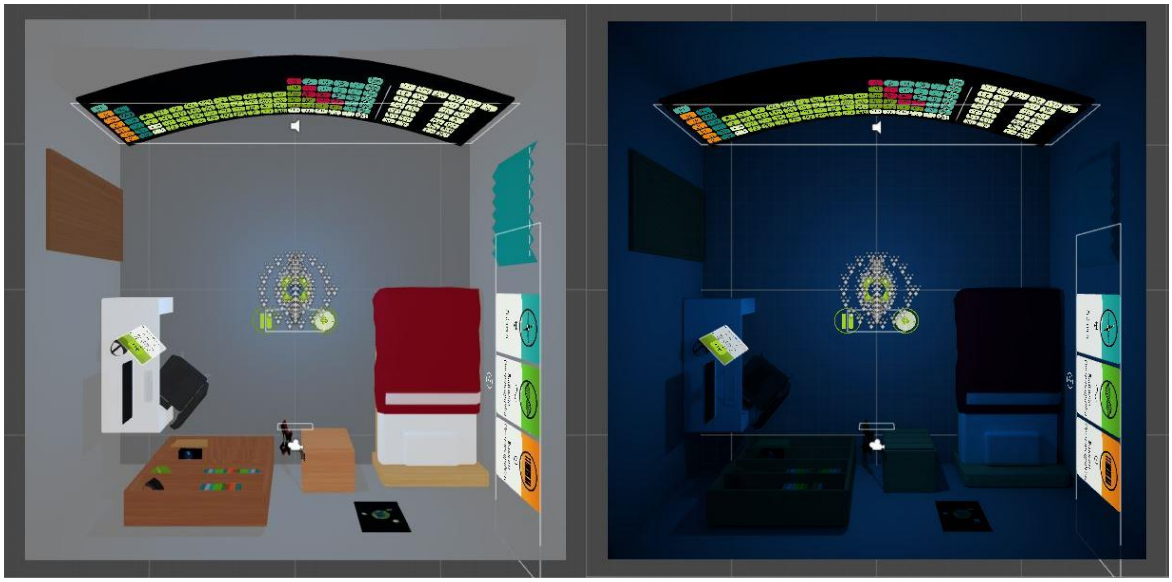


Figura 47. Antes y después de baked lighting en el nivel de VR. Fuente: Captura de pantalla de Unity.

El tercer nivel de la aplicación se enfoca en la implementación de AR y está dividido en dos secciones, la primera muestra la tabla periódica por medio de una interfaz gráfica al usuario y la segunda, muestra el átomo seleccionado en AR.

3.7. SECCIÓN: "TABLA PERIÓDICA Y CARACTERÍSTICAS"

Permite visualizar los elementos químicos de la tabla periodica (figura 48). Igual que en el nivel de VR, esta sección hace uso de los componentes *Scroll Rect* y *Scroll Bar* para la visualización de la tabla periódica y de las características de los elementos químicos.

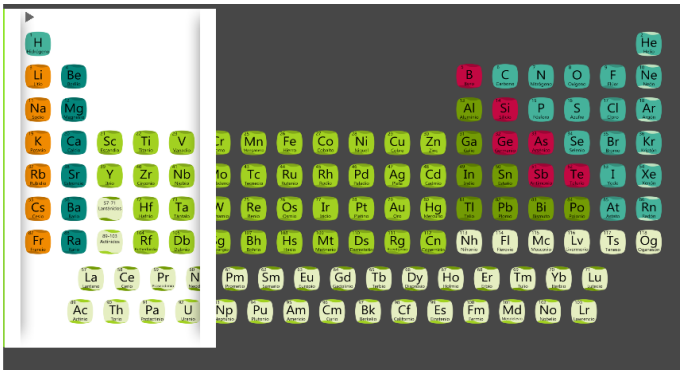


Figura 48. Tabla periódica dentro de un Scroll Rect para poder visualizar dentro del área blanca. Fuente: Elaboración propia.

En la sección “el visualizador” del nivel anterior se implementaron *colliders* y *raycast* para que el usuario sea capaz de interactuar con los elementos químicos. La tabla periódica mostrada en la figura anterior y perteneciente al nivel de AR implementa únicamente el componente *Button* (Botón) para la interacción por parte del usuario.

3.8. SECCIÓN: “AR”

Esta sección podría considerarse como la más sencilla de todas, pues su funcionamiento depende en su mayoría de la programación y no de componentes. Los únicos componentes usados en esta sección son:

- *Raw Image*: Imagen que representa cualquier tipo de textura y que es usada en el presente caso práctico para renderizar las imágenes captadas por la cámara del dispositivo.
- *Collider*: Para el uso de *raycast* y así, obtener una vista cercana del átomo seleccionado.
- *Button*: Usado para mostrar el botón de regreso a la sección anterior y que se encuentra ubicado en la parte superior de la pantalla.
- *Aspect Ratio Fitter*: Componente que permite ajustar la relación de aspecto de los elementos mostrados en la pantalla con base en las dimensiones de la pantalla física del dispositivo. *Aspect Ratio Fitter* permite visualizar de manera correcta las imágenes captadas por la cámara del dispositivo.

La siguiente figura se aprecia un átomo elaborado con sistemas de particulas dentro de la sección de AR en tiempo de ejecución.

Figura 49. Vista del átomo en AR. Fuente: Elaboración propia.

3.9. PROGRAMACIÓN DE INTERACCIONES

Tanto el nivel de VR como el de AR tienen como objetivo mostrar los átomos de los ciento dieciocho elementos químicos conocidos, por lo tanto, se desarrolló una manera de mostrar cada uno de los átomos e información con relación a la selección del usuario.

Se creó un script por nivel para mostrar el átomo seleccionado por el usuario y la información relacionada a éste. Para conocer qué átomo e información mostrar, en cada script se declaró un arreglo tipo *string* con los ciento dieciocho símbolos pertenecientes a los elementos químicos, de tal manera que al interactuar con la imagen del elemento el script recibiera un dato del mismo tipo y se procediera a una comparación entre dicho dato y todos los elementos del arreglo.

Para lograr la comparación entre un valor y todos los elementos pertenecientes a un arreglo se requirió del uso del *bucle for* y del condicional *if*, el primero para iterar entre los elementos del arreglo, y el segundo para conocer si el valor dado es igual a uno de los elementos del arreglo.

La siguiente función se declaró para realizar la comparación y ayuda a comprender el uso de la declaración *break* como una buena práctica en el proceso de programación y como técnica de optimización.

```
public void elementAR(string value){
            this.GetComponent<AudioSource> ().Play ();
            TextCont.SetActive (true);
            StartCoroutine (fadeout(Table));
            loadar.objectToAR (value);
            for(int i=0;i<elements.Length;i++){
                  if(elements[i].name == value){
                        elements [i].SetActive (true);
                        objectActived = elements [i];
                        break;}
            }
      }
```

La función anterior pertenece al nivel de AR y fue declarada para mostrar la información del elemento químico y permitir ver la representación del átomo en AR a partir de la selección del usuario. Una vez que se ha comparado con los elementos del arreglo, el valor es enviado a la función *objectToAR* para posteriormente mostrar el átomo en AR.

En el nivel de VR se programó una función similar para poder mostrar el átomo, el espectro electromagnético y la información del elemento químico con relación a la selección del usuario, por lo tanto, al seleccionar un elemento mediante *raycast* se ejecutan las tres acciones, comenzando con la comparación mediante el *bucle for* y el condicional *if*.

3.9.1. Programación y uso del raycast en VR

Anteriormente se explicó el funcionamiento del *raycast*, su manera de interactuar con los *colliders* y cómo ayudan a las interacciones dentro de un nivel de VR. En el caso práctico actual el *raycast* fue implementado para todas las interacciones permitidas en el nivel de VR, las cuales se refieren a:

- Desplazarse de un punto a otro dentro de la escena mediante el sistema de navegación.
- Pausar o iniciar la animación de los electrones del átomo.
- Expandir o contraer las orbitas de los electrones del átomo.
- Iniciar la animación para desplazar las opciones sobre los temas del átomo, el espectro y la radiación electromagnéticos, y seleccionar uno de éstos dentro de la sección “información”.
- Desplazar la información del átomo mostrada en la sección “características”.
- Seleccionar uno de los ciento dieciocho elementos de la tabla periódica dentro de la sección “el visualizador”.

Para todas las interacciones anteriores se implementó un solo rayo, cuyo origen es el centro de la cámara (vista del usuario) y su dirección en todo momento hacia enfrente (sin importar el movimiento o rotación por parte del usuario).
Sin embargo, es necesario mencionar que el rayo es creado en cada *frame* dentro del tiempo de ejecución de la aplicación, lo anterior refiere a que en cada *frame* se determina el origen y dirección del rayo, su longitud y si éste ha “golpeado” un *collider* con una determinada *layer* (etiqueta o capa para conocer si el rayo ha detectado determinados objetos).

La estructura del *raycast* usada para el nivel de VR fue la siguiente:

```
void Update () {
      ray.origin = transform.position;
      ray.direction = transform.forward;
      if (Physics.Raycast (ray, out hit, 11f, elementMask)) {
            }
      }
```

Donde:

- *ray.origin* es el origen del rayo.
- *ray.direction* la dirección del rayo.
- *ray*, el rayo con determinado origen y dirección.
- *hit*, el nombre dado a una estructura *RaycastHit* para conocer si ha “golpeado” un colisionador.
- *11f*, la longitud del rayo.
- *elementMask*, el nombre dado a la etiqueta asociada con las imágenes de los ciento dieciocho elementos químicos dentro del visualizador.

Una forma de hacer buen uso del *raycast* fue limitar lo mejor posible el código dentro de la condición usada para reducir la demanda de recursos por parte de las interacciones y de hacer uso de un solo rayo durante todo el tiempo de ejecución.

3.9.2. Programación de efectos visuales

Una parte importante en el aspecto visual de la aplicación fue la implementación de efectos visuales como animaciones de orbitales, y efectos de *fade in* y *fade out* mediante programación.

Para el nivel de VR, fue necesario animar los orbitales de todos los átomos de la tabla periódica, para esto, se requirió de modificar las propiedades de los *gameobjects* contenedores de los sistemas de partículas que forman los electrones.

La figura 50 muestra un orbital (*gameobject* principal) y cómo los electrones (sistemas de partículas) fueron colocados formando una circunferencia que posteriormente rotara sobre su eje central mediante código.

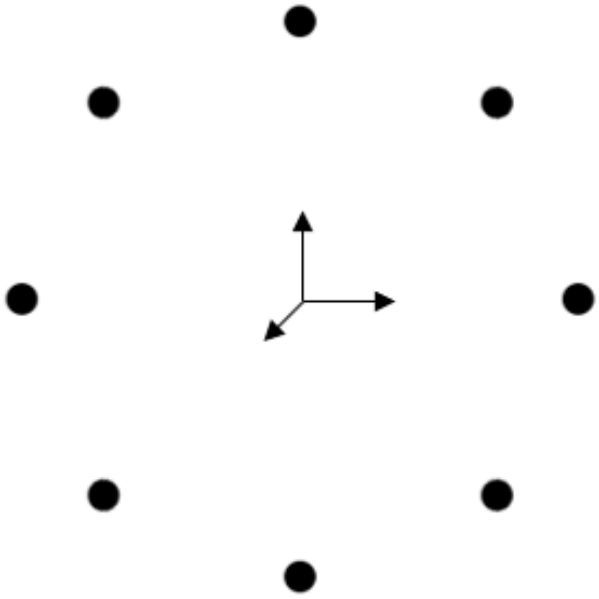


Figura 50. Sistemas de partículas rodeando el centro de un gameobject padre. Fuente: Elaboración propia.

Se decidió elaborar la rotación de los electrones mediante programación, debido a que el usuario puede visualizar la animación por tiempo indeterminado, siendo necesario que los orbitales del átomo estén rotando en todo momento.

```
transform.RotateAround(transform.position,
transform.TransformDirection(Vector3.up), speed * Time.deltaTime);
```

3.8.3. Activación y desactivación de objetos

Debido a que el usuario es capaz de visualizar cualquiera de los ciento dieciocho átomos creados con sistemas de partículas, para el nivel de VR se requirió de una función que permitiera ocultar el átomo visualizado en la sección principal y mostrar el átomo perteneciente a la nueva selección del usuario. Una manera de realizar la acción anterior pudo ser:

- Crear los ciento dieciocho átomos, declararlos como *assets* dentro de una carpeta llamada *Resources* de la ventana *Project*.
- Remover el átomo actual mediante el método *Destroy*.

- Colocar el átomo perteneciente a la nueva selección del usuario en la sección principal mediante el método *Instantiate*.

Sin embargo, una práctica que se considera importante para obtener un mejor desempeño es hacer el menor uso posible del método *Instantiate*, dicho método clona un *gameobject* y coloca el clon dentro de la escena en tiempo de ejecución, lo que genera un consumo alto de procesamiento.

Para evitar la generación de un mayor consumo de procesamiento en tiempo de ejecución, se prefirió realizar la acción de ocultar y mostrar los átomos mediante el uso del método *SetActive*, el cual permite activar y desactivar *gameobjects* que ya se encuentran dentro de la escena. Por lo tanto, la acción se realizó de la siguiente manera:

- Se crearon los ciento dieciocho átomos dentro de la escena de VR y se colocaron en la misma posición de la sección principal.
- Se desactivaron ciento diecisiete átomos en la ventana *Inspector* para permitir únicamente la visualización de uno de ellos.
- En tiempo de ejecución, al seleccionar un elemento químico distinto al mostrado se desactiva el átomo actual mediante *SetActive(false)* y se muestra el perteneciente a la nueva selección con *SetActive(true)*.

