TP Videojuegos 2
Práctica 1

Fecha Límite: 10/03/2020 a las 09:00.

En esta práctica vamos a desarrollar una variación del juego clásico Asteroids. Se recomienda leer todo el enunciado antes de empezar.

Descripción General

En el juego Asteroids hay 2 actores principales: *el caza* y *los asteroides*.

El objetivo del caza es destruir los asteroides disparándoles. El caza tiene 3 vidas y cuando choca con un asteroide explota y pierde una vida, si tiene más vidas el jugador puede jugar otra ronda. El juego termina cuando el caza no tiene más vidas (pierde) o destruye a todos los asteroides (gana). Al comienzo de cada ronda colocamos **10** asteroides aleatoriamente en los bordes de la ventana, con velocidad aleatoria. Los asteroides tienen un *contador de generaciones* con valor inicial aleatorio entre **1** y **3** (cuando se crean al comienzo de la ronda).

Cuando el caza destruye un asteroide **a**, el asteroide debe desaparecer de la ventana, y si su contador de generaciones es positivo creamos **2** asteroides nuevos. El contador de generación de cada asteroide nuevo es como el de **a** menos uno, su posición es cercana a la de **a** y su vector de velocidad se obtiene girando el vector de velocidad de **a** y cambiando su magnitud para que sea más rápido (más adelante explicamos cómo calcular la posición y la velocidad). El caza gana **1** punto al destruir un asteroide.

El caza está equipado con un arma que puede disparar (más detalles abajo). El número de vidas del caza se debe mostrar en la esquina superior izquierda y el total de puntos ganados en el centro. Entre las rondas y al terminar una partida, se debe mostrar mensajes adecuados y pedir al jugador que pulse cualquier tecla para continuar.

Se debe reproducir los sonidos correspondientes cuando el caza dispara una bala, cuando un asteroide explota, cuando el caza choca con un asteroide, etc. Además, cuando el juego está en marcha, no entre las rondas, se debe reproducir alguna música. En el campus virtual se puede encontrar archivos de sonido, música e imágenes, pero se pueden usar otros. Recuerda que hay que modificar los archivos `Resources.h` y `Resources.cpp`.

Ver el video del campus virtual para tener una idea de lo que tienes que implementar. A continuación detallamos las entidades del juego y sus componentes. La lista de componentes de cada entidad es una recomendación, se pueden usar otros componentes pero hay que justificarlo durante la corrección.

Entidad del caza

Es una entidad para representar el caza. La entidad tiene los siguientes componentes:

1. **`Transform:`** un componente para mantener las características físicas del caza.
2. **`FighterViewer:`** un componente para dibujar el caza. Usa "`airplanes.png`" con el corte `{47, 90, 207, 250}`.
3. **Health:** un componente para mantener el número de vidas del caza y dibujarlos en alguna parte de la ventana, p.ej., mostrando "`heart.png`" tantas veces como el número de vidas el la parte superior de la ventana. El componente tiene métodos para quitar una vida, resetear las vidas, etc. El caza tiene `3` vidas al principio de cada partida.
4. **`FighterCtrl:`** un componente para controlar los movimientos del caza. Pulsando `SDLK_LEFT` o `SDLK_RIGHT` gira el caza `5` grados en la dirección correspondiente. Pulsado `SDLK_UP` acelera el caza (*ver el apéndice para más detalles*).
5. **`Gun:`** un componente para disparar. Este componente necesita acceso al **`BulletsPool`** (ver abajo). Pulsando `SDLK_SPACE` dispara una bala en la dirección hacia donde mira el caza llamando al método **`shoot(...)`** del **`BulletsPool`** (*ver el apéndice para más detalles*). Se puede disparar sólo una bala cada `0.25sec` (usa `game_->getTime()` para consultar el tiempo actual).
6. **`FighterMotion:`** un componente para mover el caza. Suma la velocidad a la posición y desacelera automáticamente (p.ej., multiplicando la velocidad por `0.995`). El caza no puede salir de los bordes de la ventana, si su nueva posición choca con un borde hay que mantener la su posición anterior y multiplicar el vector de velocidad por `-1` (efecto de rebote).

