Instituto Puig Castellar
Ciclo formativo: DAM2A
Grado: CFGS
Crédito de Síntesis / Proyecto intermodular

AZERIS

Desarrollo de un videojuego roguelike

Autores: Mario Taipe y Dylan Castro
Tutor: Luis Elía
Curso: 2025-2026
Fecha de entrega: 17/05/2026

Resumen del proyecto

La idea de nuestro proyecto es desarrollar un **videojuego 2D** que incluya los elementos fundamentales que caracterizan a este tipo de obras como la historia, la trama y los personajes. El estilo visual estará basado en **pixel art**, un formato que aporta un aspecto retro y atractivo, además de permitirnos diseñar nuestras propias **texturas personalizadas**.

Para su desarrollo emplearemos **LibGDX**, una herramienta con la que nos sentimos cómodos debido a nuestra familiaridad con su lenguaje de programación. Esta aplicación destaca por su versatilidad y facilidad de uso, lo que nos permitirá centrarnos en la parte creativa del proyecto sin complicaciones técnicas excesivas.

En cuanto a la programación, el personaje principal será controlado mediante **scripts personalizados**, mientras que los enemigos contarán con comportamientos específicos que les permitirán perseguir al jugador e intentar impedir su avance. El **objetivo principal** será **derrotar a los enemigos** y superar distintos niveles hasta alcanzar el enfrentamiento final contra el enemigo principal. Para lograrlo, el jugador contará con la ayuda de un **artefacto esencial**, que será clave para completar la misión y finalizar la historia.

Para llevar a cabo este proyecto, utilizaremos diversas herramientas de **software libre**, entre ellas:

- **LibGDX**
- **Krita**
- **Tiled**

Elegimos LibGDX porque nos permitirá fortalecer nuestras habilidades en programación y adquirir una mayor experiencia en el desarrollo de videojuegos 2D, combinando creatividad, lógica y diseño en un entorno profesional.

Palabras clave

- Videojuego
- Java
- LibGDX
- Base de datos
- Página web

Abstract

The idea of our project is to develop a **2D video game** that incorporates the fundamental elements that characterize this type of work, such as story, plot, and characters. The visual style will be based on **pixel art**, which provides a retro and appealing look while allowing us to create our own **custom textures**.

For development, we will use **LibGDX**, a tool we are familiar with due to our prior experience with its programming language. This application stands out for its **versatility and ease of use**, allowing us to focus on the creative aspects of the project without excessive technical difficulties.

Regarding **programming**, the main character will be controlled through **custom scripts**, while enemies will have specific behaviors that allow them to **pursue the player** and try to prevent their progress. The **primary objective** will be to **defeat the enemies** and progress through multiple levels until facing the final boss. To achieve this, the player will have the assistance of a **key artifact**, essential for completing the mission and finishing the story.
To carry out this project, we will use several **open-source tools**, including:

- **LibGDX**
- **Krita**
- **Tiled**

We chose **LibGDX** because it will allow us to **strengthen our programming skills** and gain **more experience in 2D video game development**, combining creativity, logic, and design in a professional environment.

Keywords

- Video Game
- Rogue-like
- Java
- LibGDX
- Database
- Website

Índice

1. Presentación del proyecto....4
1.1 Introducción....5
1.2 Contexto....5
1.3 Justificación....6
1.4 Objetivos....6
2. Estrategia y planificación....7
2.1 Estrategia de desarrollo y viabilidad....7
2.2 Metodología de trabajo....7
2.3 Planificación....8
3. Análisis....1
3.1 Casos de uso....1
3.2 Requisitos funcionales....1
3.3 Requisitos no funcionales....2
3.4 Análisis de alternativas tecnológicas....3
4. Diseño....4
4.1 Arquitectura del sistema....4
4.2 Modelo de datos....4
4.3 Diseño de interfaz....5
5. Desarrollo....7
5.1 Estructura del proyecto....7
5.2 Implementación de funcionalidades....8
5.2.1.Control del personaje....8
5.2.2.Sistema de colisiones y niveles....8
5.2.3.Sistema de enemigos e inteligencia artificial....9
5.2.4.Sistema de autenticación y base de datos....9
5.2.5.Página web de rankings y descarga....9
5.3 Pruebas....10
6. Conclusiones....11
6.1 Conclusiones generales....11
6.2 Consecución de objetivos....12
6.3 Valoración de la metodología y planificación....13
6.4 Visión de futuro....13
7. Glosario....14
8. Bibliografía....15
9. Anexos....16
....18

1. Presentación del proyecto

1.1 Introducción

Uno de los campos más destacados y populares de la informática es el desarrollo de videojuegos y existen diversos géneros en los cual se pueden clasificar cada uno. Por tanto, el proceso de desarrollo de cada uno es diferente. En particular, el género roguelike es un subgénero de videojuegos de rol caracterizado por la generación procedural de niveles (aleatorios) y elementos de mazmorras y fantasía.

En este proyecto desarrollamos nuestro propio videojuego roguelike llamado Azeris, donde los jugadores deben de adentrarse en un castillo abandonado convertido en mazmorra, derrotando varios enemigos a través de numerosos niveles para derrotar a un jefe final. El proyecto se desarrollará en el lenguaje Java dentro del IDE IntelliJ con el framework LibGDX, centrándose en la generación procedural de niveles y competición entre jugadores.