La función fue programada de la siguiente manera:

```
public void getAtom(string nameAtom){
      for(int i=0;i<Elements.Length;i++){
            if(nameAtom==Atoms[i].name && nameAtom!=atomActive.name){
                  atomActive.SetActive (false);
                  Atoms [i].SetActive (true);
                  atomActive = Atoms [i];
                  break;
            }
      }
}
```

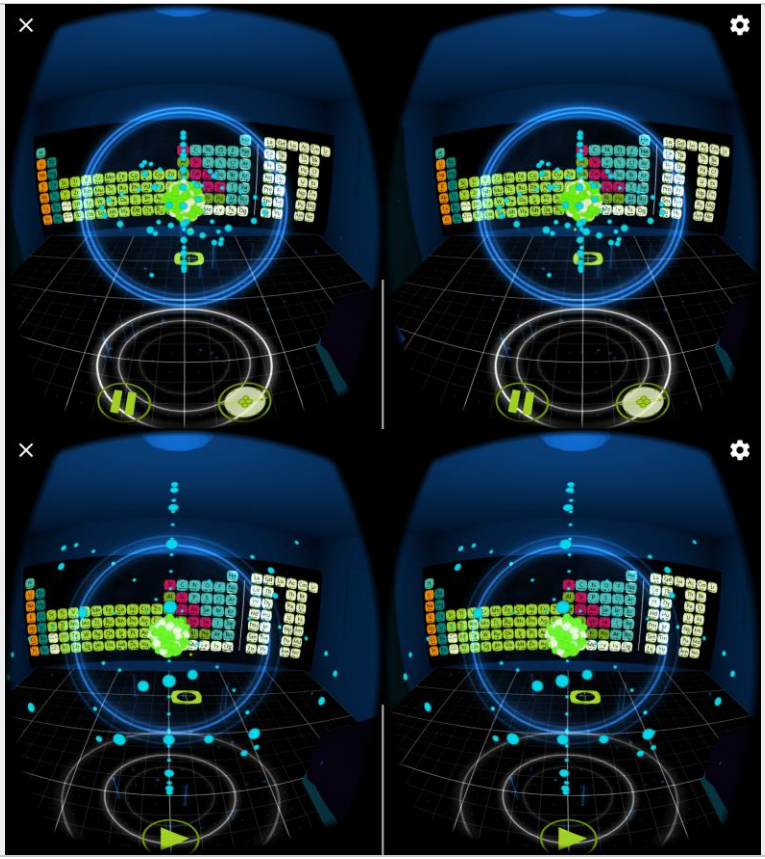
Capítulo 4. Evaluación de prototipos

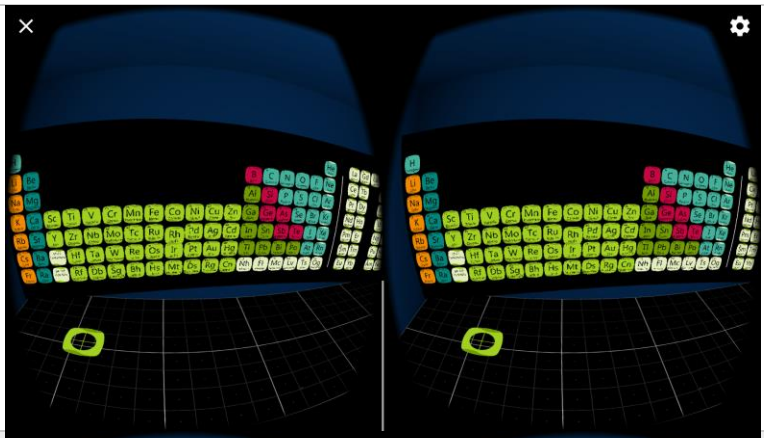
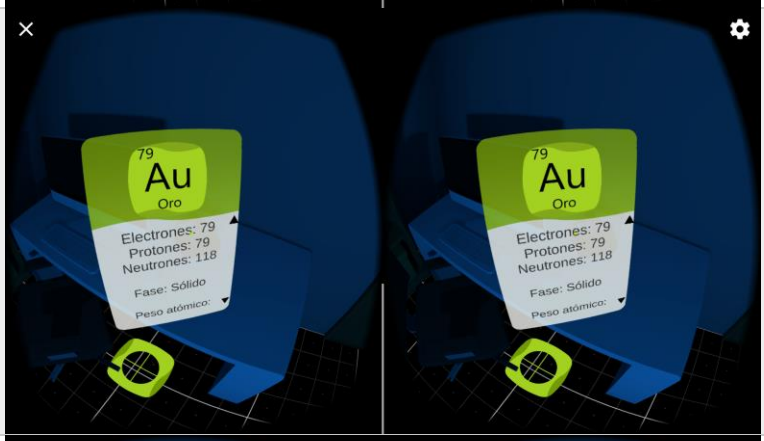
4.1. FUNCIONALIDAD

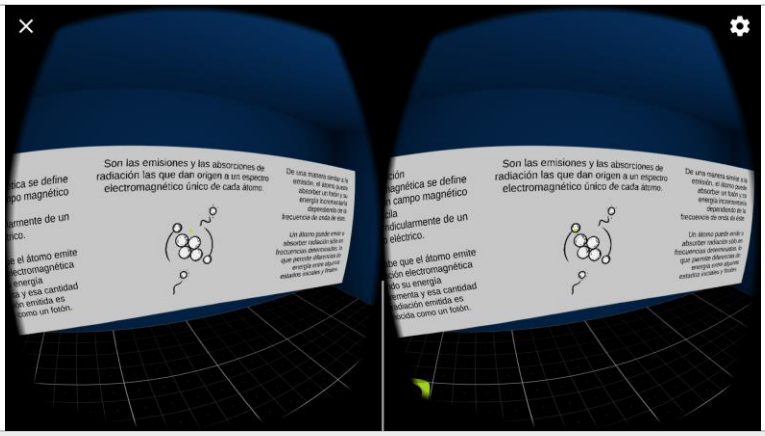
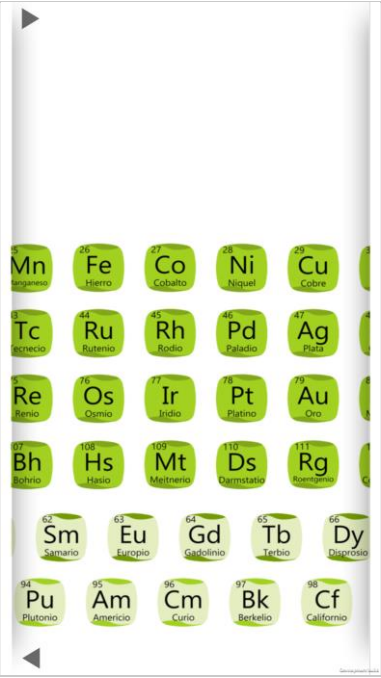
Terminado el desarrollo se procedió a comprobar la funcionalidad de cada una de las interacciones dentro de los niveles de AR y VR. La comprobación se realizó ejecutando la aplicación en Unity y en un dispositivo móvil, que permitió detectar y solucionar problemas de funcionamiento.

A continuación, se muestran las diferentes secciones en ejecución.

Tabla 3. Pruebas de funcionalidad de las distintas secciones de la aplicación.

Sección.	Ejecución.
Nivel de VR.	
Sección: “el átomo”.	

Sección: “el visualizador”.	
Sección: “características”.	
Sección: “información”.	

	
Nivel de AR.	
Sección: “Tabla periódica y características”.	

		
Sección: “AR”.		

Fuente: Elaboración propia.

4.2. PRUEBAS DE RENDIMIENTO

4.2.1. Unity Profiler

Las pruebas de rendimiento del presente caso práctico para la comprobación de la viabilidad de un desarrollo, que implementa técnicas de optimización y buenas prácticas en cada fase fueron realizadas en el tiempo de ejecución de ambas versiones del caso práctico, y mediante el uso del profiler integrado en Unity que muestra estadísticas de la demanda de hardware en varias áreas de la aplicación.

La ventana *Profiler* dentro de Unity permite realizar pruebas en tiempo de ejecución de una aplicación, generando informes de uso y estadísticas para recopilar datos de múltiples componentes (CPU, GPU, memoria, renderización, iluminación, etc.) y comparar los datos con métricas deseadas o identificar problemas de rendimiento, permitiendo la posibilidad de optimizar o mejorar secciones de la aplicación.

A continuación, se enlistan los componentes considerados importantes para las pruebas de rendimiento en dispositivos móviles y para la comprobación de la hipótesis del presente trabajo, de los cuales se obtuvieron estadísticas mediante el profiler de Unity que es capaz de generar y mostrar informes de uso:

- CPU Usage (Uso de la CPU): Muestra el uso de la CPU por múltiples subsistemas durante el tiempo de ejecución, tales como; los scripts, el renderizado, las interfaces de usuario, las animaciones, las físicas y la iluminación global.
- Memory (Memoria): Muestra la cantidad de memoria usada por distintos componentes dentro de la escena, tales como; texturas, materiales, clips de audio, animaciones y mallas de *modelos 3D* u otros elementos. También muestra la cantidad de *assets* y *gameobjects* en la escena.
- *Rendering* (Renderizado): Muestra estadisticas sobre los procesos y subsistemas de renderizado, tales como; la cantidad de *draw calls*, *batches*, *SetPass Calls*

(configuración de los materiales de cada objeto renderizado), y la cantidad de triángulos y *vértices* (geometría) mostrados por la cámara.

- Physics (Físicas): Muestra estadísticas sobre los componentes que implementan físicas dentro de la escena, tales como; *colliders*, *triggers overlaps*, *rigidbody* (cuerpo rígido, componente usado para la detección de colisiones), objetos dinámicos con animaciones y el uso de *raycast*.
- Global Illumination (Iluminación Global): Muestra estadísticas sobre la cantidad total de tiempo usado por la CPU únicamente para la iluminación, el tiempo dedicado a la actualización de luces en tiempo real, la actualización de la iluminación del entorno, de los objetos dinámicos o en movimiento, el número de materiales esperando ser procesados, y entre otros subsistemas relacionados a la existencia de iluminación en tiempo real.

4.2.2. Obtención de datos

La obtención de datos o medición de rendimiento de cada uno de los componentes mencionados anteriormente se realizó de forma remota mediante el uso del Android Debug Bridge (ADB) y el profiler de Unity debido a que dicha forma permite la obtención de datos desde el dispositivo móvil en tiempo real (figura 51). Los datos se obtuvieron desde el tiempo de ejecución de ambas versiones del caso práctico en un dispositivo móvil. Los pasos realizados para la medición de manera remota fueron:

- Se conectó de manera alámbrica el dispositivo móvil a la computadora donde se encuentra almacenado el proyecto.
- Dentro de Unity, en la ventana *Build Settings* se habilitó la opción *Development Build.*
- Se procedió a la compilación y ejecución de la aplicación mediante la opción *Build & Run*.
- Una vez que se ejecutó la aplicación en el dispositivo móvil, se abrió la ventana *Profiler* y se escogió la opción *AndroidProfiler(ADB@127.0.0.1:34999)* para la visualización de los datos en el profiler.

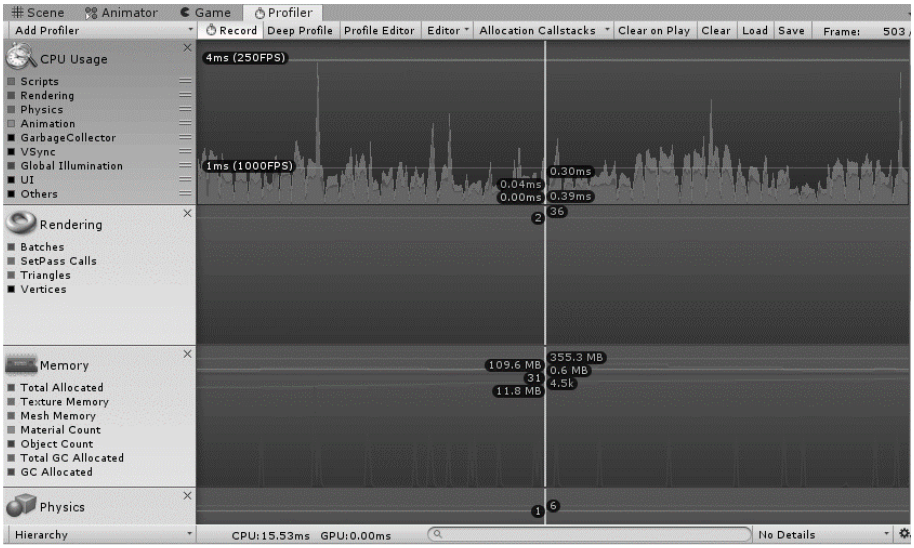


Figura 51. Ventana Profiler, la cual muestra los datos de la medición en milisegundos. Fuente: Captura de pantalla de Unity.

Para realizar la obtención de datos de manera ordenada, se fraccionó la estructura de la aplicación con base en las interacciones y/o acciones que cuentan con una cantidad mayor de programación y/o el uso de componentes que suponen una demanda de procesamiento significante como *colliders*, *raycast* y sistemas de partículas.

La siguiente tabla muestra y describe cada una de las interacciones medidas en ambas versiones del caso práctico para el análisis de rendimiento de cada uno de los componentes mencionados anteriormente.

Tabla 4. Interacciones y acciones medidas en tiempo de ejecución.

