Proyecto II

2024

SOUTHWEST STUDIOS

Technical Design Document

Universitat Politècnica de Catalunya

Diseño y desarrollo de Videojuegos

Diseño, animación y arte digital

TABLA DE CONTENIDOS

Introducción

Propósito

El propósito de este documento es proporcionar una descripción detallada de la arquitectura técnica, el diseño y la implementación del sistema que se está desarrollando. Este documento servirá como una guía para los desarrolladores, arquitectos y otras partes interesadas involucradas en el proyecto, y tiene los siguientes objetivos: Orientación para el Desarrollo, Toma de Decisiones, Documentación y Mantenimiento y Aprobación y Validación.

Definiciones, acrónimos y abreviaciones

- SDL2 (Simple DirectMedia Layer): Es una biblioteca de desarrollo de software que proporciona una interfaz de programación de aplicaciones (API) para acceder a funciones multimedia como gráficos, sonido y entrada de usuario.
- API: Se refiere a Interfaz de Programación de Aplicaciones. Es un conjunto de herramientas y definiciones de protocolos que permite la creación y la integración de software de aplicaciones.
- RAM: Es la Memoria de Acceso Aleatorio. Es un tipo de memoria volátil que se utiliza para almacenar datos temporales y ejecutar programas en un sistema informático.
- SO: Se refiere al Sistema Operativo, el software que gestiona los recursos de hardware y proporciona servicios comunes para otros programas.
- UI: Es la Interfaz de Usuario, que se refiere a los elementos visuales y de interacción que permiten a los usuarios interactuar con un dispositivo o software.
- XML: Es un Lenguaje de Marcado Extensible. Se utiliza para almacenar y transportar datos de manera legible tanto para humanos como para máquinas, utilizando etiquetas que definen la estructura y el significado de los datos.
- C++: Es un lenguaje de programación de propósito general creado como una extensión del lenguaje C. Es conocido por su eficiencia y flexibilidad, permitiendo tanto la programación de bajo nivel (manipulación de memoria) como de alto nivel (abstracciones de objetos).
- DLL: Es una Biblioteca de Vínculos Dinámicos. Es un archivo externo que contiene funciones y datos que pueden ser utilizados por otros programas.

Plataforma

Hardware

Requisitos Mínimos:

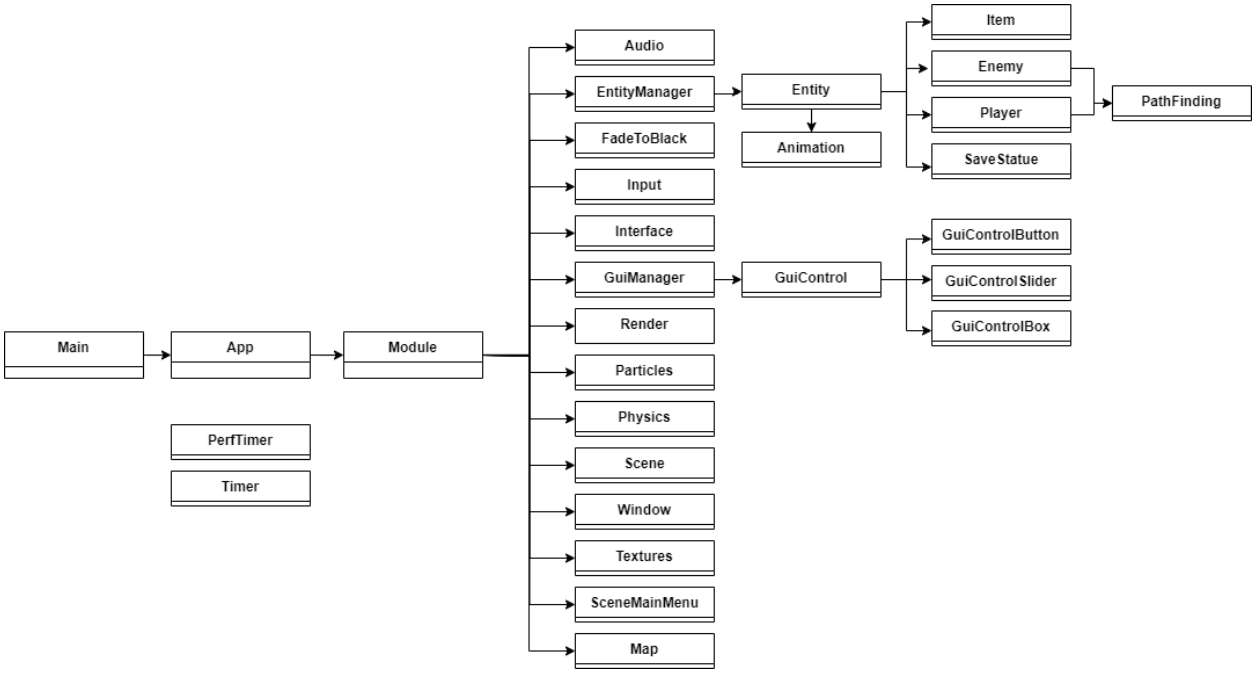
- Sistema Operativo: Windows 7
- Procesador: 2.33 GHz
- Almacenamiento: 1GB de espacio disponible en disco
- Memoria RAM: 256MB
- Resolución de pantalla: Se recomienda una resolución mínima de 1920x1080 para una experiencia de juego adecuada.

