Paper Ship

Jonatan Fernández Membrives
Diseño y Programación de Videojuegos
Área de Diseño, Creación y Multimedia

Jordi Duch Gavaldà
Jordi Duch Gavaldà

3 de enero de 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	Paper Ship
Nombre del autor:	Jonatan Fernández Membrives
Nombre del consultor/a:	Jordi Duch Gavaldà
Nombre del PRA:	Jordi Duch Gavaldà
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	Diseño y Programación de Videojuegos
Área del Trabajo Final:	Área de Diseño, Creación y Multimedia
Idioma del trabajo:	Castellano
Palabras clave	Desarrollo de Videojuego

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El diseño y el desarrollo de un videojuego es una tarea que conlleva mucho tiempo y recursos, durante todo el trayecto del curso, se han ido aprendiendo una serie de herramientas y conocimientos que en su conjunto permiten la creación completa de un videojuego. Por ello, en este proyecto se plantea aplicar todos estos conocimientos adquiridos para la realización de un videojuego desde su conceptualización hasta su producción.

A partir de una idea inicial, se establecen los diferentes hitos y requerimientos necesarios para completarla. En base a la planificación inicial, a lo largo del proyecto se evalúan y modifican los objetivos en base a las necesidades de cada hito.

Como objetivo final, se propone crear un juego de rol de acción en tercera persona que englobe las características más comunes de este género, analizando cuales son las dificultades y retos del proyecto.

Abstract (in English, 250 words or less):

Designing and developing a video game is a task that takes time and lots of resources, over the duration of the master's degree, a lot of tools and knowledge have been acquired to be able to create a video game in its whole. Therefore, in this project it is proposed to apply all these acquired knowledge, to the realization of a video game from its conceptualization to its production.

Starting from a base idea, the different milestones and requirements necessary to complete it are established. Based on the initial planning, throughout the project, the objectives are evaluated and modified according to the needs of each milestone.

As a final goal, it is proposed to create a third-person action role-playing video game that encompasses the most common characteristics of this genre, analyzing which are the difficulties and challenges of the project.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

El diseño y el desarrollo de un videojuego es una tarea que requiere de muchísimo esfuerzo y trabajo. Además, cuando no se tiene mucha experiencia en este, se añaden complicaciones como falta de definición y errores que inducen a repetir muchas partes del proceso. La finalidad de este proyecto es pensar detalladamente en todos estos puntos y procesos mientras se desarrolla un videojuego a nuestra elección.

Empezando por la idea (ponerla por escrito, definir al máximo lo que queremos recrear en nuestro videojuego, realizar bocetos, buscar conceptos, etc.) hasta el desarrollo de un videojuego completo, es la solución encontrada para poder descubrir como de complejo es todo el proceso en sí. Se intenta entrar al máximo en todos los procesos necesarios para llegar al videojuego final y ver cuáles son las necesidades, problemas y recursos que encontramos en cada uno.

1.2 Objetivos del Trabajo

El principal objetivo del trabajo es desarrollar un videojuego desde cero con todo lo que ello conlleva.

Además de este objetivo principal, también se tiene como objetivo, como se ha comentado anteriormente, el análisis de todos los estados y procesos del diseño y desarrollo de un videojuego.

Intentar desarrollar todo lo planificado en la primera entrega requiere de buena gestión y saber planificar los recursos y tiempos requeridos para cada punto del proceso. Ver que se ha llevado a cabo y que no, ver como esto ha impactado en el producto final y que cosas se podrían haber mejorado es otro de los objetivos de este proyecto.

1.3 Enfoque i método seguido

Como ya hemos comentado, la mejor forma de ver y estudiar todos los pasos del diseño y desarrollo de un videojuego es, básicamente, realizar todos estos uno mismo y ver en primera persona como es.

Desde una simple idea, deberemos diseñar todo lo necesario para empezar a desarrollar un videojuego y con ese diseño establecer y planificar todas las tareas. Se creará un pequeño juego, donde se muestre la introducción y un poco las principales mecánicas y estética del videojuego que se ha pensado.

Además, se analizarán todos los pasos y como estos han repercutido en el producto final y en qué cosas podríamos mejorar o modificar para un futuro nuevo proyecto de este mismo alcance.

1.4 Planificación del Trabajo

La planificación se ha separado en cuatros grandes hitos: documento del diseño del videojuego, versión parcial, versión jugable y versión final (documento de memoria del proyecto incluido):

- **Documento del diseño del videojuego**: [16/09/20 – 4/10/20] Para este hito no se ha planificado con anterioridad ya que este se utiliza para planificar los otros tres hitos que veremos más adelante.
- **Versión parcial**: [5/10/20 – 01/11/20] Para este hito se ha creado un *roadmap* por semanas indicando las tareas de cada parte del desarrollo del videojuego.

Ilustración 1. Roadmap versión parcial

- **Versión jugable**: [2/11/20 – 06/12/20] Para este hito, como en el anterior, se ha creado un *roadmap* por semanas indicando las tareas de cada parte del desarrollo del videojuego.

Ilustración 2. Roadmap versión jugable

- **Versión final**: [7/12/20 – 03/01/21] Para este hito, como en el anterior, se ha creado un *roadmap* por semanas indicando las tareas de cada parte del desarrollo del videojuego y la memoria.

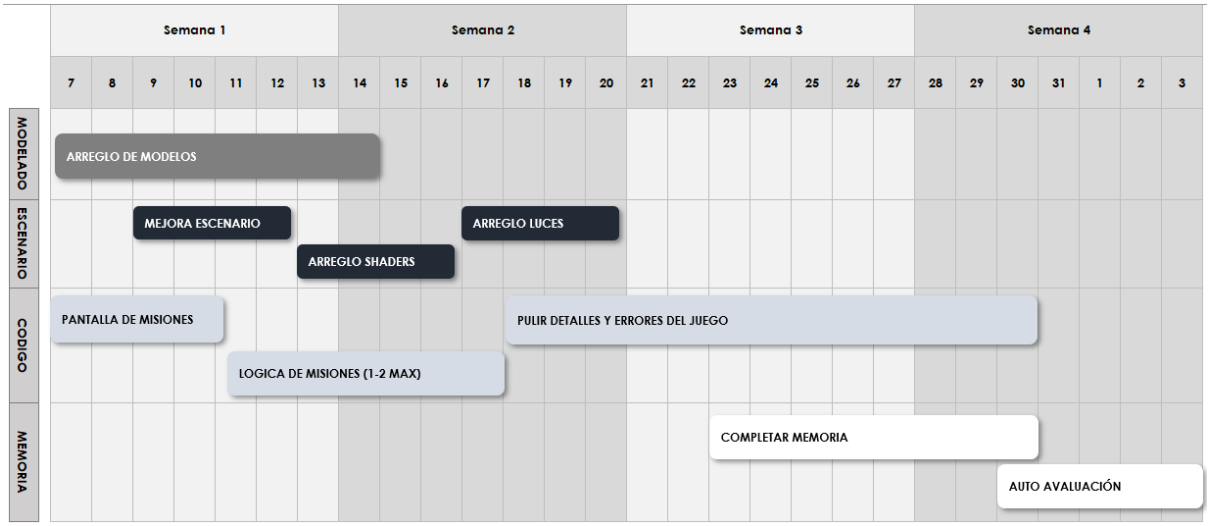

Ilustración 3. Roadmap versión final

1.5 Breve sumario de productos obtenidos

Se han obtenido una serie de diferentes productos, en su mayoría audiovisuales durante el transcurso del proyecto:

- Ejecutable del videojuego ***Paper Ship*** para la plataforma de PC Windows.
- Múltiples videos explicativos del videojuego que han sido utilizados en cada entrega del proyecto.
- Video tráiler promocional del videojuego.
- *Roadmap* de cada parte del proyecto con la planificación seguida.
- Repositorio en GitHub con todo el contenido del proyecto y tags de cada una de las *releases* y entregas realizadas.

1.6 Breve descripción de los otros capítulos de la memoria

A continuación, se expondrán los contenidos de los capítulos que conforman la memoria de este trabajo final:

- **Estado del arte**: Revisión del género rol de acción en tercera persona y títulos del mismo género y que se asemejan a este proyecto.
- **Definición del juego**: Conceptualización del juego, personajes, historia y ambientación decididos para el proyecto.
- **Diseño técnico**: Entorno de desarrollo escogido y su motivo, así como los elementos, lógicas y algoritmos desarrollados más importantes del videojuego.

- **Diseño de niveles**: Mostraré el diseño de Kattegat, la isla a la que llega nuestro protagonista después de un accidente de barco.
- **Manual de usuario**: Como jugar y controles que tiene el videojuego para facilitar su uso para una persona primeriza en los videojuegos de PC.
- **Conclusiones**: Conclusiones obtenidas durante la realización de todo el proyecto de Final de Máster. Que cosas cambiaría y que cosas mantendría del proceso.

2. Estado del arte

El género de rol de acción o *ARPG*, se caracteriza por la inmersión en un mundo virtual, donde somos el protagonista del juego, y donde con una serie de parámetros (vida, defensa, daño, armas, etc.) las acciones que realicemos tendrán un impacto en el progreso de la historia.