Interacción o acción medida.	Descripción.
Sección "el átomo" (VR).	
• Interrupción de la animación de orbitales.	Interacción por parte del usuario para detener la animación de los orbitales del átomo.
• Reproducción de la animación de orbitales.	Interacción por parte del usuario para continuar la animación de los orbitales del átomo.
• Expansión de orbitales.	Interacción por parte del usuario para reproducir la animación de expansión de

	los orbitales del átomo (los orbitales se alejan lentamente del núcleo).
• Contracción de orbitales.	Interacción por parte del usuario para reproducir la animación de contracción de los orbitales del átomo (los orbitales se acercan lentamente del núcleo).
• Interrupción de la animación de orbitales estando expandidos.	Interacción por parte del usuario para detener la animación de los orbitales del átomo después de haber reproducido la animación de expansión.
• Reproducción de la animación de orbitales estando expandidos.	Interacción por parte del usuario para continuar la animación de los orbitales del átomo después de haber reproducido la animación de expansión.
Sección "el visualizador" (VR).	
• Selección de un elemento dentro de la tabla periódica.	Interacción por parte del usuario que muestra una animación de acercamiento de la imagen seleccionada, las características, el espectro electromagnético y el átomo del nuevo elemento químico seleccionado.
Sección "información" (VR).	
• Animación de rotación de los temas.	Interacción por parte del usuario para reproducir la animación de rotación de los diferentes temas relacionados al átomo.
• Selección de uno de los temas y visualización de la información en la sección "el visualizador".	Interacción por parte del usuario para mostrar la información de uno de los temas relacionados al átomo en la sección "el visualizador".
Ejecución del nivel de AR.	Acción de iniciar el nivel de realidad aumentada, el cual está formado en su mayoría de imágenes.
Sección "Tabla periódica y características" (AR).	
Animación del contenedor de los temas.	Interacción por parte del usuario para reproducir la animación que muestra los diferentes temas relacionados al átomo.
Selección de un elemento dentro de la tabla periódica.	Interacción por parte del usuario que muestra las características.
Sección "AR".	
Visualización del átomo en AR.	Interacción por parte del usuario que muestra el átomo del elemento químico seleccionado en realidad aumentada.

Fuente: Elaboración propia.

4.2.3. Comparación de datos

Para una mejor comparación de datos se tomaron valores máximos y mínimos proporcionados por el profiler de Unity en cada medición, y se calcularon porcentajes con base en los valores obtenidos de la versión no optimizada para determinar si hubo o no una mejora de rendimiento en cada componente medido de la versión optimizada de la aplicación.

A continuación, se muestra en qué momento (frames) fueron obtenidos los valores máximos y mínimos de cada interacción en ambas versiones, y en las siguientes tablas se muestran los valores obtenidos en cada componente.

Tabla 5. Momentos en los que se obtuvieron los valores máximos y mínimos.

Interacción	Ambas versiones	
	Valor máximo	**Valor mínimo**
Ejecución del nivel de VR.	El frame en el que la aplicación abrió el nivel de VR.	Un frame antes de abrir el nivel de VR.
Interrupción de la animación de orbitales.	El frame en el que fue pausada la animación de los orbitales.	Un frame antes de la interacción.
Reproducción de la animación de orbitales.	El frame en el que fue reproducida la animación de los orbitales.	Un frame antes de la interacción.
Expansión de orbitales.	El frame en el que los orbitales comenzaron a expandirse.	El frame en el que los orbitales dejaron de expandirse.
Contracción de orbitales.	El frame en el que los orbitales comenzaron a encogerse.	El frame en el que los orbitales dejaron de encogerse.
Interrupción de la animación de orbitales estando expandidos.	El frame en el que fue pausada la animación de los orbitales estando expandidos.	Un frame después de que la animación fue pausada.
Reproducción de la animación de orbitales estando expandidos.	El frame en el que fue reproducida la animación de los orbitales estando expandidos.	Un frame después de que la animación fue reproducida.
Selección de un elemento dentro de la tabla periódica.	El frame en el que fue seleccionado un elemento	El frame en el que se mostró el espectro y el átomo de la selección.

	distinto de la tabla periódica.	
Animación de rotación de los temas.	El frame en el que el menú empezó a rotar.	El frame en el que el menú dejó de rotar.
Selección de uno de los temas y visualización de la información.	El frame en el que se seleccionó un tema.	El frame en el que desapareció la tabla periódica y apareció la información del tema seleccionado.
Ejecución del nivel de AR.	El frame en el que se abrió el nivel de AR.	Un frame después del inicio del nivel.
Animación del contenedor de los temas (AR).	El frame en el que inició la animación para mostrar el contenedor.	El frame en el que terminó la animación.
Selección de un elemento dentro de la tabla periódica (AR).	El frame en el que se mostró la ventana de características del átomo y empezó el efecto de fade out de la tabla periódica.	El frame en el que terminó el efecto de fade out de la tabla periódica.
Visualización del átomo en AR.	El frame en el que se abrió el modo AR y se visualizó el átomo.	Un frame después de que se inició el modo AR.

Fuente: Elaboración propia.

Tabla 6. Comparación de resultados obtenidos del componente CPU.

Componente: CPU	Se midieron los tiempos en milisegundos que tomó cada interacción o acción (valores máximos) y los tiempos antes o después (valores mínimos).			
Interacción	**Versión no optimizada**		**Versión optimizada**	
	Valor máximo	**Valor mínimo**	**Valor máximo**	**Valor mínimo**
Ejecución del nivel de VR.	6688.8 ms (100%)	18 ms (100%)	7818.8 ms (117%)	15.9 ms (88.3%)
			Hubo un decremento de desempeño de 17% al iniciar el nivel de VR en la versión optimizada.	Antes de iniciar el nivel de VR hubo una mejora en el desempeño de 11.7%.
Interrupción de la animación de orbitales.	86.1 ms (100%)	79.6 ms (100%)	52.5 ms (61%)	41.5 ms (52.1%)
			Esta interacción presentó una mejora en el desempeño de 39%.	Antes de iniciar la interacción hubo una mejora en el desempeño de 47.9%.

Reproducción de la animación de orbitales.	87.7 ms (100%)	80 ms (100%)	53.2 ms (60.7%)	45.3 ms (57%)
			Esta interacción presentó una mejora en el desempeño de 39.3%.	Antes de iniciar la interacción hubo una mejora en el desempeño de 43%.
Expansión de orbitales.	121 ms (100%)	79 ms (100%)	56.4 ms (46.6%)	38 ms (48%)
			Esta interacción presentó una mejora en el desempeño de 53.4%.	Al terminar la interacción hubo una mejora en el desempeño de 52%.
Contracción de orbitales.	86.6 ms (100%)	75.6 ms (100%)	56.2 ms (65%)	40 ms (53%)
			Esta interacción presentó una mejora en el desempeño de 35%.	Al terminar la interacción hubo una mejora en el desempeño de 47%.
Interrupción de la animación de orbitales estando expandidos.	86.6 ms (100%)	77.7 ms (100%)	53.4 ms (61.7%)	41.2 ms (53%)
			Esta interacción presentó una mejora en el desempeño de 38.3%.	Al terminar la interacción hubo una mejora en el desempeño de 47%.
Reproducción de la animación de orbitales estando expandidos.	83.3 ms (100%)	73.7 ms (100%)	54 ms (64.8%)	43 ms (58.3%)
			Esta interacción presentó una mejora en el desempeño de 35.2%.	Al terminar la interacción hubo una mejora en el desempeño de 41.7%.
Selección de un elemento dentro de la tabla periódica.	161.8 ms (100%)	25.4 ms (100%)	121.9 ms (75.3%)	17.7 ms (69.7%)
			Esta interacción presentó una mejora en el	Al terminar la interacción hubo una mejora en el

			desempeño de 24.7%.	desempeño de 30.3%.
Animación de rotación de los temas.	71 ms (100%)	70 ms (100%)	53 ms (74.6%)	52.8 ms (75.4%)
			Esta interacción presentó una mejora en el desempeño de 25.4%.	Al terminar la interacción hubo una mejora en el desempeño de 24.6%.
Selección de uno de los temas y visualización de la información.	101.8 ms (100%)	30.9 ms (100%)	40.6 ms (39.9%)	26.3 ms (85.1%)
			Esta interacción presentó una mejora en el desempeño de 60.1%.	Al terminar la interacción hubo una mejora en el desempeño de 14.9%.
Ejecución del nivel de AR.	2106.9 ms (100%)	10.3 ms (100%)	2334.5 ms (111%)	12.1 ms (117.5%)
			Hubo un decremento de desempeño de 11% al iniciar el nivel de AR en la versión optimizada.	Hubo un decremento de desempeño de 17.5% al terminar de abrir el nivel de AR.
Animación del contenedor de los temas (AR).	16.8 ms (100%)	16.2 ms (100%)	17.4 ms (103.5%)	16.5 ms (101.8%)
			Hubo un decremento de desempeño de 3.5% al mostrar el contenedor de temas.	Hubo un decremento de desempeño de 1.8% al terminar la animación del contenedor de temas.
Selección de un elemento dentro de la tabla periódica (AR).	181.3 ms (100%)	17 ms (100%)	133 ms (73.3%)	17 ms (100%)
			Esta interacción presentó una mejora en el desempeño de 26.7%.	Al terminar la interacción hubo una mejora en el desempeño de 0%.

Visualización del átomo en AR.	1916.2 ms (100%)	20 ms (100%)	1848.1 ms (96.5%)	20 ms (100%)
			Esta interacción presentó una mejora en el desempeño de 3.5%.	Al terminar la interacción hubo una mejora en el desempeño de 0%.

Fuente: Elaboración propia.

La comparación anterior muestra una mejora de desempeño en las distintas interacciones de los niveles, sin embargo, al iniciar cualquiera de éstos se obtuvo un decremento de hasta el 17.5% comparado con los datos de la versión no optimizada.

Tabla 7. Comparación de resultados obtenidos del componente *Rendering*.

Componente: *Rendering* (Renderizado)	Se midieron en ambas versiones, la cantidad de *draw calls* y polígonos que fueron renderizados al inicio de las acciones o durante las interacciones dentro de los niveles (valores máximos), y también la cantidad antes de cada acción o después de cada interacción (valores mínimos).			
Interacción	**Versión no optimizada**		**Versión optimizada**	
	Valor máximo	**Valor mínimo**	**Valor máximo**	**Valor mínimo**
Ejecución del nivel de VR.	364 *draw calls* (100%) 8,200 Tris (100%)	5 *draw calls* (100%) 0 Tris (100%)	313 *draw calls* (85.9%) 5,100 Tris (62.2%)	5 *draw calls* (100%) 0 Tris (100%)
			Hubo una mejora de desempeño de 14.1% en *draw calls* y de 37.8% en cantidad de polígonos.	Antes de iniciar el nivel de VR no hubo cambio alguno entre las versiones.
Interrupción de la animación de orbitales.	714 *draw calls* (100%) 17,400 Tris (100%)	698 *draw calls* (100%) 16,400 Tris (100%)	621 *draw calls* (87%) 8,200 Tris (41.1%)	613 *draw calls* (87.8%) 7,200 Tris (44%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un	Antes de iniciar la interacción, hubo una disminución de *draw calls* de un 12.2% y de

			13% y de 58.9% en polígonos.	56% en polígonos.
Reproducción de la animación de orbitales.	706 *draw calls* (100%) 14,300 Tris (100%)	698 *draw calls* (100%) 12,300 Tris (100%)	645 *draw calls* (91.4%) 8,200 Tris (57.3%)	637 *draw calls* (91.3%) 7,200 Tris (58.5%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 8.6% y de 42.7% en polígonos.	Antes de iniciar la interacción, hubo una disminución de *draw calls* de un 8.7% y de 41.5% en polígonos.
Expansión de orbitales.	690 *draw calls* (100%) 11,300 Tris (100%)	526 *draw calls* (100%) 10,200 Tris (100%)	641 *draw calls* (92.9%) 7,200 Tris (70.6%)	479 *draw calls* (91%) 6,100 Tris (60%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 7.1% y de 29.4% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 9% y de 40% en polígonos.
Contracción de orbitales.	692 *draw calls* (100%) 11,300 Tris (100%)	534 *draw calls* (100%) 11,300 Tris (100%)	637 *draw calls* (92%) 7,200 Tris (63.7%)	433 *draw calls* (81.1%) 6,100 Tris (54%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 8% y de 36.3% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 18.9% y de 46% en polígonos.
Interrupción de la animación de orbitales estando expandidos.	698 *draw calls* (100%) 12,300 Tris (100%)	614 *draw calls* (100%) 17,400 Tris (100%)	623 *draw calls* (89.2%) 7,200 Tris (58.5%)	518 *draw calls* (84.4%) 8,200 Tris (47%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 10.8% y de	Al terminar la interacción, hubo una disminución de *draw calls* de un 15.6% y de

			41.5% en polígonos.	53% en polígonos.
Reproducción de la animación de orbitales estando expandidos.	698 *draw calls* (100%) 12,300 Tris (100%)	666 *draw calls* (100%) 11,300 Tris (100%)	639 *draw calls* (91.5%) 7,200 Tris (58.5%)	557 *draw calls* (83.6%) 7,200 Tris (63.7%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 8.5% y de 41.5% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 16.4% y de 36.3% en polígonos.
Selección de un elemento dentro de la tabla periódica.	281 *draw calls* (100%) 1,000 Tris (100%)	43 *draw calls* (100%) 1,000 Tris (100%)	265 *draw calls* (94.3%) 1,000 Tris (100%)	23 *draw calls* (53.5%) 0 Tris (0%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 5.7% y de 0% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 46.5% y de 100% en polígonos.
Animación de rotación de los temas.	102 *draw calls* (100%) 9,200 Tris (100%)	100 *draw calls* (100%) 9,200 Tris (100%)	67 *draw calls* (65.7%) 5,100 Tris (55.4%)	65 *draw calls* (65%) 5,100 Tris (55.4%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 34.3% y de 44.6% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 35% y de 44.6% en polígonos.
Selección de uno de los temas y visualización de la información.	346 *draw calls* (100%) 7,200 Tris (100%)	100 *draw calls* (100%) 9,200 Tris (100%)	297 *draw calls* (85.8%) 2,000 Tris (27.7%)	51 *draw calls* (51%) 4,100 Tris (44.6%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 14.2% y de	Al terminar la interacción, hubo una disminución de *draw calls* de un 49% y de

			72.3% en polígonos.	55.4% en polígonos.
Ejecución del nivel de AR.	16 *draw calls* (100%) 68 Tris (100%)	5 *draw calls* (100%) 0 Tris (100%)	16 *draw calls* (100%) 4 Tris (5.9%)	5 *draw calls* (100%) 0 Tris (100%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 0% y de 94.1% en polígonos.	Antes de iniciar el nivel de AR no hubo cambio alguno.
Animación del contenedor de los temas (AR).	16 *draw calls* (100%) 74 Tris (100%)	16 *draw calls* (100%) 123 Tris (100%)	16 *draw calls* (100%) 10 Tris (13.5%)	16 *draw calls* (100%) 59 Tris (47.9%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 0% y de 86.5% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 0% y de 52.1% en polígonos.
Selección de un elemento dentro de la tabla periódica (AR).	26 *draw calls* (100%) 68 Tris (100%)	16 *draw calls* (100%) 97 Tris (100%)	26 *draw calls* (100%) 4 Tris (5.9%)	16 *draw calls* (100%) 33 Tris (34%)
			Esta interacción presentó una mejora en la disminución de *draw calls* de un 0% y de 94.1% en polígonos.	Al terminar la interacción, hubo una disminución de *draw calls* de un 0% y de 66% en polígonos.
Visualización del átomo en AR.	26 *draw calls* (100%) 0 Tris	26 *draw calls* (100%) 0 Tris	26 *draw calls* (100%) 0 Tris	26 *draw calls* (100%) 0 Tris
			No hubo ningún cambio al visualizar el átomo en AR.	

