TP Videojuegos
Ejercicio 1

Objetivos: familiarizarse con un conjunto de clases que vamos a utilizar en las prácticas de este cuatrimestre; utilizar plantillas de C++; utilizar herencia múltiple para evitar la duplicación de código y mejorar la separación de interfaces.

Paso 0: Descargar y ejecutar el código

Descarga el código correspondiente. Esta plantilla incluye ya todas la librerías que vamos a usar durante el cuatrimestre, también incluye el código necesario para este ejercicio.

Todos lo archivos `.h` y `.cpp` están en la carpeta `HolaSDL` (úsala para tus archivos `.cpp` y `.h`). Los archivo de imágenes, sonido, etc., están en la carpeta `HolaSDL/resources` (úsala para tus archivos de imágenes, sonidos, etc.)

El pájaro se controla con las teclas `S`, `A` y `Z`. El caza se controla con la teclas ←, ↑, →, ↓ y espacio.

Paso 1: Leer y entender el código

Repase las diferentes partes del código para entender cómo funciona:

1. `GameObject`: repase el código y entender sus diferentes métodos
2. `DemoGame` y `SDLGame`: repase los métodos `initGame()` y `start()` para entender cómo se inicializa el juego, cómo es el bucle principal, etc. Repase los métodos `getGame()`, `getRenderer()`, `getServiceLocator()`, etc.
3. `Bird` y `Fighter`: repase los distintos métodos, presta atención a cómo se usa `getServiceLocator()` para obtener y renderizar texturas, etc.
4. `EggsShooter` y `BulletsShooter`: estas clases implementan una variante de *"Object Poll"*. Repase el código, comenzando por los métodos `addEgg()` y `addBullet()`, para entender qué hacen estas clases. Presta atención a cómo reproducimos los sonidos, etc.

Paso 2: Refactorización

Las clases `EggsShooter` y `BulletsShooter` tienen mucho código común (i.e., duplicado):

- Los atributos son iguales, sólo el tamaño y el tipo del array cambian.
- Los métodos `handleInput()`, `update()` y `render()` son idénticos.
- Los métodos `getUnusedEgg()` and `getUnusedBullet()` son parecidos, sólo el tipo de la salida cambia.
- Los constructores son iguales, en ambos casos hay un bucle para inicializar los objetos llamando a `setGame()` and `setActive()`.

Refactorizar este código creando un *template class* que incluye las partes comunes:

```
template<typename T, int SIZE>
class ObjectPool : public GameObject {
   // ...
}
```

Modificar `EggsShooter` y `BulletsShooter` para que heredan de `ObjectPool` con el tipo y tamaño adecuados:

```
class EggsShooter : public ObjectPoll<Egg,20> {
  // ...
}

class BulletsShooter : public ObjectPoll<Bullet,100> {
  // ...
}
```

Paso 3: Separación de interfaces

Queremos ocultar de `Bird` y `Fighter` todos los métodos innecesarios de las clases `EggsShooter` y `BulletsShooter`, es decir, todos excepto `addEgg` y `addBullet`:

1. Definir dos interfaces correspondientes que incluyen sólo estos métodos.
2. Usar las interfaces en las definiciones de `EggsShooter` y `BulletsShooter`.
3. Cambiar `Bird` y `Fighter` para que usen esas interfaces en lugar de usar `EggsShooter` y `BulletsShooter` directamente.