Universitat Oberta de Catalunya

uoc.edu

Hermit King

Jose Manuel Picó Gómez
Máster Universitario en Diseño y Programación de Videojuegos
Trabajo Final de Máster

Consultores:
Helio Tejedor Navarro
Jordi Duch Gavaldà

Profesor responsable de la asignatura:
Joan Arnedo Moreno

Junio 2021


Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Hermit King*
Nombre del autor:	*Jose Manuel Picó Gómez*
Nombre del consultor/a:	*Helio Tejedor Navarro*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Máster Universitario en Diseño y Programación de Videojuegos*
Área del Trabajo Final:	*Trabajo Final de Máster*
Idioma del trabajo:	*Castellano*
Palabras clave	*Videojuego, desarrollo, simulación*

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

En este documento se trata la planificación y desarrollo de un videojuego rogue-like para ordenador en tercera persona sobre cangrejos ermitaños realizado con el motor Unity, con un particular uso de las conchas que pueden equipar.

Entre otros, se ofrece información acerca de la elaboración de la IA de los cangrejos enemigos para que ataquen e interactuen con el jugador y las conchas, animación procedural de las patas de los cangrejos, elaboración de shaders y lógica general que compone los distintos sistemas del juego.

También se exponen distintas decisiones de diseño en las que ha primado la accesibilidad y la cercanía al comportamiento de los ermitaños reales, así como las referencias que se han tenido en cuenta durante el proceso en cuanto a jugabilidad y estética.

En el apartado de conclusiones se realiza una valoración del producto final y el propio desarrollo. Si bien se ha conseguido un juego aparente, accesible y llamativo en la línea de lo planificado, han surgido algunos problemas que han ido alterando su desarrollo y resultado en detrimento del entretenimiento que ofrece en su versión final. Se comentan también las maneras en las que se podría suplir en futuras iteraciones estos fallos así como el proceso que se seguiría para realizar un lanzamiento comercial indie del juego.

Abstract (in English, 250 words or less):

This paper deals with the planning and development of a third-person rogue-like computer game about hermit crabs made with Unity engine, with a particular use of the shells they can equip.

Among others, information is given about the elaboration of the AI of the enemy crabs to attack and interact with the player and the shells, procedural animation of the crabs' legs, shader elaboration and general logic that composes the different systems of the game.

Different design decisions in which accessibility and the real behavior of hermit crabs have been a priority, as well as the references that have been taken into account during the process in terms of playability and aesthetics, are also presented.

In the conclusions section, an evaluation of the final product and the development itself is made. Although an apparent, accessible and striking game quite pretty much as planned has been achieved, some problems have arisen that have been altering its development and result to the detriment of the entertainment offered in its final version. It also discusses ways in which these flaws could be overcome in future iterations as well as the process that would be followed to make an indie commercial release of the game.

Índice general

Índice de figuras

1 Introducción

En los últimos años se ha vivido un incremento dentro del mundo de los videojuegos independientes de los denominados rogue-likes. Estos juegos muy centrados en el concepto de la muerte permanente y en la generación procedural son muy diversos entre sí dando pie a un gran número de obras originales e influyentes como *The Binding of Isaac*, *Enter the Gungeon o Hades*. A lo largo de este proyecto se ha buscado crear un título que siga las bases de este popular género pero dándole un toque distintivo que consiga diferenciarlo de los demás.

Hermit King es un juego basado en el estilo de vida de los cangrejos ermitaños. Aprovecha muchas de las particularidades que tienen estos crustáceos para presentar un gameplay entretenido en el que debe crecer para convertirse en el cangrejo más grande de la playa: el Rey Ermitaño. El jugador encarnará a un cangrejo que deberá ir encontrando los pedacitos de comida que están diseminados por el mapa para poder crecer. El cuerpo de los ermitaños está desprotegido en su parte trasera, por lo que en en su travesía nuestro protagonista podrá habitar distintas conchas para protegerse. La playa está plagada de otros cangrejos que deambulan en una búsqueda continua de una concha mejor y pueden llegar a reaccionar hostilmente con el jugador. Nuestro cangrejo contará con varias habilidades que le permitirán atacar y defenderse para poder derrotar a los demás cangrejos.

A medida que crece, nuestro personaje necesitará cambiar de concha y buscar una nueva más grande. Hay distintos tipos de conchas que modifican la velocidad de movimiento y protección con las que cuenta el cangrejo y que ofrecen distintas formas de abordar los enfrentamientos aprovechando sus ventajas. Al recibir varios golpes, los cangrejos sueltan su concha, pudiendo ser robada por otros ermitaños. En el momento en el que un cangrejo sin concha es golpeado, morirá. Si se trata del cangrejo del jugador, la partida volverá a comenzar desde cero y la posición de la comida, enemigos y conchas variará. En cambio, si consigue crecer lo suficiente como para entrar en la gran concha dorada que hay en el centro del mapa, se hará con la victoria.

El juego ha sido diseñado con la accesibilidad y simpleza de los elementos en mente. Se ha pretendido crear un título que sea sencillo de aprender a controlar, que dé facilidades para poder probarlo y que no agobie al usuario con una gran cantidad de elementos a tener en cuenta. No obstante, también se ha procurado crear una pequeña progresión de manera que dé cabida a la mejora personal, tanto del conocimiento de los comportamientos que forman el juego como de la habilidad necesaria para llegar más lejos en la partida, algo muy propio del género rogue-like.

El desarrollo del juego se ha realizado con el conocido motor de videojuego Unity dada la familiaridad que se tiene con esta aplicación. A lo largo de este documento se presentarán todo tipo de detalles de las partes más relevantes del proceso de implementación como pueden ser la creación de la máquina de estados que rige el comportamiento de la Inteligencia Artificial de los cangrejos enemigos, la elaboración del sistema de animación procedural de las extremidades de los cangrejos que permite adaptarse a las condiciones del suelo, la creación del shader de agrietado progresivo de las conchas, o la estructuración en clases que se ha utilizado para organizar la lógica del juego.

También se presentarán decisiones del diseño de niveles, gameplay y de la estética escogida junto con referentes ya existentes de los cuales se ha tomado inspiración. A lo largo de estas secciones se tratan temas interesantes como la elección de las habilidades del cangrejo, la preparación de la escena inicial con un tutorial diegético que facilita la entrada a los nuevos jugadores, o el cómo se ha ido dando forma al estilo estilizado del arte del juego.

Por último, se hará una retrospectiva del proyecto y se analizará el producto final. Si bien el resultado está en la línea de lo que se había planificado inicialmente, durante la implementación han surgido inconvenientes de distintos tipos que han ocasionado desvíos en el desarrollo. La versión final cuenta con fallos, tanto en el funcionamiento como en el propio gameplay. Durante este apartado se comenta cómo se realizaría una futura iteración más profesional del título estudiando estos fallos, aportando soluciones y mejoras sustanciales. Se elaborará también el proceso que habría que seguir para realizar un posible lanzamiento comercial independiente.

2 Estado del arte

2.1 Género Rogue-Like

Hermit King se trata de un juego difícil de categorizar, sin embargo se podría englobar dentro del género de los *rogue-likes*. Este género, a pesar de estar considerado como algo de nicho, se ha hecho mucho más popular desde hace unos años en el terreno de las producciones independientes de bajo presupuesto o *indies*. Actualmente se puede encontrar en el mercado un gran número de obras que están influidas por sus bases como *Spelunky* o *Nuclear Throne*.

2.1.1 Historia del género

El término rogue-like proviene del juego para ordenador *Rogue* de 1980, que se trataba de un sencillo juego de mazmorras con gráficos ASCII que se jugaba en una terminal. El juego consistía en atravesar numerosos pisos de una mazmorra plagados de enemigos para encontrar el amuleto de Yendor situado al fondo y volver a subir intentando salir de la mazmorra con vida. La peculiaridad de Rogue era su generación procedural de mazmorras así como la muerte permanente del jugador, algo que si bien se basaba en conceptos presentados por juegos anteriores, resultaba muy innovador para la época (Moss (2020)).

Las partidas comenzaba completamente desde cero, no gozando de las ventajas de ningún tipo de progreso obtenido en partidas anteriores. El mapeado se generaba completamente desde cero, y se variaba la disposición de enemigos e ítems en cada partida, generando una cantidad de contenido muy amplia dada la variabilidad del algoritmo de generación. Además, los objetos del juego eran indistinguibles entre sí antes de cogerlos, lo cual limitaba la progresión a largo plazo al incremento de la experiencia real y habilidad que iba obteniendo el jugador en apartados como el combate y el uso de objetos.

El juego ganó bastante popularidad y enseguida aparecieron juegos inspirados en él que mejoraban aspectos como la apariencia, ofreciendo una interfaz gráfica más accesible y natural que los caracteres ASCII o añadiendo contenido al juego como nuevos enemigos, tipos de interacción y objetos. De esta manera surgieron los denominados juegos rogue-like. Estas modificaciones se fueron haciendo cada vez más variadas mezclando conceptos de otros géneros dando pie a juegos enormemente dispares entre sí.

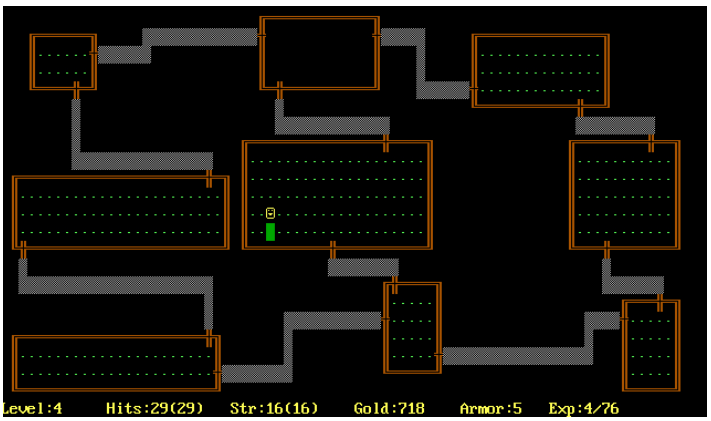

Figura 2.1: Captura del clásico juego *Rogue* lanzado en 1980

Un ejemplo muy conocido sería el clásico *Diablo* que contaba con un sistema de generación de misiones pseudo-aleatorias. Acercó mucho más el género al público general haciéndolo más accesible dotándolo de cuidados gráficos isométricos y centrando su experiencia en un combate más sencillo que descartaba los turnos. En contrapartida tenemos a *Dwarf Fortress* que se trata de un juego increíblemente complejo, que integra una enorme cantidad de sistemas distintos que generan un mundo con una cantidad de información abrumadora. Mantiene los gráficos en ASCII (en su versión original) y se trata de un rogue-like de final abierto más centrado en la simulación y construcción de colonias.

Figura 2.2: *Diablo 1*

Figura 2.3: *Dwarf Fortress*

Cabe a destacar el papel de los desarrollos *indies* en esta evolución del género que ha sucedido gracias al incremento de popularidad que ha sufrido este sector durante la última década. Han aparecido juegos como *Spelunky* o *The Binding of Isaac* que han creado escuela y que, a pesar de evidentemente ser muy distintos a la obra original Rogue, sí que están claramente influenciados por ella. Spelunky por ejemplo es un juego de plataformas, que poco o nada tiene que ver con esa exploración cenital de mazmorras con combate por turnos. Sin embargo, se mantiene la esencia de que cada partida es completamente diferente al ser generada proceduralmente, la estructuración en pisos distintos y que la muerte significa perder todo el progreso.

Figura 2.4: *Spelunky*

Figura 2.5: *The Binding of Isaac: Rebirth*

2.1.2 Rogue-likes y rogue-lites

La creciente popularidad del género ha dado lugar a numerosas variaciones e interpretaciones de la bases que presentó Rogue en su momento. Muchas de estas variaciones han abogado por mejorar la accesibilidad y minimizar el castigo que supone la derrota en los juegos del estilo, surgiendo así el nuevo concepto de rogue-lite.

Los rogue-lite mantienen el esquema básico de juego en el que al morir volvemos al punto de partida, teniendo una generación del mapeado distinta y única en cada partida. Sin embargo, en esta variante el reinicio de la partida no nos devuelve exactamente al mismo punto, si no que podemos aprovecharnos del progreso obtenido en partidas anteriores para hacer más fácil la siguiente partida. Se establece así una progresión por diseño entre partidas de la cual carecían sus predecesores.

Los rogue-likes estrictos buscan una progresión exclusivamente de la habilidad del jugador, que vaya aprendiendo de sus errores, vaya conociendo los diferentes elementos que conforman el juego y aprenda las dinámicas necesarias para alcanzar la victoria. Esto, de cara a las experiencias de cierto sector de jugadores puede llegar a resultar frustrante y tedioso. Si un jugador se queda estancado y no percibe una sensación de progresión en el juego, puede llegar a perder el interés por el mismo y no llegar a ver todo lo que el juego tendría que ofrecerle. Los rogue-lites consiguen que el jugador deba mejorar para superar el juego pero en el caso de que no consiga avanzar en varias partidas, al menos tendrá un aliciente de ciertas recompensas obtenidas que le harán la siguiente partida más sencilla, consiguiendo así una curva de dificultad más suavizada que los hace más accesibles para cierto sector de público (Brown (2019)).

En contrapartida, los rogue-lites pierden esa sensación de superación personal tan pura, haciendo que la consecución de los diferentes objetivos en la partida no sea únicamente debida a la mejora como jugador, si no que ha sido, en gran parte, por dedicarle más tiempo al título para obtener una mejora que haga más sencillo el juego. Los rogue-lite pueden causar una sensación de artificialidad, ya que al contrario que los rogue-like, están diseñados para que al

principio del juego sean prácticamente insuperables ofreciendo muchas barreras por diseño, haciendo obligatoria esa dedicación de tiempo en pos de la habilidad.

Hagamos una comparación entre dos juegos relevantes de cada variante: *Nuclear Throne* (rogue-like) y *Rogue Legacy* (rogue-lite). En Nuclear Throne un jugador experimentado puede jugar en un dispositivo en el no haya ningún progreso guardado y llegar al final del juego en su primera partida, mientras que un jugador que no haya jugado a Nuclear Throne previamente perdería inevitablemente. Sin embargo, si un jugador experimentado en Rogue Legacy elimina su progreso guardado y parte desde el principio, si bien es técnicamente posible superarlo a la primera, se encontrará con un sin fin de trabas que le parecerán engorrosas como por ejemplo que tarda una eternidad en eliminar a los enemigos por que no ha tenido la opción de aumentar el daño de su personaje, haciendo obligatorio tener que jugar un gran numero de partidas para simplemente conseguir los recursos necesarios a pesar de contar con la habilidad necesaria para completarlo (Foreman (2016)).

Figura 2.6: Captura de una partida en *Nuclear Throne*

No obstante, la mayoría de títulos que salen al mercado hoy en día suelen hacer una hibridación entre los dos extremos. Por ejemplo, juegos como Nuclear Throne o The Binding of Isaac, a pesar de tener un estilo de juego completamente Rogue-like, sí que cuentan con desbloqueables que se pueden conseguir para las siguientes partidas, que si bien no las hacen más fáciles de superar, otorgan mayor variedad al juego y favorecen esa sensación de progresión. La separación entre estas variantes es algo difusa y todavía hay discusiones entre los límites que definen a cada vertiente, llegando a haber opiniones que consideran rogue-lite a cualquier juego del estilo que muestre cualquier tipo de progreso como los mencionados previamente (Bycer (2019)).

Figura 2.7: Árbol de habilidades del personaje que aparece entre partidas en el título *Rogue Legacy*

2.2 Motores de videojuegos

El motor de un videojuego sería el núcleo del programa sobre el que implementar el juego en sí y que a su vez se divide en un gran número de componentes muy distintos como el motor gráfico que permite mostrar la escena por pantalla, el sistema de físicas que controla todos los cálculos y comportamientos de los objetos en el mundo virtual relativos a la física, o el componente de red que permite establecer una comunicación entre distintos dispositivos. El motor de un videojuego puede ser muy diferente en función del tipo de juego al que vayan dirigidos, pero siempre habrá varios componentes y funcionalidades básicas en común (Montoya (2020)).

Durante los primeros años de existencia de los videojuegos era necesario crear prácticamente desde cero el motor para poder implementar el juego. Con el tiempo se ha ido agilizando este proceso mediante la aparición de distintas herramientas que se podían combinar, evitando tener que hacer partes del desarrollo que ocupaban mucho tiempo, permitiendo así también crear juegos más punteros. Actualmente, contamos con motores de videojuegos comerciales a los que podemos acceder, por lo general, de manera sencilla y económica y que nos ofrecen una base muy adelantada sobre la que simplemente implementar nuestra lógica de juego y poder configurar los módulos que lo integran a nuestro gusto.

Hay una gran cantidad de motores a la disposición de los desarrolladores, llegando incluso a existir motores especializados en el desarrollo de un género en concreto de juego como *RPG Maker*. A continuación, comentaremos los motores más relevantes que están a disposición de un desarrollador independiente para el tipo de juego que vamos a elaborar.

2.2.1 Unreal Engine

Uno de los motores de juego más populares es *Unreal Engine* creado por *Epic Games*. Fue lanzado en 1998 y desde entonces se ha convertido en el motor comercial más utilizado en producciones AAA[1], como las sagas *Gears of War*, *Mass Effect*, y *Bioshock* entre muchos otros. Sin embargo, también es bastante utilizado en desarrollos *indies* de bajo presupuesto como *Hello Neighbor*, *A Hat in Time* y *Ashen*.

Se le considera como uno de los motores actuales mas punteros en el apartado gráfico, contando con un sorprendente sistema de partículas que es capaz de manejar hasta un millón de partículas en una escena. También ofrece una completa libertad para modificar sus componentes, lo cual permite adaptar el motor perfectamente a las necesidades del juego, obteniendo un mejor resultado. Otro de sus grandes atractivos sería la posibilidad de portar el juego a numerosas plataformas de una forma muy sencilla, lo que consigue ahorrar mucho trabajo de desarrollo, aminorar los costes del proyecto y aumentar la difusión del producto (EpicGames (2021)).

En contrapartida, se considera como un motor más difícil de utilizar respecto algunos de sus competidores como *Unity*. No obstante, desde hace unos años hasta ahora ha mejorado bastante en este aspecto, contando actualmente con un sistema de programación por *blueprints* que permiten configurar la lógica del juego sin necesidad de programar a cambio de no acceder a todas las funcionalidades que ofrece el motor.