Fuente: Elaboración propia.

Con respecto al renderizado de elementos dentro de los niveles, hubo una mejoría en la cantidad de *draw calls* y polígonos en la mayoría de las interacciones. Un caso importante

es el de la selección de un elemento dentro del nivel de VR donde se desactivan los *colliders* pertenecientes a los elementos de la tabla periódica y esto ocasiona la disminución en la cantidad de polígonos.

También existen los casos de la visualización de información en el nivel de VR y de la animación del contenedor de temas en el nivel de AR, en el que en ambas versiones se obtuvo una disminución en *draw calls* (nivel de VR) o no hubo cambio alguno (nivel de AR), se presentó un aumento de polígonos debido a la cantidad de texto mostrado después de la interacción, pero entre versiones la versión optimizada presentó mejor desempeño.

Tabla 8. Comparación de resultados obtenidos del componente Memory.

Componente: Memory (Memoria)	Se midió la cantidad de memoria en MB (megabytes) y KB (kilobytes) usada por las texturas, materiales y mallas en cada una de las interacciones (valores máximos), y antes o después de éstas (valores mínimos).	
Interacción	**Versión no optimizada**	**Versión optimizada**
	Valor máximo / **Valor mínimo**	**Valor máximo** / **Valor mínimo**
Ejecución del nivel de VR. Interrupción de la animación de orbitales. Reproducción de la animación de orbitales. Expansión de orbitales. Contracción de orbitales. Interrupción de la animación de orbitales estando expandidos.	Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones (Excepto antes de la ejecución del nivel). Texturas: 620 MB Materiales: 115 KB Mallas: 0.8 MB	Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones (Excepto antes de la ejecución del nivel). Texturas: 461.9 MB Materiales: 96 KB Mallas: 0.6 MB En esta versión se obtuvo una mejora de 25.5% en texturas, 16.5% en materiales y de 25% en mallas de *modelos 3D* y otros elementos dentro de la escena.

Reproducción de la animación de orbitales estando expandidos.				
Animación de rotación de los temas.				
Selección de un elemento dentro de la tabla periódica.	Texturas: 620 MB Materiales: 115 KB Mallas: 0.8 MB	Texturas: 620 MB Materiales: 115 KB Mallas: 0.7 MB	Texturas: 461.9 MB Materiales: 96 KB Mallas: 0.6 MB	Texturas: 461.9 MB Materiales: 96 KB Mallas: 0.48 MB
			Hubo una mejora de 25.5% en texturas, 16.5% en materiales y de 25% en mallas.	Al terminar la interacción únicamente se presentó una mejora de 31.4% en mallas debido a que dejan de renderizarse las imágenes de los elementos químicos, pero siguen almacenadas en la memoria del dispositivo.
Selección de uno de los temas y visualización de la información.	Texturas: 620 MB Materiales: 115 KB Mallas: 0.8 MB	Texturas: 620 MB Materiales: 115 KB Mallas: 0.9 MB	Texturas: 461.9 MB Materiales: 96 KB Mallas: 0.6 MB	Texturas: 461.9 MB Materiales: 96 KB Mallas: 0.7 MB
			Hubo una mejora de 25.5% en texturas, 16.5% en materiales y de 25% en mallas.	Al terminar la interacción únicamente hubo una mejora de 22.2% en mallas entre versiones, y se presentó un aumento con respecto al inicio de la

				interacción debido a la aparición de texto.
Ejecución del nivel de AR.	Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones (Excepto antes de la ejecución del nivel). Texturas: 314 MB Materiales: 11 KB Mallas: 91 KB		Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones (Excepto antes de la ejecución del nivel). Texturas: 253 MB Materiales: 11 KB Mallas: 91 KB	
Animación del contenedor de los temas (AR).			En esta versión se obtuvo una mejora únicamente del 19.4% en texturas.	
Selección de un elemento dentro de la tabla periódica (AR).	Texturas: 314 MB Materiales: 15 KB Mallas: 198 KB	Texturas: 314 MB Materiales: 11 KB Mallas: 91 KB	Texturas: 253 MB Materiales: 15 KB Mallas: 198 KB	Texturas: 253 MB Materiales: 11 KB Mallas: 91 KB
			Para esta interacción, este componente fue el único en el que el valor mínimo se produjo un frame antes del valor máximo, se obtuvo un aumento en materiales y mallas, pero entre versiones sólo hubo una mejora de 19.4% en texturas.	
Visualización del átomo en AR.	Texturas: 314 MB Materiales: 15 KB Mallas: 198 KB	Texturas: 11.9 MB Materiales: 18 KB Mallas: 5 KB	Texturas: 253 MB Materiales: 15 KB Mallas: 198 KB	Texturas: 11.9 MB Materiales: 18 KB Mallas: 5 KB
			Se obtuvo una mejora únicamente del 19.4% en texturas.	Después de iniciar el modo AR no hubo mejoras o cambios con respecto a la versión no optimizada.

Fuente: Elaboración propia.

Todas las interacciones presentaron una disminución en la cantidad de memoria con respecto a la versión no optimizada. En algunos casos la cantidad de memoria usada en mallas decrementa o aumenta debido a la visualización de texto para mostrar información o características sobre el átomo.

Tabla 9. Comparación de resultados obtenidos del componente Global Illumination.

Componente: Global Illumination (Iluminación Global)	Se midió el tiempo en milisegundos que tomó la iluminación de la escena y que forma parte del tiempo total de la CPU en cada acción o interacción.			
Interacción	**Versión no optimizada**		**Versión optimizada**	
	Valor máximo	**Valor mínimo**	**Valor máximo**	**Valor mínimo**
Ejecución del nivel de VR.	4.44 ms (100%)	0 ms (100%)	Debido a la implementación de *Baked Global Illumination*, en esta versión no se realizó ningún proceso de iluminación o sombreado en tiempo real, por lo tanto, no se obtuvo ninguna demanda de procesamiento por parte de este componente. Respecto a los resultados obtenidos en la versión no optimizada, las interacciones presentaron una mejora de 100% en este componente, lo que significa también una mejora en la CPU.	
Interrupción de la animación de orbitales.	0.5 ms (100%)	0.15 ms (100%)		
Reproducción de la animación de orbitales.	2.45 ms (100%)	0.15 ms (100%)		
Expansión de orbitales.	0.89 ms (100%)	0.12 ms (100%)		
Contracción de orbitales.	2.72 ms (100%)	0.15 ms (100%)		
Interrupción de la animación de orbitales estando expandidos.	1.47 ms (100%)	0.15 ms (100%)		
Reproducción de la animación de orbitales estando expandidos.	2.25 ms (100%)	0.14 ms (100%)		

Selección de un elemento dentro de la tabla periódica.	1.23 ms (100%)	0.37 ms (100%)	
Animación de rotación de los temas.	0.4 ms (100%)	0.4 ms (100%)	
Selección de uno de los temas y visualización de la información.	0.19 ms (100%)	0.19 ms (100%)	
Ejecución del nivel de AR.	Este nivel carece de *modelos 3D* o *assets* que requieran de iluminación para mejorar la experiencia de usuario, por lo tanto, está compuesto únicamente de elementos 2D, tales como imágenes y texto. No se obtuvo ninguna demanda de procesamiento por parte de este componente en ninguna de las versiones de la aplicación.		
Animación del contenedor de los temas (AR).			
Selección de un elemento dentro de la tabla periódica (AR).			
Visualización del átomo en AR.			

Fuente: Elaboración propia.

Tabla 10. Comparación de resultados obtenidos del componente Physics.

Componente: Physics (Físicas)	Se midió la cantidad de *colliders* en cada interacción para el uso de *raycast*.			
Interacción	**Versión no optimizada**		**Versión optimizada**	
	Valor máximo	**Valor mínimo**	**Valor máximo**	**Valor mínimo**
Ejecución del nivel de VR.	Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones. 133 *colliders*		Los valores obtenidos fueron constantes antes, durante y después de cada una de estas interacciones. 133 *colliders*	
Interrupción de la animación de orbitales.				

Reproducción de la animación de orbitales.				
Expansión de orbitales.				
Contracción de orbitales.				
Interrupción de la animación de orbitales estando expandidos.				
Reproducción de la animación de orbitales estando expandidos.				
Animación de rotación de los temas.	12 *colliders*	12 *colliders*	12 *colliders*	12 *colliders*
			No hubo cambio alguno entre versiones. El decremento entre ésta y las demás interacciones se debe a que la mayoría de los *colliders* pertenecen a la tabla periódica (en esta interacción no se visualiza y por lo tanto el profiler no mide la cantidad de *colliders* pertenecientes a ésta).	
Selección de un elemento dentro de la tabla periódica.	133 *colliders*	14 *colliders*	133 *colliders*	14 *colliders*
			No hubo cambio alguno entre versiones. El decremento entre los valores máximo y mínimo se debe a que se dejan de renderizar las imágenes de los elementos químicos y los *colliders* asociados a cada imagen son desactivados.	
Selección de uno de los temas y visualización de la información.	133 *colliders*	12 *colliders*	133 *colliders*	12 *colliders*
			No hubo cambio alguno entre versiones. El decremento entre los valores máximo y mínimo se debe a que se dejan de renderizar las imágenes de los elementos químicos y se visualiza la información relacionada al tema seleccionado.	

Ejecución del nivel de AR.	Los valores obtenidos fueron nulos antes, durante y después de cada una de estas interacciones en ambas versiones de la aplicación. 0 *colliders*			
Animación del contenedor de los temas (AR).				
Selección de un elemento dentro de la tabla periódica (AR).				
Visualización del átomo en AR.	1 *collider*	1 *collider*	1 *collider*	1 *collider*
			No hubo cambio alguno entre versiones, y el decremento entre ésta y las demás interacciones se debe a que el modo AR únicamente hace uso de un *collider* para implementar *raycast*.	

Fuente: Elaboración propia.

En este componente no se obtuvieron mejoras de desempeño entre las versiones de la aplicación. Los decrementos en la cantidad de *colliders* en algunas interacciones significan una mejora de desempeño durante éstas únicamente y comparando entre secciones de la misma versión.

4.3. EXPERIENCIA DE USUARIO

Para comprobar la viabilidad de desarrollo de la aplicación junto con las pruebas de rendimiento, se diseñó un instrumento para evaluar la percepción de treinta estudiantes de ingeniería en computación al usar ambas versiones de la aplicación. La evaluación tuvo el objetivo de recolectar datos suficientes para determinar si la experiencia del usuario al usar la versión optimizada de la aplicación fue afectada, positiva o negativamente.

A continuación, se muestra el instrumento diseñado para la recolección de datos de los usuarios, elaborado con base en las interacciones mencionadas anteriormente en la tabla

4 y dividido en dos secciones (nivel de VR y nivel de AR). Con el fin de comparar los resultados obtenidos al usar cada versión de la aplicación, el instrumento fue implementado después del uso de cada una de éstas.

Nivel de realidad virtual.

Las interacciones con el átomo fueron fáciles.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

Las interacciones con la tabla periódica fueron fáciles.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

La interacción para desplazar el texto informativo del elemento químico fue fácil.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

El desplazamiento dentro de la escena fue fluido y permitió interactuar con las distintas secciones.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

La escena carece de “congelamientos”, retrasos en las reproducciones o de animaciones no fluidas.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

Las texturas de los objetos y las imágenes dentro de la escena no son “borrosas”.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

La escena contiene sombras con igual opacidad en todos los objetos (no existen sombras más oscuras o claras con base en la distancia de los objetos al átomo).

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

El tiempo que tomó pausar o reproducir la animación y expandir o contraer los orbitales del átomo fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

El tiempo que tomó mostrar el átomo e información del elemento químico después de su selección en la tabla periódica fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

El tiempo que tomó desplazar el texto informativo del elemento químico fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

Nivel de realidad aumentada.

