Universidad del País Vasco Euskal Herriko Unibertsitatea
VITORIA-GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

Grado en Ingeniería Informática de Gestión de Sistemas de Información

TRABAJO FIN DE GRADO

Realidad Virtual en Unity

Adaptación del Baloncesto en Silla de Ruedas a un entorno VR

Alumno: Ramos Perdomo, David

Director: González Nalda, Pablo

Curso: 2021-2022

Fecha: 5 de Abril de 2022

Resumen

Este documento recoge la memoria del Trabajo Fin de Grado del alumno David Ramos Perdomo del curso 2020/2021 de la Escuela de Ingeniería de Vitoria-Gasteiz en el Grado en Ingeniería Informática de Gestión y Sistemas de Información.

La finalidad del trabajo desarrollado es realizar un estudio de las mecánicas y movimientos característicos del Baloncesto en Silla de Ruedas, con el objetivo de desarrollar una aplicación de realidad virtual donde simular estos movimientos, creando una experiencia interactiva con distintos niveles de dificultad. Para desarrollar la aplicación se utilizará Unity como motor gráfico, además de otras herramientas como Blender para el modelado 3D de las figuras y personajes.

Palabras claves: Realidad Virtual, Baloncesto en Silla de Ruedas, Unity.

Abstract

This document contains the memory of the Final Degree Project of the student David Ramos Perdomo in the course 2020/2021 of the School of Engineering of Vitoria-Gasteiz in the Degree in Computer Engineering of Management and Information Systems.

The purpose of the work developed is to study the mechanics and characteristic movements of Wheelchair Basketball, with the goal of developing a virtual reality application to simulate these movements, creating an interactive experience with different levels of difficulty. To develop the application, Unity will be used as the graphic engine, in combination with other tools such as Blender for the 3D modeling of the figures and characters.

Keywords: Virtual Reality, Wheelchair Basketball, Unity.

Índice General

Índice de Tablas

Índice de Figuras

1. Introducción

La informática, la ciencia que estudia el tratamiento automático de la información, se ha introducido en prácticamente todas las tareas que desarrollamos en nuestro día a día. Más del 50 % de la población actual tiene acceso a internet mediante un ordenador o cualquiera de sus variantes (teléfonos móviles, tabletas, etc...). Por esta razón, es muy interesante aprovechar la masificación de la tecnología con el fin de agilizar tareas o simplificar procesos complejos.

La realidad virtual es una de las ramas de la informática con mayor auge en los últimos años. Ha demostrado un gran potencial para simular, con gran realismo, situaciones del mundo real, sin que suponga ningún riesgo para el usuario. Desde entrenar a trabajadores de centrales nucleares, hasta crear aplicaciones para crear reflejos en deportistas. Por esta razón, la realidad virtual como concepto es ideal para desarrollar herramientas que necesiten de un alto grado de inmersión y sensación de realismo.

1.1 Contexto

En el punto en el que estamos como sociedad los deportes adaptados no son ninguna novedad en el ambiente. Esto es algo que se ha desarrollado y trabajado a lo largo del tiempo, contando ya con deportes muy consolidados. El Baloncesto en Silla de Ruedas no es la excepción, y a pesar de que lleva bastantes años con una base competitiva muy sólida, carece de popularidad si lo comparamos con deportes más "mainstream" como el fútbol. Por esta razón es muy interesante crear estudios que aporten valor y ayuden a que el BSR o Baloncesto en Silla de Ruedas sea más conocido por la gente de a pie.

Sumando a lo anteriormente dicho, es indudable que la empatía es un recurso muy importante que todo ser humano debe tener desarrollado, y sobre todo los niños. La principal diferencia entre dos personas debe ser su mente, no su físico y lo que sea capaz de hacer con él. Por esta razón, ayudar a aceptar los diferentes tipos de discapacidades que pueda tener una persona es algo importante, sobre todo en edades tempranas. Este estudio tiene mucho que hacer en ese aspecto, puesto que no solo va dirigido a mejorar deportivamente a los jugadores del BSR, sino, a que se vea de una forma más normal y natural el deporte adaptado.

1.3 Objetivo del Proyecto

Con este proyecto se pretende crear una aplicación de realidad virtual que recoge ciertas características del BSR. La idea es estructurar la experiencia en tres situaciones, contando cada una de ellas con 3 niveles de dificultad. La primera situación sería el simple tiro a canasta, situándose al jugador en una posición cercana al aro para que pueda realizar un tiro. La segunda situación estará dirigida a la realización de pases, el jugador tendrá que dar un pase a un compañero. Por último tenemos la intercepción de pases, donde dos o más contrincantes van a realizar un pase entre ellos y el jugador tiene que interceptar dicho pase. A estas tres situaciones se le sumaría un nivel tutorial donde se explican las dinámicas de la aplicación, de tal forma que quede claro cómo interactuar con la aplicación.

Cabe destacar que cada una de estas situaciones tendrá diferentes niveles de dificultad. En la mayoría de los casos el nivel de dificultad aumentará al situarse más defensores o enemigos en el campo para dificultar las acciones del usuario.

1.4 Cómo surgió la idea

La idea de utilizar VR para simular situaciones de la vida real no es algo nuevo, se lleva haciendo desde hace años en otros sectores de la industria. En cambio, el sector del deporte adaptado está muy verde en este aspecto. Si lo comparamos con otros deportes como el fútbol, la falta de financiación y presupuesto se deja ver, siendo un factor determinante para el crecimiento de proyectos de este tipo.

Por motivos personales, puesto que soy jugador de Baloncesto en Silla de Ruedas, tenía la idea de desarrollar algo que aporte valor al deporte. Mi idea inicial era desarrollar e implementar sensores que pudieran ser colocados en las sillas utilizadas para practicar BSR, de tal forma que se pudiesen sacar estadísticas de cada jugador al final de un entrenamiento, distancia recorrida, agilidad con la silla y demás. La idea, aunque tenía potencial, en comparación con la temática de este proyecto carecía de replicabilidad y escalabilidad. Estas dos últimas palabras son muy importantes a la hora de decidir en qué proyecto invertir recursos, y puesto que la idea es aportar valor a un deporte infravalorado, es indispensable tener la mayor repercusión posible.

Por esta razón, la balanza se inclinó para un lado, desarrollar una aplicación en VR simulando ciertas características del BSR.

2. Planificación

Este apartado describe la estructura utilizada en la realización de la aplicación, las fases y tareas del proyecto, así como la agenda de trabajo y los recursos materiales empleados.

2.1 Análisis de requisitos

Los requisitos son pequeñas descripciones de las necesidades de un cliente o usuarios finales. Al redactar los requisitos se debe tener en cuenta la funcionalidad y el resultado esperado. Además, debe venir acompañado de los criterios de aceptación.

2.1.1. Requisitos funcionales iniciales

Realizar un breve estudio del BSR, para posteriormente centrar la atención en una serie de mecánicas y situaciones características del deporte:

- Situaciones de tiro, posiciones de los jugadores y defensa del mismo.
- Toma de decisión al dar un pase, qué se tiene que tener en cuenta para realizar un pase seguro y eficaz a un compañero.
- Situaciones de robo de balón, en qué jugadas y posiciones se puede dar esta acción.

Con los datos obtenidos se desarrollará una aplicación en VR que simule estas situaciones con el mayor realismo posible utilizando Unity como motor gráfico, y las Oculus Quest 2 como las gafas de VR.

2.1.2 Tecnología y herramientas a utilizar

Gafas de VR

La intención es desarrollar la aplicación para una plataforma que permita gran movilidad e independencia, de tal forma que las gafas de VR elegidas no necesiten cables de por medio. Esto es algo que mejora mucho la experiencia y la escalabilidad de la aplicación, puesto que al no depender de cables se le da mucha libertad de movimiento al usuario. En el mercado europeo, a fecha de realización de este proyecto, solo existe un modelo de gafas accesible al usuario común que cumple la premisa de no necesitar cables. Las Oculus Quest, en cualquiera de sus dos versiones, utilizan

Android como sistema operativo y cuentan con buena conectividad así como batería interna, permitiendo una autonomía de 2h. La potencia de renderizado no es muy alta, puesto que se asemeja mucho en características a un teléfono móvil de gama alta. A pesar de esto, es más que suficiente para mostrar un nivel de detalle óptimo y una experiencia fluida.

Entorno de desarrollo

Para el desarrollo de esta aplicación se ha elegido como motor gráfico Unity, por su gran modularidad y facilidad para configurar muchos aspectos de la escena. En cuanto a desarrollar los modelos y entorno 3D, se utilizará Blender por ser un software libre con una gran comunidad de usuarios detrás. En los Anexos I y II se detallan las características de estas dos herramientas.

2.2 Diagrama de Gantt

El diagrama de Gantt es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

En el caso de este proyecto se han definido 4 grandes bloques:

- Estudio y documentación del BSR
- Fase de aprendizaje, estudio de Unity y otras herramientas necesarias para el desarrollo.
- Desarrollo de la herramienta
- Pruebas de funcionamiento con usuarios. Retrospectiva.

El proyecto comienza el día 01 de febrero del 2021 y finaliza el 31 de agosto del mismo año. En la siguiente figura se puede apreciar la planificación en días dedicados por mes del proyecto, es decir, en el mes de febrero se dedicaros 15 días repartidos a lo largo del mes al estudio y documentación del BSR. Posteriormente se detallarán cuántas horas corresponden a los días de cada tarea.

Actividad	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto
Estudio y documentación del BSR	15 Días						
Unity		20 Días					
Blender			10 Días				
Desarrollo de la aplicación			10 Días	10 Días	20 Días	20 Días	

Pruebas con usuarios.							5 Días
Retrospectiva							10 Días

Figura 1. Diagrama de Gantt

2.3 Estimación de tiempos

En este apartado se visualizará una estimación de tiempos para las tareas que se han llevado a cabo en la realización del proyecto. Cabe destacar que no se está trabajando a tiempo completo en el proyecto y por esta razón los tiempos son aproximados.

Tabla 1: Estimación de tiempos

Actividad	Estimación del tiempo
Estudio del BSR - Definir los límites de la aplicación y las mecánicas y situaciones que se van a simular en VR.	15h de estudio, repartidas en: - tiempo de visualización de videos - asistencia a entrenos **TOTAL: 15h**
Unity, conocimiento de la herramienta - Curso en Udemy introductorio a Unity	65h de curso en Udemy 65h dedicadas a realizar ejercicios y tareas derivadas de los cursos. **TOTAL: 130h**
Blender, conocimiento de la herramienta - Curso introductorio de Blender realizado en la plataforma Udemy - Tutoriales avanzados en Youtube	25h curso Udemy 10h tutoriales de Youtube 10h de tareas derivadas del estudio. **TOTAL: 45h**
Desarrollo de la aplicación	

- Parte 1: desarrollo de los Assets a utilizar, así como preparar una primera demo con funcionalidades básicas. - Parte 2: Desarrollo de los niveles 1, 2 y 3 - Parte 3: Desarrollo del nivel tutorial	**TOTAL: 250h**
Revisión final - Prueba de la aplicación con usuarios - Solución de errores, ajustes en los niveles… etc.	**TOTAL: 50h**

Todas estas tareas, hacen una aproximación total de 490 horas para el desarrollo del proyecto.

2.4 Estimación de Costes

Aparte de considerar el factor humano, se debe considerar también el material y el software empleado en el desarrollo. En los costes materiales se han incluido la formación y sus costes derivados.

Dentro de los programas gratuitos se tienen:

- Unity: entorno de desarrollo.
- Blender: diseño de los avatares.
- Audacity: edición de audio

En la Tabla 2, se puede ver el desglose de precios:

Tabla 2: Estimación de Costes

Tipo de Recurso	**Recurso**	**Coste de adquisición**
Rol/especialización	Salario	10€/hora * 490 h = 4900€
Material	Gafas de VR Oculus Quest 2	350 €
	Ordenador	1200€
Formación	Curso en Udemy introductorio a Unity	70€

	Curso introductorio de Blender realizado en la plataforma Udemy	40€
Licencias	Windows	120€
	Total	6680€

2.5 Análisis de riesgos

El análisis de riesgos es el proceso que permite la identificación de las amenazas que pueden perjudicar al desarrollo del proyecto y determinar el impacto o grado de perjuicio que pueden ocasionar. Implica analizar las amenazas y vulnerabilidades que puedan darse y se valoran en términos de probabilidad de ocurrencia y gravedad del impacto sobre el proyecto.

A continuación se establece una escala (de menor a mayor gravedad) para valorar el impacto que causarían los riesgos que se describen a continuación:

- Muy grave: De consecuencias fatales para el desarrollo del proyecto, requerirá replantearse, incluso, si merece la pena continuar con el proyecto.
- Grave: Problemas que podrían imposibilitar terminar el proyecto en el plazo previsto. Requeriría realizar una profunda planificación ampliando el número de horas a dedicar o reduciendo el trabajo a realizar.
- Perjudicial: Pequeños retrasos relativamente sencillos de solventar, aunque pueden resultar peligrosos si no se identifican y subsanan a tiempo.

Los posibles riesgos que se tienen en cuenta en la realización del proyecto son los siguientes:

- Elección equivocada de las herramientas de trabajo:

  Descripción: La elección del motor de físicas a utilizar, puede ser determinante para el curso del proyecto y las dificultades que se puedan presentar. Además, también se podría tener problemas con las gafas de VR, puesto que no todas son válidas para las mismas tareas.

  Probabilidad: 20%

Alcance: Perjudicial

Medidas preventivas: no es necesario adoptar medidas preventivas ya que el desarrollador conoce las herramientas y tiene experiencia con las mismas.

- Desconocimiento del software elegido:

  Descripción: Este es un problema que persigue a los desarrolladores de casi cualquier proyecto. Esta no será la excepción, puesto que se trabajarán con herramientas que tienen una curva de aprendizaje bastante marcada, significando un gran esfuerzo por parte de los desarrolladores.

  Probabilidad: 90%

  Alcance: Perjudicial

  Medidas preventivas: no es necesario adoptar medidas preventivas ya que el desarrollador conoce las herramientas y tiene experiencia con las mismas.

- Estancamiento en el desarrollo:

  Descripción: Existe la posibilidad de que llegados a un punto se entre a un callejón sin salida, llevado al proyecto a estancarse en un mismo sitio.

  Probabilidad: 20%

  Alcance: Perjudicial

  Medidas preventivas: El desarrollo del proyecto se divide en secciones, de tal forma que realizando pequeñas tareas se avanza en el desarrollo, evitando así el estancamiento por la imposibilidad de realizar grandes cambios.

- Pérdida de los datos del proyecto por fallo del hardware o software maligno

Descripción: La realización de backup y la documentación debería solucionar este problema, aunque siempre existe la posibilidad de que se pierda la información.

Probabilidad: 10%

Alcance: Muy grave, pues supondría empezar de cero.

Medidas preventivas: Se ha implantado un sistema de control de versiones (git) y una copia de seguridad del entorno de desarrollo, además de herramientas antivirus y antimalware.

- Baja del desarrollador de la aplicación:

  Descripción: Podría suceder que, por enfermedad o accidente, el desarrollador del proyecto quede incapacitado durante una temporada.

  Probabilidad: 5%

  Alcance: Perjudicial, puesto que impide la realización del proyecto.

  Medidas preventivas: No aplicables.

- Análisis o planificación muy optimistas

  Descripción: El tiempo de realización del proyecto no siempre es el óptimo y se puede ser optimista en la planificación, llevando a mucho tiempo de retraso en el desarrollo.

  Probabilidad: 10%

  Alcance: Perjudicial, puesto que retrasaría la fecha de entrega del proyecto.

  Medidas preventivas: Estudiar detalladamente los tiempos estimados y dejar un pequeño margen de error que permita suplir cualquier problema durante el desarrollo.

2.6 Particularidades de este proyecto

En este proyecto se va a desarrollar una herramienta que es la unión de muchas partes y conocimientos. Por un lado se tiene la parte técnica e informática que permite desarrollar el entorno

3D con todas las funcionalidades necesarias. Por otro lado están los conocimientos del BSR y las mecánicas específicas de este deporte, además de las intenciones que se tengan con la simulación de estas acciones concretas.

Por esto, es la unión de dos mundos bastante separados hasta ahora. Esto supone un trabajo extra, no solo en la parte informática, sino en la documentación del propio deporte y en la dificultad de conseguir que ciertas ideas o conceptos se puedan trasladar a VR.

3. Baloncesto y Realidad Virtual

En este capítulo se entrelazan los conceptos de Realidad virtual y deporte, en concreto con el BSR, de esta forma se irá avanzando en los detalles técnicos que hacen a la realidad virtual una tecnología con mucho potencial en este ámbito.

3.1 Realidad Virtual (VR). Conceptos básicos.

El término VR o Realidad Virtual define bastante bien el objetivo de esta tecnología, simular situaciones o entornos de la vida real en un mundo virtual que se muestra a través de una pantalla. La definición ya deja intuir qué ventajas puede tener esta tecnología. La primera de ellas se hereda de la informática en general, la capacidad de replicar un programa o escena en concreto las veces que se desee, y con características concretas que en la vida real sería difícil de lograr. La segunda también se deja ver, la capacidad de simular entornos reales con bastante grado de realismo, aunque más que realismo, esta tecnología brinda una experiencia más inmersiva. En el Anexo III se detalla el funcionamiento interno de esta tecnología.

Estas razones hacen que sea algo a tener en cuenta cuando se desarrollan herramientas que necesiten de estas características.

3.2 Características del Baloncesto en Silla de Ruedas (BSR).

El Baloncesto en Silla de Ruedas o comúnmente llamado BSR es un deporte adaptado. Esto quiere decir que las personas que lo juegan tienen una discapacidad física, por ejemplo amputación de una extremidad, una lesión medular, entre otras lesiones. Esto hace que sea muy diferente a un deporte en el que solo participan personas que cuentan con una discapacidad física.

La principal diferencia está en el medio que se utiliza para jugar, la silla de ruedas. Estas sillas distan mucho de la idea que se tiene de una silla de hospital muy pesada y poco estética. Las sillas de competición son muy ligeras, apenas sobrepasan los 9 kg y están específicamente diseñadas para cada jugador.

En el Anexo IV se habla con más detalle sobre este deporte.

3.3 Beneficios de la VR en la práctica de un deporte.

La necesidad de realizar un deporte competitivo va más allá de la ventaja de mantener una forma física robusta. En el fondo existe la necesidad de dejar evidencia de que individuo o colectivo de individuos están por encima de otros. Esto es algo que está dentro de cada uno de nosotros, y es el instinto competitivo que muchas veces nos da vida, siempre y cuando se sepa gestionar con criterio.

En la práctica de un deporte competitivo, las herramientas que permitan mejorar el rendimiento con relativamente poco esfuerzo son muy perseguidas. En este caso, se suele pensar en mejoras físicas como las propias sillas de ruedas, y aunque también influyen mucho, se suele descuidar la herramienta más importante que tiene el ser humano, su mente. Por esta razón hay que potenciar las herramientas que desarrollen la capacidad mental de un jugador, ya sean en cuanto a visión de juego, capacidad de reacción, concentración, etc...

La realidad virtual encaja perfectamente en esta necesidad, puesto que prácticamente todo lo que potencia o trabaja es la mente.

3.4 Aplicación Teórica de VR a BSR.

Para ilustrar la idea de aplicar el VR al Baloncesto en Silla de Ruedas es mejor utilizar un ejemplo práctico.

Imaginemos que un jugador está lesionado y le es imposible realizar la actividad deportiva con normalidad. Esto puede suponer un problema muy grave, puesto que lo más importante para un deportista es su cuerpo... ¿o no? Como se ha mencionado anteriormente la mente juega un papel crucial, y el poder entrenar a este jugador aunque su cuerpo no lo permita temporalmente también supone un paso muy importante. Aquí es donde entra en juego la realidad virtual, haciendo posible que se pueda seguir entrenando algunas partes del deporte.

Este es un ejemplo muy concreto, puesto que la idea principal es complementar la práctica deportiva con este tipo de herramientas.

4. Diseño de la Aplicación.

Para lograr el objetivo de la herramienta, es necesario ante todo saber que se va simular, puesto que sobre estas ideas se desarrollará el proyecto. Después de algunas horas de observación y teniendo en cuenta que el desarrollador de este proyecto es jugador de BSR, se ha llegado a la siguiente conclusión.

La aplicación se basará en tres escenas, cada una de ellas engloba a una idea:

1. Realización de **tiros** a canasta: estar posicionado a cierta distancia de la canasta con el balón en las manos, tendiendo como objetivo encestar.
2. Realización de **pases**: el usuario deberá realizar un pase a otro NPC (Non Playable Character) o a un sitio en concreto de la escena.
3. **Interceptar** pases: el usuario tiene que interceptar un pase entre dos posiciones, ya sea estando en medio o en otro sitio donde tenga la posibilidad de realizar esta acción.

Cada escena como bien se ha mencionado anteriormente engloba a una idea, tiros, pases y robo de balón. A esto hay que incluir niveles de dificultad para que la experiencia no sea monótona. Cada escena tendrá 3 niveles de dificultad.

El concepto de dificultad en este tipo de aplicaciones puede ser muy variado, que por lo general viene dado por el retoque de ciertos parámetros en la propia escena, o simplemente utilizando recursos que den la sensación de presión al usuario.

A continuación se detallan las características de cada nivel de dificultad:

Escena 1 (Tiro)

Nivel 1: Usuario en una posición de tiro donde el rival está a cierta distancia. Se tiene el objetivo de que el usuario tenga la sensación de cierto nivel de dificultad para poder encestar el balón en la canasta.

Nivel 2: Usuario en una posición de tiro donde hay dos rivales que están a una distancia bastante corta. Objetivo de que el usuario tenga la sensación de un mayor nivel de dificultad para poder encestar el balón en la canasta en comparación con el nivel anterior.

Nivel 3: Usuario en una posición de tiro donde tres rivales están a una distancia bastante corta. Objetivo de que el usuario tenga la sensación de un mayor nivel de dificultad para poder encestar el balón en la canasta en comparación con el nivel anterior.

Escena 2 (Pases)

Nivel 1: Se le colocará al usuario un rival a cierta distancia para que, al dar el pase al compañero, se tenga la sensación de que el rival tiene cierta posibilidad de interceptar el balón.

Nivel 2: El usuario tendrá a dos rivales a una distancia más corta que en el anterior nivel para que, al dar el pase al compañero, se tenga la sensación de que los rivales tienen cierta posibilidad de interceptar el balón.

Nivel 3: Usuario ante tres rivales a una distancia más corta que en el anterior nivel para que, al dar el pase a alguno de los dos compañeros que están en el campo, se tenga la sensación de que los rivales tienen cierta posibilidad de interceptar el balón.

Escena 3 (Interceptar pases)

Nivel 1: El usuario tiene que interceptar el pase entre dos rivales. Respecto a su posición en relación a los rivales, el usuario se encontrará aproximadamente en la mitad de ambos.

Nivel 2: El usuario tiene que interceptar el pase del contrario. En vez de haber dos rivales, habrá tres. El usuario sigue estando en medio de la trayectoria del balón. Objetivo adivinar a quién de los dos rivales irá y poder interpretarlo.

Nivel 3: El usuario tiene que interceptar el pase del contrario. Sigue habiendo tres rivales, pero la dificultad en este nivel reside en que, si el balón no es interceptado, se le restará la puntuación obtenida. El usuario sigue estando en medio de la trayectoria del balón hacia los rivales. Objetivo adivinar a quién de los dos rivales irá y poder interpretarlo.

Nivel Tutorial

Aparte de estos tres niveles por cada escena, se incluirá uno más, el nivel tutorial para que el usuario sepa que tiene que hacer en cada nivel.

Objetivo: Demostración desde el propio juego (a modo de tutorial) de cómo tiene que coger el balón, qué botones tiene que apretar para poder hacerlo y luego qué hay que hacer para poder llevar a cabo la acción de manera correcta (tirar, pasar o robar el balón).

Escena 1: El usuario con la canasta de frente, sin nadie alrededor, tirar. Objetivo que consiga encestar el balón en el menor tiempo posible.

Escena 2: El usuario va a dar un pase a su compañero/a de equipo, sin nadie alrededor. Objetivo poder dar un pase de manera correcta sin tener ningún tipo de presión en cuanto a rivales se refiere.

Escena 3: El usuario en una situación de robo de balón en la que va a tener que interceptar el pase entre dos rivales. Objetivo poder robar el balón de manera efectiva.

Tiempos y puntuación de los niveles.

A continuación se muestran las tablas de tiempos y puntuaciones.

Tabla 3: Tiempos de juego

	Escena 1	**Descanso**	*Escena 2*	**Descanso**	*Escena 3*
Nivel 1	40”	**10”**	40”	**10”**	40”
Descanso 15”					
Nivel 2	40”	**10”**	40”	**10”**	40”
Descanso 15”					
Nivel 3	40”	**10”**	40”	**10”**	40”

Tabla 4: Puntuación de los niveles

	Puntuación		
	Nivel 1	*Nivel 2*	*Nivel 3*
Encestar en la canasta	1 punto	2 puntos	3 puntos
Pase acertado al compañero/a	1 punto	2 puntos	3 puntos
Robo de balón	1 punto	2 puntos	3 puntos

5. Herramientas

En esta sección se definirán las herramientas más idóneas para solucionar el problema planteado y realizar un proyecto útil y funcional.

5.1 Entorno virtual

Este apartado es muy crítico en proyectos de este tipo, puesto que el hardware que se elija va a condicionar el desarrollo de la aplicación. Primero que todo hay que definir qué se espera del hardware, en este caso las gafas de VR.

Movilidad: este es sin lugar a dudas el punto más importante a tener en cuenta, puesto que es necesario un sistema que permite gran movilidad, es decir, que no necesite de componentes externos que influyan en la experiencia del usuario. Como en muchas situaciones si se tiene una característica generalmente se sacrifica alguna otra, y suele ser el rendimiento. En este caso, la aplicación desarrollada tiene que ser ante todo funcional, el apartado gráfico sólo tiene que crear una base convincente. Por esta razón el hardware elegido debe ser cómodo de llevar y a ser posible no necesitar cables y componentes externos.

Rendimiento: al no ser una aplicación que se base en la parte visual para lograr su objetivo, en este apartado simplemente se espera fluidez y rapidez en la interfaz, puesto que no se intenta lograr una experiencia realista, sino funcional.

Plataforma y software: en el desarrollo cualquier herramienta que se pueda reutilizar agiliza mucho el proceso, por esto cualquier paquete o driver que se incluya en Headset VR que aumente su compatibilidad con Unity puede agilizar mucho el proceso de desarrollo.

Teniendo en cuenta lo anteriormente dicho, y viendo las alternativas existentes en el mercado (ver Anexo III), la mejor opción para este apartado es elegir las Oculus Quest 2. La movilidad que ofrecen es buena puesto que no necesitan cables y cuenta con una batería interna, dando una autonomía de unas 2h. En el apartado de rendimiento cuentan con un Qualcomm® Snapdragon™ XR2 como CPU y 6gb de RAM, no se debería tener problemas en este apartado. Por último corren una versión de Android como sistema operativo, por lo que heredan muchas características de este

sistema, además incluye un paquete de integración “Oculus Integration” que facilita el desarrollo de aplicaciones con Unity.

5.2 Motor gráfico

Para la elección del motor gráfico en este proyecto hay un factor en concreto que tuvo mucho peso, y es el hecho de que al empezar no se contaba con ninguna experiencia en el desarrollo de aplicaciones 3D. Por esta razón es importante valorar qué opciones tienen más soporte por la comunidad, de esta forma se pueden encontrar cursos y tutoriales que faciliten el desarrollo. En este caso, se dio la posibilidad de optar por una formación brindada por la universidad de Deusto en Bilbao, especializada en Unity. Como es lógico esta decisión marcó definitivamente que motor gráfico se utilizará para desarrollar la aplicación.

Independientemente de esta situación, en el mercado existen principalmente dos opciones a elegir en cuanto a motores gráficos, Unity y Unreal. Particularmente en este proyecto el factor gráfico no es lo más importante, en cambio lo que sí es un requisito indispensable es la modularidad y la facilidad de configurar los aspectos más pequeños del entorno. Por esta razón, la idea de utilizar Unity es acertada.

En el Anexo I, dedicado a Unity, se detallan las características de esta herramienta.

5.3 Diseño 3D

En proyectos donde se crean entornos virtuales es necesario utilizar software específicos de modelado 3D para crear objetos y personajes. En el mercado existen principalmente dos opciones en cuanto a software de este tipo, Blender y Maya. En este caso, ya se contaba con experiencia modelando con Blender, por esta razón se ha elegido este programa. Además, el factor económico también influye mucho, y teniendo en cuenta que la licencia de Maya es muy costosa, la decisión de optar por Blender es más que acertada.

En el Anexo II, dedicado a Blender, se detallan las características de esta herramienta.

5.4 Otras herramientas

En el desarrollo se pueden necesitar otras herramientas a medida que se va desarrollando la aplicación. Por ejemplo, se pueden necesitar algunos conocimientos de edición de audio para generar sonidos con el fin de aumentar el realismo. Además se utiliza git como sistema de gestión de copias.

6. Desarrollo De La Aplicación

En este capítulo se explicarán los detalles técnicos del desarrollo así como las decisiones tomadas durante este proceso. Recordando un poco la idea, se pretende desarrollar una aplicación de VR para simular aspectos y mecánicas del BSR, utilizando Unity como motor gráfico y las Oculus Quest 2 como gafas de realidad virtual. Para entender correctamente este capítulo es aconsejable revisar el Anexo I para tener cierto nivel de entendimiento sobre Unity.

Configuración de las gafas de VR con Unity

El primer paso es entrelazar las Gafas Oculus con Unity. Las Oculus Quest 2 utilizan Android como sistema operativo, por lo que la conexión se tendrá que realizar mediante ADB. Recordemos que ADB (Android Debug Bridge) es una herramienta de Android que permite conectar un dispositivo móvil con un ordenador mediante un cable USB, pudiendo realizar tareas como instalar APKs directamente desde el ordenador.

Luego es necesario habilitar el modo desarrollador en las Oculus. Para hacer esto hay que activar nuestra cuenta de facebook asociada a las Oculus como una cuenta de desarrollador. Con esto ya podemos activar el modo desarrollador. En la web oficial de Oculus se detallan los pasos.

Una vez la conexión está establecida, se tiene que configurar Unity para que reconozca las gafas. En **File => Build Settings**, se debe seleccionar las Oculus Quest 2. Además se debe cambiar la plataforma de desarrollo a Android.