Requisitos Recomendados:

- Sistema Operativo: Windows 10
- Procesador: 3.5 GHz o superior
- Almacenamiento: 1GB de espacio disponible en disco
- Memoria RAM: Se recomienda una capacidad de al menos 512MB para un rendimiento óptimo del juego.
- Resolución de pantalla: Se recomienda una resolución mínima de 1920x1080 para una experiencia de juego inmersiva y de alta calidad.

Arquitectura

Diagrama



Librerías Externas

SDL2 (Simple DirectMedia Layer) es una biblioteca de desarrollo de software que proporciona una interfaz de programación de aplicaciones (API) para acceder a funciones multimedia, como gráficos, sonido y entrada de usuario.[1]

SDL2_image es una biblioteca de carga de archivos de imagen para SDL2. Permite cargar imágenes en forma de superficies y texturas SDL, y admite varios formatos, como BMP, GIF, JPEG, PNG, TGA, TIFF, WEBP, XCF, XPM y más. [2]

libjpeg es una biblioteca gratuita que maneja el formato de datos de imágenes JPEG. Proporciona un códec (para codificar y decodificar) imágenes JPEG, junto con diversas utilidades para trabajar con datos JPEG.[3]

libpng16-16 es una biblioteca que maneja el formato de datos de imágenes PNG. Proporciona un códec para codificar y decodificar imágenes en formato PNG. Además, incluye utilidades para trabajar con datos PNG.[4]

Libtiff-5 es una biblioteca para leer y escribir archivos en formato TIFF. Es útil para manipular imágenes TIFF en aplicaciones y sistemas compatibles.[5]

Libwebp es una biblioteca desarrollada por Google para codificar y decodificar imágenes en formato WebP. Esta biblioteca es esencial para trabajar con imágenes en este formato. [6]

SDL2_mixer es una biblioteca de mezcla de audio de múltiples canales para SDL2. Permite cargar y reproducir archivos de audio en formatos como FLAC, MP3, Ogg, VOC y WAV. Es esencial para agregar sonido y música a tus aplicaciones y juegos basados en SDL2. [7]

Libogg es una biblioteca que se utiliza para leer y escribir archivos en formato Ogg. Esta biblioteca es esencial para manipular datos de imágenes y sonido en este formato. [8]

SDL2_ttf es una biblioteca que permite usar fuentes TrueType para renderizar texto en aplicaciones SDL. Funciona como un envoltorio alrededor de las bibliotecas FreeType y Harfbuzz. Con SDL2_ttf, puedes cargar y mostrar fuentes TrueType sin tener que escribir tu propia rutina de renderizado de fuentes. [9]

libFLAC es una biblioteca que implementa codificadores y decodificadores de referencia para FLAC nativo y Ogg FLAC, junto con una interfaz de metadatos. [10]

LibFreetype es una biblioteca utilizada para leer y escribir archivos en formato TrueType. Es especialmente útil para trabajar con fuentes de texto en aplicaciones y sistemas que requieren soporte para fuentes TrueType. La biblioteca proporciona funciones para cargar y renderizar fuentes, lo que permite mostrar texto de manera legible en pantalla. [11]

Libmikmod es una biblioteca que nos permite reproducir ficheros de audio modulares. Estos ficheros, creados por secuenciadores populares como MOD, IT, S3M, STM y XM, almacenan muestras de sonido de instrumentos que se organizan en secuencias o patrones. [12]

