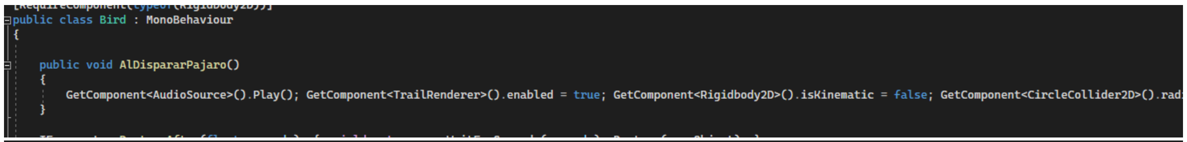
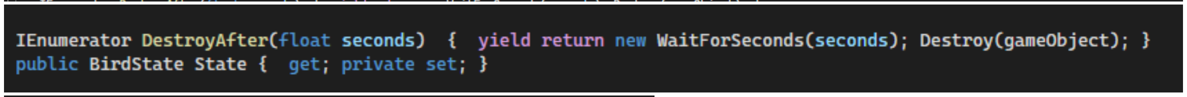
1. **Revisión y refactorización de código**

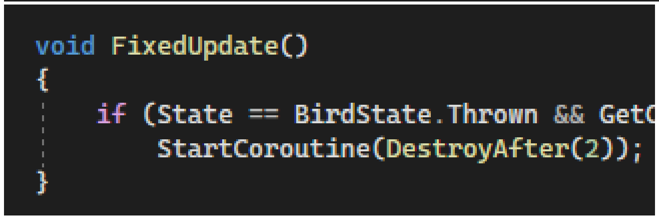
   Bird.cs Monobehaviour OLD



```
[RequireComponent(typeof(Rigidbody2D))]
public class Bird : MonoBehaviour
{

    public void AlDispararPajaro()
    {
        GetComponent<AudioSource>().Play(); GetComponent<TrailRenderer>().enabled = true; GetComponent<Rigidbody2D>().isKinematic = false; GetComponent<CircleCollider2D>().rad
    }
```



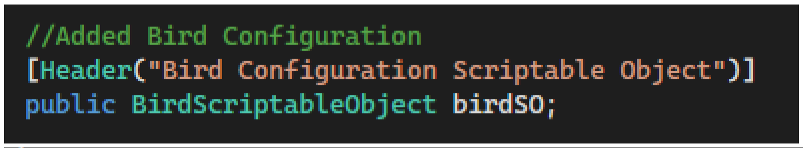
```
IEnumerator DestroyAfter(float seconds)  {  yield return new WaitForSeconds(seconds); Destroy(gameObject); }
public BirdState State {  get; private set; }
```



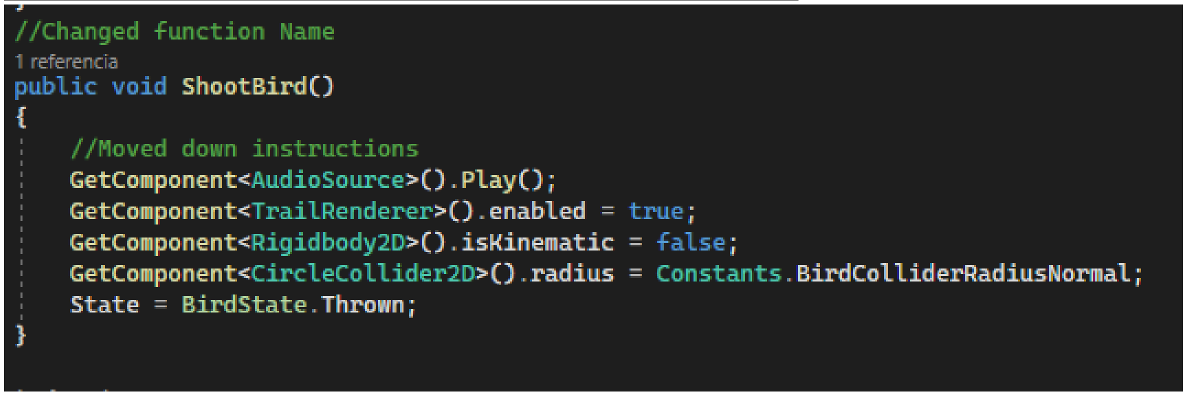
```
void FixedUpdate()
{
    if (State == BirdState.Thrown && GetC
        StartCoroutine(DestroyAfter(2));
}
```

- Cambio de nombre de funciones (español -> inglés)
- Se movieron las instrucciones por legibilidad
- Cambio de números mágicos por declarados o configurados de ScriptableObject: Bird

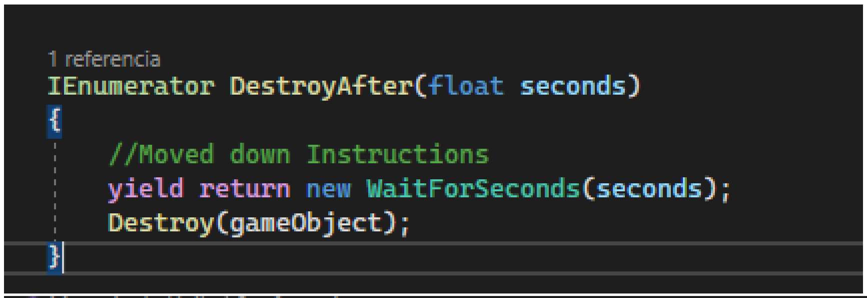
Bird.cs Monobehaviour NEW



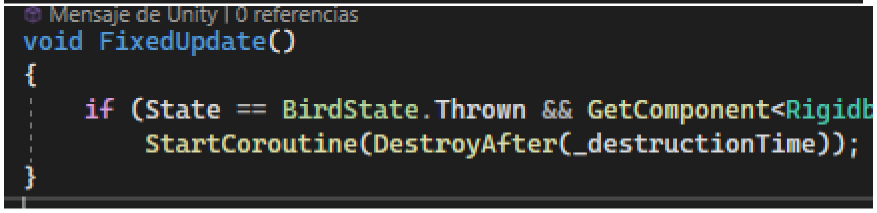
```
//Added Bird Configuration
[Header("Bird Configuration Scriptable Object")]
public BirdScriptableObject birdSO;
```



```
//Changed function Name
1 referencia
public void ShootBird()
{
    //Moved down instructions
    GetComponent<AudioSource>().Play();
    GetComponent<TrailRenderer>().enabled = true;
    GetComponent<Rigidbody2D>().isKinematic = false;
    GetComponent<CircleCollider2D>().radius = Constants.BirdColliderRadiusNormal;
    State = BirdState.Thrown;
}
```

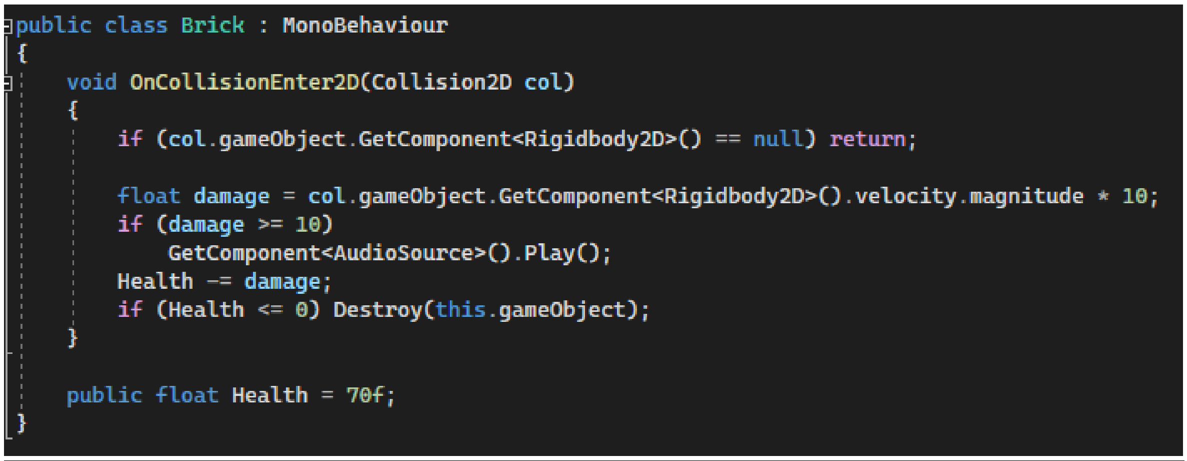


```
1 referencia
IEnumerator DestroyAfter(float seconds)
{
    //Moved down Instructions
    yield return new WaitForSeconds(seconds);
    Destroy(gameObject);
}
```



```
Mensaje de Unity | 0 referencias
void FixedUpdate()
{
    if (State == BirdState.Thrown && GetComponent<Rigidb
        StartCoroutine(DestroyAfter(_destructionTime));
}
```