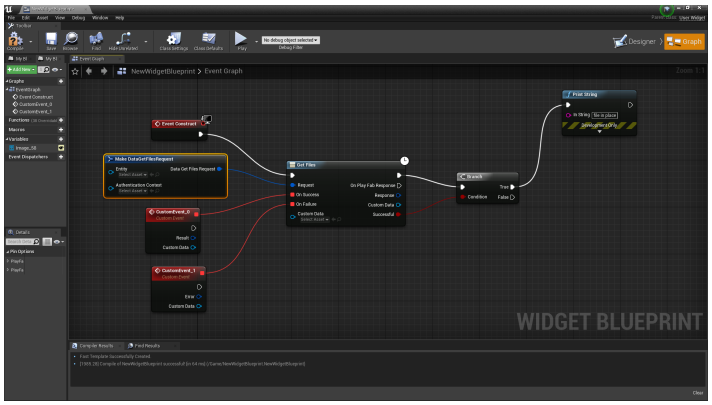

Figura 2.8: Interfaz de el diseño con *blueprints* que incluye Unreal Engine

Su precio también resulta llamativo, ya que cuenta con diversos planes de pago para adaptarse a las necesidades del equipo de desarrollo, permitiendo su uso completamente gratuito para productos internos no comercializados y estudiantes. En el caso de lanzar un videojuego con Unreal Engine es gratuito hasta alcanzar el millón de dólares estadounidenses de ingresos, a partir del momento en el que comenzarán a recibir un 5% de los ingresos del juego. Al

[1]Se considera videojuego AAA o triple A a grandes producciones realizadas por empresas de gran envergadura que suelen ser bastante conocidas en el sector y que cuentan con un presupuesto muy elevado y multitud de personal

margen de este plan básico existe otro plan de 1500$ anuales por puesto de trabajo y otro plan con un acuerdo personalizado negociable.

2.2.2 Unity

Actualmente, el motor más popular entre los desarrolladores independientes es Unity estando presente en muchos juegos famosos como *Hearthstone*, *Ori and the Blind Forest* o *Firewatch*. Se trata de un motor más sencillo de utilizar que la mayoría de sus competidores, otorgando también muchas herramientas de modificación del componentes del motor. Además, cuenta con una comunidad muy activa y extensa, lo cual facilita todavía más que se pueda encontrar información acerca de cómo utilizarlo, así como que exista una gran cantidad de herramientas a disposición del desarrollador que añaden funcionalidades al motor base.

Figura 2.9: Interfaz de Hearthstone

Se trata además de un motor económico, ya que su uso es gratuito hasta alcanzar cierto nivel de negocio, lo cual lo hace muy atractivo para pequeños desarrolladores o estudiantes del medio. En el momento en el que la empresa o equipo de desarrollo supere ingresos o fondos de más de 100.000 dólares estadounidenses anuales comenzará a cobrar una cantidad por puesto de trabajo. Esta cantidad oscila desde 399$ anuales por puesto (si se superan los 100.000$ de recaudación) a 2.000$ mensuales por 10 puestos de trabajo en el caso de contar con un equipo formado por más de 10 personas y tener ingresos superiores a 200.000$ (UnityTechnologies (2021)).

Sin embargo, Unity cuenta con el inconveniente de tener oculto su núcleo, al contrario que sucedía con Unreal Engine. Esto implica que la capacidad de modificación de los módulos internos del motor está muy limitada, lo cual puede resultar una traba para proyectos más ambiciosos que necesiten reestructurarlos para obtener mejoras en apartados como el rendi-

miento y los gráficos. Desde Unity se ha ido intentando ofrecer cada vez más opciones de modificación de este aspecto, pero sigue sin proporcionar la libertad con la que sí cuentan algunos de sus competidores.

2.2.3 Cryengine

CryEngine es otro de los motores más potentes y dominantes que tenemos hoy en día. Diversos juegos del mercado lo utilizan actualmente, como el reciente *Prey*, la saga *Crysis* o *Kingdom Come: Deliverance*. Lo que hace que el CryEngine sea digno de mención son sus capacidades gráficas que podríamos considerar como equivalentes a las de Unreal Engine. Un ejemplo de su capacidad gráfica sería su sistema de renderizado volumétrico de niebla y nubes, que consigue un aspecto muy realista de estos fenómenos. También resulta una opción muy atractiva de cara a proyectos con realidad virtual.

El principal inconveniente que tiene este motor es su curva de aprendizaje, la cual resulta más elevada que la de sus competidores. Puede resultar demasiado complejo para principiantes que no haya utilizado ningún otro motor de antemano. No obstante, su sistema de pagos sería mucho más asequible que el de su competidor más directo Unreal Engine, ya que CryEngine no requiere que los usuarios paguen un porcentaje de las ganancias de los juegos que lo utilicen, si no que únicamente es necesario pagar mensualmente una cuota de 9,90$ para acceder a la plataforma de CryEngine. Además, CryEngine cuenta con un foro de preguntas y respuestas llamado CryEngine Answers, que ayuda a aclara dudas y consultas, mejorando la experiencia (Crytek (2021)).

2.2.4 GameMaker Studio

Game Maker Studio puede ser considerado como el motor más sencillo de utilizar de los aquí mencionados. Al margen de un lenguaje de programación sencillo, cuenta con un sistema *drag and drop* que apenas requiere de conocimientos de programación para ser utilizado, lo cual lo hace muy atractivo para otro tipo de perfiles menos técnicos o para prototipado de juegos. En contrapartida, esta sencillez también limita las herramientas de las que si gozan otros lenguajes. Cuenta soporte para juegos 2D y 3D, pero su sistema 3D es muy limitado.

Hay una buena cantidad de títulos *indie* conocidos que han utilizado este motor como *Hyper Light Drifter*, *Downwell* o *Nuclear Throne*, lo cual de muestra que es una opción atractiva para este tipo de producciones de bajo presupuesto. Su sistema de pagos es bastante económico también y parte desde 39$ anuales para poder publicar en Windows o Mac. También hay licencias permanentes para exportar a distintas plataformas que oscilan entre 99$ y 199$ en funcion de la plataforma. En el caso de querer exportar a consolas, la licencia por plataforma cuesta 799$ anuales con la opción de pagar 1500$ para tener la posibilidad de exportar a todas las plataformas disponibles.

2.2.5 Godot

Godot es un motor de videojuegos similar a Unity pero totalmente gratuito. Como lenguaje de programación utiliza Python y ofrece una curva de aprendizaje muy baja respecto a otros motores, lo cual lo hace ideal para comenzar a entrar contacto con el desarrollo de juegos contando con nociones básicas de programación. Se trata de un motor de código libre, lo cual implica que se puedan modificar al completo todos sus componentes. Ofrece soporte a juegos 2D y 3D bastante solventes, aunque en el apartado 3D está más limitado que los de la mayoría de competidores, pero sigue siendo superior que, por ejemplo, el de GameMaker (GDQuest (2020)).

Se trata de un motor con una comunidad muy activa que puede ayudar a resolver dudas sobre su uso, pero dado que también se trata de un motor menos utilizado, su comunidad también es menor, lo cual lo deja en clara desventaja respecto a su competidor más directo Unity.

3 Definición del juego

A lo largo de este apartado se va a exponer un reflejo de la fase de diseño del juego. Se comentarán detalles acerca de conceptos, mecánicas y dinámicas utilizadas en el juego, así como detalles acerca de la ambientación, trama y estética del juego, entre otros.

3.1 Idea del juego

El diseño del juego partió del deseo de realizar un juego que reflejara la naturaleza de los cangrejos ermitaños. El estilo de vida y las características de estas criaturas son bastante curiosas si las comparamos con otros animales. Los cangrejos ermitaños a pesar de su nombre, son más similares a las langostas, pero con una importante diferencia: su parte trasera carece de caparazón que lo proteja, quedando completamente expuestos a depredadores. Este es el motivo por el que buscan conchas vacías que han generado otros animales (ellos son incapaces de generarlas) para meterse dentro y conseguir así estar protegidos.

A lo largo de su vida, los cangrejos van creciendo en tamaño, por lo que necesitan cambiar su concha actual por otra nueva que sea más grande. Es habitual que en algunas zonas haya escasez de conchas más grandes debido a la gran afluencia de turistas que las cogen como recuerdo. Esto ocasiona que los ermitaños que no hayan encontrado una concha opten por soluciones desesperadas y traten de cambiar su concha por cualquier objeto que encuentren, habitualmente latas, botellas y plásticos encontrados en la basura a pesar de que estos suelen resultarles bastante incómodos al no poder sujetarse bien.

A pesar del nombre, los cangrejos ermitaños suelen ser bastante sociables y viven junto a otros cangrejos ermitaños. Cuando un ermitaño cambia su concha y suelta la anterior, pueden haber luchas entre cangrejos más pequeños aspirantes a entrar en la concha liberada.

Todos estas curiosas cualidades hacen que sean unos animales interesantes de reflejar en un juego. Se podía crear una aventura en la que controláramos a un cangrejo ermitaño cuya meta fuese hacerse el más grande de la playa. Debería ir recogiendo comida situada por el mapa y creciendo poco a poco, teniendo que ir cambiando de concha continuamente. Para conseguir las conchas debería enfrentarse a otros cangrejos que lo atacarían y le podrían quitar su concha, de manera que en el momento en el que se quedara sin caracola, el cangrejo solo podría recibir un golpe antes de morir, dando pie a lo que es actualmente Hermit King.

3.1.1 Desarrollo de la idea

Finalmente se decidió abordar la idea con un juego 3D en tercera persona en el que controlaríamos a un ermitaño. Situados en todo el mapa habría bolitas de fango que el jugador debería ir recogiendo para alimentarse. El cangrejo crecería hasta el punto de que su concha actual no le sirviera, momento en el que debería buscar otra más grande. El objetivo final del juego sería crecer hasta conseguir habitar la enorme caracola dorada situada en el centro de la playa, lo cual le convertiría en el rey de los ermitaños.

En el mapa también estarían situados otros ermitaños que estarían buscando también conchas de mayor tamaño a la suya y que no dudarían en dirigirse al jugador para atacarle si se diera el caso. La jugabilidad, por tanto, estaría enfocado al combate y la movilidad del cangrejo principal para defenderse y atacar a los demás. Podría atacarles con sus pinzas para intentar expulsarlos de su concha tras asestar varios golpes con sus pinzas, pero también defenderse de ataques enemigos si actúa en el momento adecuado. En el caso de que un cangrejo contara con concha, soportaría varios golpes antes de soltarla, pero si recibiera un solo golpe sin estar en la concha, moriría automáticamente debido a que estaba desprotegido.

El juego contaría con distintos tipos de conchas con características diferenciadoras. Unas conchas podrían hacer que el cangrejo aguantara más golpes a costa de ralentizar su movimiento y otras que permitieran movernos más rápido, pero serán más débiles. En el caso de que el cangrejo no tuviera ninguna concha equipada sería la manera más rápida de moverse, pero a la vez, solo podría recibir un golpe. Esto combinaría muy bien con la posibilidad de dejar la caracola a voluntad, de manera que el jugador pudiera elegir entre movilidad y resistencia en función de la situación y estilo de juego. Para intensificar esto, el jugador al salir de la concha tendría la opción de salir despedido de la concha a la vez que la deja, lo cual le podría librar de situaciones en la que varios cangrejos le persiguen o podría utilizarlo para alcanzar un punto mas rápidamente.

También se decidió que se podría implementar otra manera de liberarse de la concha, esta vez ofensiva. El cangrejo podría lanzar su concha hacia los enemigos, de manera que se destruyera en el proceso, pero infligiendo una gran cantidad de daño en área a los cangrejos cercanos al explotar en pedazos. De esta manera, se favorecería de nuevo esa dinámica de riesgo-recompensa en función del estilo de juego que quisiera llevar el jugador.

Cada concha tendría un rango de tamaño del cangrejo que la puede habitar, siendo imposible meterse dentro de una concha si no se está dentro de este rango. En el caso de que el ermitaño del jugador creciera hasta superar el tamaño máximo admitido por su actual concha, esta comenzaría a sentirse incómoda, hasta llegar al punto de que se soltaría automáticamente del jugador teniendo que buscar otra más grande. Se crearía así una dinámica en la que se está continuamente buscando nuevas conchas, teniendo que adaptarse a la situación.

En cuanto a la progresión de la dificultad, a lo largo que creciera el jugador se iría incrementando la amenaza de los enemigos. A media que se avanzase en el juego aparecerían

más cangrejos, los cuales serían cada vez más grandes. Esto también implicaría que cada vez aparecería conchas más grandes, pero que deberíamos arrebatárselas a nuestros adversarios, incrementando la dificultad. Además los cangrejos enemigos, cuanto más grandes fueran respecto al cangrejo del jugador, mayor daño le harán en caso de golpearle. En caso de ser derrotado, se perderá todo el progreso, teniendo que comenzar una partida de nuevo. Esto eliminaría la progresión del juego entre partidas, para centrarse en la mejora de la habilidad del jugador.

Dado que se trata de un rogue-like, para hacer que cada partida fuera distinta se utilizaría un sistema de generación progresivo aleatorio de los elementos del mapeado, que haría aparecer en distintas ubicaciones a los cangrejos, conchas y comida alterando aleatoriamente también sus propiedades entre partidas.

3.1.2 Plataforma y control

El juego pretende ser todo lo accesible posible, por lo que se ha enfocado a ser controlado con mando, que sería el control más común, pero también se ofrece la posibilidad de jugarlo con teclado y ratón en el caso de que el jugador no disponga de un mando. Por el mismo motivo, el juego está diseñado como aplicación de escritorio, para que pueda ser ejecutado en prácticamente cualquier ordenador. Se ha utilizado la URP[1] que permite evitar consumir demasiados recursos a cambio de disminuir la calidad gráfica del juego.

Al margen de las limitaciones que supondría el desarrollo para otras plataformas como consolas, en las que es necesario un kit de desarrollo, hoy en día tener un ordenador esta ampliamente extendido, por lo que es altamente probable que un juego de estas características pueda ser ejecutado por una gran cantidad de público. De hecho, con un proceso de adaptación sencillo, sería posible portarlo a móviles.

Siguiendo en la línea de la accesiblidad, se ha procurado utilizar controles bastante intuitivos de manejar. Se han aprovechado un mapeo de controles tanto del mando como de teclado y ratón que habitualmente se pueden encontrar en otros juegos. De esta manera, en el caso de que el jugador tenga cierta familiaridad con cualquiera de estos títulos, podrá controlar fácilmente los de este juego, sin apenas tener que informarse de cómo jugar.

Un ejemplo de esto sería el uso del joystick izquierdo para desplazar al jugador, y del joystick derecho para controlar la cámara, que se ha convertido en el estándar de los juegos en tercera persona. También se ha procurado simplificar el uso de botones combinando acciones entre sí, como por ejemplo en el caso del salto desde la concha. Al pulsar y soltar rápidamente el botón servirá para únicamente salir de la concha y quedarse en el sitio, pero manteniendo y soltándolo después podremos saltar además de dejar la concha, lo cual minimiza las

[1] La Universal Render Pipeline se trata de una metodología de procesamiento del renderizado que incorpora Unity enfocada a ser utilizada en sistemas con pocos recursos como móviles y tabletas u ordenadores y consolas poco potentes

asignaciones de botones que debe aprender el usuario de una mantera intuitiva.

3.2 Conceptualización

3.2.1 Historia, ambientación y trama

El juego se sitúa en una playa de estilo caribeño en la que viven los cangrejos ermitaños. Desde que nacen, los cangrejos pueden ver la gigantesca concha que se sitúa en medio de la arena, la cual lleva ahí desde hace generaciones. Ningún cangrejo ha conseguido crecer lo suficiente como para poder hacer de esa caracola su casa. Al igual que muchos otros, nuestro cangrejo protagonista intentará por todos los medios crecer hasta conseguirlo. Tendrá que buscar caracolas deshabitadas o arrebatar a otros sus conchas conforme crezca evitando que los cangrejos enemigos rompan su casita.

3.2.2 Definición de personajes y elementos

Como personajes, el juego solo cuenta con nuestro cangrejo protagonista y el elenco de cangrejos enemigos. Estos podrán contar con caracolas o tener la mala suerte de no disponer de ninguna. Todos los cangrejos siempre andarán buscando una nueva caracola mejor que la actual, teniendo que enfrentarse entre sí a menudo.

Todas las conchas no son iguales, si no que cada una contará con unas características distintivas en función de su forma de manera que se puedan reconocer visualmente. Existen caracolas en espiral que son ligeras y débiles, conchas de caracol equilibradas y latas robustas pero pesadas. Algunas de estas conchas estarán vacías y algún cangrejo afortunado podrá habitarla sin la necesidad de pelear para conseguir una.

Además de cangrejos, conchas y rocas, en la playa también podremos encontrar bolitas de fango que servirán de alimento para nuestro cangrejo principal. Conseguir esas bolitas son la clave para crecer y obtener la victoria haciéndose con la gran concha dorada situada en el centro del mapa, convirtiéndose así en el Rey Ermitaño.

3.2.3 Interacción entre actores

Todos los cangrejos codician una concha mejor que la suya, por lo que en el momento en el que detecten que nuestro cangrejo protagonista tiene una concha mejor que les pueda servir comenzarán a perseguirlo. Cuando se acerquen lo suficiente al protagonista comenzarán a atacarle con sus pinzas, ante lo cual el jugador deberá reaccionar esquivándolo mediante el movimiento o defendiéndose en el momento correcto.

Los cangrejos más grandes que el jugador tienden a ignorarlo, pero cualquier ermitaño puede llegar a ser hostil en el caso de que nos acerquemos lo suficiente como para suponer una amenaza para ellos o que directamente sea atacado por el jugador. En cualquier otro caso, los cangrejos se limitarán a vagar tranquilamente por la playa.

3.2.4 Objetivos del juego

El objetivo del juego es hacer todo lo grande posible a nuestro ermitaño para llegar más lejos. Como meta última tendría que crecer lo suficiente hasta habitar la concha central, completando el juego. Para conseguir esto se debe conseguir comida, vencer a otros cangrejos e ir cambiando la concha en función de la disponibilidad, preferencias y tamaño del cangrejo.

Mediante la progresión del juego se busca que el jugador en sus primeras partidas no pueda conseguir la concha central, y a lo largo de unas cuantas partidas mejore su habilidad lo suficiente como para finalizar el juego, siguiendo la estructura de un rogue-like típico.

3.2.5 Apartado artístico

Como ya se ha comentado previamente, Hermit King está bastante influenciado por el comportamiento del cangrejo ermitaño real, por lo que se han tomado ciertos detalles acerca de su estilo de vida real y anatomía para utilizarlos en las distintas partes del juego. No obstante, evidentemente se han tomado ciertas licencias para adaptarlo a un videojuego con un estilo colorido y desenfadado que fuera entretenido.

Por este motivo el diseño de los cangrejos se ha originado con unas nociones realistas de la anatomía del cangrejo en mente que se han estilizado posteriormente. Por ejemplo, los ermitaños reales son decápodos y cuentan con un par tenazas, dos pares de patas para desplazarse y otros dos pares traseros que utilizan para agarrarse a la concha que ocupan. Para simplificarlo se han eliminado estas patas traseras del mismo modo que en dibujos animados como *Los Simpson* las manos de los personajes tienen 4 dedos.

Para el diseño de los cangrejos también se revisaron diseños ya existentes de ermitaños estilizados a modo de inspiración. Esto ha servido de apoyo para encontrar un estilo artístico adecuado a las intenciones del juego. En el caso de las conchas se ha tomado inspiración de conchas que realmente llevan estos cangrejos para después simplificarlas.

El terreno del juego se ha realizado con cierta inspiración de *Journey*, un juego famoso entre muchas otras cosas por su gran cuidado de los terrenos arenosos. Dado que la arena es la mayor parte del terreno se ha procurado cuidar más su aspecto dotándola de suaves dunas y puntitos que reflejan la luz en función de la posición del jugador.

