JUEGO RPG

Documento de Especificación Funcional

Índice

1. INTRODUCCIÓN

Breve descripción del juego:

JuegoRPG es un videojuego de rol (RPG) de combate por turnos desarrollado en Java.
El juego narra la historia de un aventurero que entra en una mazmorra en busca de respuestas, pero se encuentra atrapado en un laberinto sin fin. Para escapar, el jugador debe derrotar a diferentes enemigos, comprar equipamiento en una tienda y progresar hasta enfrentarse al jefe final "JEFE FINAL - Dueñas" al alcanzar el nivel 5.

El juego se ejecuta en consola y utiliza entrada por teclado para la interacción.
El jugador puede realizar las siguientes acciones principales:

- Luchar contra enemigos en combates por turnos
- Comprar ítems en la tienda para mejorar sus estadísticas
- Consultar sus estadísticas actuales
- Salir del juego

El sistema de combate funciona por turnos alternados, donde el jugador ataca primero y luego el enemigo. El daño se calcula restando la defensa + vida del atacado al ataque del atacante. Al derrotar enemigos, el jugador gana oro y sube de nivel, mejorando automáticamente sus estadísticas.

Objetivos del proyecto:

Los objetivos principales que nos marcamos al inicio del proyecto fueron:

1. Crear un juego RPG en Java que demuestre los conocimientos que hemos ganado en lo que llevamos de curso
2. Implementar un sistema de combate por turnos con mecánicas claras y balanceadas
3. Crear un sistema de economía (oro) que permita al jugador mejorar su personaje mediante compras
4. Implementar diferentes tipos de enemigos con estadísticas escaladas según el nivel del jugador
5. Añadir un jefe final especial que sirva como objetivo final del juego
6. Crear una interfaz de usuario clara y fácil de usar en consola
7. Documentar bien el código con comentarios JavaDoc

