Práctica 5
Informática 2
Facultad de Ingeniería
Universidad de Antioquia

1 Motivación

1.1 Introducción

El mundo en el que habitamos está en constante movimiento. Desde el lanzamiento de un cohete hasta el simple rebote de una pelota están regidos por principios físicos que pueden ser modelados y simulados. Comprender y aplicar estos principios es fundamental en campos como la ingeniería, la robótica y el desarrollo de videojuegos.

Esta práctica permitirá a los estudiantes adquirir destreza en el modelamiento y la simulación de sistemas físicos utilizando la programación orientada a objetos (POO). Se abordarán conceptos clave de la física, como la dinámica del movimiento y las colisiones (tanto elásticas como inelásticas), y se simularán sus comportamientos en un entorno computacional. La primera parte de la práctica se centrará en el desarrollo de la lógica de simulación en tiempo discreto, creando un programa que calcule y almacene las trayectorias de los objetos en un archivo de texto, sin la necesidad de una interfaz gráfica. Esto permitirá a los estudiantes concentrarse en la implementación de las ecuaciones de movimiento y de conservación del momento en una colisión.

Posteriormente, en la segunda parte, la práctica evolucionará hacia la creación de un juego interactivo. Se utilizarán las simulaciones desarrolladas en la primera parte para desarrollar una aplicación con una interfaz gráfica de usuario (GUI) funcional. Este enfoque progresivo permitirá a los estudiantes entender cómo los principios de la física se integran en la programación para crear videojuegos, que se basan en fenómenos físicos como el movimiento y las colisiones. A lo largo de este proceso, se consolidarán habilidades en el diseño y la implementación de aplicaciones usando POO y la gestión de datos para la visualización.

Al finalizar la práctica, el estudiante será capaz de diseñar y programar un sistema físico simulado mediante POO, comprendiendo la relación entre los modelos físicos y su implementación computacional, además de aplicar estos principios en el desarrollo de una aplicación interactiva con interfaz gráfica.

1.2 Objetivos

- Adquirir destreza en el modelamiento y simulación de sistemas físicos utilizando los recursos propios de la programación orientada a objetos.
- Afianzar el desarrollo de aplicaciones usando interfaz gráfica en el IDE Qt Creator.
- Implementar las ecuaciones de movimiento y de conservación del momento de los sistemas físicos propuestos para simular fenómenos físicos en un entorno gráfico.

1.3 Simulación de fenómenos físicos en tiempo discreto

La simulación de un sistema físico puede ser compleja, especialmente si no se puede resolver de manera analítica debido al gran número de variables presentes en el fenómeno. Para simular estos sistemas, se utilizan herramientas que calculan el estado del sistema en pequeños intervalos de tiempo (Δ_t).

Este enfoque de simulación se basa en tomar el estado actual del sistema y aplicar las ecuaciones que lo modelan para calcular su estado en el siguiente instante de tiempo. Los valores obtenidos se convierten

en los valores actuales para el próximo cálculo. Este proceso se repite definida o indefinidamente para simular el comportamiento del sistema a lo largo del tiempo. Sin embargo, en algunos casos, existen ecuaciones que facilitan el cálculo de la posición en cualquier tiempo t.

Un ejemplo de este método es la simulación de un movimiento parabólico. En lugar de usar una ecuación continua, el sistema calcula la posición y velocidad de un objeto en intervalos muy pequeños (Δ_t). Para cada intervalo, se actualizan las coordenadas de posición (x, y) y las componentes de la velocidad (v_x, v_y). Así, un lanzamiento se puede simular paso a paso, recreando su trayectoria curva a lo largo del tiempo. Se puede tomar una simplificación de las ecuaciones del movimiento parabólico para calcular, conociendo las condiciones iniciales del lanzamiento, las posiciones (x, y) en cada instante ($t_i + \Delta_t$) y las velocidades iniciales en ambos ejes (v_x, v_y). Para un movimiento parabólico donde la componente en y es de caída libre (con $a_y = g$, donde g es la aceleración de la gravedad) y la componente en x es un movimiento rectilíneo uniforme (donde $a_x = 0$) el valor de las componentes de la posición en cada instante de tiempo se modela de la siguiente forma:

$$x = x_0 + v_x t \tag{1}$$

$$y = y_0 + v_y t - \frac{1}{2} g t^2 \tag{2}$$

En el simulador oPhysics , se puede ver un ejemplo de una simulación donde para cada instante $t_i + \Delta_t$ se calculan los valores en x y y. Ahora bien, si escogemos un Δ_t lo suficientemente pequeño, dicha simulación podría parecer una animación.

