Motor de Videojuegos: “MKengine”

Mikel Iturrlde Salguero

Motores Gráficos y plugins: Diseño y desarrollo de videojuegos

Descripción general

Este proyecto consiste en la realización de un motor de videojuegos en lenguaje C++, utilizando como apoyo diversas librerías de renderizado de gráficos.

Contexto

El desarrollo de este proyecto se da en el contexto académico, con un profesor que ha participado activamente en la resolución de algunos problemas, tanto de falta de conocimiento como de mala arquitectura. También se ha apoyado a nivel de código en ejemplos obtenidos del repositorio público: "https://bitbucket.org/angel-esne/".

Objetivos

El desarrollo de este proyecto cumple con los siguientes objetivos:

- La escena consta de un espacio cerrado entre cuatro paredes.
- En el centro de la escena hay un personaje cuyo movimiento puede ser controlado por el usuario y también cuatro objetos dispuestos cerca de las cuatro esquinas de la escena.
- El personaje no puede abandonar el espacio cerrado aunque el usuario lo intente.
- Los objetos se mueven hacia el personaje siguiéndole allá donde esté, pero a una velocidad inferior, de modo que el personaje puede evitar que le alcancen.
- Si alguno de los objetos termina tocando al personaje controlado por el usuario, todo se restablece a su posición inicial y se vuelve a empezar.

Descripción técnica

El motor se basa en una arquitectura ECS y se estructura en escenas. Cada escena alberga sus propios sistemas independientes y son gestionados por un controlador que utiliza un patrón singleton.

Sistema ECS

El patrón ECS (Entity Component System) se basa en clases principales que vertebran las jerarquías dentro del proyecto.

Entidad

Una entidad es un objeto que contiene componentes y que realiza los comportamientos propios de los mismos. Otros motores comerciales como Unity son los denominados GameObjects.

En el caso de este motor todas las entidades pertenecen a la clase Entity. Pueden albergar cualquier número de componentes, pero siempre deben contar al menos con un componente del tipo Transform. Este contiene la información de su posición, rotación y escala dentro del espacio tridimensional de la escena.
Las entidades se incluyen dentro de una escena y poseen un identificador único.

Componente

Los componentes se añaden a las entidades para dotarlas de funcionalidades. Todos los componentes se heredan de la clase abstracta Component.
En este motor existen tres tipos de componentes básicos, cada uno dependiente de su propio sistema.

Transform

El componente transform es el único que no pertenece a ningún sistema. Este se crea de forma automática siempre que se crea una entidad y contiene la información relativa a la posición, rotación y escala de la entidad a la que pertenece.

El componente transform también es el encargado de establecer las jerarquías de pertenencia de entidades. Las entidades pueden anidar unas dentro de otras agregando sus transformaciones unas a otras. Es así cómo pueden crearse objetos complejos con movimientos agrupados.

El componente Transform permite acceder a los valores de forma local o global. Al hacerlo de forma global se suman todos los valores de sus padres.
También permite aplicar transformaciones sencillas como trasladar, cambiar la escala o orientar un objeto en un ángulo determinado.

Renderizado

El Render Component guarda la información necesaria para que la entidad interactúe con el sistema de renderizado de la escena. Dependiendo del tipo de componente interactúa de forma diferente.Los diferentes tipos de componentes de renderizado heredan el componente abstracto básico.
Todos los componentes de renderizado deben poseer un identificador único a la hora de crearse y añadirse al sistema de renderizado.

Cámara

Un componente de renderizado de tipo cámara indica al sistema desde donde ha de renderizar la escena, así como todas las variables propias de la cámara (planos de renderizado, campo de visión, proporciones…).
Una entidad con este componente se comportará como una cámara dentro de la escena.

Luz

Un componente de luz se encarga de dar la información de iluminación a la escena, necesaria para que ésta se renderice correctamente.
Una entidad con este componente se comportará como un punto de luz dentro de la escena.

Malla

Las entidades con este tipo de componente son los objetos tridimensionales que se renderizan en la ventana de la escena. Para crear este componente a parte del identificador único hay que especificar una ruta relativa donde se encuentra el modelo que contiene la

información para crear esta maya. En este proyecto se utilizan archivos del tipo .obj para crear estos componentes en la escena.

Controlador

Los componentes de este tipo simulan un sistema de scripting. Otorgando comportamientos varios a las entidades que los poseen. Estos heredan de una clase abstracta base, que les incluye las funciones de start y awake.

En este proyecto concreto se utilizan para implementar los comportamientos del jugador y los enemigos.

Sistema

Un sistema dentro de este proyecto se encarga de gestionar todos los componentes pertenecientes a este. Los crea, los almacena y ejecuta funciones basándose en sus datos. Los sistemas son en esencia tipos de tareas como por ejemplo renderizar los gráficos. Es por esta razón que los sistemas tienen prioridades para ordenarse dentro del kernel y poder ejecutarse en el orden correcto.

Los sistemas a su vez son propios de una escena y se encargan de los componentes de las entidades de la misma.

