El objetivo de este ejercicio es usar el primer diseño de componentes que vimos en clase, para crear un caza que se puede girar, acelerar, etc.

Antes de empezar, y muy recomendable estudiar la transparencias **"La clase SDLUtils (y sus clases auxiliares)"** y **"La clase Vector2D"**. Ver cómo se usa en el juego ping pong (que vais a descargar en este ejercicio) o en la **`sdlutils/sdlutils_demo.cpp`.**

PARTE 0

1. Descarga la plantilla de proyecto SDL y verifica que funcione correctamente.
2. Descarga el juego Ping Pong **`ver_4.zip`** y descomprímelo.

a. Copia la carpeta game al proyecto (dentro **`TPV2/TPV2/src`**, al mismo nivel de **`sdlutils`**).

b. Copiar el **`main.cpp`** al proyecto reemplazando **`TPV2/TPV2/src/main.cpp`**.

c. Crea un filtro con el nombre **`game`** (uno en *Header Files* y otro en *Source Files*). Los filtros son una forma de organizar los archivos (en **`Visual Studio`**) para que tengan una estructura (en el *Project Explorer*) que corresponde a la estructura de carpetas.

d. Añade el contenido de la carpeta **`game`** al proyecto (los **`.h`** en **`Header Files -> game`** y los **`.cpp`** en **`Source Files -> game`***)*.

3. Descarga **`resources.zip`** y descomprímelo en **`TPV2/TPV2`** (reemplazando la carpeta **`resources`** actual).
4. Ejecuta el programa y verifica que funcione correctamente.

PARTE 1

1. Copia **`resources/config/resources.json`** a **`resources/config/test.resources.json`**.
2. Añade una entrada en **`test.resources.json`** para la imagen **`resources/images/fighter.png`**. La imagen ya existe en esa carpeta.
3. En el método **`init`** de **`Game.cpp`**, cambia **`pingpong.resources.json`** a **`test.resources.json,`** y comentar el resto de instrucciones del método **`init`** (deja solo la inicialización de **`SDLUtil`**).
4. En la clase **`Game`** comenta (o barra) el método **`checkCollisions`** y la llamada correspondiente en el método **`start`**, y los atributos **`ball_`**, **`leftPaddle_`**, **`rightPaddle_`** y **`gm_`**.
5. Añade un atributo **`fighter_`** de tipo **`Container*`** en **`Game.h`**.
6. En el método **`init`**, crea un **`Container`** y asignarlo al atributo **`fighter_`**.
7. Modifica la posición del **`fighter_`** para que esté en el centro de la ventana – mira como se hace para la pelota de ping pong.
8. Modifica el tamaño del **`fighter_`** (por ejemplo a **`50x50`**).
9. Añade un componente **`ImageRenderer`** a **`fighter_`** para mostrar la imagen **`fighter.png`**.

10. En el método **`init`**, añade **`fighter_`** al vector **`objs_`** (usando **`objs_.push_back(fighter_)`**).
11. Si ejecutas el programa tendrás que ver la imagen del caza en el centro de la pantalla

PARTE 2

1. Añade un atributo **`rot_`** de tipo **`float`** a la clase **`GameObject`** y inicializarlo a **`0.0f`** en la constructora. Lo usamos para representar la rotación del objeto.
2. Añade métodos **`setRotation`** y **`getRotation`** a la clase **`GameObject`** para cambiar y consultar el valor de **`rot_`**.
3. Cambia el componente **`ImageRenderer`** para que use **`render(dest,r)`** en lugar de **`render(dest)`**, donde **`r`** es la rotación del **`GameObject`** correspondiente. Eso hace que muestre la imagen con rotación.
4. En el método **`init`**, cambia la rotación de **`fighter_`** a **`90.0f`** para que mire hacia la derecha.
5. Si ejecutas el programa tendrás que ver la imagen del caza mirando hacia la derecha

PARTE 3

1. Escribe un componente de entrada **`FighterCtrl`** (y añadelo a **`fighter_`**) con el siguiente comportamiento: cuando el jugador pulsa **`SDLK_LEFT`** o **`SDLK_RIGHT`** se gira el **`GameObject`** en **`5.0f`** grados en la dirección correspondiente, es decir añade **`5.0f`** o **`-5.0f`** a la rotación actual.
2. Si ejecutas el programa tendrás que ver la imagen del caza mirando hacia la derecha, y puedes girarlo con las teclas **`SDLK_LEFT`** o **`SDLK_RIGHT`.**

PARTE 4

1. Añade un componente **`SimpleMove`** a **`fighter_`**. Este componente ya existe, simplemente mueve el caza en la dirección de su vector de velocidad.
2. Modifica **`FighterCtrl`** para que cuando el jugador pulsa la tecla **`SDLK_UP`** acelere el **`GameObject`** correspondiente (el caza en este caso).

   Para acelerar, suponiendo que **`vel`** es el vector de velocidad actual y **`r`** es la rotación, el nuevo vector de velocidad **`newVel`** sería **`vel+Vector2D(0,-1).rotate(r)*thrust`** donde **`thrust`** es el factor de empuje (usar **`0.2`** por ejemplo). Además, si la magnitud de **`newVel`** supera un límite **`speedLimit`** (usar **`3.0f`** por ejemplo) modifícalo para que tenga la magnitud igual a **`speedLimit`**, es decir a **`newVel.normalize()*speedLimit`**.

   Recuerda que las operaciones de la clase **`Vector2D`** no modifican el objeto actual, sino que devuelven una nueva instancia de **`Vector2D`**. Recuerda que el caza puede mirar hacia una dirección (depende de su rotación) pero se mueve en otra dirección (depende de su vector de velocidad).
3. Si ejecutas el programa, podrás girar el caza como antes y moverlo usando **`SDLK_UP`**. Cuando el caza llega a los límites de pantalla sale y no vemos más (arreglamos esto en el siguiente apartado).

PARTE 5

1. Escribe un componente de física **`ShowAtOpposieSide`** (y añadelo a **`fighter_`**) con el siguiente comportamiento: cuando el **`GameObject`** correspondiente sale de un borde de la ventana (por completo) empieza a aparecer en el borde opuesto. No hace falta tener la rotación en cuenta. Recuerda que **`sdlutils().height()`** y **`sdlutils().width()`** devuelven el tamaño de la ventana.
2. Si ejecutas el programa, ahora cuando el caza sale de un lado entra por el lado opuesto.

PARTE 6

1. Escribe un componente de física **`DeAcceleration`** (y añadelo a **`fighter_`**) que desacelera el caza automáticamente en cada iteración del juego, por ejemplo multiplicando su vector de velocidad por `0.995f`.
2. Si ejecutas el programa, cuando dejas de pulsar **`SDLK_UP`** el caza desacelera.