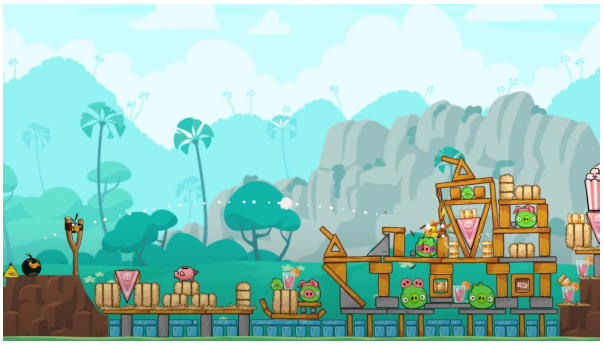
Elisa Todd Rodríguez

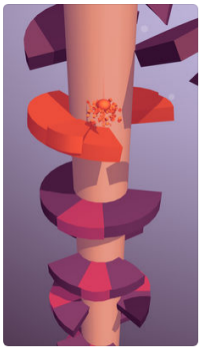
PROYECTO FINAL SIMULACIÓN FÍSICA DE VIDEOJUEGOS

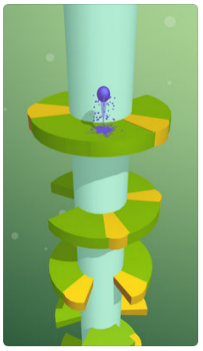
Para el proyecto de la final de la asignatura he desarrollado un juego en el que se controla a un jugador en forma de pelota, al que desplazamos por el escenario mediante saltos, con el objetivo de llegar a una meta o final.

El mini-juego está inspirado en los juegos móviles con la misma dinámica, en los que controlamos diferentes formas geométricas (usualmente pelotas) que van rebotando por el mundo de juego según una trayectoria inicial que le dicta el jugador.

Otra de las inspiraciones principales del juego es Angry Birds, por la forma en la que se ve representada la trayectoria del jugador en los momentos previos a su lanzamiento:



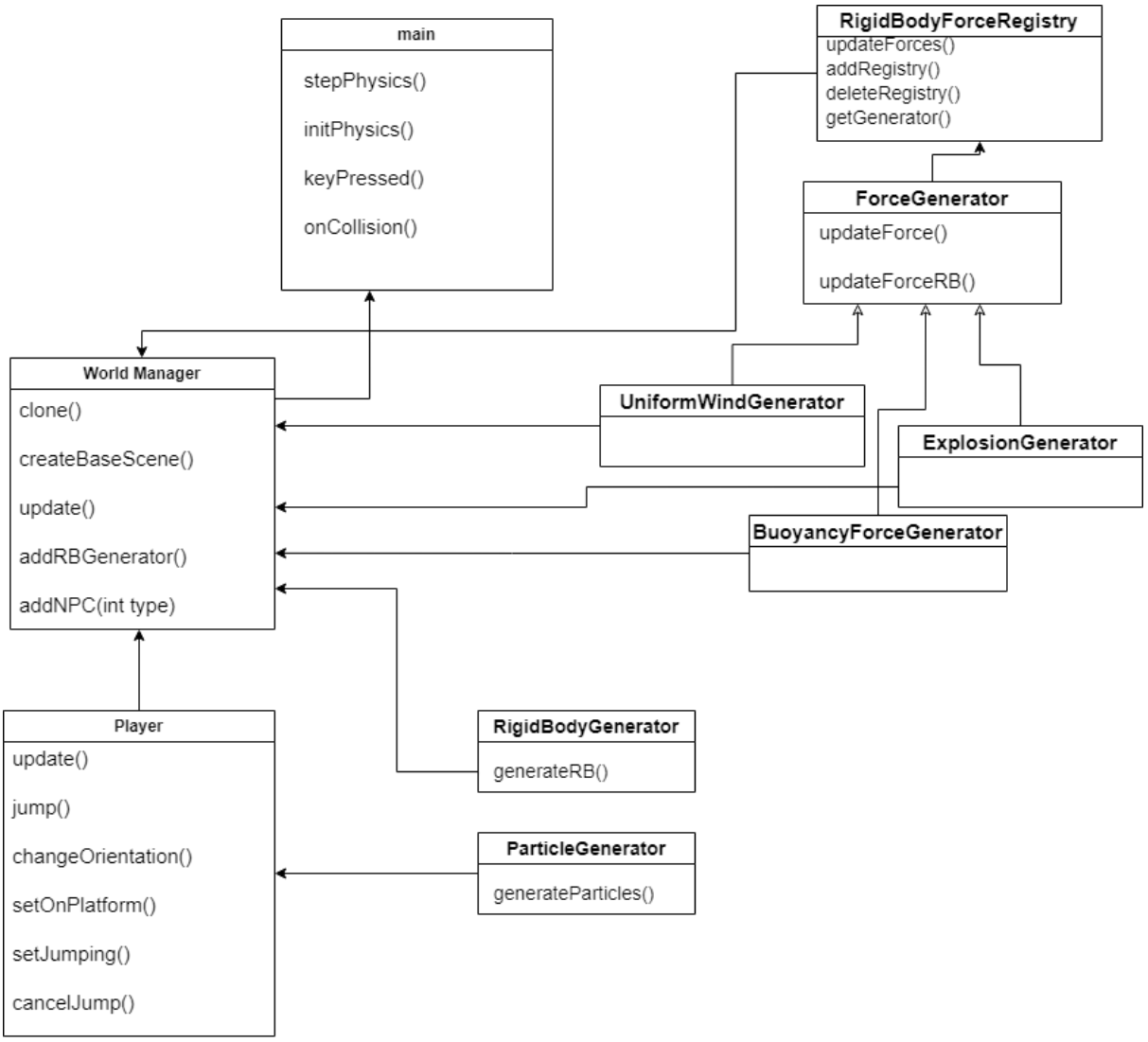




Los niveles desarrollados consisten en subir por una serie de plataformas hasta llegar a la que se encuentra por encima de todas, siendo esta la meta.
A medida que vaya subiendo, encontrará varias adversidades en el camino que le impedirán el paso o dificultarán el camino, obligando al jugador a buscar una forma alternativa para subir.