Figura 3.1: Imagen de un cangrejo ermitaño joven

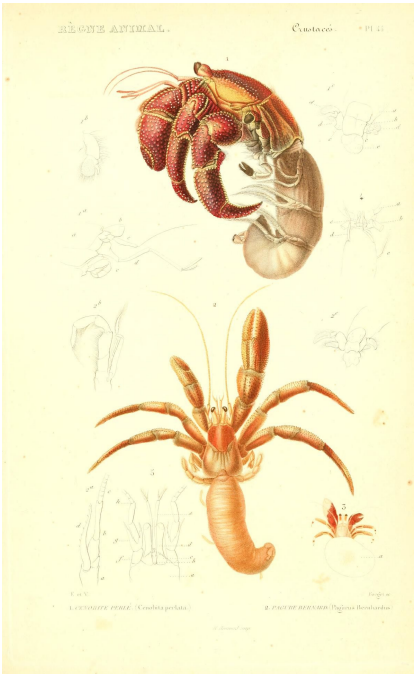

Figura 3.2: Anatomía del cangrejo ermitaño

Figura 3.3: Bocetos realizados durante el proceso de diseño del cangrejo

Esto se ha acompañado de una serie de rocas estilizadas que van a caballo entre lo realista y el *low poly*[2]. De la misma manera, el agua de la playa tiene un contorno de la espuma bastante marcado y estilizado, contando con una textura muy plana y ondulaciones muy leves.

En cuanto a la animación de los personajes, se ha utilizado un sistema de animación procedural para desplazar y rotar las patas de los cangrejos, de manera que el cangrejo

[2]estilo de modelado caracterizado por contar con un bajo número de polígonos, que suelen estar muy remarcados, creando superficies trianguladas

Figura 3.4: Imágenes y diseños tomados como referencia para encontrar un estilo artístico

apoyase sus patas automáticamente sobre la superficie cumpliendo ciertas condiciones que permitan emular el movimiento de un ermitaño real.

Evidentemente, no se pretende hacer una simulación realista de su movimiento, si no hacer que el ermitaño sea identificable como tal en el juego. De hecho, en este apartado también podemos ver un proceso de estilización y caricaturización del movimiento. Un ejemplo de esto es la animación de defensa, donde el cangrejo da la sensación de estar asustado y se protege con sus pinzas de una manera similar a como lo haría una persona.

Para el color e iluminación se ha seguido en la misma línea que los anteriores apartados y se ha presentado unas texturas que dan a entender perfectamente a lo que referencian de la realidad, pero también con un aspecto estilizado y simplificado que ayuda a ofrecer ese aspecto desenfadado del juego, de la misma manera que sucede en juegos como *Crash Bandicoot* o *Yooka Laylee*. Se han utilizado texturas bastante planas con poco detalle y colores alegres para todas las superficies del juego, de manera que se cree una estética cercana al *cel shading*.

Figura 3.5: Captura del aclamado juego *Journey*

Con ayuda de un posterior post-procesado de la imagen, se ha conseguido adaptar mejor los colores para que estén dentro de la misma paleta de tonos anaranjados, marrones, rojos y amarillos. Como contraste se tienen los colores azules del cielo y el mar que si bien resaltan dentro de esta paleta de colores cálidos también han sido modificados para ser mas amarillentos y no contrastar tanto como a priori. Este contraste si que se ha reforzado para elementos como las bolitas de comida para llamar la atención del usuario y ser fácilmente localizables.

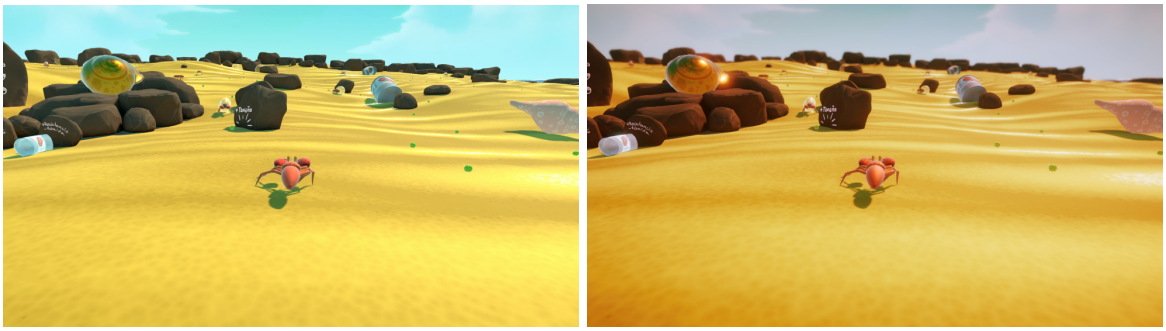

Figura 3.6: Corrección de color y sensación de profundidad conseguida con el post-procesado de la imagen (izquierda: sin post-procesado; derecha: con post-procesado)

En cuanto a la interfaz, se ha buscado que fuera lo más minimalista posible, de manera que no entorpezca al usuario y sea más agradable de ver. Se ha sustituido un medidor de resistencia por una indicación diegética integrada en el juego mediante un sistema de agrietado de la concha y se ha hecho que se pueda saber fácilmente si una concha es o no habitable en función si está o no iluminada. De la misma manera, los cangrejos que son mas grandes y por tanto hacen más daño al jugador se han tintado de un rojo intenso para dar a entender que son más peligrosos. Se evita de esta manera cualquier tipo de indicación textual que ensucie la interfaz del juego pero se sigue aportando la misma información de una manera intuitiva para el jugador.

Tanto esta interfaz minimalista como la composición del campo de visión con el jugador en el centro y cierto espacio alrededor que permite visualizar el entorno han sido inspirados en

otros juegos en tercera persona con secciones de ”plataformeo” como *Yooka Laylee* o *Journey*.

Figura 3.7: Interfaz de *Yooka Layle*

4 Diseño técnico

4.1 Motor utilizado

Para la elaboración de este proyecto se ha utilizado Unity como motor. Como ya pudimos ver en el apartado de estado del arte, Unity ofrece una gran cantidad de ventajas. Es fácil de utilizar contando con algunas nociones de programación, cuenta con una gran comunidad de la que poder aprender como solucionar muchos problemas y obtener herramientas que agilicen el desarrollo, y además es gratuito para un proyecto de estas características.

En cuanto al apartado gráfico, serían más recomendables otros motores como Unreal Engine o Cryengine, los cuales son superiores en este apartado. Sin embargo, dado el estilo visual que queremos conseguir esto no parece necesario, y en el caso de querer hacerlo se podría utilizar la HDRP de Unity.

Aunque quizás el factor con mayor peso a la hora de escoger Unity es el conocimiento que se tiene del mismo respecto a las otras opciones. Prácticamente todo el máster universitario cursado se articula en torno a Unity como herramienta, lo cual sumado a la experiencia previa con el motor lo hace la opción más recomendable.

4.1.1 Pipeline

No se ha utilizado la versión básica que viene por defecto al instalar Unity, si no que se han utilizado distintas herramientas que vienen incluidas por defecto. Una de las primeras decisiones que debemos tomar antes de comenzar a desarrollar un juego con Unity es la *render pipeline* que vamos a utilizar. Contamos con varias configuraciones de pipeline a nuestra elección:

- **Pipeline por defecto:** Es la pipeline que viene incluida por defecto en un proyecto de Unity. Antiguamente era la única tubería que ofrecía Unity oficialmente y actualmente Unity está tratando de sustituirla directamente por la URP, dejándola prácticamente en desuso. Cuenta con muy poca capacidad de adaptación y muchas menos opciones de personalización que las siguientes y tampoco cuenta con soporte para herramientas tan útiles como *Shader Graph*, la cual permite crear *shaders* de manera visual mediante nodos.

- **Render Pipeline o URP:** Es la anteriormente conocida como *Lightweight Render Pipeline* y como su nombre indica está enfocada a ser utilizada en cualquier tipo de dispositivo gracias a su alta eficiencia. Se trata de una tubería de bajo rendimiento, por lo que no puede conseguir la misma calidad gráfica que la que tienen otros motores o la propia HDRP. No obstante, ofrece una gran cantidad de opciones para personalizarla que consiguen efectos visuales bastante sorprendentes. Tiene una alta compatibilidad con sistemas de menor rendimiento como móviles, tabletas o la consola Nintendo Switch, para los cuales resulta prácticamente imprescindible.

- **High Definition Render Pipeline:** Es la tubería más avanzada que dispone Unity y está enfocada a conseguir efectos muy realistas y ambiciosos a cambio de perder compatibilidad con dispositivos menos potentes. Para ello aprovecha convenientemente la capacidad de la GPU para conseguir efectos bastante similares si no iguales que los que se pueden conseguir con motores como Unreal o Cryengine.

Tanto la HDRP como la URP son configuraciones predefinidas de la SRP o Scriptable Render Pipeline que es la verdadera tubería de renderizado que utiliza Unity. El nombre proviene de que es completamente configurable mediante scripts escritos en lenguaje C# por lo que también es posible crear una tubería completamente personalizada desde cero, aunque evidentemente sea mucho más laborioso. La SRP es un añadido relativamente reciente que se añadió para suplir la baja capacidad de modificación del sistema de renderizado que se le achacaba al sistema y de la que si gozaba la competencia.

Dadas las características de la estética de Hermit King parece evidente que no aboga por ser realista, de manera que no sería necesaria la HDRP. La URP cuenta con opciones más que suficiente para permitir reflejar el juego tal y como estaba en mente, con el añadido de que exista la posibilidad de ser compatible con más plataformas como móviles.

4.1.2 Herramientas extra

Unity cuenta con una gran cantidad de herramientas útiles incluidas por defecto, sin embargo es posible añadirle otras muchas. En este caso se ha decidido incluir varios paquetes oficiales de Unity para facilitar la labor en ciertos apartados.

- **Cinemachine:** Se trata de un añadido de Unity más popular para manejar el control de las cámaras de la escena. Cuenta con numerosas opciones para realizar *travelings*, seguir a un objeto, encuadrar un grupo de objetos, amortiguación del movimiento, etc.

- **Text Mesh Pro:** Los textos que utiliza Unity por defecto en su interfaz tienen un problema bastante notorio que es el pixelado de la fuente en ciertas situaciones. Text Mesh Pro soluciona este problema asegurando un renderizado de texto muy fiel además de otras opciones útiles como la creación de sombreados y bordes mas complejos entre otros.

- **Shader Graph:** Herramienta exclusiva de la SRP que permite crear shaders de una manera mucho más sencilla e intuitiva evitando la programación. Cuenta con un sistema de nodos de distintos tipos con propiedades que se comunican entre sí de manera que se vaya definiendo el proceso que sigue el shader. Cada nodo cuenta con una representación gráfica de su salida, agilizando en gran medida su configuración.

- **Animation Rigging:** *Asset* todavía en estado experimental creado por Unity para realizar alteraciones en el *rig* y animaciones de los modelos importados en Unity. Permite aplicar ciertos tipos de restricciones al movimiento de los huesos de manera que se puedan manejar dentro de Unity, pudiendo sobrescribir el movimiento de huesos en animaciones o crear animaciones nuevas directamente. Incluye restricciones útiles en este proyecto como las de *Inverse Kinematics* o IK, las cuales permiten articular automáticamente varios huesos para que un punto del esqueleto esté en una posición concreta, especificando el rango de movilidad del mismo.

- **New Input System:** Se trata de un nuevo sistema creado por Unity para controlar las entradas de recibidas por los distintos dispositivos conectados. La intención es que este sistema sustituya al incluido por defecto dentro de poco, ya que incluye una manera mucho más sencilla, rápida e intuitiva de especificar las relaciones entre eventos de dispositivos y acciones del juego. Hace que la adaptación del control del juego a distintos dispositivos sea mucho más sencilla y además su uso asegura que no se quede obsoleto en poco tiempo.

4.2 Software auxiliar

Al margen del motor de videojuegos son necesarias otras aplicaciones que nos ayuden a realizar distintas tareas. En el caso de este proyecto se han utilizado las siguientes:

- **Maya:** Software de animación, modelado, simulación y renderizado en 3D muy extendido en la industria de la animación y los videojuegos. Aunque existen otras opciones muy válidas en el mercado como 3D Studio Max, ZBrush o Blender, se ha elegido Maya debido a que se cuenta con una licencia gratuita para estudiantes para poder utilizarlo, y es el programa utilizado durante la asignatura de Media para videojuegos. Se ha utilizado para todo el proceso de modelado, despliegue de UVs, rigging y animación del cuerpo del cangrejo. También se ha utilizado para adaptar modelados gratuitos obtenidos de internet a las necesidades del proyecto, minimizando el número de polígonos utilizados, haciendo retoques en un forma y realizando el desplegado de UVs, de las mallas resultantes.

- **Audacity:** Programa de edición de audio más conocido y utilizado de código abierto. Se ha utilizado para cortar y alterar sonidos obtenidos de librerías de manera que se adaptaran a la estética y ambientación del juego.

- **Procreate:** Popular aplicación para iPad con la que se han pintado las texturas de cangrejo y caracolas. Tras obtener el contorno de los polígonos desplegados en la hoja de UVs con Maya se ha introducido en esta aplicación a modo de guías. Después se han utilizado los distintos pinceles que ofrece para pintar las texturas utilizando un lápiz táctil.

- **Photoshop:** Sistema de edición de imágenes por excelencia. Se ha utilizado para realizar retoques en algunos *sprites* y texturas utilizados en el proyecto como por ejemplo realizar cambios de tamaño, eliminar fondos de imágenes, modificar propiedades de color, contraste e iluminación o aplicar filtros estéticos.

- **Overleaf:** Aplicación web utilizada para redactar textos en formato LaTeXmuy extendida en el mundo académico.

- **Libre Office:** Editor de texto libre y de código abierto utilizado como programa auxiliar durante la redacción de la documentación.

- **DaVinci Resolve:** Software de edición de vídeo utilizado para la composición de los vídeos adjuntos a este documento que se ha popularizado en los últimos años.

4.3 Assets

En este apartado se presentarán los assets utilizados para elaboración del juego, describiendo el proceso de creación y modificación y/o las fuentes de las que se han obtenido si han partido del producto de terceros. Se podrían considerar también como assets a partes de la lógica del juego, del código o del comportamiento de los shaders, pero por una cuestión organizativa estos elementos se verán en apartados posteriores centrados en los mismos. Dicho como assets completamente propios tendríamos el modelo del cangrejo, el cual se ha realizado completamente desde cero. También hay otros assets como las conchas para los cuales se ha partido de una base ya hecha y se han realizado numerosas modificaciones, y finalmente hay assets que se han incluido tal cual estaban en el proyecto y apenas se han modificado. A continuación, daremos una explicación más detallada acerca de todos ellos.

4.3.1 Cangrejo

Como ya se ha comentado en apartados anteriores, se han realizado todas las fases del diseño, modelado, texturizado, *rigging* y animación del cangrejo desde cero utilizando las herramientas de Maya y Procreate. Más tarde, ya en Unity, se hizo uso de las facilidades que provee la herramienta Animation Rigging para realizar la animación procedural de las patas del cangrejo adaptándose al terreno.

Como ya vimos en apartados anteriores, el diseño del cangrejo partió de realizar varios bocetos de la forma del cuerpo tomando inspiración de distintas imágenes y vídeos de referencia. Teniendo estos bocetos en mente se pasó a la fase de modelado y partiendo de distintas primitivas se fueron extruyendo, y modificando la geometría del cangrejo para adaptarla a los diseños. Una herramienta muy utilizada durante el proceso debido a la forma más redondeada del cangrejo fue la deformación suavizada, que permite aplicar transformaciones a puntos cercanos a nuestra selección con un efecto menor en función de su lejanía a la misma, permitiendo generar deformaciones más orgánicas que evitan mostrar los polígonos que forman la maya. En el proceso se tuvo muy en cuenta la complejidad de la geometría, ya que al tratarse de un modelo que se va instanciar muchas veces es importante es relevante no abusar del número de polígonos.

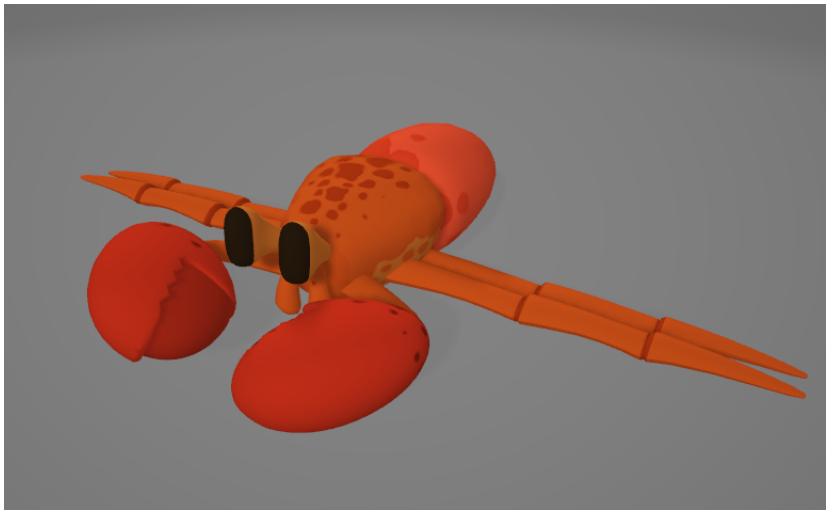

Figura 4.1: Resultado final del modelo del ermitaño

Con el modelo terminado se comenzó con el despliegue de UVs separando las caras de la geometría en grupos denominados islas. En este apartado se priorizó minimizar la tensión[1] de las caras que componían las islas y mantener una organización sencilla y visual de las mismas dentro de la hoja de UVs. Como el texturizado se realiza de una forma bastante básica y manual esto ayuda a mejorar los resultados y a agilizar el proceso de pintado.Una vez realizado el despliegue y organización de UVs se ha pasado el contorno de las caras como esquemas a la herramienta de dibujo Procreate, en la cual se ha pintado manualmente con un lápiz táctil, obteniendo las texturas.

Posteriormente comenzó la fase de rigging, en la cual se creó el esqueleto del cangrejo de manera que se pudieran articular las patas, pinzas, ojos, bigotes y trasero y se trabajó en la

[1]El concepto de tensión aplicado en el terreno de las UVs se refiere a la cantidad de deformación que se le va a aplicar a las caras que componen el modelo al pasarlas a una representación 2D como son las hojas de UVs que se utilizan para aplicar las texturas. Un modelo que tenga todas sus caras completamente separadas tendrá una tensión nula, mientras que una figura con mucho volumen como por ejemplo una esfera a la cual apenas se le hayan hecho cortes y sus caras formen parte de una única isla tendrá una gran cantidad de tensión

distribución de los pesos sobre la piel de manera que se comportaran de una manera lógica en función de lo rígida que fuera cada parte del cangrejo. Al tratarse de un cangrejo, muchas de las articulaciones son completamente rígidas, por lo que no se realiza ninguna deformación progresiva. Simplemente al desplazar un hueso del rig, se desplazan al unísono los vértices de la maya que lo rodean, al contrario que ocurriría en personajes humanos donde se formarían partes de degradado de manera que se simulara la elasticidad de la piel y los músculos.

