Actividad Guiada: Refactorización en C++

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.

Fases de la Actividad

Fase 1: Versión inicial (todo en main)

```
#include <iostream>
#include <random>

using namespace std;

int main() {
    int heroHP = 100, heroAttack = 15;
    int enemyHP = 80, enemyAttack = 12;

    random_device rd;
    mt19937 gen(rd());

    cout << "A battle begins between the Hero and the Enemy!" << endl;

    while (heroHP > 0 && enemyHP > 0) {
        uniform_int_distribution<> heroDis(1, heroAttack);
        int danioHeroe = heroDis(gen);
        enemyHP -= danioHeroe;
        cout << "Hero attacks and deals " << danioHeroe << " damage."
<< endl;
        if (enemyHP <= 0) {
            cout << "Enemy is defeated! Hero wins!" << endl;
            break;
        }
        cout << "Enemy's remaining HP: " << enemyHP << endl;

        uniform_int_distribution<> enemyDis(1, enemyAttack);
        int danioEnemigo = enemyDis(gen);
        heroHP -= danioEnemigo;
        cout << "Enemy attacks and deals " << danioEnemigo << "
damage." << endl;
        if (heroHP <= 0) {
            cout << "Hero is defeated! Enemy wins!" << endl;
            break;
        }
        cout << "Hero's remaining HP: " << heroHP << endl;
    }

    cout << "\nThe battle is over." << endl;
```

```
    return 0;
}
```

Fase 2: Refactorización con funciones sin parámetros

```
#include <iostream>
#include <random>

using namespace std;

int heroHP = 100, heroAttack = 15;
int enemyHP = 80, enemyAttack = 12;

int generarDaño(int maxDaño) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, maxDaño);
    return dis(gen);
}

void turnoHeroe() {
    int danioHeroe = generarDaño(heroAttack);
    enemyHP -= danioHeroe;
    cout << "Hero attacks and deals " << danioHeroe << " damage." <<
endl;
    if (enemyHP <= 0) cout << "Enemy is defeated! Hero wins!" << endl;
    else cout << "Enemy's remaining HP: " << enemyHP << endl;
}

void turnoEnemigo() {
    int danioEnemigo = generarDaño(enemyAttack);
    heroHP -= danioEnemigo;
    cout << "Enemy attacks and deals " << danioEnemigo << " damage."
<< endl;
    if (heroHP <= 0) cout << "Hero is defeated! Enemy wins!" << endl;
    else cout << "Hero's remaining HP: " << heroHP << endl;
}

void batalla() {
    while (heroHP > 0 && enemyHP > 0) {
        cout << "\n--- New Turn ---" << endl;
        turnoHeroe();
        if (enemyHP <= 0) break;
        turnoEnemigo();
        if (heroHP <= 0) break;
    }
    cout << "\nThe battle is over." << endl;
}

int main() {
    batalla();
    return 0;
}
```

Fase de contexto: Versión final

```
#include <iostream>
```

```
#include <random>

using namespace std;

int generarDaño(int maxDaño) {
    random_device rd;
    mt19937 gen(rd()); //"mt199937" obligatorio para generar un número
random
    uniform_int_distribution<> dis(1, maxDaño); //Dis crea uan
distribución uniforma entre 1 y el máxDaño
    return dis(gen);
}

int turnoHero(int& heroHP, int& enemyHP, int heroAttack) {
    int dañoHero = generarDaño(heroAttack);
    enemyHP -= dañoHero;
    cout << "\nNuestro colegon se cree Jackie Chan y causa " <<
dañoHero << " de daño." << endl;

    if (enemyHP <= 0) {
        cout << "El malechor huye, vaya pringao" << endl;
    }
    else {
        cout << "Al malechor aun le quedan " << enemyHP << "punticos
de vida" << endl;
    }

    return enemyHP;
}

int turnoEnemi(int& heroHP, int& enemyHP, int enemyAttack) {
    int monsterAttack = generarDaño(enemyAttack);
    heroHP -= monsterAttack;
    cout << "\nEl malechor mos mete una que nos hace " <<
monsterAttack << " de daño." << endl;

    if (heroHP <= 0) {
        cout << "Nos han ganau, nos quedamos sin movil" << endl;
    }
    else {
        cout << "A mi colegon le quedan " << heroHP << " puntos de
vida" << endl;
    }

    return heroHP;
}

void batalla() {
    int heroHP = 100;
    int heroAttack = 15;

    int enemyHP = 80;
    int enemyAttack = 12;

    cout << "Nuestro colegon va por la rambla y se encuentra un
malechor, esto va a ser una batalla epica" << endl;

    while (heroHP > 0 && enemyHP > 0) {

        enemyHP = turnoHero(heroHP, enemyHP, heroAttack);
        if (enemyHP <= 0) {
```

```
            break;
        }

        heroHP = turnoEnemi(heroHP, enemyHP, enemyAttack);
        if (heroHP <= 0) {
            break;
        }
    }

    cout << "\nSa' acabau \n\nINSERT COIN TO CONTINUE" << endl;
}

int main() {
    batalla();
    return 0;
}
```

Fase 3: Refactorización con parámetros por valor

```
int turnoHero(int heroHP, int enemyHP, int heroAttack) {
    int dañoHero = generarDaño(heroAttack);
    enemyHP -= dañoHero;
    cout << "\nNuestro colegon se cree Jackie Chan y causa " << dañoHero << " de daño." << endl;
```

En la función “generarDaño”, necesitamos pasarle el valor del ataque del héroe para ponerlo como límite al generarse el número aleatorio de daño

Fase 4: Refactorización con valores de retorno

```
        cout << "A mi colegon le quedan " <<
    }

    return heroHP;
}
```

Fase 5: Refactorización final con paso por referencia

```
int turnoEnemi(int& heroHP, int& enemyHP, int enemyAttack) {
    int monsterAttack = generarDano(enemyAttack);
    heroHP -= monsterAttack;
    cout << "\nEl malechor mos mete una que nos hace " << monsterAttack <<
```