

ANGRY PIGS

Práctica 2: *Angry Pigs*

Imagen Digital

Pablo Fernández González

Introducción

1. Introducción	2
2. Motivos de elección del proyecto	2
3. Descripción del juego y alcance	3
4. Requisitos de uso	3
5. Explicación del programa	4
6. Posibles ampliaciones	6
7. Agradecimientos	6
8. Anexo: Códigos de los programas desarrollados	7
8.1. Anexo I: Código del script "AngryBird.cs"	7
8.2. Anexo II: Código del script "CameraManager.cs"	9
8.3. Anexo III: Código del script "GameManager.cs"	10
8.4. Anexo IV: Código del script "IconHandler.cs"	14
8.5. Anexo V: Código del script "InputManager.cs"	15
8.6. Anexo VI: Código del script "Piggy.cs"	16
8.7. Anexo VII: Código del script "SlingShot_Area.cs"	18
8.8. Anexo VIII: Código del script "SlingShot_Handler.cs"	19
8.9. Anexo IX: Código del script "SoundManager.cs"	25

1. Introducción

En el siguiente documento, se detalla el trabajo realizado durante la segunda práctica de Imagen Digital, en la que se solicitó la creación de un programa mediante la plataforma de desarrollo Unity.

Se decidió optar por la recreación del videojuego multiplataforma **Angry Birds**, lanzado en 2009 por la compañía finlandesa **Rovio Entertainment Corporation**, recreando una pequeña demo con niveles jugables.

El objetivo de este documento consiste en describir el trabajo realizado durante su desarrollo. Asimismo, se incluyen manuales de usuario del software para facilitar su uso.

2. Motivos de elección del proyecto

Se escogió Angry Birds como proyecto de desarrollo debido a una serie de factores técnicos y personales:

- **El juego es altamente popular**, por lo que resulta sencilla la obtención de sprites y sonidos necesarios para su creación, así como proyectos similares que ofrezcan apoyo durante su desarrollo.
- **No se implementa una gran cantidad de mecánicas**, por lo que resulta adecuado como una primera toma de contacto con la plataforma de desarrollo utilizada. Del mismo modo, facilita el aprendizaje del lenguaje de programación requerido en su creación.
- Al ser un juego con el que poseo gran **familiaridad** desde mi infancia, su desarrollo resulta entretenido, más aún [tras el cierre del juego original](#).

3. Descripción del juego y alcance

Angry Birds es un videojuego de rompecabezas en el que los jugadores utilizan un tirachinas para lanzar pájaros contra construcciones de distintos materiales elaboradas por cerdos verdes que, según cuenta la historia, han robado los huevos de los primeros.

El juego se hace valer de la precisión y la fuerza de los lanzamientos para derribar las estructuras, y su complejidad avanza según progresan los niveles.

Este proyecto cubre las mecánicas principales del juego, e incluye animaciones, efectos de sonido y cámara para acercarlo más al producto final deseado.

Se incluyen cinco niveles iniciales, con la posibilidad de poder expandirse en un futuro e implementar un sistema de selección de niveles.

4. Requisitos de uso

Con respecto a cuestiones de compatibilidad, este software puede ser ejecutado en sistemas *Windows* (en formato *.exe*) y dispositivos *Android* (en formato *.apk*). El almacenamiento requerido por cada versión es de **112 MB** y **~36.000 KB**, respectivamente.

Del mismo modo, necesita una resolución de pantalla de **16:9** en ordenadores, y el uso del **modo panorámico** en móviles.

5. Explicación del programa

El proyecto está compuesto por una serie de scripts que dan funcionalidades a ciertos componentes de este. Con respecto a la propia jugabilidad, destacan los siguientes:

- **SlingShot_Handler.cs**: implementa la lógica de un **tirachinas** en el entorno de nuestro proyecto. Su principal objetivo es gestionar la interacción del jugador con el tirachinas y el lanzamiento de pájaros hacia un objetivo.
 - **SlingShot_Area.cs**: se encarga de gestionar la **detección de la interacción del jugador** con el área del tirachinas, a partir del componente **LayerMask** **slingshotAreaMask**. Utilizado por **SlingShot_Handler.cs**.
 - **InputManager.cs**: gestiona los **controles** del usuario relacionados con el ratón o el toque en pantalla táctil. Permite la compatibilidad del juego tanto con sistemas de entrada *Windows* como los utilizados por dispositivos *Android*.
- **AngryBird.cs**: se asocia a los **pájaros** del juego. Permite sus lanzamientos, la orientación que poseen basada en su velocidad y sus estados al colisionar.

