

### **Buenas prácticas y convenciones de programación:**

(Creo que no nos van a estar mirando mucho el código, así que esto es principalmente para facilitar el trabajo conjunto, no para presentarlo a nadie)

Disclaimer: esto depende de a quién le preguntes y va por gustos, aunque hay algunas más generales que otras y muchas dependen del lenguaje usado. Voy a poner a las que estoy acostumbrada yo por tutoriales y por motores recientemente.

#### **Nombres:**

**guion bajo** antes de nombre de variable **privada** en **minúscula**

“*\_instance*” variable privada

**métodos** primera letra **mayúscula**, independientemente de si público o privado (se da por hecho que son descriptivos de lo que hacen)

“*Instance*” método público getter de la variable privada.

si son **varias palabras**, siempre la siguiente empieza por **mayúscula**, nada de separarlo por guion bajo ni nada.

“*\_transRights*”

“*TransRights*”

las **constantes** son siempre **mayúsculas**

“PAPOPEPO = 4206669”

a mi personalmente no me podría dar más igual si es en inglés o español

Ej:

(SCRIPT GAMEMANAGER)

```
private int _coins = 0;
```

```
private GameManager instance = this;
```

```
public GameManager Instance { get { return _instance; } };
```

lo del get / return es porque es así en c#, idky

```
public static void AddCoin() { _coins++; }
```

GameManager.Instance.AddCoin() permite acceder DESDE CUALQUIER SITIO si la instancia de GameManager existe, si no, dará error de referencia.

(SCRIPT COLLISIONMANAGER) // añadir moneda

```
if(collider.blabla == coin blabla) { GameManager.Instance.AddCoin() };
```

convendría hacer un if (GameManager.Instance != null) para evitar errores pero vaya

#### **Convenio de llaves:**

Las llaves **siempre** así:

```
void papopepo()
```

```
{
```

```
    papopepo
```

```
}
```

```
nunca así
void papopepo() {
    papopepo
}
```

para métodos de una línea como los getters puede ser en la misma línea tmb aunque es menos legible la mayoría de las veces.

```
void papopepo() { papopepo };
```

Orden de código: metidos en #region #endregion cada uno excepto los de tick

enums para las state machines

referencias variables que referencian a otros componentes u objetos

properties variables internas

métodos propios que hagamos nosotros

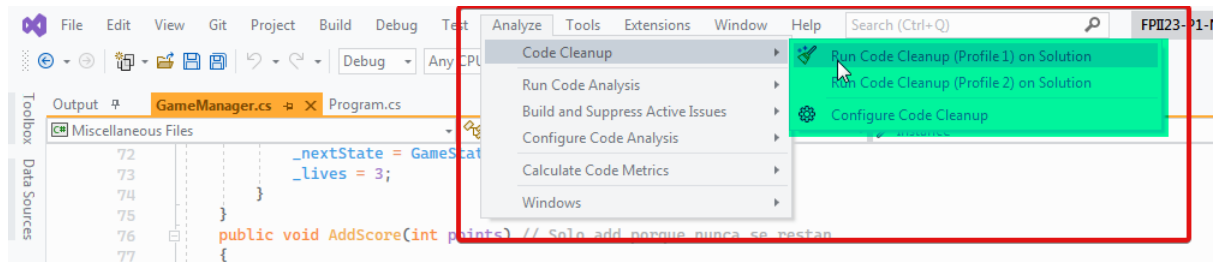
métodos internos (tick): van en este orden

```
awake
start
fixedupdate
update
lateupdate
```

Orden de carpetas del proyecto: STC pero más o menos el planteamiento general

- Assets:
  - Scripts
  - Scenes
  - Sprites
    - UI
    - Characters
    - ...
  - Prefabs
    - ...
  - Animations (Cuando animaciones)
  - Sounds (Cuando sonidos)
    - Background (?)
    - SFX (?)
  - ... (Según lo que necesitemos, si tenemos varios recursos de un mismo tipo se creará su propia carpeta (fonts por ejemplo si usamos varias), pero por lo general cuanto más simple sea mejor)

Y en general algo **muy útil es limpiar el código**: hace una limpieza más estética y legible que otra cosa, pero siempre viene bien. (ej: (i+4)%2 pasa a ser ( i + 4 ) % 2



Y **comentarlo!!!** después de cada sesión procurad coger la costumbre de comentar lo que hayáis hecho, vuestro future self os lo agradecerá (no tiene por que ser de manera exhaustiva tampoco)

[Convenio de C# según microsoft](#) (usamos camel)

plus: **organización de github:**

Haremos la arquitectura básica (gamemanager + movimiento) en la rama main y ya luego cada HU feature según assignees por **escenas propias** y **ramas**.