Segunda práctica: “MKatapulta”

Mikel Iturrlde Salguero

Animación 3D: Diseño y desarrollo de videojuegos

Descripción general

Este proyecto consiste en desarrollar un nivel 3D, donde se deben cumplir una serie de condiciones de jugabilidad. En este nivel se controla una catapulta sobre una plataforma flotante. Se deberá recoger una llave que abre una puerta en un muro y activa una plataforma. Con esto se podrá acceder a la siguiente isla. Donde se podrá disparar a una torra presente en la tercera isla terminando aquí el nivel.

Contexto

El desarrollo de este proyecto se da en el contexto académico. Se ha utilizado como apoyo el código presente en el repositorio público: "https://bitbucket.org/angel-esne/". Esta práctica se ha desarrollado sobre el motor creado anteriormente con el objetivo de facilitar la ejecución y ahorrar trabajo. Para el desarrollo de los comportamientos físicos de este proyecto se podían utilizar bibliotecas de ayuda. No obstante, con el propósito de marcar una característica diferencial, así como de profundizar en el aprendizaje de la implementación de físicas sencillas, se ha optado por la implementación de un sencillo sistema de físicas simples.

Objetivos

El desarrollo de este proyecto cumple con los siguientes objetivos:

- Una catapulta con ruedas que se controla con el teclado (u otro dispositivo de entrada). La catapulta debe poder avanzar, retroceder y girar hacia los lados. Tiene un brazo que dispara proyectiles esféricos girando sobre su eje.
- Un bloque vertical que se debe poder abatir cuando recibe un golpe.
- Un muro con una puerta cerrada que separa la catapulta del bloque por la mitad de la escena: el bloque está a un lado y la catapulta en el lado contrario. El muro tiene una altura tal que la catapulta no puede disparar proyectiles por encima de él.
- Un ítem que representa una llave ubicada en algún lugar y que la catapulta puede coger cuando pasa a través de él. Cuando se coge la llave, la puerta del muro se abre.
- Un agujero en el suelo, entre el muro y la catapulta, que esta no puede cruzar.
- Una plataforma que se desplaza horizontalmente y que permite llevar la catapulta hacia el muro cuando la catapulta se sube a ella (la plataforma se mueve automáticamente cuando la catapulta está encima).
- Un agujero en el suelo que separa la catapulta del bloque cuando esta pasa al otro lado del muro y que no se puede cruzar de ninguna manera.

Descripción técnica

Este proyecto se ha desarrollado sobre el motor propio creado en una práctica anterior. Sobre éste se han creado componentes de control nuevos necesarios para cumplir con la funcionalidad de la entrega.

Motor

La descripción técnica del motor se ha desarrollado en más profundidad en su respectiva memoria. Al ser una parte tan importante de esta práctica se ha adjuntado un anexo al final de este documento donde se explica nuevamente la descripción del motor.

Componentes de Control

Para desarrollar la funcionalidad requerida en la escena propuesta por la práctica han sido necesarios varios componentes con funcionalidades diversas.

Catapulta

Este es el componente principal de la práctica. Gestiona por un lado el movimiento de la catapulta, y por otro lado la mecánica de poder disparar.

El movimiento de la catapulta no está implementado con motores sino que ha sido simplificado por falta de tiempo. También cabe destacar que la catapulta en su totalidad responde a las colisiones con el suelo solo en su eje de coordenadas, por lo que ésta caerá solo si este sale fuera de las plataformas. Si esto sucede la catapulta reaparece en su posición inicial.

Para crear esta clase es necesario tener una referencia al brazo de la catapulta para animarlo al disparar, así como a la bala para activar su comportamiento de físicas.

Suelo

Este comportamiento forma parte de todas las plataformas. Tiene el propósito de simular las físicas de un collider cuadrado. Cuando un objeto marcado como referencia entra colisiona con el este lo sitúa en su parte superior impidiendo que sea traspasado.

En la escena, tanto la catapulta como la bala son objetos que interactuan con todos los elementos con este componente.