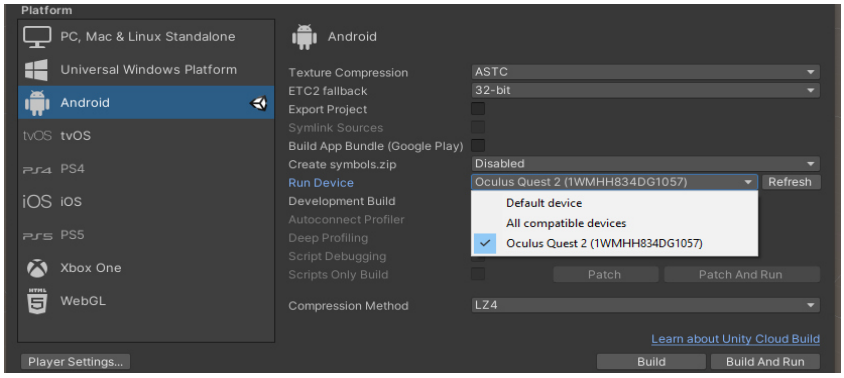


Figura 2. Build Settings

Paquete Oculus Integration

El paquete Oculus Integration es una herramienta necesaria cuando se desarrollan aplicaciones para las gafas de VR de Oculus. Nos ofrece funciones, componentes, scripts y plugins para facilitar y mejorar el proceso de desarrollo. Es importante recalcar que ya Oculus proporciona una documentación detallada de los componentes de este paquete y cómo se pueden utilizar en favor del desarrollo de una aplicación con las Quest 2. En este capítulo solo se brinda un resumen y se centra en los componentes usados en este desarrollo en concreto.

En Unity Asset Store podemos descargar este paquete e importarlo al proyecto.

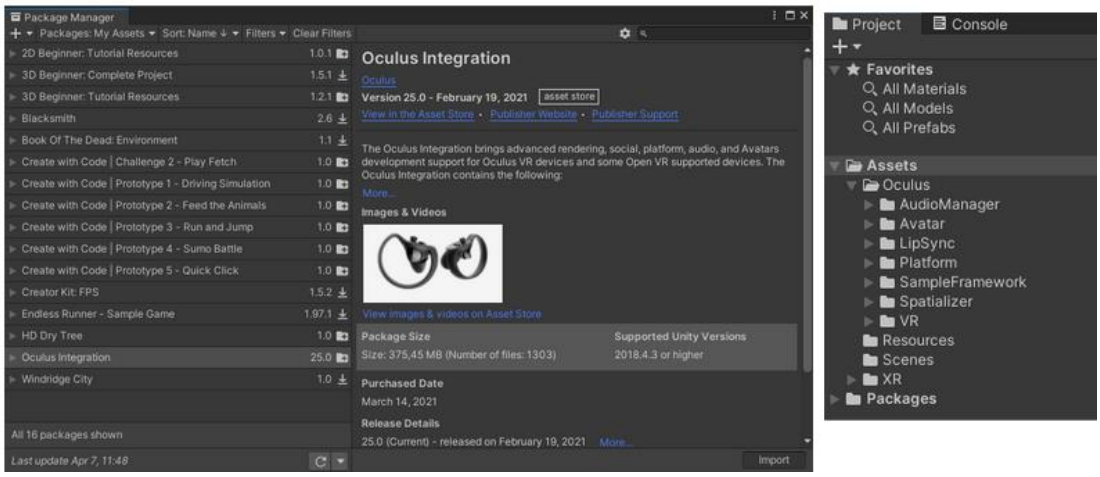


Figura 3. Paquete Oculus Integration

En cuanto a la configuración del proyecto, es interesante mirar la web oficial de Oculus, donde hacen una serie de recomendaciones al respecto.

Una vez realizada la configuración y añadida la escena actual a la compilación, se podrá ver en las oculus la escena creada. Ahora se puede pasar a utilizar las funciones que ofrece el paquete Oculus Integration, y construir una demo inicial que sirva de base.

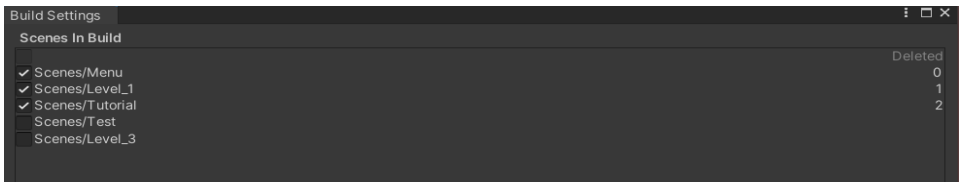
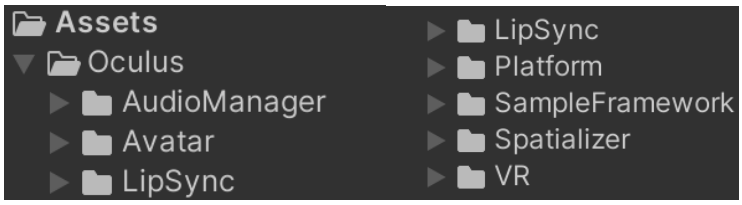


Figura 4. Build Settings

Una vez la configuración esté lista, es necesario saber qué funciones nos ofrece el paquete Oculus Integration para facilitar el desarrollo. Dentro de la carpeta Oculus aparece todo el contenido importado. A continuación se muestra un resumen de los recursos:

Tabla 5: Contenido de la carpeta Oculus



Carpeta	**Descripción**
AudioManager	Colección de clips de audio que puede utilizar durante el desarrollo.
SampleFramework	Contiene ejemplo de algunas funcionalidades que tienen las Oculus Quest 2
VR => Editor	Scripts que añaden funcionalidad al editor de Unity y mejoran otros scripts de componentes C#.
VR => Prefabs	Colección de prefabricados que se pueden incluir en una escena, como OVRCameraRig, OVRHandPrefab, y OVRPlayerController.
VR => Scenes	Ejemplos de escenas que ilustran conceptos comunes.
VR => Scripts	Archivos C# que VR framework y los componentes de Unity.

Lo más interesante a destacar son los Prefabs que se incluyen. En Unity, los prefabs son gameobject reutilizables, como formas 3D, luces, audio o cámaras. Para utilizar estos prefabs, basta con arrastrarlos a la escena. La carpeta **Assets/Oculus/VR/Prefabs/** incluye:

- OVRCameraRig: Una cámara de VR que optimiza el renderizado para una pantalla estereoscópica como la que usan las gafas de VR. Facilita mucho el proceso de desarrollo, puesto que es algo que no es necesario programar. El objeto OVRCameraRig contiene un script OVRManager que es el encargado de configurar este prefab, en la documentación de Oculus están detalladas las opciones incluidas. A continuación se resumen las diferentes partes de este gameobject:

Figura 5. Estructura del prefab OVRCameraRig

- TrackingSapce: en este objeto se incluyen todos los componentes que representan el movimiento de las gafas, como son los mandos, las cámaras correspondientes a cada ojo, etc.
- LeftEyeAnchor, CenterEyeAnchor y RightEyeAnchor: representan las cámaras utilizadas para renderizar la imagen de la escena. Si se desea cambiar algún parámetro específico como el color de fondo cuando no hay imagen para renderizar, se debe hacer en estos tres objetos.
- LeftHandAnchor y RightHandAnchor: todo objeto que sea hijo de estos dos padres se moverá y tendrá la misma posición que los mandos de las oculus. De esta forma el prefab de los mandos deben ser incluido en estos objetos. Este movimiento y rotación está realizado automáticamente por el prefab OVRCameraRig y no es necesario programarlo.

  Como se puede apreciar en la imagen anterior, se asignan los prefab CustomHandLeft y CustomHandRight a sus correspondientes anclajes, así se moverán y tendrán la misma posición y la rotación que los mandos de las Oculus cuando se reproduzca la aplicación. Cabe destacar que también se incluyen los objetos LeftControllerAnchor y RightControllerAnchor para mayor flexibilidad, pero como ya se ha mencionado anteriormente heredan sus padres.

- OVRPlayerController: Este prefab es igual que el anterior, la única diferencia es que la cámara (OVRCameraRig) tiene un padre (OVRPlayerController) que incluye scripts que permiten el movimiento por la escena con los controles de los mandos.

- OVRHandPrefab: Implementa Hand Tracking, para utilizar las manos en vez de los mandos.
- CustomHandLeft y CustomHandRight: Simulan las manos en el entorno virtual, cuando se agarra el mando y se presiona un botón, las manos virtuales se deforman e imitan el gesto en dependencia del botón pulsado. Se incluye el script OVRGraber por defecto, el cual contiene la funcionalidad necesaria para agarrar objetos, debe ser configurado como se muestra en la siguiente imagen, además también contiene Colliders que permiten la detección de los objetos a agarrar.

Figura 6. CustomHandLeft y CustomHandRight

El paquete también cuenta con una serie de clases de tipo .cs que permiten obtener información del sistema y agregar funcionalidad.

Tabla 6: Scripts fundamentales en el desarrollo

Script	Descripción
OVR Boundary.cs	Proporciona acceso al sistema de límites de Oculus
OVRHaptics.cs	Proporciona retroalimentación háptica en el controlador de seguimiento
OVRInput.cs	Proporciona un sistema de entrada unificado para los mandos y gamepads de Oculus
OVROverlay.cs	Este script permite renderizar ciertas formas de contenido, como texto, vídeo o texturas, con una calidad superior a la que puede alcanzar el renderizado tradicional.

OVRGrabber.cs	Permite agarrar y lanzar objetos utilizando los mandos
OVRGrabbable.cs	Este script se asigna a un objeto para que este se pueda agarrar y lanzar con los mandos.
OVRModeParms.cs	Registra cuando la aplicación entra en el modo de ahorro de energía y te permite llegar a un nivel de CPU/GPU de bajo consumo con sólo pulsar un botón.
OVRResetOrientation.cs	Permite restablecer VR input tracking con una pulsación del botón del mando.

Una vez más, los script más importantes que se utilizarán son OVRInput.cs, OVRGrabber.cs, y OVRGrabbable.cs. Estos componentes son fundamentales para definir una funcionalidad inicial que permite agarrar objetos y realizar acciones con los botones de los mandos.

Input System

Algo importante a tener en cuenta es el Input System de las gafas, en la imagen a continuación mostrada se puede apreciar el mapeo de los controles.

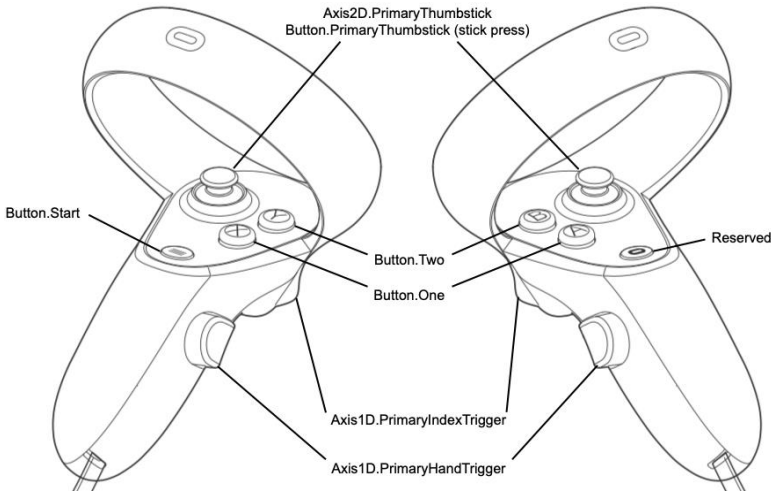


Figura 7. Input System de la Oculus quest 2

Para utilizar los controles a continuación se muestran algunos ejemplos de cómo a través de la clase OVRInput se puede realizar la comunicación.

```
// returns true if the primary button (typically "A") is currently pressed.
OVRInput.Get(OVRInput.Button.One);

// returns true if the primary button (typically "A") was pressed this frame.
OVRInput.GetDown(OVRInput.Button.One);

// returns a Vector2 of the current state of the thumbstick owned by the left
controller.
// (X/Y range of -1.0f to 1.0f)
OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick, OVRInput.Controller.LTouch);

// returns a Vector2 of the current state of the thumbstick owned by the
right controller.
// (X/Y range of -1.0f to 1.0f)
OVRInput.Get(OVRInput.Axis2D.PrimaryThumbstick, OVRInput.Controller.RTouch);
```

Figura 8. Código de ejemplo para acceder al Input System

Una vez se sabe con qué herramientas se cuenta, se puede empezar a construir las bases. Lo primero sería diseñar un objeto Player que tenga los controles necesarios para permitir al usuario interaccionar con el entorno mediante el. Además se deben crear los modelos 3D necesarios para que la experiencia sea satisfactoria y con cierto grado de realismo. A continuación se detalla la creación de los componentes principales de la aplicación.

Componentes

Para que la aplicación tome forma, es necesario crear una serie de componentes que permitan que la experiencia sea realista y divertida.

Modelos 3D

Para la creación de los modelos 3D se utiliza Blender, comencemos con la creación de las sillas de ruedas.

Sillas de ruedas

Como se trata de una aplicación dedicada a imitar el BSR, es imprescindible contar con modelos de sillas de ruedas para dar más realismo. En este caso se han creado 4 diferentes modelos, representado las sillas que utilizarían 4 prensas con un grado de movilidad distinto.

En primer lugar tenemos la silla que utilizaría una persona con una puntuación correspondiente a un 4 o un 4.5. Como ya se cuenta con un plano de una silla real de un jugador con puntuación de 4.5 se puede utilizar para crear el modelo.

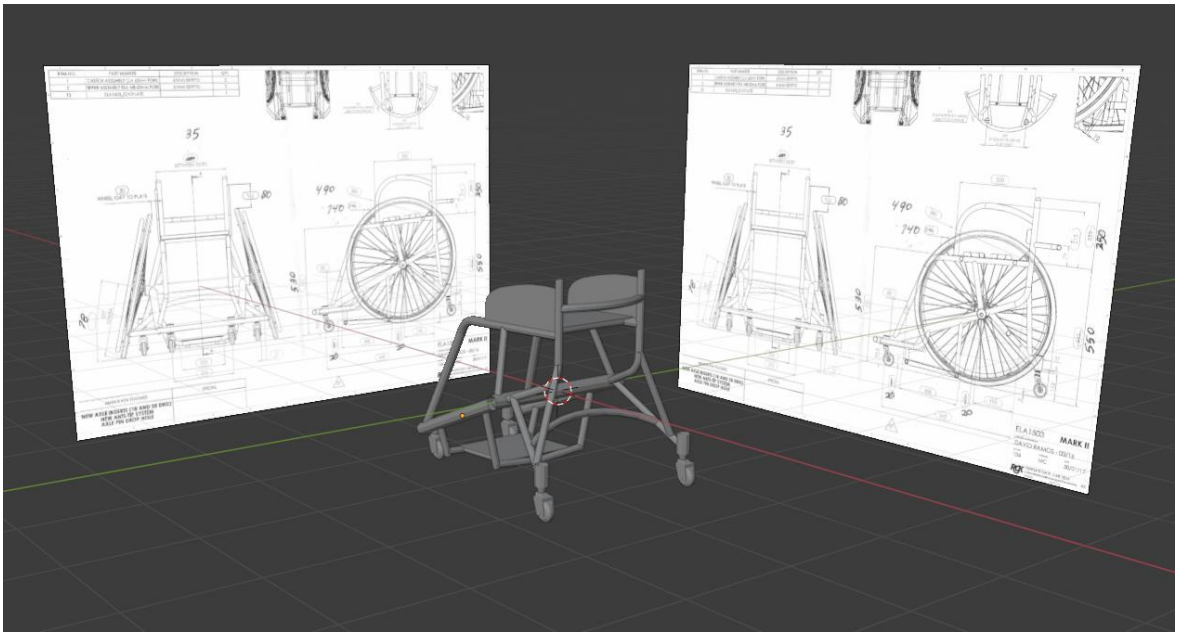


Figura 9. Proceso de modelado de una asilla de BSR

El objetivo es lograr una referencia que permita saber dónde ubicar cada componente, estando la silla formada mayoritariamente por tubos con diferentes formas.

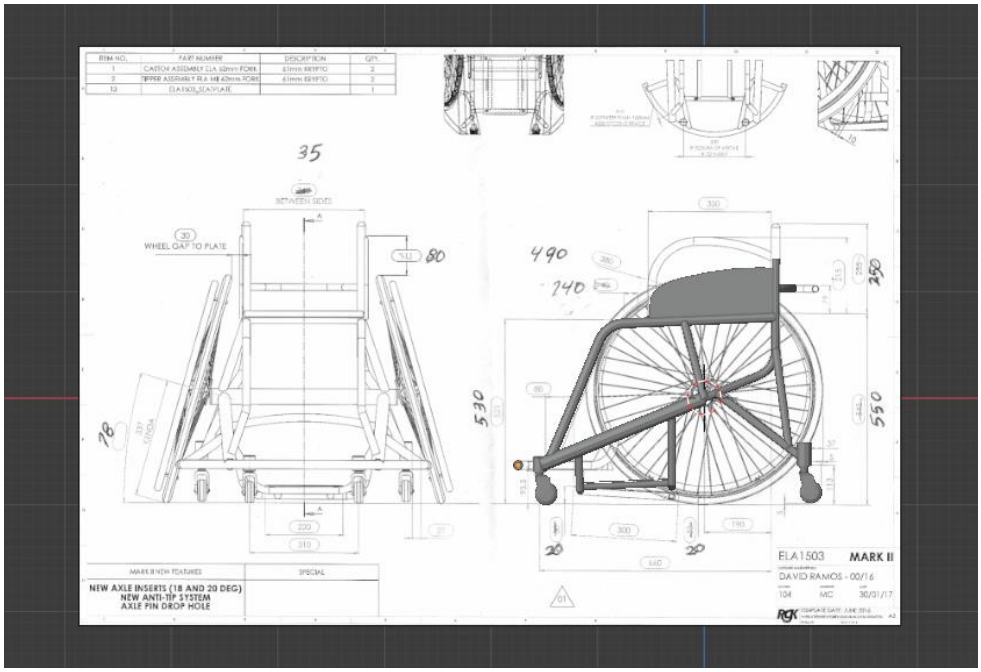


Figura 10. Vista lateral de la silla sin terminar.

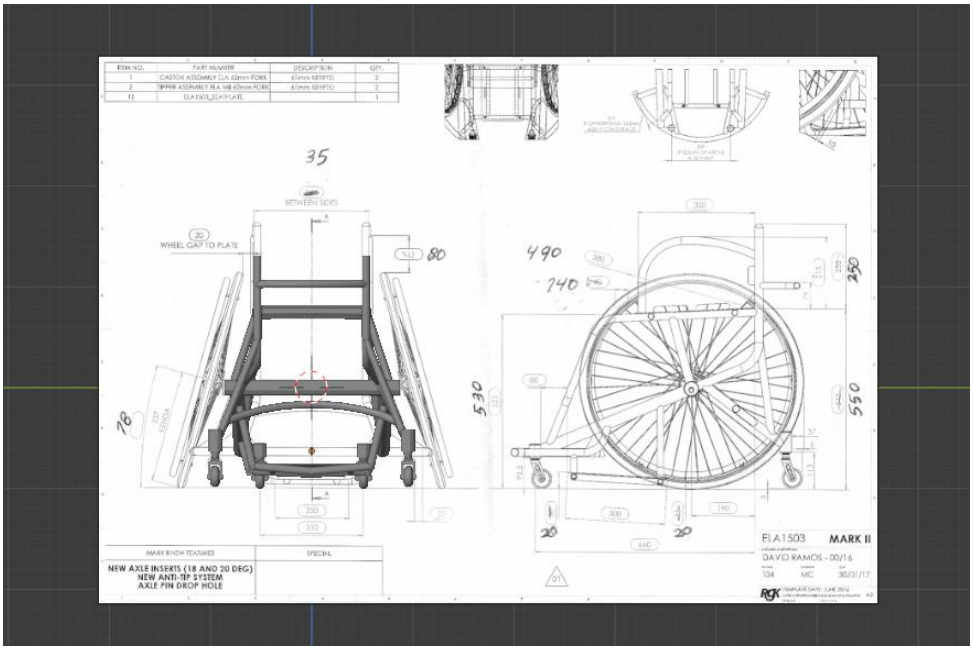


Figura 11. Vista frontal de la silla sin terminar.

Como se puede apreciar en las figuras anteriores, se ubica el plano de tal forma que la vista ortogonal lateral y frontal coinciden con el modelo que se está creando, de esta forma se sabe qué dimensión dar a las figuras. Después de esto, el proceso de creación es sencillo, es cuestión de crear tubos con diferentes formas y diámetros, y colocarlos según los planos utilizados como referencia.

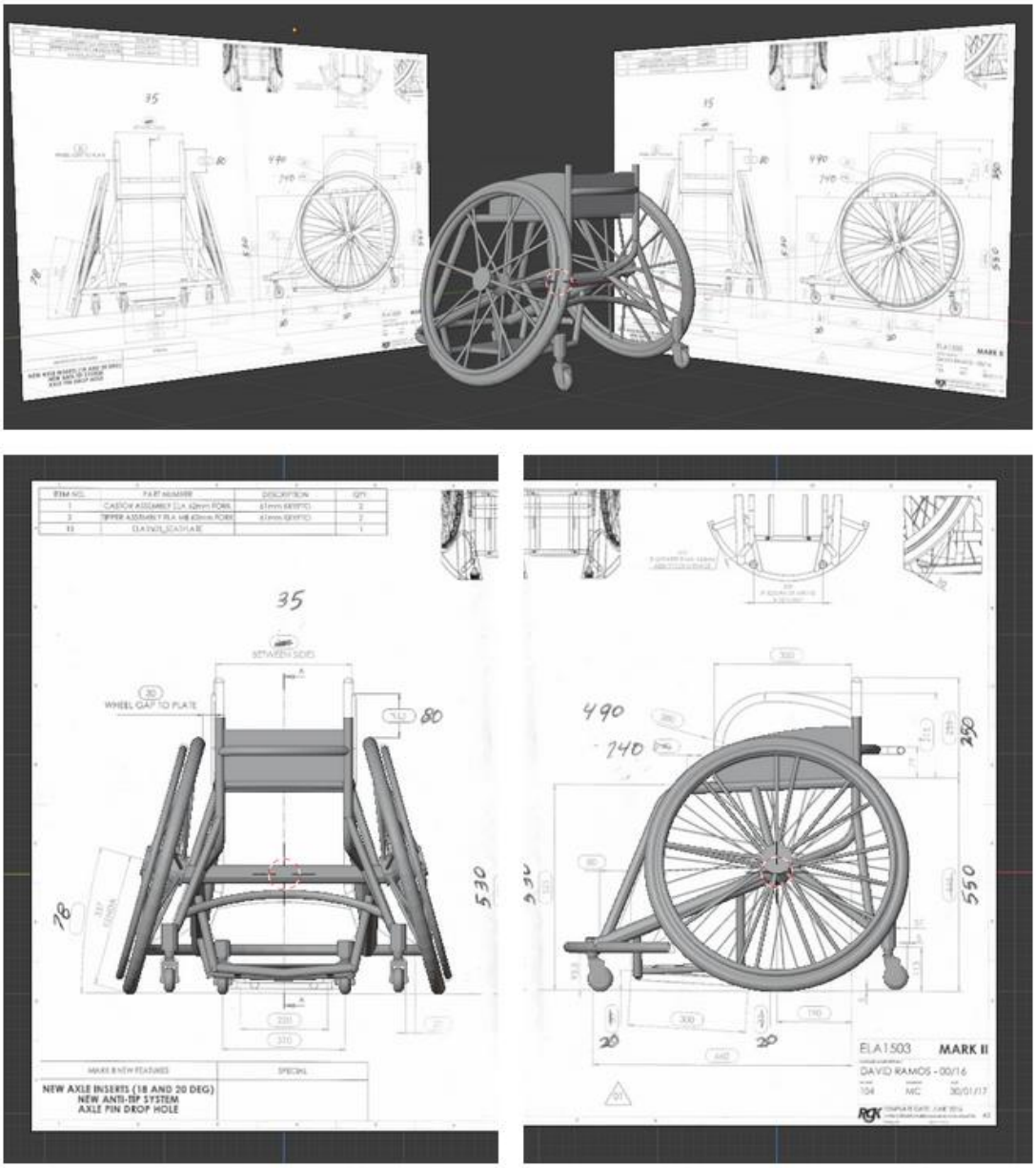


Figura 12. Vista frontal y lateral de la silla terminada, con las ruedas incluidas.

Para las tres sillas restantes, se utilizará como referencia las sillas correspondientes a jugadores con puntuaciones de 3, 2 y 1 respectivamente. Para ahorrar trabajo, y teniendo en cuenta que las sillas a menor puntuación, más pequeñas o más pegadas al suelo suelen estar, se utilizará como base la silla

ya creada para el jugador de puntuación 4, y se reducirá la escala modificando el respaldo y el ancho del asiento.

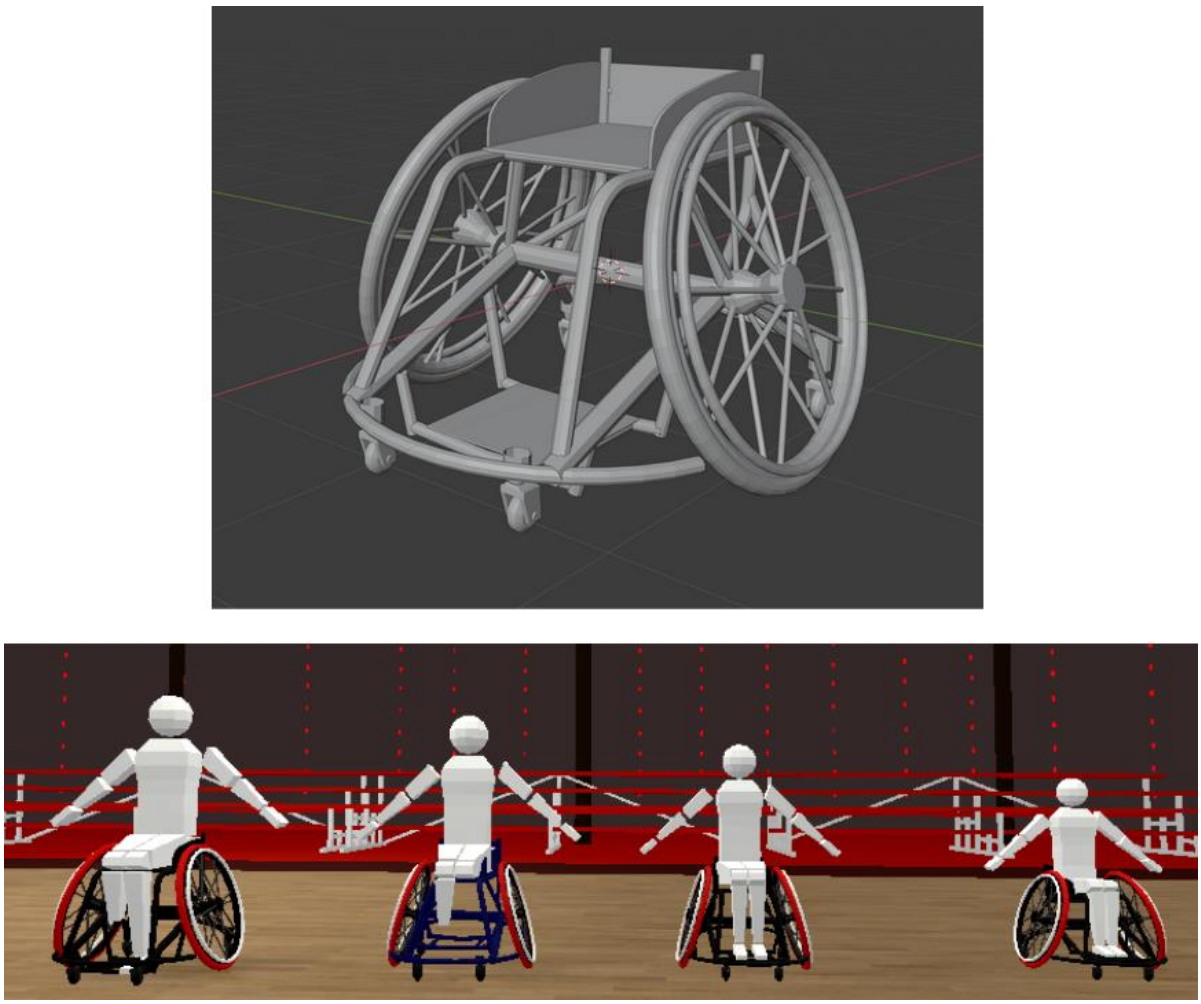


Figura 13. Sillas de ruedas terminadas.

Avatares

Para la creación de los personajes o los NPC (Personaje no jugador o Non Playable Character) se ha utilizado un técnica de modelado llamada Low Poly, que quiere decir utilizar pocos polígonos en la creación del modelo 3D. Esta elección tiene mucho sentido ya que se trata de una plataforma con una capacidad de renderizado reducida, por lo que se tiene que cuidar que modelos 3D se utilizan.

Para la realización de los 4 avatares se utilizó una técnica similar a la usada con las sillas, primero se modela el avatar correspondiente al jugador con una puntuación de 4, y luego se modifica

ese modelo para adaptarlo al resto de puntuaciones. También se han agregado detalles como la falta de alguna extremidad inferior en los avatares, para que coincida con su puntuación.

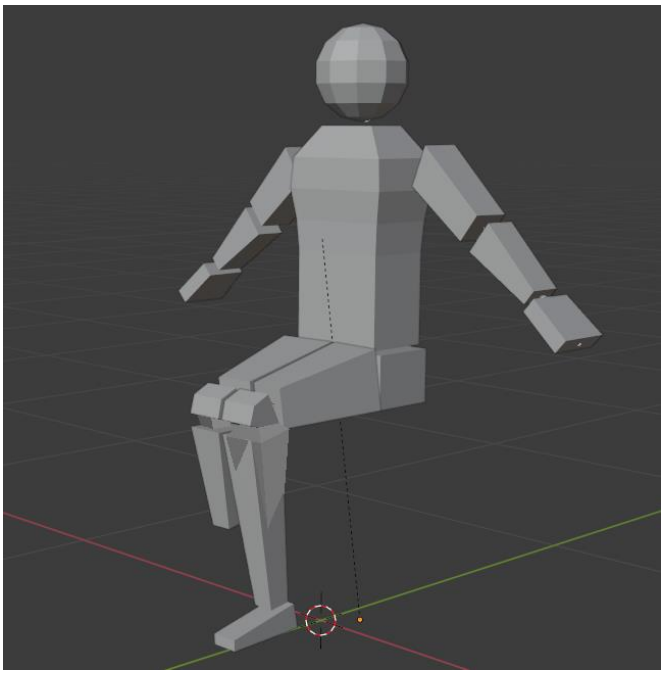


Figura 14. Modelo 3D de un personaje con puntuación de 4 o 4.5.

