UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
[GRADUADA/O EN INGENIERÍA DEL SOFTWARE]

Desarrollo de un videojuego de mundo abierto utilizando técnicas de generación procedimental.

Development of an open world video game using procedural generation techniques.

Realizado por
Marco António De Sousa Gonçalves

Tutorizado por
Antonio José Fernández Leiva

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2020

Fecha defensa: Septiembre 2020

Resumen

En este proyecto se ha abordado el desarrollo de un videojuego de mundo abierto utilizando Unity y C#, cuyo mapeado es un planeta generado de forma procedimental. Utilizando las herramientas proporcionadas por Unity y la función matemática de ruido de Perlin, se ha implementado un algoritmo capaz de generar planetas de cualquier radio, con una topografía realista y distintos biomas que aportan diversidad al mundo.

La jugabilidad es en primera persona, con combates cuerpo a cuerpo y recolección de alimentos para la supervivencia, pudiendo el jugador moverse por los continentes del planeta de forma libre.

Se han desarrollado las mecánicas de juego, la generación del mapeado, las interfaces de usuario y la progresión en el juego, que lleva al jugador por los distintos continentes del planeta generado. El algoritmo que se utiliza para la generación del planeta utiliza semillas aleatorias para crear planetas diferentes cada vez que se ejecuta, y permite que el jugador guarde su partida; de manera que se guardan las semillas para volver a generar el planeta de forma idéntica.

Palabras clave:

Videojuego

Unity

C#

Perlin

Abstract

In this project, an open world video game has been developed using Unity and C#, whose map is procedurally generated. Using Unity's tools and the Perlin noise function, an algorithm capable of generating planets with any radius, with a realist topography and distinct biomes that give diversity to the world has been developed.

The gameplay is on first person, with melee combats and food gathering for survival, being the player free to explore the planet's continents.

The game mechanics, the generation of the map, the user interfaces and the progression in the game, which takes the player through the different continents of the generated planet, have been developed. The algorithm used to generate the planet uses random seeds to create different planets each time it runs, and allows the player to save their game; so that the seeds are saved to regenerate the planet in an identical way.

Keywords:

Video game

Unity

C#

Perlin

Índice

1

Introducción

En este capítulo se contextualiza el proyecto. Se exponen los principales motivos, objetivos y datos generales del proyecto, como las tecnologías y metodologías utilizadas.

1.1 Motivación

La industria del videojuego es a día de hoy la más grande en el mundo del arte y el entretenimiento. La cantidad de personas que los consumen crece a cada día que pasa, así como la cantidad de juegos desarrollados, su calidad y su tamaño.

El incesante auge del medio ha propiciado un enorme crecimiento en el interés no solo por jugar a videojuegos, sino por desarrollarlos; lo cual ha dado lugar a la aparición de tecnologías y plataformas en las que pequeños equipos, e incluso individuos, son capaces de desarrollar videojuegos que hace 20 años eran impensables fuera de los grandes estudios y distribuidoras. Entre estas tecnologías destacan los motores gráficos, los cuales antes formaban parte del proceso de desarrollo de cualquier videojuego, pero que, con la aparición de motores comerciales como Unity, el motor gráfico pasa a ser la base sobre la que parte el desarrollo, permitiendo así ahorrar muchos recursos para el desarrollo en sí, así como lo hace un *framework* en el ámbito del desarrollo web.

Esta apertura al mercado de los videojuegos ha supuesto la aparición de una nueva industria: los videojuegos independientes o *indie*. Esta industria se compone de desarrolladores que no dependen de grandes distribuidoras para publicar sus juegos, con lo que tienen la capacidad de desarrollar sin imposiciones externas a su estudio; manteniendo así la libertad artística y dando paso a muchos videojuegos que han sido revolucionarios.

Sin duda el videojuego más popular y exitoso nacido de la industria de los videojuegos independientes es Minecraft, el cual revolucionó el mundo de los videojuegos y se convirtió en el videojuego más vendido de la historia, con 200 millones de copias vendidas (The Verge, 2020). Gran parte del atractivo de este juego se debe a la generación de los mundos para cada partida, para lo cual utiliza tecnologías de generación procedimental; siendo capaz de generar para cada jugador un mundo único en base a criterios pseudoaleatorios y su algoritmo de generación de contenido. Con este algoritmo el juego es capaz de generar para cada jugador un mundo con diversos ecosistemas, topografías y mazmorras; manteniendo en todo momento un grado de coherencia más que suficiente para llegar pensar que es un mundo creado a mano por los desarrolladores.

Estas son, pues, las principales motivaciones detrás de este proyecto; en el cual se ha desarrollado un videojuego basado en generación procedimental de contenido por una única persona, utilizando recursos artísticos gratuitos (como modelos 3D, animaciones, etc.).

1.2 Objetivos

En este proyecto se ha propuesto desarrollar un videojuego 3D cuyos contenidos (mapeado, elementos del entorno y enemigos) se generan de manera procedimental al ejecutar el juego.

En cuanto al mapeado se busca la generación procedimental de un planeta esférico, el cual puede tener el radio que se desee; y que contiene 6 continentes y océanos, pudiendo los continentes tener ecosistemas y topografías de diferente estilo. Cada nueva partida se generará un planeta completamente diferente a la anterior, gracias a la aleatorización de los parámetros de generación de los contenidos. El jugador

se moverá por la superficie de este planeta, pudiendo explorar la totalidad del terreno, siempre que sus movimientos lo permitan (p.ej. podría no poder subir a una montaña demasiado escarpada). Entre los ecosistemas variarán los siguientes parámetros: topografía del terreno (montañosa o planicie); disposición y cantidad de fauna y flora; y el tipo de fauna y flora que podrá encontrarse el jugador.

El jugador podrá realizar acciones variadas, tales como luchar contra enemigos, explorar el terreno y comer de las plantas que lo permitan. Entre los movimientos de los que dispone el jugador se encuentran: andar, correr, saltar, golpear y esquivar.

El juego tendrá disponible un sistema de guardado de partida, pudiendo el jugador guardar su partida en el disco duro para después cargarla, procediendo el juego a generar todo el contenido igual que la última vez al utilizar la misma semilla para la generación de números pseudoaleatorios.

El resto de información detallada sobre el diseño del videojuego se encuentra en el Documento de Diseño de Juego (Anexo I), así como en el apartado 2 de esta memoria.

También se llevará a cabo la documentación técnica mediante modelos UML de los principales módulos del videojuego, incluyendo modelos estáticos y dinámicos con los que se especificará en detalle el funcionamiento interno de los algoritmos utilizados, así como al estructura interna de los objetos que colaboran en la ejecución de estos algoritmos.

1.3 Estructura de la memoria

En adelante, la memoria estará estructurada de la siguiente manera:

- En el segundo capítulo se hablará de los antecedentes de este proyecto, haciendo hincapié en las tecnologías y herramientas utilizadas durante el desarrollo, así como la metodología utilizada y la planificación seguida.
- El tercer capítulo habla sobre el diseño del videojuego, desde la idea inicial hasta el Documento de Diseño de Juego que describe el videojuego en su totalidad. También se detallarán algunas decisiones de diseño (de forma más resumida que en el mencionado documento de diseño), así como se mencionarán los videojuegos que más han influido en el diseño de Rings.

- El cuarto capítulo constará de las fases del desarrollo, explicando en detalle las tareas llevadas a cabo en cada sprint, incluyendo las explicaciones técnicas de lo implementado y los resultados obtenidos. Se dividirá en subcapítulos, los cuales representan un sprint cada uno.
- En el quinto capítulo se expondrán las conclusiones del proyecto, incluyendo los resultados finales, el proceso de aprendizaje durante el proyecto y las posibles mejoras y líneas futuras para este trabajo.

2

Antecedentes

2.1 Estudio de las tecnologías a utilizar

Para el desarrollo del videojuego planteado se ha optado por utilizar el lenguaje de programación C# .NET (Troelsen y Japikse, 2017) así como el motor gráfico Unity en su versión 2019.3.3f1 (Okita, 2019).

La elección del motor gráfico se ha debido al conocimiento previo del mismo y su uso junto con el mencionado lenguaje de programación, además de ser el que más abundante información tiene disponible en Internet (principalmente para el desarrollo junto con C#).

Unity ofrece una gran variedad de herramientas para el desarrollo de videojuegos, entre ellas: aplicación de patrones de diseño, gran biblioteca de físicas y cálculos vectoriales, manejo de las animaciones, manejo de la iluminación, manejo de escenas, etc. Todas estas funcionalidades, entre otras, están incluidas en una interfaz de alto nivel preparada para agilizar el desarrollo lo máximo posible; ofreciendo todo lo necesario para desarrollar cualquier videojuego que pueda plantearse en el entorno de desarrollo independiente.

Entre la oferta de herramientas que ofrece Unity, destaca su implementación del patrón de diseño *Composite* (Wikipedia Composite, 2020), que permite la creación de objetos (entendidos como elementos de la escena en el videojuego) a partir de la composición de otros; los cuales pueden comunicarse entre sí para llevar a cabo las

funcionalidades que se requieran. Algunos componentes esenciales de un objeto en una escena de Unity son: el *MeshRenderer* (Unity MeshRenderer, 2019), que renderiza el objeto en pantalla; el *Collider* (Unity Collider, 2019), que define la forma geométrica con la que el objeto interacciona físicamente con los demás; o el *Rigidbody* (Unity RigidBody, 2019), que convierte al objeto en un elemento físico sujeto a gravedad y fuerzas; entre otros. Unity implementa este patrón colocando como objeto raíz al *GameObject* (Unity GameObject, 2019), que tiene la capacidad de habilitarse, deshabilitarse o destruirse junto con todos los demás objetos con los que se relaciona por composición y a los cuales delega todas las funcionalidades del juego.

En cuanto a la API de programación (Unity API, 2019.3), lo más destacable es su clase *MonoBehaviour* (Unity Monobehaviour, 2019), de la cual debe heredar cualquier clase de la que se vaya a instanciar un objeto para un GameObject. Esta clase ofrece funcionalidades esenciales como el acceso a los demás componentes del GameObject (incluido el mismo GameObject) y el uso de métodos especiales como *Awake*, *Start*, *Update*, *FixedUpdate* y LateUpdate. Estos métodos cumplen con los siguientes papeles en la ejecución del videojuego:

- Awake (Unity Awake, 2019) se invoca tras la instanciación del objeto al que está adherido el script.
- Start (Unity Start, 2019) se invoca tras la ejecución de todos los métodos Awake de los objetos MonoBehaviour de la escena y antes de la ejecución de cualquier Update.
- Update (Unity Update, 2019) se ejecuta antes de la renderización de cada fotograma para actualizar el estado del juego (tanto con cambios internos como a partir de la entrada que aporte el jugador)
- FixedUpdate (Unity FixedUpdate, 2019) funciona de forma similar a Update, pero se ejecuta con una frecuencia de tiempo fija coordinada con los cálculos del motor físico, por lo que cualquier operación que conlleve la aplicación de alguna fuerza en un objeto debe hacerse en este método.
- LateUpdate (Unity LateUpdate, 2019) funciona también de forma similar a Update, con la diferencia de que se ejecuta después de haberse cargado el

presente fotograma, siendo sus efectos visibles en el siguiente fotograma cargado.

Otras clases indispensables dentro de la API son *Input* (Unity Input, 2019), que controla cualquier entrada por parte del jugador (ratón, teclado, mando, etc.); *Rigidbody*, que permite la aplicación de fuerzas lineales o rotacionales al objeto, además de poder modificar directamente el vector velocidad del objeto de ser necesario; o *Transform* (Unity Transform, 2019), que almacena la posición, rotación y tamaño del objeto, y aplica movimiento de forma cinemática al mismo (sin la intervención de fuerzas).

Una de las clases más importantes de la API durante el desarrollo del presente proyecto ha sido *Mesh* (Unity Mesh, 2019). Esta clase representa una malla, que es un conjunto de vértices y triángulos que componen todos los objetos geométricos (figura 2.1), de forma que cada vértice está asociado a dos triángulos, o lo que es lo mismo, se forma un triángulo con cada 3 vértices de la malla uniéndolos en sentido antihorario.

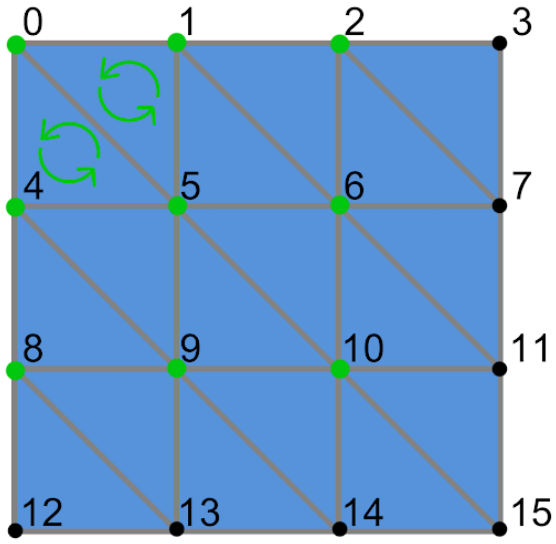


Figura 2.1: Vértices y triángulos de un mesh

Los objetos de esta clase permiten modificar en tiempo de ejecución la composición de una malla, pudiendo generar trozos del terreno mientras el jugador avanza en el mapa. Estos cálculos son considerablemente costosos, por lo que se han implementado en hebras separadas de la hebra principal de Unity, utilizando el sistema de hebras de C# (Troelsen y Japikse, 2017). De esta manera estos cálculos afectan lo mínimo posible a la carga de los fotogramas del juego.

Para la implementación del sistema de generación de contenido procedimental, se hace uso de la función matemática Mathf.PerlinNoise, que viene documentada en (Unity Perlin, 2019.3). Esta técnica fue inventada por el doctor Ken Perlin, profesor de Ciencias de la Computación en la universidad de Nueva York, mientras trabajaba produciendo los efectos especiales para la película TRON. Los fundamentos y resultados pueden hallarse en su artículo *An Image Synthesizer* (Perlin, 1985), en los que detalla el funcionamiento del hoy conocido como ruido de Perlin, referencia en la industria del cine y los videojuegos. Esta función toma 2 argumentos, que representan las coordenadas cartesianas del punto del mapa de ruido generado por la función, del cual se observa un fragmento en la figura 2.2. El resultado de la función es un valor en el intervalo [0,1] (representando en la figura 2.2 el 0 con el negro y el 1 con el blanco), el cual en el caso de este trabajo se ha utilizado para calcular la elevación del terreno en cada vértice de la malla que lo compone; simulando así la topografía de un terreno real, pudiendo utilizar parámetros externos para que el resultado sea el más adecuado para el diseño buscado. Más adelante se detalla en profundidad el funcionamiento del algoritmo (capítulo 3, apartado 3.3).

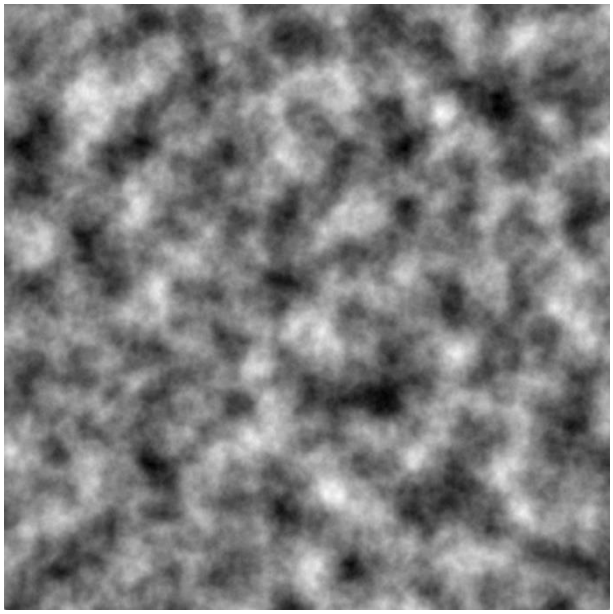


Figura 2.2: Ruido de Perlin.