Puerta

La puerta es en esencia un muro gigante, el cual tiene un comportamiento que le permite generar un hueco en su mitad de un tamaño variable que actúa como puerta. Este comportamiento se puede activar de forma externa cuando sea necesario.

Plataforma

La plataforma, aparte de contar con el componente de suelo, también tiene la funcionalidad de poder detectar cuando la catapulta está sobre ella y moverse entre dos posiciones llevando consigo la catapulta.

Tiene varios estados, realiza transiciones de uno a otro según ciertas condiciones:

- Reposo. Estado inicial en el que la plataforma no se mueve ni se activa.
- Preparada. La plataforma está, o se dirige, a la primera isla y espera a la catapulta.
- Activa. La plataforma está, o se dirige, a la segunda isla trasladando sobre ella a la catapulta.

Llave

La llave es un objeto de la escena que activa comportamientos de otros al detectar la la catapulta acercarse a su posición. También tiene un pequeño comportamiento estético para diferenciarla dentro de la escena como un objeto importante.

Bala

La bala tiene la funcionalidad necesaria para realizar un movimiento parabólico al ser disparada. También puede regresar a su estado original al caerse por la escena y así poder ser disparada de nuevo por la catapulta.

Pared

La pared genera el comportamiento de rebote con el proyectil invirtiendo su dirección en el eje de colisión. Está presente en el muro y en el bloque final del ejercicio.

Bloque

El bloque contiene la lógica para simular el comportamiento de impacto y posterior caída al vacío producido por la colisión del proyectil. Calcula la fuerza de la colisión basándose en el punto de contacto y con esto genera una velocidad angular que junto con la gravedad generan la caída.

Diario de desarrollo

Al apoyarse sobre el motor, el desarrollo de este proyecto parecía de menor envergadura. No obstante la cantidad de contenido de la escena. Así como la inclusión de un sistema propio de físicas simplificadas ha generado casi tanto trabajo como el desarrollo del propio motor.

Sistema de físicas propio

La decisión de no utilizar una librería de físicas nace de la intención por marcar un factor diferencial frente a otras prácticas. Aun sabiendo que supondría más trabajo y que el resultado sería más limitado, se tomó esta decisión. Esto se debe a la curiosidad y las ganas de experimentar con la idea de generar unas físicas propias.

Diseño de componentes

El diseño de los componentes necesarios para esta escena fue bastante orgánico. La propia escena iba pidiendo nuevas funcionalidades y esto fue generando los distintos comportamientos y encapsularlos en nuevos componentes.

Desarrollo de componentes

El trabajo de desarrollo de componentes fue más metódico que complicado. Al tener el diseño ya establecido simplemente fue necesario crear los ficheros y rellenarlos con las funciones ya pensadas. Ninguna de estas funciones por separado son de elevada complejidad por lo que no supusieron un problema a la hora de implementarlas..

Reflexiones

Este proyecto ha servido para reflexionar no solo en el desarrollo de esta práctica sino también en la del motor. Ya que al utilizarlo como herramienta ha salido a la luz debilidades y fortalezas de este que podrían ser mejoradas en el futuro.

Posibles mejoras

Este desarrollo deja dos grandes mejoras pendientes de realizarse.

El sistema de control.

Este debe ser más escalable si se pretende hacer del motor algo viable para proyectos grandes. Incorporar un sistema de scripting probablemente sería la mejor opción. Ya que crear una función nueva cada vez que se crea un comportamiento es una práctica muy poco escalable.

El sistema de físicas

Este sistema aunque muy funcional no es generalista. Ha sido desarrollado con un propósito muy concreto y si se pretende usar en otros proyecto sería necesario replantearse la estructura. Probablemente habría que crear un sistema dentro del motor y generar componentes genéricos de físicas que utilicen los comportamientos ya desarrollados.

Conocimientos adquiridos

Este proyecto ha permitido asentar enormemente los conocimientos previos aprendidos sobre el motor. Así como, gracias a la implementación de físicas, adquirir muchos conocimientos sobre este ámbito y cómo aplicarlos a un entorno de programación.

Anexo 1: Descripción técnica del motor

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.