Desde los inicios de los juegos de mazmorras salieron los primeros videojuegos de rol, pero en sus inicios la batalla no sucedía en tiempo real, si el jugador no se movía los enemigos tampoco. Durante la década del 1980 se empezaron a explorar las posibilidades de videojuegos de rol en tiempo real, que acabaron siendo los videojuegos de rol de acción. Uno de los primeros juegos del mercado fue *Dragon Slayer*.

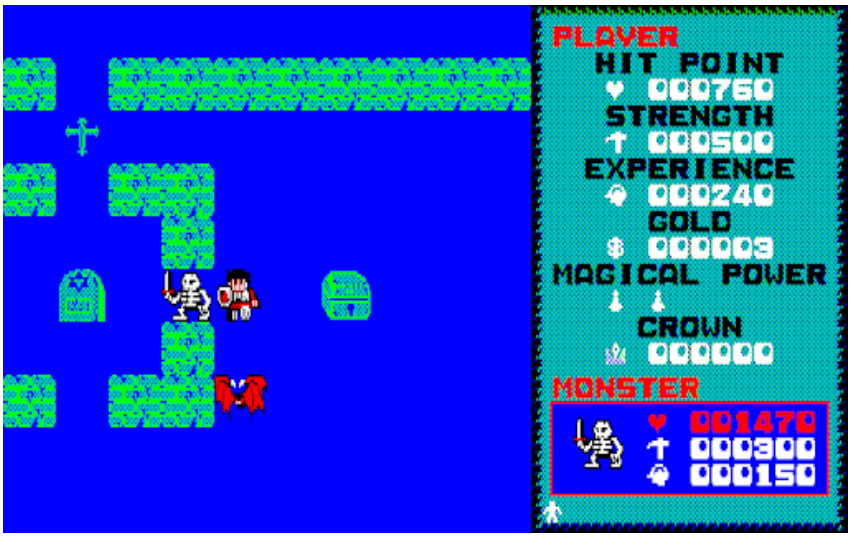

Ilustración 4. Dragon Slayer

Con el avance el tiempo y de las tecnologías, de juegos como Dragon Slayer hemos podido llegar a juegos con mejores gráficos y con cámara en tercera persona. Como seria uno de los grandes juegos de estos últimos años en este género, *The Legend of Zelda: Breath of the Wild*.

Ilustración 5. The Legend of Zelda: Breath of the Wild

3. Definición del juego

3.1 Historia y ambientación

Nuestro protagonista es un niño que despierta en un barco de papel durante una tormenta, que le arrastra hacia una isla vikinga. Al llegar conoce a Eyra, una habitante del pueblo, que le explica la historia del reino.

Antaño, el reino de X gozaba de vida y prosperidad. Gobernados por el rey X, los habitantes vivían en tranquilidad, las calles estaban repletas de color y música, los niños jugaban libremente sin preocupaciones. Pero toda esta felicidad llegó a su fin cuando la reina X cayó enferma. El rey devastado por la triste noticia, movió mares y montañas para encontrar una cura para su reina: rezó día y noche, hizo traer a todos los curanderos del reino…; pero no parecía ayudar a la reina.

Sin lograr su cometido, abandonado por todos sus dioses y creencias, el rey en el lecho de muerte de su amada reina, juró que daría lo que fuera necesario para salvarla. En ese momento, de entre las sombras, un dios oscuro se manifestó ofreciéndole un pacto. Yo salvaré a tu reina, pero a cambio quiero tu reino, dijo el ser. El rey sin pensárselo dos veces accedió al trato sin saber lo que ello supondría para los habitantes de su reino.

Hoy, un año después de esos sucesos, todo el reino esta infestado y controlados por los súbditos del dios oscuro que controla actualmente el reino. Esta isla, aunque parezca prácticamente deshabitada, anteriormente estaba llena de gente que vivía y crecía en ella, pero con la llegada del mal, muchos se fueron, otros fueron tomados prisioneros, y algunos fueron asesinados. Los pocos que quedamos somos solo niños y abuelos que no suponen ningún peligro para el dios oscuro.

3.2 Definición de personajes

Estos son los personajes de **Paper Ship**, debido a la corta duración de la demo, solo se pueden llegar a ver en detalle al protagonista y a Eyra, los demás personajes se darían a conocer según la trama del videojuego avanzase:

- **Protagonista**: Niño que no recuerda nada de su pasado, pero ayudará a los habitantes del reino a deshacerse del dios oscuro que los tiene controlados. Durante su aventura ira descubriendo cosas sobre su pasado y sobre quien es, que le ayudaran a entender de donde viene.
- **Eyra**: Niña vikinga valiente, atrevida, aventurera e independiente. Es la primera en encontrar y ayudar al protagonista en la historia, también es la que explica la situación y la historia al jugador.

- **Rey del reino**: Hombre mayor, de con un poco de sobrepeso, en su tiempo fue un gran guerrero, pero desde que se convirtió en rey no ha salido en muchas aventuras. Vive por su gran amor, la reina y en cuanto cae enferma, el rey se vuelve loco por intentar salvarla.
- **Reina**: Hermosa mujer, de constitución débil y de nobles valores.
- **Dios oscuro**: Ser maligno que solo quiere controlar todo el reino y hacer sufrir a sus habitantes, en cuanto consigue mandar sobre el reino, manda a todas sus tropas a capturar a los habitantes fuertes y luchadores para tenerlos controlados y que no puedan recuperar el reino.
- **Enemigos**: Servidores del dios oscuro, no son muy inteligentes. Siguen las órdenes del dios oscuro.

3.3 Interacción entre los actores del juego

Los enemigos serán siempre dóciles, pero si el jugador se acerca demasiado a ellos, estos se volverán agresivos y atacarán al jugador. Para derrotarlos el jugador deberá reducirles la vida a 0 sin morir en el intento.

Las interacciones con Eyra serán algo más complejas, ella nos dará pistas sobre como avanzar en caso que estemos bloqueados y nos recomendará que hacer si le preguntamos.

3.4 Concept art

Ilustración 6. Concept art protagonista y Eyra

Ilustración 7. Concept art dios oscuro

Ilustración 8. Concept art isleños vikingos

Ilustración 9. Concept art isla

4. Diseño técnico

4.1 Evaluación de game *engines*

Para la realización de este proyecto evalué los dos principales *engines* más conocidos del mercado, Unreal y Unity. Tratándose de un proyecto de la envergadura de este juego, no creo que se pueda desperdiciar mucho tiempo aprendiendo como funciona un *engine* nuevo, y debido a que la plataforma de destino es ordenador, creo que el *engine* más optimo es con el que tengo más experiencia trabajando: Unity.

4.2 Herramientas utilizadas

Durante todo el proyecto se han utilizado una serie de herramientas no solo para la realización del proyecto sino también para organizar y gestionar el proyecto.

A continuación, se listan todas ellas según su categoría.

Desarrollo:

- **Unity 2020.1.5f1**: *Game engine* escogido para todo el desarrollo del videojuego **Paper Ship**.
- **Visual Studio 2017**: Editor de *scripts*.
- **GitHub**: Control de versiones, *tags* y *releases*.

Diseño/Arte:

- **Unity Asset Store**: Repositorio de *assets* en línea de la comunidad de Unity.
- **Adobe Photoshop CC 2020**: Creación y edición de imágenes y UI del juego.
- **Autodesk Maya 2018**: Creación y edición de modelos 3D.
- **Mari 4.6**: Texturización de los modelos 3D.
- **Adobe Premiere CC 2019**: Creación y edición de videos.
- **OBS Studio**: Captura de pantalla para grabar el *gameplay* del juego.

Gestión y planificación:

- **Microsoft Word**: Documentación de todas las entregas y memoria del proyecto.
- **Microsoft Excel**: *Roadmap* y planificación de los hitos del proyecto.
- **Super Productivity**: Control y priorización de tareas, así como control del tiempo empleado en cada tarea.

4.3 En busca de un estilo

Uno de los puntos más importantes que definirían todas las decisiones artísticas y de diseño, era encontrar un estilo que caracterizase al juego, que tipo de gráficos tendría el juego. El *low poly* se caracteriza por ser más sencillo de crear, pero también requiere de un poco de post procesado en Unity para que no parezca demasiado plano. En cuanto al *high poly*, es mucho más complejo de realizar para generar resultados realistas y los requerimientos de nuestro juego subirían ya que necesitaría de más recursos para poder procesar todos los polígonos.

Por ello, en primera instancia se ha decidido realizar un videojuego *low poly*, o al menos con un número de polis reducido. Con esto en mente, empecé a buscar en la *Unity Asset Store* en busca de ejemplo o posibles *assets* para utilizar en el proyecto.

No se llegó a encontrar nada que coincidiese con lo que tenía en mente para los personajes ni UI, así que esta parte fue completamente generada desde cero. Por otro lado, se encontraron buenos *assets* para generar una isla *low poly.*

Una vez con los *assets* para el mundo decididos y añadidos a una escena de prueba en Unity. Los resultados visuales no eran exactamente lo buscado, para asemejarse a la idea en mente se realizaron una serie de modificaciones que le dan el aspecto final al videojuego.

Pestaña *Quality* del *Project Settings:*

- **Shadow Projection**: Stable Fit => Close Fit
- **Shadow Resolution**: High Resolution => Very High Resolution

Directional Light de la escena:

- **Normal Bias**: 0.4 => 0
- **Color**: se ha modificado el color para cambiar el tono de la escena.

