Actividad Guiada: Refactorización en C++

Daniel Garcia Brun

1-DAM 13/01/2025

Contenido

Introducción

En esta actividad, aprenderás el concepto de refactorización, que consiste en mejorar la estructura interna de un programa sin cambiar su funcionalidad externa. El objetivo es convertir un programa básico en C++ en una versión modular, legible y fácil de mantener, siguiendo una progresión por etapas.

Instrucciones Generales

Crearás un programa que simule un combate entre un héroe y un enemigo. Ambos personajes tendrán puntos de vida (Hit Points) y un valor de ataque. El programa deberá controlar el flujo del juego utilizando estructuras condicionales y bucles. Finalmente, refactorizarás el código en varias fases.

Fase 1: Versión inicial (todo en main)

Escribe todo el código en el método main. Utiliza variables globales para los datos del héroe y el enemigo. Puedes reusar el código del videojuego creado en el modulo de programación.

En esta etapa, el código se presenta en su versión más elemental. Todo el proceso del juego se gestiona en el interior de la función. main , empleando factores globales para mantener la vida tanto del héroe como del enemigo.

```
#include <iostream>
using namespace std;

string opcion;
int heroh = 2;
int enemyh = 2;

int main() {
    while (heroh > 0 && enemyh > 0) {
        cout << "Quieres atacar o esquivar?";
        cin >> opcion;

        if (opcion == "atacar" || opcion == "Atacar" || opcion == "ATACAR")
{
            enemyh--;
            cout << "El enemigo pierde un punto de vida" << endl;
            if (enemyh <= 0) {
                cout << "Felicidades has ganado el combate!" << endl;
                break;
            }

            heroh--;
            cout << "Tu enemigo ataca" << endl;
            if (heroh <= 0) {
                cout << "Has sido derrotado y la ciudad destruida" << endl;
                break;
            }
        }
        else if (opcion == "esquivar" || opcion == "Esquivar" || opcion ==
"ESQUIVAR") {
            cout << "Tu enemigo intenta atacarte pero lo esquivas" << endl;
        }
        else {
            cout << "Algo ha salido mal" << endl;
        }
    }

    return 0;
}
```

Fase 2: Refactorización con funciones sin parámetros

Refactoriza el código dividiéndolo en funciones que no usen parámetros, pero trabajen con variables globales.

Ejemplo de funciones:

- `void heroAttackEnemy()`
- `void enemyAttackHero()`
- `void printStatus()`

En esta fase, comenzamos el proceso de mejora la estructura del código al dividirlo en funciones. Cada función lleva a cabo una función específica, lo que facilita la lectura y la observación del código. A pesar de que aún se usan variables globales, la distinción de responsabilidades ayuda a una mejor interpretación del progreso del juego.

```
#include <iostream>
using namespace std;

string opcion;
int heroh = 2;
int enemyh = 2;

void introduccion() {
    cout << "La ciudad esta siendo atacada y no hay mas heroes disponibles,
eres la ultima esperanza. Derrota al villano que esta asolando la ciudad."
<< endl;
}

void ataqueEnemigo() {
    heroh--;
    cout << "Tu enemigo ataca." << endl;
    if (heroh <= 0) {
        cout << "Has sido derrotado y la ciudad destruida." << endl;
    }
}

void ataqueHeroe() {
    enemyh--;
    cout << "El enemigo pierde un punto de vida." << endl;
    if (enemyh <= 0) {
        cout << "Felicidades, has ganado el combate!" << endl;
    }
}

void status() {
    cout << "A tu enemigo le quedan " << enemyh << " puntos de vida." <<
endl;
    cout << "A tu heroe le quedan " << heroh << " puntos de vida." << endl;
}

void opciones() {
    cout << "Quieres atacar, esquivar, ver el estado o huir?" << endl;
    cin >> opcion;
}

void logicacombate() {
    while (heroh > 0 && enemyh > 0) {
        opciones();
```

```
        if (opcion == "atacar" || opcion == "Atacar" || opcion == "ATACAR")
{
            ataqueHeroe();
            if (enemyh <= 0) {
                break;
            }
            ataqueEnemigo();
            if (heroh <= 0) {
                cout << "Has perdido, pero no pasa nada, inténtalo de
nuevo!" << endl;
                break;
            }
        }
        else if (opcion == "esquivar" || opcion == "Esquivar" || opcion ==
"ESQUIVAR") {
            cout << "Tu enemigo intenta atacarte, pero lo esquivas." <<
endl;
        }
        else if (opcion == "estado" || opcion == "Estado" || opcion ==
"ESTADO") {
            status();
        }
        else if (opcion == "huir" || opcion == "Huir" || opcion == "HUIR") {
            cout << "Huyes; en consecuencia, la ciudad ha sido destruida y
miles de personas masacradas." << endl;
            break;
        }
        else {
            cout << "Opción no valida. Intentalo de nuevo." << endl;
        }
    }
}

int main() {
    introduccion();
    logicacombate();
    return 0;
}
```

Fase 3: Refactorización con parámetros por valor

Elimina las variables globales y utiliza parámetros por valor en las funciones.

