
# Guion de prácticas

*Proyecto Final - Parte B*


Metodología de la Programación

Curso 2018/2019

# Introducción

Este guion contiene las instrucciones para el desarrollo de la segunda parte del proyecto final de la asignatura. Se requiere disponer (al menos) de la implementación de las clase `ConjuntoParticulas` planteada en el guión anterior. Al ser esta una clase que utiliza memoria dinámica, ha tenido que completar la implementación de la clase con la sobrecarga del operador de asignación, constructor de copia y destructor.

# Extensión de la `Clase ConjuntoParticulas`

Agregue a la clase, un constructor por defecto. Debe crear un conjunto de partículas vacío.

Luego, para probar el funcionamiento de los métodos implementados, utilice el código de la Fig. 2 como guía. Evalúe el funcionamiento correcto de la memoria dinámica utilizando la herramienta `valgrind`.

# Clase Simulador

Esta clase se encargará de mover y visualizar un conjunto de partículas utilizando la biblioteca `MiniWin`. Dependiendo de la complejidad del sistema que querramos simular, la clase se puede extender.

Construiremos una simulación donde hay objetos móviles y objetos fijos (partículas con velocidad 0).

Se propone la siguiente definición para los datos miembro:

```
class Simulador{
private:
ConjuntoParticulas moviles;
ConjuntoParticulas fijas;
// tamaño de la pantalla
int ancho, alto;
```

Respecto a los métodos, serán necesarios:

1. Constructor con parámetros: recibe dos objetos de la Clase `ConjuntoParticulas` y dos enteros representando el ancho y alto de la ventana para la visualización. Asigna los valores en la lista de inicialización. El método crea la ventana de `MiniWin`.

2. Destructor: cierra la ventana de `MiniWin`.

3. Set/Get: para los conjuntos de partículas.

4. `Pintar()`: pinta las partículas `moviles` y `fijas`. Implemente métodos privados para pintar las partículas en función de si son fijas o móviles (por ejemplo, las fijas con rectángulos y las móviles con círculos). Llama a las funciones `refresca` y `espera` de `MiniWin`. El tiempo de espera se recibe como parámetro.

```
int main() {
int ancho = 600, alto = 400;
srand(time(0));
ConjuntoParticulas parts(15);
ConjuntoParticulas obstaculos(5);
ConjuntoParticulas aux;

// ponga a 0 las velocidades de las particulas
// en el objeto obstaculos
Simulador mySim(parts, obstaculos, ancho, alto);

while (tecla() != ESCAPE){
  mySim.Step();
  mySim.Pintar(25);
  aux = gestor.GetParticulasFijas();
  Particula p;
  aux.AgregaParticula(p);
  mySim.SetParticulasFijas(aux);
}

return 0;
}
```

Figura 1: Modelo de programa para probar la clase `Simulador`.

5. `Step()`: realiza un paso de la simulación. Inicialmentesolo mueve y gestiona las colisiones entre las partículas del dato miembro `moviles`:
   `moviles.Mover(ancho, alto);`
   `moviles.GestionarChoques();`

   Por el momento, las partículas fijas no tendrán ningún comportamiento.

Para probar la funcionalidad de las clases implementadas, puede seguir el modelo de programa que se muestra en la Fig. 1.

**Método** `Rebotes()`

Implemente un método `Rebotes()` en la `Clase Simulador` que gestione las colisiones entre los objetos fijos y los móviles. Si una partícula móvil $p_1$ "choca" contra otra fija $p_2$, la colisión solo afectará a $p_1$, cambiando su velocidad para simular el rebote. Modifique la clase `Particula` si lo considera necesario.

Extienda el método `Step()`, para que haga la llamada al método `Rebotes()` y repita las pruebas.

```
int main() {
srand(time(0));
ConjuntoParticulas base(3);

// constructor de copia
ConjuntoParticulas nuevo(base);
nuevo.Mover(800, 800);

// constructor por defecto
ConjuntoParticulas otro;

// sobrecarga de asignacion y +
otro = nuevo + base;
otro.Mover(800, 800);

// mostrar
cout << "\n Conjunto Inicial" << endl;
cout << base;
cout << "\n Conjunto Movido 1" << endl;
cout << nuevo;
cout << "\n Conjunto Movido 2" << endl;
cout << otro;
cout << endl;
return 0;
}
```

Figura 2: Modelo de programa para probar la clase `ConjuntoParticulas` completa.

## Número variables de Partículas Móviles

En este apartado debe implementar un comportamiento que dé lugar a la aparición/desaparición de nuevas partículas (fijas y/o móviles). Algunas posibilidades son:

1. Si una partícula móvil $p_1$ colisiona contra un obstáculo $p_2$ del mismo color, entonces se genera una nueva partícula móvil $p_1^+$ que tiene la misma velocidad y color que $p_1$ (y la posición se genera al azar). Si $p_1$ y $p_2$ tienen colores diferentes, entonces $p_1$ desaparece.

2. Cambiar la gestión de las colisiones en la `Clase ConjuntoParticulas` para que las partículas reboten, aparezcan o desaparezcan (por ejemplo, en base a los colores).

3. Al presionar alguna tecla que decida, se “regenera” el conjunto de partículas moviles y/o el de fijas.

4. Al presionar una tecla, se agrega una nueva partícula fija/móvil al conjunto correspondiente.

Si tiene alguna propuesta adicional para extender la práctica, consúltela primero con su profesor de prácticas.

# Material a entregar

Las instrucciones para la entrega se publicarán próximamente.