Entidad de los asteroides

Es un entidad para representar los asteroides. Vamos a usar un *Object Pool* en lugar de tener una entidad para cada asteroide. Primero definir una clase `Asteroid` con atributos de *posición*, *velocidad*, *tamaño*, *rotación* y *generaciones*. Los componentes de esta entidad son los siguientes:

- ➔ **`AsteroidPool:`** un componente que tiene (como atributo) un `ObjectPool` de `Asteroid` de tamaño `30` y tiene los métodos:
  - ◆ **`generateAsteroids(int n):`** genera "n" asteroides con características aleatorias (*ver el apéndice para más detalles*).
  - ◆ **`disablAll():`** desactiva todos los asteroides (marca todos los objetos como no usados).
  - ◆ **`onCollision(Asteroid* a, Bullet* b):`** se llama a este método cuando un asteroide choca con una bala. Lo que tiene que hacer es dividir el asteroide **a** en `2` (*ver el apéndice para más detalles*).

- ◆ **`getNumOfAsteroid():`** devuelvo el número de asteroides activos (no contar cada vez, mantener un atributo!).
- ◆ **`getPool():`** devuelve el vector de asteroides (usando `getPool()` de `ObjectPool`).

➔ **`AsteroidsMotion:`** un componente para mover los asteroides, sumando la velocidad a la posición y girando cada asteroide en `0.5` grados (probar con otros valores). Si un asteroide sale de un lado de la ventana tiene que aparecer en el lado opuesto.

➔ **`AsteroidsViewer:`** un componente para dibujar los asteroides. Usa "`asteroid.png`".

Entidad de las balas

Es una entidad para representar las balas, es muy parecida a la entidad de los asteroides. Vamos a usar un *Object Pool* en lugar de tener una entidad para cada bala. Primero definir una clase `Bullet` con atributos de *posición*, *velocidad*, *tamaño* y *rotación*. Los componentes de esta entidad son los siguientes:

➔ **`BulletsPool:`** un componente que tiene (como atributo) un `ObjectPool` de `Bullet` de tamaño `10` y tiene los métodos:

- ◆ **`shoot(Vector2D pos, Vector2D vel, double w, double h):`** se usa para añadir una bala al juego con posición "`pos`", velocidad "`vel`", anchura "`w`", altura "`h`" y rotación como la dirección del vector "`vel`", es decir "`Vector(0,-1).angle(vel)`".
- ◆ **`disablAll():`** desactiva todas las balas (marca todos los objetos como no usados).
- ◆ **`onCollision(Bullet*b, Asteroid* a):`** se llama a este método cuando una bala choca con un asteroide. Desactiva la bala (marcar el objeto como no usado).
- ◆ **`getPool():`** devuelve el vector de balas (usando `getPool()` de `ObjectPool`).

➔ **`BulletsMotion`:** un componente para mover las balas, sumando la velocidad a la posición. Si una bala sale de la ventana la desactiva (marca el objeto como no usado).

➔ **`BulletsViewer:`** un componente para dibujar las balas. Se puede dibujar cada bala como un rectángulo lleno teniendo en cuenta la rotación (o más fácil usar "`whiterect.png`").

Entidad de manejo del juego

Es responsable de manejar el juego. Los componentes de esta entidad son los siguientes:

➔ **`GameCtrl:`** este componente necesita acceso a componentes de otras entidades, en particular a **`AsteroidsPool`** y **`Health`**. Si el estado del juego es *parado*, muestra el mensaje *"Press Any Key To Start"* y si el jugador pulsa cualquier tecla empieza una nueva ronda

llamando a **`generateAsteroids(10)`** del **`AsteroidsPool`** y cambia el estado del juego a *no parado*. Si el juego ha *terminado* resetea el marcador y las vidas (`3` vidas y `0` puntos).

- ➔ **`ScoreManager:`** un componente para mantener los puntos ganados, el estado del juego, y el ganador. El estado del juego puede ser *parado*, *no parado*, *terminado*, o *no terminado*. Además, tiene métodos para consultar y modificar los atributos.
- ➔ **`ScoreViewer:`** un componente para mostrar el marcador y un mensaje correspondiente si el juego ha terminado (*"Game Over! You won!*" o "*Game Over! You lost*").
- ➔ **`GameLogic:`** este componente necesita acceso a componentes de otras entidades, en particular a **`AsteroidsPool`**, **`BulletsPool`**, **`Health`** y **`Transform`** del caza. Si el estado del juego es *no parado*, para cada asteroide activo comprueba lo siguiente:
  - ◆ Si choca con el caza, en ese caso desactiva todos los asteroides y las balas, quita una vida al caza, marca el juego como *parado* y como *terminado* si no quedan vidas (pierde el caza), y pone al caza en centro de la ventana con velocidad y rotación `0`.
  - ◆ Si choca con una bala activa, en ese caso llama a **`onCollision`** de **`AsteroidsPool`** y **`BulletsPool`**, añade un punto al marcador, si no hay más asteroides en la ventana, marca el juego como *terminado* (gana el caza), y pone al caza en centro de la ventana con velocidad y rotación `0`.