Cuando se obtuvo el movimiento de las articulaciones deseado se realizaron dos animaciones mediante Forward Kinematics[2] para el ataque y defensa del cangrejo. Tras exportar el modelo desde Maya en formato FBX, se importó dentro del proyecto de Unity, donde se separaron las animaciones y se adaptaron en el animator del cangrejo. Más tarde se combinarían estas animaciones "tradicionales" con el sistema de animación procedural de las patas, que veremos más adelante.

4.3.2 Assets de terceros

Evidentemente, no ha sido posible realizar manualmente todos los assets del juego desde cero como se ha hecho con modelos del cangrejo, ya que sería demasiado costoso y el perfil de esta titulación no está enfocado a artistas. Por este motivo se han utilizado numerosos assets de acceso gratuito y libre.

- **Mar:** Se ha utilizado el asset *URP Stylized Water Shader* (BitGem (2021)) obtenido gratuitamente de la Asset Store. En él podemos encontrar unos componentes que nos permiten delimitar un volumen de agua mediante cubos, detectando el contorno de los objetos que toquen el volumen de agua generando un efecto de espuma. Cuenta con muchas opciones de personalización que permiten alterar el tamaño, velocidad y frecuencia del oleaje, textura, color y transparencia del agua, etc. También cuenta con una opción para hacer objetos que floten en la superficie como madera o boyas. Se ha configurado de tal manera que case con la estilización del resto de objetos de la escena y que de la sensación de ser un mar calmado al estar en la orilla de la playa.

- **Rocas:** Pack de modelos texturizados obtenido gratuitamente de un banco de modelos 3D (vgvladimir (2017)). se ha escogido por casar con la estética del juego al ser una mezcla entre realista y estilizado. Se han importado dentro del proyecto de Unity y se han preparado los modelos por separado en prefabs. También se han tenido que configurar las propiedades del material para poder aprovechar en Unity las texturas de albedo y normales que incluía. Se ha descartado la textura de oclusión ambiental que incluían los modelos para utilizar directamente un *bake* de la iluminación de la escena generado por el propio Unity.

[2]Se trata de un método de cambiar la orientación de los huesos de un esqueleto en el cual se debe indicar la rotación que se aplica a cada hueso para conseguir que los huesos se coloquen en la posición deseada. Es el proceso inverso de Inverse Kinematics, en el cual se indica la posición deseada del hueso y se calculan las rotaciones de los huesos involucrados para que se llegue a esa postura

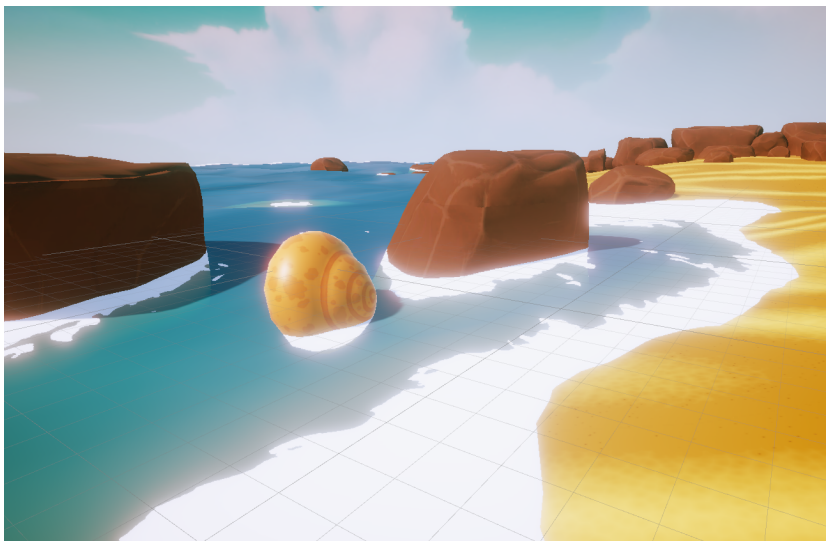

Figura 4.2: Asset de agua estilizada en funcionamiento. Genera contornos con espuma alrededor de cualquier malla que entre en contacto con el volumen de agua del mar

- **Conchas:** Como punto de partida de los modelos de las conchas se han utilizado modelos obtenidos gratuitamente de bancos de modelos 3D. Después se les tuvieron que aplicar múltiples procesos para obtener el resultado actual.

  - **Concha de caracol:** Se redujeron los millones de polígonos con los que se contaba inicialmente intentando mantener la forma mediante las herramientas *decimate* y *retopologize*[3]. Después de este proceso automático se revisó la geometría con más detalle, eliminando detalles innecesarios como los del interior de la caracola y arreglando la malla para poder hacer el desplegado de UVs del modelo. Cierta geometría residual que creaban geometrías convexas daban fallos al realizar este proceso automático.

  - **Caracola en espiral:** se realizó un proceso similar, aunque a una escala mucho menor dado que se trataba de una malla mucho más sencilla.

  - **Lata:** Se arreglaron ciertos problemas con geometría redundante y polígonos superpuestos que hacían más complejo su manejo y se deformó su forma creando la abolladura para darle un aspecto de desecho.

  - **Texturizado:** Finalmente se descartaron las texturas y UVs incluidas en los modelos y se cortaron nuevas UVs para los 3 modelos. Las guías se pasaron a Procreate y se pintaron manualmente las nuevas texturas con un aspecto más adecuado al

[3]Tanto *decimate* como *retopologize* son herramientas presentes en los programas de modelado 3D que permiten reducir el número de polígonos de una maya intentando mantener la forma original todo lo posible. Decimate intenta eliminar vértices y aristas que estén a menos de cierta distancia de otros, combinando la geometría en el proceso, mientras que retopologize intenta crear una topología o forma en la que se estructuran los vértices y aristas más alineada y correcta, eliminando también geometría en el proceso si es necesario

juego.

Figura 4.3: De izquierda a derecha: Modelos de conchas ligera, equilibrada y pesada

- **Shader de arena:** Se ha seguido paso a paso un tutorial (NedMakesGames (2021)) para elaborar un shader de generación procedural de una superficie de arenosa parametrizada. Se han utilizado los scripts provistos en el mismo para crear distintos generaciones de ruido necesarias en el proceso. Finalmente se añadieron algunas modificaciones en los nodos para hacer más presentes los brillos de la arena desde la perspectiva del cangrejo.

- **Sonidos y música:** Los sonidos se han grabado directamente u obtenido de bancos de sonido gratuitos. Después se les ha realizado algún proceso de edición para adaptarlos al juego, a excepción de las dos canciones que incorpora el título que están reproducidas tal y como fueron obtenidas de fuentes *royalty free* (Sapajou (2020); BisonTracksMusic (2017)).

- **Pack de botones de UI:** para los botones de la interfaz se ha aprovechado el asset *Simple Button Set 01* (BitGem (2019)) que cuenta con varios modelos de diseños de botones combinables con iconos que indiquen su función, lo cual resultaba muy práctico para los menús del juego.

- **Skybox:** Obtenido de la Asset Store del pack de skybox de cielo *Fantasy Skybox FREE* (RenderKnight (2020)).

- **Fuentes de texto:** Se han utilizado varios tipos de fuentes obtenidas del archivo de fuentes gratuitas Dafont.com (Dafont (2021)). Estas fuentes son *Scratch Boys*, *Candy Cake*, *Poppins* y *Leaves and ground*.

4.3.3 Shaders y efectos visuales

Hermit King cuenta con algunos efectos visuales concretos para mejorar el apartado estético del juego, así como su usabilidad. En esta categoría podríamos incluir a los shaders personalizados que utilizan los distintos materiales del juego, los efectos de partículas o los

efectos de post-procesado que se aplican sobre la imagen final modificando propiedades como las curvas de color.

4.3.4 Efectos de partículas

El juego cuenta con varios sistemas de partículas para mejorar la sensación de juego y añadir información acerca de los eventos sucedidos durante la partida. Todos se han creado con la herramienta de sistemas de partículas que incluye por defecto Unity, aunque también cabría la posibilidad de haberlos realizado con *Visual Effect Graph*[4]. A continuación comentaremos los sistemas incluidos en el juego así como algunos aspectos relevantes de su implementación. Estos son los efectos utilizados:

- Nube de polvo al caminar: Este efecto simplemente genera partículas con un sprite de humo aleatorio dentro de un array en el momento en el que se detecta movimiento del cangrejo mediante la opción de generación de partículas en función de la distancia. Los sprites emitidos son semitransparentes, por lo que se ha modificado las propiedades del shader para que haga una mezcla de color aditiva, de manera que se ilumine en mayor medida en los puntos en los que se superponen más sprites.

Figura 4.4: Partículas de polvo emitidas al desplazarse el cangrejo

- Gotas de agua cuando un cangrejo resulta herido: Simplemente se generan sprites de gota con rotación y velocidad aleatorias hacia arriba con un desvanecimiento progresivo de su opacidad. Este sistema escala a medida que el cangrejo crece, por lo que además de modificar su escala de generación progresivamente ha sido necesario aumentar también

[4]Herramienta de Unity que permite crear efectos de diversos tipos mediante nodos. VFX Graph puede conseguir sistemas de partículas más complejos y eficientes que los que permite los sistemas de partículas "convencionales" de Unity al aprovechar los recursos de la GPU en vez de los de la CPU

el multiplicador de gravedad para que realizaran unas parábolas similares independientemente del tamaño por como están implementados estos sistemas de partículas.

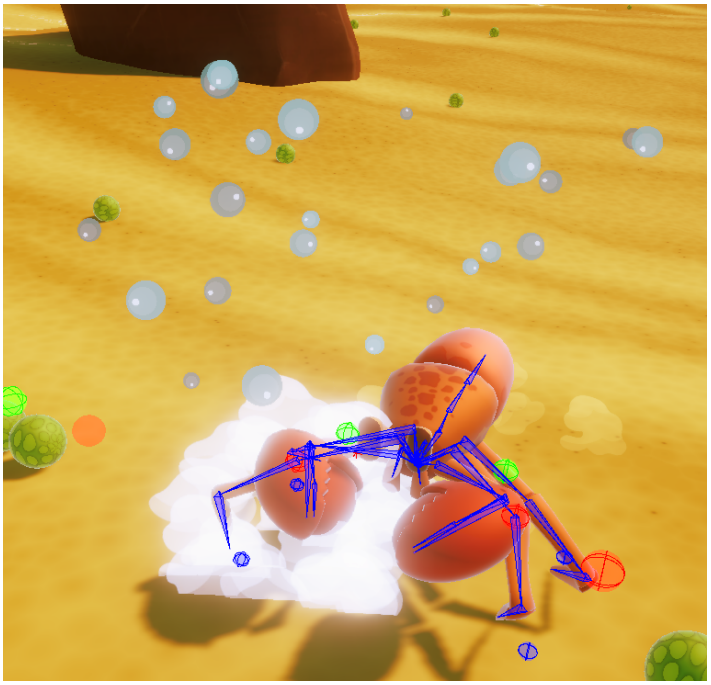

Figura 4.5: Muerte de un cangrejo enemigo con partículas de humo y gotas

- Chispas al defender efectivamente un golpe: Se trata de un sistema prácticamente igual al anterior, solo que en este se han utilizado las opciones de *trail* para realizar un trazo de la trayectoria realizada por cada partícula, y se ha aplicado emisión del color dentro de su shader.

- Nube de humo al morir un cangrejo: Se trata de un sistema prácticamente igual al de generación de polvo, pero con una emisión esférica de partículas más grandes y de color distinto.

- Explosión de la concha al lanzarla: Creado a partir de 3 sistemas:

  – Nube de humo: Sistema de humo utilizado previamente.

  – Explosión: Generación de sprites semitransparentes de explosión superpuestos aditivos con emisión de luz.

  – Fragmentos de concha: Para este apartado se han creado sprites de trozos de concha para cada una de las conchas. Para ello se han recortado varios trozos de una imagen de la concha y se han creado se han colocado en una sola imagen

Figura 4.6: Explosión de una concha pesada

a modo de *sprite sheet*[5]. Además se ha utilizado la opción de añadir colliders a las partículas de manera que interactúen con el escenario y desaparezcan antes al estar en contacto con algún otro collider. Las partículas además rotan en función de la velocidad, por lo que se van parando al tocar el suelo.

4.3.4.1 Shader de conchas

Para informar de la resistencia de las conchas se decidió realizar un shader para los modelos de las conchas que fuera variando en función de los golpes recibidos por otros cangrejos o el exceso de tamaño del cangrejo admitido por la concha.

Para ello, sen creó un shader con Shader Graph que permite aumentar el agrietado en mayor o menor medida con una variable que oscila entre 0 (concha intacta) y 1 (concha a punto de soltarse). Cuanto más grande es este valor más grietas aparecen, de mayor tamaño y más brillantes, dando a entender al jugador de una forma clara, visual y bien integrada en el gameplay del estado de su concha.

Se podría decir que el resultado final se compone de 3 capas de efectos que se superponen:

[5]Los sprites se pueden agrupar dentro de una misma imagen llamada *sprite sheet*. Normalmente esto se suele utilizar para agrupar por cuestiones de eficiencia sprites que corresponden a la misma animación o a una categoría. La distribución de los sprites dentro de la hoja suele ser cuadricula, aunque Unity dota de la opción de identificarlos dentro de la hoja independientemente de su posición y tamaño. En estas tareas resulta muy útil la herramienta *Sprite Editor* de Unity.

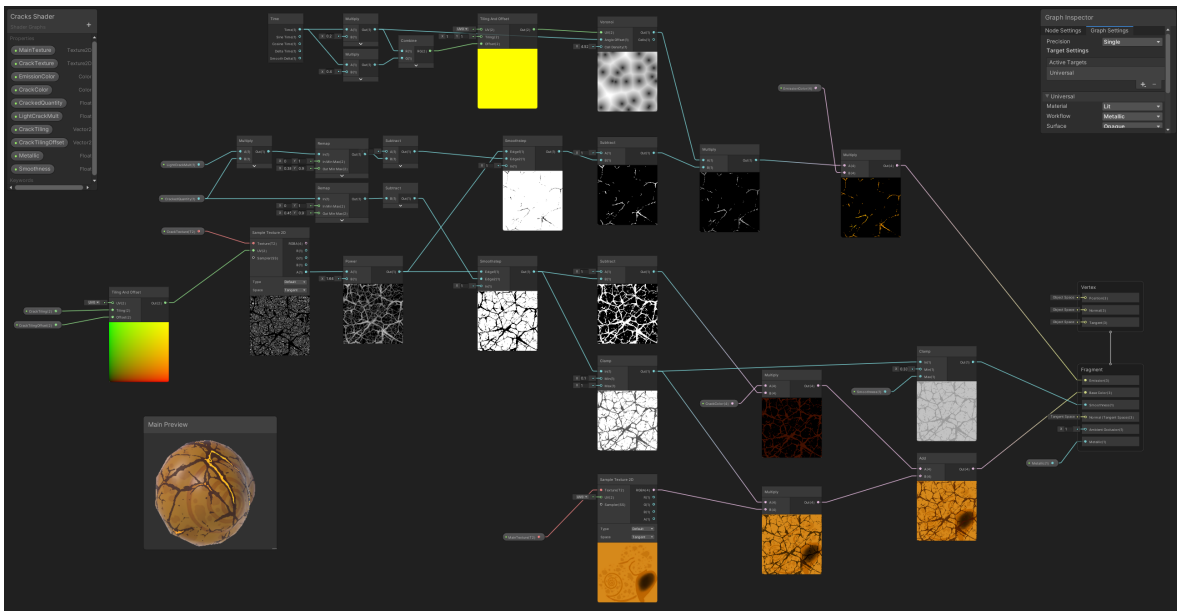

Figura 4.7: Nodos de Shader Graph que componen el efecto de agrietado progresivo del shader de las conchas

- Textura Principal: Se han habilitado parámetros para poder introducir la textura base de la concha opciones para poder modificar las cantidades de *smoothness* y metalizado de la misma forma que se hace en shaders convencionales que incluye Unity.

- Surcos de las grietas: Se ha utilizado una textura de agrietado que se procesa de tal manera que se creen las líneas estilizadas que se pueden apreciar en las imágenes. Lo más relevante de este apartado sería el uso del nodo *SmoothStep* que suaviza el contorno de las grietas de la textura de manera que modificando su variable *Edge 2* se consigue el efecto de mostrar más o menos agrietado. Simplemente se ha hecho un remapeado de la entrada de la variable que indica la cantidad de agrietado con un valor entre 0 y 1 para que se adapte a los valores que debe tomar Edge 2. El resultado de este nodo se superpone a la textura principal y se pinta de un color indicado por un parámetro del shader, de manera que se pueda modificar en función de la concha.

- Núcleo incandescente de las grietas: Se ha realizado un proceso muy similar al anterior. En este caso se utiliza un duplicado del efecto anterior que utiliza un valor siempre menor de su variable *Edge 2* que el utilizado en la textura de los surcos. El resultado se pinta con un color HDR combinado con un mapa de emisión para que esa textura emita su propia luz. Se ha añadido también un efecto palpitante de la textura que se va moviendo utilizando un patrón de ruido Voronoi que se va desplazando progresivamente en las UVs de la textura. Al multiplicar esa textura por la región del núcleo de las grietas obtenida anteriormente se consigue un efecto similar al magma fluyendo muy marcado que llama la atención del jugador.

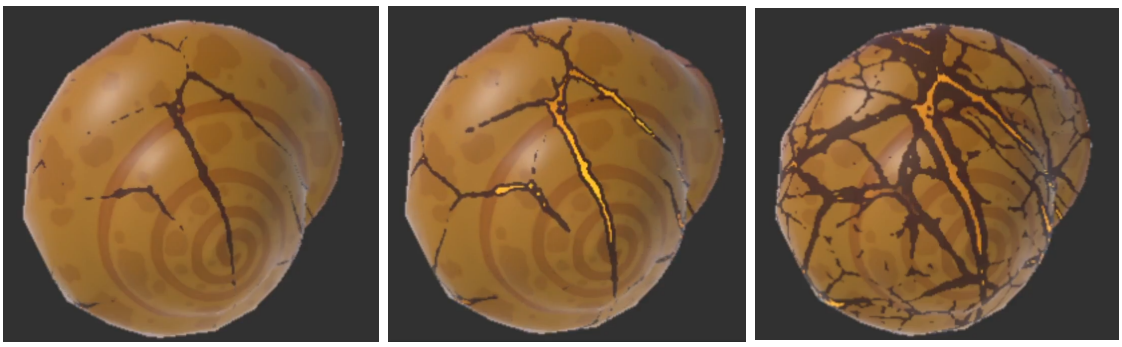

Figura 4.8: Efecto progresivo del shader de agrietado de las conchas

Además del efecto de agrietado, se decidió añadir también al shader información acerca de la disponibilidad de esa concha en función de su tamaño y el del cangrejo principal. Esto ayuda mucho al jugador para así poder detectar fácilmente qué conchas puede coger y cuáles no, de manera que pueda elaborar una estrategia.

En el caso de que la concha demasiado grande para el cangrejo el color no se altera, mientras que si la concha es habitable con el tamaño del cangrejo actual se iluminará, utilizando un incremento de la emisión del material y de la oclusión ambiental. En caso de que se haya superado el tamaño que admite la concha, el material se volverá oscuro disminuyendo la oclusión ambiental, y la iluminación y saturación de la textura base.