1.2 Contexto

En los últimos años, la industria de los videojuegos ha crecido rápidamente gracias a varias herramientas (Unity, Unreal Engine y Godot) que permiten a cualquier persona desarrollar de manera totalmente gratuita e independiente (desarrolladores que trabajan por su cuenta sin pertenecer a una empresa se llaman desarrolladores indie).

Azeris nace de nuestro propio interés por aprender sobre el desarrollo de videojuegos y profundizar en áreas como la programación, las bases de datos y el diseño gráfico 2D. El proyecto combina diferentes tecnologías actuales: LibGDX, Firebase y Vercel. Para crear un producto completo que une entretenimiento, competición y aprendizaje técnico.

1.3 Justificación

Dentro del ámbito de la informática, el desarrollo de videojuegos es un tema que toca y combina bastantes elementos de programación, diseño interactivo y creación audiovisual.

En el contexto de este proyecto, siempre hemos tenido el deseo de desarrollar nuestro propio juego, y consideramos que esta iniciativa representa una excelente oportunidad para aplicar y ampliar los conocimientos adquiridos a lo largo de nuestra formación académica.

1.4 Objetivos

El **objetivo general** es desarrollar un videojuego con generación procedural, gestión de estadísticas y configuraciones del jugador mediante BBDD y una arquitectura modular que garantice rendimiento, mantenimiento y escalabilidad.

También tenemos **objetivos específicos**, que son los siguientes:

- Implementar un sistema de generación procedural de niveles que cree escenarios aleatorios de forma automática.
- Diseñar e integrar una base de datos para almacenar el progreso, puntuaciones y estadísticas.
- Desarrollar una arquitectura modular utilizando patrones de diseño que faciliten el mantenimiento y ampliación del código.
- Crear una página web informativa y funcional para descargar el juego y consultar las clasificaciones.
- Implementar un sistema de login seguro para autenticar a usuarios y vincular su progreso a la base de datos.

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

Para comenzar hemos creado un proyecto nuevo de LibGDX con IntelliJ, ya que al no tener experiencia previa desarrollando un videojuego, no poseemos ningún proyecto personal que podamos reutilizar o adaptar.

En el equipo somos dos integrantes, entonces hemos acordado al principio comenzar el desarrollo por partes distintas del videojuego. Una persona se encarga de escribir la lógica que permite generar las habitaciones de cada nivel mientras que la otra persona se encarga de diseñar el personaje del jugador y programar la lógica de las colisiones de los recursos del mapa. Una vez terminado el trabajo que le toca a cada integrante del equipo, se incorporarán los dos proyectos en uno a través de GitHub con el repositorio del proyecto y los commits adecuados. Finalmente, se comenzará a desarrollar la base de datos y el diseño de la página web para el proyecto.

Aunque desarrollar un videojuego no se considera tarea fácil, confiamos en nuestras capacidades para crear por lo menos una demo que ponga en práctica los conceptos que queremos trabajar, que en este caso son una aplicación ejecutable en ordenador (videojuego Azeris), una base de datos (que recoja estadísticas de cada partida), y una página web que refleje la información de la base de datos y a la vez sirva como punto de descarga para Azeris.

2.2 Metodología de trabajo

El proyecto se irá desarrollando por fases en el siguiente orden: Azeris, base de datos y finalmente la página web.

Es importante primero tener una versión básica y funcional de Azeris antes de comenzar el desarrollo de la base de datos o la página web, ya que de Azeris obtendremos datos con los cuales podremos trabajar. En cualquier caso, consideramos que Azeris es la parte más importante de este proyecto, razón por la cual la

priorizaremos en todo momento por encima de la BD y la página web, que son prioridades secundarias. En la fase del desarrollo de Azeris, se ha comenzado primero por la generación aleatoria de niveles y el diseño visual de los enemigos y el jefe final, seguido por implementar hasta 30 diseños diferentes para las habitaciones que se encuentran en cada nivel, y finalmente integrando las diferentes mecánicas del juego que veamos necesarias

Para hacer un seguimiento completo del desarrollo de Azeris utilizamos Taiga.io, una herramienta que proporciona un tablero (Kanban) dividido en columnas, cada una representando un estado de tareas de programación. Para ir guardando y compartir el código utilizamos Github, manteniendo diferentes ramas para partes distintas de Azeris.

2.3 Planificación

El proyecto entero consiste de las siguientes fases:

Fase 1: Lluvia de ideas y conceptualización

En esta primera fase previa es donde el grupo se pone de acuerdo en el género del videojuego que se desarrollará, y acordamos en que sería un juego de explorar mazmorras mientras te enfrentas a enemigos y a un jefe final. A partir de esta idea general profundizamos en las funcionalidades que debe tener y planteamos la estructura de la base de datos mediante un diagrama entidad-relación (ER).

Fase 2: Despliegue de Entorno de Desarrollo

Se configura la IDE que se utilizará para el desarrollo, que en este caso es IntelliJ IDEA junto con el framework de LibGDX especializado para programar en Java. También iniciamos un repositorio Git.