2. ARQUITECTURA DEL SOFTWARE

Diagrama de clases

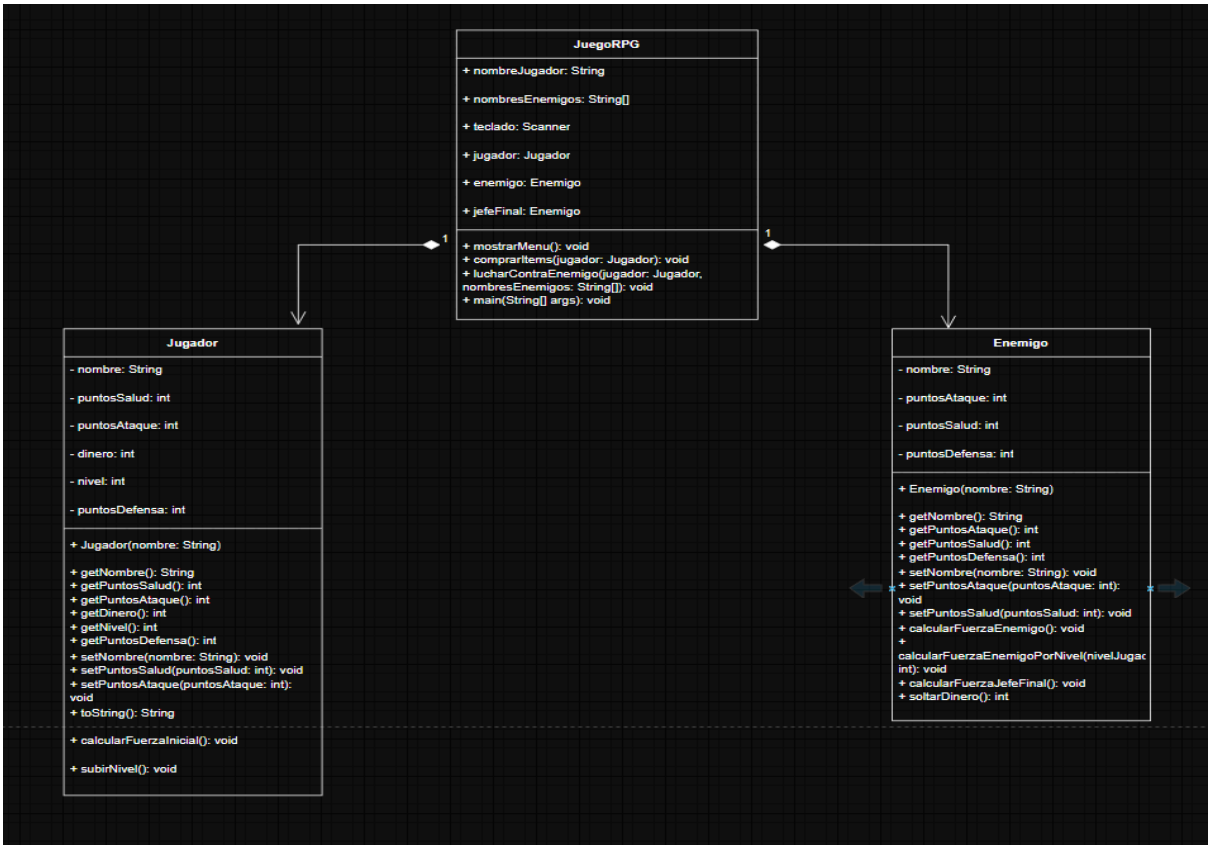
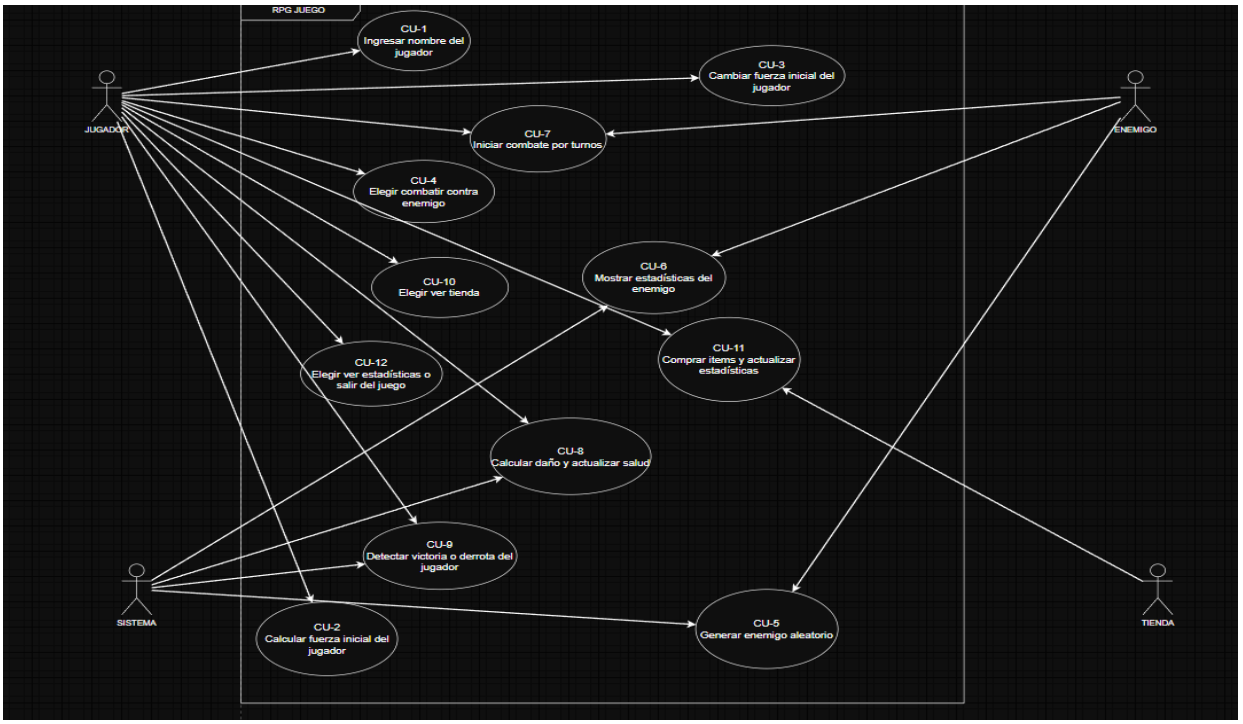


