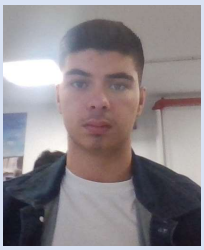
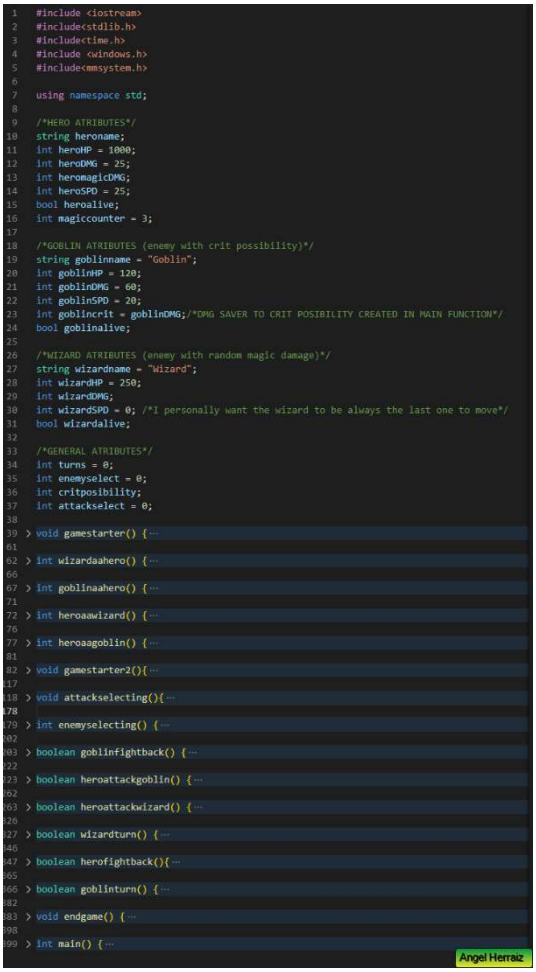
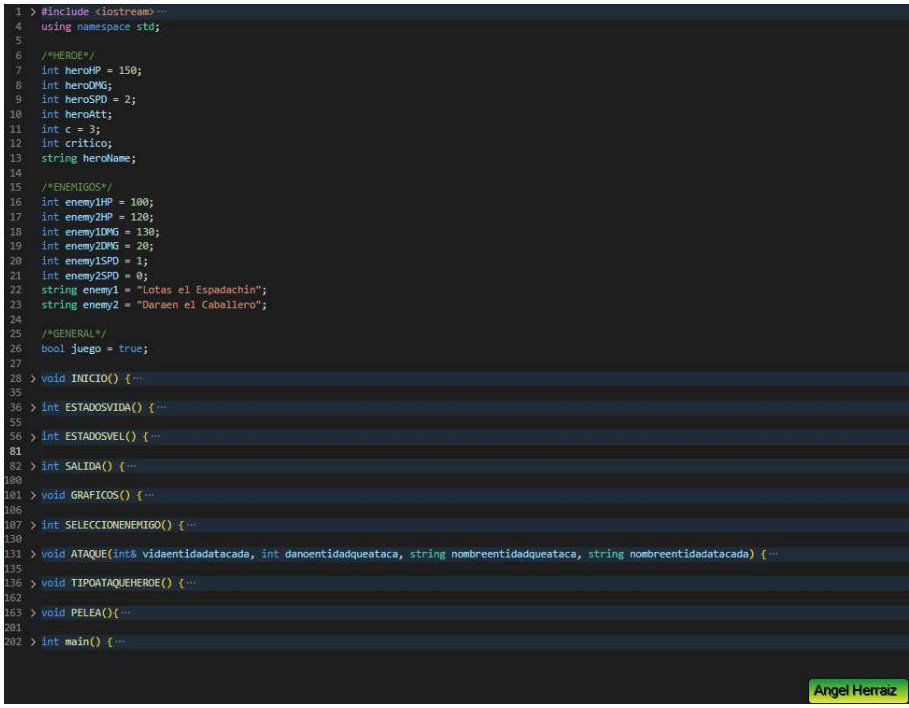
MONLAU FORMACIÓN PROFESIONAL	**M5 ENTORNOS DE DESARROLLO**	
	UF2-OPTIMIZACIÓN DE PROGRAMAS	
REFACTORIZACIÓN DE CODIGO		
Apellidos: Herraiz Foz	**Nombre: Angel**	**Fecha: 13/01/2023**

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.