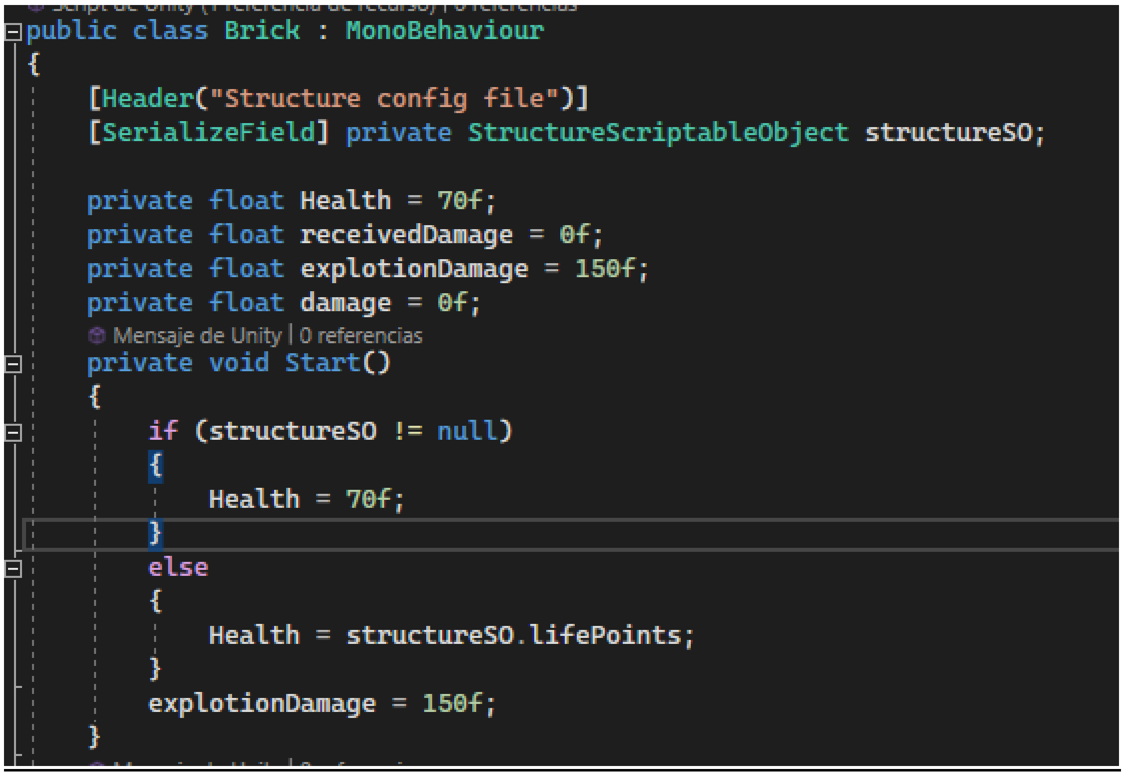
<u>Bricks.cs Monobehaviour OLD</u>



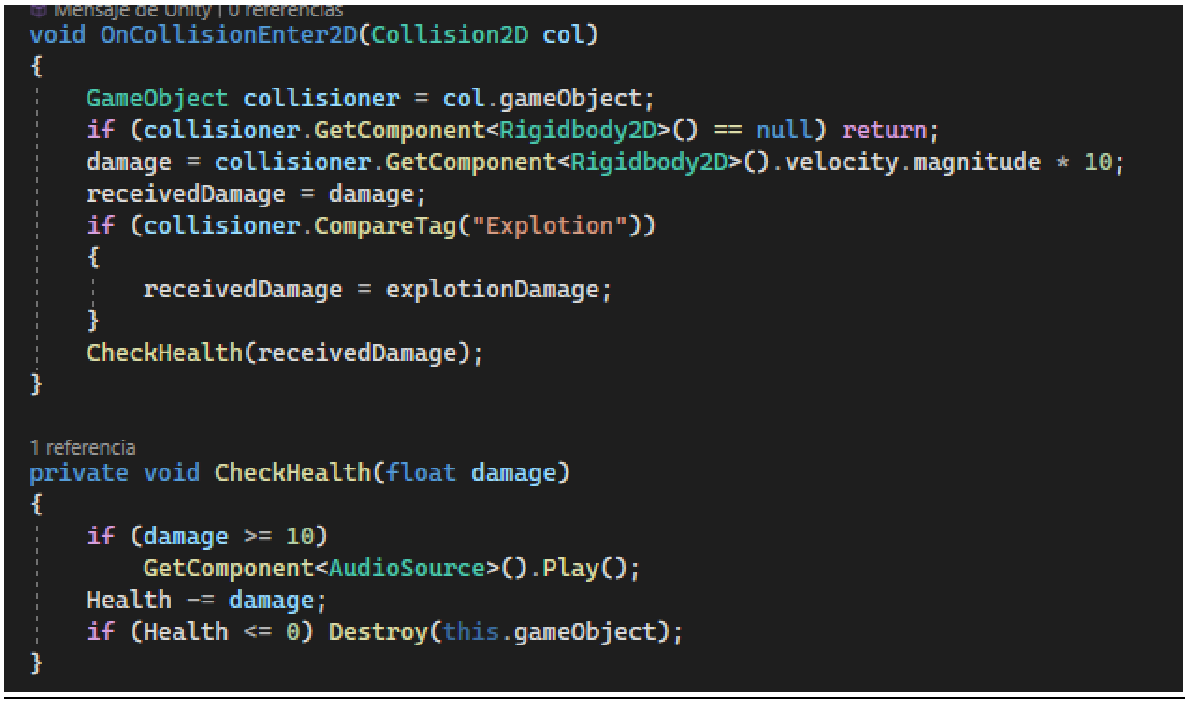
```
public class Brick : MonoBehaviour
{
    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.GetComponent<Rigidbody2D>() == null) return;

        float damage = col.gameObject.GetComponent<Rigidbody2D>().velocity.magnitude * 10;
        if (damage >= 10)
            GetComponent<AudioSource>().Play();
        Health -= damage;
        if (Health <= 0) Destroy(this.gameObject);
    }

    public float Health = 70f;
}
```

- Creación de archivo configurable StructureScriptableObject
- Cambio de números mágicos por declarados o configurados de ScriptableObject: StructureScriptableObject
- Separación de chequeo de puntos de vida de la estructura al llamarse CollisionEnter
- Reemplazo de == por CompareTag

Bricks.cs Monobehaviour NEW

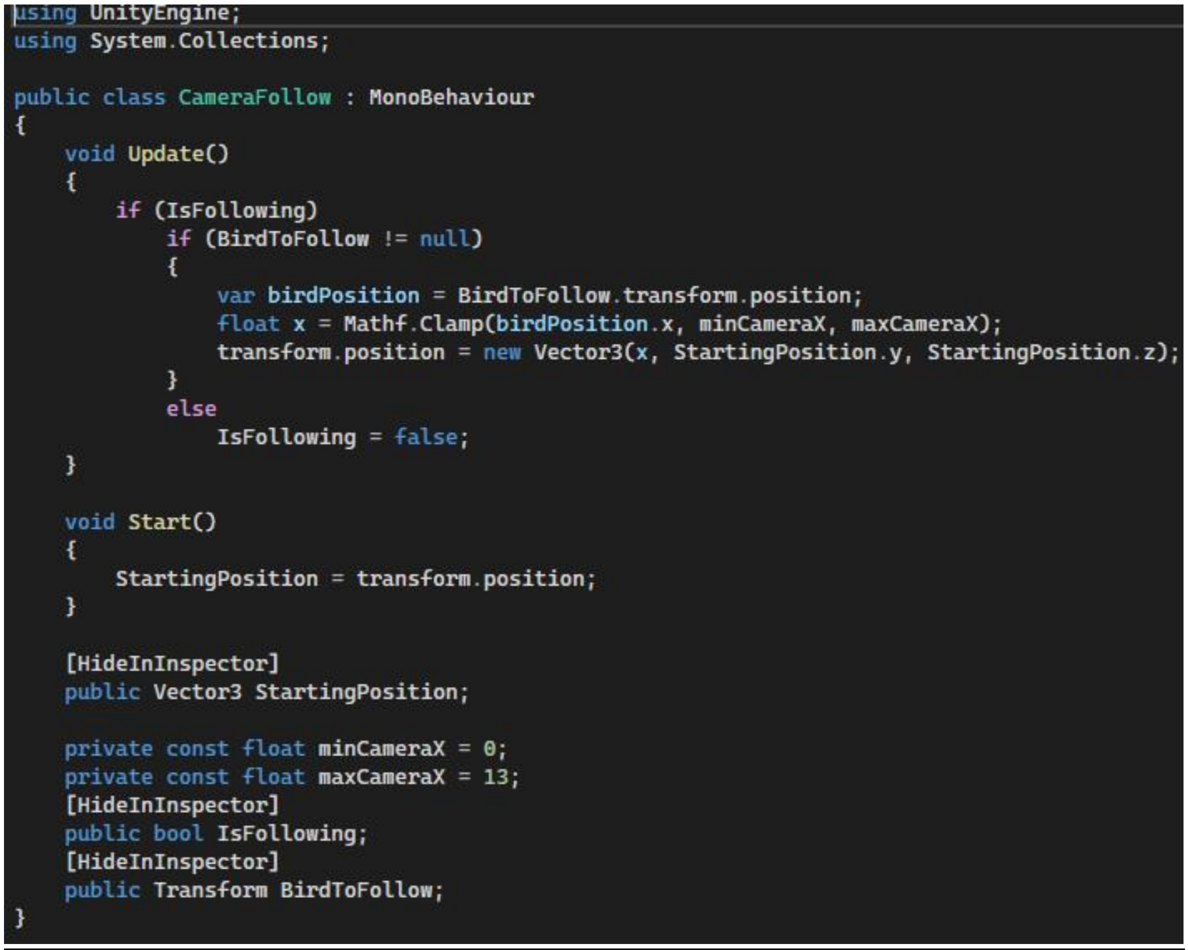


```
Script de Unity (1 referencia de recurso) | 0 referencias
public class Brick : MonoBehaviour
{
    [Header("Structure config file")]
    [SerializeField] private StructureScriptableObject structureSO;

    private float Health = 70f;
    private float receivedDamage = 0f;
    private float explotionDamage = 150f;
    private float damage = 0f;
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        if (structureSO != null)
        {
            Health = 70f;
        }
        else
        {
            Health = structureSO.lifePoints;
        }
        explotionDamage = 150f;
    }
```



```
Mensaje de Unity | 0 referencias
void OnCollisionEnter2D(Collision2D col)
{
    GameObject collisioner = col.gameObject;
    if (collisioner.GetComponent<Rigidbody2D>() == null) return;
    damage = collisioner.GetComponent<Rigidbody2D>().velocity.magnitude * 10;
    receivedDamage = damage;
    if (collisioner.CompareTag("Explotion"))
    {
        receivedDamage = explotionDamage;
    }
    CheckHealth(receivedDamage);
}

1 referencia
private void CheckHealth(float damage)
{
    if (damage >= 10)
        GetComponent<AudioSource>().Play();
    Health -= damage;
    if (Health <= 0) Destroy(this.gameObject);
}
```