2.2 Metodología de trabajo empleada

El proyecto se ha abarcado utilizando la metodología ágil SCRUM (Wikipedia Scrum, 2020), que consiste en el siguiente procedimiento: el trabajo se divide en pequeñas etapas de aproximadamente 2 semanas llamadas *sprints*. En cada sprint se desarrollan los requisitos que se propongan para el mismo en conjunto con el cliente, y al finalizar se hace una reunión con el cliente para analizar los resultados, valorarlos, o proponer cambios en lo implementado o incluso en los requisitos aún por implementar. De esta manera, se consigue una flexibilidad y agilidad muy superiores a los de la tradicional metodología en cascada; lo cual es un elemento crítico para el desarrollo, dada la continua evolución del diseño del videojuego durante el proyecto; debida a la creación de la idea al inicio del proyecto de forma original y el carácter especialmente subjetivo y artístico que conlleva el desarrollo de un videojuego.

A causa de la situación de alarma sanitaria global, las reuniones se han sustituido por la comunicación por correo electrónico, en la que se le enviaba al tutor (en el rol de cliente) los avances realizados sobre los entregables propuestos, y se revisaban para su posterior ampliación o corrección.

Dada la envergadura del proyecto, se han ejecutado un total de 10 sprints de entre 2 y 3 semanas en los que se ha ido continuamente puliendo el diseño del videojuego, ampliando la documentación y agregando, eliminando o modificando requisitos según el avance del proyecto y las revisiones hechas. En el capítulo 4 se abarcará en detalle la estructura de estos sprints, así como los avances hechos en cada uno de ellos.

2.2.1 Planificación

Para la planificación, como ya se ha mencionado, se ha dividido el trabajo en sprints de entre 2 y 3 semanas, dividiendo a su vez cada sprint en una serie de tareas. Las tareas se dividieron en las siguientes clases: documentación, diseño y desarrollo; que en el diagrama se representaban con los colores verde, azul y morado, respectivamente.

Dependiendo de la duración del sprint, de la envergadura de las tareas y de la disponibilidad horaria, los sprints se han dividido en entre 3 y 10 tareas; siendo algunas dependientes de otras o simultáneas en el tiempo. Cada tarea contenía la siguiente

información: fecha de inicio, fecha de fin, tipo de tarea, título, descripción y progreso. El progreso se actualizaba al final de cada sesión de trabajo.

La herramienta utilizada para la planificación del proyecto ha sido Gantt Project, dada su simpleza, su efectividad y su gratuidad. En la figura 2.2 pueden verse las tareas de los sprints 6 y 7 contiguamente.

Con este proyecto se adjunta, separado de la memoria, el archivo .gan con toda la planificación del proyecto para su consulta.

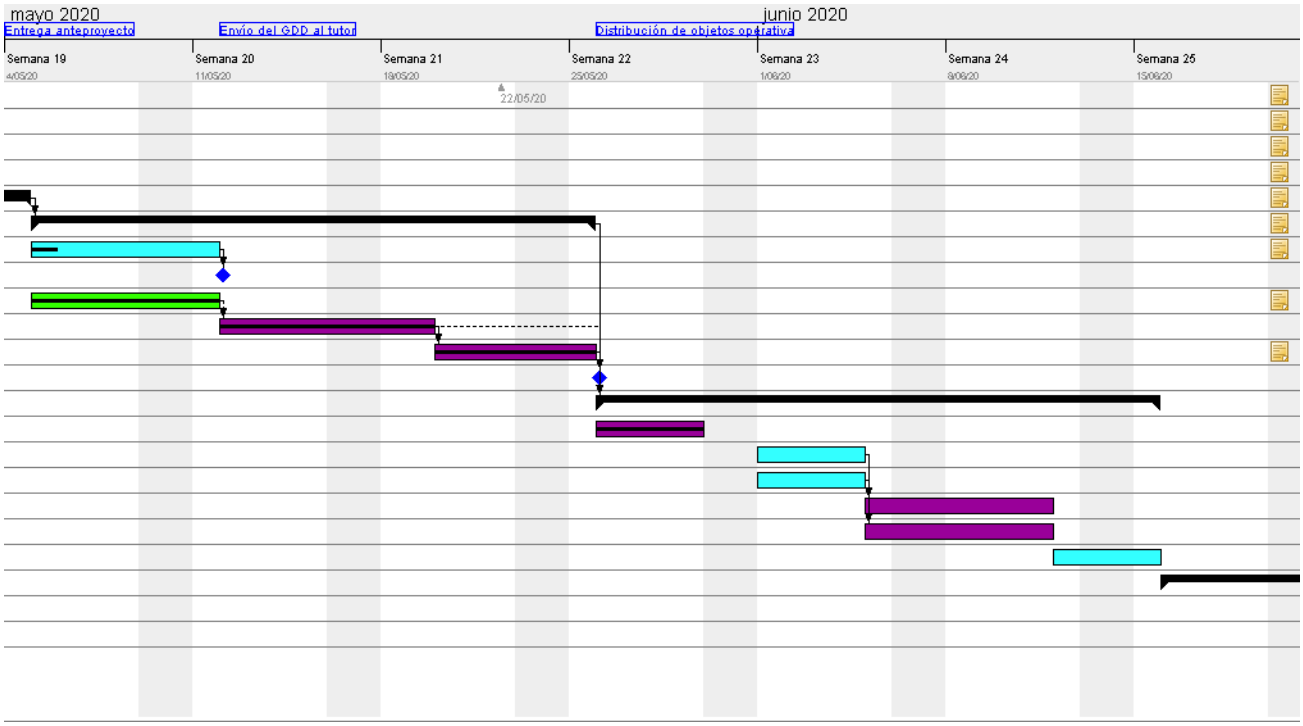


Figura 2.2: Sprints 6 y 7 vistos en Gantt Project

2.3 Videojuegos de referencia

El diseño de Rings, como cualquier otro videojuego, toma prestados muchos conceptos e ideas de videojuegos anteriores, a los que se suele denominar sus referentes. Los referentes son extremadamente importantes en el proceso de diseño de un videojuego, ya que de ellos se extrae tanto las buenas decisiones como las que no lo fueron tanto, para así dirigir el diseño del videojuego que el diseñador tiene en mente hacia una experiencia jugable lo mejor posible; sirviendo a su vez posiblemente de referente para futuros videojuegos y haciendo evolucionar la industria del videojuego en la dirección que los jugadores demanden.

Entre estos referentes, los siguientes son los más destacables:

- Minecraft. Por su orientación a la libertad del jugador y su mundo abierto prácticamente infinito generado por procedimientos y dividido en biomas.

Figura 2.3: Aldea y bioma de bosque en Minecraft

- The Legend of Zelda: A Link to the Past y The Legend of Zelda: Breath of the Wild. Por sus narrativas centradas en el protagonismo del jugador y la inmersión, así como su enfoque a la exploración.

Figura 2.4: Paisaje inicial en The Legend of Zelda: Breath of the Wild

- Bloodborne. A pesar de su estética opuesta a la de los dos anteriores y al mismo Rings, de este juego se toma su sistema de combate y parte de su gestión de los puntos de vida y energía.

Figura 2.5: Batalla contra un jefe en Bloodborne

- Shadow of the Collosus. Por el desarrollo de su historia, basado en buscar y derrotar a una serie de enemigos repartidos por el mundo.

Figura 2.6: Batalla contra jefe en Shadow of the Colossus

3

Diseño

En este capítulo se aborda el diseño del videojuego, desde sus fases iniciales hasta el Documento de Diseño de Juego, el cual describe el videojuego a desarrollar en su totalidad.

3.1 Diseño conceptual

La primera fase del diseño del videojuego fue el diseño conceptual. El diseño conceptual es el nivel más abstracto y se suele realizar de forma previa a redactar un Documento de Diseño de Juego (o GDD, por sus siglas en inglés). Esta fase del diseño se realizó en los inicios del proyecto, y se definieron varios pilares clave que luego se concretarían en el mencionado GDD:

- **Generación procedimental del mundo.** La primera decisión tomada respecto al videojuego fue que su mapeado estaría generado de forma procedimental y que sería único para cada partida y jugador.
- **Mundo abierto.** Se decidió que el mundo sería explorable de forma libre y sin limitaciones, pudiendo el jugador hacer lo que le apeteciera en cada momento.
- **Mundo con forma de planeta.** Se decidió que el mapeado tendría forma de planeta, es decir, que el jugador jugaría sobre un planeta esférico. De esta manera se evitaba poner límites al mapa a la vez que se evitaba que el mapa fuera infinito como en el caso de Minecraft.
- **Supervivencia.** Se decidió que habría factores de supervivencia en el juego, que aún no estaban decididos concretamente. Estos factores serían diferentes

mecanismos por los que el jugador tendría que velar activamente por su supervivencia en todas las fases del juego.

- **Cámara en primera persona.** Se decidió que la vista del juego sería en primera persona a modo de mejorar la inmersión, así como para aumentar la sensación de pequeñez respecto del mundo.
- **Combate cuerpo a cuerpo.** Se decidió que el combate sería cuerpo a cuerpo y que trataría de ser lo más dinámico posible, evitando las dinámicas de combate vistas habitualmente en juegos de primera persona, usualmente aburridas.
- **Enemigos principales.** Por último, se decidió que en cada continente habría un enemigo final que sería el principal objetivo de ese nivel, y que la tarea del jugador era buscarlo y derrotarlo.

Para más detalle respecto al diseño conceptual, véase el Anexo II.

3.2 Documento de Diseño de Juego (GDD)

En el Documento de Diseño de Juego (César León, 2019) se concretan, modifican y añaden características respecto a lo descrito en el Diseño Conceptual del videojuego. Este documento debe contener toda la información necesaria para el desarrollo del producto, teniendo en mente la posibilidad de que otras personas trabajen en el proyecto, por lo que el nivel de detalle debe ser suficiente para permitirles desarrollar con fluidez. También es importante seguir un enfoque esquemático, que garantice que todos los aspectos del videojuego están descritos en el documento, ya que el documento debe ser la representación textual del videojuego que se pretende desarrollar.

A continuación se enumeran resumidamente las que se han considerado como las partes más fundamentales del GDD redactado, disponible en el Anexo I.

3.2.1 Fundamentos del diseño

Se hace un resumen de lo ya descrito en el Diseño Conceptual, añadiendo además ciertos detalles; como juegos de referencia o el origen y estilo de los assets utilizados.

3.2.2 Género y público objetivo

A pesar de ser 2 capítulos distintos en el GDD, ambos guardan una importante relación, ya que el género de un videojuego determina en enorme medida su público objetivo.

Los géneros en los que puede englobarse la idea de videojuego descrita son: aventura, acción y supervivencia. Dado el enfoque hacia el mundo abierto y la libertad del jugador, está altamente enfocado a jugadores que disfrutan de tomarse su tiempo explorando y disfrutando de la atmósfera de un videojuego. Se recomienda una edad superior a 7 años, por la relativa dificultad de los controles y la necesidad de un pensamiento resolutivo para avanzar en el juego.

En general, se recomienda para cualquier amante de juegos como The Legend of Zelda: A Link to the Past o Minecraft, al ser ejemplos muy populares de juegos de mundo abierto.

3.2.3 Jugabilidad

Se empieza haciendo un resumen de las principales mecánicas del juego, las cuales son las siguientes: exploración, recolección, combate cuerpo a cuerpo y supervivencia; para seguidamente exponer el bucle de juego (figura 3.1), es decir, la secuencia de acciones que hará el jugador en su paso por el juego. Se refiere a esta como bucle porque, habitualmente, durante el transcurso de un videojuego el jugador deberá repetir acciones de una forma similar a cómo las hizo en el pasado. Esto es importante ya que, de no ser así, el jugador no tendría capacidad de aprender y mejorar en el videojuego, lo cual puede llegar a ser frustrante.

Se pasa a hacer una reflexión sobre qué emociones o pensamientos pretende el videojuego causar en el jugador. En el caso de este juego el foco de atención se pone sobre la inmersión, la tranquilidad, la curiosidad y la sensación de odisea alrededor del mundo. Este apartado es importante, ya que les da a los desarrolladores una dirección subjetiva a la que deben orientar lo que están desarrollando, más allá de los aspectos más rígidos u objetivos del videojuego, al final lo primordial no deja de ser qué experiencia se le aporta al jugador.

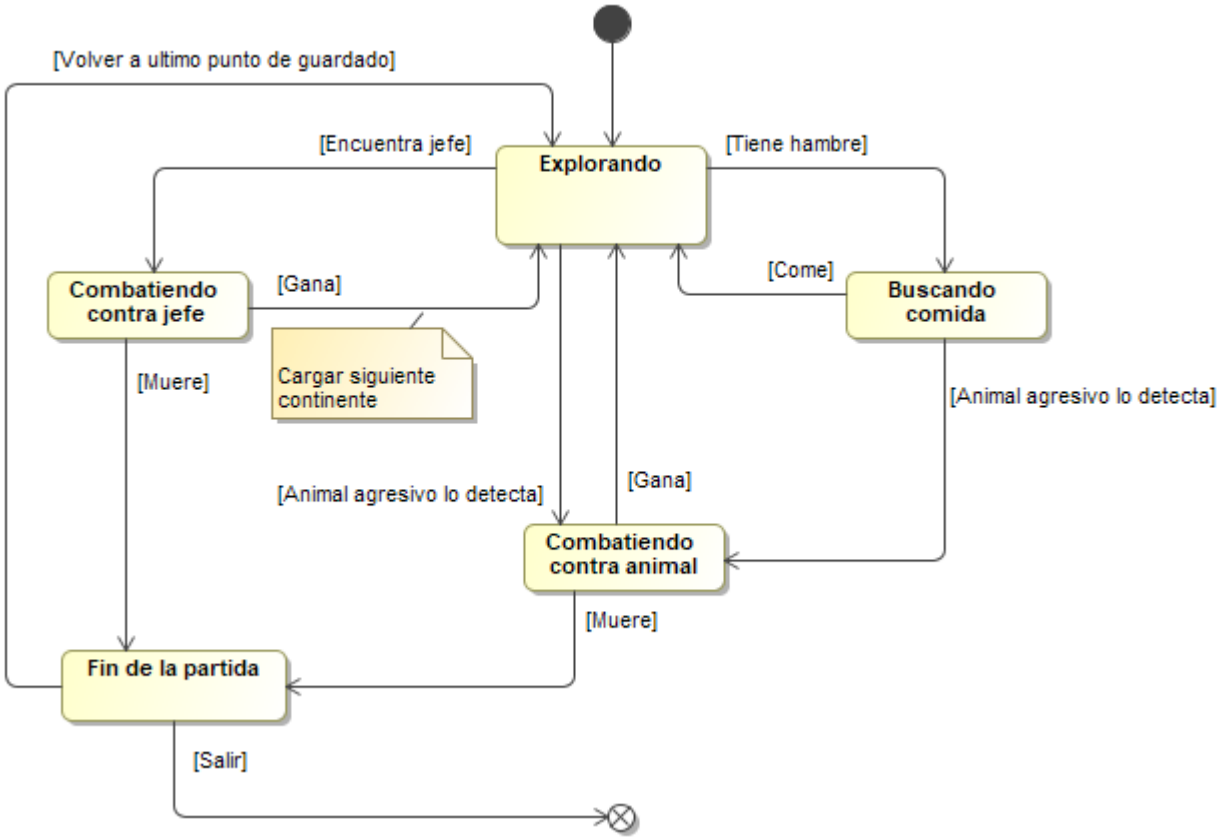


Figura 3.1: Diagrama del bucle de juego

A continuación se enumeran todas las mecánicas disponibles para el jugador, desde correr hasta golpear a sus enemigos o recolectar comida. De nuevo, para no sobrecargar este apartado, en el GDD (Anexo I) se halla una explicación en detalle.