Fase 3: Desarrollo de funcionalidades básicas

Implementamos mecánicas básicas del juego (movimiento, ataques, animaciones, colisiones, enemigos, etc). Diseñamos diferentes habitaciones que aparecerán en el nivel de manera aleatoria diseñadas desde un programa gratuito llamado Tiled. La generación aleatoria de niveles y la implementación y posicionamiento de los enemigos y útiles del juego que ayudarán al jugador a poder completar el juego.

Fase 4: Desarrollo de base de datos

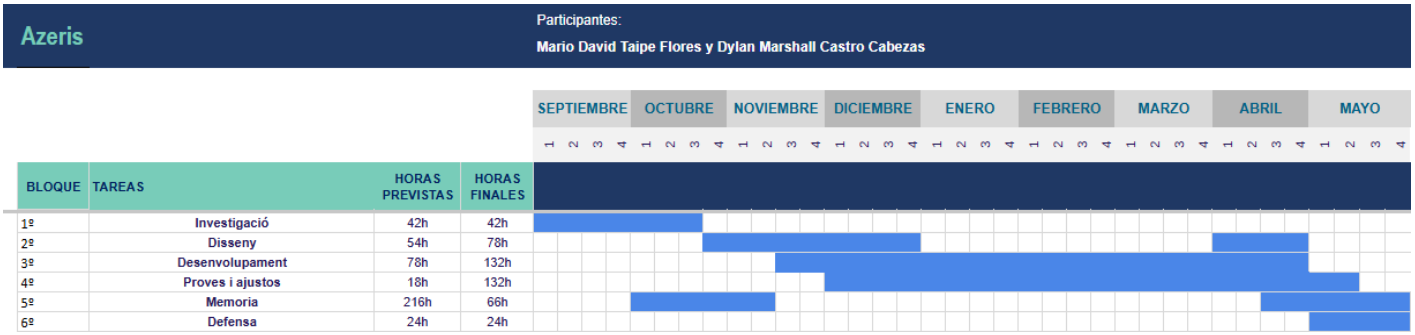
Al terminar con las funcionalidades básicas del juego, dará comienzo con la implementación de base de datos que será sustentado por la plataforma de desarrollo de aplicaciones Firebase debido a estructura que es acordé a lo que queremos que se registre.

Fase 5: Desarrollo de pagina web

Finalizando con el agregado de la base de datos será crear una página web en que aloje tanto los links suficientes para la descarga del juego completo y otra pantalla distinta para alojar y visualizar las mejores puntuaciones de los jugadores.

Fase 6: Prueba final

Tenemos la siguiente distribución de tiempo a lo largo del curso escolar:



BLOQUE	TAREAS	HORAS PREVISTAS	HORAS FINALES
1º	Investigació	42h	42h
2º	Disseny	54h	78h
3º	Desenvolupament	78h	132h
4º	Proves i ajustos	18h	132h
5º	Memoria	216h	66h
6º	Defensa	24h	24h

Este esquema Gantt se ha hecho al principio del curso, entonces es solo una aproximación de las horas que se deberían invertir en cada fase.

3. Análisis

3.1 Casos de uso

El usuario al entrar por primera vez al juego, puede crear una cuenta introduciendo un nombre de usuario, correo electrónico y contraseña. En caso que no quiera, la opción de visualizar estadísticas queda deshabilitada. En caso de olvidar la contraseña, el usuario podrá solicitar un correo de recuperación mediante Firebase Authentication introduciendo el correo con el que se registró.

El jugador puede iniciar una partida y visualizar cada componente que conlleva el juego. Y al finalizar una partida, el sistema almacenará estadísticas relevantes del jugador como:

- Número de enemigos eliminados.
- Muertes.
- Puntuación total.
- Tiempo completado.
- Fecha de la partida.

El jugador puede acceder a la página web y consultar diferentes tablas de clasificación de: mejores puntuaciones, mayor cantidad de enemigos eliminados y menores tiempos a través de un botón implementado en la pantalla principal del juego para visitar la página web. Y desde la web oficial, el usuario podrá descargar Azeris para Windows y Linux mediante enlaces de descarga.

3.2 Requisitos funcionales

El juego debe permitir el movimiento del personaje dentro del escenario y detectar correctamente las colisiones entre entidades del juego. También el generar niveles de forma aleatoria permitiendo que cada partida sea distinta de habitaciones y enemigos.

El videojuego debe incluir mecánicas de combate, enemigos con comportamiento propio y un jefe final como objetivo principal de la partida.

El sistema debe implementar registros de usuarios mediante correo electrónico y contraseña. Los jugadores deben poder registrarse, iniciar sesión, cerrar sesión, recuperar sus contraseñas mediante correo electrónico y borrar sus cuentas. Además, el sistema debe almacenar localmente la sesión iniciada para evitar que el usuario tenga que autenticarse constantemente.

Para la base de datos, el sistema debe almacenar información de los usuarios y registrar estadísticas de cada partida. También debe actualizar automáticamente los datos del jugador y permitir consultar rankings globales desde la página web.

Y la página web debe mostrar información general del videojuego y ofrecer navegación entre distintas tablas de clasificación. También debe incluir enlaces de descarga del juego y adaptarse correctamente a todo tipo de ordenadores.