1.3.1 Ejemplo

El profesor realizará un ejemplo donde simulará un movimiento parabólico, calculará las posiciones en ambos ejes (x, y) y exportará los resultados a un archivo de texto. Luego, este archivo de texto será graficado utilizando una herramienta de preferencia, como se muestra en la Figura 1.

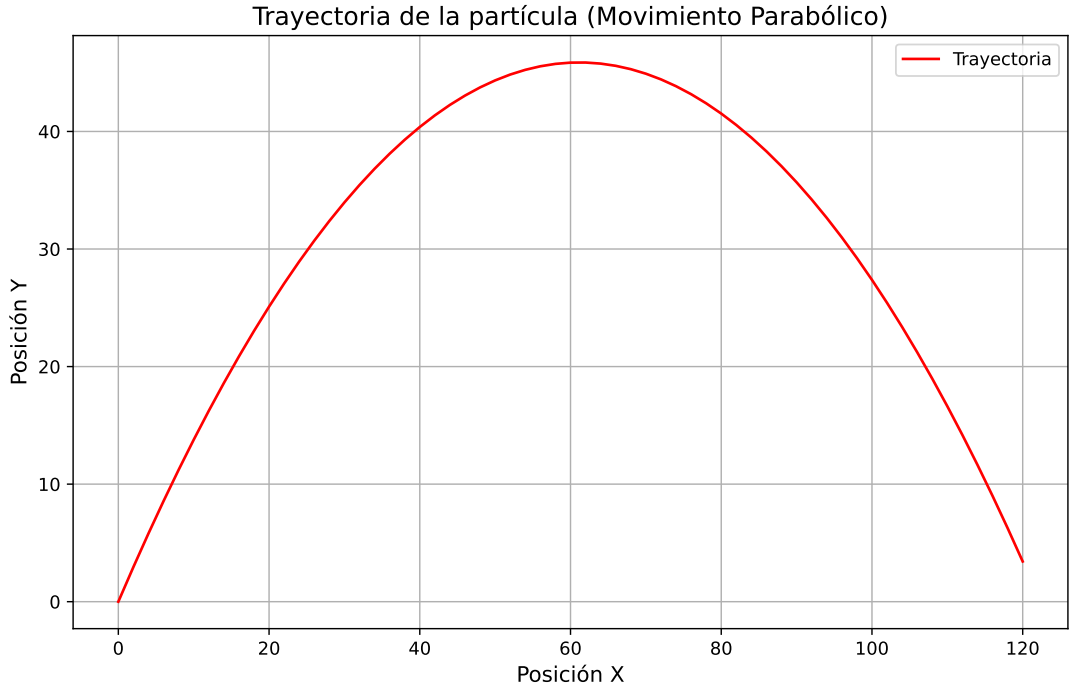

Figura 1: Muestra de trayectoria graficada usando matplotlib de python

2 Práctica

La parte práctica está dividida en dos actividades. Primero se modelará un fenómeno físico sin interfaz gráfica . Luego se aprovechará lo desarrollado en la primera parte para crear un juego de estrategia por

turnos.

2.1 Actividad: Simulación de un sistema de colisiones múltiples

Duración: dos semanas.

El objetivo de esta actividad es que los estudiantes pongan en práctica sus conocimientos de programación orientada a objetos (POO) y física para simular el comportamiento de múltiples partículas que se mueven y chocan dentro de un espacio cerrado. La simulación debe ser capaz de modelar diferentes tipos de colisiones, lo que requiere una comprensión clara de la conservación del momento y el uso de coeficientes de restitución. **El estudiante debe demostrar el avance continuo de esta actividad a través de actualizaciones periódicas del código en el repositorio.**

2.1.1 Requerimientos de la actividad

El programa deberá simular el movimiento de n partículas dentro de un área rectangular que representa una "caja". Cada partícula circular tendrá una velocidad inicial y una masa. Dentro de esta caja habrá cuatro obstáculos cuadrados estáticos en posiciones definidas por el estudiante. La simulación debe manejar tres tipos de colisiones:

1. **Colisiones entre partículas y las paredes de la caja:** Estas colisiones deben ser **perfectamente elásticas**. Cuando una partícula choca con una pared, su componente de velocidad perpendicular a la pared debe invertirse, conservando la energía cinética total.

