

DIV - Desarrollo de videojuego para personas con diversidad funcional



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Jessica Hernández Gómez

Tutor:

Carlos J. Villagrá Arnedo

Julio 2022

JUSTIFICACIÓN

El mundo de los videojuegos está en constante expansión y actualmente se considera una forma de ocio más, dejando de ser algo trivial que sólo entendían algunos. Hoy en día podemos encontrar videojuegos de todas clases y desarrollados para todas las edades, haciéndolos aún más accesibles para todos excepto para las personas con diversidad funcional. Adaptarlos para ellos es un tema que requiere tiempo y esfuerzo y no todas las compañías están dispuestas a invertir en ello, esto es otro motivo más por el cual están siendo discriminados por la sociedad. Por este motivo se pretende desarrollar en Unity un único videojuego compuesto por 2 minijuegos para dispositivos móviles donde se pueda jugar sean cuales sean las características del usuario en cuestión.

Se revisará cuál es la situación actual a la que se enfrentan las personas con diversidad funcional, qué referentes se encuentran y cómo se pueden adaptar los videojuegos sin necesidad de perder el objetivo en dicha adaptación y que finalmente el juego sea completamente diferente tras su adaptación. Posteriormente, toda la información recabada será utilizada para desarrollar un videojuego accesible.

El objetivo de dicho videojuego es que independientemente de las características del usuario, los minijuegos estén lo suficientemente adaptados para que sea algo completamente indiferente y se pueda llevar a cabo una partida sin problema alguno.

AGRADECIMIENTOS

Entregar este documento supone finalizar una etapa, una etapa donde he crecido personal y profesionalmente. Cuando comencé esta aventura no pensaba que viviría tantas experiencias, he conocido a personas increíbles y aunque la vida nos lleve a cada uno por caminos diferentes, tendrán un hueco en mi corazón siempre. Quiero agradecer en este documento a mis compañerxs de carrera Charo, Bea, Naru y Juanru, por haber sido un gran pilar en esta etapa, cuando he querido abandonar han estado ahí para recordarme que era capaz de conseguir mis objetivos y de que no estaba sola.

Esta etapa no puede llevarse a cabo sin la figura del profesorado, sin duda me llevo la lección de que hay que ponerle empeño a lo que se hace para aprender. Se nota muchísimo cuando un profesor está realmente implicado y le gusta la docencia y cuando no. Me gustaría agradecer a Alicia Garrido y a Luisa Micó el hecho ir más allá del papel de profesorado preocupándose realmente por sus alumnos. Siempre voy a recordar el apoyo y la preocupación recibida por su parte cuando tuve un accidente; haciendo hincapié en que la salud es lo primero. No puedo finalizar este apartado sin agradecer también a Carlos Villagrá por el magnífico trabajo realizado como tutor. Gracias Carlos por ayudarme a darle forma a este documento y al juego, aportándome otras perspectivas y haciéndome llegar así a nuevas ideas y nuevos enfoques. Gracias también por comprender mi situación laboral y ajustar todas las tutorías sin problema alguno y por todo el feedback recibido.

Y por último, pero no por ello menos importante, a quien más le debo agradecer esta etapa es sin duda a mi madre Raquel. Gracias mamá por todo lo que has hecho siempre por mí, por ser madre y padre. Por pasarte madrugadas en vela para estar pendiente de que no me ocurría nada cuando era un bebé. Por haber creído siempre en mi capacidad para todo y no haberme aislado cuando era una niña. Por apoyarme siempre con los estudios, aunque eso haya supuesto un esfuerzo económico el cual hemos superado con dificultades. Gracias por matarte a trabajar y por todos los sacrificios que has hecho para poder pagar todos los gastos que esta carrera ha conllevado. Gracias por enseñarme que las lágrimas desinfectan las heridas del corazón y que soy capaz de todo.

DEDICATORIA

Este trabajo está dedicado a Pepi, la educadora de Educación Especial que dio apoyo al alumnado con necesidades educativas especiales durante mi etapa en el instituto Las Fuentes de Villena. Gracias por compartir miles de risas y de vivencias con todos nosotros. Pese a que has abandonado este mundo de manera inesperada, espero que allá donde estés puedas estar orgullosa de que tus *niños* están consiguiendo todo lo que se proponen. Gracias por enseñarnos que no somos menos que nadie, pese a las características de cada uno. Gracias por creer en nosotros, gracias por hacer todo lo posible siempre para adaptar todas las actividades realizadas para que nuestras experiencias y oportunidades fuesen las mismas que las del resto. Este proyecto nace sin duda de ese ideal que has dejado en nosotros.

Y sin duda está dedicado a mi pareja Clau, quien desde el minuto 0 ha creído en mí. Quien me ha soportado y acompañado en mis desesperaciones durante el ABP, quien ha madrugado para llevarme a las prácticas sin importarle hacerlo y sobre todo quien me ayudó a decidirme sobre este tema de TFG sobre otros más convencionales. Quien me ha enseñado que hay que apostar por hacer lo que nace desde el corazón. Gracias por animarme cada vez que tenía que ponerme con este trabajo porque me bloqueaba y no sabía por dónde cogerlo. Gracias por crear la música de Gunkour, la de este proyecto y por supuesto, darle una melodía especial a mi vida. Y sobre todo, junto con los michis Stitch y Mozart, gracias por formar una pequeña familia conmigo. *Love me like you do.*

“No elegimos cómo empezamos en esta vida.

La verdadera grandeza es qué hacemos con lo que nos toca.”

Sully – Uncharted 3

ÍNDICE DE CONTENIDO

JUSTIFICACIÓN.....	1
AGRADECIMIENTOS	2
DEDICATORIA.....	3
ÍNDICE DE CONTENIDO	5
ÍNDICE DE ILUSTRACIONES	7
1. INTRODUCCIÓN	6
2. MARCO TEÓRICO	8
2.1. INTRODUCCIÓN	8
2.2. LA DIVERSIDAD FUNCIONAL EN DATOS.....	8
2.2.1. AFECTADOS EN RELACIÓN CON LA POBLACIÓN, POR COMUNIDADES AUTÓNOMAS	9
2.2.2. PERSONAS CON DIVERSIDAD FUNCIONAL POR SEXO.....	10
2.2.3. PERSONAS CON DIVERSIDAD FUNCIONAL POR EDAD.....	11
2.2.4. PERSONAS CON DIVERSIDAD FUNCIONAL POR GRADOS	12
2.3. CATEGORÍAS.....	13
2.4. ESTIGMAS SOCIALES	14
2.5. LAS TIC Y LA DIVERSIDAD FUNCIONAL.....	17
2.5.1. LAS TIC Y LA DIVERSIDAD FUNCIONAL FÍSICA	17
2.5.2. LAS TIC Y LA DIVERSIDAD FUNCIONAL SENSORIAL.....	18
2.5.3. LAS TIC Y LA DIVERSIDAD FUNCIONAL INTELECTUAL	18
2.5.4. LAS TIC Y LA DIVERSIDAD FUNCIONAL PSICOSOCIAL (MENTAL)	19
2.6. VIDEOJUEGOS ADAPTADOS	19
2.6.1. CELESTE	21
2.6.2. THE LAST OF US PARTE II	23
3. OBJETIVOS.....	26
4. METODOLOGÍA.....	27
4.1. METODOLOGÍA DE CONTROL Y GESTIÓN DEL PROYECTO.....	27
4.1.1. MICROSOFT PLANNER.....	27
4.2. METODOLOGÍA DE DESARROLLO DEL SOFTWARE.....	29
4.3. CONTROL DE VERSIONES	30
5. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO.....	32
5.1. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE DIV	32
5.1.1. FICHA TÉCNICA	32
5.1.2. CONCEPTO DEL VIDEOJUEGO	32
5.1.3. CONTROLES.....	33
5.1.4. DIAGRAMAS DE FLUJO	33
5.1.5. BOQUETOS DE PANTALLAS.....	34
5.1.6. ARTE FINAL	34
5.1.7. SONIDO	35
5.2. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE NINJA RUN	35
5.2.1. FICHA TÉCNICA	35
5.2.2. CONCEPTO DEL VIDEOJUEGO	35
5.2.3. MECÁNICAS	36

5.2.4.	CONTROLES.....	36
5.2.5.	DIAGRAMAS DE FLUJO.....	36
5.2.6.	BOCETOS DE PANTALLAS.....	38
5.2.7.	PERSONAJE PRINCIPAL	38
5.2.8.	PLATAFORMAS	39
5.2.9.	FONDO Y EFECTO PARALLAX.....	40
5.2.10.	INTERFAZ DE USUARIO (UI).....	43
5.2.11.	MÚSICA Y EFECTOS DE SONIDO.....	44
5.3.	DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE HIGHER & LOWER.....	44
5.3.1.	FICHA TÉCNICA	44
5.3.2.	CONCEPTO DEL VIDEOJUEGO.....	44
5.3.3.	MECÁNICAS	45
5.3.4.	CONTROLES.....	45
5.3.5.	DIAGRAMAS DE FLUJO	45
5.3.6.	BOCETOS DE PANTALLAS.....	46
5.3.7.	ARTE FINAL	46
5.3.8.	SONIDO	47
6.	DESARROLLO	48
6.1.	MOTOR DE DESARROLLO.....	48
6.2.	CARACTERÍSTICAS DE UNITY	49
6.3.	CONCEPTOS BÁSICOS DE UNITY.....	50
6.3.1.	ESCENA	50
6.3.2.	GAME OBJECTS.....	51
6.3.3.	COMPONENTES.....	52
6.4.	DESARROLLO DE DIV	54
6.4.1.	PROTOTIPO 1 – GESTIÓN DE ESCENAS DE DIV	54
6.4.2.	PROTOTIPO 2 – DESARROLLO DEL PRIMER PROTOTIPO DE NINJA RUN.....	56
6.4.3.	PROTOTIPO 3 – CAMBIO DE DISEÑO DE NINJA RUN, DESARROLLO DEL NUEVO PROTOTIPO	59
6.4.4.	PROTOTIPO 4 – AJUSTES, MEJORAS VISUALES Y SONORAS DE NINJA RUN	64
6.4.4.1.	AJUSTES DE NINJA RUN	64
6.4.4.2.	IMPLEMENTACIÓN DE INTERFAZ DE USUARIO	67
6.4.4.3.	IMPLEMENTACIÓN EFECTO PARALLAX SCROLLING.....	69
6.4.4.4.	INCLUSIÓN DE MÚSICA Y EFECTOS DE SONIDO	72
6.4.5.	PROTOTIPO 5 – PROTOTIPO DE HIGHER & LOWER	74
6.4.5.1.	GESTIÓN Y REPRODUCCIÓN DE SONIDOS E IMPLEMENTACIÓN DE LA VIBRACIÓN	74
6.4.5.2.	IMPLEMENTACIÓN DE LA INTERACCIÓN Y LA INTERFAZ DE USUARIO	77
7.	CONCLUSIONES	81
7.1.	CONCLUSIONES SOBRE EL CUMPLIMIENTO DE OBJETIVOS	81
7.2.	CONCLUSIONES SOBRE EL DESARROLLO	82
	BIBLIOGRAFÍA	84

ÍNDICE DE ILUSTRACIONES

ILUSTRACIÓN 1 - GRÁFICO DE PERSONAS CON DIVERSIDAD FUNCIONAL RECONOCIDA COMPARADO ENTRE CCAA.....	9
ILUSTRACIÓN 2 - GRÁFICO DE PERSONAS CON DIVERSIDAD FUNCIONAL DIVIDIDO POR SEXOS	10
ILUSTRACIÓN 3 - DISTRIBUCIÓN DE AFECTADOS POR SEXO Y POR COMUNIDAD AUTÓNOMA ..	10
ILUSTRACIÓN 4 - GRÁFICO DONDE SE MUESTRA EL TOTAL DE PERSONAS AFECTADAS DIVIDIDOS POR RANGO DE EDADES	11
ILUSTRACIÓN 5 - GRÁFICO DEL PORCENTAJE DE PERSONAS AFECTADAS POR RANGOS DE EDAD EN CADA CCAA.....	11
ILUSTRACIÓN 6 - GRÁFICO DISTRIBUIDO SEGÚN EL PORCENTAJE DE AFECTADOS POR GRADOS DE DIVERSIDAD FUNCIONAL.....	12
ILUSTRACIÓN 7 - GRÁFICO DE NÚMERO DE PERSONAS CON DIVERSIDAD FUNCIONAL DISTRIBUIDO POR GRADOS, SEXO Y RANGOS DE EDAD	13
ILUSTRACIÓN 8 - ACTIVAR MODO DE ASISTENCIA CELESTE.....	21
ILUSTRACIÓN 9 – MENSAJE EXPLICATIVO DEL MODO ASISTENCIA SIN CONNOTACIONES NEGATIVAS	22
ILUSTRACIÓN 10 - OPCIONES DEL MODO ASISTENCIA	22
ILUSTRACIÓN 11 - PERSONALIZACIÓN DE CONTROLES THE LAST OF US PARTE 2	25
ILUSTRACIÓN 12 - IMAGEN DEL ESTADO ACTUAL DEL TABLERO DE MICROSOFT PLANNER.....	28
ILUSTRACIÓN 13 - GRÁFICOS CON ESTADÍSTICAS DE LAS TAREAS	29
ILUSTRACIÓN 14 - DIAGRAMA DE FLUJO DE DIV	33
ILUSTRACIÓN 15 - BOCETO CON LA PANTALLA INICIAL DE DIV	34
ILUSTRACIÓN 16 - IMAGEN DE FONDO DE LA PANTALLA INICIAL	34
ILUSTRACIÓN 17 - DIAGRAMA DE FLUJO DE NINJA RUN.....	37
ILUSTRACIÓN 18 - BOCETO DE LA PANTALLA DE JUEGO DE NINJA RUN	38
ILUSTRACIÓN 19- REPRESENTACIÓN DEL SALTO EN NINJA RUN.....	38
ILUSTRACIÓN 20 - SPRITE SHEET ANIMACIÓN DEL NINJA PARTE 1	39
ILUSTRACIÓN 21 - SPRITE SHEET ANIMACIÓN DEL NINJA PARTE 2	39
ILUSTRACIÓN 22 - SPRITE DE LAS PLATAFORMAS.....	39
ILUSTRACIÓN 23 – SPRITE DE PLATAFORMA EN UNITY SIN CORRECCIÓN DE COLOR	40
ILUSTRACIÓN 24 - SPRITE DE PLATAFORMA EN UNITY CON UNA MÁSCARA GRISÁCEA	40
ILUSTRACIÓN 25 - COLOR DE FONDO APLICADO	41
ILUSTRACIÓN 26 - CAPA DE ÁRBOLES LEJANA APLICADA	41
ILUSTRACIÓN 27 - CAPA DE ÁRBOLES MEDIA CON LOS RAYOS DE SOL	42
ILUSTRACIÓN 28 - CAPA DE ÁRBOLES MÁS CERCANA, FONDO COMPLETO	42
ILUSTRACIÓN 29 - NINJA RUN CON LA INTERFAZ DE USUARIO	43
ILUSTRACIÓN 30 - DIAGRAMA DE FLUJO DE HIGHER & LOWER	45
ILUSTRACIÓN 31 - BOCETO DE LA PANTALLA DE JUEGO DE HIGHER & LOWER	46
ILUSTRACIÓN 32 - IMAGEN DEL DISEÑO FINAL DE LA PANTALLA DE HIGHER & LOWER	46
ILUSTRACIÓN 33 - ESCENAS DIV Y GAME OBJECTS DE NINJA RUN	51
ILUSTRACIÓN 34 - COMPONENTES PLAYER NINJA RUN	52
ILUSTRACIÓN 35 - DETALLES DE ALGUNOS COMPONENTES DEL PLAYER DE NINJA RUN.....	53

ILUSTRACIÓN 36 - BOTÓN QUE INVOC A LA ESCENA DE NINJA RUN	55
ILUSTRACIÓN 37 - SCRIPT PARA EL MANEJO DE ESCENAS INICIAL	56
ILUSTRACIÓN 38 - PRIMER PROTOTIPO DE NINJA RUN.....	57
ILUSTRACIÓN 39 - EXTRACCIÓN DEL CÓDIGO DE SALTO QUE POSTERIORMENTE SE REUTILIZÓ 58	
ILUSTRACIÓN 40 - PROTOTIPO NINJA RUN REHECHO	59
ILUSTRACIÓN 41 - CÓDIGO DE LAS PLATAFORMAS.....	60
ILUSTRACIÓN 42 - BOX COLLIDER 2D INVISIBLE QUE ACTIVA LA VIBRACIÓN	
ILUSTRACIÓN 43 - TRANSFORM ENDPOSITION	61
ILUSTRACIÓN 44 - COLISIÓN PARA QUE EL DISPOSITIVO VIBRE.....	62
ILUSTRACIÓN 45 - SCRIPT DESDE EL CUAL SE CONTROLA PRÁCTICAMENTE TODO EL JUEGO	63
ILUSTRACIÓN 46 - SPAWN PLATFORM ACTUALIZADO	65
ILUSTRACIÓN 47 - LISTA DE TIPOS DE PLATAFORMAS EN EL EDITOR DE UNITY	65
ILUSTRACIÓN 48 - ESKUEMA QUE ILUSTR A LA LISTA DE PLATAFORMAS PARA COMPRENDER LA PROBABILIDAD DE CADA CASUÍSTICA	66
ILUSTRACIÓN 49 - SCRIPT PARA ACTUALIZAR EL TEXTO EN PANTALLA.....	68
ILUSTRACIÓN 50 - GESTIÓN DE LA INTERFAZ DE USUARIO EN EL EDITOR DE UNITY	68
ILUSTRACIÓN 51 - ACTUALIZACIÓN DE LA PUNTUACIÓN.....	69
ILUSTRACIÓN 52 - CORTES DE ZONAS DE CAPAS.....	69
ILUSTRACIÓN 53 - EJEMPLO DE UN GAME OBJECT DE UNA CAPA, CON SU SCRIPT ASOCIADO ..	70
ILUSTRACIÓN 54 - ACTUALIZACIÓN DE MOVIMIENTO DE CADA CAPA	71
ILUSTRACIÓN 55 - IMPLEMENTACIÓN SONIDO DEL SALTO.....	72
ILUSTRACIÓN 56 - MÚSICA DE FONDO Y SONIDOS DE FIN DE JUEGO	73
ILUSTRACIÓN 57 - SCRIPT ENCARGADO DE LA GESTIÓN DE CADA SONIDO Y ASIGNACIÓN DE VALOR EN UNITY.....	74
ILUSTRACIÓN 58 - REPRODUCCIÓN AUTOMÁTICA DE LA DESCRIPCIÓN AL INICIO DEL JUEGO..	75
ILUSTRACIÓN 59 - GESTIÓN DE SONIDOS Y REPRODUCCIÓN DE ESTOS EN CADA RONDA	76
ILUSTRACIÓN 60 - ACCIÓN BOTONES AGUDO Y GRAVE.....	78
ILUSTRACIÓN 61 - REPRODUCCIÓN DE VOCES AL FINAL DEL JUEGO DEPENDIENDO DEL RESULTADO FINAL.....	79
ILUSTRACIÓN 62 - IMPLEMENTACIÓN DE INTERFAZ DE USUARIO DE HIGHER & LOWER	80

1. INTRODUCCIÓN

Actualmente las Tecnologías de la Información y la Comunicación (TIC) tienen cada vez más presencia en el día a día y los videojuegos han dejado de ser vistos como algo vejatorio y han ganado popularidad entre toda la sociedad. De hecho, hay gente que profesionalmente dedica su vida a ser un profesional especializado en algún videojuego y se gana la vida compitiendo en grandes torneos y ligas. Este auge ha tenido como consecuencia la creación de muchos puestos de trabajo a causa de los diferentes perfiles que existen en el propio desarrollo que conlleva un videojuego. Sin embargo, una pieza que debería ser clave actualmente en la sociedad tan desigual en la que vivimos es hacer que sean accesibles para todos y desafortunadamente a penas se dedican recursos para hacer los videojuegos adaptados para cualquiera que sea la situación física del usuario.

Pero, antes de nada, ¿qué se entiende por videojuego? Según la Real Academia Española encontramos 2 definiciones (Real Academia Española, 2021):

- M. Juego electrónico que se visualiza en la pantalla.
- M. Dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor, una computadora u otro dispositivo electrónico.

La primera definición se queda un poco corta, con lo cual se parte de la base de la segunda. Esta definición plantea que un videojuego es un producto electrónico con el que se interactúa a través de los mandos apropiados y del cual se obtiene una respuesta visual. Pero ¿qué ocurre cuando el usuario en cuestión no puede interactuar con el mando que se plantea?, ¿o si no puede ver la respuesta proyectada en la pantalla?, ¿o si sí puede realizar ambas acciones, pero no puede recibir toda la información proporcionada por el videojuego para interaccionar en consecuencia? A estos problemas se enfrentan diariamente personas que por circunstancias de la vida cuentan con alguna diversidad funcional que no les permite interactuar con el entorno tal y como está planteado. Además, dado que el ser humano se caracteriza por ser un ser sociable, no poder jugar a videojuegos es una razón más para que estas personas sean discriminadas y no puedan participar en todos los círculos sociales.

Por ello, a través de este proyecto, se quiere realizar un videojuego desarrollado en C# utilizando el motor Unity donde el objetivo es que sea capaz de ser jugable por personas con diversidad funcional o no; para que así esas personas puedan estar tranquilamente con sus amigos y jugar todos juntos. Se han elegido dos diversidades funcionales diferentes: el déficit visual y el déficit auditivo y en base a ellas se desarrollarán 2 minijuegos en 2D adaptados. Para el déficit visual se pretende realizar un *endless runner*¹ y para el déficit auditivo, un juego arcade.

Finalmente, en el presente documento se expondrá todo el desarrollo que seguirá este proceso. Se realizarán los estudios previos convenientes, se realizará la planificación del proyecto que se apoyará en las metodologías ágiles ya aprendidas y aplicadas en otros proyectos a lo largo del grado; además se investigará sobre todo el software necesario. Para ello en primer lugar se contextualizará sobre la situación actual de los videojuegos adaptados y se entrará en detalle en los conceptos necesarios para así hilarlo todo posteriormente en la exposición del desarrollo donde se mostrará cómo se han aplicado los conocimientos previos, así como lo expuesto en el GDD que se realizará previamente.

¹ El término *endless runner* proviene del inglés y su traducción directa es la de "corredor infinito". Podemos entenderlo como la acción en la que el jugador avanza sin parar en una dirección. Además, podrá ir superando obstáculos que irán dificultando su camino y en caso de no superarlos, el jugador morirá.

2. MARCO TEÓRICO

2.1. INTRODUCCIÓN

La discapacidad está descrita propiamente por la RAE como ‘Situación de la persona que por sus condiciones físicas o mentales duraderas se enfrenta con notables barreras de acceso a su participación social’. A lo largo de todo este apartado se tratarán conceptos importantes a conocer sobre la discapacidad, cuántas personas hay afectadas, los diferentes tipos que hay, cómo afecta esto socialmente a los que las padecen y por último cómo se adaptan los videojuegos ante estas condiciones.

Antes de seguir se debe hacer una aclaración, aunque el término a nivel reglamentario, académico u oficial sea *discapacidad* o *minusvalía* y ambos términos aparecen en la Real Academia Española; durante todo el documento se van a sustituir dichos términos por *diversidad funcional*, aunque no esté aún incluido en la RAE. Diversidad funcional es un término cuyo uso se promueve desde las personas directamente afectadas y pretende sustituir los términos antes comentados dado que éstos tienen una connotación negativa ya que las personas con diversidad funcional no tienen menos capacidades o son menos válidos para realizar cualquier actividad, simplemente las llevan a cabo de una forma diferente, dando así lugar a la diversidad. El término fue propuesto por Javier Romañach Cabrero en el Foro de Vida Independiente en Enero del 2005 (Romañach Cabrero, 2005).

2.2. LA DIVERSIDAD FUNCIONAL EN DATOS

La propia definición de la Real Academia Española ha resaltado un aspecto importante que pocas veces se tiene en cuenta y es que las personas con diversidad funcional tienen, a parte de las propias barreras que les conlleva el propio padecimiento de cada uno, enfrentar las barreras de acceso a su participación social. Para esto debemos entender mejor cuán extendido está en nuestra sociedad y a su vez, los diferentes tipos y cómo afecta cada uno a cada persona.

El Instituto de Mayores y Servicios Sociales pone a disposición de los ciudadanos una Base de Datos Estatal de personas con discapacidad donde se recogen las valoraciones hechas en las diferentes Comunidades Autónomas que constituyen el Estado español, incluyendo Ceuta y Melilla. Esta base de datos está actualizada a fecha de 31 de diciembre

de 2019 y sus datos nos van a servir para poder poner cifras y entender mejor cuál es la extensión total de las personas afectadas (Ministerio de Derechos Sociales y Agenda 2030, 2020).

De ahí podemos obtener que, a fecha de 31 de diciembre de 2019, el total de la población española estaba en 47.450.795 de personas, de las cuales 3.255.843 eran personas con un grado de diversidad funcional reconocido mayor o igual al 33%. Esto supone un 6'86% de la población española total, si redondeamos, 7 de cada 100 habitantes tienen una diversidad funcional reconocida.

De esta base de datos se van a extraer más datos y a analizar diversos aspectos que esta información nos brinda, para ello, con dichos datos se van a elaborar diversas gráficas para comprender mejor la amplia magnitud que abarca todo este tema.

2.2.1. AFECTADOS EN RELACIÓN CON LA POBLACIÓN, POR COMUNIDADES AUTÓNOMAS

En el gráfico mostrado en la ilustración 1 se representa el porcentaje de personas con un grado de diversidad funcional reconocido mayor o igual al 33% sobre la población, comparado con el resultado obtenido en cada Comunidad Autónoma. Como se ha comentado antes, en España estadísticamente 7 de cada 100 personas tienen una diversidad funcional reconocida. Donde más incidencia de personas afectadas respecto al total de habitantes es en Melilla, el Principado de Asturias y Ceuta.

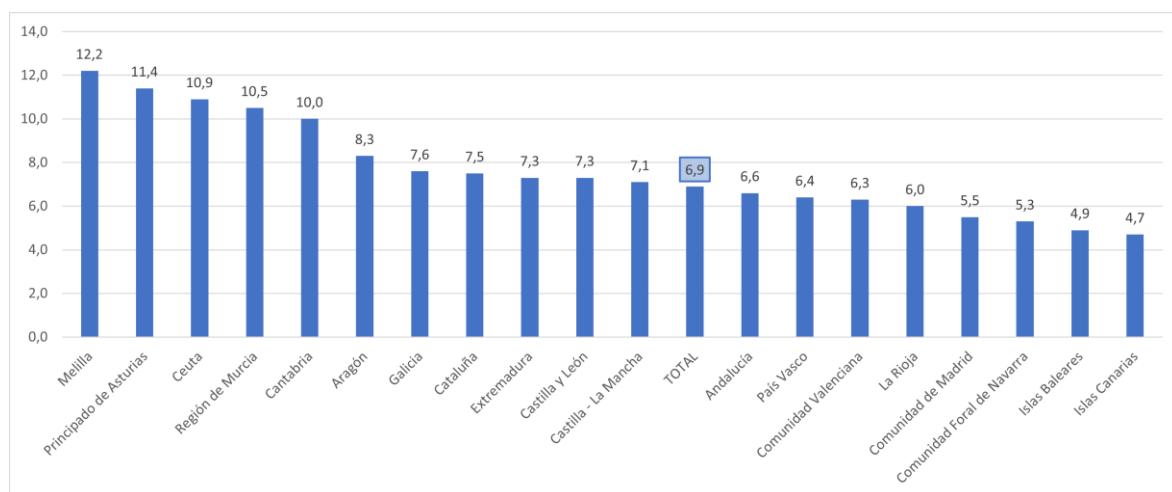


Ilustración 1 - Gráfico de personas con diversidad funcional reconocida comparado entre CCAA

Fuente: Elaboración propia

2.2.2. PERSONAS CON DIVERSIDAD FUNCIONAL POR SEXO

Se ha comentado que, del total de la población española, un 6'86% tiene una diversidad funcional reconocida, de las cuales 1.634.687 son hombres y 1.621.156 mujeres, es decir, el 50'20% de personas afectadas son hombres y el 49'80% mujeres. De estos porcentajes se concluye que la diversidad funcional no discrimina entre sexos.

