

Proceso de producción de un videojuego: Berserk 2D

Iker Roa Ameigidez

Resumen— Berserk 2D es un proyecto que nace de la iniciativa de querer abarcar todos los aspectos que forman el desarrollo de un videojuego: Diseño de niveles, de personajes, realización de cinemáticas 2D, creación de los escenarios y codificación de la lógica del juego, entre otros, teniendo en cuenta estándares de calidad en la jugabilidad y visualmente, logrando así un producto destinado a un hipotético videojugador de ordenador. Para este proyecto se ha tomado en cuenta el manga Berserk, concretamente su primer capítulo, escrito y dibujado por el autor japonés Kentaro Miura, trasladando su esencia estética y narrativa, a un juego en 2D con combate y exploración con una estética pixelart en el motor Unity y en un lenguaje C#.

Paraules clau— Berserk, 2D, videojuego, manga, pixelart, combate, exploración, Unity, C#

Abstract— Berserk 2D is a project that was born from the initiative of wanting to cover all the aspects that make up the development of a video game: Level design, character design, creation of 2D cinematics, creation of scenarios and coding of the game logic, among others, taking into account quality standards in playability and visually, thus achieving a product intended for a hypothetical computer gamer. For this project, the Berserk manga has been taken into account, specifically its first chapter, written and drawn by the Japanese author Kentaro Miura, transferring its aesthetic and narrative essence to a 2D game with a pixelart aesthetic in the Unity engine and in a C language#.

Index Terms— Berserk, 2D, videogame, manga, pixelart, combat, exploration, Unity, C#



Junio de 2024, Escola d'Enginyeria (UAB)

1 INTRODUCCIÓN - CONTEXTO DEL TRABAJO

Guts, nuestro guerrero protagonista, apodado el guerrero negro, tras ser encerrado en una mazmorra por los soldados de la realeza (FIGURA 1), ve a través de los barrotes de la ventana de su celda que da al exterior como la aldea ha entrado en un caos de llamas y muerte (el estigma de su cuello, símbolo de un peligro no humano, se ha activado). Tras que Puck, su “compañero” elfo, le haya dejado las llaves que abren la puerta de su celda, Guts recoge su espada y armadura para salir de la mazmorra sorteando los peligros de esta, sale del castillo y se adentra en la aldea que está ahora llena de llamas y cadáveres (FIGURA 2), donde el Barón Serpiente, antes señor de este pueblo y del castillo, ha mutado en un ser similar a una serpiente, ahora sediento de sangre.

Guts debe derrotarlo si quiere avanzar y escapar de ese lugar.



FIGURA 1 Mazmorra en la que está nuestro personaje al inicio



FIGURA 2 Salida del castillo para la batalla contra el jefe

La creación y desarrollo de un videojuego es capaz de juntar el arte en forma de animación, escenarios y paisajes creados a mano, con sistemas tan complejos como la Inteligencia Artificial de ciertos entes, funcionalidades, y leyes lógicas del entorno.

Las secciones en las que se dividirá el nivel serán 2: La zona inicial que servirá para familiarizarse con los controles de movimiento sorteando obstáculos y desplazándose horizontalmente, y la segunda y última situada a la salida del castillo y llegada a la aldea en la que te enfrentarás contra un jefe final. Cabe aclarar que la primera fase y la tercera son las más “inamovibles” en cuanto a la escala y lo que se quiere lograr, aunque ambas fases crecerán de forma paulatina en complejidad hasta la fase final del desarrollo.

2 PROYECTO - LA SOLUCIÓN A IMPLEMENTAR Y OBJETIVOS

El objetivo, como ya se ha mencionado, es realizar un nivel al más alto nivel en el tiempo dedicado al proyecto (300 horas), este proceso de producción abarca cinemáticas animadas en nivel, diseños artísticos y a nivel de programación del jugador, enemigos y jefe final, y aspecto y comportamiento del mundo del nivel.

Los objetivos a cumplir durante y con en el proyecto son los siguientes:

Creación de una Experiencia Visualmente Impactante

- Estética visual: Estilo oscuro y detallado de Berserk usando pixelart.
- Elementos estéticos: Inspirados en Blasphemous, con referencias religiosas y folclóricas.

Implementación de Mecánicas de Juego Desafiantes y Estratégicas

- Sistema de combate: Exigente y táctico, inspirado en Bloodborne y Dark Souls.
- Exploración y plataformas: Mecánicas metroidvania, interacción con el entorno y búsqueda de secretos.

Desarrollo de un Nivel Completo y Coherente

- Secciones de nivel: Dos secciones distintas con progresión de dificultad gradual.
- Estructura narrativa: Fidelidad a Berserk, proporcionando contexto significativo para la acción del juego.