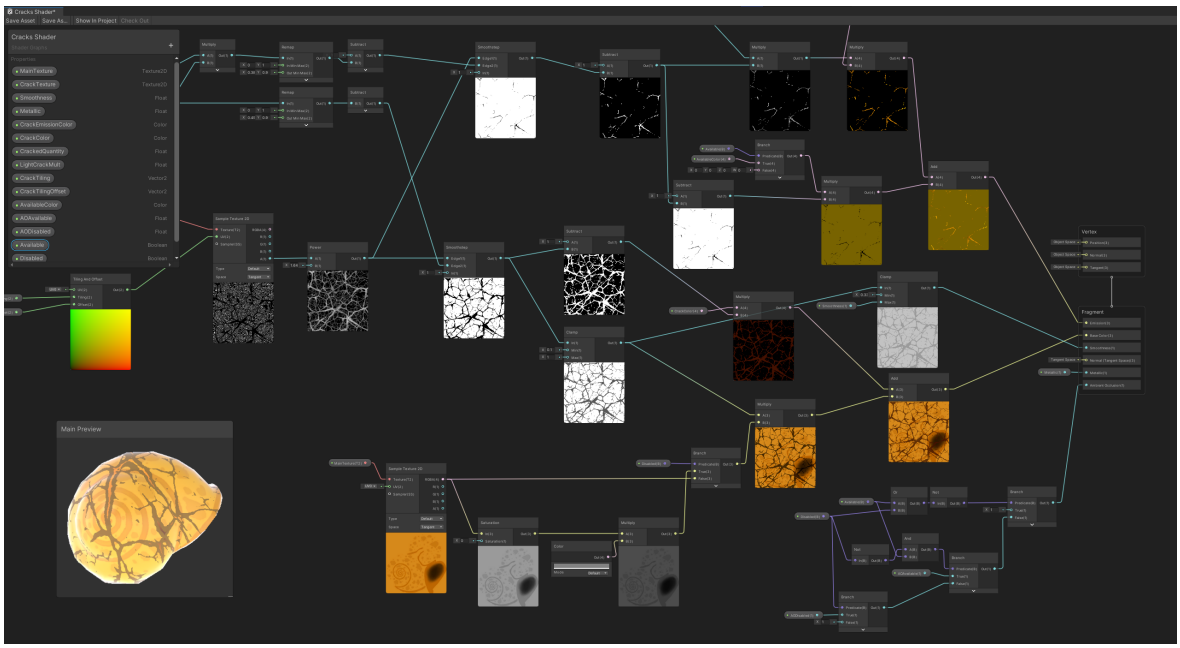

Figura 4.9: Ampliación de los nodos del shader de las conchas para añadir los efectos de estar habitable o deshabilitada completamente

Estos modos se ponen a través de dos variables booleanas que activan un modo u otro

según la lógica del shader. Para que cada concha cambie automáticamente de modo al variar el tamaño del cangrejo se ha utilizado un evento con una función *callback* situado en el controlador del jugador y que se llama en la función de modificación de tamaño. A este evento están suscritas todas las conchas, de manera que al variar el tamaño cada una comprueba si hay que cambiar su visualización. En el momento en el que se sobrepasa el tamaño admitido por la concha, se desuscribe del evento ya que no lo va a volver a utilizar, y de la misma manera, en el destructor del cangrejo se quitan todas las suscripciones para evitar problemas al reiniciar.

4.3.5 Lógica del juego y arquitectura

A pesar de que hay más clases auxiliares que tienen un papel en la lógica del juego, en la siguiente figura 4.10 se muestran las clases troncales del comportamiento de la escena principal del juego en sí.

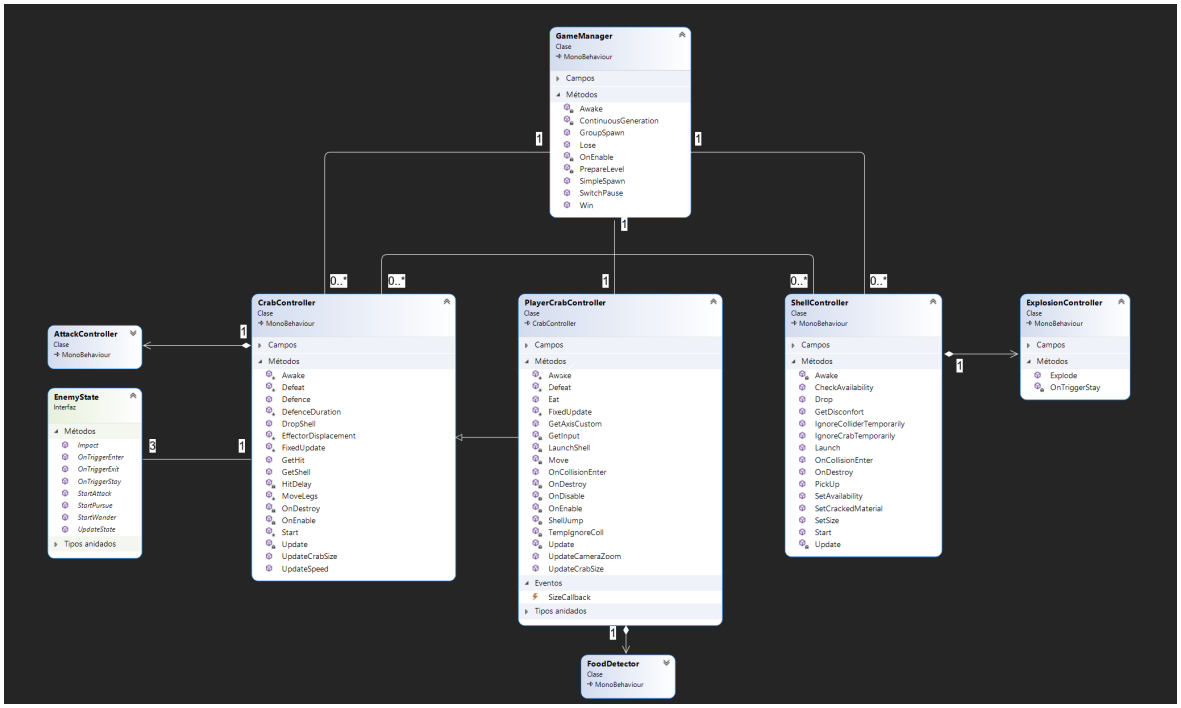

Figura 4.10: Diagrama de clases del núcleo del juego. Muestra los métodos de cada clase así como las relaciones establecidas entre sus instancias

A continuación, entraremos más en detalle en el comportamiento de algunas de estas clases, de forma que se pueda entender cómo está estructurada la lógica del juego.

4.3.5.1 GameManager

Se trata de una clase *singleton*[6] que está encargada de preparar cada partida y de regular su progresión a lo largo de esta.

Para la preparación de la partida se utiliza un sistema de generación de puntos que utiliza el algoritmo de *Poisson Disc Sampling* . Este algoritmo garantiza que los puntos generados en una superficie plana se mantengan a una distancia mínima definida que sería el radio del disco que se menciona en su nombre. En otros algoritmos de generación de puntos como podría ser uno completamente aleatorio veremos que en muchas situaciones los puntos acaban solapándose. En nuestro juego utilizaremos estos puntos para instanciar a parte de comida conchas y cangrejos de distintos tamaños, lo cual puede suponer un impedimento si generamos varios de estos objetos cerca. De la misma manera, con este sistema obtenemos una generación más distribuida más natural que no genera patrones como podría ser una cuadrícula.

El algoritmo de Poisson devuelve una serie de puntos situados en el plano formado por los ejes X y Z y están en los límites que engloba un collider cúbico, de manera que una vez obtenida la serie de puntos se proyecta un rayo paralelo al eje Y sobre el terreno. Para cada rayo se detecta si se ha colisionado con un objeto marcado como suelo con un tag. En el caso de que así sea se instanciará en el punto de colisión con el suelo el objeto que se esté generando en ese momento, y si no se descartará el punto. Se puede controlar de este modo mediante la colocación de colliders en el terreno que se generen objetos al lado de zonas claves como el cangrejo principal y la concha dorada. La serie de puntos es común para todos los tipos de objetos generables, por lo que existen unas variables que indican la proporciones de los puntos que corresponden a cangrejos enemigos, conchas y comida.

Una vez comenzada la partida, es necesario seguir creando cangrejos enemigos que vayan ofreciendo un mayor reto al jugador, por lo que hay un sistema de generación progresivo que crea nuevas instancias de enemigos conforme el jugador crece. Cada cierto incremento de la variable de tamaño del jugador principal se llama a la función de generación de un nuevo cangrejo en una posición aleatoria. Los cangrejos generados siempre serán un poco más grandes que el jugador y podrán contar con un tipo aleatorio de concha ya equipado o no. Para evitar que haya demasiado cangrejos en escena que podrían suponer un problema de rendimiento se ha limitado el número de cangrejos a un máximo, de manera que se vaya eliminando los cangrejos más antiguos mediante una cola.

Al margen de esto también cuenta con métodos para pausar y despausar la partida, así como mostrar las pantallas de derrota o victoria.

[6] El singleton es un patrón de programación por el cual solo puede haber una instancia de una clase. Suelen contar con una referencia pública estática a sí mismas para que se pueda acceder más fácilmente desde cualquier punto del código

4.3.5.2 CrabController

Esta es la clase que controla a cada cangrejo enemigo en todas las acciones que realiza. Almacena una gran cantidad de atributos que rigen propiedades como la velocidad, tamaño, resistencia, concha equipada y comportamiento de la IA, entre otras.

Uno de sus funcionalidades más relevantes sería la del desplazamiento. En cada iteración del bucle principal se llama a la función de actualización del estado de la inteligencia artificial, de manera que la máquina de estados indique cómo debe desplazarse ayudándose de su componente de *NavMeshAgent*[7]. Durante el desplazamiento, cabe a mencionar el papel de la función *MoveLegs()*, encargada de controlar la animación procedural de las patas en función del terreno. Tanto el comportamiento de la IA como el funcionamiento de la animación procedural lo veremos más adelante con más detalle.

Como podemos ver cada instancia del cangrejo cuenta con un componente indivisible AttackController, encargado de realizar los ataques. Cuenta con una función Attack llamada por el CrabController que se encarga de realizar la animación de ataque y activar temporalmente los colliders situados en los huesos de las pinzas del rig del cangrejo para que puedan ser detectados por el jugador. Cuenta además con una corrutina que controla el retardo mínimo que debe de haber entre ataques consecutivos. Por otro lado, el CrabController cuenta con una función GetHit encargada de detectar los golpes recibidos por el cangrejo principal disminuyendo la resistencia y soltando la concha si fuera necesario con la función DropShell que desvincula de la jerarquía el objeto concha y vacía la referencia a su ShellController.

4.3.5.3 ShellController

Cada objeto concha cuenta con un componente ShellController que cuenta con variables para almacenar sus atributos como el tamaño, resistencia, peso y rango de tamaño de cangrejo admitido entre otras. Cada cangrejo puede tener referenciado o no un componente ShellController que pertenece a la concha que está equipada en ese momento. En el momento en el que el cangrejo toca una concha habitable se llama a su método GetShell, el cual a su vez llama al método GetDisconfort de la concha que devuelve indica si está o no disponible para habitar y si es así llama al método PickUp de la concha colocando el objeto concha en un punto relativo al cuerpo del cangrejo marcado con un objeto vacío y pasando los atributos de la concha al cangrejo.

Por otro lado, las conchas necesitan variar los parámetros de su shader para que muestren su disponibilidad y resistencia al usuario. Aquí es donde entran en juego los métodos SetCrackedMaterial y SetAvailability. El primer método es llamado en la función GetHit del cangrejo

[7] *NavMesh* es la herramienta que incluye Unity por defecto para realizar el desplazamiento de agentes controlados por Inteligencia Artificial. Estos se mueven por superficies previamente construidas denominadas NavMesh que limitan la parte del escenario que es navegable. Todos los objetos controlados por este sistema deben contener un componente NavMeshAgent

que la habita el cual modifica el valor entre 0 y 1 del shader de agrietado. La segunda es una función suscrita al evento del cangrejo principal de cambio de tamaño, por lo que cada vez que se llama a la función SetSize se ejecuta esta función en todas las conchas de la escena modificando la visualización de la concha a inaccesible, accesible o deshabilitada.

Además este componente cuenta con funciones para el lanzamiento de la concha, de manera que se desvincule la concha del cangrejo principal y se le comunique una fuerza al *rigid body* de la concha. Esto e combina con funciones que desactivan temporalmente las colisiones con el cangrejo principal y el suelo, para evitar que colisione no deseadas durante el lanzamiento. Como las conchas deben explotar al tocar el suelo u otros cangrejos, cada ShellController cuenta a su vez con un ExplosionController que cuenta con un método Explode llamado al recibir una colisión durante el lanzamiento de la concha. Este cuenta a su vez con un *trigger collider* con el radio de la explosión que detecta a los cangrejos que están en contacto y les comunica el daño para después eliminar la concha.

4.3.5.4 PlayerCrabController

Esta es la clase que controla al cangrejo del jugador, la cual extiende de CrabController alterando algunos de sus métodos para adaptarlos y añadiendo algunos nuevos. A diferencia de CrabController solo hay una instancia de la clase al haber solo un jugador por lo que es una clase singleton auto-referenciada.

Una de las funciones que se han sobreescrito son las de su movimiento, ya que en este caso no se utiliza el desplazamiento con NavMesh, si no que se detectan las entradas recibidas por los dispositivos con el New Input System y interpretan. De esta manera, la función de movimiento aplica fuerza sobre el rigid body del cangrejo en la dirección indicada por los controles de movimiento (joystick o WASD). El cangrejo siempre se dirige a la dirección en la que está mirando, por lo que siempre se tiene como referencia del movimiento el vector forward del objeto cangrejo. En la función de movimiento del cangrejo también se rota progresivamente al propio cangrejo recogiendo el ángulo de visión en el cual está mirando la cámara, realizando una rotación amortiguada.

Por otro lado, el cangrejo principal cuenta con muchas habilidades que no utilizan los enemigos, por lo que también se detectan los eventos recibidos del input system para llamar a las funciones de LaunchShell, ShellJump y Defence.

- LaunchShell: Se encarga de llamar a las funciones que comentábamos previamente del lanzamiento de la concha con los atributos actuales del cangrejo.

- ShellJump: Para salir despedido de la concha se detecta el tiempo que se mantiene el botón de salto y al soltarlo se llama a la función de ShellJump pasándole dicho tiempo como atributo, de manera que se calcule la potencia de salto del cangrejo. En esta función también se desvincula la concha y se ignora temporalmente durante el salto,

pero en vez de comunicar fuerza a la concha se le comunica al cangrejo.

- Defence: Funciona de una manera similar que la función de ataque, pero en este caso hay un tiempo durante el cual el cangrejo no pueden recibir daños.

Otra diferencia respecto a los enemigos es que el jugador puede crecer durante la partida comiendo. Para ello cuenta con un componente FoodDetector que se encarga de detectar cuando se ha tocado una bolita de comida, momento en el que llama a la función Eat del cangrejo y se destruye la bolita. La función Eat varía la variable de tamaño del cangrejo, modificando a u vez la escala. Aunque en un principio pudiera parecer que la variación del tamaño es un simple cambio, esto afecta a muchas funcionalidades del cangrejo. Por ejemplo, conforme el cangrejo se hace grande debe aumentar de velocidad, ya que si no la sensación de movimiento será mucho menor. Lo mismo sucede con la altura a la que se coloca el cangrejo respecto al suelo, la velocidad a la que se cambian de posición las patas, el tamaño de los colliders, la fuerza con la que se salta y se lanzan las conchas, etc. En estas situaciones se han realizado modificaciones para que dichos parámetros escalen en función de valor del tamaño del cangrejo.

4.3.6 Animación procedural

La herramienta experimental Animation Rigging de Unity nos permite modificar las animaciones que vienen por defecto en los modelos que importamos. De esta manera, pone a nuestra disposición distintas *constraints* o restricciones que nos ayudan a que ciertos huesos del modelo describan un movimiento más acorde con nuestro escenario. Las que nos interesan en este caso son las relacionadas con Inverse Kinematics o IK, las cuales permiten rotar automáticamente las articulaciones a partir de puntos auxiliares como la posición en la que debe estar la punta de la pata.

Para el caso del cangrejo en el cual se animan 4 patas con dos articulaciones cada una, se ha utilizado una combinación de de dos de estas constraints:

- Multi-Aim Constraint: para situar el primer tramo de cada pata (la parte mas cercana al tronco). Esta constraint nos permite indicar con la posición de un objeto llamado *effector* a qué punto debe mirar este hueso. Sería la forma más básica de IK. Esta parte no se articula durante el movimiento del cangrejo, por lo que simplemente es para modificar la posición inicial en la que está completamente estirada y girarla hacia adelante como cuando está dentro de la concha. Se ha añadido una limitación de la rotación en el eje X para evitar que la pata se girase sobre si misma mostrando la parte inferior por arriba y creando una rotación extraña.

- Two Bone IK Constraint: para mover los otros dos huesos de la punta de la pata. Estas serían las que describirían la animación al caminar. Esta constraint cuenta con dos effectors que describen la posición de la pata. Uno indica la posición en la que está la

punta de la pata en todo momento, y el otro indica la posición de la rodilla de manera que los dos huesos se doblen hacía fuera.

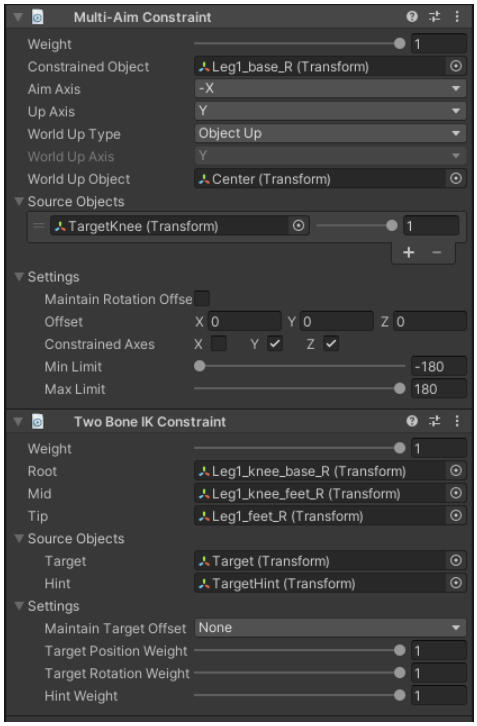

Figura 4.11: Configuración de los componentes de IK de las patas del cangrejo

En la figura 4.12 se pueden ver las posiciones de los effectors y el resultado final del cangrejo con las patas correctamente colocadas por IK, siendo modificable su posición al cambiar la ubicación de los effectors en runtime. Los puntos verdes son de las patas de atrás y los rojos de las de delante. Se puede ver como hay effectors en la punta de la pata, en la primera rodilla y otro en el aire indicando la hint sobre la que sebe rotar la segunda rodilla.

