SUBWAY SURFERS ***PERO MAL***

Objetivo y bases:

El proyecto realizado es un runner procedural. Donde el jugador deberá llegar al final del mapa superando las diferentes pruebas generadas de forma aleatoria sin ser golpeado o salirse de la pista; en cuyo caso deberá de volver a comenzar desde el principio.
El jugador es una pelota y puede desplazarse en las cuatro direcciones rotando sobre sí mismo. Además de los diferentes obstáculos diferentes pueden afectar sobre él diferentes fuerzas que tendrá que tener en cuenta a la hora de avanzar.
Los objetos sólidos desplazarán la player pudiendo hacerle caer, las partículas; si golpean al player, harán que este vuelva a empezar desde cero; al igual que si cae al vacío.

Referencias:

Para crear el juego me he basado en los diferentes runners de móvil. Cambiando los 3 carriles pasando de un lado a otro por un movimiento libre que dé más libertad al jugador y dando, por la asignatura, mucha más importancia a los componentes físicos.

Elementos y clases utilizadas:

*Se señalizarán aquellos cambios realizados sobre el proyecto realizado en la última práctica. Omitiendo aquellos campos no modificados, o modificados muy limitadamente. Aquellos elementos marcados con un * son clases totalmente nuevas creadas para la práctica final.*

- Sólidos rígidos:

1. Clase World Manager para la gestión de los sólidos-rígidos, interacciones y mantener un registro sobre ellos.
    - Implementado método GenerateFloor para generar una superficie como base de las diferentes zonas.
    - Implementados métodos para generar las diferentes zonas del mapa, sus obstáculos y generadores. Algunos de estos llaman al Particle System para los basados en partículas.
    - Implementado método GenerateLevel para generar un mapa aleatorio usando los métodos anteriores.
    - Añadidas modificaciones para el control mediante un RigidBodyForceRegistry y listas de generadores, fuerzas…
    - Control de la posición del player y otros valores para reubicarlo o lanzar el evento de victoria una vez alcance la distancia determinada.

2. * Clase RigidBodyForceRegistry usada como apoyo para el WorldManager con el que gestionar y controlar los diferentes cuerpos dinámicos generados. Usa los métodos del ParticleForceRegistry adaptados.

3. Clase RigidBodyForceGenerator modificado para que en la misma generación de objetos se les asigne un generador de fuerzas asociado a dicho generador.

4. UniformRigidBodyForceGenerator modificado para cumplir las especificaciones nuevas del anterior.

5. * Clase StaticRigidBodyForceGenerator para generar objetos dinámicos sin valor de aleatoriedad. Usado en secciones donde se quieran crear siguiendo un patrón determinado.

6. * Clase HorizontalForceGenerator para ejercer fuerza sobre los objetos dinámicos en una dirección determinada horizontalmente.

7. Clase RotationGenerator modificada para adaptarse a la nueva estructura pero con la misma función.

8. * Clase PlayerController. Objeto sólido que controla directamente el jugador y sobre el que se basa la experiencia.
    - Métodos get y set para obtener y modificar diferentes atributos del jugador.

- Métodos de actualizar fuerzas, añadirlas y de colisión para su interacción con el mundo.
- Diferentes variables almacenadas para el correcto funcionamiento de estos métodos.
- Constructor para instanciar el cuerpo del jugador al inicio del nivel y almacenar todos los datos pertinentes.

- Partículas y fireworks

1. Clase ParticleSystem modificada para los nuevos requisitos y su relación con el WorldManager:
    - Método CheckParticlePlayerCollision para comprobar colisiones entre el jugador y todas aquellas partículas que pueden afectarle.
    - Implementados métodos para la generación de zonas solicitadas por el WorldManager, creando los generadores y fuerzas de partículas pertinentes.
    - Nuevas plantillas de partículas y generadores para las diferentes áreas que se pueden dar.
    - Método LaunchFireWorksWin que es llamado por el WorldManager cuando el jugador alcanza la meta que genera una serie de fuegos artificiales en diferentes puntos en torno a la posición de victoria.
    - Método UpdateTrailPlayer que actualiza un sistema de colisiones que hace las veces de estela del jugador.
    -
2. Clase GaussianForceGenerator modificada para las nuevas necesidades:
    - Poder pasar generadores de fuerza para que estos sean asignados automáticamente al crear las nuevas partículas.
    - Diferentes Set diferentes para cambiar la configuración y que el random solo ocurra en un eje, diferentes valores a la hora de aleatoriedad…

3. Clase ExplosionGenerator y FloatGenerator modificadas para que también afecten a los RigidBodies. La primera con un propósito más general y la segunda para que el jugador pueda flotar en su zona.

- Generales

1. Modificación del constructor main para que se llamen los diferentes managers en el orden correcto así como los inicializadores de la cámara y otros componentes.