Libmodplug es una biblioteca que se utiliza para leer y escribir archivos en formato MOD (módulos de música). Estos archivos MOD contienen datos de muestras de sonido e instrucciones de reproducción, lo que permite crear música en secuencias. La biblioteca está basada en el código de renderizado de sonido de ModPlug, que es ampliamente considerado como uno de los mejores reproductores de archivos MOD disponibles.[13]

Libvorbis es la implementación de referencia del códec Vorbis. Vorbis es un formato de compresión de audio sin pérdida, contemporáneo al AAC de MPEG-4 y TwinVQ. A diferencia de los formatos patrocinados por MPEG (y otros formatos propietarios como RealAudio G2 y la moda del mes de Windows), la especificación del códec Vorbis pertenece al dominio público. Todos los detalles técnicos están publicados y documentados, y cualquier entidad de software puede utilizar el formato sin tarifas de licencia, regalías o preocupaciones de patentes.[14]

Libvorbisfile es una biblioteca que simplifica el uso de archivos de audio en formato Vorbis. Proporciona una interfaz conveniente para cargar y reproducir archivos Vorbis sin tener que lidiar directamente con detalles de bajo nivel.[15]

zlib1 es una biblioteca de compresión de datos sin pérdida diseñada para ser gratuita, general y legalmente desvinculada de patentes. Puede utilizarse en prácticamente cualquier hardware y

sistema operativo. A diferencia del método de compresión LZW utilizado en Unix compress(1) y en el formato de imagen GIF, zlib1 no expande los datos y su huella de memoria es independiente de los datos de entrada. [16]

Box2D es una biblioteca de cuerpos rígidos en 2D. Permite crear físicas realistas, incluyendo colisiones, gravedad y movimiento. [17]

Sistema de documentos

Nomenclaturas:

- **Archivos de código:**
  - Nombres descriptivos de la funcionalidad del documento
  - Si son dos o más palabras, separadas por mayúsculas: *EntityManager.cpp, EnemyAI.cpp*

- **Archivos de recursos:**
  - Prefija los nombres con un identificador que indique el tipo de recurso.
  - Guiones bajos (_) para indicar espacios.
  - Por ejemplo: *textura_personaje.png, textura_pueblo.png*

- **Documentación y archivos de diseño:**
  - Nombres claros y descriptivos para identificar el contenido.
  - Versión y/o fecha de los archivos si es necesario.
  - Por ejemplo: *Documento_Diseño_V1.pdf, Especificaciones_Técnicas.pdf*

- **Configuración y archivos de recursos adicionales:**
  - Usa nombres que reflejen su propósito y función en el juego.
  - Mantén la consistencia en la nomenclatura para facilitar la búsqueda y gestión de archivos.
  - Por ejemplo: *Licencia.txt, Tutorial.pdf*

Rutas

- ★ **Output:** Carpeta contenedora de todas las dll's, xml's y assets necesarios
    - **Assets:**
        - **Texturas:** Contiene todas las texturas del juego, se subdivide en:
            - Entidades
                - Jugador
                - Enemigos
                    - Enemigo1
                    - Enemigo2
                    - Etc.
                - Items
                    - Espadas
                    - Armaduras
                    - Pociones
                    - Etc.
            - Interfaz
                - Menú
                - Inventario
                - Dentro del juego
                - Etc.
            - Mapas
                - Metadatas
                - Pueblo
                - Mazmorras
                    - Nivel1
        - **Audios:** Contiene todos los archivos de sonido del juego, se subdivide en
            - Música
            - FX
        - **Mapas:** Contiene todos los archivos de mapas y niveles, se subdividen en:
            - Pueblo
            - Mazmorras
                - Nivel1
        - **Fuentes:** Contiene todas las fuentes del juego.