3.3 Requisitos no funcionales

El videojuego debe ejecutarse de forma fluida y mantener un rendimiento estable durante las partidas. La generación de niveles debe realizarse rápidamente para evitar tiempos de carga excesivos. Del mismo modo, la página web debe cargar de forma rápida y eficiente.

En el apartado de seguridad, las contraseñas de los usuarios no deben almacenarse en texto plano y la autenticación será gestionada mediante Firebase Authentication.

La interfaz del videojuego y de la página web debe ser intuitiva y sencilla de utilizar, permitiendo que cualquier usuario pueda navegar fácilmente por el sistema.

El sistema debe estar diseñado de forma modular para facilitar futuras ampliaciones y mantenimiento. La arquitectura permitirá añadir nuevas mecánicas, enemigos o funcionalidades sin necesidad de rehacer completamente el proyecto.

3.4 Análisis de alternativas tecnológicas

Durante el desarrollo del proyecto, pensamos en diferentes tecnologías antes de seleccionar la que fue escogida.

Se consideraron crear el juego con Unity y Godot pero no fueron seleccionados porque el objetivo principal era trabajar directamente con programación Java y profundizar en conceptos internos del desarrollo de videojuegos. Además de dar visibilidad a más programas de creación de videojuegos y tratar que más personas conozcan más opciones para sus futuros proyectos. Al final se eligió LibGDX debido a que permite desarrollar videojuegos utilizando Java que ofrece un buen rendimiento para juegos 2D y proporciona mayor control sobre la lógica y arquitectura del proyecto. Además, por la previa experiencia en el uso de java.

En relación con la base de datos, inicialmente se consideró utilizar MySQL debido a su popularidad y aprendizaje previo. Pero esta opción requería configurar servidores propios y desarrollar una API adicional para conectar el videojuego con la BBDD. También se pensó en SQLite por su simplicidad, pero fue descartado porque no resultaba adecuado para rankings online y sincronización de datos entre usuarios. Finalmente usamos Firebase ya que proporciona autenticación integrada, almacenamiento en la nube y facilidad de conexión tanto con aplicaciones Java como con aplicaciones web.

Para el desarrollo de la página web, pensamos usar HTML, CSS y JavaScript puro, aunque estas opciones se descartaron porque dificulta la organización de interfaces complejas y la reutilización de componentes. La página web terminó desarrollándose con Vue e Ionic debido principalmente a nuestra experiencia previa del lenguaje. Esta combinación permite crear interfaces modernas, organizadas y adaptables a dispositivos móviles y ordenadores.

Finalmente, para el despliegue de la página web siempre se consideró usar Vercel porque permite integrar fácilmente proyectos Vue con GitHub, realizar despliegues automáticos, ofrecer buen rendimiento de carga y previa experiencia con la web. No pensamos en otro sitio web para el alojamiento de nuestra web.

4. Diseño

4.1 Arquitectura del sistema

El proyecto tiene tres partes: El videojuego (Azeris), la base de datos y la página web oficial.

Azeris ha sido desarrollado exclusivamente en IntelliJ junto con el framework de LibGDX, todo pensado para trabajar con Java. Azeris es la parte central del proyecto, se ejecuta de manera local en el sistema operativo mediante un ejecutable que se puede descargar fácilmente. Después de finalizar una partida de Azeris, los datos que se recogen se envían a una base de datos alojada en Firebase.

La base de datos en Firebase actúa como intermediario entre Azeris y la página web oficial. Recibe datos cada vez que se termina una partida en Azeris, los almacena y luego los utiliza para actualizar la información visible en la página oficial.

La página web oficial de Azeris fue creada con Vue y subida a un repositorio GitHub antes de ser desplegada con el servicio gratuito de Vercel. En esta página se puede visualizar la información disponible (estadísticas de los jugadores, puntos, muertes, victorias, etc) que es brindada por la base de datos. Se puede acceder desde cualquier navegador sin necesidad de registrarse ni descargar Azeris.

4.2 Modelo de datos

La base de datos externa gestionada por Firebase recibe las estadísticas generadas durante un partida cuando se derrota al jefe final. El modelo de datos es bastante simple en su diseño. Solo existe una tabla Jugador con una ID única generada automáticamente por Firebase como clave primaria. El resto de campos son los siguientes: Nombre del usuario, email, enemigos derrotados, muertes, partidas completadas, puntos, fecha de registro, fecha de la partida completada más reciente, y el mejor tiempo para completar una partida.

De los datos que se han listado solo se envían los enemigos derrotados, los puntos y el mejor tiempo para actualizar los rankings de la página web oficial.

4.3 Diseño de interfaz

Nuestro proyecto tiene dos interfaces distintas con las que los usuarios pueden interactuar:

Interfaz del videojuego

Durante una partida normal, la mayor parte de la pantalla muestra una habitación de la mazmorra con sus distintos enemigos. En la parte izquierda superior se encuentra la cantidad de vida del jugador, la cantidad de enemigos derrotados durante la partida y el tiempo de espera para usar la habilidad principal del jugador.

La diferentes pantallas que se pueden ver en Azeris son las siguientes:

Pantalla de inicio: La pantalla que se muestra al abrir el archivo ejecutable. En la parte izquierda, tenemos varios botones. De arriba hacia abajo, empezar partida, puntuación, configuración y finalmente los créditos.