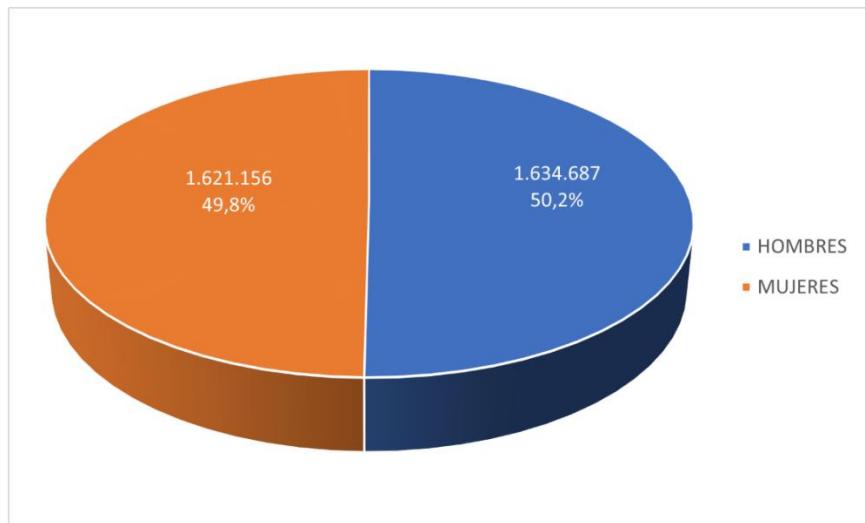


Ilustración 2 - Gráfico de personas con diversidad funcional dividido por sexos

Fuente: Elaboración propia

De estos totales, se puede observar la distribución del total de personas afectadas dividida por sexos en cada Comunidad Autónoma. Las 3 comunidades autónomas con más personas afectadas son Cataluña, Andalucía y la comunidad de Madrid.

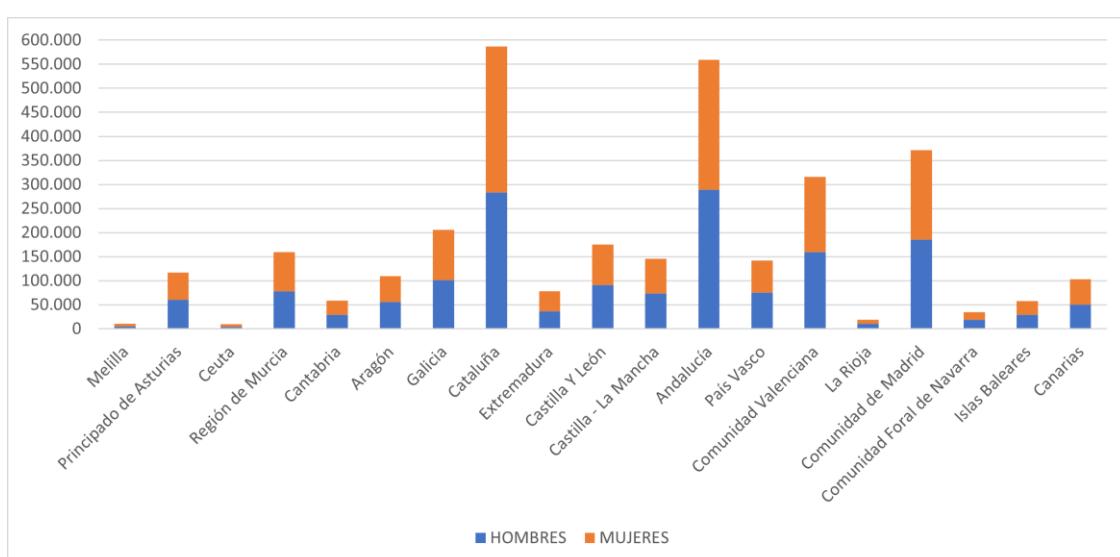


Ilustración 3 - Distribución de afectados por sexo y por Comunidad Autónoma

Fuente: Elaboración propia

2.2.3. PERSONAS CON DIVERSIDAD FUNCIONAL POR EDAD

A continuación, se tratan los datos de personas afectadas divididos por rangos de edad, se puede observar que un 45'4% del total de afectados son personas que tienen 65 años o más, comprensible también por lo que la propia vejez trae consigo, aunque muy de cerca con un 42'75% comprenderían a personas entre 35 a 64 años, siendo ésta la población adulta de España y que está en un rango para estar activa laboralmente.

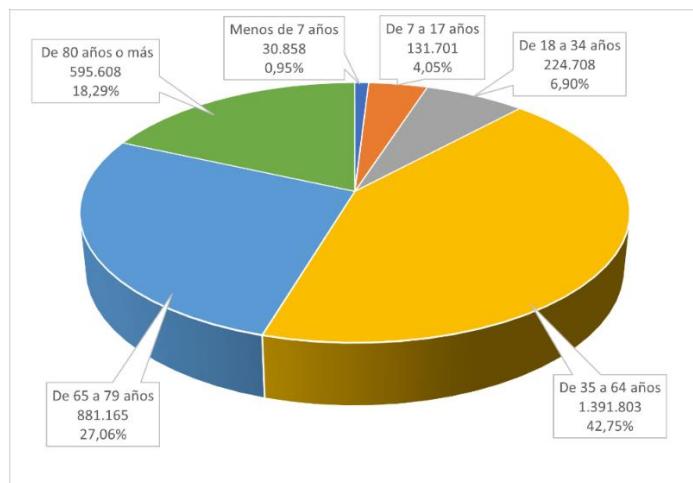


Ilustración 4 - Gráfico donde se muestra el total de personas afectadas divididos por rango de edades

Fuente: Elaboración propia

Como ya se ha observado antes el total de personas afectadas en cada Comunidad Autónoma, sobre el 100% de afectados de cada una de ellas se muestra una división por los mismos rangos de edad del total de afectados.

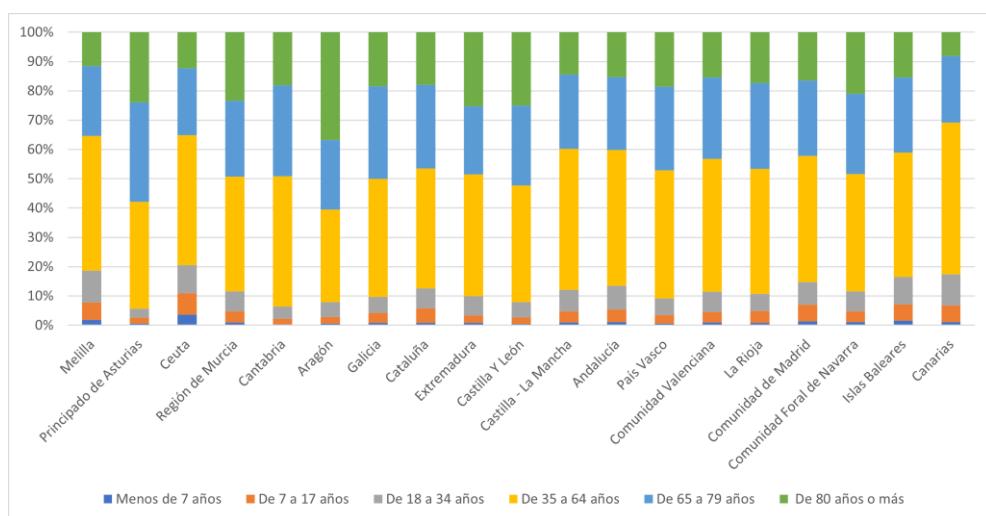


Ilustración 5 - Gráfico del porcentaje de personas afectadas por rangos de edad en cada CCAA

Fuente: Elaboración propia

2.2.4. PERSONAS CON DIVERSIDAD FUNCIONAL POR GRADOS

Para que una persona se le reconozca si tiene diversidad funcional, es examinada por un tribunal médico y éstos dictaminan la sentencia. En dicho dictamen se obtiene un porcentaje dependiendo de la cuantía total de afecciones que tengan y de la categoría a la que pertenezcan, de las que se hablará más adelante. Seguidamente encontramos un gráfico donde se contempla la distribución de las personas con diversidad funcional por grado. El 42% de personas afectadas tienen reconocido un grado de diversidad funcional que está entre el 33% y el 45%.

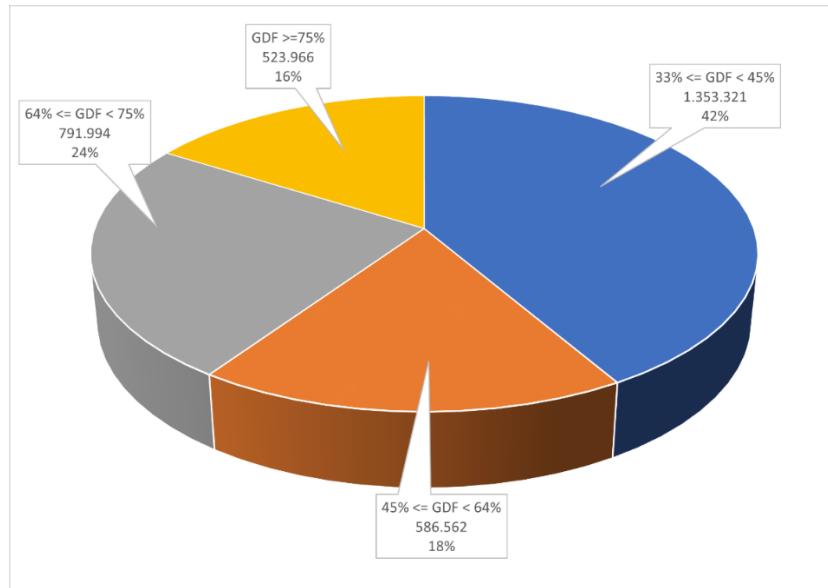


Ilustración 6 - Gráfico distribuido según el porcentaje de afectados por grados de diversidad funcional

Fuente: Elaboración propia

Por último, en el gráfico de la ilustración 7 se unifican todos los datos recopilados, número de personas con diversidad funcional por sexo, edad y grado de afectación. Se puede observar que estadísticamente lo más probable es que los rasgos de una persona con diversidad funcional sean: hombre, con una edad comprendida entre 35 y 64 años y con un grado de diversidad funcional entre el 33% y el 45%.

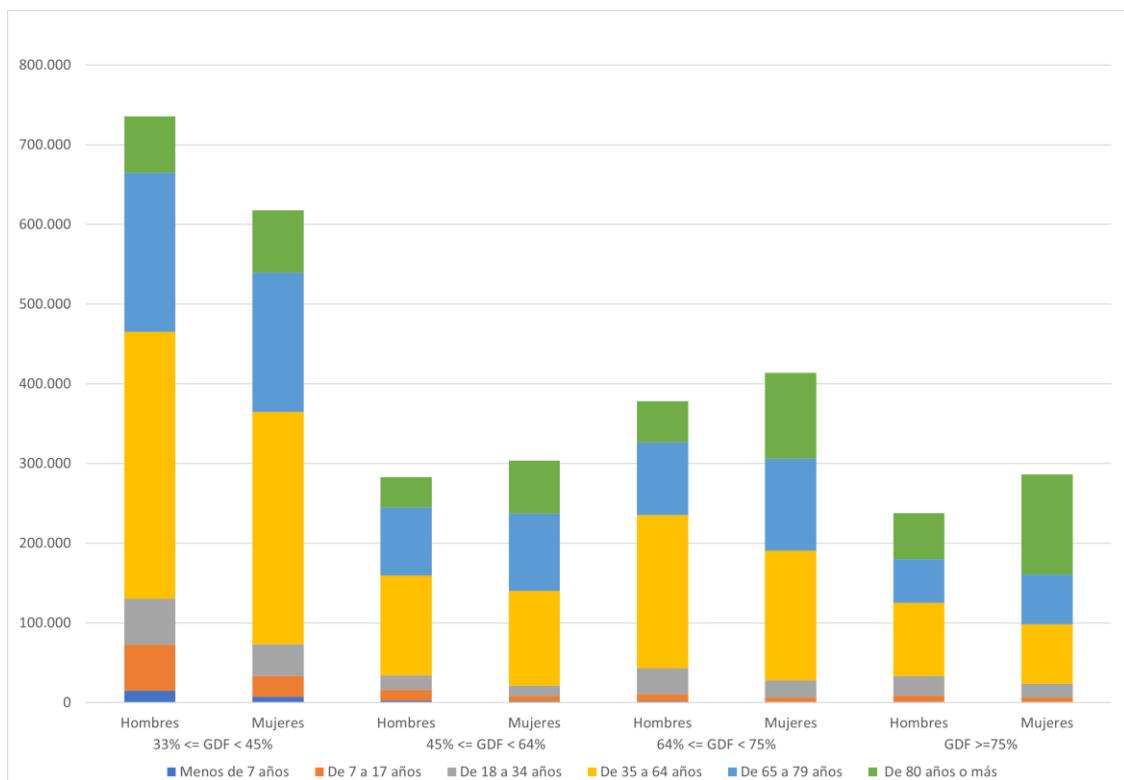


Ilustración 7 - Gráfico de número de personas con diversidad funcional distribuido por grados, sexo y rangos de edad

Fuente: Elaboración propia

2.3. CATEGORÍAS

Tras poner en valor la cantidad de personas afectadas y sus distribuciones, se puede observar que estamos tratando ante una problemática bastante extensa y que no diferencia entre sexo o edad. A su vez también se ha hablado de diferentes grados de afectación, pero no todas las personas cuentan con los mismos tipos de diversidad funcional, no todas afectan de la misma forma ni se tratan igual; hay demasiados tipos concretos de diversidad funcional y sería imposible numerarlos todos en este documento.

Con ayuda de la Clasificación Internacional del Funcionamiento aprobada el 22 de Mayo del 2001 por los 191 países que integran la OMS (Ministerio de Trabajo y Asuntos Sociales, 2001), se clasifican en las siguientes categorías, las cuales no son mutuamente excluyentes:

- **Diversidad funcional física:** Abarca alteraciones corporales que dificultan el movimiento y/o motricidad, restringiendo la realización de las actividades cotidianas. A su vez se divide en funcional y orgánica.

- **Diversidad funcional motriz (discapacidad funcional):** Es la referida a alteraciones que afectan al funcionamiento del sistema neuromuscular y/o esquelético y que dificultan o limitan el movimiento.
- **Diversidad funcional orgánica:** Corresponde a las que afectan a los procesos fisiológicos u órganos internos: sistema digestivo, metabólico, endocrino, respiratorio, etc.
- **Diversidad funcional sensorial:** Es la que se relaciona con las estructuras sensoriales. Puede ser auditiva, visual o afectar a otros sentidos.
- **Diversidad funcional intelectual:** Comprende las alteraciones en la función intelectual, significativamente por debajo del promedio, dificultando así la comprensión y/o respuesta ante distintas situaciones que se dan comúnmente en la vida diaria.
- **Diversidad funcional psicosocial (mental):** Engloba las alteraciones en la conducta adaptativa, con afectación de las facultades mentales y las estructuras neurológicas.

Este proyecto pretende focalizarse sobre la diversidad funcional sensorial, concretamente en la diversidad funcional visual y la auditiva.

2.4. ESTIGMAS SOCIALES

Tras conocer un poco más estadísticamente sobre las personas con diversidad funcional, se va a tratar el lado más humano. Es un hecho que las personas con diversidad funcional afrontan obstáculos estigmatizantes y discriminatorios a la hora de realizar cualquier actividad cotidiana y esto se acentúa si se trata algo más concreto que esté directamente relacionado con su afectación.

Sobre todo, hay un tipo de violencia de la que no se habla, una violencia silenciosa y que es capaz de hacer incluso más daño que cualquier confrontación directamente física, y es la violencia social. Los niños afectados por alguna diversidad funcional tienen mayor riesgo de sufrir acoso en la etapa escolar; este tipo de violencia se empieza a sufrir desde

edades tempranas dado que cuando se es infante no se hace el hincapié necesario en la educación sobre este tema o se hace incorrectamente.

Por ello es importante concienciar adecuadamente, se debe inculcar que su compañero tiene ciertas dificultades para realizar algunas cosas o que las hará de forma diferente pero no porque sea *especial*. De la misma manera que la persona afectada no debe recibir un trato diferente por parte del adulto, puesto que esto incentivaría más aún el hecho de que sus compañeros lo vean como alguien que no es igual. Algo que se hace de forma diferente y que no se hace como todo el mundo no tiene que ser algo especial, ni la persona afectada es alguien super diferente al humano de a pie. Se debe entender que la persona afectada es una persona igual que el resto, no es alguien situado por encima ni por debajo de nosotros, es alguien afín.

- Que la persona afectada no pueda correr o saltar, no lo hace diferente, llevará a cabo la acción de desplazarse de otra manera.
- Que la persona afectada no pueda leer un libro, no impide que pueda conocer la literatura, simplemente lo hará a través de otros medios.
- Que la persona afectada no pueda escuchar el diálogo de una serie, no impide que pueda disfrutar de ella, la verá y comprenderá los diálogos de otra forma.

Y se puede seguir así con más ejemplos, pero estos son los más comunes que podemos encontrar a diario.

La capacidad alude a una relación entre las disposiciones del cuerpo y las estructuras socioculturales y materiales en el tipo de sociedad concreta en la que se inscribe. Las capacidades representan modos de relación de la persona con el entorno.

Nuestras capacidades están tan condicionadas por nuestra constitución orgánica como por las características del contexto. Toda capacidad se realiza en un contexto que la hace posible. En este sentido, consideramos que las capacidades se construyen socialmente; el ser humano tiene actitudes, creencias y sesgos sobre la normalidad.

Estas actitudes han llevado por ejemplo al desarrollo de piernas y brazos artificiales, por ejemplo, en el caso de la talidomida. La talidomida era un sedante que comenzó a

venderse a partir de 1957 y que se administraba como complemento inocuo para tratar las náuseas, la ansiedad, el insomnio y los vómitos matutinos de las embarazadas, pero que causó graves malformaciones en los fetos, así como ausencia de sus extremidades al nacer.

Se ha tenido un enfoque muy normativo dado que el objetivo ha sido dar miembros artificiales, es decir, normalizar a las personas a pesar de que estas herramientas eran toscas, estéticas y a veces poco funcionales. Aquí se ve reflejado la puesta en valor de la normatividad, ¿estamos ante una solución estética o funcional?

Encontramos alternativas por las que podrían haber optado las personas afectadas, pero sobre las que hay prejuicios sociales: gatear está menos aceptado y esto se ve como algo inferior a utilizar una silla de ruedas, y esto a su vez se considera inferior al uso de piernas artificiales, especialmente si parecen naturales. Esta jerarquía realmente no está basada en la funcionalidad de la persona, sino en la rígida aceptación de un esquema corporal normativo.

La mirada capacitista no es otra cosa que la multitud de prácticas, representaciones y valores que se utilizan para la creación del cuerpo normativo y de su carácter regulador como norma y criterio de normalidad. Esta noción de *mirada* va mucho más allá del simple *ver*. Estamos ante un *modo de ver* que amplía lo que la persona está viendo dado que le aplica un código normativo. La mirada pone de manifiesto una relación de poder del sujeto que mira sobre la persona observada.

A continuación, se puede ver atentamente a lo que se refiere esto con un breve ejemplo:

- Si una mujer no puede acceder a un edificio, estamos ante un caso de discriminación.
- Si una persona de otra religión no puede acceder a un edificio, estamos ante un caso de discriminación.
- Si una persona de otra raza no puede acceder a un edificio, estamos ante un caso de discriminación.
- Si una persona en silla de ruedas no puede acceder a un edificio, se considera discapacidad.

Y aquí se hace hincapié en la palabra discapacidad porque el hecho de que no pueda acceder al edificio no se considera discriminación por el hecho de que simplemente la persona no tiene la capacidad necesaria y normal que tiene cualquiera para realizar esa acción.

2.5. LAS TIC Y LA DIVERSIDAD FUNCIONAL

Hay una capacidad humana muy relevante y frecuente que no se tiende a incluir en el marco normativo de las capacidades y se trata de la capacidad de adaptar el funcionamiento a elementos externos. Anteriormente se han comentado algunas de las acciones comunes que las personas con diversidad funcional no pueden llevar a cabo como todos conocemos, las harán de forma diferente, pero ¿cómo?

El uso de recursos tecnológicos es un punto clave en el desarrollo de herramientas que faciliten este objetivo. Las TIC, acrónimo de Tecnologías de la Información y la Comunicación, juegan un papel esencial en la eliminación de las barreras, creando herramientas para superar los obstáculos que los usuarios puedan encontrarse.

Seguidamente se exponen ejemplos concretos de herramientas que podemos encontrar para ayudar a las personas afectadas en las categorías anteriormente mencionadas.

2.5.1. LAS TIC Y LA DIVERSIDAD FUNCIONAL FÍSICA

Como ejemplos de herramientas para complementar las limitaciones al nivel de la motricidad, podemos encontrar teclados adaptados con diversas funciones. Primeramente, se pueden ampliar o disminuir su tamaño para facilitar su pulsación dependiendo de la amplitud de movimiento de los usuarios. También existen ratones de bola, los cuales permiten dirigir el movimiento del cursor con la bola central que posee, sin necesidad de tener que desplazarlo sobre la mesa.

Y, por último, un ámbito que puede aportar mucha independencia es la domótica, que a su vez se sirve muchas veces de un software de reconocimiento de voz para comprender y ejecutar las órdenes recibidas. A través de herramientas como Alexa y una instalación de bombillas adecuadas podremos ordenarle que encienda y apague las luces de una estancia sin necesidad de darle al interruptor.

2.5.2. LAS TIC Y LA DIVERSIDAD FUNCIONAL SENSORIAL

Para afectaciones visuales podemos encontrar hoy en día multitud de lectores de textos para personas con un grado de ceguera bastante severo, los propios móviles ya empiezan a traerlo integrado para hacer así mucho más fácil el uso de teléfonos móviles inteligentes.

Es bien conocido que los móviles ya no se usan sólo para llamar y como hemos dicho antes una diversidad funcional puede traer muchas barreras sociales. Con la actual expansión de las redes sociales es muy importante que personas con diversidad funcional puedan socializar igual que cualquier otra persona. Esto les abre muchas puertas, incluso para conocer a gente en su misma situación y que así no se sientan tan incomprendidos y vean que no están solos.

Asimismo, también podemos encontrar con lupas ampliadoras de pantalla, por si el grado de pérdida de visión no fuese tan extremo, pero sí considerable. Para las afectaciones auditivas, nos encontramos con teléfonos o herramientas de videollamadas que permiten la conversión de voz en texto. Además, la mensajería instantánea es un punto clave para que pueda comunicarse con cualquier persona de una manera rápida y eficaz.

Por último, no se podía cerrar este apartado sin hacer una mención a [OMU](#), el cual es un servicio de intérprete virtual de lengua española a lengua de signos española. El proyecto fue realizado por el grupo OmuSapiens durante el curso 2019-2020 en el itinerario de Gestión de Contenidos de este grado.

2.5.3. LAS TIC Y LA DIVERSIDAD FUNCIONAL INTELECTUAL

Las personas con esta afectación suelen tener muchas dificultades de aprendizaje, así que gracias al desarrollo de la tecnología podemos encontrar nuevos entornos de aprendizaje virtuales que también pueden ofrecer una disponibilidad completa a cualquier hora del día, para que así sirven de refuerzo y ejercitación y no pierda el trabajo realizado sólo hecho a una determinada hora en un día concreto.

Además, estas tecnologías también son utilizadas para la creación de programas específicos para el diagnóstico y el tratamiento de algunas deficiencias, dado que se pueden tener diversas unidades didácticas para diversos grados. Este material didáctico

se deja en disposición del usuario para su consulta cuando quiera, solventando problemas de extravío de los materiales.

Estas tecnologías se podrán apoyar también en otras ya descritas antes, por ejemplo, el uso de lectores de texto que le permiten acceder a información de textos escritos si esta persona aún no ha desarrollado la habilidad de leer.

2.5.4. LAS TIC Y LA DIVERSIDAD FUNCIONAL PSICOSOCIAL (MENTAL)

El uso de *smartphones* o *tablets* permiten que la persona pueda suplir, aumentar o mejorar sus habilidades de comunicación verbal. Algunos recursos se utilizan como medio para hacer una función mediadora. El uso de aplicaciones de videollamadas permite una interacción personal más directa y esto puede transmitir una mayor tranquilidad a la persona o a sus familiares.

El uso de aplicaciones que apoyen a la persona para que pueda organizarse y estructurar su entorno mejor para hacerlo predecible y así permitirle una mayor adaptación a su contexto y se capaz de dar una mejor respuesta a las exigencias que se le presenten. Por ejemplo, el uso de calendarios, alarmas y recordatorios integrados.

Por último, mencionar también que en este grado podemos encontrar otros Trabajos de Fin de Grado que han tratado sobre la realización de videojuegos adaptados para personas con parálisis cerebral, tales como [Footb-all](#) realizado por David Gómez Davó en 2014, [Fórmula Chair](#) realizado por Aitor Font Puig en 2015 y [Fisio Run](#) realizado por Alberto Martínez Martínez en 2016.

2.6. VIDEOJUEGOS ADAPTADOS

Una vez ya se ha puesto en base todos pilares sobre la diversidad funcional tales como el alcance, los tipos que hay, las dificultades que se pueden encontrar las personas afectadas y qué tipo de herramientas podemos encontrar hoy en día para tratar de hacer la vida de estas personas más fácil; no se debe olvidar el objetivo de todo este estudio y es la creación de un videojuego completamente adaptado.

Retomando lo que se ha comentado en la introducción sobre los expandidos que están los videojuegos actualmente y continuando con cómo afecta a un niño tener una diversidad funcional, dentro del ámbito social los videojuegos actualmente ya son otro factor a tener

en cuenta. A su vez, como también se ha nombrado antes, las redes sociales juegan un papel cada día más relevante en esta sociedad y es una vía muy utilizada para que la gente se comunique y socialicen entre sí.

Entre los niños de hoy en día está muy expandido el tema de los *influencers* y hay un numeroso público juvenil que sigue de cerca a los que juegan a videojuegos, ya sea para entretenerse y pasar el rato, como para aprender a jugar mejor. Saber sobre la actualidad del mundo del videojuego es algo tan común como hablar de fútbol. Los creadores de contenido a menudo se juntan para crear competiciones entre sí y fácilmente en una conversación de colegio puedes encontrar a unos hablando de quién ha ganado la liga o quién ha quedado primero en un torneo donde han participado diversos famosos.

