Editor y motor de juegos 2D para no programadores

TRABAJO DE FIN DE GRADO

Pablo Fernández Álvarez
Yojhan García Peña
Iván Sánchez Míguez

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2023

Documento maquetado con TEXIS v.1.0+.

Editor y motor de juegos 2D para no programadores

Memoria que se presenta para el Trabajo de Fin de Grado

Pablo Fernández Álvarez, Yojhan García Peña e Iván Sánchez Míguez

Dirigida por el Doctor

Pedro Pablo Gómez Martín

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2023

Agradecimientos

Queremos expresar nuestro sincero agradecimiento a todos aquellos que han contribuido de manera significativa en nuestro desarrollo como estudiantes. En primer lugar, extendemos nuestro reconocimiento a nuestros respetados profesores, cuya orientación y sabiduría han sido fundamentales para guiarnos a lo largo de este proceso académico. Sus conocimientos compartidos y su apoyo constante nos han permitido crecer y prosperar en este proyecto. Además, deseamos mostrar nuestro agradecimiento a nuestras familias, cuyo inquebrantable respaldo y ánimo han sido una fuente inagotable de motivación. Su apoyo emocional y comprensión han sido esenciales para superar los desafíos y celebrar los logros. Nuestro más sincero agradecimiento a todos aquellos que han estado a nuestro lado en este viaje, ayudándonos a alcanzar este hito académico.

Resumen

Editor y motor de juegos 2D para no programadores

Un motor de videojuegos es un entorno de desarrollo que proporciona herramientas para la creación de videojuegos. Estas herramientas evitan al desarrollador implementar gran cantidad de funcionalidad para centrase en mayor medida en el desarrollo del videojuego. Algunos ejemplos de funcionalidades que aportan los motores son: rendereizado gráfico, motor de físicas, sistema de audio, gestión de input del jugador, gestión de recursos, gestión de red, etc.

Además, pueden llevar integrado un editor. Los editores son herramientas visuales cuyo objetivo es comunicar al motor las acciones que realiza el desarrollador. Por lo tanto, forman parte del entorno de desarrollo del motor. Los editores suelen tener una curva de aprendizaje, especialmente para aquellos que no están familiarizados con el motor en particular o con el desarrollo de videojuegos en general. Sin embargo, una vez que los desarrolladores se familiarizan con las herramientas, pueden acelerar significativamente el proceso de creación del juego y mejorar la productividad.

Esto supone una gran ventaja a los desarrolladores experimentados pero motores como Unity o UnrealEngine pueden albergar demasiada complejidad para personas sin experiencia en programación, incluso aunque su objetivo sean juegos sencillos en 2D. Una herramienta muy útil para solucionar este problema es la programación visual. Este tipo de programación permite a los usuarios crear lógica mediante la manipulación de elementos gráficos en lugar de especificarlos exclusivamente de manera textual. Unity cuenta con su Unity Visual Scripting y UnrealEngine con los Blueprints.

El motor de este trabajo de fin de grado consiste en un entorno de desarrollo de videojuegos 2D autosuficiente. Esto quiere decir que permitirá gestionar los recursos del videojuego, las escenas y los elementos interactivos. Además, dará soporte para la creación de comportamientos a través de programación visual basada en nodos, la ejecución del juego en el editor y la creación de ejecutables finales del juego para su distribución.

Abstract

Game engine and editor 2D for non-programmers

A game engine is a development environment that provides tools for creating video games. These tools allow the developer to avoid implementing a large amount of functionality and focus more on the game's development. Some examples of functionalities that engines provide include graphic rendering, physics engine, audio system, player input management, resource management, network management, etc.

In addition to the mentioned features, game engines can include an editor. Editors are visual tools aimed at communicating the developer's actions to the engine. Therefore, they are part of the engine's development environment. It is worth noting that game engine editors often have a learning curve, especially for those unfamiliar with the specific engine or game development in general. However, once developers become familiar with the tools, they can significantly accelerate the game creation process and improve productivity.