Por último dentro del apartado de jugabilidad, se ha incluido el diseño de niveles. En muchos casos se incluye este apartado de forma independiente, pero al tratarse de un juego basado en la generación procedimental en lugar de ser diseñado a mano, se ha incluido aquí ya que los contenidos de este apartado no ameritaban un capítulo completo del GDD. En este último apartado se describen los 3 tipos de biomas disponibles en el juego: boscoso, nevado y desértico. Un continente será de uno de estos 3 tipos de bioma, además de poder tener 2 tipos de terreno: montañoso o plano. De forma aleatoria, un continente tendrá 1 tipo de terreno combinado con cualquiera de los 3 tipos de biomas. Además, cada bioma tiene su tipo de fauna y flora correspondientes.

En este último apartado dedicado al diseño de niveles, también se incluyen los tipos de enemigos, con sus modelos 3D y sus comportamientos explicados textual y gráficamente.

3.2.4 Interfaces de usuario

En este apartado se define qué interfaces estarán en el juego, qué forma tendrán y cómo estarán conectadas. Las interfaces a desarrollar son las siguientes:

- Menú inicial: El menú que aparecerá al iniciar el juego, y desde el que se podrá cargar una partida o iniciar una nueva.
- Menú de pausa: Este menú contará con 3 botones orientados verticalmente y en el centro de la pantalla que permitirán retomar el juego, volver al menú y guardar la partida.
- Interfaz in-game: Esta interfaz es la que se muestra en todo momento mientras el jugador está jugando y fuera del menú de pausa. Contará con los siguientes elementos: barra de salud, barra de energía, barra de oxígeno y la barra de hambre. Además, se incluye el brillo de la espada del jugador como interfaz integrada en el juego, ya que sustituye la función que tendría una brújula en la interfaz, indicando al jugador hacia dónde debe dirigirse para progresar en el juego.

4

Desarrollo

En este capítulo se abordará en su totalidad el desarrollo del proyecto, entrando en detalle en cada sprint realizado, así como en los resultados obtenidos al cerrar cada uno de estos. Se incluye tanto desarrollo de software como cualquier tarea involucrada en el avance del proyecto.

4.1 Idea inicial (03/02/2020 - 14/02/2020)

Este sprint marcó el inicio del proyecto, y en él se desarrolló la idea inicial, la cual se le presentó al tutor el día 14. Debido a la coincidencia del sprint con los exámenes del primer cuatrimestre, en este solo se realizó un pequeño documento con el diseño conceptual del juego, exponiendo en líneas generales la dirección deseada para el proyecto (mundo abierto, generación procedimental, 3D, etc.).

Resultados: la idea general fue aprobada por el tutor, con ciertos apuntes y sugerencias para la próxima reunión, en la que se acordó la entrega de un documento de diseño conceptual o *high concept* versión 1.0 para concretar ciertos puntos de la idea y desarrollarla hasta tener un buen punto de partida para el diseño.

4.2 Prototipo (17/02/2020 - 05/03/2020)

En este sprint se continuó con la redacción del documento high concept, con la intención de aclarar ciertos puntos inciertos en el diseño, así como para abarcar los campos restantes, como pueden ser la cámara en primera persona, el combate cuerpo a cuerpo, la inclusión de biomas en el planeta, etc. También se empezó a investigar sobre técnicas

de generación procedimental, y se decidió que la tecnología a utilizar sería el ruido de Perlin. Se desarrolló un pequeño prototipo de generador de planetas, con el que se generaba el planeta completo, sin texturas y formado por 6 mallas. Para ello se hizo un uso extensivo de la clase Vector3 de Unity, que representa vectores en un espacio vectorial tridimensional, además de la clase Mesh (Unity Mesh, 2019).

Para generar cada malla se seguía un procedimiento como el siguiente:

```
Vector3 meshNormal = direccionDelMeshDesdeElCentro
Vector3 meshX = Perpendicular(meshNormal)
Vector3 meshY = Perpendicular(meshNormal, meshX)
Vector3[] vertices = new Vector3[resolucionMesh^2]
int[] triangulos = new int[resolucionMesh^2  * 6]
int indiceVertice = 0
for i in 0..resolucionMesh
        for j in 0..resolucionMesh
                posicionVertice = meshNormal + i / resolucionMesh * meshX
                                                + j / resolucioMesh * meshY
                vertices[indiceVertice] = posicionVertice.normalized*(radio+ ruido[i,j])
                calcularTriangulosAsociados()
mesh.setVertices(vertices)
mesh.setTriangles(triangulos)
```

Resultados: documento de high concept versión 1.0, en el que se detallaban aspectos del juego como la jugabilidad, el estilo visual, la elección del planeta como forma de mapeado y los objetivos del juego. Buena parte de este diseño inicial fue modificado durante el proyecto. También se pudo probar el prototipo de generador de planetas, el cual serviría como base para el generador real.

4.3 Generación del planeta (06/03/2020 - 23/03/2020)

Empieza el desarrollo del videojuego a partir del prototipo del segundo sprint. El prototipo se amplía para poder generar planetas con el radio que se desee, implementando el algoritmo para la generación de ruido representado en la figura 4.3. Este algoritmo compone varias capas de ruido de Perlin (Perlin, 1985), para conseguir una apariencia más realista del terreno, a cada capa nueva se le llama octava. Para

conseguir un efecto de mayor cantidad de pequeños montículos, lo que se hace es disminuir la fuerza con la que aporta cada octava al ruido final, pero aumentar la frecuencia del ruido a cada octava, aumentando así el número de elevaciones en la malla pero haciendo que las elevaciones de últimas octavas sean mucho más débiles que las de las primeras. Además, se genera a la vez un mapa de ruido que incrementa su valor mientras se acerca a los bordes del continente, cuyo valor se resta al valor de ruido de Perlin procesado de ese punto. Este mapa de ruido se consigue con la siguiente función:

$$k = max(indiceXDelChunk, indiceYDelChunk)$$

$$falloff(k) = \frac{k^3}{k^3 + (2 - 2 * k)^3}$$

Consiguiendo un mapa de ruido para cada continente como el de la figura 4.2 (blanco = 0, negro = 1).

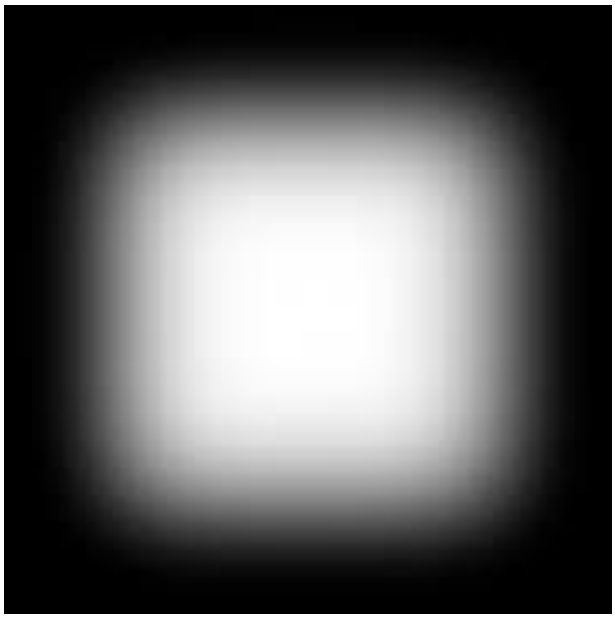


Figura 4.2: Mapa falloff.

Se añaden biomas, tipos de terreno, océano, y se desarrolla una versión inicial de los controles del personaje para poder investigar el terreno generado, junto con los componentes necesarios para simular la gravedad en el planeta: AttractedBody y Gravity. El funcionamiento de la gravedad, implementado en el método Attract de la clase Gravity, corresponde al siguiente fragmento de código:

```
public void Attract(Transform body)
{
Vector3 gravityDirection = (body.position - transform.position).normalized;
Vector3 bodyUp = body.up;

body.GetComponent<Rigidbody>().AddForce(gravityDirection * gravityMagnitude,
ForceMode.Acceleration);

Quaternion desiredRotation = Quaternion.FromToRotation(bodyUp,
gravityDirection) * body.rotation;
body.rotation = Quaternion.Slerp(body.rotation, desiredRotation, 50f *
Time.deltaTime);
}
```

Para simular la gravedad, además de aplicar una fuerza al cuerpo atraído en dirección al centro del planeta, se fuerza su rotación para que el sentido de su eje Y local coincida con el radio del planeta en ese punto. De esta manera, el jugador siempre está de pie.

En el lado del cuerpo atraído, se congela su rotación para que el motor físico no interfiera en ella, dejando el control a la gravedad del planeta. También se desactiva la gravedad por defecto del motor físico. Para mantener la gravedad del planeta actuando, en cada ejecución del método Update del cuerpo atraído se llama a Attract(transform), siendo transform el componente Transform del GameObject siendo atraído.

Cada continente generado sigue siendo único, simplemente con estos parámetros lo que se consigue es una coherencia que se mantiene entre los terrenos y biomas del mismo tipo.

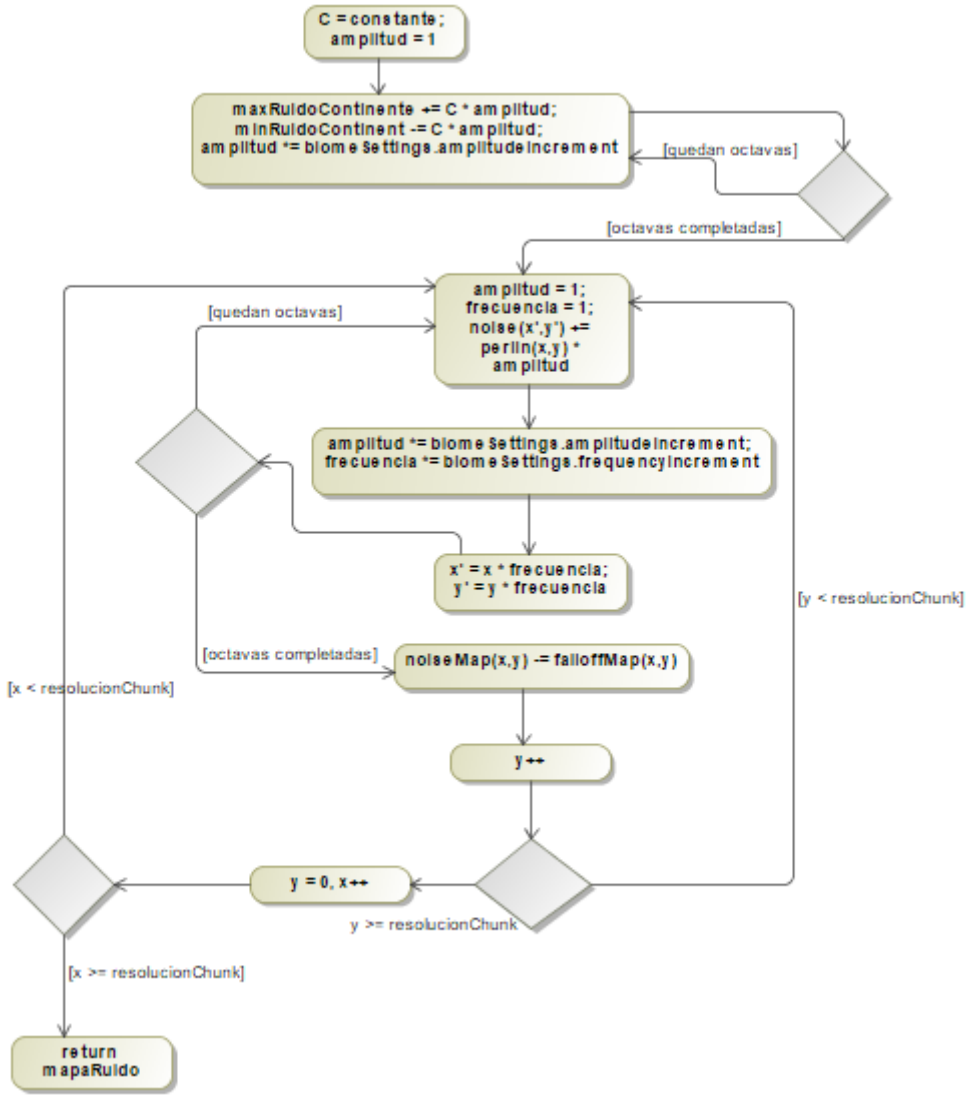


Figura 4.3: Algoritmo para la generación de mapas de ruido de Perlin con una resolución dada.

Se añade un generador de texturas basado en el ruido de Perlin, que utiliza el valor obtenido del algoritmo de generación del terreno para decidir qué color utilizar para el suelo según la altura de ese punto en el terreno. Este algoritmo recorre los vértices comprobando su valor de ruido asociado, consulta en los ajustes del bioma del continente a qué región de color pertenece y le asigna ese color al vértice.

Se sigue ampliando el documento high concept, para empezar a convertirlo en un documento de diseño completo, con todos los detalles para el desarrollo.

Resultados: documento high concept versión 2.0, generador de planetas capaz de generar planetas de diverso tamaño, con diferentes biomas y diferentes topografías.

4.4 Troceado del mapa y concurrencia (24/03/2020 - 13/04/2020)

Se implementa el sistema de hebras para la generación del terreno en tiempo real. Para ello se divide cada malla (que antes representaba un continente entero), en un número par de mallas más pequeñas, con el fin de poder cargarlas en memoria o borrarlas según se necesite, dependiendo de la localización del jugador en el mapa. Se implementa una estructura como la de la figura 4.4. En el diagrama de clases se observa la relación de agregación entre el planeta y el océano con sus MeshBuilders, que son los objetos encargados del cálculo y generación de los trozos del mapa en tiempo de ejecución. Estos tienen como principales atributos los 3 vectores de dirección (el normal al chunk, el horizontal y el vertical), con los cuales se calculan las posiciones de cada vértice de la malla a generar. Además tienen también los objetos terrainSettings y biomeSettings, que contienen información sobre la forma del terreno y los detalles del bioma del continente del que forma parte el chunk.

La clase BiomeSettings se relaciona con la clase Region, que determina las regiones que tiene cada bioma, las cuales son, básicamente, pares de <altura,color> para decidir qué color tiene el suelo del terreno dependiendo del bioma y de la altura a la que se encuentre.

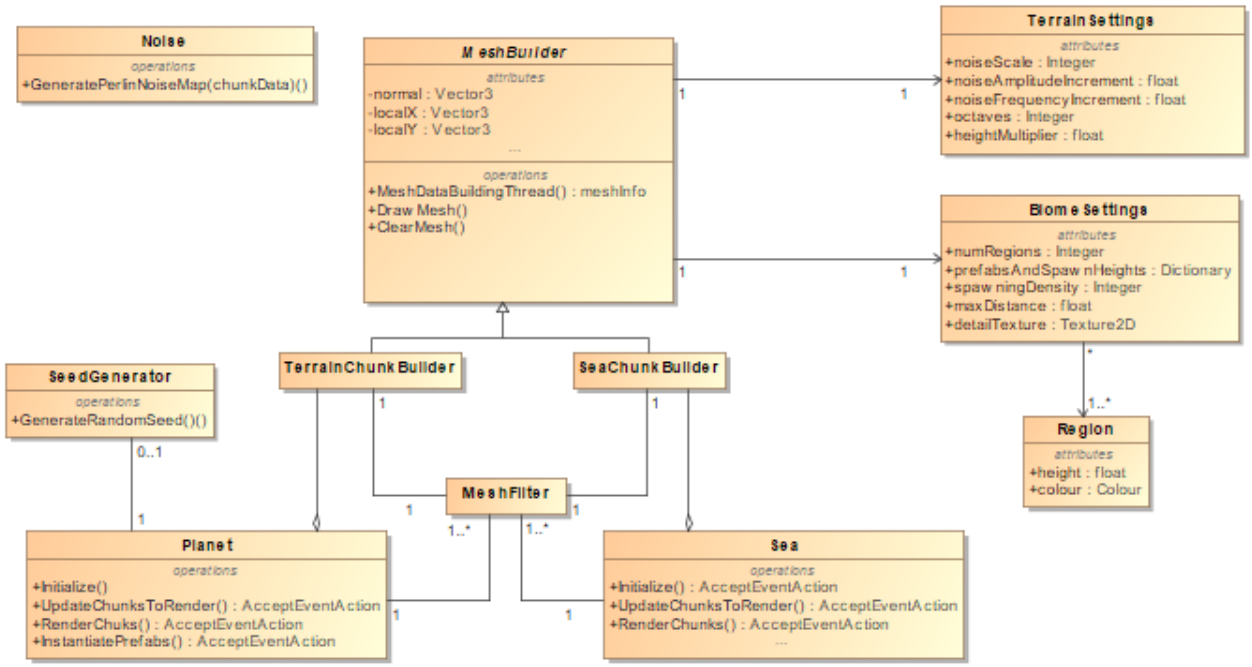


Figura 4.4: Diagrama con las clases responsables de generar el planeta.