Se combina con componentes de física (tales como **Rigidbody2D** y **CircleCollider2D**) para simular un desplazamiento realista en el aire.

- **Piggie.cs**: gestiona el comportamiento de los **cerdos**: controla la salud, el daño recibido por colisiones y su “muerte”, incluyendo efectos visuales y sonoros.
- **GameManager.cs**: es el **núcleo del juego**, encargado de manejar el estado general de la partida. Esto incluye:
 - La gestión de los disparos disponibles.
 - El seguimiento de los cerdos (enemigos) restantes.
 - La lógica de victoria o derrota de la partida.
 - El reinicio de niveles y la posibilidad de acceder al siguiente.

Del mismo modo, se incluyen los siguientes scripts, que gestionan otros ámbitos relacionados con otras áreas del juego:


- **CameraManager.cs**: gestiona el cambio entre **cámaras** del juego: una es estática ("*idle*"), mientras que otra se invoca para seguir al pájaro durante su lanzamiento.
- **IconManager.cs**: controla la actualización visual de los iconos de la interfaz de usuario, que representan los tiros disponibles en un juego.

Cambiar el color de uno de los iconos de pájaro de la interfaz cada vez que se utiliza un tiro, indicando visualmente que se ha consumido.

- **SoundManager.cs**: maneja los sonidos reproducidos en el juego, ya sean sonidos determinados o aleatorios pertenecientes a un listado.

6. Posibles ampliaciones

Se consideran las siguientes adiciones y modificaciones como posibles ampliaciones futuras del proyecto:

- Adición de más pájaros, junto con sus respectivos poderes.
- Posibilidad de romper bloques de estructuras.
- Selector de niveles y menú principal.
- Sistema de puntuación.
- Guardado del progreso alcanzado.
- Adición de bloques de dinamita.
- Adición de “huevos de pascua” (disparadores del juego escondidos capaces de desbloquear niveles secretos) 

7. Agradecimientos

Este proyecto no hubiera sido posible sin el uso de imágenes y sonidos pertenecientes a *Rovio Entertainment Corporation*. Del mismo modo, la implementación de este trabajo se debe en gran medida al asesoramiento recibido por el canal de YouTube anglosajón [sasquatchbgames](https://www.youtube.com/channel/UCsasquatchbgames).

Finalmente, agradezco al profesor de la asignatura de Imagen Digital, Pablo García Rodríguez, quien me apoyó en la toma de decisiones de este proyecto y al que le agradezco su pasión por el contenido que enseña y el trabajo realizado por sus alumnos.

8. Anexo: Códigos de los programas desarrollados

A continuación, se detalla el funcionamiento de cada uno de los *scripts* desarrollados durante este proyecto, con comentarios entre líneas que detallan la función de cada una de sus funciones.

