Guion de prácticas

Proyecto Final - Parte I

DECSAI

Metodología de la Programación

Curso 2024/2025

Introducción

Este guion contiene las instrucciones para el desarrollo de la primera parte del proyecto final de la asignatura. Se requiere disponer de la implementación de las clase `Vector2D` y `Particula` planteadas en el guión anterior. Se implementará una clase `ConjuntoParticulas` para almacenar y manipular un conjunto de objetos de la clase `Particula`.

Métodos adicionales para `Particula`

Agregue el siguiente método a la clase `Particula`:

- `wrap()`: recuerde que la partícula se mueve en un "mundo" rectangular. Este método permite que, cuando la partícula llega a uno de los lados, aparezca por el lado contrario (esto se consigue cambiando la coordenada correspondiente en el `Vector2D` asociado a la posición). Tenga en cuenta el radio del objeto para decidir la nueva posición.

Clase ConjuntoParticulas

Se define la siguiente clase para almacenar un conjunto de objetos de tipo `Particula`:

```
class ConjuntoParticulas{
private:
  Particula *set = nullptr;   // un array de particulas
  int capacidad = 0;          // capacidad del array
  int utiles = 0;             // posiciones ocupadas
```

Respecto a los métodos, serán necesarios (por el momento):

1. Constructor con un parámetro por defecto `n = 0`. Si `n >0`, reserva memoria y crea un conjunto con `n` partículas generadas al azar (recuerde la funcionalidad del constructor de la clase `Particula`). Los datos miembro `capacidad` y `utiles` se fijan a `n`.

2. Constructor de copia.

3. Destructor: libera la memoria reservada.

4. Métodos `getUtiles` y `getCapacidad` para elementos útiles y capacidad del conjunto. Desde el punto de vista del diseño, este último método no es necesario. Se pide solamente para su uso en las pruebas de la clase asociadas a la redimensión del array.

5. `agregar`: agrega una partícula al conjunto. Si no hay espacio suficiente, se aumenta la capacidad del array `TAM_BLOQUE` unidades. La constante `TAM_BLOQUE = 3` se define en el fichero `ConjuntoParticulas.h`.

6. `borrar`: Elimina la partícula de una posición dada. Si la posición es inválida, no hace nada. Para borrar, reemplaza la partícula a borrar con aquella que se encuentra en la última posición util y reduce el valor de elementos útiles. Si luego del borrado, se verifica que `(capacidad - utiles) >TAM_BLOQUE`, entonces debe redimensionar el conjunto para que la nueva capacidad coincida con el número de partículas.

7. `obtener`: devuelve una referencia a la partícula de una posición dada.

8. `reemplazar`: en una posición `pos`, almacena una partícula `part` dada, sobreescribiendo la existente. Los parámetros se reciben en ese orden.

9. `mover (int tipo = 0)`: en función del valor del parámetro, mueve cada partícula del conjunto usando diferentes opciones:

   - `tipo = 0`, a cada partícula se le aplica el método `mover`.
   - `tipo = 1`, se le aplica el método `mover`, seguido de `rebotar`.
   - `tipo = 2`, se le aplica el método `mover`, seguido de `wrap`.

   En cualquier otro caso, se considera `tipo = 0`.

10. `gestionarColisiones`: Evalúa todos los pares de partículas. Si un par de partículas colisionan, entonces "chocan" con el método correspondiente de la clase `Particula`. Asuma que una partícula i solo puede chocar una vez con otra partícula j, con $(j > i)$.

11. `toString()`: muestra la capacidad y el número de partículas del conjunto y luego, una partícula por línea (utilizando el método de la clase `Particula`).

Observaciones

Proximamente se publicarán los ficheros de prueba y las instrucciones para la entrega.

Sea cuidadosa/o con la implementación. Tenga en cuenta la indentación, piense el tipo de los parámetros que reciben y devuelven los métodos, no repita código y utilice métodos privados para evitarlo (por ejemplo para `reservarMemoria, liberarMemoria, redimensiona`,etc.).

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