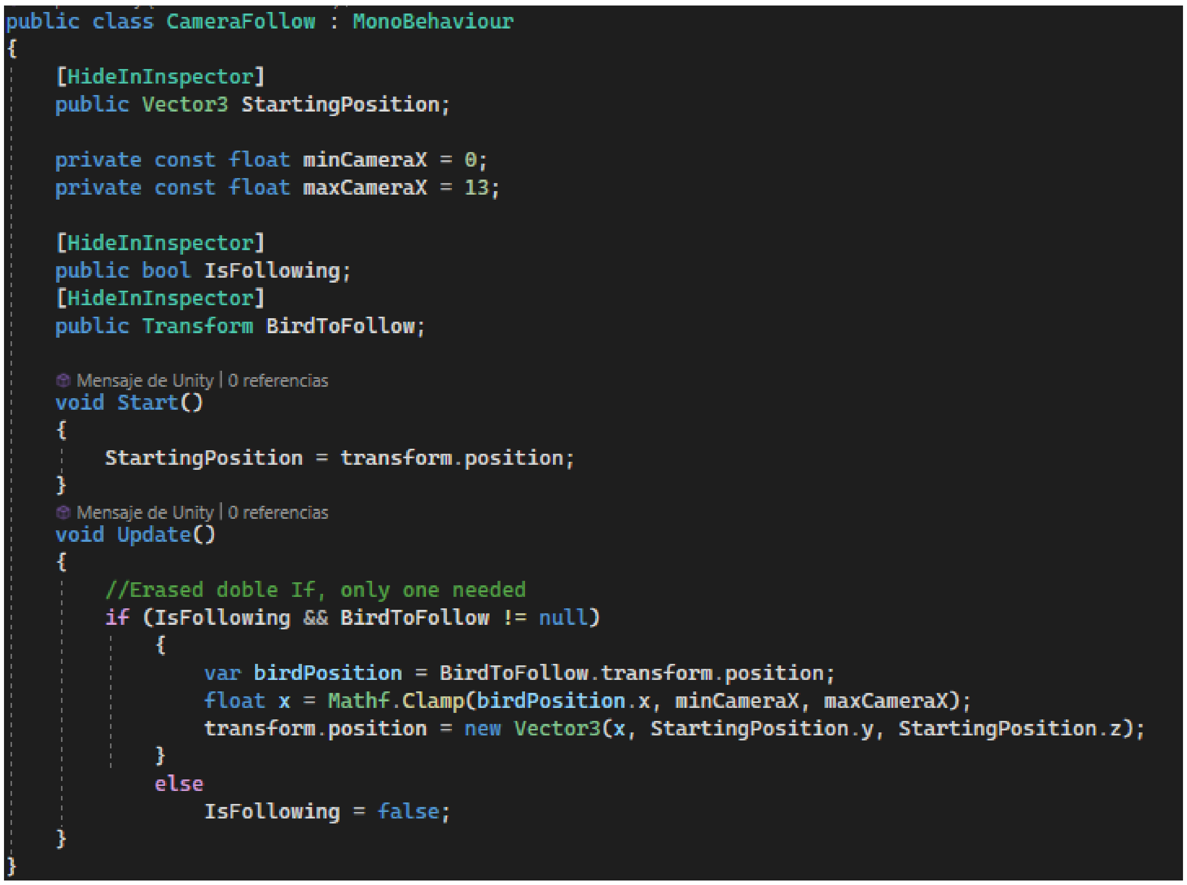
CameraFollow.cs Monobehaviur OLD



```
using UnityEngine;
using System.Collections;

public class CameraFollow : MonoBehaviour
{
    void Update()
    {
        if (IsFollowing)
            if (BirdToFollow != null)
            {
                var birdPosition = BirdToFollow.transform.position;
                float x = Mathf.Clamp(birdPosition.x, minCameraX, maxCameraX);
                transform.position = new Vector3(x, StartingPosition.y, StartingPosition.z);
            }
            else
                IsFollowing = false;
    }

    void Start()
    {
        StartingPosition = transform.position;
    }

    [HideInInspector]
    public Vector3 StartingPosition;

    private const float minCameraX = 0;
    private const float maxCameraX = 13;
    [HideInInspector]
    public bool IsFollowing;
    [HideInInspector]
    public Transform BirdToFollow;
}
```

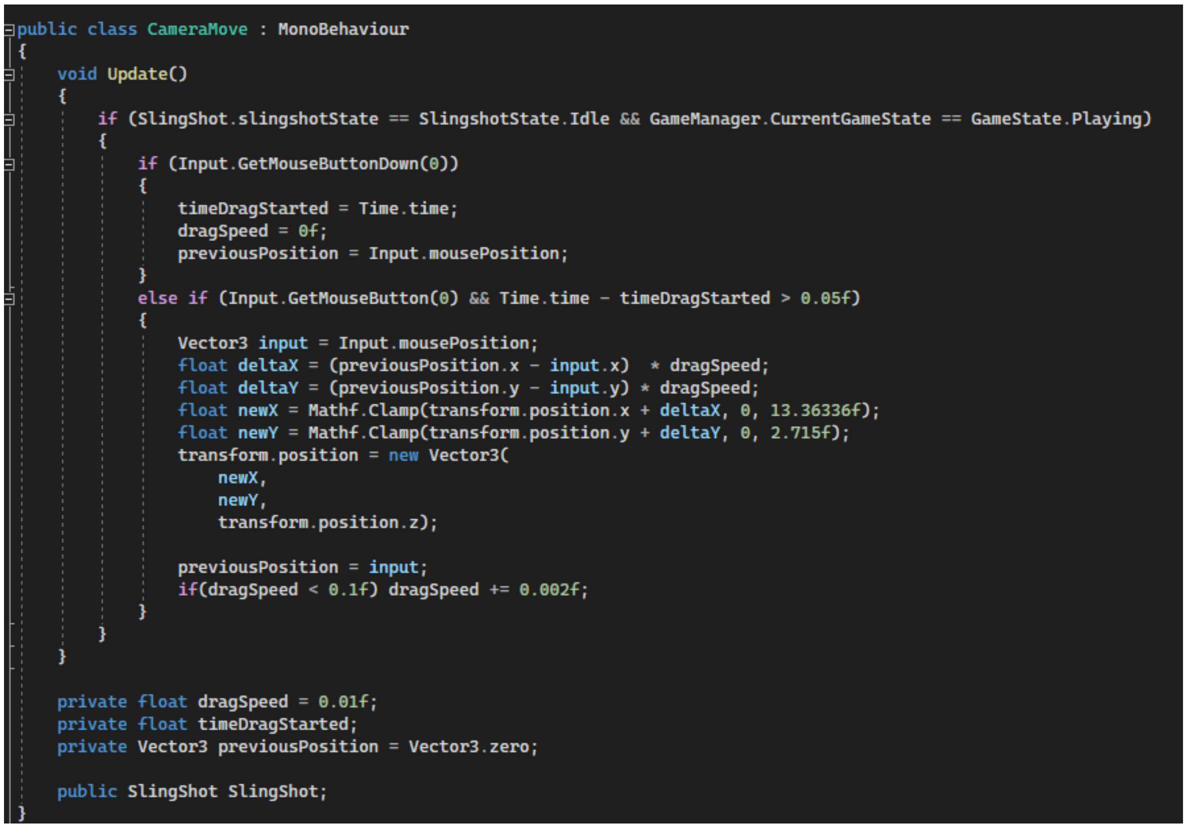
- Cambio de orden de variables por legibilidad
- Cambio de orden de Start y Update por legibilidad

CameraFollow.cs Monobehaviur NEW



```
public class CameraFollow : MonoBehaviour
{
    [HideInInspector]
    public Vector3 StartingPosition;

    private const float minCameraX = 0;
    private const float maxCameraX = 13;

    [HideInInspector]
    public bool IsFollowing;
    [HideInInspector]
    public Transform BirdToFollow;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        StartingPosition = transform.position;
    }
    Mensaje de Unity | 0 referencias
    void Update()
    {
        //Erased doble If, only one needed
        if (IsFollowing && BirdToFollow != null)
            {
                var birdPosition = BirdToFollow.transform.position;
                float x = Mathf.Clamp(birdPosition.x, minCameraX, maxCameraX);
                transform.position = new Vector3(x, StartingPosition.y, StartingPosition.z);
            }
            else
                IsFollowing = false;
    }
}
```