Diagrama de Casos de uso



Requisitos Funcionales

- RF-1: Interfaz / Menú Principal

RF-1.1 Mostrar menú principal
RF-1.2 Seleccionar opción de luchar contra enemigo
RF-1.3 Seleccionar opción de comprar ítems (tienda)
RF-1.4 Consultar estadísticas del jugador
RF-1.5 Salir del juego

- RF-2: Jugador

RF-2.1 Introducir nombre del jugador
RF-2.2 Calcular fuerza inicial aleatoria
RF-2.3 Permitir mejorar fuerza inicial pagando oro
RF-2.4 Modificar estadísticas (salud, ataque, defensa, oro, nivel)
RF-2.5 Subir de nivel al ganar combates
RF-2.6 Mostrar estadísticas del jugador

- RF-3: Enemigos

RF-3.1 Generar enemigos según el nivel del jugador
RF-3.2 Mostrar estadísticas del enemigo
RF-3.3 Calcular fuerza o atributos aleatorios por tipo de enemigo
RF-3.4 Soltar oro al morir
RF-3.5 Generar jefe final cuando el jugador tiene nivel ≥5

- RF-4: Combate

RF-4.1 Iniciar combate por turnos
RF-4.2 Calcular daño entre jugador y enemigo
RF-4.3 Detectar victoria o derrota
RF-4.4 Entregar oro y experiencia al ganar
RF-4.5 Finalizar el juego si se derrota al jefe final

- RF-5: Tienda

RF-5.1 Mostrar ítems disponibles
RF-5.2 Comprar ítems usando oro
RF-5.3 Mejorar estadísticas según el ítem comprado
RF-5.4 Comprobar que el jugador tiene suficiente oro
RF-5.5 Salir de la tienda

3. DISEÑO DEL JUEGO

Mecánicas de juego:

El juego comienza mostrando la historia y pidiendo el nombre del jugador. Después se crea el personaje, se calcula su fuerza inicial y se ofrece la opción de mejorarla pagando 1 oro.

A partir de ahí, el jugador entra en un menú principal en bucle con cuatro opciones:

1. **Luchar:**
   - Se genera un enemigo según el nivel (o el jefe final si el jugador es nivel 5).
   - Se realiza un combate por turnos hasta que uno quede sin salud.
   - Si gana, obtiene oro y sube de nivel (excepto contra el jefe final).

2. **Comprar ítems:**
   - Se muestra la tienda, se valida el oro y se actualizan las estadísticas.

3. **Consultar estadísticas:**
   - Se muestran los atributos actuales del jugador.

4. **Salir del juego:**
   - Finaliza el programa.

El juego termina si el jugador derrota al jefe final o si muere en combate.

Reglas del juego:

Reglas de combate:

- No se puede luchar con salud ≤ 0.
- El daño mínimo es 0.
- Combate obligatorio por turnos.
- No se puede huir.

Reglas de compras:

- No se pueden comprar ítems sin suficiente oro.
- Los ítems mejoran permanentemente las estadísticas.

Reglas de progresión:

- Se sube de nivel solo derrotando enemigos normales.
- Las mejoras por nivel son automáticas.
- El jefe final aparece al nivel 5.

Reglas de economía:

- Se empieza con 2 de oro.
- El oro se gana solo derrotando enemigos.
- Se gasta en ítems o mejora de fuerza inicial.

Reglas de enemigos:

- Se generan aleatoriamente entre 4 nombres.
- Sus estadísticas escalan con el nivel del jugador.
- El jefe final aparece una única vez.

