

	M5 ENTORNOS DE DESARROLLO		
	UF2-OPTIMIZACIÓN DE PROGRAMAS		
REFACTORIZACIÓN DE CODIGO			
Apellidos: Herraiz Foz	Nombre: Angel	Fecha:	13/01/2023

Después de diversos intentos de optimizar mi código de poco más de 400 líneas, me di por vencido. Decidí crear un nuevo proyecto y reestructurarlo de cero. El resultado fue bastante satisfactorio, ya que el código del juego pasó de 467 líneas a tan solo 213.

```

1 #include <iostream>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <windows.h>
5 #include <msystem.h>
6
7 using namespace std;
8
9 /*HERO ATRIBUTES*/
10 string heroname;
11 int heroHP = 1000;
12 int heroDMG = 25;
13 int heromagicDMG;
14 int heroSPD = 25;
15 bool heroalive;
16 int magiccounter = 3;
17
18 /*GOBLIN ATRIBUTES (enemy with crit possibility)*/
19 string goblinname = "Goblin";
20 int goblinHP = 120;
21 int goblinDMG = 60;
22 int goblinSPD = 20;
23 int goblincrit = goblinDMG; /*DMG SAVER TO CRIT POSSIBILITY CREATED IN MAIN FUNCTION*/
24 bool goblinalive;
25
26 /*WIZARD ATRIBUTES (enemy with random magic damage)*/
27 string wizardname = "Wizard";
28 int wizardHP = 250;
29 int wizardDMG;
30 int wizardSPD = 0; /*I personally want the wizard to be always the last one to move*/
31 bool wizardalive;
32
33 /*GENERAL ATRIBUTES*/
34 int turns = 0;
35 int enemyselct = 0;
36 int critpossibility;
37 int attackselect = 0;
38
39 > void gamestarter() { ...
40
41 > int wizardaahero() { ...
42
43 > int goblinaahero() { ...
44
45 > int heroaawizard() { ...
46
47 > int heroaagoblin() { ...
48
49 > void gamestarter2() { ...
50
51 > void attackselecting() { ...
52
53 > int enemyselcting() { ...
54
55 > boolean goblinfightback() { ...
56
57 > boolean heroattackgoblin() { ...
58
59 > boolean heroattackwizard() { ...
60
61 > boolean wizardturn() { ...
62
63 > boolean herofightback() { ...
64
65 > boolean goblinturn() { ...
66
67 > void endgame() { ...
68
69 > int main() { ...

```



```

1 > #include <iostream>
2 using namespace std;
3
4 /*HERO*/
5 int heroHP = 150;
6 int heroDMG;
7 int heroSPD = 2;
8 int heroAtt;
9 int c = 3;
10 int critico;
11 string heroname;
12
13 /*ENEMIGOS*/
14 int enemy1HP = 100;
15 int enemy2HP = 120;
16 int enemy1DMG = 130;
17 int enemy2DMG = 20;
18 int enemy1SPD = 1;
19 int enemy2SPD = 0;
20 string enemy1 = "Lotas el Espadachin";
21 string enemy2 = "Daraen el Caballero";
22
23 /*GENERAL*/
24 bool juego = true;
25
26 > void INICIO() { ...
27
28 > int ESTADOSVIDA() { ...
29
30 > int ESTADOSVEL() { ...
31
32 > int SALIDA() { ...
33
34 > void GRAFICOS() { ...
35
36 > int SELECCIONENEMIGO() { ...
37
38 > void ATAQUE(int& vidaentidadatacada, int danoentidadqueataca, string nombrentidadqueataca, string nombrentidadatacada) { ...
39
40 > void TIPOATAQUEHEROE() { ...
41
42 > void PELEA() { ...
43
44 > int main() { ...

```

Angel Herraiz

Previamente existían cuatro funciones de ataque en mi código. Dos para los enemigos y dos para el héroe. Simplifiqué estas cuatro, en una sola a la que pasarle varios valores por referencia.

```

int wizardaahero() {
    heroHP = heroHP - wizardDMG;
    return heroHP;
}

int goblinaahero() {
    heroHP = heroHP - goblinDMG;
    return heroHP;
}

int heroaawizard() {
    wizardHP = wizardHP - heroDMG;
    return wizardHP;
}

int heroaagoblin() {
    goblinHP = goblinHP - heroDMG;
    return goblinHP;
}

```

Angel Herraiz

==>
==>
==>

```

void ATAQUE(int& vidaentidadatacada, int danoentidadqueataca, string nombrentidadqueataca, string nombrentidadatacada) {
    cout << "Es " << nombrentidadqueataca << " quien blande su espada contra " << nombrentidadatacada << "! (" << danoentidadqueataca << " daña a " << nombrentidadatacada << " por " << danoentidadqueataca << " puntos de vida." << endl;
    vidaentidadatacada = vidaentidadatacada - danoentidadqueataca;
}

