Documentación Proyecto de Motor de Videojuegos

Visión General

El proyecto del motor de juego presentado es una implementación basada en el patrón Entity-Component-System (ECS), buscando y usando de referencia en aspectos de estructura al motor Unity, que proporciona una estructura modular y eficiente para el desarrollo de juegos en 3D.

Este motor ofrece una arquitectura flexible y extensible, donde las entidades representan los objetos del juego, los componentes definen su comportamiento y atributos, y los sistemas gestionan la lógica de renderizado, entrada del usuario y actualización de la lógica del juego.

El diseño modular del motor permite una fácil integración de nuevos componentes y sistemas, facilitando la creación de juegos complejos y permitiendo una gestión eficiente del ciclo de vida del juego. Con una combinación de clases y sistemas bien organizados, este proyecto proporciona una base sólida para el desarrollo de una amplia variedad de juegos en 3D.

Veamos las clases que conforman el proyecto para ver cómo funciona

1. Clase Entity

   - La clase Entity representa un objeto básico del juego que puede tener múltiples componentes asociados. Una entidad se identifica mediante un ID único y puede contener diferentes componentes que definen su comportamiento y atributos. En el constructor, se inicializan las entidades y se les asigna un Transform por defecto. Los métodos principales incluyen Add_Component para añadir componentes a la entidad, Get_Component para recuperar componentes específicos (comentado en el código), y métodos para obtener y establecer el ID y el estado activo de la entidad. La clase también maneja la relación de transformación entre la entidad y su posible entidad padre.

2. Clase Transform

   - La clase Transform se utiliza para gestionar la posición, rotación y escala de una entidad en el espacio 3D. Esta clase contiene métodos para obtener y establecer estas propiedades, así como para aplicar transformaciones como la traducción y la

rotación. El método Get_Matrix devuelve la matriz de transformación compuesta que representa la posición, rotación y escala de la entidad, considerando también las transformaciones de cualquier entidad padre.

3. Clase Component y Subclases

- Los componentes definen comportamientos específicos que pueden ser añadidos a las entidades. Algunas de las subclases importantes son:
    - CameraComponent gestiona la configuración de la cámara para la renderización.
    - LightComponent define una fuente de luz en la escena.
    - MeshComponent maneja la geometría y la apariencia visual de la entidad mediante la asignación de modelos 3D.
    - PlayerController gestiona la lógica de movimiento del jugador, incluyendo la velocidad y la dirección.
    - ReciverComponet actúa como un receptor de eventos de entrada, permitiendo que las entidades respondan a las acciones del usuario.
    - PlayerReader recibe eventos de entrada y ajusta las propiedades de PlayerController en consecuencia, facilitando el control del jugador.
    - BoundingBoxComponent define un volumen de colisión para la entidad, utilizado en la detección de colisiones.
    - Cada uno de estos componentes se asocia a una entidad y se integra con el sistema de renderizado y otros sistemas para ser visualizados y actualizados en la escena.

4. Clase RenderSystem

- El RenderSystem es responsable de gestionar todos los aspectos de la renderización en el motor de juego. Esta clase inicializa la ventana de renderizado y mantiene una lista de componentes de renderizado que deben ser dibujados cada frame. Los métodos clave incluyen Run, que se ejecuta en un bucle continuo para renderizar la escena, y métodos para añadir componentes de renderizado, como CreateAndAddMeshToRender, CreateAndAddCameraToRender y

CreateAndAddLightToRender. El sistema actualiza las transformaciones de los componentes antes de renderizarlos y maneja el intercambio de buffers para actualizar la ventana.

5. Clase InputSystem

- El InputSystem se encarga de manejar los eventos de entrada del usuario, como las pulsaciones de teclas y los eventos del ratón. Utiliza la biblioteca SDL para capturar eventos y los distribuye a los receptores de entrada registrados, que son instancias de ReciverComponet. El método principal Run procesa la cola de eventos y llama al método Reciver de cada receptor de entrada para manejar eventos específicos.

6. Clase UpdateSystem

- El UpdateSystem coordina la lógica de actualización de todas las entidades en el juego. Mantiene una lista de componentes actualizables, como PlayerController, y llama a sus métodos Update en cada ciclo de juego para aplicar la lógica de actualización. Este sistema asegura que todas las entidades respondan de manera coherente a las entradas del usuario y otros eventos del juego.

7. Clase Kernel

- La clase Kernel es el núcleo del motor que coordina la inicialización, ejecución y finalización de todos los sistemas registrados. Mantiene una lista de tareas (Task) que representan los diferentes sistemas (renderizado, entrada, actualización) y gestiona su ciclo de vida. El método Run mantiene el bucle principal del juego, llamando a Run en cada sistema en cada iteración del bucle.

8. Clase Scene y SceneManager

- Scene representa una colección de entidades y sistemas que componen una escena del juego. Incluye sistemas específicos para la escena, como RenderSystem, InputSystem y UpdateSystem. La clase SceneManager maneja la creación, activación y desactivación de escenas. También facilita la transición entre diferentes escenas del juego y gestiona la ventana principal del motor de demostración.

9. Clase Window

- La clase Window crea y gestiona la ventana de renderizado del juego utilizando SDL y OpenGL. Inicializa el contexto de OpenGL, maneja la limpieza y el intercambio de buffers, y proporciona métodos para obtener las dimensiones de la ventana. Esta clase es fundamental para la configuración del entorno de renderizado y la presentación visual del juego.

10. Clase Task

- La clase Task es una clase abstracta que define una interfaz para las tareas que pueden ser ejecutadas dentro del Kernel. Cada tarea debe implementar el método Run, que contiene la lógica principal que se ejecutará en cada iteración del bucle de juego. Las subclases de Task incluyen sistemas como RenderSystem, InputSystem y UpdateSystem, que heredan de Task y proporcionan su implementación específica del método Run.