Introducción a la programación de videojuegos

Sebastián Díaz Arancibia

Universidad Técnica Federico Santa María - 2023

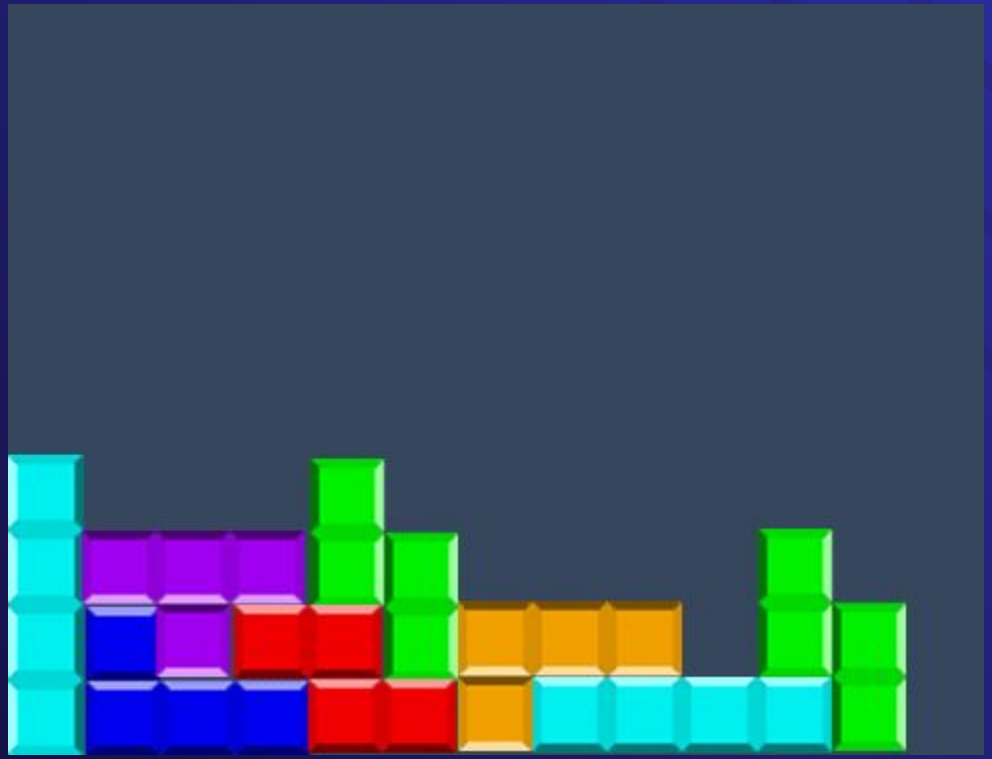
Tabla de contenido

01.

Crear GameObjects

Existen muchas mecánicas acerca de la creación de objetos en la escena. Desde proyectiles, enemigos, plataformas, etc.

BRAND NAME



Crear GameObjects

Existen muchos videojuegos que crean objetos en un videojuego y con distintos propósitos, donde esta mecánica juega un papel importante para el progreso del jugador y la personalización de la experiencia de juego:

- **Minecraft** es un juego de construcción y aventura en el que los jugadores pueden recolectar recursos de su entorno y utilizarlos para crear una amplia variedad de objetos y estructuras. Los jugadores combinan diferentes materiales y elementos para crear herramientas, armas, bloques, etc.

- **The Binding of Isaac** es un juego de mazmorras roguelike en el se exploran mazmorras generadas aleatoriamente. Durante el juego, se pueden recolectar diferentes objetos y power-ups que afectan las habilidades y atributos de su personaje, lo que les permite enfrentarse a enemigos más difíciles y desbloquear nuevas áreas.

Crear GameObjects

```
public GameObject coin;

// Start is called before the first frame update
void Start()
{
    Instantiate(coin);
}
```

Instantiate(GameObject)

En Unity, un GameObject se puede crear usando la función Instantiate que hace una nueva copia de un objeto existente. El objeto será creado según la posición y rotación que tenga este GameObject.

En estos casos se recomienda crear **prefabs** debido a que si utiliza un GameObject de la escena podría tener futuros problemas en la implementación.

Para crear un prefab, se debe arrastrar el objeto desde la jerarquía hasta una carpeta en el explorador de archivos de Unity.

Crear GameObjects

Un prefab es un término utilizado para referirse a un objeto o entidad preconfigurada que se puede reutilizar en múltiples instancias en tu juego. Un prefab es una plantilla o modelo que contiene una configuración completa de componentes, propiedades y comportamientos de un objeto en particular.

Un prefab en Unity es un archivo que representa un objeto completo o una entidad dentro de tu juego. Contiene toda la información necesaria para crear instancias del objeto, incluyendo sus componentes, transformación, materiales, colisiones, scripts adjuntos y cualquier otro elemento asociado.

Puedes crear un prefab seleccionando un objeto en la escena y arrastrándolo al Project Window. Una vez creado, puedes editar el prefab para ajustar sus propiedades y componentes. Los cambios que hagas en el prefab se aplicarán a todas las instancias de ese prefab en tu juego.

Crear GameObjects

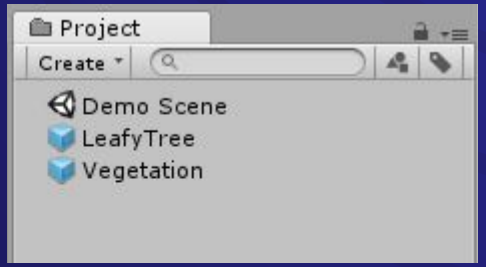
Una de las ventajas principales de los prefabs es su capacidad de reutilización. Puedes instanciar múltiples copias de un prefab en diferentes partes de tu juego.

Si necesitas realizar cambios en el prefab, los cambios se reflejarán automáticamente en todas las instancias del mismo. Esto te permite mantener una consistencia, modularidad y eficiencia en tu juego, facilita la actualización de objetos en tiempo real, ahorra tiempo y evita tener que realizar cambios individuales en cada objeto.

Los prefabs también pueden contener otros prefabs, lo que significa que puedes crear jerarquías complejas de objetos en un solo prefab.

Cuando un GameObject se convierte en prefab, su icono en la jerarquía se vuelve celeste.

En resumen, un prefab en Unity 3D es un modelo o plantilla que contiene información completa sobre un objeto o entidad en tu juego. Te permite crear instancias reutilizables de ese objeto, facilita la actualización y el mantenimiento, promueve la eficiencia y modularidad en el desarrollo y ofrece una forma flexible de estructurar y organizar tus objetos en el juego.



Crear GameObjects

Instantiate(GameObject, Transform)

En este caso, los GameObjects que se crean quedan anidados a un GameObject padre. Como consecuencia de esto, la rotación y posición de los GameObjects clonados son en relación a su padre.

Es decir, si un padre se mueve en la escena, sus objetos hijos se moverán junto con él.

```
public GameObject gO;
public Transform parent;

Unity Message | 0 references
void Start()
{
    Instantiate(gO, parent);
}
```

Crear GameObjects

Instantiate(GameObject, Vector3, Quaternion)

Usando la función Instantiate con estos 3 parámetros, los GameObjects que se crean quedan en la posición del Vector3 y su rotación va determinada por el Quaternion indicado.

Quaternion.identity significa que la rotación del objeto instanciado, tendrá relación con la rotación con la que viene el objeto.

```
Unity Message | 0 references
void Start()
{
    Instantiate(gO, new Vector3(0, 0, 0), Quaternion.identity);
}
```

Crear GameObjects

GameObject variable = Instantiate(GameObject)

Al crear un nuevo objeto, podemos guardarlo en una variable para poder modificar algunos otros componentes del objeto recién creado.

Por ejemplo, si quisiéramos crear un objeto y modificar el valor de su velocidad, o los puntos que entregará al colisionar con el jugador, esta forma de instanciar objetos lo permite.

```
public GameObject g0;

Unity Message | 0 references
void Start()
{
    GameObject aux = Instantiate(g0);
    aux.transform.localScale = new Vector3(2, 2, 1);
}
```

Crear GameObjects

Instantiate( )

Por ejemplo, si se desea clonar un objeto y especificar su posición y rotación, el código se vería algo como esto:

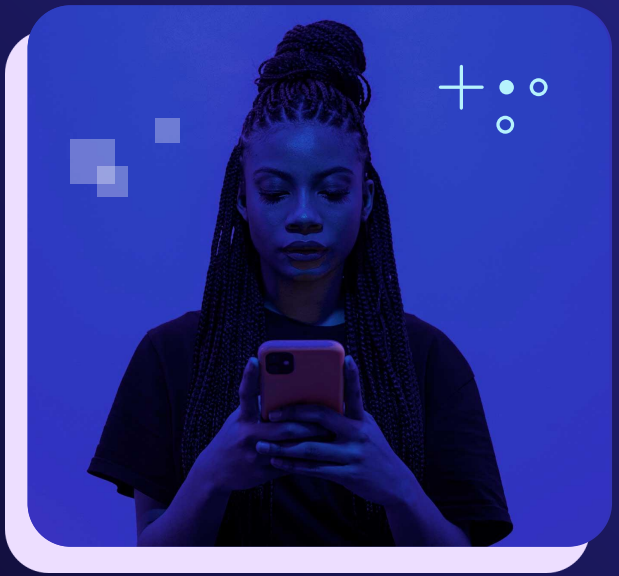
```
public class ClonarObjeto : MonoBehaviour
{
    public GameObject originalObject; // Referencia al objeto que deseas clonar
    public Vector3 newPosition; // Nueva posición para la instancia clonada
    public Quaternion newRotation; // Nueva rotación para la instancia clonada

    Unity Message | 0 references
    void Start()
    {
        // La instancia clonada se ubicará en la posición y rotación especificadas
        GameObject clonedObject = Instantiate(originalObject, newPosition, newRotation);
    }

}
```

Crear GameObjects

Instantiate( )

Por otro lado, si se desea clonar un objeto y realizar acciones sobre la instancia recién creada, el código tomaría una forma como la siguiente:

```
public class ClonarObjetoAcciones : MonoBehaviour
{
    public GameObject originalObject; // Referencia al objeto que deseas clonar

    Unity Message | 0 references
    void Start()
    {
        // Realizar acciones sobre la instancia clonada
        GameObject clonedObject = Instantiate(originalObject);
        clonedObject.transform.Translate(Vector3.forward * 2f);
        clonedObject.GetComponent<Renderer>().material.color = Color.red;
    }

}
```



Ejercicio 1

Crear un nuevo Script llamado Spawner que cada 3 segundos crear un nuevo GameObject coin. Crear 4 GameObject nuevos y vacíos. Arrastrar el Script a cada uno de estos objetos. Dar play para ver el comportamiento.

Ejercicio 1

```
public GameObject coin = null;
public float timer = 0;
public int timeToSpawn = 3;

// Update is called once per frame
void Update()
{
    timer += Time.deltaTime;
    if (timer >= timeToSpawn)
    {
        Instantiate(coin, gameObject.transform);
        timer = 0;
    }
}
```

02.
Destruir GameObjects

Todo videojuego toma la destrucción de los distintos elementos del juego como una mecánica fundamental de juego. Algunos deciden tener un contador de enemigos eliminados, otros lo utilizan para coleccionar objetos en un mapa, etc.

Destruir GameObjects

Destroy(GameObject)

Esta función destruye el objeto que se entrega como parámetro inmediatamente después del ciclo Update.

Al destruir un objeto, se pierden todas las referencias a este, los componentes dejan de funcionar, incluyendo las funciones de los scripts.

```
public GameObject gO;

Unity Message | 0 references
void Start()
{
    Destroy(gO);
}
```

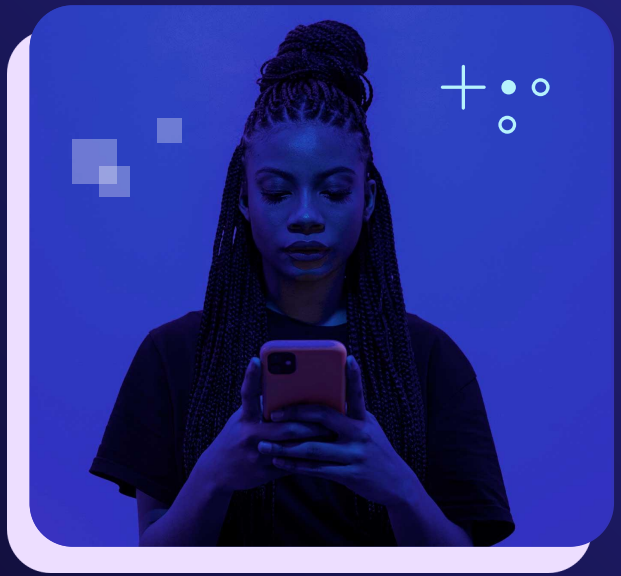
Destruir GameObjects

```
public GameObject gO;
public float timeToDestroy = 2f;

Unity Message | 0 references
void Start()
{
    Destroy(gO, timeToDestroy);
}
```

Destroy(GameObject, float)

Si se desea destruir un GameObject luego de una cierta cantidad de tiempo, se puede agregar un parámetro float para definir esta acción.

Esta función destruye el objeto que se entrega como parámetro en la cantidad de segundos a partir de ejecutada la instrucción dependiente del valor flotante que también se entrega como parámetro.

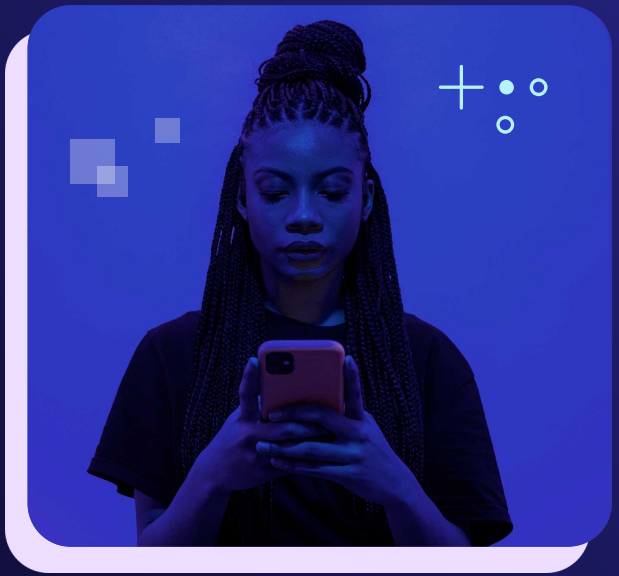


Ejercicio 2

Crear un objeto que se pueda mover en eje X e Y. Modificar el método OnCollisionEnter2D de la nave para que cuando colisione con un objeto, el objeto colisionado se destruya (no la nave). Dar play para ver el comportamiento.

Ejercicio 2

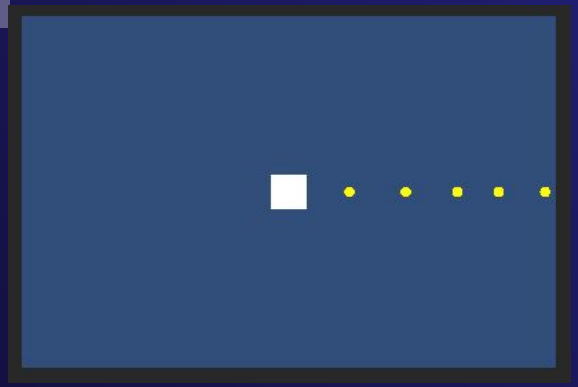
```
public void OnCollisionEnter2D(Collision2D collision)
{
    Destroy(collision.gameObject);
}
```



Ejercicio 3

Crear un script para disparar un proyectil desde un GameObject usando el clic izquierdo del mouse. Crear otro script para los proyectiles, los cuales deben moverse hacia la derecha usando su transform y ser destruidos al colisionar con cualquier otro objeto o luego de 3 segundos.

Ejercicio 3



```
public class Projectile : MonoBehaviour
{
    public float speed = 10f;

    Unity Message | 0 references
    void Update()
    {
        transform.position += transform.right * speed * Time.deltaTime;
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        Destroy(gameObject);
    }
}
```

```
public class Shoot : MonoBehaviour
{
    public GameObject projectile;
    public Transform aim;

    Unity Message | 0 references
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Instantiate(projectile, aim.position, Quaternion.identity);
        }
    }
}
```

03.

Números aleatorios

Una de las funciones más utilizadas en mecánicas de videojuegos es la generación de números aleatorios. Desde el manejo de spawners hasta una IA básica de un enemigo, las utilidades de los números aleatorios son casi infinitas.

Números aleatorios

Random.Range(int, int)

Es muy común el uso de valores aleatorios en los videojuegos, power ups en las cajas de Mario Kart, premios secretos en los cofres de Clash Royale, tiempo aleatorio para spawnear objetos en el Snake, etc. Unity provee el método Random.Range()para generar números aleatorios, recibiendo como parámetros el mínimo y el máximo del rango dentro del que se quiere el número.

Utilizando números enteros como parámetros de entrada, uno para el mínimo y otro para el máximo, ambos valores se incluyen en el rango de posibilidades. El resultado en este caso, será int por usar parámetros enteros.

```
public int numEntero = 0;

Unity Message | 0 references
void Start()
{
    numEntero = Random.Range(0, 100);
}
```

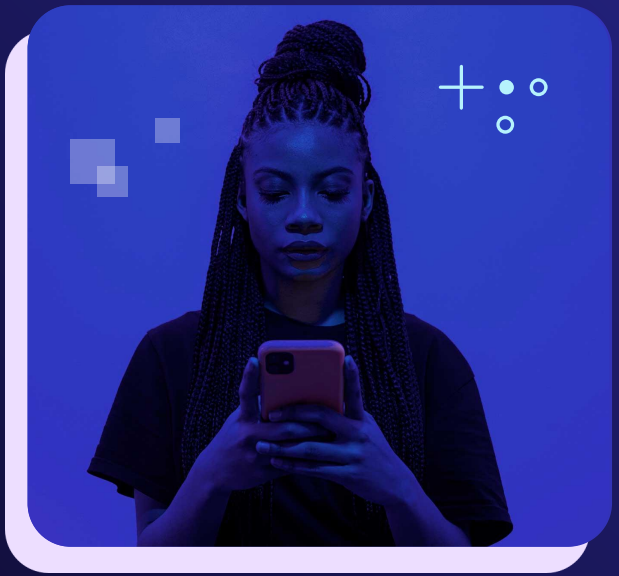
Números aleatorios

```
public float numFlotante = 0f;

Unity Message | 0 references
void Start()
{
    numFlotante = Random.Range(0f, 100f);
}
```

Random.Range(float, float)

La otra forma de utilizar Random.Range es utilizando números flotantes como parámetros de entrada. En este caso el resultado será un valor float.

Esta opción tiene un abanico de posibilidades más grande ya que por ejemplo entre 1.0f y 2.0f hay mínimo 10 opciones (1.0f, 1.1f, 1.2f...).

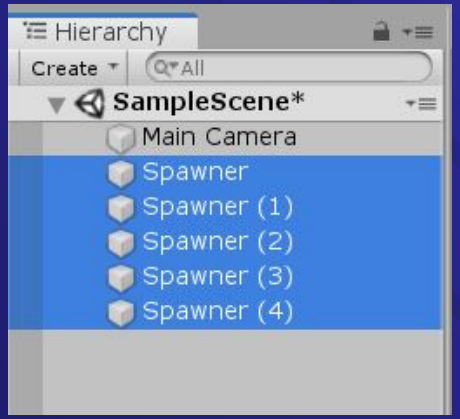


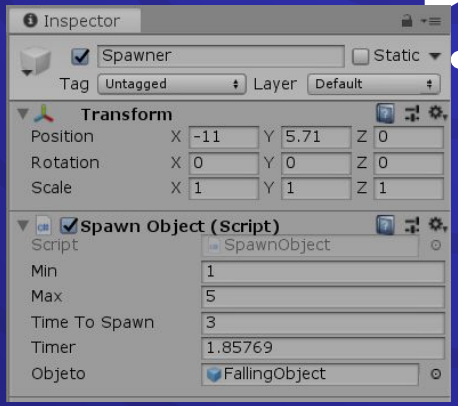
Ejercicio 4

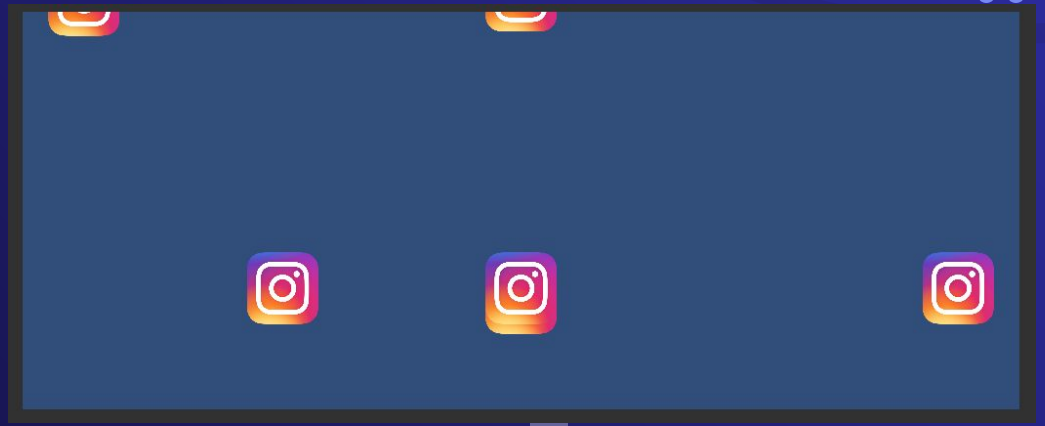
Desarrolle un nuevo script que cada una cantidad aleatoria entre 1 y 5 segundos, cree un nuevo objeto con rigidbody 2d. Dar play para visualizar los resultados.

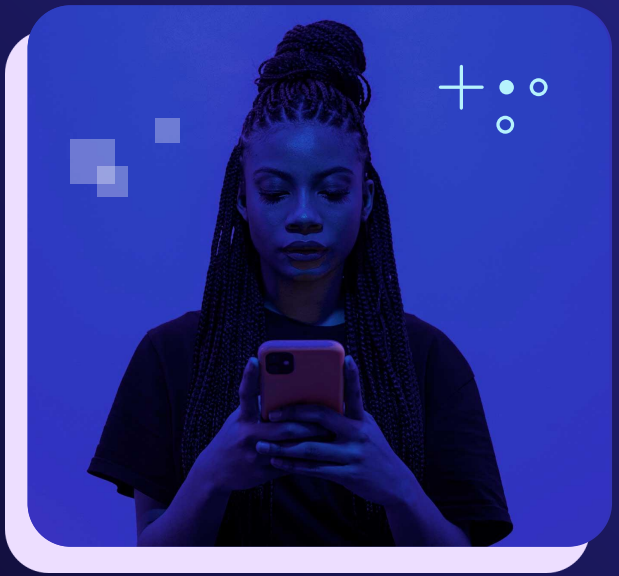
Ejercicio 4

```
public class SpawnObject : MonoBehaviour
{
    public int min = 1;
    public int max = 5;
    public int timeToSpawn = 0;
    public float timer = 0;
    public GameObject objeto = null;

    // Start is called before the first frame update
    void Start()
    {
        timeToSpawn = Random.Range(min, max);
    }

    // Update is called once per frame
    void Update()
    {
        timer += Time.deltaTime;
        if (timer >= timeToSpawn)
        {
            Instantiate(objeto, transform);
            timer = 0;
            timeToSpawn = Random.Range(min, max);
        }
    }
}
```





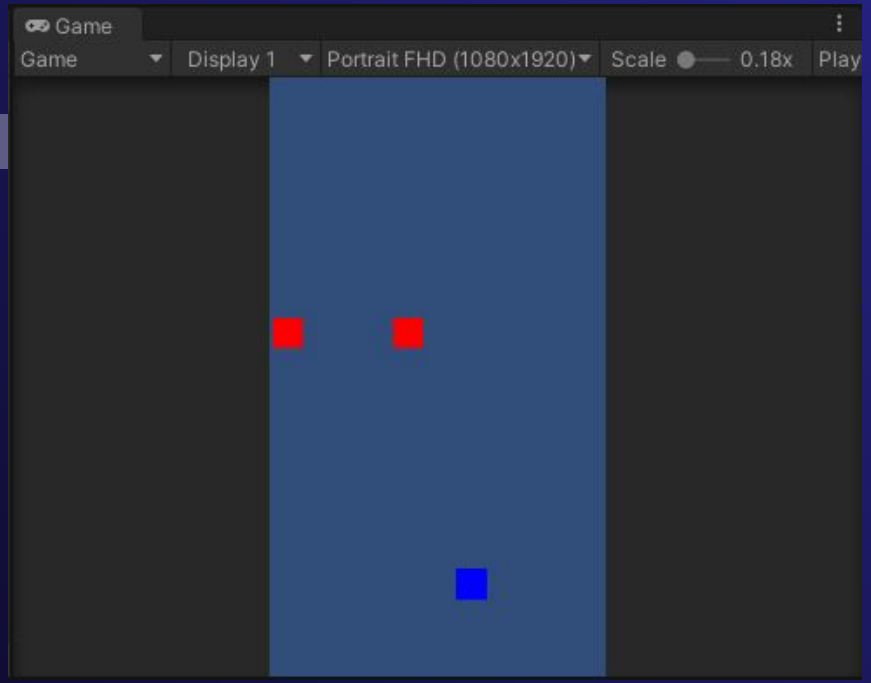




Ejercicio 5

Crear un prototipo de videojuego del clásico Road Fighter de NES. Crear un spawner que cada 4 segundos instancie un auto que se deba esquivar. El jugador puede moverse sólo de izquierda a derecha. El jugador gana 10 puntos por cada segundo de gameplay.

Ejercicio 5

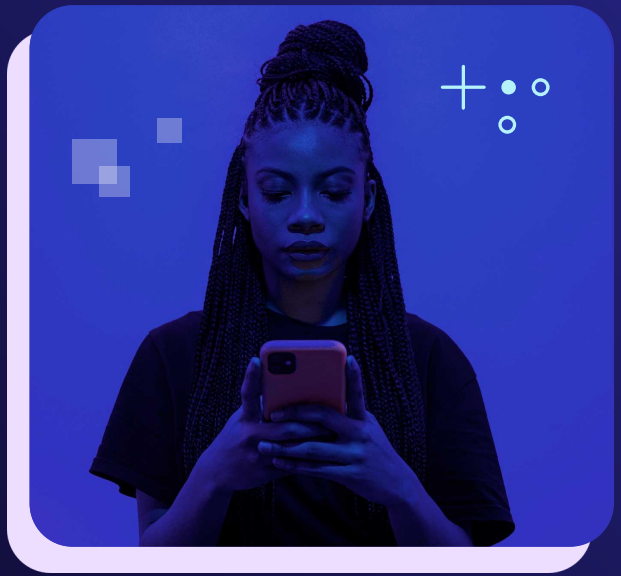



```
public class Enemy : MonoBehaviour
{
    public float speed;

    Unity Message | 0 references
    void Update()
    {
        transform.position += -transform.up * speed * Time.deltaTime;
    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```

Ejercicio 5

```
public class Player : MonoBehaviour
{
    public int score;
    public float speed;
    float x;
    float timer;

    Unity Message | 0 references
    private void Start()
    {
        timer = 0;
    }

    Unity Message | 0 references
    void Update()
    {
        x = Input.GetAxisRaw("Horizontal");

        transform.position += new Vector3(x, 0) * speed * Time.deltaTime;

        timer += Time.deltaTime;
        if (timer >= 1)
        {
            score += 10;
            timer = 0;
        }
    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```

```
public class Spawner : MonoBehaviour
{
    public float timeToSpawn;
    public GameObject obj;
    public float timer;

    Unity Message | 0 references
    private void Start()
    {
        timer = 0;
    }

    Unity Message | 0 references
    void Update()
    {
        timer += Time.deltaTime;
        if (timer >= timeToSpawn)
        {
            timer = 0;
            Instantiate(obj, transform.position, Quaternion.identity);
        }
    }
}
```

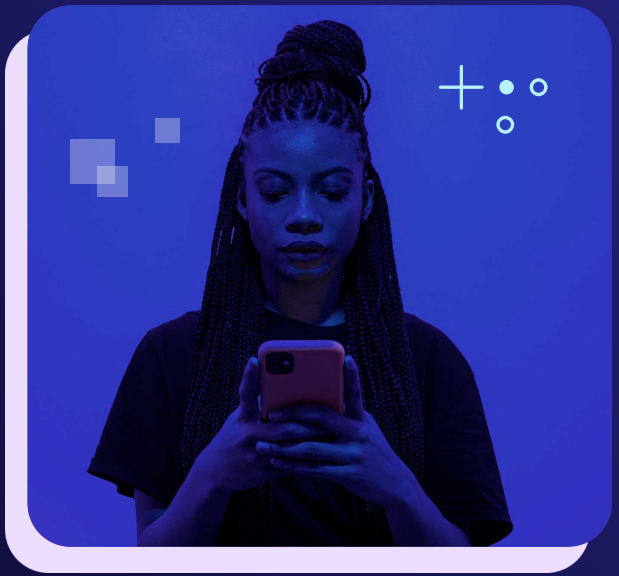


Ejercicio 6

Usando el ejercicio anterior, edite el script de los spawners para generar instancias en un tiempo aleatorio entre 2.5 y 6 segundos.

Ejercicio 6

```
public class Spawner : MonoBehaviour
{
    public float timeToSpawn;
    public GameObject obj;
    public float timer;

    Unity Message | 0 references
    private void Start()
    {
        timer = 0;
        timeToSpawn = Random.Range(2.5f, 6.0f);
    }

    Unity Message | 0 references
    void Update()
    {
        timer += Time.deltaTime;
        if (timer >= timeToSpawn)
        {
            timer = 0;
            timeToSpawn = Random.Range(2.5f, 6.0f);
            Instantiate(obj, transform.position, Quaternion.identity);
        }
    }
}
```

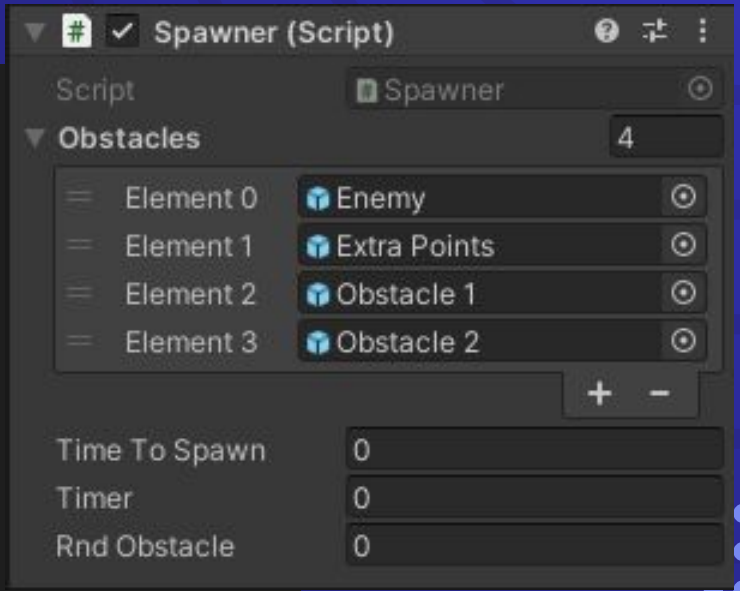


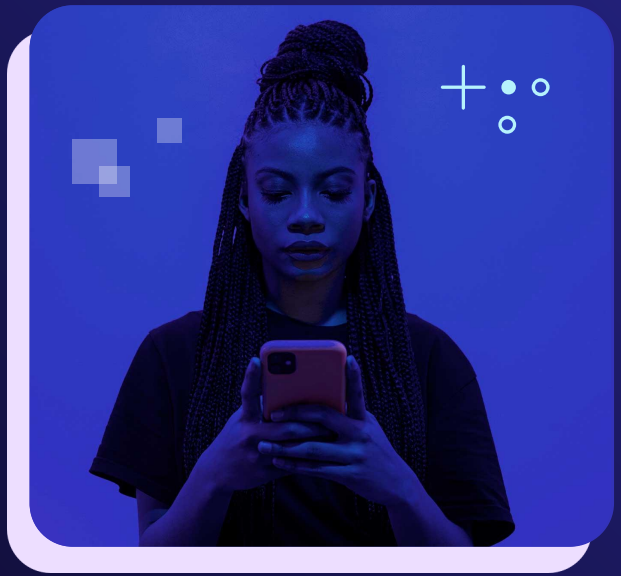
Ejercicio 7

Usando el ejercicio anterior, debe modificar el Script de Spawner agregando un arreglo de obstáculos. Debe seleccionar uno de estos obstáculos con un Random para luego ser instanciado generando un spawn aleatorio de obstáculos.

Ejercicio 7

```
public class Spawner : MonoBehaviour
{
    public GameObject[] obstacles;
    public float timeToSpawn;
    public float timer;
    public int rndObstacle;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        timer = 0;
        timeToSpawn = Random.Range(2.5f, 6f);
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        timer += Time.deltaTime;
        if (timer >= timeToSpawn)
        {
            timer = 0;
            timeToSpawn = Random.Range(2.5f, 6f);
            rndObstacle = Random.Range(0, obstacles.Length);
            Instantiate(obstacles[rndObstacle], transform.position, Quaternion.identity);
        }
    }
}
```



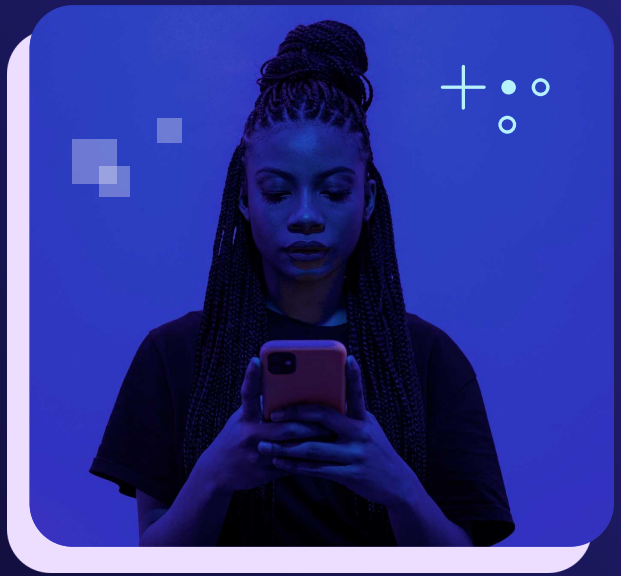


Ejercicio 8

Al Script Enemy agregar una función de destrucción del objeto a los 4 segundos.

Ejercicio 8

```
public class Enemy : MonoBehaviour
{
    public float speed;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        speed = Random.Range(3, 8);
        Destroy(gameObject, 4f);
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        transform.position += -transform.up * speed * Time.deltaTime;
    }

    Mensaje de Unity | 0 referencias
    private void OnCollisionEnter2D(Collision2D collision)
    {
        Destroy(gameObject);
    }
}
```



Ejercicio 9

Usando el ejercicio anterior, agregar un objeto que entregue puntos al jugador al colisionar con él. Este objeto entregará 50 puntos.

Ejercicio 9

```
public class ExtraPoints : MonoBehaviour
{
    public int points;
    public float speed;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        speed = Random.Range(3, 8);
        Destroy(gameObject, 4f);
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        transform.position += -transform.up * speed * Time.deltaTime;
    }

    Mensaje de Unity | 0 referencias
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.gameObject.GetComponent<Player>().score += points;
        }
        Destroy(gameObject);
    }
}
```

IMPORTANTE

Cuentan con herramientas y conocimientos suficientes para crear prototipos de videojuegos: **salto, movimiento, colisiones, tiempo, input de teclado y mouse, colaboración entre objetos y componentes, spawn de objetos, etc**.

De ahora en adelante, su imaginación y motivación serán su límite, que tengan un hermoso viaje creando videojuegos.

BIENVENIDOS A LA PROGRAMACIÓN DE VIDEOJUEGOS!

Introducción a la programación de videojuegos

Sebastián Díaz Arancibia

Universidad Técnica Federico Santa María - 2023