Neiva, 22 de noviembre de 2021

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Ciudad

El (Los) suscrito(s):

JONATAN HUERGO AGUILAR _____, con C.C. No. 1075291397_____,

LEONARDO AUGUSTO TAFUR GONZALEZ _____, con C.C. No. 1130744002_____,

_____, con C.C. No. _____,

_____, con C.C. No. _____,

Autor(es) de la tesis y/o trabajo de grado o _____

titulado _ COQUERSOULS 0.3.1 BASADO EN LA GENERACIÓN PROCEDIMENTAL DE MUNDOS Y AGENTE INTELIGENTE DE AUMENTO DE DIFICULTAD DE NIVEL_____

_____

_____

presentado y aprobado en el año __2021___ como requisito para optar al título de

__INGENIERO DE SOFTWARE_____;

Autorizo (amos) al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales “open access” y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que, de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, “Los derechos morales sobre el trabajo son propiedad de los autores” , los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE:

Firma: _____

EL AUTOR/ESTUDIANTE:

Firma: Jonatan Huergo _____

TÍTULO COMPLETO DEL TRABAJO: COQUERSOULS 0.3.1 BASADO EN LA GENERACIÓN PROCEDIMENTAL DE MUNDOS Y AGENTE INTELIGENTE DE AUMENTO DE DIFICULTAD DE NIVEL

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
HUERGO AGUILAR TAFUR GONZÁLEZ	JONATAN LEONARDO AUGUSTO

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
ROJAS ROJAS	FERNANDO

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre
N/A	N/A

PARA OPTAR AL TÍTULO DE: INGENIERO DE SOFTWARE

FACULTAD: INGENIERÍA

PROGRAMA O POSGRADO: PROGRAMA DE INGENIERA DE SOFTWARE

CIUDAD: NEIVA **AÑO DE PRESENTACIÓN:** 2021 **NÚMERO DE PÁGINAS: 69**

TIPO DE ILUSTRACIONES (Marcar con una **X**):

Diagramas___ Fotografías___ Grabaciones en discos___ Ilustraciones en general___ Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos___ Retratos___ Sin ilustraciones___ Tablas o Cuadros__

SOFTWARE requerido y/o especializado para la lectura del documento: N/A

MATERIAL ANEXO: N/A

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*:

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

	Español	Inglés
1.	**Agente inteligente**	**Intelligent agent**
2.	**Generación procedimental de contenido**	**Procedural generation of content**
3.	**Motor de videojuego**	**Video game engine**
4.	**Videojuego 2D**	**2D video game**

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

Este proyecto está direccionado a la creación del videojuego COQUERSOULS 0.3.1 bajo la filosofía de Demon´s Souls, basado en generación procedimental de contenido y agente inteligente que aumenta progresivamente el nivel de dificultad, dependiendo de las habilidades del jugador durante la ejecución de la partida.

COQUERSOULS 0.3.1 se desarrolló en Godot 3.3.4 que es un motor de videojuegos de código abierto, liviano y multiplataforma. Además, provee un lenguaje de programación para la creación de contenido de alto nivel similar a Python llamado GDScript y soporta C++, C#, VisualScript, haciéndolo flexible y óptimo para la fabricación de cualquier tipo de videojuego 2D o 3D.

En el proceso de construcción de COQUERSOULS 0.3.1 se usó la metodología ágil XP que dentro de sus ventajas se destaca el fomento de la comunicación entre los desarrolladores y el cliente, la mejora continua a través de la corrección periódica de fallas y la efectividad del proceso que conlleva a que sea un desarrollo rápido y económico.

El código fuente del videojuego quedará disponible a la comunidad universitaria en general, como aporte académico para promover el desarrollo de nuevas versiones de COQUERSOULS y profundizar a través de proyectos de investigación en la implementación de nuevas técnicas o soluciones de Inteligencia artificial en la producción de nuevos videojuegos de cualquier tipo.

ABSTRACT: (Máximo 250 palabras)

This project is directed to the creation of the video game COQUERSOULS 0.3.1 under the philosophy of Demon's Souls, based on procedural generation of content and intelligent agent that progressively increases the level of difficulty, depending on the player's skills during the execution of the game.

COQUERSOULS 0.3.1 was developed in Godot 3.3.4, which is an open source, light and multiplatform videogame engine. Besides, it provides a programming language for the creation of high-level content similar to Python called GDScript and supports C++, C#, VisualScript, making it flexible and optimal for the production of any type of 2D or 3D videogame.

In the construction process of COQUERSOULS 0.3.1 the agile methodology XP was used, which within its advantages stands out the promotion of the communication between the developers and the client, the continuous improvement through the periodic correction of failures and the effectiveness of the process that leads to a fast and economic development.

The source code of the videogame will be available to the university community in general, as an academic contribution to promote the development of new versions of COQUERSOULS and to deepen through research projects in the implementation of new techniques or solutions of Artificial Intelligence in the production of new videogames of any kind.

APROBACION DE LA TESIS

Nombre Presidente Jurado: N/A

Firma:

Nombre Jurado: JORGE ELIECER MARTINEZ GAITÁN

Firma:

Nombre Jurado: YHON JERSON ROBLES PUENTES

Firma:

COQUERSOULS 0.3.1 BASADO EN LA GENERACIÓN PROCEDIMENTAL DE MUNDOS Y AGENTE INTELIGENTE DE AUMENTO DE DIFICULTAD DE NIVEL

JONATAN HUERGO AGUILAR

LEONARDO AUGUSTO TAFUR GONZÁLEZ

UNIVERSIDAD SURCOLOMBIANA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SOFTWARE

NEIVA

2021

COQUERSOULS 0.3.1 BASADO EN LA GENERACIÓN PROCEDIMENTAL DE MUNDOS Y AGENTE INTELIGENTE DE AUMENTO DE DIFICULTAD DE NIVEL

JONATAN HUERGO AGUILAR

LEONARDO AUGUSTO TAFUR GONZÁLEZ

Proyecto de grado para optar el título de ingeniero de software

DIRECTOR

FERNANDO ROJAS ROJAS Mg.

Jefe de Programa Ingeniería de Software

UNIVERSIDAD SURCOLOMBIANA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SOFTWARE

NEIVA

2021

CONTENIDO

LISTA DE TABLAS

LISTA DE IMÁGENES

1 RESUMEN

Este proyecto está direccionado a la creación del videojuego COQUERSOULS 0.3.1 bajo la filosofía de Demon´s Souls, basado en generación procedimental de contenido y agente inteligente que aumenta progresivamente el nivel de dificultad, dependiendo de las habilidades del jugador durante la ejecución de la partida.

COQUERSOULS 0.3.1 se desarrolló en Godot 3.3.4 que es un motor de videojuegos de código abierto, liviano y multiplataforma. Además, provee un lenguaje de programación para la creación de contenido de alto nivel similar a Python llamado GDScript y soporta C++, C#, VisualScript, haciéndolo flexible y óptimo para la fabricación de cualquier tipo de videojuego 2D o 3D.

En el proceso de construcción de COQUERSOULS 0.3.1 se usó la metodología ágil XP que dentro de sus ventajas se destaca el fomento de la comunicación entre los desarrolladores y el cliente, la mejora continua a través de la corrección periódica de fallas y la efectividad del proceso que conlleva a que sea un desarrollo rápido y económico.

El código fuente del videojuego quedará disponible a la comunidad universitaria en general, como aporte académico para promover el desarrollo de nuevas versiones de COQUERSOULS y profundizar a través de proyectos de investigación en la implementación de nuevas técnicas o soluciones de Inteligencia artificial en la producción de nuevos videojuegos de cualquier tipo.

2 INTRODUCCIÓN

El principal propósito del presente proyecto de grado es el de crear un videojuego 2D de género soulslike y código abierto versión demo, basado en la generación procedimental de contenido y agente inteligente de dificultad de nivel, como requisito para la obtención del título de Ingeniero de Software.

En la historia reciente se evidencia como el uso de la inteligencia artificial ha revolucionado la industria de los videojuegos, generando experiencias de juego más reales y mejorando factores relacionados con la calidad gráfica, funcionalidad y optimización de recursos, en especial aquellos que están asociados a la limitación de recursos de hardware, el cual ese uno de los mayores obstáculos a la hora de desarrollar videojuegos.

Las técnicas de generación procedimental de contenido semi-aleatorio, usadas en videojuegos se han vuelto populares, puesto que producen situaciones de juego que pueden llegar a ser diferentes o únicas en cada partida y varían los niveles de dificultad según la habilidad del jugador, logrando de esta manera una mejor jugabilidad y un menor costo computacional.

Para la creación de COQUERSOULS 0.3.1 a través de la metodología de desarrollo de software XP, se usó el motor de videojuego GODOT que, a diferencia de otros motores, tales como Unity o Unreal, es más liviano, gratuito, multiplataforma de código abierto y que proporciona un gran conjunto de herramientas comunes para desarrollar videojuegos de cualquier tipo, de una manera fácil y eficiente.

Por último, el código fuente de COQUERSOULS 0.3.1 quedará disponible a la comunidad universidad para su mejoramiento, optimización, creación de nuevos videojuegos de género soulslike o como recurso para futuros proyectos de investigación de inteligencia artificial en el área de los videojuegos.

3 JUSTIFICACIÓN.

Con el desarrollo de este proyecto, se busca crear una versión Demo de videojuego 2D del género soulslike, usando algoritmos de generación procedimental y un motor de videojuegos liviano multiplataforma de código abierto que permitan minimizar las deficiencias en optimización de videojuegos 2D que se aprecia en la falta de capacidad en el aprovechamiento eficiente de los procesadores de más de cuatro núcleos y ocho hilos.

Sumado a lo anterior, se pretende disminuir las limitaciones de requerimientos de hardware y software indispensables para un rendimiento eficiente de este tipo de videojuegos, de modo que pueda ser escalable, funcional y con buena experiencia de juego, dando como resultado la satisfacción de las necesidades y exigencias del jugador en materia de jugabilidad.

Además, en lo concerniente al costo computacional que produce esta clase de videojuegos, se desea disminuir aquellas dificultades ligadas a la construcción de representaciones gráficas de objetos, generación de contenido multimedia y aumento de la calidad gráfica requerida, con el fin de minimizar el consumo de memoria y espacio en disco duro.

4 FORMULACIÓN DEL PROBLEMA.

En la actualidad muchos videojuegos 2D de distintos géneros, (incluyendo el género soulslike) que cuentan con excelente apartado gráfico presentan deficiencias en optimización que se ve reflejada en la incapacidad de aprovechar eficientemente procesadores de más de cuatro núcleos y ocho hilos, producto del bajo uso de la CPU.

Adicionalmente, las limitaciones de los requerimientos mínimos de hardware necesarios para que el videojuego corra tanto en pc como en consolas, hacen que este presente constantemente fallas de funcionalidad, generando bajos índices de jugabilidad y escalabilidad que se evidencia en una mala experiencia de juego.

Así mismo, al desarrollar videojuegos 2D de género soulslike se evidencia inconvenientes relacionados con el costo computacional tales como: la construcción de representaciones gráficas de objetos con recursos limitados, aumento de la calidad gráfica requerida, la complejidad creciente de generación de contenido multimedia (personajes, niveles, imágenes de textura, entre otros) que logren recrear escenas de hecho, que conllevan al alto consumo de memoria y espacio en el disco duro.

¿Cómo desarrollar un videojuego 2D de género soulslike con calidad gráfica óptima y buena experiencia de juego de tal manera que los recursos mínimos computacionales necesarios para su funcionamiento no sean un impedimento?

5 OBJETIVOS

5.1 Objetivo General

Crear un Videojuego 2D versión Demo de tipo soulslike, basado en la generación procedimental de mundos e implementación de agente inteligente de aumento progresivo del grado de fortaleza de los enemigos a vencer en un nuevo nivel.

5.2 Objetivos Específicos

- Analizar la filosofía del género soulslike para videojuegos 2d y los requisitos de funcionalidad.
- Diseñar videojuego 2D versión demo de género soulslike.
- Implementar algoritmo de generación procedimental de contenido de mundos.
- Emplear un agente inteligente de aumento progresivo de dificultad en niveles.

6 ALCANCES Y LIMITACIONES.

El presente proyecto contempla la creación de un Videojuego 2D versión Demo de género soulslike, basado en la generación aleatoria de mundos acorde a lo seleccionado por el jugador y aplicación de un algoritmo que aumente progresivamente la fortaleza de los enemigos a vencer en un nuevo nivel, considerando el tipo de jugador (personaje), comportamiento, cantidad de ataques, planeación de movimientos en el nivel anterior y el nivel de fortaleza del enemigo principal.

7 ANTECEDENTES

El género Soulslike se origina en Japón, fundamentado en el género Metroidvania, perteneciente al subgénero de videojuego de acción-aventura, el cual está basado en un concepto de plataformas no lineal [1] y la serie de videojuegos de acción-aventura The Legend of Zelda, creada por los diseñadores japoneses Shigeru Miyamoto y Takashi Tezuka y desarrollada por Nintendo Su trama por lo general describe las heroicas aventuras del joven guerrero Link, que debe enfrentarse a peligros y resolver acertijos para ayudar a la Princesa Zelda a derrotar a Ganondorf y salvar su hogar, el reino de Hyrule. [2]

Los juegos Soulslike generalmente tienen elementos comunes como alta dificultad, combates de alto riesgo con enemigos contundentes, puntos de control dispersos y enemigos que sueltan almas (o algún otro recurso utilizado para mejorar estadísticas y / o armas que se pierden al morir), pero el jugador tiene una oportunidad de recuperar las almas caídas si pueden llegar al lugar de su muerte sin volver a morir.

El primer juego de la serie souls lanzado en 2009 para PlayStation 3 fue Demon´s Souls el cual contiene un sistema de creación de personajes y destaca la recolección de recompensas durante el combate con los enemigos en una serie no lineal de ubicaciones variadas[3]. Posteriormente en el año 2011 fue el lanzamiento del segundo videojuego de la serie Souls llamado Dark Souls de rol de acción, desarrollado por FromSoftware para las plataformas PlayStation 3, Xbox 360 y Microsoft Windows, distribuido por Bandai Namco Entertainment.

Dark Souls tiene lugar en el reino ficticio llamado Lordran. Los jugadores asumen el papel de un personaje humano maldito que ha sido elegido para realizar un peregrinaje para descubrir el destino de los no muertos. El argumento del juego se va contando fundamentalmente a través de descripciones de objetos del juego, y

---

[1] *Wikipedia/Metroidvania*. (09 de septiembre de 2021). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Metroidvania.

[2] *Wikipedia/The_Legend_of_Zelda*. (17 de 07 de 2021). Obtenido de https://es.wikipedia.org/wiki/The_Legend_of_Zelda

[3] *Wikipedia/Souls_Serie.* (09 de 10 de 2021). Obtenido de https://es.wikipedia.org/wiki/Souls_(serie)

diálogos con personajes no jugables (PNJs). Los jugadores deben ir reuniendo pistas para poder entender la historia.

Dark Souls se labró un gran reconocimiento por su exigente dificultad y duro desafío. El mundo del juego está lleno de armas, armaduras y objetos consumibles que tienen como objetivo ayudar al jugador durante su viaje. Para el año 2014 se estrenó el tercer videojuego Dark Souls II el cual tiene lugar en el reino ficticio de Drangleic, donde el jugador debe encontrar una cura para la maldición de los muertos vivientes. Si bien, está adaptado al mismo universo que Dark Souls, no tienen conexión directa entre sí.

Seguidamente en el año 2016 se lanzó el cuarto videojuego Dark Souls III como continuidad de la saga Dark Souls, en donde se resalta la jugabilidad, la cual es más rápida y fluida que la de sus predecesores y que es atribuida en parte a un juego de FromSoftware de rol de acción llamado Bloodborne que no es precisamente un souls.

Ahora bien, en lo referente a la aplicación de técnicas de generación procedimental en videojuegos, se conoce que aproximadamente en el año de 1980 se comenzó con esta práctica debido a las grandes limitaciones de memoria, por lo que muchos juegos de la época usaban de alguna manera este tipo de técnicas, de los cuales el más destacado es Rogue, donde se emplea caracteres ASCII para dibujar todos los elementos visuales, implementando algoritmos de generación procedimental de contenido de mazmorras[4]

Otros ejemplos de videojuegos procedimentales de la década de los 80 que sobresalen son Elite, desarrollado por Acornsoft y publicado en 1984 para la computadora BBC Micro y Acorn Electron, en dónde el universo en el cual se lleva a cabo el videojuego era generado procedimentalmente (Elite Dangerous ESP, 2017) y The Sentinel (videojuego de comercio espacial), creado por Geoff Crammond y publicado bajo el sello de Firebird en 1986 para máquinas como la BBC Micro, Comodore 64, Amstrad CPC, ZX Spectrum, Atari ST, Amiga y PC, tratándose de uno de los primeros juegos de perspectiva 3D con polígonos rellenos y que debido a las limitaciones de memoria de los microprocesadores de 8 bit era complejo almacenar los 10000 mundos que suponía tener, por lo que a través de la generación procedimental, cada mundo se generaba a partir de un pequeño paquete de datos: un número de 8 dígitos obtenido al terminar el mundo anterior.