En la parte derecha superior tenemos la opción de dirigirnos a la pantalla de usuario.

Pantalla de usuario: En esta pantalla se pueden ver todas las estadísticas de un usuario. Si el jugador no tiene cuenta, solo se mostrará una recomendación de que se cree una cuenta. Con una cuenta, esta pantalla muestra los siguientes campos: Usuario, email, kills, muertes, victorias, puntos, fecha registro, fecha de la victoria más reciente, mejor tiempo.

Pantalla de juego: Esta pantalla es la que se muestra cuando se inicia una partida. La cámara siempre estará centrada en el jugador.

Pantalla de pausa: Si el jugador en cualquier momento pulsa la tecla del espacio, el juego se pondrá en pausa.

Pantalla de fin de partida: Cuando se derrota al jefe final, se mostrarán una serie de imágenes explicando los eventos que suceden después de la victoria del personaje antes de volver a la pantalla de inicio.

Interfaz de la aplicación web

La página web tiene una estructura simple con dos pantallas principales:

Pantalla de información y descarga: Esta pantalla es la primera que se ve cuando se accede a la página web oficial. Se explica y promociona Azeris y se puede descargar el ejecutable.

Pantalla de estadísticas: En este apartado se pueden ver diversas páginas en un tab bar con diferentes estadísticas de cada jugador en una ranking/leaderboard ordenada por enemigos derrotados, mayor cantidad de puntos y el menor tiempo que se ha tardado en completar el juego.

Todas las pantallas descritas en este apartado se pueden encontrar en el **Anexo A**.

5. Desarrollo

5.1 Estructura del proyecto

La estructura general del proyecto se divide principalmente en dos partes: la carpeta assets, donde se almacenan todos los recursos utilizados por el juego, y el módulo core donde se encuentra toda la lógica programada en Java mediante LibGDX.

```
assets/
├── audio/
├── enemy/
├── maps/
├── pantalla/
├── player/
└── utils/

core/
└── src/main/java/
    ├── firebase/
    └── ui/projecto/
        ├── mecánicas/
        ├── personajes/
        └── Screens/
```

assets/ contiene todos los recursos necesarios para el funcionamiento visual y sonoro. Dentro de **audio/** se organizan los efectos de sonido y música separados entre sonidos del jugador, enemigos, menús y utils.

La carpeta **enemy/** almacena todos los sprites y animaciones de los enemigos del juego. En **maps/** se encuentran todos los mapas creados en Tiled, alojandose la estructura, configuraciones y texturas de los escenarios. **pantalla/** contiene los recursos visuales; menús, fondos y botones.

Por otro lado, el módulo *core* contiene toda la lógica principal del videojuego. El paquete **firebase/** incluye las clases responsables de la autenticación de usuarios, almacenamiento de estadísticas y sesiones utilizando Firebase.

En **ui.projecto/** se organiza el núcleo principal del videojuego. **mecánicas/** contiene sistemas generales: Animaciones, generación procedural y control de objetos interactivos. **personajes/** está dividido entre jugador y enemigos. Cada enemigo dispone de una estructura propia organizada en clases. Esta arquitectura nos permite reutilizar la lógica y simplifica la creación de nuevos enemigos.

Finalmente, el paquete **Screens/** aloja todas las pantallas del videojuego; menú principal, login, registro, pantalla de juego, recuperación de contraseña y más.

5.2 Implementación de funcionalidades

5.2.1.Control del personaje

El movimiento del jugador se implementa mediante una clase llamada Player.java encargada de detectar las entradas del teclado y actualizar la posición del personaje en cada frame. El sistema permite desplazamiento en ocho direcciones, sprint, ataques y detección de daño. Las animaciones del personaje se controlan mediante un sistema de estados que cambia automáticamente dependiendo de la acción realizada por el jugador.

La detección de colisiones se realiza utilizando rectángulos de colisión asociados al personaje y objetos del mapa. El jugador dispone de sonidos y animaciones independientes gestionados mediante clases específicas para facilitar la organización del código.

5.2.2.Sistema de colisiones y niveles

Los niveles del juego se diseñaron utilizando el programa Tiled mediante mapas con formato .tmx. Cada habitación del juego se encuentra almacenada dentro de la carpeta **maps/mapas** y posteriormente es seleccionada aleatoriamente durante la partida.

La generación procedural se implementó mediante la clase DungeonManager.java, encargada de seleccionar habitaciones aleatorias y conectarlas entre sí para formar mazmorras distintas en cada ejecución. En los mapas ya controla el posicionamiento de spawn de los enemigos, puertas y objetos útiles.

Se decidió utilizar generación procedural porque aumenta la rejugabilidad y encaja con el género roguelike planteado inicialmente.

5.2.3.Sistema de enemigos e inteligencia artificial

Los enemigos fueron implementados utilizando distintas clases propias para comportamiento, animaciones, sonidos y estados. El sistema de inteligencia artificial se basa en máquinas de estados que controlan acciones como patrullar, perseguir al jugador, atacar, recibir daño o morir.