El método MeshDataBuildingThread implementa el algoritmo representado en la figura 4.5, que es ya la versión definitiva del algoritmo para generar el terreno, salvando ciertas puntualizaciones posteriores. En este algoritmo se itera sobre la resolución definida para cada chunk, colocando cada vértice teniendo en cuenta la posición que debe tener respecto al (0,0) del chunk, además de los offset X e Y que se le suman dependiendo de en qué parte del continente se halle el chunk. Los métodos DrawMesh y ClearMesh instancian la malla calculada o la destruyen, respectivamente. Los métodos UpdateChunksToRender, RenderChunks e InstantiatePrefabs implementan los algoritmos para detectar qué chunks deben cargarse o destruirse, cargar los chunks y cargar los objetos sobre esos chunks; respectivamente.

Los métodos mencionados en el párrafo anterior tienen el siguiente funcionamiento interno:

UpdateChunksToRender

Este método, que se ejecuta como una corrutina de Unity (Unity Corrutinas, 2019), espera 2 segundos cada vez que se ejecuta, antes de volver a ejecutarse. Su funcionamiento se basa en calcular la distancia del jugador a todos los MeshBuilders que componen el planeta. Si un MeshBuilder se encuentra a una distancia menor que la distancia de visión, se abre una hebra con el método MeshDataBuildingThread de ese chunk para construir su terreno. Terminado este proceso, el chunk se agrega a la lista de chunks por renderizar que se almacena en el objeto Planet, para posteriormente ser instanciado en la escena antes de que el jugador llegue a ver esa zona del mapa, dando así la sensación de que siempre ha estado ahí.

Al calcular las distancias a cada chunk, si se detectan chunks activos en ese momento y que sin embargo se encuentran a una distancia superior a la distancia definida para su eliminación, se destruyen tanto el chunk como todos los objetos asociados a él, liberando ese espacio en memoria para los chunks más cercanos al jugador.

Para evitar posibles congelaciones de la pantalla durante la ejecución del método, se limitan los chunks que se analizan por cada fotograma, dividiéndose así la carga y los cálculos de distancia entre varios fotogramas.

RenderChunks

En este método se itera sobre la cola de chunks por renderizar, la cual es actualizada por el método anteriormente descrito. Para cada MeshBuilder que se halla en esa cola, se invoca a su método DrawMesh, que como ya se ha explicado tiene la función de instanciar la malla anteriormente calculada. Tras instanciar el trozo de terreno correspondiente al MeshBuilder, se abre una hebra con el método PositionPrefabsByPerlinHeightMap de la clase PrefabPositioner, el cual calcula los prefabs que deberán instanciarse sobre el trozo de terreno calculado.

Para terminar, se añade este chunk a la lista de chunks pendientes de instanciar sus prefabs, también localizada en el objeto Planet.

InstantiatePrefabs

Esta corrutina, que también se ejecuta cada 2 segundos, se encarga de recorrer la lista de chunks con prefabs por instanciar, para luego iterar sobre las parejas <prefab, posición> de cada chunk e instanciar los objetos en su posición correspondiente. La rotación del objeto se calcula de manera que su eje Y sea perpendicular a la superficie del planeta, y se le da una rotación sobre su eje Y aleatoria, para que no estén todos los objetos orientados hacia la misma dirección, dándoles un aspecto más realista y aleatorio.

Además, si inició la redacción del Documento de Diseño de Juego (Anexo I), utilizando como base el diseño conceptual (Anexo II) desarrollado hasta el anterior sprint.

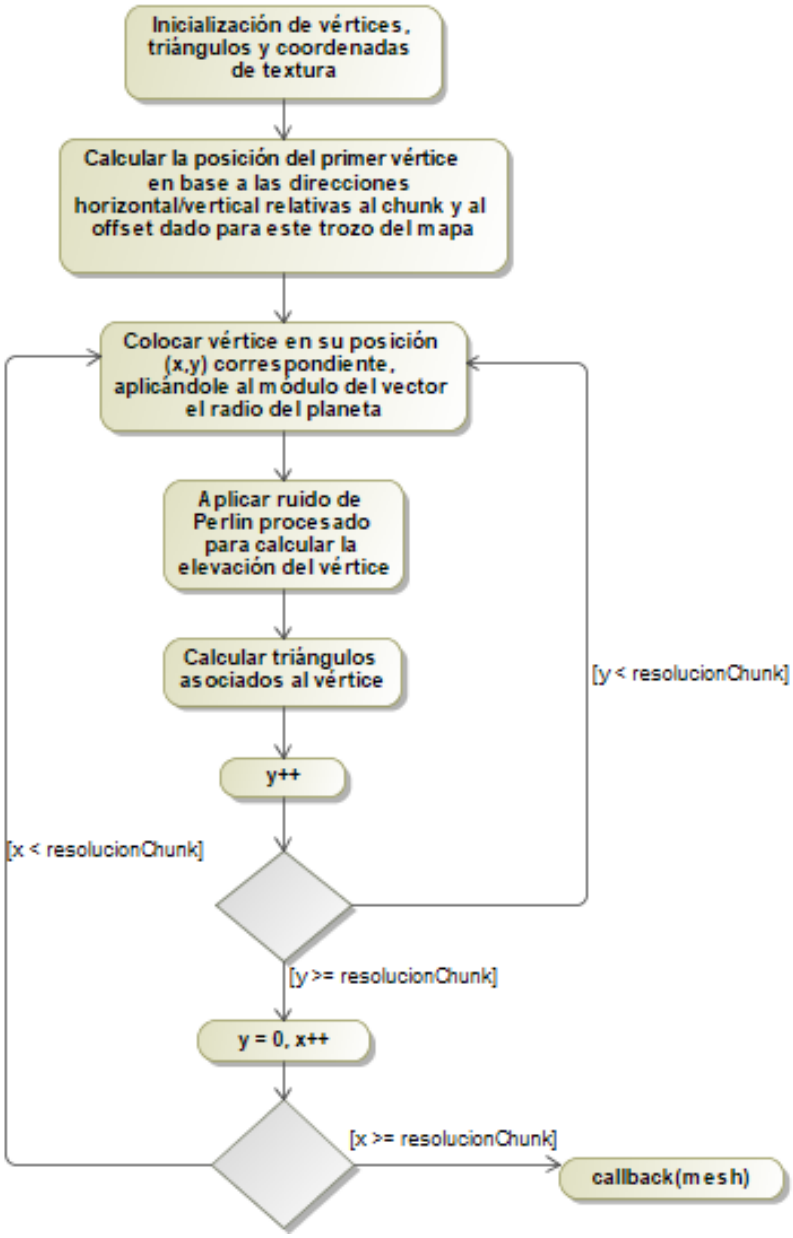


Figura 4.5: Funcionamiento de MeshDataBuildingThread

Resultados: algoritmo funcionando en tiempo real, habiendo mejorado la eficiencia enormemente y permitiendo la subida de la resolución del mapeado, al no necesitar tenerlo cargado en su totalidad. Versión inicial del Documento de Diseño de Juego.

4.5 Anteproyecto (14/04/2020 - 04/05/2020)

La primera tarea en este sprint se trató de optimizar y refactorizar ciertas porciones del código desarrollado en el sprint anterior. Se resolvieron errores de conflictos entre las hebras, así como se reordenó el código para hacerlo más flexible, reutilizable y legible.

Tras esto se empezó la redacción del anteproyecto, el cual se envió para su corrección el día 23 de abril, se terminó de redactar el día 4 de mayo y se entregó. Durante la redacción también se implementó la iluminación del juego, la cual funcionaba en tiempo real, con un sol que rotaba alrededor del planeta (posteriormente, en el séptimo sprint, se abandonó el ciclo de día/noche por complicaciones técnicas). También se implementó un sistema de controles del personaje mejorado, que es el utilizado para el juego final, salvando el sistema de combate que es implementado en el octavo sprint.

Resultados: anteproyecto entregado, algoritmo de generación procedimental optimizado, sistema de controles mejorado, iluminación con ciclo día/noche (incompleto).

4.6 Posicionamiento de prefabs (05/05/2020 - 25/05/2020)

En este sprint se empezó por aumentar el detalle del documento high concept, con la intención de convertirlo en un documento de diseño de juego, en el que el nivel de detalle fuera suficiente como para desarrollar el videojuego siguiéndolo como guía.

Posteriormente, y con la generación del terreno completada, se procedió a implementar el algoritmo que posiciona los objetos en el mapeado (figura 4.7). Estos elementos son, por ejemplo: árboles, rocas, animales, enemigos, plantas, etc. Para ello también se utilizó en parte el mapa de ruido de cada trozo del terreno, definiendo para cada tipo de objeto la altitud a la que es más probable hallarlo, para luego decidir en un vértice concreto qué objeto colocar. El resultado se ve en la figura 4.6, en la que se muestra el bioma nevado como ejemplo.

Figura 4.6: Bioma nevado con objetos

Junto al desarrollo del algoritmo, se configuraron los recursos que se utilizarían dentro de Unity, incluyéndolos en el proyecto y desarrollando otro sistema que los carga en memoria de forma previa a su carga en el mapa.

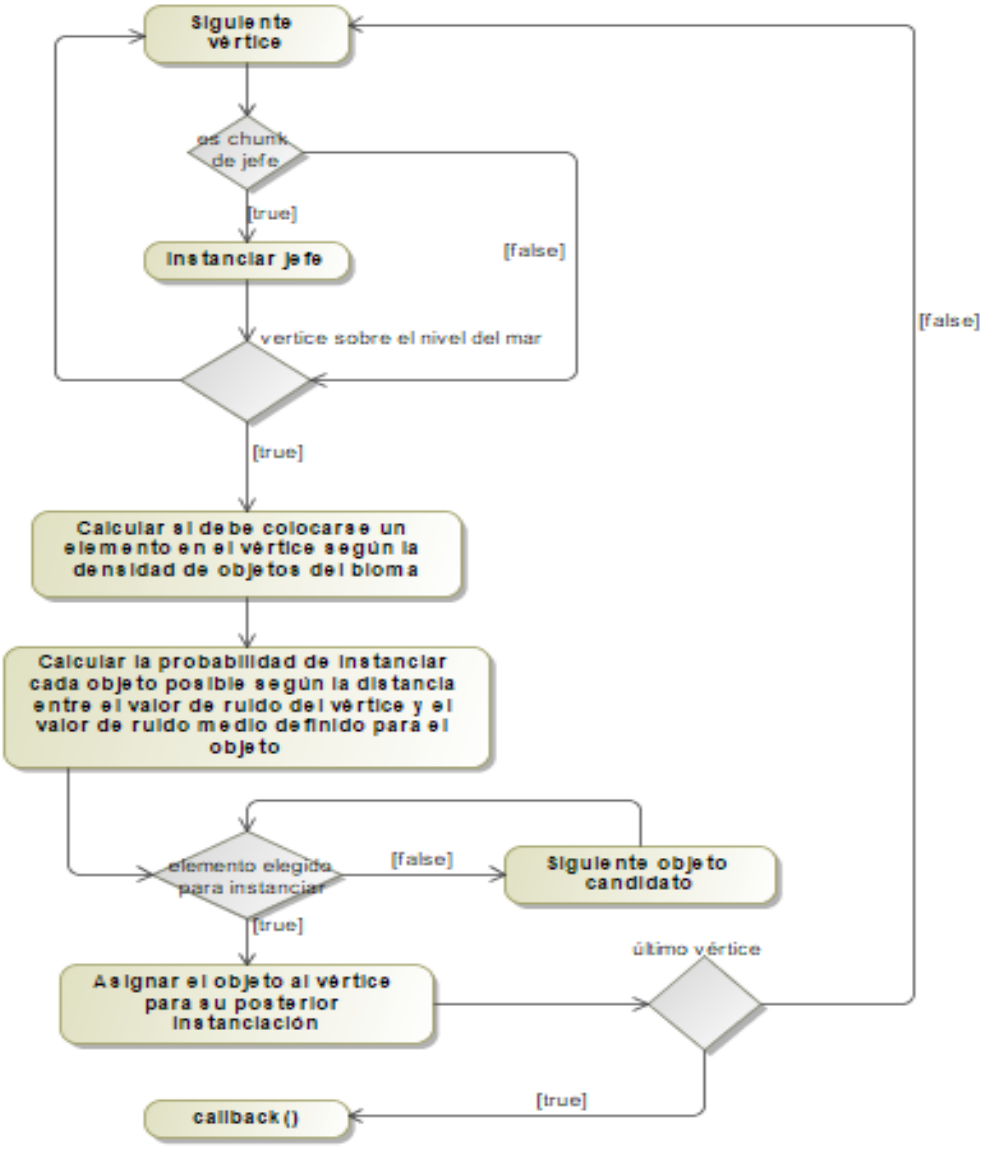


Figura 4.7: Algoritmo de posicionamiento de objetos a nivel conceptual

Resultados: algoritmo de posicionamiento de objetos implementado, sistema de cargado de recursos en memoria implementado, documento de diseño de juego versión 1.0.

4.7 IA (26/05/2020 - 15/06/2020)

La primera tarea en este sprint fue la eliminación del ciclo día/noche. Era una parte interesante a implementar, pero por falta de tiempo dadas las alturas del proyecto y las dificultades técnicas para implementarlo junto a la inexistencia de información sobre el tema, se tuvo que abandonar este sistema; ya que se llegó a la conclusión de que para lograr el efecto atmosférico deseado no había herramientas para simularlo, había que implementar una atmósfera basada en principios físicos. Esto no era viable como parte del proyecto, ameritando un proyecto entero.

Para sustituir a este sistema se implementó una skybox de color plano, y se hizo que la luz del sol estuviera siempre sobre el jugador; con lo cual siempre es de día.

Posteriormente se empezó el diseño de los enemigos del juego a nivel de comportamiento. Se diseñaron 3 tipos de PNJ (personajes no jugadores): animales pacíficos, animales agresivos y enemigos finales. El comportamiento de cada uno viene expuesto en las 4.8 y 4.9, siendo el comportamiento de los jefes igual al de los animales agresivos, cambiando sin embargo sus ataques, que son a distancia, y sus mejores atributos.