2. Modificación del keyPress y relacionados para que WASD llame al PlayerController y modificar sus fuerzas.

3. Modificación de la cámara para que no se modifique con el teclado y métodos para poder modificar su posición externamente.

4. Modificación de StepPhysics en main para que actualice constantemente la cámara respecto a la posición del player.

Ecuaciones físicas usadas:

Se han mantenido las ecuaciones físicas usadas durante la práctica. Se señalizarán los de los diferentes generadores de fuerzas:

1. Partículas:

- Explosion: fuerza como la vista en clase

```
if (!enabled)
    return;

const double euler = std::exp(1.0);
auto pos = particle->getPos();
auto difX = pos.x - point.x;
auto difY = pos.y - point.y;
auto difZ = pos.z - point.z;

auto r2 = pow(difX, 2) + pow(difY, 2) + pow(difZ, 2);

if (r2 > pow(R, 2))
{
    return;
}

auto x = (k / r2) * difX * pow(euler, (-t / w));
auto y = (k / r2) * difY * pow(euler, (-t / w));
auto z = (k / r2) * difZ * pow(euler, (-t / w));

Vector3 force(x, y, z);


particle->addForce(force);
```

```
const double euler = std::exp(1.0);
auto pos = rigid->getGlobalPose().p;
auto difX = pos.x - point.x;
auto difY = pos.y - point.y;
auto difZ = pos.z - point.z;

auto r2 = pow(difX, 2) + pow(difY, 2) + pow(difZ, 2);

if (r2 > pow(R, 2))
{
    return;
}

auto x = (k / r2) * difX * pow(euler, (-t / w));
auto y = (k / r2) * difY * pow(euler, (-t / w));
auto z = (k / r2) * difZ * pow(euler, (-t / w));

Vector3 force(x, y, z);


rigid->addForce(force);
```

- GravityForceGenerator: fuerza como la vista en clase

```
if (!enabled) return;

if (fabs(particle->getInverseMass()) < 1e-10)
    return;

particle->addForce(_gravity * particle->getMass());
```

- UniformWindGenerator: fuerza como la vista en clase

```
auto p = particle->getPos();

Vector3 v = particle->getVel() - air;
float drag_coef = v.normalize();
Vector3 dragF;
drag_coef = (_k1 * drag_coef) + _k2 * drag_coef * drag_coef;
dragF = -v * drag_coef;


particle->addForce(dragF);
```

-

- Float generator: fuerza como la vista en clase

```
float h = p->getPos().y;
float h0 = _liquid_particle->getPos().y;

Vector3 f(0, 0, 0);
float immersed = 0.0;

if (h - h0 > _height * 0.5)
{
    immersed = 0.0;
}
else if (h0 - h > _height * 0.5) {
    immersed = 1.0;
}
else
{
    immersed = (h0 - h) / _height + 0.5;
}

f.y = _liquid_density * p->getVolume() * immersed * 9.8;

p->addForce(f);
```

```
float h = rigid->getGlobalPose().p.y;
float h0 = _liquid_particle->getPos().y;

Vector3 f(0, 0, 0);
float immersed = 0.0;

if (h - h0 > _height * 0.5)
{
    immersed = 0.0;
}
else if (h0 - h > _height * 0.5) {
    immersed = 1.0;
}
else
{
    immersed = (h0 - h) / _height + 0.5;
}

f.y = _liquid_density * _volume * immersed * 9.8;

rigid->addForce(f/4);
```

- SpringForceGenerator: fuerza como la vista en clase

```
Vector3 force = _other->getPos() - p->getPos();

const float lenght = force.normalize();

const float delta_x = lenght - _resting_length;

force *= delta_x * _k;

p->addForce(force);
```