Tecnología

Lenguajes tecnológicos

- **C++:**
  - C++ es un lenguaje de programación de propósito general creado como una extensión del lenguaje C.
  - Es conocido por su eficiencia y flexibilidad, permitiendo tanta programación de bajo nivel (manipulación de memoria) como de alto nivel (abstracciones de objetos).
  - C++ es ampliamente utilizado en el desarrollo de sistemas operativos, software de aplicaciones, juegos, dispositivos embebidos, entre otros.
  - Ofrece características como la programación orientada a objetos, plantillas (templates), gestión de memoria manual y polimorfismo, entre otros.
  - Es un lenguaje compilado, lo que significa que el código fuente se traduce a un código de máquina específico para la plataforma de destino antes de su ejecución.
- **XML:**
  - XML es un lenguaje de marcado que se utiliza para almacenar y transportar datos de manera legible tanto para humanos como para máquinas.
  - Su sintaxis consiste en etiquetas que definen la estructura y el significado de los datos.
  - XML es extensible y flexible, lo que significa que los usuarios pueden definir sus propias etiquetas y estructuras de datos específicas para su dominio.
  - Se utiliza ampliamente en la comunicación entre sistemas, intercambio de datos, configuración de aplicaciones, almacenamiento de configuraciones, y en muchos otros contextos.
  - Aunque XML es legible para los humanos, no está diseñado para ser eficiente en términos de tamaño de archivo o velocidad de procesamiento, por lo que a menudo se prefiere otros formatos como JSON para ciertos casos de uso.

Código base

El código base que emplearemos para poder crear nuestro proyecto es el template que nos proporcionó el profesor Pedro Omedas en la asignatura de Desarrollo de Videojuegos.

Podemos encontrarlo en este link:
https://github.com/pomedas/citm_desvj_project_template?tab=readme-ov-file

Esta plantilla nos proporciona todo lo necesario para desarrollar nuestro proyecto, utilizando SDL2 como principal librería, y C++ como lenguaje de programación.

Estilo de programación

Principio SOLID

Aplicaremos los principios SOLID para crear un código modular, flexible y mantenible:

- S (Single Responsibility Principle): Cada clase debe tener una sola razón para cambiar y debe ser responsable de una sola tarea.
- O (Open/Closed Principle): Las clases deben estar abiertas para la extensión pero cerradas para la modificación.
- L (Liskov Substitution Principle): Los objetos deben ser reemplazables por instancias de su subtipo sin afectar la integridad del programa.
- I (Interface Segregation Principle): Las interfaces deben ser específicas para los clientes que las utilicen, evitando interfaces monolíticas.
- D (Dependency Inversion Principle): Los módulos de alto nivel no deberían depender de módulos de bajo nivel. Ambos deberían depender de abstracciones. Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

Convenciones

Seguiremos las convenciones de nomenclatura establecidas para mantener un código consistente y fácilmente comprensible:

- **Nombres Descriptivos**: Utilizaremos nombres descriptivos y significativos para variables, métodos, clases y otros elementos del código.
- **CamelCase:** Seguiremos la convención CamelCase para **nombrar variables**(p. ej., nombreVariable).
- **PascalCase:** Utilizaremos PascalCase para nombrar **clases** y **métodos** tipos (p. ej., NombreClase, CalcularCosto()).
- **ScreamingSnakeCase:** Utilizaremos ScreamingSnakeCase para nombrar **tipos** (p. ej, TIPO_DATO, PLATAFORMA_MOVIBLE).

Documentación del Código

Documentaremos adecuadamente el código para facilitar la comprensión y el mantenimiento:

- Comentarios Descriptivos: Incluiremos comentarios descriptivos que expliquen el propósito y la funcionalidad de las secciones críticas del código.
- Documentación de Métodos: Documentaremos los métodos con claridad, explicando sus parámetros, su comportamiento y su propósito. Esta explicación estará compuesta de la siguiente manera:

  ```
  /**
  * Descripción del metodo
  * @param [Nombre del parametro]: Para qué sirve ese parámetro
  * @return Explicación de que devuelve
  */
  ```

Refactorización del Código

Practicaremos la refactorización continua para mejorar la calidad del código sin cambiar su comportamiento externo:

- Eliminación de Duplicaciones: Identificaremos y eliminaremos duplicaciones de código para mejorar la legibilidad y la mantenibilidad.
- Mejora de la Estructura: Refactorizaremos el código para mejorar su estructura y su diseño, siguiendo los principios de diseño limpio.

Físicas y Colisiones

Para el sistema de físicas y colisiones usaremos **box2d,** un motor de física en dos dimensiones, “Open Source”, escrito en c + + y creado por Erin Catto. El motor se ha usado en proyectos como “Shovel Knight”, “Angry Birds”y “Happy Wheels” entre otros.

