# FUTURITOS

---

**Proyecto Desarrollado en Unity**

**Documentación de código**


**Autores**

Los futuritos

Ruslan Elias Benavides Zadorozhna

Kevin David Garcia Morales

Juan Luis Castillo Sarmiento

José Miguel Grajales Herrera

Alfonso Ramírez Gamboa

**Códigos de audio**

---

**AudioManager.cs**

**void Awake()**

Inicializa el singleton, carga los volúmenes guardados y reproduce la música según la presencia del jugador en la escena.

**void Start()**

Bloquea y oculta el cursor al comenzar el juego.

**void PlayMusicBasedOnPlayerPresence()**

Determina si se debe reproducir la música del menú o del juego según si hay un jugador en la escena.

**void PlaySoundEffect(AudioClip clip, float volume = 1f)**

Reproduce un efecto de sonido una vez, con el volumen especificado.

**void PlayMusic(AudioClip musicClip, float volume = 1f, float fadeDuration = 0f)**

Reproduce una pista de música, opcionalmente con un fundido de entrada. Evita repetir la misma pista si ya se está reproduciendo.

**void StopMusic(float fadeDuration = 0f)**

Detiene la música actual, con opción de fundido de salida.

**IEnumerator FadeInMusic(float duration)**

Corrutina que hace un fundido de entrada al reproducir música.

**IEnumerator FadeOutMusic(float duration)**

Corrutina que hace un fundido de salida al detener la música.

**void SetSceneMusic(AudioClip newClip, float fadeDuration = 0.5f)**

Reproduce una nueva pista de música si es diferente a la actual, con opción de fundido.

**AudioUIManager.cs**

**void Awake()**

Busca y asigna los sliders de audio y música, registra sus eventos, carga los valores guardados y aplica los volúmenes correspondientes.

**void Start()**

Desactiva el panel de pausa al iniciar la escena.

**void SetAudioVolume(float sliderValue)**

Ajusta el volumen de efectos de sonido con una escala no lineal y guarda el valor del slider.

**void SetMusicVolume(float sliderValue)**

Ajusta el volumen de la música con una escala no lineal y guarda el valor del slider.

**MusicaNivel.cs**

**void Start()**

Asigna una nueva pista de música al inicio del nivel usando el AudioManager, si está disponible.

**SliderInput.cs**

**void Update()**

Permite modificar el valor del slider con las teclas de dirección mientras está seleccionado, ajustando la velocidad de cambio con el tiempo.

## Códigos del nivel 1

---

### ColorCycler.cs

#### void Start()

Inicializa las variables del script. Selecciona aleatoriamente un SpriteRenderer para excluirlo del ciclo de cambio de color y guarda el color original de cada sprite en la lista. También establece el temporizador para el cambio de color.

#### void Update()

Actualiza el ciclo de colores. Cuando el temporizador llega a cero, cambia el color de todos los SpriteRenderer en la lista, excepto el sprite excluido. En el segundo cambio de color, también habilita un script llamado GenerarAutos si está presente en el SpriteRenderer.

### Movimiento.cs (Enemigo)

#### void Inicializar(float nuevaVelocidad)

Ajusta la velocidad del objeto sumando nuevaVelocidad a la velocidad actual.

#### void Update()

Mueve el objeto hacia la izquierda cada frame, aplicando la velocidad y el tiempo transcurrido.

### MovimientoVertical.cs (Jugador)

#### void Start()

Inicializa las variables necesarias, incluyendo el Rigidbody2D, la posición inicial en el eje Y (startY), y el componente Animator.

#### void Update()

Detecta la entrada vertical del jugador (arriba o abajo). Si el objeto no ha alcanzado el límite superior o inferior en el eje Y, permite el movimiento y actualiza el estado de la animación. Si está en el límite, mantiene la animación en estado de reposo.

**void FixedUpdate()**

Actualiza la posición del objeto en el eje Y, restringiéndola dentro de los límites establecidos (positiveYLimit y negativeYLimit), y mueve el objeto con el Rigidbody2D de forma suave.

**void SetAnimState(int estado)**

Controla el estado de la animación del objeto, pasando un valor entero que representa el estado de movimiento (arriba, abajo o detenido).

**PlayerHurtbox.cs**

**void OnTriggerEnter2D(Collider2D collision)**

Detecta cuando el jugador entra en contacto con un objeto que tiene el tag "Enemy". Si el jugador tiene vida (vida != null) y está vivo (vida.IsAlive()), le resta salud al jugador llamando a vida.TryTakeDamage() con el daño especificado (damagePerHit).

**Vida.cs**

**void Start()**

Inicializa las variables del script, establece la salud actual igual a la salud máxima, configura el sprite del personaje, anima y controla la UI de las barras de vida.

**void Update()**

Actualiza el progreso del temporizador de supervivencia, comprueba las condiciones de invulnerabilidad y actualiza las iconos de corazones de la UI. También gestiona el estado de la victoria y la activación de la animación de muerte.

**void DisableSpawner()**

Desactiva el objeto del spawner cuando la salud del jugador llega a cero o el temporizador de supervivencia alcanza cierto valor.

**public void TryTakeDamage(int amount)**

Intenta recibir daño, activando la invulnerabilidad y llamando a TakeDamage() solo si el jugador está vivo y no es invulnerable.

**public void TakeDamage(int amount)**

Reduce la salud del jugador al recibir daño. Si la salud llega a cero, activa la animación de muerte y otros efectos relacionados.

**private void TriggerDeath()**

Gestiona la muerte del jugador, iniciando la animación de muerte, haciendo desaparecer el objeto y mostrando la pantalla de Game Over.

**public void Heal(int amount)**

Recupera salud, asegurándose de que no se supere el máximo de salud.

**public bool IsAlive()**

Devuelve un valor booleano que indica si el jugador está vivo.

**public void ResetSurvivalProgress()**

Resetea el progreso del temporizador de supervivencia y actualiza la UI.

**private void BecomeInvulnerable()**

Activa el estado de invulnerabilidad y establece el temporizador de invulnerabilidad.

**private void UpdateHeartIcons()**

Actualiza los iconos de los corazones en la UI en función de la cantidad de salud actual.

**private void BlinkHeartIcons(float alpha)**

Hace que los iconos de corazones parpadeen con un valor de opacidad (alpha), típicamente cuando el jugador es invulnerable.

**private void ResetHeartIconAlpha()**

Restaura la opacidad de los iconos de los corazones a su valor original (totalmente visible).

**BulletShooter.cs**

**void Update()**

Este método ejecuta cada frame y gestiona la lógica del spawning de los enemigos. Se calcula la dificultad, la tasa de disparo ajustada, y cambia el modo de disparo (recto o circular) con el tiempo

**void SpawnBullet(Vector2 direction, Vector3 origin, float difficulty)**

Instancia un enemigo en la posición de origen con la dirección proporcionada. Ajusta la velocidad del enemigo dependiendo de la dificultad y otros parámetros. Configura las propiedades del enemigo, como la dirección, el punto de origen, la velocidad, y las características de movimiento.

**ElementosFondo.cs**

**void Start()**

Este método se ejecuta al inicio del juego. Dependiendo de las configuraciones del script, inicia el patrón de spawn de elementos de fondo. Si se activa el modo de depuración (debugSpawnInstant), se generarán varios elementos de fondo instantáneamente. Si se usa la distancia en lugar del tiempo para generar los elementos (useDistanceInsteadOfTime), se comienza el patrón de spawn basado en la distancia. Si no, se inicia un spawn temporal en intervalos regulares.

**void Update()**

Este método se ejecuta cada frame. Si el modo de spawn usa distancia en lugar de tiempo, calcula la distancia recorrida desde el último punto de spawn y genera un nuevo elemento de fondo cuando se alcanza la distancia establecida. Si no se encuentra en modo depuración, el spawn de elementos se activará solo cuando se haya recorrido la distancia suficiente.

**IEnumerator SpawnTimedPattern()**

Este método genera elementos de fondo en intervalos regulares de tiempo, según el spawnDelay. Los elementos se generan indefinidamente mientras el juego se esté ejecutando.

**void SpawnMultiple(int buildingCount)**

Este método genera múltiples elementos de fondo. Los elementos de fondo se alternan entre edificios y arbustos, según el valor de spawnBuildingNext. El número de elementos generados está determinado por el parámetro buildingCount.

**void SpawnNextElementWithDistance()**

Este método genera el siguiente elemento de fondo basándose en la distancia. Calcula la posición de spawn en el eje X y luego llama a SpawnNextElementAt() para crear el objeto en esa posición.

**void SpawnNextElementAt(Vector3 spawnPos)**

Este método genera un nuevo elemento de fondo en la posición especificada (spawnPos). Alterna entre edificios y arbustos dependiendo de la variable spawnBuildingNext. También configura el movimiento, el color y la escala de los objetos generados, y ajusta el orden de los sprites en función de la capa de orden (sortingLayerOffset). Después de generar un edificio o arbusto, alterna la variable spawnBuildingNext para determinar el siguiente tipo de objeto a generar.

**GeneradorEnemigos.cs**

**void Start()**

Este método se ejecuta al iniciar el juego. Inicializa el temporizador de spawn llamando a ResetSpawnTimer(), para establecer cuándo se generará el primer enemigo.

**void Update()**

Este método se ejecuta cada frame. Si el componente SpawnControl existe y no permite el spawn (AbleToSpawn == false), se detiene la lógica. Si el spawn está habilitado, el temporizador de spawn disminuye y, al llegar a cero, se genera un nuevo enemigo con Spawn() y se reinicia el temporizador con ResetSpawnTimer().

**void ResetSpawnTimer()**

Calcula el tiempo que debe esperar antes de generar el próximo enemigo, ajustándolo según el nivel de dificultad. Aplica un retardo base (baseSpawnDelay) modificado por la dificultad y agrega un valor aleatorio (baseRandomOffset), también ajustado por la dificultad. Se asegura de que ambos valores estén dentro de los rangos mínimos y máximos definidos.

**void Spawn()**

Instancia un nuevo enemigo aleatorio desde el arreglo prefabsToSpawn, en la posición del objeto actual. Si el enemigo tiene un componente BulletEnemy, se ajusta su velocidad en función de la dificultad, aplicando una bonificación de velocidad limitada por los valores establecidos en baseBulletSpeedBonus y maxBulletSpeedBonus.

## GenerarAutos.cs

**void OnEnable()**

Este método se ejecuta automáticamente cuando el objeto que contiene el script se activa en la escena. Inicia la corrutina SpawnCars() para comenzar a generar autos.

**IEnumerator SpawnCars()**

Corrutina que genera tres autos en la posición definida por spawnPlace. Cada auto es elegido aleatoriamente del arreglo carPrefabs. Entre cada aparición hay un pequeño retardo definido por spawnDelay. Al finalizar, desactiva el script (enabled = false) para evitar que vuelva a ejecutarse.

## SpawnControl.cs

**void Start()**

Este método se ejecuta automáticamente al iniciar la escena. Desactiva temporalmente la aparición de enemigos (AbleToSpawn = false) y programa el inicio del juego llamando a StartGame() después de un retardo definido por startDelay.

**void StartGame()**

Método llamado después del retardo inicial. Marca que el juego ha comenzado (hasGameStarted = true) y permite que se inicien las apariciones de enemigos (AbleToSpawn = true).

**void Update()**

Se ejecuta una vez por frame. Si el juego ha comenzado y está habilitado el spawn, acumula el tiempo transcurrido. Cuando el

temporizador alcanza el valor de spawnDelay, lo reinicia, desactiva temporalmente el spawn y programa una aparición con retrasos a través de Invoke() llamando a SpawnWithDelays() después del tiempo preSpawnBlockTime.

## Códigos del nivel 2

---

### AgarrarrCosas.cs

#### void Start()

Se ejecuta al comenzar la escena. Inicializa el temporizador (timerRemaining) con el valor de timerDuration y obtiene la referencia al componente MovimientoHorizontal del jugador.

#### void Update()

Se ejecuta cada frame. Si el juego no ha terminado, actualiza el temporizador y verifica si el valor del slider llegó a 1. Si es así, finaliza el juego como victoria llamando a EndGame(true).

#### void OnTriggerEnter2D(Collider2D collision)

Detecta colisiones con objetos marcados como "Correcto" o "Incorrecto".

Si es "Correcto", aumenta el valor del slider, muestra retroalimentación visual correcta y destruye el objeto.

Si es "Incorrecto", disminuye el valor del slider, muestra retroalimentación visual incorrecta y destruye el objeto.

No se ejecuta si el juego ya terminó.

#### void ModifySlider(float amount)

Modifica el valor del slider dentro de los límites de 0 a 1, sumando o restando la cantidad indicada.

#### void ShowFeedback(ref Coroutine currentRoutine, GameObject feedbackObject)

Activa el objeto de retroalimentación visual (correcto o incorrecto). Si ya hay una corrutina activa, la detiene. Luego, inicia una nueva corrutina para desactivar el objeto tras cierto tiempo.

**IEnumerator DisableAfterDelay(GameObject obj, float delay)**

Corrutina que espera una cantidad de segundos y luego desactiva el objeto visual pasado como parámetro.

**void UpdateTimer()**

Actualiza el temporizador del juego y muestra el tiempo restante en formato minutos:segundos.

Si faltan 10 segundos o menos, cambia el color del texto a rojo.

Si faltan 5 segundos o menos, activa una corrutina para que el texto parpadee.

Si el tiempo llega a cero, termina el juego como derrota llamando a EndGame(false).

**IEnumerator BlinkText()**

Corrutina que hace parpadear el texto del temporizador mientras el tiempo restante sea menor o igual a 5 segundos.

**void EndGame(bool win)**

Finaliza el juego y detiene la aparición de nuevos objetos. Si el parámetro win es verdadero, muestra el objeto de victoria. Si es falso, muestra el objeto de derrota. Además, desactiva el movimiento del jugador.

## MovimientoHorizontal.cs

**void Start()**

Inicializa las referencias necesarias: Rigidbody2D, Animator, y SpriteRenderer. También guarda la posición inicial en X del objeto para limitar su movimiento.

**void Update()**

Se ejecuta cada frame.

Lee la entrada horizontal del jugador y determina si puede moverse hacia la izquierda o derecha, dependiendo de los límites establecidos (positiveXLimit y negativeXLimit).

Establece el estado de animación según la dirección de movimiento y actualiza el vector movimiento.

**void FixedUpdate()**

Se encarga de aplicar el movimiento del jugador de manera física usando Rigidbody2D. Calcula una nueva posición en X y la limita dentro del rango permitido para evitar que el jugador se salga del área designada.

**void SetAnimState(int estado)**

Cambia el parámetro "Estado" del Animator para actualizar la animación del jugador dependiendo de si está quieto (0), moviéndose a la derecha (1) o a la izquierda (2).

## Códigos del nivel 3

---

### SpriteSorting.cs

**void Awake()**

Se ejecuta cuando se inicializa el objeto. Obtiene y guarda la referencia al SpriteRenderer del GameObject para poder modificar su orden de renderizado más adelante.

**void LateUpdate()**

Se ejecuta después de todos los Update() en cada frame. Calcula y asigna dinámicamente el sortingOrder del sprite basado en su posición vertical (Y). Esto asegura que los objetos más abajo en pantalla se dibujan por encima de los que están más arriba, creando un efecto de profundidad.

### RetiradaEnemiga.cs

**void Start()**

Se ejecuta al comenzar el juego. Busca y guarda el Collider2D del jugador. Lanza un error si el objeto no tiene un Collider2D, ya que es necesario para detectar enemigos cercanos.

**public void TriggerRetreat()**

Activa la animación de retirada en enemigos cercanos que tengan el tag "Enemy" y colisionen con el jugador mediante triggers. Usa

OverlapCollider para detectar enemigos en contacto con el Collider2D del jugador. Para cada enemigo detectado: Activa el trigger "Retirada" en su Animator si lo tiene. Detiene su movimiento si tiene el script BulletEnemy. Añade al enemigo a una lista (triggeredEnemies) para evitar que se vuelva a activar inmediatamente. Inicia una corrutina que elimina al enemigo de la lista después de un tiempo (retriggerCooldown).

**private IEnumerator RemoveAfterCooldown(GameObject enemy)**

Corrutina que espera retriggerCooldown segundos antes de permitir que el mismo enemigo pueda ser afectado otra vez por TriggerRetreat. Esto previene que el enemigo reciba múltiples activaciones seguidas.

## Juego3Spawner.cs

**void Start()**

Se ejecuta al comenzar el juego. Verifica si el componente SpawnControl está desactivado o permite generar objetos (AbleToSpawn). Si es así, genera un objeto llamando a Spawn() y luego inicializa el temporizador con ResetSpawnTimer().

**void Update()**

Se ejecuta en cada frame. Detiene la ejecución si SpawnControl no permite generar nuevos objetos. Suma el tiempo transcurrido total (elapsedTime) y reduce el temporizador (spawnTimer). Si el temporizador llega a cero, genera un nuevo objeto y reinicia el temporizador con ResetSpawnTimer().

**void ResetSpawnTimer()**

Calcula el tiempo entre cada aparición. Ajusta dinámicamente la dificultad usando elapsedTime, interpolando entre baseSpawnDelay y minSpawnDelay. Añade un desfase aleatorio usando baseRandomOffset, limitado entre minRandomOffset y maxRandomOffset. Asigna este tiempo al spawnTimer.

**void Spawn()**

Genera una instancia del objeto prefabToSpawn en una posición aleatoria dentro del área delimitada por corner1 y corner2.

Además, tiene un 50% de probabilidad de reflejar el objeto en el eje X invirtiendo su escala horizontal, para añadir variedad visual.

## NivelAgua.cs

### private void Awake()

Se ejecuta antes del Start.

Asigna la instancia actual a la variable estática instance para poder acceder a este script desde otros scripts fácilmente.

### private void Start()

Inicializa el slider al valor máximo (1f) y calcula el tiempo total de cuenta regresiva combinando minutos y segundos. Muestra el tiempo inicial en el texto del temporizador usando UpdateTimerText().

### private void Update()

Se ejecuta en cada frame. Si el juego ya terminó (gameEnded), no hace nada. Si queda tiempo, lo reduce con Time.deltaTime, lo limita a un mínimo de 0 y actualiza el temporizador. Si llega a cero, llama a Victory(). Además, revisa el valor del slider. Si llega a cero, llama a GameOver().

### public void ReduceSlider(float amount)

Disminuye el valor del slider en la cantidad indicada si el juego no ha terminado. Si el valor llega a cero o menos, se llama a GameOver().

### private void UpdateTimerText()

Convierte los segundos restantes a minutos y segundos. Actualiza el texto del temporizador (timerText) para mostrar el tiempo en formato MM:SS.

### private void GameOver()

Marca el juego como terminado (gameEnded = true). Activa la interfaz de Game Over si gameOverUI no es null.

### private void Victory()

Marca el juego como terminado (gameEnded = true). Activa la interfaz de victoria si victoryUI no es null.