Complementaria a esta idea, encontramos que igual que los niños juegan en el patio al fútbol y tratan de imitar la plantilla del Real Madrid o cualquier equipo y sueñan con algún día formar parte de él, también pasa con los videojuegos, también sueñan con ser profesionales de los *e-sports*, dado que es una industria que cada vez está en un mayor auge.

Pero para llegar a ser el mejor no basta con ver lo que hacen, uno no se hace buen jugador de fútbol viendo muchos partidos de fútbol; uno mejora su habilidad a base de entrenamientos, partidos y esfuerzo. De la misma forma, un joven mejorará su habilidad en un videojuego echándole horas, entrenando, jugando y esforzándose en aprender. O simplemente el joven quiere pasar un rato divertido con sus amigos echándose una pachanga, de la misma forma que puede querer pasar un rato divertido con sus amigos jugando a cualquier juego cooperativo en línea.

Las necesidades de los usuarios son diferentes y por ello las respuestas en forma de opciones de accesibilidad deberían serlo también. El primer paso que ya se puede encontrar en casi cualquier videojuego es añadir la opción de los subtítulos, pero el pilar para que un juego sea accesible para cualquier persona debe permitir personalizar también la experiencia mecánica del videojuego.

Se pueden encontrar amplios ejemplos de videojuegos que permiten modificar y elegir la dificultad y así poder hacer el juego alcanzable a diferentes audiencias. Llegados a este

punto se va a tratar en qué videojuegos podemos encontrar esto verdaderamente reflejado y cómo lo han llevado a cabo.

2.6.1. CELESTE

Un ejemplo de producto que ha llevado esto a cabo de una manera muy completa es sin duda Celeste: un videojuego de plataformas cuya historia trata precisamente de la superación ante las adversidades y para ello cuenta su historia a través de un viaje hasta la cima de la montaña Celeste. Pese a ser un juego de plataformas desafiante, existe la posibilidad de modificar la velocidad del juego y activar la invulnerabilidad de la protagonista o la energía infinita.

Asimismo, no trata al jugador con connotación negativa si utiliza estas opciones, al revés, siempre trata de hacerle llegar al jugador que el reto no lo debe poner el juego, sino la propia persona y que de nada sirve la dificultad si se deja de lado el punto de la diversión; esta es una forma bastante positiva de hacerle entender ese punto a los jugadores y hace mucho más liviana y pone en común la idea de la accesibilidad.

En este modo Asistencia, el propio juego lo define como que es un modo hecho para que puedas disfrutar del juego, haciendo hincapié en que el objetivo de cualquier juego es pasarlo bien y disfrutar. A continuación, en las siguientes imágenes sacadas directamente del juego podemos observar cómo se ha integrado lo comentado:

Inicialmente cuando creas o cuando seleccionas una partida para cargarla, puedes activar o desactivar el modo de asistencia.



Ilustración 8 - Activar modo de asistencia Celeste

Fuente: Captura del juego de elaboración propia

Si lo activamos, se nos muestran una serie de mensajes donde se explica en qué consiste el modo y el objetivo del propio.

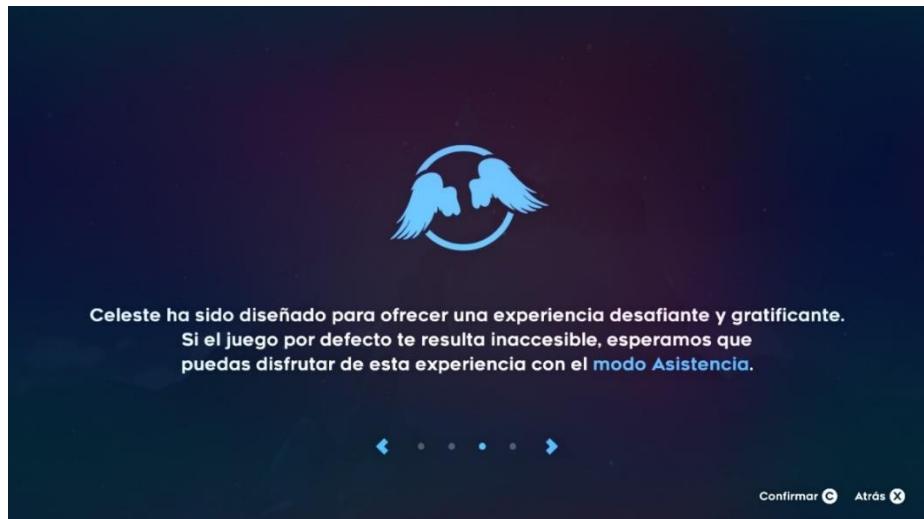


Ilustración 9 – Mensaje explicativo del modo asistencia sin connotaciones negativas

Fuente: Captura del juego de elaboración propia

En este último mensaje mostrado en la ilustración 9 podemos observar lo que se ha comentado antes, de cómo hacen hincapié en que el objetivo es que el juego sea desafiante, pero sin llegar a ser frustrante, tratando de hacer que el jugador pueda disfrutar de la experiencia.

Por último, las opciones que ofrece este modo asistencia se encuentran a través del menú de pausa, seguidamente en la ilustración 10 podemos ver todas las opciones que hay.

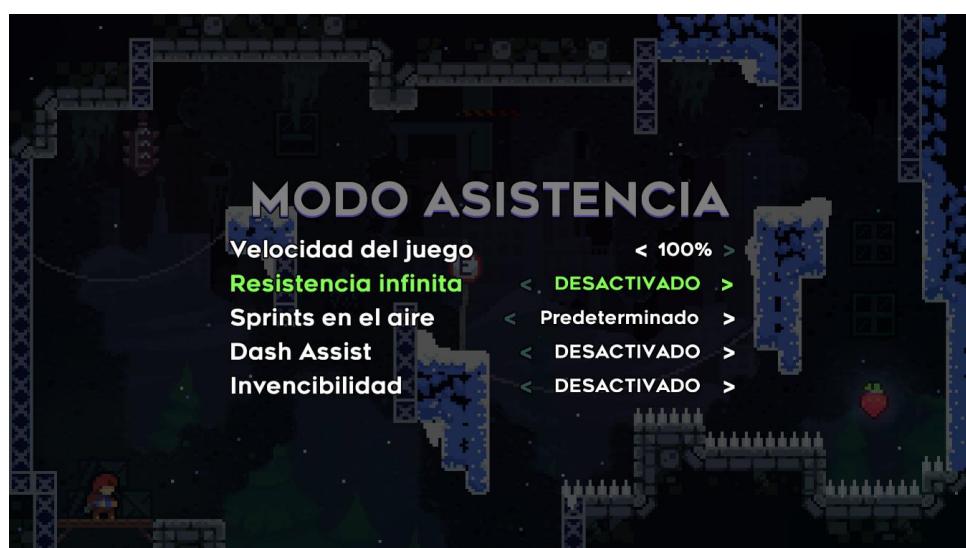


Ilustración 10 - Opciones del modo asistencia

Fuente: Captura del juego de elaboración propia

Como conclusión, añadir todas estas opciones no sólo hace el juego más accesible a personas con diversidad funcional, sino que para el resto de las personas también puede suponer una manera de no generar frustración.

Este tipo de modos no suponen ninguna penalización sobre el modo inicial, así que simplemente añaden contenido adicional para esas personas que por cualquier motivo son menos habilidosas, pudiendo disfrutar de la experiencia completamente.

Esto permite también que la comunidad crezca dado que el público del juego es mayor, con lo cual los esfuerzos invertidos inicialmente en desarrollar estas opciones, luego se van a ver reflejados, dado que, si haces el juego mucho más accesible y amplias tu público, más personas lo comprarán.

Celeste fue creado originalmente como un prototipo en cuatro días durante una Game Jam y aunque han conseguido desarrollar un videojuego con un modo de accesibilidad más amplio que lo que se puede encontrar normalmente; a continuación, se va a analizar un videojuego mundialmente conocido: The Last Of Us Parte II, desarrollado por Naughty Dog, estudio que pertenece a Sony.

2.6.2. THE LAST OF US PARTE II

The Last of Us Parte II es un videojuego insignia de PlayStation y es considerado por muchos un gran juego, pero sobre todo es un videojuego que ha desarrollado de manera excelente todo su apartado de accesibilidad. Como prueba de esto es la puntuación media obtenida en la página [Can I Play That?](#), página dedicada a analizar cuán accesible es un videojuego en diversos apartados.

Las puntuaciones obtenidas para The Last of Us Parte II son de un 9'5 para el apartado de adaptación para las deficiencias visuales (Kombat, 2020) y un 10 para el apartado sonoro (Craven, 2020). Tal es la importancia que en la propia página de información del videojuego de PlayStation se puede encontrar un apartado dedicado a la accesibilidad y los más de 60 ajustes de accesibilidad implementados (Sony Interactive Entertainment, 2020).

Partiendo de la información oficial obtenida, los ajustes se pueden clasificar en 3 grupos: ajustes para la accesibilidad visual, ajustes para la accesibilidad sonora y ajustes para la accesibilidad motora. Los ajustes más resaltados para cada uno de ellos son los siguientes:

Accesibilidad motora

Se pueden encontrar configuraciones para bloquear automáticamente el movimiento en los objetivos, cambiar automáticamente de armas, recolección de objetos automática, asistente de cámara, eliminar el desvío de armas, omitir la opción de puzzle.

Accesibilidad visual

Algunas de las configuraciones son transcribir el texto a voz, imagen de alto contraste, escalado del HUD, pistas de audio de combate y transversales y algunas mencionadas anteriormente pero que también darían apoyo a la accesibilidad visual como el fijar al apuntar y omitir la opción de puzzle.

Accesibilidad auditiva

Se habilitan opciones como indicadores de percepción del enemigo, notificaciones de recogida de objetos, subtítulos de la historia, subtítulos del combate, inclusión del nombre en los subtítulos y señales de vibración de combate y del uso de la guitarra.

Tras mencionar los ajustes más destacados de cada grupo, otra de las medidas de accesibilidad desarrollada es la de controles alternativos, donde se pueden personalizar completamente los controles.

Esto va desde cambiar la orientación del mando hasta reasignar todos los comandos a diferentes botones, incluidas las opciones de deslizar el panel táctil y agitar el mando. También se permite personalizar individualmente la ejecución de las acciones, como mantener pulsado pase a una pulsación alterna o que una acción de pulsar repetidamente se ejecute manteniendo el botón. Algunos de estos ajustes se pueden observar a continuación en la ilustración 11.

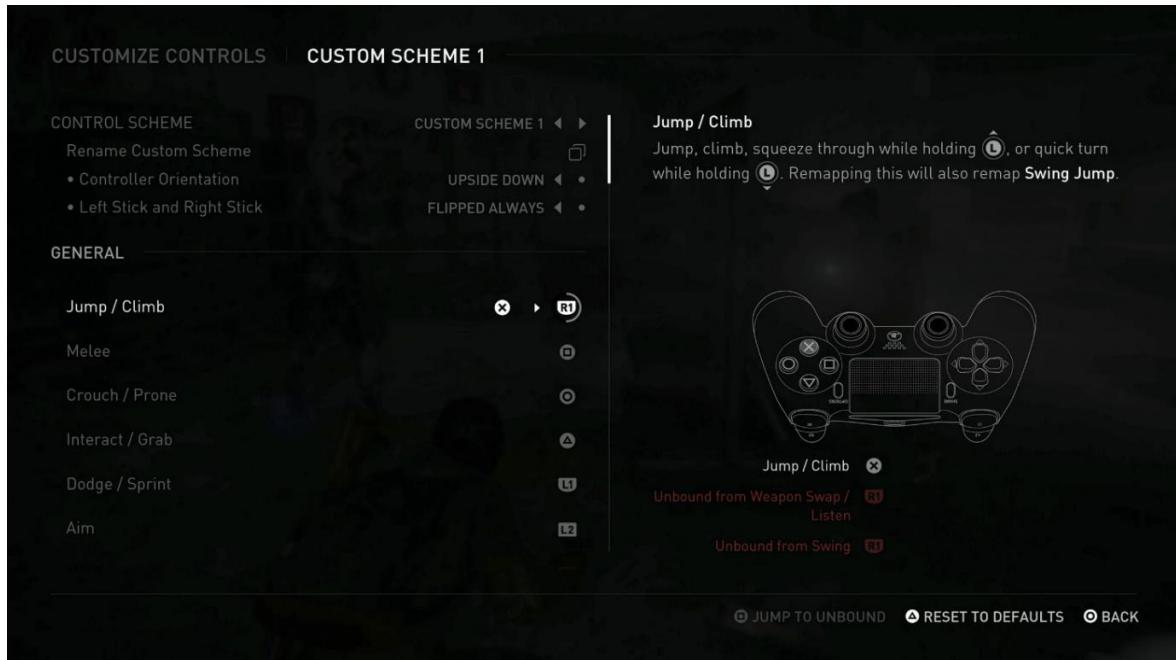


Ilustración 11 - Personalización de controles The Last of Us Parte 2

Que juegos tan relevantes se planteen las opciones de accesibilidad dentro de su presupuesto de desarrollo es algo muy positivo para empezar a cambiar la sociedad y que se tengan más en cuenta a las personas con diversidad funcional.

Por otra parte, Microsoft tampoco se ha quedado atrás y en su propia documentación sobre el desarrollo de videojuegos han incluido una guía con pautas de mejoras y consejos para que el desarrollo sea más accesible; es una guía bastante amplia, se puede encontrar en inglés en el siguiente [enlace](#).

Con toda esta información recopilada sobre cómo diversos estudios han implementado cada uno sus opciones de accesibilidad y las casuísticas que se pueden encontrar las personas afectadas a la hora de jugar a un videojuego, se pretenden aplicar los conocimientos adquiridos para el correcto desarrollo de DIV, un videojuego inclusivo enfocado principalmente en la diversidad funcional visual y auditiva.

3. OBJETIVOS

Una vez hecha la puesta en valor de las personas con diversidad funcional, la gran extensión de personas afectadas y cómo el sector tecnológico y el mundo de los videojuegos están evolucionando para realizar juegos inclusivos, el objetivo principal de este trabajo es el desarrollo de dos videojuegos 2D dentro de uno mismo. DIV, el cual es el nombre de la aplicación completa, incluirá dos minijuegos: Ninja Run y Higher & Lower. Ninja Run se trata de un *Endless Runner* y Higher & Lower está dentro del género Arcade. Dichos juegos deberán ser capaces de ser jugados por cualquier usuario independientemente de las características de este y para ello se tomarán como base las lecciones aprendidas en el apartado anterior.

Este objetivo principal puede dividirse a su vez en los siguientes objetivos más concretos:

- Aprendizaje sobre el motor Unity y a su vez del lenguaje de programación C#. Todo el proyecto se desarrollará en el motor 2D de Unity.
- Aplicación de las metodologías ágiles aprendidas durante todo el grado, especialmente durante el Itinerario de Creación y Entretenimiento Digital, de 4º Curso. Todo el conocimiento de las herramientas serán un gran apoyo para la gestión y planificación del proyecto.
- Diseño e implementación de todos los apartados que engloben los videojuegos.
 - En Ninja Run lo entendemos como la generación infinita y aleatoria del nivel, el diseño del nivel, el personaje y el sonido.
 - En Higher & Lower abarcará la generación aleatoria del orden de los sonidos y la interfaz de usuario.
- Adaptación de los videojuegos, una vez se asienten los principales cimientos de cada juego, se deberá modificar para hacer que las características de la persona que va a jugar no influyan en la jugabilidad.

4. METODOLOGÍA

La metodología empleada para el desarrollo de este trabajo ha sido una combinación de la metodología ágil y el desarrollo por prototipos. Para la parte de control y gestión del proyecto se ha usado una metodología ágil, en concreto la metodología Scrum. Por otra parte, para el desarrollo del producto se ha empleado una metodología basada en modelos por prototipos. Asimismo, se ha llevado un control de versiones a través de GitHub.

4.1. METODOLOGÍA DE CONTROL Y GESTIÓN DEL PROYECTO

La metodología empleada para llevar a cabo un control y una correcta gestión del proyecto se ha basado en una metodología ágil tipo Scrum. Este tipo de metodología se suele aplicar en proyectos llevados a cabo grupalmente, ya que así las tareas se pueden separar correctamente para que todos los miembros tengan un trabajo que no dependa de ningún otro y así se obtengan rápidamente resultados (Proyectos Ágiles, 2020).

Aunque este proyecto ha sido desarrollado por una única persona, se ha aplicado esta metodología puesto que ya se ha trabajado con ella antes exitosamente y ayuda al desarrollo de una forma correcta y organizada, siendo así más rápida.

Asimismo, se ha llevado a cabo un desarrollo iterativo, en el cual se planificaron 5 iteraciones temporales. 1 iteración para la gestión de la pantalla principal, 3 para el desarrollo de Ninja Run y por último, 1 para Higher & Lower. Teniendo en cuenta la duración de cada una y asignándoseles una carga de trabajo correspondiente al tiempo real efectivo de éstas. Estas iteraciones se sincronizaban con las reuniones con el tutor de forma que las reuniones servían para corroborar cuán efectivas habían sido, haciendo una retrospectiva del trabajo realizado y en ellas se planificaban los objetivos para la siguiente iteración.

4.1.1. MICROSOFT PLANNER

Como complemento para un correcto uso de esta metodología, se ha utilizado Microsoft Planner. Esta herramienta trata de un software de administración de proyectos con interfaz web. Permite administrar planes de trabajo creando tableros Kanban con tarjetas

de tareas, de tal forma que facilita asignar de manera fácil y organizada diferentes actividades a una o más personas incluidas en el proyecto.

También se puede marcar una fecha objetivo además del grado de avance de la misma; en las tarjetas se da la posibilidad de compartir archivos y añadir notas e incluye un chat para hablar sobre el trabajo para estar al día del progreso.

Gracias al convenio de la Universidad de Alicante con Microsoft, los estudiantes pueden utilizar esta herramienta de forma gratuita dado que está incluida en el plan de Office de las cuentas de MSCloud.

En este proyecto se ha utilizado para planificar las actividades llevadas a cabo en cada iteración y el estado de estas, así como para añadir anotaciones de ideas sobre tareas que no eran objetivo de esa iteración pero que no se quería que cayesen en el olvido.

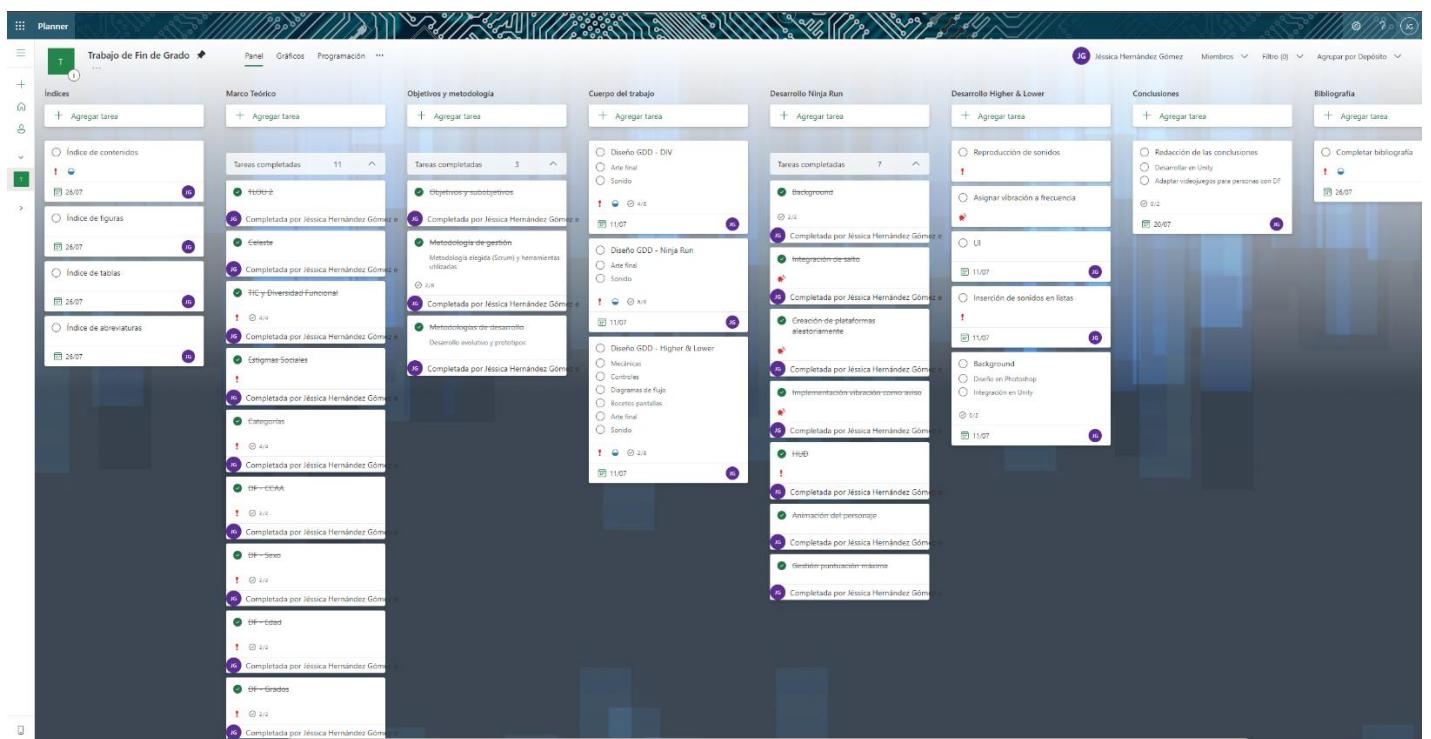


Ilustración 12 - Imagen del estado actual del tablero de Microsoft Planner.

Fuente: Captura de elaboración propia

Las tarjetas que aparecen tachadas en la ilustración 12 son tareas que ya se han completado, las que no, son tareas que están pendientes aún, con su fecha máxima de finalización, asimismo cada tarea tiene una prioridad, si tienen el símbolo de ! significa que la prioridad de la tarea es importante y si tienen el símbolo de campana es que es

urgente. El tablero está separado en columnas que corresponden a las diversas secciones del trabajo.

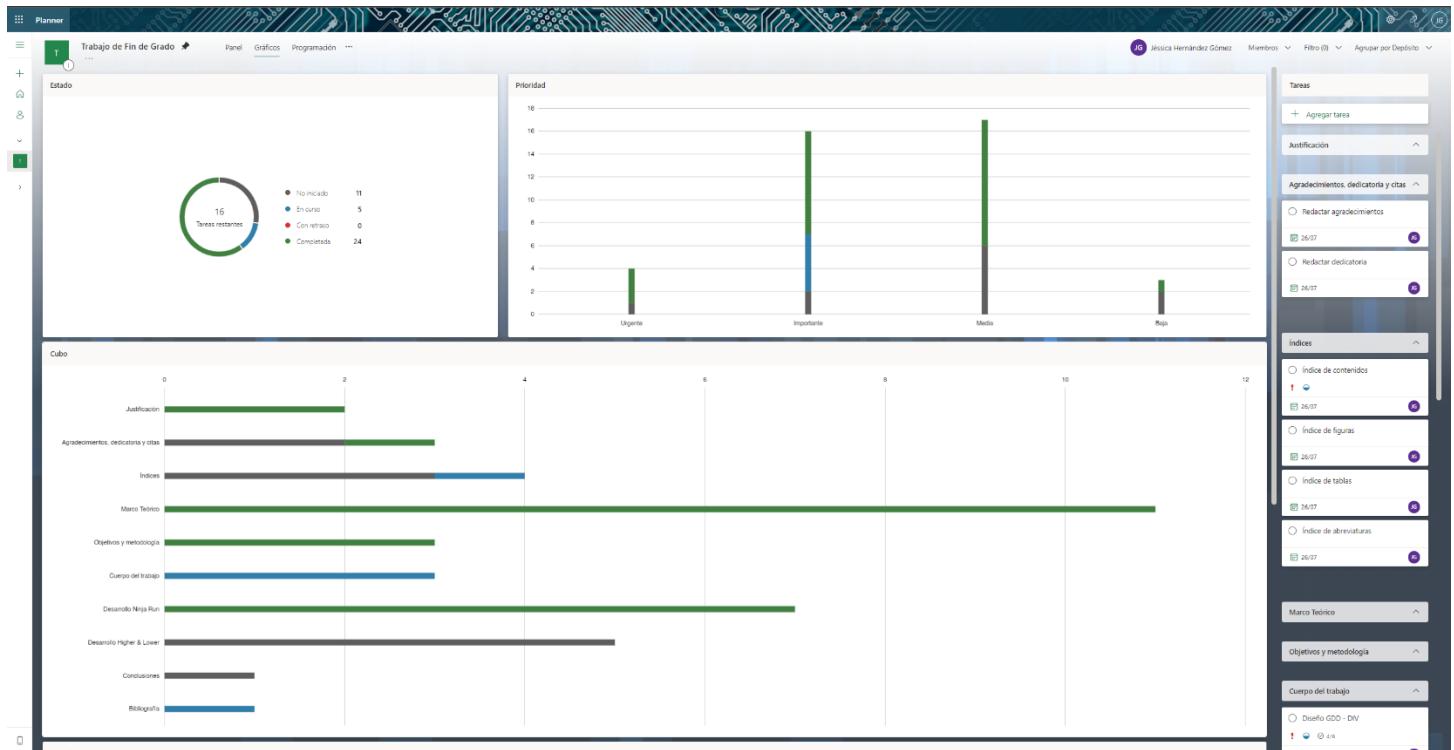


Ilustración 13 - Gráficos con estadísticas de las tareas

Fuente: Captura de elaboración propia

Como se puede ver en la ilustración 13, Microsoft Planner también cuenta con un apartado de Gráficos donde podemos encontrar estadísticas detalladas sobre las tareas de nuestro tablero.

4.2. METODOLOGÍA DE DESARROLLO DEL SOFTWARE

La metodología utilizada para el desarrollo del software ha sido el modelo por prototipos, este modelo trata de un desarrollo evolutivo. Empieza con la definición de los objetivos globales y tras ello se identifican los requisitos conocidos y las áreas donde se necesita más definición. Se trata de un buen modelo a utilizar para dar al usuario una vista preliminar de parte del software (EcuRed, 2019). Está conformado por diversas etapas.

- Diseño rápido del producto final
- Construcción del prototipo
- Evaluación del prototipo

- Pulir y ajustar el prototipo
- Obtención del producto final

Este desarrollo tiene ciertas ventajas tales como que no se modifica el flujo del ciclo de vida, reduce los costes y aumenta la probabilidad de éxito, exige contar desde el principio con las herramientas adecuadas y ello reduce los posteriores retrasos por no contar con ellas o por tener que habituarse a nuevas herramientas a mitad de desarrollo.

El motivo por el cual se ha utilizado este modelo en este tipo de proyecto es porque así se maximizará la calidad del producto final, contando con las limitaciones de tiempo disponible para el desarrollo.

Se han generado 5 prototipos, primero uno básico para entender rápidamente el funcionamiento de Unity y cómo maneja las escenas de los diversos juegos y posteriormente en cada juego se ha podido ir avanzando independientemente el uno del otro.

