Guion de prácticas

Proyecto Final, Parte 1

DECSAI

Metodología de la Programación

Grado en Ingeniería Informática

Prof. David A. Pelta

Introducción al guion

En este guion volveremos a utilizar Clases y compilación separada.

Se trabajará a partir de una clase básica para representar una partícula, que contiene las coordenadas en el plano, la velocidad y un radio (todas definidas como float). La implementación de esta clase está disponible en PRADO. Posteriormente, debe construir una clase ConjuntoParticulas que permitirá almacenar objetos de la clase Partícula utilizando un array dinámico.

Finalmente, utilizando una biblioteca gráfica que permitirá visualizar las partículas del conjunto deberá implementar un minijuego.

Parte 1: Conjunto de Partículas

Tiene disponible en PRADO la implementación de la clase `Particula` en el fichero `particula.zip`. El fichero `Particula.h` contiene la descripción de cada uno de los métodos disponibles.

A partir de dicha clase, se pide implementar una clase `ConjuntoParticulas` con la siguiente estructura:

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

Respecto a los métodos, serán necesarios:

1. Constructor sin parámetros: inicializa el array con `MIN_SIZE = 5` partículas. Defina la constante en el fichero `ConjuntoParticulas.h`.
2. Constructor con parámetro: recibe un entero representando la capacidad C del conjunto e inicializa el array con C partículas.
3. Destructor: libera la memoria reservada.
4. Métodos `Get` para capacidad y elementos útiles del conjunto.
5. `AgregaParticula`: agrega una partícula al conjunto. Si no hay espacio suficiente, se redimensiona el array extendiendo su capacidad en `TAM_BLOQUE` unidades. Defina la constante en el fichero `ConjuntoParticulas.h`.
6. `BorraParticula`: haciendo desplazamientos a izquierda, elimina la partícula de una posición dada. Si luego del borrado, se verifica que `(capacidad - utiles) >TAM_BLOQUE`, entonces debe redimensionar el conjunto para que la nueva `capacidad = capacidad - TAM_BLOQUE`.
7. `ObtieneParticula`: devuelve la partícula de una posición dada.
8. `ReemplazaParticula`: reemplaza la partícula de una posición dada con otra.

9. `Mover`: recibe como parámetros los valores de `ancho` y `alto` en los que las partículas se pueden mover. Luego, mueve cada partícula usando el método correspondiente.

10. `Rebotar`: recibe como parámetros los valores de `ancho` y `alto` en los que las partículas se pueden mover. Luego, aplica a cada partícula el método `RebotaBordes`.

11. `Mostrar`: muestra por pantalla los valores de `capacidad` y `utiles`, y luego los datos de cada partícula (utilice el método `ToString` de la clase `Particula`).

Más adelante se agregarán la sobrecarga de operadores y el constructor de copia.

Tareas

El programa debe estar correctamente modularizado y organizado en las carpetas `src, include`. Cada clase debe tener un fichero .h y el correspondiente .cpp. Sea cuidadoso/a con la implementación. Tenga en cuenta la indentación, piense cuidadosamente el tipo de los parámetros que reciben y devuelven los métodos, no repita código y utilice métodos privados para evitarlo (como `reservarMemoria, liberarMemoria, redimensiona`, etc.).

Utilice el código provisto en PRADO (fichero `pruebaConjunto.cpp`) para probar la clase `ConjuntoParticulas`. Debe comprobar que la ejecución del programa utilizando `Valgrind`, no reporta errores en ninguna de las funciones de test.

Parte 2: Captura Ovnis

En esta sección se describe un minijuego que servirá de ejemplo de aplicación de las clases implementadas. Tiene disponible en PRADO un video de demostración donde puede ver lo que se quiere obtener.

Paso 1: instalación de raylib

Para mostrar las partículas utilizaremos la biblioteca raylib [1]. Es una biblioteca orientada a la programación de videojuegos muy completa, de la que solo utilizaremos sus funcionalidades más básicas. De manera resumida, los pasos son para instalar la biblioteca son los siguientes. [2]

En primer lugar, se instalan bibliotecas adicionales:

[1] https://www.raylib.com/

[2] Puede encontrar una descripción completa en https://github.com/raysan5/raylib/wiki/Working-on-GNU-Linux

```
  sudo apt install libasound2-dev mesa-common-dev
libx11-dev libxrandr-dev libxi-dev xorg-dev
libgl1-mesa-dev libglu1-mesa-dev
```

Luego se descarga el código, "clonando" el proyecto desde `GitHub`:

```
git clone https://github.com/raysan5/raylib.git raylib
```

Se compila para construir la biblioteca:

```
  cd raylib/src/
make PLATFORM=PLATFORM_DESKTOP RAYLIB_LIBTYPE=SHARED
```

y se instala en el sistema:

```
sudo make install RAYLIB_LIBTYPE=SHARED
```

Para probar que la instalación se ha realizado correctamente, descargue desde PRADO el fichero `ejemplos.zip`. Descomprima el fichero. Compile y ejecute el fichero `testRaylib.cpp` de la siguiente manera:

```
  g++ testRaylib.cpp -lraylib -o test
./test
```

Observe que aparecen tres ficheros adicionales de ejemplo (además del módulo `Particula`). Puede compilar cada uno de ellos haciendo:

```
  g++ ejemplo.cpp Particula.cpp -lraylib -o ejem
./ejem
```

Estudie el código de cada ejemplo pues encontrará los elementos necesarios para el desarrollo del minijuego.