Años más tarde en 1996 es publicado por Blizzard Entertainment el videojuego

---

[4] Wichman, G. R. (1984). Obtenido de A BRIEF HISTORY OF ROGUE - Glenn R. Wichman: https://www.free-culture.ir/A_brief_history_of_rogue.html

Diablo, que consiste en un videojuego de rol, donde se aplicaron elementos procedimentales a través de la generación aleatoria de la estructura de las mazmorras en forma de gráficos 2D imitando una perspectiva de 3 dimensiones isométrica de gran detalle y los ítems, incorporando una categoría de colores que clasificaba los objetos por rareza, donde las estadísticas de estos se generaban en el momento de la creación. [5]

A partir del 2008 se desarrollaron múltiples videojuegos donde la aplicación de algoritmos de generación procedimental de mundos era la principal característica, tales como Spelunky, Minecraft en el 2009, The Binding of Isaac en el 2011, No Man´s Sky en el 2015 y Dead Cells en el 2017.

[5] *Blizzard Entertainment*. (1996). Obtenido de http://ftp.blizzard.com/pub/misc/Diablo.PDF

8 MARCO TEÓRICO

8.1 Generación Procedimental De Contenido

8.1.1 Definición

Según Caballero[6]: "*En computación, la generación procedimental puede definirse como el método que permite la creación de datos mediante la aplicación de algoritmos evitando la recolección de datos sobre un fenómeno concreto o la producción de los mismos de forma manual*."

Sin embargo, en los videojuegos, la generación procedimental de contenido (PCG, Procedural Content Generation) se refiere a la creación automática de contenido, utilizando algoritmos que le faciliten este proceso a los involucrados. Dicha generación no es una labor fácil, debido a que requiere satisfacer las restricciones del artista, y la experiencia del jugador. El contenido a generar suele ser "pseudo aleatorio" ya que se tiene que asegurar que este sea apto para ser jugado.

8.1.2 Tipos de generación procedimental

Con la existencia de múltiples soluciones de generación procedimental de contenido, es necesario clasificarlas teniendo en cuenta la semejanza de problemas que tratan resolver. Si bien, Andrew Doull [7] propone una taxonomía que procura ser definitiva y otros proponen clasificaciones alternativas[8], en ambos casos se evidencian conceptos similares. A continuación, se definen los tipos de generación procedimental más representativos

---

[6] Caballero, P. S. (09 de 06 de 2020). Obtenido de https://riull.ull.es/xmlui/bitstream/handle/915/19774/Generacion%20procedimental%20de%20entornos%20exteriores%20para%20videojuegos%203D.pdf?sequence=1

[7] Doull, A. (2007). *PCG Wiki.* Obtenido de http://pcg.wikidot.com/the-death-of-the-level-designer

[8] Togelius, J. (2011). *julian.togelius.* Obtenido de http://julian.togelius.com/Togelius2011Searchbased.pdf

8.1.2.1 Generación aleatoria de niveles en tiempo de ejecución

Es el tipo más general y el primero que se tiene en cuenta cuando se trata de generación procedimental de contenido en videojuegos.

La generación en tiempo de ejecución es la generación de contenido que se produce mientras la aplicación o el juego está en marcha, independientemente de que se produzca antes de cargar un nivel o durante el progreso del mismo.

En relación con la generación de niveles es indispensable que estén bien definidos los límites de este, así como los parámetros base para el algoritmo dado que normalmente el juego debe almacenar el nivel completo en memoria y es necesario definir sus límites de manera concreta.

Dentro de los niveles, las distintas zonas de este deben estar siempre conectadas de alguna manera, lo que conlleva a que las reglas de generación sean más complejas debido a que deben asegurar este requerimiento.

Tanto la conectividad de zonas dentro de un nivel como entre distintos niveles suelen estar representados por algún tipo de grafo.

8.1.2.2 Diseño de contenido de niveles

Hace referencia al uso de técnicas de generación para crear los recursos del juego, como pueden ser enemigos, armas, objetos, etc. El objetivo es ayudar al diseñador o programador a generar contenido rápidamente y poblar el nivel atendiendo a los requerimientos del juego. Es un tipo de generación bastante útil en proyectos de pocos recursos, desarrolladores independientes con equipos de pocas personas, donde pueden dejar la creación del contenido relacionado a objetos y enemigos al generador, pudiendo así abarcar un mundo más grande sin tener que crear y posicionar cada elemento o moldear el terreno manualmente.

8.1.2.3 Generación dinámica de mundos

La generación dinámica hace referencia al contenido que se crea al comenzar o durante una partida en marcha pero que no es almacenado permanentemente por el juego.

Para generar este contenido se usa una semilla o un hash que permanece constante para una partida o mundo, de esta manera se puede volver a visitar sin que se produzcan cambios en este. Sin embargo, los cambios permanentes que el jugador

pueda producir, sí se almacenan en disco excepto el mundo inicial debido a que la semilla permite reconstruirlo cada vez que se requiera.

Una vez generada la semilla se expande la creación, empleando un generador de números semi-aleatorios, el cual toma la semilla como base, permitiendo al algoritmo tomar decisiones sobre el tipo de contenido a crear en base a las reglas establecidas por el diseñador.

Por lo general, el diseño de estos algoritmos se produce de una manera jerárquica, donde se comienza con una decisión aleatoria en base a la semilla y a partir de cada paso anterior se genera en cadena el resto de contenido.

En caso de que se realice un cambio en las etapas iniciales la generación de contenido será diferente. Esta estructura tiene el problema de dificultar el ajuste del contenido hasta ciertas etapas finales de la generación, donde empieza a tomar la forma final.

8.1.2.4 Instanciación de entidades del juego

Es una técnica cada vez más común que utiliza middleware (lógica de intercambio de información entre aplicaciones) para modificar dinámicamente ciertos parámetros de las entidades de la aplicación y crear la mayor variedad posible de una misma entidad, minimizando la repetición.

Esto se puede ver claramente en sistemas naturales como bosques, donde cada árbol puede provenir de una entidad base, pero con ciertos cambios sobre esta.

Normalmente se usan números aleatorios de manera que el juego no necesita almacenar la información de cada uno de los objetos, sino simplemente de las entidades base. Esta manera de instanciar objetos crea una sensación genérica de tal manera que no son reconocibles como objetos individuales y pertenecen a un grupo de objetos de características similares.

8.2 Métodos para la generación procedimental de mapas 2D

Noor Shalke[9] en su libro Generación de Contenido en Juegos (Content Generation in Games) describe los diferentes algoritmos usados en la creación de mazmorras, basados en la división recursiva del espacio aplicando unas reglas que dan como resultado una jerarquía de celdas en forma de árbol, además explica algunos algoritmos de generación a través de agentes y autómatas celulares. (Shalke, 2016)

8.2.1 Partición del Espacio Binario

Teniendo un área inicial A, esta se va subdividiendo de manera recursiva cumpliendo unas reglas de tamaño mínimo por celda, Generando dos celdas hijas hasta alcanzar el número de divisiones deseado o no poder subdividirse más. Gracias a este tipo de división, las celdas pueden representarse como un árbol binario.

A partir de esta división, se localizan las habitaciones dentro de los límites de las celdas, las cuales se unen mediante pasillos, comenzando desde las últimas ramas generadas y subiendo por el árbol hasta llegar a la raíz.

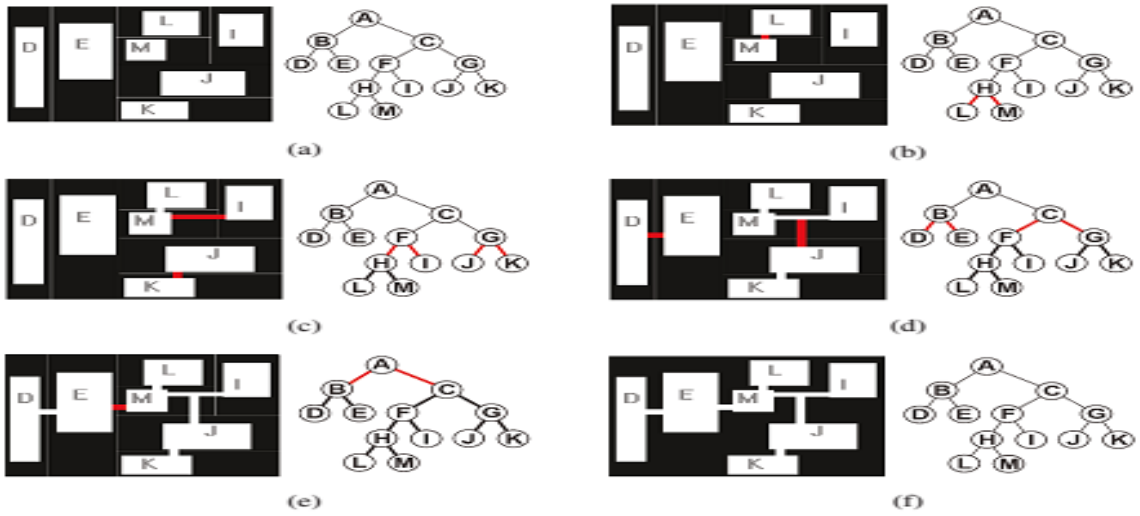


Ilustración 1 Ramas del Árbol Binario como pasillos de la mazmorra. Imagen extraída de Shalke, N 2016

[9] Shalke, N. (2016). *Generación de Contenido en Juegos.* Springer.

8.2.2 Partición del Espacio en Cuatro Cuadrantes (Quadtree)

Este algoritmo se utiliza frecuentemente en aplicaciones de gráficos para organizar la información de una imagen y manipularla de forma más eficiente. Al ser usado en la generación de mazmorras, su funcionamiento es similar al algoritmo de Árboles Binario con la diferencia que, en lugar de generar dos hijos en cada subdivisión, genera cuatro.

La lógica aplicada para crear las habitaciones en las celdas generadas afecta al aspecto de la mazmorra (Ilustración 2). Por lo tanto, al crearse una habitación en cada celda (b) se obtiene como resultado una mazmorra mucho más “cuadriculada” a diferencia de tener en cuenta cada celda como un espacio en blanco o a rellenar, de manera aleatoria (a).

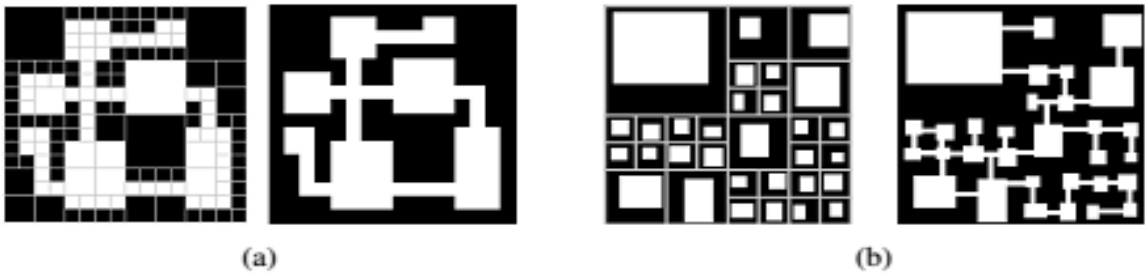


Ilustración 2 Generación de mazmorras a través de Quadtree. Imagen extraída de Shalke, N 2016

```
1: start with the entire dungeon area (root node of the BSP tree)
2: divide the area along a horizontal or vertical line
3: select one of the two new partition cells
4: if this cell is bigger than the minimal acceptable size:
5:   go to step 2 (using this cell as the area to be divided)
6: select the other partition cell, and go to step 4
7: for every partition cell:
8:   create a room within the cell by randomly
     choosing two points (top left and bottom right)
     within its boundaries
9: starting from the lowest layers, draw corridors to connect
   rooms corresponding to children of the same parent
   in the BSP tree
10:repeat 9 until the children of the root node are connected
```

Ilustración 3 Pseudocódigo de alto nivel de algoritmo generativo Imagen extraída de Shalke, N 2016

8.2.3 Generación de mazmorra basada en Agentes.

Normalmente se usa un agente que al mismo tiempo recorre y escarba el mapeado, obteniendo como resultado una mazmorra mucho más orgánica, pero desordenada. La apariencia dependerá considerablemente de prueba y error y del tipo de agente que la crea, por ejemplo, con un agente de comportamiento estocástico el resultado puede ser confuso, mientras que con un agente que solo avanza hacia adelante, el resultado será una apariencia no tan definida.

Los comportamientos para agentes son infinitos, pero tienen en común que en cada jugada o movimiento existe una probabilidad ajustable de cambiar la dirección en la que cavan y una probabilidad de generar una habitación de tamaño aleatorio. La probabilidad de cambiar de dirección aumenta con cada jugada que ejerce el agente, mientras que la de generar una habitación disminuye. Cuando cambia de dirección, ambas se reinician (Ilustración 4).

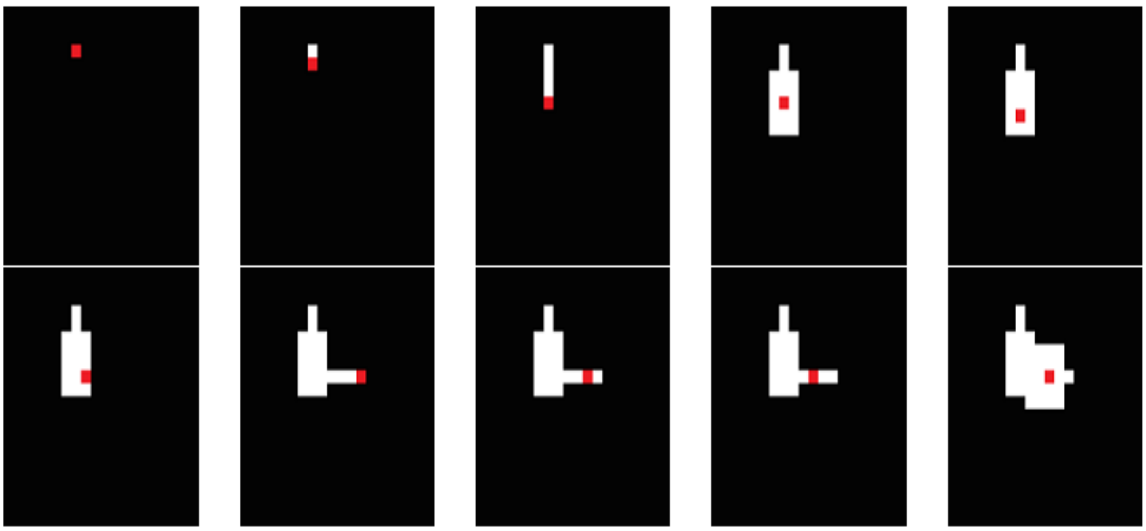


Ilustración 4 Generación de una mazmorra por medio de un agente estocástico. Imagen extraída de Shalke, N 2016

```
1: initialize chance of changing direction Pc=5
2: initialize chance of adding room Pr=5
3: place the digger at a dungeon tile and randomize its direction
4: dig along that direction
5: roll a random number Nc between 0 and 100
6: if Nc below Pc:
7:   randomize the agent's direction
8:   set Pc=0
9: else:
10:  set Pc=Pc+5
11:roll a random number Nr between 0 and 100
12:if Nr below Pr:
13:  randomize room width and room length between 3 and 7
14:  place room around current agent position
14:  set Pr=0
15:else:
16:  set Pr=Pr+5
17:if the dungeon is not large enough:
18:  go to step 4
```

Ilustración 5 Pseudocódigo generador de mazmorra por medio de un agente estocástico. Imagen extraída de Shalke, N 2016

Este procedimiento no asegura generar habitaciones duplicadas, mazmorras homogéneas o la concentración de las habitaciones en una esquina del área del mapa.

Como prevención de la duplicidad se pueden usar agentes con mayor información sobre el tipo de mazmorra a generar y la ubicación de las habitaciones, de tal manera que al generar unas habitaciones sea posible evaluar si resultará una composición habitación-habitación o habitación-pasillo (Ilustración 6).

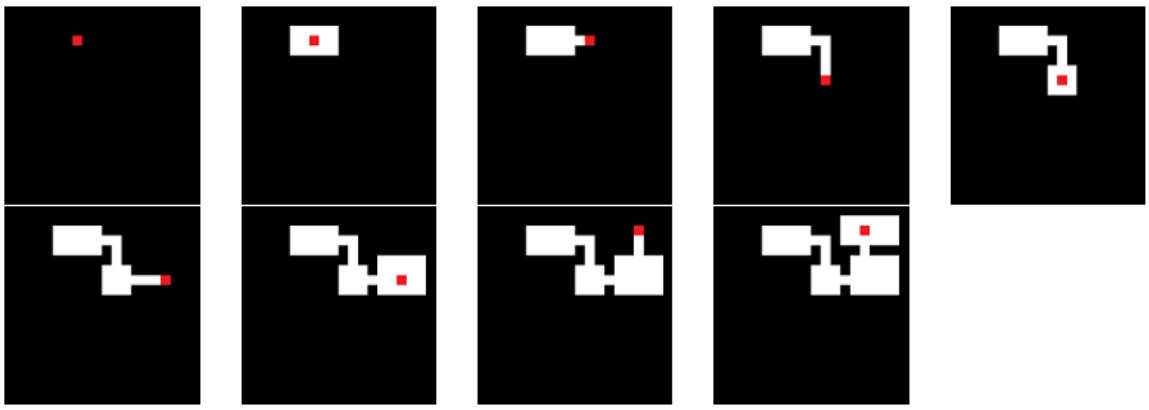


Ilustración 6 Generación de mazmorra por medio de agente con información de escenario

```
1: place the digger at a dungeon tile
2: set helper variables Fr=0 and Fc=0
3: for all possible room sizes:
3:   if a potential room will not intersect existing rooms:
4:     place the room
5:     Fr=1
6:     break from for loop
7: for all possible corridors of any direction and length 3 to 7:
8:   if a potential corridor will not intersect existing rooms:
9:     place the corridor
10:    Fc=1
11:    break from for loop
12:if Fr=1 or Fc=1:
13:  go to 2
```

Ilustración 7 Pseudocódigo generador de mazmorra por medio de agente con información de escenario

8.2.4 Autómata celular

Un autómata celular está formado por una malla de n dimensiones, un conjunto de estados y un conjunto de reglas de transición. La mayoría de los autómatas celulares son unidimensionales (vectores) o bidimensionales (matrices). Cada celda puede estar en uno de varios estados;

La distribución de los estados de las células al principio de un experimento (en el momento t0) es el estado inicial del autómata celular. A partir de ahí, el autómata evoluciona en pasos discretos basados en las reglas de ese autómata en particular. En cada momento t, cada célula decide su nuevo estado basándose en su propio estado y en el de todas las células de su entorno en el momento t. las células de su vecindad en el momento t - 1.