La animación al desplegar el menú lateral fue fluida.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

Las interacciones con la tabla periódica fueron fáciles.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

La interacción con el menú lateral fue fácil.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

Las animaciones del átomo fueron fluidas.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

La escena carece de “congelamientos”, retrasos en las reproducciones o de animaciones no fluidas.

Muy en desacuerdo.	En desacuerdo.	De acuerdo.	Muy de acuerdo.

El tiempo que tomó mostrar la información de un elemento químico al ser seleccionado en la tabla periódica fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

El tiempo que tomó mostrar y ocultar el menú lateral fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

El tiempo después de seleccionar la opción “AR” para visualizar el átomo en realidad aumentada fue:

Demasiado largo.	Largo.	Corto.	Demasiado corto.

Los pasos realizados para la medición de la experiencia de usuario fueron los siguientes:

- Se explicó a los estudiantes el objetivo del proyecto sin darles a conocer cuál de ambas versiones es la versión optimizada.
- Se permitió el uso de la versión optimizada a los estudiantes.
- Se aplicó el instrumento, y se indicó a los estudiantes responder con relación a la versión usada anteriormente.
- Se permitió el uso de la versión no optimizada a los estudiantes.
- Se aplicó el instrumento, y se indicó a los estudiantes responder con relación a la versión usada anteriormente.

4.3.1. Comparación de datos

Los siguientes gráficos muestran los resultados obtenidos en cada una de las preguntas pertenecientes al nivel de AR de cada versión.

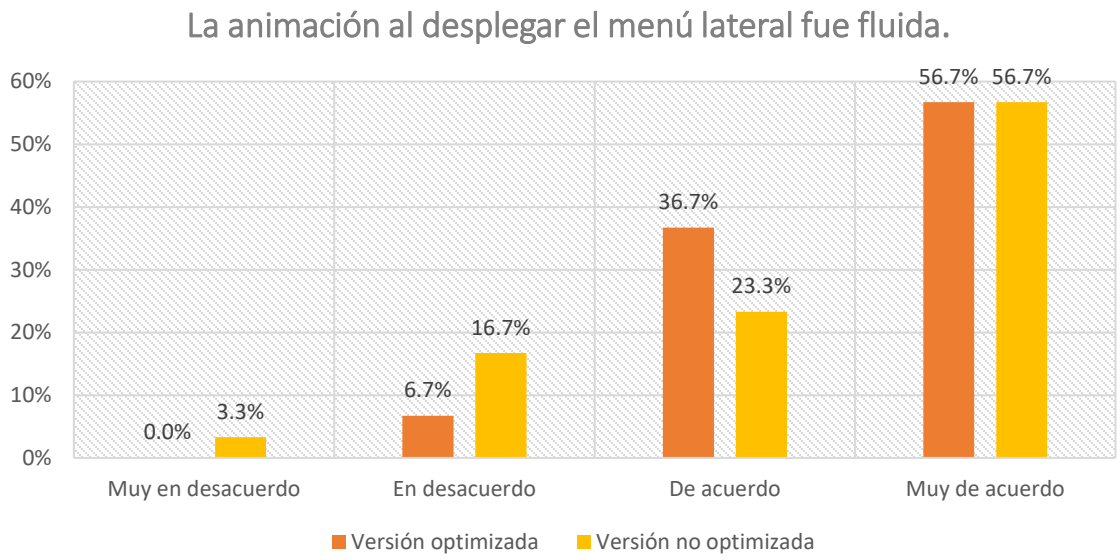


Figura 52. Comparación de resultados de la pregunta 1 del nivel de AR. Fuente: Elaboración propia.

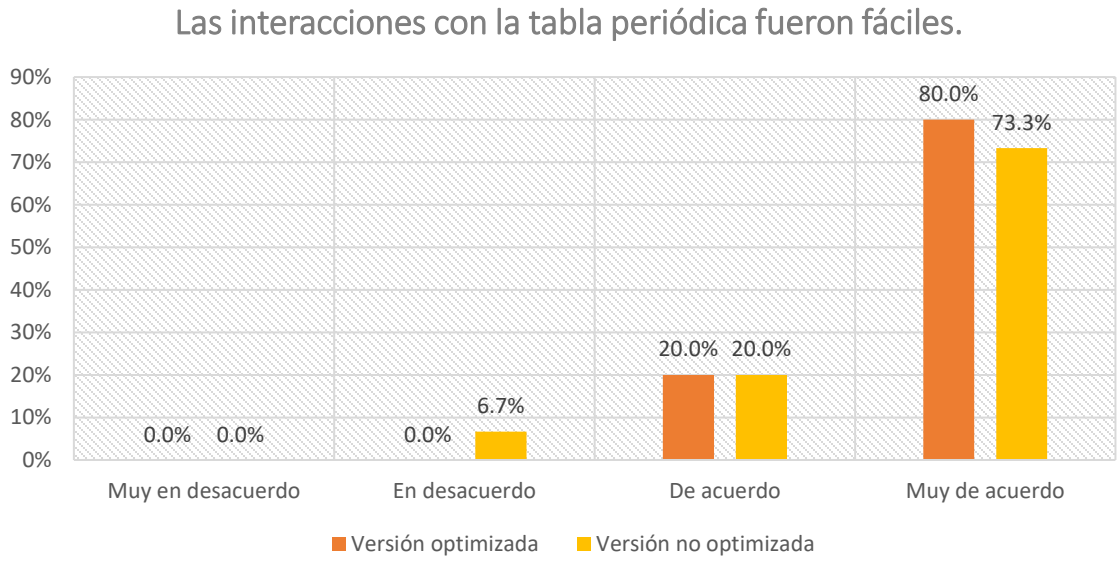


Figura 53. Comparación de resultados de la pregunta 2 del nivel de AR. Fuente: Elaboración propia.

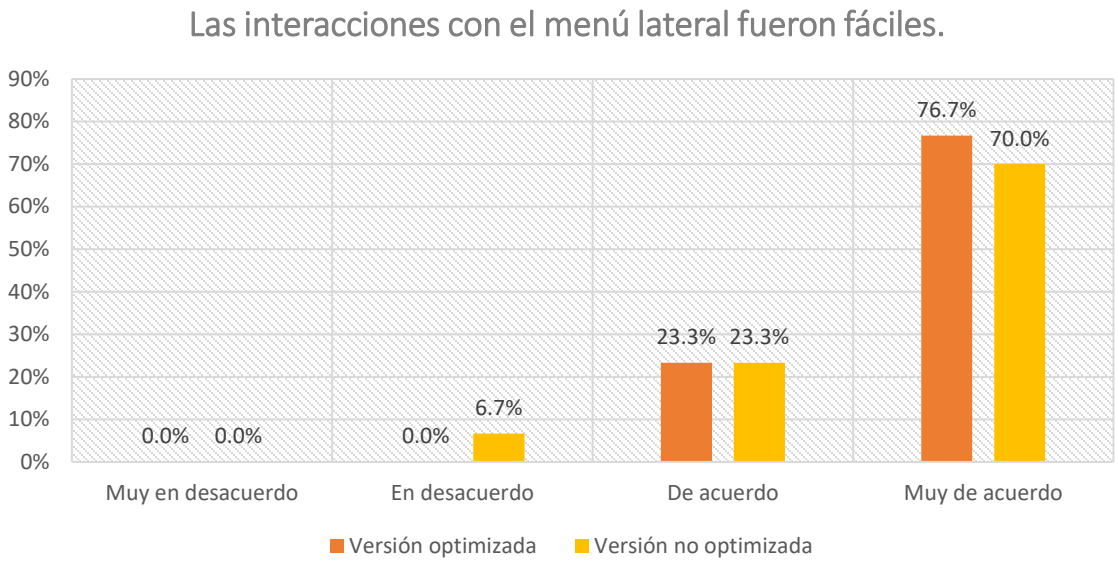


Figura 54. Comparación de resultados de la pregunta 3 del nivel de AR. Fuente: Elaboración propia.

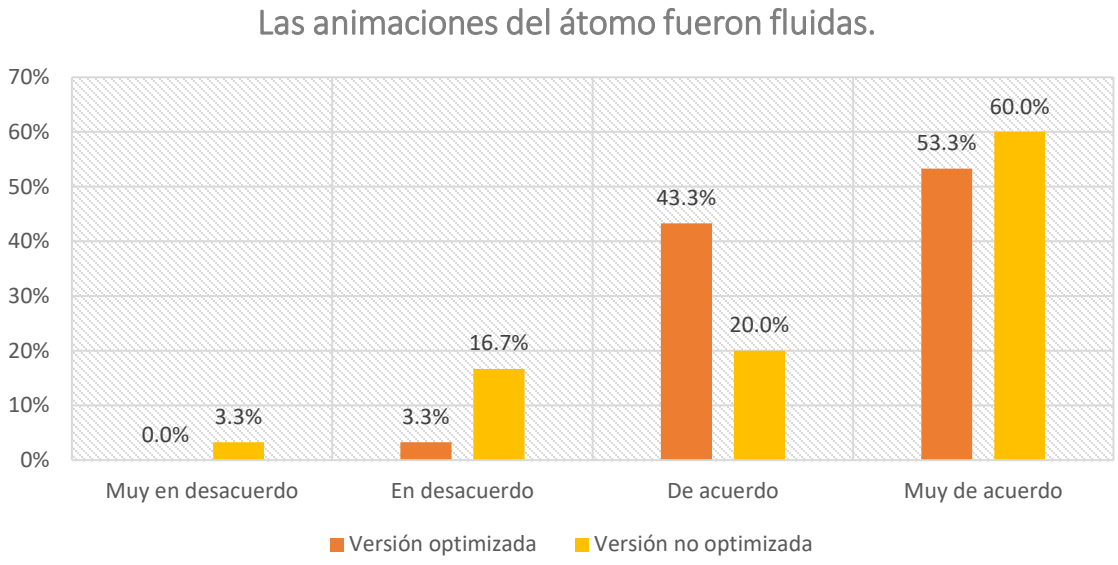


Figura 55. Comparación de resultados de la pregunta 4 del nivel de AR. Fuente: Elaboración propia.

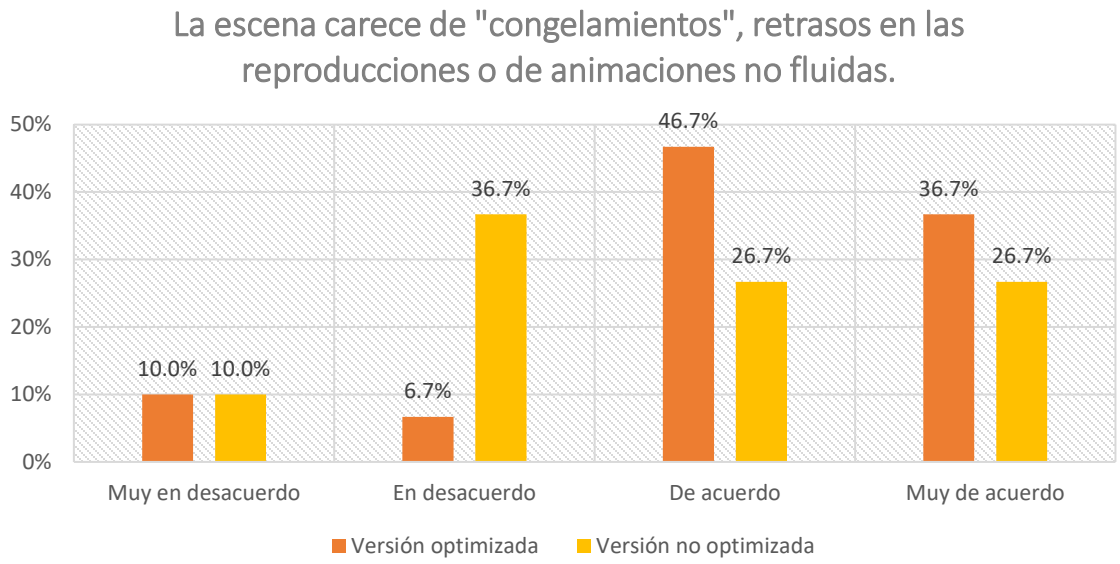


Figura 56. Comparación de resultados de la pregunta 5 del nivel de AR. Fuente: Elaboración propia.

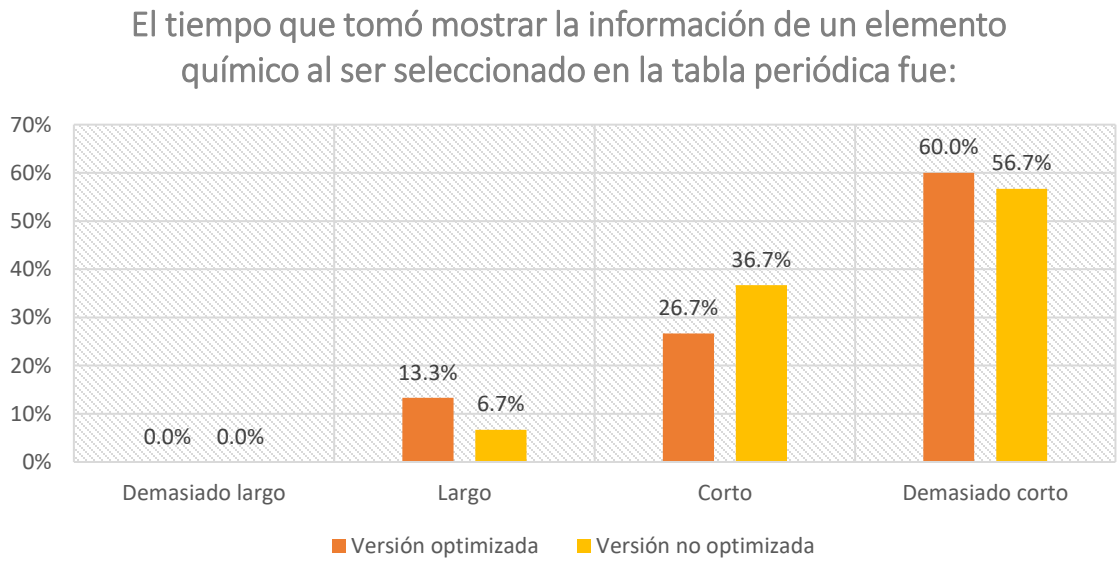


Figura 57. Comparación de resultados de la pregunta 6 del nivel de AR. Fuente: Elaboración propia.

