TP Videojuegos 2

Práctica 2

Fecha Límite: 10/05/2021 a las 09:00.

En esta práctica vamos a desarrollar el juego Asteroids usando el patrón de diseño ECS donde los componentes tienen sólo datos y los comportamientos están definidos en sistemas. **Hay que usar `ecs_4.zip` o `ecs_6.zip`**. A continuación proponemos un diseño de sistemas y componentes que podéis usar. **Es sólo una recomendación, se puede usar otro diseño**. El diseño está hecho para comunicación directa entre sistemas, sin usar mensajes, pero se puede hacer usando mensajes para algunas/todas partes de la comunicación (***si no lo haces, es muy recomendable practicar cómo hacerlo después de entregar la práctica porque este tema está incluido en el examen***).

Propuesta de diseño de sistemas y componentes

Los componentes son como en la práctica 1, pero mejor no usar `Image` y decidir en el sistema correspondiente qué imagen usar (por lo menos para las balas y asteroides). Para distinguir entre los tipos de asteroides se puede usar un nuevo componente `AsteroidType` que incluye un atributo para indicar el tipo (en la práctica 1 había dos tipos de asteroides).

GameCtrlSystem

```
class GameCtrlSystem: public System {
public:
	// - a este método se le va a llamar cuando muere el caza.
	// - desactivar los asteroides y las balas, actualizar el estado del juego, etc.
	void onFighterDeath() …

	// - a este método se le va a llamar cuando no haya más asteroides.
	// - desactivar todas las balas, etc.
	void onAsteroidsExtinction() …

	// - devuelve el estado del juego.
	GameState getGameState() …

	// - inicializar el estado del juego si es necesario, etc.
	void init() override …

	// - si el juego está parado y el jugador pulsa SDLK_SPACE cambia el estado como
	//   en la práctica 1, etc.
	void update() override …
}
```

TP Videojuegos 2, Práctica 2, Fecha Límite: 10/05/2021 a las 09:00.

AsteroidsSystem

```
class AsteroidsSystem: public System {
public:
	// - añade n asteroides al juego como en la práctica 1.
	void addAsteroids(int n) …

	// - desactivar el asteroide “a” y crear 2 asteroides como en la práctica 1.
	// - pasamos una referencia a la bala aunque no se usa de momento (en el futuro
	//   se puede usar para tener comportamientos distintos depende del tipo de
	//   bala, etc.)
	// - si no hay más asteroides avisar al GameCtrlSystem
	void onCollisionWithBullet(Entity *a, Entity *b) …

	// - si el juego está parado no hacer nada.
	// - mover los asteroides como en la práctica 1.
	void update() override …
private:
	std::size_t numOfAsteroids_;
}
```

BulletsSystem

```
class BulletsSystem: public System {
public:
	// - añadir una bala al juego, como en la práctica 1. La rotación de la bala
	//   sería vel.angle(Vector2D(0.0f,-1.0f))
	void shoot(Vector2D pos, Vector2D vel, double width, double height) …

	// - desactivar la bala “b”
	// - pasamos una referencia al asteroid aunque no se usa de momento (en el futuro
	//   se puede usar para tener comportamientos distintos depende del tipo de
	//   asteroid, etc).
	void onCollisionWithAsteroid(Entity *b, Entity *a) …

	// - si el juego está parado no hacer nada.
	// - mover las balas y desactivar las que se salen de la ventana
	void update() override …
}
```

TP Videojuegos 2, Práctica 2, Fecha Límite: 10/05/2021 a las 09:00.

FighterSystem

```
class FighterSystem: public System {
public:
	// - poner el caza en el centro con velocidad 0 y rotación 0. No hace falta
	//   desactivar la entidad (no dibujarla si el juego está parado en RenderSystem).
	// - avisar al GameCtrlSystem que se ha muerto el caza (se puede también avisar
	//   directamente en el CollisionSystem)
	void onCollisionWithAsteroid(Entity *a) …

	// - crear la entidad del caza, añadir sus componentes (Transform, Health, etc.)
	//   y asociarla con el handler correspondiente.
	void init() override …

	// - si el juego está parado no hacer nada.
	// - actualizar la velocidad del caza y moverlo como en la práctica 1.
	void update() override …
}
```

FighterGunSystem

```
class FighterGunSystem: public System {
public:
	// - si el juego está parado no hacer nada. Si el jugador pulsa SDLK_S, llamar
	//   a shoot del BulletsSystem para añadir una bala al juego, etc.
	// - se puede incluir ese comportamiento en el FighterSystem directamente en
	//   lugar de definirlo en un sistema separado
	void update() override …
}
```

CollisionSystem

```
class CollisionSystem: public System {
public:
	// - si el juego está parado no hacer nada.
	// - comprobar colisiones como en la práctica 1 y avisar a los sistemas
	//   correspondientes en caso de colisiones
	void update() override …
}
```

TP Videojuegos 2, Práctica 2, Fecha Límite: 10/05/2021 a las 09:00.

RENDERSYSTEM

```
class RenderSystem: public System {
public:
	// - dibujar asteroides, balas y caza (sólo si el juego no está parado).
	// - dibujar el marcador y las vidas (siempre).
	// - dibujar el mensaje correspondiente si el juego está parado (como en la
	//   práctica 1)
	// - Otra posibilidad es definir un método render en la clase System, y distribuir
	//   este código en los sistemas correspondientes en lugar de tener un sólo
	//   sistema para rendering
	void update() override ...
}
```