La vecindad define qué celdas alrededor de una celda c afectan al estado futuro de c. En los autómatas celulares unidimensionales, la vecindad se define por su tamaño, es decir, cuántas celdas a la izquierda o a la derecha se extiende la vecindad.

En el caso de los autómatas bidimensionales, los dos tipos de vecindades más comunes son las vecindades de Moore y de von Neumann. Ambas vecindades pueden tener un tamaño de cualquier número entero mayor a 1. La vecindad de Moore es un cuadrado (*(2r + 1)2 – 1 de distancia r*) una de tamaño 1 está formada por las ocho celdas que rodean inmediatamente a c, incluidas las que la rodean en diagonal. La vecindad de von Neumann es como una cruz centrada en c: una vecindad de von Neumann de tamaño 1 está formada por las cuatro celdas que

rodean a c por encima, por debajo, a la izquierda y a la derecha

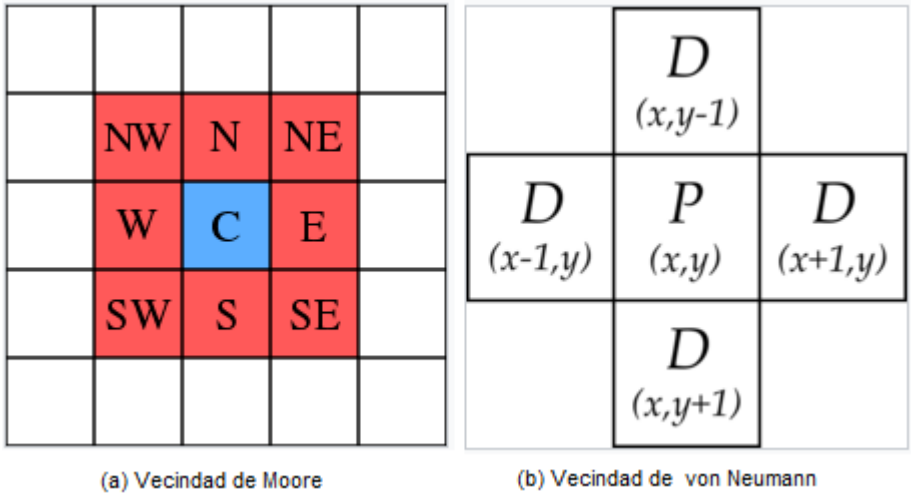


Ilustración 8 Tipo de vecinos en una malla 2D de células autómatas. Imagen Extraída de Wikipedia

El primer acercamiento a la generación de mapas usando autómatas celulares se realizó con el fin de obtener mazmorras de tipo cueva de aspecto orgánico, donde el mapa se va generando a medida que el jugador avanza.

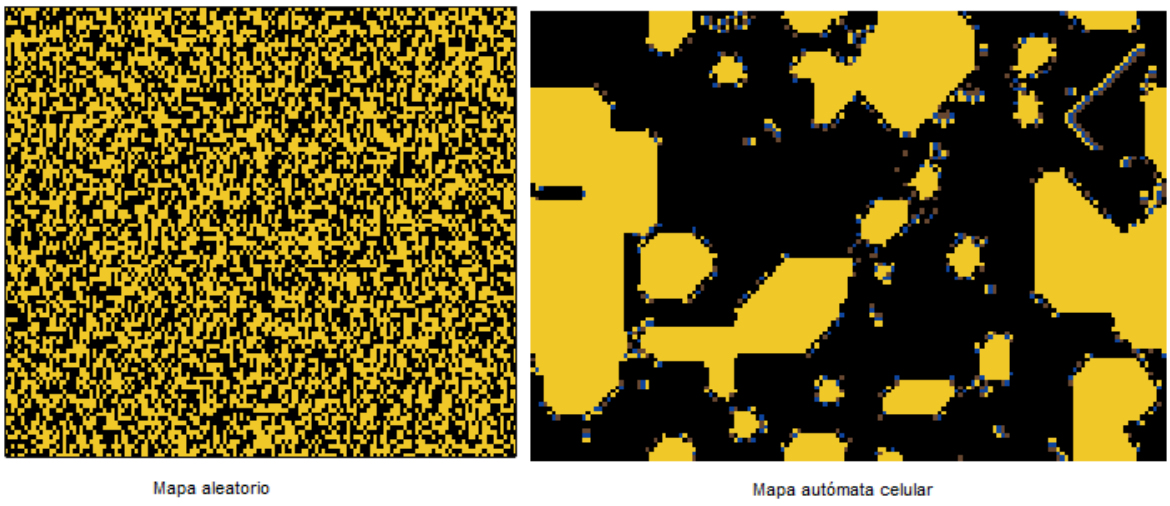


Ilustración 9 Mapa antes y después de la interacción de un autómata celular. Imagen extraída de (Hu, R., 29 de agosto de 2017)

8.3 Motor de Videojuego

El motor de videojuegos seleccionado para la creación del videojuego 2D fue GODOT, el cual según Wikipedia[10] es “un motor de videojuegos 2D y 3D multiplataforma, libre y de código abierto, publicado bajo la Licencia MIT[11] y desarrollado por la comunidad de Godot. El motor es funcional en sistemas Windows, OS X, Linux y BSD. Permite exportar los videojuegos creados a PC (Windows, OS X y Linux), teléfonos móviles (Android, iOS), y HTML5 y que cuenta con las siguientes características generales:

- Soporta los lenguajes de programación c++ c# y VisualScript, siendo GDScript el principal lenguaje de scripting similar a python que fue creado especialmente para Godot, lo cual lo hace flexible y óptimo para programar videojuegos en el motor.

- Cuenta con un editor visual de sombreadores el cual consiste en un programa informático que realiza cálculos gráficos escrito en un lenguaje de sombreado GLSL (OpenGL Shading Language por sus siglas en inglés) que se puede compilar independientemente.

- Utiliza un lenguaje simplificado de sombreadores que puede ser utilizado para mapeos de texturas, post procesamiento y 2D.

- A diferencia de otros motores, no existe la necesidad de simular 2D en espacio 3D. El motor soporta luces, sombreadores, GUIs, sprites, tilesets, desplazamiento de paralaje, polígonos, animaciones, física, partículas y más. También es posible de combinar 2D con 3D, o 3D con 2D utilizando nodos viewport. En la versión de Godot 3.2, se ha implementado la posibilidad de desarrollar juegos 2.5D.

- soporta múltiples plataformas.

- Posee un sistema de animación sofisticado y completo, con soporte para editar animación esquelética, blending, árboles de animación, morphing y cinemáticas. El sistema de animación permite animar las propiedades de los nodos y ejecutarlas simplemente llamando funciones.

---

[10] *Wikipedia/Godot.* (octubre de 2021). Obtenido de https://es.wikipedia.org/wiki/Godot

[11] *Wikipedia.* (20 de octubre de 2021). Obtenido de https://en.wikipedia.org/wiki/MIT_License

- Ofrece varios objetos de colisión en 2D y 3D para proveer tanto detección como respuesta a colisión. Tiene su propio motor de físicas juegos 2D y 3D con detección de colisión, cuerpo rígido, cuerpo estático, personajes, vehículos, raycasts y uniones.[12]

[12] Godot, C. (2021). *GODOT DOCS.* Obtenido de https://docs.godotengine.org/es/stable/about/list_of_features.html

9 METODOLOGÍA.

Como metodología de desarrollo se optó por la metodología ágil XP, la cual se basa en la simplicidad, comunicación y realimentación del código desarrollado.

Algunas de sus características fundamentales son las siguientes:

- Desarrollo iterativo e incremental.
- Pruebas unitarias.
- Programación en parejas.
- Corrección antes de crear una nueva funcionalidad.
- Refactorización de código
- Grupo pequeño y muy integrado mínimo 2 personas y máximo 12

9.1 Planificación

Teniendo en cuenta el alcance del proyecto previamente definido, los requisitos funcionales y no funcionales identificados para la versión Demo y el tiempo estipulado para el desarrollo del videojuego 2D, el cual será de 90 días calendario (3 meses), se establecieron las siguientes historias de usuario:

- **Creación de nueva partida**

Tabla 1 Historia de Usuario 1

	Historia de Usuario
Número: 1	**Usuario:** JUGADOR
Nombre historia: Creación de nueva partida	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada: 1**

Programador responsable: Jonatan Huergo Aguilar
Descripción: El jugador inicia el juego creando una nueva partida donde elige el tamaño del mundo: Corto – Mediano - Largo
Observaciones:

- **Creación de nuevo mundo**

Tabla 2 Historia de Usuario 2

Historia de Usuario	
Número: 2	**Usuario:** JUGADOR
Nombre historia: Creación de nuevo mundo	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 4	**Iteración asignada: 1**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: El jugador selecciona el tamaño del mundo: Corto – Mediano - Largo y se crea aleatoriamente el mundo	
Observaciones: Se debe ejecutar automáticamente el algoritmo de generación procedimental de mundos al seleccionar algunos de los tres tipos de tamaño del mundo.	

- **Jugar Partida**

Tabla 3 Historia de Usuario 3

Historia de Usuario	
Número: 3	**Usuario:** JUGADOR
Nombre historia: Juego de la partida creada	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 4	**Iteración asignada: 2**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: El jugador comienza a jugar la partida creada según el mundo elegido. Los enemigos se van fortaleciendo progresivamente según el nivel en el que se encuentre el jugador y su historial de juego	
Observaciones: Se debe ejecutar automáticamente el algoritmo de aumento progresivo de la fortaleza de los enemigos a medida que el jugador avanza de nivel.	

- **Guardar Partida**

Tabla 4 Historia de Usuario 4

Historia de Usuario	
Número: 4	**Usuario:** JUGADOR
Nombre historia: Guardar partida en juego	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada: 2**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: El jugador pausa el juego y guarda la partida en curso para posteriormente reanudarla	
Observaciones: Se debe reanudar el videojuego en el punto donde el jugador realizó su último movimiento antes de guardar la partida.	

- **Reanudar Partida**

Tabla 5 Historia de Usuario 5

Historia de Usuario	
Número: 5	**Usuario:** JUGADOR
Nombre historia: Reanudar partida guardada	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: El jugador visualiza en el menú principal la partida guardada y procede a continuar con el juego en el último nivel donde se encontraba	
Observaciones: el menú muestra las opciones: continuar con la partida guardada (si existe) e iniciar una nueva partida	

- **Visualización de Ranking**

Tabla 6 Historia de Usuario 6

Historia de Usuario	
Número: 6	**Usuario:** JUGADOR
Nombre historia: Visualización de Ranking	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada: 3**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: El jugador selecciona en el menú principal la opción de ranking para visualizar el historial de resultados según el tamaño del mundo: Corto - Mediano - Largo	
Observaciones:	

- **Finalizar partida**

Tabla 7 Historia de Usuario 7

Historia de Usuario	
Número: 7	**Usuario:** JUGADOR
Nombre historia: Finalizar partida	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada: 3**
Programador responsable: Jonatan Huergo Aguilar	
Descripción: La partida finaliza cuando el jugador no cuente con almas suficientes para continuar o cuando el jugador seleccione la opción de finalizar la partida	
Observaciones:	

9.2 Asignación de roles del proyecto

Se definieron los siguientes roles:

Tabla 8 Roles

Roles	**Asignado A:**
Programador	Jonatan Huergo Aguilar Leonardo Tafur
Cliente	Jonatan Huergo Aguilar
Encargado de pruebas (Tester)	Jonatan Huergo Aguilar
Encargado de seguimiento (Tracker)	Leonardo Tafur
Entrenador (Coach)	Leonardo Tafur
Consultor	Jonatan Huergo Aguilar
Gestor (Big Boss)	Leonardo Tafur

9.3 Cronograma de entrega

Tomando como base las historias de usuario definidas para el desarrollo de la versión Demo del videojuego 2D se ha elaborado el siguiente cronograma de entregas donde se relacionan las historias de usuario que se llevarán a cabo en cada iteración, teniendo en cuenta la prioridad y el esfuerzo:

HISTORIAS	ITERACIÓN	PRIORIDAD	ESFUERZO	MES 1				MES 2				MES 3			
				SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4
Historia 1	1	Alta	3												
Historia 2	1	Alta	4												
Historia 3	2	Alta	4												
Historia 4	2	Alta	2												
Historia 5	2	Alta	3												
Historia 6	3	Alta	3												
Historia 7	3	Alta	3												

Ilustración 10 Cronograma de Entrega

9.4 Ciclo de vida (Videojuego 2D versión Demo de género soulslike)

9.4.1 Primera Iteración

En esta primera iteración se desarrollaron la historia de usuario 1 “Creación de nueva partida” y la historia de usuario 2 “Creación de nuevo mundo”.

9.4.1.1 Tareas de Ingeniería

Se definieron las siguientes tareas de ingeniería para las historias 1 y 2:

Tabla 9 Tareas de Ingeniería Historias de Usuario 1 y 2

NÚMERO DE HISTORIA	NÚMERO DE TAREA	NOMBRE DE LA TAREA
1	1	Diseño interfaz de inicio y menú principal
1	2	Diseño interfaz nueva partida con elección del tamaño del mundo
1	3	Selección del personaje héroe, enemigos, y boss
2	4	Desarrollo del algoritmo de generación procedimental del tamaño mundo
2	5	Validación del tamaño del mundo

9.4.1.2 Descripción Tareas de Ingeniería

Tabla 10 Descripción Tarea de Ingeniería 1

TAREA DE INGENIERÍA	
Número de Tarea: 1	Número de Historia: 1
Nombre de Tarea: Diseño interfaz de inicio y menú principal	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 1 - Semana 1	Fecha Fin: Mes 1 -Semana 2
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se desarrollará la interfaz de inicio y menú principal del juego	

Tabla 11 Descripción Tarea de Ingeniería 2

TAREA DE INGENIERÍA	
Número de Tarea: 2	Número de Historia: 1
Nombre de Tarea: Diseño interfaz nueva partida con elección del tamaño del mundo	
Tipo de Tarea: Desarrollo	Punto Estimados: 1
Fecha Inicio: Mes 1 - Semana 1	Fecha Fin: Mes 1 - Semana 2
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se desarrollará la interfaz de creación de nueva partida con la opción de selección del tamaño del mundo: pequeño, mediano, grande, para dar inicio al juego.	

Tabla 12 Descripción Tarea de Ingeniería 3

TAREA DE INGENIERÍA	
Número de Tarea: 3	Número de Historia: 1
Nombre de Tarea: Selección del personaje héroe, enemigos, y boss	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 1 - Semana 1	Fecha Fin: Mes 1 - Semana 2
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se seleccionará el héroe principal (jugador), 2 tipos de enemigos y el boss a vencer, los cuales se usarán automáticamente en todos los niveles del juego.	