Para la realización de los modelos se han utilizado figuras básicas como rectángulos y un círculo para la cabeza, de esta forma se simplifica el proceso de creación y se logra el objetivo de un modelo low poly.

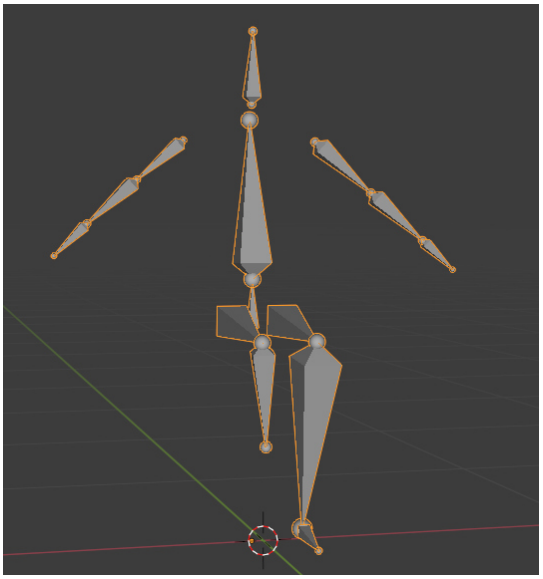


Figura 15. Esqueleto de un personaje.

En la confección del esqueleto se ha creado la jerarquía de huesos acorde a las figuras utilizadas. Posteriormente en Unity se realizará la animación de los avatares.

Campo de baloncesto

Para la confección del campo simplemente se ha utilizado un plano interno con una textura aplicada y un plano exterior con un color aplicado, de esta forma se delimita la superficie del campo.

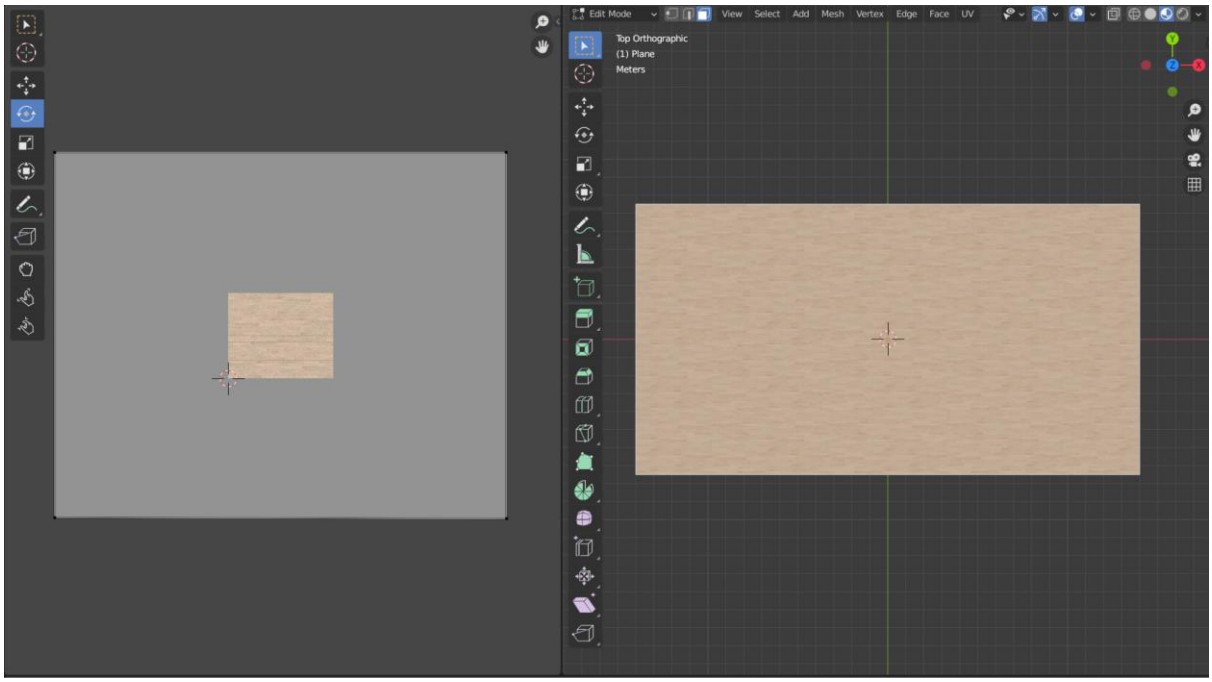


Figura 16. Parte interior del campo de baloncesto con una textura que imita al suelo de un campo real.

Figura 17. Vista superior del campo de baloncesto en Unity.

Otros modelos

Otros modelos como el balón, las gradas, el marcador y demás, son sencillos de crear. Por ejemplo para el balón se utiliza una textura que imita a la de un balón real, esta textura se aplica sobre un círculo creado en Unity.

Figura 18. Balón de Baloncesto.

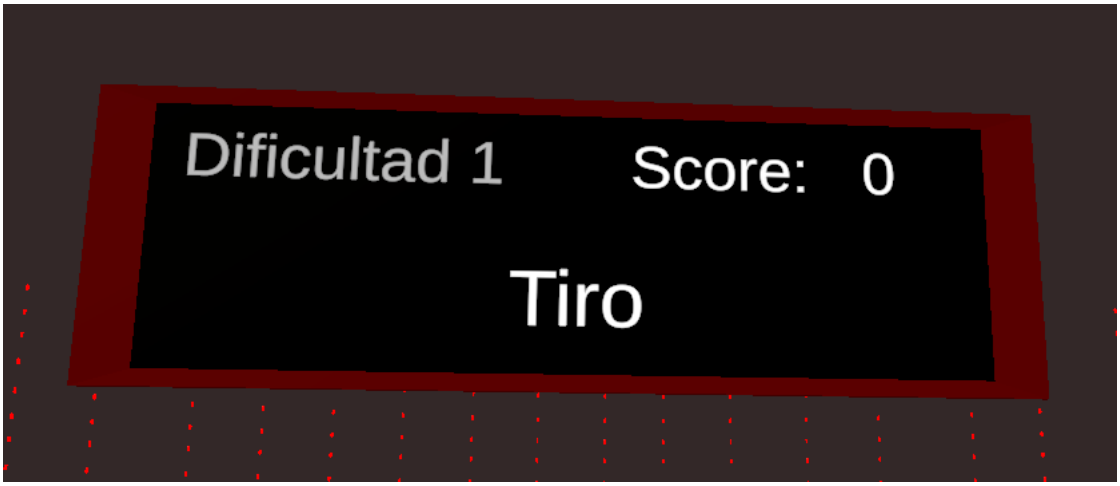


Figura 19. Marcador utilizado para indicar el flujo del juego y la puntuación acumulada por el jugador.

Animación de los Avatares

Unity tiene una forma muy simple de animar modelos 3D una vez estos tienen los componentes anteriormente mostrados, como el esqueleto. En este caso, solo se necesitan animaciones básicas y estáticas, como mover la mano en cierta dirección o simular que se está dando un pase.

El proceso es sencillo, primero se selecciona el objeto en la escena que se quiere animar y luego se añade un componente Animator. Este componente tiene un campo que es el Controller, el encargado de controlar al Animator. Por último solo queda ir a la pestaña de animación y crear una animación.

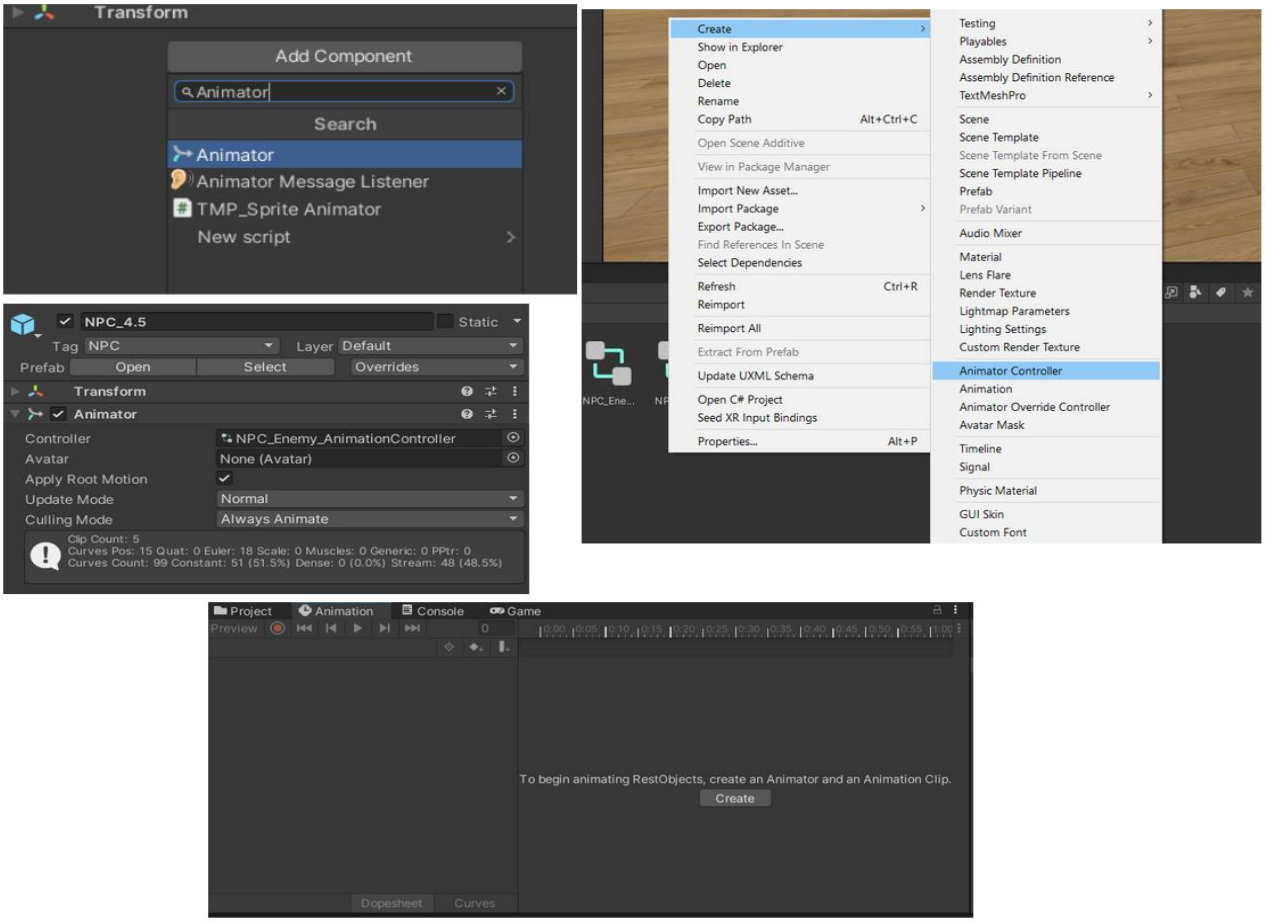


Figura 20. Proceso de creación de animaciones para modelos 3D.

Para animar el personaje se mueven los huesos del avatar para crear una pose como se ve en la siguiente figura.

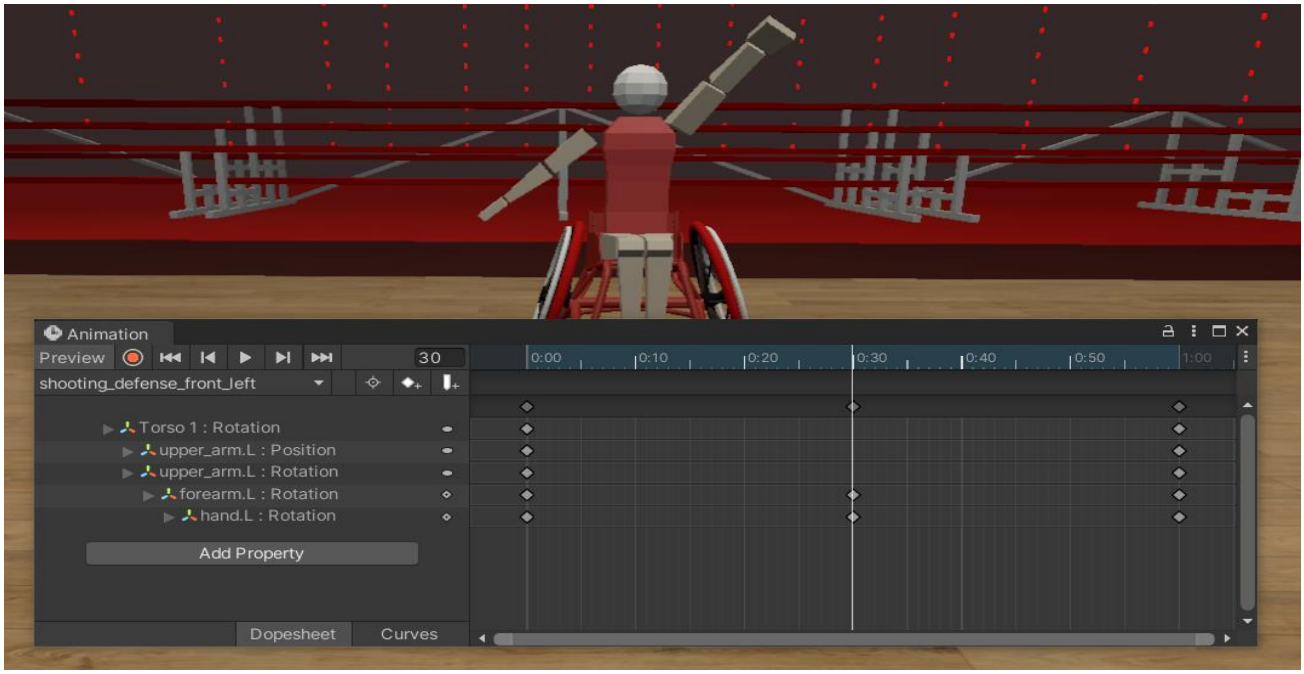


Figura 21. Proceso de creación de animaciones para modelos 3D.

Movimiento de la silla de ruedas

El movimiento de una silla de ruedas en la vida real se logra moviendo las ruedas hacia adelante o hacia atrás, es decir, se necesitan dos componentes que se muevan y uno que pivote y de estabilidad. Para simular esto en Unity se podría hacer que las ruedas roten y muevan el cuerpo de la silla, pero hay que recordar que se trata de una aplicación 3D, por lo que no se tiene que replicar la realidad. Además, realizar el modelo con el movimiento de las ruedas contra el suelo, supondría tener un balance necesario para que las ruedas toquen el suelo, como pasa en la vida real con los antivuelcos trasteros, por lo que se tendrían vibraciones indeseadas en la cámara del jugador.

La solución es simple, se crean dos anclas (RightAnchor y LeftAnchor) a los laterales de la silla y se congela la rotación y posición del objeto para permitir que solo se mueva en dos de los tres ejes, en X y Z. Las anclas se unen al objeto principal mediante el componente Fixed Joint, que como su nombre indica, realiza una conexión fija entre las anclas y su objeto padre.

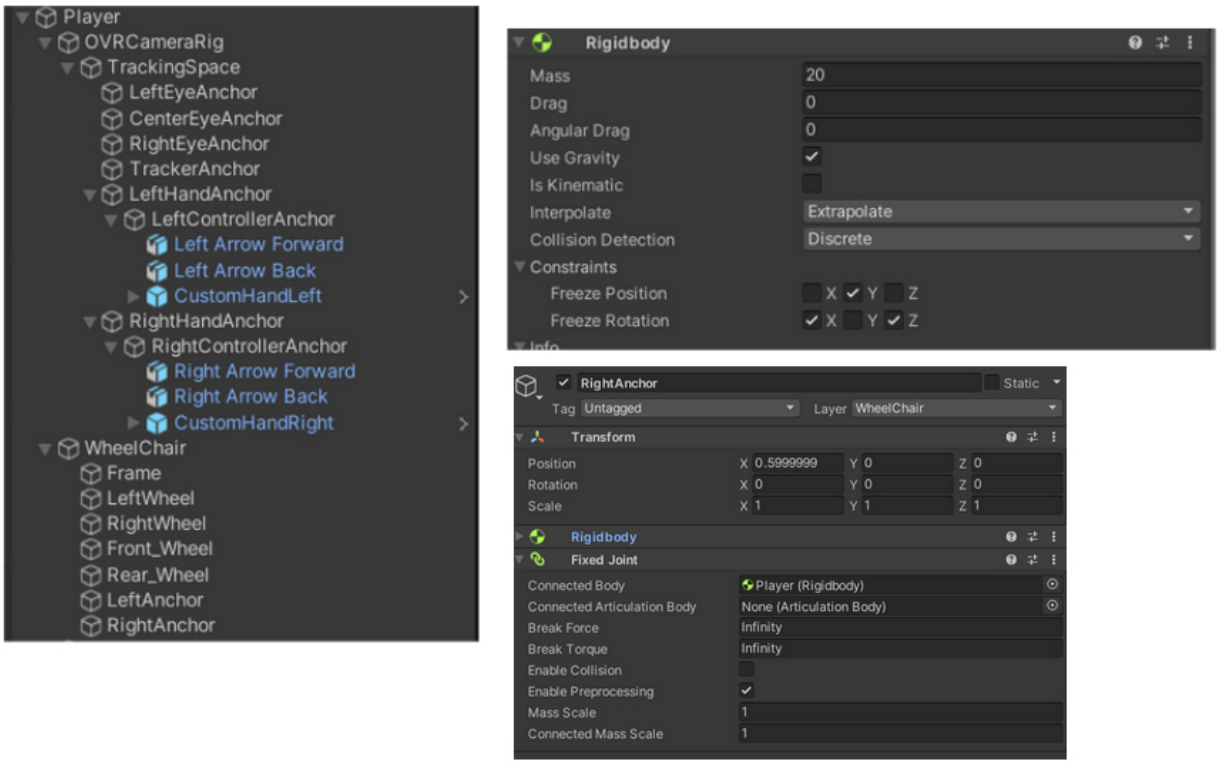


Figura 22. Jerarquía del objeto Player y configuración.

Por último se crea un script que controle cuando se usan los joystick izquierdo o derecho de cada mando y transfiera ese movimiento al ancla correspondiente. Cabe destacar que se han elegido los joystick de los mando para controlar este moviendo al ser estos un "botón" variable, es decir, devuelven un vector de dos dimensiones con un valor entre -1 y 1 en cada componente. De esta forma se le permite al usuario realizar un movimiento controlado y preciso.

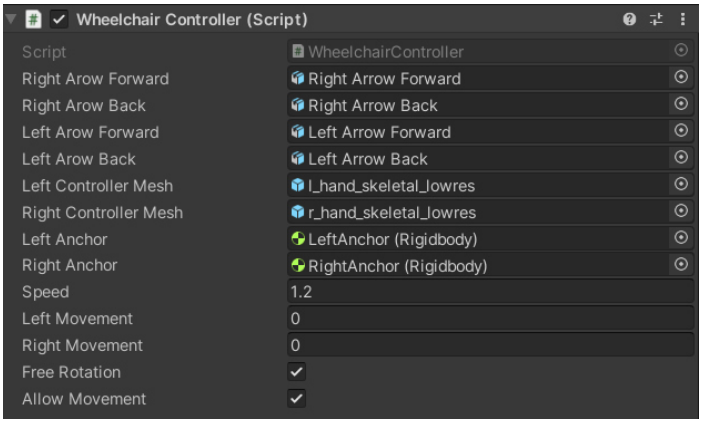


Figura 23. Script que controla el movimiento del personaje, el objeto Player debe tener este script asignado.

El movimiento se logra impulsando hacia adelante o hacia atrás cada ancla mediante sus componentes Rigidbody de forma que simule el efecto que tendría mover la silla mediante las ruedas. El movimiento es posible debido a que ambas anclas están unidas al objeto padre mediante el Fixed Join, por lo que al moverse una de las anclas se moverá también su padre, o el Player.

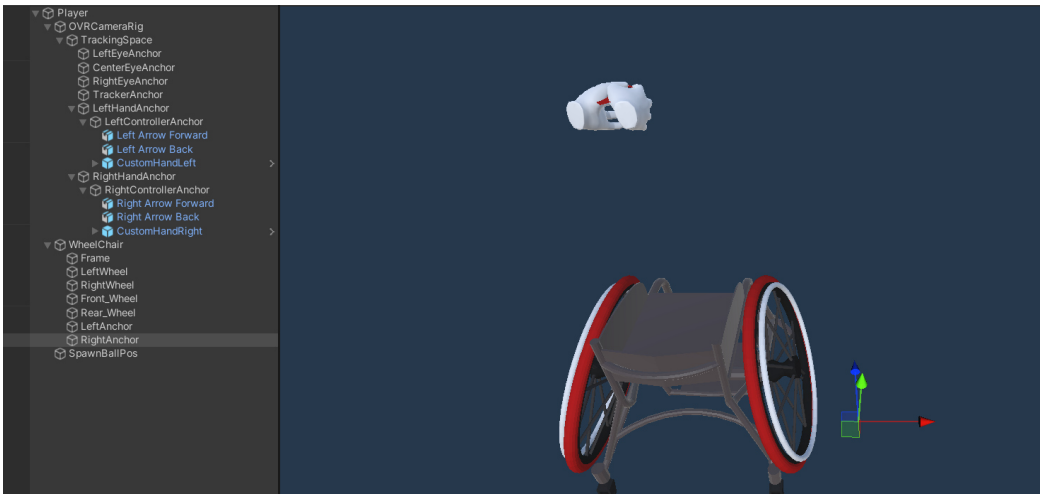


Figura 24. Ubicación del ancla derecha que mueve el objeto Player.

Para indicar al usuario en qué sentido se está moviendo cada rueda, aparece una flecha roja en la posición de las manos indicando el movimiento

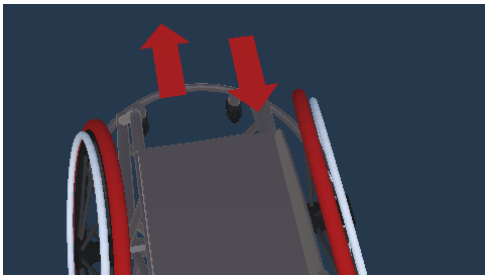


Figura 25. Flechas que indican en qué sentido se está moviendo cada joystick.

Construcción del Player

La construcción del personaje una vez se tiene el movimiento de la silla es muy simple, solo se necesita incluir el prefab OVRCameraRig del paquete de integración de Oculus, que ya incluye lo necesario para que funciones correctamente la cámara. Este prefab se debe incluir dentro del objeto Player, para que se mueva al mover el objeto padre.

Construcción de una escena

Una vez se tienen los Assets listos, se puede construir una escena inicial y hacer que todo esté en sintonía. Lo primero es saber con qué se cuenta y que se puede utilizar del paquete Oculus Integration:

1. Player, como se ha visto anteriormente incluye toda la funcionalidad básica, con un movimiento definido por el joystick de los controles además del prefabs de la cámara.
2. Para los Assets del entorno es necesario crear u obtener modelos 3D para construir la escena de la forma más realista posible, como ya se ha visto anteriormente. Para esto se han modelado el campo y otros modelos, además de que se ha incluido algunos Assets más del paquete de Oculus Integration.
3. Para iluminar las escenas en Unity existen diferentes técnicas. Se puede realizar con materiales que emitan luz, con una luz general para toda la escena o con puntos de luz estáticos. En este caso se utiliza una mezcla de estas técnicas. Primero definimos una luz general muy suave para mejorar las sombras, y posteriormente se posicionan luces direccionales encima del campo para iluminar la escena.
4. Una parte muy importante de todo escenario es un GameManager. Es una figura abstracta que gestiona todo el proceso lógico y configura los componentes para su correcto funcionamiento. En Unity se suele crear un objeto vacío que contiene un script, donde se define las funcionalidades y se gestionará la escena.

A continuación una imagen de la jerarquía de la escena dedicada al tutorial.

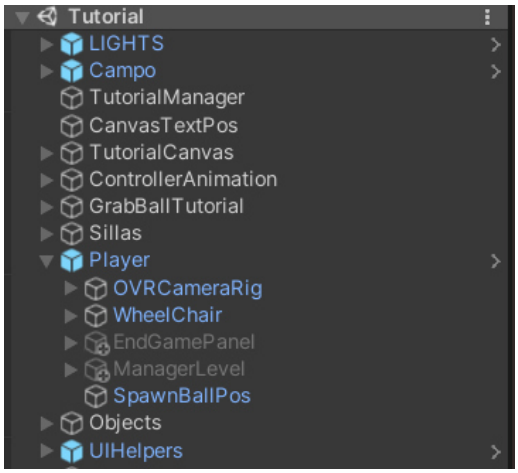


Figura 26. Jerarquía de la escena.

Estéticamente todo se tiene que posicionar a mano, de tal forma que exista cierta simétrica, puesto que se intenta simular un deporte competitivo.

Figura 27. Escena correspondiente al Tutorial.

Funcionalidad inicial y estructura

Llegados a este punto, se puede empezar a definir la funcionalidad y estructura de la aplicación. Comencemos con la estructura.

Estructura de las escenas

En estos casos, es importante definir una correcta abstracción así como una organización clara, que permitan realizar cambios sin que suponga un gran esfuerzo. Para lograr esto, es necesario tener un alto grado de modularidad, evitando las dependencias directas. Comencemos con la disposición de las escenas en Unity.

En la siguiente figura se puede apreciar la estructura en cuanto a escenas de la aplicación además del menú inicial que se muestra. Explicándolo con palabras el proceso sería el siguiente, cuando se inicia la aplicación automáticamente se carga la escena 1 y se muestra un menú para elegir entre empezar desde cero, pasando el tutorial y luego a la escena 3, o bien si ir directamente a las escena 3. Cuando la escena 3 finaliza se vuelve automáticamente al escena 1.

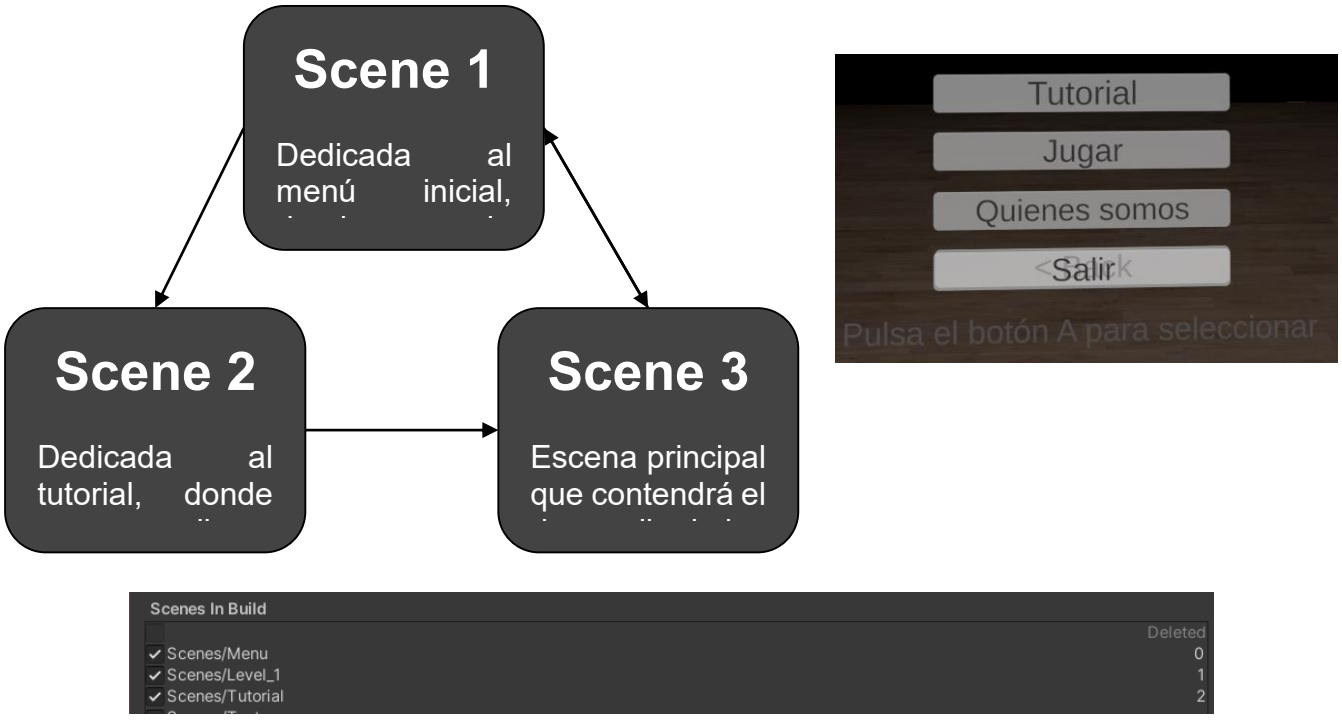


Figura 28. Estructura de la aplicación y menú inicial.

El orden que deben tener las escenas en los Build Settings en Unity es el mostrado en la imagen anterior.

La decisión de qué estructura y cuantas escenas utilizar en una aplicación de este tipo está basada en las partes que contendrá la aplicación. En este caso se tiene un menú principal, y dos niveles, uno de tutorial y otro de juego. Si se elige una sola escena y se centra todo en una sola jerarquía, para cambiar entre niveles se tendría que realizar muchos cambios en objetos como la iluminación y ciertas partes del campo, por lo que sería más trabajo. De lo contrario, separando las escenas se logra que al cargarlas estos objetos ya estén en la posición y ubicación que deseamos, sin necesidad de tocarlos. Es cierto que al usar tres escenas aumenta un poco el tamaño del ejecutable .apk de la aplicación, pero en este caso un aumento de unos pocos MB no supone un problema.

Escena 1: Menú inicial

Siguiendo con lo mencionado anteriormente, la construcción de la escena 1 es relativamente fácil, basta con crear un menú, utilizar el prefab de OVRCameraRig y definir un script que sea el gestor de la escena.