Utilización Eficiente de Recursos y Tecnologías

- Motor gráfico y programación: Uso de Unity y C# para un desarrollo eficiente y colaborativo.
- Elementos artísticos: Creación en Procreate, combinando recursos propios con assets libres de derechos.

Cumplimiento de Estándares de Calidad y Rendimiento

- Pruebas de rendimiento y calidad: Garantizar funcionamiento en varios dispositivos y configuraciones.
- Feedback de terceros: Identificación y corrección de errores o mejoras en jugabilidad y experiencia del usuario (o en su defecto, que realice las pruebas el propio desarrollador).

3 TAREAS DEL PROYECTO

Como parte del replanteamiento de la planificación del proyecto, se añadió una metodología para clasificar por prioridad las tareas que eran básicas para nuestro proyecto, hasta llegar a aquellas que no supondría ninguna diferencia o poca en añadir valor al producto final. Esto ha sido posible gracias al método MoSCoW, tal y como se muestra en la tabla (FIGURA 7).

El método MoSCoW es una técnica de priorización utilizada en la gestión de proyectos para clasificar los requisitos o funcionalidades en función de su importancia. La palabra MoSCoW es un acrónimo que representa cuatro categorías de priorización:

Must Have (Debe tener): Son los requisitos o funcionalidades imprescindibles para el éxito del proyecto. Son elementos que se consideran esenciales y que deben ser entregados en la primera fase del proyecto.

Should Have (Debería tener): Son los requisitos o funcionalidades importantes pero no críticos para el éxito inicial del proyecto. Estos elementos se abordan después de los "Must Have", pero antes que los de menor prioridad.

Could Have (Podría tener): Son requisitos o funcionalidades deseables pero no esenciales. Si es posible, se incluirán en la entrega del proyecto, pero pueden ser pospuestos para futuras fases si es necesario.

Won't Have (No tendrá): Son requisitos o funcionalidades que se han identificado como no necesarios en la versión actual del proyecto. Estos elementos se pueden considerar para futuras iteraciones, pero no se abordarán en la fase actual.

El método MoSCoW permite al equipo del proyecto y a los interesados priorizar de manera efectiva qué requisitos son críticos y cuáles pueden ser pospuestos. Esto ayuda a garantizar que los recursos se asignen de manera óptima y que el proyecto se enfoque en la entrega de valor de manera incremental.

En el desarrollo de un videojuego es muy importante debido a que por falta de tiempo y fechas de entrega ajustadas suelen quedar algunas tareas sin ser implementadas a tiempo, por lo que es esencial priorizar antes las importantes que hacen funcionar y destacar a nuestro proyecto.

	TAREA	Categoría MoSCoW
1	Investigación y documentación sobre productos similares e inspiraciones externas	Must-have
2	Formación sobre Unity / C# / IA	Must-have
3	Formación Animación / Sprites y Tiles 2D	Should-Have
4	Realizar Sprites Jugador	Must-Have
5	Realizar Sprites Boss Final	Must-Have
6	Pixelart (Tiles) del Escenario y fondo	Could-Have
7	Sprites Enemigo Base y Arquero	Could-Have
8	Diseñar nivel	Must-Have
9	Programar movimiento del jugador (despl. horizontal, salto, rodar)	Must-Have
10	Programar ataques del Jugador (ataque débil, fuerte y especial)	Should-Have
11	Programar IA Enemigos: Base, Arquero y Final	Must-have
12	Diseñar interfaz	Should-Have
13	Añadir partículas al nivel (sangre, iluminación, tierra...)	Won't-Have
14	Añadir música y efectos de sonido al nivel	Could-Have
15	Diseñar menú inicial	Could-Have
16	Animar cinemática Inicial 2D	Won't-Have
17	Diseñar pantalla Final Partida (Victoria/Derrota)	Could-Have

FIGURA 3 Tabla de tareas con método MoSCoW

4 METODOLOGÍA

Tras investigar sobre metodologías de trabajo individuales que se adaptaran a la filosofía de este proyecto, se ha llegado a la conclusión de hacer uso de la metodología (adaptada al trabajo individual) Scrum.

Scrum permitirá, de forma iterativa, llegar antes y de una forma más controlada al core del nivel y de cada uno de sus aspectos: las etapas tempranas cubrirán las necesidades principales de funcionalidad y aspecto, para con cada iteración y llegando a las últimas etapas, se pulen y añaden complementos para enriquecer la experiencia del usuario, es decir, la calidad del producto final.

Al comienzo del proyecto se realizará una lista de tareas (product backlog) a realizar que forman al proyecto, ordenadas por prioridad y con una descripción.