Una vez creadas las restricciones IK de las patas se pasó a modificar las posiciones de las puntas de las patas mediante la proyección de puntos fijos respecto a la posición del cuerpo. Se tienen unos puntos situados en las posiciones originales de cada punta, considerada como la posición ideal para el cangrejo. Estos puntos son hijos del cangrejo, por lo que se desplazan con él, pudiendo proyectar un rayo en el eje Y obteniendo el punto sobre el suelo en el que debería estar situada la punta de la pata. Con esta posición óptima en la que deberían estar las patas del cangrejo simplemente es necesario calcular la distancia actual de la punta respecto a la posición calculada. En el momento en el que esa distancia supere un umbral se deberá cambiar el effector de la pata a la posición óptima en ese instante, de manera que se quede ahí hasta que se vuelva a superar el umbral.

Después se han desvinculado los effectors de las puntas del cangrejo haciendo un prefab con estos, de manera que cada vez que se crea un cangrejo automáticamente se crean estos

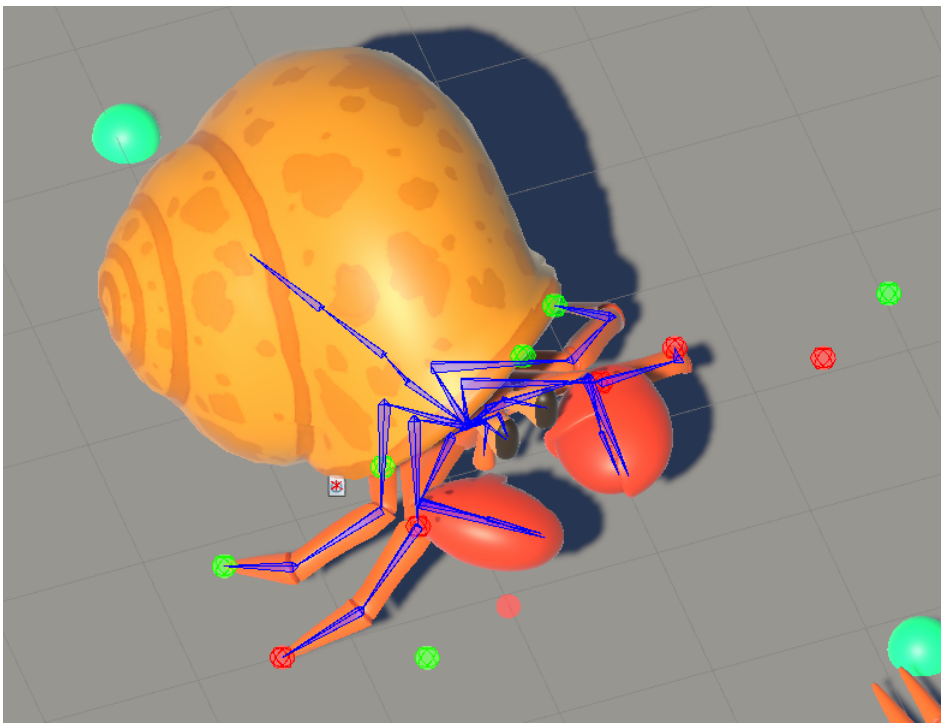

Figura 4.12: Localización de los effectors de las constraints IK que rigen el movimiento de las patas (verdes: patas traseras, rojos: patas delanteras)

effectors y se asocian a los targets de las constraints de IK de ese cangrejo. Tal y como funciona este paquete todavía experimental de Unity surgieron ciertos problemas ya que no se puede alterar los targets que utilizan las colisiones una vez ha comenzado la ejecución. Esto se solventó simplemente desactivando el componente RigBuilder[8] inicialmente, asignando los targets y finalmente activando el RigBuilder una vez terminado.

Para mejorar el efecto se ha iterado para hacer que el desplazamiento no sea inmediato, si no que se desplace hasta el punto óptimo poco a poco en un tiempo parametrizado. De esta manera, en las situaciones que los cambios de posición de las patas no sean tan frecuentes no resulta tan notorio este cambio, al estar suavizado. Para hacer este desplazamiento progresivo se ha hecho uso de corrutinas que se encargan de desplazar mediante interpolación del punto original erróneo al óptimo en cierta cantidad de tiempo. En el momento en el que se vuelve a superar la distancia umbral si hay una corrutina de desplazamiento en marcha para esa pata se detiene y se genera una nueva.

Además del movimiento de las patas, también se ha realizado un sistema básico para controlar la altura a la cual se colocaba el tronco del cangrejo respecto a la posición de sus patas. Dado que surgieron ciertos inconvenientes en una implementación más transparente para el juego, el cuerpo del cangrejo se nivela con un box collider que hay situado en su parte

[8]Componente troncal que hay que añadir a un modelo para que se puedan aplicar constraints de la herramienta Animation Rigging

superior, el cual eleva al cangrejo a su posición deseada. La altura que tiene este collider se modifica constantemente para que se eleve una cantidad X de altura respecto a la altura media de las puntas de las patas. De esta manera se consigue regular la altura en las pendientes. Además se controla el tamaño máximo que puede tener el collider para que no actúe a la hora de saltar de la concha.

4.3.7 Inteligencia Artificial

Como ya hemos comentado previamente, los cangrejos enemigos deben mostrar cierto comportamiento en función de las acciones del jugador y las conchas que encuentren en su entorno. Para ello se ha creado una máquina de estados compuesta de 3 estados en los que se puede encontrar cada enemigo:

- *Wander* o paseo: Estado en el que se encuentra por defecto el cangrejo, en el cual se limita a ir de una posición a otra del mapa y pausándose en ciertas ocasiones. El rango del ángulo y la distancia en los cuales se puede encontrar el siguiente así como el rango de tiempo y probabilidad de las pausas están parametrizados.

- *Pursue* o persecución: Al detectar una concha mejor que la suya que pueda equipar, ya sea una concha libre o la del jugador, el enemigo se dirigirá rápidamente a su posición. En el caso de recibir un impacto también se perseguirá al jugador. Si el objetivo sale del radio de huida entonces se dejará de perseguir y se volverá al estado wander.

- *Attack* o ataque: En el momento en el que el cangrejo del jugador entre en el radio de ataque comenzará a atacar cada cierto tiempo hasta que se aleje. El intervalo de tiempo entre golpes es aleatorio y su rango de valores también está parametrizado.

Para la implementación de la máquina de estados se ha creado una interfaz con todos los métodos que debe incorporar cada estado que compone la máquina y acto seguido se han creado una clase que extienda de la interfaz para cada estado, de manera que se definiese su comportamiento. De esta manera se puede manejar a los 3 estados de una manera de la misma manera independientemente del estado en el que se encuentre. De esta forma en cada iteración del bucle principal se llama a la actualización del estado actual y todos los estado cuentan con funciones para cambiar a otro estado y detección de eventos.

Para detectar cuando es necesario un cambio de estado, se utilizan varios colliders vinculados al cangrejo que permiten detectar eventos para cambiar el comportamiento. Estos colliders tienen forma esférica y son de tipo *trigger*, por lo que se pueden atravesar sin afectar a *rigid bodies*. Además, se han configurado para ser ignorados por el trazado de rayos, ya que podrían interferir en el funcionamiento de otros módulos del juego. Cuando se detectan eventos de entrada y salida de conchas o el cangrejo principal en cada uno de los collider trigger se le comunican al estado actual de la IA del cangrejo para que interprete esa señal y cambie o no su comportamiento. Por cuestiones de eficiencia se ha utilizado la asignación

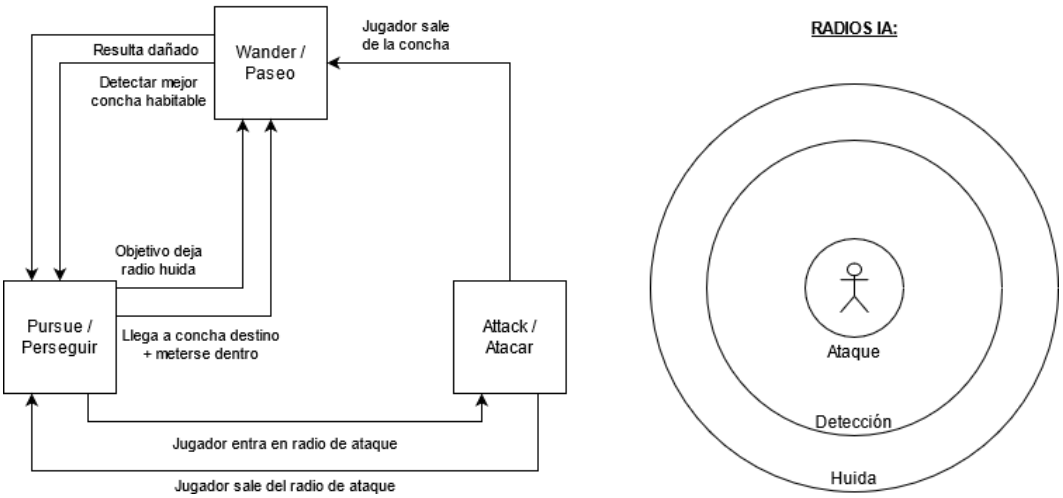

Figura 4.13: Izquierda: Esquema de comportamiento de la máquina de estados de la inteligencia Artificial de los cangrejos enemigos. Derecha: Esquema de la colocación de los 3 colliders de detección que intervienen en el comportamiento

de los objetos detectables a capas distintas, de manera que solo se detecten las colisiones con este tipo de objetos. Además de los eventos de los triggers también se utiliza el evento de recibir daño.

5 Diseño del nivel

Algo muy importante a la hora de crear un videojuego es su diseño de niveles. El cómo estén situados los elementos de la escena de juego pueden suponer causar sensaciones distintas en el jugador. Es habitual pensar que este diseño solo se aplica a juegos con una división de niveles muy marcada y simplificada, pero en realidad esto afecta a todos los tipos de juego, incluyendo a los rogue-like a pesar de su importante factor de generación procedural.

En nuestro caso, el juego consta de un mapeado fijo con una variación de los puntos de aparición de enemigos, conchas y comida. Se trata de un diseño más *sandbox*[1] por lo que no se fuerza a que el jugador siga un camino guiado. Al inicio de la partida el jugador ya conoce los objetivos del juego y debe explorar por su cuenta para conseguir ganar. Sin embargo en el proceso, el diseño del juego puede ayudarlo en su tarea.

Un claro ejemplo de esto, es la posición inicial del cangrejo. Nada más empezar, la cámara muestra a nuestro cangrejo en el centro, lo cual presenta rápidamente al avatar que va a encarnar el jugador, algo que rápidamente entenderá al tocar los controles y ver que se desplaza.

Justo enfrente de el se muestran las tres conchas disponibles en el juego con carteles con la menos cantidad de texto posible que expliquen las diferencias entre ellas. Estas conchas están iluminadas, por lo que llaman la atención del jugador e instintivamente las prueba, viendo claramente sus efectos diferenciadores gracias a los letreros y a la propia experimentación del jugador. Además, al ser el inicio la parte de la partida con dificultad más reducida la colocación de estas 3 conchas también cumple la función de ofrecer la posibilidad de equiparse con una concha a su elección, ofreciéndole cierta ventaja y protección iniciales.

En la parte superior de la pantalla también podremos ver una concha muy grande y llamativa colocada encima de un montículo de piedras. Dado que este elemento es único en el mapa y se nos presenta nada más comenzar la partida, nos da a entender que juega un papel relevante. Además la concha se diferencia de las demás que tenemos en pantalla por tener un acabado dorado, dando a entender que es de oro y por ende valiosa. Si a esto le sumamos el propio título del juego, todo indica que pertenece a la ”realeza” de los cangrejos. No deja de ser una concha como las demás, por lo que debe de poderse equipar, pero parece demasiado grande e inaccesible, al menos de momento. Una vez el jugador descubra como las

[1] Los juegos de tipo *sandbox* son aquellos en los que se le presenta al jugador un terreno de juego que puede explorar libremente. Algunos ejemplos muy conocidos de este tipo de juegos serían la saga *Grand Theft Auto*, *Minecraft* o *The Witcher*.

Figura 5.1: Captura del estado inicial y ángulo de visión al iniciar una partida

conchas se van haciendo habitable conforme crezcamos podrá atar todos los cabos llegando a la conclusión casi inevitable de que se trata del objetivo final que nos otorgará el título del rey ermitaño.

Si seguimos analizando la visión desde la posición inicial de nuestro cangrejo podremos además ver a los laterales otros carteles con unas palabras y poco texto, descubriendo así un tutorial de los controles incluido dentro del propio mundo del juego. Estos letreros son casi imposibles de pasar por alto por lo que el jugador se interesará por ellos. Esto sucede de la misma manera en otros juegos como por ejemplo el conocido rogue-like *The Binding of Isaac* que tiene sus controles impresos en el suelo de la primera cámara del juego. Las rocas de tutoriales están colocadas de tal manera que no quepan todas en el recuadro de visión inicial, para no agobiar al jugador con demasiada información. Solo se muestra parte de ellos, de forma que al desplazarse para ir a verlos vea las demás rocas. El orden de visualización de estas rocas también está pensado para comenzar viendo conocimientos más básicos como el ataque y pausa a los más avanzados y opcionales de salto y lanzamiento.

Se podría considerar que este tutorial es obligatorio de ver en todas las partidas, pero tiene la ventaja de que se puede ignorar fácilmente en partidas posteriores, no estorbando al jugador. Por otro lado, los controles del juego, son muy sencillos de aprender al ser muy parecidos a los de muchos juegos en 3D, por lo que no es casi necesario aprender cómo explorar el mapa si el jugador está mínimamente familiarizado con los videojuegos en general. Se realizan asociaciones habituales con otros tipos de juegos como los shooters, que utilizan el clic izquierdo o el gatillo derecho trasero para atacar y el espacio o el botón A para saltar, minimizando la curva de entrada al juego.

Dada la sencillez del mapeado, los elementos clave destacan más, por lo que al poco de empezar el jugador descubrirá rápidamente que hay más conchas situadas a lo largo del mapa,

Figura 5.2: Habitación tutorial en la que se comienza cada partida en *The Binding of Isaac*

y que la mayoría parecen más grandes que las 3 conchas iniciales. Otra diferencia que podrá ver claramente es que esas conchas no están iluminadas y que a poco que se intente interactuar con ellas no serán habitables. Además, conforme vaya aumentando de tamaño verá que esas conchas comienzan a iluminarse. Con todos estos datos el jugador puede llegar fácilmente a la conclusión de que las conchas iluminadas son las que podemos ocupar con nuestro cangrejo, y que, a medida que crezcamos, podremos ocupar conchas más grandes. Se aprovecha de esta manera el uso de la iluminación para guiar al jugador a las conchas que le pueden ser útiles, que ofrece unas indicaciones implícitas que el jugador seguirá instintivamente.

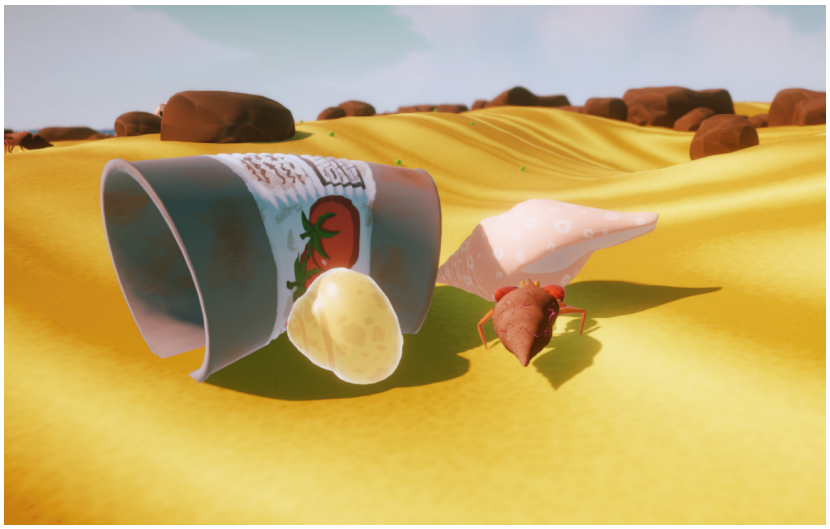

Figura 5.3: Indicación de la disponibilidad de las conchas en función del tamaño de la concha y el del jugador

Si el jugador sigue creciendo, verá que las conchas pequeñas por el contrario comienzan a

apagarse y que se vuelven a hacer inocupables. Por una sencilla asociación de efectos opuestos el jugador puede extrapolar que si las conchas iluminadas eran las que se podían ocupar en algún momento, las oscuras que parecen deshabilitadas y parecen mucho más pequeñas, efectivamente son las conchas con un tamaño inferior al que podemos aceptar. Esto será todavía más sencillo de ver si además el jugador crece demasiado con una concha equipada, ya que inmediatamente se volverá oscura y comenzará a agrietarse hasta caerse.

Dado el crecimiento constante del cangrejo y las limitaciones de tamaño que tienen las caracolas, se crea un ciclo en el que se fuerza al jugador a ir cambiando la concha que tiene equipada. De esta manera se consigue obligar al jugador a salir de su zona de confort y adaptarse a cada situación. Esta mecánica ocasiona que se descubra más contenido del juego otorgando una mayor diversidad. Esto se combina con el uso de las habilidades de salto y lanzamiento que implican perder la concha temporal o permanentemente, las cuales resultan mucho más útiles al tratar con conchas de una naturaleza más efímera.

Por otro lado, el jugador enseguida se topará con los demás cangrejos del mapa. Es aquí donde comienza la dificultad de la progresión del jugador a lo largo de las partidas. El jugador inicialmente desconocerá el comportamiento de los cangrejos. Lo único que podrá deducir es que se tratan de cangrejos iguales que los de su avatar, y que cuentan con habilidades similares como la de equiparse conchas. A medida que el juego avance, descubrirá en mayor profundidad los comportamientos que tienen estos cangrejos mediante la experimentación, como por ejemplo que si se acerca mucho a ellos lo perseguirán y lo atacarán, o que si detectan a una mayor distancia que lleva una concha mejor que la suya intentarán robársela atacándole.

Figura 5.4: Los cangrejos más rojos son lo suficientemente más grandes que el jugador como para causarle más daño que un golpe de cangrejo convencional. Mediante el tintado rojo se indica al jugador que son peligrosos

A priori, estos encontronazos iniciales sin información alguna pueden parecer injustos, pero en realidad son una manera de aportar información acerca del juego. Por ejemplo, si durante estos primeros enfrentamientos el cangrejo principal muere, el jugador será capaz de entender que si lleva una concha conforme le golpeen, se irá agrietando más hasta llegar al punto de perderla, y que en el caso de no llevar ninguna concha equipada al recibir un golpe, morirá. Conforme avance y se enfrente a más cangrejos aprenderá lógica más avanzada del juego como que unas conchas soportan más golpes que otras a cambio de velocidad de movimiento, y así sucesivamente hasta dar con la manera de superar el juego.

Se crea así una progresión entre partidas propia de los rogue-likes que obliga al jugador a que mejore no solo sus habilidades con los controles durante el combate, si no también su conocimiento de todas las mecánicas del juego, algo imprescindible para hacerse con la victoria. Un jugador inexperto por ejemplo, podría no hacer uso de la defensa correctamente al no conocer los patrones y tiempos de ataque de los enemigos, pereciendo en el intento, mientras que un jugador experimentado podrá esquivar más golpes y hacer uso del lanzamiento de conchas o el salto para sobrevivir a más enfrentamientos.