Renderizado

El sistema de renderizado se encarga de renderizar los gráficos utilizando la biblioteca SDL2, para funcionar correctamente necesita al menos un componente del tipo cámara y otro del tipo luz. Este sistema necesita una referencia a una ventana donde poder renderizar los gráficos.

Controlador

El sistema de control se encarga de ejecutar las funcionalidades de los componentes de su sistema. Al ser un sistema encargado de las funciones varias del sistema de scripting tan solo ejecuta las funciones de start y update.

En el ejemplo del motor los componentes de control que se utilizan son los encargados de controlar al jugador y a los enemigos.

Input

El sistema de input es el único presente en el motor que no utiliza componentes. El diseño de este sistema es un patrón singleton. Registra los inputs del jugador y los actualiza en variables que pueden ser utilizadas por el resto de sistemas del motor, especialmente el de control y sus componentes.
En el ejemplo del motor se utilizan dos ejes, el horizontal y el vertical que se controlan pulsando las teclas w,a,s,d. Este sistema se inspira en el sistema de InputManager de Unity que también utiliza este sistema de ejes.

Escenas

Una escena es el conjunto de entidades con sus componentes, los sistemas que ejecutan la funcionalidad de los componentes y el kernel que gestiona los sistemas. El motor organiza el flujo por escenas y estas se parecen bastante a la estructura de Unity.

Para el ejemplo implementado en la entrega, la función main solamente crea la ventana y una escena default. Esta gestiona la creación de todos los sistemas que necesita para funcionar correctamente.

Kernel

El kernel es el que se encarga de organizar los sistemas por prioridad. Estos al ser tareas cuentan con una variable que indica el orden en el que deben ejecutarse. Esta es utilizada por el kernel para ordenarlos y ejecutarlos cíclicamente hasta que se le ordene parar la ejecución.

Manager

El Scene Manager se encarga de gestionar el flujo dentro de las aplicaciones desarrolladas con este motor. Puede cargar escenas, reiniciar escenas y cerrar el programa. Para que una escena puede ser utilizada por el manager antes ha debido de crearse.

Diario de desarrollo

Este proyecto se trata del más grande que he desarrollado en el lenguaje de programación C++. La experiencia previa utilizando motores de videojuegos, especialmente Unity, han inspirado en gran parte la arquitectura de este motor. La asistencia por parte del profesor ha supuesto una increíble ayuda a la hora de resolver conflictos durante el desarrollo y diseñar algunos de los patrones de diseño que están presentes en el motor.

Implementación de librerías

La primera parte del desarrollo consistió en la creación de un entorno donde estuvieran bien implementadas todas las librerías necesarias. Esto puede parecer trivial a simple vista pero no lo fue en absoluto. Esta fue una de las partes más conflictivas del desarrollo ya que los problemas de enlace de las librerías fueron muy frecuentes y sin ayuda no habríamos podido resolverlos.

Por suerte estos problemas han conseguido que este esencial conocimiento esté mucho más asentado y es probable que en futuros desarrollos esta parte quede como algo anecdótico.

Creación de sistemas

La siguiente parte del desarrollo consistió en el diseño de los sistemas que iban a conformar el motor. Para esto la mayor inspiración fue el motor con el que más he trabajado en el pasado, Unity.

Durante esta fase del desarrollo también se implementó el sistema de creación de ventanas, así como una primera versión de lo que se convertiría en el sistema de renderizado. Con esto validamos la creación de gráficos tridimensionales y comenzamos a estructurar los diseños y la arquitectura final del motor.

Implementación de sistemas

La última parte del desarrollo fue la implementación del motor tal y como se diseñó anteriormente. También se lidio con los distintos errores y con algún cambio que se implementó a raíz de ver cómo se comportaba el motor.

Reflexiones

Este proyecto ha significado un reto importante, pero creo que se ha conseguido un resultado bastante satisfactorio. Se han cumplido gran parte de los objetivos propuestos y el escenario demostrativo es perfectamente funcional.

Posibles mejoras

A nivel de mejoras ha habido funcionalidades que no han sido posibles implementarlas por falta de tiempo. Algunas de estas pueden ser:

- Sistema de mensajería.
- Cargado de escenas mediante archivos externos xml.
- Separación del proyecto en dos partes independientes, una con la funcionalidad base del motor y otra con el ejemplo implementado.

También podría haberse hecho un mejor trabajo en cuanto a los comentarios del código. Y también una mejor estructuración de las carpetas.

Conocimientos adquiridos

El desarrollo de esta práctica ha asentado mucho los conocimientos de C++, tanto conocimientos generales, como específicos de las librerías implementadas.

También se han adquirido conocimientos básicos sobre arquitectura de motores, estos son de gran ayuda no solo para desarrollar un motor propio como en el caso de esta práctica, si no para entender mejor cómo funcionan los motores comerciales de uso común.

Por último se ha profundizado bastante en el patrón de diseño ECS, el cual ha permitido estructurar toda la arquitectura del motor de una manera muy efectiva.