Reglas de salud:

- Salud ≤ 0 = derrota.
- Solo puede aumentarse comprando ítems o subiendo de nivel.

4. IMPLEMENTACIÓN

Lenguaje de programación utilizado:

El juego está desarrollado completamente en **Java**, utilizando programación orientada a Objetos.

Bibliotecas utilizadas:

- **Scanner** para leer entrada del usuario.
- **Math.random()** para generar valores aleatorios.

Herramientas y Tecnologías utilizadas:

- **NetBeans** como IDE.
- **JDK** para compilar y ejecutar el juego.
- **Draw.io** para el diagrama de clases y casos de uso
- Comentarios **JavaDoc** para documentar el código.
- **Github** para subir nuestros progresos y trabajar en equipo
- **Visual Studio** para editar el github

5. PRUEBAS

Resultados de pruebas y correcciones:

PRUEBA 1: Creación del jugador

- Prueba realizada: Verificar que el jugador se crea correctamente con los valores iniciales esperados.
- Resultado: El jugador se creaba correctamente con salud 20, defensa 4, dinero 2 y nivel 1. El ataque se calculaba aleatoriamente como queriamos.
- Correcciones: Ninguna.

PRUEBA 2: Cálculo de fuerza inicial

- Prueba realizada: Verificar que la fuerza inicial se calcula correctamente y que la opción de mejorarla funciona.
- Resultado: La fuerza inicial se calculaba correctamente. Sin embargo, encontramos que si el jugador no tenía oro suficiente, el sistema intentaba mejorar la fuerza de todas formas.
- Corrección: Se añadió una validación para verificar que el jugador tenga al menos 1 oro antes de permitir mejorar la fuerza inicial.

PRUEBA 3: Sistema de combate

- Prueba realizada: Verificar que el combate por turnos funciona correctamente, que el daño se calcula bien y que el combate termina cuando alguien llega a 0 de salud.
- Resultado: El combate funcionaba sin problemas y hacia lo que queriamos.
- Corrección: Ninguna

PRUEBA 4: Generación de enemigos

- Prueba realizada: Verificar que los enemigos se generan correctamente según el nivel del jugador.
- Resultado: Los enemigos se generaban, pero las estadísticas no funcionaban correctamente según el nivel del jugador
- Corrección: Se ajustaron las estadísticas de los enemigos para que sean más dificiles a medida que el jugador sube de nivel, pero no imposibles.

PRUEBA 5: Sistema de compras

- Prueba realizada: Verificar que las compras funcionan correctamente y que el sistema de oro funciona bien en la tienda
- Resultado: La tienda funciona correctamente sin problemas.
- Corrección: Ninguna.

PRUEBA 6: Subida de nivel

- Prueba realizada: Verificar que al derrotar enemigos, el jugador sube de nivel y recibe las mejoras correctas.
- Resultado: La subida de nivel funcionaba correctamente, añadiendo +5 salud, +2 ataque y +1 defensa como se esperaba.
- Correcciones: Ninguna.

PRUEBA 7: Jefe final

- Prueba realizada: Verificar que el jefe final aparece al alcanzar nivel 5 y que tiene las estadísticas correctas.
- Resultado: El jefe final aparecía correctamente, pero inicialmente también permitía subir de nivel al derrotarlo.
- Corrección: Se añadió un booleano "esJefeFinal" para evitar que el jugador suba de nivel al derrotar al jefe final, ya que este es el objetivo final del juego.

PRUEBA 8: Opciones del menú

- Prueba realizada: Verificar que el menú maneja correctamente opciones inválidas.
- Resultado: Si el usuario introduce un número fuera del rango 1-4, el programa no mostraba ningún mensaje.
- Corrección: Se añadió un caso "default" en el switch del menú principal para mostrar un mensaje de "Opción inválida" cuando se introduce un valor que no está en el menu.