También se implementaron estos diseños, junto con sus animaciones y sus modelos ya integrados en el juego; para lo cual se utilizó el componente Animator de Unity, que permite activar las animaciones de los modelos mediante el acceso a sus métodos desde el script de control del jugador, como puede verse en el siguiente fragmento de código:

```
void Wander()
{
anim.speed = 1.3f;
anim.SetInteger("Walk", 1);
rb.MovePosition(transform.position + transform.forward * movementSpeed *
Time.fixedDeltaTime);

CheckForTarget();
}
```

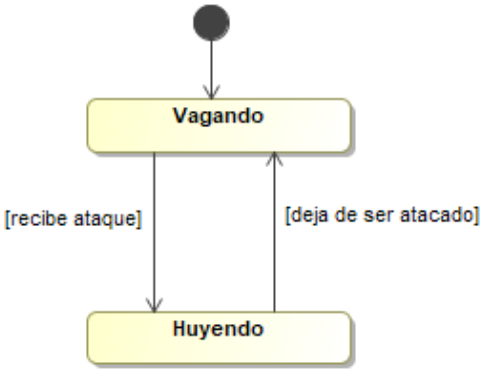


Figura 4.8: Diagrama de estados animal pacífico

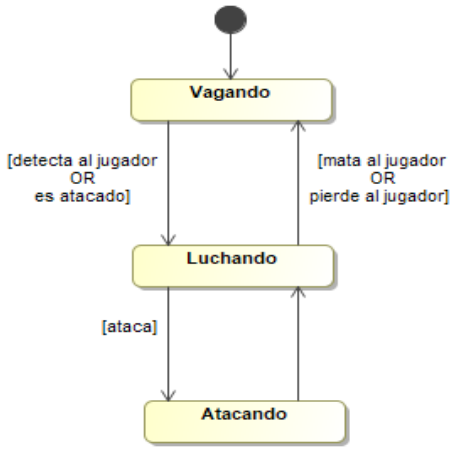


Figura 4.9: Diagrama de estados animal agresivo

Para terminar, se diseñó el sistema de combate del que dispondrá el jugador para avanzar, el cual se observa en el diagrama de estados de la figura 4.10. El jugador podrá fijar la cámara en sus enemigos y así cambiar la funcionalidad del botón "espacio", que normalmente activa el salto, a la de esquivar, más apropiada para el combate.

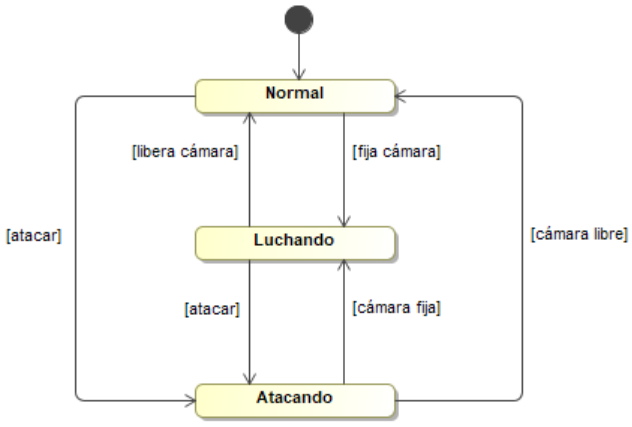


Figura 4.10: Diagrama de estados del combate

Resultados: diseño del sistema de combate y de los enemigos terminado, skybox corregida y comportamientos de los enemigos implementados.

4.8 Sistema de combate (16/06/2020 - 01/07/2020)

Se empezó por implementar el sistema de combate diseñado en el anterior sprint. Para ello se integraron las animaciones del personaje con el comportamiento que debía llevar a cabo (figura 2.10), así como se ajustaron ciertos detalles, como la posición de la cámara y las transiciones entre animaciones. Los estados que pueden observarse en la figura 2.10 se traducen en lo siguiente:

- Die: animación de morir.
- Idle_01: animación del personaje parado.
- Run: animación de correr.
- Jump: animación de saltar.
- Attack_01: primera animación de ataque.
- Attack_02: segunda animación de ataque.
- Attack_03: tercera animación de ataque.

Las transiciones son similares a las de la figura 4.11, cambiando el hecho de que en este caso son los posibles estados de la máquina de estados de animación del personaje en cualquier momento, no solo combatiendo.

Se le hicieron pruebas dentro del juego, con las que se detectaron varios errores en el combate, así como en los comportamientos de algunos enemigos. Por tanto, se pulió el sistema de combate y los comportamientos y animaciones de todos los PNJ.

Resultados: sistema de combate completo, animaciones del personaje y los enemigos ajustadas e integradas con los comportamientos programados, agilización del combate aumentando la velocidad de ciertos movimientos.

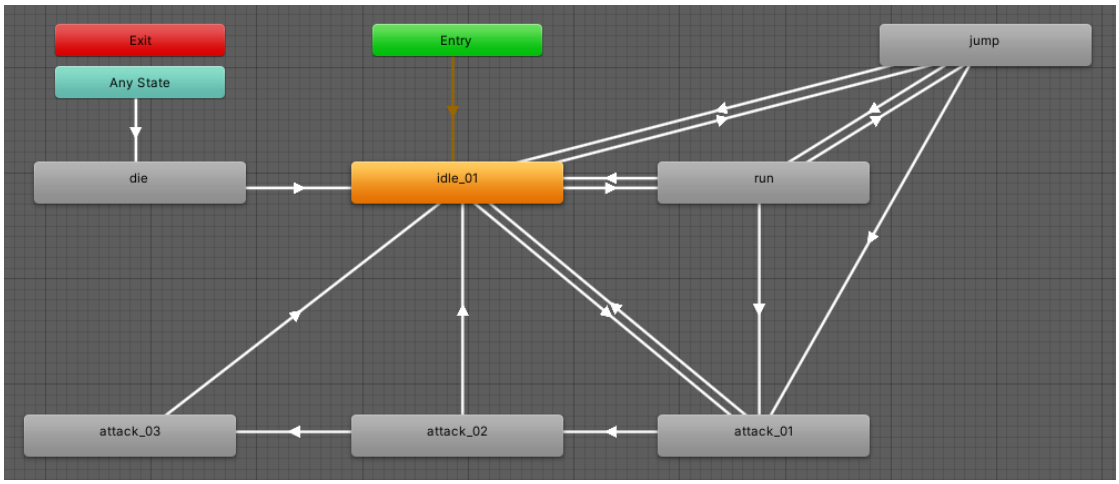


Figura 4.11: Diagrama de estados de animaciones del personaje en Animator.

4.9 Interfaces y sistema de guardado (02/07/2020 - 20/07/2020)

En este sprint se diseñaron e implementaron las interfaces de usuario del juego. Se empezó por diseñar las interfaces del menú principal, el menú de pausa y la interfaz dentro del juego. En esta última se decidió mostrar la vida, la energía y el hambre del jugador, y su diseño se encuentra en el documento de diseño de juego adjuntado con el trabajo. Para ello se utilizó la librería Unity.UI, que utiliza anclas (figura 4.12) para los elementos de la interfaz, de manera que esta es responsiva al tamaño y forma del dispositivo en el que se ejecute el videojuego. Los elementos con los que se puede interactuar (como botones) cuentan con distintas opciones de personalización, como los colores o texturas que tienen al ser pulsados o al pasar el ratón encima, así como un sistema de eventos (figura 4.13) que permite conectar el evento a una función desde el editor.

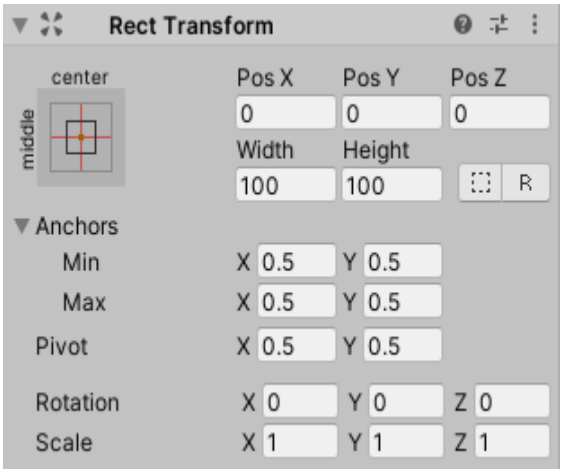


Figura 4.12: Transform de un elemento de interfaz.

Figura 4.13: Opciones de un botón en Unity.

Tras la implementación de las interfaces, se empezó a trabajar en el sistema de guardado del juego, el cual se encarga de hacer persistente la información de la partida del jugador. Para ello, se almacenan varios datos sobre el usuario y el juego: su posición,

su vida, su hambre y las semillas con las que se generó el mundo originalmente, para permitir volver a generarlo exactamente igual, como se muestra en la figura 4.14.

El día 15, tras terminar con lo anterior, se empezó un proceso de varios días de refactorizaciones del código y optimizaciones, incluyendo ajustes en la configuración del terreno para lograr un resultado más realista y apropiado, ajustes en los controles del personaje y optimizaciones en el algoritmo de generación del terreno. También se implementó un sistema para instanciar a los jefes de cada continente dentro de los límites de este, asegurando solo haber uno por continente.

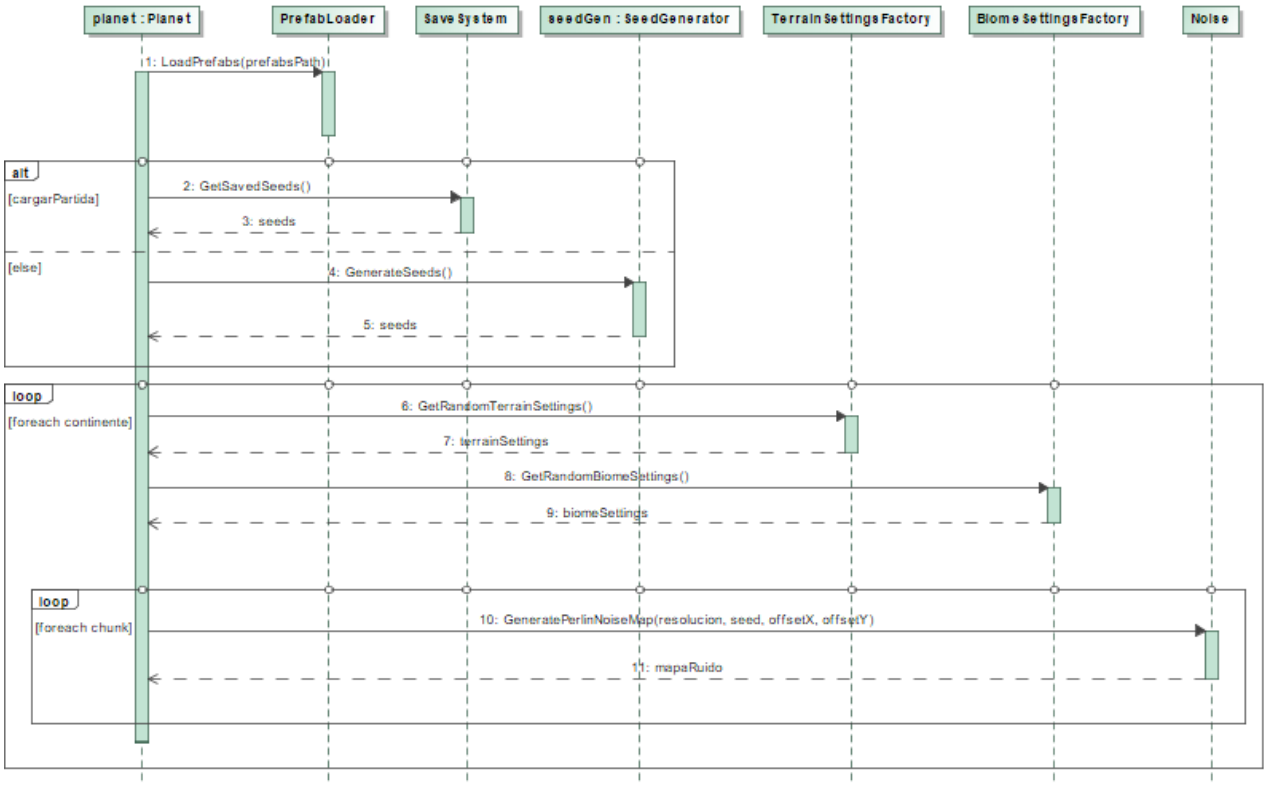


Figura 4.14: Inicialización del planeta con los datos guardados

Resultados: interfaces implementadas, sistema de guardado de partida implementado, terreno mejorado y sistema de instanciación de los jefes implementado.

4.10 Redacción del grueso de la memoria (21/07/2020 - 21/08/2020)

La tarea principal de este último sprint es la de la redacción de la memoria, la cual contaba con ciertos capítulos planteados, necesitando aún el grueso de la redacción. El objetivo para este sprint es tener la primera versión de la memoria para su revisión con el tutor.

Intercalando con la redacción de la memoria, se agregaron ciertas funcionalidades y mejoras al juego, tales como la indicación de la posición del jefe del continente mediante un sistema de partículas en la espada del personaje (figura 4.15), ajustes en los atributos de los animales y jefes (vida, daño, etc.) y ajustes en los parámetros de generación del terreno.

Figura 4.15: Sistema de partículas que indica la dirección hacia el jefe del continente

Resultados: memoria preliminar redactada, ajustes varios en el videojuego desarrollado.

4.11 Pruebas y ajustes (22/08/2020 - 01/09/2020)

Durante la revisión de la memoria por parte del tutor se llevaron a cabo pruebas jugables llevadas a cabo por mí mismo, en las que se probó el sistema de combate a fondo, las físicas, los sistemas de hambre, salud y energía, etc.

Se encontraron varios errores:

- Un fallo al encadenar las animaciones de golpeo del jugador, que hacía que se gastara la energía sin realizar el golpe.
- Un fallo en el sistema de salud, que impedía que se recuperara salud al estar fuera de combate.
- Un fallo en el sistema de oxígeno, que provocaba que el jugador siguiera ahogándose tras salir del agua.

- Un fallo en las físicas, que hacía que los perros saltaran una distancia inmensa, dado un fallo de cálculo en las fuerzas a aplicar respecto a su masa.

Resultados: se resolvieron todos los errores encontrados.

4.12 Revisión de la memoria (02/09/2020 - 11/09/2020)

Durante este último sprint se atendieron las puntualizaciones hechas por el tutor respecto a la memoria, añadiendo capítulos y reestructurando buena parte de ella para ajustarse mejor al proyecto llevado a cabo.

Tras este último sprint, no se realizaron ya grandes tareas salvo la última corrección de detalles de la memoria y algunos pequeños ajustes en el videojuego.

Resultados: memoria corregida enviada al tutor para la última revisión.

5

Conclusiones

Se ha implementado un videojuego de mundo abierto, en el que el mapeado y los objetos de la escena de juego son generados de forma procedimental y aleatoria, consiguiendo que cada partida creada se genere un planeta diferente. Para ello se ha utilizado una metodología ágil, en este caso SCRUM, en la que se ha dividido el trabajo en sprints, donde los requisitos a implementar y el diseño del videojuego son flexibles.

El videojuego implementado es apto para ser jugado y completado por un jugador, así como también admite posteriores investigaciones y mejoras tanto en sus mecánicas de juego como en el algoritmo de generación procedimental.

5.1 Mejoras y trabajo futuro

Dada la envergadura del proyecto, muchas posibles ampliaciones se descartaron en el camino para asegurar la compleción y calidad de lo que se ha decidido implementar. El videojuego desarrollado puede ser ampliado y corregido en varios aspectos, que se enumeran a continuación.

La principal mejora que podría plantearse es el sistema de posicionamiento de objetos en la escena, el cual tiene sus limitaciones en cuanto a la variedad de objetos que puede colocar de forma realista, al igual que carece de una forma para incrementar el número de un cierto objeto en particular, manteniendo los demás. Esto es notable en cuanto a los animales del planeta, cuyo número aumenta necesariamente si se quiere aumentar la cantidad de vegetación.

Una de las características descartadas pronto en el proyecto fue la posibilidad de viajar libremente entre los continentes del planeta. El juego tiene la capacidad de generar cualquier parte del planeta en función de su distancia con el jugador, sin embargo no ha sido posible desarrollar un sistema de navegación en barco (o parecido) que permita surcar los océanos y explorar el mundo con total libertad. Sería una mejora notoria el aprovechar los océanos como parte de los elementos explorables del juego, que también supondría ciertos desafíos a nivel técnico.