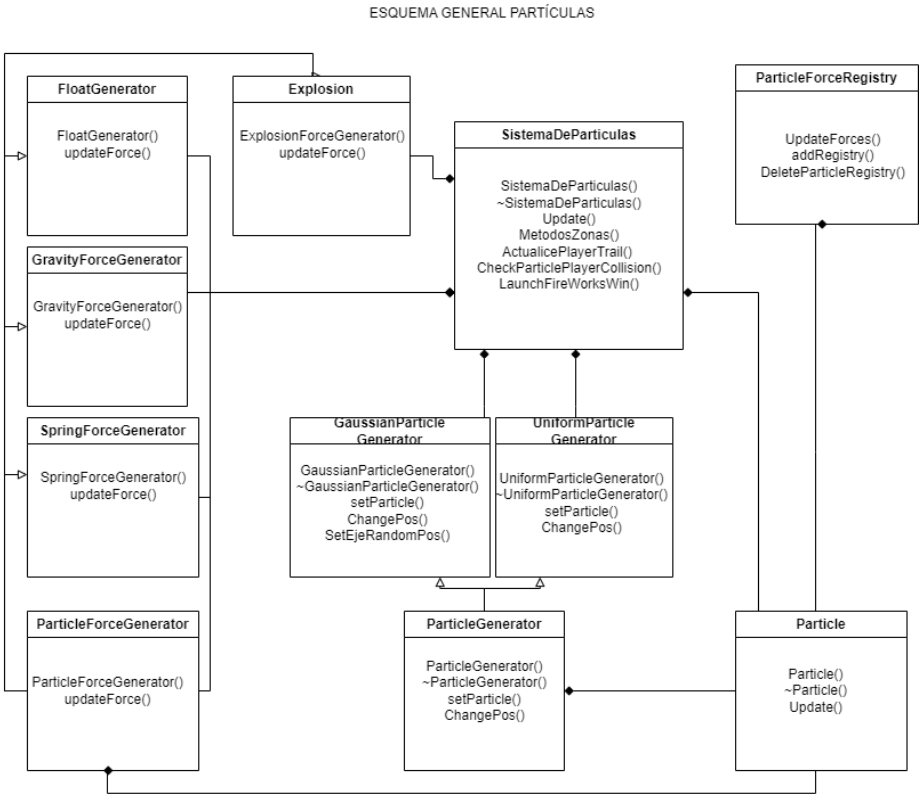
- RotationForceGenerator: genera una fuerza centrífuga a todos los objetos sólidos en un área.

```
auto pos = rigidBody->getGlobalPose().p;

auto difX = pos.x - point.x;
auto difY = pos.y - point.y;
auto difZ = pos.z - point.z;

auto r2 = pow(difX, 2) + pow(difY, 2) + pow(difZ, 2);

if (r2 > R*R)
    return;

//puedes añadirle otro vector que mire para el centro
rigidBody->addTorque(Vector3(difX,difY, difZ) * k);
```

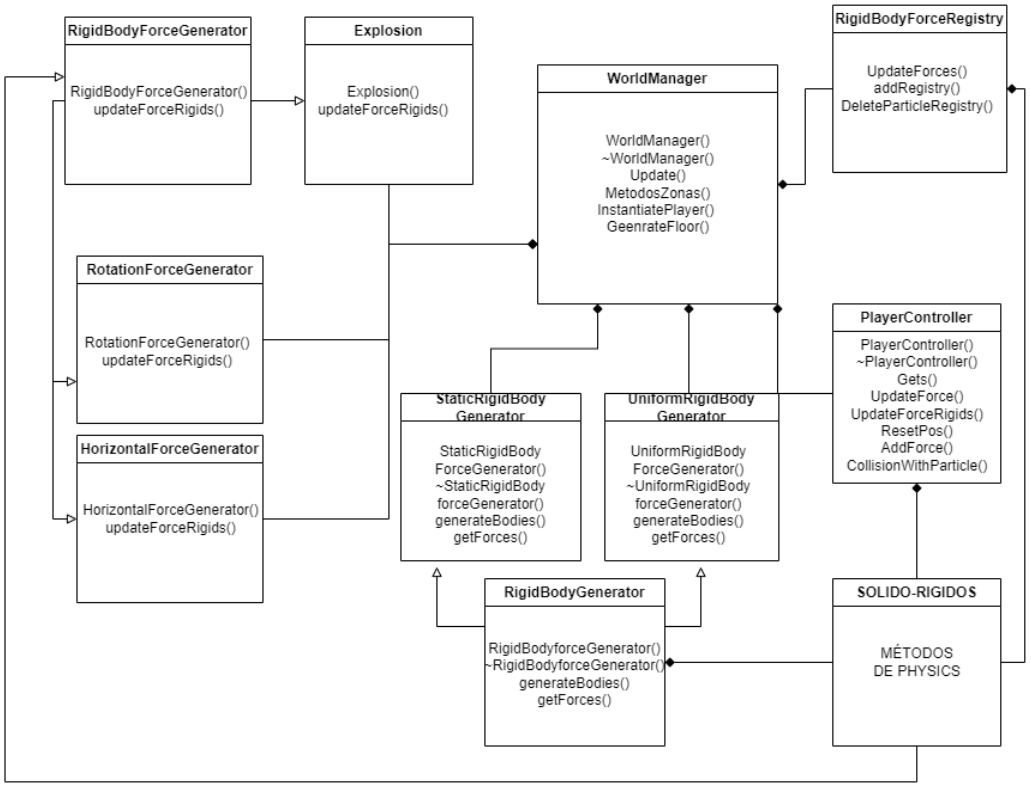
- HorizontalForceGenerator: genera una fuerza lateral continua que invierte su dirección cada x tiempo.

```
currentTime += t;

if (currentTime > time + lastChange)
{
    lastChange = currentTime;
    left = !left;
}

if (time == lastChange)
    rigidBody->setLinearVelocity({ 0,0,0 });


if (left)
    rigidBody->addForce(Vector3(-1, 0, 0) * k * rigidBody->getMass());
else
    rigidBody->addForce(Vector3(1, 0, 0) * k * rigidBody->getMass());
```