La detección del jugador se realiza mediante sistemas de visión y distancia implementados dentro de las clases EnemyVision.java y EnemyPathFinder.java. Cada enemigo tiene estadísticas diferentes permitiendo crear enfrentamientos variados durante la partida.

5.2.4.Sistema de autenticación y base de datos

El sistema de usuarios se implementó utilizando Firebase Authentication y Firestore; Firebase Auth para el registro y Firestore para guardar los datos del usuario. Los jugadores pueden registrarse, iniciar sesión y recuperar su contraseña mediante diferentes pantallas desarrolladas dentro del paquete Screens.

La comunicación con Firebase se realiza mediante clases específicas como FirebaseAuthService.java y FirebaseFirestoreService.java. Las estadísticas del jugador se almacenan automáticamente en Firestore al finalizar la partida o cuando el jugador muere.

5.2.5.Página web de rankings y descarga

La página web permite consultar rankings y sirve también como plataforma informativa y de descarga del videojuego.

Los rankings muestran las mejores puntuaciones, enemigos eliminados y mejores tiempos. Esta información se obtiene desde Firestore y se ordena automáticamente antes de mostrar por pantalla.

5.3 Pruebas

Durante el desarrollo del proyecto se realizaron diferentes tipos de pruebas para comprobar el correcto funcionamiento de las funcionalidades.

Las primeras pruebas realizadas fueron la comprobación de las funciones básicas del jugador. Verificando el movimiento, detección de colisiones, ataques, daño recibido y animaciones.

Posteriormente se realizaron pruebas sobre el sistema de generación procedural y que el jugador se encuentre con distintas habitaciones cada vez.

Con los enemigos, se probó individualmente todos los estados de comportamiento para verificar que reaccionaban ante la presencia del jugador.

Con Firebase, se comprobó el registro de usuarios, inicio de sesión, recuperación de contraseña y almacenamiento de estadísticas en la base de datos. Además, se verificó que la información almacenada se mostrará correctamente en la página web de rankings.

En la web se comprobó la estructura en diferentes tamaños de pantalla y navegadores. Durante estas pruebas, se detectaron problemas relacionados con las tablas y separación de elementos que fueron corregidos mediante ajustes CSS.

6. Conclusiones

6.1 Conclusiones generales

El desarrollo de Azeris nos ha permitido crear un videojuego funcional del género roguelike que integra generación de niveles, combate contra enemigos en tiempo real, almacenamiento de estadísticas mediante base de datos y una página web conectada al sistema de puntuación. El resultado obtenido cumple con la idea inicial planteada al comienzo del proyecto y demuestra que es posible combinar distintas tecnologías para construir una aplicación completa relacionada con el desarrollo de videojuegos.

A nivel técnico, el proyecto nos ha dado una oportunidad para aplicar nuestros conocimientos adquiridos durante la formación académica y ampliarlos con las nuevas mecánicas que hubo que implementar para que este proyecto sea un éxito. Así mismo, trabajar con LibGDX y Firebase ha permitido comprender mejor cómo se estructura un videojuego real y cómo se conectan diferentes sistemas externos a una aplicación interactiva.

Uno de los aspectos más positivos del proyecto fue la implementación de la generación procedural de niveles y la arquitectura modular para separar enemigos, pantallas, animaciones, audio y mecánicas del juego. Esta organización ha facilitado el mantenimiento del código y ha permitido trabajar de manera más ordenada durante el desarrollo.

Aunque, también han surgido dificultades durante el desarrollo como las colisiones, animaciones y la sincronización, requirió más tiempo de lo esperado debido a la complejidad técnica y a la falta de experiencia en el desarrollo de videojuegos. Del mismo modo, algunas funcionalidades tuvieron que simplificarse para garantizar la estabilidad general del proyecto y poder finalizar una versión funcional dentro del tiempo establecido.

En definitiva, el proyecto ha supuesto una experiencia muy positiva tanto a nivel académico como personal ya que nos permitió desarrollar competencias técnicas y organizativas aplicadas a un entorno real de desarrollo de software.

6.2 Consecución de objetivos

El objetivo general del proyecto, desarrollar un videojuego roguelike con generación procedural, almacenamiento de estadísticas y arquitectura modular, se ha cumplido satisfactoriamente. Azeris cuenta con niveles aleatorios, diferentes enemigos, sistema de puntuaciones, autenticación de usuarios y conexión con Firebase para almacenar información de las partidas.

El objetivo de implementar un sistema de generación procedural de niveles también se ha alcanzado. El juego selecciona automáticamente diferentes habitaciones diseñadas previamente y genera recorridos distintos en cada partida, aumentando la rejugabilidad del videojuego.

La integración de una base de datos para almacenar estadísticas y progreso del jugador se ha completado mediante Firebase Firestore. El sistema permite guardar puntuaciones, enemigos eliminados y tiempos de partida, además de mostrar esta información posteriormente en la página web.

El objetivo relacionado con la arquitectura modular se ha cumplido parcialmente de forma muy satisfactoria. El proyecto está dividido en paquetes independientes para enemigos, jugador, pantallas, mecánicas y servicios de Firebase, lo que facilita la ampliación futura del juego. No obstante, todavía existen partes del código que podrían ser factorizadas para mejorar aún más la reutilización de componentes.