La escena se estructura en los siguientes componentes:

1. Menú inicial: Para la creación del menú, primero creamos dos Canvas, correspondientes al menú principal y otro dedicado a mostrar información de los creadores de la aplicación, en el script de "MenuManager" se define la interacción de estos menús. Por último creamos un objeto vacío que contendrá a estos dos canvas. Es importante recordar que al tratarse de una aplicación de VR, los menús tienen que estar en el mundo, por esto se debe cambiar el parámetro Render Mode a World Space.

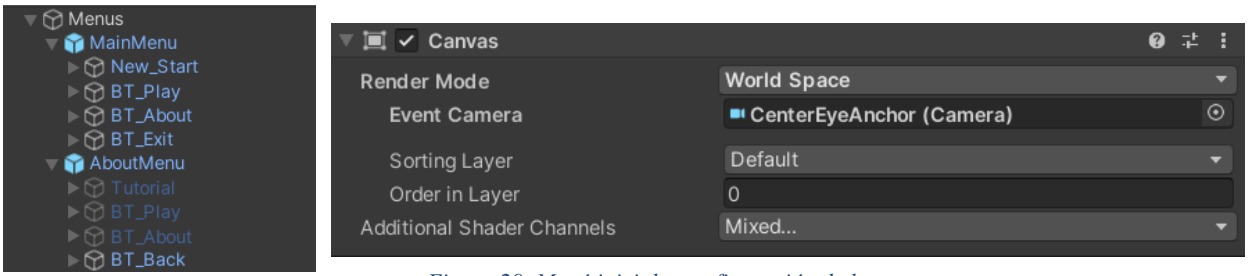


Figura 29. Menú inicial y configuración de los canvas.

También es necesario incluir el prefab UIHelpers a la escena y el script OVRRaycaster a los Canvas, si no la interacción con los menús no sería posible.

2. OVRCameraRig: solo se necesita que el usuario sea capaz de seleccionar las opciones adecuadas en el menú principal, por esto con incluir solo la cámara es más que suficiente. Recordemos que también se debe incluir el prefab de las manos, puesto que por defecto no viene incluido.

3. MenuManager: en este script se recogen las funcionalidades de esta escena. Para usarlo es necesario asignarla a un objeto vacío en la escena. A continuación se describen los principales métodos de este script y su función.

   `AboutProject()` muestra u oculta el menú y la información del proyecto.

   `exitgame()` Se cierra la aplicación

   `LoadSceneByName(string levelName)` Carga la escena con el mismo nombre que el parámetro de entrada.

4. Ambientación: esto se realiza a prueba y error, puesto que se trata de ambientar la escena y posicionar todo los objetos de la misma. En este caso se ha decidido hacer una ambientación oscura mostrando solo el menú principal y muy poco la pista de baloncesto.

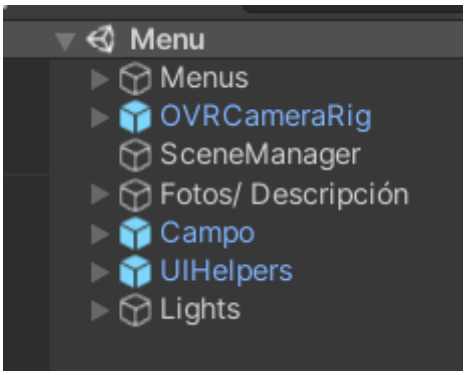


Figura 30. Jerarquía de la escena 1.

Escena 2: Tutorial

En esta escena se va a explicar brevemente cómo interactuar correctamente con la aplicación, además, se le dará libertad al usuario para que pueda realizar parte de los niveles de forma libre y sin presión de tiempo.

La primera parte del tutorial consistirá en una breve explicación con diferentes partes. En cada uno se mostrará información crítica para interactuar correctamente con la aplicación.

En la segunda parte el usuario tendrá la libertad de interactuar con los niveles (Tiro, pases e intercepción de pases) sin límite de tiempo, decidiendo cuando quiere pasar al siguiente. De esta forma se logra que la interacción posterior con la escena 3, donde si se cuenta el tiempo, sea más fluida.

Componentes del tutorial

Para que las explicaciones se entiendan correctamente por el usuario es necesario hacer que sean lo más gráficas posibles, de forma que con pocos segundo se entienda que se quiere decir.

Mandos: En el paquete de integración de oculus se incluye un prefab llamado OVRControllerPrefab que permite visualizar los mandos de la gafas. Esto es ideal para mostrarle al usuario la forma de los mandos y con qué botones se cuenta. Estos prefabs se escalan para que sean más visuales.

Figura 31. Mandos de ejemplo que estarán a la derecha del jugador para que se pueda ver qué botones se tienen.

Sillas: Para enseñar el movimiento de la silla, primero se muestra a la derecha del jugador el modelo de dos sillas de ruedas, y posteriormente se le incita a que se mueva por el campo e intente girar alrededor de un cono que tiene detrás

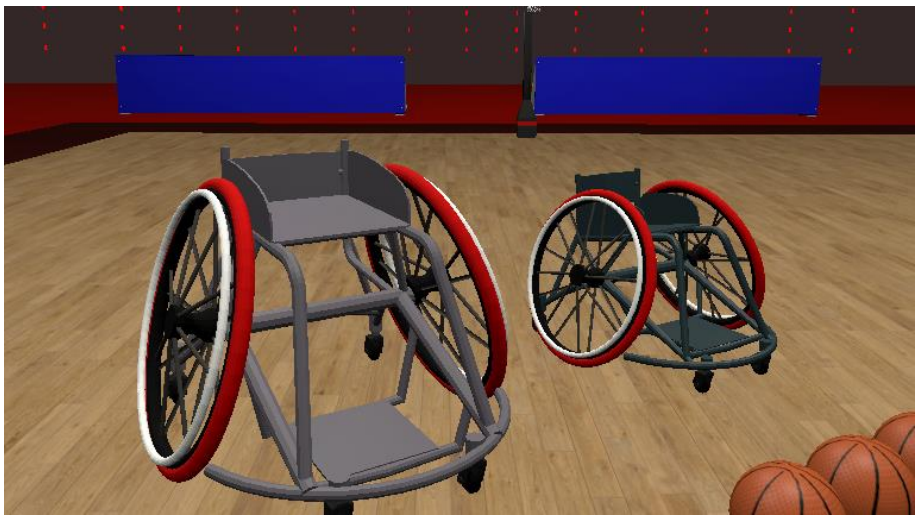


Figura 32. Sillas de ruedas de ejemplo.

Coger el balón: También es necesario que se explique la acción de agarrar objetos, por esto se incluye dos filas de balones a los laterales del jugador.

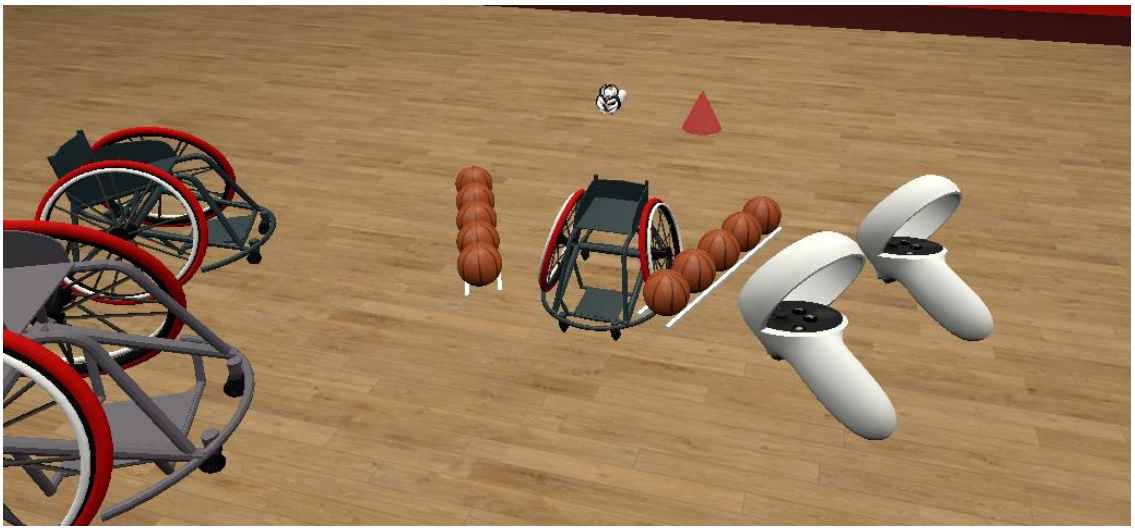


Figura 33. Fila de balones que se pueden agarrar como ejemplo para que el jugador se familiarice con el gesto de agarrar.

Explicaciones: A todo esto se suma, que delante del jugador se muestran unas letras blancas con fondo negro indicando que se debe hacer en cada momento. En la siguiente imagen se pueden ver todos los Canvas mostrados en fila, luego durante el transcurso del tutorial se irán posicionado por código para que se muestran delante del jugador.

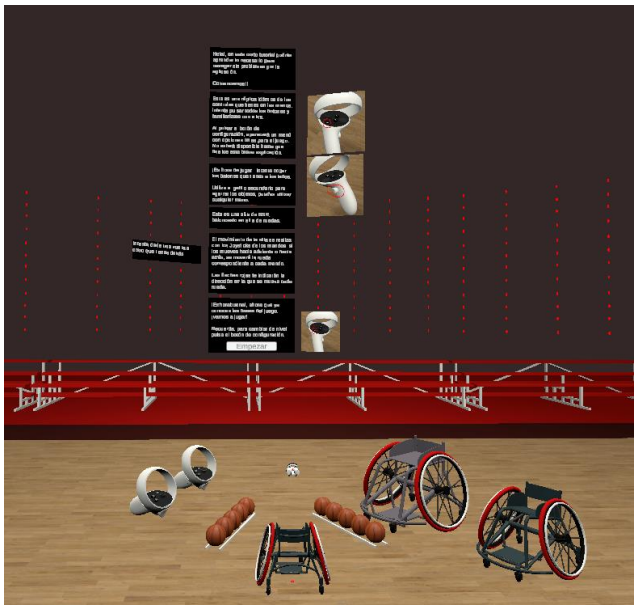


Figura 34. Objetos del tutorial.

Posteriormente, luego de las explicaciones, el usuario se podrá mover libremente entre las tres situaciones de tiro, pases e intercepción de pases sin límite de tiempo mediante un menú. Además podrá elegir en cualquier momento cuándo salir de la App.

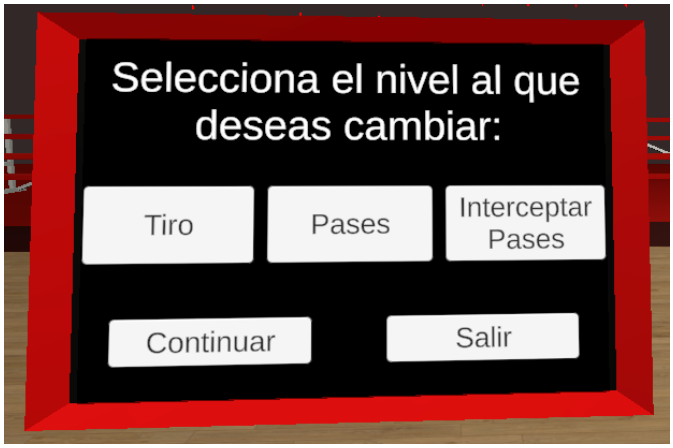


Figura 35. Menú que permite cambiar de escenario en el tutorial.

A continuación se explicará cómo se construyen los niveles de tiro, pases e intercepción de pases. En el nivel tutorial se incluirá una versión simplificada del primer nivel de dificultad, puesto que no se exigirán tiempo.

Escena 3: Niveles

En esta escena se desarrollan las funcionalidades y mecánicas de las que se habla anteriormente. Como es de esperar, la construcción de esta escena no será fácil, al juntarse todos los escenarios y las diferentes dificultades, la jerarquía será compleja y la comunicación entre todas las partes debe ser fluida. Empecemos por definir el flujo de la escena.

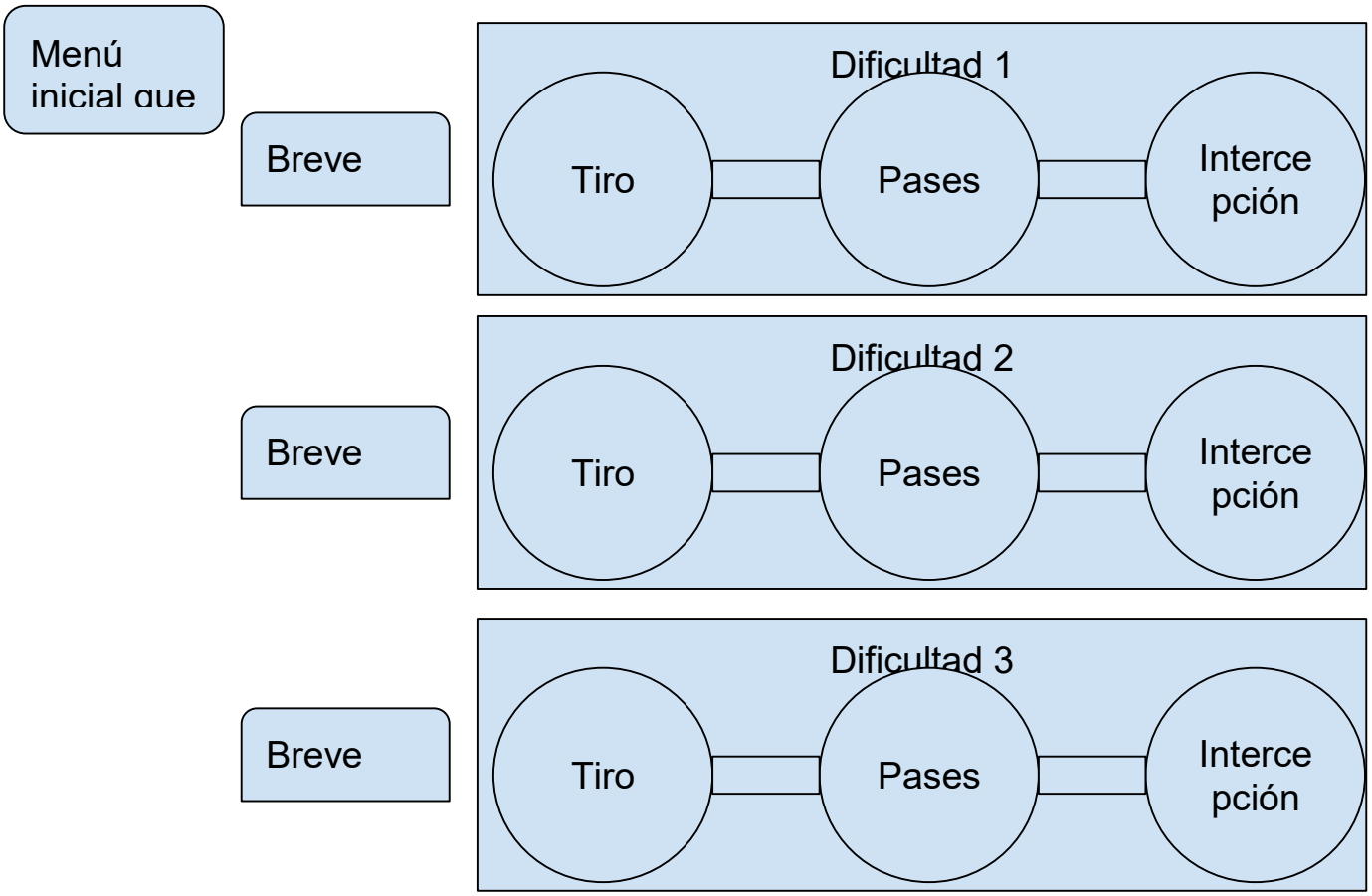


Figura 36. Estructura de la escena 3

En la figura anterior se puede ver la estructura ya mencionada anteriormente.

Para que la explicación de cómo se ha desarrollado la escena 3 y sus niveles sea clara y comprensible, se dividirá en varias partes.

Arquitectura óptima en proyectos con Unity

Antes de empezar a crear scripts y programar, es necesario saber qué criterios se deben seguir y cómo afrontar el aumento de complejidad. Primero que todo definamos 5 principios básicos a seguir en todo momento.

- Sistema SIN DEPENDENCIAS directas: logra que eliminar un componente en concreto no suponga que toda la aplicación se paralice, o al menos no suponga mucho trabajo realizar este cambio. Para lograr esto se puede utilizar la comunicación a través de eventos, que al ser lanzados y escuchados por los scripts, no hay una dependencia directa en la comunicación.
- MÍNIMA TRANSFERENCIA de información entre escenas: esto significa que cuando se lanza una escena desde otra escena, se pasen la menor cantidad de datos posibles, de tal forma que todo sea cargado de la base de datos.
- MAXIMIZAR EL USO DE PREFABS en escenas: hacer uso de prefab significa crear un único objeto en el proyecto que será utilizado por diferentes escenas, de tal forma que si se modifica este objeto se modificarán todas sus copias.
- CONFIGURAR DATOS sin programar: esto se logra mediante el inspector, la ventana que generalmente está a la derecha y cuando se pulsa sobre un objeto nos da información acerca de este objeto.
- MODIFICAR LÓGICA sin programar: para conseguir este objetivo, es necesario tener una correcta configuración de datos mediante el inspector. Para esto hay que diseñar los script de tal forma que tengan parámetros claves que sean visibles y editables en la interfaz.

Estos criterios son importantes si se desea que la estructura del proyecto sea modular, puesto que al aumentar la complejidad y el número de componentes usados, es más fácil gestionarlos si por ejemplo se ha definido una comunicación sin dependencias directas y con eventos.

GameManager y Gestores de niveles

El gestor del juego, o comúnmente llamado GameManager, es un concepto abstracto que se aplica al desarrollo de videojuegos, pero que en Unity se suele aplicar de una forma específica. El objetivo de este script es definir un lugar donde crear la lógica principal, de esta forma se mantiene una estructura limpia.

Primero se crea en la escena un GameObject vacío, es decir, no cuenta con componentes de física ni mallas que le hagan tener forma, por lo que no tiene presencia en el render de la cámara. A este gameobject se le agrega usualmente un script, que tendrá el mismo nombre, GameManager. Este script contendrá toda la funcionalidad que hace referencia al nombre.

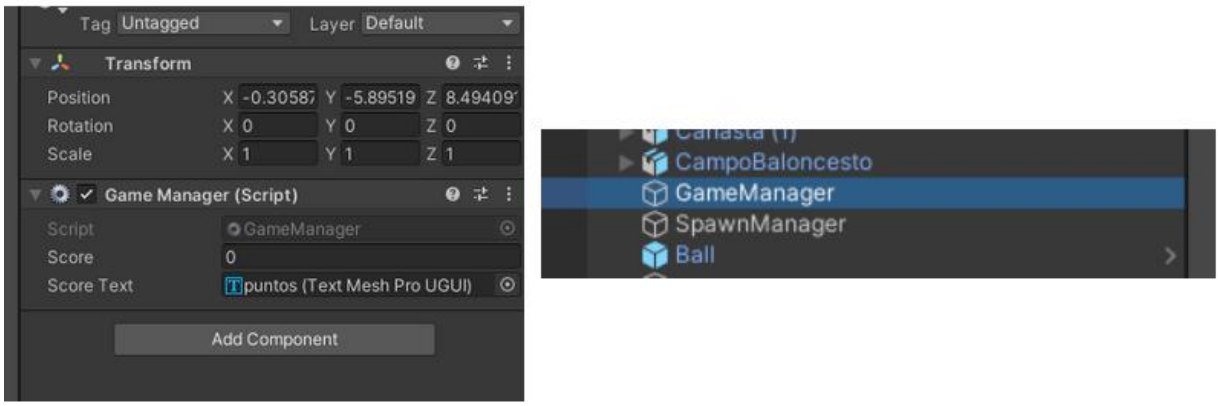


Figura 37. GameManager

Este script no es el único que se utiliza para gestionar el flujo del juego. Se han creado otros 6 scripts dedicados a gestionar los niveles, las puntuaciones, las posiciones del jugador y los avatares en la escena.

LevelManager_1: encargado de gestionar el nivel de Tiro, en todas sus dificultades.

LevelManager_2: encargado de gestionar el nivel de Pases, en todas sus dificultades.

LevelManager_3: encargado de gestionar el nivel de Interceptar Pases, en todas sus dificultades.

RestManager: Gestiona las pausas entre nivel de dificultad, explicando al usuario el siguiente nivel de dificultad.

ScoreManager: Este script se encarga de decidir cuándo una acción realizada por el usuario es buena y por lo tanto se suma la puntuación que se va obteniendo.

PositionManager: Posiciona en la escena a todos los objetos involucrados en cada nivel.

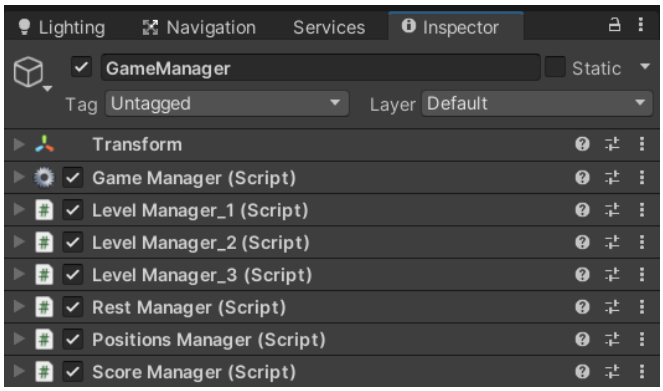


Figura 38. Scripts utilizados para gestionar el flujo de la aplicación.

Como es de imaginar en el script GameManager se concentra el flujo de la escena, de forma que es en este script donde se decide qué hacer en cada momento. Lo que no se decide es cómo hacerlo, puesto que esta funcionalidad se delega a los demás script logrando que sea fácil cambiar algo en el flujo sin alterar la funcionalidad. Para que el resto de componentes de la escena decidan qué acciones realizar en un momento concreto se crearon dos variables actualizadas por el GameManager que definen el estado de la escena en cada momento.

```
[SerializeField] public GameSatus;
[SerializeField] public Difficulty;

public enum GameSatus
{
    waiting,
    level_1,
    level_2,
    level_3,
    rest
}

public enum Difficulty
{
```

```
    difficulty_1,
    difficulty_2,
    difficulty_3
}
```

Figura 39. Variables que definen el estado del juego en todo momento.

Control del flujo de la aplicación

Para crear el flujo de la aplicación y definir cuándo se muestra o se ejecuta cierta acción se utilizan las corrutinas. En Unity son ampliamente utilizadas para crear flujos de ejecución paralelos que permiten crear comportamientos y eventos. A continuación se muestra parte del código de la corrutina que dirige el flujo principal de la escena 3.

```
[SerializeField] private int restTime = 15;
[SerializeField] private int timeOfGame = 40;
[SerializeField] private int timeOfInactivity = 10;

IEnumerator StartGame()
{
    GameState.game_state(0,1);
    Timer.set_up_time(3, restTime);
    yield return new WaitForSeconds(restTime);

    //.....................................Difficulty 1
    Scoreboard.scoreboard_state(1,1);
    difficulty = Difficulty.difficulty_1;
    gameSatus = GameSatus.level_1;
    GameState.game_state(1,1);
    Timer.set_up_time(1, timeOfGame);
    yield return new WaitForSeconds(timeOfGame);
    EndGameLevels.End_Game(1);
    gameSatus = GameSatus.rest;
    Timer.set_up_time(2, timeOfInactivity);
    yield return new WaitForSeconds(timeOfInactivity);

...

}
```

Figura 40. Corrutina que dirige el flujo principal de la escena 3.

Como se puede apreciar se cambia el valor de las variables que definen el estado del nivel y además se lanzan eventos que advierten a otros script de realizar ciertas acciones.

Comunicación a través de Eventos

Para la comunicación entre los scripts se utilizaran eventos, de esta forma se simplifica el proceso de comunicación, puesto que no se tiene que configurar a mano en el inspector de Unity. Además, se cumple con el criterio de evitar las dependencias directas, con el fin de maximizar la modularidad.

Primero es necesario crear la clase que contiene al evento, en este caso se crea dentro del script de GameManager.

```
public static class EndGameLevels
{
    public static Action<int> End_Game;
}
```

Para suscribir un método al evento lo hacemos en el método Start de la clase, de esta forma según empiece "la vida" del script se suscribe al evento el método que indiquemos.

```
void Start()
{
    EndGameLevels.End_Game += EndLevel_1;
}

public void EndLevel_1(int level)
{
    if (level == 1)
    {
        _positionsManager.TurnOffAssets();
        StopAllCoroutines();
    }
}
```

Por último lanzamos el evento.

```
EndGameLevels.End_Game(1);
```

Otros scripts que aportan funcionalidad

Para realizar pequeñas acciones y mantener una buena modularidad del código, es necesario separar ciertas funciones. A continuación se explican algunos pequeños scripts utilizados para agregar funcionalidad a al nivel.

SpawnManager: se encarga de instanciar un balón delante del jugador para que se pueda agarrar y lanzar a canasta o realizar cualquier otra acción exigida. Está suscrito al evento SpownABall e instancia un balón cuando la acción spownABallAtPlayerPos es lanzada.

BallAtBasket: detecta cuando el balón ha entrado por una de las canastas y suma la puntuación lanzando el enevent Scoreboard.game_score().

ChangePosRandom: cambia el target u objetivo de posición, esto es útil en el nivel de pases, puesto que el target es a donde el usuario tiene que dirigir el pase para puntuar.

TimerManager: como su nombre lo indica, este script reacciona al evento Timer.set_up_time, y forma parte del objeto que conforma el marcador, se encarga de cambiar el tiempo en dependencia del número pasado por parámetro.

DetectCollisionsHand: detecta cuando el balón ha sido agarrado por alguna de las dos manos. Esto es útil en el nivel de intercepción de pases.

Acción de agarrar el balón

En este caso, la acción de agarrar objetos se resume solo en una, agarrar un balón de baloncesto. La funcionalidad necesaria para lograr la acción ya está incluida en el paquete de integración de Oculus.

Para realizar las acciones de agarrar un objeto y soltarlo, recordemos que se utilizan los scripts de OVRGrabber.cs, y OVRGrabbable.cs. El primero se asigna al objeto que represente el mando y el segundo al objeto que se pretende agarrar, de esta forma ambos objetos se comunican y permiten el movimiento.

Detección de buenas acciones

Cuando el usuario realiza una acción que debe ser puntuada positivamente, como lanzar a canasta durante el nivel de tiro y encestar, es necesario que la lógica lo detecte y sea capaz de diferenciar de una acción buena y una acción mala.

En Unity, la detección de objetos se realiza mediante el componente Collider, este componente puede tomar varias formas según se necesite.

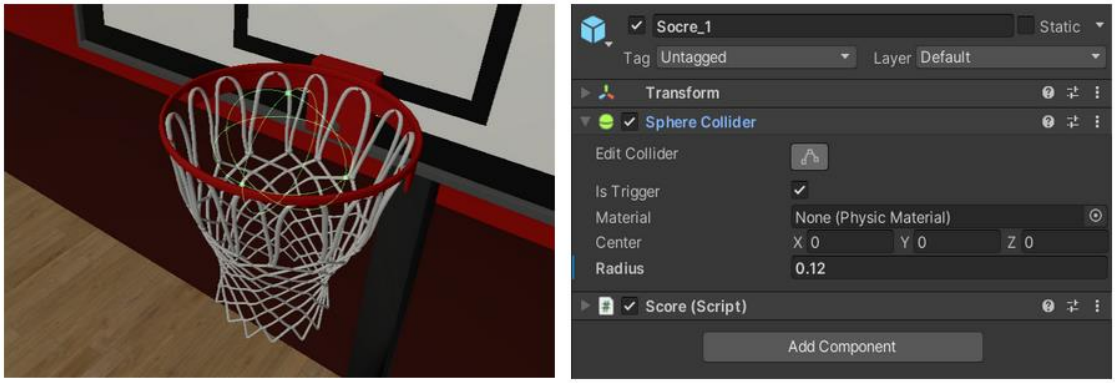


Figura 41. Componente utilizado para la detección del balón al entrar a la canasta.