Se trabajará en planificar en Sprint de 2 semanas de duración, en los que se especificará los aspectos en los que se trabajará en este periodo de esa lista. Al finalizar el Sprint, se recogerá el trabajo realizado del planificado para analizar si se ha podido cumplir dicho Sprint, y de forma satisfactoria. El trabajo faltante o revisable se desplazará al siguiente Sprint.

4.1 Jira: La herramienta para la planificación/seguimiento del proyecto

Para planificar los Sprints y tener controladas las tareas realizadas, realizándose y próximas a ser realizadas se utilizará la herramienta Jira

Jira es una herramienta de gestión de proyectos flexible y poderosa que ofrece seguimiento visual del progreso, colaboración efectiva, integraciones con herramientas populares, gestión avanzada de problemas y escalabilidad para adaptarse a cualquier tamaño de equipo o proyecto. Y es que Jira ofrece una gran variedad de formas de representar/organizar las tareas:

En forma de cronograma o diagrama de Gantt (ANEXO 1), Backlog, con etiquetas de cada tarea/objetivo según la prioridad o un tablero que muestra las tareas por hacer, las que están en curso, y las finalizadas.

4.2 MVP y las diferentes versiones del proyecto

Tras un periodo de tiempo de desarrollo y crecimiento del proyecto, se llegó a la conclusión de que el proyecto debía crecer, incrementando todos sus aspectos al mismo tiempo, pues estos dependían directa o indirectamente unos de otros.

Por poner un ejemplo, resultaría prácticamente imposible programar el ataque de nuestro personaje y su alcance exacto si no tiene una animación asignada que muestre el movimiento para calcular esos tiempos de impacto y rango de ataque que se programan en los scripts en C#, principalmente.

Es por esto que se deshace la planificación iniciada basada en Sprints con tareas diferentes asignadas anteriormente realizada, para dar paso a una nueva división del

proyecto en Sprints, pero basando todas las tareas y sus mejoras en versiones: Es decir, mientras que la versión 1 de la realización del escenario es la mínima para la navegación y finalización del nivel (MVP, es decir Minimum Viable Product), la versión 3 será la versión del escenario con la visión completa en lo general y en los detalles del nivel del proyecto en este apartado.

Expandiendo y formalizando la definición de Minimum Viable Product, es la versión más básica de un producto que aún ofrece valor al usuario. Se crea con la menor cantidad de recursos posible para validar la idea del producto y recopilar comentarios de los usuarios reales. El objetivo es aprender rápidamente y mejorar el producto en iteraciones futuras.

5 ESTADO DEL ARTE

Cómo en una novela, cómic o película, existen diferentes géneros para clasificar una obra: terror, thriller, comedia, romance... Esto también aplica a los videojuegos, pero además hay que añadir otro género que clasifica a este según la forma en la que este se juega o el jugador interactúa:

Cuando el videojuego te deja tomar decisiones y esas decisiones se convierten en un puzzle a resolver que afectará al desarrollo y desenlace del videojuego, esto se le denomina Aventura gráfica (similar a los libros de crear tu propia aventura).

Este proyecto se le ha clasificado con términos como Metroidvania (FIGURA 4) y Soulslike. Metroidvania es aquel juego que mezcla un juego de plataformas (Super Mario Bros.) añadiendo secciones de combate, habitualmente con espadas o similares.

El término proviene de los dos videojuegos que popularizaron este nuevo género: Metroid (propiedad de Nintendo) y Castlevania (propiedad de Konami)(FIGURA 3).

El concepto del proyecto es trasladar el primer capítulo del manga Berserk, de Kentaro Miura, titulado "El Guerrero Negro" a un nivel en 2D con estética pixelart centrado en el combate "con un estilo soulslike" y metroidvania con una mentalidad concreta: "Que tenga un acabado lo más profesional posible para vivir en todos los aspectos, tanto los artísticos como la lógica, del proceso de desarrollo de un videojuego".



FIGURA 4 Castlevania Symphony of the Night, desarrollado por Konami

Soulslike es atribuido a cualquier juego que siga unas normas parecidas en jugabilidad a los juegos desarrollados por FromSoftware y que comenzaron a popularizarse por la serie de juegos Dark Souls (una vez más, el género nace de un fragmento del nombre del videojuego): Un combate con una dificultad considerada elevada en comparación a otros juegos, un sistema que penaliza al jugador tras perder en el que el punto de control está alejado de donde se encontraba previamente, jefes finales con una estética muy trabajada y diferenciada (habitualmente basándose la estética en la fantasía oscura).

