# Estructura Básica: Motor Gráfico

Adrián Ponce Balseiro.
G 4.3 DDVJ
adrianpb95@gmail.com

Esta práctica está dividida en dos proyectos:

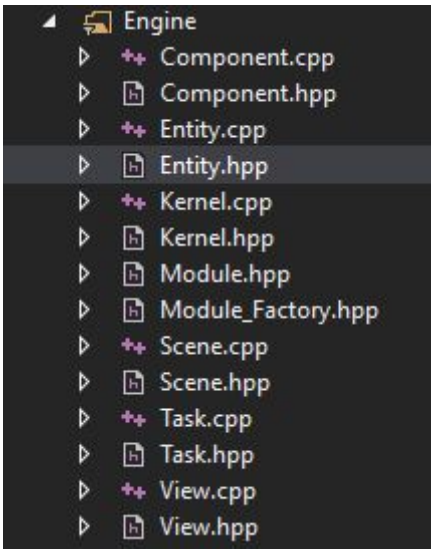
- Demo → Crea la escena y la ejecuta. (main)
- Motor → Gestiona todos los elementos y tareas que participan en esta escena.

## *Motor*

El motor a su vez está dividido en diferentes partes:

- **Engine:**

Es la base del motor. Son los elementos que se encargan de las tareas más básicas, sobre lo que se construye el resto.



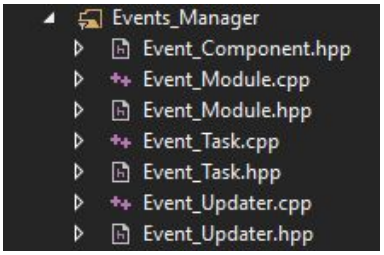

1. **Scene** → Creada desde la demo, posee una ventana (View) donde mostrar por pantalla todos los elementos del motor, y además inicializa el Kernel de tareas.
2. **View** → Renderiza las mallas por pantalla.
3. **Kernel** → La base del funcionamiento del motor. Se encarga de ejecutar las tareas (Task) activas según su prioridad.
4. **Task** → Clase padre virtual, que posee una prioridad de ejecución y trabaja con Entities y Components.
5. **Components y Entities** → Los componentes son los elementos básicos de trabajo que poseen información. Las entidades por su lado, están formadas por un conjunto

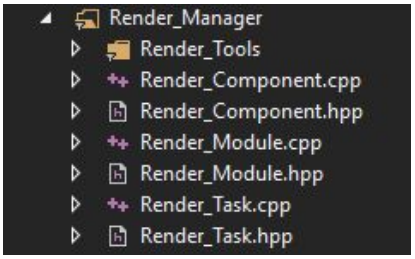
de componentes. Dependiendo del tipo de componente creado, hay que crear un Module acorde para poder trabajar con ellos y tratar su información (Por ello Component, también es una clase padre virtual).

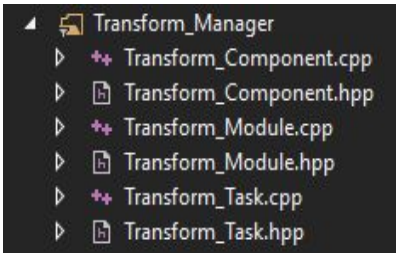
6. **Module** → Gestor de Task y sus componentes, encargado de realizar diversas tareas dependiendo de su tipo.

- **Managers:**

Elementos ya especializados en realizar diversas tareas.




Como mencionaba antes, las clases Component, Module y Task son prácticamente virtuales, por lo tanto en esta parte del motor vamos a crear hijas de estas para poder desempeñar diferentes tareas:

- **Event Manager** → Se encarga de los eventos de ventana (close y resize) y de los eventos de teclado (inputs del jugador). La clase Event Updater, mueve al jugador y actualiza la misión.
- **Render Manager** → Se encarga de limpiar la pantalla y renderizar los elementos de forma adecuada. Posee un Render Tools, que son una serie de clases creadas por mi para realizar la práctica de OpenGL (Renderizado de mallas).
- **Transform Manager** → Se encarga de tratar las posiciones y movimientos de cada elemento de la escena.

Por último, he creado un sistema de mensajería basado en Dispatchers y Listeners, capaces de enviar mensajes con información de un lado a otro del motor, de tal forma que podemos realizar acciones, como el movimiento del jugador, a través del **Message Manager**.