Box2D realiza una simulación de cuerpo rígido restringido. Puede simular cuerpos compuestos por polígonos convexos, círculos y formas de bordes. Los cuerpos están unidos mediante articulaciones y sobre ellos actúan fuerzas. El motor también aplica gravedad, fricción y restitución.

En nuestro juego usaremos un módulo llamado “physics” el cual se encargará de crear y gestionar los diferentes cuerpos, así como aplicar diferentes fuerzas a los objetos o al mundo.

En cuanto a las colisiones, generaremos un tipo de colisión propia para cada cuerpo y gestionaremos su comportamiento mediante una función “OnColision()”, que gracias a box2d identificara la colisión entre dos cuerpos y reaccionará a nuestra voluntad.

Inteligencia artificial y algoritmo pathfinding

Usaremos inteligencia artificial para el algoritmo de “pathfinding”, concretamente usaremos “A*”. Este se utiliza en muchos campos de la informática debido a su integridad, optimización y eficiencia óptima. Dado un gráfico ponderado, un nodo fuente y un nodo objetivo, el algoritmo encuentra el camino más corto desde el origen hasta el objetivo. “A*”, a diferencia de “Dijkstra”, usa heurística para una mayor eficacia.

En nuestro proyecto, crearemos un módulo de pathfinding de A*. Este módulo tendrá una función principal que, a partir de un origen y un objetivo, encontrará el camino más corto y eficiente posible. La llamada a esta función estará en entidades que deban buscar o perseguir a otras entidades, por ejemplo la entidad “enemigo”, que llamara a la función “CreatePath()” para encontrar el camino entre él y el jugador.

El algoritmo A* se basa en la evaluación de los nodos del grafo utilizando dos funciones:

1. **Función de costo real** g(n): Representa el costo real de llegar desde el nodo inicial hasta el nodo actual.
2. **Función heurística** h(n): Estima el costo restante desde el nodo actual hasta el nodo objetivo.

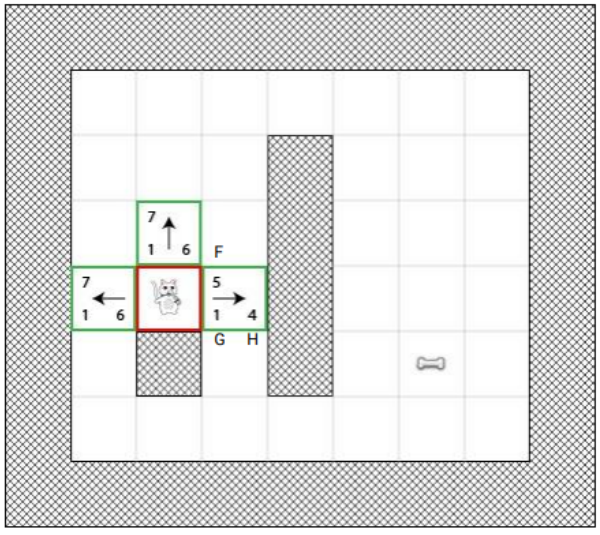
A partir de estas funciones, se calcula una funcion de evaluación f(n) = g(n) + h(n) para cada nodo, donde n es un nodo en el grafo. El algoritmo selecciona los nodos para su exploración basándose en esta funcion de evaluacion.

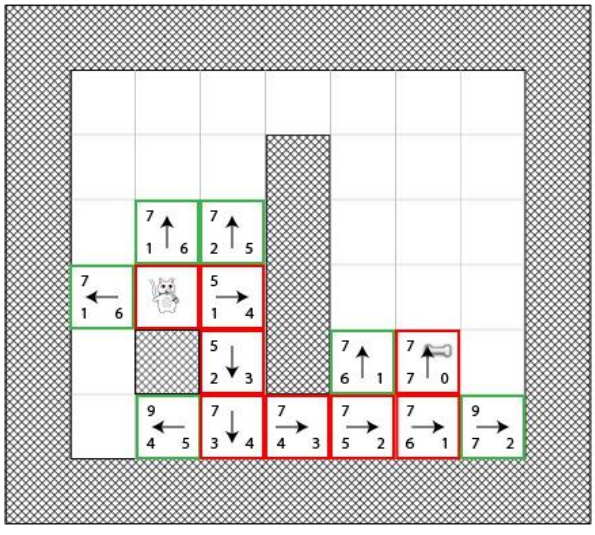
Pseudocódigo del algoritmo

```
Función A*(inicio, objetivo):
    Agregar el nodo de inicio a la lista abierta
    Mientras la lista abierta no esté vacía:
        Seleccionar el nodo con el menor valor de f(n) de la lista abierta
        Si el nodo seleccionado es el objetivo:
            Retornar el camino reconstruido
        Mover el nodo seleccionado de la lista abierta a la lista cerrada
        Para cada vecino del nodo seleccionado:
            Si el vecino no es transitable o está en la lista cerrada:
                Continuar al siguiente vecino
            Calcular los valores de g(n) y h(n) para el vecino
            Si el vecino no está en la lista abierta o tiene un valor de f(n) menor:
                Establecer el nodo seleccionado como el padre del vecino
                Actualizar los valores de g(n) y h(n) del vecino
                Si el vecino no está en la lista abierta:
                    Agregar el vecino a la lista abierta
    Retornar "No se encontró camino"
```