4.3. CONTROL DE VERSIONES

Para llevar a cabo un correcto control de versiones, se ha utilizado un repositorio en GitHub. Esta plataforma es utilizada tanto para alojar como para gestionar proyectos y sus diversas versiones. Fue fundada en 2008 y tras su gran éxito entre los desarrolladores, 10 años más tarde Microsoft adquirió la compañía por 7000 millones de dólares. El sistema trabaja bajo el control de versiones Git, cuyo propósito es llevar registro de los cambios en archivos del repositorio con los cambios locales del ordenador en el que se ejecuta Git, coordinando así el trabajo realizado por varias personas en caso de tratarse de archivos compartidos (Fernández, 2019).

Esta herramienta será muy útil para el almacenamiento de todo el proyecto, así como para llevar una correcta organización y control de este. Esto está motivado también porque permite diferenciar cada versión ya sea por fechas, como por el título que nosotros le hayamos dado a cada versión, siendo rápidamente identificables en caso de querer volver a una versión antigua.

Se ha utilizado [Git for Windows](#), para poder utilizar el terminal para gestionarlo todo dado que ya se cuentan con conocimientos previamente adquiridos sobre el uso de la herramienta, adquiridos durante el estudio del grado.

5. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO

En este apartado se va a hacer todo un recorrido sobre las diferentes partes que conformarán el propio videojuego, analizando así cada aspecto con el que contará el juego, ayudando así a que el desarrollo sea más ágil dado que todo habrá estado definido previamente. Como se tiene el escenario general del videojuego DIV y dentro de él se pueden observar dos escenarios completamente diferentes dentro de un mismo juego, se van a analizar primeramente todos los detalles generales de DIV y a continuación como dos videojuegos independientes, se analizará Ninja Run y después Higher & Lower.

5.1. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE DIV

5.1.1. FICHA TÉCNICA

- **Título:** DIV
- **Plataforma:** Dispositivos móviles Android
- **Género:** Arcade
- **Audiencia:** Todas las edades
- **Formato:** Apaisado
- **Número de jugadores:** Un solo jugador
- **Idioma:** Español

5.1.2. CONCEPTO DEL VIDEOJUEGO

DIV es un videojuego donde el principal objetivo es pasar el rato intentando superar la puntuación más alta. Otro de sus puntos clave es que todos puedan jugarlo, sin importar las características del usuario. Se plantea como un juego arcade, donde puedes jugar y pasar un rato divertido y puedes volver a jugarlo cuando quieras, donde las partidas no duren más de 5 minutos. El título de DIV realmente son las siglas de Diversión, Inclusión y Videojuegos, que es la propia esencia del videojuego. En él encontraremos dos videojuegos y el usuario podrá elegir a cuál jugar: Ninja Run y Higher & Lower.

5.1.3. CONTROLES

El control principal es la pulsación sobre la pantalla. Dependiendo de la zona donde se realice la pulsación, ocurrirán unas acciones u otras.

5.1.4. DIAGRAMAS DE FLUJO

La gestión de la aplicación general y los juegos se realizará mediante las escenas de Unity, en el desarrollo se ampliará la información sobre éstas, en una escena se almacenan y ejecutan todos los elementos necesarios de esa escena. No se debe confundir las escenas con los estados del juego, una escena es por así decirlo una sección del juego y en una escena podemos encontrar diversos estados. En DIV contaremos con 3 escenas: la escena inicial, la escena de Ninja Run y la escena de Higher & Lower. A continuación, se muestra el diagrama de flujo de la escena inicial y cuando se analice cada juego, se mostrará el diagrama de flujo de sus escenas. Cuando iniciemos el juego cargaremos la escena principal, se reproducirá un audio indicando que se debe girar el dispositivo y para jugar a Ninja Run hay que pulsar en cualquier parte del lado izquierdo de la pantalla y para jugar a Higher & Lower hay que hacer lo mismo, pero sobre la parte derecha de la pantalla, así como una pequeña descripción de cada juego. No será necesario que el usuario espere a que termine de reproducirse el audio para elegir una opción.

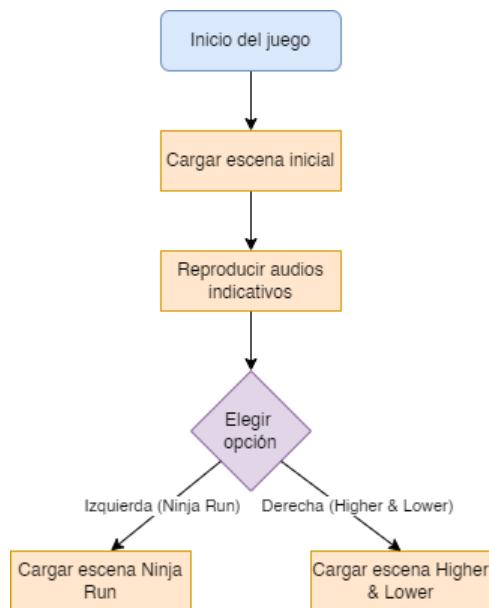


Ilustración 14 - Diagrama de flujo de DIV

Fuente: Elaboración propia

5.1.5. BOCETOS DE PANTALLAS

La pantalla inicial será muy básica y simple. Contará con una imagen de fondo donde en una mitad habrá una imagen de Ninja Run y en la otra mitad una imagen de Higher & Lower y arriba encontraremos el logo de DIV. No contará con botón de salir dado que para ello será tan fácil como pulsar el botón Atrás del dispositivo.

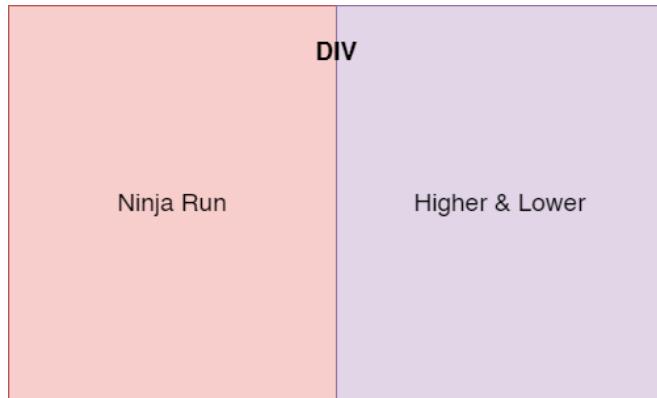


Ilustración 15 - Boceto con la pantalla inicial de DIV

Fuente: Elaboración propia

5.1.6. ARTE FINAL

La pantalla inicial queda diseñada tal y como se muestra en la siguiente ilustración, en la zona izquierda se ha puesto una imagen representativa de cada juego:



Ilustración 16 - Imagen de fondo de la pantalla inicial

Fuente: Elaboración propia

5.1.7. SONIDO

Cuando iniciamos el juego se reproduce el siguiente audio:

Bienvenido a DIV, antes de nada, asegúrate de que el dispositivo esté en posición horizontal. Para jugar a Ninja Run, un juego donde serás un Ninja que deberá saltar sobre plataformas para lograr superar su récord pulsa sobre la zona izquierda de la pantalla. O si quieres jugar a Higher & Lower, un juego donde tendrás que acertar si el sonido que escuchas es más grave o agudo que el anterior, pulsa sobre la zona derecha de la pantalla.

La voz del audio explicativo y la grabación de la misma ha sido llevada a cabo por la compositora Claudia Moreno Vidal, conocida como [Niels Prods](#) en las redes, quien ya participó previamente en la elaboración de la música del proyecto de ABP del Itinerario de Creación y Entretenimiento Digital, [Gunkour](#), creado por el grupo Blue Amber durante el curso 2020/21.

Además, Claudia también ha colaborado durante el curso 2021/22 en la elaboración de la música del corto de animación del Trabajo de Fin de Grado 'Fuego a la vista', desarrollado por Sofía Ivars Buyolo, Juan Carlos Soriano Martínez y Dolores Santiago Castillo.

5.2. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE NINJA RUN

5.2.1. FICHA TÉCNICA

- **Título:** Ninja Run
- **Plataforma:** Dispositivos móviles Android
- **Género:** Endless Runner - Arcade
- **Audiencia:** Todas las edades
- **Formato:** Apaisado
- **Número de jugadores:** Un solo jugador
- **Idioma:** Español

5.2.2. CONCEPTO DEL VIDEOJUEGO

Ninja Run es un *endless runner* donde el objetivo es simple, saltar, saltar para cambiar de plataforma y que el personaje pueda seguir corriendo. Las plataformas aparecerán

aleatoriamente al mismo nivel del jugador, más arriba o más abajo. El único fin que habrá en Ninja Run sucederá cuando el usuario no logre saltar a tiempo y se caiga. Para adaptar este juego se pretende hacer que cuando jugador se encuentre al final de una plataforma y esté lo suficientemente cerca para saltar, el dispositivo vibre.

La puntuación del juego consistirá en el número de plataformas superadas. Cuando la partida finalice, se reproducirá un audio indicando si se ha superado o no el récord.

5.2.3. MECÁNICAS

La mecánica principal del protagonista será el salto, donde el jugador se desplazará en el eje Y. El jugador siempre estará quieto respecto al eje X, aunque daremos la sensación de movimiento con la animación del correr y el desplazamiento del fondo mediante la técnica *parallax scrolling*².

Por otra parte, las plataformas se crearán aleatoriamente; contamos con 3 tipos de plataformas: una se crea a una altura mayor que la plataforma actual, otra se crea al mismo nivel que la plataforma actual y, por último, una que se crea por debajo de la altura de la plataforma actual; su única mecánica será el desplazamiento en el eje X, para poder acceder de unas plataformas a otras el jugador deberá saltar.

5.2.4. CONTROLES

Como contamos únicamente con la mecánica del salto para personaje principal, este saltará cuando el usuario toque cualquier zona de la pantalla.

5.2.5. DIAGRAMAS DE FLUJO

El siguiente diagrama de flujo mostrado en la ilustración 17 parte desde la decisión del jugador de jugar a Ninja Run, se detalla todo lo que ocurriría en una partida completa.

² El efecto *parallax scrolling* es una técnica donde las imágenes del fondo pasan más lentamente por la cámara que las imágenes de primer plano, creando así una ilusión de profundidad en una escena 2D y añadiendo un sentido de inmersión a la experiencia. Posteriormente se explicará más detalladamente cómo se ha llevado a cabo esta técnica.

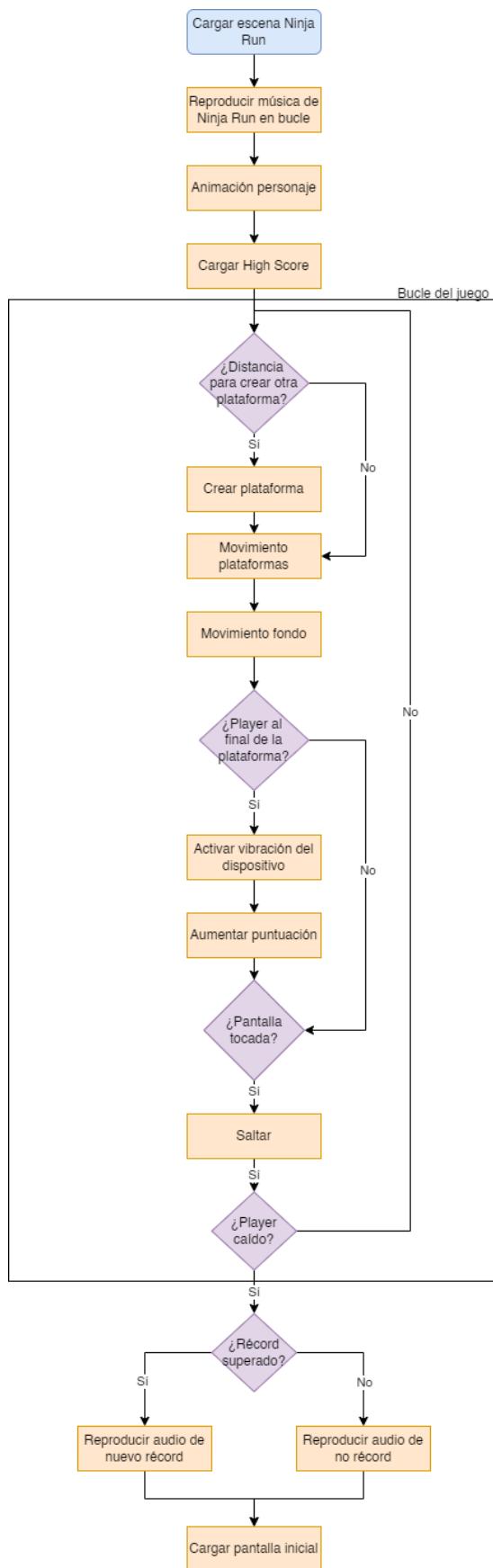


Ilustración 17 - Diagrama de flujo de Ninja Run

Fuente: Elaboración propia

5.2.6. BOCETOS DE PANTALLAS

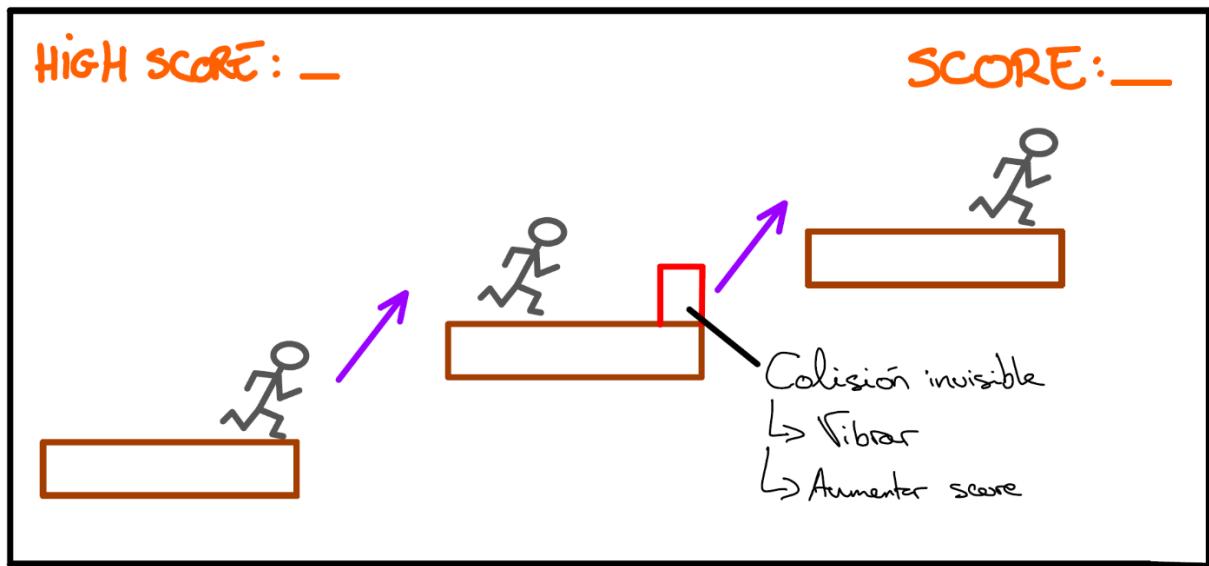


Ilustración 18 - Boceto de la pantalla de juego de Ninja Run

Fuente: Elaboración propia

Al final de cada plataforma, se encontrará una colisión invisible, cuando el jugador pase por ahí ocurrirán dos eventos: el dispositivo vibrará para avisar al jugador de que debe de saltar y se aumentará la puntuación.

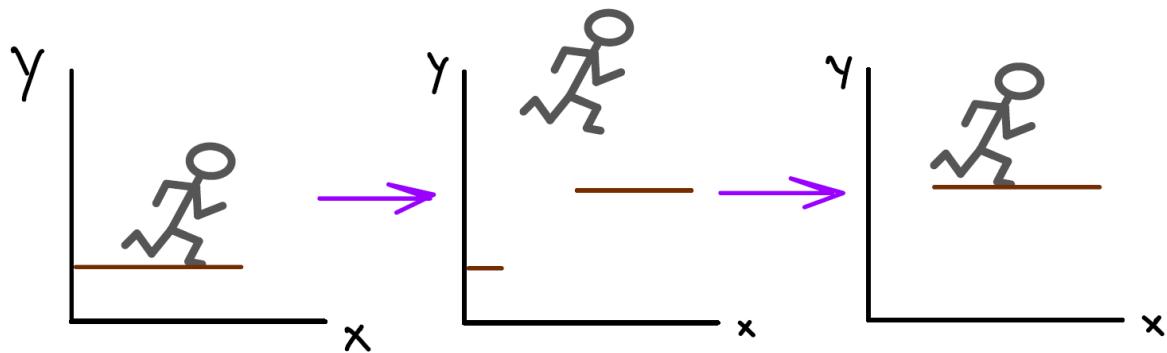


Ilustración 19- Representación del salto en Ninja Run

Fuente: Elaboración propia

La sensación del salto sólo produce movimiento en el eje Y puesto que serán las plataformas las que se acerquen. Asimismo, como ya se ha comentado antes la sensación de movimiento se incentivará con el efecto *parallax scrolling*.

5.2.7. PERSONAJE PRINCIPAL

El protagonista del videojuego es un Ninja, ya que es un personaje perfectamente capaz de correr y saltar para superar las plataformas.

El personaje no fue diseñado de 0, sino que se adquirió su licencia a través de la Unity Asset Store, la propia tienda de artículos de Unity donde las personas pueden publicar los artículos que creen bien de manera gratuita o poniéndoles un precio a los packs.

Del pack que se adquirió, el cual es [2D Character – Ninja](#) creado por [Copy Spright](#), se ha empleado el *Sprite* y la animación del ninja caminando y esta animación se reproduce en bucle durante todo el juego.

El *Sprite Sheet* donde se muestran todos los *Sprites* que conformarán la animación es el siguiente:

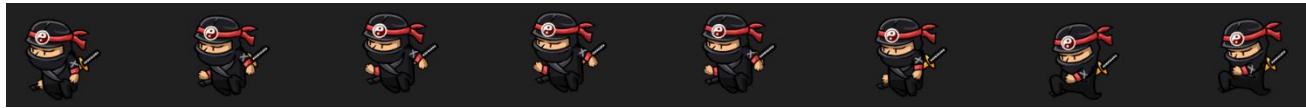


Ilustración 20 - *Sprite Sheet animación del ninja parte 1*

Fuente: Captura de elaboración propia

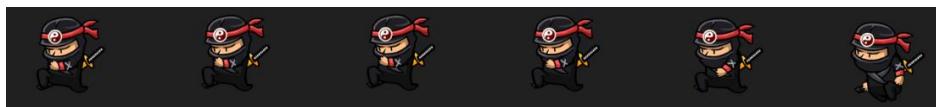


Ilustración 21 - *Sprite Sheet animación del ninja parte 2*

Fuente: Captura de elaboración propia

5.2.8. PLATAFORMAS

Las plataformas tienen un diseño simple, siendo éstas bloques de tierra con césped por encima, las plataformas independientemente de que sean más largas o cortas tienen el mismo *Sprite* base, el cual es el siguiente:



Ilustración 22 - *Sprite de las plataformas*

Fuente: Captura de elaboración propia

Al ver el contraste con el fondo quedaba demasiado claro y para lograr una mayor homogeneidad con el entorno se le aplicó una máscara grisácea directamente en Unity para reducir su tonalidad. A continuación, en la imagen 23 se muestra la plataforma done

el primer color aplicado es blanco, es decir el *Sprite* se queda con su color inicial y luego en la ilustración 24, el *Sprite* tiene una máscara grisácea aplicada para que quede más oscuro, se han incorporado las imágenes seguidas para un mayor contraste.

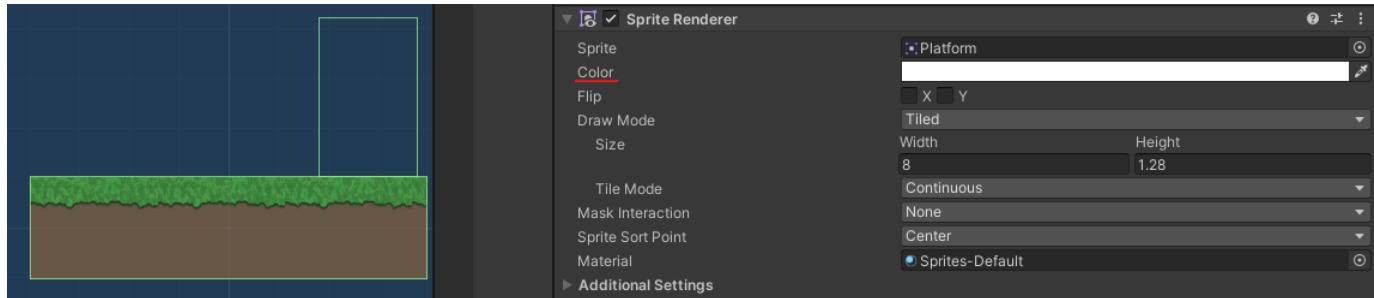


Ilustración 23 – Sprite de plataforma en Unity sin corrección de color

Fuente: Captura de elaboración propia

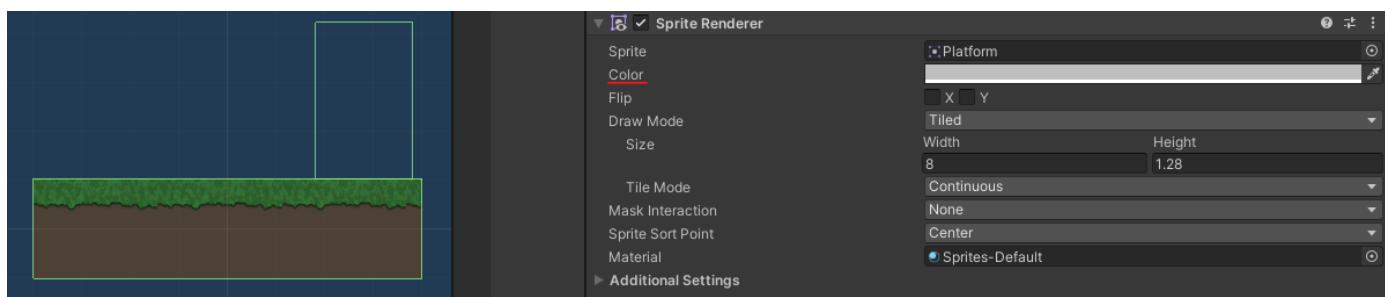


Ilustración 24 - Sprite de plataforma en Unity con una máscara grisácea

Fuente: Captura de elaboración propia

5.2.9. FONDO Y EFECTO PARALLAX

Para el fondo se han utilizado unas imágenes también de licencia gratuita disponibles en itch.io. El paquete de recursos se llama [Daemon Woods Parallax Background](#) y como su propio título indica está preparado para aplicarlo en un efecto *Parallax Scrolling*, ya que el fondo viene en diversas imágenes, las cuales serían las capas del fondo.

El *Parallax Scrolling* es una técnica mediante la cual las imágenes de fondo las cuales están separadas en diferentes capas se desplazan a diferentes velocidades cada una. Mediante este efecto se consigue crear una ilusión de un fondo más natural ya que hay unas figuras en diferentes planos al moverse a distintas velocidades y el cerebro piensa que está viendo un paisaje en 3D, aunque sean imágenes en 2D.

En Ninja Run hay 3 niveles de profundidad, diferenciados por 3 imágenes diferentes y un color sólido de fondo ya que las imágenes tienen transparencias. Seguidamente se

describen las capas empleadas desde la capa más profunda hacia la más cercana y cuál es el resultado superponiendo una sobre otra.

- Color de fondo: Es aplicado directamente desde la cámara principal, en el apartado del desarrollo se darán más detalles. Su código hexadecimal es #DA5E53.

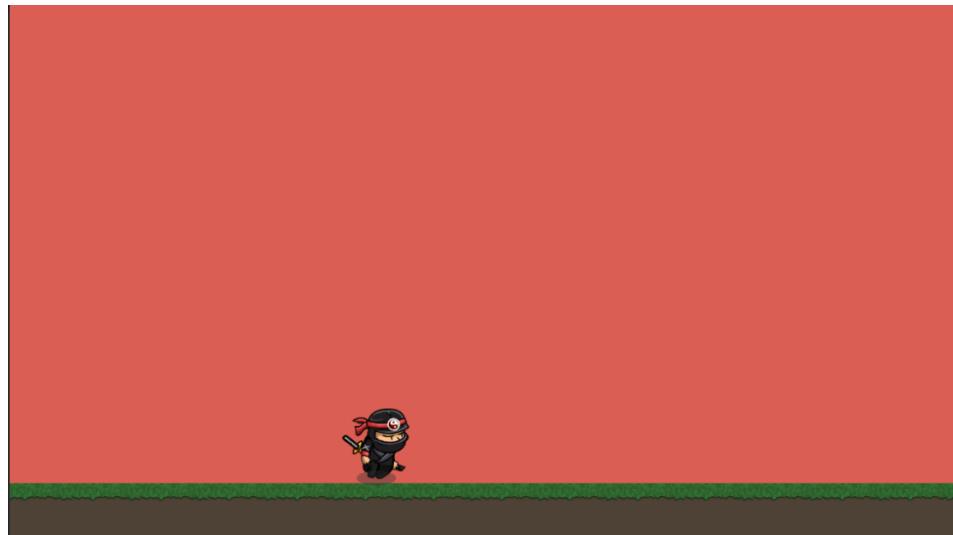


Ilustración 25 - Color de fondo aplicado

Fuente: Captura de elaboración propia

- Capa de árboles más lejana: Tiene un color parecido al del fondo y como se puede apreciar entre los árboles, tiene huecos transparentes, los cuales rellenan con el color de fondo.



Ilustración 26 - Capa de árboles lejana aplicada

Fuente: Captura de elaboración propia

- Capa de árboles media: En esta capa encontramos colores más oscuros, así como líneas oblicuas que dan la sensación de los rayos de sol que se cuelan entre los árboles.



Ilustración 27 - Capa de árboles media con los rayos de sol

Fuente: Captura de elaboración propia

- Capa de árboles cercana: Esta es la última capa y la más cercana, donde se encuentran mejor definidos los árboles y, junto con los setos, tienen sombras del mismo color que la capa media y lejana.

