

# Proceso de producción de un videojuego: Berserk 2D

Iker Roa Ameigidez

**Resum**—Resum del projecte, màxim 10 línies. ....

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Paraules clau**—Paraules clau del projecte, màxim 2 línies. ....

---

---

**Abstract**—Versió en anglès del resum. ....

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Index Terms**—Versió en anglès de les paraules clau. ....

---

---



Mes" de 2024, Escola d'Enginyeria (UAB)

## 1 INTRODUCCIÓN - CONTEXTO DEL TRABAJO

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.

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”.

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).

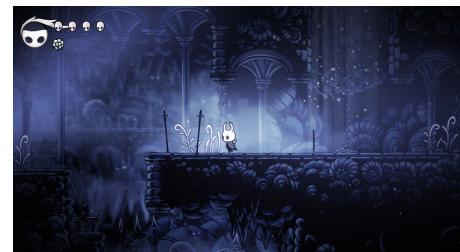
Este proyecto se le ha clasificado con términos cómo Metroidvania y Soulslike. Metroidvania es aquel juego que mezcla un juego de plataformas (Super Mario Bros.) añadiendo secciones de combate, habitualmente con espadas o similares.

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



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).



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, 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](#) 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.



Blasphemous, desarrollado por The Game Kitchen



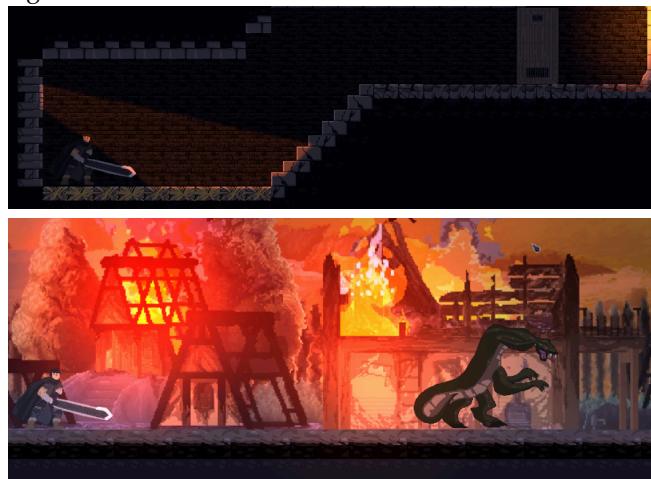
Bloodborne, desarrollado por FromSoftware

Las secciones en las que se dividirá el nivel serán 3: La zona inicial que servirá para familiarizarte con los controles enfrentándote al enemigo que custodia la celda. La salida del castillo y llegada a la aldea en la que te enfrentarás a más enemigos de tipo base y arqueros. Y la última que será un enfrentamiento 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 las 3 fases crecerán de forma paulatina en complejidad hasta la fase final del desarrollo.

La estructura narrativa es la siguiente: Guts, nuestro guerrero protagonista, apodado el guerrero negro, tras ser

encerrado en una mazmorra por los soldados de la realeza, 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 Puck, su “compañero” elfo, le ha 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, 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.



A la hora de decidir en qué motor 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. Por motivos que más adelante se expondrán, finalmente se decidió trabajar en Unity.

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.

## 2 PROYECTO - LA SOLUCIÓN A IMPLEMENTAR

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.

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](#), 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.

## 3 OBJETIVOS DEL PROYECTO

### 3.1 Creación de una Experiencia Visualmente Impactante

Desarrollar una estética visual fiel al estilo oscuro y detallado del primer capítulo de Berserk, utilizando pixelart para los personajes, escenarios y paisajes del juego.

Integrar elementos estéticos inspirados en [Blasphemous](#) para enriquecer la atmósfera del juego con referencias religiosas y folclóricas.

### 3.2 Implementación de Mecánicas de Juego Desafiantes y Estratégicas

Diseñar un sistema de combate exigente y táctico, inspirado en los estándares de calidad de [Bloodborne](#) y Dark Souls, que proporcione una experiencia desafiante y gratificante para el jugador.