Al implementar el algoritmo A*, se deben considerar varios aspectos importantes:

La función heurística debe ser admisible y consistente para garantizar la optimalidad y la eficiencia del algoritmo.
Se debe implementar una estructura de datos eficiente para almacenar la lista abierta y la lista cerrada, como una cola de prioridad o un montículo binario.
Es crucial elegir una heurística adecuada para el problema específico, ya que puede afectar significativamente el rendimiento del algoritmo.





Estrategias de optimización

Recursos

Optimizaremos el uso de recursos del sistema para reducir la carga y mejorar el rendimiento:

- Gestión de Memoria: Implementaremos prácticas de gestión de memoria eficientes para minimizar las fugas de memoria y reducir el uso innecesario de recursos.
- Optimización de Algoritmos: Revisaremos y optimizaremos los algoritmos existentes para mejorar la eficiencia temporal y espacial.
- Paralelismo y Concurrencia: Utilizaremos técnicas de paralelismo y concurrencia para distribuir la carga de trabajo y aprovechar los recursos del hardware de manera eficiente.

Mapeado

En el desarrollo de videojuegos y aplicaciones gráficas, la eficiencia en la carga y renderizado de texturas es crucial para asegurar un rendimiento óptimo y una experiencia de usuario fluida. Una técnica ampliamente utilizada para mejorar esta eficiencia es la gestión de texturas mediante "chunks". Esta técnica divide el mapa completo en pequeñas secciones manejables, permitiendo que solo las texturas necesarias se carguen y rendericen en cualquier momento dado.

El sistema de optimización de la carga de texturas se basa en los siguientes componentes y funcionalidades:

1. **División del Mapa en Chunks**:
    - El mapa completo se divide en varias secciones más pequeñas, llamadas "chunks".
    - Cada chunk contiene una parte específica de las texturas del mapa.
2. **Carga Dinámica de Texturas**:
    - Las texturas de los chunks se cargan dinámicamente en función de la posición del jugador o cámara.
    - Solo los chunks visibles o cercanos al jugador se cargan en la memoria, reduciendo el uso de recursos.
3. **Descarte de Texturas No Necesarias**:
    - Los chunks que salen del área visible o de interés se descargan de la memoria.
    - Esto asegura que la memoria solo contenga las texturas necesarias en cualquier momento dado.
4. **Actualización Continua**:
    - El sistema continuamente monitorea la posición del jugador y ajusta la carga y descarga de texturas en tiempo real.
    - Esto permite una experiencia fluida y sin interrupciones.

Cache

Implementaremos estrategias de cacheado de datos para reducir el tiempo de acceso a datos y mejorar la velocidad de respuesta:

- Cache de Memoria: Utilizaremos caches en memoria para almacenar datos frecuentemente accedidos y evitar la recuperación repetitiva desde el almacenamiento persistente.
- Cache de Resultados: Cachearemos los resultados de operaciones costosas para evitar su recálculo innecesario en futuras solicitudes.

Monitoreo y Ajuste Continuo

Implementaremos sistemas de monitoreo para medir el rendimiento del sistema en tiempo real y realizar ajustes según sea necesario:

- Monitoreo de Rendimiento: Utilizaremos herramientas de monitoreo para medir y analizar métricas de rendimiento clave, como tiempo de respuesta, utilización de recursos y carga de trabajo.
- Ajuste Dinámico: Realizaremos ajustes dinámicos en la configuración y la infraestructura del sistema para optimizar el rendimiento en función de los cambios en la carga de trabajo y los patrones de uso.