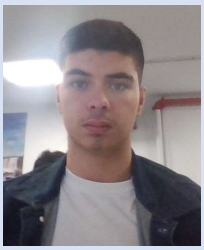
Previamente existían cuatro funciones de ataque en mi código. Dos para los enemigos y dos para el héroe. Simplifiqué estas cuatro, en una sola a la que pasarle varios valores por referencia.

==>

==>

==>

```
void ATAQUE(int& vidaentidadatacada, int danoentidadqueataca, string nombreentidadqueataca, string nombreentidadatacada) {
    cout << "¡Es " << nombreentidadqueataca << " quien blande su espada contra " << nombreentidadatacada << "! (" << danoen
    vidaentidadatacada = vidaentidadatacada - danoentidadqueataca;
}
```

MONLAU FORMACIÓN PROFESIONAL	**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)) {
        gamestarter2();
        attackselecting();
        /*ENEMY SELECTOR*/
        enemyselecting();
    /*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();
    }
}
```

Angel Herraiz

==>

==>

==>

```
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 ()”.

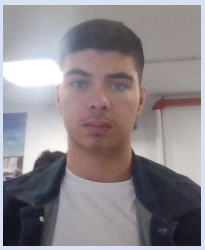
```
void TIPOATAQUEHEROE() {
    int opcion=0;
    while ((opcion < 1 || opcion>3)) {
        cout << "Introduce el tipo de ataque que quieres realizar:\n[1]-Ataque Normal\n[2]-Ataque Especial ["<<<<<" usos disponibles]\n[3]-Golpe\n";
        cin >> opcion;
        if (C == 0 && opcion==2) {
            cout << "¡Ya no te quedan puntos de poder!";
            opcion = 0;
        }
    }
    if (opcion == 1) {
        heroDMG = 25;
    }
    else if (opcion == 2) {
        heroDMG = 40;
        critico = rand() % 20;
        int salecrit = rand() % 20;
        if (critico == salecrit) {
            heroDMG *= 1'5;
        }
        C--;
    }
    else if (opcion == 3) {
        heroDMG = 20;
    }
}
```

Angel Herraiz

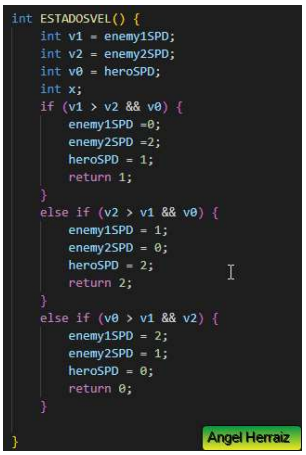
```
int SELECCIONENEMIGO() {
    int opcion;
    cout << "¿A que enemigo quieres atacar?\n";

        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;
}
```

Angel Herraiz

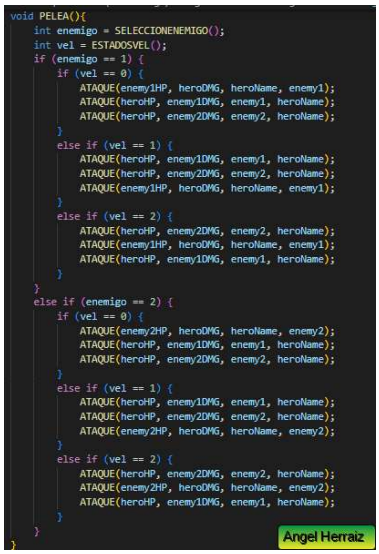
Esta función se compone de otras dos, la primera de ellas es la función “SELECCIONENEMIGO ()”. Lo que hace es permitirnos elegir entre una lista de enemigos vivos, al que queremos atacar, y darnos un aviso en caso de que ese enemigo ya esté muerto.

MONLAU FORMACIÓN PROFESIONAL	**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 funcion “PELEA ()”. Esta última ejecuta todos los ataques al enemigo seleccionado mediante un sistema de turnos basado en velocidad.



Por último, he creado la funcion “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.