PostProcess (se ha añadido el *asset Post Processing Stack* para editar la imagen que genera la cámara):

- **Camara**: Añadir *post-process layer* para que utilice la configuración creada con *post-process*.
- **PostProcess GameObject**: Nuevo *GameObject* en la escena que contiene un *post-process* volumen con los cambios realizados en la imagen. En este caso se ha añadido un *Color Grading* para modificar el color del juego a uno más cercano a la idea buscada.

Lighting window, Environment tab:

- **Sun Source**: *None => Directional Light* de la escena (también servirá de luz solar en nuestro juego).
- **Environment Lighting Source**: *Skybox => Color*.
- **Ambient Color**: se ha modificado el color para cambiar el tono de la escena.
- **Fog:** *Unchecked => checked.*
- **Fog Color:** se ha modificado el color para cambiar el tono del *fog* de la escena.
- **Fog End:** 150 => 300.

Con todos estos cambios la escena, luces y colores del juego llegaron al punto deseado.

Ilustración 10. Escena final sin ningún cambio de luces ni colores aplicados

Ilustración 11. Escena final con los cambios de luces y colores aplicados

4.4 Personajes

Una vez con la estética del juego decidida se necesitan personajes que ocuparan la escena. Debido a la falta de *assets* en este aspecto se optó por generar los personajes desde cero mediante la herramienta Adobe Maya.

Debido a la gran carga de trabajo detectada al final del primer módulo, se decidió mantener los modelos 3D que iba a generar desde cero al mínimo. Por ellos tan solo se generaron los modelos del protagonista y del *PNJ* Eyra.

Comenzando por el protagonista, lo primero fue el modelado de la malla base, sin ningún tipo de ropa o elementos extra, solo el cuerpo desnudo del protagonista.

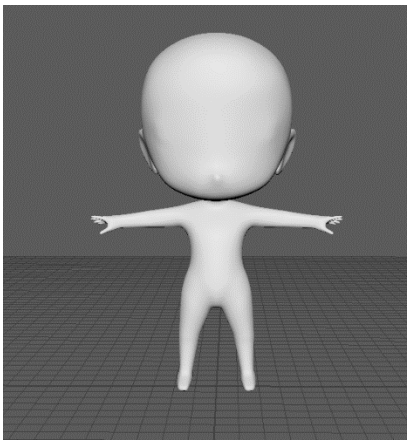

Ilustración 12. Malla base protagonista sin props

Una vez con esta base, se generaron todos los *props* del personaje: pelo, bandana, ropa, saco, etc; y se le añadieron colores planos para ver como quedaba todo en conjunto.

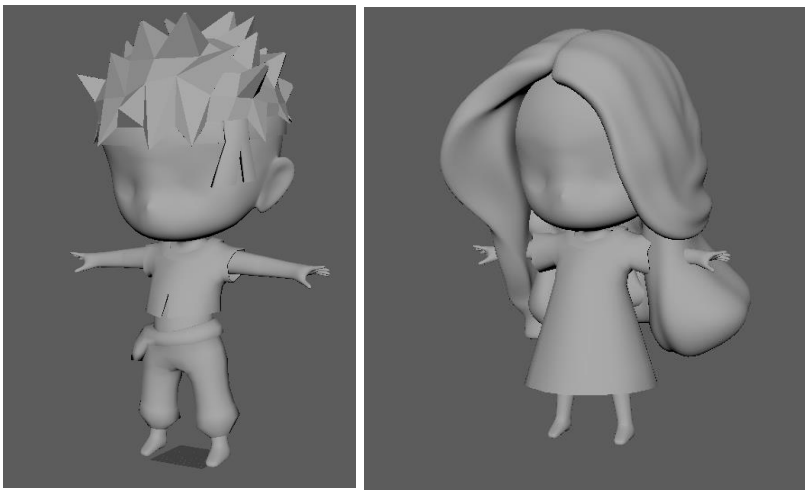

Ilustración 13. Malla base protagonista y Eyra con props

Por último, se han arreglado y perfilado detalles del modelo. Se han extraído las *UV*s del modelo para poder pintar las texturas finales del modelo. Y mediante el programa *Mari* se ha pintado la textura del personaje, ya que este nos permite ver en tiempo real como le afectan las texturas aplicadas a través *UV* en el modelo 3D.

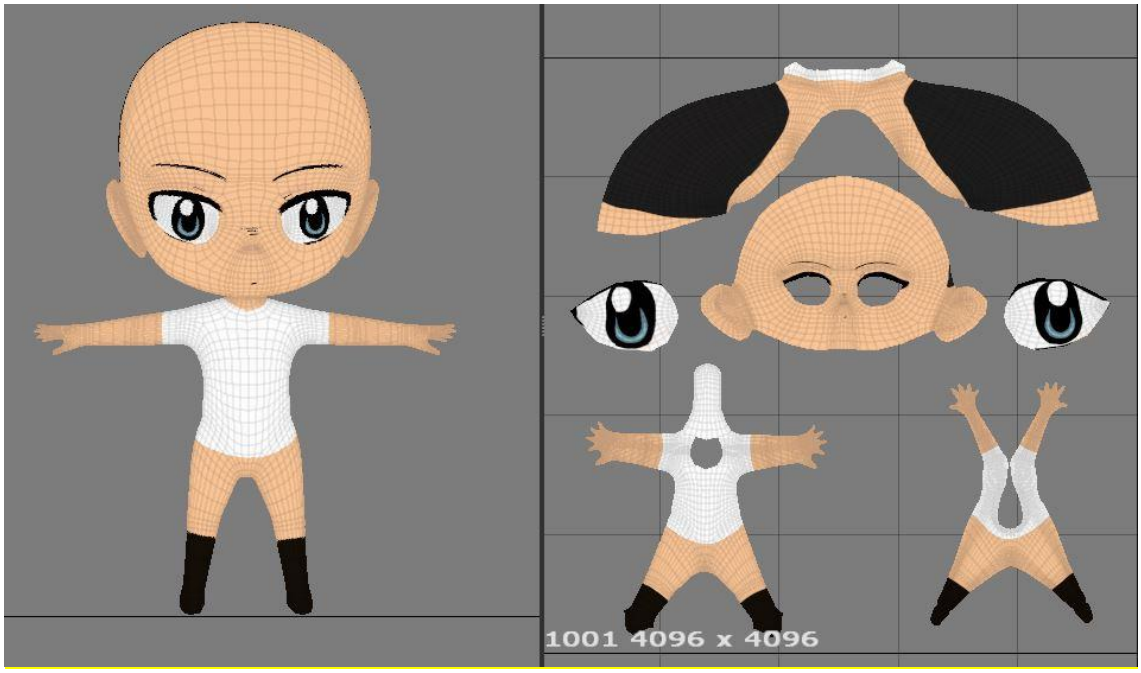

Ilustración 14. Texturización Mari modelo protagonista

Ilustración 15. Modelo protagonista finalizado

Para el personaje de Eyra se ha utilizado el mismo modelo de la ilustración 12 y se han seguido los mismos pasos que con el protagonista, para llegar a conseguir el modelo final con sus *UVs* con sus texturas.

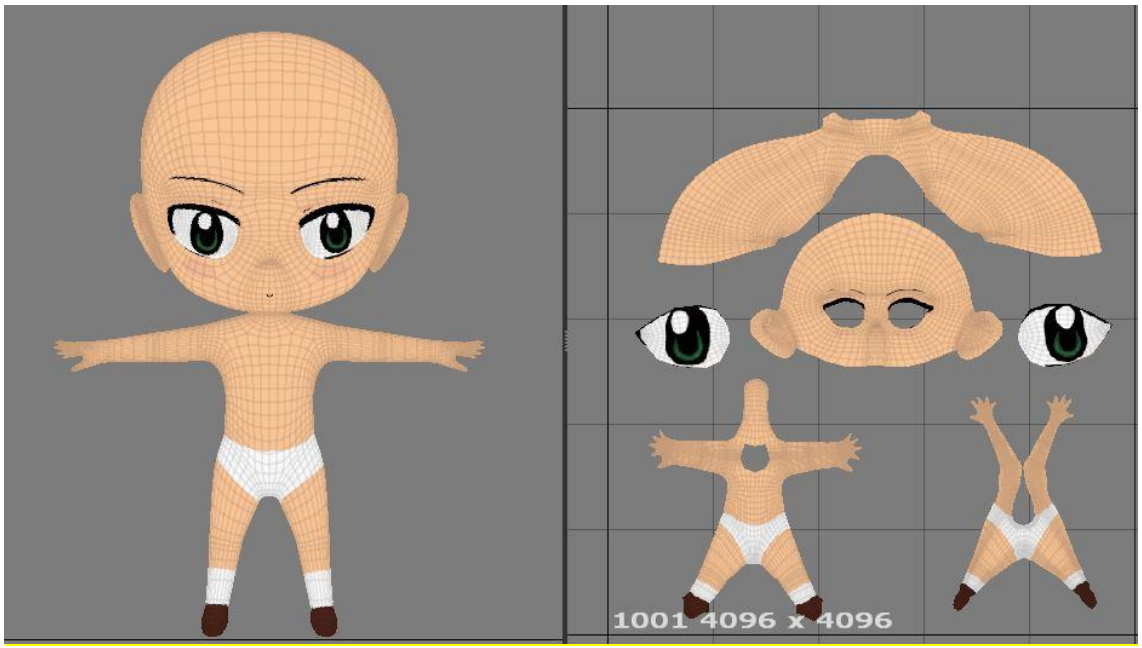

Ilustración 16. Texturización Mari modelo Eyra

Ilustración 17. Modelo Eyra finalizado

En primera instancia se pensó en generar el *rigging* y animaciones de ambos personajes desde cero, pero debido a la gran cantidad de animaciones que debía ejecutar el protagonista (caminar, saltar, girar, atacar, agacharse, morir, etc.) se planteó como alternativa hacer uso de la herramienta gratuita *online Mixamo.*

Creamos una cuenta gratuita con Adobe y ya podemos subir nuestros personajes a esta plataforma. Por defecto, *Mixamo* reconoce de forma bastante precisa los modelos y genera un esqueleto bastante preciso, en nuestro caso al tratarse de un modelo humano, pero de proporciones antinaturales necesita un poco de nuestra ayuda para generar el esqueleto. Una vez *Mixamo* recoge nuestro modelo, nos pregunta donde están algunas partes de nuestro modelo y con ello genera un esqueleto que funciona con nuestro modelo.