Tabla 13 Descripción Tarea de Ingeniería 4

TAREA DE INGENIERÍA	
Número de Tarea: 4	Número de Historia: 2
Nombre de Tarea: Desarrollo del algoritmo de generación procedimental del tamaño mundo	
Tipo de Tarea: Desarrollo	Punto Estimados: 1
Fecha Inicio: Mes 1 - Semana 3	Fecha Fin: Mes 2 - Semana 1
Programador Responsable: Jonatan Huergo Aguilar - Leonardo Tafur	
Descripción: Se necesita desarrollar un algoritmo de generación procedimental de mundos basado en el tamaño del mundo seleccionado por el jugador	

Tabla 14 Descripción Tarea de Ingeniería 5

TAREA DE INGENIERÍA	
Número de Tarea: 5	Número de Historia: 2
Nombre de Tarea: Validación del tamaño del mundo	
Tipo de Tarea: Desarrollo	Punto Estimados: 1
Fecha Inicio: Mes 1 - Semana 3	Fecha Fin: Mes 2 - Semana 1
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se necesita probar el algoritmo y validar los resultados obtenidos para su posterior implementación	

9.4.1.3 Tarjetas CRC

Tabla 15 Tarjeta CRC Iteración 1

Mundo	
RESPONSABILIDAD	**COLABORACIÓN**
Crear mundo aleatorio de forma automática	

9.4.1.4 Pruebas de aceptación

Las pruebas generales para la iteración 1 son las siguientes:

Tabla 16 Pruebas Iteración 1

NÚMERO DE LA PRUEBA	NÚMERO DE LA HISTORIA	NOMBRE DE LA PRUEBA
1	1	Creación de nueva partida
2	2	Creación de nuevo mundo

9.4.1.5 Descripción pruebas de aceptación

Tabla 17 Descripción caso de prueba 1 Historia de Usuario 1

CASO DE PRUEBA	
Código: 1	**No de Historia de Usuario**: 1
Historia de Usuario: Creación de nueva partida	
Condiciones de Ejecución: El jugador debe de visualizar una vista de inicio con un menú principal para comenzar a interactuar con el videojuego	
Entrada/Pasos de Ejecución: Dar clic en el botón de inicio de juego, Dar clic en la opción nueva partida del menú principal	
Resultado Esperado: Acceso a una nueva ventana para seleccionar el tamaño del mundo	
Evaluación de la prueba: Prueba superada satisfactoriamente	

Tabla 18 Descripción caso de prueba 2 Historia de Usuario 2

CASO DE PRUEBA	
Código: 2	**No de Historia de Usuario**: 2
Historia de Usuario: Creación de nuevo mundo	
Condiciones de Ejecución: El jugador debe seleccionar alguna de los 3 tipos de tamaño del mundo: Corto - Mediano - Largo y se debe crear el mundo de forma automática	
Entrada/Pasos de Ejecución: Dar clic en alguno de los 3 tipos de tamaño. Se crea automáticamente el mundo. Se inicia el juego con la caída libre del héroe al mundo creado.	
Resultado Esperado: Creación automática del mundo según el tamaño seleccionado y caída libre del héroe en el mundo para dar inicio al juego	
Evaluación de la prueba: Prueba superada satisfactoriamente	

9.4.1.6 Diseño

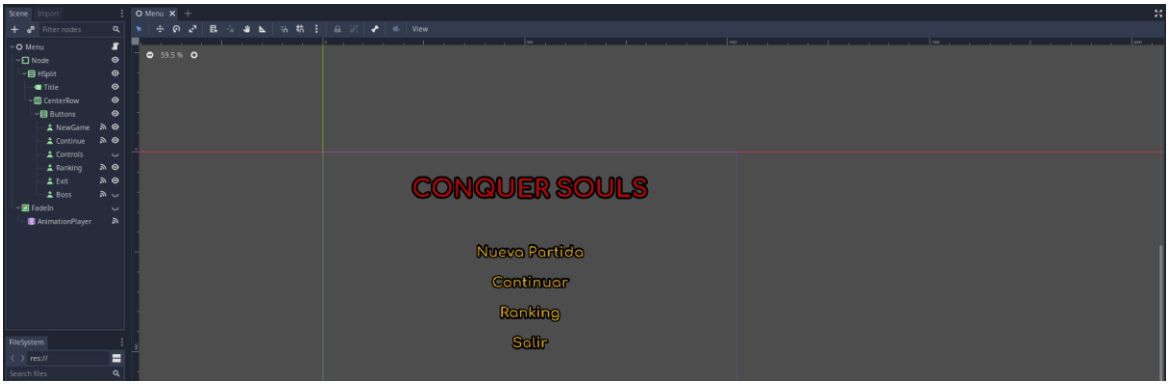


Ilustración 11 Diseño vista de inicio y menú principal

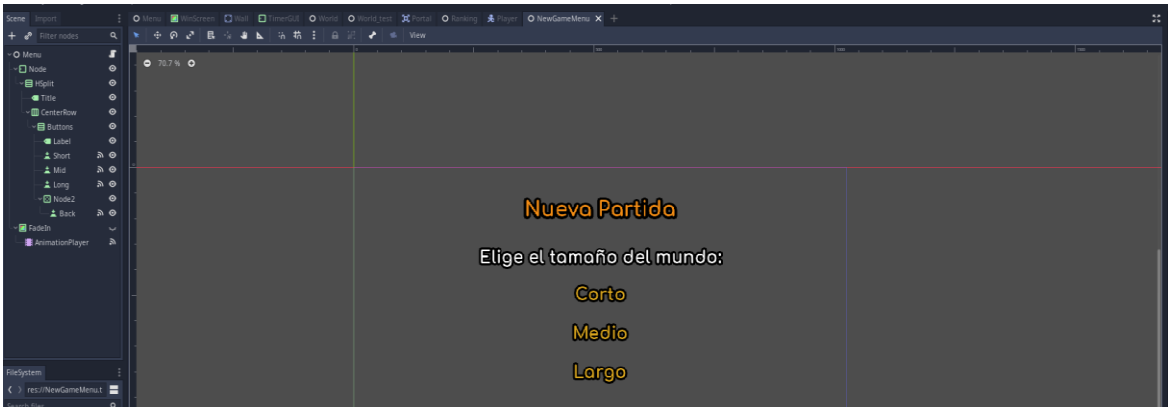


Ilustración 12 Diseño de la vista de elección del tamaño del mundo

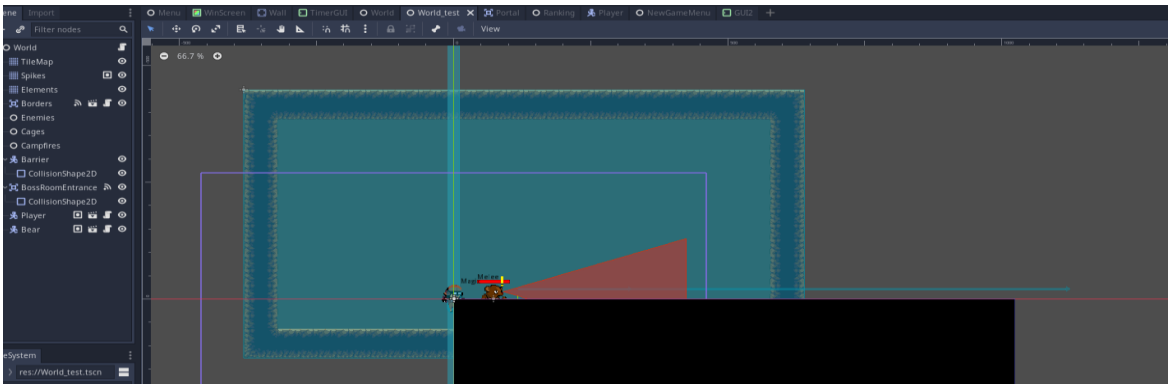


Ilustración 13 diseño de generación del mundo

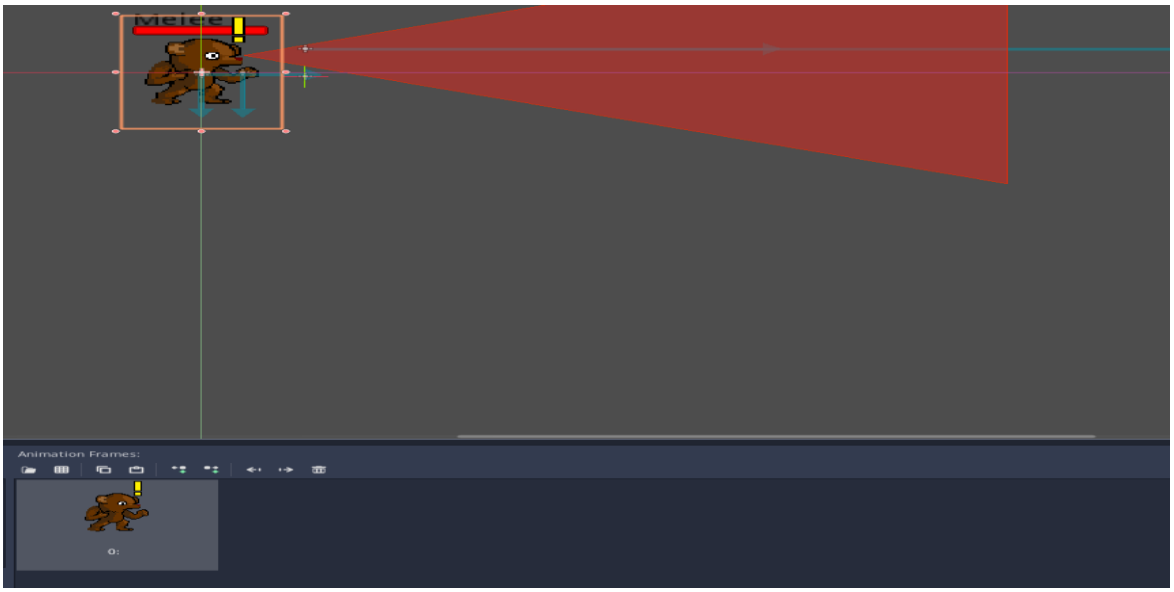


Ilustración 14 Diseño del personaje “enemigo”

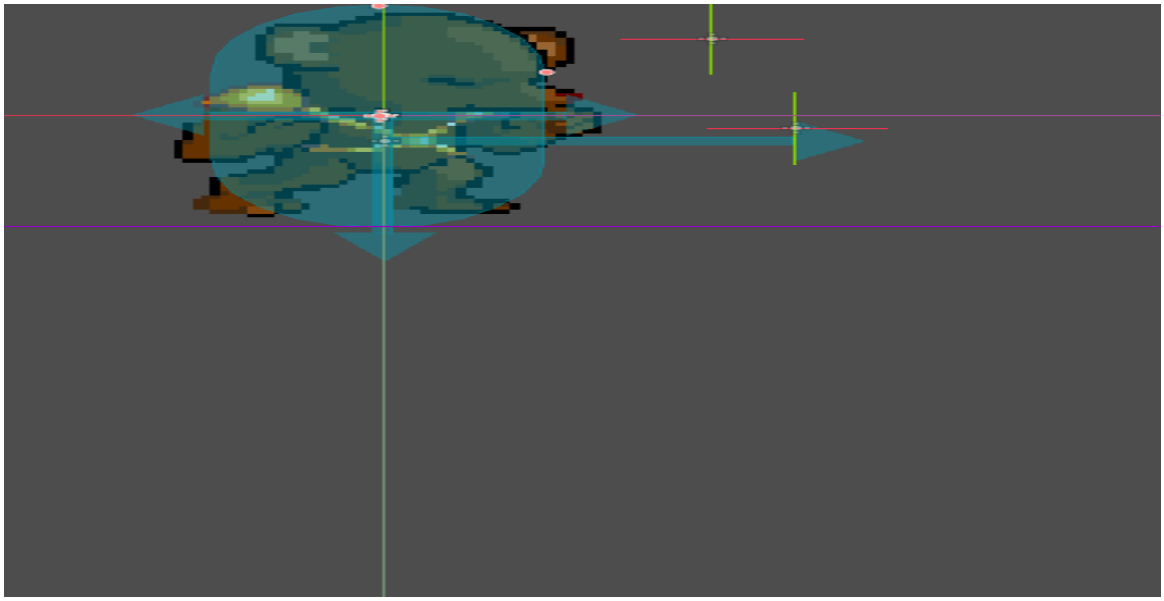


Ilustración 15 Diseño Jefe Enemigo Principal

9.4.1.7 Codificación

```gdscript
extends Node

const SAVE_DIR = "user://saves/"
var save_path = SAVE_DIR + "Save.dat"

export (String, FILE, "*.tscn") var world_scene
export (String, FILE, "*.tscn") var new_world_menu
export (String, FILE, "*.tscn") var controls_scene
export (String, FILE, "*.tscn") var ranking_scene

var scene_to_load

func _ready():
	$Node/HSplit/CenterRow/Buttons/NewGame.grab_focus()
	var file = File.new()
	if not file.file_exists(save_path):
		$Node/HSplit/CenterRow/Buttons/Continue.disabled = true


func _on_NewGame_pressed():
	scene_to_load = new_world_menu
	get_tree().change_scene(scene_to_load)
	#$FadeIn/AnimationPlayer.play("fade_in")
	#$FadeIn.show()


func _on_Continue_pressed():
	scene_to_load = world_scene
	$FadeIn/AnimationPlayer.play("fade_in")
	$FadeIn.show()


func _on_Ranking_pressed():
	scene_to_load = ranking_scene
	get_tree().change_scene(scene_to_load)


func _on_Exit_pressed():
	get_tree().quit(0)

func _on_Boss_pressed():
	get_tree().change_scene("res://World_test.tscn")


func _on_AnimationPlayer_animation_finished(anim_name):
	get_tree().change_scene(scene_to_load)

```

Ilustración 16 Código interfaz de inicio y menú principal

```gdscript
extends Node

export (String, FILE, "*.tscn") var world_scene
export (String, FILE, "*.tscn") var menu_scene

# Called when the node enters the scene tree for the first time.
func _ready():
	$Node/HSplit/CenterRow/Buttons/Short.grab_focus()


func _on_Button_pressed(name):
	$FadeIn/AnimationPlayer.play("fade_in")
	$FadeIn.show()
	print(name)
	match name:
		"short":
			Global.world_size = 100
		"mid":
			Global.world_size = 200
		"long":
			Global.world_size = 300
	Save.delete_data()

func _on_AnimationPlayer_animation_finished(anim_name):
	get_tree().change_scene(world_scene)


func _on_Back_pressed():
	get_tree().change_scene(menu_scene)

```

Ilustración 17 Código interfaz nueva partida con elección del tamaño del mundo

9.4.1.8 Capturas de Pantalla

Ilustración 18 Captura menú principal

Ilustración 19 Captura elección tamaño del mundo

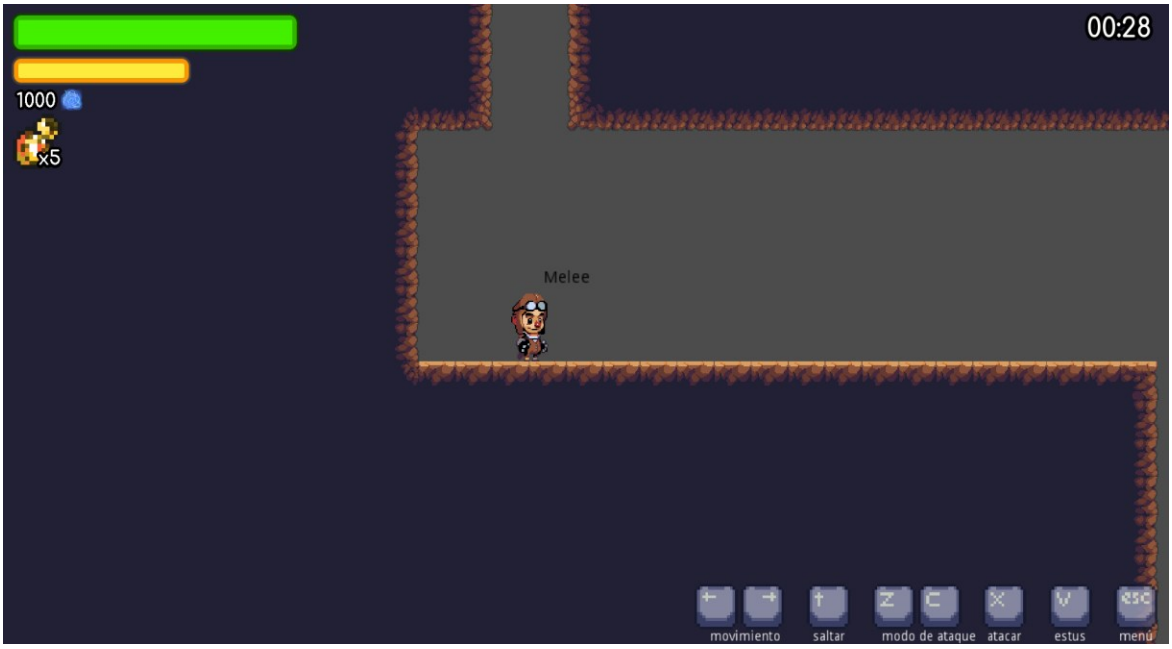


Ilustración 20 Captura mundo generado

9.4.1.9 Resultado:

Como resultado de entrega de la primera iteración del Videojuego 2D versión Demo de género soulslike se desarrolló satisfactoriamente la interfaz de inicio que contiene el menú principal, adicionalmente se logró la implementación del algoritmo de generación procedimental de mundos con base al tamaño seleccionado previamente y por último se crearon los personajes del juego (Héroe, osos enemigos y Enemigo principal - Jefe boss), quedando pendiente el desarrollo de las habilidades del héroe, el cual se realizará en la siguiente iteración.

9.4.2 Segunda Iteración

En esta segunda iteración se desarrollaron la historia de usuario 3 “Juego de la partida creada”, la historia de usuario 4 “Guardar partida en juego” y la historia de usuario 5 “Reanudar partida guardada”.