Figura 5.5: Al evadir un golpe enemigo con éxito se indica con unas chispas y un sonido, aportando un *feedback* útil para el usuario

El papel del terreno también está diseñado para ayudar al jugador. En primer lugar, el terreno de juego es una extensión cuadrada notoriamente delimitada. Se trata de una superficie con una forma simple que resulta sencilla de explorar. Esto sumado a la progresiva pendiente que tiene la playa, el mar que solo está situado en la parte baja de la playa, y la concha dorada situada en el centro del mapa, ayudan al jugador a tener muchos puntos de referencia con los que saber fácilmente en qué lugar del mapa se encuentra el jugador. De esta manera se puede prescindir de indicadores no diegéticos que ensucien la interfaz del juego como un minimapa o flechas de indicación de elementos interesantes, siguiendo con el deseo de mantener una interfaz lo más simplificada posible.

El crecimiento del cangrejo del jugador también juega un papel importante en cuanto a las sensaciones del jugador. Inicialmente el cangrejo del jugador no cuenta con apenas información del terreno en el que se sitúa, teniendo una cámara muy cercana al suelo. Sin embargo conforme se incrementa su tamaño, el jugador es capaz de vislumbrar más parte del terreno dotándolo un mayor control del entorno. Esto favorece la sensación de que cuanto más grande es el cangrejo más poderoso es. Es capaz de ver lo que ha avanzado al ver compararse con el terreno y con otros cangrejos más pequeños que podrían ser como era él al comienzo de la partida causando en el jugador una sensación de empoderamiento.

6 Manual de usuario

En este capítulo se comentará toda la información relevante para que el jugador pueda jugar a Hermit King correctamente. Para ello se darán explicaciones acerca de los controles, mecánicas y dinámicas que son necesarios conocer para completar el juego. También se expondrán los requisitos de los dispositivos con los que debe contar el jugador para poder jugar con unas condiciones aceptables al juego.

6.1 Instrucciones de juego

Este sería el mapeo de controles y acciones que incorpora el juego finalmente y que se puede ver en el tutorial (se han utilizado el nombre de los botones de un mando de XBox, pero sería valido para cualquier tipo de mando):

- Desplazamiento del cangrejo: Joystick izquierdo / WASD
- Rotación de la cámara: Joystick derecho / Ratón
- Ataque: Gatillo RT / Click izquierdo
- Defensa: Gatillo LT / Click derecho
- Saltar de la concha: Botón A / Espacio
- Lanzar concha: Botón X / Q
- Poner/Quitar pausa: Start / Escape

Como ya hemos visto, el juego se genera aleatoriamente para cada nueva partida y da libertad al jugador para desplazarse por el mapa, por lo que al contrario que otros juegos, no hay una serie de pasos fijos a seguir que nos lleve a la victoria, si no una serie de estrategias que nos acercarán a la concha dorada.

La estrategia más evidente para conseguir la victoria es recoger todas las bolitas necesarias para alcanzar el tamaño necesario para ocupar la concha dorada, evitando el enfrentamiento

completamente. No obstante, este camino resulta bastante tedioso y largo, y no asegura que no podamos encontrar amenazas. Podemos acercarnos demasiado a un cangrejo, o que otro detecte la concha que llevamos equipada y se convierta en hostil. La opción más correcta sería enfrentarse a los cangrejos más a menudo, ya que pueden dejar caer unas cuantas bolitas de comida simultáneamente que nos puedan hacer crecer de una manera más rápida que la exploración. Además, conforme se vayan cogiendo las bolitas repartidas por el mapa, serán más escasas y por tanto laboriosas de encontrar, por lo que se insta cada vez más al enfrentamiento.

Para enfrentarse a los cangrejos es importante conocer su comportamiento y características para poder actuar en consecuencia. En capítulos anteriores ya se ha explicado el comportamiento con más detalle, por lo que daremos esto por sabido y nos centraremos en las estrategias más eficientes para abordar un encuentro con un enemigo.

Inicialmente, los cangrejos no son hostiles, por lo que podemos aprovechar esta baza para atacarlos rápidamente por la espalda evitando las pinzas, que podrían dañarnos. Esto es muy útil para los cangrejos que no tienen concha o que incluso tienen una concha poco resistente, a los cuales podremos derrotar rápidamente, consiguiendo mucha comida fácil. En caso de que las peleas con los cangrejos sean más largas resulta imprescindible aprovechar su dificultad para girar, moviéndonos en círculo atacándole mientras que evitamos sus pinzas.

A la hora de enfrentarse a un rival, la concha equipada tiene un papel relevante. Evidentemente, es preferible equiparse cualquier concha antes de entrar en combate, ya que resulta fácil que un enemigo nos hiera, sobre todo si tiene una concha robusta. La elección del tipo de concha también condicionará la forma en la que combatiremos. En una situación en la que contemos con una mayor resistencia podemos aprovechar para atacar frontal y rápidamente al enemigo, soportando los golpes y consiguiendo una victoria.

Sin embargo, en el caso de equipar una concha ligera o directamente no contar con ninguna, podemos aprovechar nuestra velocidad superior para evitar los golpes y asestar los nuestros. Esta se trata de una opción más arriesgada, pero en la que podemos resultar ilesos si aprovechamos que entre cada ataque enemigo hay cierto margen de tiempo. En estas situaciones también es muy útil la habilidad de defenderse para esquivar los golpes. En el momento en el que veamos que el enemigo comienza la animación de atacar podemos pulsar el botón de defender para bloquearlo. Si somos lo suficientemente rápidos es posible bloquear cada ataque en cuanto comience su animación y además gozaremos de unas décimas de segundo durante las que el cangrejo estará cubierto.

El juego está diseñado de tal manera que se deba cambiar de concha constantemente, por lo que frecuentemente encontraremos que nuestra concha se está quedando pequeña disminuyendo su resistencia, o directamente que están muy agrietadas, o incluso nos topamos con una situación en la que tenemos demasiadas conchas y no solo podemos llevarnos una. En estos casos resulta muy útil la habilidad de lanzamiento de la concha.

Esta causa mucho daño a los enemigos, y en la mayoría de situaciones hará que los enemigos

suelten su concha, siendo mucho más fácil conseguir una concha o incluso vencer enemigos para conseguir comida. Por tanto es muy aconsejable utilizarla siempre que tengamos conchas con poca utilidad. Las explosiones tienen daño en área en un pequeño radio, por lo que también puede ser interesante su uso para cangrejos que están muy juntos, como puede ser durante una persecución de varios cangrejos.

Por otro lado, la habilidad de salto nos aumenta la movilidad por lo cual agiliza el movimiento a través del mapeado. En una situación desesperada puede servir también para escapar de un combate, ya que nos ofrecerá una forma rápida de alejarnos de nuestra posición y además al quedarnos sin concha podremos movernos más rápido, pudiendo incluso salir del radio de detección de nuestro enemigo.

Conforme nuestro cangrejo crece, los cangrejos que aparecerán también serán más grandes, y por ende nos podrán hacer más daño y sus radios de detección serán más grandes. Cuando somos pequeños es poco recomendable enfrentarnos a ellos, ya que partimos en desventaja, otorgan la misma recompensa al vencerlos que los demás y no podremos utilizar su concha por el momento. Conforme avance la partida, los cangrejos pequeños irán desapareciendo, quedando solo los grandes, por lo que debemos tener en cuenta que hay muchas más probabilidades de que nuestra concha sea buscada por otro cangrejo.

6.2 Requisitos de hardware

Para averiguar los requisitos de hardware necesarios para ejecutar Hermit King se han realizado pruebas en varios ordenadores con versiones de escritorio y navegador. En la tabla 6.1 se pueden ver las características de los equipos utilizados durante las pruebas de rendimiento. Se realizaron pruebas en las mismas condiciones y se anotaron los fotogramas por segundo (FPS) que se obtenían de media, así como el porcentaje de uso del procesador y la tarjeta gráfica, obteniendo los resultados de la tabla 6.2 para ambas versiones.

Equipo	Tipo	Pantalla	CPU	GPU	RAM
1	Sobremesa	2560x1440px (144Hz)	Intel i7-6700K (4,20 GHz)	Nvidia GTX 1060 (6GB)	24GB
2	Portátil	1920x1080px (60Hz)	Intel i5-10300H (4,50 GHz)	Nvidia GTX 1650 Ti (4GB)	16GB
3	Portátil	1920x1080px (60Hz)	Intel i7-8550U (4,00 GHz)	Nvidia MX150 (2GB)	8GB

Tabla 6.1: Características de los equipos utilizados para las pruebas de rendimiento

Basándonos en los juegos y plataformas que han salido en los últimos años podríamos considerar que el mínimo de fluidez exigido a un juego sería que alcanzara los 30 FPS estables corriendo el juego a una resolución de 1080p (1920x1080 píxeles). Como podemos ver, la versión de escritorio supera con creces esta cifra y roza o incluso supera los 60 FPS en los 3

Equipo	Versión	FPS	Uso GPU (%)	Uso CPU (%)
Equipo 1	Escritorio	135-144	70-95	25-35
	Navegador	65-115	55-70	20-30
Equipo 2	Escritorio	60	25-35	20-30
	Navegador	25-35	100	25-40
Equipo 3	Escritorio	50-60	75-90	15-25
	Navegador	10-15	100	30-40

Tabla 6.2: Resultados obtenidos por cada equipo en las pruebas de rendimiento de las versiones de escritorio y navegador del juego.

equipos, que podríamos considerar como la cifra óptima. En cuanto a la versión de navegador, se puede ver que tiene un rendimiento mucho peor. Esto no es extraño, ya que los navegadores no están tan enfocados a este tipo de contenidos y sobre todo a la visualización de modelos 3D. Sin embargo, todavía podemos ver que muestra un rendimiento aceptable en el segundo equipo y que en el ordenador de sobremesa se siguen alcanzando estos 60 FPS.

No se han podido realizar pruebas en equipos que no contaran con gráfica dedicada, aunque viendo los resultados del tercer equipo que cuenta con una gráfica de gama baja para portátiles es posible que se pudiera jugar a la versión de escritorio de Windows a unos 30 FPS en un ordenador con procesador Intel i5 y con la tarjeta gráfica que viene integrada en los ordenadores de Intel. En el caso de contar con una gráfica dedicada como las de la económica serie MX de Nvidia o superior es posible que se alcanzaran incluso con un procesador Intel i3 o AMD Ryzen 3.

De todos modos, los requisitos mínimos recomendados y rondar los 60 FPS para jugar a Hermit King serían los del Equipo 3 o similares. Parece que la cantidad de memoria RAM no afecta demasiado al rendimiento del juego, pero dados los sistemas actuales se recomendaría al menos contar con 8GB de memoria RAM. Tampoco se ha podido probar en dispositivos Mac, pero dados los resultados se estima que podría funcionar en casi todos los dispositivos Mac que han salido en los últimos 5 años.

En cuanto a periféricos, si bien es recomendable utilizar un mando, su uso no es obligatorio, pudiendo utilizar cualquier combinación de ratón y teclado. Es recomendable el uso de un mando de XBox, que cuentan con una mejor integración en ordenadores. Además el tutorial del juego muestra el mapeo de botones con los controles de XBox. No obstante sigue siendo posible utilizar un mando de Play Station o cualquier otra marca compatible con ordenador.

7 Conclusiones

Durante este capítulo se harán reflexiones acerca del resultado conseguido en este proyecto, así como del proceso de desarrollo. También se expondrán los pasos que se deberían realizar a continuación en el caso de querer conseguir un producto comercial de mayor envergadura.

7.1 Evaluación del resultado y consecución de objetivos

La versión final de Hermit King consta de un ciclo de juego completo que ofrece ciertos retos al jugador, una pequeña progresión con contenido suficiente para invitar a jugar a realizar varias partidas, y un apartado visual agradable y un concepto original que llaman la atención del jugador, por lo que en términos generales se puede considerar que se ha conseguido un resultado acorde a lo planificado.

Entrando más en detalle de los objetivos conseguidos también podemos considerar que se ha conseguido un juego accesible con una interfaz minimalista que consigue guiar al jugador con elementos bien integrados en el mundo sin la necesidad de tener un HUD que pueda agobiar con demasiados elementos al jugador. Cuenta con unos controles bastante sencillos que son compatibles con mando y teclado, mejorando la adaptación del jugador. Estos controles además están presentados con un tutorial conforme comienza la partida que aporta la información de una manera más visual y sin texto que pueda perder la atención del jugador.

El tutorial está situado de tal manera que es imposible pasar inadvertido para jugadores nuevos, pero con la posibilidad de que jugadores más experimentados puedan ignorar al tratarse de unos elementos situados en el propio mundo. En cuanto al resto de comportamientos que están presentes en el juego no se explican textualmente, si no que se deja al jugador que los descubra por su cuenta dando herramientas que lo faciliten, como la iluminación de las conchas.

En cuanto a los actores participantes en el juego se ha conseguido reflejar el comportamiento de los cangrejos ermitaños a través de una combinación de un sistema de inteligencia artificial, y un modelo con las animaciones adecuadas. Cabe a destacar en este apartado la implementación con éxito del sistema de animación procedural de las patas que se adapta al escenario y que inicialmente se tomaba como uno de los mayores escollos que incluía el desarrollo por la inexperiencia en ese apartado. Si bien no se ha conseguido crear un sistema tan pulido como estaba en mente sí que se ha conseguido un sistema bastante robusto, versátil y

aparente del movimiento de las articulaciones.

Gracias a la IA y habilidades del cangrejo, se ha conseguido una experiencia de combate con cierta profundidad que ofrece varias opciones de afrontar un combate y variaciones relativas a la mecánica de las distintas conchas. Se consigue así sacar al jugador de su zona de confort para que pruebe distintas posibilidades y pruebe todo el contenido del juego, mejorando el entretenimiento que puede conseguir con el título.

En cuanto al apartado visual, se ha conseguido un aspecto en la línea de lo que se buscaba, con una estética simplificada y cercana a lo naif y a lo *cartoon* en cuanto a modelados, texturas, animaciones y efectos. En este apartado podemos destacar aparte del modelo y texturas de los cangrejos y conchas, la elaboración del útil shader de agrietado de la concha que se ha realizado completamente desde cero, así como la mejora visual que ha aportado la configuración de las opciones de iluminación y post-procesado del juego que han permitido añadir una sensación de distancia focal, y modificar los colores para formar una paleta concreta de tonos predominantemente anaranjados. Todos los elementos del juego han participado para crear una estética desenfadada y animada que se buscaba desde un inicio, con mención especial a los sonidos y efectos de partículas.

Además de esto se ha conseguido que Hermit King goce de un buen rendimiento permitiendo que se pueda ejecutar en un mayor abanico de dispositivos. Se han realizado pruebas en distintos dispositivos e incluso se han probado builds en navegador, que por defecto suelen ofrecer un peor rendimiento, y el resultado ha sido bastante positivo ofreciendo una buena fluidez que permita jugar en buenas condiciones.

7.2 Problemas encontrados

No obstante, al igual que sucede en la mayoría de proyectos, han surgido inconvenientes y problemas no contemplados durante su desarrollo. La división y planificación del trabajo por tareas que se realizó inicialmente, si bien ha sido una buena referencia que seguir, no ha reflejado al completo el seguimiento real del desarrollo. Conforme se avanzaba en el proyecto iban surgiendo nuevas tareas que no estaban contemplados a priori, un aumento de la complejidad de algunas tareas que hacían sopesar si eran o no llevadas a cabo, e incluso cambios en el diseño del propio juego que hacían que el desarrollo cambiara de dirección. Esta disparidad entre lo planeado y la situación real no se finalmente no se han distanciado en exceso, pero si que se la distancia entre ellas se hacía mayor a medida que avanzaba el desarrollo.

Un ejemplo de esto ha sido por ejemplo el desarrollo del sistema de animación procedural. En la planificación estaba contemplado que el sistema estuviera mucho mas pulido, permitiendo al cangrejo nivelar y rotar en todo momento su cuerpo respecto a la superficie, y alternar el movimiento de las patas en un patrón de zig-zag. No obstante, finalmente se prescindieron de estos detalles debido a la relación de coste que suponían en tiempo de desarrollo y la mejora que supondrían en el comportamiento del cangrejo. El zig-zag se descartó porque de por sí

las patas ya se desplazaban alternamente la gran mayoría del tiempo. En cuanto al nivelado y rotación del cuerpo en función de las posición de las patas se intentó abordar de distintas maneras que fracasaron debido, entre otros motivos, a que el cangrejo debía controlarse con un rigid body no kinematic que complicaban su implementación. Finalmente se descarto rotar el cuerpo del cangrejo y se creó un rudimentario sistema de añadir un collider de altura variable que lo separase del suelo para hacer el nivelado por una cuestión de tiempo.

La estimación del tiempo de dedicación de tareas tampoco se ha desviado demasiado por lo general, pero por lo general han tendido a necesitar más tiempo del estimado. Un claro ejemplo ha sido la tarea de implementación de la inteligencia artificial, que ha llevado aproximadamente el triple de tiempo del estimado. Inicialmente se pensaba crear una inteligencia muy simple que no costase mucho tiempo por lo que se hizo una estimación demasiado optimista, pero a medida que avanzaba el desarrollo nuevas condiciones tenía que contemplar la inteligencia artificial, consumiendo mucho más tiempo de desarrollo.

Muchos de estos problemas se podrían haber evitado con una mejor estructura de las clases y funcionalidades por módulos del código. Si bien el código inicialmente estaba correctamente estructurado y separado, a medida que crecía la complejidad del juego se han creado clases demasiado largas que agrupan demasiadas funcionalidades y que resultarían más complejas de mantener en el caso de alargarse el desarrollo, requiriendo un laborioso proceso de refactorización. Esto está en gran parte ocasionado por las desviaciones respecto a lo planificado inicialmente, la falta de tiempo y la poca experiencia en el desarrollo de proyectos del estilo. En este caso, esto no resulta tan problemático al no pretenderse continuar el desarrollo, y sobre todo no contar con otros participantes que tuvieran que toparse con el, pero sigue siendo un punto en que hay cierto margen de mejora.

Otro problema relevante no contemplado a priori fue el impacto que tendría en todos los sistemas el cambio de tamaño del cangrejo. Al modificar la escala del personaje principal se alteraba el funcionamiento de la física de sus habilidades, las animaciones, tamaño de los colliders, el daño causado a otros enemigos, la velocidad de movimiento, el comportamiento de la IA, etc. Fue necesario aplicar modificaciones a todos estos sistemas involucrados para que muchos de sus parámetros escalaran en función del tamaño que tenía en ese momento. Esto también hacía el periodo de testeo del juego más laborioso, ya que era necesario comprobar el funcionamiento de todos los sistemas relacionados con el cangrejo principal, con distintas escalas del cangrejo.

También han aparecido complicaciones relativas al propio diseño del juego. El interesante concepto de crear un juego sobre cangrejos ermitaños, si bien ofrece un concepto original y llamativo para un juego, ha resultado más difícil de adaptar a una experiencia divertida. Parte de esta complicación está ocasionada precisamente por la innovación que supone el concepto. Al no contar con claros referentes sobre los que tomar nota de su diseño resulta más laborioso crear la lógica del juego que si el concepto elegido hubiera sido, por ejemplo, un *shooter* en primera persona tradicional con una ambientación concreta, para el cual hay una gran cantidad de juegos de los que poder tomar nota.