8.1. Anexo I: Código del script “AngryBird.cs”

```
using UnityEngine;

public class AngryBird : MonoBehaviour
{
    // Sonidos de colisión
    [SerializeField] private AudioClip[] _hitClips;

    // Componentes de físicas
    private Rigidbody2D _rb;
    private CircleCollider2D _circleCollider;

    // Bandera que indica si el pájaro ha sido lanzado
    private bool _hasBeenLaunched;
    // Bandera que indica la orientación del pájaro (cara)
    private bool _shouldFaceVelDirection;
    // Fuente de sonidos del pájaro
    private AudioSource _audioSource;

    // Awake: Preparación pájaro (componentes y fuente de audio)
    private void Awake(){
        _rb = GetComponent<Rigidbody2D>();
        _circleCollider = GetComponent<CircleCollider2D>();
        _audioSource = GetComponent<AudioSource>();
    }

    // Start: Colisionador desconectado y estado de Rigidbody modificado
    private void Start()
    {
        _rb.bodyType = RigidbodyType2D.Kinematic;
        _circleCollider.enabled = false;
    }

    // FixedUpdate: Si se lanza el pájaro y se encuentra en la dirección del
    // lanzamiento, su transformada pasa a ser la velocidad lineal de su componente
```



```
Rigidbody
private void FixedUpdate()
{
    if (_hasBeenLaunched && _shouldFaceVelDirection){
        transform.right = _rb.linearVelocity;
    }
}

// LaunchBird: Cambian las físicas del pájaro y se activa su
colisionador. Se añade una fuerza a su lanzamiento
public void LaunchBird(Vector2 direction, float force)
{
    _rb.bodyType = RigidbodyType2D.Dynamic;
    _circleCollider.enabled = true;

    _rb.AddForce(direction * force, ForceMode2D.Impulse);

    _hasBeenLaunched = true;
    _shouldFaceVelDirection = true;
}

// OnCollisionEnter2D: AL chocar con algo, deja de orientarse a una
posición definida, se reproduce un sonido y se destruye su instancia (NO
DEL JUEGO)
private void OnCollisionEnter2D(Collision2D collision)
{
    _shouldFaceVelDirection = false;
    SoundManager.instance.PlayRandomClip(_hitClips, _audioSource);
    Destroy(this);
}
}
```

Listing 1. Código del programa AngryBird.cs.

8.2. Anexo II: Código del script “CameraManager.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Cinemachine;

public class CameraManager : MonoBehaviour
{
    // Posición de cámara por defecto
    [SerializeField] private CinemachineVirtualCamera _idleCam;
    // Posición de cámara de seguimiento del pájaro
    [SerializeField] private CinemachineVirtualCamera _followCam;

    // Awake: Invocación de cámara por defecto al inicio
    private void Awake()
    {
        SwitchToIdleCam();
    }

    // SwitchToIdleCam: Cambio a cámara por defecto
    public void SwitchToIdleCam()
    {
        _idleCam.enabled = true;
        _followCam.enabled = false;
    }

    // SwitchToFollowCam: Cambio a cámara del pájaro lanzado
    public void SwitchToFollowCam(Transform followTransform)
    {
        _followCam.Follow = followTransform;