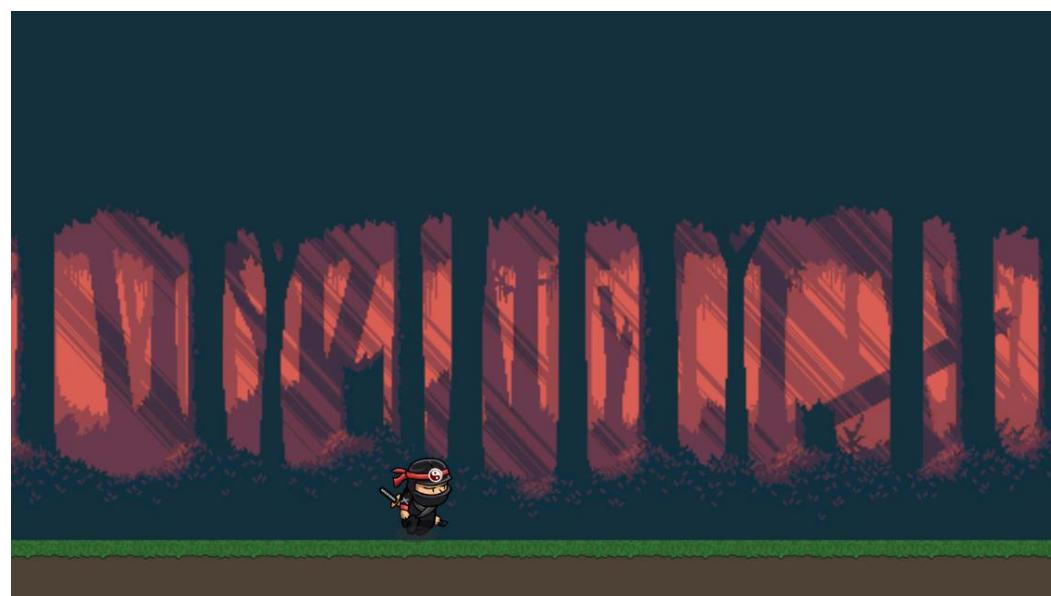


Ilustración 28 - Capa de árboles más cercana, fondo completo

Fuente: Captura de elaboración propia

5.2.10. INTERFAZ DE USUARIO (UI)

Para finalizar con todos los elementos visuales, no nos podemos olvidar de la interfaz de usuario o *User Interface (UI)* en inglés. La escena no podía estar sobrecargada con demasiada información ya que el usuario tiene que estar pendiente de las plataformas así que sólo habrá dos etiquetas diferentes.

En la esquina superior izquierda se encuentra la etiqueta de puntuación máxima donde al cargar el juego se llenará con la puntuación máxima obtenida hasta la fecha en el dispositivo. Si en el transcurso de la partida se supera el récord, el valor se irá actualizando con el nuevo valor que vaya tomando.

En la esquina superior derecha se encuentra la etiqueta de puntuación la cual se actualizará en todo momento en el juego, una vez el jugador supere una plataforma se sumará 1 a la puntuación total.

Cuando el jugador muera se reproducirá la frase: ¡Enhorabuena, has superado el récord! O ¡No has conseguido superar el récord, vuelve a intentarlo!, dependiendo de si en el transcurso de esta ha superado el récord. Tras ello se volverá a la pantalla inicial.

La fuente elegida para el texto en pantalla es Nothing You Could Do Bold (Geswein, 2014) y el resultado en el juego es el mostrado en la ilustración 29:



Ilustración 29 - Ninja Run con la interfaz de usuario

Fuente: Captura de elaboración propia

5.2.11. MÚSICA Y EFECTOS DE SONIDO

Durante todo Ninja Run se reproduce una canción cuya duración es de 29 segundos, pero está puesta en bucle, además de las frases comentadas anteriormente. La grabación de voz de los audios hablados y la creación de la canción ha sido llevada a cabo por la compositora Claudia Moreno Vidal.

Dado que el juego tiene una temática oriental, la música debía ir acorde y acentuar la inmersión en este mundo. Para ello se han contado con instrumentos digitales orientales tales como el Koto, el Guzheng y los Taiko Drums, los cuales son muy utilizados para la percusión. Durante todo el desarrollo se le fue mostrando avances a la compositora y se le pasaron demos jugables para que pudiera inspirarse.

Para el salto se ha utilizado un sonido disponible de manera gratuita y libre de derechos, bajo la licencia Creative Commons 0, en el banco de sonidos de Free Sound, se puede encontrar [aquí](#).

5.3. DOCUMENTO DE DISEÑO DEL VIDEOJUEGO DE HIGHER & LOWER

5.3.1. FICHA TÉCNICA

- **Título:** Higher & Lower
- **Plataforma:** Dispositivos móviles Android
- **Género:** Arcade
- **Audiencia:** Todas las edades
- **Formato:** Apaisado
- **Número de jugadores:** Un solo jugador
- **Idioma:** Español

5.3.2. CONCEPTO DEL VIDEOJUEGO

El objetivo es superar las rondas. Al inicio de cada ronda se reproducirán 2 sonidos, haciendo una pausa entre ellos. El usuario tiene que identificar si el segundo sonido es más agudo o grave que el primero. Para adaptar este juego se pretende hacer que cuando se reproduzcan los sonidos, el dispositivo vibre en función de la frecuencia de estos. Si un

sonido es más agudo tendrá una mayor frecuencia, con lo cual la vibración será mayor que la vibración de un sonido grave. Al final de la partida se reproducirá un audio dependiendo de si se ha superado el récord de rondas o no.

5.3.3. MECÁNICAS

La mecánica principal es el sonido y vibración.

5.3.4. CONTROLES

El control principal es la pulsación sobre la pantalla. Dependiendo de la zona donde se realice la pulsación, ocurrirán unas acciones u otras.

5.3.5. DIAGRAMAS DE FLUJO

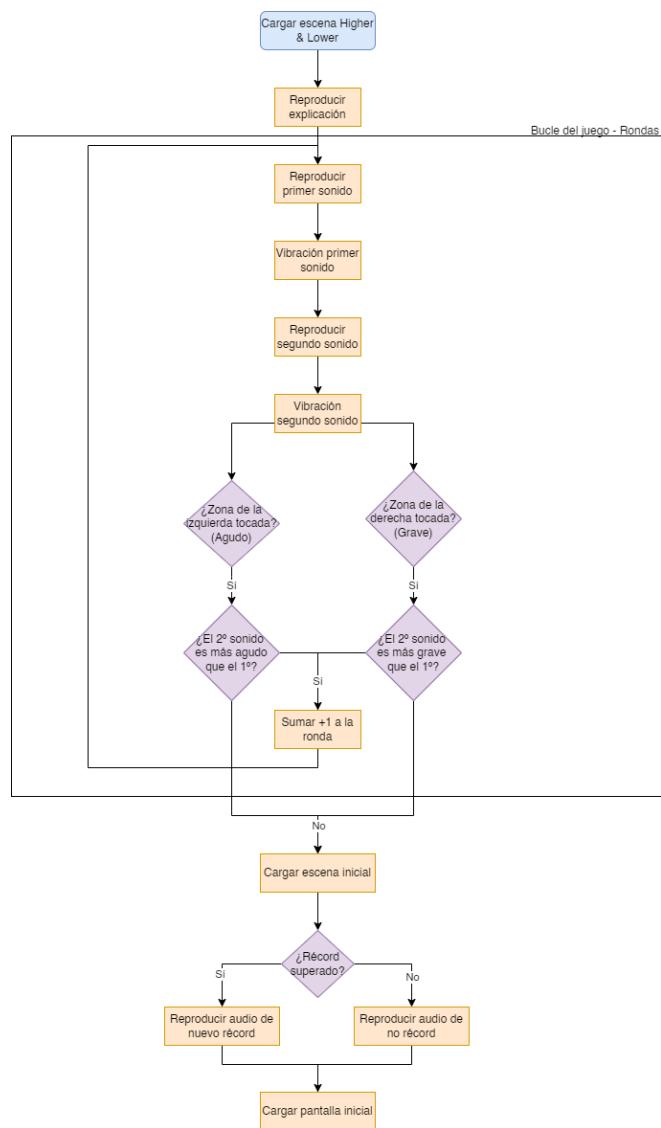


Ilustración 30 - Diagrama de flujo de Higher & Lower

Fuente: Elaboración propia

5.3.6. BOCETOS DE PANTALLAS

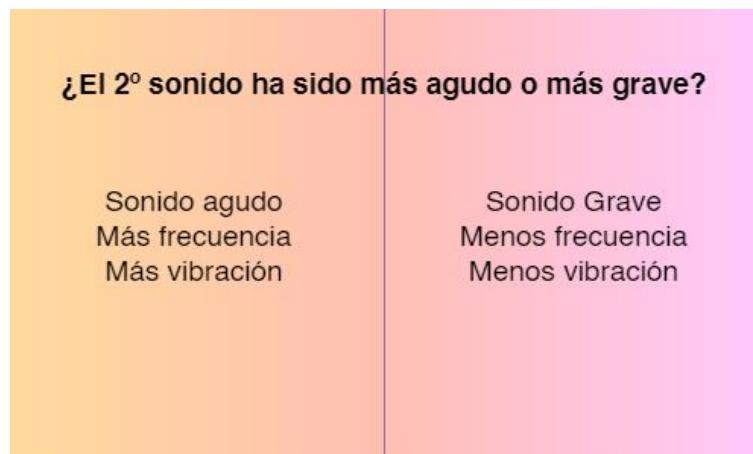


Ilustración 31 - Boceto de la pantalla de juego de Higher & Lower

Fuente: Elaboración propia

Cuando el botón invisible de la zona izquierda sea pulsado, se comprobará si el segundo audio ha sido más agudo que el primero. Si es correcto, se pasará a la siguiente ronda donde se seguirá el mismo proceso: reproducir sonidos con la vibración y esperar una respuesta. En caso de ser pulsado el botón que ocupa toda la mitad derecha de la pantalla, se comprobará si el sonido ha sido más grave, si acierta ocurrirá lo antes comentado. Si el jugador falla en la respuesta, se reproducirá un sonido indicando si se ha superado el récord y se volverá a la pantalla inicial.

5.3.7. ARTE FINAL

La pantalla inicial queda diseñada tal y como se muestra en la siguiente ilustración:



Ilustración 32 - Imagen del diseño final de la pantalla de Higher & Lower

Fuente: Elaboración propia

5.3.8. SONIDO

Cuando se inicia el juego, se reproducirá la siguiente frase:

Bienvenido a Higher & Lower, a continuación se van a reproducir 2 sonidos. ¿Eres capaz de diferenciar si el segundo sonido es más agudo o grave que el primero? Si aciertas, pasarás de ronda y volverás a escuchar 2 sonidos con el mismo objetivo.

Cuando termine el juego, se utilizarán las frases grabadas para Ninja Run para que el jugador sepa si ha superado el récord o no. En caso de que el jugador supere todas las rondas, se reproducirá una nueva frase: Enhorabuena, has superado todas las rondas de Higher & Lower.

Al igual que en las anteriores frases, Claudia se ha encargado de la grabación de estas y de doblarlas.

6. DESARROLLO

En este capítulo se detallará el proceso realizado para conseguir plasmar todo lo documentado en el GDD y cómo se ha implementado. Es el apartado con más peso de todo el trabajo, puesto que sin un correcto desarrollo no se podrá lograr un buen producto final.

En este apartado se va a hacer un amplio recorrido sobre todas las decisiones tomadas para la implementación del software requerido para desarrollar DIV. Antes de nada, se encuentra una introducción al motor elegido para el desarrollo del videojuego. Se justificará la elección de este motor, para ello también se hará una pequeña comparativa con otros motores actuales del mercado. Esta información será útil ya que todo este apartado va a tratar de la completa implementación del videojuego en dicho motor.

6.1. MOTOR DE DESARROLLO

¿Qué es un motor de videojuegos o *game engine*? Un motor de videojuegos es un conjunto de herramientas que ayudan a agilizar el proceso de desarrollo de un videojuego. Éstos proveen de herramientas al desarrollador para poder agilizar el desarrollo haciendo que no se gaste tiempo en otras tareas. Entre las herramientas que proporcionan se pueden destacar: motor de renderizado, motor de físicas y detección de colisiones, motor de sonido, inteligencia artificial y administración de memoria (Ruelas, 2017).

Para el desarrollo de este proyecto, a diferencia de en los videojuegos desarrollados en el Itinerario de Creación y Entretenimiento Digital no se ha desarrollado todo un motor de desarrollo propio con todos sus componentes debido a que es un proyecto individual y sería demasiado costoso. Por ello se ha buscado un motor que cumpla las necesidades requeridas, tales como que la plataforma objetivo va a ser dispositivos móviles y va a ser un juego en dos dimensiones; con estas características no se encuentran demasiadas alternativas.

Actualmente a nivel comercial en el sector encontramos dos grandes motores: Unity y Unreal Engine.

Ambos son dos grandes opciones en cuanto a potencia, por ello han sido elegidos para desarrollo de juegos AAA. El término ‘triple A’ se utiliza en la industria de los

videojuegos para calificar a los juegos que tienen una producción de gran envergadura y coste detrás. Algunos ejemplos de videojuegos desarrollados en estos motores son Hollow Knight y Cuphead en Unity y Fortnite y Street Fighter 5 en Unreal Engine. Ambos motores ofrecen herramientas para hacer que el desarrollo sea más fácil, dentro de toda la complejidad que este conlleva, pero cada motor cuenta con sus propias características que los hace más idóneos para unos juegos u otros.

El motor elegido para el desarrollo de este proyecto es Unity, pero ¿por qué? Principalmente porque Unity ofrece una mayor facilidad para el desarrollo de juegos en 2D y para dispositivos móviles. El entorno de desarrollo es más intuitivo y encontramos una curva de aprendizaje menor; además, en las asignaturas de Prácticas Externas 1 y 2, realizadas en [GGTech Entertainment](#) durante el anterior verano, se adquirió experiencia con este motor ya que se desarrollaron diversos minijuegos para la plataforma [Gamerdy World](#); con lo cual se puede poner en práctica algunos conocimientos adquiridos en estas asignaturas.

6.2. CARACTERÍSTICAS DE UNITY

Para realizar un correcto desarrollo se debe comprender mejor las características ofrecidas por Unity, así también se comprenderá mejor el desarrollo realizado. Unity es un motor de videojuegos multiplataforma creado por Unity Technologies, fundada en Copenhague en Mayo de 2002. Además, Unity está disponible para Windows, Mac OS, y Linux, lo que lo hace muy versátil a la hora de trabajar con él (Unity Technologies, 2022).

Sus principales características son:

- Más de 25 plataformas disponibles para crear tu videojuego, entre las más destacadas: iOS, Android, Windows, Linux, PlayStation 4, PlayStation 5, XboxOne, Nintendo Switch, Xbox One, Oculus, Google Stadia, tvOS, etc (Unity Technologies, 2022).
- Diseño y desarrollo combinados: Unity permite combinar programas de modelado tales como Blender o Maya y permite importar assets directamente sin tener que convertirlos en otro tipo de archivo, esto permite que sea más rápido cargarlos en

en el juego para tener una previsualización. Esto también está disponible para juegos 2D con herramientas como Photoshop (Unity Technologies, 2022).

- Facilidad de aprendizaje: Unity pone a disposición de los usuarios de forma gratuita más de 750 horas de sesiones de aprendizaje paso a paso, asimismo se puede encontrar un extenso manual con la [documentación](#) de cada una de las funciones de cada versión de Unity, con ejemplos de cómo usarlas (Unity Technologies, 2022).
- Licencias: Unity pone a disposición de sus clientes licencias gratuitas y de pago. Las principales diferencias es quitar la pantalla de inicio cuando comienza el juego donde sale el logo de Unity, otra característica de la versión plus es la herramienta de diagnóstico *Cloud Diagnostics Advance*, para las plataformas donde esté disponible nuestro juego. Dicha herramienta permite recibir retroalimentación de los posibles bugs o errores que se encuentran en nuestro juego y puedes elegir dónde quieras que lleguen esos reportes, ya sea a través de un correo electrónico, Discord u otro medio (Unity Technologies, 2022).

6.3. CONCEPTOS BÁSICOS DE UNITY

Tras comprender qué es Unity, se puede ahondar y comenzar a explicar a grandes rasgos cómo se lleva a cabo el desarrollo de proyectos en este motor de videojuegos. A continuación, se muestran algunas definiciones necesarias para comprender las explicaciones siguientes.

6.3.1. ESCENA

Las escenas contienen los entornos y menús del juego. Concibamos cada escena como un nivel único. En cada escena, se colocará el entorno, los obstáculos, las decoraciones, es decir, todos los objetos de ese nivel. Esto hará que el juego esté diseñado y construido en trozos (Unity Technologies, 2022).

En DIV encontramos 3 escenas: la escena inicial con el menú de selección de juego, la escena de Ninja Run y la escena de Higher & Lower.

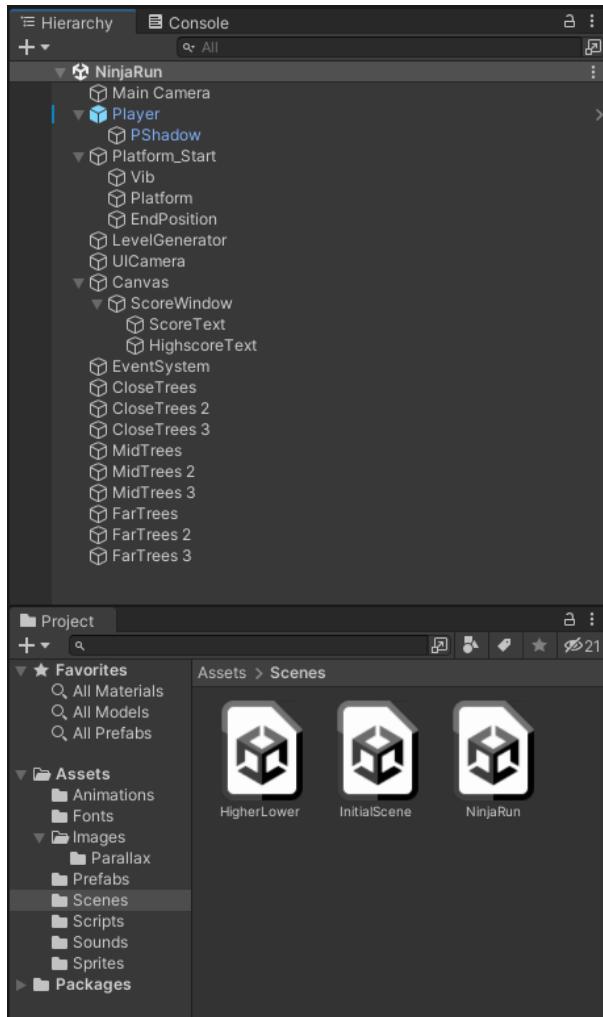


Ilustración 33 - Escenas DIV y Game Objects de Ninja Run

Fuente: Captura de elaboración propia

En la ilustración 33 podemos observar en la parte de abajo las 3 escenas del juego y en la parte de arriba todos los *Game Objects* iniciales de la escena de Ninja Run.

Cada escena puede tener diferentes concepciones, en este proyecto se han concebido como cada juego único y la pantalla de selección. Hay que entender que podemos tener diversas escenas cargadas, pero sólo se podrá tener una única escena activa simultáneamente.

6.3.2. GAME OBJECTS

Los *Game Objects* son objetos fundamentales en Unity que generalmente representan personajes, *props* y el escenario. Estos no logran nada por sí mismos, pero funcionan como contenedoras para Componentes, los cuales implementan la verdadera funcionalidad (Unity Technologies, 2022).

El *Game Object* Player de Ninja Run contiene los siguientes componentes:

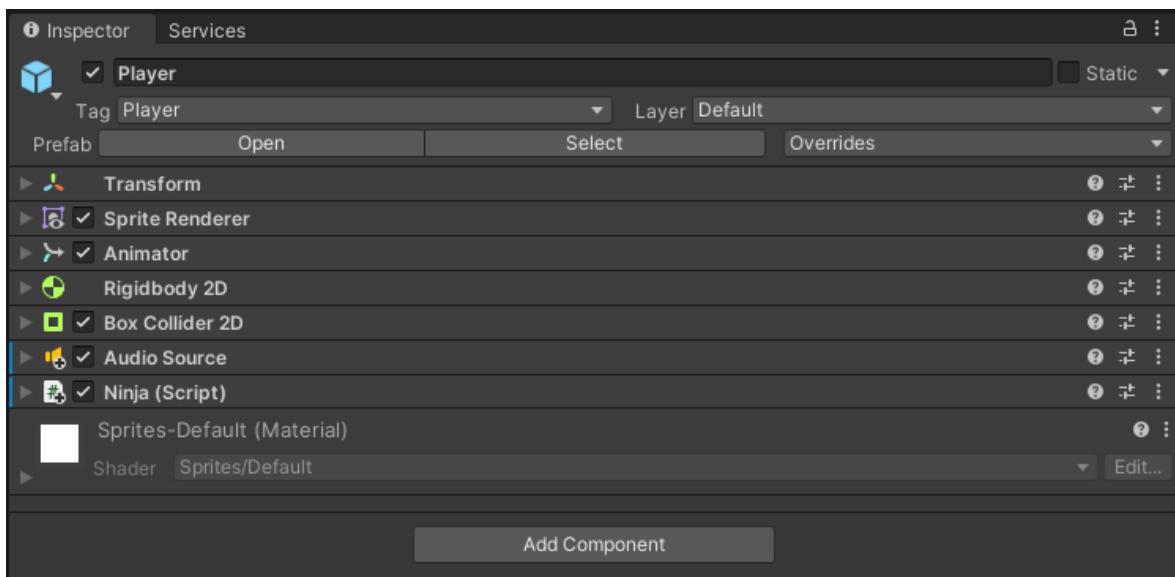


Ilustración 34 - Componentes Player Ninja Run

Fuente: Captura de elaboración propia

6.3.3. COMPONENTES

Los componentes son las tuercas y los tornillos de los objetos y comportamientos de un juego. Son las piezas funcionales de cada *Game Object*. Un *Game Object* es un contenedor de uno o varios Componentes diferentes y cada Componente dotará al *Game Object* de unas propiedades. Por defecto todos los *Game Object* tienen un componente *Transform*, el cual da la información sobre la posición del objeto, así como su rotación y su escalado.

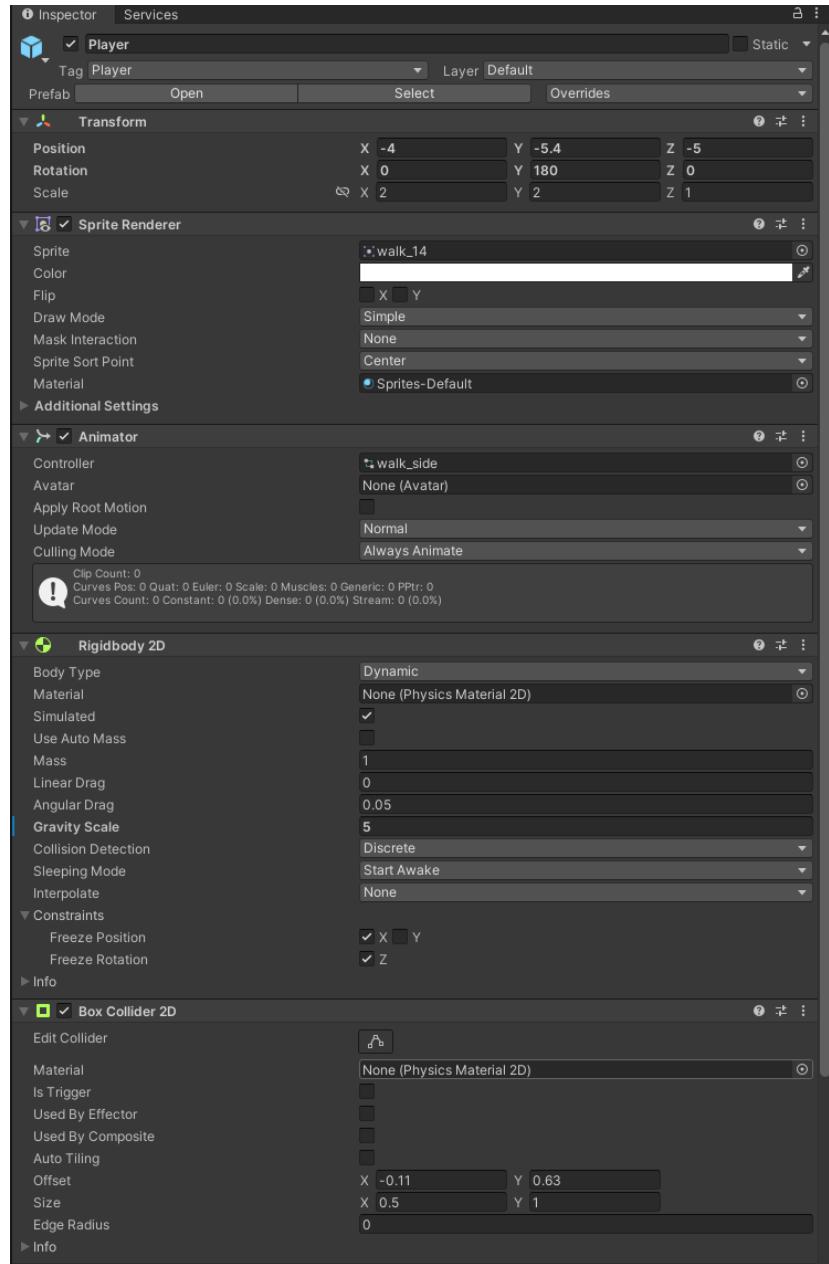


Ilustración 35 - Detalles de algunos componentes del Player de Ninja Run

Fuente: Captura de elaboración propia

En la ilustración 35 se pueden observar detalladamente algunos componentes del personaje principal de Ninja Run. Algunos componentes son el *Transform* anteriormente mencionado, el *Sprite Renderer* que contiene el *Sprite* a dibujar, el componente *Animator* para gestionar la animación del personaje, el *Rigidbody 2D* dota al personaje de características físicas tales como que le afecte la gravedad, pero que su posición X y Z nunca se vea alterada. Con el *Box Collider 2D* determinaremos la caja de colisión de

nuestro personaje y manejaremos las acciones que debe hacer cuando colisione con diversos objetos de la escena.

Como conclusión de este apartado, comprendemos que únicamente puede haber una escena activa de todas las que tengamos en nuestro proyecto. Dicha escena estará formada por diferentes *Game Objects* y éstos, a su vez, contendrán una multitud de componentes diferentes, cada uno con unas propiedades diferentes para hacer a cada objeto característico.

Con esta información básica descrita, se procede a explicar las decisiones tomadas para el desarrollo en cuestión, siguiendo el hilo de las iteraciones realizadas en cada escena.

La línea de tiempo del desarrollo ha sido la siguiente:

- **Prototipo 1:** Desarrollo de la ventana principal de DIV y la gestión de escenas entre la escena inicial, Ninja Run y Higher & Lower.
- **Prototipo 2:** Desarrollo del primer prototipo de Ninja Run.
- **Prototipo 3:** Cambio de diseño de Ninja Run, desarrollo del nuevo prototipo.
- **Prototipo 4:** Implementación de UI, fondo parallax, música, efectos de sonido y ajustes finales de Ninja Run.
- **Prototipo 5:** Desarrollo del prototipo de Higher & Lower, incluyendo sonidos e interfaz de usuario.

6.4. DESARROLLO DE DIV

En los siguientes puntos se van a tratar cada uno de los prototipos, ahondando en las características de cada uno y cómo se han desarrollado.

6.4.1. PROTOTIPO 1 – GESTIÓN DE ESCENAS DE DIV

Como ya se ha comentado previamente, las escenas en Unity tienen diversas interpretaciones y todas ellas son válidas. En este proyecto se le ha dado el uso a cada escena para cada juego, con sus *Game Objects* únicos y a su vez, sus correspondientes componentes.

El flujo de las escenas se ha detallado en cada diagrama de flujo en el GDD de cada juego, pero en resumen inicialmente se carga la escena inicial, la escena de DIV donde el usuario puede seleccionar qué juego quiere jugar. Una vez cargada la escena, a través de un *AudioSource* se reproducirá un *AudioClip*, el cual será un audio donde se le indicará al jugador que si quiere jugar a Ninja Run tiene que pulsar en la zona de la izquierda y si quiere jugar a Higher & Lower deberá pulsar en la zona de la derecha. Dependiendo de la selección, se cargará la escena de Ninja Run o la de Higher & Lower. Cuando la partida termine, automáticamente se volverá a cargar la escena inicial.

