Informática Gráfica

Ingeniería en Informática

Curso 08-09. Práctica 3. Versión 1

Carácter: Obligatorio

Fecha de entrega: Miércoles 14 de Enero

Objetivo: colisiones, rebotes, simulación de movimiento

Descripción: se trata de implementar el movimiento lineal de una *pelota* en una habitación cerrada en la que se encuentran varios *obstáculos* 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 cada una de estas dos categorías:

1. Polígonos convexos
2. Segmentos

Las pelotas son círculos que se desplazan con velocidad constante por la habitación. Entre sus atributos se incluyen el polígono regular que la aproxima, 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. Al moverse pueden colisionar con las paredes o con los obstáculos, en cuyo caso la pelota rebotará por reflexión. Por último, la generación de las pelotas tendrá lugar en una determinada posición de la habitación que debe estar suficientemente apartada de las paredes y de los obstáculos, para garantizar que pueden generarse sin problemas. El sentido de cada pelota se calcula de forma aleatoria.

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 segmentos 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. Por ello, incluso los segmentos emplearán el algoritmo de Cyrus-Beck para detectar colisiones en este modo.

Detalles de la implementación: Estructura de la información

Debes estructurar tu código usando entre otras las siguientes clases. 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, determinando si habría colisión y, en caso afirmativo, devolviendo la normal implicada.
3. *`Convexo`* y *`Segmento`* extienden `Obstaculo` y resuelven la colisión contra pelota de forma específica.

4. *`Obstaculo_Recubierto`* contendrá dos obstáculos convexos, 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 el círculo que usamos para dibujarla.

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.
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 `double` en lugar de float también puede resultar beneficioso.
    b. Las paredes de la habitación, pueden considerarse como un objeto de una nueva clase de obstáculos, la de los polígonos simples no convexos, ya que su interior es el exterior de la habitación. 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 polígonos convexos 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 ha agotado 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.
    d. El sentido de la pelota tras el choque se calcula mediante reflexión. Su tamaño debe coincidir con el del antiguo sentido, para así garantizar que la velocidad de la pelota permanece constante.
    e. Siempre debes evitar la penetración parcial o total de la pelota.
4. La generación de números aleatorios se consigue con los métodos `randomize()` y `randon(int num)` de la clase `Math`.

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 habitación cerrada
Etapa II. Una pelota rebotando en habitación cerrada con polígonos convexos
Etapa III. Una pelota rebotando en habitación cerrada con segmentos
Etapa IV. Una pelota rebotando en habitación cerrada con obstáculos recubiertos en modo de no penetración.

Parte Opcional

[+] Incluye en la escena múltiples pelotas que interactúen entre sí. Para ello debes resolver dos tareas:

1. Cálculo del instante de contacto entre dos pelotas.
   Se basa en 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.