La clave a la hora de utilizar un Collider para detectar otros objetos con un componente Collider es activar el campo “Is Trigger”, de esta forma se elimina las colisiones con cualquier objeto y simplemente hacer la función de disparador o detector de otros objetos.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Ball"))
    {
        if (other.gameObject.transform.position.y > 2.5f)
        {
            ScoreManager.game_score();
        }
    }
}
```

Figura 42. Método OnTriggerEnter utilizado para detectar colisiones.

Para lograr esto se utiliza un script asociado al objeto y el método OnTriggerEnter que detecta a otros objetos, luego se puede comprobar que efectivamente se trata del balón y que está por encima del aro, así nos aseguramos de que la canasta es válida.

Nivel de Tiro

El nivel dedicado al tiro se consigue, posicionando al jugador cerca de la canasta de tal forma que sea posible realizar un tiro. Para aumentar la dificultad se puede jugar con dos factores, la distancia del jugador a la canasta, y la presión que pueda sentir al aumentar el número de enemigos.

Nivel 1 de dificultad:

Figura 43. Tiro a canasta con un enemigo.

Nivel 2 de dificultad:

Figura 44. Tiro a canasta con dos enemigos.

Nivel 3 de dificultad:

Figura 45. Tiro a canasta con tres enemigos.

Nivel de Pases

El nivel dedicado a la realización de pases se consigue posicionando al jugador a cierta distancia de un compañero al que tiene que realizar el pase. Para aumentar la dificultad se puede jugar con dos factores, la distancia del jugador al compañero, y la presión que pueda sentir al aumentar el número de enemigos. En este nivel el pase tiene que estar dirigido a un objetivo que tiene un color rojo, este objetivo puede cambiar de posición.

Nivel 1 de dificultad:

Figura 46. Realizar pases con un enemigo, el compañero es de color azul.

Nivel 2 de dificultad:

Figura 47. Realizar pases con dos enemigos.

Nivel 3 de dificultad:

Figura 48. Realizar pases con tres enemigos.

Nivel de Interceptar el Pase

El nivel dedicado a la intercepción de pases se consigue posicionando al jugador a cierta distancia de un enemigo que va a realizar un pase a otro enemigo. Para aumentar la dificultad se pueden modificar ciertos aspectos del pase a interceptar, como la velocidad y el ángulo. Además al aumentar el número de enemigos al que puede estar dirigido el pase, también crea cierta incertidumbre y por lo tanto se aumenta la dificultad.

Nivel 1 de dificultad:

Figura 49. Interceptar pases con solo un posible destino.

Nivel 2 de dificultad:

Figura 50. Interceptar pases con dos posibles destinos.

Nivel 3 de dificultad:

Figura 51. Interceptar pases con dos posibles destinos.

Descanso entre niveles

Entre cada cambio de dificultad existirá un descanso en el cual se posicionará al jugador frente a una pizarra donde se mostrará información sobre el siguiente nivel de dificultad a afrontar.

Figura 52. Descanso entre niveles de dificultad.

Imágenes del resultado

Al tratarse de una aplicación gráfica, hay muchos conceptos que son difíciles de entender si no se tiene una imagen que los describa, por esta razón a continuación se muestran una serie de imágenes de los diferentes niveles y escenas.

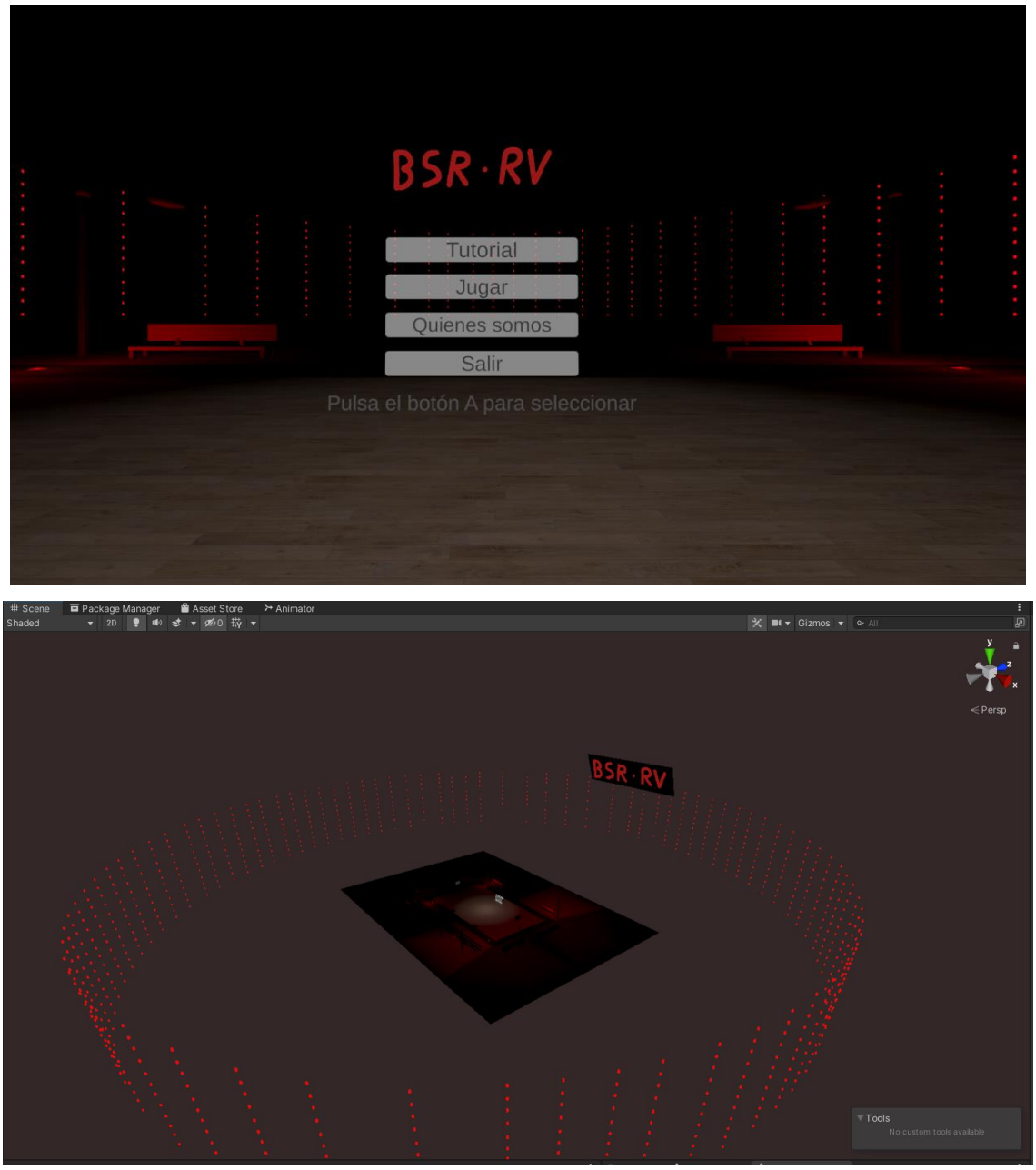


Figura 53. Escena inicial y menú principal.

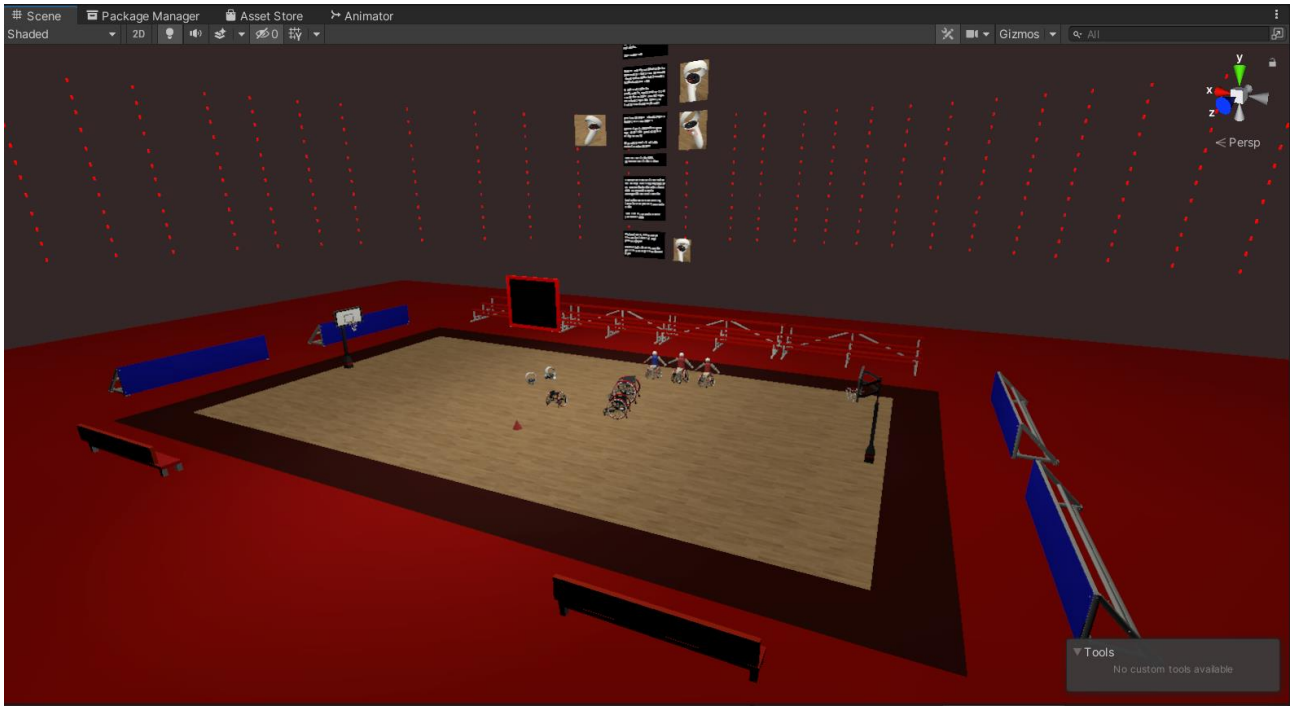


Figura 54. Visión global de la escena correspondiente al Tutorial.

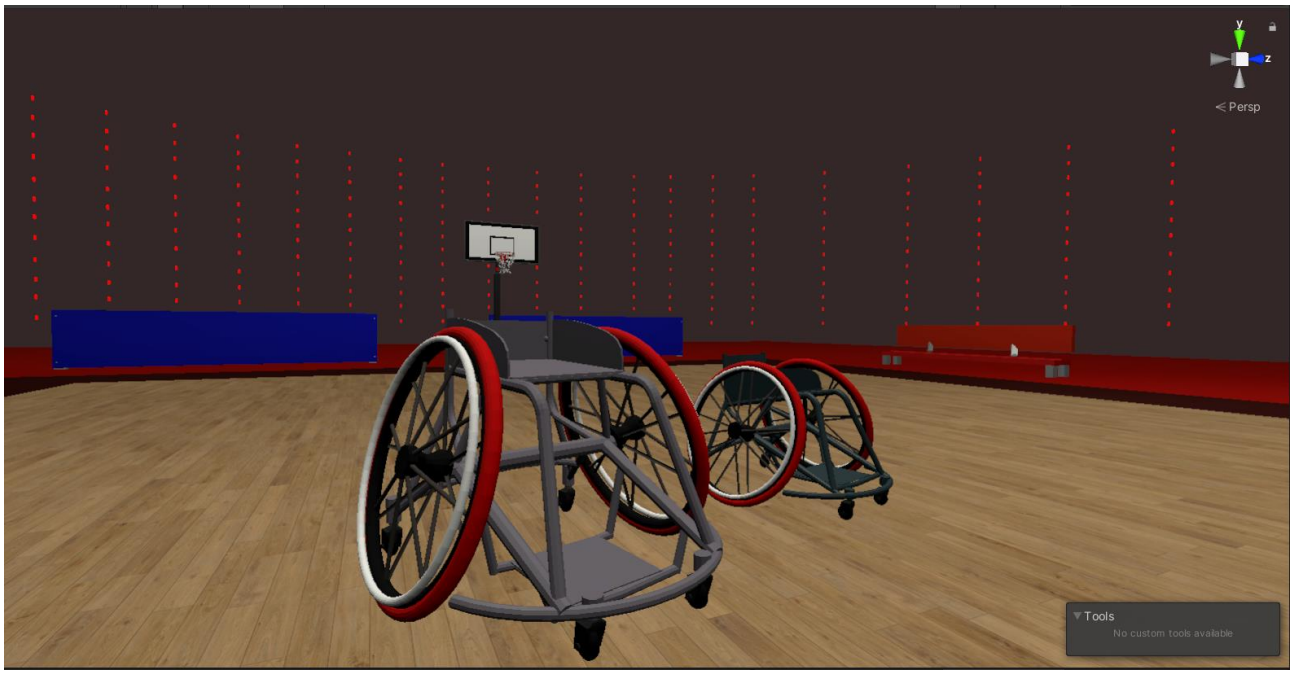


Figura 55. Sillas de ruedas de competición mostradas en el Tutorial.

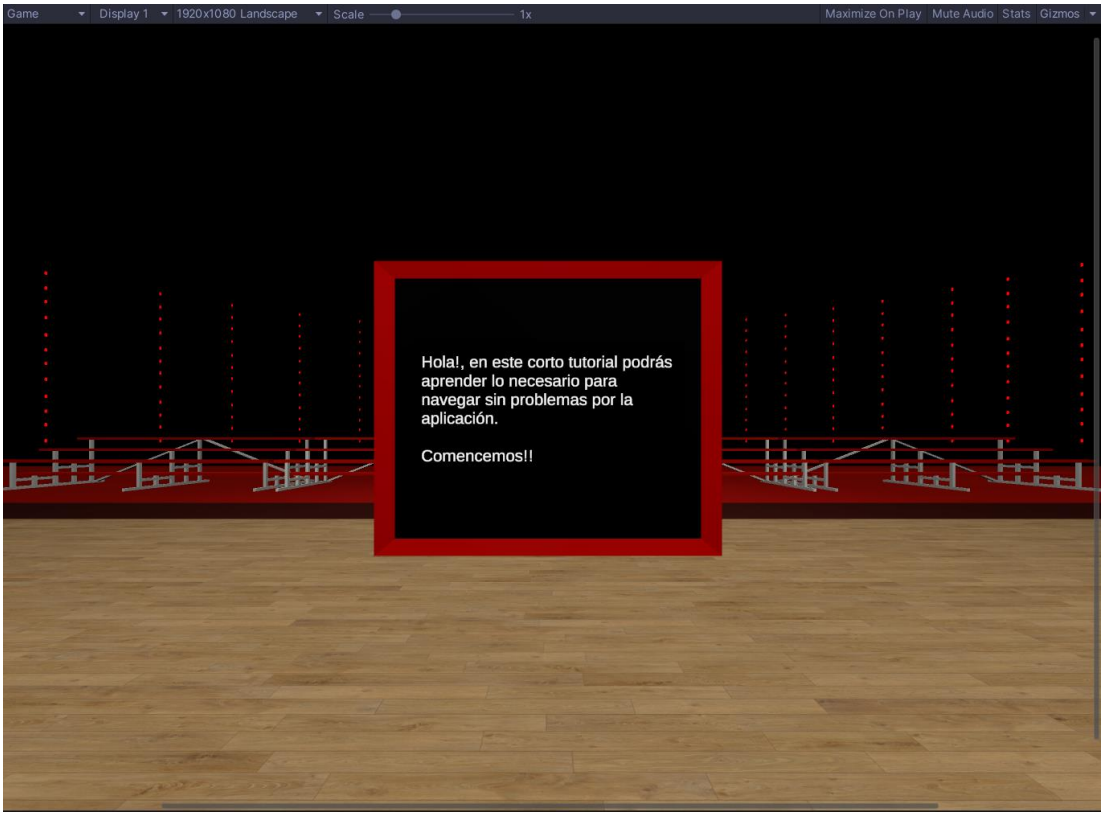


Figura 56. Primer texto que se muestra en el tutorial.

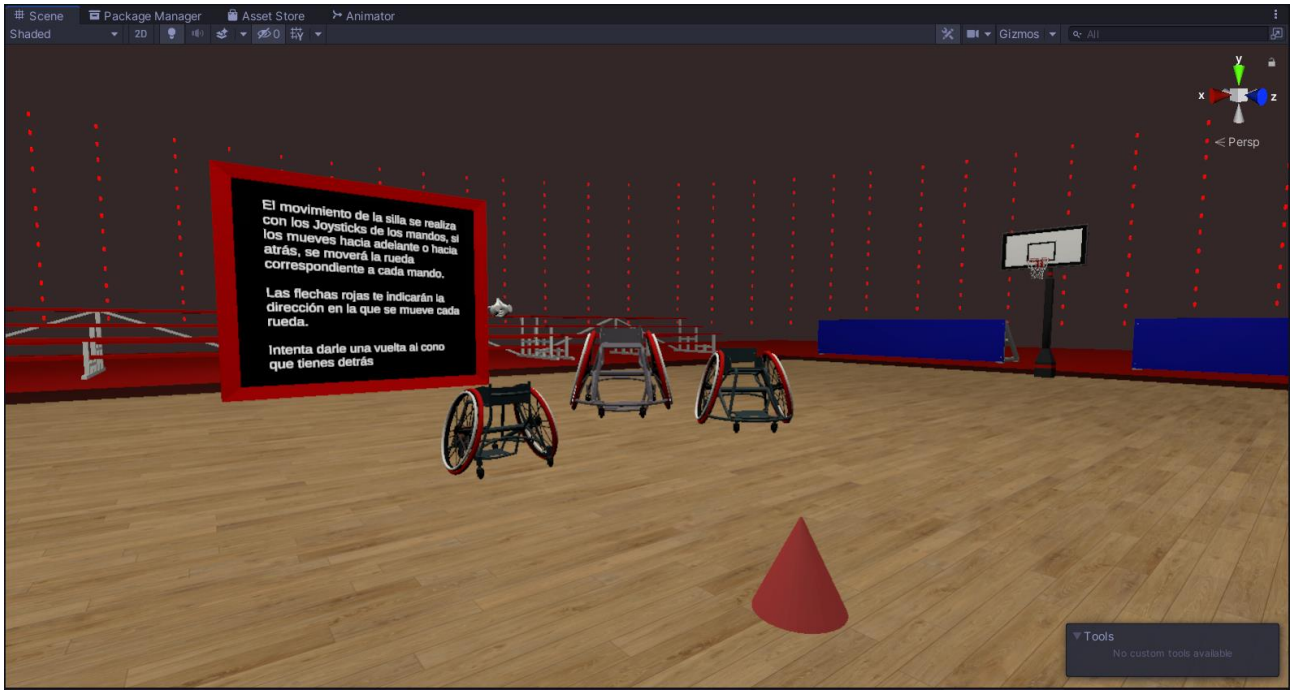


Figura 57. Explicación del movimiento de la silla en el tutorial.

Sonidos

Para mejorar la experiencia y aumentar la sensación de realismo de los usuarios es necesario incluir sonidos en la aplicación. Audacity es una aplicación multiplataforma que se puede usar para grabación y edición de audio. En este proyecto se ha utilizado Audacity puesto que ya se contaba con experiencia usando el software además de que no supone coste adicional al ser software libre.

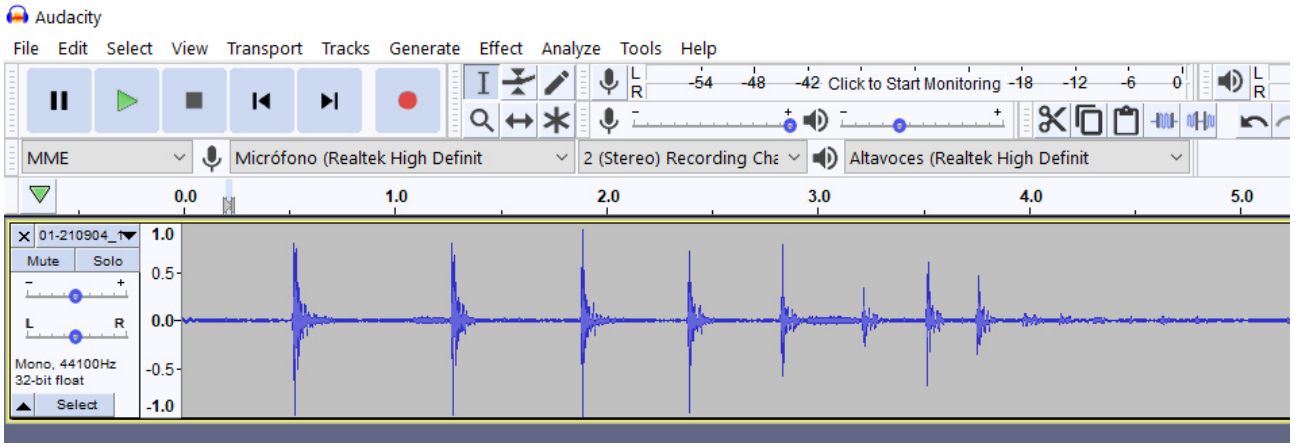


Figura 58. Sonido correspondiente al bote del balón.

Primero que todo necesitamos añadir el componente Audio Source al objeto que deseamos que reproduzca el sonido en cuestión y arrastrar el sonido al campo AudioClip de este componente. Para grabar estos sonidos se ha decidido utilizar un teléfono móvil, puesto que la calidad final después de retocar los sonidos ha resultado satisfactoria.

En este proyecto se han incluido tres tipos de sonidos:

- Bote del balón: cuando el balón toca el suelo este sonido será reproducido por el objeto que representa el balón. En la figura anterior se ven diferentes "picos" que representan el sonido de un balón que se deja caer desde una cierta altura.
- Sonido al encestar el balón: como su nombre lo indica se reproducirá en el momento que el balón pase por el aro.
- Música de fondo: esta música sólo se reproducirá en el menú principal y fue creada por el desarrollador de este proyecto exclusivamente para este fin.

Para reproducir sonidos en Unity, una vez se tiene el componente Audio Source añadido al objeto, se crea un script si no se ha creado para ese objeto y se añade el código que aparece en la siguiente figura. Como se puede apreciar lo que hace el siguiente código es detectar cuando el balón toca el suelo y en ese caso reproduce el sonido.

```
public class PlayAudio : MonoBehaviour
{
    public AudioSource audioSource;