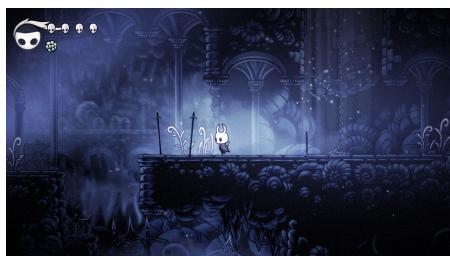


FIGURA 5 Metroidvania: Juego en 2D con saltos, exploración y combate

Gracias a la ambientación de fantasía oscura y gran diseño de personajes del material original, Berserk, el videojuego se verá beneficiado estéticamente, teniendo ya una identidad visual básica, y con una estructura narrativa del nivel ya definida. Además, se añaden referencias estéticas de otros juegos, tales como Blasphemous (FIGURA 5), un videojuego español del mismo género y estética que Berserk 2D, este mismo, basado en el folklore derivado de la religión y festividades en España, u otros que han sido tomados como pautas a la hora de diseñar las mecánicas de juego como Bloodborne (FIGURA 6) o Dark Souls, ambos juegos desarrollados por FromSoftware, en el que el videojuego gira en torno a la recompensa de un combate exigente y meticulosamente bien diseñado.



FIGURA 6 Blasphemous, desarrollado por The Game Kitchen



FIGURA 7 Bloodborne, desarrollado por FromSoftware

A la hora de decidir en qué motor gráfico se trabajaría, había muchas posibles opciones de entrada: Unreal Engine, el motor desarrollado por Epic, se encuentra ya en su quinta iteración, siendo, probablemente a día de hoy, el motor más utilizado dentro del desarrollo de videojuegos de gran producción en 3D, especialmente. Entrando ya en los motores especialmente centrados y populares en el desarrollo en 2D, encontramos 3 motores: GameMaker, Godot y Unity.

Se ha decidido desarrollar este videojuego en el motor Unity, por ser el ideal para el género y perspectiva 2D que tendrá el juego (el motor Unreal Engine de Epic se descartó por ser uno claramente enfocado al desarrollo en 3D), además de que dentro de este tipo de motores en 2D, es el que cuenta con una mayor comunidad. Ya sea la que forman desarrolladores, que han ido entregando grandes juegos como Hollow Knight (FIGURA 4), Cuphead, o el propio mencionado Blasphemous, también existe un gran grupo de desarrolladores “amateur” o centrados en la educación y formación del motor.

En consecuencia, a la hora de programar la lógica del nivel, se utilizará C# dentro del IDE Visual Studio Code, por la familiaridad que ya se tiene con este editor de código y la variedad de extensiones con las que cuenta para enriquecerlo aún más.

Para crear todos los aspectos “artísticos” del nivel se hará uso de la aplicación Procreate disponible en iPad. Cabe mencionar que para agilizar algunos procesos, se hará uso de assets libres de derechos, pudiendo incluso modificarlos para las necesidades de la producción.

6 PASOS DE IMPLEMENTACIÓN

6.1 Formación

Unity y C#

Se ha realizado un curso de 18.5 h en la plataforma Udemy, impartida por GameDev.tv Team llamado Complete C# Unity Game Developer 2D [4], en el que además de mostrar las capacidades del motor en diferentes proyectos, también se ha trabajado en C# para las funcionalidades de estos.

Regularmente, también se realizan consultas para el proyecto, a la documentación oficial del motor Unity, en foros de Reddit, en el propio foro de Unity o en algunos vídeos de YouTube.

Apartado artístico

Se han realizado consultas puntuales sobre las bases en el diseño de sprites y assets para el proyecto, para sacarle el máximo partido a la plataforma Procreate en el que se han realizado.

Para la animación 2D ha sido todo por iniciativa y conocimientos propios.

6.2 Inspiración/referencias y productos similares

Estilo Artístico: Pixelart Inspirado en Berserk y Blasphemous

El proyecto se adentra en el universo del manga Berserk, en particular, al adaptar el primer capítulo de esta obra concebida por Kentaro Miura. Inspirándose en la riqueza estética de este capítulo, los diseños de personajes y escenarios se nutren de la iconografía distintiva de este capítulo. La oscuridad temática y la brutalidad estilística de Berserk se tratan de traducir en el juego, creando así una experiencia fiel al material original.

Además, se busca enriquecer aún más la atmósfera del juego mediante la influencia estética del videojuego Blasphemous, desarrollado por The Game Kitchen, para ayudar a traducir el arte del manga ultrarrrealista a un arte píxel en 2D. Esta obra, con su particular visión de la fantasía oscura, aporta elementos religiosos y folclóricos, así como la representación de paisajes desolados y perturbadores, lo que resulta compatible visualmente con Berserk para inspirarse.