En este punto, eliminamos las variables globales y comenzamos a pasar parámetros a las funciones. Esto mejora la administración y permite que cada función funcione con sus propios datos, reduciendo así el peligro de fallos. También, la aplicación de parámetros por valor ayuda a que el código sea más predecible.

```
#include <iostream>
using namespace std;

void introduccion(string mensaje) {
    cout << mensaje << endl;
}

void ataqueEnemigo(int& heroh) {
    heroh--;
    cout << "Tu enemigo ataca." << endl;
    if (heroh <= 0) {
        cout << "Has sido derrotado y la ciudad destruida." << endl;
    }
}

void ataqueHeroe(int& enemyh) {
    enemyh--;
    cout << "El enemigo pierde un punto de vida." << endl;
    if (enemyh <= 0) {
        cout << "Felicidades, has ganado el combate!" << endl;
    }
}

void status(int enemyh, int heroh) {
    cout << "A tu enemigo le quedan " << enemyh << " puntos de vida." <<
endl;
    cout << "A tu héroe le quedan " << heroh << " puntos de vida." << endl;
}

void opciones(string& opcion) {
    cout << "¿Quieres atacar, esquivar, ver el estado o huir?" << endl;
    cin >> opcion;
}

void logicacombate(int heroh, int enemyh) {
    string opcion;
    while (heroh > 0 && enemyh > 0) {
        opciones(opcion);
        if (opcion == "atacar" || opcion == "Atacar" || opcion == "ATACAR")
{
            ataqueHeroe(enemyh);
            if (enemyh <= 0) {
                break;
            }
            ataqueEnemigo(heroh);
            if (heroh <= 0) {
                cout << "Has perdido, pero no pasa nada, inténtalo de
nuevo!" << endl;
                break;
            }
        }
```

```
        else if (opcion == "esquivar" || opcion == "Esquivar" || opcion ==
"ESQUIVAR") {
            cout << "Tu enemigo intenta atacarte, pero lo esquivas." <<
endl;
        }
        else if (opcion == "estado" || opcion == "Estado" || opcion ==
"ESTADO") {
            status(enemyh, heroh);
        }
        else if (opcion == "huir" || opcion == "Huir" || opcion == "HUIR") {
            cout << "Huyes; en consecuencia, la ciudad ha sido destruida y
miles de personas masacradas." << endl;
            break;
        }
        else {
            cout << "Opción no válida. Inténtalo de nuevo." << endl;
        }
    }
}

int main() {
    string mensaje = "La ciudad está siendo atacada y no hay más héroes
disponibles, eres la última esperanza. Derrota al villano que está asolando
la ciudad.";
    int heroeVida = 2;
    int enemigoVida = 2;

    introduccion(mensaje);
    logicacombate(heroeVida, enemigoVida);

    return 0;
}
```

Fase 4: Refactorización con valores de retorno

Elimina las variables globales y utiliza parámetros por valor en las funciones.

