TP Videojuegos 2
Práctica 1

Fecha Límite: 06/04/2021 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**. Además, algunos asteroides actualizan su vector de velocidad continuamente para avanzar en la dirección del caza.

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 a partir del vector de velocidad de **"a"** (más adelante explicamos cómo calcular la posición y la velocidad).

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. Entre las rondas y al terminar una partida, se debe mostrar mensajes adecuados y pedir al jugador que pulse SPACE para continuar.

Se debe reproducir sonidos correspondientes cuando el caza dispara una bala, cuando el caza acelera, cuando un asteroide explota, cuando el caza choca con un asteroide, etc. Archivos de sonido e imágenes están disponibles en el campus virtual (pero se pueden usar otros).

Ver el video para tener una idea de lo que tienes que implementar. El los detalles a continuación, los componentes y las entidades son una recomendación, se pueden usar otros componentes/entidades pero hay que justificarlo durante la corrección.

Entidad del caza

Es una entidad para representar el caza con los siguientes componentes:

1. **`Transform:`** mantiene las características físicas y mueve la entidad sumando la velocidad a la posición.
2. **`DeAcceleration:`** desacelera el caza automáticamente en cada iteración del juego (p.ej., multiplicando la velocidad por `0.995f`).
3. **`Image:`** dibuja el caza usando **`fighter.png`**.
4. **`Health:`** mantiene el número de vidas del caza y las dibuja en alguna parte de la ventana, p.ej., mostrando **`heart.png`** tantas veces como el número de vidas en el la parte superior de la ventana.Además, tiene métodos para quitar una vida, resetear las vidas, consultar el número de vidas actual, etc. El caza tiene `3` vidas al principio de cada partida.
5. **`FighterCtrl:`** controla los movimientos del caza. Pulsando `SDLK_LEFT` o `SDLK_RIGHT` gira `5.0f` grados en la dirección correspondiente y pulsado `SDLK_UP` acelera *(más información en el apéndice).* Al acelerar hay que reproducir el sonido **`thrust.wav.`**
6. **`Gun:`** pulsando `SDLK_S` añade una bala al juego y reproduce el sonido **`fire.wav`** *(màs información en el apéndice)*. Se puede disparar sólo una bala cada `0.25sec` (usar `sdlutils().currTime()` para consultar el tiempo actual).
7. **`ShowAtOpposieSide:`** cuando el caza sale de un borde (por completo) empieza a aparecer en el borde opuesto *(ver el video)*.

Entidad de un asteroide

Es una entidad para representar un asteroide con los siguientes componentes (hay dos tipos de asteroides, `A` y `B`):

1. **`Transform:`** mantiene las características físicas y mueve la entidad sumando la velocidad a la posición.
2. **`FramedImage:`** dibuja el asteroide usando la imagen **`asteroids.png`** para el tipo `A` y **`asteroids_gold.png`** para el tipo `B`. Hay que cambiar el frame cada `50ms`.
3. **`ShowAtOpposieSide:`** cuando el asteroide sale de un borde (por completo) empieza a aparecer en el borde opuesto *(ver el video)*.
4. **`Generations:`** mantiene el número de generaciones del asteroide.
5. **`Follow:`** actualiza el vector de velocidad para que siga al caza *(màs información en el apéndice).* Se usa sólo para asteroides de tipo B.

La entidad pertenece al grupo **`Asteroid_grp`**.

TP Videojuegos 2, Práctica 1, Fecha Limite: 06/04/2021 a las 09:00.

Entidad de una bala

Es una entidad para representar una bala con los siguientes componentes:

1. **`Transform:`** mantiene las características físicas y mueve la entidad sumando la velocidad a la posición.
2. **`Image:`** dibuja la bala usando la imagen **`fire.png`**
3. **`DisableOnExit:`** desactiva la bala cuando sale por completo de la ventana.

La entidad pertenece al grupo **`Bullet_grp`**.

