INTEGRANTES:

Carlos Jhosef Perez Maldonado – 192364

Valerin Fernanda Hernandez Prada - 192425

|  |  |
| --- | --- |
| **adventurecraft** | Descripción breve  un videojuego interactivo donde el jugador puede moverse, enfrentarse a enemigos, superar obstáculos, y avanzar por diferentes niveles, todo mientras disfruta de un entorno visual y sonoro atractivo |

Tabla de contenido

1. Roles de los integrantes -------------------- pág. 3
2. Introducción ----------------------------------- pág. 3
3. Objetivos --------------------------------------- pág. 3
4. Alcance del proyecto ----------------------- pág. 4
5. Documentación técnica ------------------- pág. 4, 5,6
6. SDLC--------------------------------------------- pág. 7, 8
7. Diagramas UML------------------------------- pág. 9, 10,11,12,13

Proyecto - videojuego adventurecraft

1. **Roles de los integrantes :**

**Desarrollador de Lógica del Juego (**Carlos Jhosef Perez Maldonado**):** programar la lógica principal del juego, asegurar que las entidades (como el jugador, los enemigos y los objetos) interactuaran correctamente, y manejar el flujo entre los diferentes estados del juego.

**Desarrollador de Interfaz y Visuales (**Valerin Fernanda Hernandez Prada**):**

Diseñar los elementos visuales del juego y programar la interfaz gráfica para que sea atractiva y fácil de usar.

1. **Introducción :**

El proyecto consiste en el desarrollo de un videojuego 2D llamado AdventureCraft, que combina elementos de aventura, acción y resolución de desafíos. Su creación busca resolver el reto de construir un sistema interactivo en el que un jugador pueda controlar un personaje, enfrentarse a enemigos, superar obstáculos y avanzar a través de niveles progresivos, todo mientras disfruta de un entorno visual y sonoro atractivo.

El videojuego está diseñado como una herramienta para aprender y aplicar conceptos de programación orientada a objetos, diseño modular y desarrollo de videojuegos. A través de este proyecto, se simulan situaciones reales de creación de software, como la gestión de recursos visuales y de audio, el manejo de animaciones y la interacción entre múltiples componentes.

1. **Objetivos:**

**Objetivo general:**

Crear un videojuego 2D que combine elementos de aventura y acción, con niveles progresivos y dinámicas desafiantes.

**Objetivos específicos:**

-Implementar un sistema de control para el jugador que incluya movimiento, ataques y gestión de recursos (vida y energía).

-Diseñar enemigos con comportamientos únicos y niveles con obstáculos que aumenten la dificultad.

-Integrar efectos visuales y sonoros que mejoren la experiencia del usuario.

-Desarrollar una interfaz amigable con menús interactivos, opciones configurables y transiciones fluidas entre estados del juego.

1. **Alcance del proyecto :**

Incluye:

Un sistema funcional con múltiples niveles, enemigos y un jugador controlable.

Menús interactivos (inicio, opciones, créditos).

Efectos visuales (animaciones, lluvia, diálogos emergentes) y de sonido (música de fondo y efectos).

1. **Documentación técnica:**

**Detalle de los Módulos Desarrollados**

El proyecto se divide en módulos principales que encapsulan las funcionalidades clave del videojuego. Cada módulo tiene responsabilidades específicas para garantizar la modularidad y escalabilidad del sistema.

**Gestión del Juego (Game):**

-Controla el flujo principal del juego: inicialización, actualización y renderización.

-Cambia entre diferentes estados del juego (menú, opciones, jugando, créditos).

**Jugador (Player):**

Gestiona el movimiento, ataques y estado de salud del personaje controlable.

Incluye:

-Movimiento horizontal y saltos.

-Detección de colisiones con enemigos y objetos

-Gestión de barras de vida y energía.

**Enemigos (Enemy):**

Clases específicas para cada enemigo (Crabby, Shark, Pinkstar), heredando de una clase base Enemy.

Cada enemigo tiene comportamientos únicos:

- Crabby: Movimiento básico y ataques.

-Shark: Movimientos rápidos simulando nado.

-Pinkstar: Ataques en rodillo.