Durante las últimas pruebas del videojuego, se apreció que al aumentar sustancialmente el detalle del mapeado (y por tanto las resoluciones de los chunks y los mapas de ruido), el tiempo de carga inicial del videojuego crecía exponencialmente. Esto se debe a que, a pesar de cargar los contenidos en tiempo real, el algoritmo hace el cálculo de los mapas de ruido de todos los chunks del planeta de forma previa a iniciar el juego. Se podría plantear modificar la clase Noise, que se ocupa del cálculo del ruido, para que cada chunk tuviera una instancia de esta con todos los datos necesarios para generar el mapa de ruido, pero que sin embargo solo lo calculara realmente cuando fuera necesario para cargar ese chunk.

Una parte del videojuego a la que no se ha podido dedicar el tiempo necesario han sido las inteligencias artificiales de los enemigos, que son bastante básicas. Se podría ampliar el juego haciendo que los enemigos fueran más variados y complejos; así como mejorar el sistema de combate, añadiendo más opciones de cara a las animaciones del personaje y sus transiciones.

A nivel de historia, las posibilidades de desarrollo han sido mínimas. sería buena opción optar por aportarle una historia al videojuego, con contenidos alineados con ella generados también de forma procedimental. Esto puede suponer un reto al tener que compatibilizar una historia lineal con un mundo que se genera de forma aleatoria.

Por último, podría plantearse el utilizar el algoritmo desarrollado para desarrollar un videojuego con múltiples planetas, a los que el jugador pudiera desplazarse con libertad por el espacio. Para ello debería usarse un sistema adicional de posicionamiento procedimental, que decidiera en qué punto del espacio colocar cada planeta.

5.2 Aprendizaje personal

A nivel personal, este trabajo me ha aportado muchos conocimientos y experiencia en el campo del desarrollo de videojuegos. Se han asentado mis conocimientos previos de Unity, además de haberse ampliado en todos los aspectos.

En cuanto a Unity, he profundizado en su API de C# (Unity API, 2019.3), habiendo hecho uso de sistemas que desconocía como los ofrecidos por la clase Mesh, que permite la creación y modificación de mallas en tiempo real, o en la utilización de los vectores y rotaciones dentro de la escena, en lo cual he tenido que profundizar bastante al tener que ajustarlo todo a cualquier punto en el planeta. También he profundizado en otros sistemas que aporta Unity, como lo son el sistema de animación, el sistema de iluminación, la creación de materiales, los sistemas de partículas y el manejo de modelos 3D dentro del motor. Todo ello me ha elevado a un nivel de desarrollo en Unity bastante mejor que el que tenía previamente al trabajo, tanto que probablemente sea suficiente como nivel base para empezar de forma profesional.

He descubierto el mundo de la generación procedimental en los videojuegos, aprendiendo técnicas como el ruido de Perlin o el ruido Simplex (el cual finalmente no se usó en el proyecto), lo cual me ha ayudado a comprender como funcionan muchos de los juegos que emplean estas tecnologías, así como también me ha iniciado en este tipo de técnicas para poder seguir explorándolas en mayor profundidad en el futuro; aprendiendo también en el camino a manejar (aunque a un nivel superficial) el sistema de hebras de C#, y el funcionamiento interno del motor de Unity a nivel de hebras.

He asentado mis conocimientos en cuanto a la metodología SCRUM, al haber llevado a cabo un proyecto de duración considerable y cumpliendo generalmente los objetivos de cada sprint.

5.3 Problemas técnicos encontrados

Las mayores dificultades se presentaron en las fases iniciales del proyecto, con todo lo relacionado con la generación del planeta de forma procedimental. Al ser un campo nuevo el proceso de aprendizaje fue difícil y las primeras fases del desarrollo estuvieron llenas de contratiempos. Esto hizo que el proyecto se alargara y se tuviera que renunciar a ciertas ideas por falta de tiempo. Los principales problemas vinieron por los cálculos espaciales para poder generar planetas de cualquier radio, así como del sistema de

hebras, que debía hacer los cálculos para después modificar ciertas estructuras de datos que posteriormente procesaría la hebra principal de Unity, ya que el motor no permite la modificación de objetos de la escena en otras hebras, teniendo por ello que implementar un sistema de hebras que garantizara la exclusión mutua y evitara el bloqueo entre las hebras, que eran numerosas (decenas en tiempo de ejecución).

Dadas las circunstancias excepcionales a causa del Covid-19, las reuniones presenciales no fueron posibles, haciendo más difícil la comunicación y las revisiones de los avances del proyecto, aunque se hizo lo posible para mantener una comunicación fluida y revisiones constantes del proyecto.

Referencias

[Troelsen y Japikse, 2017] Andrew Troelsen, Philip Japikse, Pro C# 7: With .NET and .NET Core, Eighth Edition, 2017

[Okita, 2019] Alex Okita, Learning C# programming with Unity 3D, Second Edition, 2019

[Unity API, 2019.3] Referencia de Lenguaje de Unity (Accedido en agosto de 2020) https://docs.unity3d.com/es/530/ScriptReference/index.html

[Unity Perlin, 2019.3] Mathf.PerlinNoise (Accedido en abril de 2020) https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html

[Perlin, 1985] An Image Synthesizer (Accedido en julio de 2020) https://www.cs.drexel.edu/~david/Classes/Papers/p287-perlin.pdf

[The Verge, 2020] Ventas de Minecraft (Accedido en agosto de 2020) https://www.theverge.com/2020/5/18/21262045/minecraft-sales-monthly-players-statistics-youtube

[Unity Corrutinas, 2019] Corrutinas en Unity (Accedido en agosto de 2020) https://docs.unity3d.com/es/530/Manual/Coroutines.html

[Unity Monobehaviour, 2019] Clase Monobehaviour (Accedido en julio de 2020) https://docs.unity3d.com/es/530/ScriptReference/MonoBehaviour.html

[Unity MeshRenderer, 2019] Componente MeshRenderer (Accedido en abril de 2020) https://docs.unity3d.com/Manual/class-MeshRenderer.html

[Unity Collider, 2019] Colliders en Unity (Accedido en agosto de 2019) https://docs.unity3d.com/es/2018.4/Manual/CollidersOverview.html

[Wikipedia Composite, 2020] Patrón Composite (Accedido en agosto de 2020)
https://es.wikipedia.org/wiki/Composite_(patr%C3%B3n_de_dise%C3%B1o)#:~:text=El%20patr%C3%B3n%20Composite%20sirve%20para,estructura%20en%20forma%20de%20%C3%A1rbol.

[Unity RigidBody, 2019] Componente Rigidbody (Accedido en julio de 2020)
https://docs.unity3d.com/es/2019.4/Manual/class-Rigidbody.html

[Unity GameObject, 2019] Componente GameObject (Accedido en julio de 2020)
https://docs.unity3d.com/es/2018.4/Manual/class-GameObject.html

[Unity Awake, 2019] Método Awake de MonoBehaviour (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

[Unity Start, 2019] Método Start de MonoBehaviour (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html

[Unity Update, 2019] Método Update de MonoBehaviour (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

[Unity FixedUpdate, 2019] Método FixedUpdate de MonoBehaviour (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html

[Unity LateUpdate, 2019] Método LateUpdate de MonoBehaviour (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html

[Unity Input, 2019] Interfaz Input (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/Input.html

[Unity Transform, 2019] Componente Transform (Accedido en agosto de 2020)
https://docs.unity3d.com/es/2018.4/Manual/class-Transform.html

[Unity Mesh, 2019] Clase Mesh (Accedido en agosto de 2020)
https://docs.unity3d.com/ScriptReference/Mesh.html

[Wikipedia Scrum, 2019] Metodología Scrum (Accedido en agosto de 2020)
https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

[César León, 2019] Documento de Diseño de Juego (Accedido en septiembre de 2020)
https://indielibre.com/2019/04/17/la-importancia-del-gdd-game-design-document/

Apéndice A

Manual de Instalación

Requerimientos:

Windows 10, cualquier distribución.

El videojuego se ha desarrollado sobre un portátil con las siguientes características:

- Procesador Intel i5 -7200U
- Tarjeta gráfica Nvidia GeForce GTX 950M
- Memoria RAM de 8GB
- Windows 10 Home

Con las características listadas , el videojuego se ejecuta a una tasa de fotogramas de aproximadamente 80 fotogramas por segundo, con lo cual un hardware inferior debe ser capaz de ejecutarlo con menor rendimiento. Se recomiendan las siguientes características mínimas:

- Procesador Intel i3 o similar
- Tarjeta gráfica Nvidia GeForce 930M o similar
- Memoria RAM de 6GB

Instalación:

Para ejecutar el juego únicamente deberá abrirse la carpeta del proyecto (descomprimiéndola primero, si estuviera comprimida) y ejecutar Rings.exe.

Anexo I

Documento de Diseño de Juego

A continuación se adjunta el Documento de Diseño de Juego redactado durante este proyecto. Este documento ha evolucionado conjuntamente con el desarrollo del juego para adaptarlo a las necesidades y cambios propuestos en cada sprint; y en base al criterio de desarrollar el videojuego más disfrutable posible para el jugador.

Documento de diseño de

Redactado por:

Marco De Sousa

Última actualización:

08/09/2020

Índice

Fundamentos del diseño

Este juego se desarrolla en un planeta con 6 continentes que el jugador podrá explorar con libertad y tomando las decisiones sobre qué hacer y hacia dónde ir en cada momento dentro de los continentes. El planeta será esférico, con vistas a que sea completamente explorable en el futuro (incluidos océanos). Cada jugador tendrá un planeta único ya que en cada partida se genera un planeta de manera procedimental, con parámetros aleatorios que hacen que cada mundo sea distinto. Cada continente podrá tener un bioma: boscoso, nevado o desértico.

El jugador interactuará con el entorno a través de una perspectiva en primera persona, en 3D, pudiendo recolectar víveres para alimentarse, cazar, y luchar contra enemigos a través de un sistema de combate con influencias en títulos como Dark Souls, alejándose las mecánicas de movimiento y combate del estilo estático que muchos juegos en primera persona utilizan.

El estilo artístico será minimalista y colorido, con modelos 3D de bajo pulido; similar a Minecraft. Para ello se utilizarán modelos 3D procedentes de la Asset Store de Unity, con el estilo más apropiado para el aspecto que se busca para el videojuego. Se buscará una suficiente variedad tanto de fauna como de flora, con el objetivo de conseguir un paisaje variado y que no se haga aburrido para el jugador.

Concepto clave

Rings es un juego de aventura, acción y supervivencia con un mapa generado procedimentalmente en el que te enfrentarás tanto a un entorno deseando ser explorado como a enemigos desafiantes, a través de un sistema de combate dinámico en primera persona.

Género

Aventura, acción, supervivencia.

Plataformas

PC Windows 10.

Público objetivo

El juego será apto para jugadores de cualquier edad, aunque preferiblemente mayores de 7 años por la relativa complejidad de los controles.

El juego estará enfocado en gran medida a personas que prefieren las experiencias que conlleven cierta motivación intrínseca a ellos mismos, es decir, que prefieran proponerse sus propios objetivos y marcar su propio camino dentro del juego. Este perfil encaja con personas que disfruten de juegos como Minecraft, The Legend of Zelda: Breath of the Wild, The Elder Scrolls, etc. En definitiva, amantes de los mundos abiertos.

Sin embargo, no se dejará de lado la motivación extrínseca por medio de objetivos dados por el juego. En este sentido, no sólo se enfocará a jugadores que disfruten de la exploración, sino que se ofrece un sistema de combate dinámico que incentive el seguir los objetivos que a menudo llevarán a algún tipo de enfrentamiento.

Dado el estilo de mundo abierto, con un mapa de tamaño considerable, es un videojuego pensado para personas que disfruten de experiencias pausadas y relajantes, por lo cual no está muy enfocado a personas que disfruten de una acción continua en los videojuegos o que necesiten de constantes estímulos externos que los empujen a avanzar en el videojuego.

Historia

El jugador aparece en un planeta desconocido. Su nombre es simplemente ‘héroe’, y una voz en su cabeza lo guía en su búsqueda de los 6 anillos de poder esparcidos por el mundo.

Para liberar al planeta del azote de unos demonios de origen desconocido debe enfrentarse a ellos y encontrar los 6 anillos de poder que están en manos los demonios. En cada continente hay un anillo que está en manos de uno de los demonios.

Al recolectar los 6 anillos, el planeta queda libre y termina la misión, pudiendo regresar a su hogar.

Jugabilidad

Resumen de la Jugabilidad

Las mecánicas de juego serán principalmente la exploración, la recolección y el combate.

La exploración será principalmente la búsqueda de los demonios, así como la recolección de comida y caza de animales para sobrevivir; y el sistema de combate en primera persona adaptará lo mejor de la inmersión de la primera persona con una profundidad en las mecánicas que irán desde esquivar, fijar a los enemigos, controlar las distancias y golpear cuando se presente la oportunidad.

El jugador tendrá una luz en su espada que le indicará cuándo se dirige hacia el demonio de ese continente.

Mecánicas clave:

- Exploración
- Recolección
- Combate en primera persona
- Sobrevivir al entorno

El bucle de juego será como el de la figura 1.

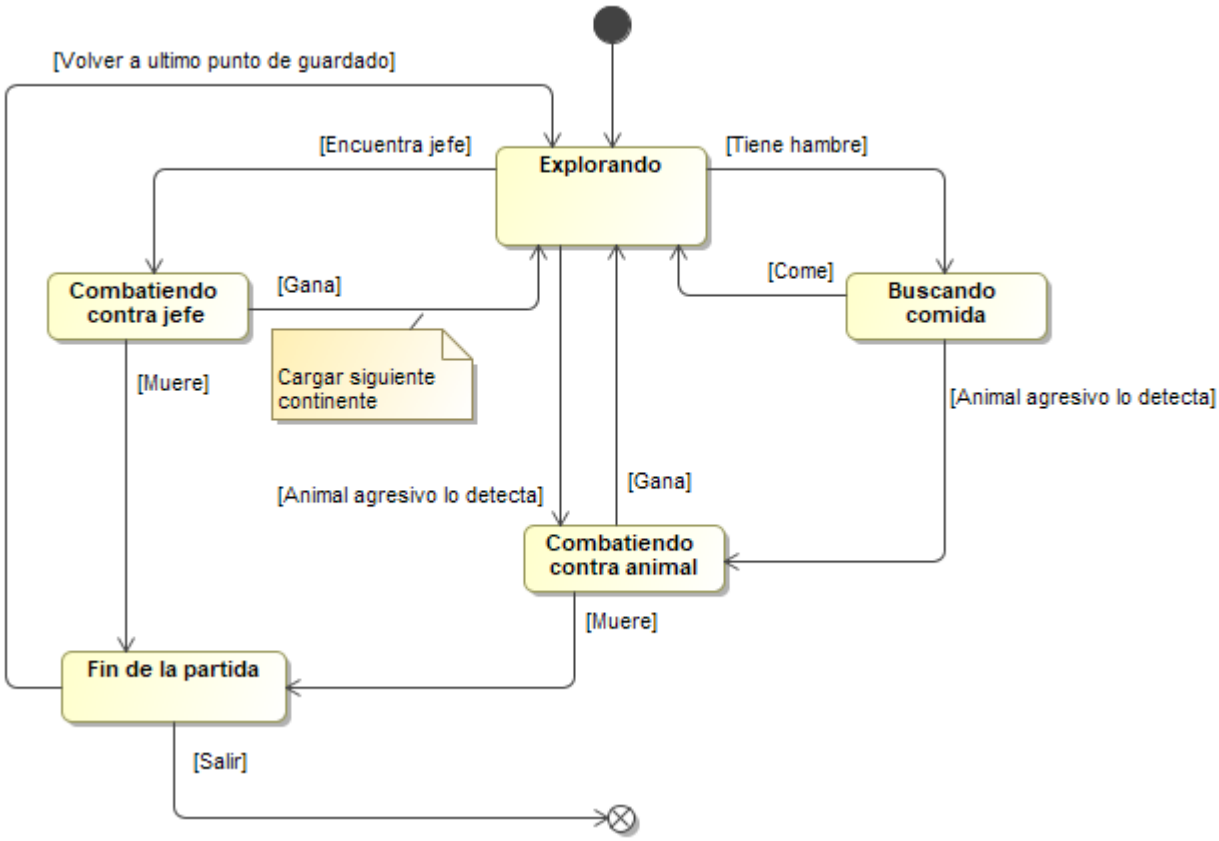


Figura 1. Bucle de juego

Experiencia Jugable

El juego te proporcionará una experiencia completa. Sentirás tranquilidad y curiosidad por explorar el mundo en busca de tesoros y paisajes nuevos; mientras por otro lado sentirás la presión por sobrevivir y la emoción del combate.

