Informática Gráfica

Ingeniería en Informática

Curso 09-10. Práctica 3.2

Carácter: Obligatorio

Fecha de entrega: Miércoles 20 de Enero

Objetivo: colisiones, rebotes, simulación de movimiento

Descripción: se trata de implementar el juego del billar bajo las siguientes condiciones. La *pelota* se desplaza dentro de la *mesa* sólo con movimiento lineal, es decir, eliminamos el movimiento rotacional y los efectos que produce. En la mesa se encuentran varios *obstáculos* que siempre permanecerán inmóviles. Aunque tienes libertad para elegir cuántos obstáculos introduces, cuál es su tamaño y cuál es su forma, al menos debes incluir un obstáculo de estas dos categorías:

1. Polígonos convexos
2. Círculos

Entre los atributos que caracterizan una pelota, se incluyen la figura que usamos para visualizarla, su radio, la posición de su centro y el sentido de su movimiento. Puedes entender que el módulo de su sentido corresponde a su velocidad, o puedes usar vectores de módulo 1 y llevar por separado el tamaño que debe aplicarse cada vez que la pelota se desplaza. Al moverse, las pelotas pueden colisionar con las paredes o con los obstáculos, en cuyo caso la pelota rebotará por reflexión. En cualquier caso la velocidad de la pelota debe disminuir para simular el gasto que ocasiona el roce con la superficie de la mesa, más aún después de una colisión.

En cuanto a los obstáculos, éstos deben resolver la cuestión "¿la pelota me ha golpeado?" mediante el algoritmo apropiado. Concretamente, para los polígonos convexos usarás el algoritmo de Cyrus-Beck, mientras que para los Círculos emplearás el procedimiento específico que vimos en clase. Tu práctica debe admitir dos modos de ejecución. En el más sencillo se permitirá la penetración de la pelota, ya que supondremos que la pelota es una partícula sin masa. En este modo, los algoritmos de intersección son exactamente los vistos en clase. En el más complicado, evitarás la penetración recubriendo cada obstáculo con una corteza cuyo grosor sea el radio de la pelota. Otra posibilidad es adaptar los algoritmos de intersección para manejar peloats en lugar de partículas (esta aproximación también se describió en clase).

La interacción con el juego consistirá en golpear la pelota hacia el punto que usuario marque en el borde interior de la mesa. Para ello, el usuario podrá desplazar dicho punto usando las flechas del teclado. El golpe se produce cuando la pelota está en reposo pulsando enter.

Detalles de la implementación: Estructura de la información

Debes estructurar tu código usando clases como las que se enumeran a continuación. Otorga a cada clase el comportamiento que le corresponde.

1. *PV* guarda las coordenadas de un punto o vector en dos dimensiones.
2. *Obstaculo* es una interfaz que define métodos abstractos para dibujarse y determinar si existe colisión con una pelota, devolviendo en caso afirmativo el correspondiente t_{HIT} y la normal implicada.
3. *Convexo* y *Circulo* extienden Obstaculo y resuelven la colisión contra pelota de forma específica.
4. *Obstaculo_Recubierto* contendrá dos obstáculos, uno que usará para dibujar el obstáculo y otro invisible, que usará para resolver el test de intersección.
5. *Pelota* guarda su radio, la posición de su centro, su sentido y la figura que usamos para dibujarla.
6. *Mesa* guarda el obstáculo correspondiente a sus paredes, y el punto que se usa para establecer la dirección de la pelota tras el golpeo.