La gestión de escenas es simple, en nuestra escena inicial tenemos diversos componentes *Button*; encontramos un botón para jugar a Ninja Run y otro para Higher & Lower, la escena podrá cerrarse dándole al botón de ‘Atrás’ del dispositivo.

Cada botón tiene un método *OnClick()*, como se puede observar en la ilustración 36, mediante el cual podemos elegir qué método invocar cuando se realice la acción de pulsar sobre él, para ello se ha creado un *script* muy simple donde tenemos las funciones a invocar por dicho botón.

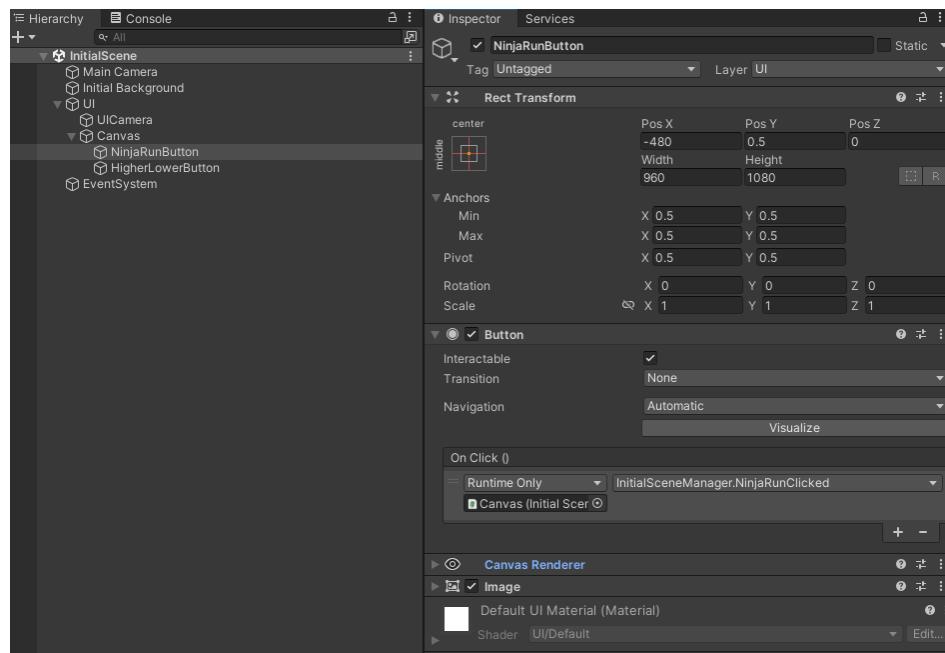


Ilustración 36 - Botón que invoca la escena de Ninja Run

Fuente: Captura de elaboración propia

Cada botón de juego, ocupa media pantalla, es un botón transparente para que se pueda ver la imagen de fondo diseñada. Como hemos podido observar en la ilustración 36 en el

componente *Button*, en el apartado *OnClick()* se le ha asociado el *Script InitialSceneManager*, el cual contiene el método *NinjaRunClicked()*.

En la ilustración 37, se muestra el *script* desarrollado, donde podemos observar que, si pulsamos tecla de Escape, la cual en dispositivos móviles se trata del botón Atrás, se cerrará la aplicación. Para cambiar de escena, simplemente hemos necesitado añadir la librería *UnityEngine.SceneManagement*. Una vez importada, simplemente invocamos el método *LoadScene(string sceneName, SceneManagement.LoadSceneMode mode)*; el *string* será el nombre de la escena que queremos cargar y el modo de carga puede ser *Single* o *Additive*. Seleccionamos *Single* porque queremos que cuando se cargue una escena, cierre la escena actual y cargue la siguiente.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using UnityEngine.UI;
6
7  public class InitialSceneManager : MonoBehaviour {
8
9      private void Update() {
10          if(Input.GetKey(KeyCode.Escape)){
11              Application.Quit();
12          }
13      }
14
15      public void NinjaRunClicked() {
16          SceneManager.LoadScene("NinjaRun", LoadSceneMode.Single);
17      }
18
19      public void HigherLowerClicked() {
20          SceneManager.LoadScene("HigherLower", LoadSceneMode.Single);
21      }
22
23  }
```

Ilustración 37 - Script para el manejo de escenas inicial

Fuente: Captura de elaboración propia

El método de *LoadScene()* será invocado también en las escenas de Ninja Run y Higher & Lower cuando se dé el fin de partida para volver a la pantalla inicial.

6.4.2. PROTOTIPO 2 – DESARROLLO DEL PRIMER PROTOTIPO DE NINJA RUN

La idea inicial de Ninja Run era implementar un *endless runner*, o en español corredor infinito. Es un género donde el jugador debe avanzar de manera irremediable en una

misma dirección y cuyo objetivo es avanzar lo máximo posible antes de morir. La acción principal era el salto para así poder esquivar los obstáculos.

Con esto se creó un prototipo donde se generaban enemigos aleatoriamente y se implementó la mecánica del salto. Para la adaptación se pretendía crear un colisionador invisible antes del objetivo a esquivar, para que así cuando colisionase con el jugador el dispositivo vibrase a modo de aviso.

Se llegó a implementar una versión muy básica, con sólo el *Sprite* del jugador, la generación aleatoria de enemigos y la gestión para eliminar las entidades, pero a medida que se jugaron diversas partidas, se llegó a la conclusión de que el juego no era divertido y era demasiado fácil morir. Si se le bajaba la dificultad haciendo que se generasen menos enemigos, el juego no aportaba variedad y era monótono, llegando a ser aburrido.

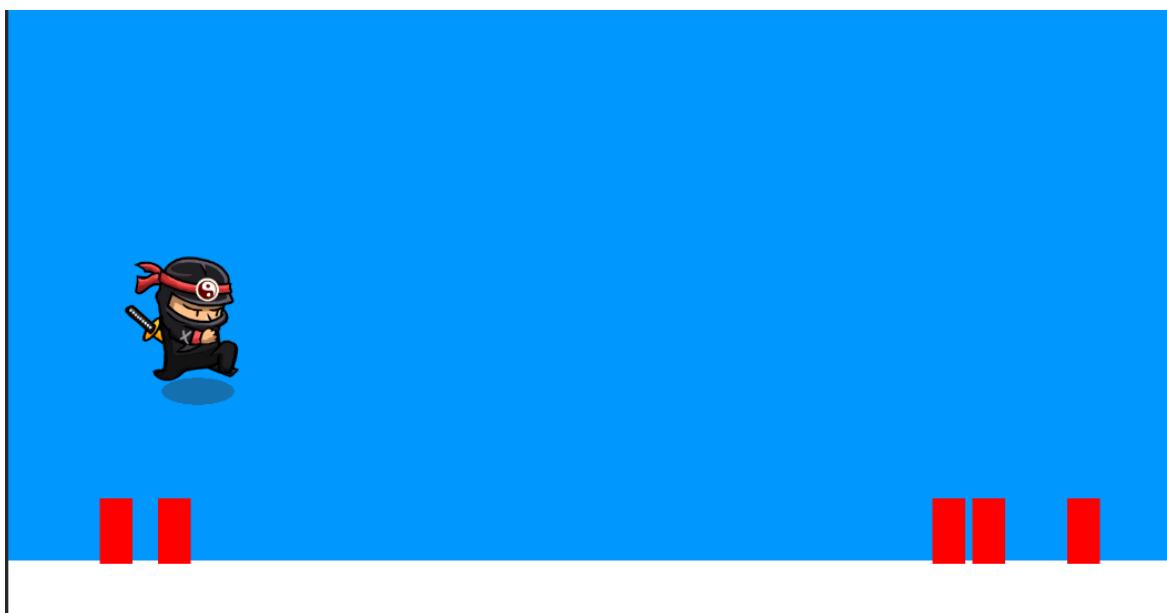


Ilustración 38 - Primer prototipo de *Ninja Run*

Fuente: Captura de elaboración propia

La única idea que se tuvo para hacerlo más divertido era añadir un doble salto para tener mayor versatilidad para esquivar obstáculos. Pero esto suponía un problema para la idea principal del juego: que estuviese adaptado para que cualquier persona pudiera jugarlo. El público objetivo para la adaptación eran personas con deficiencias visuales, si se añadía un doble salto, ¿cómo iba esta persona a saber si estaba en un salto simple o uno doble?

¿cómo se iban a controlar los avisos si una persona podía estar esquivando obstáculos con un doble salto no previsto?

Para que el juego estuviese adaptado a personas con déficit visual, el juego tenía que ser simple, no se podían añadir muchas mecánicas porque sería un lio para esa persona. Pero si el juego era simple, era aburrido. Por ello, se descartó esta idea y se rediseñó Ninja Run, aprovechando la idea principal del ninja como personaje principal y reutilizando la mecánica del salto.

La mecánica fue fácil de implementar, como se ha comentado anteriormente el jugador tiene un componente *RigidBody 2D*, el cual le aporta propiedades físicas al *Game Object*. Se creó un *Script* propio para el jugador donde, al pulsar la tecla espacio o tocar la pantalla, se le añadiría al personaje una fuerza aplicada hacia arriba, y dado que el personaje también tiene una fuerza de gravedad asignada, la actualización de la aplicación de las fuerzas hace que se forme este efecto de salto. El salto sólo se podía llevar a cabo si había tocado el suelo previamente, se controló esto con un *bool*, para no tener saltos infinitos.

```
1  private void Update() {
2      if (Input.GetKeyDown(KeyCode.Space) || Input.touchCount > 0) {
3          if(isGrounded){
4              rb.AddForce(Vector2.up * JumpForce);
5              isGrounded = false;
6              Debug.Log("Jump");
7          }
8      }
9      if(!isAlive){
10         SceneManager.LoadScene("InitialScene", LoadSceneMode.Single);
11     }
12 }
13
14 private void OnCollisionEnter2D(Collision2D collision) {
15     if(collision.gameObject.tag == "Ground") {
16         if(!isGrounded) {
17             isGrounded = true;
18         }
19     }
20     if(collision.gameObject.tag == "Enemy") {
21         isAlive = false;
22     }
23 }
```

Ilustración 39 - Extracción del código de salto que posteriormente se reutilizó

Fuente: Captura de elaboración propia

6.4.3. PROTOTIPO 3 – CAMBIO DE DISEÑO DE NINJA RUN, DESARROLLO DEL NUEVO PROTOTIPO

Ninja Run era aburrido, no se podía continuar por ese camino, pero tampoco se podía abandonar la idea de desarrollar Ninja Run. ¿Qué otra forma había de desarrollar Ninja Run manteniendo su ambientación y el género *endless runner*? ¿Para qué saltan los ninjas aparte de para sortear enemigos? Para desplazarse rápidamente.

Así que se partió de esa base, un juego de un ninja donde tenga que correr y saltar en plataformas para llegar lo más lejos posible. Se había consumido tiempo de desarrollo, pero para tener un buen producto final era necesario tomar ese cambio de rumbo, estas cosas pasan, la idea puede parecer buena pero una vez implementada salen defectos que no se habían tenido en cuenta.

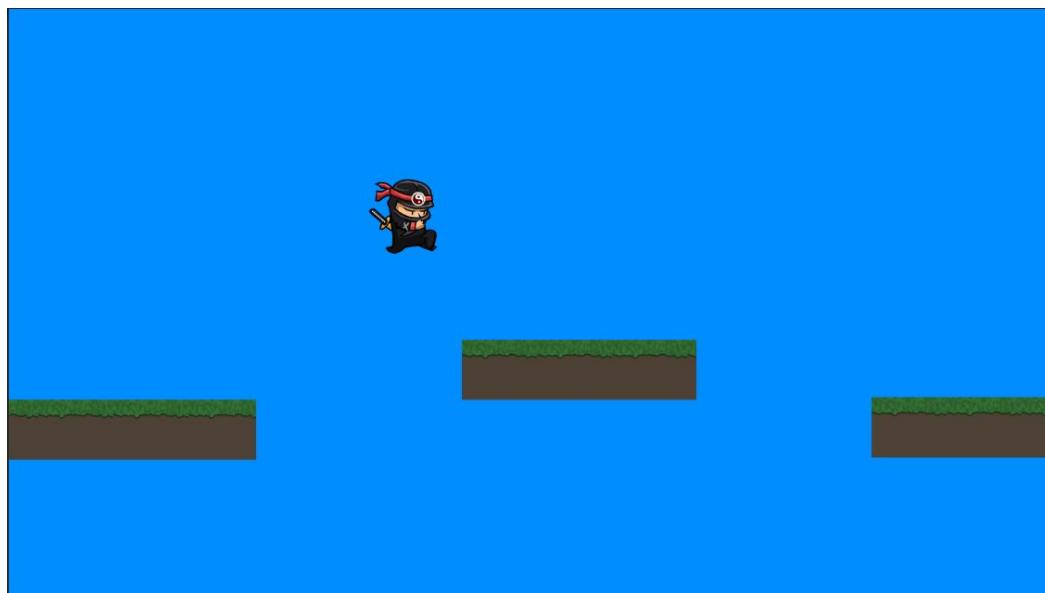


Ilustración 40 - Prototipo Ninja Run rehecho

Fuente: Captura de elaboración propia

Se rehizo Ninja Run, la mecánica del salto se reutilizó del anterior como se ha comentado, así que lo siguiente era la generación de las plataformas y comprobar que se podían instanciar.

Se crearon 3 tipos de plataformas dependiendo del nivel donde queríamos generarla, ésta se colocaría al mismo nivel que la plataforma anterior, más elevada o más abajo. Para ello se crearon 3 tipos de *Prefabs*. Un *Prefab* es un tipo de *Asset* que permite almacenar un *Game*

Object con componentes y propiedades predefinidas antes de comenzar la ejecución. El *Prefab* actúa como una especie de plantilla de un objeto, a partir de la cual se pueden crear nuevas instancias de ese objeto en la escena, sin necesidad de realizar todo el proceso desde 0, añadir los componentes y darles valores; los valores se cogerán de esa plantilla.

Los *Prefabs* de las plataformas, tenían un *script* llamado *Platform*, el cual se observa en la siguiente ilustración:

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  public class Platform : MonoBehaviour {
5      private const float PlatformMoveSpeed = 10.0f;
6      public Transform ownTransform;
7      public int type;
8
9      public Transform getPlatformTransform() => ownTransform;
10     public int getType() => type;
11
12     public void MoveToInitialPosition(Vector3 spawnPosition) {
13         ownTransform.position = spawnPosition;
14         if(type == 1){
15             ownTransform.position = ownTransform.position + new Vector3(10 , 1, 0);
16         }
17         if(type == 2){
18             ownTransform.position = ownTransform.position + new Vector3(10 , -1, 0);
19         }
20         if(type == 3){
21             ownTransform.position = ownTransform.position + new Vector3(10 , -2, 0);
22         }
23     }
24
25     public void Move(){
26         ownTransform.position += new Vector3(-1, 0, 0) * PlatformMoveSpeed * Time.deltaTime;
27     }
28
29     public void DestroySelf(){
30         Destroy(ownTransform.gameObject);
31     }
32 }
33 }
```

Ilustración 41 - Código de las plataformas

Fuente: Captura de elaboración propia

Cuando se instancia una nueva plataforma, internamente ese *Prefab* lleva asignado un tipo, la variable es pública, así que externamente en el editor de Unity se le dio el valor. La elección de la plataforma a instanciar depende del número aleatorio que se haya generado. Una vez se haya creado la plataforma, se invocará el método *MoveToInitialPosition(Vector 3 spawnPosition)*, al que se le pasará un vector de 3 posiciones con la posición donde se ha creado la plataforma; así dependiendo

del tipo de plataforma que hayamos creado, ésta se desplazará a la derecha para dejar hueco para el salto y variará su posición Y dependiendo del tipo.

Asimismo, cada plataforma tiene un método para moverse, el cual cuando se invoque generará el movimiento. Así como un método para eliminarse a sí mismas que será invocado cuando sea necesario para la correcta gestión de la memoria.

Las plataformas cuentan con dos *Game Objects* en su interior, uno denominado Vib, que es una caja de colisión invisible, la cual cuando el jugador colisione con ella se activará la vibración del dispositivo, esto se maneja con un script propio para la vibración. El otro se denomina EndPosition y es un *Game Object* cuyo componente *Transform* indica la posición final de la plataforma, esto será usado para saber cuán cerca está el jugador de la última posición de la última plataforma generada. Si está lo suficientemente cerca, se generará otra plataforma.



Ilustración 42 - Box Collider 2D invisible que activa la vibración

Fuente: Captura de elaboración propia

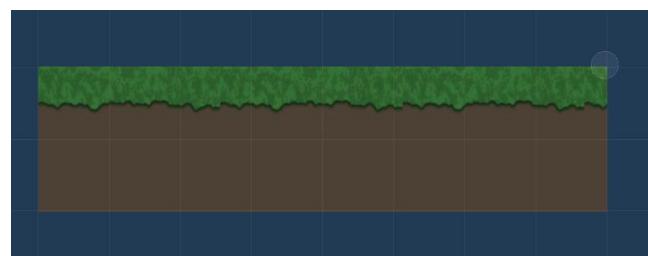


Ilustración 43 - Transform EndPosition

Fuente: Captura de elaboración propia

Cuando la caja de colisión invisible colisione con el jugador, se activa una vibración corta del dispositivo, que sirve como aviso al usuario para saber que está al final de la plataforma y debe saltar, el código asociado se muestra en la ilustración 44.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Vibrate : MonoBehaviour {
6      private void OnCollisionEnter2D(Collision2D other) {
7          if(other.gameObject.tag == "Player") {
8              //Debug.Log("Aviso");
9              Handheld.Vibrate();
10         }
11     }
12 }
```

Ilustración 44 - Colisión para que el dispositivo vibre

Fuente: Captura de elaboración propia

De la gestión de las plataformas se encarga el Level Generator, el cual es un *Game Object* que simplemente tiene el *script* que se encuentra en la siguiente ilustración:

```

1  sing System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class LevelGenerator : MonoBehaviour {
6      [SerializeField] private Transform platformStart;
7      [SerializeField] private Ninja ninja;
8      [SerializeField] private List<GameObject> platformsTypes;
9      private const float platformDestroyPositionX = -50f;
10     private const float PlayerPlatformDistance = 50f;
11     private List<Platform> platformList;
12     private Vector3 lastEndPosition;
13
14     private void Awake() {
15         lastEndPosition = platformStart.Find("EndPosition").position;
16         platformList = new List<Platform>();
17         platformList.Add((GameObject.Find("Platform_Start")).GetComponent<Platform>());
18         SpawnPlatform();
19     }
20
21     private void Update() {
22         float distance = lastEndPosition.x - ninja.getPosition().x;
23         if(distance < PlayerPlatformDistance){
24             SpawnPlatform();
25         }
26         UpdatePlatforms();
27     }
28
29     private void SpawnPlatform() {
30         GameObject chosenPlatform = platformsTypes[Random.Range(0, platformsTypes.Count)];
31         //GameObject chosenPlatform = platformsTypes[1];
32         Transform lastPlatformTransf = SpawnPlatform(chosenPlatform, lastEndPosition);
33         lastEndPosition = lastPlatformTransf.Find("EndPosition").position;
34     }
35
36     private Transform SpawnPlatform(GameObject platform, Vector3 spawnPosition) {
37         var plat = Instantiate(platform.GetComponent<Platform>());
38         plat.MoveToInitialPosition(spawnPosition);
39         platformList.Add(plat);
40         return plat.getPlatformTransform();
41     }
42
43     private void UpdatePlatforms() {
44         for(int x = 0; x < platformList.Count; x++){
45             if(platformList[x].getPlatformTransform().position.x <= platformDestroyPositionX){
46                 platformList[x].DestroySelf();
47                 platformList.Remove(platformList[x]);
48             }else{
49                 platformList[x].Move();
50             }
51         }
52         lastEndPosition = platformList[platformList.Count - 1].getPlatformTransform().Find("EndPosition").position;
53     }
54 }

```

Ilustración 45 - Script desde el cual se controla prácticamente todo el juego

Fuente: Captura de elaboración propia

Este *script* se encarga de generar las plataformas a la derecha de la pantalla, actualizar su movimiento para que parezca que avanzan y, una vez han desaparecido de la pantalla, eliminarlas.

Entrado en detalles del *script* mostrado en la ilustración 45, la primera parte es la generación de plataformas, el método `Awake()` se invoca automáticamente cuando comienza la ejecución del juego, aquí se actualiza el valor de la última posición final de la plataforma, para saber a partir de dónde se debe generar una nueva plataforma. Además,

se crea una lista donde se irán almacenando las plataformas creadas y se invoca al método para instanciar una nueva plataforma.

La segunda parte es el bucle del juego, primeramente se calcula la distancia entre la última posición de la plataforma y el jugador, para saber cuán cerca está esta plataforma del jugador y si esta es menor que el mínimo establecido para que se genere una plataforma, se invoca el método para crear una nueva plataforma `SpawnPlatform()`, cada vez que se genere una plataforma ésta se añadirá a la lista de plataformas, por último se actualiza el movimiento de las plataformas.

Para actualizar el movimiento de las plataformas, se recorrerá la lista de plataformas creadas. Primero se comprueba que no haya salido de la pantalla, de ser así, se invoca su método para que se destruya a sí misma y se elimina de la lista, esto ayudará a la gestión de memoria y que el dispositivo no se sature con miles de instancias de plataformas. Seguidamente, si la plataforma se encuentra en la parte visible del juego, se invoca su método propio de movimiento. Por último, se actualiza el valor de la posición de la última plataforma, se recuerda que este valor es el utilizado para calcular si se debe crear una nueva plataforma o no.

6.4.4. PROTOTIPO 4 – AJUSTES, MEJORAS VISUALES Y SONORAS DE NINJA RUN

6.4.4.1. AJUSTES DE NINJA RUN

El primero de los cambios realizados en esta iteración fue un simple renombre, el *script* de LevelGenerator pasó a llamarse LevelManager, ya que debido a sus funcionalidades el nombre era más acertado.

Los cambios realizados en el LevelManager afectaron a una correcta instanciación de las plataformas, haciendo que si el jugador estaba en la parte baja de la pantalla, no se creasen más plataformas hacia abajo; se siguió la misma lógica cuando el jugador se situaba en la parte de arriba.

Para la selección de la instanciación de las plataformas, se creó una lista con los diversos *Prefabs* existentes de las plataformas, más de uno se repitió para jugar con la probabilidad y favorecer más la creación de unas plataformas u otras. En la ilustración 46 se puede ver el método de `SpawnPlatform()` modificado, así como la ilustración 48 muestra un

esquema sobre el que se va a apoyar la explicación para que quede más claro su funcionamiento.

```

1  private void SpawnPlatform() {
2      GameObject chosenPlatform;
3      float ninjaPosY = ninja.getPosition().y;
4
5      if(ninjaPosY <= (-6)){ //Ninja down spawn plat 1 y 2
6          chosenPlatform = platformsTypes[Random.Range(0, 2)];
7      }
8      else if(ninjaPosY >= 1.95){ //Ninja up spawn plat 2 y 3
9          chosenPlatform = platformsTypes[Random.Range(2, 5)];
10     }
11     else{ //Ninja middle, spawn plat 1, 2 y 3
12         chosenPlatform = platformsTypes[Random.Range(1, 4)];
13     }
14     Transform lastPlatformTransf = SpawnPlatform(chosenPlatform, lastEndPosition);
15     lastEndPosition = lastPlatformTransf.Find("EndPosition").position;
16 }
```

Ilustración 46 - Spawn Platform actualizado

Fuente: Captura de elaboración propia

Los elementos de la lista se asignaron externamente en el editor de Unity, como se muestra en la ilustración 47, de forma que cuando comenzase la ejecución del juego ya estuviese preparada para ser utilizada.

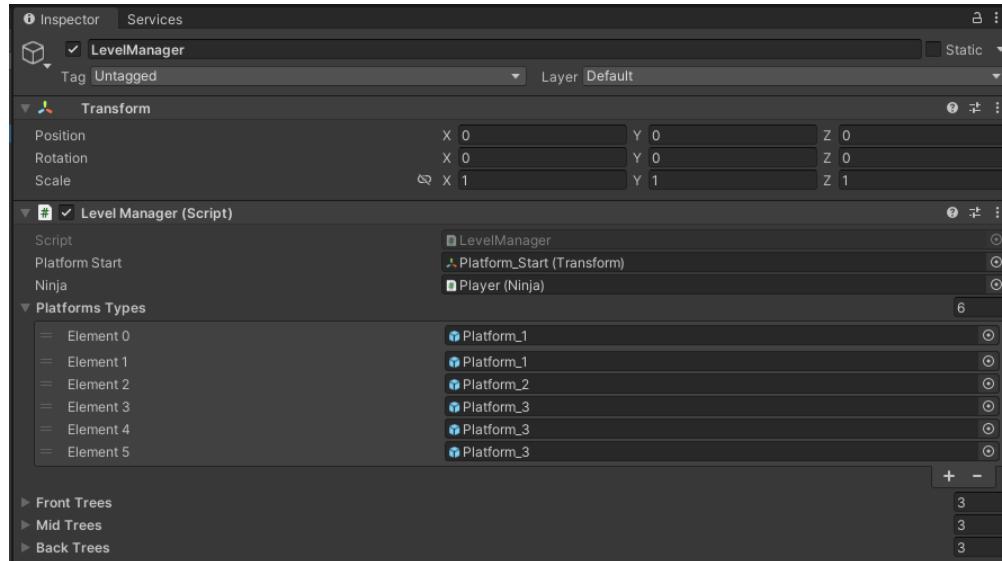


Ilustración 47 - Lista de tipos de plataformas en el editor de Unity

Fuente: Captura de elaboración propia

Pero lo más importante, ¿cómo funciona esto? ¿cómo afecta que haya elementos repetidos? Para explicar esto, como se ha mencionado antes se va a utilizar el siguiente

esquema de la ilustración 48 para utilizarlo como apoyo y comprenderlo de una forma más fácil asimilándolo visualmente.

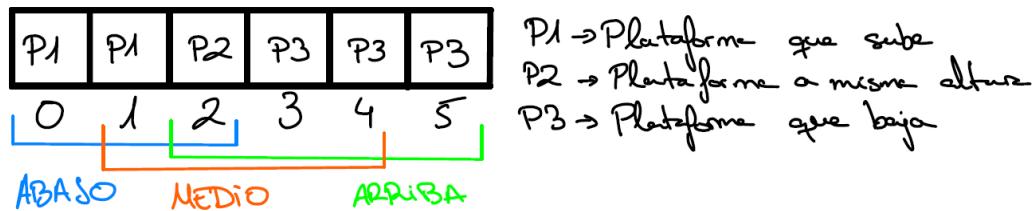


Ilustración 48 - Esquema que ilustra la lista de plataformas para comprender la probabilidad de cada casuística

Fuente: Elaboración propia

Existe una lista de 6 elementos, cada rango de elementos va a ser utilizado para una casuística diferente; la casuística en cuestión depende de la posición Y del jugador. Si el jugador está en una posición por debajo de -6 unidades, el jugador estará por la parte de abajo; sin embargo, si está en una posición mayor a 1.95 unidades, se considerará que el jugador está arriba. Si el jugador no está ni por arriba ni por abajo, se considerará que está en medio de la pantalla. Cada casuística funciona de la siguiente manera:

- Jugador situado en la parte baja: Se generará un número aleatorio entre 0 y 2, ya que son las plataformas que se deben crear, si el jugador está debajo no se pueden crear plataformas que vayan hacia abajo porque saldría de la pantalla. ¿Por qué hay dos plataformas de tipo 1? Para modificar la probabilidad. No se quiere que la probabilidad sea un 50/50, si el jugador está abajo lo que va a dar más variedad es que el jugador suba, por eso se añade una segunda plataforma que va hacia arriba, de tal forma las probabilidades en esta casuística queda tal que: 66'6% de probabilidades de que el jugador suba y un 33'3% de probabilidades de seguir recto.
- Jugador situado arriba: El número aleatorio generado será entre 2 y 5. Las probabilidades de instanciación de las plataformas son: 25% de probabilidad de ir recto y 75% de probabilidades de bajar. Siguiendo la misma lógica que anteriormente, se prefiere que el jugador baje.

- Jugador situado en medio: El numero aleatorio generado será entre 1 y 4. Las probabilidades quedan tal que: 25% de probabilidad de subir, 25% de probabilidad de ir recto y 50% de probabilidad de bajar.

Si se observan las probabilidades, el objetivo de éstas es que el jugador siempre acabe situado más bien por la zona de abajo, esto es una decisión de diseño, ya que se observó que, si el jugador estaba demasiado por la zona de arriba, se podría generar alguna plataforma que se viese y donde el jugador pudiese saltar, pero al realizar el salto el personaje saldría de pantalla o podría cortarse alguna parte de su *sprite* y puede ser molesto y confuso. Con estas probabilidades generalmente por muy alto que se generen las plataformas, no llegarán demasiado alto como para hacer que el jugador desaparezca saltando.

6.4.4.2. IMPLEMENTACIÓN DE INTERFAZ DE USUARIO

Tras estos cambios, se implementó la interfaz de usuario. Para ello se crearon diversos *Game Objects*, comenzando por un Canvas y una Cámara asociada a éste. Esta cámara renderizará todos los elementos incluidos en el Canvas. Como hijo, se creó un *Game Object* vacío (Score Window), el cual contiene el *script* NinjaScoreWindow, el cual se encarga de actualizar el texto en pantalla. Se muestra en la ilustración 47. Sucesivamente a éste, se crearon dos *Game Objects*, cada uno de ellos maneja un apartado, uno es para la puntuación máxima y el otro para la puntuación actualizada durante la partida.

Como se puede observar en la ilustración 50, el *Game Object* está compuesto por un *Rect Transform* encargado de la posición, el alto y el ancho de la zona donde irá el texto. Asimismo, cuenta con un componente *Text* en el cual podemos definir un texto base, también se puede configurar la fuente, el estilo de ésta, su tamaño, la alineación, el color y demás opciones. Todo esto se realiza desde el editor de Unity.

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.UI;
6
7  public class NinjaScoreWindow : MonoBehaviour{
8      private Text scoreText;
9      private Text highscoreText;
10
11     private void Awake() {
12         scoreText = transform.Find("ScoreText").GetComponent<Text>();
13         highscoreText = transform.Find("HighscoreText").GetComponent<Text>();
14     }
15
16     private void Update() {
17         scoreText.text = "Puntuación: " + (LevelManager.GetInstance().getScore()).ToString();
18         highscoreText.text = "Récord: " + PlayerPrefs.GetInt("Highscore").ToString();
19     }
20 }