Mecánicas de Juego: Inspiradas en Bloodborne y Dark Souls

El diseño de las mecánicas de juego se inspira principalmente en Bloodborne y Dark Souls. Estas obras son reconocidas por su enfoque en el combate desafiante, y un trabajado diseño de jefes finales. Se prioriza la implementación de un sistema de combate desafiante y fluido, representando un desafío significativo que demande habilidad y táctica por parte del jugador.

6.3 Análisis de requisitos y selección de tecnologías y recursos

Se realiza un detallado análisis de requisitos/funcionalidades que tendrá el producto, tanto las de su aspecto como lógica del producto.

Es importante al inicio del proyecto escoger las tecnologías en las que se desarrollará el producto y la gran mayoría de recursos predefinidos para arrancar el proyecto (assets, efectos de sonido, plantillas de sprites... todo libre de derechos).

6.5 Diseño del jugador

Desarrollar todas las funcionalidades del jugador: desplazamiento, combate, vida y resistencia. Esto comprende tanto la parte de código que las hace posibles, con su interacción con el resto de elementos del nivel (terreno, gravedad, enemigos), como la creación de los sprites necesarios para representar estas funcionalidades.

Se han diseñado los sprites que forman todas las acciones y estados en los que se puede encontrar el personaje: Avanzar horizontalmente, ataque débil, ataque fuerte, el de estado idle (quieto), muerte y rodar.

Cada uno de estos sprites está asignado a su respectiva animación (ANEXO 2), que se activa o bien presionando una tecla asignada o bien por el entorno en el caso de muerte (fin de la partida).

Controles:

- **A/D:** Desplazamiento horizontal.
- **Space:** Salto.
- **Q:** Rodar.
- **Click Izquierdo:** Ataque Débil.
- **Click Derecho:** Ataque Fuerte.



FIGURA 8: Sprites del personaje controlable

6.6 Diseño de enemigos

Se ha trabajado en el enemigo final del nivel, convirtiendo al juego en un videojuego de tipo **Boss Rush** (es decir, un juego dónde los únicos enemigos a vencer son jefes).

Como el videojuego toma inspiración del primer capítulo de Berserk, el aspecto visual así como el set de movimientos del jefe final son del personaje *El Barón Serpiente* (FIGURA 9), un humano ahora transformado en una bestia humanoide con características de una serpiente, con quien el protagonista se enfrenta.

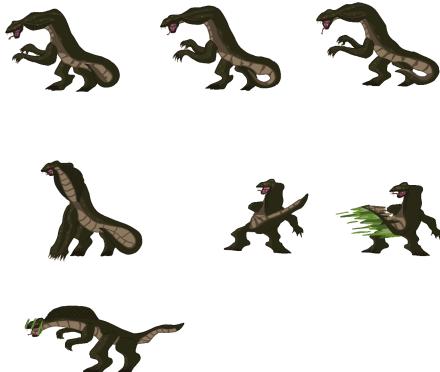


FIGURA 9: Sprites del jefe final inspirados en el manga Berserk

En cuanto a la lógica del enemigo, se han desarrollado una serie de comportamientos y toma de decisiones para una experiencia de combate contra el jugador: Un comportamiento que según la distancia entre este y el jugador realizará un ataque de corto alcance, de largo alcance o avanzará/voltará para acercarse al jugador, activando las correspondientes animaciones. También, estos ataques tienen un daño asignado (el de corto alcance 20, el de largo 10 de daño).

```
0 references
void Attack()
{
    Collider2D[] hitPlayer = Physics2D.OverlapCircleAll(attackController.position, attackRange);
    randomAttack = myAnimator.GetInteger("RandomAttack");
    snakeAttack1.Play();
    foreach(Collider2D player in hitPlayer)
    {
        if (player.CompareTag("Player") && !player.GetComponent<Player>().isRolling)
        {
            player.GetComponent<Player>().TakeDamage(attackDamage);
        }
    }
}

0 references
private void OnDrawGizmos()
{
    //Ataque de la cola = corta distancia
    Gizmos.color = Color.red;
    Gizmos.DrawWireSphere(attackController.position, attackRange);

    //Ataque mordisco = larga distancia
    Gizmos.color = Color.blue;
    Gizmos.DrawWireCube(attack2position.position, attack2dimension);

}

0 references
void Attack2()
{
    Collider2D[] hitPlayer = Physics2D.OverlapBoxAll(attack2position.position, attack2dimension, 0f);
    snakeAttack2.Play();
    foreach(Collider2D player in hitPlayer)
    {
        if (player.CompareTag("Player") && !player.GetComponent<Player>().isRolling)
        {
            player.GetComponent<Player>().TakeDamage(attack2Damage);
        }
    }
}
```

FIGURA 10: Extracto de código de la lógica de ataques del enemigo

El rectángulo azul corresponde a la zona de ataque del ataque de largo alcance, el círculo rojo el del ataque de corto alcance, y los elementos verdes son para la caja de colisiones con el jugador y el terreno (FIGURA 11).