        _followCam.enabled = true;
        _idleCam.enabled = false;
    }
}
```

Listing 2. Código del programa CameraManager.cs.

8.3. Anexo III: Código del script “GameManager.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using DG.Tweening;
using UnityEngine.UI;

public class GameManager : MonoBehaviour
{
    public static GameManager instance;

    // Número máximo de Lanzamientos
    public int MaxNumberOfShots = 3;
    // Tiempo de espera hasta dar por perdida una partida
    [SerializeField] private float _secondsToWaitBeforeDeathCheck = 6f;
    // Objeto de reinicio de escenario
    [SerializeField] private GameObject _restartScreenObject;
    // Script del tirachinas
    [SerializeField] private SlingShot_Handler _slingShotHandler;
    // Imagen (botón) de acceso al siguiente nivel
    [SerializeField] private Image _nextLevelImage;

    //Número de tiros gastados
    private int _usedNumberOfShots;

    // Gestor de iconos de UI
    private IconHandler _iconHandler;

    // Lista de cerdos en escena
    private List<Piggie> _piggies = new List<Piggie>();

    // Awake: detección del gestor de iconos, almacenamiento de cerdos
    // en escena
    // y pestaña de cambio de nivel desactivada
    private void Awake()
    {
        // Patrón singleton
        if (instance == null)
        {
            instance = this;
        }
    }
}
```

```
        _iconHandler = FindFirstObjectByType<IconHandler>();

        Piggie[] piggies =
Object.FindObjectsByType<Piggie>(FindObjectsSortMode.None);
        for (int i = 0; i < piggies.Length; i++)
        {
            _piggies.Add(piggies[i]);
        }

        _nextLevelImage.enabled = false;
    }

    // UseShot: actualización del número de lanzamientos y de la UI.
    // Se comprueba si es el último tiro
    public void UseShot()
    {
        _usedNumberOfShots++;
        _iconHandler.UseShot(_usedNumberOfShots);
        CheckForLastShot();
    }

    // HasEnoughShots: comprobación de la existencia de tiros
    // disponibles (Usada en Slingshot_Handler)
    public bool HasEnoughShots()
    {
        if (_usedNumberOfShots < MaxNumberOfShots)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    // CheckForLastShot: Comprobación del último tiro
    public void CheckForLastShot()
    {
        if (_usedNumberOfShots == MaxNumberOfShots)
        {
            StartCoroutine(CheckAfterWaitTime());
        }
    }

    // CheckAfterWaitTime: Tiempo de espera hasta reiniciar el nivel
    private IEnumerator CheckAfterWaitTime()
```

```
{
    yield return new WaitForSeconds(_secondsToWaitBeforeDeathCheck);

    if (_piggies.Count == 0)
    {
        WinGame();
    }
    else
    {
        RestartGame();
    }
}

// RemovePiggy: borra un cerdo y comprueba si queda alguno en
pantalla
public void RemovePiggy(Piggy piggy)
{
    _piggies.Remove(piggy);
    CheckForAllDeadPiggies();
}

// RemovePiggy: comprueba si quedan cerdos en pantalla para activar
el estado de victoria
private void CheckForAllDeadPiggies()
{
    if (_piggies.Count == 0)
    {
        WinGame();
    }
}

#region Win/Lose

// WinGame: se desactiva el tirachinas y aparece la interfaz de
reinicio y siguiente nivel, si lo hay
private void WinGame()
{
    _restartScreenObject.SetActive(true);
    _slingShotHandler.enabled = false;

    int currentSceneIndex =
SceneManager.GetActiveScene().buildIndex;
    int maxLevels = SceneManager.sceneCountInBuildSettings;
    if (currentSceneIndex + 1 < maxLevels)
    {
        _nextLevelImage.enabled = true;
    }
}
```

```
    }  
}  
  
// RestartGame: se reinicia el nivel  
public void RestartGame()  
{  
    DOTween.Clear(true);  
  
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);  
}  
  
// RestartGame: se carga el siguiente nivel  
public void NextLevel()  
{  
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex  
+ 1);  
}  
  
#endregion  
  
// ExitGame: cierre del juego  
public void ExitGame () {  
    Application.Quit ();  
    Debug.Log("Bye!");  
}  
}
```

Listing 3. Código del programa GameManager.cs.

8.4. Anexo IV: Código del script “IconHandler.cs”

```
using UnityEngine;
using UnityEngine.UI;

public class IconHandler : MonoBehaviour
{
    // Listado de iconos de la UI
    [SerializeField] private Image[] _icons;

    // Color utilizado para indicar que un pájaro se ha lanzado
    [SerializeField] private Color _usedColor;

    // UseShot: Actualización de iconos de pájaros en la UI.
    // Se colorean los iconos en función de los pájaros utilizados
    public void UseShot(int shotNumber)
    {
        for (int i = 0; i < _icons.Length; i++)
        {
            if (shotNumber == i + 1)
            {
                _icons[i].color = _usedColor;
                return;
            }
        }
    }
}
```

Listing 4. Código del programa IconHandler.cs.

8.5. Anexo V: Código del script “InputManager.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class InputManager : MonoBehaviour
{
    public static PlayerInput PlayerInput;

    private InputAction _mousePositionAction;
    private InputAction _mouseAction;

    public static Vector2 MousePosition;
    public static bool WasLeftMouseButtonPressed;
    public static bool WasLeftMouseButtonReleased;
    public static bool IsLeftMousePressed;

    // Nota: código compatible con inputs desde PC y Android

    // Awake: obtención de la fuente de input del jugador
    private void Awake()
    {
        PlayerInput = GetComponent<PlayerInput>();

        _mousePositionAction = PlayerInput.actions["MousePosition"];
        _mouseAction = PlayerInput.actions["Mouse"];
    }

