PRÁCTICA 0

Amanda Oliver Calero

48773796L Fundamentos de los Videojuegos

Implementación de la práctica

Creando un Arkanoid

La clase Rectangulo

La clase Rectangulo es una clase auxiliar que será de la cual hereden las clases Nave y Ladrillo. He incluido en ella aquellos métodos y atributos que comparten en común, que son los siguientes:

- Atributos:
  - `RectangleShape shape` : La forma que tendrá nuestro objeto, en el caso de los ladrillos y la nave, cuadrado.
- Métodos:
  - `float get_x()` : Método que devuelve la posición x de la forma del objeto.
  - `float get_y()` : Método que devuelve la posición y de la forma del objeto.
  - `float izquierda()`: Método que calcula el borde izquierdo del objeto restando la posición x en la que comienza la forma menos el tamaño total que ocupa en el eje x dividido entre dos, es decir, el centro en el eje x.
  - `float derecha()` : Método que calcula el borde derecho.
  - `float arriba()` : Método que calcula el borde superior.
  - `float abajo()` : Método que calcula el borde inferior.

La clase Nave

- Atributos:
  - `float ancho_nave, alto_nave :` El alto y el ancho de la nave.
  - `float velocidad_nave :` La velocidad de la nave para poder simular las colisiones.
  - `Vector2f velocidad :` Un vector velocidad controlar el movimiento con shape.move y procurar que no se salga de los límites de la ventana.
  - `Texture textura :` La textura que vamos a usar en ella.
- Métodos:
  - `Nave(float,float) :` constructor de la nave, se le pasan por parámetro las posiciones x e y del objeto.
  - `RectangleShape get_shape()` : devuelve la forma del objeto para poder dibujarla.
  - `float get_velocidad_nave()` : devuelve la velocidad de la nave.
  - `void set_velocidad_nave(float)` : asigna un nuevo valor a la velocidad de la nave.

- `void actualizar_movimiento(float,float)` : actualiza el movimiento de la nave dentro de los límites de la ventana (pasados por parámetro) y controla que la nave no se pueda salir de ellos.

La clase Ladrillo

- Atributos:
  - `float ancho_ladrillo, alto_ladrillo` : El ancho y el alto que ocupará el ladrillo.
  - `bool destruido` : Indica si el ladrillo ha sido destruido o no, por defecto es false.

- Métodos:
  - `Ladrillo(float,float)` : El constructor del ladrillo, se le pasan las posiciones en el eje x e y. Se inicializa el shape con .setPosition , .setOrigin y .setSize, y .setFillColor
  - `RectangleShape get_shape()` : Devuelve la forma del ladrillo para poder dibujarla.
  - `bool get_destruido() const` : Devuelve el estado de la variable destruido.
  - `void set_destruido(bool)` : Asigna un nuevo valor a la variable destruido.
  - `float get_ancho_ladrillo()` : Devuelve el ancho del ladrillo.
  - `float get_alto_ladrillo()` : Devuelve el alto del ladrillo.

La clase Bola

- `Variables:`
  - `float x, y` : posición de la bola.
  - `float velocidad_bola` : velocidad de la bola.
  - `float radio` : radio de la bola.
  - `CircleShape shape` : forma de la bola.
  - `Vector2f velocidad` : vector velocidad de la bola.
  - `Texture textura` : textura a aplicar.
  - `int alpha` : transparencia de la bola.

- `Métodos:`
  - `Bola(float,float)` : crea la bola en la posición indicada e inicializa el shape asignándole una posición y una textura dentro de la imagen sprites.png.

- `float get_x()` : devuelve la posición en el eje x
- `float get_y()` : devuelve la posición en el eje y
- `CircleShape get_shape()` : devuelve la variable shape para poder dibujar el objeto.
- `void shape_set_position(float, float)` : asigna una nueva posición al shape del objeto.
- `float get_velocidad_x()` : devuelve la velocidad del vector velocidad en el eje y.
- `float get_velocidad_y()` : devuelve la velocidad del vector velocidad en el eje x.
- `float get_velocidad_bola()` : devuelve el valor de la variable velocidad_bola.
- `void set_velocidad(Vector2f)` : asigna un nuevo valor al vector de velocidad.
- `void set_velocidad_x(float)` : asigna un nuevo valor a la x del vector velocidad.
- `void set_velocidad_y(float)` : asigna un nuevo valor a la y del vector velocidad.
- `void set_velocidad_bola(float)` : asigna un nuevo valor a la velocidad de la bola.
- `float arriba()` : calcula el borde superior.
- `float abajo()` : calcula el borde inferior.
- `float izquierda()` : calcula el borde izquierdo.
- `float derecha()`: calcula el borde derecho.
- `void actualizar_movimiento(float,float)` : actualiza el movimiento de la bola dentro de los límites de la ventana.
- `template<class T1,class T2> bool isIntersecting(T1&, T2&)` : comprueba si dos objetos están colisionando. He usado una clase template de c++ para poder comprobar tanto la colision de la bola con la nave y la colisión de la bola con el ladrillo en un mismo método.
- `void colision_ladrillo_bola(Ladrillo&, Bola&)` : si el ladrillo y la bola están colisionando (se comprueba con isIntersecting)dependiendo de por qué lado ha colisionado la bola (se combrueba con las operaciones arriba(), abajo() izquierda() y derecha() de los dos objetos) se invertirá la velocidad de la bola en el eje y o en el eje x y al ladrillo se le cambiará la variable destruido a true.

- `void colision_nave_bola(Nave&, Bola&)` : la misma comprobación de colisiones que colision_ladrillo_bola pero entre la nave y la bola.

- `void parpadeo()` : aplica un color a la bola con una transparencia alpha que va cambiando de intensidad.

La aplicación Arkanoid

Desde la aplicación Arkanoid incluiremos todas las clases creadas hasta el momento y manejaremos los eventos con la clase Event.

Primeramente renderizamos una ventana con RenderWindow y le añadiremos un límite de frames a 60 frames por segundo.

Después crearemos un objeto Bola, un objeto Nave y un vector de ladrillos.

También con la clase Text crearemos un contador para las vidas y las opciones que se dibujaran en pantalla.

A parte crearemos un RectangleShape que será la línea límite donde, si toca la bola , perderemos una vida.

Crearemos un bucle donde, mientras la ventana sigue abierta:

- Leeremos los eventos introducidos por teclado.
- Si las vidas llegan a cero, el vector de ladrillos se generará otra vez.
- Se comprobarán las colisiones entre nave y bola (colision_nave_bola())
- Se comprobarán las colisiones entre Ladrillo y bola (colision_ladrillo_bola()) y si se cumple se eliminará el ladrillo del vector.
- Dibujaremos todos los objetos por pantalla.

Para más información, el mismo código de la práctica está comentado.

Preguntas propuestas

- ¿Que pasa si un sprite sale de la pantalla visible? ¿Porqué crees que esto ocurre así?

  Se deja de ver pero sigue existiendo. Esto pasa porque al establecer un límite de ventana, si nos pasamos de los bordes tanto en el eje y como en el x, los valores de la posición del sprite serán negativos sin afectar a su integridad.

- ¿Se ejecutara tu juego igual independientemente de la potencia del ordenador donde se ejecute?

  Establezco una restricción para que no se superen los 60fps, por lo tanto debería ser estable.