Ante todo, el jugador sentirá deseo por descubrir el mundo, con sus distintos ecosistemas y continentes, fauna y flora; así como diferentes enemigos para los que tendrás que utilizar todas tus habilidades en el cuerpo a cuerpo.

Además, estarás en una constante búsqueda en todos los continentes para encontrar el demonio que los asola; por lo que siempre podrás optar por disfrutar del entorno o continuar con el objetivo principal.

En definitiva, el juego pretende sentirse como una odisea donde el jugador es el único protagonista; una aventura por la que pasará por cada continente de un planeta gigantesco para finalmente poder lograr su objetivo.

Aptitudes puestas a prueba

Exploración: El jugador debe ser capaz de explorar su entorno, de manera que consiga localizarse en su continente (el cual tiene un tamaño considerable), así como encontrar los recursos que necesita para sobrevivir.

Supervivencia: Para sobrevivir al entorno, el jugador debe tener cuidado y mirar bien por dónde se aventura. De lo contrario podría verse en una situación en la que los enemigos lo superen en número y ser eliminado; o caerse desde un precipicio y tener que volver a recorrer el mismo camino, consumiendo su valioso medidor de hambre. También deberá estar atento tanto a su medidor de hambre (e ir en busca de alimento cuando este esté bajo), como a su medidor de oxígeno, que sólo aparecerá cuando el jugador esté en el agua y puede provocar que se ahogue.

Combate: Para salir victorioso de un combate no bastará con golpear al enemigo sin ninguna estrategia, ya que la mayoría de estos producen bastante daño con sus golpes y tienen una considerable cantidad de puntos de vida. Para vencer en la mayoría de los combates, el jugador deberá reaccionar ante los ataques que le lleguen, esquivando los máximos posibles y aprovechando los momentos de debilidad del enemigo para atacarlo.

Orientación: El jugador deberá hacer lo posible para mantenerse orientado durante el juego, ya que de lo contrario podría perderse. Su espada lo guiará hacia el demonio del continente en el que se encuentra, pero nada más. Si el jugador desea explorar otras partes de la isla deberá guiarse por el paisaje y su memoria.

Mecánicas de Juego

Lista detallada de las mecánicas.

Atributos de los personajes	
Personaje	Movimientos y acciones disponibles
Jugador	1. Movimiento en 3 dimensiones. 2. Rotar sobre sí mismo. 3. Mirar hacia arriba y abajo. 4. Saltar. 5. Esprintar. 6. Atacar cuerpo a cuerpo con su espada. 7. Esquivar los ataques de los enemigos. 8. Fijar la mirada en un enemigo.

El jugador tendrá diferentes acciones disponibles dependiendo de su situación (detallado más adelante en el apartado “Controles”).

Movimiento en 3 dimensiones

El jugador podrá moverse hacia cualquier dirección que el terreno permita, pudiendo subir por pendientes que no sean excesivamente escarpadas, bajar por cualquier pendiente descendente. Podrá caminar hacia adelante, los lados y hacia atrás.

Rotar sobre sí mismo

El jugador podrá rotar su cuerpo hacia la izquierda y la derecha, rotando también consigo la cámara. De esta manera puede inspeccionar sus alrededores y cambiar la trayectoria de su movimiento.

Mirar hacia arriba y abajo

El jugador podrá inclinar su cabeza hacia arriba y abajo, con un límite de 60 grados de rotación. La cámara también rotará de la misma manera.

Saltar

El jugador podrá saltar, lo cual puede resultarle útil para ayudarse a escalar una montaña o salir del agua si se está ahogando.

Esprintar

El jugador podrá esprintar, aumentando la velocidad a la que se desplaza. Esto consumirá su indicador de energía y aumentará su consumo del indicador de hambre, por tanto debe ser precavido con el uso de esta habilidad.

Atacar cuerpo a cuerpo con su espada

El jugador podrá atacar al enemigo con su espada, en combinaciones de 3 estocadas. Cada golpe consume parte de su barra de energía, por lo que debe tener cuidado ya que si la gasta entera no podrá esquivar un posible ataque enemigo. El jugador puede elegir entre hacer la combinación completa de 3 golpes, así como dar solamente 1 o 2 golpes para ahorrar energía o esquivar un ataque.

Esquivar

Un esquive hacia la dirección en la que se esté moviendo el jugador. Es un movimiento rápido que hace invulnerable al jugador mientras lo realiza. Este movimiento consume puntos de energía, con lo cual debe usarse con cautela para no acabar sin energía ante el enemigo. Es necesario tener la cámara fijada en un enemigo para realizarlo.

Fijar la mirada en un enemigo

Esta es la mecánica principal del combate. Orienta al jugador hacia el enemigo, no pudiendo girar hacia sus lados. El jugador queda orientado hacia ese enemigo hasta que cancela el fijado, manteniéndose mirando al enemigo sin importar en qué dirección se mueva.

En la figura 2 se muestra el diagrama de estados de las animaciones del personaje, pudiendo observarse desde qué animación puede transitarse a otra.

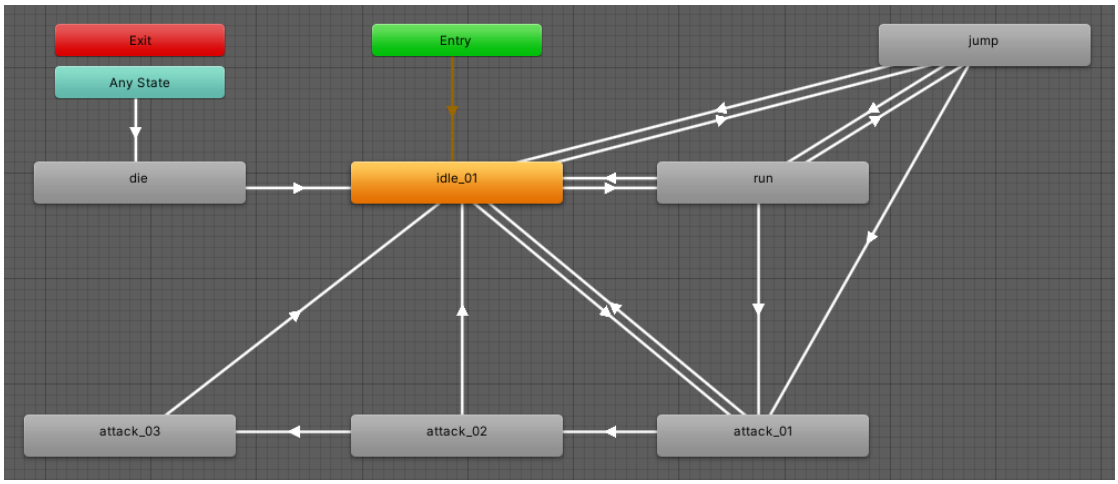


Figura 2. Diagrama de estados de las animaciones

El aspecto visual del jugador puede observarse en la figura 3, aunque durante el juego lo único que podrá verse de este es su espada y sus piernas, ya que la cámara está en primera persona.

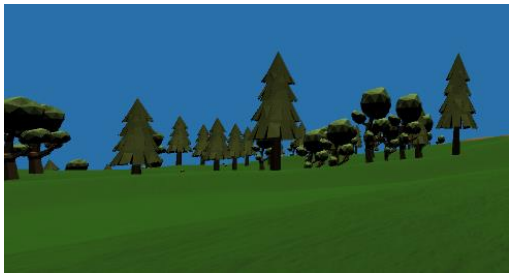
Figura 3. Jugador

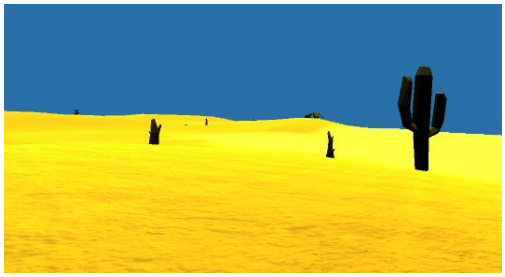
Diseño de Niveles

Niveles y enemigos presentes. Todos los elementos artísticos (modelos 3D) procederán de la Asset Store de Unity. Las texturas para las partículas y terreno se obtendrán de imágenes sin copyright en Internet.

Ciertos tipos de vegetación pueden ser usados para obtener comida y rellenar el indicador de hambre.

Cada continente pertenece a uno de los tres tipos de biomas: boscoso, nevado o desértico. Cada bioma se diferencia de los demás visualmente, pero además tienen ciertas implicaciones a nivel jugable, como se ve en la siguiente tabla:

Niveles	Descripción
Bioma boscoso 	Es un bioma con gran cantidad de vegetación que se destaca por colorido primaveral. Mayor densidad de fauna y flora entre los biomas. Animales pacíficos: Gallina Animales agresivos: Lince Este tipo de nivel se caracteriza por tener más facilidades que en los otros dos, pero a la vez cuenta con más dificultades. Al haber más presencia tanto de flora como de fauna, la comida será abundante. Sin embargo, también significa que la presencia de animales agresivos es mayor, y el jugador deberá tener más cuidado a la hora de desplazarse para no ser atacado por sorpresa por algún enemigo.
Bioma nevado 	Es un bioma en el que abundan los pinos y se destacan paisajes nevados. Tiene menos vegetación y fauna que el bioma boscoso, pero sigue siendo abundante. Animales pacíficos: Gallina Animales agresivos: Lobo

	En este bioma hay un equilibrio entre abundancia y amenaza. Al tener menos fauna que el bioma boscoso, es menos peligroso ya que serán menos frecuentes los animales agresivos en el terreno. Esto queda contrarrestado por su carencia de vegetación comestible, con lo cual el jugador es forzado a cazar para sobrevivir.
Bioma desértico 	Es un bioma en el que se encuentra poca vegetación y grandes desiertos. Animales pacíficos: Gallina Animales agresivos: León Este bioma, que es con diferencia el más árido, se caracteriza por su poca abundancia de todo tipo de recursos. La comida es escasa y debe aprovecharse toda la que se encuentre; sin embargo, tiene la ventaja de ser el bioma que menos densidad de animales agresivos posee, por lo cual el jugador tendrá pocas amenazas aparte de morirse de hambre.

Además de los biomas, los continentes pueden tener dos tipos de topografías: tipo montañoso y tipo planicie. El tipo de bioma puede combinarse con cualquiera de los dos tipos de terreno. Los terrenos montañosos son más difíciles de transitar, ya que el jugador

debe pasar por desniveles importantes y grandes pendientes y precipicios. La ventaja que tiene es la mayor facilidad para orientarse, ya que desde la cima de una montaña tiene una visibilidad muy buena del entorno.
Por otro lado, el terreno plano es más cómodo de transitar, pero carece de la visibilidad que aportan las montañas, con lo cual será más difícil orientarse y más fácil perderse.

La flora del planeta es la siguiente:

Pinos (figura 4): este árbol se halla tanto en el bioma boscoso como en el nevado, siendo el único tipo de vegetación disponible en el bioma nevado. No proveen de alimento al jugador.

Árboles frutales (figura 5): esté árbol es exclusivo del bioma boscoso. Son abundantes en prácticamente cualquier altitud, salvo cerca de los picos de las montañas. Proveen de alimento al jugador.

Arbusto (figura 6): esta planta es exclusiva del bioma boscoso. No provee alimento y es común en las mismas altitudes que los árboles.

Cactus (figura 7): esta planta es exclusiva del bioma desértico. No provee alimento y se encuentra en altitudes medias.

Chumbera (figura 8): esta planta es exclusiva del bioma desértico. Provee de alimento al jugador y se encuentra en altitudes medias.

Palmera (figura 9): esta planta es exclusiva del bioma desértico. Provee al jugador de alimento y se encuentra a altitudes bajas, cerca del agua.

Tronco seco (figura 10): esta planta es exclusiva del bioma desértico. No provee de alimentos y se encuentra a altitudes medias.

Figura 4. Pino

Figura 5. Árbol frutal

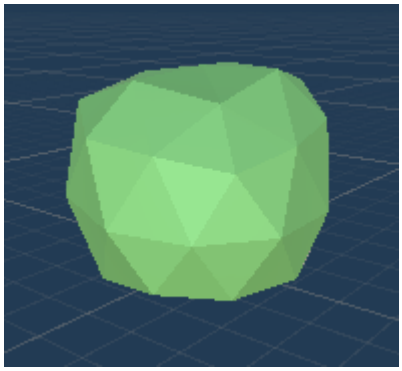


Figura 6. Arbusto

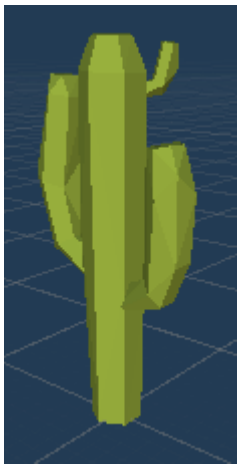


Figura 7. Cactus

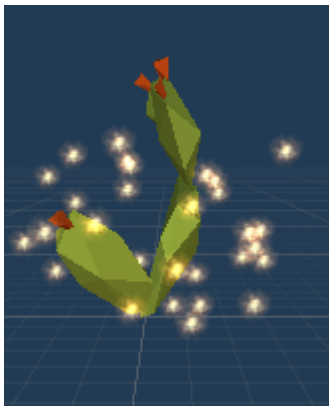


Figura 8. Chumbera

Figura 9. Palmera

Figura 10. Tronco seco

Enemigos

Enemigo	Comportamiento
Animales pacíficos *(Gallina)*	Este tipo de animales no atacará al jugador. Deambulan por el terreno y huyen al ser atacados.
Animales agresivos *(Lince, Lobo, León)*	Este tipo de animales atacarán al jugador cuando lo vean, con ataques cuerpo a cuerpo que vendrán precedidos de señales para que el jugador pueda evitarlos. El resto del tiempo deambulan por el terreno. Tendrán el mismo comportamiento, pero distintos atributos (vida, daño, velocidad…).
Demonio	Estos enemigos son los más peligrosos. Atacarán al jugador en caso de verlo, y sus ataques serán a distancia utilizando bolas de fuego. Al matarlos se obtiene uno de los 6 anillos, siendo transportado a otro continente para buscar el siguiente.

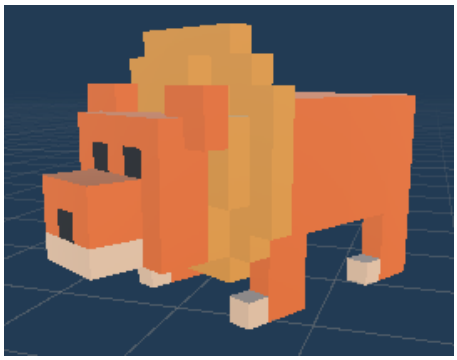


Figura 11. León.

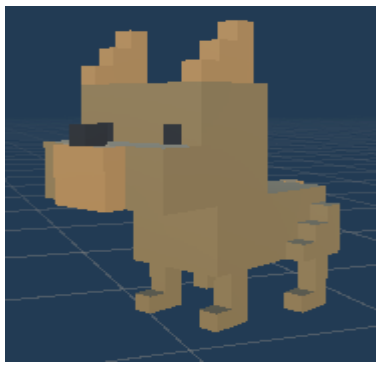


Figura 12. Lobo.

Figura 13. Lince.

