Abril 2019

Clases implementadas y métodos

**class Box2D\_WorldManager:**

Clase encargada de la gestión del mundo, desde la creación e instanciado de todos los componentes del mundo, hasta el control de los componentes animados de la escena, ya sean animados automáticamente, como las plataformas (Las cuales se mueven automáticamente al detectar una colisión con el jugador), o animados manualmente, como el vehículo del jugador (El cual reacciona al input del teclado(A / D o Izq. / Der).

Posee los siguientes métodos:

**Create\_World(),** el cual crea el mundo físico y sus componentes.

**GetMainVehicle(),** el cual devuelve un shared\_ptr con la dirección del vehículo.

**AnimateBodies(),** encargado de comprobar el comportamiento de los componentes animados automáticamente.

**PlatformCollision(),** función llamada por el CollisionListener al detectar una colisión valida.

**class Box2D\_RigidBody:**

Encapsulación de componentes B2D, esta clase permite crear un RigidBody de manera más simple y mecanizada, el constructor recibe los parámetros básicos para crear un RigidBody, además existen múltiples setters para configurar el elemento creado, por otro lado, existen diferentes getters para acceder a los componentes del RigidBody.

**Box2D\_RigidBody():** Constructor que necesita los valores básicos para crear un rigidbody

**SetShape():** Permite preseleccionar una de las formas aportadas en el enum Shapes, según las cuales existen diferentes formas preestablecidas en un switch dependiendo de la forma elegida, las formas añadidas son: Box ( Cuadrado ), Circle (Circulo), TriangleRectangle ( Triángulo Rectángulo ), Ramp ( Cuadrado inclinado, rampa de 45º), InvertedRamp (Cuadrado inclinado con una rampa de 225º)

**Setters**: Permiten ajustar parámetros adicionales del rigidbody

**class Box2D\_GameObject:**

Otra clase enfocada a la encapsulación de elementos B2D, en este caso a mas alto nivel, puesto que esta clase es la engargada de crear el vehiculo principal del ejemplo, usando rigidbodies, amortiguación, motores o uniones, además permite modificar los parámetros del motor para poder ajustar la dirección del vehiculo.

**SetDirecction()**: Permite ajustar la velocidad de los motores en las ruedas para ajustar la dirección del desplazamiento del vehiculo

**class CollisionListener**

Clase que hereda de la clase aportada por B2D b2ContactListener, esta clase es un listener de eventos de colisión en la escena, es utilizada para recibir una colisión, descifrar que dos componentes están colisionando y actuar según se haya indicado. En este ejemplo, cuando las ruedas están o dejan de estar en contacto con las plataformas, estas ascienden o descienden según la acción recibida.

**class Renderer**

Clase encargada del renderizado de elementos en la escena, permite convertir el espacio de los estándares B2D a SFML2, así como representar los elementos de B2D visualmente con gráficos SFML2. Esta clase ha sido construida a partir del ejemplo aportado, debido a su simpleza de comprensión y sencillez de implementación, ya que recorriendo los componentes de la escena y obteniendo tanto cuerpos físicos como fixtures, podemos sacar la forma que estos poseen y representarlas con elementos SFML2 fácilmente.

**Controles:**

- Movimiento ( W/A/S/D) o Flechas Direccionales

- Reinicio de la escena ( R )