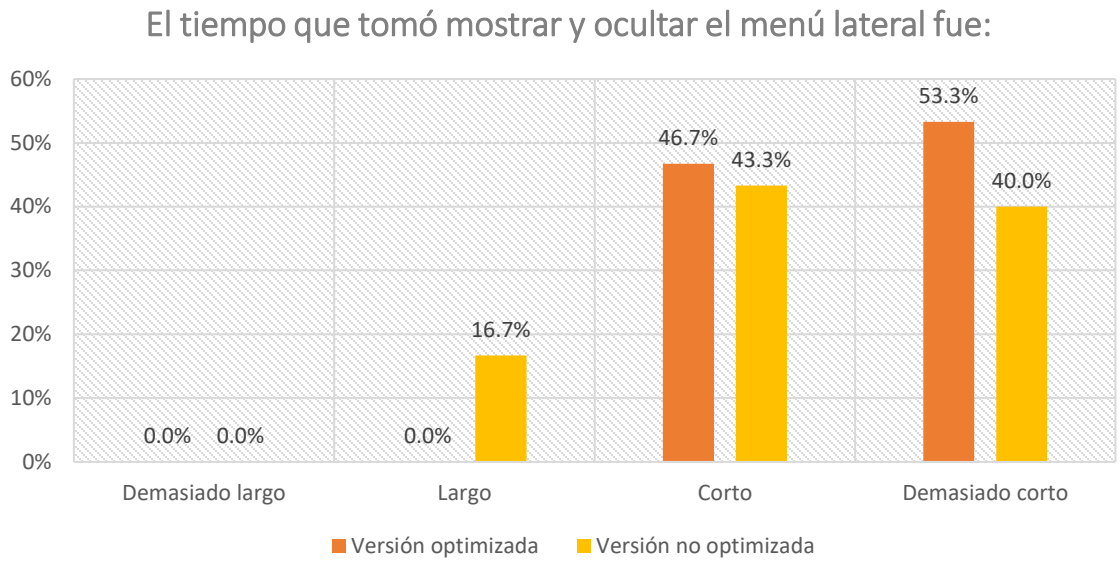


Figura 58. Comparación de resultados de la pregunta 7 del nivel de AR. Fuente: Elaboración propia.

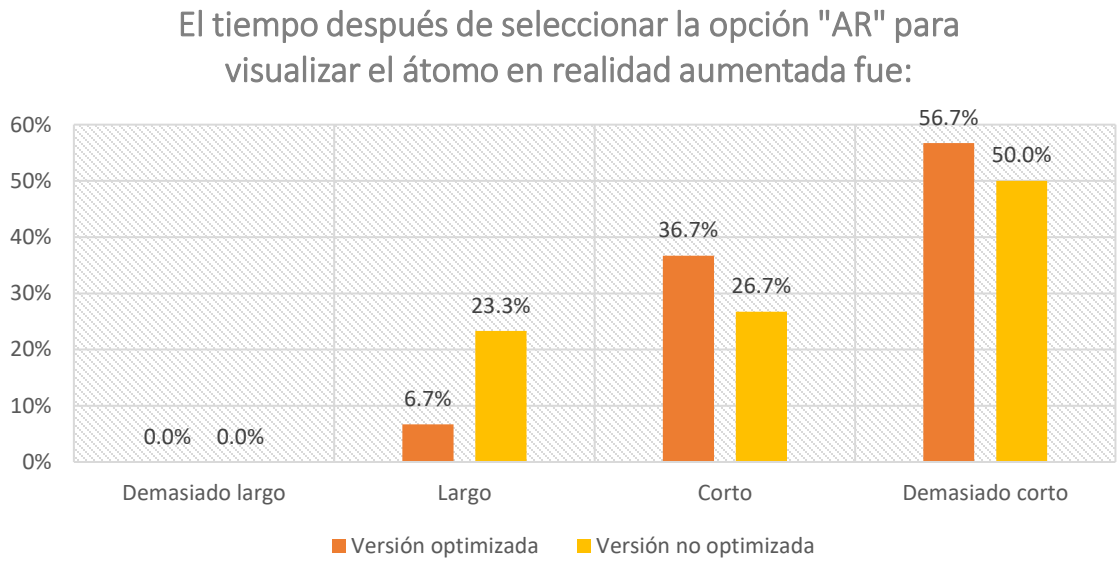


Figura 59. Comparación de resultados de la pregunta 8 del nivel de AR. Fuente: Elaboración propia.

En la versión optimizada, el 97.5% de los estudiantes respondió “de acuerdo” o “demasiado de acuerdo” en que las interacciones fueron fáciles y las animaciones fluidas en comparación al 86.6% de la versión no optimizada. El 83.3% respondió “de acuerdo” o “demasiado de acuerdo” en que la escena carece de congelamientos o retrasos para la versión optimizada y 53.3% para la versión no optimizada. En cuestión de tiempos de

respuesta por parte de las interacciones, el 93.3% respondió “corto” o “demasiado corto” en la versión optimizada y 84.4% en la versión no optimizada.

A continuación, se muestran los resultados obtenidos en cada una de las preguntas pertenecientes al nivel de VR de cada versión.

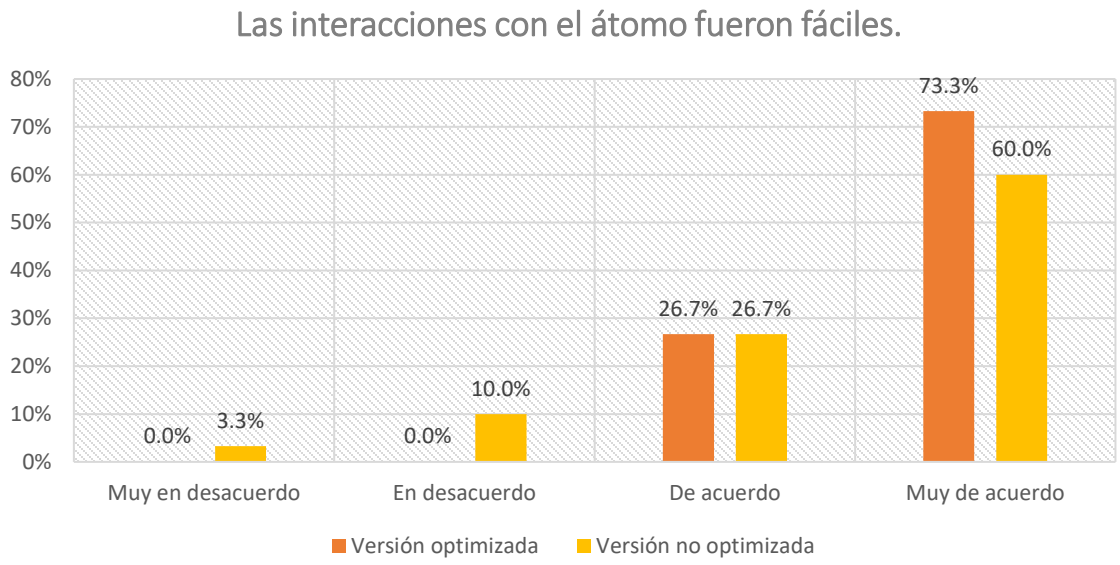


Figura 60. Comparación de resultados de la pregunta 1 del nivel de VR. Fuente: Elaboración propia.

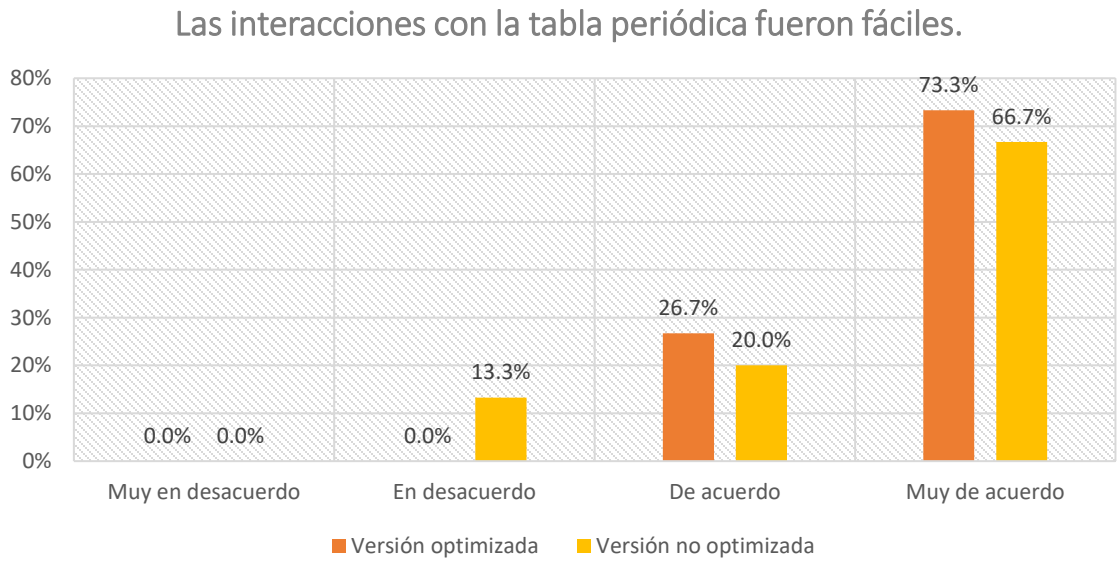


Figura 61. Comparación de resultados de la pregunta 2 del nivel de VR. Fuente: Elaboración propia.

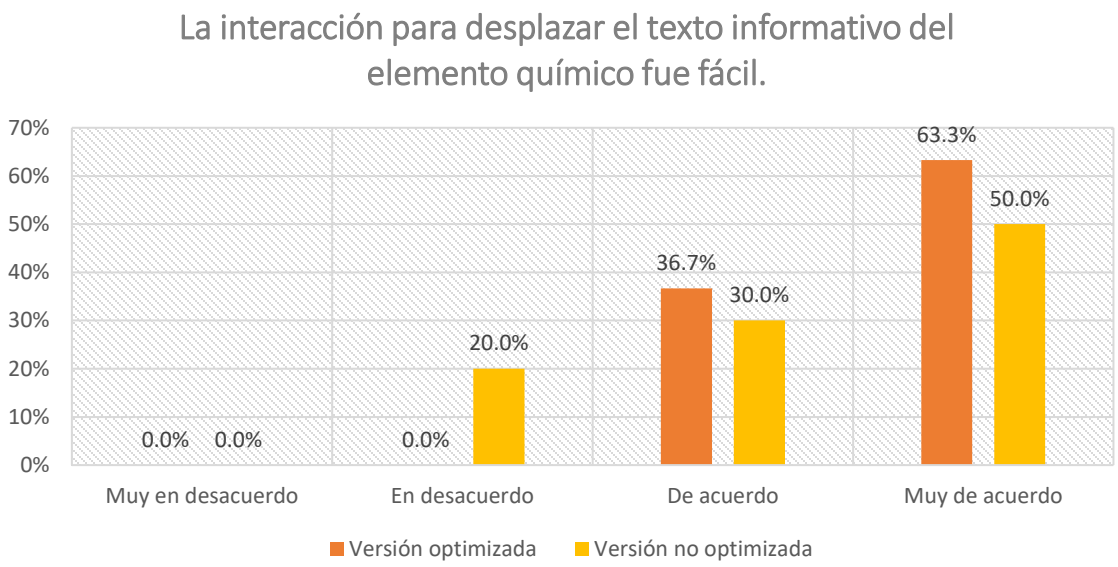


Figura 62. Comparación de resultados de la pregunta 3 del nivel de VR. Fuente: Elaboración propia.

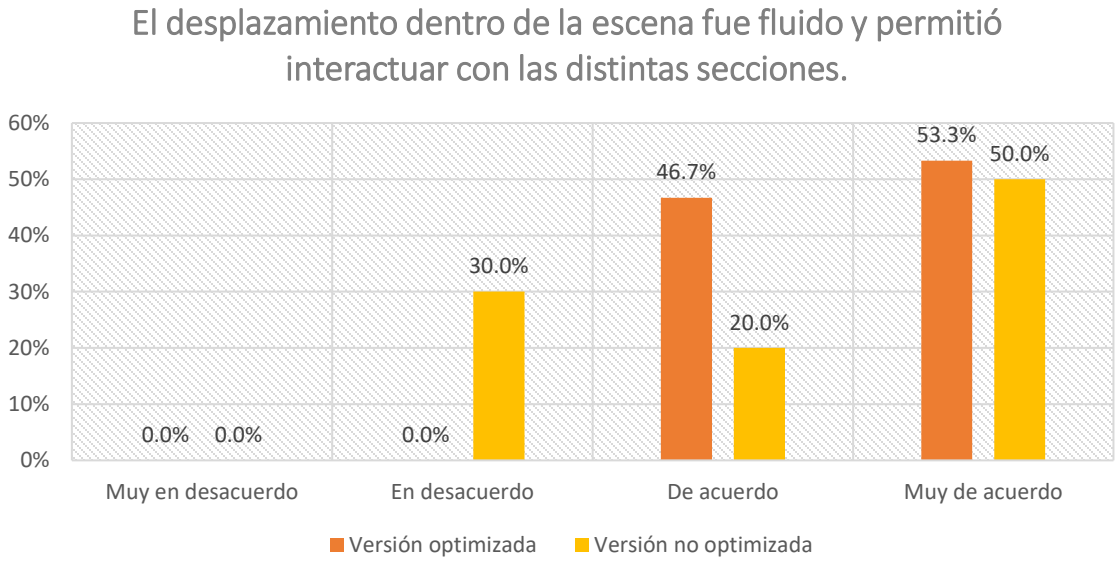


Figura 63. Comparación de resultados de la pregunta 4 del nivel de VR. Fuente: Elaboración propia.

