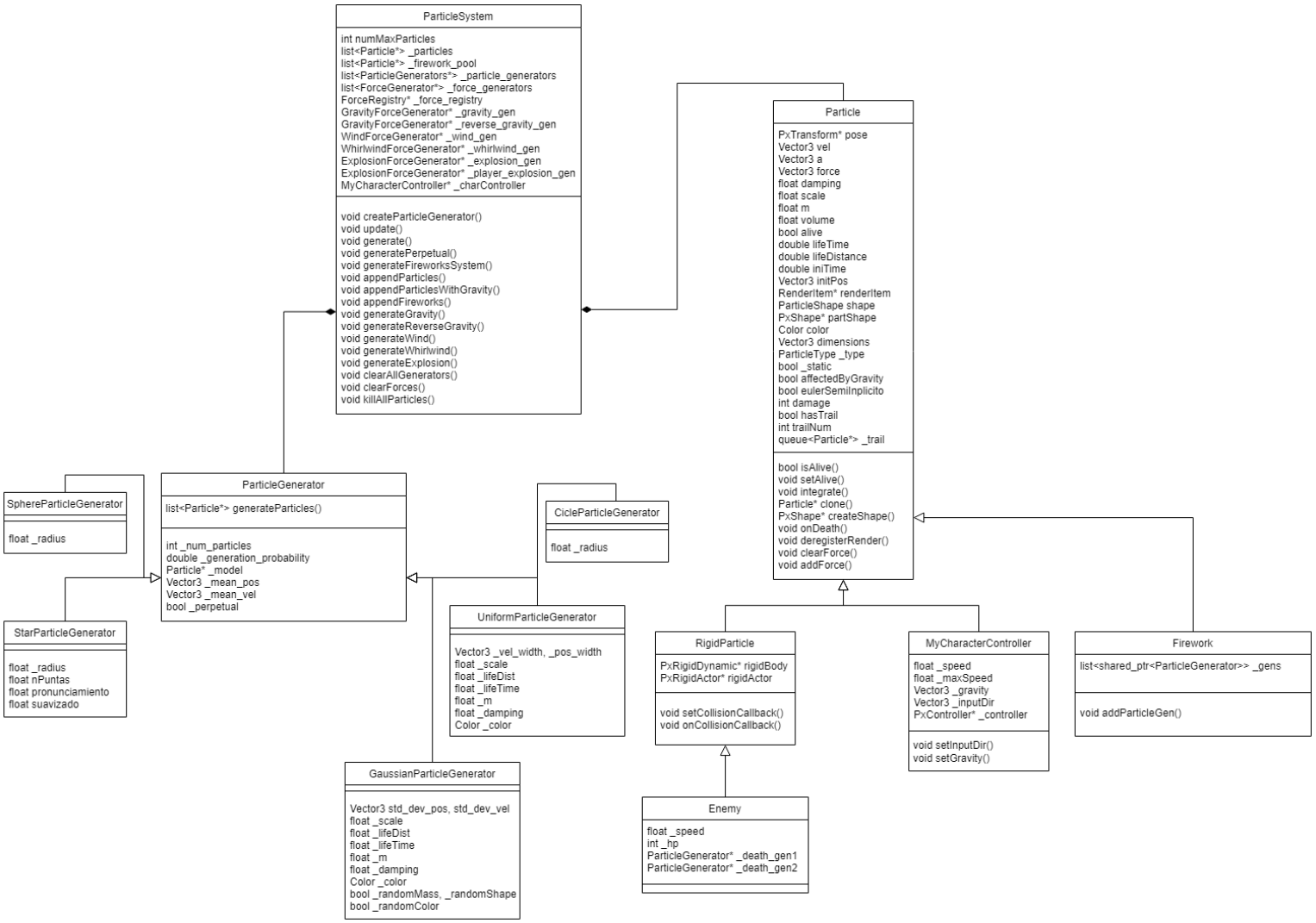
Memoria trabajo final de Simulación Física para Videojuegos

1. Temática:

El proyecto consiste en un shooter básico, aprovechando el motor físico basado en rigid bodies que hemos realizado en la asignatura.