Además, si el jugador falla en sus saltos y se cae de las plataformas, caerá a una masa de agua, desde la cual se le transportará a la plataforma de inicio pasados unos segundos.

Diagrama de clases:



Esquema general del código, éste tiene más clases secundarias que especifican el comportamiento del juego y varios métodos más para modificar variables.

Además, hay un particle system que se ocupa de los fireworks, instanciado en el main().

Como el resto de los objetos de la escena son RigidBodies, son controlados desde WorldManager.

Ecuaciones físicas usadas:

Se han usado ecuaciones físicas en el juego a la hora de implementar generadores de fuerzas sobre los Rigid Bodies.

1- Explosión:

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

//double r = sqrt(pow((pos.x - _centre.x), 2) + pow((pos.y - _centre.y), 2) + pow((pos.z - _centre.z), 2));
auto difX = pos.x - _centre.x;
auto difY = pos.y - _centre.y;
auto difZ = pos.z - _centre.z;
auto r = sqrt(pow(difX, 2) + pow(difY, 2) + pow(difZ, 2));
Vector3 forceDir = { 0, 0, 0 };

if (r <= _R) {
    double first = _K / pow(r, 2);
    double power = -(t / _const);
    double second = pow(e, power);

    forceDir = first * Vector3(difX, difY, difZ) * second;
}

Vector3 v = rb->getAngularVelocity() - forceDir;
//  Vector3 v = rb->getLinearVelocity() - forceDir;
float coef = v.normalize();
Vector3 expForce;
coef = (_K * coef) + _K * coef * coef;
expForce = -v * coef;

//  std::cout << "Explosion force: " << expForce.x << "\t" << expForce.y << "\t" << expForce.z << "\n";

expForce.z = 0.0; // para simular 2D
rb->addForce(expForce);
```

2- Flotación:

```
        F.y =
_liquid_density * _volume
* immersed;
        rb->addForce(F);
```

Uso para la flotación la densidad del agua, y el volumen del objeto se calcula mirando sus características:

```
// Miro la forma que hay asociada a este RB
physx::PxShape* s;
physx::PxU32 u = rb->getNbShapes();
rb->getShapes(&s, u);

// Miro su tamaño para calcular altura y densidad

auto type = s->getGeometryType();
switch (type) {
case physx::PxGeometryType::eBOX:
{
    physx::PxBoxGeometry box;
    s->getBoxGeometry(box);
    _height = box.halfExtents.y * 2;
    // volumen = l * w * h
    _volume = (box.halfExtents.x * 2) * _height * (box.halfExtents.z * 2);

    break;
}
case physx::PxGeometryType::eSPHERE:
{
    physx::PxSphereGeometry sphere;
    s->getSphereGeometry(sphere);
    _height = sphere.radius * 2;
    // volumen = 4/3 * pi * radio^3
    _volume = (4 / 3) * pi * pow((_height / 2), 3);
    break;
}

}
```

3- Viento:

```
// Check if the particle is inside the mass of wind
auto pos = rb->getGlobalPose().p;
if (pos.x > origin.x + areaX || pos.x < origin.x - areaX ||
    pos.y > origin.y + areaY || pos.y < origin.y - areaY ||
    pos.z > origin.z + areaZ || pos.z < origin.z - areaZ)
    return;

// Compute the wind force == drag force + wind
Vector3 v = rb->getLinearVelocity() - _wind; // lo mismo que drag pero ahora sumamos el viento
float dragCoef = v.normalize();
Vector3 windForce;
dragCoef = _k1 * dragCoef + _k2 * dragCoef * dragCoef;
windForce = -v * dragCoef;

// apply the wind force
std::cout << "Wind force: " << windForce.x << "\t" << windForce.y << "\t" << windForce.z << "\n";
//std::cout << "Particle position: " << p->getPose().p.x << "\t" << p->getPose().p.y << "\t" << p-
rb->addForce(windForce);
```

Efectos incorporados

1. Hay una masa de agua bajo el jugador que imita un océano. En éste hay “animales” representados por cubos con diferente masa, que flotan.
   Además si el jugador cae al agua también flotará, y a los pocos segundos reaparecerá sobre la plataforma de inicio.
2. Al tocar el botón rojo de la escena, aparece una explosión alrededor de este que empujará al jugador.
3. Sobre la segunda plataforma del lado derecho, hay una ráfaga de viento invisible que empujará al jugador al agua si éste entra dentro de la masa de aire.
4. Hay un generador de rigid bodies que genera una fuente de pelotas con el objetivo de empujar al jugador, una vez más, hacia el agua.
5. Si el jugador consigue alcanzar la plataforma de la meta, saldrán fuegos artificiales.
6. Al prepararse para saltar, el jugador, genera una fila de partículas que marcan la trayectoria que seguirá cuando ejecute su salto, y esta cambiará según las teclas pulsadas.
7. Cuando efectúe el salto, el jugador se moverá según la trayectoria prevista por el punto anterior.

Controles

- SPACE para ver la trayectoria de salto.
- SPACE de nuevo, para ejecutar el salto.
- Z para salir de la vista de trayectoria.
- Q y E para cambiar la trayectoria del salto.