**Gestión de Niveles (LevelManager):**

- Maneja la carga de niveles, incluyendo la ubicación de enemigos y objetos.

- Se encarga de la transición entre niveles.

**Interfaz de Usuario (UI):**

-Diseña menús interactivos (inicio, opciones, créditos) utilizando botones.

-Integra barras de estado (vida y energía) en el juego.

**Audio (AudioPlayer):**

-Carga y reproduce música de fondo y efectos sonoros.

- Sincroniza sonidos con eventos como saltos, ataques y cambios de nivel.

**Efectos Visuales (Rain, DialogueEffect):**

-Rain: Implementa el efecto de lluvia con partículas animadas.

-DialogueEffect: Muestra iconos como signos de exclamación e interrogación en momentos específicos del juego.

**Tecnologías Empleadas**

**Lenguaje de Programación:**

Java: Elegido por su orientación a objetos, robustez y amplia biblioteca estándar.

**Librerías Gráficas:**

Java AWT y Swing: Para renderizar gráficos y manejar eventos de usuario.

**Gestión de Audio:**

Java Sound API: Utilizada para cargar y reproducir archivos de audio (.wav) tanto para música como para efectos sonoros.

**Arquitectura y Diseño:**

Patrón State: Implementado para manejar los diferentes estados del juego (menú, opciones, jugando, créditos).

Hojas de Sprites: Utilizadas para animaciones del jugador y enemigos.

**Justificación de Decisiones Técnicas**

**Uso de Java:**

Java es una opción ideal para proyectos educativos debido a su sintaxis clara y su orientación a objetos.

Las bibliotecas estándar de Java permiten gestionar gráficos y sonido sin necesidad de librerías externas complejas.

**Modularidad:**

Dividir el juego en módulos como Player, Enemy y LevelManager facilita el mantenimiento y la escalabilidad.

Cada módulo tiene responsabilidades claras, lo que permite futuras extensiones, como añadir más enemigos o niveles.

**Efectos Visuales:**

La implementación de efectos como lluvia y diálogos emergentes mejora la inmersión del jugador.

El uso de partículas simuladas garantiza un equilibrio entre calidad visual y rendimiento.

**Patrón State:**

Este patrón asegura que los cambios entre estados del juego (como del menú al juego) sean fluidos y organizados.

**Hojas de Sprites:**

Reducen el consumo de memoria al almacenar múltiples frames de animación en una sola imagen.

Facilitan la creación de animaciones fluidas y visualmente atractivas.

**Gestión del Audio:**

La sincronización de sonidos con eventos del juego aumenta la percepción de inmersión.

Utilizar la API estándar de Java evitó dependencias adicionales.

**6. SDLC (Software Development Life Cycle):**

**1. Análisis**

Objetivo: Crear un videojuego de estilo retro con múltiples niveles, enemigos y un sistema de audio interactivo.

**Requisitos funcionales:**

-Gestión de personajes (jugador, enemigos).

-Niveles con obstáculos y desafíos.

-Sistema de audio (música, efectos).

-Diferentes estados de juego: Menú, Juego, Créditos.

-Animaciones y efectos visuales.

**2. Planificación**

Cronograma

- Día 1-2: Configuración inicial del entorno y diseño del sistema.

- Día 3-6: Desarrollo de la lógica del juego (entidades, niveles).

- Día 7-8: Implementación de audio y efectos visuales.

- Día 9-10: Pruebas unitarias y de integración.

- Día 11: Presentación.

**3. Diseño**

Arquitectura:

- Patrón basado en Estados para manejar el flujo del juego.

Modularidad para separar entidades, lógica del juego y audio.

Diagrama:

**Módulos principales:**

-Gestión de audio.

-Entidades (jugador, enemigos).

-Niveles y objetos interactivos.

-Interfaz gráfica (UI).

Diseño visual:

-Gráficos 2D de píxeles.

-Animaciones de sprites para personajes y enemigos.

**4. Desarrollo**

-Implementar clases de entidades (Jugador, Enemigos) y manejar sus comportamientos.

-Crear sistemas de colisión y detección (por ejemplo, cajas de impacto).

-Añadir efectos visuales (lluvia, diálogos emergentes).

-Configurar audio (canciones, efectos sonoros).