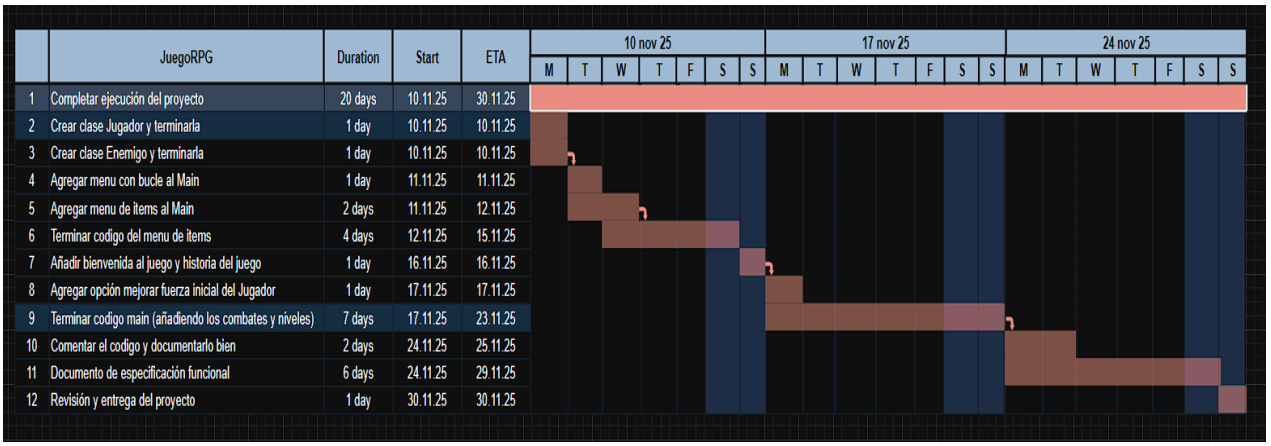
PRUEBA 9: Flujo completo del juego

- Prueba realizada: Jugar una partida completa desde el inicio hasta derrotar al jefe final.
- Resultado: El juego funcionaba correctamente y solo tuvimos que poner más bonito como se mostraba en la consola los mensajes

6. GESTIÓN DE PROYECTO

Planificación temporal y seguimiento:

DIAGRAMA DE GANTT



	JuegoRPG	Duration	Start	ETA
1	Completar ejecución del proyecto	20 days	10.11.25	30.11.25
2	Crear clase Jugador y terminarla	1 day	10.11.25	10.11.25
3	Crear clase Enemigo y terminarla	1 day	10.11.25	10.11.25
4	Agregar menu con bucle al Main	1 day	11.11.25	11.11.25
5	Agregar menu de items al Main	2 days	11.11.25	12.11.25
6	Terminar codigo del menu de items	4 days	12.11.25	15.11.25
7	Añadir bienvenida al juego y historia del juego	1 day	16.11.25	16.11.25
8	Agregar opción mejorar fuerza inicial del Jugador	1 day	17.11.25	17.11.25
9	Terminar codigo main (añadiendo los combates y niveles)	7 days	17.11.25	23.11.25
10	Comentar el codigo y documentarlo bien	2 days	24.11.25	25.11.25
11	Documento de especificación funcional	6 days	24.11.25	29.11.25
12	Revisión y entrega del proyecto	1 day	30.11.25	30.11.25

Problemas encontrados y soluciones:

PROBLEMA 1: Gestión de la lógica del combate

- Descripción: Inicialmente, teníamos los combates metidos dentro directamente del caso 1 del menú, y eso hacia que el código se viera mal y para ajustar cosas era más problemático
- Solución: Separamos la lógica del combate en un método específico (lucharContraEnemigo). Esto hizo el código más sencillo para modificar y fácil de entender.

PROBLEMA 2: Balance del juego

- Descripción: Inicialmente, el juego era demasiado demasiado difícil dependiendo de la suerte con la fuerza inicial.
- Solución: Ajustamos las fórmulas de cálculo de estadísticas de enemigos. También ajustamos los precios de los ítems para que sean más balanceados.