Integrar mecánicas de exploración y plataformas características de los juegos metroidvania, promoviendo la interacción con el entorno y la búsqueda de secretos.

### 3.3 Desarrollo de un Nivel Completo y Coherente

Crear tres secciones de nivel distintas, cada una con su propio diseño y desafíos, que sigan una progresión de dificultad gradual y coherente.

Establecer una estructura narrativa que guíe al jugador a través del nivel, manteniendo la fidelidad al material original de Berserk y proporcionando un contexto significativo para la acción del juego.

### 3.4 Utilización Eficiente de Recursos y Tecnologías

Emplear el motor gráfico Unity y el lenguaje de programación C# para el desarrollo del juego, aprovechando la comunidad y las herramientas disponibles para garantizar un proceso de desarrollo eficiente y colaborativo.

Hacer uso de la aplicación Procreate para la creación de los elementos artísticos del juego, combinando recursos propios con assets libres de derechos para agilizar el proceso de producción.

### 3.5 Cumplimiento de Estándares de Calidad y Rendimiento

Realizar pruebas exhaustivas de rendimiento y calidad para garantizar que el juego funcione correctamente en una variedad de dispositivos y configuraciones, cumpliendo con requisitos mínimos de rendimiento y tiempos de carga aceptables.

Incorporar feedback de terceros durante el proceso de desarrollo para identificar y corregir posibles errores o mejoras en la jugabilidad y la experiencia del usuario.

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.

## 4 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 supondrían 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](#).

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.

	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	Could-Have

	nivel	
15	Diseñar menú inicial	Could-Have
16	Animar cinemática Inicial 2D	Won't-Have
17	Diseñar pantalla Final Partida (Victoria/Derrota)	Could-Have

## 5 METODOLOGIA

Tras investigar sobre metodologías de trabajo individuales que se adaptasen 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 sana al core del nivel y de cada uno de sus aspectos: las etapas tempranas cubrirían las necesidades principales de funcionalidad y aspecto, para con cada iteración y llegando a las últimas etapas, se pulirían y añadirían 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.

### 5.1 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, 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.

### 5.2 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 desecha 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), 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 MVP, 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.

## 6 PASOS DE IMPLEMENTACIÓN

### 6.1 Formación

#### 6.1.1 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, 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.

#### 6.1.2 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

### 6.2.1 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 ultrrealista a un arte píxel en 2D. Esta obra, con su visión particular 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.

### 6.2.2 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, representar un desafío significativo que demande habilidad y táctica por parte del jugador.

## 6.3 Análisis de requisitos

Detallado análisis de requisitos/funcionalidades que tendrá el producto, tanto las de aspecto como lógica del producto.

## 6.4 Selección de tecnologías y recursos predefinidos

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

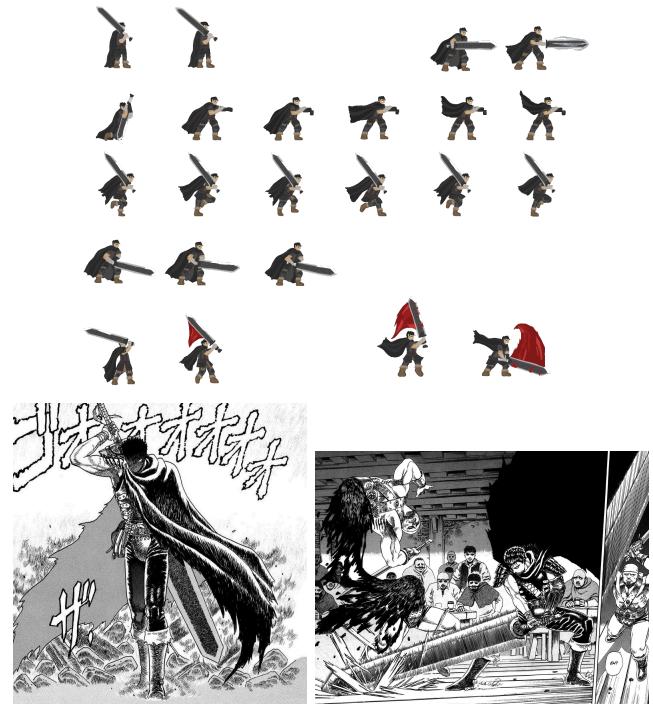
Desarrollar todas las funcionalidades del jugador: desplazamiento, combate, vida y resistencia. Esto comprende tanto la parte de código que las hace posibles, con su interacción con el resto de elementos del nivel (terreno, gravedad, enemigos), como la creación de los sprites necesarios para representar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