This represents a significant advantage for experienced developers, but engines like Unity or UnrealEngine can be overly complex for people without programming experience, even if their goal is to create simple 2D games. A very useful tool to solve this problem is visual programming. This type of programming allows users to create logic by manipulating graphical elements instead of exclusively specifying them in text. Unity has its Unity Visual Scripting and UnrealEngine has Blueprints.

The engine for this final degree project consists of a self-sufficient 2D game development environment. This means that it will allow managing game resources, scenes, and interactive elements. Additionally, it will support the creation of behaviors through visual programming based on nodes, the game's execution in the editor, and the creation of final game executables for distribution.

Índice

Capítulo 1

Introducción

1.1. Motivación

En lo relacionado a motores de videojuegos durante el grado, en primero aprendimos las bases de Unity, en segundo desarrollamos videojuegos 2D sencillos con SDL con una capa de abstracción para facilitar el aprendizaje y en tercero tuvimos que salir de la zona de comfort y aprender que hay detrás de las herramientas que nos proporcionaban durante el grado desarrollando así un motor de videojuegos 3D usando OGRE como motor gráfico, BulletPhysics como motor físico, FMOD como librería de sonido y Lua como lenguaje de scripting.

Con esta experiencia nos dimos cuenta de varias cosas. Es más comodo trabajar con un motor que cuenta con un editor integrado en vez de acceder a él directamente a través de programación. Además, la forma de programar los scripts debe ser cómoda y inuida ya que es la tarea que más más tiempo va a ocupar al desarrollaor.

En cuarto y con la reciente experiencia de desarrollo de un motor decidimos que la propuesta del trabajo de fin de grado sería un motor con la siguientes características:

- `Autosuficiente`: Esto siginifca que aporta funcionalidad para manejar recursos, crear escenas y objetos desde el editor y cuenta con la creación de ejecutables finales del juego para su distribución. Además, el motor podrá mostrar la ejecución del juego directamente en el editor durante su desarrollo. Obviamente no tiene toda la funcionalidad necesaria como para no depender de ninguna herrmienta externa durante el desarrollo pero si aporta lo básico para el ciclo de desarrollo de un videojuego 2D.
- `Editor integrado`: La principal herramienta de un motor autosuficiente es el editor, encargado de comunicar al motor las accciones del desarrollador. No teníamos experiencia en desarrollo de aplicaciones de

escritorio, ya que para el motor de tercero no hicimos editor, por lo que sabíamos que podía ser un reto.

- `Programación visual basada en nodos`: Esta es la parte a destacar de nuestro motor. Debido a la complejidad que presentan algunos motores de la actualidad para desarrolladores principiantes o inexpertos, la programación visual es una herramienta muy útil e intuitiva para crear lógica y comportamiento en el videojuego. Sabíamos que podía ser un reto a nivel técnico pero aportaría mucha comodidad, fluidez y por supuesto abriría las puertas de nuestro motor a muchos desarrolladores con poca experiencia en programación.

Por último, optamos por el 2D debido a la experiencia obtenida durante la carrera. Además, supone menor complejidad a nivel técnico durante el desarrollo del motor.

1.2. Objetivos

El objetivo principal del trabajo de fin de grado es desarrollar el motor de videojuegos 2D autosuficiente con editor integrado y programación visual basada en nodos. Con esto, los usuarios dispondrán de una herramienta para desarrollar cualquier tipo de videojuego 2D.

Con la autosuficiencia del motor queremos conseguir que los usuarios puedan desarrollar sus videojuegos con la mínima dependencia posible de herramientas externas de tal forma que todo el trabajo pase por el editor.

Con el editor queremos conseguir que el desarrollo sea cómodo, fluido, visual evitando así que el desarrollador se tenga que comunicar con el motor vía programación.