La creación de una página web funcional también se ha conseguido. La web permite descargar el juego, consultar rankings y mostrar información general sobre Azeris utilizando Vue, Ionic y Firebase como tecnologías principales.

Por último, el sistema de login seguro se ha implementado mediante Firebase Authentication, permitiendo registrar usuarios, iniciar sesión y recuperar contraseñas.

En general, todos los objetivos principales del proyecto han sido alcanzados, aunque algunas funcionalidades podrían seguir ampliándose en futuras versiones sin afectar al funcionamiento actual del sistema.

6.3 Valoración de la metodología y planificación

Azeris es el primer videojuego que nosotros hemos desarrollado. Teníamos muchas ideas al principio de mecánicas y adiciones al videojuego en sí que al final no se han acabado de implementar en la versión final del juego que atribuimos a la falta de tiempo y de conocimiento suficiente en LibGDX. Debido a nuestro estatus como principiantes en el desarrollo de los videojuegos, se ha invertido bastantes horas en aprender el funcionamiento básico del framework LibGDX además de sus otras funcionalidades más complejas cuando ha hecho falta como parte de las horas que hemos acordado dedicar al desarrollo de Azeris.

Para planificar nuestro progreso en el desarrollo de Azeris nos ha servido utilizar Taiga.io, donde hemos podido utilizar la metodología Kanban para asignar tareas específicas a cada miembro del equipo. Para el resto hemos hecho servir el seguimiento semanal.

6.4 Visión de futuro

Azeris como videojuego tiene mucho potencial y una variedad enorme de posibilidades de mejora. Entre las más interesantes podemos destacar:

- Tres clases distintas para escoger al empezar el juego: Elfo, Duende, y Humano. Cada uno con habilidades propias.
- Diferentes tipos de armadura, armas, pociones, y items para utilizar.
- Tres tipos de daño: Cortante, Perforante, Contundente. Enemigos y clases son más débiles o más resistentes a ciertos tipos de daño. Esto se puede modificar con armaduras, otros ítems o habilidades.

Por lo que concierne el backend y el frontend, aunque el sistema funciona correctamente, todavía podrían añadirse mejoras adicionales relacionadas con validaciones más avanzadas o perfiles personalizados para cada usuario.

7. Glosario

Término	Definicion
API (Application Programming Interface)	Conjunto de reglas y herramientas que permiten que diferentes programas se comuniquen entre sí.
Arquitectura modular	Forma de organizar un programa separándolo en partes independientes llamadas módulos, para facilitar mantenimiento y ampliación.
Assets	Archivos de recursos usados en un videojuego, como imágenes, sonidos, mapas o animaciones.
Diagrama entidad-relación (ER)	Esquema visual usado para diseñar bases de datos mostrando relaciones entre datos.
Framework	Conjunto de herramientas y estructuras que facilitan el desarrollo de software.
Generación procedural	Técnica que crea contenido automáticamente mediante algoritmos, como niveles aleatorios en videojuegos.
IDE (Integrated Development Environment)	Programa que reúne herramientas para programar, escribir código y ejecutar proyectos.
Indie (desarrollador independiente)	Desarrollador que crea videojuegos sin pertenecer a una gran empresa.
Kanban	Método de organización de tareas mediante columnas que representan estados del trabajo.
LibGDX	Framework especializado en desarrollo de videojuegos con Java.
Máquina de estados	Sistema de programación donde un objeto cambia de comportamiento dependiendo de su estado actual.
Mecánicas del juego	Reglas y sistemas que definen cómo funciona un videojuego.
Pixel Art	Estilo gráfico formado por píxeles visibles, común en videojuegos retro.
Procedural	Relacionado con contenido generado automáticamente mediante procesos o algoritmos.

Ranking / Leaderboard	Tabla de clasificación donde aparecen las mejores puntuaciones o estadísticas de los jugadores.
Roguelike	Subgénero de videojuegos caracterizado por niveles aleatorios, dificultad elevada y exploración de mazmorras.

8. Bibliografía

Documentacion Oficial

- LibGDX (2026). *LibGDX Wiki*. LibGDX. https://libgdx.com/wiki/
- Tiled Documentation Writers (2023). *Tiled Documentation.* Tiled. https://doc.mapeditor.org/en/stable/

Herramientas

- IntelliJ Idea: https://www.jetbrains.com/idea/
- LibGDX: https://libgdx.com/
- Krita: https://krita.org/es/
- Tiled: https://www.mapeditor.org/
- Audacity: https://www.audacityteam.org/
- Visual Studio Code: https://code.visualstudio.com/
- Launch4J: https://launch4j.sourceforge.net/

Videos de Referencia

- "LibGDX & Tiled RPG Tutorial" by QuillRaven
- "Java Game Development (Libgdx)" by Phillip Mode Dev