9.4.2.1 Tareas de Ingeniería

Se definieron las siguientes tareas de ingeniería para las historias 3, 4 y 5:

Tabla 19 Tareas de Ingeniería Historias de Usuario 3, 4 y 5

NÚMERO DE HISTORIA	NÚMERO DE TAREA	NOMBRE DE LA TAREA
3	6	Diseño de la primera escena del videojuego con un suelo y el personaje héroe afectado por la gravedad
3	7	Diseño de los ataques y movimientos del héroe, enemigos y boss
3	8	Desarrollo de los ataques y movimientos del héroe, enemigos y boss
3	9	Diseño del estado de monitoreo del nivel de vida, nivel de ataque del héroe, enemigos y boss
3	10	Desarrollo del estado de monitoreo del nivel de vida de ataque del héroe, enemigos y boss
3	11	Desarrollo del temporizador, contador de almas y puntuación adquirida por el jugador en el nivel
3	12	Desarrollo del agente inteligente de aumento progresivo de la fortaleza de los enemigos
4	13	Desarrollo del guardado automático del videojuego
5	14	Creación de la opción continuar en el menú principal en la interfaz de inicio

9.4.2.2 Descripción Tareas de Ingeniería.

Tabla 20 Descripción Tarea de Ingeniería 6

TAREA DE INGENIERÍA	
Número de Tarea: 6	Número de Historia: 3
Nombre de Tarea: Diseño de la primera escena del videojuego con un suelo y el personaje héroe afectado por la gravedad	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 -Semana 3
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se desarrollará la primera escena del videojuego partiendo de una imagen pixelada de suelo y con la acción de caída libre del personaje Héroe al mundo previamente creado	

Tabla 21 Descripción Tarea de Ingeniería 7

TAREA DE INGENIERÍA	
Número de Tarea: 7	Número de Historia: 3
Nombre de Tarea: Diseño de los ataques y movimientos del héroe, enemigos y boss	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se realizará el diseño de los ataques y movimientos de los personajes (Héroe, enemigos y Boss) que son parte del videojuego	

Tabla 22 Descripción Tarea de Ingeniería 8

TAREA DE INGENIERÍA	
Número de Tarea: 8	Número de Historia: 3
Nombre de Tarea: Desarrollo de los ataques y movimientos del héroe, enemigos y boss	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se desarrollarán las funciones correspondientes a los ataques y movimientos de los personajes (Héroe, enemigos y Boss) en el videojuego	

Tabla 23 Descripción Tarea de Ingeniería 9

TAREA DE INGENIERÍA	
Número de Tarea: 9	Número de Historia: 3
Nombre de Tarea: Diseño del estado de monitoreo del nivel de vida, nivel de ataque del héroe, enemigos y boss	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se diseñarán las barras de visualización del nivel de vida, nivel de ataque de los personajes (Héroe, enemigos y Boss) en el videojuego	

Tabla 24 Descripción Tarea de Ingeniería 10

TAREA DE INGENIERÍA	
Número de Tarea: 10	Número de Historia: 3
Nombre de Tarea: Desarrollo del estado de monitoreo del nivel de vida de ataque del héroe, enemigos y boss	
Tipo de Tarea: Desarrollo	Punto Estimados: 1
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar - Leonardo Tafur	
Descripción: Se desarrollarán las funciones de aumento o disminución del nivel de vida y nivel de ataque en las barras de visualización de los personajes (Héroe, enemigos y Boss) en el videojuego	

Tabla 25 Descripción Tarea de Ingeniería 11

TAREA DE INGENIERÍA	
Número de Tarea: 11	Número de Historia: 3
Nombre de Tarea: Desarrollo del temporizador, contador de almas y puntuación adquirida por el jugador en el nivel	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Se desarrollará la función temporizador que controlará el tiempo máximo que tendrá el jugador para pasar de un nivel a otro, el contador de almas y la puntuación adquirida en el nivel	

Tabla 26 Descripción Tarea de Ingeniería 12

TAREA DE INGENIERÍA	
Número de Tarea: 12	Número de Historia: 3
Nombre de Tarea: Desarrollo del agente inteligente de aumento progresivo de la fortaleza de los enemigos	
Tipo de Tarea: Desarrollo	Punto Estimados: 1
Fecha Inicio: Mes 2 - Semana 1	Fecha Fin: Mes 2 - Semana 3
Programador Responsable: Jonatan Huergo Aguilar - Leonardo Tafur	
Descripción: Se debe desarrollar un agente inteligente basado en la técnica de decisión de piedra, papel o tijera para aumentar automáticamente la fortaleza de los personajes (enemigos, Boss) en el siguiente nivel	

Tabla 27 Descripción Tarea de Ingeniería 13

TAREA DE INGENIERÍA	
Número de Tarea: 13	Número de Historia: 4
Nombre de Tarea: Desarrollo del guardado automático del videojuego	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 - Semana 3	Fecha Fin: Mes 2 - Semana 4
Programador Responsable: Jonatan Huergo Aguilar - Leonardo Tafur	
Descripción: Al cerrar el juego o pérdida de vida del jugador (Héroes), el juego se debe guardar automáticamente para su posterior reinicio hasta que el jugador no cuente con el suficiente número de almas para reiniciar el juego, o inicie una nueva partida	

Tabla 28 Descripción Tarea de Ingeniería 14

TAREA DE INGENIERÍA	
Número de Tarea: 14	Número de Historia: 5
Nombre de Tarea: Creación de la opción continuar en el menú principal en la interfaz de inicio	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 3 - Semana 1	Fecha Fin: Mes 3 - Semana 2
Programador Responsable: Jonatan Huergo Aguilar - Leonardo Tafur	
Descripción: Implementar la opción continuar en el menú principal en la interfaz de inicio para que el jugador pueda seguir jugando en el punto de la última acción realizada.	

9.4.2.3 Tarjetas CRC

Tabla 29 Tarjeta CRC Iteración 2

Personaje (Héroe, Enemigos, Boss)	
RESPONSABILIDAD	**COLABORACIÓN**
Mover el Personaje Héroe hacia arriba, hacia abajo o al frente según comando ejecutado por el jugador	
Mover automáticamente el Personaje (Enemigos, Boss) hacia arriba, hacia abajo o al frente	
Realizar ataque	
Morir el personaje (Héroe, Enemigos, Boss) cuando el nivel de vida esté vacío	
reanimar el personaje Héroe cuando el número de almas sean suficientes	

9.4.2.4 Pruebas de aceptación

Las pruebas generales para la iteración 2 son las siguientes:

Tabla 30 Pruebas Iteración 2

NÚMERO DE LA PRUEBA	NÚMERO DE LA HISTORIA	NOMBRE DE LA PRUEBA
3	3	Juego de la partida creada
4	4	Guardar partida en juego
5	5	Reanudar partida guardada

9.4.2.5 Descripción pruebas de aceptación

Tabla 31 Descripción caso de prueba 3 Historia de Usuario 3

CASO DE PRUEBA	
Código: 3	**No de Historia de Usuario**: 3
Historia de Usuario: Juego de la partida creada	
Condiciones de Ejecución: El Personaje Héroe debe caer por gravedad al mundo generado, moverse hacia adelante, arriba o abajo según los comandos ejecutados por el jugador, realizar ataques contra los personajes (Enemigos y Boss), acumular puntuación durante el desarrollo del nivel en juego.	
Entrada/Pasos de Ejecución: Dar click en alguna de las 3 opciones de tamaño del mundo: corto, mediano, largo, el juego inicia automáticamente, el jugador mueve al personaje Héroe y realiza ataques a través de comandos, los personajes (Enemigos y Boss) se mueven y realizan ataques automáticamente, las barras de visualización de los niveles de vida y de ataque aumentan o disminuyen según el desarrollo del juego	
Resultado Esperado: Inicio automático de la partida y desarrollo del juego	
Evaluación de la prueba: Prueba superada satisfactoriamente	

Tabla 32 Descripción caso de prueba 4 Historia de Usuario 4

CASO DE PRUEBA	
Código: 4	**No de Historia de Usuario**: 4
Historia de Usuario: Guardar partida en juego	
Condiciones de Ejecución: La partida debe guardarse automáticamente	
Entrada/Pasos de Ejecución: El Personaje Héroe muere, se guarda automáticamente la última acción y el puntaje obtenido	
Resultado Esperado: Creación y/o actualización de archivo con extensión. psk que contiene los datos guardados automáticamente.	
Evaluación de la prueba: Prueba superada satisfactoriamente	

Tabla 33 Descripción caso de prueba 5 Historia de Usuario 5

CASO DE PRUEBA	
Código: 5	**No de Historia de Usuario**: 5
Historia de Usuario: Reanudar partida guardada	
Condiciones de Ejecución: la partida debe reanudarse	
Entrada/Pasos de Ejecución: después de que el Personaje Héroe muera y el jugador pulse la tecla “Enter”, se reanuda el juego siempre y cuando cuente con la cantidad mínima de almas requeridas para revivir. La partida se puede reanudar desde el menú principal al dar click en la opción continuar si existe algún archivo pck guardado	
Resultado Esperado: La partida se reanuda después de pulsar la tecla “Enter” La partida se reanuda desde el menú principal, pulsando la tecla “Enter”, después de seleccionar la opción continuar.	
Evaluación de la prueba: Prueba superada satisfactoriamente	

9.4.2.6 Diseño

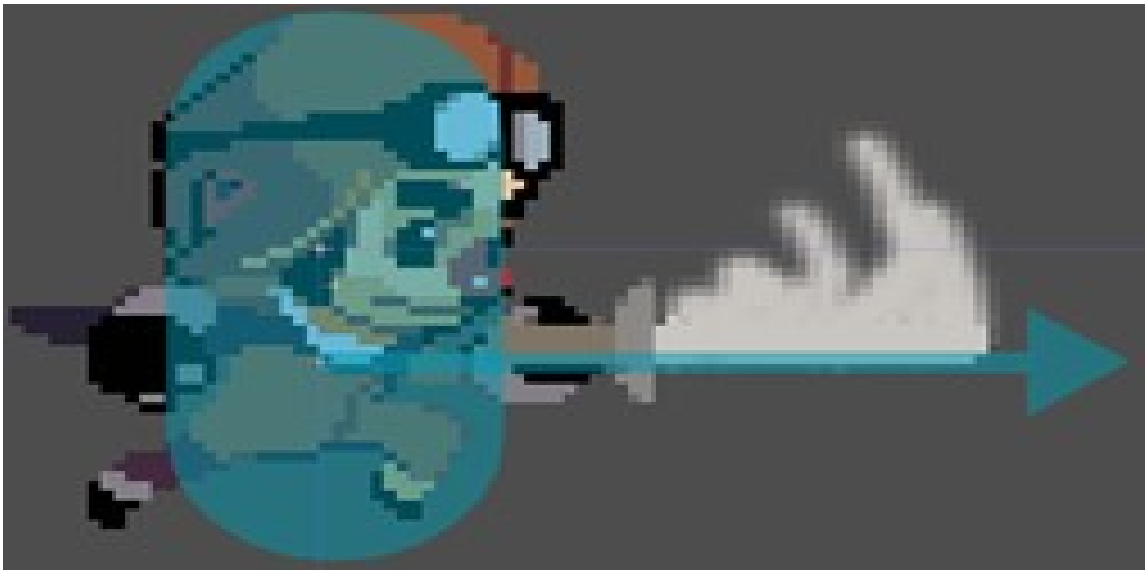


Ilustración 21 Diseño de movimiento y ataque del personaje héroe

Ilustración 22 Diseño de las barras de visualización del nivel de vida, nivel de ataque, puntaje acumulado y número de almas

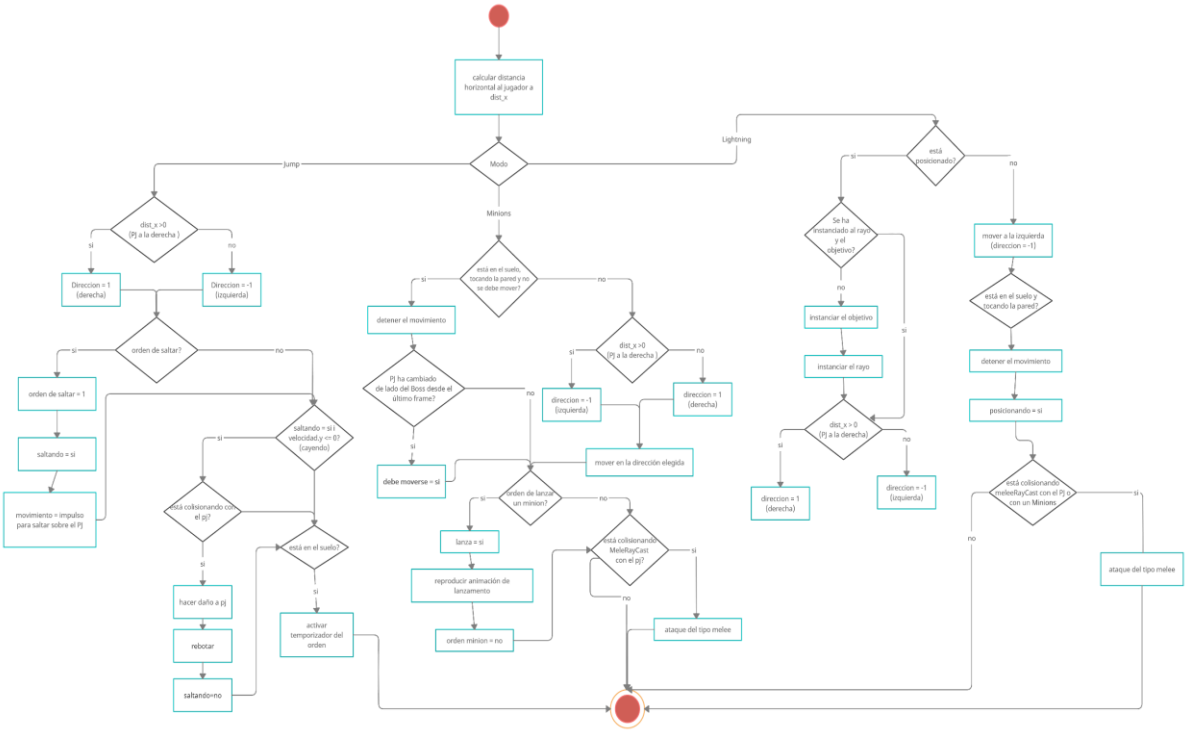


Ilustración 23 Diagrama de flujo de movimiento y ataque del Personaje Boss

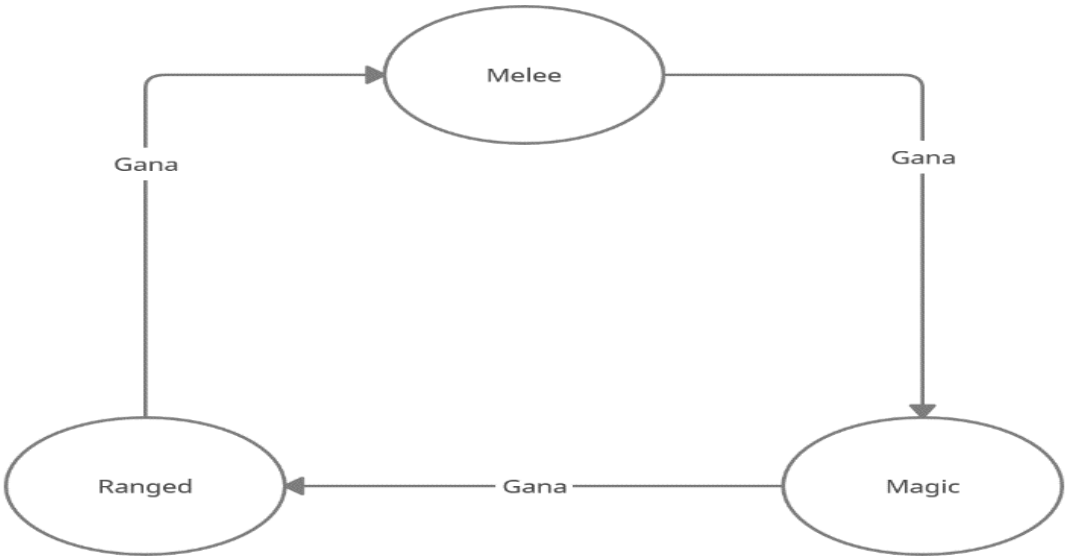


Ilustración 24 Jerarquía piedra - papel - tijera implementado en el agente inteligente de aumento progresivo de la fortaleza de los enemigos

9.4.2.7 Codificación

```
extends KinematicBody2D

var motion = Vector2(0,0)
export var MAX_SPEED = 250
export var G = 10
export var JUMP_FORCE = -400
export var ACC = 50
export var MELEE_DMG = 1
export var RANGED_DMG = 2
export var MAGIC_DMG = 3
export var soul_value = 100
const UP = Vector2(0,-1)

export var direction = 1
var is_dead = false

var max_hp = 1
var hp
var max_sta = 3
var sta

var is_being_hit = false
var is_attacking = false

var state = "Melee"

const MELEE_STA = 1

var MeleeObjective : Object

var player_position = Vector2()

var x_dist = 0

var is_positioned = false

func _ready():
    if direction == -1:
        $MeleeRayCast.cast_to.x *= -1
        $Position2D.position.x *= -1
        $AnimatedSprite.flip_h = true

    hp = max_hp
    sta = max_sta
```

Ilustración 25 Código del personaje jefe final - Boss parte 1

```gdscript
func hit(dmg,type,direction):

    hp -= dmg * Global.modifyer(type,state)
    
    is_being_hit = true
    $AnimatedSprite.play("hit")

    if hp <= 0: # Die
        is_dead = true
        motion = Vector2(0,0)
        $AnimatedSprite.play("dead")
        if direction == 1:
            $AnimatedSprite.rotation_degrees = 160
        else:
            $AnimatedSprite.rotation_degrees = -160

        $CollisionShape2D.rotation_degrees = 90
        $AnimatedSprite.position.y = 4
        
        #$CollisionShape2D.set_deferred("disabled", true) # Important fer-ho deferred
        set_collision_mask_bit(0, false)
        set_collision_layer_bit(0, false)
        set_collision_mask_bit(2, true)
        set_collision_layer_bit(2, true)
        $DespawnTimer.start()
        
        get_tree().call_group("player","recieve_souls",soul_value)
    #else:
    if not is_positioned:
        motion.x = Global.KNOCKBACK_X * direction
        motion.y = Global.KNOCKBACK_Y
    else:
        motion.x = Global.KNOCKBACK_X * direction
        motion.y = Global.KNOCKBACK_Y * 0.7
        #is_positioned = false

func _physics_process(delta):
    motion.y += G
    if not is_dead:
        if not is_attacking and not is_being_hit:
            player_position = get_tree().get_nodes_in_group("player")[0].global_position
            
            x_dist = player_position.x - global_position.x
            
            is_positioned = false
            ########## MOVEMENT
```