    private void OnTriggerEnter(Collider other)
    {
       if (other.gameObject.CompareTag("Ground"))
       {
           audioSource.Play();
                                                                    }
    }
}
```

Figura 59.Código para detectar la colisión con el suelo del balón y reproducir un sonido.

Pruebas con usuarios

Finalmente una vez se tiene una versión estable de la aplicación es hora de llevarla al terreno real y obtener feedback de los usuarios. Los candidatos para esta tarea fueron los deportistas del Club Deportivo Zuzenak. En este caso es de agradecer contar con deportistas que practiquen habitualmente el deporte que se intenta imitar puesto que las valoraciones contarán con más criterio.

Después de que los usuarios probaron la aplicación se obtuvieron las siguientes conclusiones:

- En la escena correspondiente al tutorial era necesario ajustar los tiempos de espera y mejorar las explicaciones dadas a los usuarios sobre cómo utilizar la aplicación. En este caso simplemente se acortó las explicaciones y se aumentó el tiempo de espera entre cada una de ellas.
- El desarrollo de la aplicación es muy demandante con la lógica necesaria para que todo funcione correctamente. Como es de esperar el apartado gráfico quedó en un segundo plano

y no se cuidaron ciertos aspectos. Esto es algo que los usuarios transmitieron al terminar de probar la aplicación. La solución a esta cuestión es sencilla pero laboriosa, puesto que es necesario mejorar los modelos 3D y los colores usados.

- Los usuarios eran libres de moverse por todo el campo sin restricciones entre los niveles. Esto supone un grave problema puesto que si el usuario se posiciona lejos del aro en un nivel de tiro y luego es libre de moverse debajo del aro y encestar, el nivel de dificultad no cumpliría su objetivo.

La solución al problema de libertad de los usuarios durante los niveles no es muy compleja, pero lleva tiempo de implementarse. Primero que todo se necesita incluir un objeto más que se posicione justo donde debe ir el usuario en cada nivel y sus correspondientes dificultades. Este objeto será un simple cuadrado plano desplegado en el suelo, debajo del usuario, que delimita la zona donde el usuario debe estar para que la acción que se está realizando sea contada. Posteriormente se necesita crear un script que indique si el usuario está posicionado encima del cuadrado. Este script tendrá una variable pública que será consultada por el ScoreManager antes de incrementar la puntuación.

Figura 60. Cuadrado debajo del usuario que le indica a donde retornar si se mueve de posición.

7. Conclusiones

La realidad virtual es una tecnología que ha demostrado tener un alto potencial para simular situaciones de la vida real con un gran nivel de realismo. Esta cualidad se está explotando actualmente en muchos sectores de la industria, teniendo especial repercusión en sector del bienestar y la salud con el tratamiento de fobias y creación de herramientas para uso médico.

Tal y como se ha visto en este documento, enfocar esta tecnología en el deporte y explotar sus cualidades para mejorar el rendimiento de deportistas es algo que se lleva haciendo algunos años. En el ámbito del deporte adaptado, en cambio, no es muy común ver este tipo de herramientas.

El Baloncesto en Silla de Ruedas o BSR es una adaptación del baloncesto convencional para lograr que personas con algún tipo de discapacidad motora puedan practicarlo. En este trabajo se ha utilizado este deporte como base sobre la cual crear una herramienta que sirvan tanto a deportistas que lo practiquen como a personas ajenas al deporte. Para lograr esto, se ha diseñado un sistema de niveles donde cada nivel engloba un movimiento o acción propia del deporte. Con esto se ha conseguido que tanto usuarios ajenos al deporte como deportistas encuentren útil la aplicación.

Para la creación de la aplicación 3D se eligieron Unity como motor gráfico y Blender como software de modelado 3D. Unity tiene una curva de aprendizaje relativamente alta, aunque no es un motor gráfico complejo de usar. Su principal ventaja es la libertad que proporciona a los creadores, por esta razón es necesario tener claro cómo estructurar un proyecto y qué criterios seguir para agregar contenido. En cuanto a Blender, la complejidad está definida por el modelo 3D que se quiera construir, puesto que a mayor complejidad del modelo, más recurso y herramientas necesitamos conocer del programa.

En cuanto a los aspectos técnicos el desarrollo de este proyecto ha aportado una visión global del proceso de desarrollo de aplicaciones en Unity, conociendo no solo el propio motor gráfico, sino otras herramientas necesarias involucradas en el proceso. Y aunque ha habido problemas importantes a la hora de desarrollar por falta de experiencia en el ámbito, se ha realizado un buen proceso de documentación y aprendizaje que ha derivado en conocimientos sólidos de esta plataforma.

7.1 Desviaciones de tiempos

Los retrasos en el desarrollo del proyecto se han producido principalmente por encontrar dificultades al desarrollar, debido al desconocimiento de las principales herramientas que se utilizan en la implementación de la aplicación. Este desconocimiento de Unity producía fallos en la aplicación, sobre todo por malas decisiones tomadas en la implementación de funcionalidades del motor.

También podría llegar a tomarse como un retraso funcionalidades que se desarrollan a prueba y error, debido al hecho de no conocer la mejor alternativa para solucionar una cuestión dada.

En la planificación original se destinó 250h al desarrollo y 50h a la revisión final. En este caso el tiempo planificado para el desarrollo se ajustó bien al real. En cuanto al tiempo dedicado a la revisión final y ajustes de los niveles, las horas planificadas no se ajustaron a las reales. El problema reside en los pequeños ajustes que se hacían necesarios a medida que se probó más a fondo la aplicación con los usuarios. En concreto se detectó que era necesario ajustar mejor los tiempos de espera y las explicaciones dadas al usuario sobre cómo utilizar la aplicación. Además se incluyó un sistema para evitar que el usuario se moviese de la posición en la que aparece al inicio de cada nivel, evitando así posibles "trampas". Se estima que la desviación fue de unas 40h de trabajo añadido a las 50h calculadas inicialmente.

7.2 Trabajo futuro

El resultado final que se obtuvo es más que satisfactorio teniendo en cuenta los recursos con lo que se contaba. La falta de experiencia en el ámbito fue una de los problemas más grandes a sobrepasar, además de la dificultad añadida de tener que crear algo desde cero, con todos los altibajos que eso conlleva. A pesar de esto, el resultado final cumple los requisitos iniciales.

Al tratarse de una aplicación que se podría catalogar perfectamente de ser un videojuego, sumar funcionalidades no siempre es la mejor opción. Esto se debe a que si las mecánicas que están implementadas no funcionan todo lo bien que deberían, no tiene sentido agregar más si las que ya están se pueden mejorar. A continuación se listan algunas funciones que quedarían pendientes de mejorar.

- Interacción del usuario con el balón: aunque el nivel logrado es bueno, es cierto que en algunos momentos hay interacciones sobre todo con el balón que no terminan de sentirse sólidas. Esto pasa al agarrar el balón y lanzar a canasta, puesto que el gesto de la mano no acompaña a la acción realizada. Para pulir esta interacción es necesario adaptar el gesto de la mano al agarrar el balón, y el gesto que se ejecuta al finalizar el proceso de agarrar el balón. La dificultad recae en las diferentes situaciones en las que se pueden soltar y agarrar el balón, ya sea para tirar a canasta, dejarlo caer o realizar un pase.
- Interacción del usuario con la silla de ruedas: para lograr esta interacción se utilizaron los joysticks de los mandos, utilizando solo el desplazamiento vertical de estos. El valor de este desplazamiento se transmite a las dos anclas situadas al lado de la silla haciendo la función de las ruedas. El problema con este desplazamiento reside en que no se está realizando la acción de forma real, puesto que el gesto debería ser el de agarrar el aro de las ruedas y moverlo, de esta forma se movería la silla. Por esto, quedará pendiente para una futura actualización de la aplicación, mejorar esta funcionalidad e intentar realizar el movimiento de la forma más real posible con la acción de agarrar las ruedas.
- "Inteligencia" de los NPCs de la aplicación: en muchos juegos, la inteligencia que demuestran sus personajes o comúnmente llamados NPC (Non Playable Character) es una cuestión muy importante para lograr una experiencia satisfactoria del usuario. En este caso, se ha definido unas animaciones básicas y ciertos movimientos que se adaptan al comportamiento del usuario. Aunque para este caso son suficientes, lograr comportamientos realistas es importante si se pretende construir una experiencia satisfactoria. Por esto quedaría pendiente para una futura actualización mejorar el comportamiento de los NPCs.

Acrónimos

VR - Virtual Reality / Realidad Virtual

BSR - Baloncesto en Silla de Ruedas

Asset - Recursos que se utilizan para la construcción de una escena.

Prefab - Objetos reutilizables, creados dentro de la vista del proyecto.

GameObject - Objeto generalmente vacío, que solo contiene un componente de transform, es el pilar de Unity, puesto que prácticamente todo es un GameObject, lo que diferencia unos de otros son los componentes que contienen.

NPC - Non Playable Character / personaje no jugador o personaje no jugable.

Bibliografía

1. Unity

- Documentación oficial
  https://docs.unity3d.com/es/530/Manual/UnityManual.html
  https://unity3d.com/learn/tutorials
- Nicolas Alejandro Borromeo, 2020. “Hands-On Unity 2020 Game Development. Build, customize, and optimize professional games using Unity 2020 and C#.”
- Otras fuentes.
  https://docs.unity3d.com/es/530/Manual/ExecutionOrder.html
  https://unity3d.com/es/get-unity/download

2. Blender

- Documentación oficial
  https://docs.blender.org/manual/es/latest/
- Oscar Baechler y Xury Greer. 2020. “Blender 3D By Example: A project-based guide to learning the latest Blender 3D, EEVEE rendering engine, and Grease Pencil, 2nd Edition”
- Otras fuentes.
  https://www.blender.org/about/license/
  https://www.blender.org/download/

3. Documentación de Oculus

- Documentación oficial
  https://developer.oculus.com/documentation/unity

- Otras fuentes.

  https://developer.oculus.com/documentation/unity/unity-enable-device/

  https://developer.oculus.com/documentation/unity/unity-conf-settings/

  https://developer.oculus.com/documentation/unity/unity-utilities-overview/

  https://developer.oculus.com/documentation/unity/unity-add-camera-rig

  https://developer.oculus.com/documentation/unity/unity-ovrinput/

  https://developer.android.com/studio/command-line/adb

4. BSR

- Discapnet. 2018. “Historia del Baloncesto en Silla de Ruedas”
- Alejandro Rodriguez Montero. 2014. “Guía didáctica para el aprendizaje del baloncesto en silla de ruedas”
- Aarón Dávila García. 2016. “Manual Para El Entrenador De Baloncesto Sobre Silla De Ruedas”

5. Otras fuentes

- https://www.3dgep.com/understanding-quaternions/#Square_of_Complex_Numbers
- https://www.youtube.com/watch?v=ueKCh0pnJE8
- https://www.youtube.com/watch?v=toyqORA-i3k&t=276s
- https://www.youtube.com/watch?v=CPQjaenplJQ
- http://builder.openhmd.net/blender-hmd-viewport-temp/modeling/modifiers/index.html
- https://cgcookie.com/articles/blender-cycles-vs-eevee-15-limitations-of-real-time-rendering
- https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API/Concepts

- http://bsr.feddf.es/pdfs/normativas/7/REGLAS%20IWBF%205x5%20ESL%202021%20Marzo%20V2.pdf

Anexo I: Unity

Unity es un motor gráfico generalmente utilizado para el desarrollo de videojuegos. Reúne muchas características y es lo suficientemente flexible como para hacer casi cualquier juego o aplicación que se pueda imaginar. Con funcionalidades multiplataforma muy bien logradas, Unity es popular tanto entre los desarrolladores aficionados como entre los grandes estudios. Se ha utilizado para crear juegos como Pokemon Go, Hearthstone, Rimworld, Cuphead y muchos más.

Este motor gráfico ofrece una versión gratuita para que los desarrolladores puedan lanzar juegos sin tener que pagar por el software, siempre que sus ingresos sean menores a 100.000 dólares al año.

Cabe destacar que Unity ya cuenta con un manual de usuario donde se explican las características fundamentales del motor gráfico. Por esta razón, no tiene sentido crear otra documentación para este proyecto, simplemente se explicarán algunos conceptos muy básicos del motor.

¿Cómo descargar Unity?

Unity es muy sencillo de descargar e instalar. Para hacerlo, tenemos que dirigirnos a su página de descargas. Una vez se tenga Unity Hub, que es un gestor de descargas que permite gestionar diferentes versiones del software, junto con cualquier característica adicional que puedas necesitar. Para ello es necesario crear una cuenta y aceptar sus términos y condiciones.

Con Unity Hub, se puede elegir que versión para descargar. El instalador simplifica la instalación en sencillos pasos, si se desea desarrollar para Android, es aconsejable utilizar la versión 2019 o posterior, puesto que la comprobación de las herramientas SDK y NDK de Android y OpenJDK se realiza automáticamente. En este proyecto en concreto se utilizará la versión 2021.1.5f1.

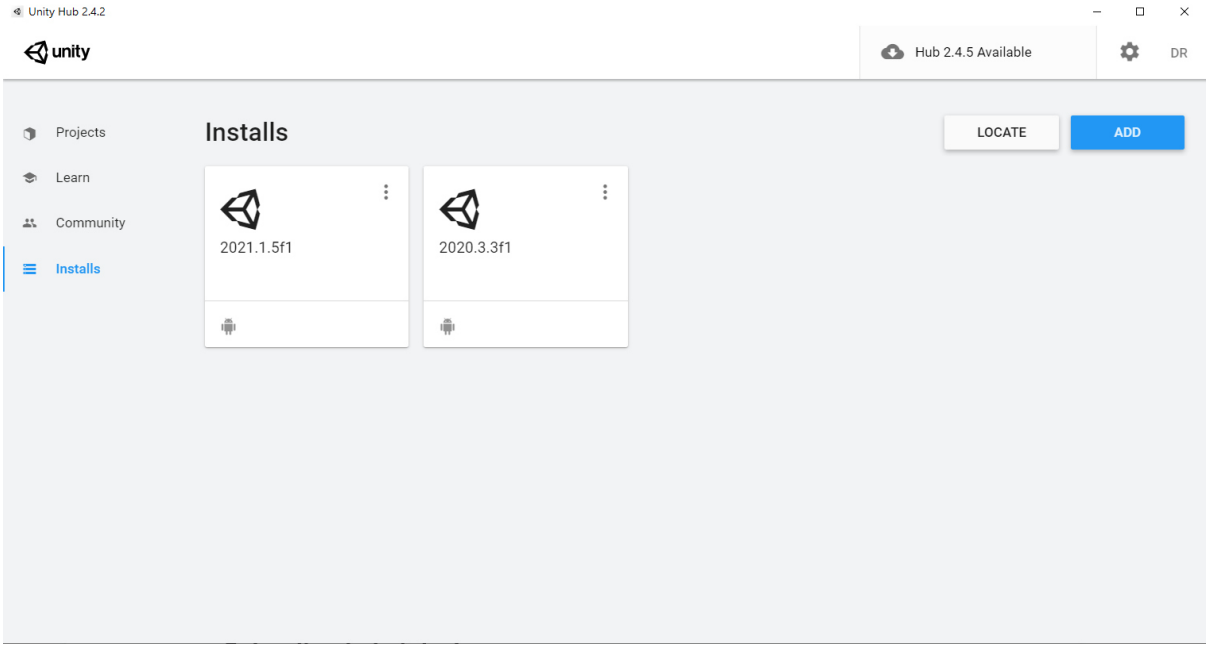


Figura 61. Unity Hub

¿Cómo aprender a utilizar este motor gráfico?

Aprender sobre Unity hoy en día es muy fácil, sobre todo por la gran comunidad de usuarios que tiene. Aparte de eso, la mejor opción es ir directamente a la documentación oficial o a los cursos para principiantes que ofrecen en su web oficial. De esta forma estaremos siempre actualizados con los últimos cambios que se realicen en el motor.

Entendiendo la interfaz

Cuando Unity arranca por primera vez, puede que el número de ventanas, iconos y opciones resulte abrumador. Por suerte, las cosas son más sencillas de lo que parecen.

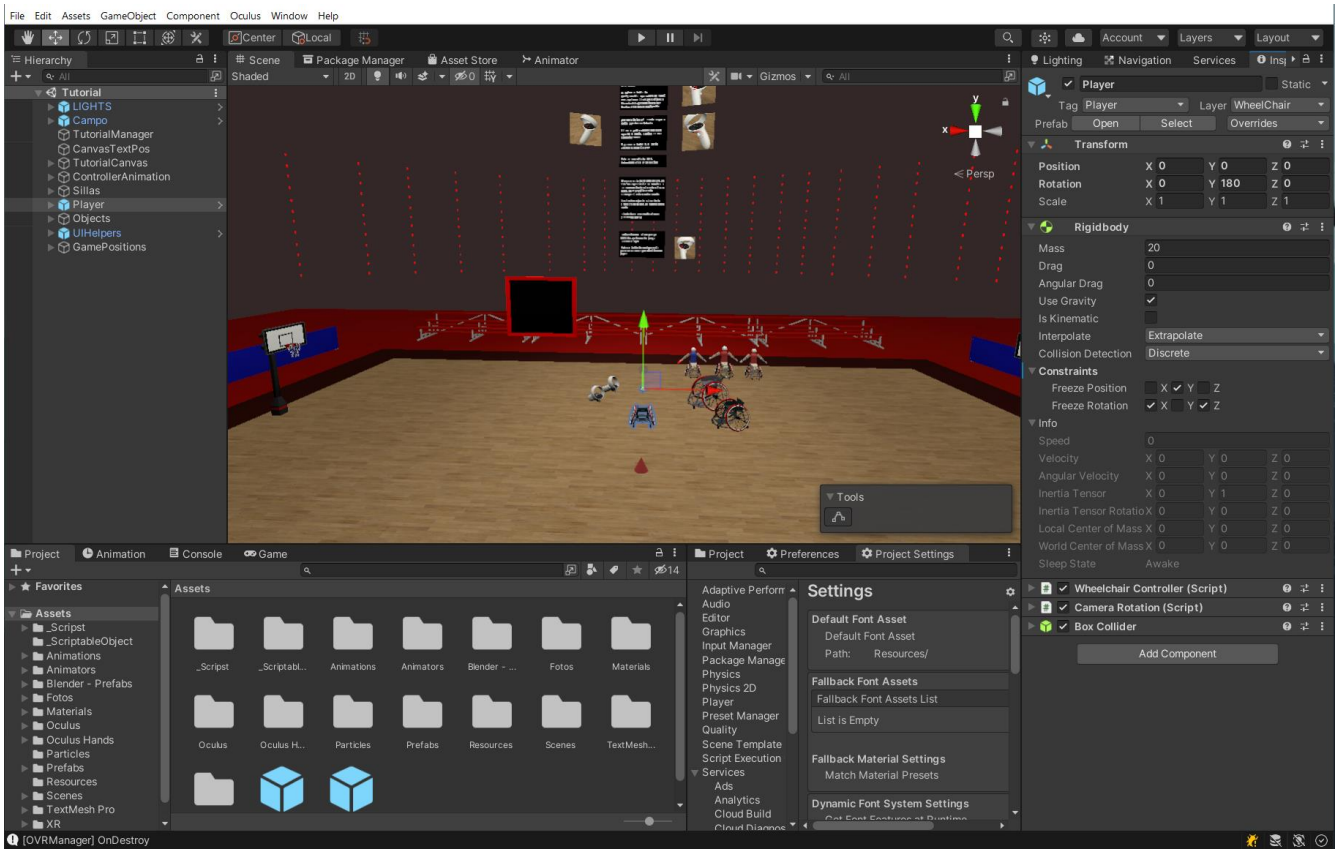


Figura 62. Interfaz de Unity.

Estas son las principales ventanas se usan y lo que hace cada una de ellas:

Jerarquía: por defecto, está en el extremo izquierdo, y muestra una larga lista de todos los GameObjects en la "escena". Esto hace que sea fácil localizar y seleccionar rápidamente cualquier aspecto del juego con el fin de cambiar sus propiedades. Los GameObjects son simplemente elementos que se incluyen en su juego, teniendo en cuenta que en Unity cualquier elemento que se pueda incluir en la escena es un GameObject, esté vacío o no.

Escena: es la ventana más grande en el centro. Esto muestra una vista del nivel actual, menú, o mundo del juego con el que estás trabajando actualmente (llamado "escena"). Esta ventana es donde se pueden arrastrar, soltar, crecer y encoger libremente GameObjects.

Los iconos que se encuentran a lo largo de la parte superior izquierda cambian la forma de interactuar con GameObjects y la escena, los llamados botones QWERTY. Nos permiten arrastrar la vista alrededor, mientras que las flechas permiten mover objetos en el espacio 3D a lo largo de los tres ejes.

Game: suele estar oculto detrás de la ventana de la escena y se puede acceder a él pulsando la pestaña de la parte superior. La vista de Juego muestra cómo se verá la escena tal y como es la escena en el juego. Esto significa que tendrá la misma perspectiva que la cámara y no se podrán mover los objetos. Aquí es también donde se reproduce el juego cuando se prueba.

Asset Store: la tienda de activos también se encuentra en una pestaña y dará acceso a los "activos" que han sido desarrollados por la comunidad.

Inspector: esta ventana se encuentra en el extremo derecho de la interfaz de usuario. El Inspector permite ver y editar las propiedades de un GameObject seleccionado. Esto podría significar cambiar el tamaño (escala) o la posición (transform), o podría significar añadir "componentes" como script C# o colliders.

Project: la ventana del proyecto se encuentra en la parte inferior de su pantalla y mostrará todos los archivos que componen el juego. Aquí es donde se crearan los scripts de C#. También se puede arrastrar y soltar archivos 3D o texturas aquí si se quiere usarlos en tu juego.

Console: finalmente, la consola es donde se puede ver la información del propio Unity. Esto permite saber si hay errores o advertencias en el código, o si hay problemas que necesitan ser abordados con la configuración del software en sí.

La base de Unity, MonoBehaviour

Cuando creamos un nuevo Script en Unity, junto a la cabecera de la clase aparece el nombre "MonoBehaviour", esto en términos simples nos está diciendo que el Script creado hereda funcionalidades de la clase MonoBehaviour. En otras palabras, Unity se encargará de ejecutar ciertas funciones automáticamente.

Por ejemplo la función "Start" se ejecutará cuando el GameObject al cual asignamos nuestro Script aparezca en la escena, al comenzar o cuando se instancie. Otra función que se ejecuta automáticamente es "Update", la cual se ejecuta antes de mostrar cada frame de la aplicación, por lo cual es una función que nos permite actualizar el estado. Esas dos funciones vienen definidas por defecto cuando creamos un nuevo Script en Unity, pero además hay otras funciones que podemos definir y se ejecutarán automáticamente en determinado momento del tiempo de vida de un MonoBehaviour.

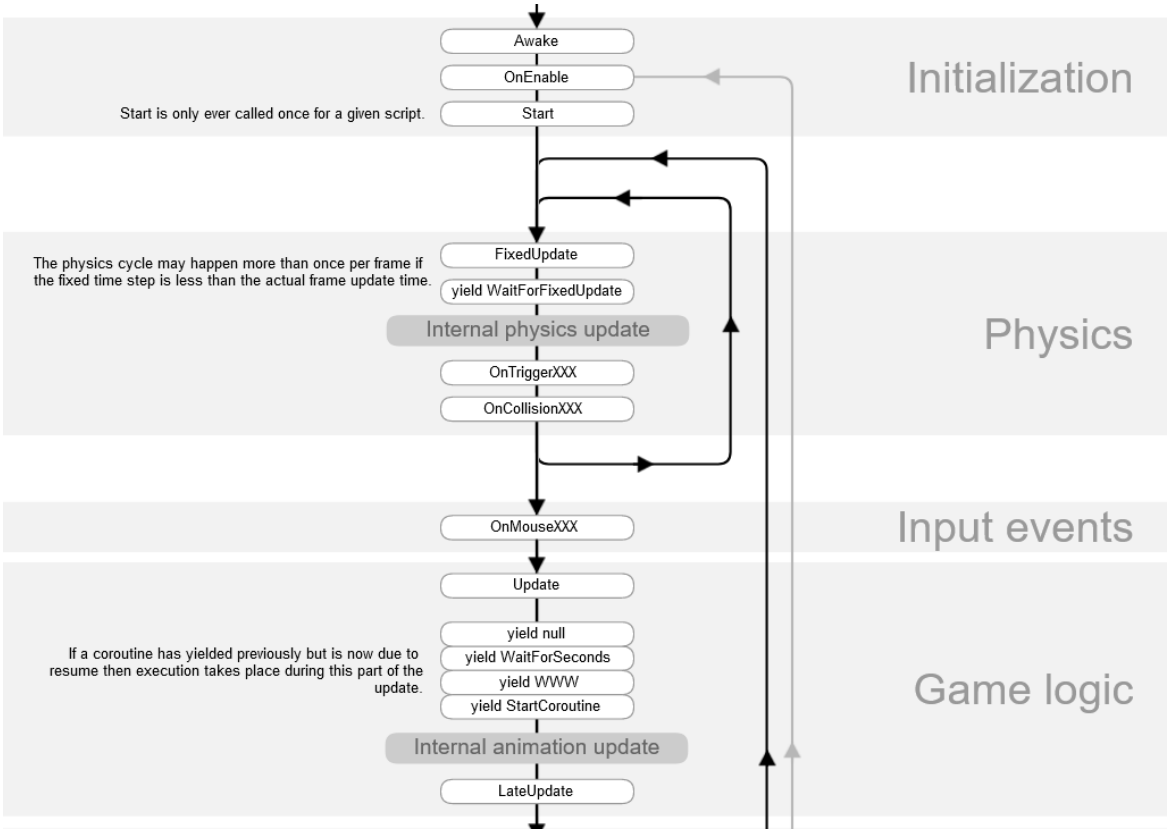


Figura 63. Orden de Ejecución de Funciones de Evento.

Como podemos apreciar en la figura anterior, hay muchos métodos que facilitan el desarrollo de la lógica de una aplicación. A continuación vemos el código de un scripts, que al ser asociado a un objeto, comprobar si el gameObject ha sido activado, y de ser cierto lo desactiva después de 3s.

```
public class DestroyAfterTime : MonoBehaviour
{
    [SerializeField] private int DestroyTime = 3;

    private void FixedUpdate()
    {
        if (gameObject.activeSelf)
        {
            StartCoroutine(DestroyObject());
        }
```