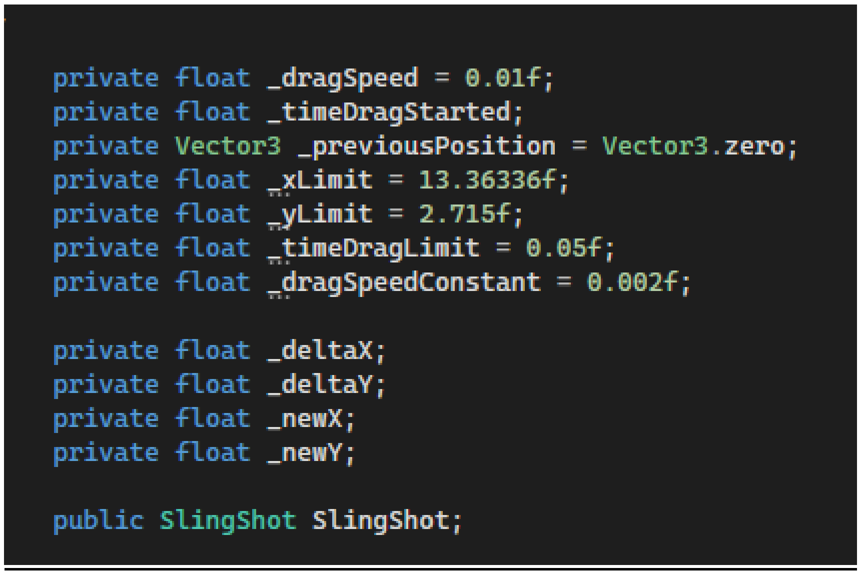
CameraMove.cs Monobehaviur OLD



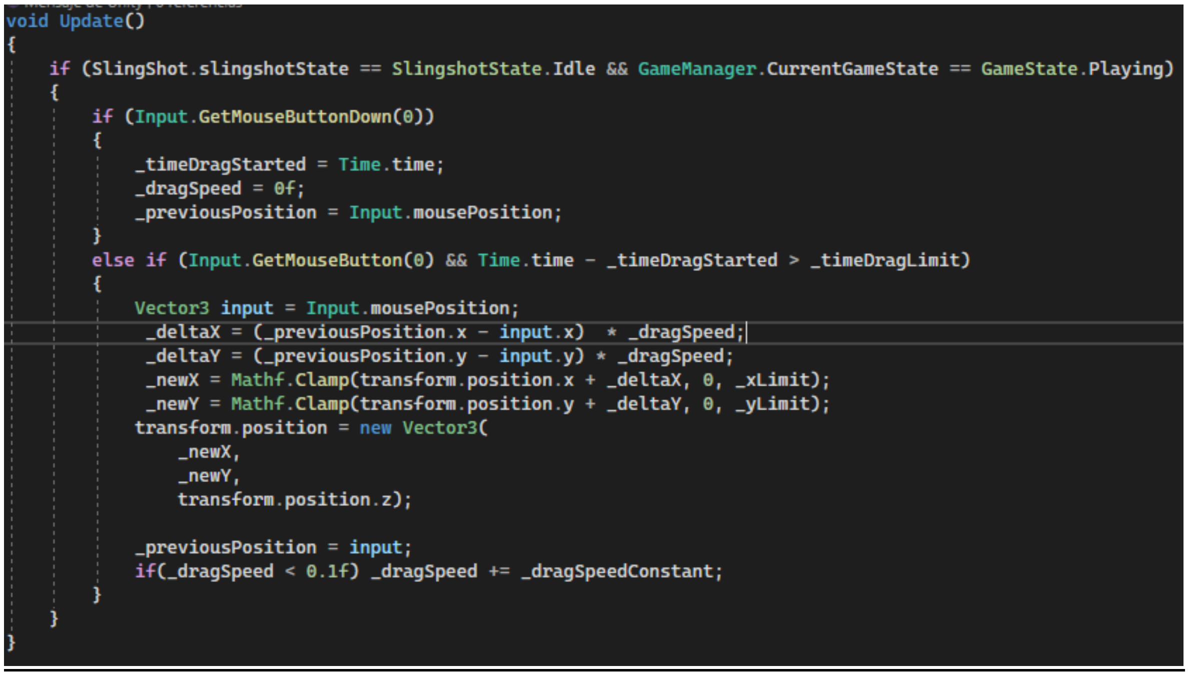
```
public class CameraMove : MonoBehaviour
{
    void Update()
    {
        if (SlingShot.slingshotState == SlingshotState.Idle && GameManager.CurrentGameState == GameState.Playing)
        {
            if (Input.GetMouseButtonDown(0))
            {
                timeDragStarted = Time.time;
                dragSpeed = 0f;
                previousPosition = Input.mousePosition;
            }
            else if (Input.GetMouseButton(0) && Time.time - timeDragStarted > 0.05f)
            {
                Vector3 input = Input.mousePosition;
                float deltaX = (previousPosition.x - input.x)  * dragSpeed;
                float deltaY = (previousPosition.y - input.y) * dragSpeed;
                float newX = Mathf.Clamp(transform.position.x + deltaX, 0, 13.36336f);
                float newY = Mathf.Clamp(transform.position.y + deltaY, 0, 2.715f);
                transform.position = new Vector3(
                    newX,
                    newY,
                    transform.position.z);

                previousPosition = input;
                if(dragSpeed < 0.1f) dragSpeed += 0.002f;
            }
        }
    }

    private float dragSpeed = 0.01f;
    private float timeDragStarted;
    private Vector3 previousPosition = Vector3.zero;

    public SlingShot SlingShot;
}
```

- Cambio de números mágicos por variables declaradas
- Cambio de variables creadas en cada instancia del update por variables creadas en declaración.

CameraMove.cs Monobehaviur NEW

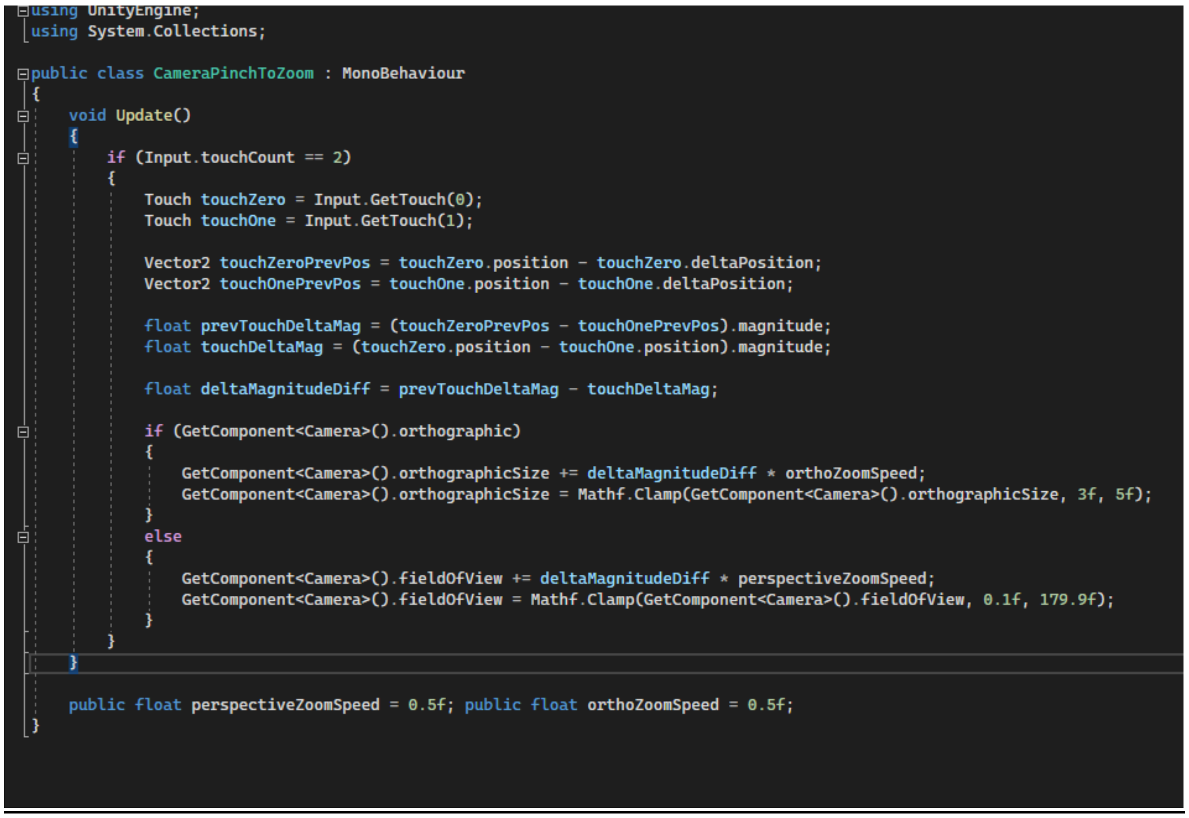


```
private float _dragSpeed = 0.01f;
private float _timeDragStarted;
private Vector3 _previousPosition = Vector3.zero;
private float _xLimit = 13.36336f;
private float _yLimit = 2.715f;
private float _timeDragLimit = 0.05f;
private float _dragSpeedConstant = 0.002f;

private float _deltaX;
private float _deltaY;
private float _newX;
private float _newY;

public SlingShot SlingShot;
```



```
void Update()
{
    if (SlingShot.slingshotState == SlingshotState.Idle && GameManager.CurrentGameState == GameState.Playing)
    {
        if (Input.GetMouseButtonDown(0))
        {
            _timeDragStarted = Time.time;
            _dragSpeed = 0f;
            _previousPosition = Input.mousePosition;
        }
        else if (Input.GetMouseButton(0) && Time.time - _timeDragStarted > _timeDragLimit)
        {
            Vector3 input = Input.mousePosition;
             _deltaX = (_previousPosition.x - input.x)  * _dragSpeed;
             _deltaY = (_previousPosition.y - input.y) * _dragSpeed;
             _newX = Mathf.Clamp(transform.position.x + _deltaX, 0, _xLimit);
             _newY = Mathf.Clamp(transform.position.y + _deltaY, 0, _yLimit);
            transform.position = new Vector3(
                _newX,
                _newY,
                transform.position.z);

            _previousPosition = input;
            if(_dragSpeed < 0.1f) _dragSpeed += _dragSpeedConstant;
        }
    }
}
```