  Para comprobar colisiones usa el método `Collisions::collidesWithRotation.`

Otros requisitos

- Es obligatorio usar el sistema de componentes del ejemplo "`ver_7.zip`" del juego *Ping Pong*.
- Limpiar el código para que no tenga ninguna clase del juego *Ping Pong* (se puede reusar componentes de ese juego, pero hay que borrar lo que no se usa). Renombrar la clase principal `PingPoing.cpp` a `Asteroids.cpp`
- Clases que usan datos concretos (como teclas, texturas, constantes), tienen que recibir esos valores en la constructora o tener métodos para cambiarlos.
- **OPCIONAL**: usar un archivo de configuración (formato JSON) para proporcionar la configuración del juego: número de vidas, disparos por segundo, etc.

Otros comentarios

- Durante la corrección, aparte de la funcionalidad, se va a tener en cuenta la claridad y organización del código.
- Para los distintos sonidos usa "`gunshot.wav`" y "`explosion.wav`" y para la música de fondo usa "`imperial_march`.wav".
- Para reproducir/parar la música usa los siguientes métodos del *Audio Manager*: `playMusic`, `haltMusic`, `pauseMusic`, `resumeMusic`, `etc.`
- Para reproducir efectos de sonido usa el método `playChannel` del *Audio Manager*.

APÉNDICE

Cómo acelerar el caza

El movimiento del caza está basado en empujones, eso hace que el juego sea más difícil. Además, como hemos visto antes, la desaceleración se hace de manera automática, el jugador no puede desacelerar. 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 (usa p.ej. `0.5`). Además, si la magnitud de "newVel" supera un límite "`speedLimit`" (usa p.ej., `2`) modifícalo para que tenga la magnitud igual a "`speedLimit`", es decir modificalo a "newVel`.normalize()*speedLimit`".

Cómo calcular la posición y dirección de la bala

Sea "`pos`" la posición del caza, "`r`" su rotation, "`w`" su anchura, y "`h`" altura. El siguiente código calcula la posición y la velocidad de la bala:

```
Vector2D bulletPos = p + Vector2D(w/2, h/2) + Vector2D(0, -(h/2 + 5.0)).rotate(r);
Vector2D bulletVel = Vector2D(0, -1).rotate(r)*2;
```

En principio, la posición es un punto cerca del frente del caza (teniendo en cuenta su rotación) y la velocidad es un vector de magnitud `2` en la dirección a donde mira el caza. Se puede cambiar `2` por otro número para controlar la velocidad de la bala.

Cómo crear un asteroide en el método `generateAsteroids`

Al crear un nuevo asteroide, hay que asignarle posición y velocidad aleatorias. Además, hay que elegir el vector de velocidad de tal manera que el asteroide mueva hacia la zona central de la ventana.

Primero elegimos su posición "`p`" de manera aleatoria en los bordes de la ventana. Después elegimos una posición aleatoria "`c`" en la zona central usando "`c=(cx,cy)+(rx,ry)`" donde "`(cx,cy)`" es el centro de la ventana y "`rx`" y "`ry`" son números aleatorios entre `-50` y `50` (se puede probar con otros valores). Ahora su vector de velocidad sería "`(c-p).normalize()*(m/10.0)`" donde "`m`" es un número aleatorio entre `1` y `10.`

El número de generaciones del asteroide es un número aleatorio entre `1` y `3`. La anchura y altura del asteroide dependen de su número de generaciones, p.ej., `10+3*g` donde "g" es el número de generaciones.

Cómo dividir un asteroide

En el método **onCollision** de **AsteroidsPoll**, al destruir un asteroide **a** la idea es desactivarlo y crear otros 2 con el número de generaciones como el de **a** menos uno (si el número de generaciones de **a** es 0 no se genera nada). La posición de cada nuevo asteroide es cercana al de **a** y tiene que moverse en otra dirección. Por ejemplo, suponiendo que "`pos`" y "`vel`" son la velocidad y la posición de **a,** se puede usar el siguiente código para calcular la posición "p" y velocidad "v" de i-ésimo asteroid:

```
Vector2D v = vel.rotate(i*45);
Vector2D p = pos + v.normalize()
```

Recuerda que la anchura y altura del asteroide dependen de su número de generaciones, p.ej., `10+3*g` donde "g" es el número de generaciones.