

Proyecto Intermedio

Iván Palomino Rodriguez

● Introducción

El juego propuesto se trata de un lanzamiento a canasta. El jugador deberá encestar en la c canasta para así anotar puntos. Habrá tres tipos de canastas (unas más grandes y otras más pequeñas) para cada tipo de bola. Cuanto más tiempo pase el juego se volverá más complicado haciendo que las canastas se muevan, zonas de viento que afecten a las bolas y hagan que cambie la dirección de estas.

● Controles

Manteniendo el espacio -- Cargar disparo (Cuanto más pulses el espacio más fuerza sale la bala)
Soltar espacio -- Disparar

Tecla 1 -- Cambiar a la bola tipo 1 (la normal)

Tecla 2 -- Cambiar a la bola tipo 2 (la bola más grande y más pesada)

Tecla 3 -- Cambiar a la bola tipo 3 (la bola más pequeña y menos pesada)

Tecla 4 -- Activar explosion (solo se podra activar la explosion si hay una bala en pantalla y afectará a la última bala creada)

Tecla T -- Activar/Desactivar los generadores de partículas que simulan los confeti

Tecla R -- Activar/Desactivar la fuerza del torbellino

Tecla V -- Activar/Desactivar la fuerza del viento

● Requisitos de Programación

-Requisito A) → clase “Particle.h” y “Particle.cpp”

Requisito A.1) →

Implementado en: “Particle.cpp” líneas 18-35.

Método: void integrate(double t);

Explicación: este método aplica la integración de Euler Semi Explícito.

Requisito A.2) →

Implementado en:

“Particle.h” línea 30, atributo float mass.

“Particle.cpp” línea 4 , inicializo el parámetro mass_ en el constructor .

“main.cpp” línea 96 o “TiroCanasta.cpp” linea 124, le paso diferentes parametros al generador de particulas de los cuales en este caso 1.0f es la masa del primero y 0.5f es la masa del segundo. Dentro del metodo de “ParticlesGen.cpp” linea 49-69 metodo void generateParticle() ; Se crea la Particula con esos parametros incluyendo la masa : Particle* p = new Particle(posRandom, vel, damping, lifeTime, color, tam, mass);.

Requisito A.3) →

Implementado en:

“Particle.cpp” linea 21, calculo la aceleración teniendo en cuenta la masa..

“Particle.h” linea 18 y 19, defino 2 métodos para la gestión de fuerzas: void addForce(const Vector3D f) { force = force + f; } y void clearForce() { force = Vector3D(0, 0, 0); }.

-Requisito B) → clase “Projectile.h” y “Projectile.cpp”.

Clase “Projectile.cpp” linea 7-18 dentro del constructor de projectile hago el escalado de velocidad, energía y fuerzas.

Clase “TiroCanasta.cpp” linea 149-178 metodo void crearBola(Vector3D pos, Vector3D dir, float fuerza); creo tres posibles tipos de proyectil distintos con masa, tamaño, color y velocidad distintos.

-Requisito C) → clase “ParticleSys.h” , “ParticleSys.cpp”, “ForceSys.h” y “ForceSys.cpp”.

Clase “ParticleSys” se encarga de gestionar los generadores de partículas, actualizando cada conjunto el cual previamente se tiene que haber añadido al sistema con el método void addParticle(ParticleGen* pg); . También el sistema borra los generadores que están controlados por tiempo o que ya no tengan partículas.

Clase “ForceSys” se encarga de gestionar las diferentes fuerzas que actúan sobre cada partícula. Tengo un unordered_map que asocia la partícula con la fuerza aplicada. en el update recorre todas las partículas, aplica la fuerza, actualiza cada generador de fuerzas y elimina las partículas que no están vivas.

-Requisito D) → clase “ParticleGen.h” y “ParticleGen.cpp”.

Requisito D.1) →

Implementado en: “ParticleGen.cpp” línea 49-69.

Método: void generateParticle() ;

Explicación: Cada generador de partículas contiene como atributos los parámetros de una partícula modelo con (la posición, la velocidad media, tiempo de vida, tamaño, masa y color).

Requisito D.2) →

Implementado en: “ParticleGen.cpp” línea 71-86.

Método: float getRandomRange(float a, float b) ;

Explicación: cuando creo el generador de partículas el cual le paso un parametro que decide que distribucion probabilistica usar en cada caso, estas distribuciones se usan para variar la posición inicial y la velocidad de las partículas en el método void generateParticle() ;

Requisito D.3) →

Implementación en : “ParticleSys.cpp” y “ParticleGen.cpp”.

Explicación: El sistema de partículas tiene una lista de generadores de partículas, “ParticleSys.h” línea 8, el sistema de partículas recurre esa lista en cada frame y actualiza los diferentes generadores de partículas, “ParticleSys.cpp” línea 29, dentro del update de los generadores de partículas se crean nuevas partículas y se actualizan las existentes, “ParticleGen.cpp” línea 19-47 método void update(double dt) ;. Se podrá activar y desactivar los diferentes generadores con el método void togglePausar() { pausar = !pausar; }; en “ParticleGen.h” línea 22.

En la clase “main.cpp” línea 96 y 98 creo dos generadores de partículas que simulan confeti, luego los añado al sistema de partículas con el método void addParticle(ParticleGen* pg); de “ParticleSys.h” al sistema de partículas(esto ocurre en las líneas 103 y 104 de “main.cpp”). En la línea 208-211 de main.cpp tengo que los generadores se puedan activar o desactivar.

-Requisito E) → clase “ForceGen.h”, “ForceSys.h”, “ForceSys.cpp”, “GavityForceGen.h”, “GravityForceGen.cpp”, “ExplosionForce.h”, “ExplosionForce.cpp”, “WindForceGen.h”, “WindForceGen.cpp”, “ZonaDeViento.h”, “ZonaDeViento.cpp” , etc.

Requisito E.1) →

Implementación en: “ForceGen.h”, línea 10 .

Método: virtual void updateForce(Particle* particle, double dt) = 0;.

Explicación: cada generador de fuerzas tiene un método updateForce que recibe la partícula , calcula la fuerza individualmente y luego le manda la fuerza con el método void addForce(const Vector3D f) { force = force + f; }, declarado en “Particle.h” linea 18.

Requisito E.2) →

Explicación: tengo un unordered_map en “ForceSys” que contiene la particula y un vector de generador de fuerzas esto hace que se pueda tener más de una fuerza en una particula ya que en el update actualizo las fuerzas, clase “ForceSys.cpp” linea 19-41.

Requisito E.3) →

Implementación en: “ParticleGen.cpp”.

Explicación: tengo un vector de generadores de fuerza en el generador de partículas que controla que fuerzas afectan ese generador.

Requisito E.4) →

Implementación en “TorbellinoForceGen”, “WIndForceGen”, “ZonaDeVientoGen”, “ExplosionForce”, “GravityForceGen”.

Explicación: mi fuerza de torbellino varía con la posición de la particula, mi fuerza de explosion decae con el tiempo, mi fuerza zonaDeViento sólo se aplica dentro de una zona esférica y mi fuerza de la gravedad y del viento son fuerzas básicas.

Se puede ver que mis projectiles son afectados por tres fuerzas distintas en la clase “TiroCanasta.cpp” línea 173-175.