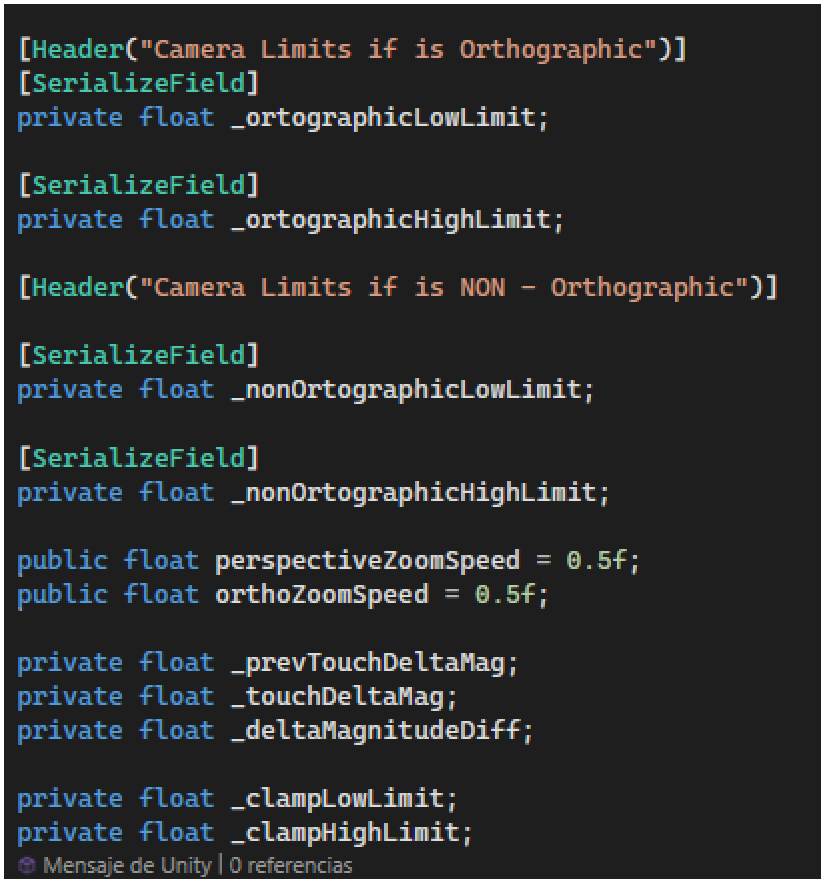
CameraPinchToZoom.cs Monobehaviur OLD



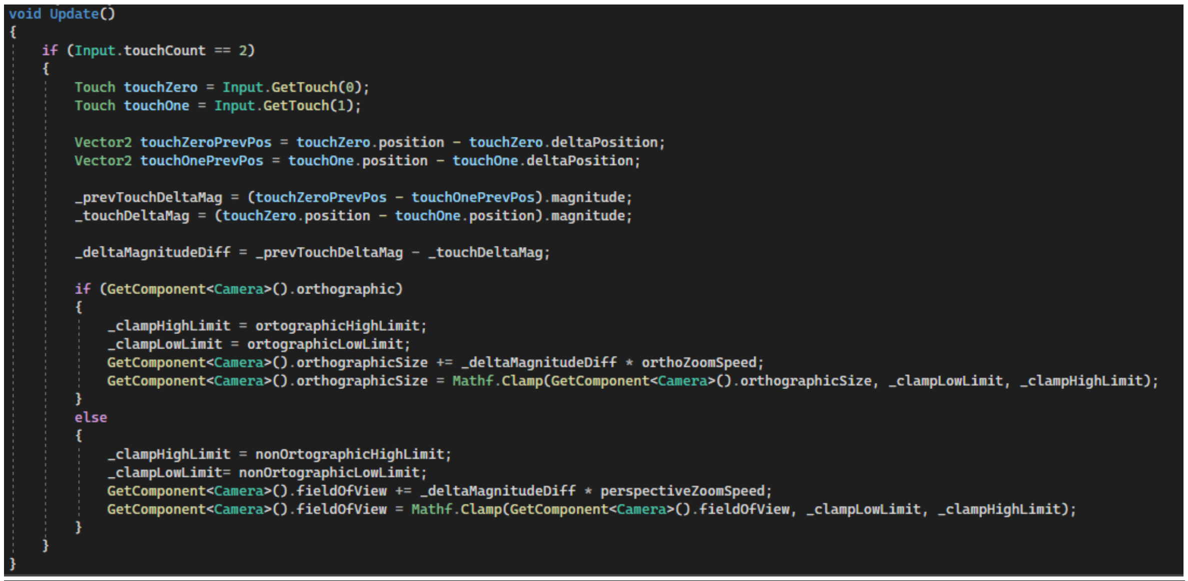
```
using UnityEngine;
using System.Collections;

public class CameraPinchToZoom : MonoBehaviour
{
    void Update()
    {
        if (Input.touchCount == 2)
        {
            Touch touchZero = Input.GetTouch(0);
            Touch touchOne = Input.GetTouch(1);

            Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

            float prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).magnitude;
            float touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

            float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;

            if (GetComponent<Camera>().orthographic)
            {
                GetComponent<Camera>().orthographicSize += deltaMagnitudeDiff * orthoZoomSpeed;
                GetComponent<Camera>().orthographicSize = Mathf.Clamp(GetComponent<Camera>().orthographicSize, 3f, 5f);
            }
            else
            {
                GetComponent<Camera>().fieldOfView += deltaMagnitudeDiff * perspectiveZoomSpeed;
                GetComponent<Camera>().fieldOfView = Mathf.Clamp(GetComponent<Camera>().fieldOfView, 0.1f, 179.9f);
            }
        }
    }

    public float perspectiveZoomSpeed = 0.5f; public float orthoZoomSpeed = 0.5f;
}
```

- Creación de variables para reemplazo de números mágicos
- Re-ordenamiento de declaración y Update

- *CameraPinchToZoom.cs Monobehaviur NEW*

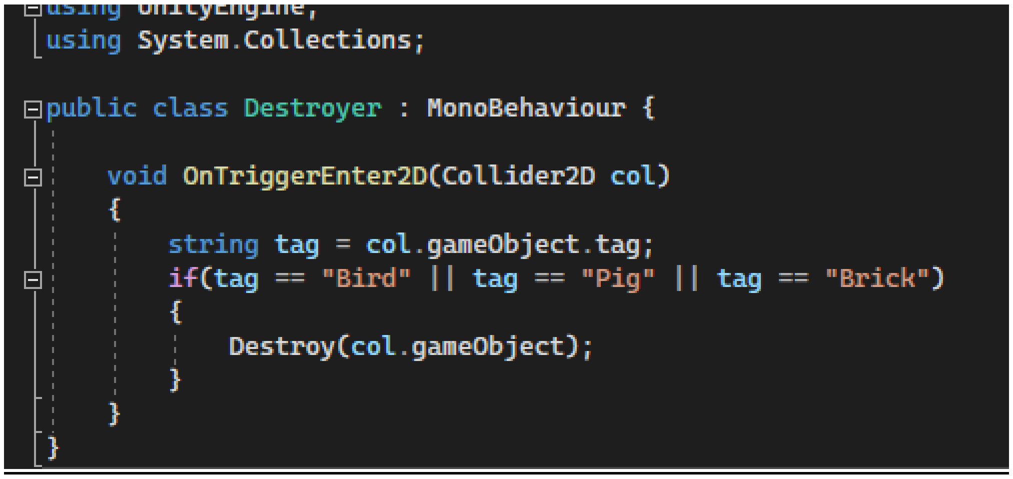


```
[Header("Camera Limits if is Orthographic")]
[SerializeField]
private float _ortographicLowLimit;

[SerializeField]
private float _ortographicHighLimit;

[Header("Camera Limits if is NON - Orthographic")]

[SerializeField]
private float _nonOrtographicLowLimit;

[SerializeField]
private float _nonOrtographicHighLimit;

public float perspectiveZoomSpeed = 0.5f;
public float orthoZoomSpeed = 0.5f;

private float _prevTouchDeltaMag;
private float _touchDeltaMag;
private float _deltaMagnitudeDiff;

private float _clampLowLimit;
private float _clampHighLimit;
Mensaje de Unity | 0 referencias
```



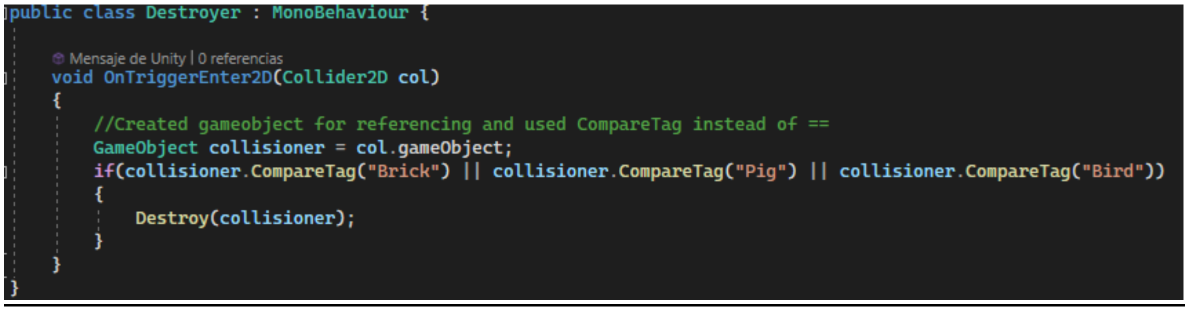
```
void Update()
{
    if (Input.touchCount == 2)
    {
        Touch touchZero = Input.GetTouch(0);
        Touch touchOne = Input.GetTouch(1);

        Vector2 touchZeroPrevPos = touchZero.position - touchZero.deltaPosition;
        Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

        _prevTouchDeltaMag = (touchZeroPrevPos - touchOnePrevPos).magnitude;
        _touchDeltaMag = (touchZero.position - touchOne.position).magnitude;

        _deltaMagnitudeDiff = _prevTouchDeltaMag - _touchDeltaMag;

        if (GetComponent<Camera>().orthographic)
        {
            _clampHighLimit = ortographicHighLimit;
            _clampLowLimit = ortographicLowLimit;
            GetComponent<Camera>().orthographicSize += _deltaMagnitudeDiff * orthoZoomSpeed;
            GetComponent<Camera>().orthographicSize = Mathf.Clamp(GetComponent<Camera>().orthographicSize, _clampLowLimit, _clampHighLimit);
        }
        else
        {
            _clampHighLimit = nonOrtographicHighLimit;
            _clampLowLimit= nonOrtographicLowLimit;
            GetComponent<Camera>().fieldOfView += _deltaMagnitudeDiff * perspectiveZoomSpeed;
            GetComponent<Camera>().fieldOfView = Mathf.Clamp(GetComponent<Camera>().fieldOfView, _clampLowLimit, _clampHighLimit);
        }
    }
}
```

Destroyer.cs Monobehaviur OLD



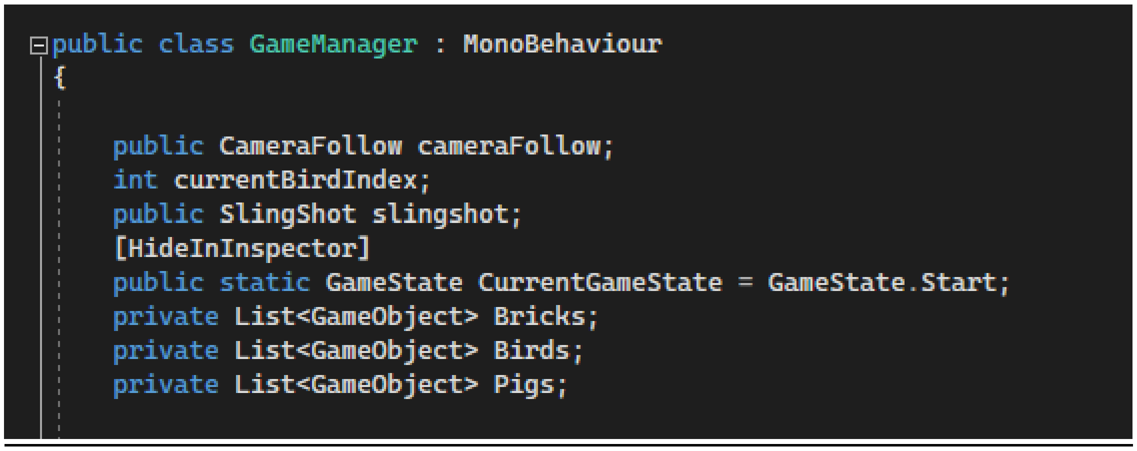
```
using UnityEngine;
using System.Collections;

public class Destroyer : MonoBehaviour {

    void OnTriggerEnter2D(Collider2D col)
    {
        string tag = col.gameObject.tag;
        if(tag == "Bird" || tag == "Pig" || tag == "Brick")
        {
            Destroy(col.gameObject);
        }
    }
}
```

- Creación de Gameobject de referencia
- Reemplazo de == por CompareTag

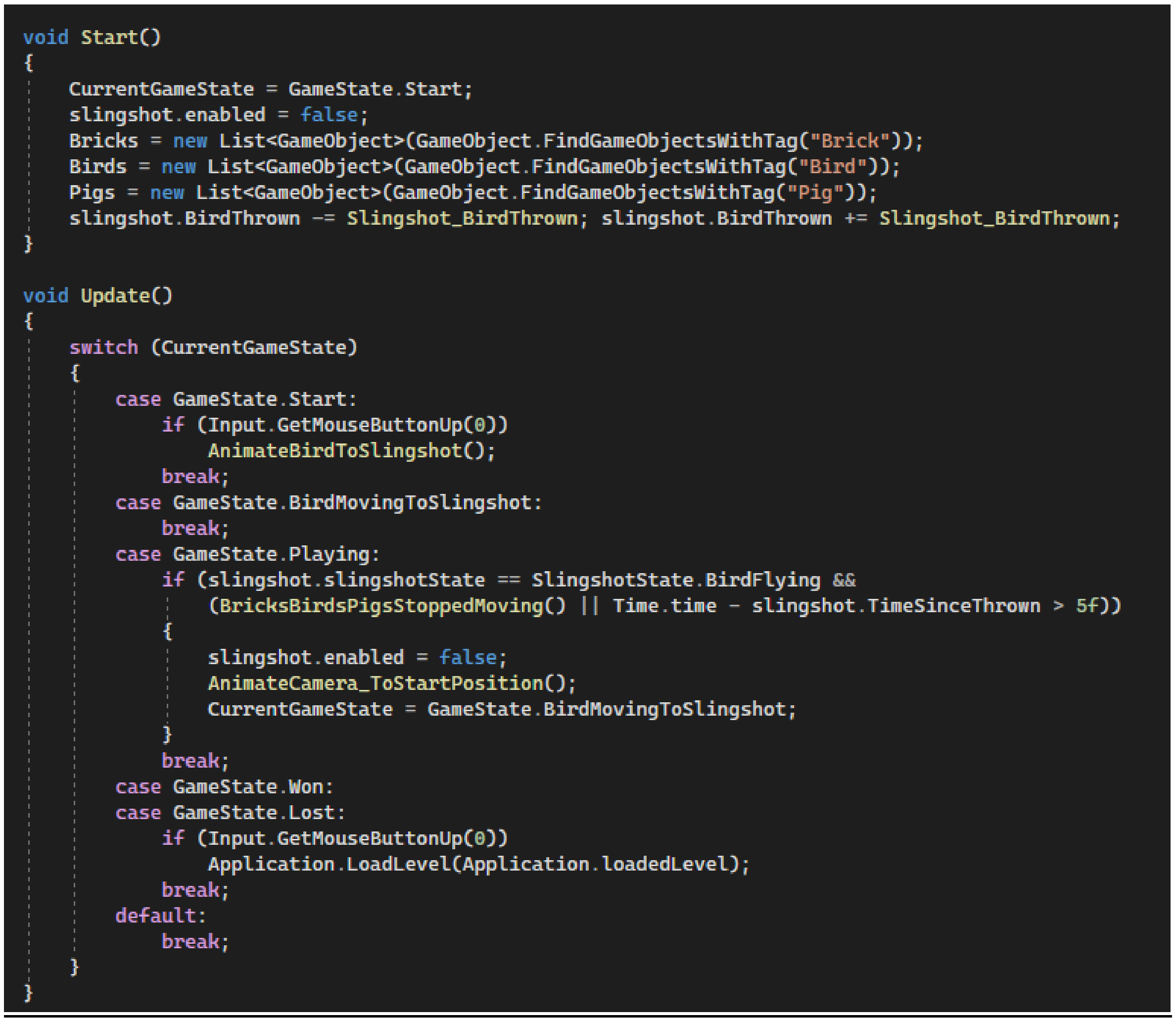


```
public class Destroyer : MonoBehaviour {

    Mensaje de Unity | 0 referencias
    void OnTriggerEnter2D(Collider2D col)
    {
        //Created gameobject for referencing and used CompareTag instead of ==
        GameObject collisioner = col.gameObject;
        if(collisioner.CompareTag("Brick") || collisioner.CompareTag("Pig") || collisioner.CompareTag("Bird"))
        {
            Destroy(collisioner);
        }
    }
}
```

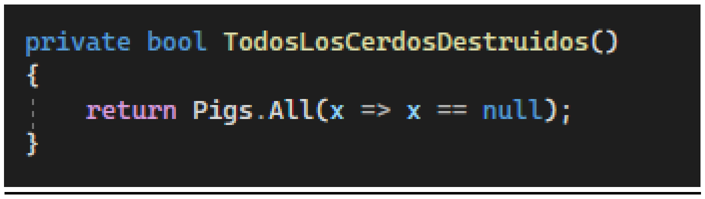
GameManager.cs Monobehaviour OLD



```
public class GameManager : MonoBehaviour
{

    public CameraFollow cameraFollow;
    int currentBirdIndex;
    public SlingShot slingshot;
    [HideInInspector]
    public static GameState CurrentGameState = GameState.Start;
    private List<GameObject> Bricks;
    private List<GameObject> Birds;
    private List<GameObject> Pigs;
```



```
void Start()
{
    CurrentGameState = GameState.Start;
    slingshot.enabled = false;
    Bricks = new List<GameObject>(GameObject.FindGameObjectsWithTag("Brick"));
    Birds = new List<GameObject>(GameObject.FindGameObjectsWithTag("Bird"));
    Pigs = new List<GameObject>(GameObject.FindGameObjectsWithTag("Pig"));
    slingshot.BirdThrown -= Slingshot_BirdThrown; slingshot.BirdThrown += Slingshot_BirdThrown;
}

void Update()
{
    switch (CurrentGameState)
    {
        case GameState.Start:
            if (Input.GetMouseButtonUp(0))
                AnimateBirdToSlingshot();
            break;
        case GameState.BirdMovingToSlingshot:
            break;
        case GameState.Playing:
            if (slingshot.slingshotState == SlingshotState.BirdFlying &&
                (BricksBirdsPigsStoppedMoving() || Time.time - slingshot.TimeSinceThrown > 5f))
            {
                slingshot.enabled = false;
                AnimateCamera_ToStartPosition();
                CurrentGameState = GameState.BirdMovingToSlingshot;
            }
            break;
        case GameState.Won:
        case GameState.Lost:
            if (Input.GetMouseButtonUp(0))
                Application.LoadLevel(Application.loadedLevel);
            break;
        default:
            break;
    }
}
```

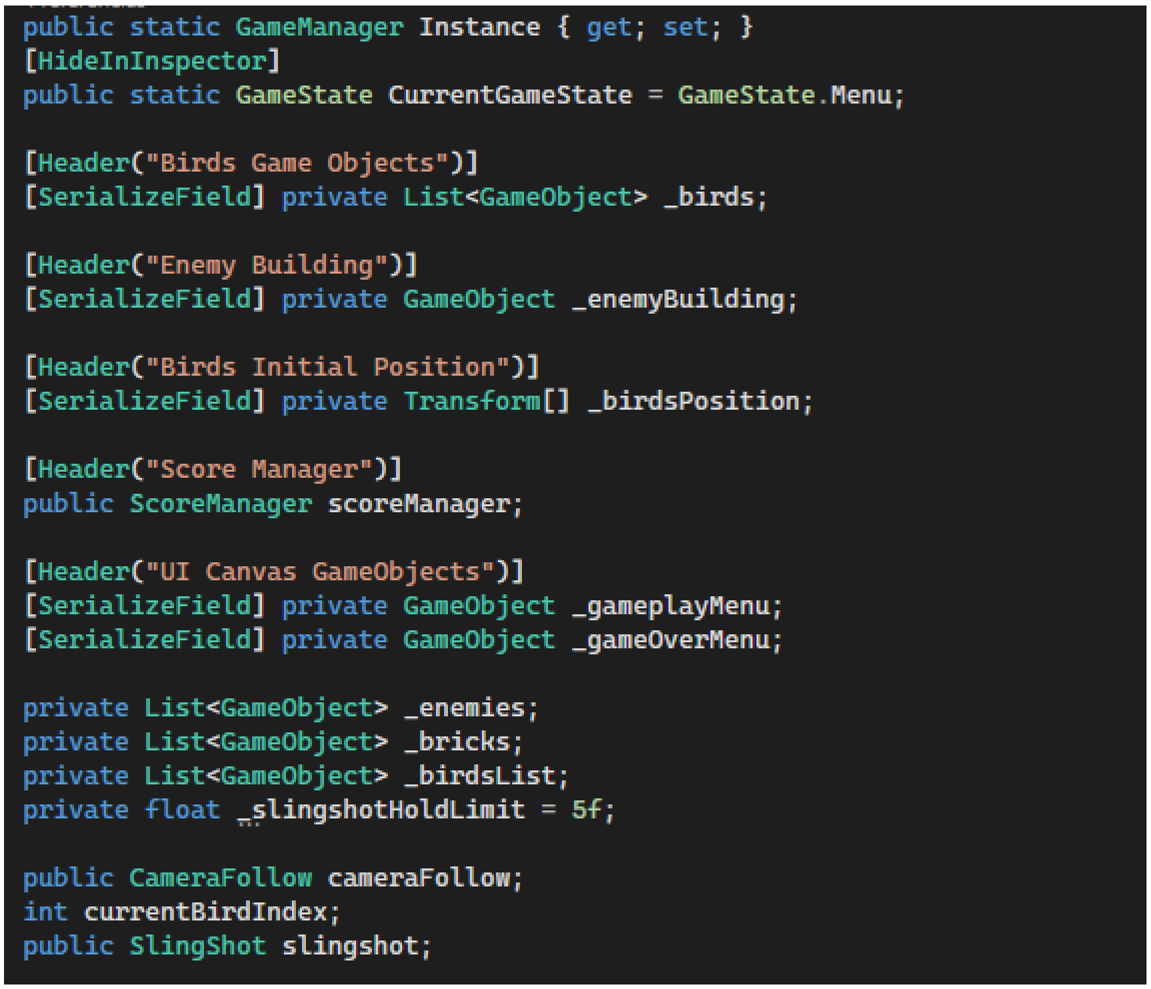


```
private bool TodosLosCerdosDestruidos()
{
    return Pigs.All(x => x == null);
}
```