En esta fase, se establecen retornos en las funciones de ataque., lo que facilita el control de los cambios en el juego. La eliminación de variables globales y la aplicación de valores de retorno aportan a una mayor seguridad y estructuración en el código.

```
#include <iostream>
using namespace std;

string opcion;
int heroh = 2;
int enemyh = 2;

void introduccion() {
    cout << "La ciudad está siendo atacada y no hay más héroes disponibles,
eres la última esperanza. Derrota al villano que está asolando la ciudad."
<< endl;
}

int ataqueEnemigo(int heroh) {
    heroh--;
    cout << "Tu enemigo ataca." << endl;
    if (heroh <= 0) {
        cout << "Has sido derrotado y la ciudad destruida." << endl;
    }
    return heroh;
}

int ataqueHeroe(int enemyh) {
    enemyh--;
    cout << "El enemigo pierde un punto de vida." << endl;
    if (enemyh <= 0) {
        cout << "Felicidades, has ganado el combate!" << endl;
    }
    return enemyh;
}

void status(int heroh, int enemyh) {
    cout << "A tu enemigo le quedan " << enemyh << " puntos de vida." <<
endl;
    cout << "A tu héroe le quedan " << heroh << " puntos de vida." << endl;
}

string opciones() {
    cout << "¿Quieres atacar, esquivar, ver el estado o huir?" << endl;
    cin >> opcion;
    return opcion;
}

void logicacombate(int heroh, int enemyh) {
    while (heroh > 0 && enemyh > 0) {
        string opcion = opciones();
        if (opcion == "atacar" || opcion == "Atacar" || opcion == "ATACAR")
{
            enemyh = ataqueHeroe(enemyh);
            if (enemyh <= 0) {
                break;
            }
            heroh = ataqueEnemigo(heroh);
```

```
            if (heroh <= 0) {
                cout << "Has perdido, pero no pasa nada, inténtalo de
nuevo!" << endl;
                break;
            }
        }
        else if (opcion == "esquivar" || opcion == "Esquivar" || opcion ==
"ESQUIVAR") {
            cout << "Tu enemigo intenta atacarte, pero lo esquivas." <<
endl;
        }
        else if (opcion == "estado" || opcion == "Estado" || opcion ==
"ESTADO") {
            status(heroh, enemyh);
        }
        else if (opcion == "huir" || opcion == "Huir" || opcion == "HUIR") {
            cout << "Huyes; en consecuencia, la ciudad ha sido destruida y
miles de personas masacradas." << endl;
            break;
        }
        else {
            cout << "Opción no válida. Inténtalo de nuevo." << endl;
        }
    }
}

int main() {
    introduccion();
    logicacombate(heroh, enemyh);
    return 0;
}
```

Fase 5: Refactorización final con paso por referencia

Introduce el uso de estructuras y referencias para hacer el código más robusto y modular.

En la etapa final, se implementa la aplicación de estructuras para mostrar a los personajes. No solo mejora la organización de los datos, sino que también facilita una gestión más precisa de las propiedades de cada personaje. La implementación de pasos por referencia en las funciones aumenta la eficacia y la claridad del código, lo que lo hace más sencillo de conservar y ampliar en el futuro.

Se añaden estructuras como por ejemplo Personaje para cada personaje. Esta estructura permite almacenar su nombre y la cantidad de vida de cada personaje, manteniendo toda la información junta y fácilmente accesible:

```
#include <iostream>
using namespace std;

struct Personaje {
    string nombre;
    int vida;
};

void introduccion() {
    cout << "La ciudad está siendo atacada y no hay más héroes disponibles,
eres la última esperanza. Derrota al villano que está asolando la ciudad."
<< endl;
}

void ataque(Personaje& atacante, Personaje& objetivo, const string& mensaje)
{
    objetivo.vida--;
    cout << mensaje << endl;
    if (objetivo.vida <= 0) {
        cout << (objetivo.nombre == "Villano" ? "Felicidades, has ganado el
combate" : "Has sido derrotado y la ciudad destruida") << endl;
    }
}

void status(const Personaje& heroe, const Personaje& villano) {
    cout << "A tu enemigo le quedan " << villano.vida << " puntos de vida."
<< endl;
    cout << "A tu héroe le quedan " << heroe.vida << " puntos de vida." <<
endl;
}

string opciones() {
    string opcion;
    cout << "¿Quieres atacar, esquivar, ver el estado o huir?" << endl;
    cin >> opcion;
    return opcion;
}

void logicacombate(Personaje& heroe, Personaje& villano) {
    while (heroe.vida > 0 && villano.vida > 0) {
        string opcion = opciones();
        if (opcion == "atacar" || opcion == "Atacar" || opcion == "ATACAR")
{
            ataque(heroe, villano, "El enemigo pierde un punto de vida.");
            if (villano.vida <= 0) break;
```

```
            ataque(villano, heroe, "Tu enemigo ataca.");
            if (heroe.vida <= 0) {
                cout << "Has perdido, pero no pasa nada, inténtalo de
nuevo." << endl;
                break;
            }
        }
        else if (opcion == "esquivar" || opcion == "Esquivar" || opcion ==
"ESQUIVAR") {
            cout << "Tu enemigo intenta atacarte, pero lo esquivas." <<
endl;
        }
        else if (opcion == "estado" || opcion == "Estado" || opcion ==
"ESTADO") {
            status(heroe, villano);
        }
        else if (opcion == "huir" || opcion == "Huir" || opcion == "HUIR") {
            cout << "Huyes; en consecuencia, la ciudad ha sido destruida y
miles de personas masacradas." << endl;
            break;
        }
        else {
            cout << "Opción no válida. Inténtalo de nuevo." << endl;
        }
    }
}

int main() {
    Personaje heroe = { "Héroe", 2 };
    Personaje villano = { "Villano", 2 };

    introduccion();
    logicacombate(heroe, villano);
    return 0;
}
```