```

	M5 ENTORNOS DE DESARROLLO		
	UF2-OPTIMIZACIÓN DE PROGRAMAS		
REFACTORIZACIÓN DE CODIGO			
Apellidos: Herraiz Foz	Nombre: Angel	Fecha:	13/01/2023

Una vez finalicé la optimización de las funciones de ataque, decidí tratar de simplificar la función main. Observé que tenía muchas funciones que se repetían en diversos sitios y finalmente logré compactarlas todas en muchas menos líneas de código.

```
int main() {
    gamestarter();
    /*RPG TURN SYSTEM CREATING*/
    while ((goblinHP >= 1) || (wizardHP >= 1) && (heroHP >= 1)) {
        gamestarter();
        attackselecting();
        /*ENEMY_SELECTOR*/
        enemysselecting();
        /*FIGHT SYSTEM ON GOBLIN*/
        /*IF HERO IS FASTER THAN THE GOBLIN*/
        if (heroSPD >= goblinsPD) {
            heroSPD = 0;
            goblinsPD = 100;

            /*IF YOU SELECT TO ATTACK GOBLIN*/
            heroattackgoblin();
            /*IF YOU SELECT TO ATTACK WIZARD*/
            heroattackwizard();
            /*GOBLIN'S TURN*/
            goblinturn();
            /*WIZARD'S TURN*/
            wizardturn();
        }
        /*IF HERO IS SLOWER THAN THE GOBLIN*/
        else if (heroSPD <= goblinsPD) {
            heroSPD = 100;
            goblinsPD = 0;
            /*THE GOBLIN ATTACKS*/
            goblinturn();
            /*IF U SELECTED GOBLIN*/
            heroattackgoblin();
            /*IF U SELECTED WIZARD*/
            heroattackwizard();
            /*WIZARD'S TURN*/
            wizardturn();
        }
        /*RESET DAMAGE FOR GOBLIN IF THIS TURN MATE ATTACKED CRITICALLY*/
        if (turns > 1) {
            goblinDMG = goblincrit;
        }
        endgame();
    }
}
```

==>
==>
==>

```
int main() {
    INICIO();
    while (juego) {
        GRAFICOS();
        TIPOATAQUEHEROE();
        PELEA();
        int check=SALIDA();
        if (check == 1) {
            break;
        }
    }
}
```

La función “GRAFICOS ()” compacta las tres barras de vida, una para cada de una de las entidades existentes en mi juego. La función “TIPOATAQUEHEROE()” me permite elegir entre tres opciones de ataque distintas para atacar a los enemigos, esta función cambia el valor de la variable heroDMG, que almacena el daño del héroe.

```
void GRAFICOS() {
    cout << enemy1 << "\t| | " << enemy1HP << " ||\n";
    cout << enemy2 << "\t| | " << enemy2HP << " ||\n";
    cout << heroName << "\t| | " << heroHP << " ||\n";
}
```

Podemos visualizar en la consola las barras de vida, y el daño del héroe cambia según la elección de un ataque u otro. Ese daño es

llamado desde la función “ATAQUE ()” mencionada previamente, la cual a su vez es llamada desde otra función “PELEA ()”.

```
void TIPOATAQUEHEROE() {
    int opcion=0;
    while ((opcion < 1 || opcion>3)) {
        cout << "Introduce el tipo de ataque que quieres realizar:\n[1]-Ataque Normal\n[2]-Ataque Especial [3]-Golpe\n";
        cin >> opcion;
        if (opcion == 0 && opcion==2) {
            cout << "¡Te no te quedan puntos de poder!";
            opcion = 0;
        }
        if (opcion == 1) {
            heroDMG = 25;
        }
        else if (opcion == 2) {
            heroDMG = 40;
            critico = rand() % 20;
            int salegcrit = rand() % 20;
            if (critico == salegcrit) {
                heroDMG *= 1.5;
            }
            c--;
        }
        else if (opcion == 3) {
            heroDMG = 20;
        }
    }
}
```

Esta función se compone de otras dos, la primera de ellas es la función “SELECCIONENEMIGO ()”. Lo que hace es permitirnos elegir entre una lista de enemigos vivos, al que queremos atacar, y darnos un aviso en caso de que ese enemigo ya esté muerto.

```
int SELECCIONENEMIGO() {
    int opcion;
    cout << "¿A que enemigo quieres atacar?\n";

    if (enemy1HP > 0) {
        cout << "[1]-" << enemy1 << "\n";
    }
    if (enemy2HP > 0) {
        cout << "[2]-" << enemy2 << "\n";
    }
    cin >> opcion;
    while (opcion < 0 || opcion>2) {
        cout << "Elige una opcion disponible";
        cin >> opcion;
        if (enemy1HP < 0 && opcion == 1) {
            opcion = -2;
        }
        if (enemy2HP < 0 && opcion == 2) {
            opcion = -2;
        }
    }
    return opcion;
}
```

	M5 ENTORNOS DE DESARROLLO		
	UF2-OPTIMIZACIÓN DE PROGRAMAS		
REFACTORIZACIÓN DE CODIGO			
Apellidos: Herraiz Foz	Nombre: Angel	Fecha:	13/01/2023

La segunda función es “ESTADOSVEL ()”, se ocupa de distribuir los turnos en función de la velocidad de las distintas entidades vivas.