Ilustración 26 Código del personaje jefe final - Boss parte 2

```
            if x_dist > 0: # Positivo -> player a la derecha del enemigo
                if direction == -1:
                    change_direction()
            else:
                if direction == 1:
                    change_direction()

            if is_on_floor():
                if direction == 1:
                    motion.x = min(motion.x + ACC, MAX_SPEED)
                else:
                    motion.x = max(motion.x - ACC, -MAX_SPEED)

            $AnimatedSprite.play("walk")

            if $FloorRayCast.is_colliding() and get_slide_count() > 0:
                for i in range(get_slide_count()):
                    var collision = get_slide_collision(i)
                    if collision.collider != null and "Player" in collision.collider.name:
                        motion = motion.bounce(collision.normal)
                        motion.y *= 2
                        motion.x *= 2

            ########## ATTACK
            if $MeleeRayCast.is_colliding():
                MeleeObjective = $MeleeRayCast.get_collider()
                if MeleeObjective in get_tree().get_nodes_in_group("player") and (sta - MELEE_STA >= 0):
                    $StaRegen.start()
                    sta -= MELEE_STA
                    is_attacking = true
                    $AnimatedSprite.play("attack")
                    #motion = Vector2(0,0)
                    motion.x = 0
                    MeleeObjective.hit(MELEE_DMG,"Melee",direction)

            if direction == 1:
                $AnimatedSprite.flip_h = false
            else:
                $AnimatedSprite.flip_h = true
        else:
            if motion.x > 0:
                motion.x -= 2
            elif motion.x < 0:
                motion.x += 2
        motion = move_and_slide(motion, UP)
```

Ilustración 27 Código del personaje jefe final - Boss parte 3

```

func change_direction():
    direction *= -1
    $MeleeRayCast.cast_to.x *= -1
    $Position2D.position.x *= -1

func _on_Despawn_timeout():
    queue_free()


func _on_animation_finished():
    is_attacking = false
    is_being_hit = false


func _on_StaRegen_timeout():
    if sta + 1 <= max_sta:
        sta = sta + 1

```

Ilustración 28 Código del personaje jefe final - Boss parte 4

```gdscript
extends KinematicBody2D

var motion = Vector2()
const MAX_SPEED = 400
const G = 10
const JUMP_FORCE = -400
const ACC = 50
const UP = Vector2(0, -1) # Vector que apunta cap a dalt
const RESPAWN_PRICE = 250

var is_attacking = false # La acción de beber estu entra aquí también
var is_dead = false
var is_contact_on_CD = false
var is_being_hit = false
var is_in_deathScreen = false

export (NodePath) var SpawnPath

const FIREBALL = preload("res://Fireball.tscn")
const ARROW = preload("res://Arrow.tscn")

export var max_hp = 10.0
export var max_sta = 10.0
export var max_estus = 5
onready var hp = max_hp
onready var sta = max_sta
onready var estus = max_estus

var souls = 1000 # Cuando haya sistema de guardado, sacarlo de allí


const MAGIC_STA = 3
const RANGED_STA = 2
const MELEE_STA = 1

export var MELEE_DMG = 1

var state = "Melee" # or "Magic" or "Ranged"
```

Ilustración 29 Código del Personaje Héroe parte 1

```gdscript

var SaveData
var start_position

var cages_cleared = 0

var barrier

var time = 0

func _ready():
$CanvasLayer / FadeOut / AnimationPlayer.play("fade_out")
if SaveData != null:
    hp = SaveData["hp"]
estus = SaveData["estus"]
sta = SaveData["sta"]
souls = SaveData["souls"]
cages_cleared = SaveData["cages_cleared"]
time = SaveData["time"]
if start_position != null:
    global_position = start_position

barrier = $"../Barrier"

func _process(delta): # este se ejecuta si o si 60 veces por segundo
if not is_dead:
    time += delta
var seconds = fmod(time, 60)
var minutes = time / 60
$CanvasLayer / TimerGUI / Label.text = "%02d:%02d" % [minutes, seconds]

func _physics_process(_delta):
motion.y += G

if not "test" in get_parent().name:
if $Camera2D.limit_left < global_position.x - 464:
    $Camera2D.limit_left = global_position.x - 464
barrier.global_position.x = global_position.x - 464 - 5

```

Ilustración 30 Código del Personaje Héroe parte 2

```
78    if not is_dead:
79  #if Input.is_action_pressed("ui_back_to_menu"):
80      if Input.is_action_just_pressed("ui_back_to_menu"):
81          get_tree().change_scene("res://Menu.tscn")
82
83    var friction = false
84  if Input.is_action_pressed("ui_right"):
85      motion.x = min(motion.x + ACC, MAX_SPEED)
86    $Sprite.flip_h = false;
87  if is_attacking == false && is_being_hit == false:
88      $Sprite.play("run")
89    if sign($Position2D.position.x) == -1: # Si es negativa(a la izq del pj)
90    $Position2D.position.x *= -1       # Le cambia el signo(* -1)
91    $MeleeRayCast.cast_to.x *= -1
92    # Funciona porque el pj está centrado
93
94    elif Input.is_action_pressed("ui_left"):
95    motion.x = max(motion.x - ACC, -MAX_SPEED)
96    $Sprite.flip_h = true;
97  if is_attacking == false && is_being_hit == false:
98      $Sprite.play("run")
99    if sign($Position2D.position.x) == 1: # Si es positiva(a la der del pj)
100   $Position2D.position.x *= -1       # Le cambia el signo(* -1)
101 $MeleeRayCast.cast_to.x *= -1
102
103         else:
104 if is_attacking == false && is_being_hit == false:
105     $Sprite.play("idle")
106   friction = true
107
108
109
110 if is_on_floor():
111     if Input.is_action_just_pressed("ui_up") or Input.is_action_just_pressed("ui_a_button"):
112   motion.y = JUMP_FORCE
113 if friction:
114     motion.x = lerp(motion.x, 0, 0.2) # linear interpolation
115 # Vamos de motion.x a 0 un 20 % cada frame
```

Ilustración 31 Código del Personaje Héroe parte 3

```
116         else:
117   if is_attacking == false && is_being_hit == false:
118     if motion.y < 0:
119       $Sprite.play("jump")
120     else:
121     $Sprite.play("fall")
122   if friction:
123     motion.x = lerp(motion.x, 0, 0.05)
124   # Para que funcione is_on_floor(arriba), se le debe pasar a move_and_slide
125   # Un vector apuntando hacia arriba para que sepa en qué dirección está el suelo
126   # move_and_slide devuelve la posición después de hacer el movimiento, habiendo calculado colisiones
127   # motion = move_and_slide(motion, UP) # Este método(de kb2D) ya tiene en cuenta delta
128
129   if Input.is_action_just_pressed("ui_fire") && is_attacking == false:
130   # $StaRegen.start() # Volver a empezar
131   match state:
132   "Magic":
133   if sta - MAGIC_STA >= 0:
134     $StaRegen.start() # Volver a empezar
135   sta -= MAGIC_STA
136   $CanvasLayer / GUI.update_sta(sta)
137
138   is_attacking = true
139   $Sprite.play("shoot")
140
141   var fireball = FIREBALL.instance() # Creación del objeto! (Como new de C)
142
143   fireball.launcher = "player"
144
145   if sign($Position2D.position.x) == 1:
146     fireball.set_fireball_direction(1)
147   else:
148   fireball.set_fireball_direction(-1)
149
150   get_parent().add_child(fireball)
151
152   fireball.position = $Position2D.global_position
153   "Ranged":
154   if sta - RANGED_STA >= 0:
```

Ilustración 32 Código del Personaje Héroe parte 4

```
    $StaRegen.start() # Volver a empezar
sta -= RANGED_STA
$CanvasLayer / GUI.update_sta(sta)

is_attacking = true
$Sprite.play("shoot") # PROVISIONAL

var arrow = ARROW.instance() # Creación del objeto

arrow.launcher = "player"

if sign($Position2D.position.x) == 1:
    arrow.set_arrow_direction(1)
else:
arrow.set_arrow_direction(-1)

get_parent().add_child(arrow)

arrow.position = $Position2D.global_position
"Melee":
if sta - MELEE_STA >= 0:
    $StaRegen.start() # Volver a empezar
sta -= MELEE_STA
$CanvasLayer / GUI.update_sta(sta)

is_attacking = true
$Sprite.play("melee")

if $MeleeRayCast.is_colliding():
    var col = $MeleeRayCast.get_collider()
if col in get_tree().get_nodes_in_group("enemies"):
    col.hit(MELEE_DMG, "Melee", sign($Position2D.position.x))

if Input.is_action_just_pressed("ui_drinkEstus") \
        && is_attacking == false && is_being_hit == false:
if estus - 1 >= 0 and hp < max_hp: #ojut
estus -= 1
$CanvasLayer / GUI.update_estus(estus)
is_attacking = true
```

Ilustración 33 Código del Personaje Héroe parte 5

```
$Sprite.play("drink")
heal(5)


if Input.is_action_just_pressed("ui_chngModeRight"):
    match state:
"Magic":
state = "Ranged"
"Ranged":
state = "Melee"
"Melee":
state = "Magic"

if Input.is_action_just_pressed("ui_chngModeLeft"):
    match state:
"Magic":
state = "Melee"
"Ranged":
state = "Magic"
"Melee":
state = "Ranged"


#########################################################################
#############################DEBUG#######################################
if OS.is_debug_build() and Input.is_action_just_pressed("debug_save"):
Save.save_data(hp, estus, sta, souls, cages_cleared, time)
print("data saved")
#########################################################################
#########################################################################


#if get_slide_count() > 0:
# for i in range(get_slide_count()):
#   if get_slide_collision(i).collider in get_tree().get_nodes_in_group("enemies"):
#     die()

#if Input.is_action_just_pressed("ui_respawn"): [de moment desactivo el respawn durant el joc]
```

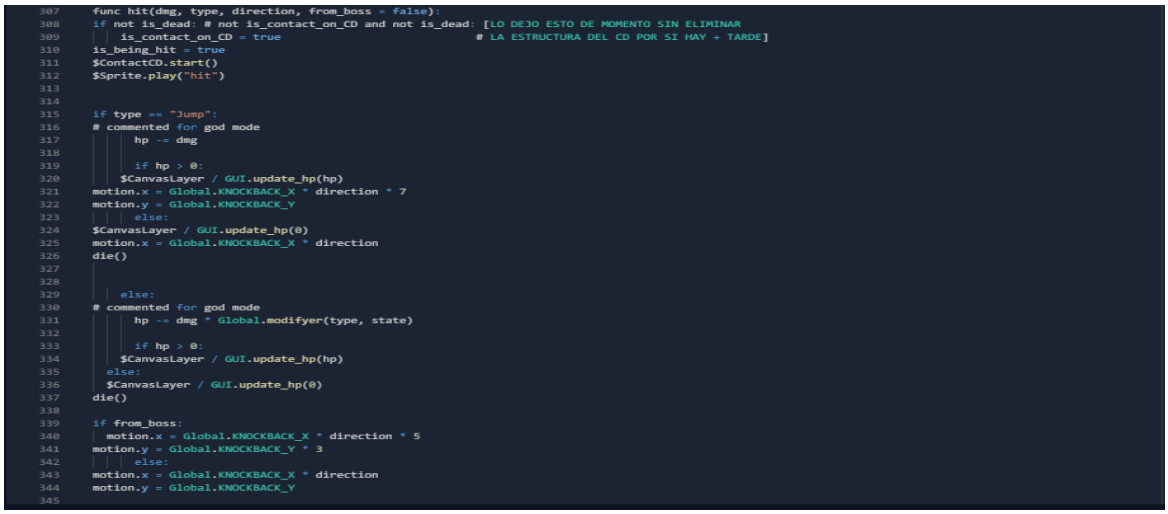
Ilustración 34 Código del Personaje Héroe parte 6

```
# respawn()

$Label.text = state

elif is_in_deathScreen: # Si està mort I a la deathScreen
$CanvasLayer / ColorRect.visible = true

if Global.controller_connected:
  $CanvasLayer / ColorRect / CanPay / Start.visible = true
$CanvasLayer / ColorRect / CanPay.text = "HAS MUERTO\n\nPULSA  ·   PARA REVIVIR\n[250 ALMAS]\n"
$CanvasLayer / ColorRect / CantPay / Start.visible = true
$CanvasLayer / ColorRect / CantPay.text = "HAS MUERTO\n\nNO TIENES LAS 250 ALMAS \nNECESARIAS PARA REVIVIR\n\nPULSA   ·   PARA \nMORIR DEFINITIVAMENTE"
    else:
$CanvasLayer / ColorRect / CanPay / Start.visible = false
$CanvasLayer / ColorRect / CanPay.text = "HAS MUERTO\n\nPULSA ENTER PARA REVIVIR\n[250 ALMAS]\n"
$CanvasLayer / ColorRect / CantPay / Start.visible = false
$CanvasLayer / ColorRect / CantPay.text = "HAS MUERTO\n\nNO TIENES LAS 250 ALMAS \nNECESARIAS PARA REVIVIR\n\nPULSA ENTER PARA \nMORIR DEFINITIVAMENTE\n"


if souls - RESPAWN_PRICE >= 0:
  $CanvasLayer / ColorRect / CanPay.visible = true
$CanvasLayer / ColorRect / CantPay.visible = false

if Input.is_action_just_pressed("ui_respawn"):
  spend_souls(RESPAWN_PRICE)
var temp_data = Save.load_data()
if typeof (temp_data) != TYPE_DICTIONARY: # si no se ha cargado bien
SaveData["souls"] = souls
SaveData["time"] = time
Save.save_dict_data(SaveData) # Guarde lo hemos cargado al principio
print(SaveData)
        else:
temp_data["souls"] = souls
temp_data["time"] = time
Save.save_dict_data(temp_data)

respawn()
    else:
```

Ilustración 35 Código del Personaje Héroe parte 7

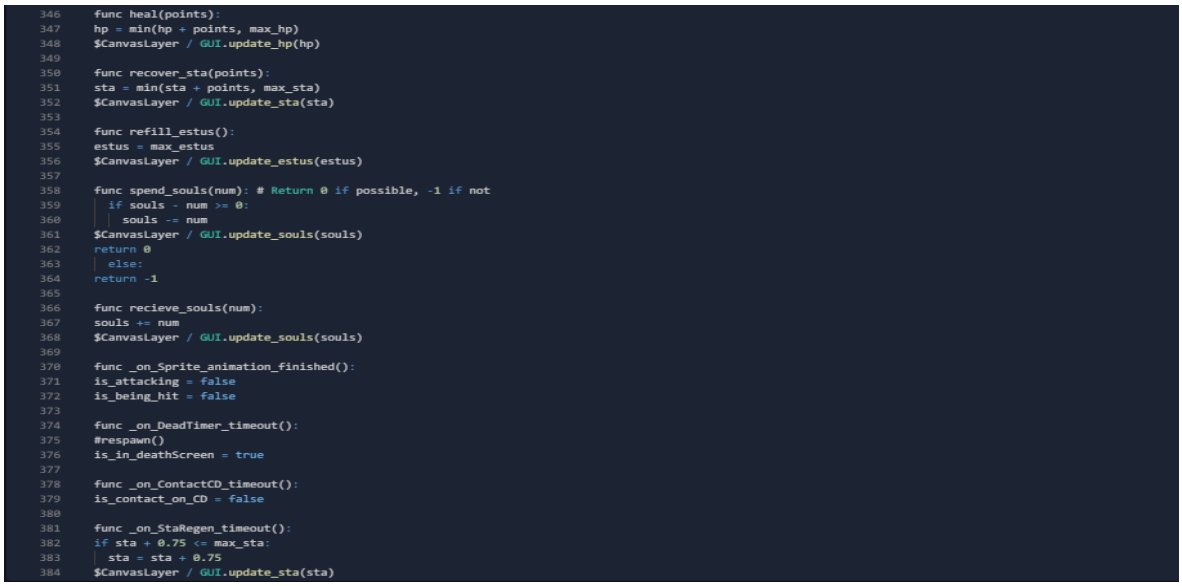
```
$CanvasLayer / ColorRect / CanPay.visible = false
$CanvasLayer / ColorRect / CantPay.visible = true
if Input.is_action_just_pressed("ui_respawn"):
  Save.delete_data()
get_tree().quit(0)

  else: # Si está muerto pero no a la deathScreen.
#Fricción para que no se mueva infinitamente por knockback
if motion.x > 0:
  motion.x -= 2
elif motion.x < 0:
motion.x += 2

motion = move_and_slide(motion, UP)
if get_slide_count() > 0:
  for i in range(get_slide_count()):
    if get_slide_collision(i).collider in get_tree().get_nodes_in_group("instakill") and not is_dead:
die()

func respawn():
# is_dead = false
# is_in_deathScreen = false
# $CanvasLayer / ColorRect.visible = false
# $CollisionShape2D.rotation_degrees = 0
get_tree().reload_current_scene()

func die():
if is_dead == false: # Por si se repite la llamada

is_dead = true
motion = Vector2(0, 0)
$Sprite.play("dead")
position = Vector2(position.x, position.y + 24)
$CollisionShape2D.rotation_degrees = 90
$DeadTimer.start()
#$CollisionShape2D.set_deferred("disabled", true)

