| | | | |
|---|---|---|---|
| | **M5: Entornos de desarrollo** | |  |
| | **UF2-Optimización de software** | | |
| **Act4: Refactorización de código** | | | |
| **Apellidos: Marqués Fernández** | **Nombre: Marc** | **Fecha:** | 16/01/2023 |

## Videojuego con parámetros por valor y referencia

Dividiré la explicación de la refactorización del videojuego en dos puntos:

1. Añadido de parámetros en las funciones.
2. Optimización a partir de añadirlos (borrado de funciones y margen de mejora en líneas en respecto al código inicial).

Las funciones de mi código serían las siguientes:

https://github.com/Dakar4/Params_por_valor_y_referencia/blob/main/funciones_videojuego.png

### 1. Parámetros

Hay varias cosas nuevas que debemos tener en cuenta. La primera y más notoria respecto al código anterior es lo que se encuentra dentro de los paréntesis de las funciones (parámetros, tanto por valor como por referencia).

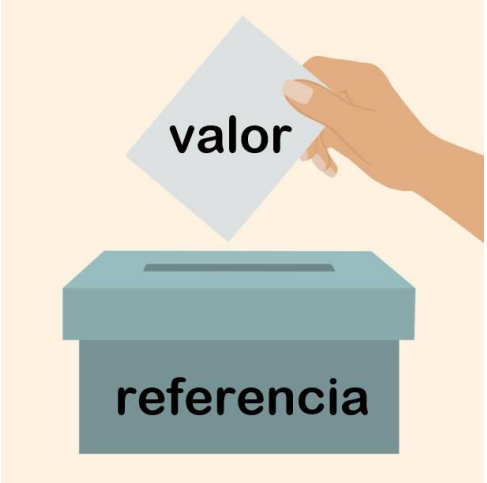
Los que están pasados por valor son simplemente inputs a los que les ponemos una etiqueta distinta para luego categorizarles la variable que nos convenga en el método principal o en el método que hayamos llamado a la función en cuestión.

**Ejemplo:** *enemyName1* (que podría tener cualquier otro nombre) en mi código hace referencia siempre a *torterraName*. De esta manera si algún día en vez de llamarse *torterraName* se llama *pikachuName* solo tendré que ir a la función en la que llamo al nombre del primer enemigo y cambiar el nombre de la variable (esto, además de reducir código, automatiza procesos).

Por otro lado, tenemos los parámetros que pasamos por referencia con el símbolo “&”.

La principal diferencia es que estos almacenan datos que no se guardan en una zona concreta de la memoria y de los cuales nos interesan sus outputs, que más tarde serán enviados a la variable que nos interese.

Una forma muy sencilla de ver esto sería con la siguiente analogía de una caja de votos:




Los resultados que se den dentro de esta caja a partir de los votos (parámetros por valor) podrán ser retirados de la misma por cualquier persona (donde cada persona seria una variable distinta, por eso estos parámetros no ocupan una posición fija en la memoria a diferencia de los que van por valor, ya que los votos entrarían en la caja, pero no saldrían porqué lo que realmente sale es el resultado de la suma de estos).

Extrapolando todo esto a mi videojuego, variables como los puntos de vida o las que detectan si las entidades están vivas o muertas las paso por referencia mientras que los ataques o los nombres por valor.

## 2. Optimización

Con esto he podido ahorrarme varias funciones booleanas que comprobaban si las entidades estaban con vida, reduciéndolas a una sola.

En principio debería poder hacer lo mismo con los ataques hacia los enemigos, pero de la manera que había estructurado el videojuego no podía debido a que según al enemigo al que atacase había ataques con mas o menos efectividad, y esto lo tenia hecho dentro de las propias funciones.

| | | | |
|---|---|---|---|
| | **M5: Entornos de desarrollo** | |  |
| | **UF2-Optimización de software** | | |
| **Act4: Refactorización de código** | | | |
| **Apellidos: Marqués Fernández** | **Nombre: Marc** | **Fecha:** | 16/01/2023 |

Quiero añadir que también he aprendido que cuando llamas a una función dentro de una función, si el nombre del parámetro que utilizas en la función interior es el mismo que utilizas en la exterior entiende la variable a la que haces referencia.

Ejemplo visual:

```
void serperiorAttacksAndKills(string enemy1Name, string enemy2Name, string heroName,
    if (decision == 1) {
        attackTorterra(enemy1Name, heroName, enemy1Hp, heroHp, enemy1Alive);
    }
    else if (decision == 2) {
        attackBlastoise(enemy2Name, heroName, enemy2Hp, heroHp, enemy2Alive);
    }
```

Marc Marqués Fernández