*Javier Murillo González y Enrique Juan Gamboa*

# PRÁCTICA 1: preguntas finales a responder

1. **En el método OnTriggerEnter2D del shot controller hemos utilizado GetComponent para comprobar si el objeto colisionado era un enemigo antes de llamar a sus métodos.**
   *¿Conoces alguna forma más sencilla de hacer esta comprobación? ¿Cuál? ¿Y por qué es preferible hacerlo como lo hemos hecho nosotros?*

   Sí, con las etiquetas que proporciona Unity para poner a cada objeto. El problema de esto es que consume muchos recursos, o sea, suele ser más lento, algo que el GetComponent soluciona en mayor o menor medida.

2. **En el caso de DeathZone no hemos hecho esa comprobación. ¿Por qué no era necesario hacerlo?**
   Porque lo único que debe de llegar a DeathZone son los enemigos. Sabiendo esto, una comprobación sería redundante.

3. **Hemos utilizado multitud de componentes para implementar comportamientos muy sencillos que podrían haberse incluido en un único componente. ¿Por qué es mejor hacerlo así?**
   En general para que el código esté más ordenado. También nos proporciona una facilidad muy grande a la hora de ajustar, añadir o reparar diferentes funciones en dicho código. Además permite que se pueda dar acceso a un mismo componente desde diferentes sitios.

4. **Hablemos de prefabs anidados:**
   *Si cambias el color de p_Enemy, ¿Qué ocurre con los prefabs p_Squad_01, p_Squad_02…? Y si cambias el color de un enemigo en p_Squad_01, ¿Qué ocurre con p_Enemy?*
   En nuestra implementación, el p_Enemy fue añadido a p_squad_XX y luego separado del mismo, cortando así la dependencia entre ellos. Si siguieran dependiendo unos de los otros, cualquier modificación en p_Enemy se vería reflejado en los prefabs de p_squad_XX.
   En cambio, si se modifiacaran los prefabs desde p_squad_XX, los cambios no se llevarían al prefab original (a menos que indiquemos que se haga).

5. **¿Qué método has utilizado para implementar el movimiento de Player, p_Shot y p_Squads? ¿Por qué?**

   Hemos usado transform.translate tanto para Player como para p_Shot, ya que ofrece un movimiento lineal muy sencillo para la bala y cambios de direcciones suaves y responsivos al input en Player.

   Debido a que p_Squads debía de tener unos movimientos más complejos y no lineales hemos optado por usar transform.position, que ofrece mayor precisión a cambio de complejidad en el método.

6. **¿Qué método has utilizado para que el resto de GameObjects pudieran acceder al GameManager? ¿Cuántas veces en la ejecución de tu juego se busca el GameManager? ¿Es una forma correcta de implementarlo?**

   Para que los objetos pudieran acceder al GameManager hemos optado por hacer uso de referencias a la clase de GameManager de la instancia actual del juego en los scripts necesarios de cada objeto.

   Durante nuestra ejecución jamás se busca al GameManager como tal, ya que interactuamos con el GameManager a base de referencias. En cuanto al número de referencias que se hace al GameManager, va a ser el mismo que el número de enemigos que aparezcan en esa instancia (ya que o son destruidos por una bala, aumentando el score, o son destruidos por el DeathZone, acabando la partida) más uno si el jugador entra en contacto con un enemigo.

   Es una correcta forma de implementarlo debido a que no se pierde tiempo buscando al GameManager, ya que “sabemos dónde está” desde el principio del juego y esto nos permite hablar con él a base de referencias, lo cual es mucho más veloz.