    // Update: obtención de acciones del ratón / pantalla táctil
    private void Update()
    {
        MousePosition = _mousePositionAction.ReadValue<Vector2>();

        WasLeftMouseButtonPressed = _mouseAction.WasPressedThisFrame();
        WasLeftMouseButtonReleased =
        _mouseAction.WasReleasedThisFrame();
        IsLeftMousePressed = _mouseAction.IsPressed();
    }
}
```

Listing X. Código del programa InputManager.cs.

8.6. Anexo VI: Código del script “Piggie.cs”

```
using UnityEngine;

public class Piggie : MonoBehaviour
{
    [Header("Pig's Health")]
    // Salud máxima de Los cerdos
    [SerializeField] private float _maxHealth = 3f;

    // Umbral de daño de Los cerdos, para que solo les afecte si se supera
    [SerializeField] private float _damageThreshold = 0.2f;

    //Partículas de muerte del cerdo
    [SerializeField] private GameObject _piggieDeathParticles;

    //Sonido de muerte del cerdo
    [SerializeField] private AudioClip _deathClip;

    // Salud actual del cerdo
    private float _currentHealth;

    // Awake: asigna la salud actual del cerdo
    private void Awake()
    {
        _currentHealth = _maxHealth;
    }

    // DamagePiggie: decrementa la salud del cerdo en función del daño realizado
    public void DamagePiggie(float damageAmount)
    {
        _currentHealth -= damageAmount;

        if (_currentHealth <= 0f)
        {
            Die();
        }
    }

    // Die: "Mata" al cerdo (borra su instancia)
    private void Die()
    {
        GameManager.instance.RemovePiggie(this);
    }
}
```

```
        Instantiate(_piggieDeathParticles, transform.position,
Quaternion.identity);

        AudioSource.PlayClipAtPoint(_deathClip, transform.position);
        Destroy(gameObject);
    }