```

Ilustración 49 - Script para actualizar el texto en pantalla

Fuente: Captura de elaboración propia

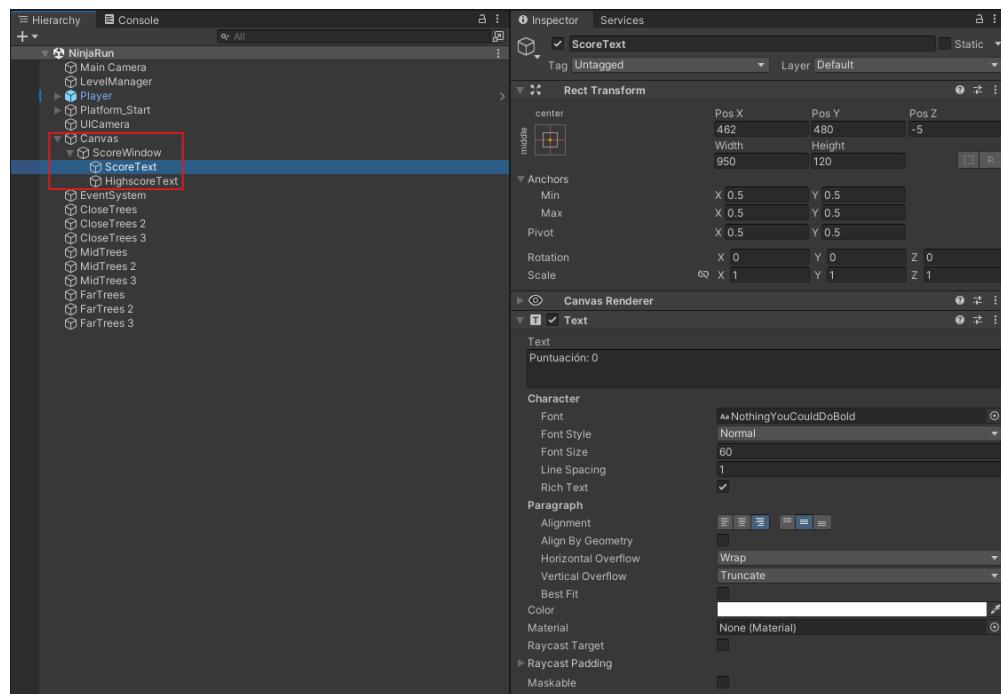


Ilustración 50 - Gestión de la interfaz de usuario en el editor de Unity

Fuente: Captura de elaboración propia

Cuando empieza una partida se almacena la puntuación máxima. En el transcurso de esta se va actualizando la puntuación cuando el jugador colisiona con la plataforma invisible. Si se supera el récord, se guarda la nueva puntuación máxima, todo esto se puede observar en la ilustración 51. Y como se ha observado en la ilustración 49, el valor de la puntuación máxima se obtiene del valor guardado en las preferencias del jugador, con lo cual se actualizará el texto a la par que el de puntuación.

The screenshot shows two open scripts in the Unity Editor:

- Vibrate.cs** (Assets > Scripts > Vibrate.cs):


```
5 public class Vibrate : MonoBehaviour {
6     private void OnTriggerEnter2D(Collider2D other) {
7         if(other.gameObject.tag == "Player") {
8             Handheld.Vibrate();
9             LevelManager.GetInstance().UpdateScore();
10    }
11 }
12 }
```
- LevelManager.cs** (Assets > Scripts > LevelManager.cs):


```
54 public void UpdateScore() {
55     score++;
56     if(score > oldHighScore){
57         PlayerPrefs.SetInt("Highscore", score);
58         PlayerPrefs.Save();
59     }
60 }
61 }
```

Ilustración 51 - Actualización de la puntuación

Fuente: Captura de elaboración propia

6.4.4.3. IMPLEMENTACIÓN EFECTO PARALLAX SCROLLING

El último apartado visual que se incluyó fue el efecto *parallax scrolling* para el fondo. Se ha explicado su funcionamiento en el GDD a nivel teórico, se va a desarrollar cómo se ha implementado esta idea en Unity.

Como podemos observar también en la ilustración 50, para cada capa se creó un *Game Object* y se cargó en ella el *Sprite* correspondiente, las 3 capas se crearon directamente en el editor. Este proceso se repitió, teniendo así 3 veces cada uno de los *Game Objects* correspondiente a cada capa y se pusieron uno a continuación del otro. Se triplicaron porque no se puede notar el corte entre una capa y otra, así cuando una capa se moviese, la siguiente le seguiría siendo imperceptible el salto. Se triplicó porque así cuando una zona de capas ha salido de la cámara, se mueve a la posición final, mientras las siguientes entran en escena. Con esto conseguimos un bucle perfecto donde no se aprecian cortes algunos. Los cortes entre zonas son los señalados en la ilustración 49.



Ilustración 52 - Cortes de zonas de capas

Fuente: Captura de elaboración propia

A cada *Game Object* se le asoció un *script* encargado del movimiento. En dicho *script* se le asoció a través del editor el componente *Transform* para gestionar su movimiento, así

como la asignación de la velocidad de desplazamiento. Esta velocidad es diferente para cada capa.

- Árboles cercanos: Son los que más rápido deben moverse, tienen una velocidad de 10 unidades.
- Árboles a media distancia: Deben tener una velocidad intermedia entre los árboles cercanos y los lejanos, tienen una velocidad de 5 unidades.
- Árboles lejanos: Deben tener una velocidad baja para causar un correcto efecto, tienen una velocidad de 2 unidades.

Todo lo comentado anteriormente se puede observar en la ilustración 53.

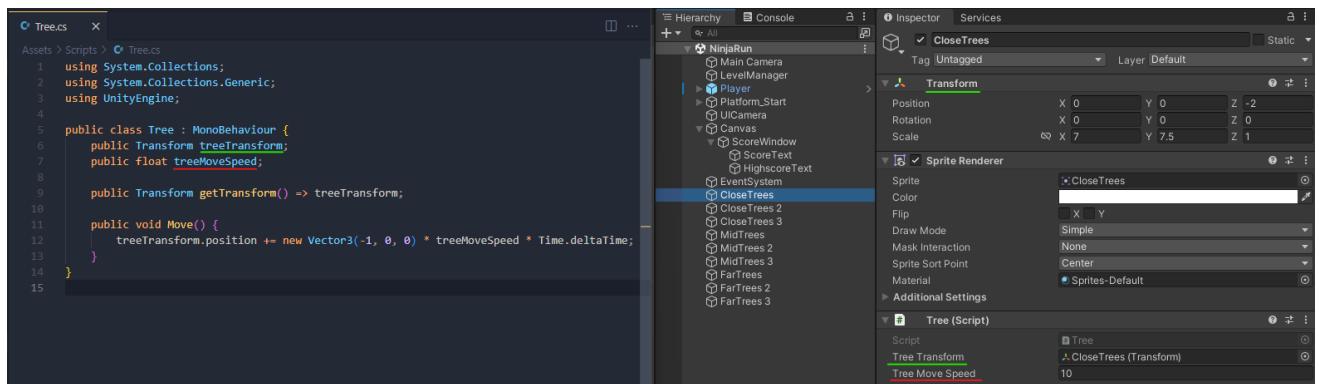


Ilustración 53 - Ejemplo de un Game Object de una capa, con su script asociado

Fuente: Captura de elaboración propia

Con esto se controla todo el movimiento independiente de cada árbol, pero hay que invocar el método `Move()` y gestionar el hecho de que cuando una capa salga de pantalla, se coloque correctamente al final de la última capa de ese tipo. Todo esto se realiza desde el `LevelManager`, a través del método `UpdateTrees()` como podemos observar en la ilustración 54.

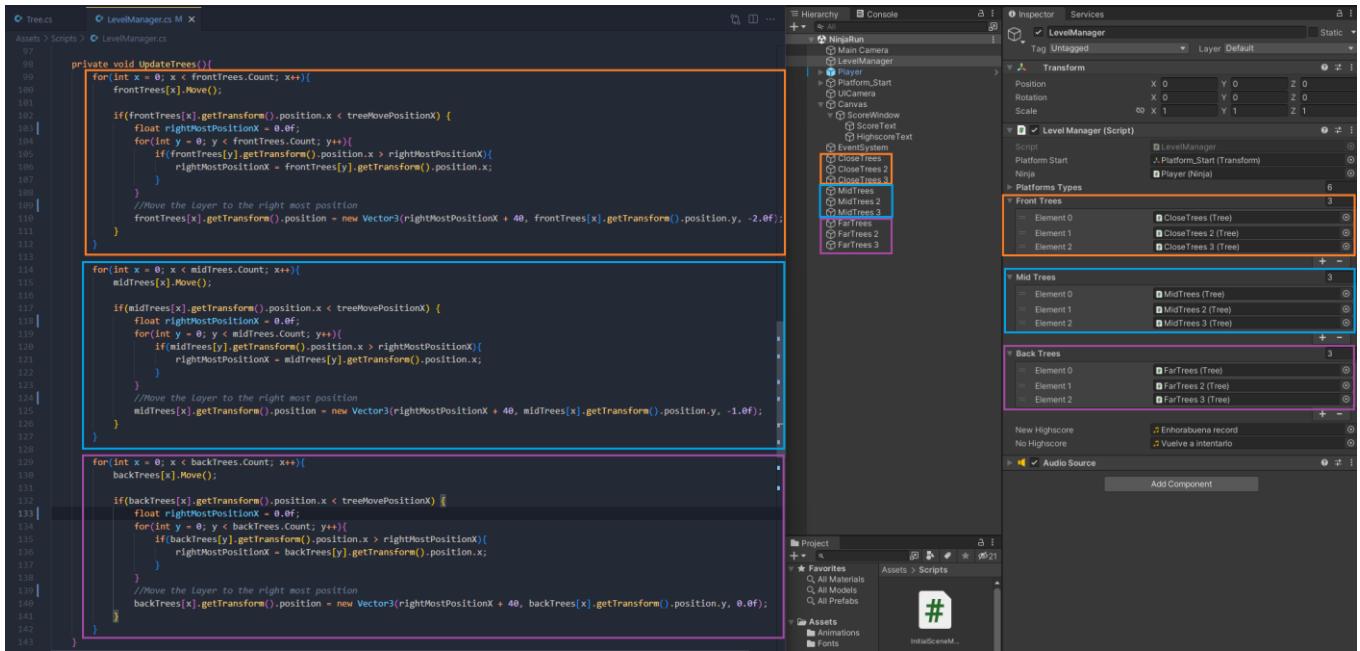


Ilustración 54 - Actualización de movimiento de cada capa

Fuente: Captura de elaboración propia

Se crearon 3 listas diferentes, una para cada grupo dependiendo de la distancia de la capa, teniendo así Front Trees para las capas cercanas, Mid Trees para las capas a media distancia y Back Trees para las capas más lejanas. Se le asignaron los valores a través del editor de Unity.

En el bucle del juego, se invocaba el método `UpdateTrees()`, este método sigue la misma lógica para cada lista, donde la única diferencia en el código es la lista que se está recorriendo y los elementos de la misma, por ello se va a describir el proceso una única vez.

Para cada elemento de la lista se invoca el método `Move()` visto en la ilustración 53. Tras ello, se comprueba si tras haberse desplazado queda fuera de la pantalla, comparándolo con una constante que tiene almacenado el valor a partir del cual se considera que está fuera. En caso de que se haya salido de la pantalla, se vuelve a recorrer la misma lista para saber cuál es la posición de la capa más lejana. Una vez averiguada, se traslada la capa a esa posición, sumándole 40 unidades a la posición X ya que la posición obtenida es la del centro del objeto y no se quiere desplazar la capa al centro de la otra, sino al final. En caso de no haber salido de la pantalla, no se hace nada más. Como se ha comentado anteriormente este método se realiza para cada lista de capas.

6.4.4.4. INCLUSIÓN DE MÚSICA Y EFECTOS DE SONIDO

Finalmente, se implementó el apartado sonoro, se procede a hablar de la implementación del sonido del salto, la música del juego y por último las voces al finalizar la partida.

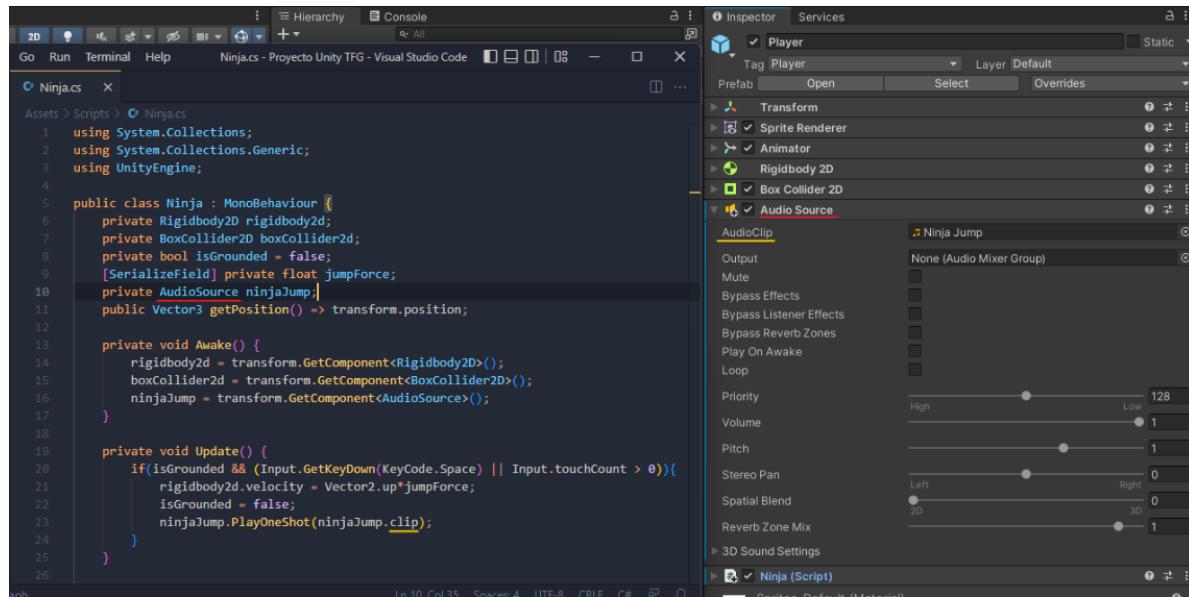


Ilustración 55 - Implementación sonido del salto

Fuente: Captura de elaboración propia

Para implementar el sonido del salto, en el *Game Object* Player, que es el contenedor de todo lo relacionado con el ninja, se añadió un componente *AudioSource* y en el apartado *AudioClip* se asignó el sonido del salto. Tras ello se añadió que cuando saltase, se reprodujese una única vez este clip. Esta implementación se refleja en la imagen 53.

El resto de los sonidos se implementaron en el *LevelManager*, como se puede observar en la ilustración 54, al igual que con el ninja, se necesitaba un componente *AudioSource* y en este caso el clip asociado sería la música de fondo, están marcadas las opciones *Play On Awake* y *Loop* para que nada más cargar la escena se empiece a reproducir la música y lo haga en bucle.

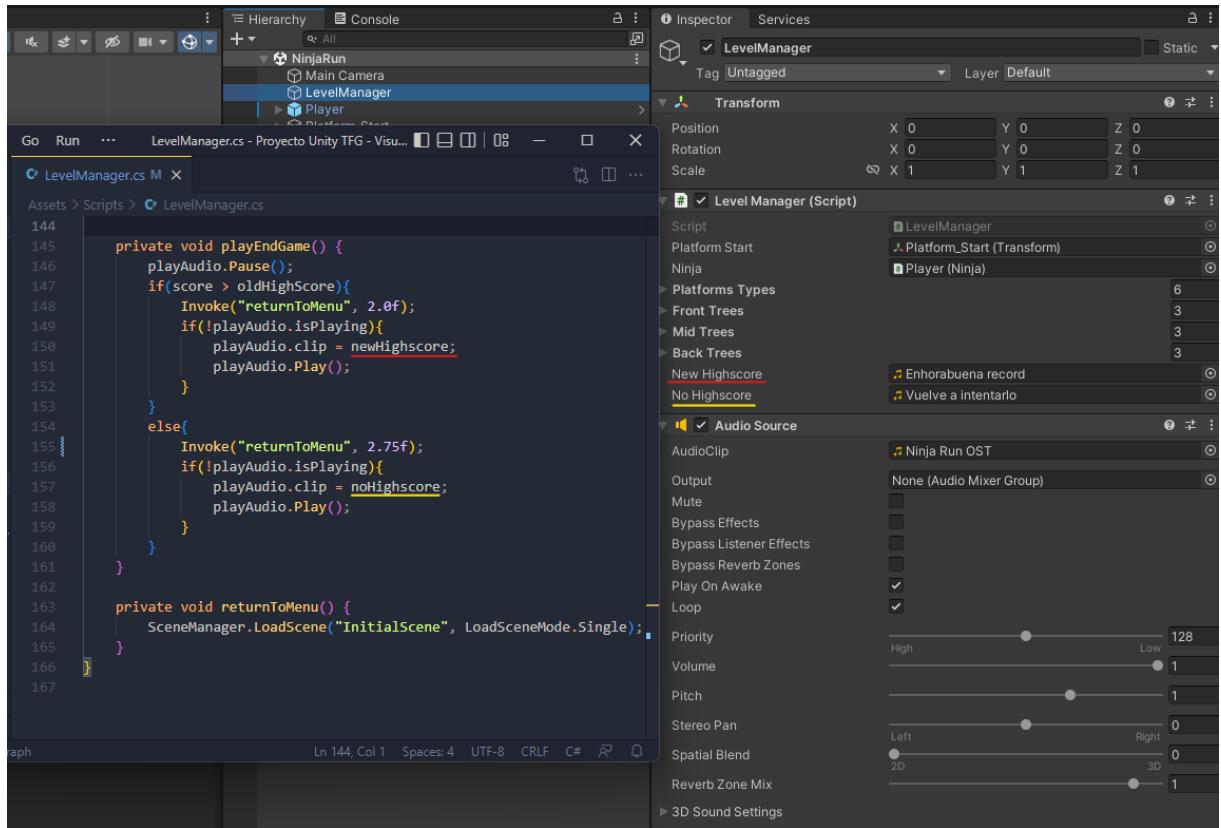


Ilustración 56 - Música de fondo y sonidos de fin de juego

Fuente: Captura de elaboración propia

En el Level Manager se cargan también los dos *AudioClips* correspondientes a las frases de fin de juego. Cuando finaliza la partida se llama al método `playEndGame()`, el cual se encarga de reproducir la música y gestionar la vuelta a la pantalla inicial. El funcionamiento es el mismo ya hayas batido el récord o no, lo único que variará es el clip que se va a reproducir. Primeramente, se pausa la música de fondo. Tras ello se comprueba si se ha superado la puntuación máxima que había al iniciar la partida o no. Se hace la llamada al método `returnToMenu()`, pero no se hace como se está acostumbrado, sino que se hace a través del método `Invoke(string methodName, float time)`, el cual permite realizar la llamada a un método tras los segundos indicados, en este caso, 2. Tras ello, se hace la comprobación de que si no hay ningún audio ejecutándose, se asigne el clip correspondiente y se reproduzca.

¿Por qué la comprobación de que no se esté reproduciendo ningún audio? Porque en el tiempo transcurrido entre la invocación de la vuelta al menú hasta que se hace efectiva, se

sigue ejecutando el bucle del juego, con lo cual se volvería a reproducir el sonido una y otra vez causando un estruendo.

Con todo esto, se da por finalizado el desarrollo de Ninja Run.

6.4.5. PROTOTIPO 5 – PROTOTIPO DE HIGHER & LOWER

Por último, se creó el prototipo de Higher & Lower, gracias a la experiencia adquirida tras el desarrollo de los anteriores prototipos este desarrollo fue mucho más ágil.

6.4.5.1. GESTIÓN Y REPRODUCCIÓN DE SONIDOS E IMPLEMENTACIÓN DE LA VIBRACIÓN

Se ha creado un *Game Object* por cada sonido que se va a reproducir en el juego, cada *Game Object* va llevar asociado un *script* Sound mostrado en la ilustración 57. En dicho *script* se almacenará el *Audio Source* del sonido en cuestión, así como la frecuencia que le vamos a asociar. Estos parámetros se asignarán manualmente en el editor de Unity. Posteriormente dicha frecuencia se utilizará para hacer que el dispositivo vibre más o menos.

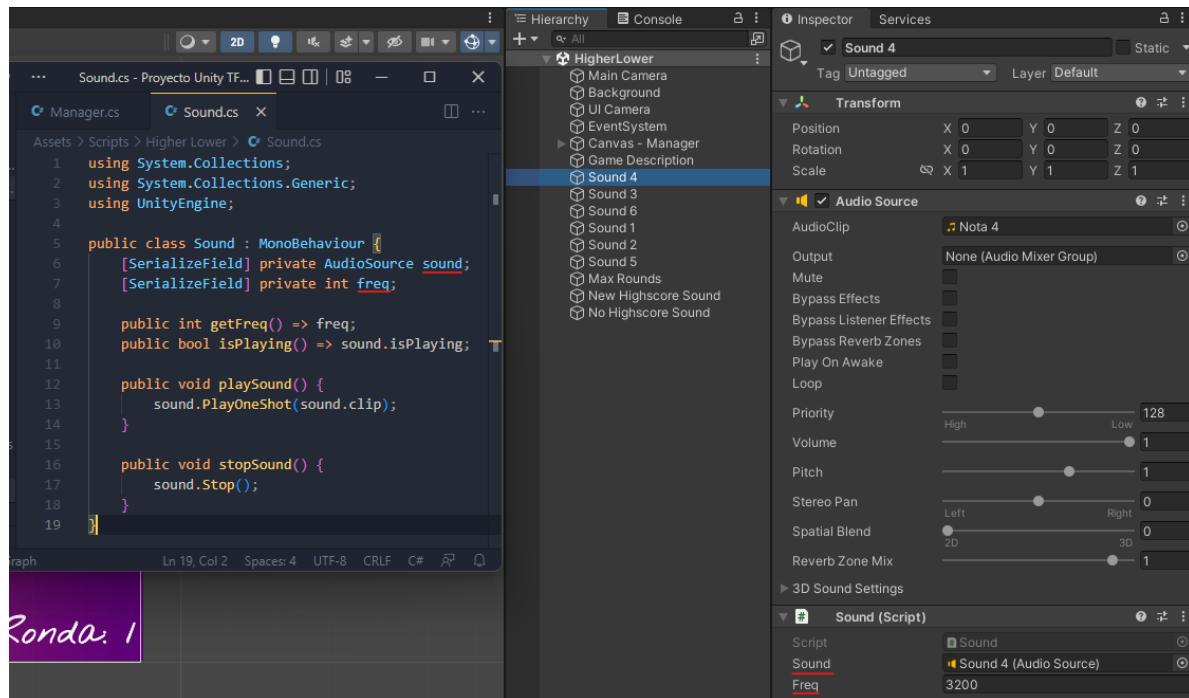


Ilustración 57 - Script encargado de la gestión de cada sonido y asignación de valor en Unity

Fuente: Captura de elaboración propia

Una vez creados todos los sonidos que iban a ser utilizados, se creó un Level Manager y en él se crearon dos listas diferentes: una lista de sonidos, la cual contendría las notas que serían reproducidas durante el juego y una lista de los sonidos de las voces, donde se localizarían las voces que suenan al final del juego e indican si el jugador ha llegado al final de rondas, si ha superado el récord o si no ha podido superarlo. Cuando se inicie el juego se reproducirá un sonido que no se almacena en ninguna lista, ya que como sólo va a ser reproducido al principio para explicarle al jugador cómo jugar, no necesita ser gestionado a través de ningún *script*. Al igual que con la música de Ninja Run, se ha marcado el check de *Play On Awake* para que se reproduzca automáticamente al inicio.