Si bien podemos encontrar muchos juegos del género rogue-like, hay una gran disparidad entre el tipo de *gameplay* que ofrecen, y no se ha encontrado un juego con un claro parecido al que se quería hacer. Este entre otros motivos, han ocasionado que ciertos comportamientos hayan variado desde su concepción inicial en pos de hacer un juego más entretenido y accesible, pero siempre ceñido al tiempo del cual se disponía. Un ejemplo de esto sería la adición del tutorial para aprender a jugar que no estaba contemplado al principio.

7.3 Futuras mejoras

Quizás el punto donde más flaquea Hermit King de cara al consumidor es la diversión que es capaz de otorgar al jugador. La experiencia que ofrece la versión final pasa por centrarse en recoger bolitas constantemente por el mapa, cabiendo la posibilidad de superar el juego evitando casi por completo las partes más interesantes del juego como son el combate y el equipamiento de conchas. Si bien es positivo dar libertad al jugador para que intenta afrontar los retos como le apetezca en este caso parece que se le de demasiada. No hay alicientes suficientes para estar equipando continuamente una concha ni para tener muchos enfrentamientos con otros cangrejos, que era lo que se pretendía con su diseño. Si el jugador evita estos puntos tan principales de la jugabilidad, su experiencia estará más centrada en una tarea repetitiva poco estimulante como es recoger bolitas sin más. De esta manera el jugador puede llegar a tener visión incompleta y mucho más negativa del título.

Una de las posibilidades que se estudiaron al inicio de la fase de diseño fue la inclusión de gaviotas enemigas. Se tratarían de enemigos que atacarían al cangrejo principal desde el aire avisando previamente de ello. En el momento de atacar una pata de la gaviota aparecería sobre el cangrejo y lo eliminaría automáticamente. Finalmente se descartó para simplificar el desarrollo y poder dedicar más tiempo a pulir el resto del juego. Sin embargo, de cara a una futura versión sería interesante su uso para solventar esa carencia de incentivos para estar continuamente en la concha. Haría mucho más relevante el papel de las caracolas y las estrategias que seguiría el jugador para no quedarse desprotegido.

Para ello se podría hacer que una gaviota apareciera a los pocos segundos de que el ermitaño principal se quedara sin concha. Iría apareciendo una amenazante sombra sobre nuestro ermitaño que se iría haciendo cada vez más grande para finalmente sonar un graznido de gaviota y apareciera una pata de gaviota que se llevara al cangrejo, terminando la partida. En el momento en el que el ermitaño entrara en una concha, la sombra desaparecería rápidamente generando así una presión constante que forzaría a buscar una concha siempre que fuera posible. Esto a su vez también dotaría de más peso al combate, los cuales se tendrían que hacer más frecuentemente y en las ocasiones en las que el protagonista se quedara sin concha deberían acabarse rápidamente. Esto también conllevaría una serie de cambios en la progresión de la partida para que se siempre se ofreciera la posibilidad de habitar una concha y no quedarse a la intemperie injustamente.

Otro punto a mejorar sería el combate por tratarse de la parte troncal de la experiencia.

Aunque se ha procurado dar ciertas herramientas a la disposición del jugador para dotarlo de cierta profundidad, sigue siendo un combate bastante rudimentario y simplificado. En una futura iteración del juego se podría mejorar este apartado para que tuviera más capas de profundidad. Para mantener un mayor tiempo entretenido al jugador sería necesario que el combate no fuese difícil de llevar a cabo a priori a un nivel básico, y que diera un margen de mejora hasta manejarlo por completo. Es decir, debería haber una mayor diferencia entre el combate que fuera capaz de realizar jugador experimentado que uno novato, a pesar de contar con las mismas herramientas iniciales en ambos casos.

Esto implicaría ajustar las mecánicas actuales (el lanzamiento por ejemplo, debería poderse regular y ser más preciso) y añadir otras nuevas de manera que resultase muy útil su utilización pero laborioso su aprendizaje. Siguiendo en esta línea por ejemplo, se podría reducir el tiempo de la habilidad de defensa que actualmente no tiene demasiada utilidad para convertirla en un contraataque o *parry*. De esa manera habría un momento muy preciso en el que el jugador podría defenderse y a cambio podría darle una ventaja, como por ejemplo cortar el ataque del enemigo y aturdirlo.

De la misma manera se podrían añadir distintos ataques, como un ataque rápido más débil y uno fuerte pero lento como sucede en muchos otros juegos como *Dark Souls*. Muchos cangrejos tienen una pinza más grande que la otra, lo cual combinaría perfectamente con estos tipos de ataques y se podrían mapear los controles de una manera más visual. Si por ejemplo la tenaza derecha fuera la grande, el clic o gatillo derechos corresponderían al ataque fuerte realizado con esa pinza y el izquierdo al ataque rápido con la otra tenaza.

Esto se combinaría también con un aumento de complejidad de la IA para que pudiera defenderse también en ocasiones y tuviera más patrones de ataque como varios ataques rápidos seguidos, un golpe fuerte o un combo de ataques rápidos y fuertes. Cada uno de estos patrones debería estar distintamente telegrafiado[1], de manera que el jugador necesite de un proceso de observación y aprendizaje del comportamiento del cangrejo para poder anticiparse correctamente a sus ataques.

Por otro lado, al juego también le vendría muy bien más contenido en general para otorgar una mayor variedad y alargar la vida del juego. Sería necesario añadir más tipos de conchas con efectos únicos que fueran más allá de modificar su peso y resistencia, como podrían ser conchas que explotan en un área mayor, que pueden congelar a los enemigos, que aumenten el daño o la velocidad de ataque, o que modifiquen la forma de atacar del cangrejo. Dado que solo hay un tipo de enemigos sin apenas variantes entre sí también enriquecería bastante el gameplay la inclusión de nuevos tipos de cangrejos enemigos. Podrían tener comportamientos distintos entre sí, variando la forma de atacar, patrones de ataque, velocidad, y estadísticas en general. Además también sería interesante que cada uno de estos tipos tuviera un tipo concreto de concha de los que comentábamos antes.

[1] En este contexto una acción telegrafiada es aquella que tiene alguna indicación previa de que va a suceder. Es muy habitual que un enemigo avise con una animación concreta o similar antes de realizar un ataque. De esta manera el jugador puede averiguar qué es lo que hará a continuación y actuar en consecuencia

En cuanto a terreno, actualmente se cuenta con un terreno único que persiste entre partidas y solo tiene dunas y algún obstáculo, por lo que no resulta demasiado estimulante de explorar pasadas las primeras partidas. Se podrían añadir ciertos elementos que tuvieran alguna funcionalidad durante el combate, power-ups e incluso cambios de altura. El tema de jugar con un terreno con varias alturas podría ser interesante darle una nueva utilidad a la habilidad de salto y se podría aprovechar para encontrar secretos o incluso para atacar más efectivamente a ciertos enemigos desde las alturas.

Para ello también sería necesario mejorar el comportamiento del cangrejo sobre las superficies para que fuera más estable y dotarle de cierta capacidad de escalada. Para evitar que el terreno se tornase repetitivo entre partidas, se debería implementar una generación procedural del propio terreno con ciertos patrones de elementos. Incluso se podrían hacer varias zonas con características y elementos únicos al igual que sucede en la mayoría de rogue-likes.

Algo en lo que obviamente se debería mejorar sería el apartad visual del juego. Dado que este proyecto no esta enfocado a un perfil artístico, se obtendría una gran mejora de la estética si se contara con artistas profesionales que mejorarían todos los modelados, texturas, interfaz y sonido sustituyendo todos los assets que se han ido utilizando de terceros por otros nuevos que estuvieran más cohesionados entre si y estuvieran específicamente creados para este título. Como ya se ha dejado ver en apartados anteriores también se mejorarían y añadirían animaciones "tradicionales" para distintas acciones del cangrejo y mejoraría el comportamiento de la animación procedural para que se adaptara más a las superficies rotando el tronco del cangrejo.

Dadas las facilidades que ofrece Unity para hacer aplicaciones multiplataforma, sería interesante realizar versiones para otros dispositivos. Hermit King no requiere de una gran capacidad de cómputo por lo que sería quizás interesante llevar una versión para *smartphones*, sobre todo Android por contar con un menor número de trabas para su desarrollo. Con las funcionalidades del nuevo Input System que utiliza la aplicación no sería demasiado complejo crear una adaptación de los controles actuales a formato táctil, por lo que también sería algo a tener en cuenta para futuras versiones del juego.

7.4 Salida al mercado

En el caso de que se intentase lanzar una versión comercial al mercado de Hermit King habría que encargarse de múltiples cuestiones. En primer lugar, evidentemente había que pulir la idea y el diseño que seguiría el juego y se realizaría una división por tareas del proceso de desarrollo. Sin embargo, un factor vital que determinará la escala del juego sería el presupuesto con el que se contaría para realizarlo. El proyecto evidentemente sería de bajo presupuesto, por lo que se podría considerar como una producción indie. Estas producciones se suelen financiar a través de plataformas de *crowdfounding*[2] como Kickstarter, Patreon o

[2]El crowdfounding se trata de un proceso de recaudación de fondos apoyados por particulares que deciden apoyar un proyecto con pequeñas cantidades de dinero. Este tipo de financiación se puede hacer a través de

Indiegogo.

Se estimaría que el coste del proyecto rondaría entorno a los 150.000-200.000€ en base a información de otras producciones de una envergadura similar. Como ejemplo para realizar la estimación se han tomado datos de juegos de una envergadura similar desarrollados en España. El estudio Crema Games, para desarrollar su título Immortal Redneck contó con el trabajo de 10 personas durante 20 meses para la versión de PC más un poco más para el port a otras plataformas lo cual dejaría un presupuesto de 300.000 a 400.000€ (teniendo en cuenta salarios, impuestos asociados, licencias, hardware, oficinas, asesoría, etc.), mientras que para su siguiente título TemTem, de una mayor envergadura, recaudaron 573.000$ en Kickstarter. Por otro lado, Digital Sun consiguió recaudar 134.000$ en su exitosa campaña de Kickstarter para el desarrollo de Moonlighter. Por último, la empresa sevillana The Game Kitchen ha obtenido 330.000$ dólares en Kickstarter para la creación de Blasphemous.

Con esas cantidades de dinero se podría conseguir un equipo de al menos otras 3 personas más especializadas en distintas ramas del desarrollo durante un par de años, que sería suficiente para sacar el juego. Una característica habitual en este tipo de producciones es la evolución del juego durante bastante tiempo después de la salida del título, siendo frecuente aparecer en primera instancia como un *early access* [3]. Esto ayudaría a desarrollar el juego mientras se reciben ingresos de las ventas durante cierta parte del desarrollo, haciendo más accesible alcanzar esa meta de presupuesto. Por ese motivo también sería una buena práctica la publicación de un Kickstarter para recaudar presupuesto y conseguir una comunidad de seguidores del producto.

Inicialmente se podría lanzar en PC y Nintendo Switch, donde se estima que los lanzamientos de estilo indie tienen mejor acogida. En el caso de PC, sería interesante lanzarlo además de en Steam, la tienda por excelencia de juegos de PC, en Epic Games Store. Epic Games, aparte de pedir solo un 12% de las ganancias de cada juego frente al 30% que exige Steam, ofrece muchas más facilidades a los desarrolladores, como ingresos constantes durante el desarrollo del juego, mayor visibilidad en la tienda o asegurar unos ingresos mínimos de ventas aunque no se alcance cierto número de copias vendidas.

En cuanto a la consola de Nintendo también ofrece varias ventajas como una mayor visibilidad del juego en la tienda y publicidad por parte de la propia Nintendo de los juegos indies que se añaden a su eShop. Esto se suma a la posibilidad de jugar con Switch en cualquier sitio debido a su portabilidad, que la hace una opción muy atractiva para los jugadores. Como objetivo secundario también se podría lanzar posteriormente para PlayStation y Xbox, donde, si bien se tienen también ciertas ventajas en el caso de tener una exclusividad con la consola, las ventas no suelen ser tan altas. Si la escala del juego no resulta excesiva en cuanto a exigencias de hardware se podría realizar una versión adaptada para móvil con un precio

distintas plataformas y para cualquier tipo de proyecto. Habitualmente se presentan este tipo de proyectos con una serie metas que prometen los desarrolladores en función de la recaudación obtenida y con un sistema de recompensas que se le otorgarían a los participantes en el caso de alcanzar el mínimo de recaudación

[3] Un juego *early access* es aquel que sale a la venta antes de ser finalizado su desarrollo, cuando el juego se encuentra en sus versiones alfa o beta, bajo la promesa de utilizar esos ingresos para completar el juego

más reducido que no supondría un sobrecoste demasiado alto de manera que se pudiera jugar en Android e iOS.

En estos casos también es habitual llegar a un acuerdo con un *publisher* a cambio de un porcentaje de los beneficios. Los publishers se encargan del lanzar el juego en distintas plataformas, crear contenido promocional y de dar algo de difusión al producto. Pueden agilizar el proceso sobre todo en su salida en las tiendas de las consolas, que suelen poner más trabas para publicar un juego que plataformas como Steam o Google Play. Además en estos casos también pueden ayudar a conseguir kits de desarrollo[4] de cada consola, lo cual es imprescindible para testear el juego antes de su salida.

Si bien el publisher puede ser de ayuda a la hora de publicitar el título, para proyectos pequeños como es este, no suele ser suficiente. La labor del publisher varía enormemente en función del acuerdo, escala y empresa, pero en mayor o menor medida resulta casi obligatorio que el equipo haga a la vez su propia difusión. Para ello, se suelen recurrir normalmente a redes sociales en los que poder mostrar pequeños fragmentos llamativos del juego que lo den a conocer. En el sector indie es habitual ver este tipo de contenido en Twitter o Reddit en formato de gifs cortos, dado que en principio la publicación de estos mensajes es gratuita, aunque en el caso de Reddit está limitada en función de la actividad normal que tengamos en la plataforma (solo uno de cada diez posts pueden ser publicitarios).

Además de esto sería necesario contar con una página para el propio juego que mostrara información relevante e imágenes, distintas formas de contacto, y un kit de materiales promocionales para la prensa para que se pueda incluir fácilmente en sus artículos y vídeos. La página debería ser simple y concisa y debería contar con un dominio propio.

Otra herramienta que se suele utilizar son las suscripciones a newsletters[5]. Si bien se tiene el concepto de que este tipo de comunicaciones no funcionan tan efectivamente como otras, existen desarrolladores que aseguran su eficacia a la hora de crear una comunidad más fiel del juego. Se puede contratar un servicio de envío periódico económico con herramientas como Mailchimp, que agilicen en buena medida el proceso.

Por último, al igual que sucede en todas las empresas, sería necesario contratar los servicios de una gestoría y y asesoría que nos ayudaran con los trámites legales y administrativos. La facturación en ciertas plataformas y países puede conllevar trámites especiales, por lo cual es altamente recomendable contar con expertos en este sector, a poder ser especializados en el mundo del videojuego.

[4] Un kit de desarrollo es una versión de una consola modificada especialmente para ejecutar juegos en proceso de desarrollo, lo cual no sería posible con versiones comerciales de la consola. Normalmente, estos kits son entregados por los fabricantes de consolas en el caso de que los solicitantes cumplan con todos los requisitos y abonen el precio correspondiente.

[5] Una newsletter es una comunicación periódica que se realiza por correo electrónico a los usuarios que han demostrado interés en un tema en concreto. Los suscriptores irán recibiendo de vez en cuando mensajes acerca del desarrollo del proyecto y acerca de su lanzamiento, de manera que se mantenga durante más tiempo el interés por el mismo.

Bibliografía

BisonTracksMusic. (2017, 4). *Caribbean breeze.* https://www.youtube.com. Descargado 31/05/2021, de https://youtu.be/tfkw1ZvYECo

BitGem. (2019, 10). *Simple button set 01.* https://assetstore.unity.com. Descargado 31/05/2021, de https://assetstore.unity.com/packages/2d/gui/icons/simple-button-set-01-153979

BitGem. (2021, 1). *Urp stylized water shader - proto series.* https://assetstore.unity.com. Descargado 31/05/2021, de https://assetstore.unity.com/packages/vfx/shaders/urp-stylized-water-shader-proto-series-187485#description

Brown, M. (2019, 1). *Roguelikes, persistency, and progression.* https://www.youtube.com. Descargado 31/05/2021, de https://www.youtube.com/watch?v=G9FB5R4wVno

Bycer, J. (2019, 11). *The roguelike debate – roguelikes vs roguelites.* https://www.gamasutra.com. Descargado 31/05/2021, de https://www.gamasutra.com/blogs/JoshBycer/20191125/354673/The_Roguelike_Debate__Roguelikes_vs_Roguelites.php

Crytek. (2021). *Cryengine.* https://www.cryengine.com. Descargado 31/05/2021, de https://www.cryengine.com

Dafont. (2021). *Dafont.com.* https://www.dafont.com. Descargado 31/05/2021, de https://www.dafont.com

EpicGames. (2021). *Unreal engine.* https://www.unrealengine.com. Descargado 31/05/2021, de https://www.unrealengine.com

Foreman, J. (2016, 7). *Rogue-lite life lessons.* https://www.gamasutra.com. Descargado 31/05/2021, de https://www.gamasutra.com/blogs/JoshForeman/20160727/277954/RogueLite_Life_Lessons.php

GDQuest. (2020, 8). *Godot vs game maker: How do they compare?* https://www.youtube.com. Descargado 31/05/2021, de https://www.youtube.com/watch?v=3KKeFK0NHc8

Montoya, J. A. (2020, 9). *¿cómo se diseña un motor de videojuegos?* https://campus.uoc.edu. Descargado 31/05/2021, de https://campus.uoc.edu/annotation/29a976745592d06c6df731cc1d4444f5/756548/PID_00246080/PID_00246080.html#w31aab5

Moss, R. C. (2020, 3). *Ascii art + permadeath: The history of roguelike games.* https://arstechnica.com. Descargado 31/05/2021, de https://arstechnica.com/gaming/2020/03/ascii-art-permadeath-the-history-of-roguelike-games

NedMakesGames. (2021, 4). *Create a sand dune shader graph in unity urp! no textures needed! 2020.3 | game dev tutorial.* https://www.youtube.com. Descargado 31/05/2021, de https://youtu.be/KqWfo6EPjCw

RenderKnight. (2020, 9). *Fantasy skybox free.* https://assetstore.unity.com. Descargado 31/05/2021, de https://assetstore.unity.com/packages/2d/textures-materials/sky/fantasy-skybox-free-18353

Sapajou. (2020, 12). *Worry less.* https://www.youtube.com. Descargado 31/05/2021, de https://youtu.be/5acxtSJdIuM

Thinkwik. (2018, 4). *Cryengine vs unreal vs unity: Select the best game engine.* https://medium.com/@thinkwik. Descargado 31/05/2021, de https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e

UnityTechnologies. (2021). *Unity.* https://unity.com. Descargado 31/05/2021, de https://unity.com

vgvladimir. (2017, 11). *Colección de piedra modelo 3d.* https://www.turbosquid.com. Descargado 31/05/2021, de https://www.turbosquid.com/es/3d-models/3d-stone-1221007