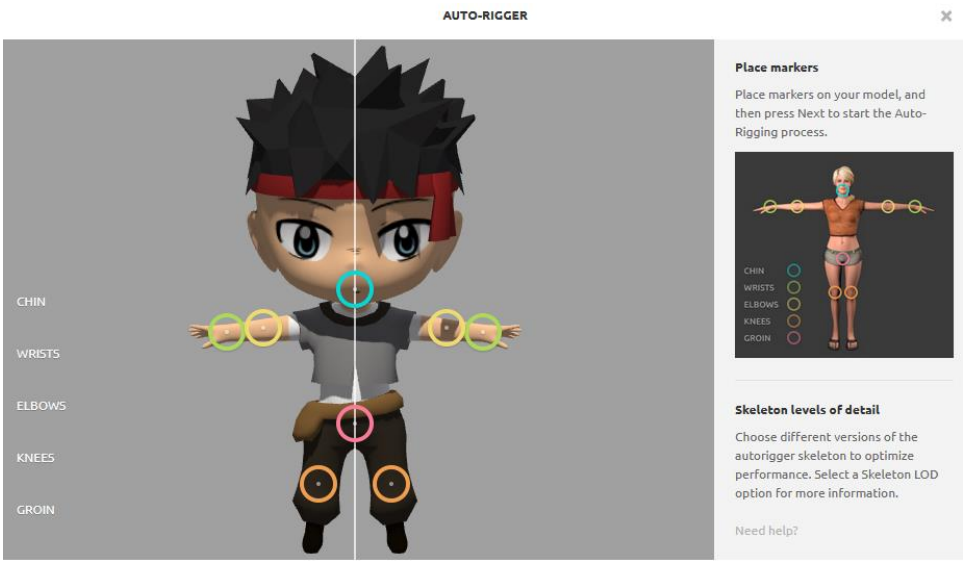

Ilustración 18. Mixamo auto-rigger pop up

Y con esto ya tenemos acceso a infinidad de animaciones que podremos usar en nuestro proyecto.

Mixamo no es 100% exacto y es necesario hacer algunos cambios en Adobe Maya para arreglar los pesos de los huesos acorde al modelo. Para ello importamos nuestro modelo con el esqueleto generado en *Mixamo* en Maya y con la herramienta de *Paint Weight* podremos arreglar algunos de los huesos generados para tener un movimiento más natural en nuestro personaje.

Por último, una vez añadido el modelo final a Unity, debemos indicar que se trata de un *Rig* de tipo *Humanoid* para que Unity sepa cómo funciona el modelo.

Arreglando detalles y problemas del modelo en Unity, se detectó que el mapeo de algunos huesos generados por *Mixamo* no estaban asociados al esqueleto de Unity y se tuvieron que añadir manualmente.

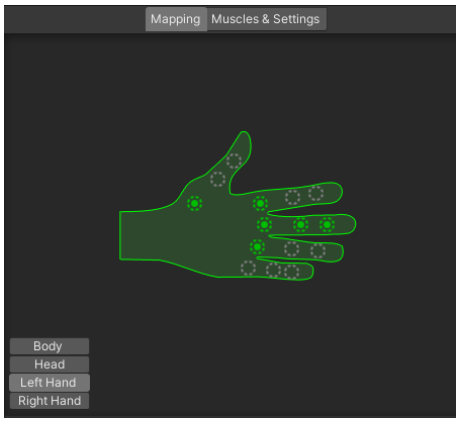

Ilustración 19. Error mapeo huesos Mixamo en Unity

4.5 Arquitectura y componentes

El juego se compone de tres escenas:

Menu Scene

Escena inicial de juego donde el jugador se encuentro con el menú inicial, el titulo del juego y un paisaje para añadir un toque visual relacionado al juego.

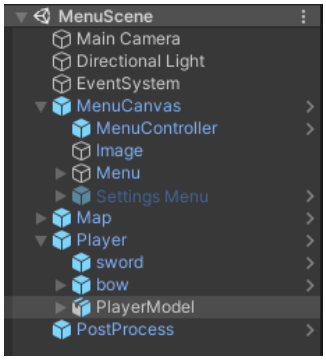

Ilustración 20. Arquitectura escena MenuScene

La escena se compone de los siguientes componentes:

- ***Main Camera***: Cámara de la escena, contiene un *post-process layer* para aplicar los cambios de color mencionados en el punto 4.3.
- ***Directional Light***: Luz direccional básica.
- ***EventSystem***: Sistema de eventos por defecto de Unity, necesario ya que la escena contiene un *canvas* con botones.
- ***MenuCanvas***: *Prefab* de canvas que contiene todo lo necesario para mostrar los menús. El punto más importante es el *prefab MenuController* que contiene cuatro scripts:
    - ***MenuManager***: Contiene todas las funcionas que llamaran los botones al ser clicados, también tiene referenciado el *GameObject* de *SettingsMenu* para poder mostrarlo y esconderlo cuando el usuario acceda a ese menú.
    - ***PrefManager***: Guarda y lee las preferencias del jugador, más concretamente el idioma seleccionado, el volumen del juego y el volumen de los sonidos del juego.
    - ***LanguageSystem***: Sistema de idioma, contiene todos los diccionarios de idiomas y cuando se modifica el idioma se encarga de actualizar todos los textos.
    - ***SettingsManager***: Utilizado por el menú *SettingsMenu* que se encarga de llamar a los respectivos *scripts* de este *prefab* para modificar el idioma o las preferencias del jugador.
- ***Map***: Mapa de Kattegat donde transcurre todo el juego.

- ***Player***: *Prefab* del jugador con todos sus *scripts* desactivados para que no pueda moverse ni interactuar con el usuario. Está constantemente en animación *idle*.
- ***PostProcess***: *Prefab* que contiene un *post-process volume* con toda la configuración de color para aplicar a la cámara.

Intro Scene

Escena de introducción al juego donde el jugador se encuentra con Eyra que le da la bienvenida y le explica dónde está.

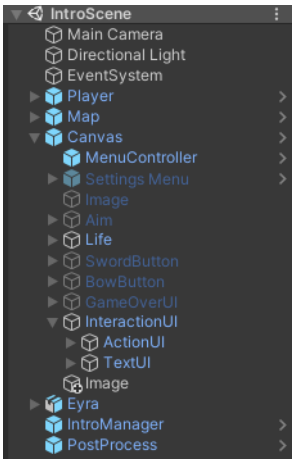

Ilustración 21. Arquitectura escena Intro Scene

La escena se compone de los siguientes componentes: (*Main Camera, Directional Light, EventSystem, Map, Player* y *PostProcess* son omitidos ya que son exactamente igual que en la escena anterior)

- ***Canvas***: Canvas muy similar al *MenuCanvas* visto en la escena anterior. Este contiene unos cuantos elementos nuevos, los desactivados en la ilustración anterior serán explicado en la siguiente escena. El elemento *Life* es la *UI* de vida del protagonista. El elemento *InteractionUI* contiene una serie de elementos para mostrar la *UI* de conversación entre el protagonista y Eyra.
- ***Eyra***: Modelo de Eyra con animación idle con todos sus *scripts* desactivados.
- ***IntroManager***: Principal *script* de la escena, es el encargado de controlar todo el progreso de esta escena. En un inicio muestra un mensaje de Eyra hablando con el protagonista preguntando si está bien. En cuanto el protagonista pula el botón de interacción, este script llama al animador del *prefab Player* para que se levante. Una vez levantado solo permite al jugador hablar con Eyra con el mismo botón de interacción. Una vez empieza la conversación, se encarga de avanzar por todos los textos y en acabar muestra una pantalla en negro y carga redirige hacia la siguiente escena.

Game Scene

Escena más compleja de todas, contiene toda la lógica del juego y es donde se desarrolla este.

