TP Videojuegos 2 - Ejercicio

En este ejercicio, vamos a practicar con el primer diseño de componentes que vimos en clase.

PARTE 0: Configuración del Proyecto

1. **Descarga y Verificación:** Descarga la plantilla del proyecto SDL y asegúrate de que funcione correctamente. Este paso es crucial para garantizar una base sólida.
2. **Integración del Juego Ping Pong:**
    - Descarga el archivo `Ping Pong ver_4.zip` y descomprímelo.
    - Copia la carpeta `game` al directorio `TPV2/src`, al mismo nivel que la carpeta `sdlutils`. La estructura de directorios debe quedar como: `TPV2/src/game` y `TPV2/src/sdlutils`.
    - Reemplaza el archivo `TPV2/src/main.cpp` con el archivo `main.cpp` incluido en el zip de Ping Pong.
3. **Recursos del Juego:**
    - Descarga `resources.zip` y descomprímelo en `TPV2`, reemplazando la carpeta `resources` existente. Asegúrate de que la estructura de carpetas sea correcta.
4. **Verificación Final:** Ejecuta el programa. Si todo está configurado correctamente, el juego Ping Pong debería funcionar.

PARTE 1: Preparación de Recursos y Entidades

1. **Copia de Configuración:** Copia `resources/config/resources.json` a `resources/config/test.resources.json`. Esta copia permitirá realizar pruebas sin modificar la configuración original.
2. **Añadir Recurso:** Añade una nueva entrada en `test.resources.json` para la imagen `resources/images/fighter.png`. Asegúrate de que la ruta sea correcta.
3. **Carga de Recursos Modificada:** En el método `init` de `Game.cpp`, modifica la línea que carga `pingpong.resources.json` para que ahora cargue `test.resources.json`. Comenta (o elimina) el resto de las instrucciones dentro del método `init`, dejando **solo** la inicialización de `SDLUtil` y `InputHandler`.
4. **Preparación de la Clase Game:**

- Comenta (o elimina) el método `checkCollisions` y su llamada correspondiente en el método `start`.
- Comenta (o elimina) los atributos `_ball`, `_leftPaddle`, `_rightPaddle` y `_gm`.
- Declara un nuevo atributo `_fighter` de tipo `Container*` en `Game.h`.

5. **Creación y Configuración del Contenedor Fighter:**
   - Dentro del método `init`, crea una instancia de `Container` y asígnala al atributo `_fighter`.
   - Modifica la posición de `_fighter` para que esté centrado en la ventana. Puedes usar `sdlutils().width()/2` y `sdlutils().height()/2`.
   - Ajusta el tamaño de `_fighter` a, por ejemplo, 50x50 píxeles.
   - Agrega un componente `ImageRenderer` a `_fighter` para que muestre la imagen `fighter.png`.
6. **Añadir Fighter al Juego:** En el método `init`, añade `_fighter` al vector `_objs` usando `_objs.push_back(_fighter)`.

PARTE 2: Rotación de Objetos

1. **Atributo de Rotación:** Añade un atributo `_rot` de tipo `float` a la clase `GameObject`. Inicialízalo a `0.0f` en el constructor.
2. **Métodos de Rotación:** Implementa los métodos `setRotation(float)` y `getRotation()` en la clase `GameObject` para modificar y acceder al valor de `_rot`.
3. **Modificación del ImageRenderer:** Modifica el componente `ImageRenderer` para que utilice `render(dest, r)` en lugar de `render(dest)`. El parámetro `r` representa la rotación del `GameObject`. Esto hará que la imagen se dibuje rotada.
4. **Rotación Inicial:** En el método `init`, rota `_fighter` 90.0f grados para que mire hacia la derecha.

PARTE 3: Control del Fighter con el Teclado

1. **Componente FighterCtrl:** Crea un nuevo componente llamado `FighterCtrl` y añádelo a `_fighter`.

2. **Implementación del Control:** Dentro de `FighterCtrl`, implementa la lógica para que cuando el jugador presione la tecla `SDLK_LEFT` o `SDLK_RIGHT`, el `GameObject` rote 5.0f grados en la dirección correspondiente (sumando o restando 5.0f a la rotación actual). Experimenta con las siguientes dos opciones para ver si se ha pulsado una tecla y comprueba cuál es la diferencia entre ellas: `ihdlr.keyDownEvent() && ihdlr.isKeyDown(...)` y `ihdlr.isKeyDown(...)`.

PARTE 4: Movimiento y Aceleración

1. **Componente SimpleMove:** Añade un componente `SimpleMove` a `_fighter`.
2. **Control de Aceleración:** Modifica `FighterCtrl` para que, cuando el jugador presione la tecla `SDL_ARROW_UP`, el `GameObject` acelere.
3. **Cálculo de la Aceleración:** Para calcular la nueva velocidad (`newVel`), usa la siguiente fórmula:

   ```
   newVel = vel + Vector2D(0, -1).rotate(r) * thrust;
   ```

   Donde:

   - `vel` es el vector de velocidad actual.
   - `r` es la rotación del objeto.
   - `thrust` es el factor de empuje (ejemplo: 0.2f).
4. **Límite de Velocidad:** Si la magnitud de `newVel` supera `speedLimit` (ejemplo: 3.0f), limita la velocidad a `speedLimit`:

   ```
   if (newVel.magnitude() > speedLimit) {
       newVel = newVel.normalize() * speedLimit;
   }
   ```

PARTE 5: Desaceleración

1. **Componente DeAcceleration:** Crea un componente `DeAcceleration` y añádelo a `_fighter`.
2. **Implementación de la Desaceleración:** Dentro de `DeAcceleration`, implementa la lógica para desacelerar el caza automáticamente en cada iteración del juego. Por ejemplo, puedes multiplicar el vector de velocidad por un factor menor a 1 (ejemplo: 0.995f).

PARTE 6: Reaparición en el Lado Opuesto

1. **Componente ShowAtOppositeSide:** Crea un componente `ShowAtOppositeSide` y añádelo a `_fighter`.
2. **Implementación de la Reaparición:** Implementa la lógica para que cuando el `GameObject` salga completamente de un borde de la ventana, aparezca en el borde opuesto. No es necesario considerar la rotación en este componente. Puedes usar `sdlutils().height()` y `sdlutils().width()` para obtener las dimensiones de la ventana.