Unity

Introducción a Motores de Videojuegos

Práctica final

Jesús Mayor: 2017-2018

Creando un videojuego para Oculus usando Unity

- Nos vamos a basar en el Hogan's Alley de la NES

Definición de requisitos

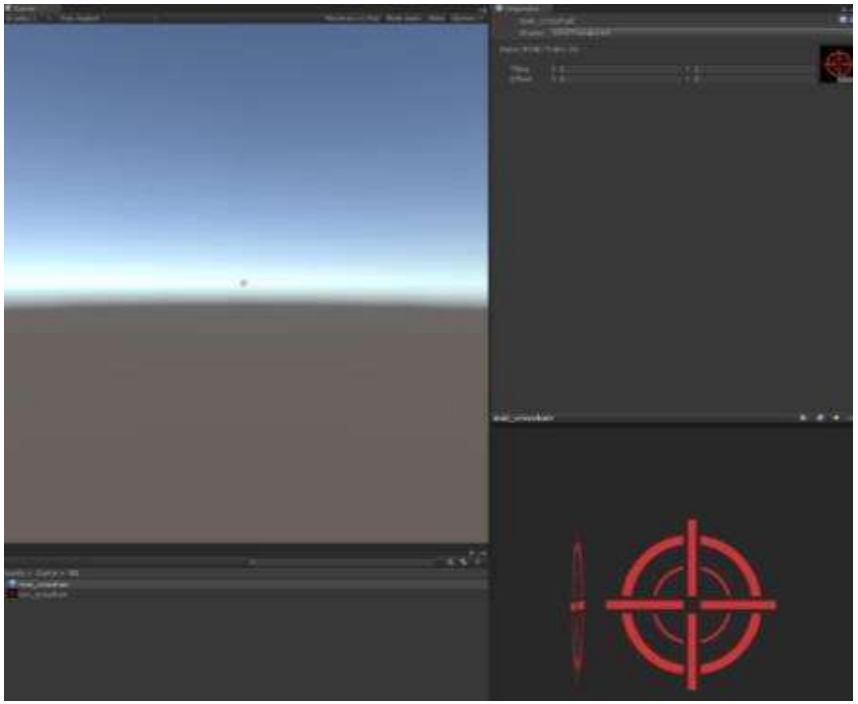
- La partida dura **1 minuto**.
- El jugador permanece inmóvil en el centro del escenario y usa **Oculus para mirar y apuntar**.
- Irán **aparecien**do diferentes personajes aleatoriamente:
  - **3 personajes buenos y 3 malos**.
  - Los malos disparan después de un tiempo.
  - Los buenos no hacen nada.
- Objetivo: **Disparar** a los malos antes de que nos disparen y evitar disparar a los buenos.
- Crear un sistema de puntuación:
  - +100 puntos por acertar a un personaje malo. -100 puntos si no acertamos a tiempo.
  - -200 si disparamos a un personaje bueno.

Pasos a seguir

- Creamos la **estructura de carpetas** enseñada.
- **Importamos** el paquete creado especialmente para **esta práctica**. (Descargar de Blackboard)
- Descargamos el paquete **Prototyping de Asset Store** de Unity. Lo usaremos para generar el escenario.