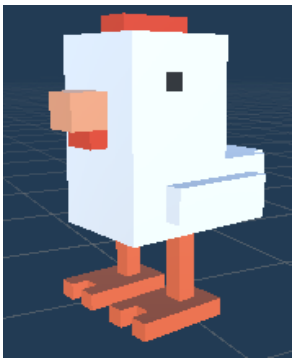


Figura 14. Gallina.

Figura 15. Demonio.

Los demonios se colocarán en un chunk aleatorio del continente (siempre en tierra), teniendo el jugador que buscarlos con su espada para matarlos. También deambulan por el terreno, con lo cual pueden alejarse considerablemente de su punto de aparición.

Los comportamientos de cada enemigo vienen descritos en las figuras 16, 17 y 18.

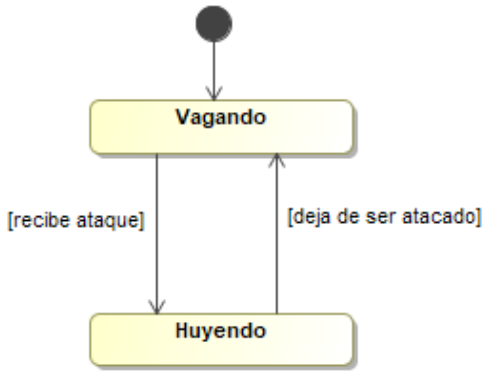


Figura 16. Estados animal pacífico.

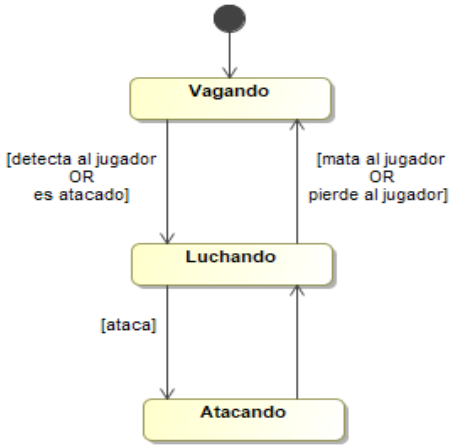


Figura 17. Estados animal agresivo.

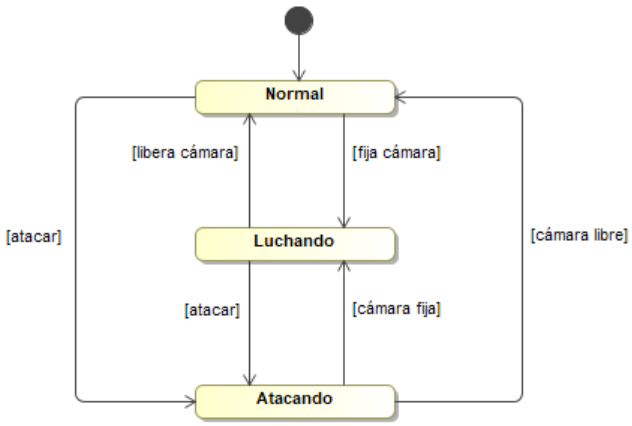
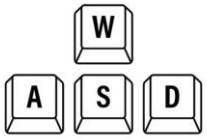
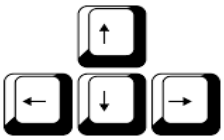
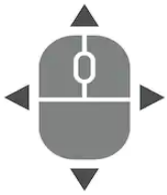
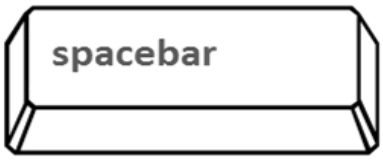
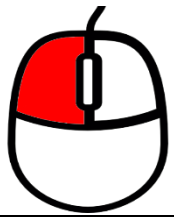
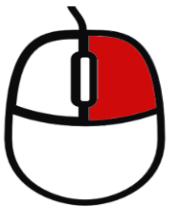
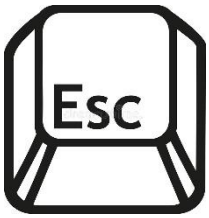


Figura 18. Estados demonio

Controles

Botones	Acción que realiza y estado necesario
  WASD y flechas	Cualquier estado: Movimiento en las 4 direcciones (hacia delante, atrás y a los lados).
 Movimiento del ratón	Sin cámara fija: Rotar cámara horizontal y verticalmente. Límite de 60º para la rotación vertical.
 Barra espaciadora	Sin cámara fija: Saltar. Con cámara fija: Esquivar hacia la dirección actual de movimiento.
 Shift izquierdo	En el suelo: Esprintar.
	

Click izquierdo	Cualquier estado: Golpear con el arma.
 Click derecho	Sin cámara fija: Fijar la cámara en el enemigo más cercano a la orientación del jugador, si hay alguno. Con cámara fija: Cancelar el fijado de cámara.
 Escape	Pausar el juego.

Interfaces de Usuario

Para las interfaces se tomará como referencias principalmente a Minecraft (menús) y Dark Souls (interfaz dentro del juego).

Interfaz dentro del juego

Esquina superior izquierda: Barra de vida y de resistencia.

Esquina inferior izquierda: Barra de hambre.

Parte superior de la pantalla: Barra de oxígeno.

Interfaz de los menús

Los menús tendrán sus botones distribuidos de forma vertical, centrados horizontalmente. El diseño de las interfaces será minimalista y se utilizarán los diseños predefinidos por Unity para los botones de los menús.

Contendrán los siguientes campos:

- Menú principal: Nueva Partida, Cargar Partida
- Menú de pausa: Continuar, Guardar, Salir

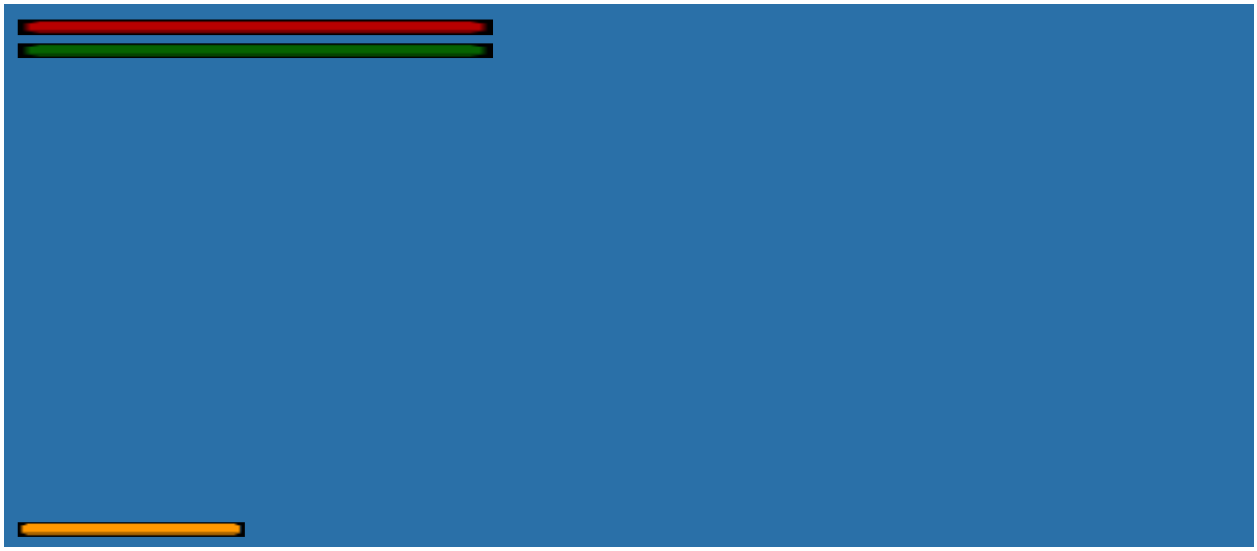


Figura 19. Interfaz in-game

El esquema de navegación entre las diferentes pantallas del juego puede observarse en la figura 20, en la que se esquematiza mediante un diagrama de actividades la conexión entre las diferentes interfaces del juego.

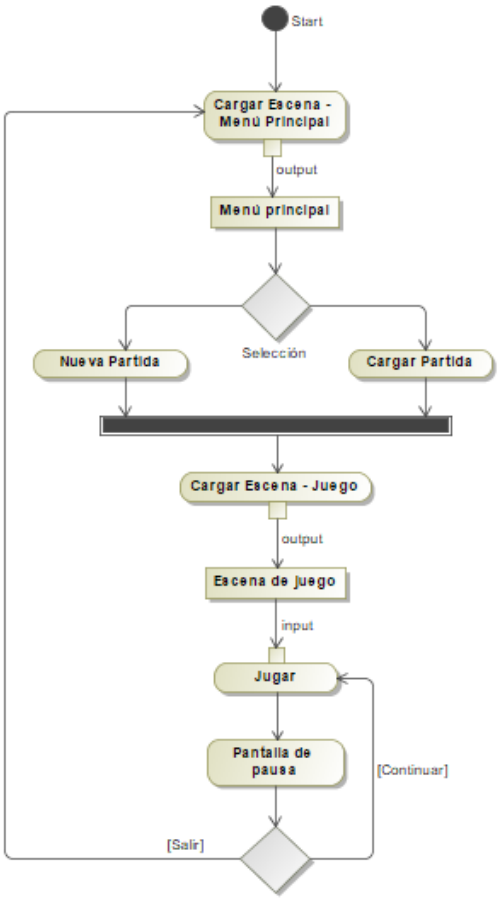


Figura 20. Flujo de pantallas del juego

Anexo II

Documento de Diseño Conceptual

A continuación se adjunta el Documento de Diseño Conceptual redactado al principio de este proyecto. Este documento sirvió de punto de partida para redactar el Documento de Diseño de Juego, así como para guiar e inspirar las decisiones tomadas respecto al videojuego.

Documento de diseño conceptual

Rings

Marco Antonio De Sousa Gonçalves

Versión 1.1

1. Pilares de diseño

1.1. Género

Aventura, supervivencia, mundo abierto.

1.2. Conceptos clave

1. La exploración es el principal objetivo que tendrá el jugador, con libertad total.

2. Cada jugador tendrá una experiencia única gracias a la generación de los contenidos de forma procedimental.
3. El jugador tendrá que sobrevivir a su entorno, tanto a los enemigos como a las condiciones del entorno.
4. Se jugará en primera persona y el sistema de combate será dinámico y ágil.

1.3. Objetivos

El juego transcurre en dos planetas. El primer planeta será parecido a la tierra. El segundo será un planeta parecido al infierno. Ambos tendrán 6 continentes. El primer planeta tendrá varios biomas, mientras que el segundo solo tendrá un tipo de bioma.

En el mapa habrá puntos de reaparición que el jugador podrá usar para revivir en ese punto la próxima vez que muera.

En el primer planeta el jugador deberá buscar los 6 anillos para poder atravesar el portal que le lleva al otro planeta, en el que están los 6 reyes de los demonios que debe matar para impedir que invadan el primer mundo.

2. Mecánicas de juego

2.1. Jugabilidad

El jugador explora un planeta salvaje en el que aparece sin explicación, no teniendo en principio objetivo.

La cámara estará en primera persona y el combate será muy dinámico, pudiendo esquivar, fijar la cámara en los enemigos (lo cual cambia los controles al modo combate) y utilizar distintos tipos de ataques. La jugabilidad en combate se inspirará en Dark Souls y The Legend of Zelda, adaptando lo necesario para la primera persona. Cada anillo que consiga dará al jugador una nueva habilidad o mejora.

El jugador cuenta con un indicador de hambre, el cual irá bajando con el tiempo y deberá rellenar cazando o recolectando comida. En caso de vaciarse por completo, se desmayará y aparecerá en su último punto de reaparición. Los puntos de vida irán ligados al nivel de hambre, sólo recuperándose hasta el nivel en que éste esté (si el indicador de hambre está al 70%, la vida solo se regenerará hasta el 70%). Si el indicador de hambre llega al 0%, el jugador empezará a perder vida en lugar de regenerarla. Cuando posea los 6 anillos y atraviese el portal, el jugador ya no tendrá indicador de hambre. A partir de este momento su vida no se regenerará, salvo cuando haga daño a un demonio.

No se dispondrá de inventario, llevando el jugador consigo sólo las armas que esté utilizando, así como cualquier equipamiento; y la comida se consumirá al momento, no pudiendo almacenarla.

Para orientarse, el jugador contará únicamente con una brújula.

2.2. Controles

Teclado y ratón

Cámara: Se moverá junto con el movimiento del ratón, sin necesidad de presionar ningún botón.

Movimiento: WASD. Espacio para saltar. Mantener shift para correr.

Acciones: Clic izquierdo para atacar. 'E' para interactuar (recolección, diálogos, etc.). Clic derecho para fijar a un enemigo (la cámara quedará fija en él).

En combate: Espacio + Botón de movimiento (WASD) para esquivar en la dirección en la que se esté moviendo el personaje. Espacio para esquivar hacia atrás. Clic derecho para cancelar fijado.

En menús: Clic izquierdo para elegir opción. Escape para volver.

2.3. Enemigos

Los enemigos atienden a las siguientes categorías:

1. Animal: No atacarán al jugador, huirán en caso de ser atacados. Al derrotarlos rellenan el indicador de hambre. Se encuentran en el primer planeta.
2. Bandido: Son agresivos con el jugador. Pueden atacar cuerpo a cuerpo y a distancia. Al ser derrotados sueltan equipamiento que el jugador puede cambiar por el suyo. Se encuentran en el primer planeta.
3. Demonio: Son agresivos con el jugador. Atacan al jugador cuerpo a cuerpo y son más fuertes, rápidos y resistentes que los bandidos. Derrotarlos recupera salud y aumenta el poder del jugador a través de la salud, resistencia y poder de ataque al absorber su poder a través de los anillos. Se encuentran en el segundo planeta.
4. Rey Demonio: Jefes del juego. Cada uno tendrá un set de movimientos diferente y habilidades únicas. Se encuentran en el segundo planeta.

2.4. Interacción con el entorno

De los árboles y plantas puede recoger alimento cada vez que éstos lo tengan disponible (estas fuentes de alimento tendrán tiempo de reutilización), rellenando así su indicador de hambre. De los animales que mate también recoge comida, en este caso le rellenará más el indicador de hambre.

En el mapa habrá tesoros repartidos, que el jugador podrá recoger. De éstos podrá obtener equipamiento.

2.5. Interacción del entorno con el jugador

Primer planeta

Según el bioma en el que el jugador se encuentre, sus indicadores de hambre y resistencia cambiará de forma diferente. En las zonas más inhóspitas estos atributos estarán más perjudicados.

Segundo planeta

El jugador pierde vida continuamente, teniendo que matar demonios para regenerarla.

3. Interfaz

3.1. Menú inicial

El menú inicial tiene como fondo una ilustración relacionada con el juego. Lleva escrito el título del juego, seguido de los botones “Nueva partida” y “Cargar partida”.

3.2. Menú de pausa

Este menú contiene los botones: “Inventario”, “Guardar partida” y “Volver al menú inicial”.

3.3. Interfaz in-game

La interfaz in-game será la mínima necesaria. Contiene el indicador de hambre, la vida y una brújula para que el jugador pueda orientarse.

4. Arte

4.1. Estilo

El estilo visual del juego será de modelos con pulido medio y un efecto visual de cel shading, con los siguientes referentes:

The Legend of Zelda: Breath of the Wild

The Legend of Zelda: Wind Waker

Borderlands

5. Sonido

5.1. Música

La música será minimalista durante los momentos de exploración. Se irá alternando entre ligeras melodías, acordes y el silencio para dejar al jugador solo con los sonidos de ambiente.

En el segundo planeta la música será oscura y pesada.

5.2. Efectos de sonido

Los efectos sonarán con un volumen superior al de la música y éstos incluirán:

- Pasos del jugador.
- Pasos de los demás seres.
- Viento (eventualmente).
- Ataques del jugador y los enemigos.
- Daño recibido por jugador o enemigos.
- Sonido del agua

5.3. Diálogos

Los PNJ emitirán onomatopeyas cuando hablen, apareciendo el texto en pantalla.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga