Tecnología de la Programación de Videojuegos 1

Grado en Desarrollo de Videojuegos · UCM · Curso 2023-2024

Examen ordinario de enero — Parte teórica (3 puntos)

Nombre		Apellido		Puesto	

Ejercicio 1 (1,25 puntos)

Los siguientes fragmentos de código pueden contener errores de compilación, en ejecución o dejar memoria sin liberar. Señala y explica los problemas que encuentres en cada recuadro y modifica lo imprescindible para que el código sea correcto sin cambiar su significado. Si hay instrucciones de entrada/salida, indica qué se imprimiría por pantalla después de corregir los errores.

a

```
int n = 1, m = 7;
int *p = &n, *q = &m;
*q = (*p)++;
++(*q);
cout << n << " " << m;
```

b

```
char c = '%';
char* p = &c;
char* q = new char;
*q = p;
delete p;
delete q;
```

c

```
int** ints = new int*[100];
for (int i = 0; i < 100; ++i)
    ints[i] = new int(3);
delete ints;
```

d

```
#ifndef GAME_H
#define GAME_H
#include "Ship.h"
class Game {
    Ship* player;  // [...]
};
#endif // GAME_H
```

```
#ifndef SHIP_H
#define SHIP_H
#include "Game.h"
class Ship {
    Game* game;  // [...]
};
#endif // SHIP_H
```

e

```
class ShooterAlien : Alien { /* ... */ }
void move(Alien* alien) { /* ... */ }
void shoot(ShooterAlien* shooter) { /* ... */ }

Alien* shooter = new ShooterAlien(game, {200, 400});
move(shooter);
shoot(shooter);
```

f

```
class Contador { int cuenta = 0; public: void suma() { cuenta++; } };

Contador* contador = new Contador;
// button tiene método connect(function<void(void)>)
button.connect(&Contador::suma);
```

Ejercicio 2 (0,5 puntos)

Explica detalladamente e ilustra mediante un ejemplo los problemas que podrían surgir al no sobrescribir el constructor por copia en una clase que utiliza memoria dinámica. Arregla el ejemplo para evitar dichos problemas.

Ejercicio 3 (1,25 puntos)

Sean las siguientes funciones y la siguiente jerarquía de clases (no es necesario corregir nada aquí):

```
void sh(const char* s, int arg) {
    cout << s << "(" << arg << ")\n";
}


class A {
protected:
    int a;
public:
    A(int x) : a(x) { sh("A", a); }
    virtual ~A() { sh("~A", a); }
    void f() { sh("A::f"); }
    virtual void g() { sh("A::g"); }
    virtual void h() { sh("A::h"); }
    virtual void m() = 0;
};
```

```
void sh(const char* s) {
    cout << s << "\n";
}


class B : public A {
    int b;
public:
    B(int a, int b) : A(a), b(b) {
        sh("B", a + b);
    }
    ~B() { sh("~B", a + b); }
    void f() { sh("B::f"); }
    void g() { sh("B::g"); }
    void m() { sh("B::m"); }
};
```

Indica para cada uno de los siguientes bloques de código si se produciría algún error en ejecución (accesos no válidos, fugas de memoria, etc.) y en tal caso cómo resolverlo (sin eliminar instrucciones) y cuál es el texto que se imprime por pantalla durante su ejecución (una vez resueltos los errores).

a

```
A a(4);
B b = a;
b.f();
```

b

```
A* a = new B(1, 7);
a->f();
a->g();
a->h();
a->m();
delete a;
```

c

```
B* b = new B(2, 3);
b->f();
b->g();
b->h();
b->m();
delete b;
```

d

```
A* a = new B(0, 0);
B* b1 = static_cast<B*>(a);
B* b2 = dynamic_cast<B*>(a);
(*b1).m();
b2->m();
```

e

```
A** as = new A*[3];
B b2(2, 0);
as[0] = new B(1, 0);
as[1] = &b2;
as[2] = as[0];
for (int i = 0; i <= 3; ++i)
    as[i]->g();
```

Tecnología de la Programación de Videojuegos 1

Grado en Desarrollo de Videojuegos · UCM · Curso 2023-2024

Examen ordinario de enero — Parte práctica (7 puntos)

En los siguientes ejercicios independientes se extiende la funcionalidad de la práctica 3 implementando nuevos objetos y estados del juego como se indica. A través del icono «Publicación docente» del escritorio se puede acceder a una demo en video para cada uno de los ejercicios.