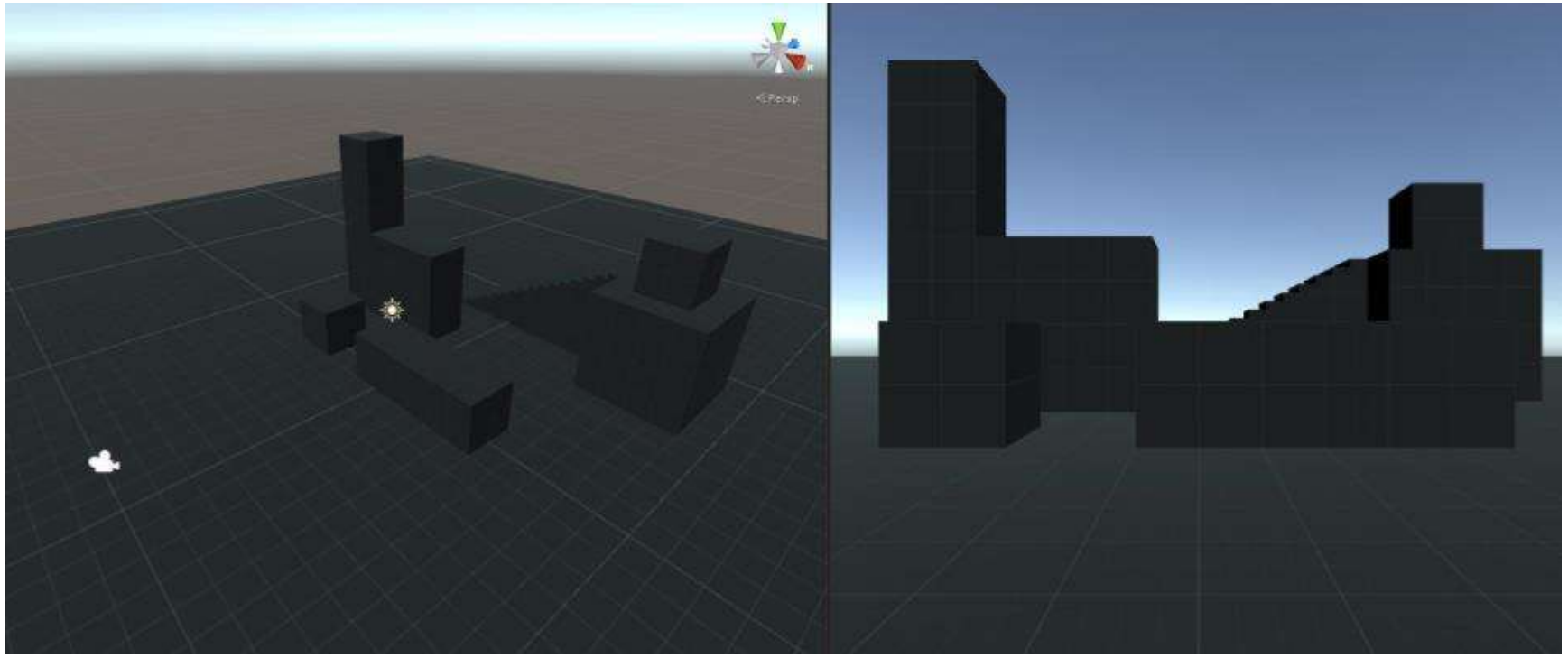
Pasos a seguir

- Creación de una mirilla
  - Quad a una distancia frontal e hija de la cámara. Eliminar su collider.
  - Material con shader unlit/transparent para mostrar transparencia y que no le afecte la luz.



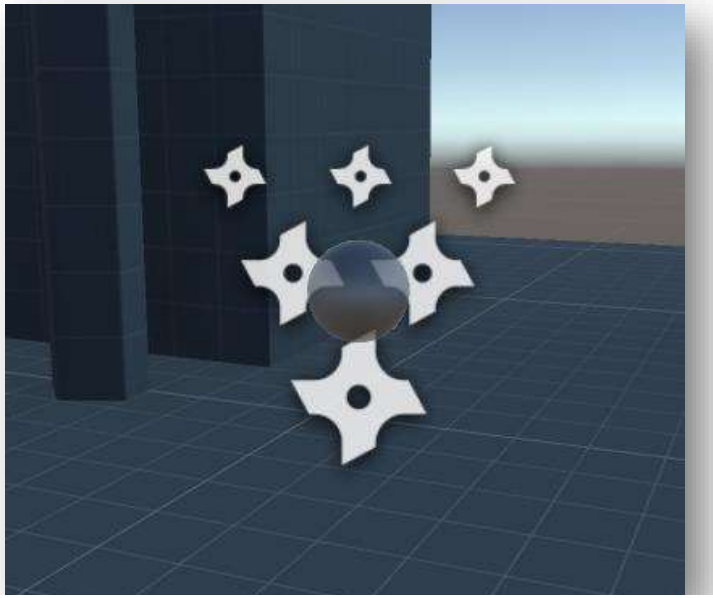
Pasos a seguir

- **Creamos el escenario** usando el paquete Prototyping de Unity.
- Usando los prefabs incluidos u la tecla **v + mover** podemos montar un campo de tiro rápidamente.



Pasos a seguir: Disparo

- Creamos un prefab con la siguiente funcionalidad:
  - Una bala de cañón que sonará al dispararse **(sonido "Fire")** y al chocar **(sonido "Hit").**
  - Crear una clase **BulletHandler** que implemente dicha funcionalidad.
  - De manera opcional, crear una método "**public void Shoot (Vector3 direction)**" que lance el sonido del disparo a la vez que impulse la bala.
  - Acordarse de implementar el **OnCollisionEnter** respectivo.



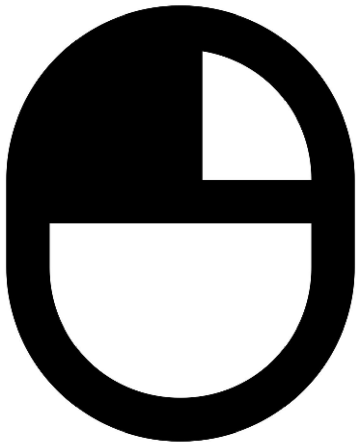
Pasos a seguir: Disparo

```
public class BulletHandler: MonoBehaviour {

    private ParticleSystem hitPs;
    private Rigidbody rb;
    private AudioSource bulletAs;
    public AudioClip shootSound;
    public AudioClip hitSound;
    public float impulseMagnitude = 10f;

    // Use this for initialization
    1 reference
    void Start () {
        hitPs = GetComponent<ParticleSystem>();
        if (hitPs == null)
            Debug.Log("No se encuentra el ParticleSystem del bullet");
        rb = GetComponent<Rigidbody>();
        if (hitPs == null)
            Debug.Log("No se encuentra el Rigidbody del bullet");
        bulletAs = GetComponent<AudioSource>();
        if (hitPs == null)
            Debug.Log("No se encuentra el AudioSource del bullet");
    }

    1 reference
    public void Shoot (Vector3 direction)
    {
        Start();
        bulletAs.PlayOneShot(shootSound);
        rb.AddForce(direction.normalized * impulseMagnitude, ForceMode.Impulse);
    }

    // Update is called once per frame
    0 references
    void OnCollisionEnter (Collision other) {
        hitPs.Play();
        bulletAs.PlayOneShot(hitSound);
    }
}
```

Pasos a seguir: Cámara

- Crear un script que permita:
  - **Instanciar un prefab de la bala** que hemos creado y llamar a su función Shoot, pasando como parámetro el vector de al que apunta la cámara.
  - **Quien crea destruye** (buena práctica de programación en general) destruir dicha instancia a las **3 segundos** de haber sido disparada.
  - Si no está activada la RV: Movimiento de la cámara con el **ratón**.
  - Si sí está activada: Movimiento natural de las **gafas**.

  - **Se dispara con el botón izquierdo del ratón.**

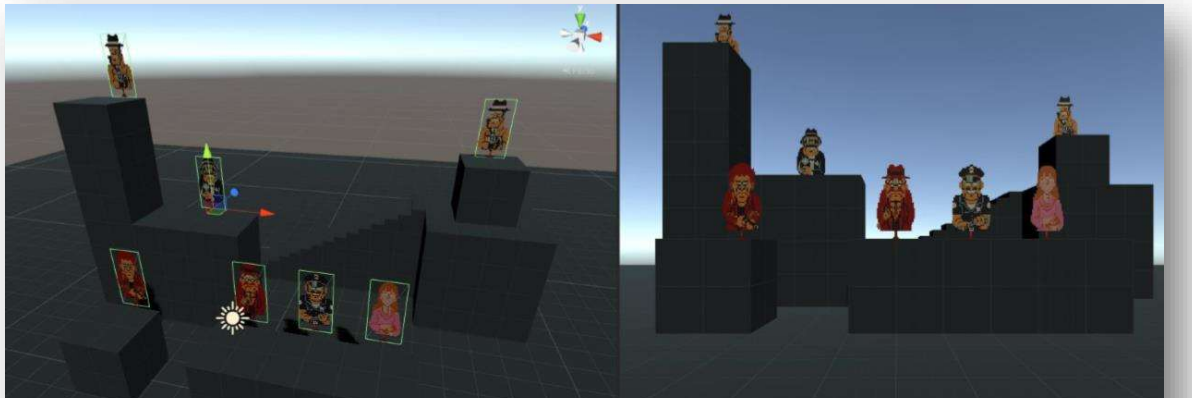


Pasos a seguir: Cámara

```
public class CameraController : MonoBehaviour{

    public float camSens = 0.50f; //How sensitive it with mouse
    Vector3 lastMouse = new Vector3(255, 255, 255);
    [SerializeField]
    GameObject bullet;

    0 references
    void Update()
    {
        //Mouse  camera angle done.
        if (!VRSettings.enabled)
            CameraPosition();

        if (Input.GetButtonDown("Fire1"))
            ShootABullet();
    }

    1 reference
    void CameraPosition()
    {
        lastMouse = Input.mousePosition - lastMouse;
        lastMouse = new Vector3(-lastMouse.y * camSens, lastMouse.x * camSens, 0);
        lastMouse = new Vector3(transform.eulerAngles.x + lastMouse.x, transform.eulerAngles.y + lastMouse.y, 0);
        transform.eulerAngles = lastMouse;
        lastMouse = Input.mousePosition;
    }

    1 reference
    void ShootABullet()
    {
        GameObject bulletInstance = Instantiate(bullet, transform.position, transform.rotation);
        BulletHandler bh = bulletInstance.GetComponent<BulletHandler>();
        if (bh != null)
            bh.Shoot(transform.forward);
        else
            Debug.Log("La Bullet no tiene Script BulletHandler");
        Destroy(bulletInstance, 3f);
    }
}
```

Pasos a seguir: Spawn points

- Se marcarán unos **puntos específicos (Empty GameObjects)** en el mapa donde se podrán spawnear tanto enemigos como aliados.
- Hay que crear una clase que se encargue:
  - Tener un **array de puntos posibles**.
  - Tener un **array de prefabs spawneables**.
  - Instanciar en un **tiempo aleatorio** (entre un máximo y mínimo) dichos prefabs en una posición también aleatoria.
  - Opcional, no necesario: Añadir un método “Stop()” que pare dicho proceso de “Spawneos”.



Pasos a seguir: Spawn points

```
public class PeopleSpawner : MonoBehaviour {

    [SerializeField]
    private GameObject[] SpawnPoints;
    [SerializeField]
    private GameObject[] PersonTypes;

    public float minSpawnTime = 1f;
    public float maxSpawnTime = 10f;

    private bool stop = false;

    // Use this for initialization
    0 references
    void Start () {
        StartCoroutine(SpawnPeople());
    }

    1 reference
    IEnumerator SpawnPeople()
    {
        while (stop == false)
        {
            yield return new WaitForSeconds(Random.Range(minSpawnTime, maxSpawnTime));
            Instantiate(PersonTypes[Random.Range(0, PersonTypes.Length)], SpawnPoints[Random.Range(0, SpawnPoints.Length)].transform);
        }
    }

    0 references
    public void Stop()
    {
        stop = true;
    }
}
```

Pasos a seguir: Los objetivos

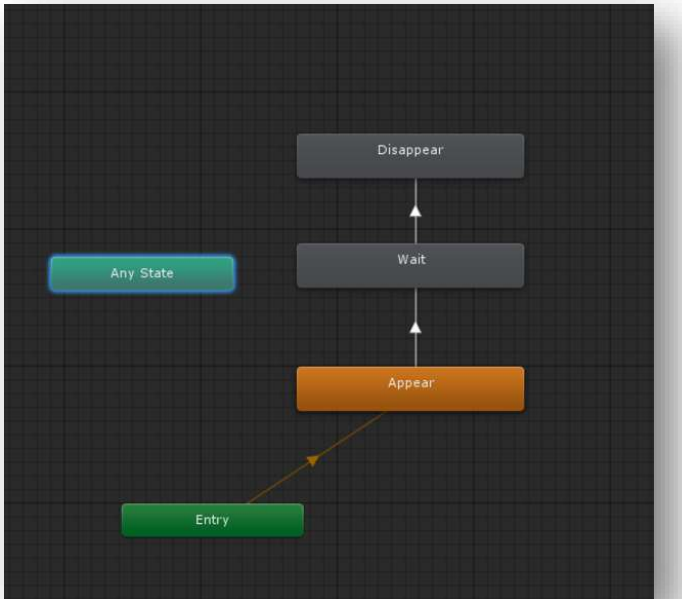
- Creamos los personajes: 3 buenos y 3 malos, aunque inicialmente crearemos solo uno con todo para replicar el prefab cambiando unos pocos datos.
  - **Quad con escala (1, 2, 1)**
  - Material Standard de tipo Cutout.
  - Añadimos un **box collider** para las colixiones. (Eliminando el del Quad)
  - Opcional: Se pueden emparentar a **un Root** para hacer que las animaciones partan de esa posición relativa. Hay otras soluciones.

Pasos a seguir: Los objetivos

- Personajes buenos
  - Aparecen -> Espera -> Desaparecen.
- Personajes malos
  - Aparecen -> Espera -> Disparo y desaparecen.
- Esta máquina de estados puede ser la del **Animator** contralada a través de Scripts.

Pasos a seguir: Animando personajes

- 3 clips de animación, uno para cada estado.
- Usando la ventana **Animation crearemos las 3 animaciones**. Animando el position del hijo evitaremos problemas posicionales de la animación.
- Usando el Animator, **crearemos una máquina de estados**. Añadir el **trigger "EndNow" como condición para transicionar al último estado**.



Pasos a seguir: Controlador de objetivos

- Crear una clase GuyActions:
  - El tiempo de espera en la posición **Stay deberá ser variable entre dos valores dados a través del inspector**.
  - Tras pasar ese valor de espera, se activará el trigger de animación con el comando**: animator.SetTrigger("EndNow").**
  - Detectar la **colisión de la bala**. En este caso otorgar unos puntos en función de si el personaje es bueno o malo.
  - Si llega al **tiempo de espera máximo** sin detectar colisión, **se quitarán puntos** en el caso de ser un personaje malo.

  - La acción de **quitar puntos** se aplicará en otra **clase GameplayManager** que haremos al final.

