



Guion de prácticas

Proyecto Final - Preparación



Metodología de la Programación

Grado en Ingeniería Informática

Prof. David A. Pelta Mochcovsky

Introducción al guion

En este guión se desarrollan dos clases básicas `Vector2D` y `Particula` que luego se utilizarán para el desarrollo del proyecto final. Ambas clases son sencillas y no requieren memoria dinámica. Cada clase se implementará en dos archivos (`.h`, `.cpp`). Recupere los ejemplos de las clases `Circulo` y `Punto2D` para recordar que se indica en cada uno de ellos.

1. Clase `Vector2D`

Para el desarrollo de esta clase debe recuperar sus apuntes de Álgebra o Cálculo. Se propone representar un vector en 2D mediante un par de valores de tipo `float`:

```
class Vector2D{
private:
    float x, y;
}
```

Se pide implementar los siguientes métodos:

- Constructor que recibe dos parámetros `_x`, `_y` de tipo `float` con valor por defecto cero.
- métodos `set/get` para los datos miembro. Puede definir también un método `setXY` para cambiar ambos valores.
- Método `sumar(...)`, recibe un vector como parámetro y se lo suma al vector actual.
- Método `escalar(float)`: multiplica el vector por un valor que se recibe como parámetro.
- Método `modulo()`: devuelve el módulo del vector.
- Método `normalizar()`: normaliza el vector, dividiendo cada una de las componentes por su módulo, convirtiéndolo en un vector unitario.
- Método `distancia(...)`: recibe otro vector y, considerando las coordenadas del vector como puntos en el plano, calcula la distancia euclídea entre ellos.
- Método `toString()`: devuelve los datos del vector como un string con el siguiente formato: `(x,y)`.

Recuerde que si un método no modifica el estado del objeto, debe definirlo como `const`.

2. Clase Particula

Esta clase permite modelizar un objeto que se mueve en un “mundo” bidimensional rectangular, que tiene como vértice superior izquierdo el punto (0,0) y como vértice inferior derecho el punto (MAX_X, MAX_Y). La representación gráfica se muestra en la Fig. 3. La definición propuesta para la clase es la siguiente:

```
class Particula{
private:
    Vector2D pos; //posicion
    Vector2D acel; // aceleracion
    Vector2D veloc; // velocidad
    float radio;
    int tipo;
}
```

Para esta clase, se pide implementar los siguientes métodos:

- Constructor con un parámetro de tipo entero llamado `tipoPart` con valor cero por defecto: si el `tipoPart` es distinto de cero, entonces crea una partícula “estática” con `radio = 3` en la posición (0,0), y con los vectores de velocidad y aceleración en cero.

Caso contrario (es decir, cuando el constructor se invoca sin parámetro), genera una partícula donde los vectores de posición, velocidad y aceleración se generan al azar. Para ello, considere que los límites para la posición son [(0,0), (MAX_X, MAX_Y)], para la velocidad [-MAX_VEL, MAX_VEL] y para la aceleración [-MAX_ACC, MAX_ACC]. El radio es un valor al azar entre [MIN_R, MAX_R].

Todas estas constantes se definen en un fichero `params.h` que deberá ser incluido en todos los módulos que lo requieran. La Figura 1 muestra el código que puede utilizar para generar números aleatorios en un intervalo determinado.

En ambos casos, el `tipo` se fija en `tipoPart`.

- Constructor con parámetros: recibe todos los parámetros necesarios para crear el objeto en este orden: posición, aceleración, velocidad, radio y tipo. Utilice la lista de inicialización.
- Métodos `set/get` para los datos miembro, salvo el `tipo`.
- Método `mover()`: actualiza el vector posición de la partícula haciendo las siguientes operaciones: 1) suma la aceleración a la velocidad, 2) si alguna componente de la velocidad es mayor que `MAX_VEL` (o menor que `-MAX_VEL`), le asigna `MAX_VEL` (o `-MAX_VEL`), y 3) suma la velocidad a la posición. Debe comprobar que la posición final se mantenga dentro del espacio del “mundo”. Si queda fuera, ajuste la posición al valor límite que corresponda.
- Método `rebotar()`: Si la partícula “choca” contra el borde derecho o el izquierdo del “mundo”, entonces se cambia el signo de la velocidad en la componente x. Si lo hace contra el borde superior o

inferior, se cambia el signo de la velocidad en la componente y del vector correspondiente.

- Método `colision(...)`: devuelve `true` si la partícula actual colisiona con otra que se pasa como parámetro. Devuelve `false` en caso contrario.
- Método `choque(...)`: implementa el choque elástico entre la partícula actual y otra que se pasa como parámetro. Asumiendo que ambas tienen la misma “masa”, el choque elástico implica intercambiar los respectivos vectores de velocidad y aceleración (investigue el por qué).
- Método `toString()`, devuelve los datos de la partícula como un string con el siguiente formato `{(pos_x, pos_y), (vel_x, vel_y), (ace_x, ace_y), radio, tipo}`. Aproveche el método `toString()` de la clase `Vector2D`.

En la Figura 2 puede ver los valores sugeridos para las constantes indicadas previamente.

Recuerde que si un método no modifica el estado del objeto, debe definirlo como `const`.

Importante

Estas instrucciones deben complementarse con las indicaciones dadas durante las sesiones de práctica.

```
float aleatorio(float min, float max) {
    float r = rand() / static_cast<float>(RAND_MAX);
    float rango = max - min;
    return (r * rango + min);
}
```

Figura 1: Código para generar números reales aleatorios entre dos valores `min`, `max`

```
const int MAX_X = 600;
const int MAX_Y = 600;

const int MAX_VEL = 7;
const float MAX_ACC = 2.0;

const float RADIO = 3.0;
const float MIN_R = 3.0;
const float MAX_R = 7.0;

const float EPSILON = 0.01;
```

Figura 2: Valores sugeridos para las constantes utilizadas en la clase `Particula`

.



Figura 3: Sistema de coordenadas para las partículas.