Herramientas

- Plataforma de Desarrollo (Visual Studio 2022):
  El proyecto se desarrollará principalmente en el entorno de desarrollo integrado (IDE) Visual Studio, utilizando las herramientas y funcionalidades proporcionadas por este entorno para facilitar el desarrollo y depuración del código.

- Gestión de Dependencias:
  Se utilizarán herramientas de gestión de dependencias, como CMake, para administrar las dependencias externas del proyecto y facilitar la compilación y el enlace de bibliotecas.

- Pruebas y Depuración:
  Se implementarán pruebas unitarias y de integración para garantizar la calidad y la estabilidad del código.
  Se utilizarán herramientas de depuración como Visual Studio Debugger para identificar y solucionar errores de manera eficiente.

- Optimización de Rendimiento (Optick):
  Para mejorar el rendimiento del proyecto, se empleará la herramienta de optimización de rendimiento "Optick". Optick es una herramienta especializada que permite el análisis detallado del rendimiento de la aplicación en tiempo de ejecución.

Flujo de trabajo

Método de diseño

- ***Método de Desarrollo Ágil:*** El desarrollo ágil enfatiza la flexibilidad y la colaboración entre los miembros del equipo. El proceso de desarrollo se divide en iteraciones cortas o "sprints", durante las cuales se desarrollan, prueban e integran características o componentes específicos del juego. El enfoque ágil permite una rápida adaptación a cambios y retroalimentación a lo largo del ciclo de desarrollo.

Estándares

- **Estándares de Diseño de Interfaz de Usuario (UI):**
    - Claridad y simplicidad en la disposición de menús e indicaciones de botones.
    - Lenguaje visual y iconografía consistentes.
- **Estándares de Diseño de Jugabilidad:**
    - Mecánicas de juego equilibradas que proporcionen una experiencia justa y atractiva.
    - Controles claros e intuitivos que respondan adecuadamente a la entrada del jugador.
    - Sistemas de progresión del jugador que ofrezcan una sensación de logro y recompensa.
- **Estándares de Diseño de Niveles:**
    - Curvas de dificultad bien ajustadas que introduzcan y desafíen gradualmente a los jugadores.
    - Uso efectivo del espacio, obstáculos y narrativa ambiental.
    - Exploración equilibrada y orientación para evitar que los jugadores se sientan perdidos.
- **Estándares de Diseño Narrativo:**
    - Narrativa convincente que involucre emocional e intelectualmente a los jugadores.
    - Construcción de mundo y lore coherentes que enriquezcan el universo del juego.
    - Elecciones y consecuencias significativas para el jugador que impacten en la experiencia del jugador.
- **Estándares de Diseño de Arte y Audio:**
    - Visuales y animaciones de alta calidad que mejoren la inmersión y el atractivo estético.
    - Diseño de sonido atmosférico y música que complementen el estado de ánimo y la ambientación del juego.
    - Optimización para el rendimiento para garantizar tasas de fotogramas suaves y tiempos de carga mínimos.

Control de Versiones

Debemos mantener un historial de cambios y revisiones realizadas en el documento a lo largo del tiempo, teniendo que cumplir unos requisitos para cada iteración del documento para poder rastrear y gestionar los cambios de manera efectiva. Para eso usaremos GitHub.

- GitHub
    - Creación del equipo y repositorio de Github.
        - Equipo: SouthWest Studios
        - Repositorio: Suregs Mask.
    - Todo el equipo debe tener acceso al repositorio y asegurarse de que esté actualizado antes de realizar cualquier cambio.
    - Cada vez que se modifique o se añada contenido del repositorio, el nombre del commit deberà ser claro y conciso, indicando los cambios que se han hecho y añadiendo una descripción si es necesario.
    - Estos cambios deberán de ser revisados y aprobados por el Jefe de programación, y en caso de no ser aprobados, deberán ser retocados hasta que cumplan con los requisitos.

Prácticas de colaboración

- Asignación de roles y responsabilidades
    - Los miembros del equipo nos dividimos en diferentes grupos (arte, programación, narrativa...) Para trabajar de manera eficiente.

- Reuniones
    - Al menos una reunión semanal del equipo entero para informar de cambios/actualizaciones. La fecha y hora de la reunión se deciden en función de la disponibilidad de los miembros del equipo.
    - Al menos una reunión semanal de los sub equipos del proyecto para repartirse el trabajo y avanzar. La fecha y hora de la reunión se deciden en función de la disponibilidad de los miembros del equipo.