Estos dos rangos de ataque son creados por código (IMAGEN 10) en la función llamada `OnDrawGizmos`. `Gizmos` es el objeto que “dibuja” el rango de estos dos ataques. Las funciones `Attack` y `Attack2` se llamarán según lo que el enemigo decida según la distancia y añadiendo un elemento de aleatoriedad con un número aleatorio que cambia desde el animator. El comportamiento de estas funciones es igual: Aquello que está dentro del rango de ataque y en el momento en el que se ejecuta el ataque, tenga el Tag “Player” y no esté rodando (momento en el que el jugador no puede recibir daño), activa la función de recibir daño del jugador con el daño del ataque por parámetro.

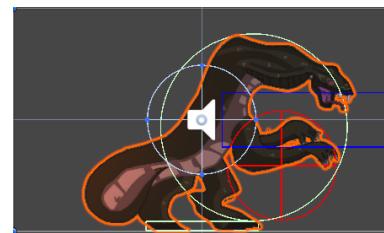


FIGURA 11: Geometría de colisiones e impactos de Boss Final

6.7 Diseño del nivel

Desarrollo de las dos secciones del nivel. Incluye la creación a boceto de este nivel y sus fases antes de implementarlo, creación y adquisición de todos los elementos visuales necesarios para representarlo, y desarrollo de la lógica y leyes del nivel. La complejidad del diseño del nivel será proporcional a la sección a la que pertenecería de un juego completo, en este caso es uno de introducción.

La primera consiste en la salida del calabozo con una sección de pinchos hasta llegar a la salida de este calabozo, momento en el que se llega a la siguiente zona.

La segunda zona consiste en una arena de combate con el jefe final, situada justo a la salida del castillo, al inicio de una aldea en llamas. Una vez derrotado este enemigo es posible finalizar el nivel llegando a la zona de la derecha de la arena.

Para estas zonas también se ha planificado la posición y límites que puede mostrar la cámara del jugador: Esta sigue al jugador y no puede superar según que límites para no romper con la experiencia del nivel de ver los límites del mapa.

6.8 UI

Se ha añadido una pantalla de inicio para acceder al nivel, una de derrota para reiniciar el nivel o volver al menú de inicio, y otra de victoria para volver al menú principal al finalizar el nivel.

Se ha añadido una barra de salud y otra de resistencia del jugador, además de una interfaz gráfica temática del juego para decorarlas (FIGURAS 11 y 12). Cuando la salud llega a 0, salta la pantalla de Game Over. La barra de resistencia se reduce cuando el jugador salta, rueda o ataca. Se va rellenando progresivamente cuando no realiza ninguna de las acciones anteriores.

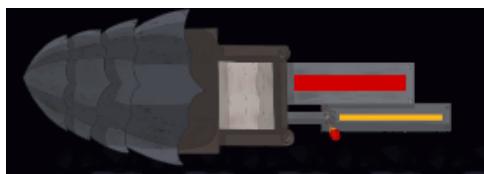


FIGURA 12: UI del jugador



FIGURA 13: UI del jefe final

Se ha añadido una barra de salud asociada al jefe final El Barón Serpiente, cuando esta llega a 0, el jugador puede progresar y finalizar en el nivel navegando hacia la derecha.

6.9 Música y efectos de sonido

Para el apartado sonoro del juego, que sería todo el conjunto de sonidos contextuales a las acciones del jugador y/o del enemigo, como otras ambientales como la música que acompaña cada nivel, se han recogido los siguientes añadidos:

- Música para los dos escenarios disponibles del nivel (mazmorra y exterior del castillo + aldea), siendo el tema Behelit de Susumu Hirasawa el que se puede escuchar en el primer escenario, y My Brother de Shiro Sagisu en el segundo escenario.
- Sonido ambiental en la mazmorra y en un rango de alcance se puede oír el caos y gritos desde la ventana de la celda de Guts (el personaje controlable) desde la que comienza el nivel.
- Un sonido del fuego añadido para todas las columnas con antorchas.
- Para los ataques de Guts hay un sonido diferente cuando da en el objetivo y cuando no.
- Un sonido de daño cuando el jugador es derrotado (salud=0).

- Un sonido de acompañamiento al movimiento de Guts "Rodar" (tecla Q).
- Un sonido diferente según el ataque del Barón Serpiente
- Un sonido para el Barón Serpiente cuando es derrotado.

