TP Videojuegos 2
Práctica 1

Fecha Límite: 14/03/2022 a las 09:00.

En esta práctica vamos a desarrollar una variación del juego clásico Asteroids. **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 explicaremos 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. En 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.

Es muy recomendable empezar la implementación de tu práctica a partir del ejemplo del **PacMan** y las estrellas.

TP Videojuegos 2, Práctica 1, Fecha Límite: 14/03/2022 a las 09:00.

Requisitos Generales (obligatorios)

1. Crea un archivo con el nombre **"resources/config/asteroid.resources.json"** y usarlo para definir referencias a todos los recursos necesarios (imágenes, sonidos, etc). Hay que usar la versión **"ecs_1.zip"** (descargar desde el campus virtual), no se puede usar otra versión.
2. La última versión de **resources.zip** (en el campus virtual) incluye todos los archivos del imágenes y sonidos necesarios para esta práctica.
3. La estructura de las carpetas tiene que ser parecida a la que uso en los ejemplos. Usando la plantilla de proyecto **TPV2**, hay que tener **3** carpetas adicionales: **src/ecs**, **src/components** y **src/game**). Poner los componentes en la carpeta **components** y la clase **Game** y el archivo **ecs_def.h** en la carpeta **game**. El archivo **main.cpp** tiene que estar directamente en la carpeta **src**, es decir al mismo nivel de las otras carpetas.
4. Hay que inicializar todos los atributos en las constructoras (en el mismo orden de la declaración).
5. Al usar la directiva **"#include"**, escribe el nombre del archivo *respetando minusculas y mayusculas*. El problema es que Windows no distingue entre mayúsculas y minúsculas pero otros sistemas operativos sí.
6. Usar el formateo (indentation) automático de **Visual Studio**, así el código es mucho más fácil de leer.
7. Entregar un **zip** con todo el proyecto, quitando el **".vs"** y todo lo que no usas de la carpeta **resources**.

Requisitos Opcionales

1. Introduce la posibilidad de configurar el juego desde un archivo de configuración usando el formato json, p.ej., **"resources/config/asteroid.config.json"**, parecido a lo que hemos hecho para el juego **Ping Pong** en clase. Puedes elegir qué valores se pueden configurar, p.ej., el número de asteroides inicial, la frecuencia de generación de asteroides, el número máximo de asteroides que puedan existir al vez, la velocidad máxima del caza, el grado de rotación del caza, el número de asteroides que generamos al destruir un asteroide, la velocidad de la balas, etc.

TP Videojuegos 2, Práctica 1, Fecha Límite: 14/03/2022 a las 09:00.

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`**), y si la magnitud del vector de velocidad llega a ser menor de un límite muy pequeño (p.ej, **`0.05f`**) cambia el vector de velocidad a **`(0.0f,0.0f)`**.
3. **`Image:`** dibuja el caza usando **`"fighter.png"`** (o cualquier otra imagen que quieras).
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 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 ← o → gira **`5.0f`** grados en la dirección correspondiente y pulsando ↑ acelera *(más información en el apéndice).* Al acelerar hay que reproducir el sonido **`"thrust.wav"`.** Recuerda que la desaceleración es automática, no se maneja en este componente.
6. **`Gun:`** pulsando **`S`** añade una bala al juego y reproduce el sonido **`"fire.wav"`** *(más 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 (pertenece al grupo **`"_grp_ASTEROIDS"`**) 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`** para conseguir el efecto de la rotación (y no cambiando la rotación en el componente **`Transform`**).
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, y tiene métodos para consultarlo, cambiarlo, etc.
5. **`Follow:`** actualiza el vector de velocidad para que siga al caza *(más información en el apéndice).* **Se usa sólo para asteroides de tipo `B`.**

Entidad de una bala

Es una entidad para representar una bala (pertenece al grupo **"_grp_BULLETS")** 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, es decir llama a su **isAlive(false)**, cuando sale por completo de la ventana.

Entidad para el control global de juego

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. **AsteroidsManager:** tiene un contador para el número de asteroides actuales y métodos para consultarlo, cambiarlo, etc. Además
   a. Su método **update** genera **1** asteroide nuevo cada **5** segundos (aparte de los **10** al principio de cada ronda), y 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)"**. No pueden existir más de **30** asteroides a la vez en la pantalla.
   b. Tiene un método **onCollision(Entity *a)** que recibe una entidad representando un asteroide que haya chocado con una bala, lo desactiva y genera otros **2** asteroides dependiendo de su número de generaciones *(más información en el apéndice)*. Recuerda que no pueden existir más de **30** asteroides a la vez en la pantalla.
3. **GameCtrl:** Si el jugador pulsa la tecla **SPACE,** si el estado es distinto de **RUNNING,** cambia el estado del juego (como el **Ping Pong**). Al empezar una ronda, es decir cuando el cambia a `RUNNING,` hay que añadir `10` asteroides al juego (pedírselo al **AsteroidsManager**).

Detectar Colisiones

Para comprobar colisiones implementa un método **checkCollisison()** que comprueba choques con todos los asteroide activos, usando el método **Collisions::collidesWithRotation**, de la siguiente manera:

a. Si choca 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 componente **`AsteroidsManager`** 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 **`0.0f`**. Recuerda que entre rondas, sólo caza aparece en la pantalla.

APÉNDICE

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. Recuerda que el caza puede mirar hacías una parte y

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. **`0.2f`**). Además, si la magnitud de **"`newVel`"** supera un límite **"`speedLimit`"** (usa p.ej., **`3.0f`**) modifícalo para que tenga la magnitud igual a **"`speedLimit`"**, es decir modifícalo a **"`newVel.normalize()*speedLimit`"**.

Calcular la posición, dirección, rotación, y tamaño de la bala

Sea **"`pos`"** la posición del caza, **"`vel`"** su vector de velocidad, **"`r`"** su rotación, **"`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 de **`5.0f`** de anchura y **`20.0f`** de altura, y la rotación es la misma que la del caza.

Crear un asteroid

Al crear un nuevo asteroide, hay que asignarle una 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`. El vector de velocidad **v** sería

```
float speed = sdlutils().rand().nextInt(1,10)/10.0f;
Vector2D v = (c-p).normalize()*speed;
```

El número de generaciones del asteroide es un número entero aleatorio entre `1` y `3`. 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.

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)`"**.

Cómo dividir un asteroide

En el método **`onCollision(Entity *a)`**, 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 tiene que ser 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"** su anchura, "h" su altura, se puede usar el siguiente código para calcular la posición y velocidad de cada asteroide nuevo:

```
auto r = sdlutils().rand().nextInt(0,360);
auto pos = p + v.rotate(r) * 2 * std::max(w,h).
auto vel = v.rotate(r) * 1.1f
```

Recuerda que la anchura y altura del nuevo asteroide dependen de su número de generaciones.