```

Ilustración 36 Código del Personaje Héroe parte 8



```
func hit(dmg, type, direction, from_boss = false):
if not is_dead: # not is_contact_on_CD and not is_dead: [LO DEJO ESTO DE MOMENTO SIN ELIMINAR
    is_contact_on_CD = true                          # LA ESTRUCTURA DEL CD POR SI HAY + TARDE]
is_being_hit = true
$ContactCD.start()
$Sprite.play("hit")


if type == "Jump":
# commented for god mode
        hp -= dmg

        if hp > 0:
    $CanvasLayer / GUI.update_hp(hp)
motion.x = Global.KNOCKBACK_X * direction * 7
motion.y = Global.KNOCKBACK_Y
        else:
$CanvasLayer / GUI.update_hp(0)
motion.x = Global.KNOCKBACK_X * direction
die()


    else:
# commented for god mode
        hp -= dmg * Global.modifyer(type, state)

        if hp > 0:
    $CanvasLayer / GUI.update_hp(hp)
  else:
   $CanvasLayer / GUI.update_hp(0)
die()

if from_boss:
  motion.x = Global.KNOCKBACK_X * direction * 5
motion.y = Global.KNOCKBACK_Y * 3
        else:
motion.x = Global.KNOCKBACK_X * direction
motion.y = Global.KNOCKBACK_Y
```

Ilustración 37 Código del Personaje Héroe parte 9



```
func heal(points):
hp = min(hp + points, max_hp)
$CanvasLayer / GUI.update_hp(hp)

func recover_sta(points):
sta = min(sta + points, max_sta)
$CanvasLayer / GUI.update_sta(sta)

func refill_estus():
estus = max_estus
$CanvasLayer / GUI.update_estus(estus)

func spend_souls(num): # Return 0 if possible, -1 if not
  if souls - num >= 0:
    souls -= num
$CanvasLayer / GUI.update_souls(souls)
return 0
  else:
return -1

func recieve_souls(num):
souls += num
$CanvasLayer / GUI.update_souls(souls)

func _on_Sprite_animation_finished():
is_attacking = false
is_being_hit = false

func _on_DeadTimer_timeout():
#respawn()
is_in_deathScreen = true

func _on_ContactCD_timeout():
is_contact_on_CD = false

func _on_StaRegen_timeout():
if sta + 0.75 <= max_sta:
  sta = sta + 0.75
$CanvasLayer / GUI.update_sta(sta)
```

Ilustración 38 Código del Personaje Héroe parte 10

```


func _on_FadeOut_animation_finished(anim_name):
$CanvasLayer / FadeOut.hide()

```

Ilustración 39 Código del Personaje Héroe parte 11

9.4.2.8 Capturas de Pantalla

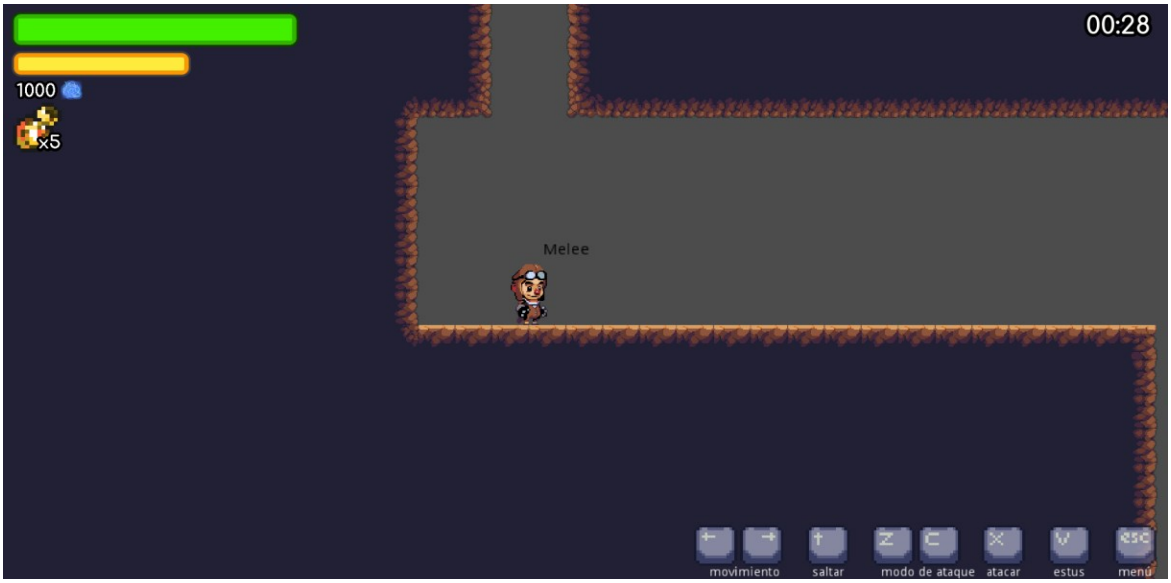


Ilustración 40 Captura de pantalla 1 iteración 2

Ilustración 41 Captura de pantalla 2 iteración 2

Ilustración 42 Captura de pantalla 3 iteración 2

Ilustración 43 Captura de pantalla 3 iteración 2

9.4.2.9 Resultado

Como resultado de entrega de la segunda iteración del Videojuego 2D versión Demo de género soulslike se desarrolló la interacción del Personaje Héroe, Personaje Enemigos y Personaje Jefe Final -Boss (movimiento y ataque) dentro del mundo creado, de igual manera se logró implementar el agente inteligente de aumento progresivo de la fortaleza de los Personajes enemigos y Boss y la reanudación de la partida acorde a las condiciones establecidas durante la iteración.

9.4.3 Tercera Iteración

En esta tercera y última iteración se desarrollaron la historia de usuario 6 “Visualización de Ranking” y la historia de usuario 7 “Finalizar partida”.

9.4.3.1 Tareas de Ingeniería

Se definieron las siguientes tareas de ingeniería para las historias 6, 7 y 8:

Tabla 34 Tareas de Ingeniería Historias de Usuario 6, 7 y 8

NÚMERO DE HISTORIA	NÚMERO DE TAREA	NOMBRE DE LA TAREA
6	15	Diseño de la interfaz de visualización de la clasificación (Ranking)
6	16	Desarrollo del historial de puntuación por tamaño de mundo
7	17	Diseño de la interfaz de finalización del juego
7	18	Desarrollo del algoritmo de finalización de la partida

9.4.3.2 Descripción Tareas de Ingeniería

Tabla 35 Descripción Tarea de Ingeniería 15

TAREA DE INGENIERÍA	
Número de Tarea: 15	Número de Historia: 6
Nombre de Tarea: Diseño de la interfaz de visualización de Clasificación (Ranking)	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 Semana 4	Fecha Fin: Mes 3 Semana 4
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Diseñar la interfaz de visualización de los últimos 5 registros de puntuación por tamaño de mundo obtenido por el jugador en la partida jugada	

Tabla 36 Descripción Tarea de Ingeniería 16

TAREA DE INGENIERÍA	
Número de Tarea: 16	Número de Historia: 6
Nombre de Tarea: Desarrollo del historial de puntuación por tamaño de mundo	
Tipo de Tarea: Desarrollo	Punto Estimados: 0.5
Fecha Inicio: Mes 2 Semana 4	Fecha Fin: Mes 3 Semana 4
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Desarrollo del historial de puntuación obtenido por el jugador durante la partida, conservando los tres puntajes más altos, los cuales deber ser mostrados en la interfaz de clasificación (ranking)	

Tabla 37 Descripción Tarea de Ingeniería 17

TAREA DE INGENIERÍA	
Número de Tarea: 17	Número de Historia: 7
Nombre de Tarea: Diseño de la interfaz de finalización del juego	
Tipo de Tarea: Desarrollo	Punto Estimados:
Fecha Inicio: Mes 2 Semana 4	Fecha Fin: Mes 3 Semana 4
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Diseñar interfaz que muestre el mensaje "has muerto" junto con el número de almas necesarias para revivir y un mensaje "has muerto", no tienes suficientes almas para revivir" si no cuenta con las suficientes almas para continuar el juego. “En hora buena has derrotado al protector del sello” si gana la partida	

Tabla 38 Descripción Tarea de Ingeniería 18

TAREA DE INGENIERÍA	
Número de Tarea: 18	Número de Historia: 7
Nombre de Tarea: Desarrollo del algoritmo de finalización de la partida	
Tipo de Tarea: Desarrollo	Punto Estimados:
Fecha Inicio: Mes 2 Semana 4	Fecha Fin: Mes 3 Semana 4
Programador Responsable: Jonatan Huergo Aguilar	
Descripción: Desarrollo de un proceso que controle la reanudación del juego, descontando del total acumulado de número de almas, la cantidad mínima de almas para continuar con el juego en el último lugar donde murió previamente el Personaje Héroe. Si el número de almas acumuladas es insuficiente para revivir, finaliza completamente el juego. Si el jugador gana la partida mostrar opción de guardar resultado y continuar a un nuevo nivel	

9.4.3.3 Tarjetas CRC

Tabla 39 Tarjeta CRC Iteración 3

Clasificación - Ranking	
RESPONSABILIDAD	**COLABORACIÓN**
Acumular la puntuación obtenida por el jugador por tamaño de mundo	Personaje (Héroe, Enemigos, Boss)
Guardar los tres puntajes mayores por tamaño de mundo	

9.4.3.4 Pruebas de aceptación

Las pruebas generales para la iteración 3 son las siguientes:

Tabla 40 Pruebas Iteración 3

NÚMERO DE LA PRUEBA	NÚMERO DE LA HISTORIA	NOMBRE DE LA PRUEBA
6	6	Visualización de Clasificación (Ranking)
7	7	Finalizar partida

9.4.3.5 Descripción pruebas de aceptación

Tabla 41 Descripción caso de prueba 6 Historia de Usuario 6

CASO DE PRUEBA	
Código: 6	**No de Historia de Usuario**: 6
Historia de Usuario: Visualización de Clasificación (Ranking)	
Condiciones de Ejecución: Se deben visualizar los tres puntajes máximos obtenidos por el jugador por tamaño de mundo	
Entrada/Pasos de Ejecución: Seleccionar la opción Ranking con la tecla Enter en el menú principal, visualizar una tabla que contiene como encabezado los tres tipos de tamaño de mundo (corto - mediano - largo) y debajo de cada tamaño, los tres puntajes mayores obtenidos por el jugador	
Resultado Esperado: ingresar correctamente a la vista de clasificación (Ranking) y visualizar en la tabla de resultados los 3 puntajes máximos obtenidos por el jugador según el tamaño de mundo	
Evaluación de la prueba: Prueba superada satisfactoriamente	

Tabla 42 Descripción caso de prueba 7 Historia de Usuario 7

CASO DE PRUEBA	
Código: 7	**No de Historia de Usuario**: 7
Historia de Usuario: Finalizar partida	
Condiciones de Ejecución: La partida finaliza cuando el personaje héroe no cuenta con suficiente cantidad de almas para revivir o gana la partida	
Entrada/Pasos de Ejecución: Después de morir el personaje, se visualiza una interfaz con la opción de revivir pulsando la tecla Enter, siempre y cuando el número de almas sea suficiente, sino se visualiza una interfaz con el mensaje de muerte definitiva, finalizando el juego pulsando la tecla Enter. Cuando el jugador termina el nivel se visualiza el mensaje “En hora buena has derrotado al protector del sello…” y la opción “ingresar el nombre para registrar el puntaje obtenido en el ranking.	
Resultado Esperado: Visualización del mensaje "Has muerto" - Cantidad de Almas - "Pulsa la tecla Enter para revivir". Visualización del mensaje "Has muerto" - Cantidad de Almas - "No cuenta con almas suficientes para revivir - Pulsa la tecla Enter para morir completamente". Visualización mensaje “En hora buena has derrotado al protector del sello …” “Ingresa tu nombre para ir al ranking”	
Evaluación de la prueba: Prueba superada satisfactoriamente	

9.4.3.6 Diseño

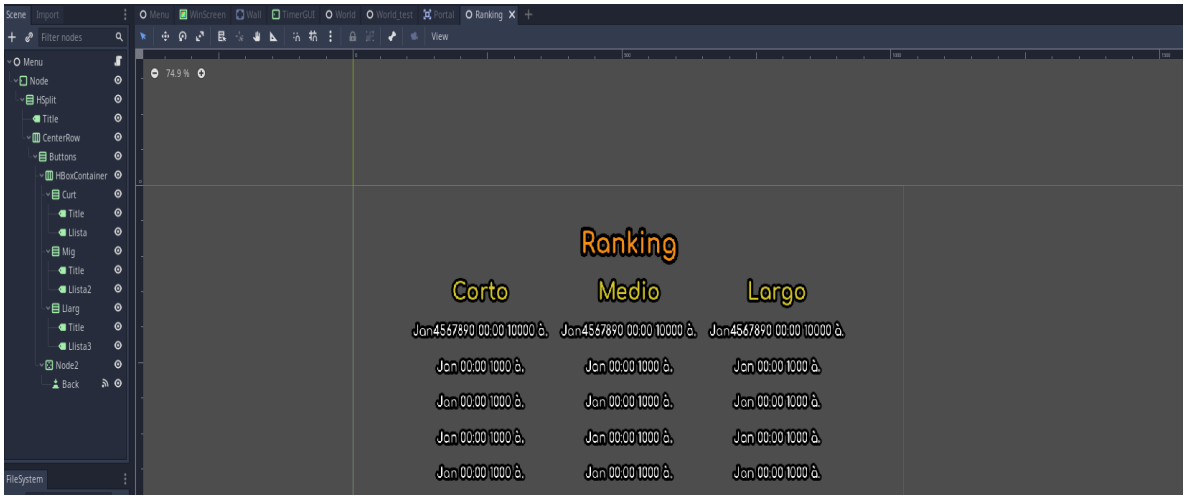


Ilustración 44 Diseño vista ranking

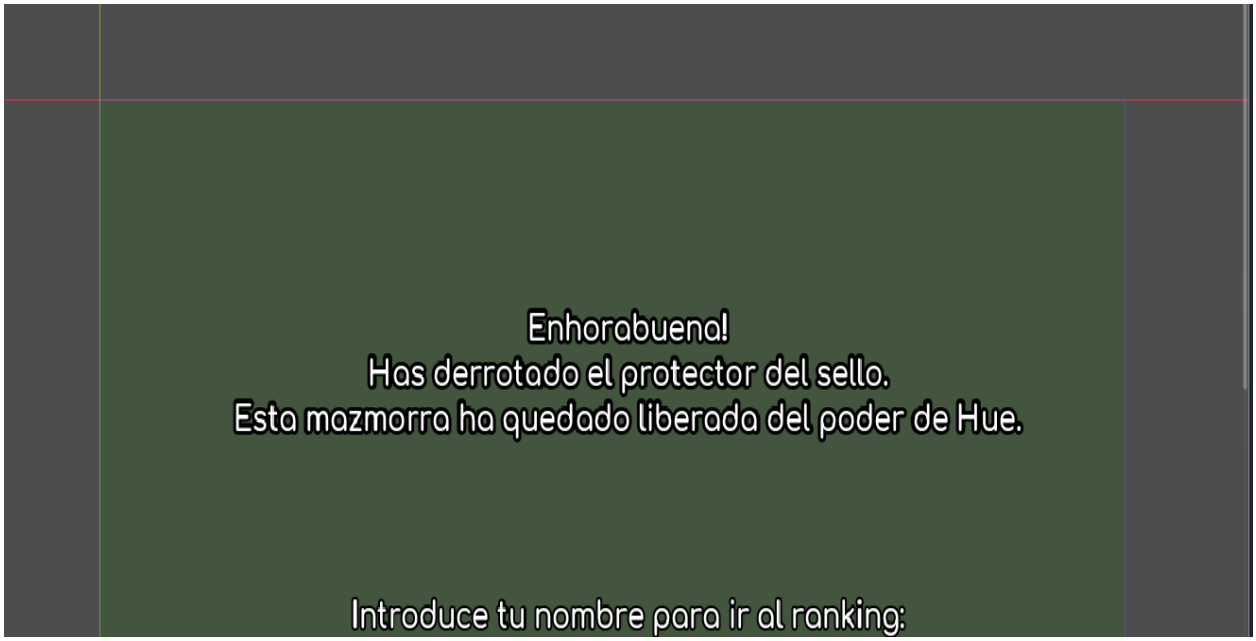


Ilustración 45 Diseño vista finalización de la partida

9.4.3.7 Codificación

```gdscript
extends Node

export (String, FILE, "*.tscn") var menu_scene

# Called when the node enters the scene tree for the first time.
func _ready():
	var short_out = ""
	var mid_out = ""
	var long_out = ""

	var ranking = Save.load_ranking()

	if typeof(ranking) != TYPE_DICTIONARY:
		short_out = "Aún no hay entradas"
		mid_out = "Aún no hay entradas"
		long_out = "Aún no hay entradas"

	else:
		print(ranking)
		if "Short" in ranking:
			for entry in ranking["Short"]:
				var time = entry["Time"]
				var seconds = fmod(time,60)
				var minutes = time / 60
				var time_text = "%02d:%02d" % [minutes, seconds]
				short_out += "%s %s %d à.\n" % [entry["Name"], time_text, entry["Souls"]]
		else:
			short_out = "Aún no hay entradas"

		if "Mid" in ranking:
			for entry in ranking["Mid"]:
				var time = entry["Time"]
				var seconds = fmod(time,60)
				var minutes = time / 60
				var time_text = "%02d:%02d" % [minutes, seconds]
				mid_out += "%s %s %d à.\n" % [entry["Name"], time_text, entry["Souls"]]
		else:
			mid_out = "Aún no hay entradas"

		if "Long" in ranking:
			for entry in ranking["Long"]:
				var time = entry["Time"]
				var seconds = fmod(time,60)
				var minutes = time / 60
				var time_text = "%02d:%02d" % [minutes, seconds]
				long_out += "%s %s %d à.\n" % [entry["Name"], time_text, entry["Souls"]]
		else:
			long_out = "Aún no hay entradas"

	$Node/HSplit/CenterRow/Buttons/HBoxContainer/Curt/Llista.text = short_out
	$Node/HSplit/CenterRow/Buttons/HBoxContainer/Mig/Llista2.text = mid_out
	$Node/HSplit/CenterRow/Buttons/HBoxContainer/Llarg/Llista3.text = long_out

	$Node/HSplit/CenterRow/Buttons/Node2/Back.grab_focus()

func _on_Back_pressed():
	get_tree().change_scene(menu_scene)