Ejercicio 1 (5 puntos)

El dron kamikaze es un nuevo tipo de arma alienígena que aparece por el borde superior de la escena y avanza en dirección al cañón para inmolarse colisionando contra él. Su vector velocidad cambia dinámicamente para apuntar a la posición del cañón, pero solo se actualiza una vez cada dos segundos y no vuelve a actualizarse cuando llega a la altura del cañón. Esta pequeña falta de agilidad permite que el cañón pueda esquivar al kamikaze, que se destruirá al abandonar la escena por la parte inferior. El dron también se destruirá al recibir el impacto de un láser del cañón, al chocar contra el cañón o contra un búnker, causando 2 puntos de daño al objeto impactado.

(1). Crea una clase `Weapon` para representar un arma del juego con métodos para consultar sus características. Adapta las clases `Laser` y `Bomb` para que sean subclases de `Weapon` y actualiza los métodos `PlayState::damage` y `SceneObject::hit` para que reciban un arma como argumento.

(2). Extiende `PlayState` con un método `getCannonPosition` para obtener la posición actual del cañón.

(3). Crea una nueva clase `DronKamikaze` como subclase de `Weapon` e implementa la funcionalidad descrita en el primer párrafo. La textura del dron debe apuntar a la dirección del movimiento en todo momento. Utiliza para ello la sobrecarga adecuada de `renderFrame` en `Texture` y recuerda que, si (x, y) es la velocidad, el ángulo con respecto a la horizontal es $180/\pi \cdot \text{atan}(x/y)$ (atan está disponible en la cabecera `cmath`).

(4). Extiende la clase `Mothership` para que genere drones kamikazes en posiciones aleatorias sobre el borde superior de la escena y a intervalos de tiempo igualmente aleatorios.

(5). Haz el resto de cambios que sean necesarios para conseguir la funcionalidad descrita.

Nota: algunos cambios en el apartado (1) están distribuidos por todo el código. Una posibilidad para abordarlos es cambiar los argumentos de la primera declaración virtual y navegar por los errores corrigiéndolos.

Ejercicio 2 (2 puntos)

Con el fin de mejorar la ambientación del juego, queremos que se muestre un texto introductorio al iniciar partida (`intro.png` en el material del examen), que se irá desplazando verticalmente hacia arriba a una velocidad que permita leerlo. Cuando se haya mostrado todo el texto, o si el jugador quiere saltarse la introducción antes pulsando cualquier tecla, se pasará al estado normal de juego. Utilizaremos un nuevo estado genérico `ScrollingState` que recibirá la imagen a mostrar (típicamente alargada en vertical) y el estado del juego que se habrá de iniciar a continuación. Con esta clase general podríamos, si quisiéramos, implementar una pantalla final de créditos cambiando solo la imagen.

(1). Añade un método `void renderRect(const SDL_Rect& source)` a la clase `Texture` que imprima el rectángulo dado de la textura cubriendo toda la ventana.

(2). Crea un nuevo estado del juego `ScrollingState` que reciba como argumentos una textura y otro estado del juego. Su funcionamiento será el que indica el párrafo anterior.

(3). Utilizando lo anterior, haz que aparezca una pantalla deslizante con la imagen `intro.png` al iniciar una nueva partida desde el menú principal y antes de que se muestre el estado normal del juego.

Instrucciones

- Se valorará la corrección, la eficiencia y la claridad del código.
- La entrega se realiza pulsando en el icono del escritorio «Exámenes en Labs...», y posteriormente, utilizando el programa que se abre, arrastrando los ficheros a entregar a la carpeta de vuestro puesto (en el lado derecho).

- Escribe un archivo `info.txt` en el directorio de la solución con tu nombre completo, DNI, puesto de laboratorio y explicaciones detalladas de qué has conseguido hacer o qué no, de qué clases has añadido y qué clases/métodos de la práctica 3 has modificado y cómo.
- Debes entregar un fichero `NombreApellidos.zip` con la carpeta de la solución limpia de archivos temporales y sin incluir la carpeta oculta `.vs` ni las carpetas `SDL2`, `SDL2_image` o `SDL2_ttf`. Para generar este archivo, ejecuta en Visual Studio la opción «limpiar solución», abre la carpeta de la solución (donde está el archivo `.sln`) con el explorador de archivos, selecciona las carpetas y archivos relevantes (entre ellos `info.txt`), haz click con el botón derecho en «7-Zip» y añadir a «NombreCarpeta.zip», que luego podrás renombrar.