Detalles de la implementación: Diseño algorítmico

1. Puedes suponer que las dimensiones de la ventana donde se presenta la habitación no pueden modificarse. Así no debes preocuparte del evento `onResize`.
2. La simulación del movimiento se consigue con un temporizador de la clase `TTimer`. Implementa el método asociado a este reloj de forma que su ejecución consista en determinar si se producirá alguna colisión primero, y en mover la pelota adecuadamente después. Podemos entender que este método representa un *paso* en la simulación. El reloj se pone en marcha cada vez que se golpea la pelota, y se detiene cuando la velocidad de la pelota sea muy pequeña (por debajo de un umbral que debes configurar).
3. Con respecto a las colisiones que sufrirá la pelota:
    a. Ten en cuenta los posibles errores de redondeo que puedan ocasionarse al implementar el algoritmo de Cyrus-Beck. Puede ser buena solución usar en ciertas ocasiones un valor muy pequeño en lugar del 0 exacto. Usar el tipo `GLdouble` en lugar de `GLfloat` también puede resultar beneficioso.
    b. Las paredes de la mesa pueden considerarse como un objeto de una nueva clase de obstáculo, la de los polígonos simples no convexos, ya que su interior es el exterior de la mesa. Para gestionar las colisiones que provocan hay varias opciones. Por una parte se puede adaptar el algoritmo de Cyrus-Beck para calcular los puntos de impacto. Por otra, puedes suponer que en realidad se trata de cuatro rectángulos (obstáculos de la clase Convexo) convenientemente colocados en la escena.
    c. Al impactar la pelota resulta interesante suponer que su movimiento (el que le corresponde avanzar en el paso actual) se agota justo al colisionar; es decir, que la pelota se detiene al impactar, aunque todavía pudiera avanzar más tras el rebote. De esta forma, la pelota empezaría a moverse en el siguiente paso en la nueva dirección.
    d. El sentido de la pelota tras el choque se calcula mediante reflexión.
    e. Siempre debes evitar la penetración parcial o total de la pelota.

Detalles de la implementación: Etapas de desarrollo

Lo que sigue son las etapas que puedes seguir para desarrollar la implementación cómodamente. Como es habitual resulta conveniente ir archivándolas tras probar que funcionan adecuadamente.

Etapa I. Una pelota rebotando en la mesa, incluyendo la implementación del golpeo
Etapa II. Una pelota rebotando en la mesa con polígonos convexos
Etapa III. Una pelota rebotando en la mesa con círculos
Etapa IV. Una pelota rebotando en la mesa con obstáculos recubiertos en modo de no penetración.

Parte Opcional

[+] Incluye en la escena otras dos pelotas, de más de la del jugador. En cada paso de la animación estas tres pelotas pueden interactuar entre sí. Por ello, debes resolver dos tareas:

1. Cálculo del instante de contacto entre dos pelotas. Hay dos opciones:
    a. Determinar el t_{HIT} exacto en que impactan las pelotas, resolviendo algebraicamente la siguiente ecuación:
    $$\|(C_A + tv_A) - (C_B + tv_B)\|^2 = (r_A + r_B)^2$$
    b. Determinar si dos pelotas están solapando en un instante en concreto. Para simular la animación de varias pelotas avánzalas un paso completo (t=1), teniendo en cuenta los obstáculos que encuentren en su camino. Si en esta configuración hay dos pelotas que solapen, pruebas a moverlas sólo la mitad del paso (t=1/2). Si hay dos que siguen solapando prueba con la cuarta parte del paso (t=1/4); de lo contrario prueba a avanzarlas tres cuartas partes del paso (t=3/4). Detén este proceso cuando hayas dado un número suficiente de iteraciones, y quédate con el máximo t antes del contacto.
2. La resolución del impacto entre pelotas.
    Supón que las pelotas A y B están en contacto, y que las posiciones de sus centros, sus velocidades y radios son C_A y C_B, v_A y v_B, y r_A y r_B, respectivamente. Para determinar si realmente van a colisionar se calculan los siguientes valores:

$$n = \frac{\overrightarrow{C_B C_A}}{\left\|\overrightarrow{C_B C_A}\right\|}$$

$$v_{rel}^{-} = n \cdot (v_A - v_B)$$

La colisión se producirá si y solo si $v_{rel}^{-} < 0$. En este caso, las nuevas velocidades se calculan mediante las siguientes expresiones:

$$j = \frac{-2v_{rel}^{-}}{\frac{1}{M_A} + \frac{1}{M_B}}$$

$$v_A' = v_A + \frac{j}{M_A} n$$

$$v_B' = v_B - \frac{j}{M_B} n$$

Puedes suponer que la masa de cada una pelota es directamente proporcional a su radio. Ajusta la constante de proporcionalidad adecuadamente.