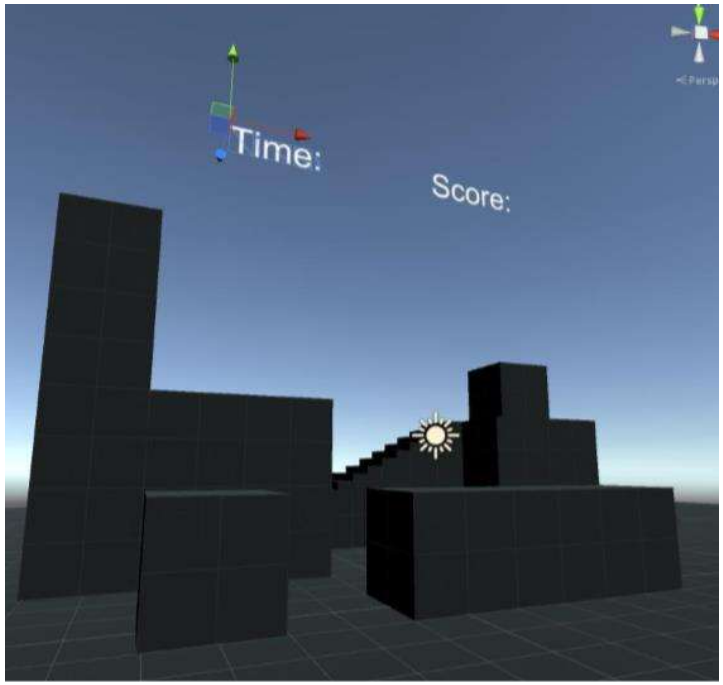
Pasos a seguir: Controlador de objetivos

```
public class GuyDie : MonoBehaviour {

    Animator anim;
    public float minStayTime = 3f;
    public float maxStayTime = 5f;
    private float timeZero = 0f;
    private float waitTime = 0f;
    public float pointsHit = 0f;
    public float pointsMiss = 0f;

    public delegate void ShootEventHandler();
    public event ShootEventHandler OnShoot;

    // Use this for initialization
    0 references
    void Start () {
        anim = GetComponent<Animator>();
        if (anim == null)
            Debug.Log("No encuentra animador");
        timeZero = Time.realtimeSinceStartup;
        waitTime = Random.Range(minStayTime, minStayTime);
    }

    // Update is called once per frame
    0 references
    void Update () {
        if (Time.realtimeSinceStartup > (timeZero + waitTime))
        {
            anim.SetTrigger("EndNow");
            GameplayManager.GetInstance().points += this.pointsMiss;
            this.enabled = false;
        }
    }

    0 references
    void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.name == "Bullet(Clone)")
        {
            anim.SetTrigger("EndNow");
            GameplayManager.GetInstance().points += this.pointsHit;
            this.enabled = false;
        }
    }
}
```

Pasos a seguir: Gameplay Manager

- Crear una clase que utilice el patrón de diseño **Singleton**
- Al ser un **Monobehavior y Singleton**, se puede almacenar los **datos** de partida de una manera sencilla y **acceder a ellos estáticamente**.
- Esta clase **almacenará puntos** y controlará que la partida **dure un minuto**.
- Tanto los puntos como el tiempo **se mostará en un TextMesh modificada cada frame**.
- Los datos se modificarán por los personajes al morir o terminar su espera.
- Para cerrar la aplicación: **Application.Quit().**

[SOLO FUNCIONA CON UNA VERSIÓN COMPILADA]



Pasos a seguir: Gameplay Manager

```
public class GameplayManager : MonoBehaviour {

    private static GameplayManager instance;

    private float timeZero;
    public float playingTime = 60f;
    public TextMesh textMesh;

    public float points = 0f;


    // Use this for initialization
    0 references
    void Start () {
        timeZero = Time.realtimeSinceStartup;
        instance = this;
    }

    2 references
    public static GameplayManager GetInstance()
    {
        return instance;
    }

    // Update is called once per frame
    0 references
    void Update () {
        if (Time.realtimeSinceStartup > (playingTime + timeZero))
            Application.Quit();
        textMesh.text = "Time: " + (int)(Time.realtimeSinceStartup - timeZero) + "\nScore: " + points;
    }
}
```

Extras: Paquete Diseñador de niveles Experto. (3pt)

- **Añadir un tiempo de disparo en los enemigos. (1pt)**
  - Ahora los enemigos disparan en un tiempo aleatorio antes de desaparecer. Al disparar, los personajes malos mosntrarán un cartel de “BANG!”. Usar el proporcionado en el material.
- **Añadir diferentes zonas, cambiarse de zona. (2pt)**
  - Habrá dianas en el escenario a las que puedes disparar para teletransportarte instantáneamente a ellas.
  - Al aparecer en una zona diferente, los spawn points serán otros diferentes, más acordes a la nueva zona.
  - Como mínimo 3 zonas diferentes, con ambientaciones diferentes.
  - El acabado debe ser mejor, no solo de cajas de prototipado.