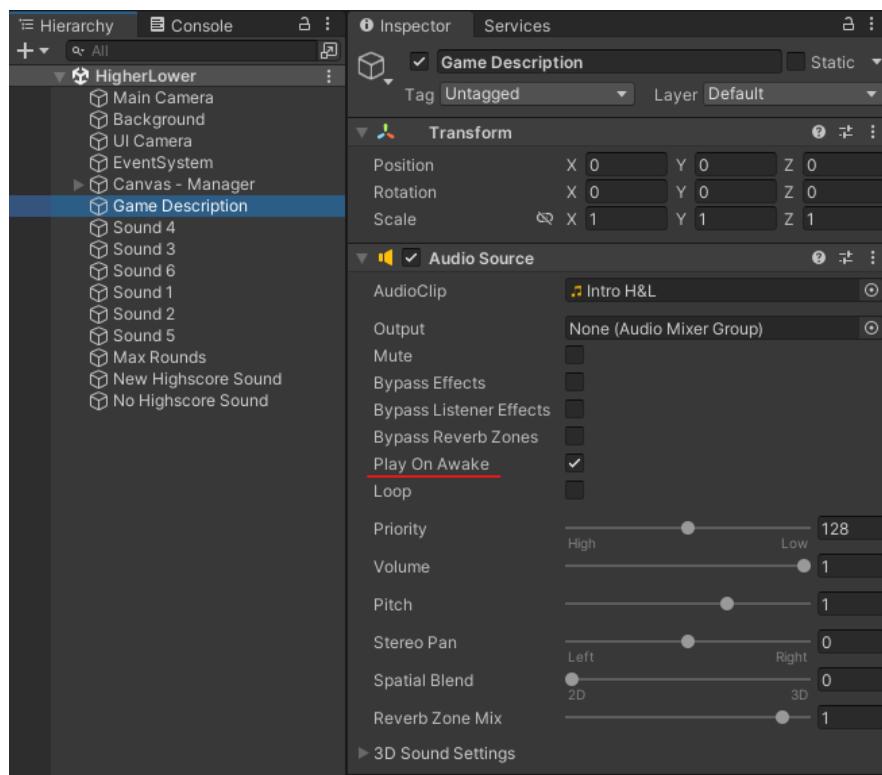


Ilustración 58 - Reproducción automática de la descripción al inicio del juego

Fuente: Captura de elaboración propia

Tras la reproducción de este audio, comenzará realmente el juego; como se observa en la ilustración 59 lo primero que se hará es mezclar aleatoriamente la lista de sonidos para que cada partida sea diferente. Para ello cuando empecemos el juego, el método de *shuffle* será llamado a través del método *Invoke()*. Así como el audio de la descripción dura 14 seg, el método se llamará 14 segundos después de iniciar el juego. La

lista se mezclará 3 veces, para evitar que algún orden de números se genere igual en alguna ejecución y esté realmente mezclada.

Una vez finalizada la mezcla, se invocará el método playFirstSound() el cual reproducirá el primer sonido de la ronda, hará que el dispositivo vibre con la frecuencia que le hemos asociado y 3 segundos después, que es la media de duración de cada sonido, se invocará la reproducción del segundo sonido.

Para saber qué posición debe reproducir, se empleará una variable que indica el elemento primario a reproducir, inicialmente la variable vale 0, con lo cual en la primera ronda se reproducirá soundsList[0] y el segundo sonido será soundList[0+1]; cuando se pase de ronda el valor de la variable se incrementará en 1. Así el segundo sonido que se ha escuchado anteriormente pasa a ser el primero.

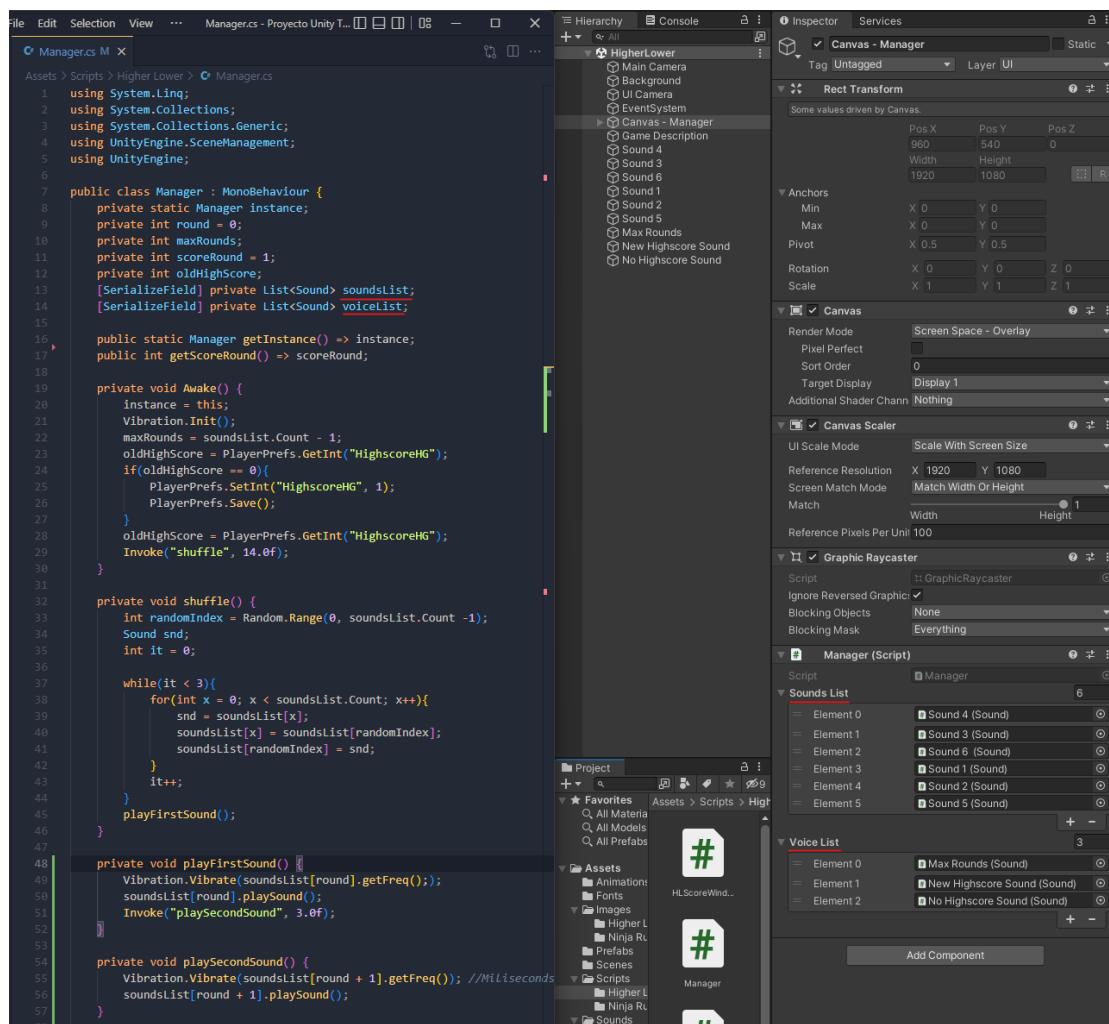


Ilustración 59 - Gestión de sonidos y reproducción de estos en cada ronda

Fuente: Captura de elaboración propia

Dado que Higher & Lower trata sobre detectar si un sonido es más grave o agudo que otro, el público objetivo para la adaptación son personas con deficiencias auditivas. Para llevar a cabo la adaptación se recurrió de nuevo a la vibración del dispositivo como medio para transmitirle la información al usuario. Para ello los sonidos que son más agudos, al tener una mayor frecuencia harían que el dispositivo vibrase más y en oposición, los graves harían que vibrase menos.

Desafortunadamente, Unity de manera nativa no tiene un método para manejar cuánto tiempo puedes activar la vibración, tiene un método predefinido para hacerlo vibrar, pero es siempre la misma vibración. Esto sirvió para Ninja Run pero no para Higher & Lower.

Para solventar esto la solución que se encontró fue un plugin gratuito desarrollado para Unity por el usuario Benoit Freslon en GitHub, mediante el cual se podían crear vibraciones personalizadas para los dispositivos móviles (Freslon, 2022). Gracias a esto, incorporando en nuestro proyecto el script 'Vibration' se podía hacer que el dispositivo vibrase la cantidad de tiempo que se quisiese para cada sonido.

Para ello sólo había que inicializar el plugin al principio con `Vibration.Init()` y para crear una vibración personalizada, se invocaba le método `Vibration.Vibrate(int milliseconds)`. Como a cada sonido se le podía asignar un valor frecuencia, este valor serían los milisegundos que vibraría el dispositivo. Por ello se asignaron valores de menor a mayor desde los sonidos graves a los agudos.

6.4.5.2. IMPLEMENTACIÓN DE LA INTERACCIÓN Y LA INTERFAZ DE USUARIO

Una vez creados y gestionados los sonidos, sólo faltaba implementar la interfaz para que el usuario pudiese interaccionar con el juego. Se partió de la misma idea que en la pantalla inicial, tener dos botones que ocupasen media pantalla cada uno. El de la izquierda servirá para responder que el sonido es más agudo y el de la derecha para responder que es más grave.

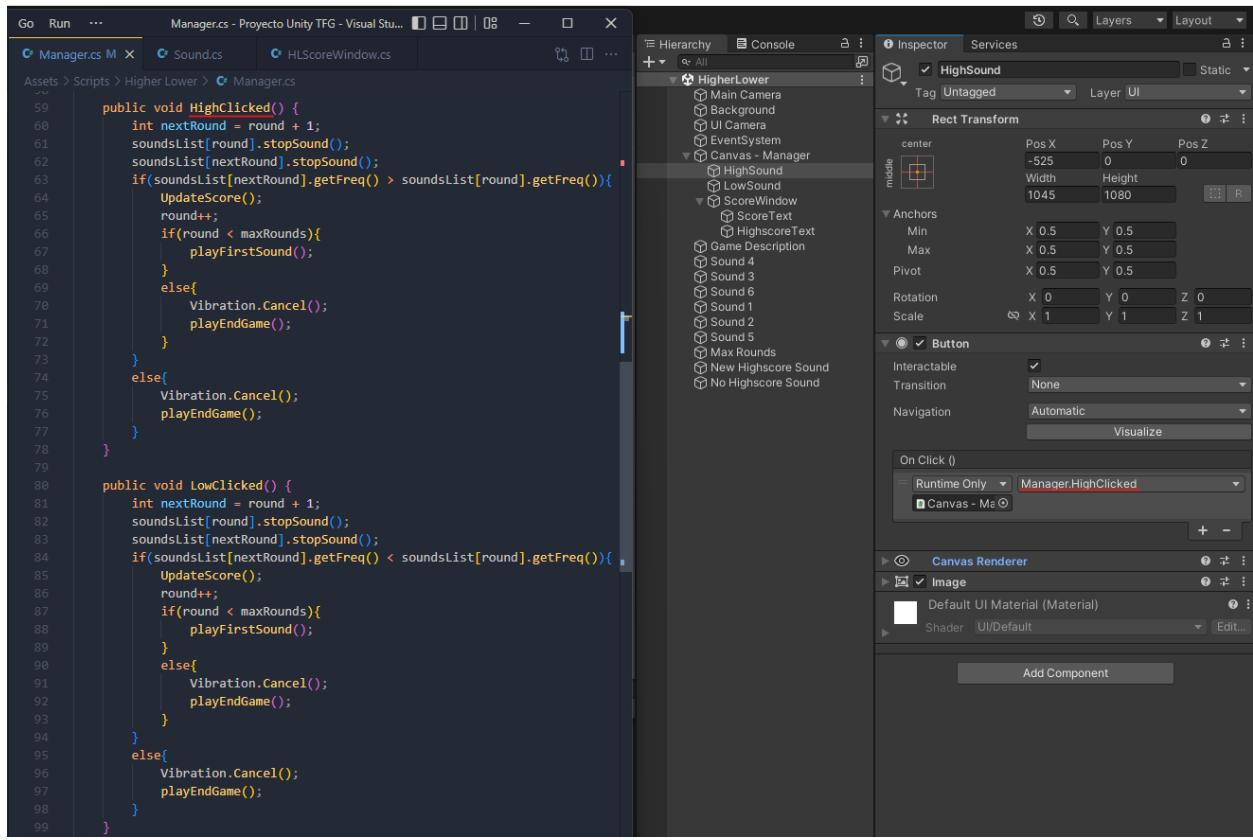


Ilustración 60 - Acción botones agudo y grave

Fuente: Captura de elaboración propia

Como se observa en la ilustración 60, cuando se pulsa el botón High, se invoca el método `HighClicked()`, esto corresponde a la respuesta agudo por parte del usuario. En este método se comparan las frecuencias asociadas a cada sonido y como ya se ha tenido en cuenta esta comparativa, los sonidos agudos tendrán mayores frecuencias.

Si efectivamente la segunda frecuencia es mayor que la primera, se considera que ha acertado, con lo cual se actualizará la puntuación y se pasará a la siguiente ronda y se volverá a hacer el mismo proceso antes mencionado para reproducir los sonidos. Si se ha llegado al máximo de rondas o si el jugador falla, se invocará `playEndGame()` donde se reproducirá el audio correspondiente como se puede observar en la imagen 61 y se volverá a la pantalla inicial.

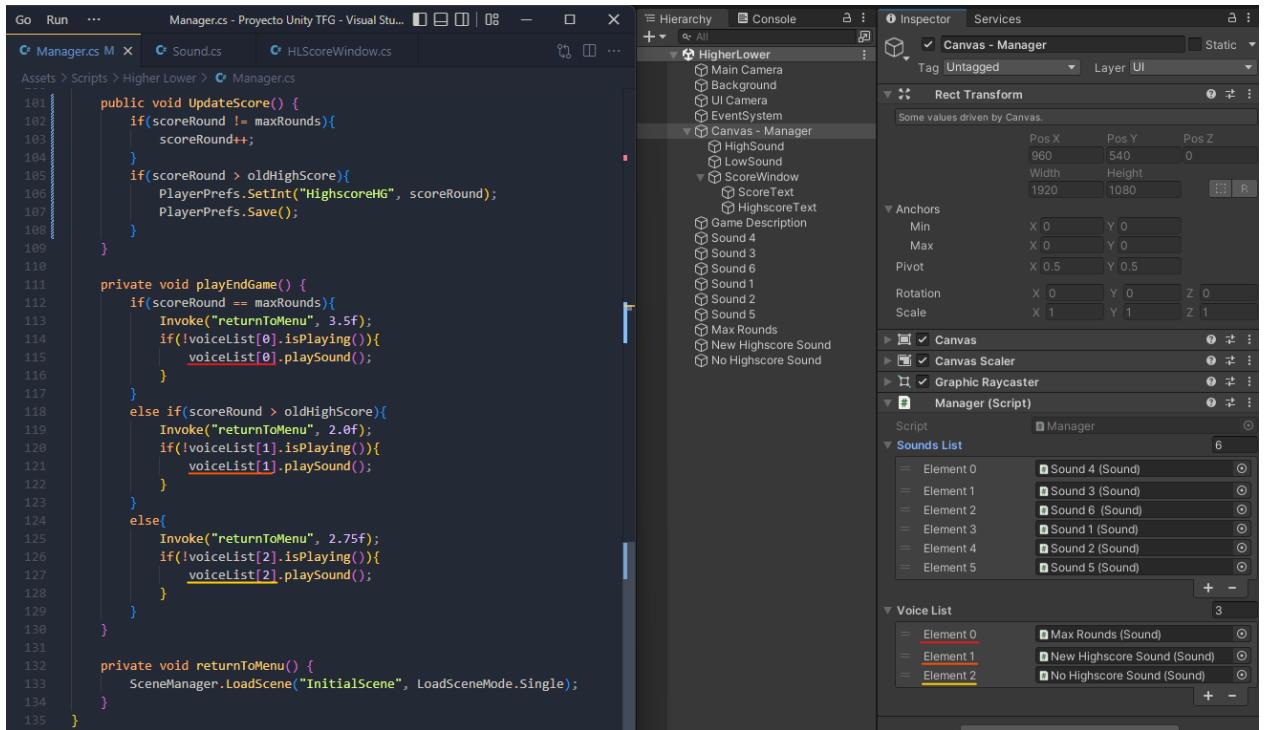


Ilustración 61 - Reproducción de voces al final del juego dependiendo del resultado final

Fuente: Captura de elaboración propia

Cuando el juego finalice se pueden dar 3 posibilidades:

- El jugador se ha pasado todas las rondas - Se reproduce el audio: Enhorabuena, has superado todas las rondas de Higher & Lower.
- El jugador no se ha pasado todas las rondas, pero ha superado su récord – Se reproduce el audio reutilizado de Ninja Run: Enhorabuena, has superado el récord.
- El jugador no se ha pasado todas las rondas ni ha superado su récord – Se reproduce el audio reutilizado de Ninja Run: No has conseguido superar el récord, vuelve a intentarlo.

Tras reproducirse el audio se volverá al menú inicial.

Por último, para finalizar con el desarrollo de Higher & Lower se implementó la interfaz de usuario. Se desarrolló un sencillo *script* para ello.

En cada actualización del bucle del juego, se actualizan los valores de las etiquetas *Score* y *Highscore* que indica el número actual de ronda y el récord alcanzado respectivamente.

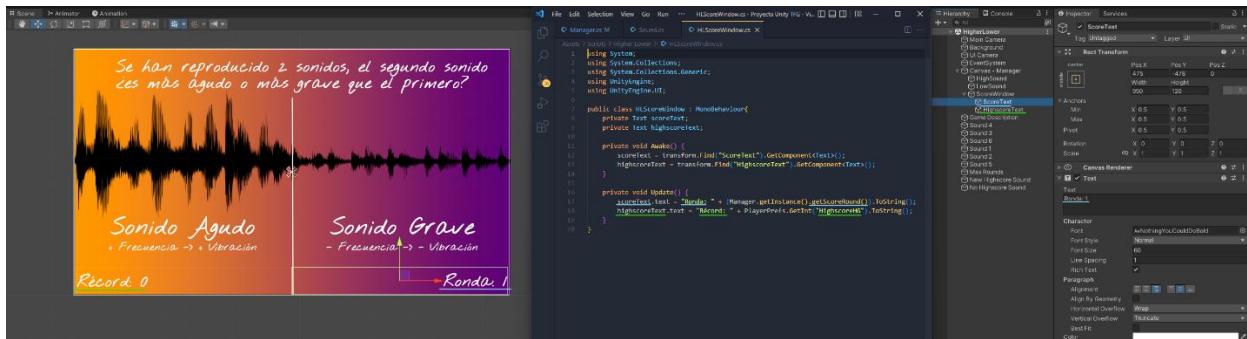


Ilustración 62 - Implementación de interfaz de usuario de Higher & Lower

Fuente: Captura de elaboración propia

Tras esta implementación, se da por finalizado el desarrollo de Higher & Lower y con ello el desarrollo de DIV.

7. CONCLUSIONES

Una vez finalizado el desarrollo de DIV, se pueden extraer diferentes conclusiones evaluando los resultados obtenidos. Estas conclusiones van a ser divididas en dos apartados: conclusiones del cumplimiento de objetivos y conclusiones sobre el desarrollo.

7.1. CONCLUSIONES SOBRE EL CUMPLIMIENTO DE OBJETIVOS

Como se ha visto en el apartado de objetivos, se propusieron 5 objetivos, los cuales han sido cumplidos. A continuación, se van a enumerar los objetivos propuestos inicialmente y el resultado obtenido:

- **Aprendizaje sobre el motor Unity y a su vez del lenguaje de programación C#.**
Todo el proyecto se desarrollará en el motor 2D de Unity. El proyecto ha sido desarrollado satisfactoriamente, obteniendo un producto final consistente. Para ello, se han debido asentar las bases de Unity para poder llevar a cabo el desarrollo. Con lo cual, se puede afirmar que este objetivo se ha cumplido y que, además, se han adquirido conocimientos suficientes como para poder plantear una continuidad del proyecto; así como el planteamiento de desarrollar nuevos proyectos en Unity.
- **Aplicación de las metodologías ágiles aprendidas durante todo el grado, especialmente durante el Itinerario de Creación y Entretenimiento Digital, de 4º Curso.** Un punto extremadamente importante en un proyecto es la correcta planificación de éste para conseguir que todo fluya y no hayan temporadas de una gran carga de trabajo. Como se comentó en el apartado de Metodologías, este objetivo ha sido aplicado a través de estas tratando de ser lo más fiel posible a esta idea y contando con la supervisión de la figura del tutor. Se considera que este objetivo ha sido alcanzado.
- **Diseño e implementación de todos los apartados que engloben los videojuegos.**
 - **En Ninja Run lo entendemos como la generación infinita y aleatoria del nivel, el diseño del nivel, el personaje y el sonido.** Todos estos apartados han sido correctamente implementados en Ninja Run y hay una correcta

cohesión entre todos ellos. Se considera así que este objetivo ha sido logrado con éxito.

- **En Higher & Lower abarcará la generación aleatoria del orden de los sonidos y la interfaz de usuario.** Estos objetivos también se han conseguido implementar satisfactoriamente en Higher & Lower tal y como se ha podido ver anteriormente. Se considera así que este objetivo ha sido logrado también con éxito.
- **Adaptación de los videojuegos, una vez se asienten los principales cimientos de cada juego, se deberá modificar para hacer que las características de la persona que va a jugar no influyan en la jugabilidad.** Este es el objetivo más importante de todo el trabajo sin duda, la idea principal del mismo. En Ninja Run se ha conseguido implementar un aviso de salto mediante la vibración y en Higher & Lower se ha conseguido gestionar la vibración para transmitir la diferencia de un sonido y otro. Por ello, se considera que el principal objetivo del trabajo ha sido cumplido.

Dado que todos los objetivos han sido cumplidos, se ha decidido subir el producto final a Google Drive, para que así se pueda descargar gratuitamente el juego a través del siguiente [enlace](#).

7.2. CONCLUSIONES SOBRE EL DESARROLLO

Finalmente, se van a exponer las distintas conclusiones a las que se han llegado a lo largo de todo el proceso de diseño y desarrollo del trabajo.

La mayor lección aprendida en este proyecto es la importancia de la empatía.

Quizás si en este proyecto se hubiese desarrollado un videojuego más normativo, no se debería haber rediseñado Ninja Run por ejemplo, ya que se podrían haber añadido nuevas mecánicas para romper con la monotonía que era. Pero este proyecto parte de la base de que todos tenemos derecho a ser felices y a disfrutar de las mismas cosas. La empatía juega un gran papel aquí ya que, aunque adaptar un videojuego es más costoso tanto a nivel de diseño como a nivel de desarrollo, las personas con dificultades también deben poder compartir la misma pasión por jugar que el resto.

Adaptar estos juegos ha supuesto que el planteamiento fuese visto desde otra perspectiva, puesto que cuando se diseña un juego se diseña pensando en el usuario estándar que podrá realizar cualquier acción para la que el dispositivo en cuestión está preparado. A veces se da por supuesto que todo el público va a ser así y desafortunadamente, no sucede de esa forma.

Adaptar estos juegos ha supuesto también más quebraderos de cabeza porque cuando se pensaba que ya estaba todo resuelto, de repente pensabas en otro usuario y realmente no se cumplía el objetivo de adaptación y tocaba modificar o rehacer partes para que funcionasen correctamente. Pero finalmente, tras ver el producto obtenido, es una gran satisfacción pensar que este juego va a poder ser usado por múltiples personas y que todas van a poder divertirse por igual. Es una satisfacción poder aportar un granito de arena a esta sociedad y poder pensar que el día de mañana un niño con diversidad funcional podrá estar jugando a DIV con sus amigos y todos podrán retarse a superar los récords sin problema alguno.

Esta satisfacción justifica todos los esfuerzos realizados.

Sin lugar a duda, todos los conocimientos adquiridos serán de gran ayuda para el desarrollo de los siguientes proyectos, así como para tener en una mayor visión de todo el público que se puede tener a la hora de desarrollar un producto.

BIBLIOGRAFÍA

- Craven, C. (2020). *The Last of Us: Part 2 - Deaf/HoH Review*. Retrieved from Can I Play That?: <https://caniplaythat.com/2020/06/12/the-last-of-us-2-deaf-hoh-review/>
- EcuRed. (2019). *Modelo de prototipos*. Retrieved from EcuRed: https://www.ecured.cu/index.php?title=Modelo_de_prototipos&oldid=3532289
- Fernández, Y. (2019). *Qué es GitHub y qué es lo que le ofrece a los desarrolladores*. Retrieved from Xataka: <https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores>
- Freslon, B. (2022). *Use custom vibrations on mobile with this native Plugin for Unity (Android & iOS)*. Retrieved from GitHub.
- Geswein, K. (2014). *Nothing You Could Do*. Retrieved from Dafont: <https://www.dafont.com/es/nothing-you-could-do.font>
- Kombat, S. (2020). *The Last of Us 2 - Blind Accessibility Review*. Retrieved from Can I Play That?: <https://caniplaythat.com/2020/06/18/the-last-of-us-2-review-blind-accessibility/>
- Ministerio de Derechos Sociales y Agenda 2030. (2020). *Base Estatal de datos de personas con valoración del grado de discapacidad*. Retrieved from Gobierno de España: https://www.imserso.es/InterPresent1/groups/imserso/documents/binario/bdepcd_2020.pdf
- Ministerio de Trabajo y Asuntos Sociales. (2001). *Clasificación Internacional del Funcionamiento, de la Discapacidad y de la Salud*. Retrieved from Gobierno de España: <https://www.imserso.es/InterPresent2/groups/imserso/documents/binario/435cif.pdf>
- Proyectos Ágiles. (2020). *Qué es SCRUM*. Retrieved from Proyectos Ágiles: <https://proyectosagiles.org/que-es-scrum/>
- Real Academia Española. (2021). *Videojuego - Definición diccionario de la lengua española*. Retrieved from Diccionario Real Academia Española: <https://dle.rae.es/videojuego>
- Romañach Cabrero, J. (2005). *Foro de Vida Independiente*. Retrieved from Wayback Machine: https://web.archive.org/web/20171031055444/http://www.asociaciones.org/vidaindepen/docs/diversidad%20funcional_vf.pdf
- Ruelas, U. (2017). *¿Qué es un motor de videojuegos?* Retrieved from Coding or not: <https://codingornot.com/que-es-un-motor-de-videojuegos-game-engine>
- Sony Interactive Entertainment. (2020). *The Last of Us Parte II - Accesibilidad (España)*. Retrieved from PlayStation: <https://www.playstation.com/es-es/games/the-last-of-us-part-ii/accessibility/>
- Unity Technologies. (2022). *Escenas*. Retrieved from Unity Manual: <https://docs.unity3d.com/es/current/Manual/CreatingScenes.html>
- Unity Technologies. (2022). *GameObject*. Retrieved from Unity Manual: <https://docs.unity3d.com/es/current/Manual/class-GameObject.html>
- Unity Technologies. (2022). *Learn Game Development w/ Unity*. Retrieved from Unity Documentation: <https://learn.unity.com/>

Unity Technologies. (2022). *Plataforma de desarrollo en tiempo real de Unity*. Retrieved from Unity: <https://unity.com/es/>

Unity Technologies. (2022). *Unity Store - Comparar planes*. Retrieved from Unity: <https://store.unity.com/es/compare-plans>

Unity Technologies. (2022). *Uso de Blender y Maya con Unity*. Retrieved from Unity: <https://unity.com/es/how-to/beginner/using-blender-and-maya-unity#design-and-development-converge>