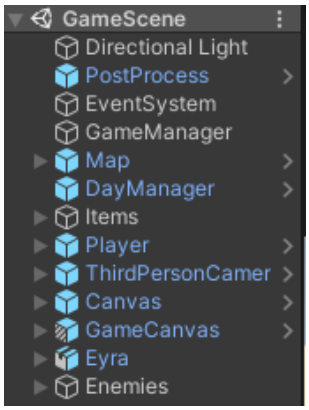

Ilustración 22. Arquitectura escena Game Scene

La escena se compone de los siguientes componentes: (*Directional Light, PostProcess, EventSystem* y *Map* son omitidos ya que son exactamente igual que en la primera escena)

- ***GameManager***: *Script* que controla el tiempo del juego, al principio muestra una pantalla en negro que se difumina hasta mostrar la escena y en ese momento inicia la partida. También se encarga de parar el juego al pulsar el botón pausa.
- ***DayManager***: *Script* que controla la rotación de la luz, simulando el avance del sol durante el día.
- ***Items***: *GameObject* que contiene todos los objetos con los que el jugador puede interactuar. Estos objetos tienen un *SphereCollider* que cuando el jugador entra en contacto habilita la opción de recoger el objeto y cuando el jugador sale de este deshabilita su recolección. Si el jugador recoge el objeto, este se destruye.
- ***Player***: *Prefab* del jugador que contiene toda la lógica de movimiento y animación del jugador. Se entra en más detalle en el punto 4.6.
- ***ThirdPersonCamera***: *Prefab* que contiene la cámara de la escena. También contiene un *script* que se encarga de rotar la cámara en base al movimiento del ratón. También cuando se utiliza la rueda del ratón, se encarga de acercar y alejar la cámara al jugador.
- ***Canvas***: Mismo *canvas* que hemos visto en la escena *IntroScene.*
- ***GameCanvas***: Mismo *canvas utilizado en la escena MenuScene pero escondiendo algunos elementos no necesarios en esta escena.*
- ***Eyra***: PNJ que contiene el *script NPCController encargado de toda la interacción con el jugador. Se entra en más detalle en el punto 4.8.*
- ***Enemies***: *GameObject* que contiene todos los enemigos de la escena, al iniciar la escena, recorre todos los hijos que lo contienen y guarda sus posiciones y rotaciones. Después cuando el protagonista descansa, todos

los enemigos que han muerto vuelven a aparecer donde estaban anteriormente.

4.6 Player

El *prefab* del jugador es uno de los más complejos del juego ya que contiene toda la lógica de movimiento, ataque y vida del personaje. Está compuesto por una serie de *scripts* y componentes de Unity necesarios para su funcionamiento.

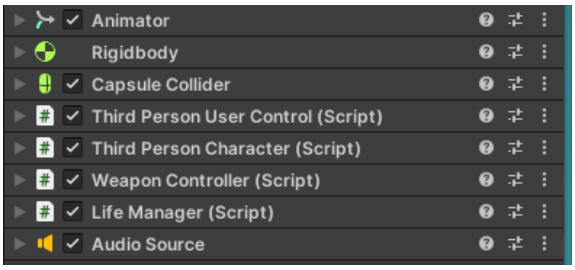

Ilustración 23. Componentes Player prefab

De principio a fin estos son los componentes y su necesidad dentro del *prefab.*

Animator

Contiene el *avatar* del modelo del jugador y el controlador de animaciones de este. El controlador contiene una serie de *layers* dispuestos para las diferentes animaciones necesarias dentro del juego. Tres *layers* para el movimiento sin arma, con espada y con arco, y cuatro *layers* con mascaras para poder mover ciertas zonas del modelo durante el juego, por ejemplo, tenemos las *layers RightArm* y *LeftArm* para poder mover solo los brazos para equipar y desequipar las armas.

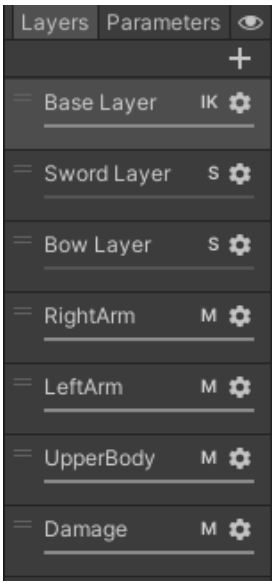

Ilustración 24. Layers del controlador de player

Las *layers Sword Layer* y *Bow Layer* son réplicas de *Base Layer*, ya que el movimiento funciona igual, pero solo cambia el tipo de animación, pero las transiciones y cambios de animaciones son las mismas.

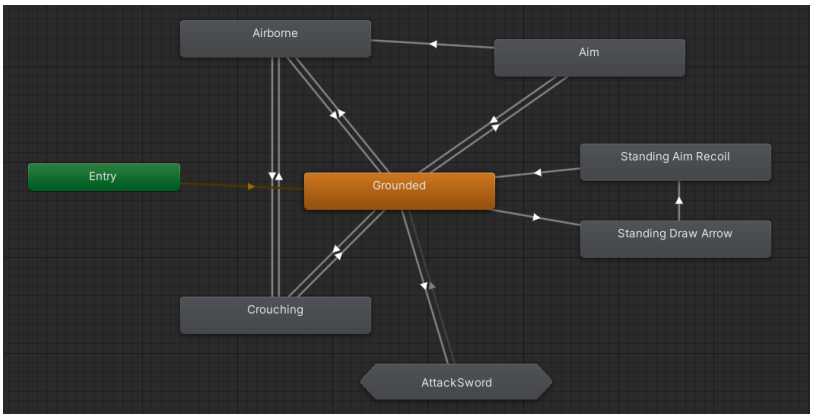

Ilustración 25. Animación del layer Base Layer, Sword Layer y Bow Layer

Cada uno de los nodos contiene un *blendtree* encargado del cambio de animación según los parámetros de cada momento del *animator.*

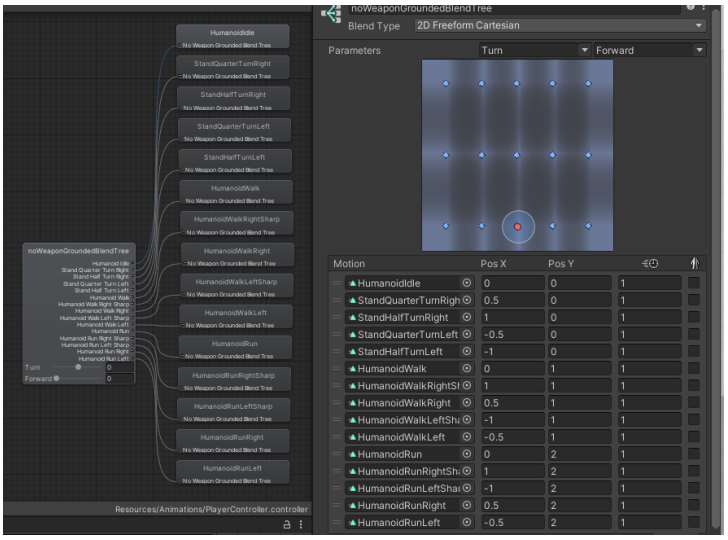

Ilustración 26. Blendtree de la animación Grounded del Base Layer

Rigidbody* y *Capsule Collider

Componentes de Unity necesarios para que el objeto tenga gravedad y tenga una zona para colisionar con objetos y no atravesarlos.

Third Person User Controller Script

Encargado de leer el input del usuario, y llamar al *script ThirdPersonCharacter* para que se mueva, salte, se agache, etc. Tiene una referencia a la cámara que para que el input del usuario siempre sea en relación a la vista de la cámara.

Third Person Character Script

Encargado de todo el movimiento del usuario, tiene referencias a todos los elementos que necesita para procesar el movimiento, como el *rigidbody*, el *animator* y el *capsulle collider.*

Recibe la llamada del *script* anterior con toda la información del input del jugador y con ello prepara una serie de valores que serán los parámetros del *animator*, con estos parámetros, el *animator* realizará una animación y otra. Y serán las animaciones quien se encargarán del movimiento del jugador.

Al añadir las animaciones a Unity, debemos marcar los *Root Transform* de estas para que modifiquen la posición o rotación del modelo acorde a la animación.

Weapon Controller Script

Controla todo lo relacionado con las armas, tiene una referencia a los objetos de las armas y en cuanto son equipadas o desequipadas las “asocia” a la mano del modelo para que se muevan a la par, simulando que el modelo tiene el arma sujetada.

En caso de que algún arma este seleccionada permite atacar y apuntar si el arma lo permite, modificando las animaciones del jugador. En el momento de atacar, activa el modo ataque del arma, para que al entrar en contacto con el enemigo le provoque una herida y si el jugador no está atacando el arma sea inofensiva.

Para realizar un ataque se han habilitado dos *BoxColliders*, uno es la punta de la espada y otro a lo largo de la flecha. Una vez el modo ataque se activa, se habilita que las armas ejerzan daño a enemigos, y con esto al entrar el *box collider* mencionado anteriormente con el enemigo, accede al *script EnemyLife* que controla la vida del enemigo y llama a la función *Hit* con el valor de daño del arma. De esta forma evitamos que el enemigo se haga daño al chocar con el arma cuando no estamos atacando.