La entidad GameManager

Es una entidad para controlar el juego con los siguientes componentes (se puede implementar directamente en la clase `Game` usando distintos métodos, no hace falta una entidad y componentes):

1. **`State:`** mantiene el estado de juego (`NEWGAME`, `PAUSED`, `RUNNING`, y `GAMEOVER`) y muestra mensajes adecuados según el estado del juego (como el `PingPong`).
2. **`GameCtrl:`** `Si` el jugador pulsa la tecla `SDLK_SPACE`, si el estado es distinto de `RUNNING`, cambia el estado del juego (como el `PingPong`). Al empezar una ronda, es decir cuando el cambia a `RUNNING`, hay que añadir `10` asteroides al juego (pedírselo al `AsteroidsManager`).
3. **`AsteroidsManager:`** Maneja la generación de asteroides:
   a. Mantiene un atributo con el número de asteroides en la ventana y permite consultarlo.
   b. Genera `1` asteroide nuevo cada `5` segundos (aparte de los `10` al principio de cada ronda).
   c. Cuando genera un asteroide, con probabilidad de `70%` genera uno de tipo `A` y el `30%` uno de tipo `B` (se puede decidir usando la condición `sdlutils().rand().nextInt(0,10)<3`).
   d. Tiene un método `onCollison` que recibe una entidad representando un asteroide que haya chocado con una bala, lo desactiva y genera otros 2 dependiendo de su número de generaciones *(más información en el apéndice)*.
4. **`CollisionsManager`**: comprueba choques para cada asteroide activo:
   a. Si hay choque con el caza, desactiva todos los asteroides y las balas, quita una vida al caza, marca el juego como `PAUSED` o `GAMEOVER` (depende de si quedan vidas) y pone al caza en el centro de la ventana con velocidad y rotación `0`.
   b. Si choca con una bala activa, desactiva la bala y llama a `onCollison` del **`AsteroidManager`** pasándole el asteroide para destruirlo, etc. Si no hay más asteroides en la ventana, marca el juego como *terminado* (gana el caza), y pone al caza en el centro de la ventana con velocidad y rotación `0`.

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

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. `0.2`). Además, si la magnitud de "`newVel`" supera un límite "`speedLimit`" (usa p.ej., `3`) modifícalo para que tenga la magnitud igual a "`speedLimit`", es decir modificarlo a "`newVel.normalize()*speedLimit`".

Cómo calcular la posición y dirección de la bala

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

```
bPos = p+Vector2D(w/2.0f,h/2.0f)-Vector2D(0.0f,h/2.0f+5.0f+12.0f).rotate(r)-Vector2D(2.0f,10.0f);
bVel = Vector2D(0.0f,-1.0f).rotate(r)*(vel.magnitude()+5.0f);
```

El tamaño de la bala es `5.0f` de anchura y `20.0f` de altura, y la rotación es la misma que la del caza.

Cómo crear un asteroide

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 se mueve 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 `-100` y `100`. Una posibilidad del vector de velocidad sería

```
(c-p).normalize()*( sdlutils().rand().nextInt(1,10)/10.0)
```

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+5*g` donde "`g`" es el número de generaciones.

Para los asteroides de tipo B, el componente `Follow` tiene que girar el vector de velocidad en `1.0f` grado en cada iteración para que el asteroide vaya hacia el caza. Si "`v`" es el vector de velocidad del asteroide, "`p`" su posición, y "`q`" la posición del caza, se puede conseguir este efecto cambiando el vector de velocidad del asteroide a `v.rotate(v.angle(q-p) > 0 ? 1.0f : -1.0f).`

TP Videojuegos 2, **Práctica 1, Fecha Limite**: 06/04/2021 a las 09:00.

Cómo dividir un asteroide

En el método **`onCollision`** de **`AsteroidsManager`**, 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 una dirección aleatoria.

Por ejemplo, suponiendo que "p" y "v" son la velocidad y la posición de **"a"**, "w" es su anchura, y "r" un número aleatorio `sdlutils().rand().nextInt(0,360)`, se puede usar el siguiente código para calcular la posición y velocidad de cada asteroide nuevo:

1. Posición: `p + v.rotate(r) * 2 * w`.
2. Velocidad: `v.rotate(r) * 1.1f`

Recuerda que la anchura y altura del asteroide dependen de su número de generaciones, p.ej., `10.0f+5.0f*g` donde "g" es el número de generaciones.