**5. Pruebas**

Pruebas unitarias:

-Movimiento del jugador.

-Comportamiento de los enemigos.

-Reproducción de audio.

Pruebas de integración:

-Interacción entre niveles y entidades.

-Transiciones de estados del juego.

Pruebas de usuario:

-Evaluar experiencia del jugador.

-Identificar problemas de rendimiento.

1. **Mantenimiento**

-Recoger feedback de usuarios para mejorar el diseño y resolver bugs.

-Actualizaciones periódicas para agregar contenido (nuevos niveles, enemigos).

-Optimización para dispositivos de bajo rendimiento.

1. **Diagrama UML**

Diagrama de Clases

+---------------------+

| Game |

+---------------------+

| - gameState: int |

| - menu: Menu |

| - player: Player |

| - levelManager: LevelManager |

| - audioPlayer: AudioPlayer |

| - rain: Rain |

| - dialogueEffect: DialogueEffect |

+---------------------+

| + start(): void |

| + update(): void |

| + render(): void |

| + changeState(int): void |

+---------------------+

+---------------------+

| Player |

+---------------------+

| - x: float |

| - y: float |

| - health: int |

| - inAir: bool |

| - powerValue: int |

| - walkSpeed: float |

+---------------------+

| + move(): void |

| + attack(): void |

| + checkCollision(): void |

| + render(Graphics): void |

+---------------------+

+---------------------+

| Enemy |

+---------------------+

| - health: int |

| - attackBox: Rectangle2D.Float |

| - walkDir: int |

| - inAir: bool |

| - active: bool |

+---------------------+

| + move(): void |

| + attack(): void |

| + takeDamage(int): void |

| + render(Graphics): void |

+---------------------+

+---------------------+

| Crabby |

+---------------------+

| + move(): void |

| + attack(): void |

+---------------------+

^

|

inherits from

|

v

+---------------------+

| Shark |

+---------------------+

| + swim(): void |

| + attack(): void |

+---------------------+

^

|

inherits from

|

v

+---------------------+

| Pinkstar |

+---------------------+

| + rollAttack(): void|

| + attack(): void |

+---------------------+

+---------------------+

| LevelManager |

+---------------------+

| - levels: Level[] |

| - currentLevel: int |

+---------------------+

| + loadLevel(int): void |

| + update(): void |

| + render(Graphics): void |

| + nextLevel(): void |

+---------------------+

+---------------------+

| UI |

+---------------------+

| - menuItems: MenuItem[] |

+---------------------+

| + showMenu(): void |

| + handleInput(): void |

+---------------------+

+---------------------+

| AudioPlayer |

+---------------------+

| - songs: Clip[] |

| - effects: Clip[] |

| - volume: float |

| - songMute: bool |

| - effectMute: bool |

+---------------------+

| + playSong(int): void |

| + stopSong(): void |

| + setVolume(float): void |

| + toggleSongMute(): void |

| + toggleEffectMute(): void |

+---------------------+

+---------------------+

| Rain |

+---------------------+

| - drops: Point2D.Float[] |

| - rainSpeed: float |

+---------------------+

| + update(): void |

| + draw(Graphics): void |

+---------------------+

+---------------------+

| DialogueEffect |

+---------------------+

| - x: int |

| - y: int |

| - aniIndex: int |

| - aniTick: int |

| - active: bool |

+---------------------+

| + update(): void |

| + reset(int, int): void |

| + isActive(): bool |

+---------------------+

Diagrama de casos

@startuml

actor Usuario

actor Sistema

Usuario -> (Iniciar Juego)

Usuario -> (Mover Jugador)

Usuario -> (Atacar Enemigos)

Usuario -> (Cambiar Nivel)

Usuario -> (Gestionar Audio)

Usuario -> (Interactuar con Menú)

(Iniciar Juego) ..> Sistema : <<include>>

(Mover Jugador) ..> Sistema : <<include>>

(Atacar Enemigos) ..> Sistema : <<include>>

(Cambiar Nivel) ..> Sistema : <<include>>

(Gestionar Audio) ..> Sistema : <<include>>

(Interactuar con Menú) ..> Sistema : <<include>>

@enduml

Link de github: https://github.com/Carlos-ufpso