```
    }

    IEnumerator DestroyObject()
    {
        yield return new WaitForSeconds(DestroyTime);
        gameObject.SetActive(false);
    }
}
```

Figura 64. Script utilizado para ocultar un objeto después de ser activado.

En el código anterior también podemos ver en uso una corrutina, *IEnumerator DestroyObject()*. Las corrutinas en Unity se utilizan para frenar el flujo de ejecución durante cierto tiempo para realizar acciones como la anteriormente definida, ocultar un objeto después de un tiempo determinado.

Para añadir un script a un objeto en la escena simplemente se arrastra el script al objeto, o se añade mediante el botón "Add component"

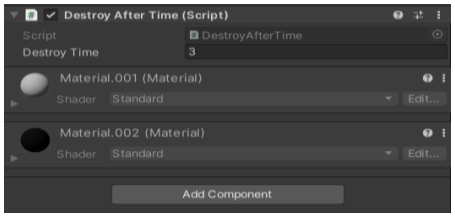


Figura 65. Script añadido a un GameObject.

Arquitectura óptima en proyectos con Unity

A la hora de hacer proyecto en Unity, hay que tener ciertos criterios en mente para hacer que la herramienta desarrollada tenga la mejor estructura posible. Para lograrlo es necesario tener en mente estas palabras: modular, escalable, configurable y depurable. Esto supone que los siguientes criterios deben cumplirse en la medida de lo posible:

- Sistema SIN DEPENDENCIAS directas: logra que remover un componente en concreto no suponga que toda la aplicación se paralice, o al menos no suponga mucho trabajo realizar este

cambio. Para lograr esto se puede utilizar la comunicación a través de evento, que al ser lanzado y escuchado por los scripts, no hay una dependencia directa en la comunicación.

- SEPARAR datos y lógica: cuando se realiza una aplicación de cualquier tipo, siempre se intentará separar la lógica o el funcionamiento de la aplicación, de los datos y el almacenamiento de los mismos. de esta forma se pueden realizar cambios en ambas partes sin que se afecten directamente. Para lograr esto se pueden utilizar los ScriptableObject para el almacenamiento de los datos y los MonoBehaviour para la lógica.
- MÍNIMA TRANSFERENCIA de información entre escenas: esto significa que cuando se lanza una escena desde otra escena, se pasen la menor cantidad de datos posibles, de tal forma que todo sea cargado de la base de datos.
- MAXIMIZAR EL USO DE PREFABS en escenas: hacer uso de prefab significa, crear un único objeto en el proyecto que será utilizado por diferentes escenas, de tal forma que si se modifica este objeto, se modificarán todas sus copias.
- CONFIGURAR DATOS sin programar: esto se logra mediante el inspector, la ventana que generalmente está a la derecha y cuando se pulsa sobre un objeto nos da información acerca de este objeto.
- MODIFICAR LÓGICA sin programar: para conseguir este objetivo, es necesario tener una correcta configuración de datos mediante el inspector. Para esto hay que diseñar los script de tal forma que tengan parámetros claves que sean visibles y editables en la interfaz,

Anexo II: Blender

Blender es un software dedicado a la creación de modelos tridimensionales. Es una herramienta muy potente que soporta todo el proceso de creación 3D: modelado, rigging, animación, simulación y renderizado, e incluso edición de vídeo y creación de juegos. Los usuarios avanzados emplean la API de Blender para la creación de scripts en Python con el fin de personalizar la aplicación y desarrollar herramientas especializadas que a menudo se incluyen en sus futuras versiones.

En este capítulo se detallarán muchas funcionalidades de Blender, pero como es de esperar faltarán muchas más por documentar. Por esta razón es importante recordar que no se pretende reemplazar la documentación oficial ya existente de este software, simplemente puntualizar ciertos aspectos muy utilizados en este proyecto.

La licencia

Como proyecto impulsado por la comunidad bajo la Licencia Pública General de GNU (GPL), el público está facultado para hacer pequeños y grandes cambios en la base de código, lo que conduce a nuevas características, correcciones de errores de respuesta, y una mejor usabilidad. La licencia concede a los usuarios una serie de libertades:

- Blender es software libre, para cualquier propósito
- Se puede distribuir libremente
- Se puede ver el funcionamiento interno y modificarlo
- Se pueden distribuir versiones modificadas de Blender

La GPL tiene como objetivo estricto la protección de estas libertades, requiriendo que todos compartan sus modificaciones cuando también compartan el software en público. Este aspecto se conoce comúnmente como Copyleft. Para más detalles sobre su licencia: license.

Descarga e Instalación de Blender

Está disponible en su página oficial para descargar de manera gratuita como se mencionó anteriormente. Su instalación es sencilla e intuitiva, no es necesario una configuración inicial.

Interfaz

Lo primero que tenemos que hacer es saber qué cosas del programa se pueden modificar. Al tratarse de un software open source, se pueden modificar muchísimos parámetros de la interfaz y el funcionamiento interno del programa. Accedemos a las preferencias, Edit => Preferences.

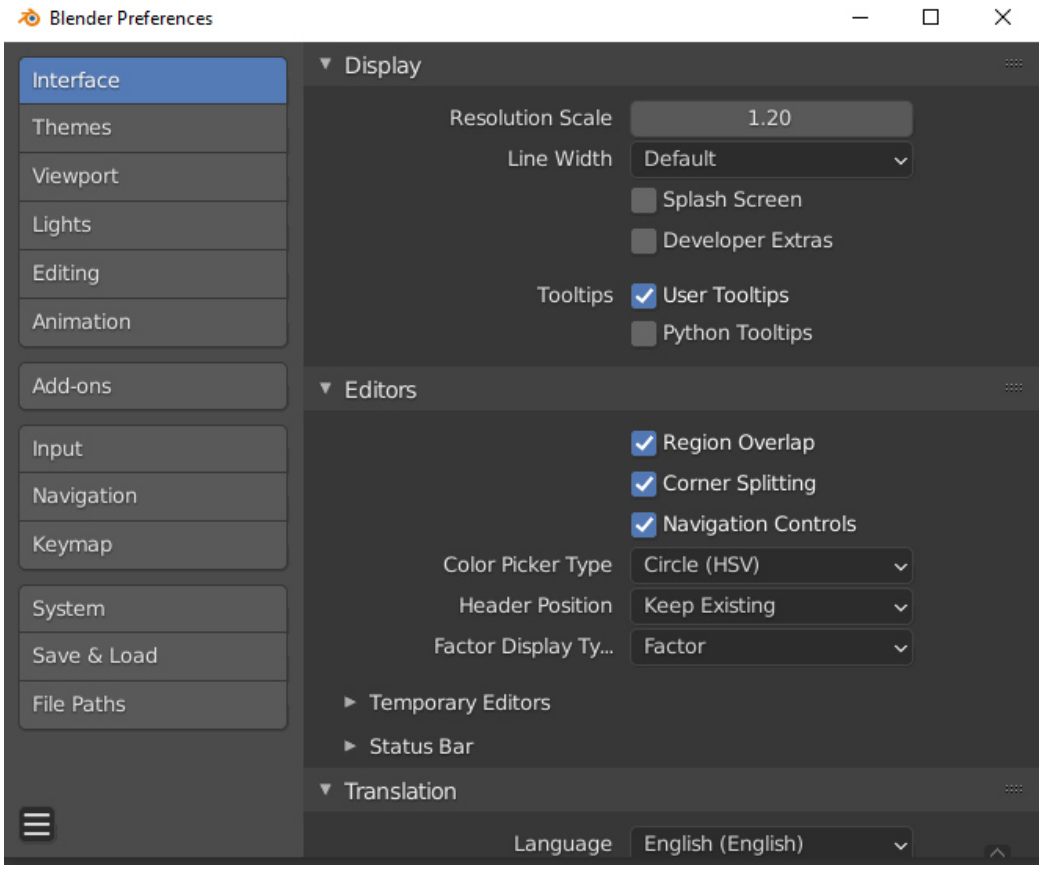


Figura 66. Panel de preferencias de Blender.

Blender, a diferencia de otros programas, no funciona con ventanas, en cambio su interfaz cuenta con un sistema de áreas y editores que se pueden configurar, en otras palabras, son como "azulejos" que podemos colocar unos al lado de otros para lograr tener la interfaz a nuestro gusto.

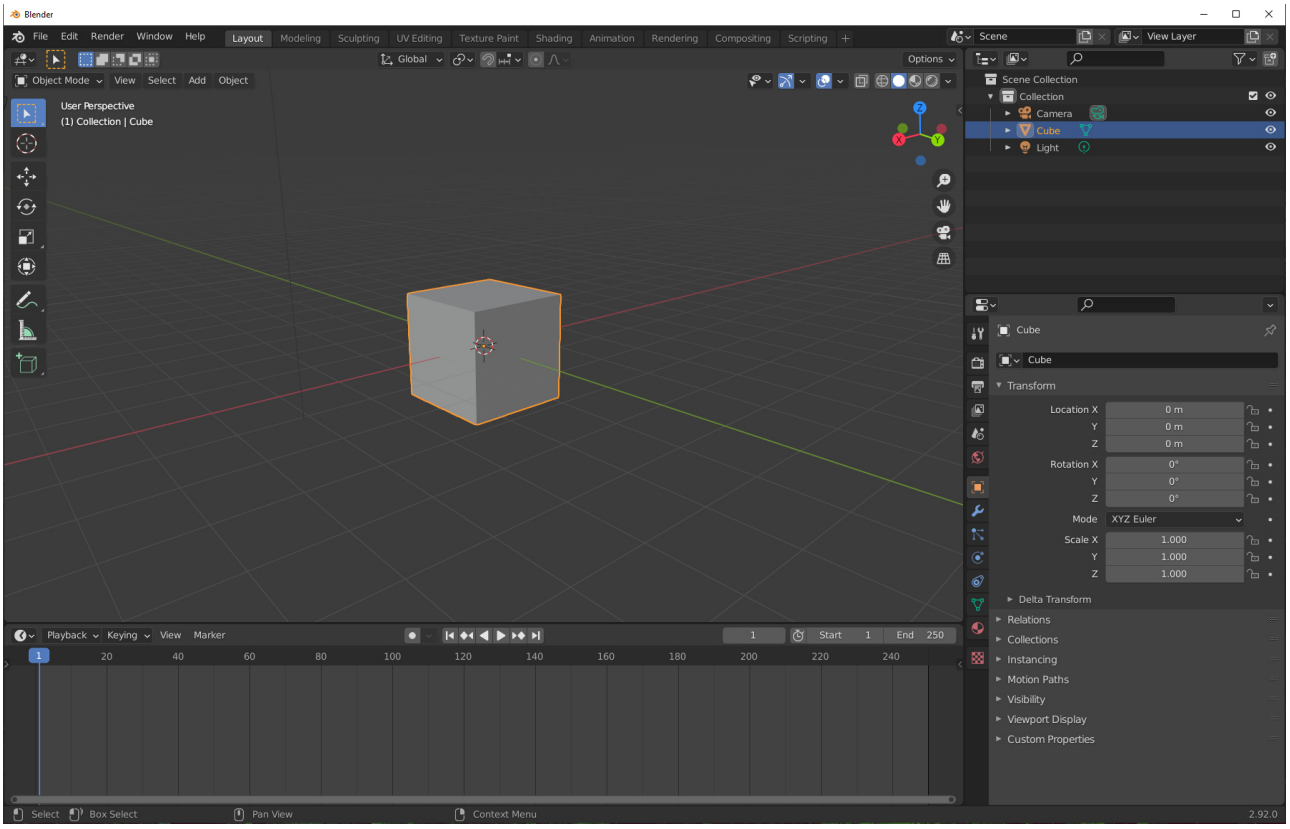


Figura 67. Interfaz por defecto.

En cada área del programa se encuentran botones característicos del panel seleccionado. En la figura anterior aparecen 4 paneles distintos. 3D ViewPort, OutLiner, Propiedades del Objeto y TimeLine. Cada uno de estos paneles puede ser cambiado utilizando un botón que siempre está a la izquierda en el panel.

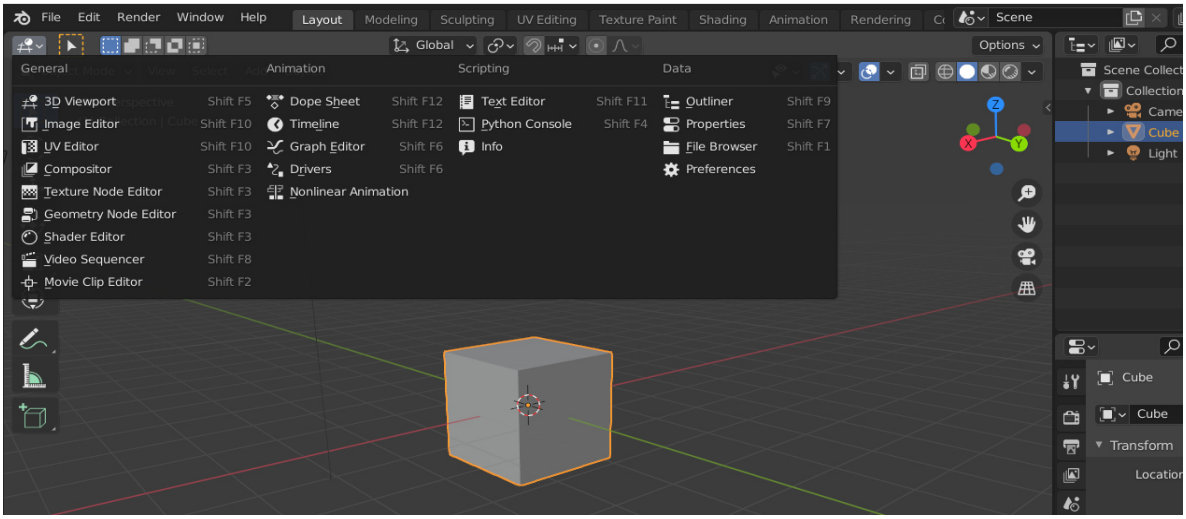


Figura 68. Pestaña que nos permite elegir el tipo de editor que se usa en cada panel o área.

Para dividir los paneles y crear tantos como se quiera, podemos pulsar encima de la línea que divide dos paneles y seleccionar la opción que nos interese. Además si se quiere duplicar un panel y crear una ventana independiente basta con acercar el cursor a una de las esquinas del propio panel y pulsar shift + click izquierdo.

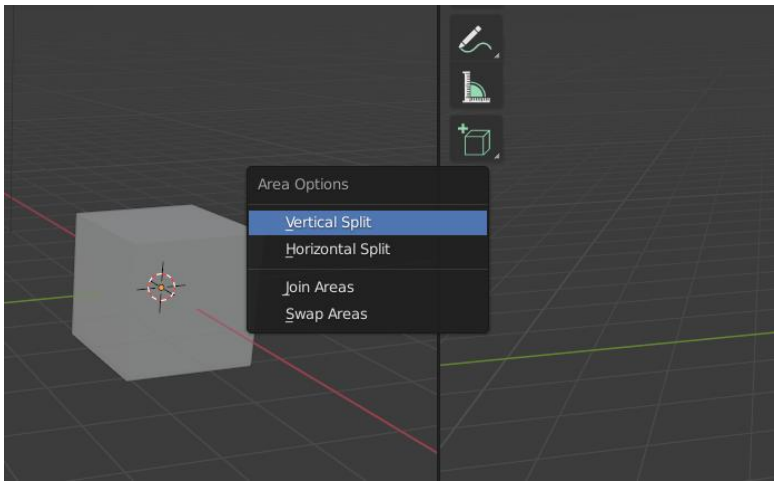


Figura 69. Opción para dividir, juntar o intercambiar los paneles.

En la parte superior central, tenemos la opción de elegir el workspace más adecuado para la tarea que estemos desarrollando. Si estamos realizando la escultura de un objeto, lo más conveniente es seleccionar "Sculpting", de esta forma el programa se cambiará automáticamente a este modo. También se tiene la posibilidad de agregar más workspaces para diferentes tareas.

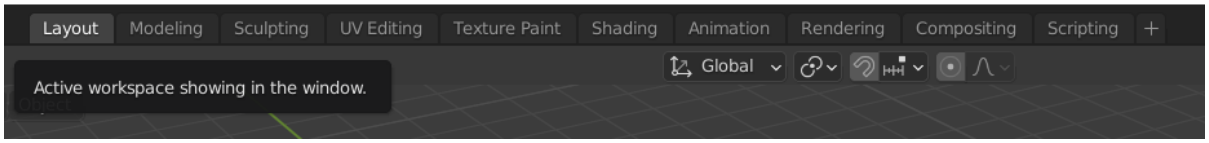


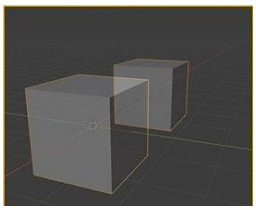
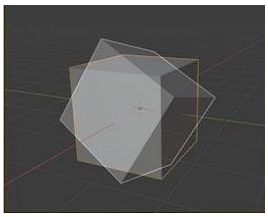
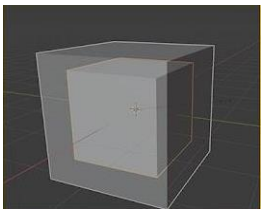
Figura 70. WorkSpaces.

Atajos de teclado

Blender es un programa que recae mucho en atajos de teclado, esto es algo bueno si ya se tiene una cierta práctica, pero la curva de aprendizaje es alta hasta que se controla con soltura. La buena noticia es que todas las opciones están disponibles a través del menú.

Tabla 7: Atajos de teclado principales en Blender

<table>
<tr><td colspan="4">Navegación</td></tr>
<tr>
<td> 
</td>
<td> 
</td>
<td> 
</td>
<td>VISTA DEL OBJETO
 
</td>
</tr>
<tr><td colspan="4">Ejes de transformación</td></tr>
<tr>
<td> 

X Y Z</td>
<td> 
MOVER
G</td>
<td> 
ROTAR
R</td>
<td> 
ESCALAR
S</td>
</tr>
<tr><td colspan="2">Generales</td><td colspan="2">Edite Mode</td></tr>
<tr>
<td colspan="2">A = SELECCIONAR TODO
AA = SELECCIONAR
SHIFT + A = AÑADIR
SHIFT + D = DIPLICAR
SHIFT + S CURSOR 3D
CTRL + T = COORDENADAS EN SHADING
CTRL + SPACE = PANTALLA COMPLETA</td>
<td colspan="2">I = INSERT
E = EXTRUIR
K = CORTAR CON EL CUCHILLO
CTRL + R = DIVIDIR
CTRL + B = BEVEL
U = PROYECTAR UVS
F = CREAR CARA
1 2 Y 3 = VÉRTICE, ARISTA Y CARA</td>
</tr>
</table>

Mover, Rotar y Escalar

Para realizar estas operaciones básicas, situadas en el panel de vista 3D, se utilizan las letras G R S para mover, rotar y escalar respectivamente. También se puede utilizar un menú que aparece a la izquierda.

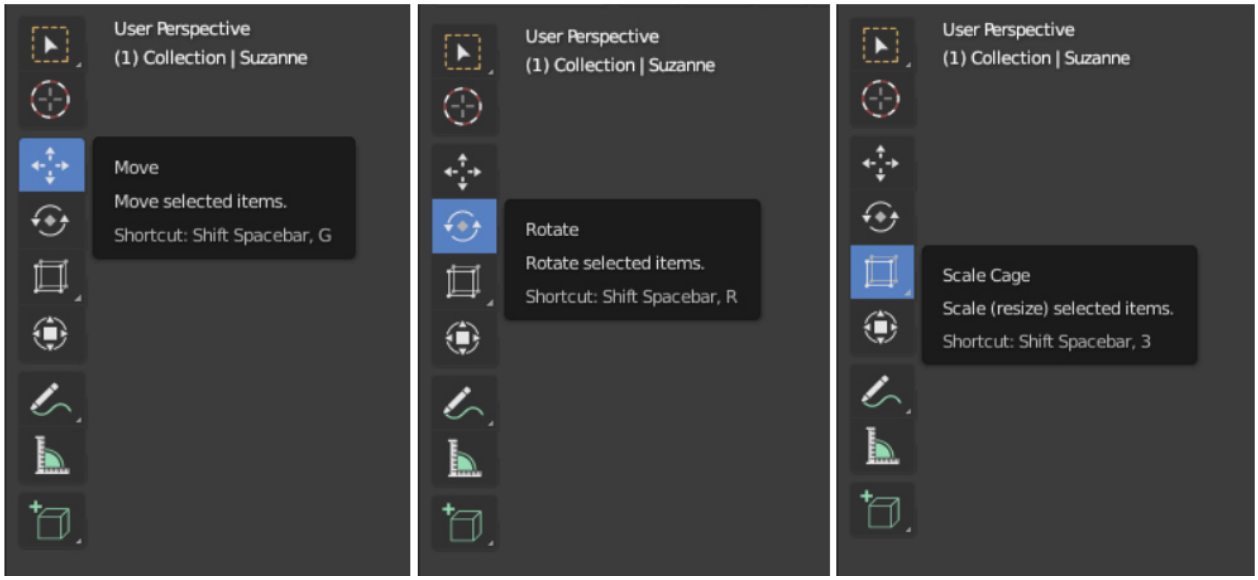


Figura 71. Opciones para mover, rotar y escalar un objeto.

Si mantenemos pulsado el click izquierdo encima de algunas de estas opciones podemos modificar la forma en que se muestra el modificar, pudiendo adaptarlo según preferencias.

Cursor 3D

El Cursor 3D es algo que llama la atención, sobre todo si viene de otros software, puesto que es algo diferenciador de Blender. Su posición está por defecto en el origen, y cualquier objeto nuevo que se cree tendrá su centro en esa posición. Para modificar la posición del cursor se pulsa la tecla n en el panel de vista 3D, y se va al apartado "3D Cursor". El menú del cursor aparece con las teclas shift + s, y permite realizar cambios como mover el un el objeto seleccionado a la posición del cursor.

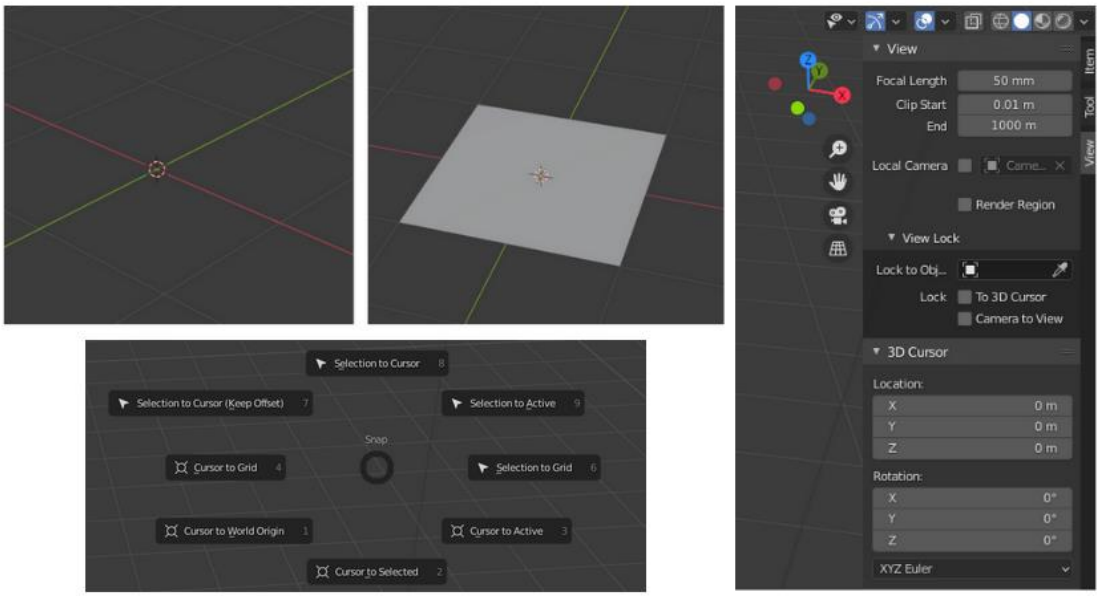


Figura 72. Cursor 3D y sus opciones.

Una de la opciones que nos da es la posibilidad de utilizar el Cursor 3D como punto de pivote. Por defecto Blender utiliza al punto medio entre uno o varios objetos para realizar las rotaciones, pero si activamos el Cursor 3D, las rotaciones pasarían a ser con respecto a este.

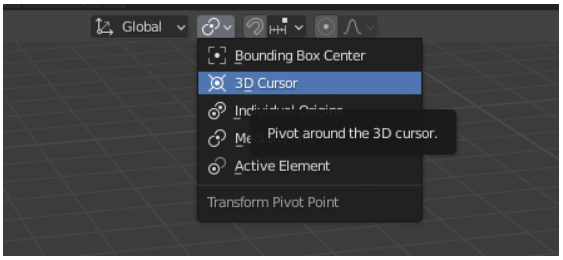


Figura 73. Cursor 3D como punto de pivote.

Modos de Interacción

Blender proporciona diferentes maneras de modificar los objetos de la escena (como modelar, texturizar, esculpir y posar), llamados Modos de Interacción. Por defecto, cuando se trabaja en el Modo Objeto, se puede mover, rotar y escalar; el Modo Objeto permite esencialmente colocar objetos en una escena. Probablemente uno de los modos más útiles es el Modo Edición, que se utiliza para modelar una malla, acceder a sus vértices, aristas y caras, y cambiar su forma.

Para seleccionar el modo de Interacción está el menú del Modo de Interacción en la cabecera de la Vista 3D, las opciones que muestra dependen del tipo de objeto que se haya seleccionado.

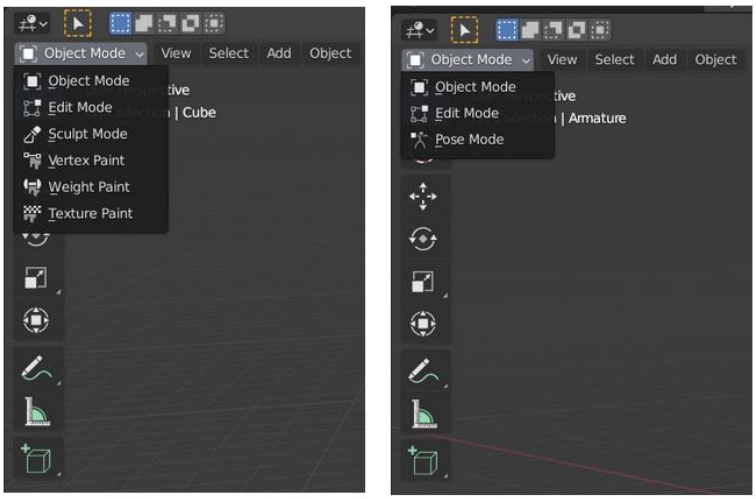


Figura 74. Modo de interacción. A la izquierda las opciones disponibles cuando se selecciona un objeto de malla; a la derecha las disponibles cuando se selecciona una armadura.

Se utiliza el Modo Objeto para crear y colocar cosas en la escena (incluso animarlas si se utilizan armaduras, que son esqueletos utilizados para animar personajes y deformar objetos). En el Modo Edición, se pueden realizar tareas de modelado en la malla, así como cambiar rápidamente entre estos modos sin tener que acceder al selector pulsando la tecla Tab del teclado.

Cuando se selecciona un “armature”, se utiliza el modo de edición para acceder a los huesos dentro de él y manipularlos, creando el estado “idle” o la pose de reposo. El modo Pose también está disponible y es el modo que se usará para crear la animación del esqueleto.

Modo de edición

Como se menciona anteriormente, este modo permite editar la malla de un objeto. Una malla es la parte exterior que recubre a un objeto y es lo que se renderiza. Los elementos que forman una malla son vértices, aristas y caras, en el modo edición se pueden modificar estos componentes individualmente o a la vez.

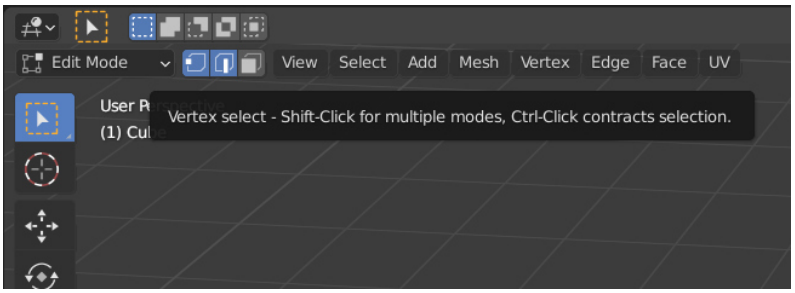


Figura 75. Selector de vértices, aristas y caras de un objeto.

Este modo cuenta con muchas herramientas que modifican la forma del objeto, cada icono tiene dentro más opciones si mantenemos pulsado el click izquierdo encima de él:

- Extruir una región: duplicar una cara creando nuevas aristas entre la cara inicial y la creada.
- Insertar una cara dentro de la cara seleccionada.
- Bevel, crea un perfil entre dos aristas seleccionadas.
- Loop cut, una cadena de artistas que están entrelazadas consecutivamente en la misma dirección, alt + click para seleccionar todo el loop.
- Knife, crea cortes en las aristas
- Spin, revolución de una selección de artistas.

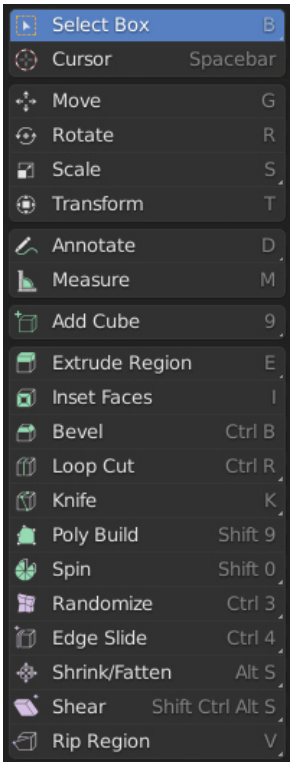


Figura 76. Shift + space.

Modos y Opciones de Visualización

En el proceso de creación de un objeto, puede que llegue un punto en el que la complejidad del mismo sea muy alta y sea difícil de ver sus partes internas. Por esto existen los modos de visualización que permiten ver el interior de una pieza o destacar ciertas zonas de otras.

Su ubicación es en la parte superior izquierda del panel de vista 3d, además cada modo cuenta con opciones para poder modificar cómo se visualiza el objeto.

- Wireframe: muestra solo las aristas y los vértices del objeto
- Solid: muestra el objeto sin texturas ni materiales.
- Material Preview: modo de previsualización alternativo al render, nos permite seleccionar distintos HDRi
- Rendered: modo de renderizado.

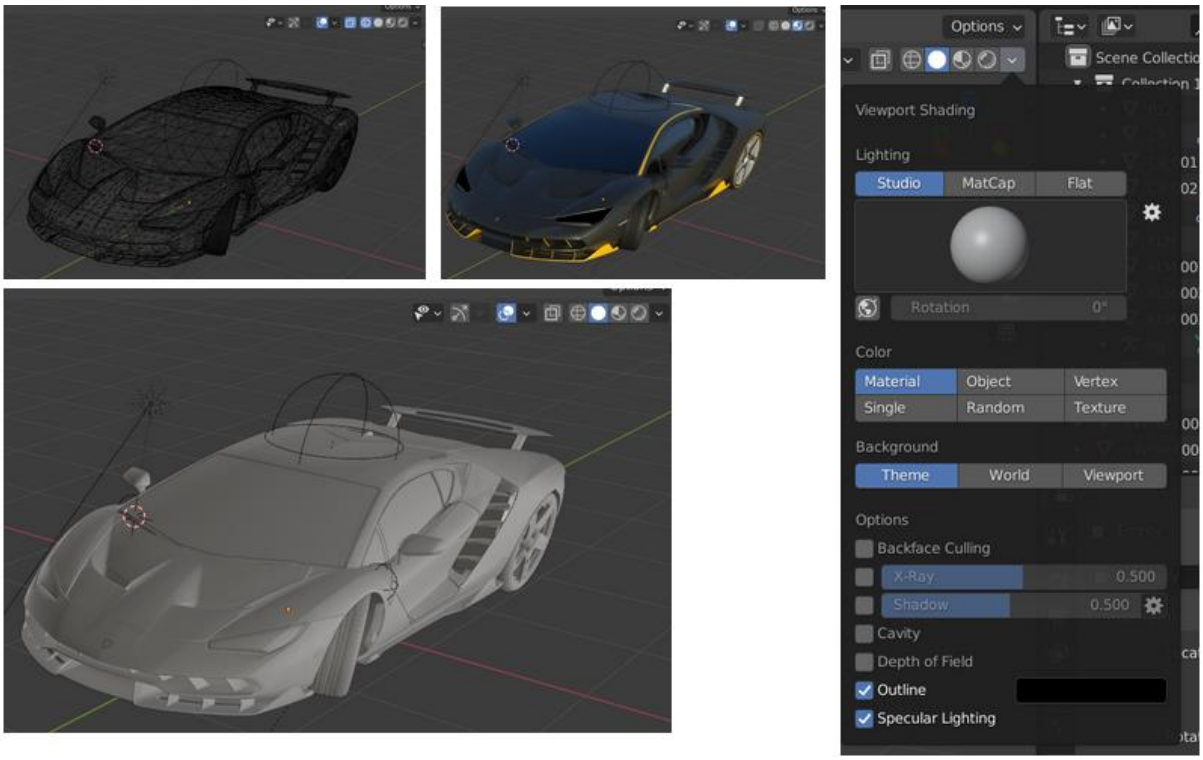


Figura 77. Modos de visualización.

Modificadores

Los modificadores son operaciones automáticas que afectan a la geometría de un objeto de forma no destructiva. Con los modificadores, se pueden realizar automáticamente muchos efectos que de otro modo serían demasiado tediosos de hacer manualmente (como las superficies de subdivisión) y sin afectar a la geometría base del objeto.

Funcionan cambiando la forma en que se muestra y renderiza un objeto, pero no la geometría, que se puede editar directamente. Puedes añadir varios modificadores a un solo objeto para formar la pila de modificadores y aplicar un modificador si quieres que sus cambios sean permanentes.

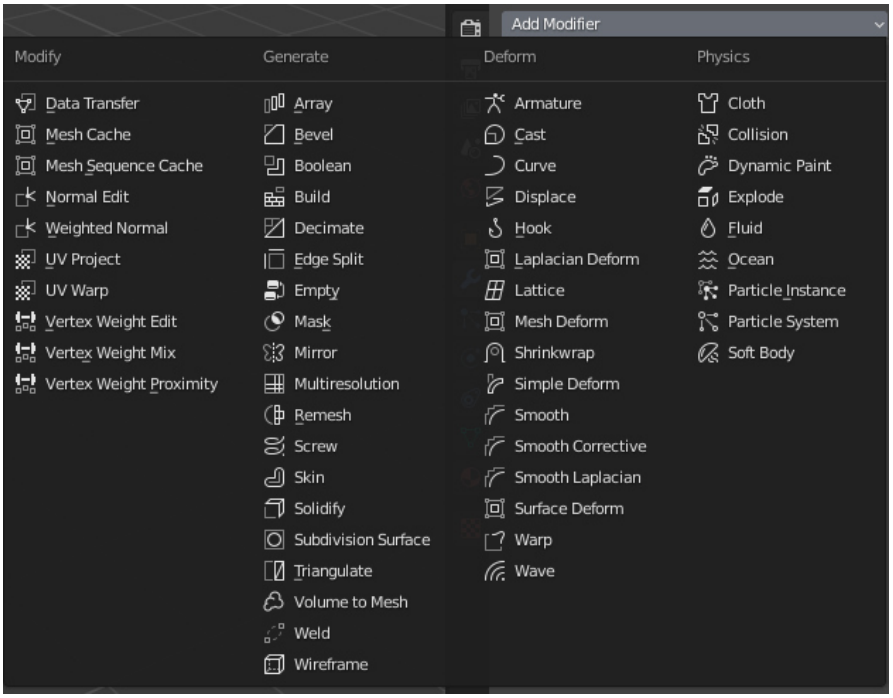


Figura 78. Modificadores.

Pueden añadirse al objeto activo mediante el menú desplegable “Add Modifier”, situado en la parte superior de la ficha de propiedades. Los nuevos modificadores se añaden siempre en la parte inferior de la pila (es decir, se aplicarán en último lugar).

Hay cuatro tipos de modificadores:

- Modificar: Son herramientas similares a las de deformación, sin embargo, no suelen afectar directamente a la geometría del objeto, sino a algún otro dato, como los grupos de vértices.
- Generar: Son herramientas constructivas/destructivas que afectan a toda la topología de la malla. Pueden cambiar la apariencia general del objeto, o añadirle nueva geometría...
- Deformar: A diferencia de las anteriores, éstas sólo cambian la forma de un objeto, sin alterar su topología.
- Simular: Representan simulaciones físicas. En la mayoría de los casos, se añaden automáticamente a la pila de modificadores cada vez que se activa un sistema de partículas o una simulación de física. Su única función es definir la posición en la pila de modificadores de la que se toman los datos base para la simulación que representan. Como tales, no suelen tener atributos, y se controlan mediante ajustes expuestos en secciones separadas de las Propiedades.

Jerarquías y Colecciones

Las colecciones son una forma muy sencilla de agrupar y organizar los elementos de la escena. Esto es particularmente útil si se tienen muchos elementos en la escena. Por ejemplo, se puede tener un grupo de objetos que conforman los elementos de fondo de la escena y otro objeto o colección de objetos en primer plano. En este caso se podría renderizar las capas de elementos por separado y si por ejemplo se necesita volver a renderizar los elementos del primer plano se podrían hacer un renderizado de esa capa solamente sin tener que volver a renderizar los elementos de fondo.

Para crear una colección, hacer clic con el botón derecho del ratón en el contorno y seleccionar nuevo. Arrastrar los elementos dentro y fuera de las colecciones.

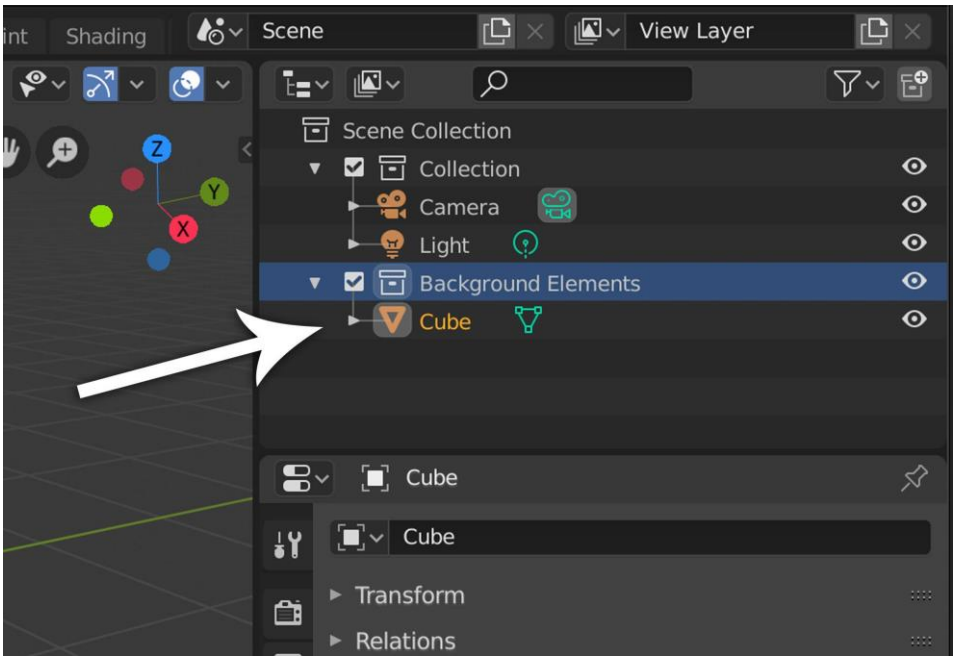


Figura 79. Jerarquías y colección en Blender.

También se puede editar las propiedades de una colección seleccionando primero la colección y luego haciendo clic en la pestaña de propiedades.

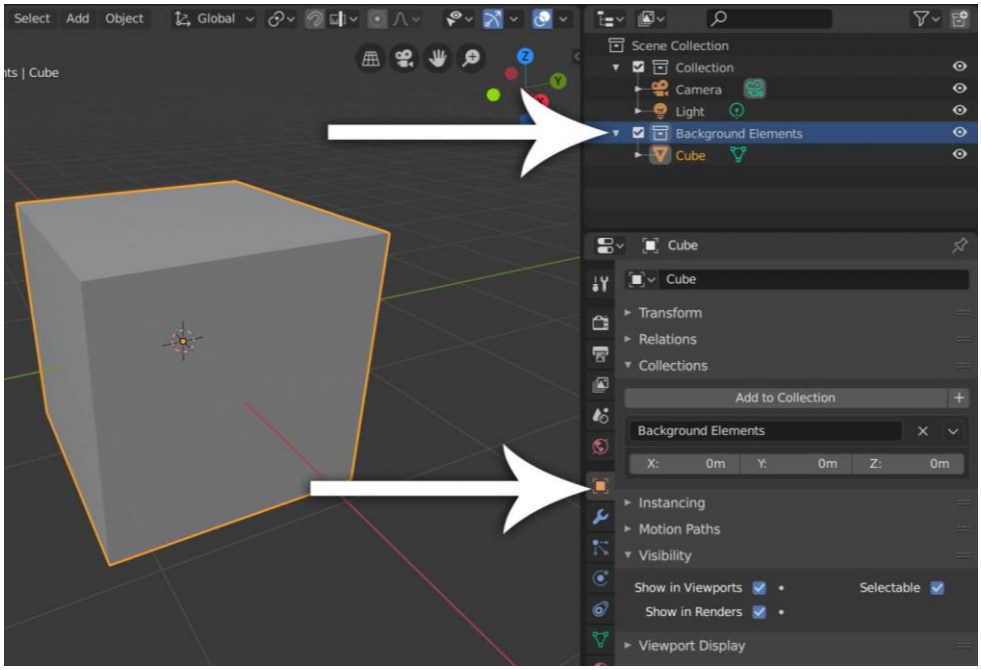


Figura 80. Propiedades de las colecciones en Blender.

Motores de Render: EEVEE y Cycles