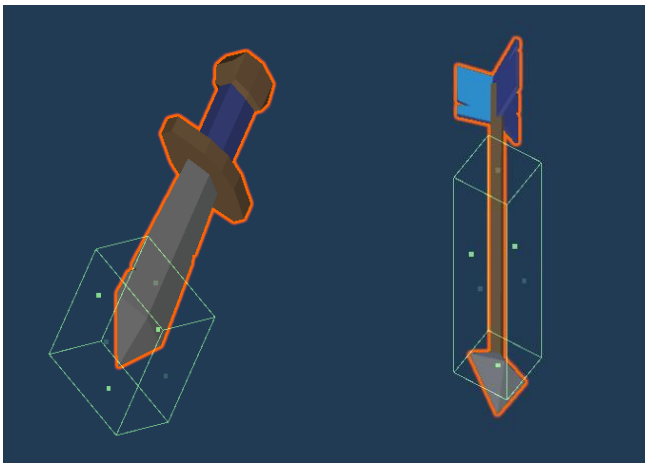

Ilustración 27. Armas con sus BoxColliders para atacar

Life Manager Script

Contiene una referencia al elemento de *UI* que muestra la vida, y controla cuanta vida tiene el jugador en cada momento. Cuando un enemigo le ataca, recibe una llamada a la función *Hit*, que reduce la vida del jugador, hace sonar un sonido de herida, actualiza la *UI* y en caso de que la vida llegue a cero, ejecuta el *GameOver*.

Audio Source

Fuente para los sonidos que emite el jugador, así como recibir una herida.

4.7 Personajes no jugables (PNJ)

Eyra es la encargada de ayudar en lo que puede al jugador, para ello se ha generado un controlador para que pueda conversar con el jugador y darle algo de soporte.

El modelo tiene un *Sphere Collider* en modo *Trigger* para cuando el jugador se acerca a Eyra pueda hablar con ella. Al entrar en el *collider* se habilita el modo hablar, que en clicar el botón interacción, se para por completo al jugador y se deshabilita su movimiento y el de la cámara.

Durante la conversación hay una serie de botones que nos permiten finalizar las misiones que no ha dado Eyra, y en finalizarlas se carga la escena final de la partida.

4.8 Enemigos

Los enemigos tienen una pequeña inteligencia artificial asociada, ya que son los enemigos del “primer” mundo del videojuego no son muy complejos ni fuertes. Se caracteriza por tener cuatro estados: *IdleState, FollowState, AttackState* y *ReturnState.*

IdleState comprueba la distancia con el enemigo y en caso de que se acerque a poca distancia entra en modo *FollowState.*

FollowState mediante un *NavMeshAgent* persigue al jugador constantemente, en caso de alejarse mucho entrará en modo *ReturnState* y en acercase al jugador entrá en modo *AttackState.*

ReturnState tiene guardada la posición inicial del enemigo y mediante el *NavMeshAgent* busca el camino más rápido para volver a este, una vez llegado a este punto, volverá al modo *IdleState.* Si durante su regreso, el jugador vuelve a acercarse a este, el enemigo volverá a entrar en modo *Follow*State.

En *AttackState* el enemigo siempre mira al jugador y le ataca cada cierto tiempo, en caso de que el jugador se aleje más de la distancia de ataque, el enemigo volverá al modo *FollowState.*

4.9 UI

Siguiendo la temática de islas y barcos. Se buscó crear una *UI* que siguiese un poco ese estilo y por ello se optó por usar madera y metal como estilo para esta.

Se comenzó con el título y el menú de la pantalla inicial, una vez ambos estaban decididos se pudo reproducir para los demás puntos donde era necesaria una *UI*. Así como la zona de conversación, el menú de configuración y la información de las armas del jugador.

Ilustración 28. UI del título del juego

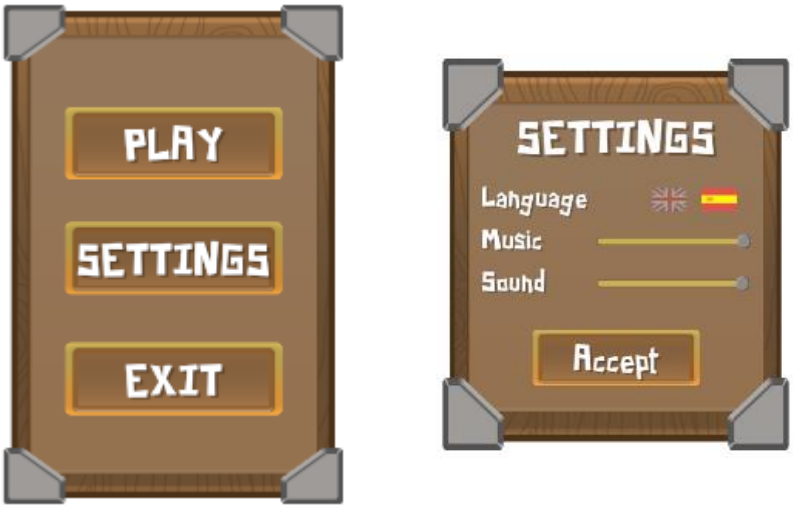

Ilustración 29. UI menú de inicio y configuración

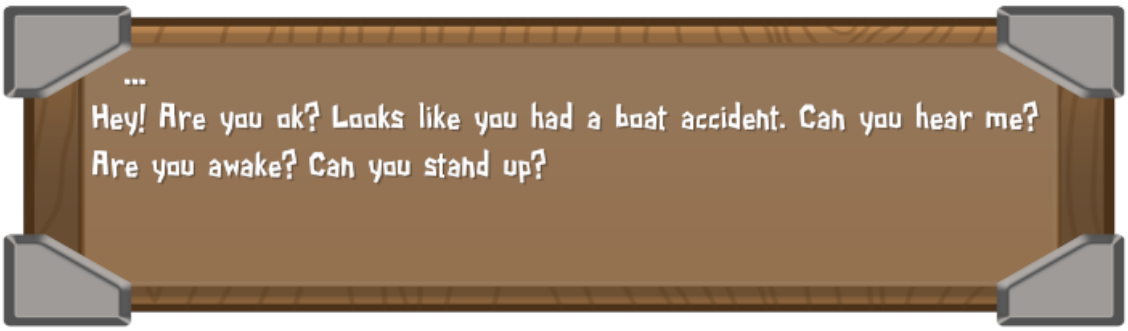

Ilustración 30. UI caja de conversación

Ilustración 31. UI armas y flechas disponibles

4.10 Configuración

Sonido

Para controlar el sonido, tanto del volumen del juego como de los ruidos dentro del juego, se ha creado un *AudioMixer,* que contiene dos grupos, uno para volumen y otro para sonidos. Exponiendo estos dos valores se consigue que desde un *script* podamos modificar los valores de estos sin tener relacionados los *AudioSource* de todos los componentes que emiten algún sonido dentro del juego.

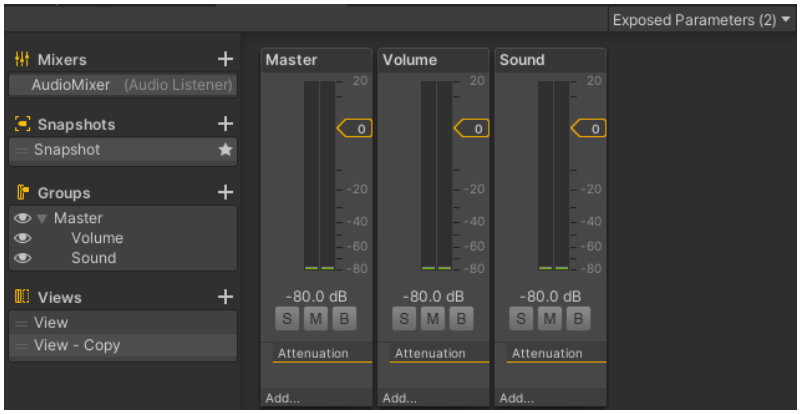

Ilustración 32. AudioMixer de Paper Ship

Sistema de idiomas

Para el control de idiomas se ha creado una serie de componentes que mantienen en todo momento el estado del idioma actual y en caso de

modificarse, actualizan todos los textos que hay en escena al nuevo valor en su respectivo idioma.

Para ello se ha creado el *script LanguageSystem* que tiene una lista de diccionarios (uno por cada idioma), donde para un identificador tiene su valor en ese idioma. Estos diccionarios se generan a partir de un archivo *XML* que contiene todos los textos en ese idioma.

en.xml

```xml
<language>
  <play>PLAY</play>
  <settings>SETTINGS</settings>
  <exit>EXIT</exit>
  <retry>RETRY</retry>
  <language>Language</language>
  <music>Music</music>
  <sound>Sound</sound>
  <accept>Accept</accept>
  <resume>Resume</resume>
  <wakeup>Wake Up</wakeup>
  <next>Next</next>
  <talk>Talk</talk>
  <grab>Grab</grab>
```

es.xml

```xml
<language>
  <play>JUGAR</play>
  <settings>AJUSTES</settings>
  <exit>SALIR</exit>
  <retry>REINTENTAR</retry>
  <language>Idioma</language>
  <music>Música</music>
  <sound>Sonido</sound>
  <accept>Aceptar</accept>
  <resume>Seguir</resume>
  <wakeup>Levantarse</wakeup>
  <next>Siguiente</next>
  <talk>Hablar</talk>
  <grab>Recoger</grab>
```

Ilustración 33. Ejemplo XML idiomas

Una vez tenemos el sistema de idiomas creado. Para cada elemento de texto que queramos actualizado por idiomas, le añadiremos el *script TextUpdate* que contiene un *Listener* para cada vez que se cambia el valor de idioma en el script *LanguageSystem.*

Si este *Listener* salta, significa que deberemos modificar nuestro valor al nuevo que tenga el sistema de idioma en el diccionario actual.