2. Diagrama de clases:



3. Efectos y ecuaciones usadas:

A lo largo del proyecto he implementado diversos objetos, las ecuaciones se ajustan a la naturaleza de cada uno.

Particle

Las partículas utilizan el integrate visto en clase. Las partículas rígidas añaden además los ajustes necesarios para actualizar la posición del rigidBody.

Generadores de fuerzas

En el juego como tal solo se emplean 2 generadores de fuerzas, el de gravedad y explosiones, sin embargo los de todas las práticas anteriores son utilizables y funcionales.

Para la gravedad se emplea un valor de g que varía entre objetos, por ejemplo el jugador tiene un valor de -100 (por motivos relacionados con el input), el ParticleSystem utiliza para sus partículas -10 por defecto y las balas utilizan un valor u otro dependiendo de sus parámetros al ser creadas.

Las explosiones utilizan un radio de explosión de 10 y una k de 10.000 para las partículas del ParticleSystem y 20.000 para el jugador.

Jugador

El jugador es una partícula que contiene la cámara.

Para moverse usa la dirección de la cámara y la dirección del input. El input no se bloquea al pulsar varias teclas a la vez, esto se consigue mediante el uso de un array de booleanos para controlar las teclas pulsadas o no pulsadas.

Se combina el uso de estos parámetros junto con el integrate básico de la partícula. Finalmente se actualiza la posición del CharacterController con el método move() y la posición de la cámara con el transform de este.

```
PxVec3 viewY = _cam->getDir().cross(PxVec3(0, 1, 0)).getNormalized();
PxVec3 viewX = -viewY.cross(PxVec3(0, 1, 0)).getNormalized();

dir = viewX * _inputDir.x + viewY * _inputDir.y;

vel += dir.getNormalized() * _speed;

PxVec2 horizonalVel = PxVec2(vel.x, vel.z);
if (horizonalVel.magnitude() > _maxSpeed)
    horizonalVel = horizonalVel.getNormalized() * _maxSpeed;

vel = Vector3(horizonalVel.x, vel.y, horizonalVel.y);
```

Dirección de la cámara e input

Colisiones

El juego utiliza las colisiones que implementan por defecto los cuerpos rígidos en physx, pero además define un shader de colisión para gestionar las interacciones entre objetos del mundo.

Para esto, cada RigidParticle posee un collisionCallback, que es una lambda function que se invoca al colisionar con otro objeto. El shader de colisiones se limita a llamar a los callbacks de los objetos que han colisionado, así solo es necesario definir dicho callback en cada partícula, no implementar un nuevo shader para cada una o llenar el actual de código.

Armas y disparos

El juego implementa 5 tipos de armas con su propia munición.

El arma es una partícula que aparece en pantalla centrada en la posición de la cámara, también existe una partícula especial llamada crosshair, que indica la posición a la que va a salir el proyectil, como una mirilla.

Al disparar el arma genera partículas emulando las chispas de la explosión en el cañón del arma.

Las balas se generan en un sistema especial llamado BulletSystem, el cual crea cada una según su tipo de munición, además, gestiona los cooldowns entre disparos y la vida de las balas.

Las balas, al ser RigidParticles tienen un collisionCallback. Cada tipo de munición implementa un collisionCallback para conseguir el efecto deseado:

- La bala normal y la bala de cañón desaparecen al colisionar, también disminuye la vida del enemigo golpeado si colisiona con uno.

- El láser no desaparece al colisionar con un muro o el suelo, pero sí al colisionar con un enemigo o pasado un tiempo, además tiene una estela.

- El lanzacohetes genera una explosión al colisionar, que afecta al jugador, al ser este una partícula (se puede utilizar como método de salto).

- El lanzacohetes de fuegos artificiales genera un fuego artificial aleatorio al colisionar, siendo posibles círculos, estrellas o esferas.

Muelles

Se han implementado 2 slinky que tienen comportamientos distintos, uno es un sistema caótico creado en horizontal y que debido a sus propias fuerzas se vuelve caótico. Al lado se encuentra un slinky estándar para comparar.

Estos tienen una k de 100 (facilita que sea caótico) y distancia en reposo de 0.5.