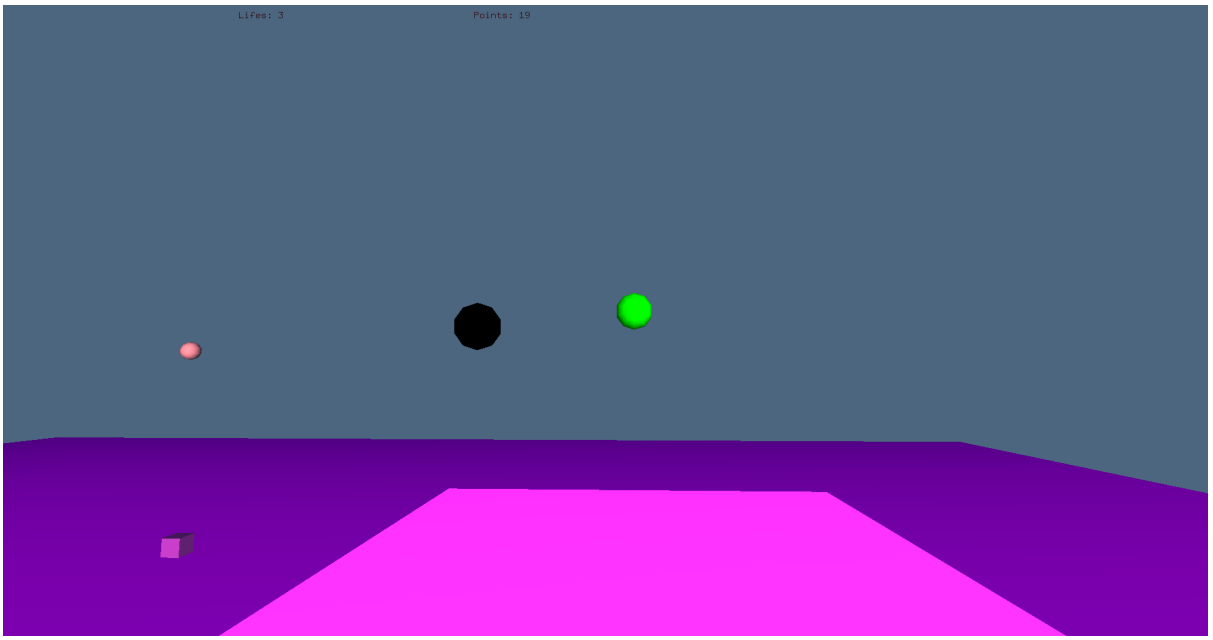
SIMULACIÓN FÍSICA PARA VIDEOJUEGOS
PROYECTO FINAL-SIMONA ANTONOVA MIHAYLOVA

¡PHYSX NINJA!

Mi proyecto final para la asignatura **Simulación Física para Videojuegos** es Physx Ninja, una adaptación del conocido juego Fruit Ninja , realizada en la librería PhysX de Nvidia.



En las partidas de Physx Ninja, el jugador debe disparar a las frutas que aparecen. Cada fruta aparece volando durante un tiempo limitado; si la fruta toca el suelo sin haber sido destruida, el jugador pierde una vida.

El jugador perderá un nivel si pierde todas sus vidas o si destruye una bomba (las bombas aparecen cada cierto tiempo, y no deben ser disparadas).

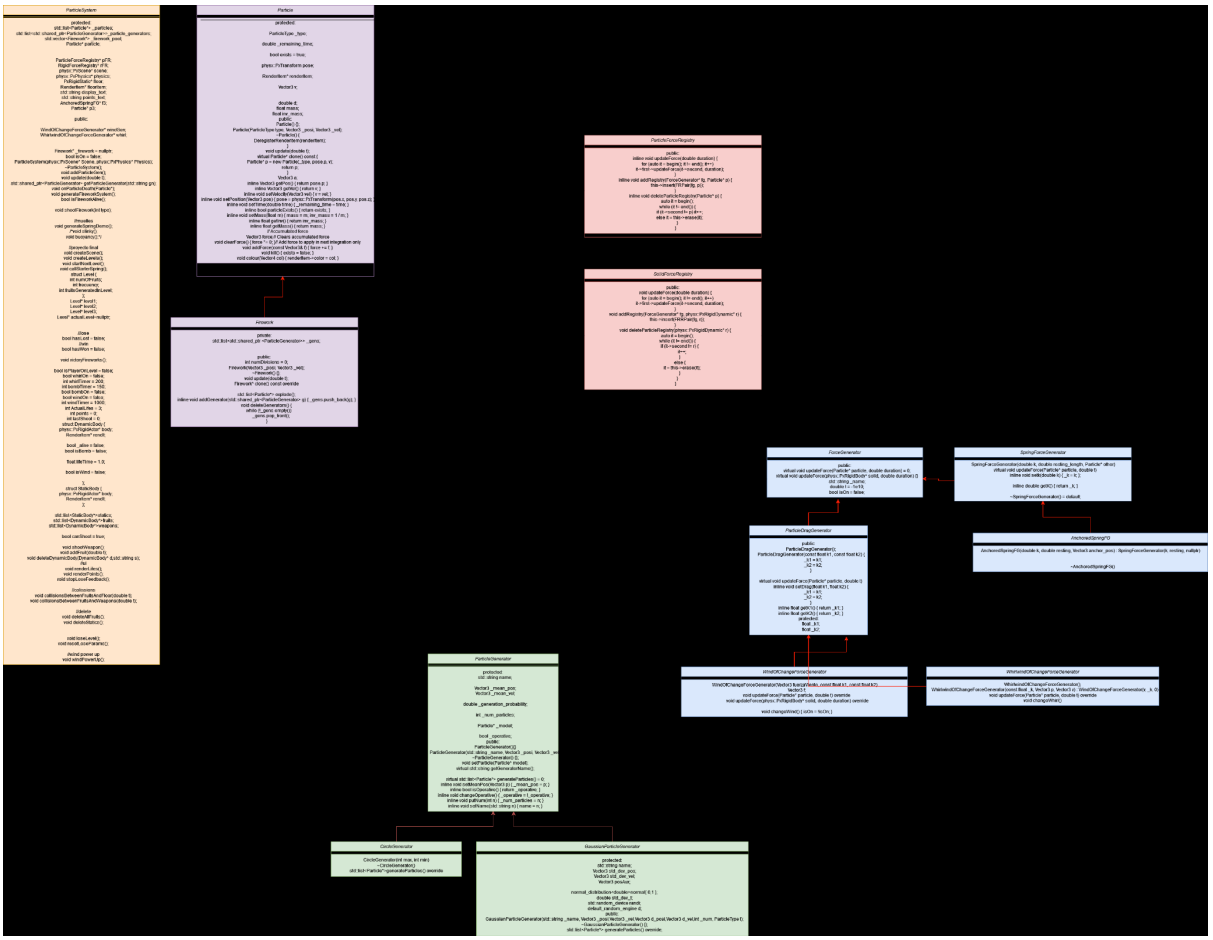
Para iniciar cada nivel, activaremos una fuerza de unión entre una partícula y un muelle, que se podrá visualizar en escena para saber que estamos inmersos en un nivel.

Existe un power-up de viento que podrá activarse una única vez durante la partida, que ayudará en la destrucción de las frutas.

Si pierdes un nivel, al volver a activar dicha fuerza se reiniciará, y si completas el último nivel, volverás de nuevo al primero, pudiendo jugar los 3 niveles en bucle.

-DIAGRAMA DE CLASES

No se aprecia muy bien, lo subire en GitHub para poder apreciarlo mejor.



-ECUACIONES FÍSICAS USADAS

Viento:

La fuerza ejercida por el viento dependerá de la diferencia de velocidades del viento con respecto a la velocidad del sólido en el punto. Lo utilizo como power-up para deshacerte de frutas,al activarlo, durante 5 segundos. El valor que le doy al generador de viento es el siguiente:

```
void WindForceGenerator::updateForce(physx::PxRigidBody* solid, double duration)
{
    if (fabs(solid->getInvMass()) < 1e-10) return;

    Vector3 v = vel - solid->getLinearVelocity();
    float drag_coef = v.normalize();
    Vector3 dragF;
    drag_coef = _k1 * drag_coef + _k2 * drag_coef * drag_coef;
    dragF = -v * drag_coef;

    solid->addForce(dragF);
}
```

El valor que le doy al generador de viento es el siguiente:

```
windGen = new WindOfChangeForceGenerator({ 80,0,0 }, 0.55, 0);//force, k1, k2
```

Torbellino:

El torbellino es considerado un campo vectorial de vientos, la velocidad del viento es proporcional a la distancia de la partícula al centro del torbellino, en dirección tangencial. Lo utilizo como feedback para el jugador para que sepa que ha perdido, con una duración de 2 segundos en escena y compuesto por partículas rojas.

```
void updateForce(Particle* particle, double t) override {
    if (fabs(particle->getInv()) < 1e-10) return;

    auto posi = particle->getPos();
    f = { k * (-(posi.z - whirlPosi.z)),k * (75 - (posi.y - whirlPosi.y)),k * (posi.x - whirlPosi.x) };

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

    particle->addForce(dragF);
}
```

El valor que le doy al generador de torbellinos es el siguiente

```
whirl = new WhirlwindOfChangeForceGenerator(7, { 50,50,-50 }, { 6,2,0 });//k, position, velocity
```

Partícula unida a una posición estática mediante un muelle:

La posición estática se representa mediante un pequeño cubo, para ver visualmente dónde está el punto de anclaje. Lo utilizo como “botón de inicio” de cada nivel, y le pongo una constante k no muy elevada para que se pueda apreciar en todo momento, ya que es feedback para el jugador.

```
virtual void updateForce(Particle* particle, double t) {
    Vector3 force = _other->getPos() - particle->getPos();

    const float length = force.normalize();
    const float delta_x = length - _resting_length;

    force *= delta_x * _k;

    particle->addForce(force);
}
```

El valor que le doy al generador de muelles es el siguiente:

```
f3 = new AnchoredSpringFG(20, 10, { 0.0,50.0,-50.0 });//k, resting length, pos
```

Colisiones:

En el juego compruebo las siguientes colisiones, todas mediante la función de physX **physx::PxGeometryQuery::overlap**

Armas-Frutas:

```
//Checks collisions between fruits and weapons
void GameManager::collisionsBetweenFruitsAndWeapons(double t) {
    auto itFruits = fruits.begin();
    auto it2 = weapons.begin();

    while (it2 != weapons.end()) {
        while (itFruits != fruits.end()) {
            if (physx::PxGeometryQuery::overlap((*it2)->rendIt->shape->getGeometry().sphere(), (*it2)->body->getGlobalPose(), (*itFruits)->rendIt->shape->getGeometry().sphere(), (*itFruits)->body->getGlobalPose())) {

                if ((*itFruits)->isBomb) {
                    //shootFirework(0);
                    auto model = new Firework({(*itFruits)->body->getGlobalPose().p, Vector3(0, 0, 0)};
                    getParticleGenerator("CIRCLE")->setParticle(model);

                    auto l = getParticleGenerator("CIRCLE")->generateParticles();
                    for (auto p : l) {
                        p->colour({ 1.0,0.0,0.0,0.0,1.0 });
                        _particles.push_back(p);
                    }
                    model->setPosition({ 8494,0,0 });
                    bombOn = true;
                    //ActualLifes = 0;
                    deleteDynamicBody(*itFruits, "f");
                    itFruits = fruits.erase(itFruits);
                }
                else
                {
                    points++;
                    deleteDynamicBody(*itFruits, "f");
                    itFruits = fruits.erase(itFruits);
                }

            }
            else itFruits++;
        }
        (*it2)->lifeTime -= t;
        if ((*it2)->lifeTime <= 0) {

            deleteDynamicBody(*it2, "w");
            it2 = weapons.erase(it2);
        }

        else it2++;
    }
}
```

Suelo-Frutas

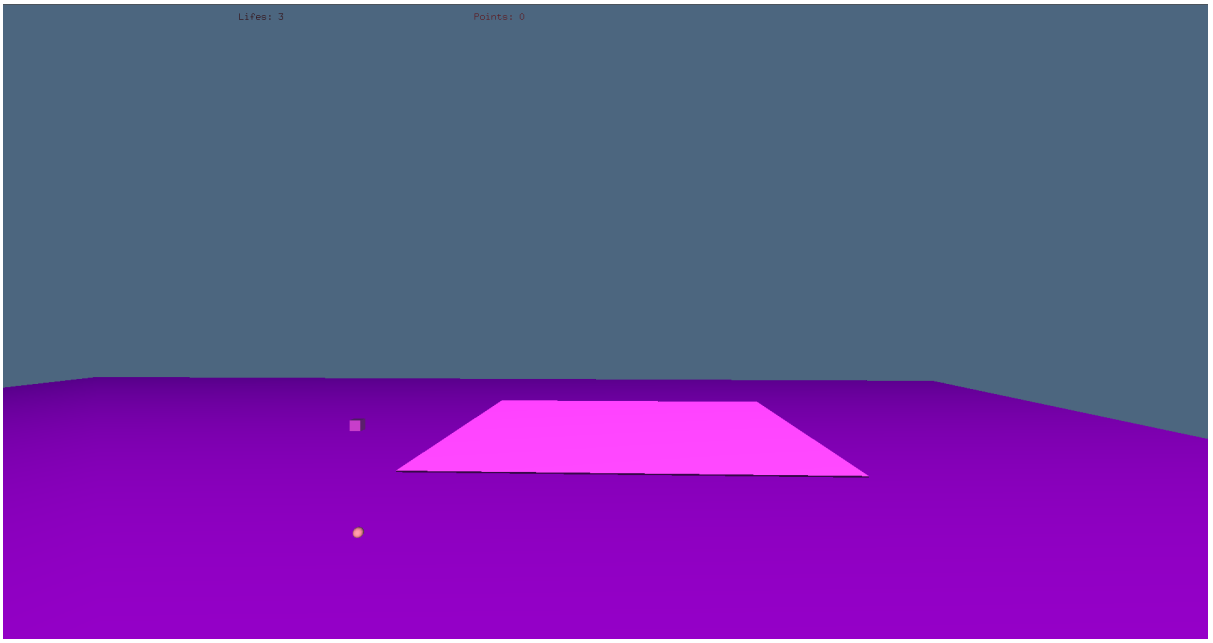
```
//Checks collisions between fruits and the floor
void GameManager::collisionsBetweenFruitsAndFloor(double t) {

    auto itFruits = fruits.begin();

    while (itFruits != fruits.end()) {
        if (physx::PxGeometryQuery::overlap((*itFruits)->rendIt->shape->getGeometry().sphere(), (*itFruits)->body->getGlobalPose(), floorItem->shape->getGeometry().box(), floor->getGlobalPose())) {

            ActualLifes--;

            deleteDynamicBody(*itFruits, "f");
            itFruits = fruits.erase(itFruits);

        }

        else {
            ++itFruits;
        }
    }

}
```

-EFECTOS INCORPORADOS

-Generación de sólidos rígidos (frutas) cada x tiempo, determinado en cada nivel.
-Destrucción de sólidos rígidos y partículas pasado un tiempo determinado.
-Destrucción de sólidos rígidos al colisionar con otros sólidos rígidos, teniendo efecto en la partida dependiendo de si son frutas y armas, frutas y el suelo, etc.
-Al completar un nivel,si has ganado, se lanzarán una serie de fuegos artificiales, si has perdido, se generará un torbellino y el cielo se teñirá de color rojo, además, los puntos se restearán.
-Se puede aplicar una fuerza de viento a sólidos rígidos.
-Contador de vidas y puntos en pantalla.

- MANUAL DE USUARIO

Al iniciar el proyecto, aparecemos en la siguiente escena. En la que podemos apreciar un plano rosa, sobre el que aparecerán las frutas, y en la parte izquierda una partícula unida por un muelle a un punto fijo de anclaje. Además, encontramos dos textos en la parte superior izquierda de la pantalla, el primero relativo a las vidas restantes, y el segundo, a los puntos logrados.

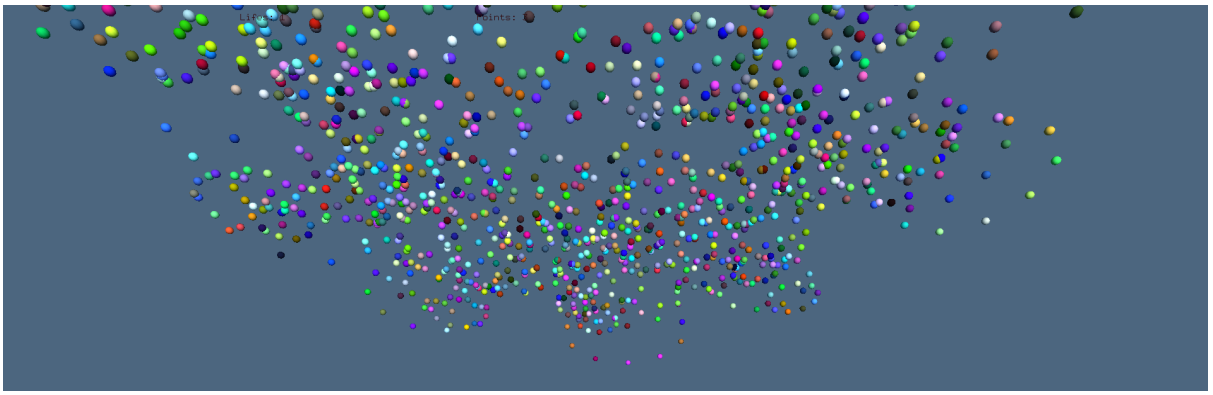


Podremos colocar la cámara a nuestro gusto con las teclas ‘W’,’A’,’S’ y ‘D’. Es recomendable acercarse al plano, debido a que si nos encontramos muy lejos, nuestras armas, que explicaré a continuación, desaparecerán antes de poder colisionar con las frutas.

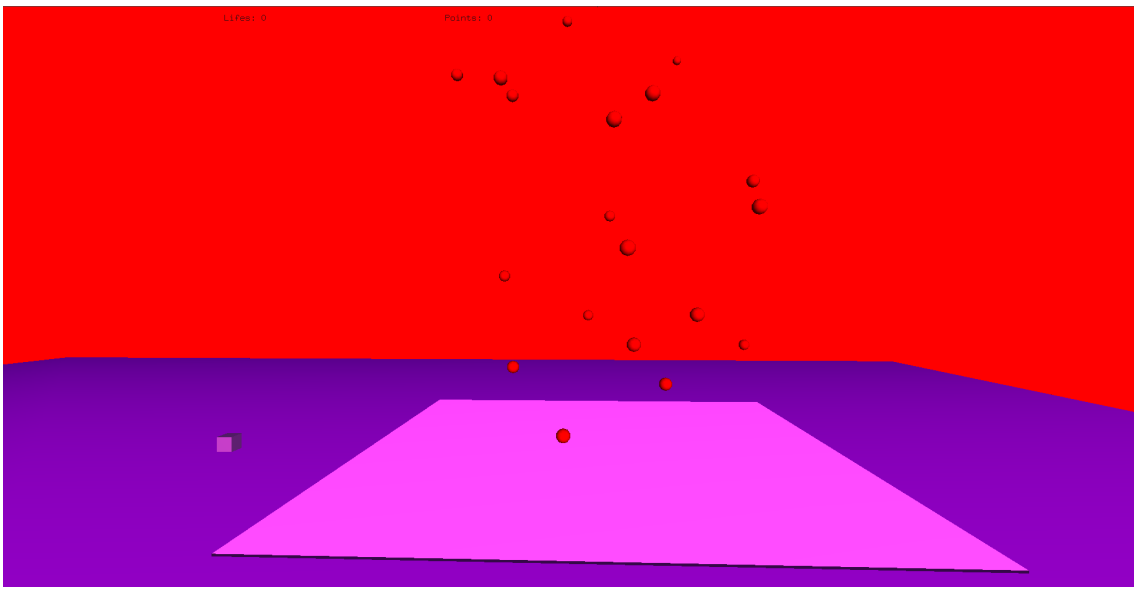
Para destruir las frutas, tendremos que hacer uso de nuestras “armas”, disparandolas con la tecla ‘X’. Para añadir dificultad al juego, está programado de manera que solo pueda haber una bala al mismo tiempo en escena, por lo que para disparar habrá que esperar a que la anterior bala disparada desaparezca (tiempo de vida: 1 segundo). Dichas armas saldrán en la dirección de la cámara, la cual podemos controlar haciendo click izquierdo y desplazando el ratón.

La partícula unida al muelle inicialmente no estará afectada por ninguna fuerza. Para iniciar un nivel, pulsaremos la tecla ‘M’, que le añadirá una fuerza a dicha partícula, y dará inicio a la aparición de frutas sobre el plano.

Como en la mayoría de juegos, en Physx Ninja puedes tanto ganar, como perder. Si has ganado, es decir, has eliminado a todas las frutas del nivel sin haberte quedado sin vidas, aparecerán una serie de fuegos artificiales en la escena.



Por otro lado, si pierdes, es decir, te has quedado sin vidas, el cielo de la escena se teñirá de rojo y aparecerá un tornado de partículas, también rojas, a los dos segundos el cielo recuperará su color inicial y el tornado desaparecerá. Además, tu contador de puntos se reiniciará a 0.



Pulsando nuevamente la tecla ‘M’, si has perdido, se reiniciará el último nivel jugado, si has ganado, avanzarás al siguiente nivel. Por otro lado, si has completado todos los niveles y quieres seguir jugando, al pulsar la tecla se volverá a empezar el primer nivel, sin reiniciar los puntos conseguidos.

Pulsando ‘V’, activaremos el power-up de viento, que durante aproximadamente 10 segundos, le aplicará una fuerza de viento a las frutas que se generen, facilitando su destrucción, para dar feedback se cambiará de color el cielo a azul mientras esté activo. Este power-up solo se puede utilizar una única vez en toda la partida.