Con el sistema de programación visual basada en nodos queremos conseguir abrir las puertas de nuestro motor a desarrolladores inexpertos o con bajos cononcimientos en programación para que así puedan desarrollar sus videojuegos. Además, supone un reto a nivel técnico que nos aportará experiencia y conocimiento.

1.3. Herramientas

Para comenzar, se ha utilizado Git como sistema de control de versiones a través de la aplicación de escritorio GitHub Desktop. Todo el código implementado se ha subido a un repositorio dividiendo el trabajo en ramas. En concreto, tal y como se cuenta en el plan de trabajo, el proyecto se ha dividido en motor, editor y programación visual.

Para el desarrollo del motor se ha hecho uso de la librería de SDL (Simple DirectMedia Layer). En concreto, se ha utilizado SDL Image como sistema

de gráficos, SDL Mixer como sistema de audio y SDL TTF como sistema de fuentes TrueType para renderizar texto. También se ha utilizado la librería Box2D para la simulación física y manejo de colisiones 2D.

Para el desarrollo del editor se ha hecho uso de la librería de interfaz gráfica ImGUI.

El código ha sido desarrollado en el entorno de desarrollo integrado (IDE) Visual Studio 2022 y escrito en C++. Por último, para la organización de tareas hemos usado la herramienta de gestión de proyectos Trello.

1.4. Plan de trabajo

El trabajo se divide en tres partes: el desarrollo del motor, el desarrollo del editor y el desarrollo del sistema de scripting visual basado en nodos.

- El motor consta de varios proyectos de Visual Studio donde se dividen las partes fundamentales del mismo:

- `Proyecto principal`: Este proyecto implementa el bucle principal del motor y de general el ejecutable de juego.
- `Proyecto de físicas`: Implementa un manager con el que acceder a la configuración del mundo físico.
- `Proyecto de gráficos`: Implementa un manager con el que acceder a la configuración de renderizado de SDL, como la ventana de juego o la cámara.
- `Proyecto de recursos`: Implementa un manager para el guardado de recursos como imágenes o fuentes de texto.
- `Proyecto de sonido`: Implementa un manager para cargar, reproducir o detener efectos de sonido o música.
- `Proyecto de utilidades`: Implementa varias clases genéricas útiles para el resto de proyectos.
- `Proyecto de input`: Implementa un manager para manejar el input del usuario, tanto de teclado y ratón como de mando, con soporte para Dualshock 4.
- `Proyecto de Entity-Component-System`: Implementa el ECS y contiene todos los componentes proporcionados por el motor.
- `Proyecto de Consola`: Implementa funcionalidad para imprimir información por consola. Útil para el desarrollo y la consola de debug del editor.

- `Proyecto de Scripting`: Implementa funcionalidad leer y crear clases en C++. Explicada en detalle posteriormente.

- El Editor consiste en 2 proyectos de Visual Studio:

- `Proyecto de componentes`: Este proyecto tiene la responsabilidad de gestionar la adquisición y procesamiento de scripts y componentes procedentes del motor subyacente. Estos elementos son preparados para su utilización posterior en el editor.

- `Proyecto de ImGUI`: En este proyecto se encuentra toda la funcionalidad relativa al editor, haciendo uso de la biblioteca ImGUI. Aquí se engloba la creación y manejo de ventanas, cuadros de diálogo emergentes, menús desplegables, renderización de la escena en edición, así como la gestión de archivos, entre otras características.

- `Proyecto main`: Este proyecto ejecuta el bucle principal del editor y genera el ejecutable.

Scripting...

- La integración entre el editor y el motor se materializa mediante un flujo de intercambio de datos. Inicialmente, el motor lleva a cabo la serialización de los componentes disponibles en un formato JSON. Posteriormente, el editor se encarga de leer y procesar estos componentes serializados, permitiendo su asignación a las diferentes entidades en desarrollo. Una vez que el proceso de desarrollo del videojuego ha concluido, el editor toma la iniciativa de serializar la escena completa. Esto abarca las entidades presentes en la escena, junto con sus respectivos componentes y scripts asociados. Con la escena debidamente serializada, el editor ejecuta el archivo ejecutable (.exe) al motor. En este punto, es el motor el que asume la responsabilidad de interpretar y cargar las escenas previamente serializadas por el editor.