6.10 Apartado Visual + Animación 2D

Todos lo que es el suelo y elementos sólidos de colisión con los que hace contacto tanto el jugador como el jefe final se han hecho en cuadrados de un mismo tamaño (128x128). El motivo es que unity ofrece la posibilidad de crear una "paleta de sprites", lo que hace que puedas agrupar sprites y añadirlos al nivel de forma rápida que si realizamos todo el nivel a mano y lo exportamos directamente a la carpeta de recursos de Unity, ofreciendo la posibilidad de resultado como en la FIGURA 13



FIGURA 14: Escenario del jefe final realizado con cuadrados que forman el terreno y otros añadidos

Por otro lado, Unity ofrece la capacidad de crear un emisor de partículas a gusto del desarrollador. Para Berserk 2D, se han creado estas partículas para diferentes propósitos:

- Partículas de humo en diferentes puntos de la arena para simular el efecto de las llamas en la aldea (FIGURA 14), así como en las antorchas del calabozo.
- Partículas de ceniza que sobrevuelan permanentemente la arena, además, de aparecer al inicio de la zona del calabozo. Estas partículas también se emiten en las zonas de llamas y humo (incluidas las antorchas).
- Polvo que levanta el jugador de forma constante al desplazarse horizontalmente, se incrementa momentáneamente al saltar y más aún al rodar.



FIGURA 15: Resultado final de diversas partículas de humo y chispas al escenario

Por último, al clicar el jugador en “Iniciar Partida” en el menú principal, se reproducirá una cinemática previa hecha enteramente en 2D para sumergir al jugador en el contexto del nivel a través de diversos fotogramas animados con distintas técnicas, desde movimientos de cámara (FIGURAS 14 y 15), como edición en la iluminación de la escena, todo con el fin de dar movimiento e inmersión. Esta escena se ha realizado con Procreate y Adobe Premiere para el montaje de sonido y vídeo.



FIGURA 16: Fotograma perteneciente al segundo corte de la animación



FIGURA 17: Fotograma perteneciente al tercer corte de la animación

7 Despliegue, Testing y Validación

El despliegue consiste en la compilación de todo el producto y realización de un simulacro y prueba completa del nivel a través de un ejecutable (.exe) que se crea al crear esta build, es decir, versión jugable del proyecto. Se han realizado en varios ordenadores el proyecto compilado en una serie de archivos y con un ejecutable asociado, para asegurar su portabilidad y rendimiento óptimo en multidispositivos.

Es imprescindible que un juego sea probado antes de ser considerado como un producto finalizado. Al igual que en el cine para ciertas películas se reúne a una audiencia para ver antes el largometraje para poder cambiar ciertos aspectos de la película y tratar de asegurar su éxito, en un videojuego se procede de forma similar: Ya sean tanto desarrolladores como personas terceras al desarrollo, se realizan constantes pruebas para asegurar que no haya errores en ciertas partes, y que otras sean simplemente divertidas, desde el movimiento del jugador o el propio combate y navegación en el nivel. Algunas veces, por la dificultad del juego, se deben de hacer muchas pruebas para que sea superable por una cantidad amplia de jugadores, teniendo que ajustar varios parámetros.

Yo mismo he hecho de tester al repetir múltiples veces durante todas las etapas del desarrollo del juego, desde las primeras etapas en Unity, pasando por el MVP, hasta el considerado hasta su testeo como la versión final.

Por ejemplo, al añadir la posibilidad al personaje de subir escaleras, se observó que el jugador era capaz de atravesarlas, así como también de atravesar paredes de la misma forma debido a que la cápsula de colisiones del jugador no estaba alineada con sus pies. También otros que perjudicaban más a la satisfacción del juego como errores en la caja de impactos del jefe final y del rango de sus ataques, o que los sprites sufrían cortes o no estaban bien alineados entre sí para su máxima fluidez, además que se añadieron otros como transición al finalizar ciertas acciones de ambos personajes.

Por último, en cuanto a cuestiones de rendimiento, se limitó el número máximo de fotogramas por segundo a 60, ya que de lo contrario podría tanto ocasionar problemas a equipos de menores prestaciones, como un esfuerzo innecesario a los de gama alta, así como creaba problemas a la hora de representar los fotogramas y velocidad de los personajes al superar esos 60 fps. Además, se ajustó la duración de archivos de audio para reducir el tiempo la carga del nivel para que no supere los 20 segundos.