Extras: Paquete Programador Experto. (3pt)

- **Los Spawnpoints se autosuscriben. -Patron de diseño observer)- (1pt)**
  - La idea es crear un **prefab de los puntos de spawn** de tal manera que sean solo el **Transform más tu nuevo script**.
  - Este script hará que **se registren todos los puntos** en la fase de **Awake()**.
  - De esta manera cuando el "spawneador" quiera iniciar (Start()), comenzará con la lista de spawnpoints autoregistrados.
  - Sería interesante que el **Spawneador** se convirtiese en **otro Singleton** para permitir el fácil registro desde los spawnpoints.

Extras: Paquete Programador Experto. (3pt)

- **Inversión del control usando delegates. (1pto)**
  - La idea es **fusionar** la clase que **spawnea** e incluir sus funciones dentro del **GameplayManager**.
  - De esta manera, el **nuevo spawneador decidirá** si **instanciar** una persona **buena o mala**.
  - Una vez creada, si la persona creada fue mala le asignará una **función en el delegate que se encarga de quitar puntos al dispararla**.
  - Ya que esta función esta dada desde el GameplayManager, resulta fácil acceder a los puntos con esta nueva función asignada.
  - **Pista: La imagen de la derecha.**

```
public class Person : MonoBehaviour
{
  public enum PersonType
  {
    Good,
    Bad
  }

  public delegate void ShootEventHandler();
  public event ShootEventHandler OnShoot;

  public PersonType Type
  {
    get
    {
      return _personType;
    }
  }

  private void Awake()
  {
    Invoke("Shoot", _shootTime);
    Invoke("Eliminate", _lifeTime);
  }

  private void Shoot()
  {
    if(OnShoot != null && _personType == PersonType.Bad)
    {
      OnShoot();
    }
  }

  private void Eliminate()
  {
    Destroy(this.gameObject);
  }

  [SerializeField]
  private PersonType _personType;
  [SerializeField]
  private float _shootTime;
  [SerializeField]
  private float _lifeTime;
}
```

Extras: Paquete Programador Experto. (3pt)

- **Menú inicial, contexto general. (1pt)**
  - La idea es generar en **otra escena un menú** donde se **muestre la mejor puntuación y un botón de comenzar partida.**
  - Esta puntuación se guardará en un "**GameManager**" diferente al GameplayManager, que tendrá un contexto general entre escenas.
  - Para que no se destruya entre escenas este manager, habrá que usar la función: **DontDestroyOnLoad()**.
  - De esta manera, **al terminar una partida volveremos a este menú** modificando la "mejor puntuación". **Para salir del juego se pulsará otra opción en el menú**.
  - La escena de menú tendrá un fondo rotatorio propiamente decorado.
  - **Ojo con los Singletons**, hay que controlar bien las referencias estáticas al cargarse varias veces las mismas escenas para que no de error.

Extras: Paquete Experto en VFX. (3pt)

- **Shader con efecto futurista, "panel de entrenamiento". (3pt)**
  - Los **personajes** se mostrarán con un **nuevo shader** programado por vosotros.
  - Este shader se hace con leves **desplazamientos de uv en función de los canales rgb**.
  - Cada canal se desplazará en una dirección diferente.
  - Una **función seno** generará **partes más oscuras o claras** en función de una de las coordenadas uv.
  - Estas líneas se desplazará muy despacio en el tiempo.
  - El resultado debe ser similar a este:

Extras: Paquete Experto en VFX. (3pt)

- Para este shader es importante no modificar la textura base, toda edición se hace a través de código.
- Un ejemplo en otro lenguaje de shaders que puede servir de referencia es: https://www.shadertoy.com/view/ldXGW4 o https://www.shadertoy.com/view/XsjSzR (parte central)

- **Pistola laser. (1pt)**
  - Cambiar la mecánica de las balas de cañón por disparos laser.
  - Para ello realizar un Raycast para detectar si hay un enemigo en la dirección de la cámra. https://docs.unity3d.com/ScriptReference/Physics.Raycast.html
  - El láser se pintará con un LineRenderer, dando como posición inicial la cámara y posición final la posición donde haya chocado el “Raycast”.