Este enfoque de intercambio de datos entre el editor y el motor permite una sincronización efectiva de la información esencial para la construcción y ejecución del videojuego, al tiempo que mantiene una clara separación de las tareas y roles entre las dos entidades.

Fase de prueba con usuarios...

Capítulo 2

Introduction

2.1. Motivation

Regarding game engines during our degree, in the first year, we learned the basics of Unity, in the second year, we developed simple 2D games with SDL using an abstraction layer to facilitate learning, and in the third year, we had to step out of our comfort zone and learn what's behind the tools provided to us during the degree. This led us to develop a 3D game engine using OGRE as the graphics engine, BulletPhysics as the physics engine, FMOD as the sound library, and Lua as the scripting language.

Through this experience, we realized several things. It's more convenient to work with an engine that has an integrated editor rather than accessing it directly through programming. Additionally, scripting should be comfortable and intuitive since it's the task that will take up the most time for the developer.

In the fourth year, with the recent experience of developing a game engine, we decided that the proposal for our final degree project would be an engine with the following characteristics:

- `Self-sufficient`: This means it provides functionality to manage resources, create scenes and objects from the editor, and can generate final game executables for distribution. Obviously, it doesn't have all the necessary functionality to be completely independent of any external tools during development, but it does provide the basics for the development cycle of a 2D game.

- `Integrated Editor`: The main tool of a self-sufficient engine is its editor, responsible for communicating the developer's actions to the engine. We had no experience in developing desktop applications, as we didn't create an editor for the third-year engine, so we knew it could be a challenge..

- `Visual Node-Based Programming`: This is the highlight of our engine. Due to the complexity that some modern engines present for beginner or inexperienced developers, visual programming is a very useful and intuitive tool for creating logic and behavior in the game. We knew it could be a technical challenge, but it would provide great convenience, fluidity, and, of course, open the doors of our engine to many developers with little programming experience.

Finally, we chose to focus on 2D due to the experience gained during our studies. Additionally, it involves less technical complexity during the development of the engine.

2.2. Goals

The main objective of the final degree project is to develop a self-sufficient 2D game engine with an integrated editor and node-based visual programming. With this, users will have a tool to develop any type of 2D video game.

By achieving self-sufficiency in the engine, we aim to enable users to develop their video games with minimal reliance on external tools, ensuring that all the work can be done within the editor.

With the editor, our goal is to make the development process comfortable, smooth, and visually-oriented, eliminating the need for developers to communicate with the engine through programming.

With the node-based visual programming system, we aim to open the doors of our engine to inexperienced developers or those with limited programming knowledge, allowing them to develop their videogames. In addition, it represents a technical challenge that will provide us with experience and knowledge.

2.3. Project Management

To start with, Git has been used as the version control system through the GitHub Desktop application. All the implemented code has been uploaded to a repository, dividing the work into branches. Specifically, as outlined in the work plan, the project has been divided into engine, editor, and visual programming.

For the development of the engine, the SDL (Simple DirectMedia Layer) library has been used. In particular, SDL Image has been used for graphics, SDL Mixer for audio, and SDL TTF for TrueType font rendering. The Box2D library has also been used for 2D physics simulation and collision handling.

For the development of the editor, the ImGUI (Immediate Mode Graphical User Interface) library has been used.

The code has been developed in the Visual Studio 2022 Integrated Development Environment (IDE) and written in C++. Finally, for task organization, we have used the Trello project management tool.

2.4. Work Plan

The work is divided into three parts: the engine development, the editor development, and the development of the node-base visual scripting system.