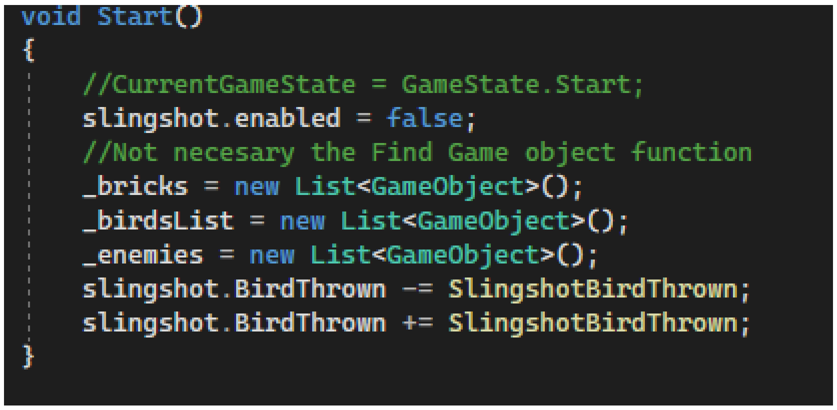
- Creación de referencias de gameobjects en Inspector para evitar uso de función ***FindGameObjectWithTag***
- Creación de Instancia de gameManager para Singleton
- Creación de variables para reemplazar números mágicos
- Reamplazo de nombre de funciones de español -> inglés
- Eliminación de guión bajo (_) en nombres de funciones
- Creación de función StartGame:
    - Referencia de ave seleccionada
    - Creación de lista de estructura (Bricks)

    - Creación de lista de enemigos (Pigs)
- Creación de función GameOver como Corutina para llamado de GameOver Gameobject en Canvas

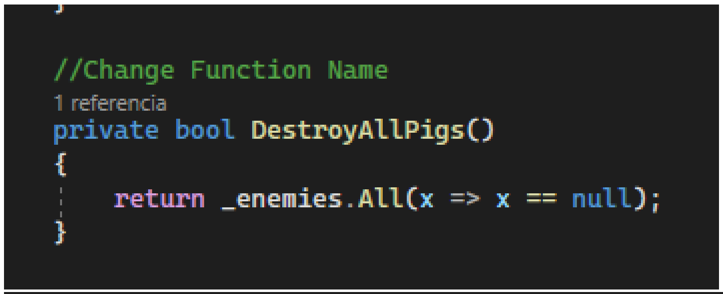
GameManager.cs Monobehaviour NEW



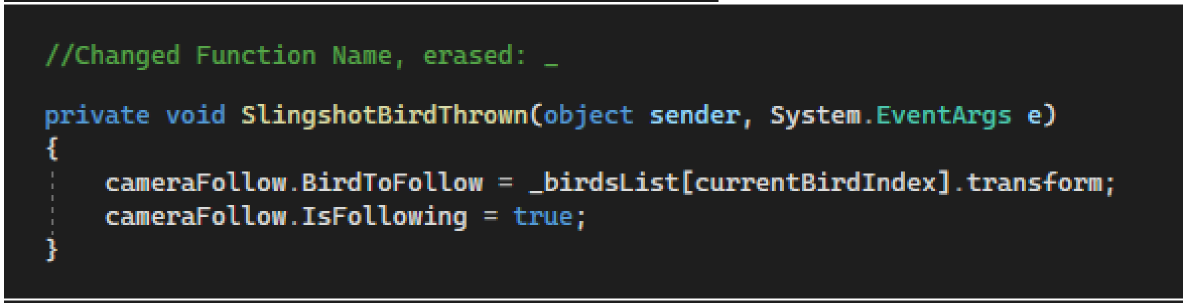
```
public static GameManager Instance { get; set; }
[HideInInspector]
public static GameState CurrentGameState = GameState.Menu;

[Header("Birds Game Objects")]
[SerializeField] private List<GameObject> _birds;

[Header("Enemy Building")]
[SerializeField] private GameObject _enemyBuilding;

[Header("Birds Initial Position")]
[SerializeField] private Transform[] _birdsPosition;

[Header("Score Manager")]
public ScoreManager scoreManager;

[Header("UI Canvas GameObjects")]
[SerializeField] private GameObject _gameplayMenu;
[SerializeField] private GameObject _gameOverMenu;

private List<GameObject> _enemies;
private List<GameObject> _bricks;
private List<GameObject> _birdsList;
private float _slingshotHoldLimit = 5f;

public CameraFollow cameraFollow;
int currentBirdIndex;
public SlingShot slingshot;
```



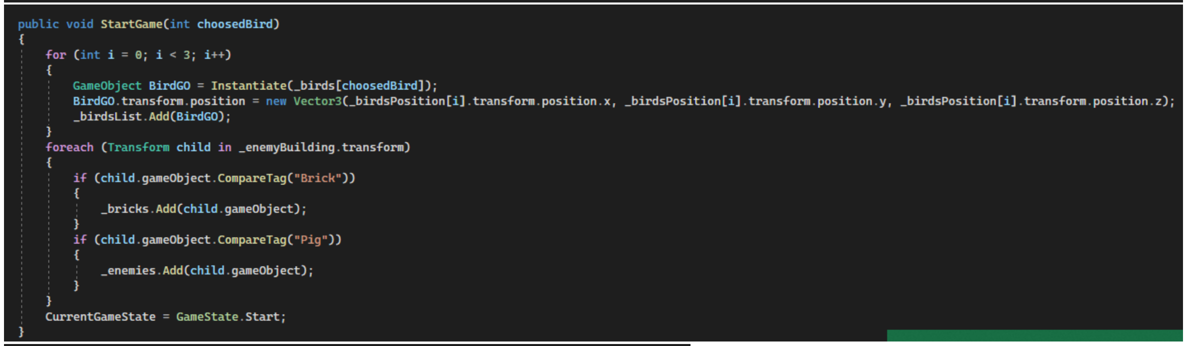
```
void Start()
{
    //CurrentGameState = GameState.Start;
    slingshot.enabled = false;
    //Not necesary the Find Game object function
    _bricks = new List<GameObject>();
    _birdsList = new List<GameObject>();
    _enemies = new List<GameObject>();
    slingshot.BirdThrown -= SlingshotBirdThrown;
    slingshot.BirdThrown += SlingshotBirdThrown;
}
```



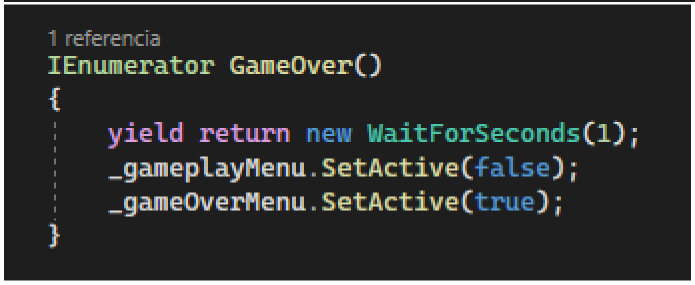
```
//Change Function Name
1 referencia
private bool DestroyAllPigs()
{
    return _enemies.All(x => x == null);
}
```



```
//Changed Function Name, erased: _

private void SlingshotBirdThrown(object sender, System.EventArgs e)
{
    cameraFollow.BirdToFollow = _birdsList[currentBirdIndex].transform;
    cameraFollow.IsFollowing = true;
}
```