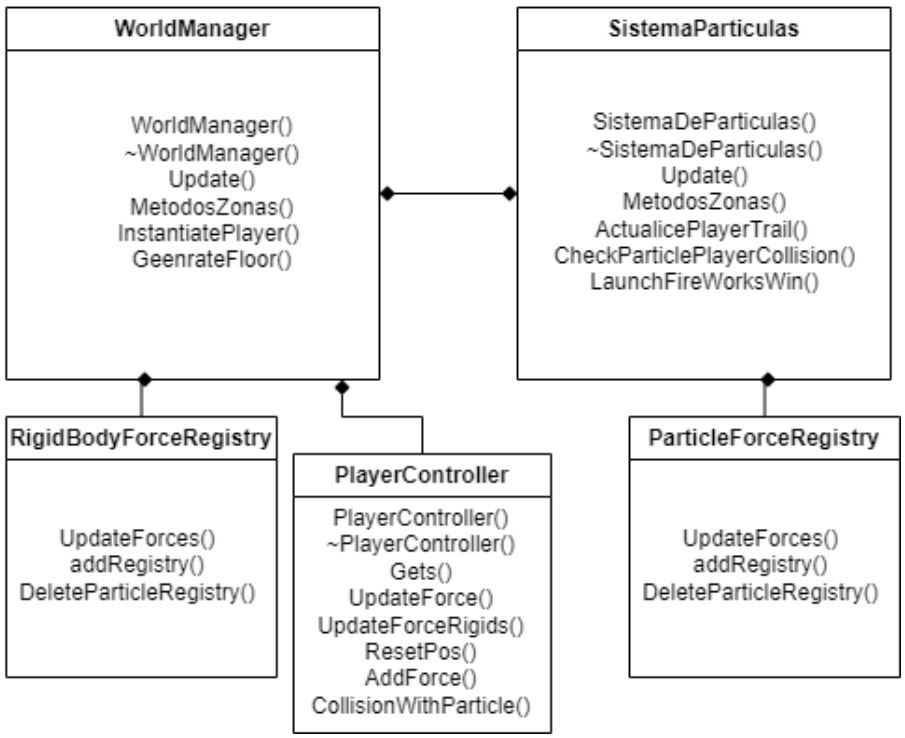
Diagramas de clases:



ESQUEMA GENERAL LOS SÓLIDOS-RÍGIDOS



ESQUEMA GENERAL DE LA RELACIÓN ENTRE
DINÁMICOS Y PARTÍCULAS



Elementos añadidos:

-Player: esfera sólida controlable por el jugador, capaz de colisionar con objetos sólidos y partículas. Es seguido por la cámara.

-2 nuevas combinaciones de fireworks para ser lanzados al completar el juego.

-1 nuevo tipo de generador de rigidbodies StaticRigidBodyGenerator estático para poder crear formas determinadas.

-1 nuevo tipo de fuerza HorizontalForceGenerator para los rigidbodies.

-6 combinaciones diferentes de zonas por las que el jugador debe pasar:

1. BouncyZone
2. RotationZone
3. HorizontalWallZone
4. CanonZone
5. Dragzone
6. WaterZone

Cada una con sus propias características pero siguiendo un patrón común que permiten expandir su número fácilmente.

-Generador procedural en el worldManager fácilmente ajustable.

Tener en cuenta que el nivel predefinido es uno donde siempre se empieza con BouncyZone y luego se va alternando entre una forma al azar y una HorizontalWallZone. Sumando un total de 10 zonas hasta la meta. Está basado en mi experiencia pero es fácilmente modificable.

Controles:

Básicos

-WASD para desplazarse.

-Si se cae al vacío se resetea a la posición inicial.

-Si el jugador es golpeado por algunas partículas se resetea a la posición inicial.

Para testear

-Se puede cambiar el número de fases en el WorldManager en el GenerateLevel modificando el número de plataformas, así como la inicial y el final (GenerateFloor); se ruega no modificar value o currentValue para no crear sobreexposición de elementos. Si se quieren ir probando diferentes secciones por separado se puede realizar de la siguiente manera:

1. generateFloor({ 0,0, 0 });
2. generateXType({0,0,value)};
3.

-Si se quiere reducir la distancia de victoria se puede modificar cambiando la variable winZ en el método GenerateLevel del WorldManager