Repositorio del proyecto

https://github.com/Dylan-alt-bot/Azeris/tree/main

9. Anexos

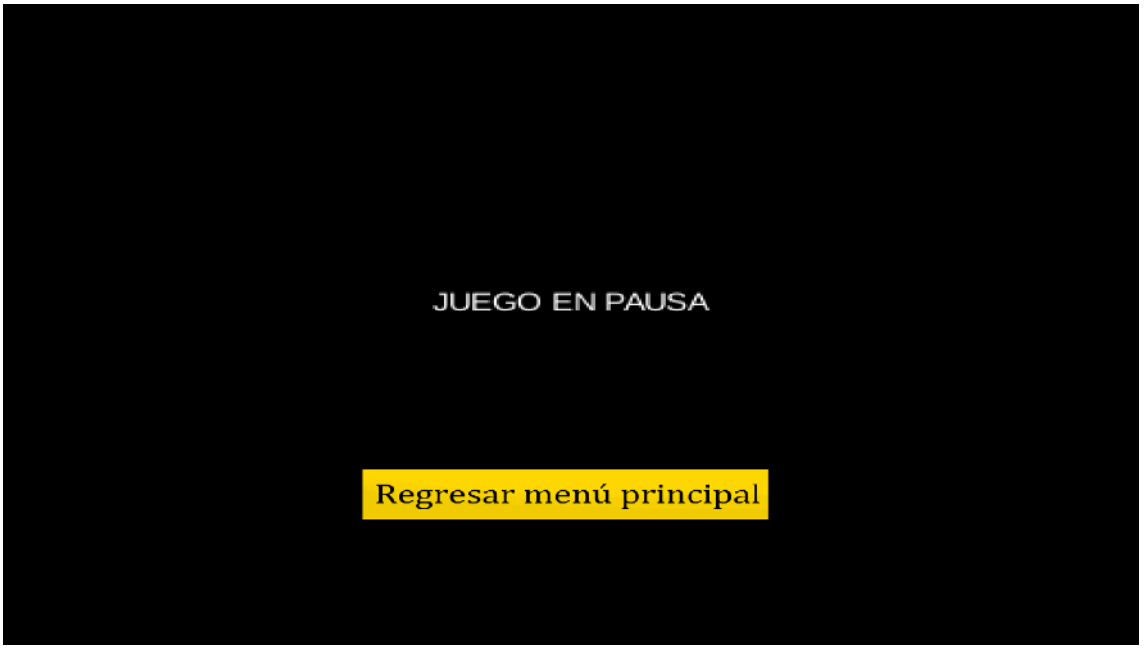
Anexo A - Pantallas Azeris

Pantalla de Inicio

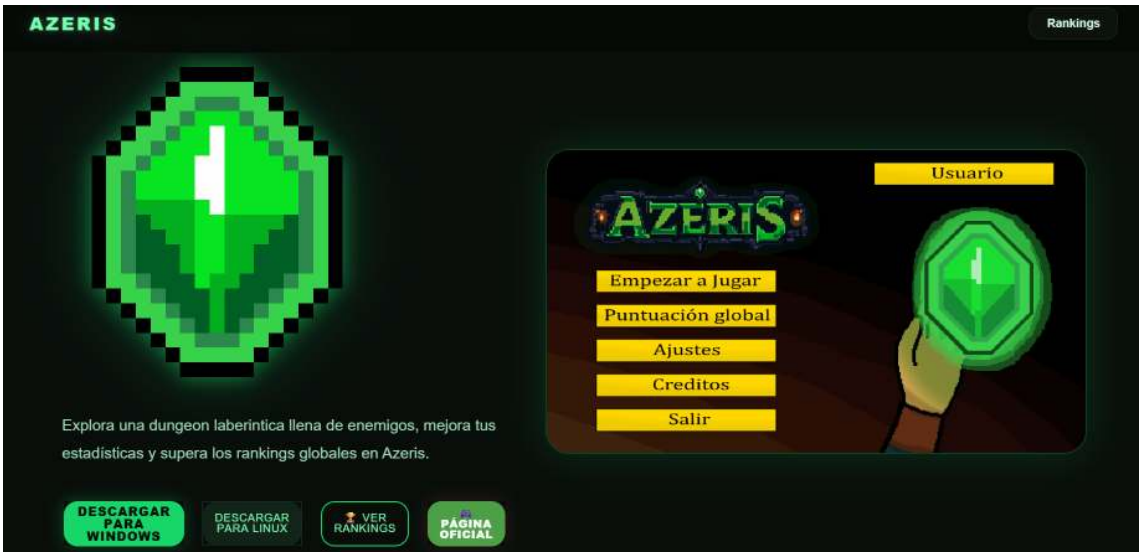
Pantalla de Juego

Pantalla de Usuario

Pantalla de Pausa



Página Web Oficial



Pantalla Web de Mejores Tiempos

Pantalla Web de Más Enemigos Derrotados

Pantalla de Mejor Puntuación

Nota sobre el uso de inteligencia artificial

Durante el desarrollo de este proyecto se han utilizado herramientas de inteligencia artificial como apoyo en determinadas tareas del proceso. Entre ellas, como herramienta de consulta sobre dudas de programación o aspectos desconocidos de LibGDX. En ningún caso el uso de estas herramientas ha sustituido el proceso de

comprensión, análisis y toma de decisiones del autor. Todo el contenido ha sido revisado, verificado y asumido como propio, garantizando que refleja fielmente el trabajo realizado y los conocimientos adquiridos a lo largo del proyecto.

Licencia

Licencia: CC BY-NC-ND 3.0 ES