El renderizado es el proceso de generar imagen foto realista, a partir de un modelo 2D o 3D. En blender, tenemos los motores de render, que toman la información de una escena digital y la sintetizan en una imagen 2D. Se encargan de interpretar y procesar los elementos de dicha escena, como las geometrías, texturas, fuentes de luz y shaders para ser exportadas en una imagen o secuencia de imágenes.

Principalmente se trabaja con dos motores. EEVEE y Cycles, se utilizan a la vez muchas veces, puesto que tienen ciertas diferencias.

EEVEE

Es el motor de render en tiempo real de Blender, construido bajo la tecnología OpenGL su objetivo es ofrecer velocidad e interactividad al renderizar materiales. Se basa en "trucos" y aproximaciones, haciendo el proceso de render muy rápido y en tiempo real. En la mayoría de los casos se suele utilizar durante el proceso de creación puesto que ahorra mucho tiempo de espera.

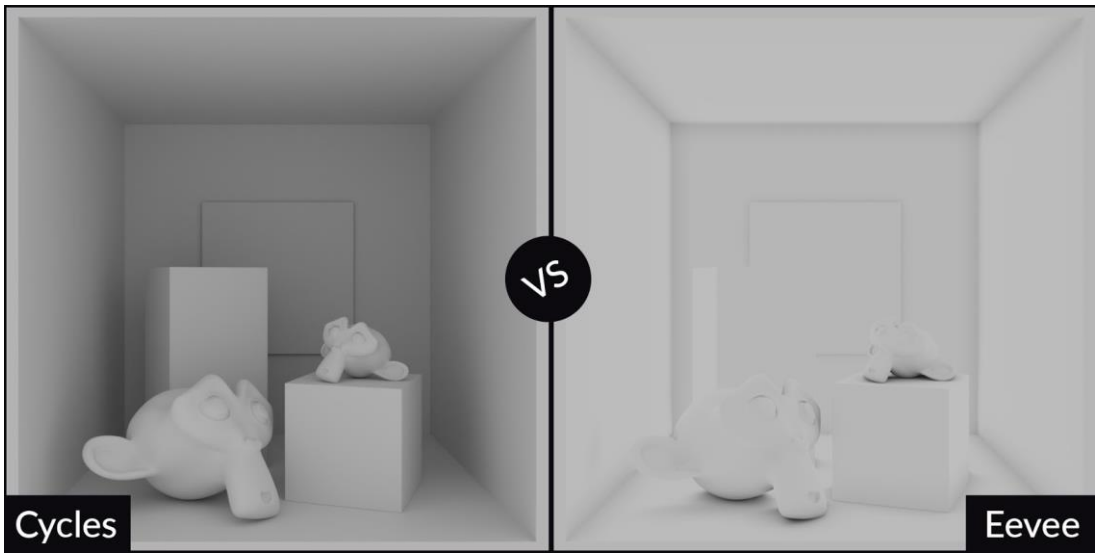


Figura 81. Comparativa entre Cycles y EEVEE.

Cycles

Cycles funciona lanzando rayos de luz desde cada píxel de la cámara hacia la escena. Se reflejan, refractan o son absorbidos por los objetos hasta que chocan con una fuente de luz o alcanzan su límite de rebote. Seguir un solo rayo de luz desde un píxel no es tan preciso, ya que muchos objetos o detalles de textura pueden ser más pequeños que un píxel, por lo que Cycles dispara rayos aleatorios adicionales desde ese píxel y promedia el resultado a lo largo del tiempo. Este enfoque de fuerza bruta de promediar muestras aleatorias se denomina simulación de Monte-Carlo, y cuando se aplica a las trayectorias de luz se denomina trazado de trayectorias.

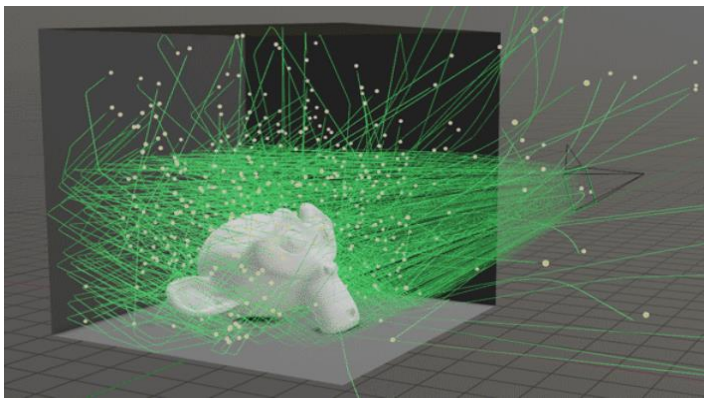


Figura 82. Rayo lanzados desde la cámara cuando se utiliza Cycles.

Cabe destacar que cuando se está en el visor 3D, en la parte superior derecha tenemos 4 modos de visualización  , y en dependencia si usamos un motor de render u otro, los dos últimos modos serán distintos. Por tanto si usamos EEVEE en el modo de visualización de Material Preview y Rendered se utilizara EEVEE, en cambio sí usamos Cycles, en el modo Material Preview se usará EEVEE pero en el modo Rendered se usará Cycles. Esto es debido a los tiempos de renderizado de cada motor, de esta forma se agiliza el proceso de modelado.

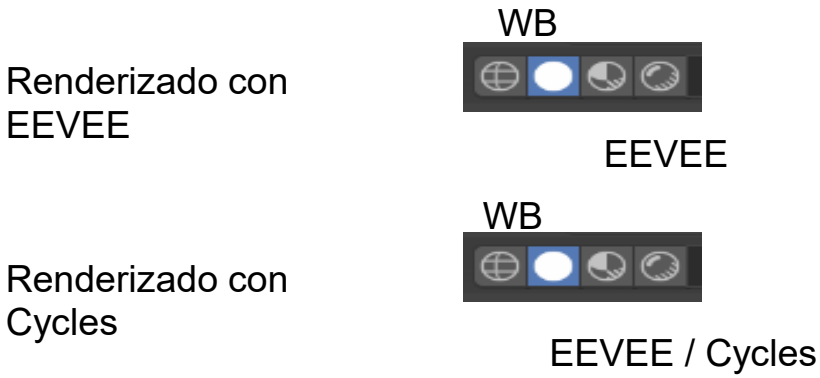


Figura 83. Diferencias entre usar un motor u otro.

Para cambiar el motor de render en Blender se puede hacer en el panel de propiedades en la pestaña de render.

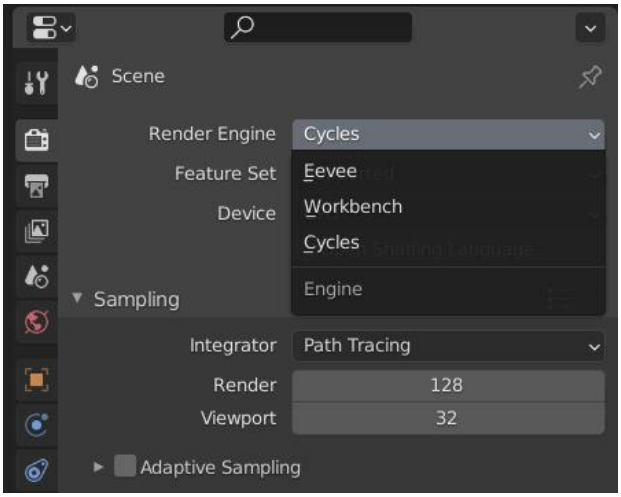


Figura 84. Motores de render.

Shading

Este es el proceso mediante el cual le asignamos un material a un objeto, dándole color, reflexión y rugosidad, entre otras propiedades. En el apartado surface podemos definir el color que tendrá el objeto.

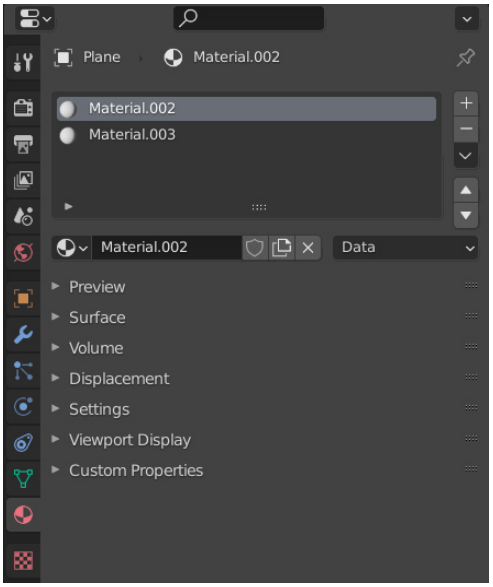


Figura 85. Shading.

Además de crear materiales como se acaba de describir utilizando todos los ajustes en todos los paneles de materiales, permite crear un material mediante el enrutamiento de materiales básicos a través de un conjunto de nodos. Cada nodo realiza alguna operación sobre el material, cambiando su apariencia cuando se aplica a la malla, y lo pasa al siguiente nodo. De este modo, se pueden conseguir apariencias de materiales muy complejos.

Cuando se crea un sistema de nodos, se está describiendo una especie de tubería de procesamiento de datos, en la que los datos "fluyen" desde los nodos inputs hacia los nodos que representan salidas o destinos. Los nodos se pueden conectar entre sí de muchas maneras diferentes y se pueden ajustar las "propiedades" o parámetros que controlan el comportamiento de cada nodo.

Este mapa de nodos se realiza en el panel Edición de Nodos. Cabe destacar que cada nodo tiene entradas y salidas, La idea es conectar cada entrada o salida con las de su mismo color, aunque hay algunas excepciones, las principales son:

Amarillas: RGB

Grisales: escala de grises o un valor numérico.

Azules: vectores, tres valores de X Y Z

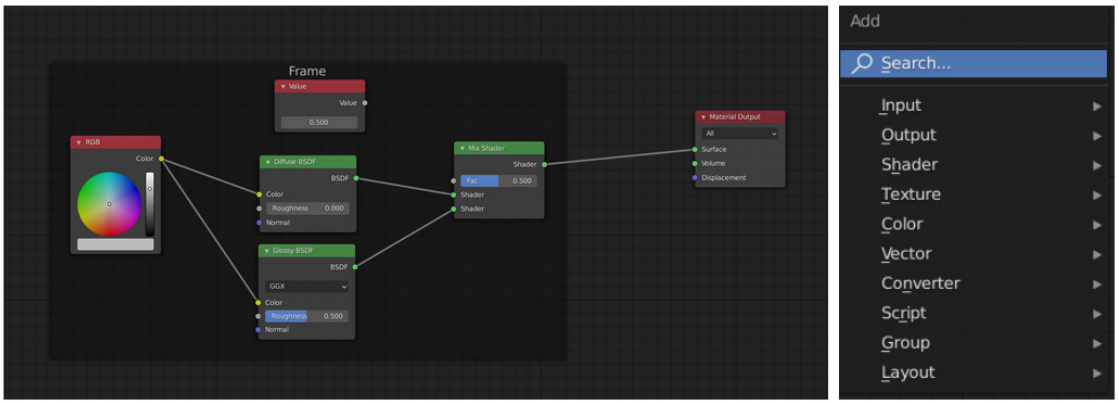


Figura 86. Shader editor, se ha añadido un frame para agrupar ciertos nodos.

Comunicación entre Blender y Unity

Este apartado es muy importante, sobre todo en este proyecto, puesto que la intención de usar Blender es hacer modelos 3D que luego se puedan llevar a Unity.

La comunicación entre Unity y Blender se puede realizar de diversas formas, pero la intención siempre debe ser buscar comodidad y rapidez, de tal forma que sea fácil modificar el modelo en Blender y ver los cambios automáticos en Unity.

En este aspecto se pueden definir dos caminos posibles:

1. Importar el archivo .blend directamente en el directorio del proyecto en Unity, es decir, guardar el modelo en un archivo .blend (el formato por defecto que utiliza Blender) y luego copiarlo directamente en el proyecto de Unity.
2. Exportar el archivo .fbx desde Blender, y luego importar el archivo en Unity.

La opción más cómoda es la primera, puesto que si guardamos el archivo .blend dentro de la carpeta del proyecto, Unity directamente lo reconocerá y podremos importarlo directamente a una escena, sin necesidad de cambiar el formato.

Anexo III: Gafas de VR

La primera patente sobre este concepto data de 1957, cuando Morton Heilig nos describió su Sensorama, un dispositivo de televisión estereoscópico montado en la cabeza. Desde entonces se ha evolucionado mucho, hasta el punto en el que ya se disponen de gafas que vienen con todo integrado, de tal forma que no es necesario conectarla a nada externo, solamente cargarlas antes de cada uso.

Ahora, veamos cómo se puede lograr este efecto a través de la visión estereoscópica. La forma normal que tienen los humanos y la mayoría de los animales para ver es la visión estereoscópica. La percepción de dos imágenes ligeramente diferentes (una de cada ojo) como una sola imagen, da como resultado la percepción de la profundidad, lo cual nos ayuda a ver el mundo en un glorioso 3D. Para recrear esto en las aplicaciones de VR, es necesario renderizar dos vistas ligeramente diferentes una al lado de la otra, que serán tomadas por los ojos izquierdo y derecho.

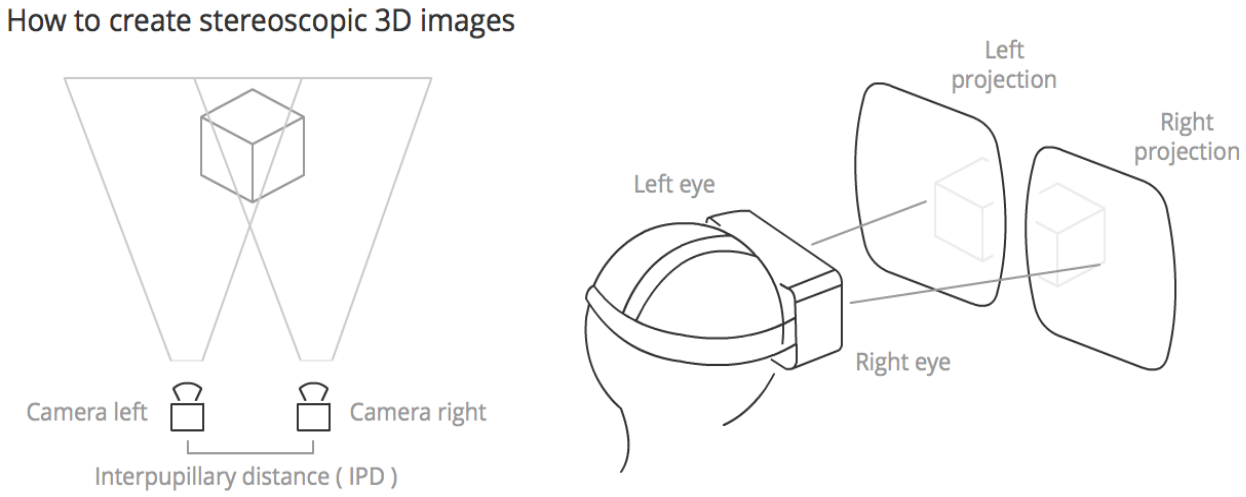


Figura 87. Visión estereoscópica.

Además, se utilizan lentes cóncavas que nos hacen parecer que la imagen está en el infinito, aunque esté a centímetros de nuestros ojos.

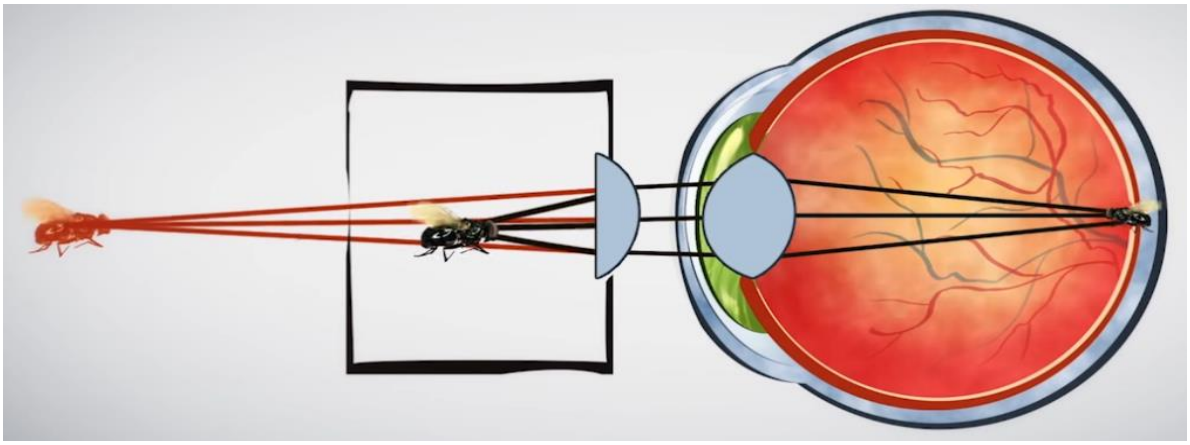


Figura 88. Cómo funcionan las lentes de las gafas VR.

Movilidad Potencia y Precio

En este proyecto en concreto no se pretende ofrecer una experiencia gráfica inigualable y artísticamente increíble. El objetivo es lograr imitar una serie de mecánicas y funcionalidades, por lo que el apartado gráfico solo cumple una función orientativa, lo suficientemente reconocible para que se "crea" lo que se está jugando. Por otro lado también es necesario que las gafas de VR sean ligeras y cómodas de usar, puesto que el usuario realiza movimientos bruscos con la cabeza y con las manos para interactuar con la aplicación.

Teniendo en cuenta los requisitos mencionados podemos llegar a la conclusión que sería ideal utilizar para este proyecto unas gafas de VR que sean un todo en 1, es decir, que no necesitan cables ni conexión a un ordenador para funcionar. Con esta característica en mente, en el mercado europeo actual, solo tenemos una opción viable para este proyecto, por precio y por disponibilidad. Hablamos de las Oculus Quest 2, son relativamente económicas además de ofrecer una relación peso potencia muy buena.

A continuación se muestran las características de las Oculus Quest 2 además de otras tres opciones que se tuvieron en cuenta para utilizar este proyecto.

Oculus Quest 2

Ligeras, cómodas y lo suficientemente potentes como para ejecutar aplicaciones de realidad virtual detalladas. Como sistema operativo tiene una versión modificada de Android, facilitando el desarrollo de aplicaciones para la plataforma. La relación potencia/movilidad es algo que hay que tener en cuenta con estas gafas, sobre todo si se desea no tener que depender de cables.

Figura 89. Oculus Quest 2.

Tabla 8: Especificaciones de las Oculus Quest 2

Peso	503 gramos
Tipo De Panel	LCD
Resolución	1.920 x 1.832 por ojo
Frecuencia De Refresco	90 Hz Capada a 72 Hz
Procesador	Snapdragon XR2
Conectividad Inalámbrica	WiFi 6 Bluetooth 5.1
Puertos	USB tipo C
Almacenamiento	64/128/256 GB
Tracking	Cámaras internas
Precio	64 GB: 349 euros

Oculus Rift S

Las Oculus Rift S son una versión con cables de las Oculus Quest 2, por lo que son dependientes de un ordenador para funcionar. Los controles son iguales a su hermano con Android. En cuanto a las características tienen una pantalla LCD que proporciona una resolución total de 2.560 x 1.440 píxeles y una tasa de refresco de 80 Hz, lo que provoca que el contenido se muestre con mayor fluidez. Su peso es ligeramente superior a los 470 gramos, pero no son incómodas de utilizar durante el tiempo que se lleven puestas.

Figura 90. Oculus Rift S.

Tabla 9: Especificaciones de las Oculus Rift S

Peso	470 gramos
Tipo De Panel	LCD
Resolución	2.560 x 1.440 píxeles en total
Frecuencia De Refresco	80 Hz
Requisitos mínimos	Intel Core i3-6100, 8 GB de memoria RAM, tarjeta gráfica NVIDIA GeForce GTX 1050 Ti y USB 3.0 Windows 10
Precio	64 GB: 349 euros

HTC Vive

Estas gafas de VR son ya todo un estándar en la industria. En su día fueron de las primeras opciones viables a nivel de usuario. Ofrecen unas características muy interesantes como su panel OLED con un nivel de contraste infinito característico de esa tecnología.

Figura 91. HTC Vive.

Tabla 10: Especificaciones de las HTC Vive

Peso	550 gramos
Tipo De Panel	OLED
Resolución	2.160 x 1.200 píxeles en total
Frecuencia De Refresco	90 Hz
Conexiones	HDMI, USB 2.0, USB 3.0
Requisitos mínimos	NVIDIA GeForce GTX 970 / Radeon R9 290 Intel Core i5-4590 o superior 4GB of RAM Salida de vídeo Displayport 1.2 o HDMI 1.4 1 x puerto USB 2.0
Precio	899 euros

Valve Index

Este modelo es relativamente nuevo en el mercado y tiene ciertas características muy interesantes y novedosas. Lo primero que llama la atención es su alta frecuencia de refresco de 120Hz y su panel LCD. Estas dos características justas han demostrado un nivel de rendimiento óptimo en aplicación con movimientos rápidos y bruscos, evitando así sensaciones de mareo de los usuarios.

Figura 92. Valve Index.

Tabla 11: Especificaciones de las Valve Index

Peso	470 gramos
Tipo De Panel	LCD
Resolución	pantallas duales de 1440 x 1600
Frecuencia De Refresco	120 Hz
Conexiones	USB 3.0, DisplayPort
Requisitos mínimos	Procesador de dos núcleos y cuatro hilos, tarjeta gráfica NVIDIA GTX 970 o AMD RX 480. Salida de vídeo Displayport 1.2 o HDMI 1.4
Precio	1.079 euros

Anexo IV: Baloncesto En Silla De Ruedas

El Baloncesto En Silla De Ruedas o comúnmente llamado BSR, es un deporte que vio sus inicios en la década del 1940 como parte del proceso de rehabilitación de los soldados que sufrían amputaciones de sus extremidades durante la guerra, y deseaban seguir jugando a baloncesto. Su primera aparición oficial fue en 1956 en la Copa Internacional de Stoke Mandeville-Games donde el equipo estadounidense "Pan Am Jets" ganó el torneo.

En nuestro territorio hace años que existe una liga organizada y con varios equipos que forman parte de ella. Uno de ellos es el Zuzenak, que reside en Vitoria-Gasteiz.

Figura 93. Equipo vitoriano de BSR Zuzenak, temporada 2016 - 2017.

Sistema de puntuaciones

El BSR, cuenta con un sistema de puntuación que lo diferencia mucho del baloncesto convencional. El objetivo de este sistema es brindar a cualquier jugador, independientemente de su grado de discapacidad, una oportunidad de jugar en este deporte.

Las puntuaciones que puede tener un jugador van desde el 1 al 4,5 aumentando en 0,5 cada vez, siendo el 1 la peor movilidad posible y el 4,5 la mayor movilidad pero con algún tipo de discapacidad, los llamados minimal handicap.

Veamos varios ejemplos de esto:

1. Lesión medular, es decir carece de total movilidad por debajo de la mitad del torso, puntuación de 1 o 1,5.
2. Amputación de una de las dos piernas por debajo de la rodilla, 4 o 4,5 en dependencia de si tiene más afectación en la cadera de la pierna amputada.
3. Alguna afectación grave en ambas piernas con dificultades para andar, puntuación 2,5 o 3, en dependencia si además existen problemas de movilidad en el torso.

Figura 94. Saque inicial en un partido de Zuzenak vs Málaga.

Como se puede apreciar, hay muchas posibilidades en dependencia de la lesión que tenga el jugador, puesto que hay muchos tipos de discapacidades y a cada jugador le afecta de una forma diferente, por esta razón la elección de la puntuación se suele realizar según la movilidad que se tenga en la silla de ruedas deportiva.

Este sistema de puntuaciones es una de las características más diferenciadoras con respecto al baloncesto convencional, ahora bien, ¿para que se utilizan las puntuaciones?

En un quinteto de juego real y en la liga española de BSR, la suma de las puntuaciones de todos sus jugadores no puede exceder los 14,5 puntos. De esta forma, un quinteto de partido podría ser el siguiente, 1 + 2 + 3 + 4 + 4,5 = 14,5, sería lo comúnmente llamado, jugar con dos puntos altos y un intermedio, puesto que se sobreentiende que los otros dos jugadores serán puntuaciones bajas.

Las sillas en el BSR y su evolución.

Con el tiempo las sillas de ruedas utilizadas para prácticas deportivas se han diferenciado mucho de las utilizadas para la calle.

Diseño

La principal diferencia que se introdujo es el "camber" o el ángulo de caída, esto permite que las ruedas tengan una inclinación, habitualmente de unos 18 grados, que ayuda a mejorar la estabilidad del jugador en la silla.

Figura 95. Silla de la marca RGK modelo Eilte X.

Como es de esperar, los jugadores tienen que ir bien atados a las sillas, en cada movimiento que se realice la silla debe acompañar y responder en consecuencia.

Variación de las sillas según el jugador

Como bien se ha descrito anteriormente, cada jugador tiene una puntuación según su movilidad y su lesión, y como es de esperar, las sillas de ruedas no serán la excepción.

La sillas de ruedas del BSR están perfectamente adaptadas a cada jugar, puesto que la silla que utiliza un "4" es por lo general más alta y más inestable que la que utiliza un "1"

Una cosa curiosa a destacar, es que los jugadores con una puntuación mayor al 3,5 inclusive, no podrán ir sentados a una altura mayor de 58 cm desde el suelo, y los jugadores por debajo de 3 inclusive, podrán ir a 63 cm. El objetivo de esta norma es no darle más ventaja a los jugadores con puntuaciones altas de la que ya tienen por su movilidad, de esta forma se limita la altura máxima a la que pueden ir sentados y se abre una posibilidad para aquellos jugadores con puntuaciones menores de 3.0

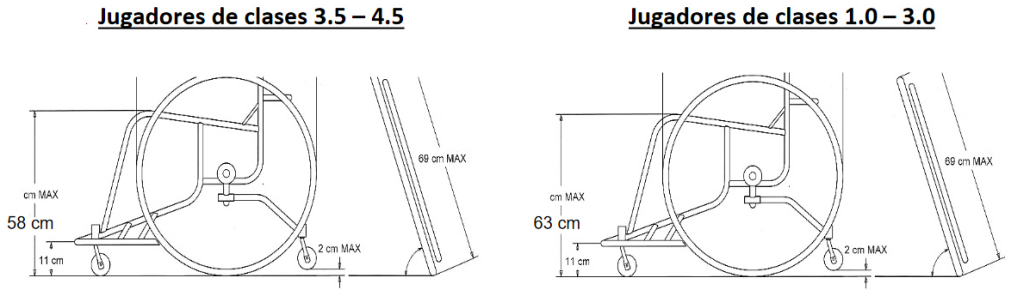


Figura 96. Variación de la silla según la puntuación.

Reglas y Normas

Una de las reglas más características de este deporte es lo que en baloncesto convencional se llaman pasos. Un jugador cuenta con tres impulsos para darle a la silla antes de tener la obligación de botar el balón, de no ser así, esto constituye falta y es posesión de balón para el otro equipo.

Cabe recalcar que se considera impulsó a cualquier acción realizada a través de las manos que sirva para mover la silla de ruedas.

Medidas del campo

La demarcación de la cancha se realiza mediante líneas blancas o un color que contraste la superficie con un grosor de 5cm. La medida oficial según FIBA para la cancha de baloncesto es de 28 metros de largo por 15 metros de ancho.

Las principales demarcaciones dentro de la cancha son:

- Círculo central de 3.6m de diámetro
- Línea de tiro libre a 5.8m de la línea de fondo y a 4.6m de la canasta.
- Línea de 3 puntos a 6.75m de la canasta

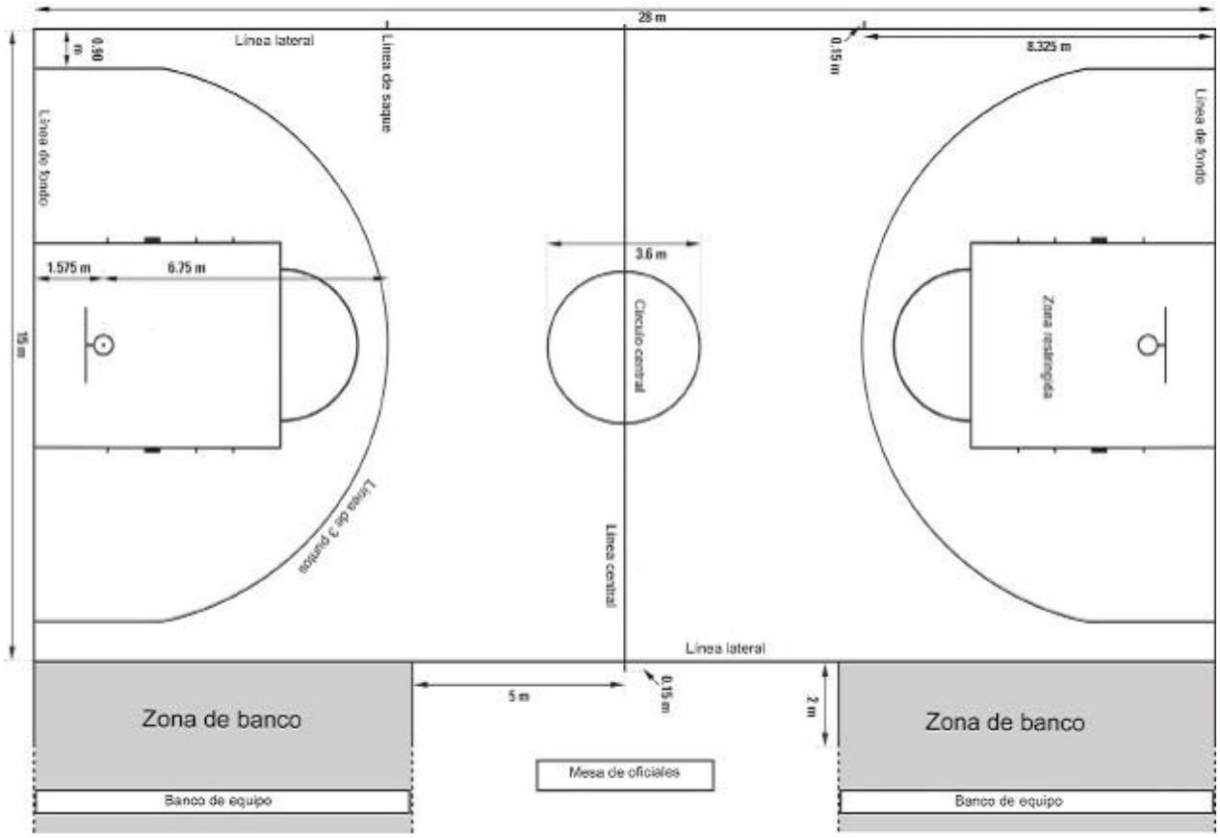


Figura 97. Medidas del campo.

Las zonas en el terreno de juego tienen las siguientes medidas:

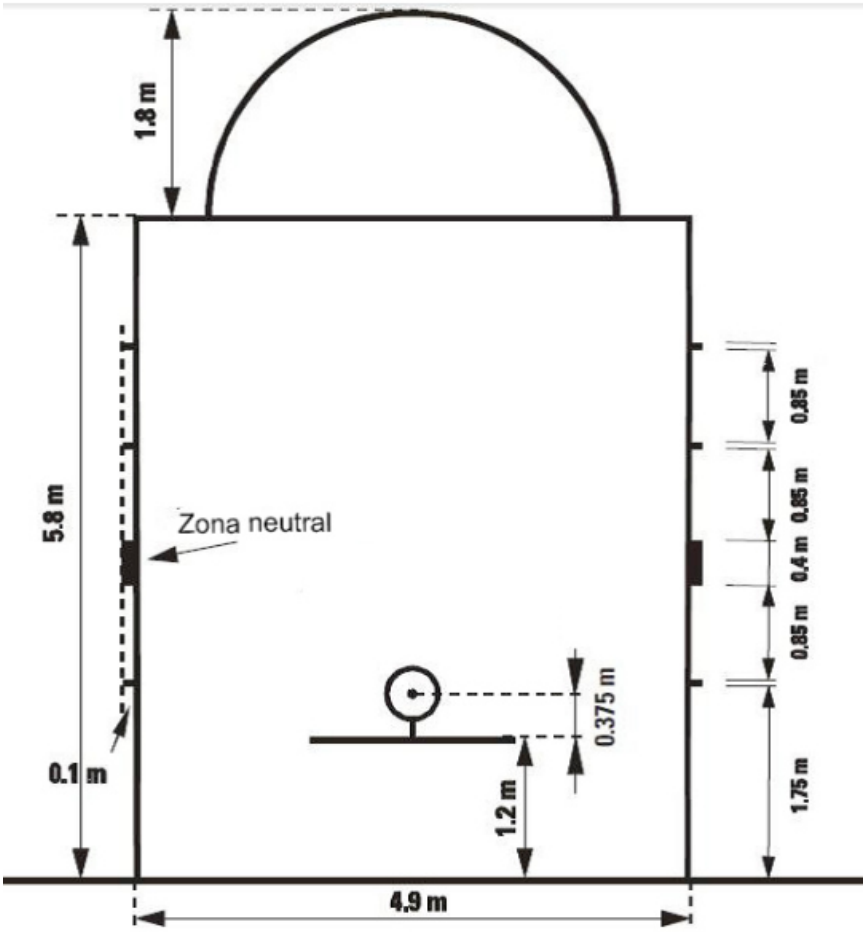


Figura 98. Medidas de la zona.