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, la cual contiene las coordenadas en el plano, la velocidad y un radio (todas definidas como variables de tipo `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 algunos mecanismos básicos para luego, desarrollar 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: crea un conjunto vacío, poniendo todo "a cero".
2. Constructor con parámetro: crea un conjunto con N partículas generadas al azar (Recuerde que cuando reserva memoria, automáticamente se llama al constructor por defecto de la clase `Particula`. Asigna N a `capacidad` y `utiles`.
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=3` 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. ¿ Cuales son las consecuencias de devolver la partícula como una referencia o como una copia?.
8. `reemplazaParticula`: reemplaza la partícula de una posición dada con otra.
9. `moverParticulas`: 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. `mostrarInfo`: 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: Breakout

En esta sección se describe un minijuego inspirado en Breakout que servirá de ejemplo de aplicación de las clases implementadas. Breakout es un videojuego arcade desarrollado por Atari, Inc. y lanzado al mercado el 13 de mayo de 1976. Fue creado por Nolan Bushnell y Steve Bristow, influenciados por el videojuego de 1972 Pong, también de Atari.

Existen infinidad de adaptaciones, pero el contenido básico se muestra en la Fig. 1. En la parte inferior de la pantalla una barra horizontal simula una raqueta de front-tenis que el jugador podía desplazar de izquierda a derecha. En la parte superior se sitúa una banda conformada por rectángulos que simulan ser ladrillos. El otro elemento del juego es una bola que el jugador debe golpear con la raqueta, entonces la pelota

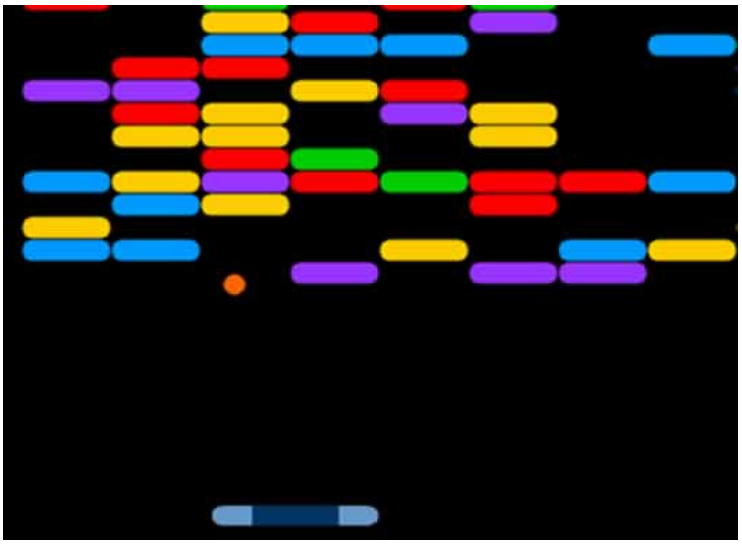

Figura 1: Imagen del juego

rebota hasta pegar en el muro, y los ladrillos tocados por la pelota desaparecen. La pelota vuelve a descender y así sucesivamente. El objetivo del juego es terminar con la pared de ladrillos [1].

Tiene disponible en PRADO un video de demostración donde puede ver lo que se quiere obtener. Todos los elementos del juego se representan mediante objetos de la clase Particula. Dependiendo del rol que desempeñen en el juego, se mostrarán de una forma u otra.

Paso 1: instalación de raylib

Para mostrar las partículas utilizaremos la biblioteca raylib [2]. 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. [3]

En primer lugar, se instalan bibliotecas adicionales:

```
  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
```

[1] https://es.wikipedia.org/wiki/Breakout_(videojuego)
[2] https://www.raylib.com/
[3] Puede encontrar una descripción completa en https://github.com/raysan5/raylib/wiki/Working-on-GNU-Linux

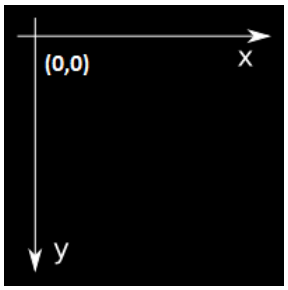

Figura 2: Sistema de coordenadas en `raylib`

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.

La Figura 2 muestra como se definen las coordenadas en `raylib`. Observe que el punto `(0,0)` se encuentra en el extremo superior izquierdo de la pantalla.

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 visua-

lización, 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.getUtiles();
 for(int i = 0; i < N; i++)
  pintarParticula(cp.obtieneParticula(i), c);
}
```

Paso 2: desafíos preliminares

Como preparación para la implementación del juego se sugieren los siguientes desafíos:

Desafío 1

1. Cree un conjunto `gas` de `N` partículas generadas al azar. El valor `N` es un parámetro a la función `main`. En cada ciclo de actualización, mueva y haga rebotar las partículas. Luego debe mostrarlas utilizando la función correspondiente del módulo `pintar`.

2. En cada ciclo de actualización, decremente la velocidad `dy` de cada partícula del conjunto. ¿Qué ocurre?.

Desafío 2

1. Cree un conjunto `pared` de `N` partículas generadas al azar. Estas partículas representan los ladrillos de la pared. El valor `N` es un parámetro a la función `main`. En cada ciclo de actualización, muestre las partículas utilizando la función correspondiente del módulo `pintar`. Decida como quiere pintar los ladrillos.

2. Ahora, cree una partícula aleatoria `bola`. En cada paso, aplique los métodos `mover` y `rebotaBordes`.

3. Compruebe si la `bola` colisiona con algún elemento de la pared. En ese caso, elimine dicho elemento. La evaluación de las colisiones debe empezar desde la última partícula del conjunto para evitar problemas con el borrado. Para simular un rebote simplificado, basta cambiar el signo de dy.

4. El programa terminará cuando no quedan ladrillos en la pared (el conjunto pared está vacío), o el usuario pulse la tecla `KEY_ESC`.

Instrucciones para la entrega

Se publicarán en PRADO.

Importante

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