9 CONCLUSIÓN

Este proyecto ha sido todo un reto de realizar. Esto se debe principalmente a la gran, y poco dimensionable, carga de trabajo, por lo abstracto y subjetivo del proyecto, que hacían difícil de saber cuando una sección del proyecto estaba finalizada o cuando era necesario volver al código o retrabajar los sprites, o que elementos debían quedar fuera del desarrollo por falta de tiempo o coherencia con el resto del videojuego. Salvo por esa cierta incertidumbre, debo decir que lo he disfrutado mucho. Desde pequeño siempre me ha interesado la producción audiovisual: me gustaba dibujar, leer, escribir, narrar y grabar para contar algo, y por supuesto, los videojuegos han formado parte de mi niñez, adolescencia, y entrada en adultez, y conforme iba creciendo, me iba preguntando cómo eran capaces algunas personas de crear un videojuego.

Independientemente del resultado final del proyecto, del cual hay que decir que estoy plenamente satisfecho, ante todo, se ha logrado responder a esa pregunta que da forma al título de este proyecto: Así es el proceso de producción de un videojuego.

AGRADECIMIENTOS

Quiero agradecer a mi familia y amigos por el apoyo que me han dado en los momentos complicados del proyecto y en mi día a día, siendo vitales para llegar hasta el final de este proyecto.

También quería agradecer a mi tutor por la excelente supervisión del proyecto, además de la cordialidad y facilidades con la que se han realizado las reuniones de seguimiento y el proyecto en cuestión.

IKER ROA AMEIGIDEZ: Proceso de producción de un videojuego: Berserk 2D

APÉNDICE

A1. DIAGRAMA DE GANTT DE PLANIFICACIÓN DEL PROYECTO

	MAR	ABR	MAY	JUN
✓ B20-46 Versión 1 ("Sprint 1")				
✓ B20-47 Formación sobre Unity / C# / IA / Animación, Spr... FINALIZADA				
✓ B20-47 Investigación y documentación sobre productos s... FINALIZADA				
✓ B20-48 v1.0 Sprites Boss Final FINALIZADA				
✓ B20-48 v1.0 Pixelart Escenario y Fondo FINALIZADA				
✓ B20-49 v1.0 Sprites jugador FINALIZADA				
✓ B20-49 v1.0 Pantalla de Inicio FINALIZADA				
✓ B20-49 v1.0 Diseño del nivel FINALIZADA				
✓ B20-49 v1.0 Movimiento jugador (despl., salto, rodar) FINALIZADA				
✓ B20-49 v1.0 Ataques del jugador (débil, fuerte y especial) FINALIZADA				
✓ B20-49 v1.0 UI FINALIZADA				
✓ B20-49 v1.0 IA Enemigo Final, Base y Arquero FINALIZADA				
✓ B20-49 v1.0 Pantalla de Game Over FINALIZADA				
✓ B20-47 Versión 2 ("Sprint 2")				
✓ B20-49 v2.0 Movimiento jugador (despl., salto, rodar) FINALIZADA				
✓ B20-49 v2.0 Ataques del jugador (débil, fuerte y especial) FINALIZADA				
✓ B20-49 v2.0 Pixelart Escenario y Fondo FINALIZADA				
✓ B20-49 v2.0 UI FINALIZADA				
✓ B20-49 v2.0 Diseño del nivel FINALIZADA				
✓ B20-49 v2.0 IA Enemigo Final, Base y Arquero FINALIZADA				
✓ B20-49 v2.0 Sprites jugador FINALIZADA				
✓ B20-49 v1.0 Partículas FINALIZADA				
✓ B20-49 v1.0 Música y efectos de sonido FINALIZADA				
✓ B20-49 v2.0 Sprites Boss Final FINALIZADA				
✓ B20-49 v1.0 Sprites Enemigos Base y Arquero FINALIZADA				
✓ B20-48 Versión 3 ("Sprint 3")				
✓ B20-49 v3.0 Movimiento jugador (despl., salto, rodar) FINALIZADA				
✓ B20-49 v3.0 Sprites Boss Final FINALIZADA				
✓ B20-49 v3.0 Sprites jugador FINALIZADA				
✓ B20-49 v3.0 Diseño del nivel FINALIZADA				
✓ B20-49 v3.0 Pixelart Escenario y Fondo FINALIZADA				
✓ B20-49 v3.0 IA Enemigo Final, Base y Arquero FINALIZADA				
✓ B20-49 v2.0 Música y efectos de sonido FINALIZADA				
✓ B20-49 v3.0 Ataques del jugador (débil, fuerte y especial) FINALIZADA				
✓ B20-49 v2.0 Partículas FINALIZADA				
✓ B20-49 v2.0 Pantalla de Game Over FINALIZADA				
✓ B20-49 v2.0 Pantalla de Inicio FINALIZADA				
✓ B20-49 v1.0 Cinematografía inicial 2D FINALIZADA				

A2. DIAGRAMA DEL ANIMATOR DEL JEFE FINAL ‘BARÓN SERPIENTE’

