Motores Gráficos y Plugins

Práctica final

Documentación del motor

Silvia Osoro García
GDDV 4.3

Descripción de los componentes del engine:

- **Kernel**: es el encargado de ejecutar todas las tareas. Tiene tareas consumibles (se ejecutan solo una vez) y otras que se estarán ejecutando hasta que finalicemos el programa.

- **Audio Manager**: esta clase es la encargada de inicializar el SDL2 y el subsistema de audio. Además contiene unas funciones para reproducir audio o música.

- **Render System**: esta clase se encarga de dibujar los objetos que hay en la escena. Renderiza los componentes de model, camera y light. Esta task actualiza las posiciones de los entities que hay en escena. Desde aquí decidimos qué cámara renderizamos, y si ponemos invisibles algún objeto.

- **Camera Component**: esta clase sirve para añadir una cámara a un entity.

- **Component**: clase base de todos los componentes.

- **Entity**: esta clase hace de contenedor de componentes para definir su comportamiento.

- **Light Component**: esta clase sirve para añadir una luz a un entity.

- **Model Component**: esta clase sirve para añadir un modelo a un entity.

- **Scene**: en esta clase añadimos las task no consumibles y les asignamos un número de prioridad (orden en el que se ejecutarán); creamos los entities; los componentes; los asignamos; colocamos, rotamos y escalamos los entities; añadimos los nodos a la escena y guardamos los entities en nuestro mapa para que puedan ser usados desde fuera de la escena. También añadimos los controladores.

- **Task**: clase base de todas las task. Obliga a elegir si son consumibles o no y a definir qué hará cada task.

- **Timer**: esta clase nos sirve como un temporizador. Devuelve segundos.

- **Transform Component**: esta clase es un componente que sirve para añadir una posición, rotación y escala a los entities.

- **Message**: esta clase contiene los datos de los mensajes (id y data).

- **Messenger**: esta clase sirve para enviar los mensajes a los observers que estén suscritos a un id del que quieran recibir su data.

- **Observer**: clase base, en la cual cuando se sobreescriba su función, servirá para recibir e interpretar los mensajes. Se envía el mensaje al observer desde la clase Messenger.

- **Controller**: clase base en la cual en su función update estarán las acciones y comportamiento del controlador.

- **Controller Component**: esta clase sirve para asignarle a un entity un controller.

- **Update Task**: esta clase ejecuta el run (el código que repetiremos en cada bucle) de todos los entities que tengan el componente Controller_Component.

- **Input Mapper**: en esta clase recibimos los mensajes del Input Task y se los pasa al Messenger.

- **Input Task**: en esta clase recogemos los tipos de mensajes de teclado mandamos al Input Mapper para luego ser interpretados por los observers suscritos al id del mensaje enviado.

- **Destroy Task**: esta clase es una task consumible que necesitamos para retirar aquellos objetos que, en tiempo de ejecución, queremos que dejen de ser renderizados (que desaparezcan). Necesitamos que haga esto bien antes de la siguiente ejecución del render. Solo lo llamamos cuando queremos que desaparezcan los números cuando el conejo los toca.

Descripción de los componentes de la demo:

- **Player Controller**: esta clase lleva la lógica del jugador. Recibe los mensajes de teclado y los interpreta.

- **Numbers Controller**: esta clase lleva la lógica de los números. Detecta si el jugador está encima suya: si cogió los anteriores números con id menor que el suyo, éste dejará de renderizarse y aumentará el número de números que lleva el jugador recogidos; si no, sonará un sonido para advertir de que ese número aún no tiene que ser cogido antes que otros.

- **Limits Controller**: esta clase lleva la lógica de los límites. Detectan si el jugador está queriendo sobrepasar ese límite (se dividen entre arriba, abajo, derecha e izquierda) y lo atrasa a la velocidad que lleva el personaje. Una vez que se coge todos los números, desaparecen.