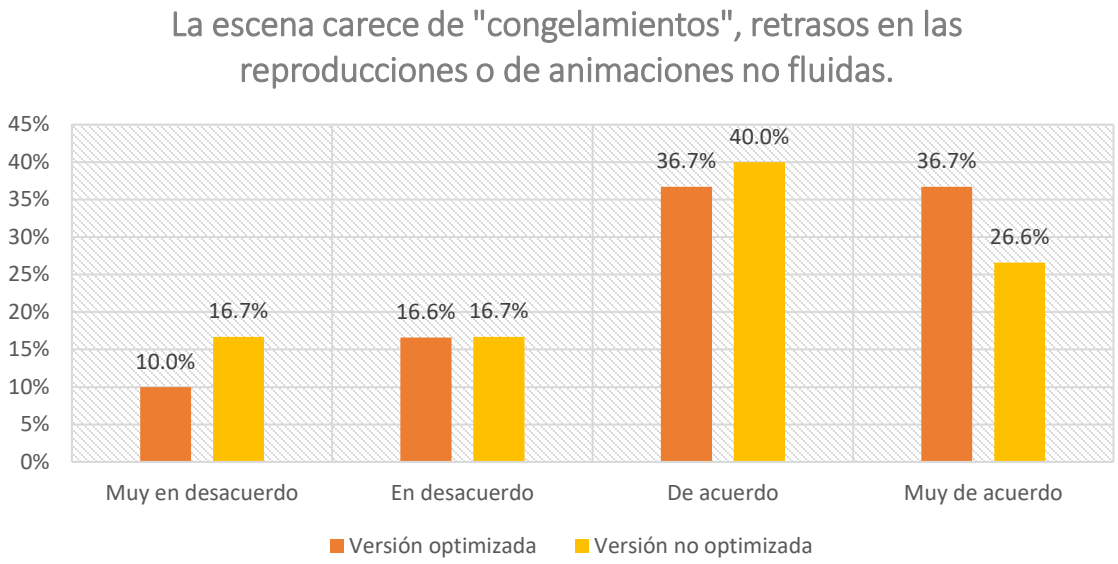


Figura 64. Comparación de resultados de la pregunta 5 del nivel de VR. Fuente: Elaboración propia.

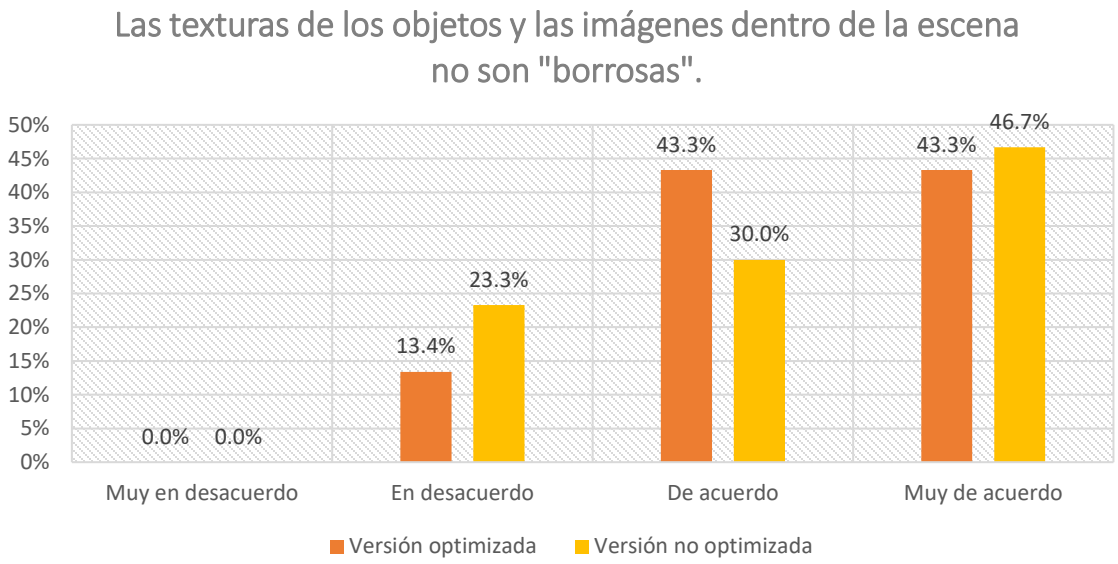


Figura 65. Comparación de resultados de la pregunta 6 del nivel de VR. Fuente: Elaboración propia.

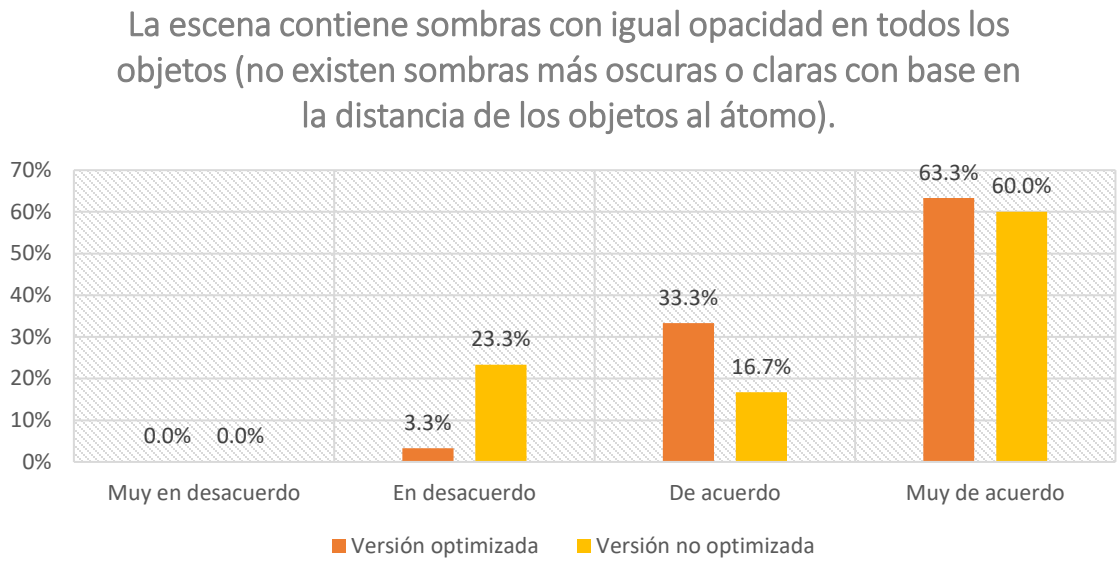


Figura 66. Comparación de resultados de la pregunta 7 del nivel de VR. Fuente: Elaboración propia.

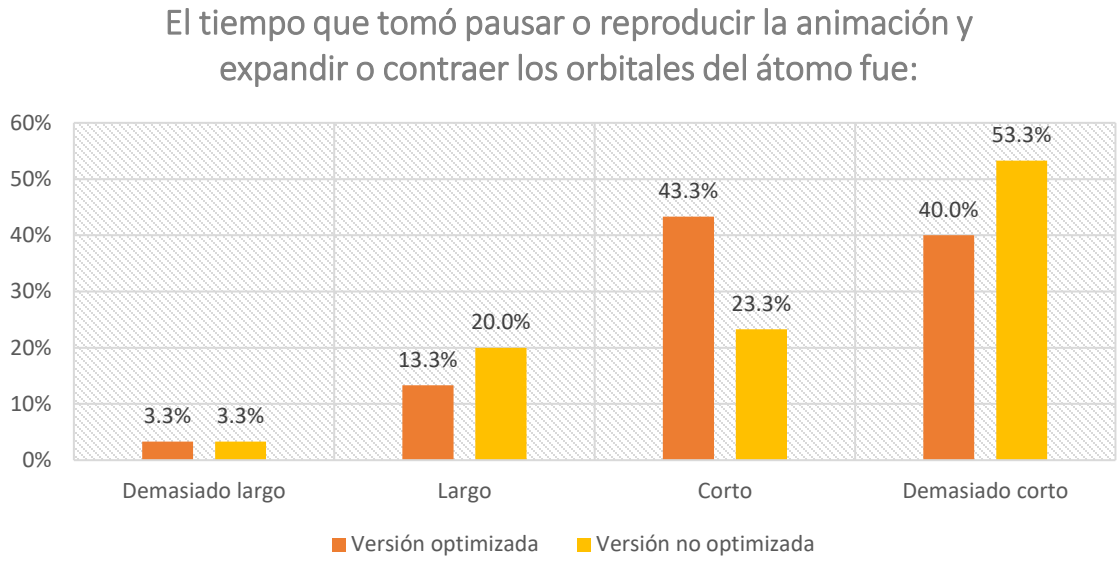


Figura 67. Comparación de resultados de la pregunta 8 del nivel de VR. Fuente: Elaboración propia.

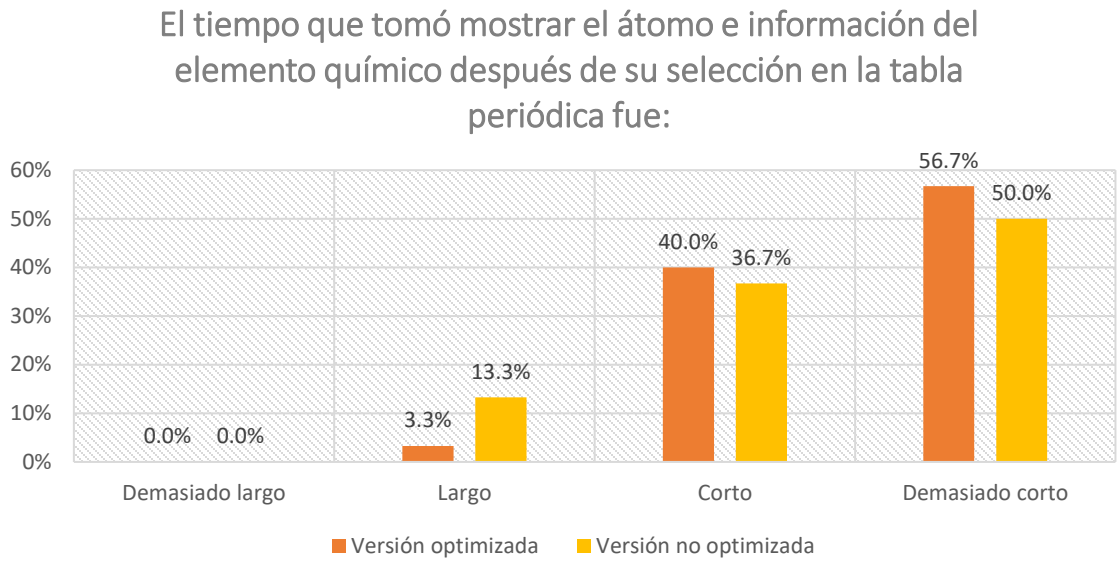


Figura 68. Comparación de resultados de la pregunta 9 del nivel de VR. Fuente: Elaboración propia.

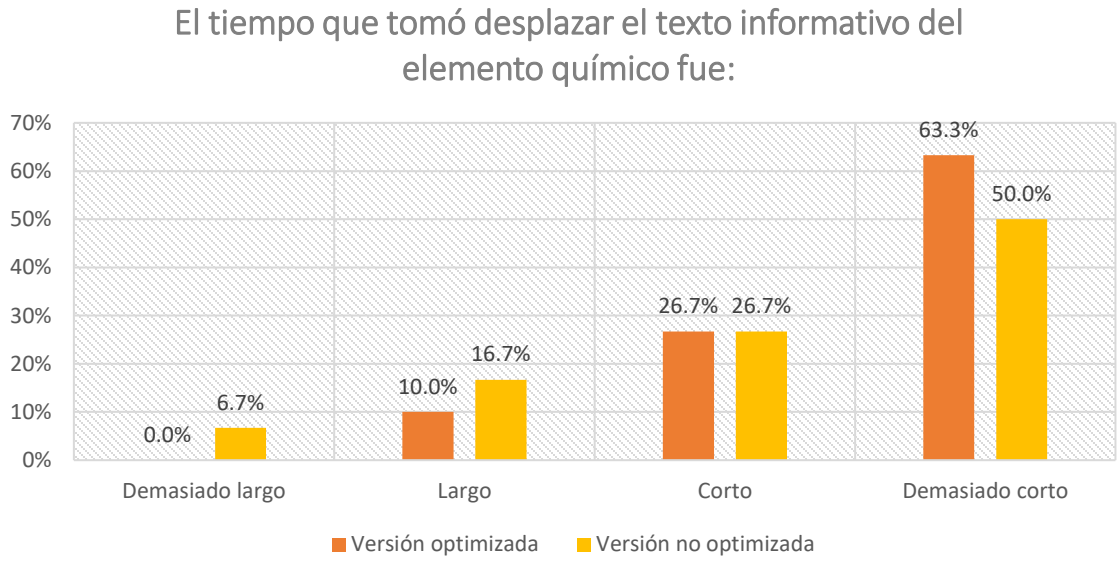


Figura 69. Comparación de resultados de la pregunta 10 del nivel de VR. Fuente: Elaboración propia.