Se ha diseñado los sprites básicos del personaje controlable: los de avanzar horizontalmente, ataque débil, ataque fuerte, el de estado idle (quieto), muerte y rodar.

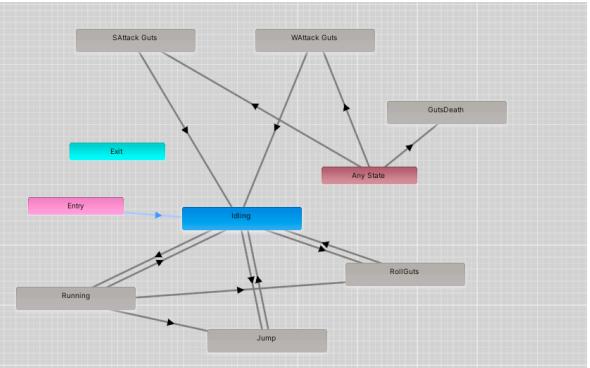
Cada uno de estos sprites está asignado a su respectiva animación, que se activa o bien presionando una tecla asignada o bien por el entorno en el caso de muerte (fin de la partida).

Controles:

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



(Sprites sujetos a cambios y añadidos)



Animator del personaje controlable

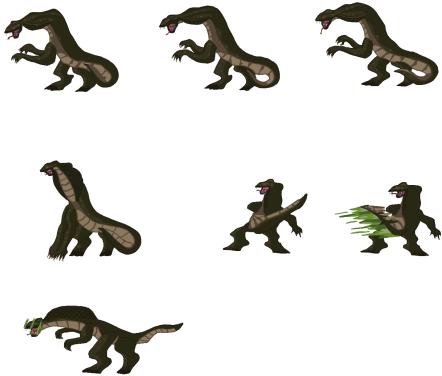
## 6.6 Diseño de enemigos

Estos para este primer nivel tendrían dos variantes: base y arquero. Se disponen de estas dos variantes, ya que el diseño del nivel (posteriormente se volverá a explicar) sería enfocado a ser uno más de introducción a un hipotético juego completo, además de ser uno especialmente enfocado en los "jefes finales". Se contempla también como diseño de enemigos, lo mencionado en la sección del jugador a desarrollar.

Se ha trabajado en el enemigo final del nivel, convirtiendo al juego temporalmente en un videojuego de tipo *Boss Rush* (es decir, un juego donde 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](#), un humano ahora transformado en una bestia humanoide con características de una serpiente, con quien el protagonista se enfrenta.



(Sprites sujetos a cambios y añadidos)

En cuanto a la lógica del enemigo, hasta ahora se ha realizado lo básico 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/volteará 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);
        }
    }
}

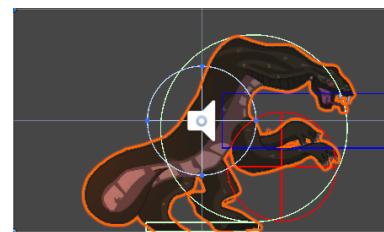
0 references
private void OnDrawGizmos()
{
    //Ataque de la cola = corta distancia
    Gizmos.color = Color.red;
    Gizmos.DrawWireSphere(attackController.position, attackRange);