```

Ilustración 46 Código del ranking

1.1.1. Capturas de Pantalla

Ilustración 47 Captura de pantalla Ranking Iteración 3

Ilustración 48 Captura de pantalla finalización 1 Iteración 3

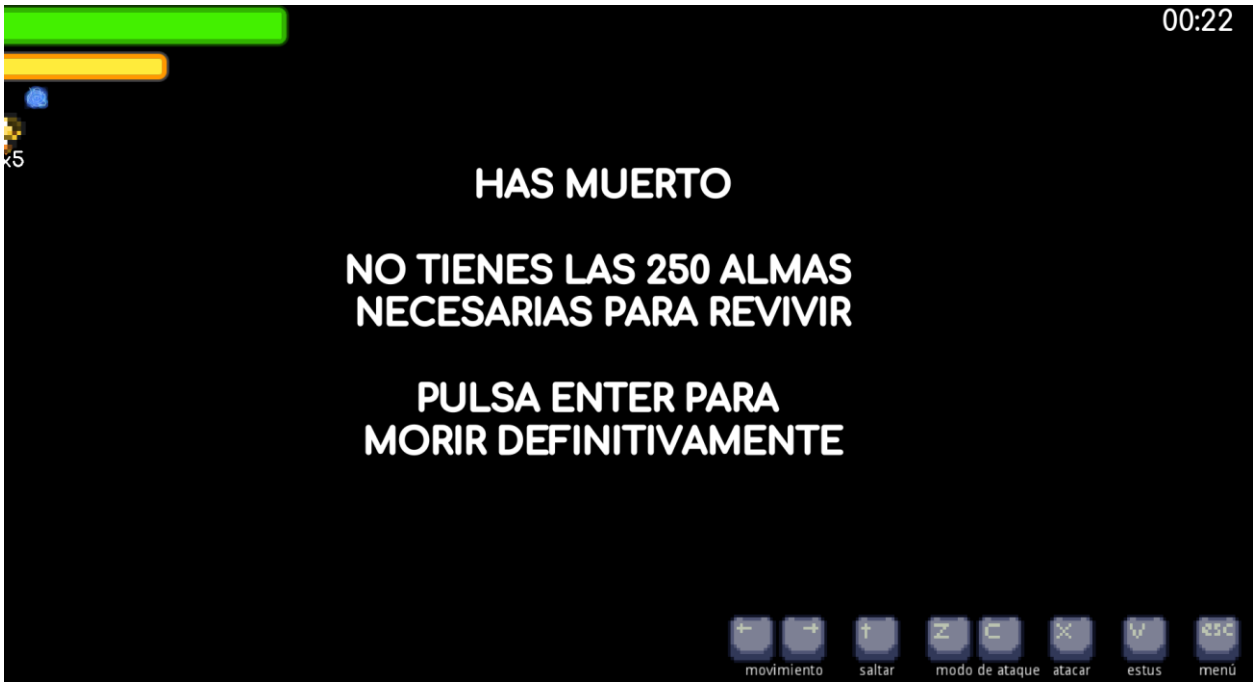


Ilustración 49 Captura de pantalla finalización 2 Iteración 3

9.4.3.8 Resultado

Como resultado de entrega de la tercera y última iteración del Videojuego 2D versión Demo de género soulslike se desarrolló interfaz de visualización de la clasificación (Ranking) e historial de puntuación por tamaño de mundo, la interfaz de finalización del juego y se implementó el algoritmo de finalización de la partida

10 CONCLUSIONES

Por ser un videojuego creado en un motor multiplataforma y de código abierto se pueden crear nuevas versiones aumentando la cantidad de personajes, enemigos y objetos o desarrollando otras funcionalidades que estén dentro de la filosofía del género soulslike.

De igual manera es posible generar ejecutables para correr en dispositivos de gama baja o media con sistemas operativos Windows, Linux, OS X, Android, iOS y HTML5 con memoria RAM mayor a 1GB, Procesador mayor de 1GHz de 32 bits y espacio de almacenamiento de disco duro mayor a 1 GB adicional al requerido por el sistema operativo, garantizando buena calidad gráfica y bajo consumo de recursos de hardware

Por otra parte, como elemento adicional para la optimización del agente inteligente, se permite modificar el código fuente del videojuego por medio del uso de datasets que contengan el mayor número posible de registros estadísticos de comportamiento y fortaleza del (los) personaje(s) y/o enemigos que mejore la experiencia de juego, teniendo en cuenta los recursos

En la generación procedimental de mundos también existe la posibilidad de usar data sets que contengan registros relacionados con la creación aleatoria de mapas para que el videojuego no se vuelva rutinario y monótono para el jugador.

Para futuras versiones es posible la integración de base de datos para la administración de usuarios y control estadístico de las partidas jugadas en línea como valor adicional a la experiencia de juego por medio de funciones multijugador.

En conclusión, el videojuego 2D del género soulslike es de software libre, por lo que el código fuente queda abierto para ser modificado y/o usado en la creación de futuros videojuegos de género soulslike, como código fuente base para la construcción de infinidad de videojuegos de otros géneros o como recurso de investigación en temas relacionados con la aplicación de modelos de inteligencia artificial en videojuegos 2D y 3D.

11 FINANCIACIÓN Y COSTOS DEL PROYECTO.

La financiación en la creación del videojuego fue asumida en su totalidad por los autores del proyecto con recursos propios distribuidos según la siguiente tabla:

Tabla 43 Inversión

Financiamiento		**Valor**
Inversores		
Recursos Propios Estudiante 1		$ 3.000.000,00
Recursos Propios Estudiante 2		$ 3.000.000,00
	Total Inversión	***$* 6.000.000,00**
Total Financiamiento		**$ 6.000.000,00**

Tabla 44 Costos

Costos		**Estimado**
Costos de única vez		
Licencias y permisos		$ 0,00
Compras de Hardware (Portátil)		$ 3.000.000,00
Muebles de oficina		$ 700.000,00
Herramientas y suministros varios		$ 20.000,00
	Costos Fijos Totales	**$ 3.720.000,00**
Costo Mensual Promedio		
Salario 2 desarrolladores de software		$ 3.000.000,00
Conexión de internet		$ 150.000,00
Servicio de energía eléctrica		$ 60.000,00
Gastos relacionados al desarrollo del videojuego		$ 50.000,00
	Costo Mensual Promedio	**$ 3.260.000,00**
	Número de meses	3
	Total Costo mensual Promedio	**$ 9.780.000,00**
Costos Totales		**$ 13.500.000,00**
Resultado de la Inversión vs Costos		***-$ 7.500.000,00***

12 BIBLIOGRAFÍA

Blizzard Entertainment. (1996). Obtenido de http://ftp.blizzard.com/pub/misc/Diablo.PDF

Caballero, P. S. (09 de 06 de 2020). Obtenido de https://riull.ull.es/xmlui/bitstream/handle/915/19774/Generacion%20procedimental%20de%20entornos%20exteriores%20para%20videojuegos%203D.pdf?sequence=1

Doull, A. (2007). *PCG Wiki.* Obtenido de http://pcg.wikidot.com/the-death-of-the-level-designer

Elite Dangerous ESP. (2017). Obtenido de https://eliteesp.es/2017/04/09/la-historia-de-elite-i-el-elite-original/

Godot, C. (2021). *GODOT DOCS.* Obtenido de https://docs.godotengine.org/es/stable/about/list_of_features.html

Hu, R. (29 de 08 de 2017). *Cellular Automata Simulator*. Obtenido de http://robinforest.net/post/cellular-automata/

Shalke, N. (2016). *Generación de Contenido en Juegos.* Springer.

Togelius, J. (2011). *julian.togelius.* Obtenido de http://julian.togelius.com/Togelius2011Searchbased.pdf

Wichman, G. R. (1984). Obtenido de A BRIEF HISTORY OF ROGUE - Glenn R. Wichman: https://www.free-culture.ir/A_brief_history_of_rogue.html

Wikipedia. (20 de 10 de 2021). Obtenido de https://en.wikipedia.org/wiki/MIT_License

Wikipedia/Godot. (10 de 2021). Obtenido de https://es.wikipedia.org/wiki/Godot

Wikipedia/Metroidvania. (09 de septiembre de 2021). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Metroidvania

Wikipedia/Souls_Serie. (09 de 10 de 2021). Obtenido de https://es.wikipedia.org/wiki/Souls_(serie)

Wikipedia/The_Legend_of_Zelda. (17 de 07 de 2021). Obtenido de https://es.wikipedia.org/wiki/The_Legend_of_Zelda

13 ANEXOS

ANEXO A: Código fuente del videojuego COQUERSOULS 0.3.1. en formato zip

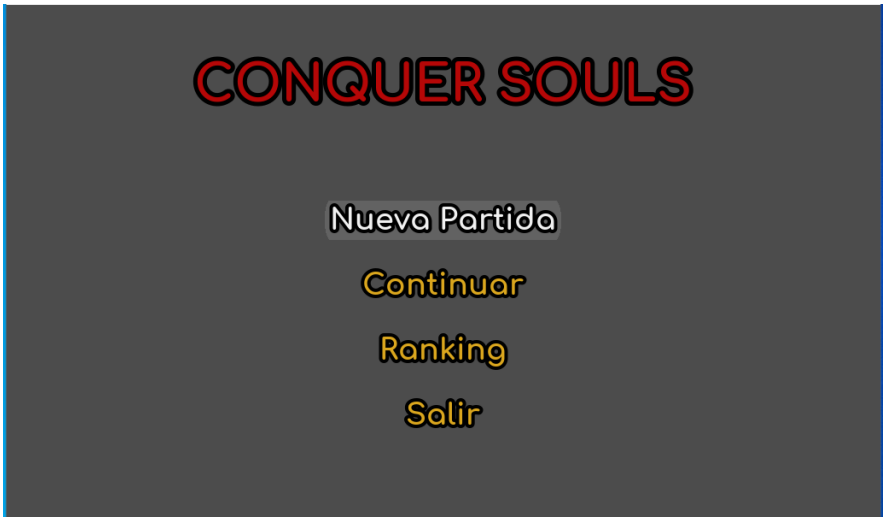
ANEXO B: Demo del videojuego COQUERSOULS 0.3.1. en forma zip para ser ejecutado en Sistemas Operativo Windows.

ANEXO C: Manual de Usuario COQUERSOULS 0.3.1

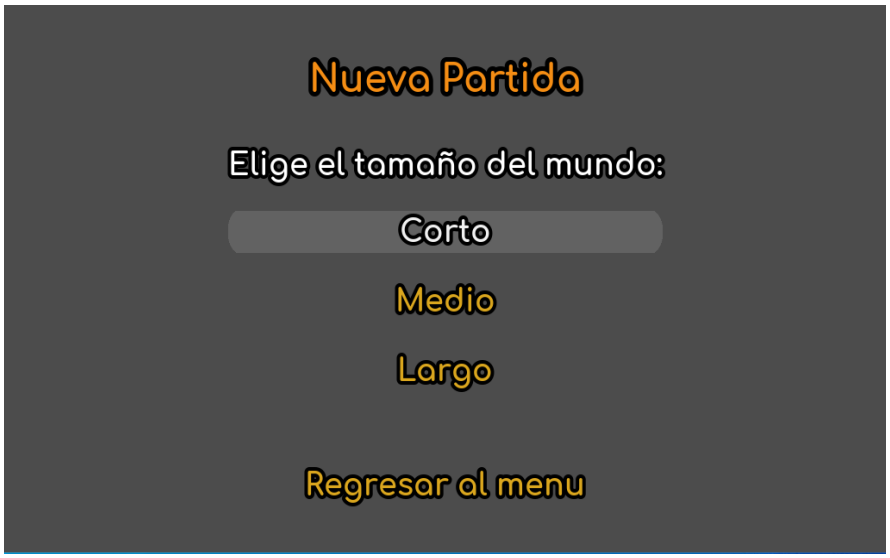
MANUAL DE USUARIO COQUERSOULS 0.3.1

El Menú principal contiene cuatro opciones:

- Nueva Partida: Inicia nueva partida

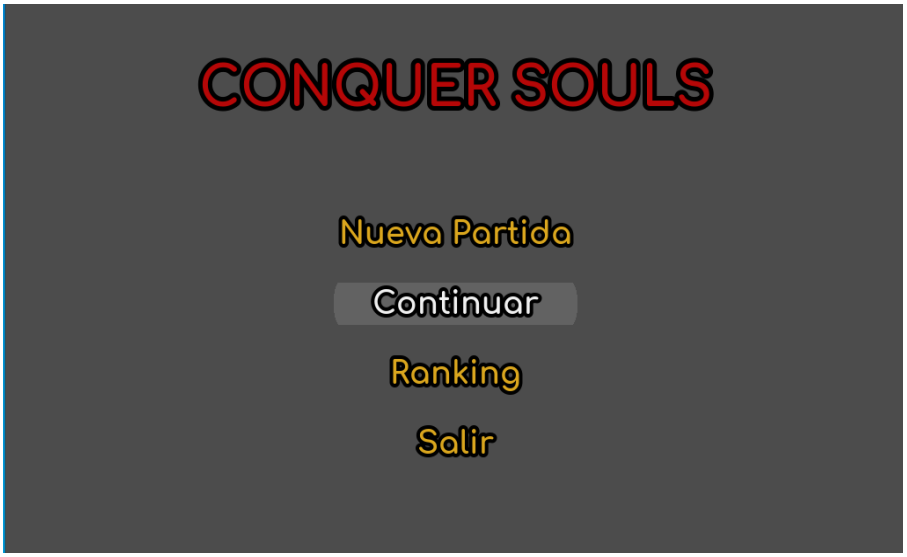


Al seleccionar “Nueva Partida” se abre el submenú “Elige el tamaño del mundo:”, el cual contiene 3 tipos de tamaño de mundo a elegir: Corto, Mediano, Largo y la opción de regresar al Menú Principal



Después de elegir el tamaño del mundo, este se crea automáticamente y se inicia el videojuego

- Continuar: Reanuda la partida previamente guardada

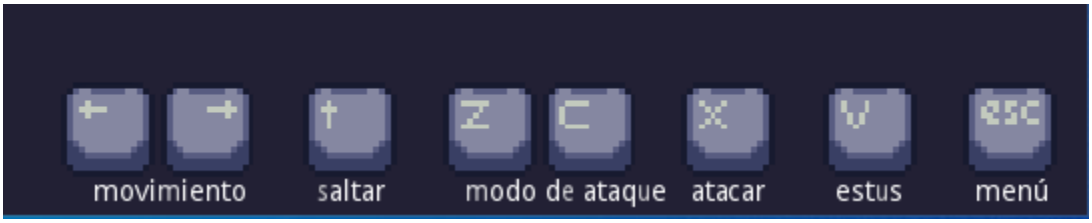


- Ranking: Visualiza los puntajes obtenidos

- Salir: Cierra el videojuego automáticamente

En la esquina superior izquierda de la ventana del videojuego en desarrollo, se encuentra la barra verde que indica el nivel de vida del personaje, la barra amarilla que indica el nivel de energía de los ataques del personaje y la cantidad de almas que tiene el personaje

En la parte inferior derecha se observan los controles para interactuar con el personaje dentro del nivel



Al presionar en el teclado las teclas: flecha a la derecha, y flecha a la izquierda. el personaje se mueve a la derecha e izquierda respectivamente. Al presionar la tecla fecha hacia arriba, el personaje salta.

Las teclas de “Z” y “C” son para cambiar el modo de ataque y la Tecla “X” es para realizar el ataque.

La tecla “V” es para reclamar las almas. Al presionar la tecla “esc” se retorna al menú principal

Para liberar los prisioneros y recuperar vida en los campamentos se presiona la tecla “B”.