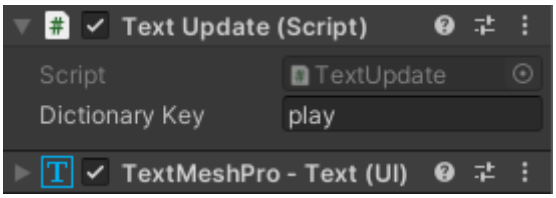

Ilustración 34. Ejemplo componente texto con script TextUpdate

5. Diseño de niveles

Desde primera instancia se planteó realizar el diseño de niveles mediante bocetos dibujados a mano y sobre los cuales poder reproducir más tarde en Unity. Una vez encontrados los *assets* finales para la isla, se decidió directamente crear el mapa de la isla sobre Unity desde cero.

Primero de todo se comenzó creando la base de la isla, dejando una zona abierta para añadir mas adelante el mar y una explanada donde iría nuestra ciudad. Debido a que el movimiento del mar suele ser costoso, se optó por hacer una isla rodeada de montañas, donde solo se puede acceder desde un solo punto al mar.

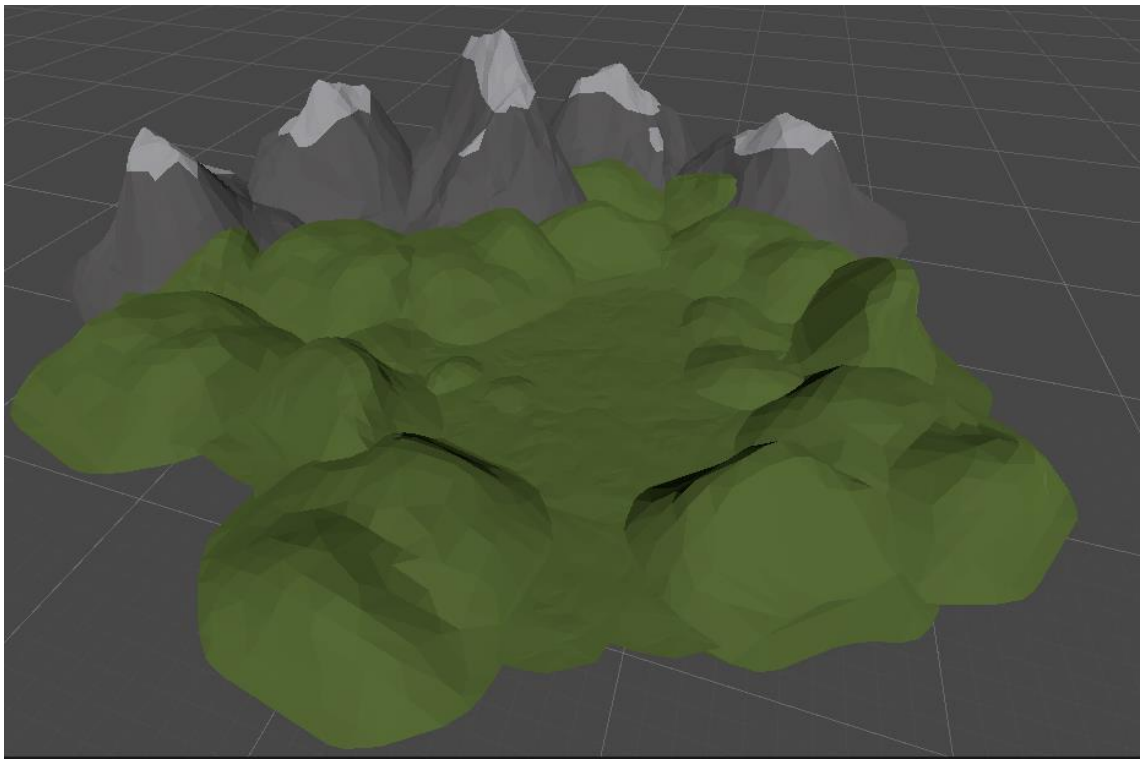

Ilustración 35. Isla sin ningún tipo de estrutura

Con la isla más o menos generada, se añadió el personaje para que recorriese la isla y ver si por algún punto se podía “salir” de la isla. Una vez el mapa esta asegurado, el jugador no puede salir de la zona de juego, se añadió el mar y la playa, donde el jugador más adelante empezaría la partida. Para el mar se utiliza un plano que mediante un script oscila simulando las olas del mar y para que el jugador no pueda entrar a este, se le añade un *BoxCollider* que actúa de muro al acercarse al mar. De nuevo, asumiendo que el mar consumirá bastantes recursos se ha optado por reducir su tamaño al máximo para que ocupe la zona visible y parezca que llega hasta el horizonte.

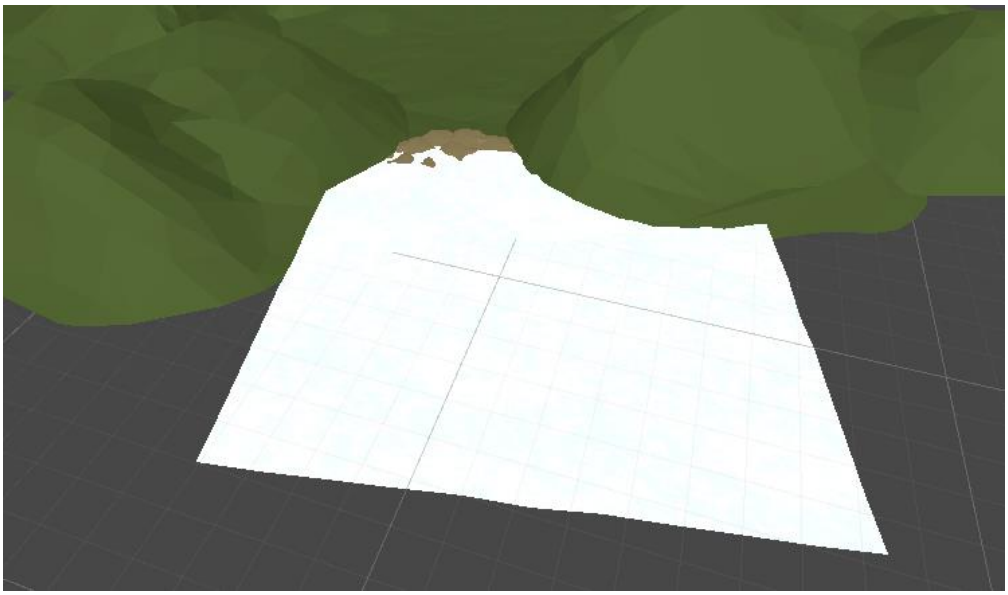

Ilustración 36. Isla con zona de playa y mar añadido

A continuación, se han añadido los edificios de la isla para tener una idea clara de como será la distribución de la isla y de sus principales puntos de interés.

Ilustración 37. Isla con edificios

Por último, se han añadido unos caminos y algunos *props* como rocas y plantas para darle un aspecto más vivo a la isla. También se añadió un barco de papel en la zona de la playa para representar al barco del protagonista.

Ilustración 38. Mapa de Kattegat finalizado

Con el mapa generado, esta misma isla, será utilizada para las escena incial del juego, donde el jugador mira al horizonte, la escena de introducción, donde habla con Eyra sobre lo que ha pasado, y la escena del juego, donde podrá moverse libremente por el mapa.

Solo resta, añadir los objetos que el jugador debe recoger en la partida y a los enemigos. Empezando por los objetos, estos se han esparcido por todo el mapa para que el jugado haya de recorrerse toda la ciudad.

Ilustración 39. Distribución de objetos dentro del mapa

Con todos los objetos que el jugador ha de recoger, se añadireon enemigos protegiendolos y algún que otro enemigo dispersado por el mapa para añadir un poco de reto al jugador.

Ilustración 40. Mapa con todos los enemigos añadidos

6. Manual de usuario

6.1 Requerimientos técnicos

Sistema operativo:

- Windows 7 (SP1+), 8, Windows 10, ambos de 64-bit
    - CPU: Soporte para el set de instrucciones SSE2.
    - GPU: DX10, DX11, o DX12.

Almacenamiento disco duro: 300 MB

6.2 Instrucciones

Para abrir el juego deberemos ejecutar el archivo “Paper Ship.exe” y una vez abierto se nos mostrará que el juego ha sido realizado utilizando el *engine* de Unity ya que hemos usado la versión gratuita que no elimina esta pantalla.

Ilustración 41. Ejecutable Paper Ship

Una vez finaliza el visual de Unity, llegaremos a la pantalla del menú de inicio donde podremos ver las opciones de este a mano derecha.

Ilustración 42. Escena inicial de Paper Ship

Si clicamos en el botón de ajustes se nos abrirá un menú donde podremos modificar el idioma del juego (disponible en inglés y castellano), así como modificar el volumen del juego y de los sonidos. Este menú también lo encontramos durante todo el juego cuando accedemos al menú de ajustes.

Ilustración 43. Escena inicial con menu de configuración

Si clicamos a jugar, el juego comenzará, la primera pantalla se nos explicará donde estamos y un poco más de la historia, simplemente pulsando el botón de interacción (Tecla E) que nos indica por pantalla, podremos continuar hacia adelante hasta acabar la introducción del juego.