    // OnCollisionEnter2D: si el cerdo recibe una colisión, se calcula
    // la velocidad del impacto y,
    // en caso de superar el umbral de daño, decrementa la salud del
    // cerdo
    private void OnCollisionEnter2D(Collision2D collision)
    {
        float impactVelocity = collision.relativeVelocity.magnitude;
        if (impactVelocity > _damageThreshold)
        {
            DamagePiggie(impactVelocity);
        }
    }
}
```

Listing 6. Código del programa Piggie.cs.

8.7. Anexo VII: Código del script “SlingShot_Area.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class SlingShot_Area : MonoBehaviour
{
    [SerializeField] private LayerMask _slingshotAreaMask;

    // IsWithinSlingshotArea: Comprobación de clics en el área del
    // tirachinas
    public bool IsWithinSlingshotArea()
    {
        Vector2 worldPosition =
        Camera.main.ScreenToWorldPoint(InputManager.MousePosition);

        if (Physics2D.OverlapPoint(worldPosition, _slingshotAreaMask))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

Listing 7. Código del programa SlingShot_Area.cs.

8.8. Anexo VIII: Código del script “SlingShot_Handler.cs”

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;
using DG.Tweening;

public class SlingShot_Handler : MonoBehaviour
{
    [Header("Line Renderers")]
    // Renderizadores de líneas
    [SerializeField] private LineRenderer _leftLineRenderer;
    [SerializeField] private LineRenderer _rightLineRenderer;

    [Header("Transform References")]
    // Posiciones de las gomas del tirachinas
    [SerializeField] private Transform _leftStartPosition;
    [SerializeField] private Transform _rightStartPosition;
    [SerializeField] private Transform _centerPosition;
    [SerializeField] private Transform _idlePosition;
    [SerializeField] private Transform _elasticTransform;

    [Header("Slingshot Stats")]
    // Distancia máxima de elasticidad del tirachinas
    [SerializeField] private float _maxDistance = 3.5f;

    // Fuerza de lanzamiento
    [SerializeField] private float _shotForce = 9f;

    // Tiempo necesario entre pájaros
    [SerializeField] private float _timeBetweenBirdRespawns = 2f;

    // Constante utilizada para animar el tirachinas
    [SerializeField] private float _elasticDivider = 1.2f;

    // Animación del tirachinas (modificable en Unity)
    [SerializeField] private AnimationCurve _elasticCurve;

    // Tiempo máximo de animación de gomas del tirachinas
    [SerializeField] private float _maxAnimationTime = 1f;

    [Header("Scripts")]
    // Script de zona de detección de tirachinas
    [SerializeField] private SlingShot_Area _slingShotArea;
```

```
// Gestor de La cámara
[SerializeField] private CameraManager _cameraManager;

[Header("Bird")]
// "Prefab" del pájaro
[SerializeField] private AngryBird _angryBirdPrefab;
// Offset del pájaro con respecto al tirachinas
[SerializeField] private float _angryBirdPositionOffset = 2f;

[Header("Sounds")]
// Sonido de tensión de gomas
[SerializeField] private AudioClip _elasticPulledClip;
// Sonido de lanzamiento del pájaro
[SerializeField] private AudioClip[] _elasticReleasedClips;

// Posiciones de Las líneas del tirachinas
private Vector2 _slingShotLinesPosition;

// Dirección
private Vector2 _direction;
private Vector2 _directionNormalized;

// Bandera para comprobar si se hace clic en el area esperada
(Lanzamiento)
private bool _clickedWithinArea;
// Bandera para comprobar si existe algún pájaro en el tirachinas
private bool _birdOnSlingshot;

// Pájaro instanciado
private AngryBird _spawnedAngryBird;

// Fuente de sonido (tirachinas)
private AudioSource _audioSource;

// Awake: Preparación tirachinas (fuente de audio y renderizadores)
private void Awake()
{
    _audioSource = GetComponent();

    _leftLineRenderer.enabled = false;
    _rightLineRenderer.enabled = false;

    SpawnAngryBird();
}
```

```
// Update: Función ejecutada cada frame
private void Update()
{
    // Si se pulsa en el área con clic izquierdo, suena el sonido de
    // la goma y se actualiza la posición de la cámara
    if (InputManager.WasLeftMouseButtonPressed &&
        _slingShotArea.IsWithinSlingshotArea())
    {
        _clickedWithinArea = true;

        if (_birdOnSlingshot)
        {
            SoundManager.instance.PlayClip(_elasticPulledClip,
            _audioSource);

            _cameraManager.SwitchToFollowCam(_spawnedAngryBird.transform);
        }
    }

    // Si se mantiene el clic izquierdo, se ha pulsado inicialmente
    // en el área y hay un pájaro en el tirachinas,
    // se dibuja su posición y la orientación del pájaro
    if (InputManager.IsLeftMousePressed && _clickedWithinArea &&
        _birdOnSlingshot)
    {
        DrawSlingShot();
        PositionAndRotateAngryBird();
    }

    // Si se levanta el clic izquierdo y se cumplen las condiciones