- Contacto
    - De manera presencial
    - Por Discord mediante el servidor del equipo SouthWest Studios.
    - Por Whatsapp para organizar las reuniones y el trabajo.

Proceso de entrega

El proceso de entrega del videojuego se llevará a cabo mediante la creación de una build en modo release y la posterior distribución del contenido en un formato comprimido. A continuación se detallan los pasos que se seguirán para realizar esta entrega:

1. Generación de la Build en Modo Release:
    - Se realizará la compilación del proyecto en modo release utilizando las herramientas de desarrollo adecuadas.
    - Durante este proceso, se optimizará el código y se eliminarán los recursos de depuración para garantizar un rendimiento óptimo del videojuego.

2. Inclusión de Archivos Necesarios:
    - La build en modo release contendrá todos los archivos necesarios para la ejecución del videojuego.
    - Se incluirán la carpeta contenedora del ejecutable del juego, junto con sus dependencias y recursos (assets) requeridos para su funcionamiento.

3. Agregado de Documentación:
    - Se adjuntará un archivo README.md que proporcionará información detallada sobre el videojuego, incluyendo instrucciones de instalación, requisitos del sistema y cualquier otra información relevante para el usuario final.
    - Además, se incluirá un archivo LICENSE.md que establecerá los términos y condiciones de uso del videojuego, garantizando la protección de los derechos de autor y la propiedad intelectual.

4. Estructura final del archivo zip:

```
Team_Name-Platformer-[Version].zip      // Game directory
[Videogame Name]                       // Game directory
    Assets                             // Assets directory, it could contain
                                       // multiple sub-dirs and files
    Game.exe                           // Main binary for the game
    Xxx.dll                            // All required DLLs to run the game
LICENSE.md                             // Game license file
README.md                              // Game detailed info
```

5. Compresión del Contenido:
    - Todos los archivos mencionados anteriormente se comprimirán en un archivo ZIP para facilitar su distribución y manejo.
    - El archivo ZIP contendrá una estructura organizada que permita una fácil comprensión y acceso a los componentes del videojuego.

6. Distribución en Plataformas Designadas:
    - El archivo ZIP generado se subirá al apartado de releases del repositorio del proyecto en GitHub.
    - Asimismo, se compartirá el archivo ZIP en la plataforma de gestión del curso, como Atenea, para que esté disponible para el profesorado.

Referencias

Referencia 1 - SDL Wiki -
SDL2/Tutorials - SDL Wiki (libsdl.org)

Referencia 2 - Introduction to SDL_Image -
SDL_Image | Take Screenshots and Load Images into SDL2 | StudyPlan.dev

Referencia 3 - Github libjpeg - @jinahphrodite -
GitHub - winlibs/libjpeg: A free library for JPEG image compression

Referencia 4 - libpng16 Wiki -
http://www.libpng.org/pub/png/libpng.html

Referencia 5 - Libtiff-5 documentation -
LibTIFF - TIFF Library and Utilities — LibTIFF 4.6.0 documentation

Referencia 6 - Libwebp documentation -
https://github.com/webmproject/libwebp

Referencia 7 - SDL_Mixer Documentation -
SDL2_mixer/Mix_Resume - SDL Wiki (libsdl.org)

Referencia 8 - libogg Documentation -
https://github.com/gcp/libogg

Referencia 9 - SDL2_ttf Documentation -
https://wiki.libsdl.org/SDL_ttf

Referencia 10 - FALC Documentation -
https://github.com/xiph/flac/releases

Referencia 11 - FreeType Documentation -
https://freetype.org/download.html

Referencia 12 - Introducción al uso de libmikmod para audio modular en el computador -
https://riunet.upv.es/handle/10251/169041

Referencia 13 - libmodplug Documentation -
https://github.com/Konstanty/libmodplug

Referencia 14 - Libvorbis Documentation -
https://github.com/xiph/vorbis

Referencia 15 - DLL Files -
https://es.dll-files.com/libvorbisfile.dll.html

Referencia 16 - zlib1 Documentation -
https://www.zlib.net/

Referencia 17 - Box2D Wiki -
Box2D: Overview

Referencia 18 - A* Wiki -
https://en.wikipedia.org/wiki/A*_search_algorithm