PROBLEMA 3: Falta de tiempo

- Descripción: Tuvimos que compaginar el desarrollo del proyecto con otras asignaturas y responsabilidades.
- Solución: Organizamos el trabajo en tareas pequeñas y manejables para cada uno, priorizando las funcionalidades principales. Trabajamos en el proyecto de forma constante pero sin sobrecargarnos, distribuyendo el trabajo a lo largo de varias semanas y sobretodo aprovechar las horas de clase.

PROBLEMA 4: Documentación del código

- Descripción: Inicialmente, el código tenía pocos comentarios, lo que dificultaba entender la funcionalidad de algunos métodos.
- Solución: Añadimos comentarios JavaDoc a todas las clases y métodos, explicando qué hace cada parte del código. Esto mejoró mucho la visibilidad del proyecto para poder entenderlo.

7. CONCLUSIONES

Lecciones aprendidas:

Durante el desarrollo de este proyecto, aprendimos varias lecciones importantes:

1. PLANIFICACIÓN ES ESENCIAL: Tener un plan claro desde el inicio nos ayudó a mantener el proyecto organizado y evitar perder tiempo.

2. PROGRAMACIÓN: Este proyecto nos ayudó a entender mejor los conceptos básicos de la programación y a imaginar ideas nuevas dentro del proyecto.

3. PRUEBAS: Desarrollar el proyecto de forma conjunta, probando cada funcionalidad conforme la implementamos, nos permitió encontrar y corregir errores rapidamente en el proceso.

4. DOCUMENTACIÓN: Aprendimos que documentar el código no es solo para otros, sino también para nosotros mismos. Los comentarios JavaDoc nos ayudaron a recordar qué hace cada parte del código cuando volvíamos a trabajar en él después de unos días.

5. BALANCE DE JUEGO: Aprendimos que crear un juego balanceado requiere muchas pruebas y ajustes. Lo que parece fácil o difícil en el papel puede ser muy diferente en la práctica.

6. GESTIÓN DEL TIEMPO: Aprendimos a distribuir el trabajo a lo largo del tiempo y a priorizar las funcionalidades más importantes cuando el tiempo es limitado.

Éxito del proyecto:

Los objetivos que hemos cumplido exitosamente son:

1. Crear un juego RPG funcional en Java: El juego funciona correctamente desde que lo iniciamos hasta que termina.

2. Implementar sistema de combate por turnos y niveles: Al principio íbamos a hacer el mínimo que se nos pedía en la práctica pero quisimos hacerlo mejor y añadimos el sistema de combate por turnos el cual funciona correctamente y el sistema de niveles para añadir al jefe final en el nivel 5, costo un poco pero ahora sabemos hacerlo que es lo importante.

3. Crear sistema de economía: El sistema de oro funciona correctamente, permitiendo ganar oro derrotando enemigos y gastarlo en ítems o mejoras.

4. Estructurar proyecto con múltiples clases: El proyecto está bien organizado en tres clases principales con responsabilidades claras.

5. Crear interfaz de usuario clara: Los menús son claros y fáciles de usar.

6. Documentar el código: Todo el código está documentado con comentarios JavaDoc.

Mejoras futuras:

Aunque el proyecto cumple con lo que se nos pide y incluso añadimos varias cosas extras que no eran obligatorias, hay varias mejoras que se podrían implementar en el futuro como por ejemplo:

1. SISTEMA DE MÚSICA Y SONIDOS: Implementar un sistema de música para que el juego capte más atención del jugador y sea más entretenido.

2. MÁS TIPOS DE ENEMIGOS: Añadir más variedad de enemigos con diferentes habilidades o comportamientos especiales.

3. SISTEMA DE HABILIDADES: Implementar un sistema de habilidades especiales que el jugador pueda usar en combate, como ataques potentes o curaciones.

4. MÁS CONTENIDO: Añadir más niveles, más ítems, misiones secundarias o una historia más elaborada.

Estas mejoras harían el juego más completo y atractivo, pero el proyecto actual cumple con todo lo necesario perfectamente.