- The engine consists of several Visual Studio projects where its fundamental parts are divided:

- `Main Project`: This project implements the main loop of the engine and generates the game executable.
- `Physics Project`: Implements a manager to access the physical world's configuration.
- `Graphics Project`: Implements a manager to access SDL rendering configuration, such as the game window or the camera.
- `Resources Project`: Implements a manager to handle resources such as images or text fonts.
- `Sound Project`: Implements a manager to load, play, or stop sound effects or music.
- `Utilities Project`: Implements various generic classes useful for the rest of the projects.
- `Input Project`: Implements a manager to handle user input, including keyboard, mouse, and controller input, with support for Dualshock 4.
- `Entity-Component-System Project`: Implements the ECS and contains all the components provided by the engine.
- `Console Project`: Implements functionality to print information through the console, useful for development and debugging within the editor.
- `Scripting Project`: Implements functionality to read and create C++ classes. Detailed explanation follows later.

- The Editor consists of 2 Visual Studio projects:

- `Componentes project`: This project is responsible for managing the acquisition and processing of scripts and components from the underlying engine.
- `ImGUI project`: In this project you will find all the functionality related to the editor, making use of the ImGUI library. This includes the creation and management of windows, pop-up dialog boxes, dropdown menus, rendering of the editing scene, as well as file management, among other features. as well as file management, among other features.
- `main project`: This project implements the main loop of the editor and generates the executable.

Scripting...

- The integration between the editor and the engine takes the form of a data exchange flow. Initially, the engine performs the serialization of the available components in a JSON format.components available in a JSON format. Subsequently, the editor is in charge of reading and processing these serialized components, allowing their assignment to the different entities under development. different entities under development. Once the video game development process has been completed, the editor takes the initiative to serialize the entire scene. This encompasses the entities present in the scene, along with their respective components and associated scripts. With the scene duly serialized, the editor runs the executable file (.exe) to the engine. At this point, it is the engine that assumes responsibility for interpreting and loading the scenes previously serialized by the editor.

This approach to data exchange between the editor and the engine allows for effective synchronization of the information essential for the construction and execution of the while maintaining a clear separation of tasks and roles between the two entities.

Fase de prueba con usuarios...

En el próximo capítulo...

En capítulo *2 Estado del Arte* se explica qué son los motores de videojuegos 2D y cuáles hemos tomado como referencia para desarrollar el nuestro. Asimismo, se expondrán las librerías que hemos decidido emplear en nuestro proyecto y se justificará la elección de cada una. Además, se menciona la diferencia entre el desarrollo de un videojuego a través de scripting y programación. Por último, se habla sobre el papel que juegan los motores en el mercado actual de los videojuegos.

Capítulo 3

Estado del arte

3.1. Motores de videojuegos 2D

Explicar qué es un motor de videojuegos 2D.

3.2. Referencias de motores de videojuegos 2D

Hablar sobre los motores que hemos tomado como referencia.

3.3. Editores en motores de videojuegos

Explicar qué es un editor en el ámbito de motores de videojuegos y qué importancia tienen para el usuario.

3.4. Scripting vs programación

Explicar la diferencia entre el desarrollo de un videojuego a través de scripting por nodos y a través de programación. Hablar también sobre los sistemas de scripting investigados y explicar sus principales características.

3.5. Librerias utilizadas

Librearias utilizadas y por que

3.6. Mercado de los motores de videojuegos 2D

¿Una sección para el mercado?

Notas bibliográficas

Referencias a los motores 2D y sistemas de scripting investigados. Asi como documentacione y repositorios utilizados.

Capítulo 4

Contribuciones

4.1. Contribuciones de Pablo

Contribuciones de Pablo Fernández Álvarez

4.2. Contribuciones de Yojhan

Contribuciones de Yojhan García Peña

4.3. Contribuciones de Iván

Contribuciones de Iván Sánchez Míguez

Parte I

Apéndices

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes