MiniFlush Golf

Breve Descripción

El Miniflush golf es un juego de minigolf en el que el jugador deberá golpear la pelota para poder hacer hoyo y convertirte en el mejor jugador del minigolf del mundo. Para ello el jugador deberá observar el ambiente y obstáculos para poder sortearlos y poder hacer hoyo con el menor numero de movimientos posible. El juego consta de dos niveles el primero es muy simple para que el jugador se adapte a las mecánicas de juego y el segundo ya tiene una mayor complicación pues hay mas obstáculos.

Explicación De Ecuaciones

-Movimiento de la cámara alrededor del jugador: La cámara se mueve y rota alrededor del jugador. Siempre y cuando este en la vista de la pelota

```
if (follow) {
    Vector3 dir = mDir; dir.y = 0; dir.normalize();
    PxVec3 relativePos = golfBall->getGlobalPose().p;
    relativePos.y += foco-foco/3;
    relativePos -= dir * foco;
    mEye = relativePos;
    cout << "x: " << mDir.x << " y: " << mDir.y << " z: " << mDir.z << "\n";
}
```

En cuanto a la rotación de la cámara lo que hacemos es que ponemos la y=0 del vector de direccion, normalizamos y sumamos un offset para dar la sensación de que la cámara se rota alrededor de la pelota , pero en verdad lo que hacemos es mover la cámara justo a la direccion de la pelota y la desplazamos un poco.

Direccion de disparo: Siempre que el jugador este quieto se hace visible unas partículas en línea q señalizan la direccion de disparo. Estas siempre que el jugador este quieto, recalculan su posición con la direccion de la cámara.

```
void ParticleSystem::updateLineaDisparo(Vector3 pos,Vector3 dir,bool desactivar){
    int i = 1; dir.y = 0; dir.normalize();
    for (auto it=list_linea.begin(); it != list_linea.end(); it++){
        auto x = *it;
        if (!desactivar)
             x->setVisible(false);
        else {
            x->setVisible(true);
            x->setPosition(Vector3(pos.x +1.5*dir.x * i, pos.y, pos.z + 1.5 * i*dir.z));
            i++;
        }
    }
}
```

-Input: Para meterle la fuerza al jugador con la direccion de la cámara, cuando el jugador le da click derecho al ratón y lo mantiene se va acumulando una fuerza la cual será aplicada en la direccion de la cámara del jugador, cuando deja de despulsar.

```
void Game::seeMouseInput(int button, int state, int x, int y){
    if (!canMove())return;
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        //std::cout << "sumandooo" << "\n";
        sumarFuerza = true;
        keepSafePosBall(golfBall->getGlobalPose().p);
    }
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP) {
        //std::cout << "fin" << "\n";
        if (!sumarFuerza)return;
        fuerzaGolfBall.x *= myCamera->getDir().x;
        fuerzaGolfBall.z *= myCamera->getDir().z;
        //cout << "x: " << fuerzaGolfBall.x << " " << "y: " << fuerzaGolfBall.y << " " << "z: " << fuerzaGolfBall.z << "\n";
        if (maxForce.x < fuerzaGolfBall.x || maxForce.z < fuerzaGolfBall.z)fuerzaGolfBall = maxForce;
        //cout << "x: " << fuerzaGolfBall.x << " " << "y: " << fuerzaGolfBall.y << " " << "z: " << fuerzaGolfBall.z << "\n";
        golfBall->setLinearVelocity(fuerzaGolfBall);
        fuerzaGolfBall = Vector3(0);
        sumarFuerza = false;
    }
}
```

-Colisiones: Miramos dos tipos de colisiones de objetos solido rígidos.

Cuando el jugador acaba el nivel, el jugador pierde la capacidad de movimiento y la cámara enfoca a los fuegos artificiales. Cuándo estos fuegos desaparecen da comienzo el nuevo nivel, eliminando de pantalla y memoria los anteriores objetos, la cámara enfoca el nuevo nivel, además de habilitarle los controles al jugador.

Cuando el jugador se sale dl mapa y colisiona con un suelo. El jugador vuele a la última posición en la k ha estado quieto.

```
void Game::collisions(PxActor* actor1, PxActor* actor2){
    if (actor2->getName() == "SUELO-BASE" && actor1->getName() == "GOLFBALL") {
        retakeBall();
    }
    else if ((actor2->getName() == "hoyo1"|| actor2->getName() == "hoyo2") && actor1->getName() == "GOLFBALL") {
        auto hoyo = static_cast<PxRigidStatic*>(actor1);
        if ((golfBall->getLinearVelocity() + golfBall->getGlobalPose().p).magnitude() >= hoyo->getGlobalPose().p.magnitude()) {
            particleSystem->generateRocketSystem("FireworkFinal", Vector3(-1));
            if(actor2->getName() == "hoyo1")myCamera->setDir({ 0.409697,0.912008,0.0197556 });
            else myCamera->setDir({0.0620697 ,0.980941, -0.184127 });
            nextLevel = actor2->getName();
            golfBall->putToSleep();
            controlActive = false;
        }
    }
}
```

-Muelle obstáculo: estos tienen tienen una fuerza AnchoredSpringFG, es decir una fuerza elástica siguiendo la ley de Hooke, la cual dicha fuerza que actúa sobre el objecto se contrae y se estira dependiendo de su posición y constante elástica.

```
Vector3 force = other->getPosition() - solid->getGlobalPose().p;
const float length = force.normalize();
const float delta_x = length - resting_lenght;
force *= delta_x * k;
solid->addForce(force);
```

-Viento Obstáculo: tienen una fuerza de viento, siempre y cuando se encuentren dentro del rango de esta fuerza, se le aplica la fuerza de rozamiento contraria al movimiento, esta consta de dos constantes las cuales k2 es 0 pq no queremos que el viento sea muy fuerte.

```
float drag_coef = v.normalize();
Vector3 dragF;
drag_coef = k1 * drag_coef + k2 * drag_coef * drag_coef;
dragF = -v * drag_coef;
return dragF;
```

-Flotación: Para simular un líquido medio burbujeante, tanto la densidad volumen y altura que se le ha asignado es(1.0,1.0,2.0) y a su vez el peso del liquido es bajito para que no se hunda y flote más y la inversión sea menor.

```
f.y = liquid_density * volume * immersed * 9.8;
```

-Materiales:

-Hielo: Para simular un supuesto hielo, la pista es de color azul y el material al que se le asocia es a la fricción dinámica 0.1 para que sea mínima y la restitución a 1.0

-Pegajoso: Para simular un supuesto elemento pegajoso, la pista es de color amarillento y el material al que se le asocia es a la fricción dinámica 1.0 para que sea máxima y la restitución a 0.5.

```
worldManager->setNewMaterial(gPhysics->createMaterial(0.3f, .1f, 1.0f));//HIELO ID=1;
worldManager->setNewMaterial(gPhysics->createMaterial(0.9f, 0.9f, 0.5f));//Pehgajoso ID=2;
```

Hay una clase define_materials de cara a sacar los colores de los elementos de los niveles, los id de los materiales.

```
#pragma once
#define MATERIAL_HIELO 1
#define MATERIAL_PEGAJOSO 2
#define COLOR_HIELO {0.5,0.8,0.9,1.0}
#define COLOR_PEGAJOSO {0.8,0.75,0.1,1.0}
#define COLOR_BASE_GENERADORES_RIGIDS {0.7,0.6,1.0,1.0}
```

Además, destaca el uso de generadores gaussianos y uniformes en el caso de los cohetes para generar todos estos objetos a excepción de la pista y los algunos de los muelles.

Efectos Incorporados

-Hay una niebla que provoca una pérdida de visibilidad, dando así una mayor funcionalidad a salirte de la cámara de la bola y medio obligando al jugador a ver a donde dirigirse.

-Rachas de viento: hay varios generadores gaussianos de solido rígidos que generan solidos como por rachas siendo así un obstáculo más para el jugador.

-Efectos de distintos tipos de carriles con distintos efectos, hielo y pegajoso

-Cuando terminas el nivel se generan una serie de cohetes artificiales random señalizando así que has acabado el nivel.

-Cuando terminas el nivel la direccion de la cámara se alza al cielo para que el jugador pueda contemplar dichos fuegos artificiales y cuando terminar los fuegos, da comienzo el nuevo nivel y la cámara enfoca al siguiente nivel.

ENLACE A DIAGRAMA DE CLASES:

https://drive.google.com/file/d/1_DSafCGCFeh2o9aJ5bPSumxxFAYOooGl/view?usp=sharing


RocketGenerator
Firework
Particle
GaussianParticleGen
ParticleGenerator
CircleGenerator
UniformParticleGenerator
ParticleSystem
ParticleForceRegistry
WindForceGenerator
WhirlWindGenerator
Main
Game
Camara
ForceGenerator
ParticleDragGenerator
AnchoredSpringFG
GravityForceGenerator
SpringForceGenerator
FloatBounceForce
TorqueForce
ElasticBand
define_materials
RigidGenerator
WorldManager
RigidSolid
GaussianRigidGenerator
RigidForceRegistry