Un elemento importante a tener en cuenta es como se definen las coordenadas en `raylib`. La Figura 1 permite observar que el punto `(0,0)` se encuentra en el extremo superior izquierdo de la pantalla.

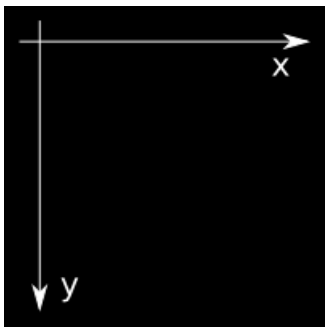

Figura 1: Sistema de coordenadas en `raylib`

NOTA: Para crear un proyecto en NetBeans siga los pasos usuales. Para que se pueda construir un programa que utilice `raylib`, debe indicar un

parámetro adicional en el "linker" (enlazador). En primer lugar, acceda a las propiedades del proyecto, y luego seleccione `linker`. En el apartado `Additional Options`, pinche en el cuadrado de la derecha y en la nueva ventana escriba el texto `-lraylib` en el apartado correspondiente.

Sobre la visualización

Es importante notar que las partículas "no saben nada" de `raylib`. No tiene sentido que una partícula sepa "pintarse". Por eso, para la visualización, utilizaremos funciones externas que reciban como parámetro un objeto de la clase `Particula` (o `ConjuntoParticulas`)y se encarguen de visualizar el objeto.

Estas funciones se incorporarán a un módulo `pintar` que contenga estas dos funciones (por el momento):

```
void pintarParticula(const Particula \& p, Color c) {
// funcion de raylib
 DrawCircle(p.GetX(), p.GetY(), p.GetRadio(), c);
}

void pintarConjunto(const ConjuntoParticula \& cp, Color c) {
 int N = cp.GetCardinal();
 for(int i = 0; i < N; i++)
  pintarParticula(cp.ObtieneParticula(i), c);
}
```

Pasos para la implementación del juego

Se sugiere desarrollar gradualmente la implementación, planteando las siguientes tareas

1. Cree un conjunto `ovnis` de `N` partículas generadas al azar (el valor `N` es un parámetro a la función `main`). En cada ciclo de actualización, aplique el método `Mover` al conjunto y luego muestre las partículas utilizando la función correspondiente del módulo `pintar`.

2. Ahora, cree una partícula `base` con los siguientes datos (`x = ancho/2.0, y = alto-20.0, dx = 5.0, dy = 0.0`). Esta párticula se moverá solo horizontalmente mediante las teclas `KEY_LEFT`, `KEY_RIGHT` aplicando respectivamente el método `MoverGrid('L', ancho, alto)` y `MoverGrid('R', ancho, alto)`. Visualice el movimiento de `ovnis` y de `base`

3. Finalmente, en cada etapa de actualización, debe evaluar si la `base` colisiona con alguna de las partículas del conjunto `ovnis`. En ese caso, dicha partícula debe borrarse. La evaluación de las colisiones debe empezar desde la última partícula del conjunto para evitar problemas con el borrado.

El programa terminará cuando no queden partículas en el conjunto o el usuario pulse la tecla `KEY_ESC`.

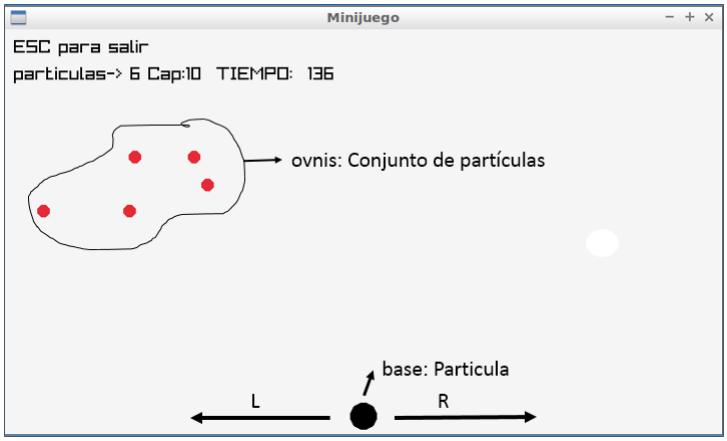

Figura 2: Esquema del juego. Las partículas rojas pertenecen al objeto `ovnis` y la partícula negra es la partícula `base`.

1. Instrucciones para la entrega

Se publicarán en PRADO.

Importante

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