Tanto “ESTADOSVEL ()” como “SELECCIONENEMIGO ()” se recogen en variables. Al ser funciones con return, printeaban un numero por pantalla si no eran recogidas en variables. Mediante estas variables creo una estructura “if, else-if” para la función “PELEA ()”. Esta última ejecuta todos los ataques al enemigo seleccionado mediante un sistema de turnos basado en velocidad.

```
void PELEA(){
    int enemigo = SELECCIONENEMIGO();
    int vel = ESTADOSVEL();
    if (enemigo == 1) {
        if (vel == 0) {
            ATAQUE(enemy1HP, hero1DMG, heroName, enemy1);
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
            ATAQUE(heroHP, enemy2DMG, enemy2, heroName);
        }
        else if (vel == 1) {
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
            ATAQUE(heroHP, enemy2DMG, enemy2, heroName);
            ATAQUE(enemy1HP, hero1DMG, heroName, enemy1);
        }
        else if (vel == 2) {
            ATAQUE(enemy1HP, enemy2DMG, enemy2, heroName);
            ATAQUE(enemy1HP, hero1DMG, heroName, enemy1);
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
        }
    }
    else if (enemigo == 2) {
        if (vel == 0) {
            ATAQUE(enemy2HP, hero1DMG, heroName, enemy2);
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
            ATAQUE(heroHP, enemy2DMG, enemy2, heroName);
        }
        else if (vel == 1) {
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
            ATAQUE(heroHP, enemy2DMG, enemy2, heroName);
            ATAQUE(enemy2HP, hero1DMG, heroName, enemy2);
        }
        else if (vel == 2) {
            ATAQUE(heroHP, enemy2DMG, enemy2, heroName);
            ATAQUE(enemy2HP, hero1DMG, heroName, enemy2);
            ATAQUE(heroHP, enemy1DMG, enemy1, heroName);
        }
    }
}
```

```
int ESTADOSVEL() {
    int v1 = enemy1SPD;
    int v2 = enemy2SPD;
    int v0 = heroSPD;
    int x;
    if (v1 > v2 && v0) {
        enemy1SPD = 0;
        enemy2SPD = 2;
        heroSPD = 1;
        return 1;
    }
    else if (v2 > v1 && v0) {
        enemy1SPD = 1;
        enemy2SPD = 0;
        heroSPD = 2;
        return 2;
    }
    else if (v0 > v1 && v2) {
        enemy1SPD = 2;
        enemy2SPD = 1;
        heroSPD = 0;
        return 0;
    }
}
```

Por último, he creado la función “SALIDA ()”. Esta pausa el juego siempre antes de terminar el bucle, para que podamos leer adecuadamente todo el texto antes de cambiar de escena, puesta que una vez salimos de esa pausa, se ejecuta un “cls”, que borra toda la consola de comandos. Mediante una subfunción “ESTADOSVIDA ()” que lee las entidades vivas y muertas, elige si el juego termina o no.

```
int ESTADOSVIDA() {
    if (heroHP <= 0) {
        return 3;
    }
    else if (enemy1HP <= 0 && enemy2HP <= 0) {
        return 4;
    }
    else if (enemy1HP >= 1 && enemy2HP <= 0) {
        cout << "El cadaver de " << enemy2 << " yace sobre el terreno de combate\n";
        enemy2HP = 0;
    }
    else if (enemy1HP <= 0 && enemy2HP >= 1) {
        cout << "El cadaver de " << enemy1 << " yace sobre el terreno de combate\n";
        enemy1HP = 0;
    }
    else if (enemy1HP >= 1 && enemy2HP >= 1) {
    }
}
```

```
int SALIDA() {
    system("pause");
    system("cls");
    int ev = ESTADOSVIDA();
    if (ev == 3) {
        juego = false;
        cout << "EL JUEGO ACABO\n";
        cout << "El heroe fue derrotado.\n";
        return 1;
    }
    else if (ev == 4) {
        cout << "El heroe vencio a los enemigos\n";
        return 1;
    }
    else {
        return 0;
    }
}
```