Ilustración 44. Escena de introducción hablando con Eyra

Una vez acabada la introducción se deja libre al jugador y los controles se vuelven un poco más complejos:

- **Movimiento personaje**: Utilizar las teclas W A S D para moverse.
- **Movimiento cámara**: Mover el ratón para cambiar hacía donde mira la cámara. Con la rueda del ratón podremos acercar y alejar la cámara del jugador.
- **(Des)Equipar espada**: Tecla 1
- **(Des)Equipar arco**: Tecla 2
- **Atacar**: Clic izquierdo del ratón cuando tenemos algún arma equipada.
- **Apuntar**: Con el arco equipado, clic derecho del ratón.
- **Hablar/Recoger**: Tecla E
- **Pausa**: Tecla Escape

7. Conclusiones

7.1 Lecciones aprendidas

Durante la realización de este proyecto se han aprendido una gran cantidad de nuevos conocimientos, así como profundizar y pulir aquellos que ya tenía con anterioridad:

- Una de las cosas más importantes que he aprendido es la gran dificultad que conlleva realizar un juego completo. Cuando se juega a un juego no se les da mucha importancia a los pequeños detalles y cree que crear el juego es mucho más sencillo de lo que en realidad es. Con este proyecto me he dado cuenta la gran cantidad y dificultad que conlleva realizar un proyecto como este.
- Esto es algo que me hubiera gustado saber al iniciar el proyecto y es a saber valorar mejor cada parte del proceso de creación de un videojuego, en el punto 7.2 se entra en más detalle, pero uno de los principales problemas durante todo el proyecto es el escaso conocimiento de algunos aspectos al iniciar el proyecto que resultaron en una mala planificación.
- Conocimientos de Unity. Con cada videojuego nuevo realizado se aprende algo nuevo de este extenso y completo *game engine.* En este caso, crear desde cero todo el movimiento y mecánicas del juego me han ayudado a aprender mucho más como funciona este *engine* a bajo nivel.
- Editar los gráficos de un juego. Es típico no darle mucha importancia al estilo visual y centrarse en el juego. Para el proyecto quería que la estética estuviese bien definida y cumpliese con lo pensado en la idea inicial. Gracias a esto he aprendido a utilizar la herramienta *Post Processing Stack*, bastante utilizado en el sector, y que aporta muy buenos resultados al juego.
- Mejor conocimiento de animación y de *blendtrees.* Este es uno de los puntos en los que más me ha gustado trabajar, gracias a las animaciones el personaje gana muchísima vida y acciones dentro del juego.
- Por último, creo que he aprendido a tener una mejor estructuración y separación de *scripts* y *prefabs* dentro de Unity. Durante todo el proyecto muchos elementos se podían reutilizar y una buena separación de *scripts* era necesaria para que todo estuviese bien estructurado.

7.2 Objetivos iniciales

Aunque el resultado final ha sido bastante cercano a los objetivos iniciales, la verdad es que ha habido muchísimos cambios al plan inicial. El principal motivo de todos estos cambios viene dado de la falta de conocimiento inicial de la

dificultad de cada una de las partes del proceso de creación de un videojuego completo.

El principal desconocimiento que más repercutió en el plan inicial y que modificó un poco el resultado final, fue el hecho de no tener ningún tipo de conocimiento de modelado 3D, en un principio se tenia pensado generar desde cero todos los habitantes de la isla y la isla entera. Al ver que esto no era factible con el limite de tiempo establecido y el poco conocimiento en el área, se optó por utilizar modelos gratuitos de la *Unity Asset Store* para la isla y los enemigos. Ya que se optó por realizar los modelos de personas a mano, una vez finalizados el protagonista y Eyra, el hecho de añadir gente en el pueblo no era factible, y los *assets* disponibles en el mercado no se parecían suficiente a la estética de los personajes del juego.

Por ello se redujeron algunos objetivos y se centró en generar un mapa sencillo, donde la única interacción es con Eyra y no con todo el pueblo.

El resultado final se asemeja bastante a la idea inicial, tan solo se han reducido algunos elementos de esta.

7.3 Planificación y metodología

En primera instancia se realizó una planificación en base a los tres hitos a entregar durante el proyecto. En base a esta planificación se inició un proceso Kanban para realizar las tareas de una en una por orden de prioridad. Al final el primer entregable, y con una nueva idea de lo que se tendría que sacrificar del plan inicial, se fueron modificando y eliminando las tareas pendientes para acercarse a un producto final óptimo.

Para ello se utilizó el programa *Super Productivity*, una herramienta para controlar y gestionar tareas, así como calcular el tiempo pendiente para una tarea, cuanto tiempo ha llevado una tarea, etc. Gracias a esta herramienta se pudieron priorizar tareas de mayor peso y trabajar antes con ellas. También al ver el tiempo consumido en algunas tareas se vio como algunas de las tareas habían sido estimadas a la baja.

7.4 Trabajo futuro

Se presentan las siguientes líneas de trabajo futuro para continuar con este proyecto:

- Añadir más jugabilidad: Más islas para poder continuar explicando un poco de historia en cada isla. Más personajes con los que poder interactuar y conocer más sobre el mundo del videojuego. Sistema de inventario y misiones más complejas.

- Modificar todos los *assets* utilizados de internet por unos creados a mano, para darle más personalidad al juego y que el jugador no pueda encontrar los mismos objetos exactos en otro juego disponible online.
- Incorporación de otros Inputs: Con la incorporación de otros inputs podremos permitir al jugador, utilizar el que más le convenga y de tal forma poder añadir más plataformas para sacar el juego como consolas.
- Pulir detalles: Tanto las animaciones como la música y sonidos, no son 100% perfectos y mejorándolos le darían un toque más profesional al videojuego y el jugador se sentiría más inmerso en éste.

8. Glosario

A continuación, se definen diversos términos que han sido usado durante el desarrollo de la memoria:

- **Roadmap**: Término en inglés para referirse a la hoja de ruta o plan estructurado a lo largo del tiempo.
- **ARPG**: Siglas en inglés de juegos de acción de rol. (Action Role-playing games)
- **Git**: Sistema de control de versiones open source.
- **Game engine**: Término en inglés que significa motor de videojuego.
- **Script**: Archivo que contiene el código de programación.
- **Gameplay**: Video donde se muestra parte del juego y su jugabilidad.
- **Low poly**: Término en inglés para referirse a un elemento que tiene pocos polígonos.
- **High poly**: Término en inglés para referirse a un elemento que tiene muchos polígonos.
- **UI**: Interfaz de usuario.
- **Directional light**: Término en inglés para luz direccional.
- **GameObject**: Objeto base de Unity. Por lo general no tienen ninguna funcionalidad hasta que no se les añade algún componente.
- **Fog**: Término en inglés para niebla.
- **Props**: Propiedades de un modelo, suele referirse a todo lo ajeno al cuerpo del modelo. Por ejemplo: pelo, ropa, armas, joyas, etc.
- **UV**: Siglas utilizadas para referirse al texturizado en 3D.
- **Layer:** Término en inglés que significa capa.
- **PNJ**: Siglas para personaje no jugador.
- **Rigging**: Término en inglés para referirse al modelado de huesos en 3D.
- **Online**: Término en inglés que significa en línea.
- **Pop up**: Término en inglés para referirse a algo que aparece de la nada en la pantalla.
- **Humanoid**: Término en inglés que significa humanoide.
- **EventSystem**: Sistema de eventos de Unity.
- **Canvas**: Objeto de Unity donde se colocan todos los elementos de la UI.
- **Prefab**: Objeto reutilizable en Unity. Se crea a partir de un GameObject cualquiera.
- **Asset**: Ítem que puede ser utilizado en un videojuego.
- **Idle**: Término en inglés para referirse a estar quieto.
- **SphereCollider**: Elemento de Unity con forma de esfera que detecta cuando otro elemento entra en contacto con él.

9. Bibliografía

1. Videojuego de rol de acción, Wikipedia [En línea]. Disponible en: https://es.wikipedia.org/wiki/Videojuego_de_rol_de_accion [Último acceso: 1 enero 2021]
2. Kanban – A brief introduction, Atlassian [En línea]. Disponible en: https://www.atlassian.com/agile/kanban [Último acceso: 30 noviembre 2020]
3. Scripting API: Animation, Unity Docs [En línea]. Disponible en: https://docs.unity3d.com/ScriptReference/Animation.html. [Último acceso: 8 diciembre 2020]
4. Animation Layers, Unity Docs [En línea]. Disponible en: https://docs.unity3d.com/Manual/AnimationLayers.html. [Último acceso: 15 diciembre 2020]
5. Blend Trees, Unity Docs [En línea]. Disponible en: https://docs.unity3d.com/Manual/class-BlendTree.html. [Último acceso: 15 diciembre 2020]
6. How to architect code as your project scales, Unity [En línea]. Disponible en: https://unity.com/how-to/how-architect-code-your-project-scales [Último acceso: 10 diciembre 2020]
7. Paint skin weights, Autodesk Knowledge [En línea]. Disponible en: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-CharacterAnimation/files/GUID-7D773C38-F9CF-4141-8ADD-4E0454838BB7-htm.html [Último acceso: 27 noviembre 2020]