    //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);
        }
    }
}
```

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.

Estos dos rangos de ataque son creados por [código](#) (el fragmento de código anterior) 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 este 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.



(Objeto sujeto a cambios y añadidos)

## 6.7 Diseño del nivel

Desarrollo de las tres 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.

Hasta ahora se ha desarrollado la primera versión del nivel: Salida del calabozo con una sección de pinchos para llegar directamente a la arena de combate con el jefe final. Una vez derrotado es posible finalizar el nivel llegando a la zona de la derecha de la arena.

## 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. 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.

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

Se ha añadido 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.

Se ha añadido 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.

Se ha añadido para todas las columnas con antorchas un sonido del fuego.

Se ha añadido para los ataques de Guts un sonido diferente cuando da en el objetivo y cuando no. También se ha añadido un sonido de daño cuando nos derrotan.

Se ha añadido para los ataques del Barón Serpiente un sonido diferenciado, así como uno cuando es derrotado.

## 6.10 Animación 2D

Realización de una cinemática previa hecha enteramente en 2D para sumergir al jugador en el contexto del nivel.



(Fotograma sujeto a cambios)



(Fotograma sujeto a cambios)

## 7 Despliegue

Compilación de todo el producto y realización de un simulacro y prueba completa del nivel.

## 8 TESTING Y VALIDACIÓN

### 8.1 Pruebas y control de calidad

Pruebas unitarias y de integración para asegurar el cumplimiento de los requisitos. Feedback de terceros que prueben el nivel y del propio desarrollador.

Se han corregido errores al subir las escaleras del personaje controlable que las atravesaba, atravesar paredes.

Se han corregido errores en la caja de impactos del jefe final y del rango de sus ataques.

### 8.2 Pruebas de rendimiento

Los objetivos incluyen alcanzar mínimo los 60 fotogramas por segundo en equipos categorizados como gama media-baja, que el uso de recursos del equipo no superen el 85% y que la carga del nivel no supere los 120 segundos.

## BIBLIOGRAFÍA

- [1] "BEST AGILE PRACTICES IN GAME DEVELOPMENT", STARLOOP STUDIOS A MAGIC MEDIA COMPANY,  
<https://starloopstudios.com/best-agile-practices-in-game-development/>

- [2] MIKE COHN, "AGILE AND SCRUM FOR VIDEO GAME DEVELOPMENT", MOUNTAIN GOAT SOFTWARE, <https://www.mountaingoatsoftware.com/presentations/agile-and-scrum-for-video-game-development>, 16 DE JUNIO DE 2012
- [3] ZAFER ELCIK, "APPLYING THE LEAN METHODOLOGY TO GAME DEVELOPMENT", <https://www.linkedin.com/pulse/applying-lean-methodology-game-development-zafer-elcik/>, 18 DE MARZO DE 2019
- [4] RICK DAVIDSON, GARY PETTIE, "COMPLETE C# UNITY GAME DEVELOPER 2D", GAMEDEV.TV TEAM,
- [5] "HECHO CON GAME MAKER", GAME MAKER TM, <https://gamemaker.io/es/showcase>
- [6] "FUNCIONES DE GAME MAKER", GAME MAKER TM, <https://gamemaker.io/es/features>
- [7] GODOT FOUNDATION, "WHY GODOT IS RIGHT FOR YOU" <https://godotengine.org/features/>

### 3 CONCLUSIÓ

.....  
.....  
.....  
.....

### AGRAÏMENTS

.....  
.....  
.....  
.....

NOM ESTUDIANT: TÍTOL TREBALL (ABREUJAT SI ÉS MOLT LLARG) 3

### APÈNDIX

#### A1. SECCIÓ D'APÈNDIX

.....  
.....  
.....  
.....

#### A2. SECCIÓ D'APÈNDIX

.....  
.....  
.....  
.....

## ANEXO: PLANIFICACIÓN - DIAGRAMA DE GANTT