2. **Colisiones entre partículas y los obstáculos cuadrados:** Estas colisiones deben ser **inelásticas**, utilizando un **coeficiente de restitución (ϵ)** para determinar la pérdida de energía. El estudiante debe definir este coeficiente. La colisión debe modelarse de manera que la dirección del rebote dependa del ángulo de impacto. El rebote se modelará aplicando el coeficiente de restitución (e) a la componente de la velocidad **perpendicular** al plano de impacto ($\vec{v}_\perp$), mientras que la componente paralela ($\vec{v}_\parallel$) se mantiene intacta.

   $$\vec{v'}_\perp = -e\vec{v}_\perp \quad \text{y} \quad \vec{v'}_\parallel = \vec{v}_\parallel$$

   El estudiante debe implementar la lógica para determinar con cuál lado del cuadrado se produjo la colisión.

3. **Colisiones entre partículas:** Estas colisiones deben ser **completamente inelásticas**. Cuando dos partículas (m_1, m_2) chocan, se deben **unir** y moverse como un solo cuerpo con una masa final $M = m_1 + m_2$ y una nueva velocidad ($\vec{v'}$). Se debe aplicar la ley de conservación del momento lineal:

   $$m_1\vec{v_1} + m_2\vec{v_2} = (m_1 + m_2)\vec{v'}$$

   La nueva velocidad del cuerpo combinado es:

   $$\vec{v'} = \frac{m_1\vec{v_1} + m_2\vec{v_2}}{m_1 + m_2}$$

   El nuevo cuerpo debe heredar las propiedades necesarias para la simulación, que serán determinadas por el estudiante (posición del centro de masa, radio/área del nuevo cuerpo, colisiones posteriores con terceros, nueva dirección del movimiento).

El estudiante puede considerar las simplificaciones necesarias, siempre y cuando no afecten lo solicitado en la actividad.

Se recomienda modelar el sistema basado en la ley de conservación del momento lineal, que establece que el momento total de un sistema aislado permanece constante. Para una colisión entre dos cuerpos, esta ley se expresa como:

$$m_1\vec{v_1} + m_2\vec{v_2} = m_1\vec{v_1'} + m_2\vec{v_2'}$$

Donde m es la masa, $\vec{v}$ es el vector de velocidad antes de la colisión, y $\vec{v'}$ es el vector de velocidad después de la colisión.

2.1.2 Implementación

Se recomienda que el programa modele las partículas y los obstáculos como **objetos**. Cada partícula debe tener propiedades como posición (x, y), velocidad (v_x, v_y) y masa. El programa debe iterar a través de un bucle de tiempo discreto, actualizando la posición de cada partícula y verificando si hay colisiones en cada paso. Cuando se detecte una colisión, el programa debe aplicar las fórmulas de conservación de momento y energía correspondientes al tipo de colisión.

Para la salida de datos, el programa debe guardar en un archivo de texto las posiciones (x, y) de cada partícula en cada paso de tiempo. Este archivo será usado posteriormente para graficar la simulación. Teniendo en cuenta lo anterior, usted debe incluir lo siguiente:

- Correcta implementación de las leyes de movimiento y conservación del momento, verificando que las posiciones y velocidades de las partículas se actualicen conforme a las ecuaciones en tiempo discreto. También se tendrá en cuenta el tratamiento de las colisiones con las paredes, asegurando que se modelen como rebotes perfectamente elásticos.ç
- Diseño e implementación de las clases que considere necesarias para la implementación. Debe incluir un diagrama de clases, indicando las relaciones entre ellas y los atributos de cada una.
- El programa debe generar un archivo de salida en formato de texto que contenga, de manera estructurada, las posiciones de cada partícula en cada instante de tiempo y las colisiones, junto con el instante de tiempo en que se dieron. Todo esto debe ser graficado en una herramienta de su preferencia.
- Una simulación que soporte al menos 4 partículas simultáneas.

2.2 Actividad: Desarrollo de un juego interactivo

Duración: dos semanas.

Ahora que ya se exploró la simulación de fenómenos físicos, es momento de usar esos conocimientos para crear un juego. En esta sección, se construirá una aplicación visual e interactiva. Usando la interfaz gráfica de Qt, se debe crear un programa donde el usuario podrá controlar los lanzamientos, ver las trayectorias y presenciar las colisiones en tiempo real. Esto ayudará a entender cómo la física computacional se aplica en los videojuegos y dará una experiencia práctica en el diseño de interfaces de usuario. **El estudiante debe demostrar el avance continuo de esta actividad a través de actualizaciones periódicas del código en el repositorio.**

2.2.1 Ejemplo

El profesor realizará un ejemplo donde retomará aspectos esenciales de la creación de interfaces gráficas usando Qt. También hará una introducción al uso de periféricos para generar acciones dentro de una interfaz gráfica. Se le recomienda al estudiante revisar la documentación de los recursos proporcionados por Qt para administrar e interactuar con elementos gráficos en dos dimensiones.