Para el nivel de VR, el 97.2% de los estudiantes respondió “de acuerdo” o “demasiado de acuerdo” en que las interacciones fueron fáciles y las animaciones fluidas en la versión optimizada, y 79.4% en la versión no optimizada. El 73.4% respondió “de acuerdo” o “demasiado de acuerdo” en que la escena carece de congelamientos o retrasos para la

versión optimizada y 66.6% para la versión no optimizada. Y el 90% respondió “corto” o “demasiado corto” en la versión optimizada y 80% en la versión no optimizada con respecto a los tiempos de respuesta por parte de las interacciones.

De manera general, el siguiente gráfico muestra la comparación de los resultados obtenidos en las dieciocho preguntas por los treinta estudiantes en ambas versiones de la aplicación. El gráfico muestra que existe una diferencia pequeña pero notable entre las versiones, dicha diferencia es una mejora por parte de la versión optimizada debido a que el 57.8% de los estudiantes respondieron estar muy de acuerdo en que las animaciones fueron fluidas, las interacciones fáciles y los tiempos de respuesta de éstas fueron demasiado cortos, en comparación al 52.6% en la versión no optimizada. También se obtuvo que el 35.5% de los estudiantes respondieron para la versión optimizada estar de acuerdo con respecto a la facilidad de las interacciones, la fluidez de las animaciones y los tiempos cortos de respuesta, comparado con el 27.2% obtenido en la versión no optimizada.

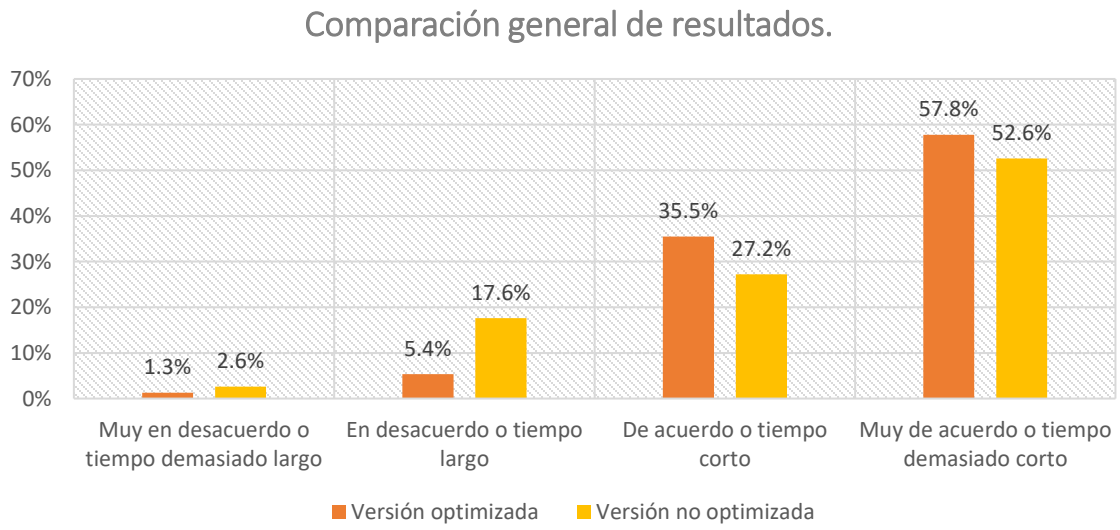


Figura 70. Comparación general de resultados. Fuente: Elaboración propia.

Por lo tanto, la implementación de técnicas de optimización obtuvo un 93.3% de aprobación por parte de los estudiantes y el desarrollo de la aplicación sin dichas técnicas un 79.8% con respecto a la experiencia de usuario.

CONCLUSIONES

El objetivo general y fundamental del presente trabajo era comprobar la viabilidad de desarrollo de una aplicación móvil con realidad virtual y aumentada utilizando técnicas de optimización para reducir la demanda de recursos de procesamiento sin afectar significativamente la experiencia de usuario. Para comprobar dicha viabilidad fue necesario el desarrollo de dos versiones del caso práctico, la identificación de métricas y la medición de éstas para la obtención de datos en ambas versiones, al igual que el diseño de un instrumento para la medición de la experiencia de usuario.

Con base en los resultados obtenidos en las pruebas de rendimiento y en la medición de la experiencia de usuario, el uso de técnicas de optimización fue viable en el desarrollo del presente caso práctico debido a la mejora en la experiencia de usuario que se obtuvo como una aprobación del 93.3% por parte de los estudiantes, y mejoras mínimas pero significantes en la demanda de recursos de procesamiento, principalmente en el nivel de VR.

En el caso de los objetivos específicos, todos fueron cumplidos sin dificultades durante el desarrollo del caso práctico y la obtención de los datos.

Durante el desarrollo del caso práctico y la obtención de datos se presentaron algunas dificultades, las cuales se enlistan a continuación:

- La elaboración de algunos *modelos 3D* presentó geometría indeseada o polígonos formados por *vértices* tan cercanos que parecían a simple vista ser uno solo, lo que implicó una revisión detallada de cada *modelo 3D* al implementar técnicas de optimización.
- En la obtención de datos mediante el profiler de Unity y el uso de ADB fue necesaria la correcta configuración del dispositivo móvil y de la computadora contenedora del caso práctico debido a que se presentaron errores en la obtención de los datos para algunas interacciones.

El desarrollo de una aplicación de realidad virtual y aumentada con técnicas de optimización significó un análisis en cada fase para la correcta implementación de dichas técnicas, debido a que se estableció un límite de éstas buscando no afectar negativamente la experiencia de usuario.

El aprendizaje significativo radicó en la optimización de diseño y desarrollo del entorno virtual *inmersivo* (nivel de VR), debido a que cualquier experiencia de VR es conocida por su nivel de *inmersión* atractivo para los usuarios, permitiendo mirar casi desde cualquier ángulo el entorno virtual e interactuando con diversos objetos o elementos multimedia dentro de éste. Para el nivel de VR fueron considerados como aspectos importantes: el diseño visual (elementos multimedia gráficos como *modelos 3D* o imágenes), el tiempo de respuesta de las interacciones, la fluidez en las animaciones (incluso en aquellas que permiten interacción por parte del usuario), y la navegación dentro del entorno virtual. Los aspectos mencionados fueron considerados en ambas versiones del caso práctico, pero únicamente en la versión optimizada se mantuvo un estricto desarrollo e implementación de técnicas con la intención de no modificar de manera visual o interactiva el nivel y sólo mejorar la parte de lógica de éste.

La optimización de cada fase de desarrollo de un nivel de VR podría llegar a ser un tema importante y relativo a debatir, debido a que depende de cómo sea realizada por parte de cada desarrollador o equipo de desarrollo y del tipo de aplicación o experiencia que se busque desarrollar.

Trabajos futuros

La investigación realizada para el presente trabajo fue enfocada en el desarrollo para dispositivos móviles como smartphones o tablets, sin embargo, en la actualidad existen dispositivos especializados para su uso con aplicaciones de VR y AR, algunos de dichos dispositivos son; Oculus Rift, Oculus Go, HTC VIVE y HoloLens de Microsoft. Los

dispositivos mencionados disponen de un nivel de procesamiento elevado debido al hardware que los compone y que en su mayoría suelen conectarse de manera alámbrica a una computadora para usarla como unidad principal de procesamiento.

Mencionado lo anterior, se consideran como trabajos futuros para el presente tema de investigación:

- Comprobar la viabilidad de desarrollo de una aplicación de VR para dispositivos especializados utilizando técnicas de optimización para reducir la demanda de recursos de procesamiento sin afectar significativamente la experiencia de usuario.
- Realizar pruebas de rendimiento y obtención de datos del presente caso práctico en diferentes arquitecturas de procesadores diseñados para dispositivos móviles.
- Comprobar la viabilidad de desarrollo al aplicar técnicas de optimización en una aplicación de VR desarrollada para dispositivos especializados y ejecutarla en dispositivos móviles como smartphones sin afectar significativamente la experiencia de usuario.

REFERENCIAS BIBLIOGRÁFICAS

- Alba, E. & Chicano, J. (2005). “Software testing with evolutionary strategies”. En Proceedings of the 2nd International Workshop on Rapid Integration of Software Engineering Techniques, LNCS 3943, 50-65.
- Android Developers. (2016, febrero 23). Threading Performance 101. (Android Performance Patterns Season 5, Ep. 1). [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=qk5F6Bxqhr4&feature=emb_logo.
- Balakrishna, B. (2013). Augmented Reality: This Is The Future Calling. The Masterbuilder, June 2013, 198-200.
- Buckley, M. (2018, diciembre 1). Profiling in Unity. [Entrada en blog]. Recuperado de https://medium.com/@mattThousand/profiling-in-unity-7e7fcedbeed4.
- Buxton, B. & Fitzmaurice, G. (1998). "HMD's, Caves & Chameleons: A Human-Centric Analysis of Interaction in Virtual Space," Computer 6raphics (Proceedings of SIGGRAPH 98, Annual Conference Series) 32(4): 64-68.
- Cawood, S & Fiala, M. (2007). Augmented Reality: A Practical Guide. Strickland Rd, #103-255: Pragmatic Bookshelf.
- Crecente, B. (2016, octubre 26). VR's long, weird history. [Entrada en blog]. Recuperado de https://www.polygon.com/2016/10/26/13401128/25-vr-greatest-innovators.
- Cruz, C., Sandin, D., DeFanti, T.Kenyon, R. & Hart, J. (1992). "The CAVE Audio Visual Experience Automatic Virtual Environment," Communications of the ACM 35(6): 65-72.
- Cushnan, D. & EL Habbak, H. (2013). Developing AR Games for IOS and Android. Birmingham B3 2PB, UK: Packt Publishing Ltd.
- Dickinson, C. (2015). Unity 5 Game Optimization. Birmingham B3 2PB, UK: Packt Publishing Ltd.

- Estayno, M., Dapozo, G., Cuenca, L. & Greiner, C. (2009). Modelos y métricas para evaluar calidad de software. Red de Universidades con Carreras en Informática (RedUNCI). 382-388.
- García Carrasco, J. M. (1992). La optimización: una mejora en la ejecución de programas. Ensayos: Revista de la facultad de Educación de Albacete, No. 7, 247-256.
- Google Developers. (2017, mayo 18). VR and AR at Google (Google I/O '17) [Archivo de vídeo]. Recuperado de https://www.youtube.com/watch?v=tto90e-DfeM
- Graham, S., Kessler, P. & Mckusick, M. (1982). Gprof: A call graph execution profiler. SIGPLAN ´82 Proceedings of the 1982 SIGPLAN symposium on Compiler construction, 120-126.
- Hassan, Y. (2017). Experiencia de Usuario: Principios y Métodos. Independently Published.
- Hassenzahl, M. & Tractinsky, N. (2006). User experience- a research agenda. Behaviour & Information Technology, Vol. 25, No. 2, 91-97.
- Hunicke, R. LeBlanc, M. & Zubek, R. (2004). MDA: A Formal Approach to Game Design and Game Research. AAAI Wirkshop-Technical Report.
- IronEqual. (2017, agosto 18). Unity: Android Optimization Guide. [Entrada en blog]. Recuperado de https://medium.com/ironequal/android-optimization-with-unity-3504b34f00b0.
- Labschütz, M., Krösl, K., Aquino, M., Grashäftl, F. & Kohl, S. (2011). Content Creation for 3D Game with Maya and Unity 3D, Institute of Computer Graphics and Algorithms, Vienna University of Technology.
- Morales, A., Alviter, L., Hidalgo, C., Amador, J., Zuñiga, J. & Mejía, C. (2015). Metodología de desarrollo evolutivo de escenarios tridimensionales para la creación de una visita virtual para el Centro Universitario UAEM Ecatepec. Revista Colombiana de Computación. Volumen 16, 7-27.
- Pérez, F. (2011). Presente y Futuro de la Tecnología de la Realidad Virtual. Creatividad y Sociedad. Volumen 16.

- Pérez, M., Zabre, E. & Islas, E. (2004). Realidad virtual: un panorama general, Boletín IIE, Instituto de Investigación Electrónicas, 28 (2) 39-44.
- Sherman, W. & Craig, A. (2002). Understanding Virtual Reality: Interface, Application, and Design. San Francisco, USA, Morgan Kaufmann Publishers.
- Srivastava, A. & Eustace, A. (1994). ATOM: a system for building customized program analysis tools. PLDI '94 Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation, 196-205.
- Steed, A. & Julier, S. (2013). Design and implementation of an Immersive Virtual Reality System based on a Smartphone Platform. 3D User Interfaces (3DUI), 2013 IEEE Symposium on.
- Unity Technologies. (2019). CPU Usage Profiler module. [Entrada en blog]. Recuperado de https://docs.unity3d.com/Manual/ProfilerCPU.html.
- Unity Technologies. (2019). Draw call batching. [Entrada en blog]. Recuperado de https://docs.unity3d.com/Manual/DrawCallBatching.html.
- Unity Technologies. (2019). Optimizing Scripts. [Entrada en blog]. Recuperado de https://docs.unity3d.com/Manual/MobileOptimizationPracticalScriptingOptimizations.html.
- Vince, J. (1995). Virtual Reality Systems. Wokingham, England: Addison-Wesley.
- Watkins, A. (2011). Creating Games with Unity and Maya How to Develop Fun and Marketable 3D Games. 30 Corporate Drive, Suite 400, Burlington, MA 01803USA: Else.