FITXA D'ACTIVITAT		
CURS: 2022-23	DATA: 15/01/2023	GRUP: 1DAM
PROFESSOR/A:	Javier Salvador Carmen Quintás	
MÒDUL/UF:	M5/UF1	
TÍTOL ACTIVITAT:	Pr4: Refactorización de código	

Refactorización del código del videojuego

En esta nueva práctica, he utilizado el código proporcionado por el profesor y lo he refactorizado, creando nuevas funciones y aplicando el concepto que hemos aprendido: los parámetros por referencia.

Antes de todo, he modificado los atributos de los personajes (nombre, puntos de vida y daño de los ataques) para darle un toque personalizado.

```
8     //Atributos del primer enemigo
9     string enemyName1 = "Freezer";
10    int enemyHP1 = 1000;
11    bool enemyIsAlive1 = true;
12
13    //Atributos del segundo enemigo
14    string enemyName2 = "Bills";
15    int enemyHP2 = 1000;
16    bool enemyIsAlive2 = true;
17
18    //Atributos de nuestro héroe
19    string heroName;
20    int heroDamage;
21    int heroHP = 2000;
22    bool heroIsAlive = true;
23
24    //Atributos del juego
25    int enemyChoosed = 0;
26    int maxKamehamehaUses = 1;
27    int patadaValue = 200;
28    int espadazoValue = 500;
29    int kamehamehaValue = 1000;
```

David Bretones López

Las funciones *gameStart()*, *chooseEnemy()* y *chooseAttack()* han quedado intactas porque, según mi criterio, no se pueden simplificar más.

Partiendo de las funciones *heroAttackEnemy1()* y *heroAttackEnemy2()*, he creado una única función llamada *heroAttackEnemy()* que se pueda llamar para atacar a cualquiera de los dos enemigos. Para ello, le añadimos un parámetro de entrada con el nombre del enemigo y otro con su vida. Este último será por referencia, de manera que podamos

modificar la vida de cualquier enemigo directamente. Por tanto, no será necesario recoger la vida resultante en ninguna variable y la función pasará a ser *void*.

```
void heroAttackEnemy(string enemyName, int damage, int& enemyHP) {
    cout << "Acabas de atacar a " << enemyName << " y le has hecho " << damage << " punto(s) de damage.\n";
    enemyHP = enemyHP - damage;
}
```

David Bretones López

También he creado la función *checkEnemyStatus()* para no tener una función diferente para cada enemigo. En esta función, que sirve para comprobar si un enemigo sigue vivo o está muerto, he añadido tres parámetros de entrada: el nombre del enemigo como valor, su vida como referencia (para poder modificarla) y su estado de vivo o muerto (para poder modificarlo también). Por tanto, la función pasará a ser *void*, ya que al modificar su estado dentro de la función no será necesario devolver un *booleano* y recogerlo en una variable.

```
void checkEnemyStatus(string enemyName, int& enemyHP, bool& enemyIsAlive) {
    if (enemyHP <= 0) {
        cout << "Te has cargado a " << enemyName << ".\n";
        enemyHP = 0;
        enemyIsAlive = false;
    }
    else {
        cout << "A " << enemyName << " le queda(n) " << enemyHP << " punto(s) de vida.\n";
        enemyIsAlive = true;
    }
}
```

David Bretones López

Para terminar, la función *enemyAttack()* ahora recibirá también el nombre del enemigo en cuestión, de manera que no tengamos que llamar una función diferente según el enemigo que nos ataque. En este caso, el parámetro no se pasará por referencia porque dentro de la función no se modifica su valor, únicamente se muestra.

```
void enemyAttack(string enemyName, int damage) {
    heroHP = heroHP - damage;
    if (heroHP > 0) {
        cout << enemyName << " te ha contraatacado con " << damage << " puntos de damage y te queda(n) " << heroHP <
    }
    else {
        cout << enemyName << " te ha contraatacado con " << damage << " y te ha matado. El mundo esta condenado.\n";
        heroIsAlive = false;
    }
}
```

David Bretones López