2.2.2 Requerimientos de la actividad

El programa consiste en un juego de estrategia militar por turnos, donde dos jugadores intentan destruir la infraestructura del rival para poder vulnerar al representante del enemigo. La infraestructura del rival está compuesta por obstáculos rectangulares que cuentan con una resistencia representada por un valor numérico. Cada jugador lanzará un proyectil por turno, al cual se le puede programar un ángulo y velocidad iniciales. Las colisiones del proyectil contra los límites del escenario (caja) serán perfectamente elásticas, mientras que las colisiones con la infraestructura serán inelásticas con un coeficiente de restitución que represente una pérdida de energía. Cada colisión de un proyectil con una parte de la infraestructura del rival afectará la resistencia del objeto; el daño infligido será **directamente proporcional al momento lineal** $(\vec{p})$ del proyectil en el momento del impacto. El momento lineal de impacto es:

$$|\vec{p}_{\text{impacto}}| = \text{masa} \times |\vec{v}_{\text{impacto}}|$$

El daño se calculará con una fórmula definida por el estudiante:

$$\text{Daño} = \text{factor\_constante} \times \text{masa}_{\text{proyectil}} \times |\vec{v}_{\text{impacto}}|$$

El alumno debe definir el *factor_constante* y el mecanismo de resta del daño a la resistencia del obstáculo. Una muestra de la interfaz simple del juego se puede observar en la Figura 2

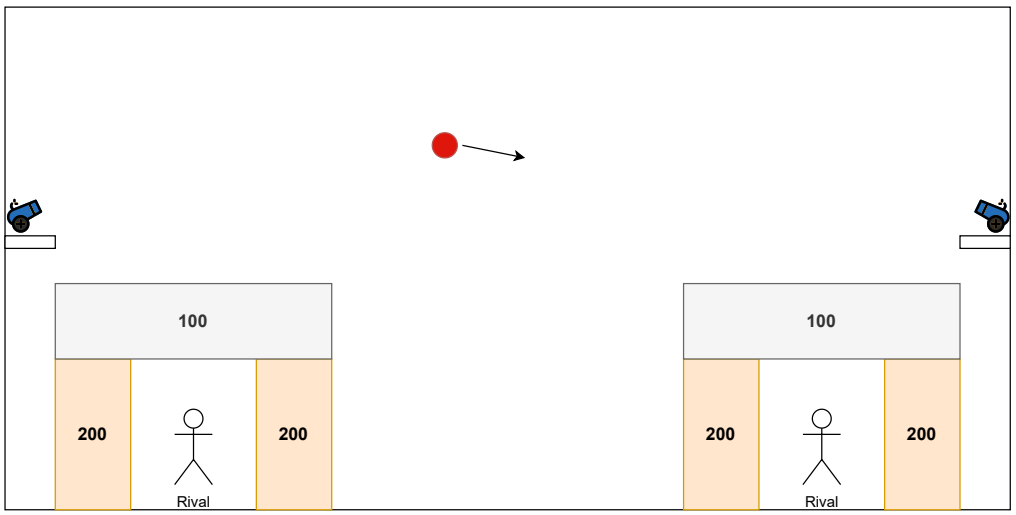

Figura 2: Muestra de interfaz simple.

2.2.3 Implementación

Para lograr el desarrollo de este juego, que integra la simulación de fenómenos físicos con la interacción del usuario a través de una interfaz gráfica (GUI), el estudiante debe implementar el código necesario para gestionar el movimiento por turnos, las colisiones y la actualización gráfica. Según lo anerior se debe incluir en la implementación:

- Diseño e implementación de las clases que considere necesarias para la implementación. Debe incluir un diagrama de clases, indicando las relaciones entre ellas y los atributos de cada una.
- Implementar correctamente la lógica de la simulación de los fenómenos físicos, que incluyen movimientos y colisiones.
- Integración gráfica para la visualización de las animaciones propias del juego. Esto debe hacerse con las clases dispuestas por Qt.
- Correcta implementación de la lógica de juego, que incluye la gestión de turnos, el cálculo de daño y las condiciones de victoria.

Referencias

- oPhysics. *Projectile motion simulation*. Disponible en: `https://ophysics.com/k8.html`
- Qt Documentation. *Graphics View Framework*. Disponible en: `https://doc.qt.io/qt-6/graphicsview.html`