    // anteriores, se lanza el pájaro, se reproduce un sonido y se renderizan
    // las nuevas líneas del tirachinas. En caso de quedar pájaros,
    // se instancian en el tirachinas
    if (InputManager.WasLeftMouseButtonReleased && _birdOnSlingshot
        && _clickedWithinArea)
    {
        if (GameManager.instance.HasEnoughShots())
        {
            _clickedWithinArea = false;
            _birdOnSlingshot = false;

            _spawnedAngryBird.LaunchBird(_direction, _shotForce);

            SoundManager.instance.PlayRandomClip(_elasticReleasedClips,
            _audioSource);
        }
    }
}
```

```
        GameManager.instance.UseShot();
        //SetLines(_centerPosition.position); -- Utilizada antes
de la creación de la función AnimateSlingShot
        AnimateSlingShot();

        if (GameManager.instance.HasEnoughShots())
        {
            StartCoroutine(SpawnAngryBirdAfterTime());
        }
    }
}

#region SlingShot Methods

// DrawSlingShot: Renderizado del estado del tirachinas
private void DrawSlingShot()
{
    Vector3 touchPosition =
Camera.main.ScreenToWorldPoint(InputManager.MousePosition);

    _slingShotLinesPosition = _centerPosition.position +
Vector3.ClampMagnitude(touchPosition - _centerPosition.position,
_maxDistance);

    Setlines(_slingShotLinesPosition);

    _direction = (Vector2)_centerPosition.position -
_slingShotLinesPosition;
    _directionNormalized = _direction.normalized;
}

// Setlines: Posicionamiento de líneas por defecto en el tirachinas
private void Setlines(Vector2 position)
{
    if (!_leftLineRenderer.enabled && !_rightLineRenderer.enabled)
    {
        _leftLineRenderer.enabled = true;
        _rightLineRenderer.enabled = true;
    }

    _leftLineRenderer.SetPosition(0, position);
    _leftLineRenderer.SetPosition(1, _leftStartPosition.position);

    _rightLineRenderer.SetPosition(0, position);
```

```
        _rightLineRenderer.SetPosition(1, _rightStartPosition.position);
    }

#endregion

#region Angry Bird Methods

// SpawnAngryBird: Instanciación del pájaro
private void SpawnAngryBird()
{
    _elasticTransform.DOComplete();
    Setlines(_idlePosition.position);

    Vector2 dir = (_centerPosition.position -
_idlePosition.position).normalized;
    Vector2 spawnPosition = (Vector2)_idlePosition.position + dir *
_angryBirdPositionOffset;

    _spawnedAngryBird = Instantiate(_angryBirdPrefab, spawnPosition,
Quaternion.identity);
    _spawnedAngryBird.transform.right = dir;

    _birdOnSlingshot = true;
}

// PositionAndRotateAngryBird: Posición y rotación del pájaro
private void PositionAndRotateAngryBird()
{
    _spawnedAngryBird.transform.position = _slingShotLinesPosition +
_directionNormalized * _angryBirdPositionOffset;
    _spawnedAngryBird.transform.right = _directionNormalized;
}

// SpawnAngryBirdAfterTime: Reinstanciación de un pájaro en el
tirachinas tras un tiempo
private IEnumerator SpawnAngryBirdAfterTime()
{
    yield return new WaitForSeconds(_timeBetweenBirdRespawns);

    SpawnAngryBird();

    _cameraManager.SwitchToIdleCam();
}

#endregion
```



```
#region Animate SlingShot

// AnimateSlingShot: Animación del tirachinas
private void AnimateSlingShot()
{
    _elasticTransform.position = _leftLineRenderer.GetPosition(0);

    float dist = Vector2.Distance(_elasticTransform.position,
    _centerPosition.position);

    float time = dist / _elasticDivider;

    _elasticTransform.DOMove(_centerPosition.position,
    time).SetEase(_elasticCurve);
    StartCoroutine(AnimateSlingShotLines(_elasticTransform, time));
}

// AnimateSlingShotLines: Corrutina de animación de gomas del
tirachinas
private IEnumerator AnimateSlingShotLines(Transform trans, float
time)
{
    float elapsedTime = 0f;
    while (elapsedTime < time && elapsedTime < _maxAnimationTime)
    {
        elapsedTime += Time.deltaTime;

        Setlines(trans.position);

        yield return null;
    }
}

#endregion
}
```

Listing 8. Código del programa SlingShot_Handler.cs.

8.9. Anexo IX: Código del script “SoundManager.cs”

```
using UnityEngine;

public class SoundManager : MonoBehaviour
{
    public static SoundManager instance;

    // Awake: Aplicación de patrón Singleton para crear una única instancia de la clase
    public void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
    }

    // PlayClip: Reproducción de un sonido determinado
    public void PlayClip(AudioClip clip, AudioSource source)
    {
        source.clip = clip;
        source.Play();
    }

    // PlayRandomClip: Reproducción de un sonido aleatorio de una lista de sonidos
    public void PlayRandomClip(AudioClip[] clips, AudioSource source)
    {
        int randomindex = Random.Range(0, clips.Length);

        source.clip = clips[randomindex];
        source.Play();
    }
}
```

Listing 9. Código del programa SoundManager.cs.