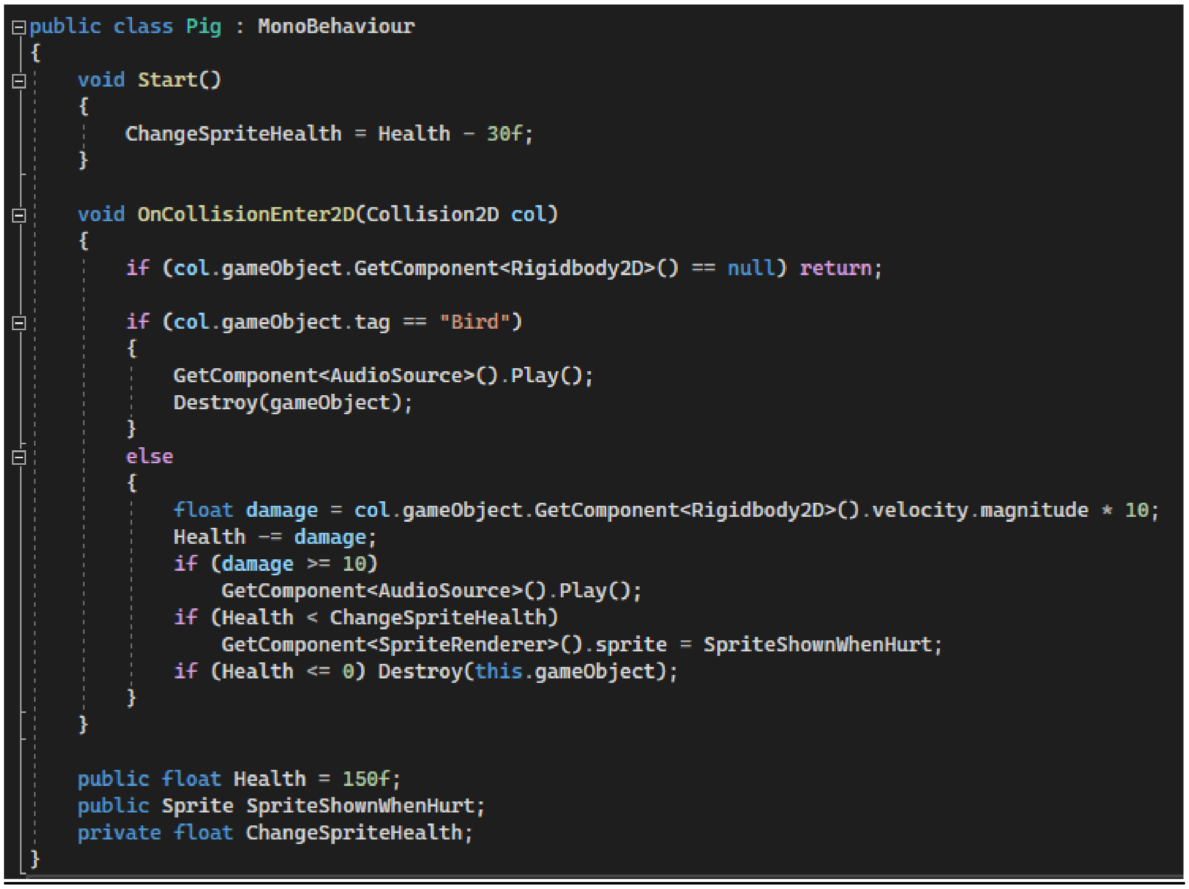


```
public void StartGame(int choosedBird)
{
    for (int i = 0; i < 3; i++)
    {
        GameObject BirdGO = Instantiate(_birds[choosedBird]);
        BirdGO.transform.position = new Vector3(_birdsPosition[i].transform.position.x, _birdsPosition[i].transform.position.y, _birdsPosition[i].transform.position.z);
        _birdsList.Add(BirdGO);
    }
    foreach (Transform child in _enemyBuilding.transform)
    {
        if (child.gameObject.CompareTag("Brick"))
        {
            _bricks.Add(child.gameObject);
        }
        if (child.gameObject.CompareTag("Pig"))
        {
            _enemies.Add(child.gameObject);
        }
    }
    CurrentGameState = GameState.Start;
}
```



```
1 referencia
IEnumerator GameOver()
{
    yield return new WaitForSeconds(1);
    _gameplayMenu.SetActive(false);
    _gameOverMenu.SetActive(true);
}
```

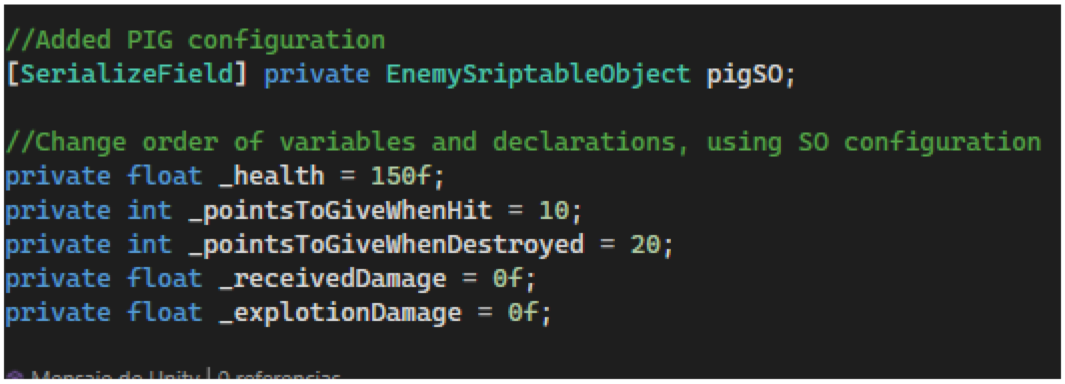
Pig.cs Monobehaviour OLD



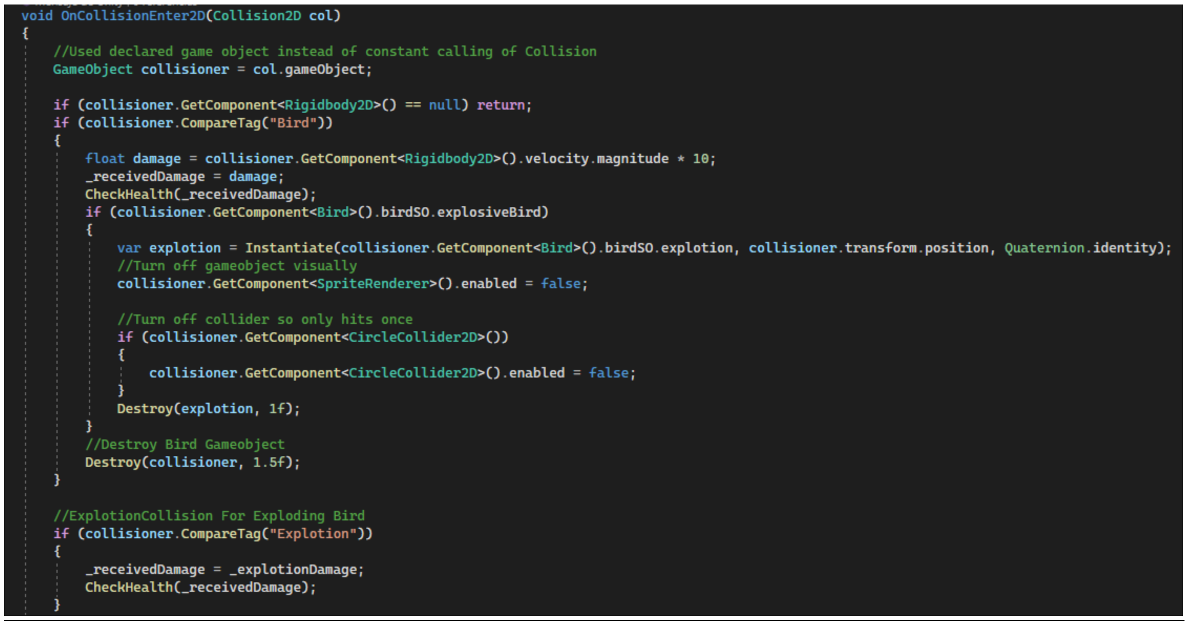
```
public class Pig : MonoBehaviour
{
    void Start()
    {
        ChangeSpriteHealth = Health - 30f;
    }

    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.GetComponent<Rigidbody2D>() == null) return;

        if (col.gameObject.tag == "Bird")
        {
            GetComponent<AudioSource>().Play();
            Destroy(gameObject);
        }
        else
        {
            float damage = col.gameObject.GetComponent<Rigidbody2D>().velocity.magnitude * 10;
            Health -= damage;
            if (damage >= 10)
                GetComponent<AudioSource>().Play();
            if (Health < ChangeSpriteHealth)
                GetComponent<SpriteRenderer>().sprite = SpriteShownWhenHurt;
            if (Health <= 0) Destroy(this.gameObject);
        }
    }

    public float Health = 150f;
    public Sprite SpriteShownWhenHurt;
    private float ChangeSpriteHealth;
}
```

- Se añadió configuración de enemigo tipo PIG como ScriptableObject
- Cambio de declaración de variables y funciones
- Se añadió función para llamar explosión al momento de colisionar con ave de tipo EXPLOSIVE BIRD
- Separación de función de chequeo de puntos de vida del enemigo
- Se creó función para añadir puntos al contacto con un enemigo

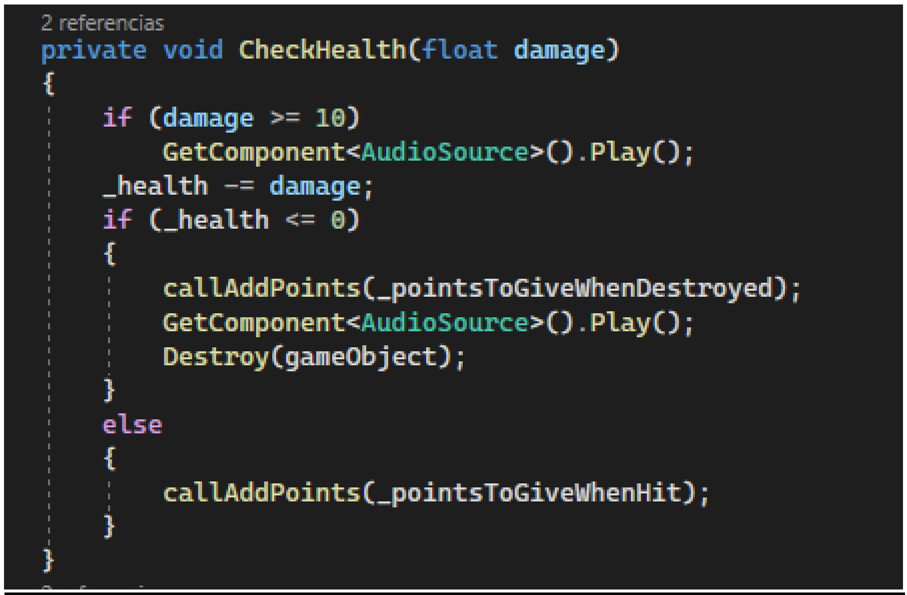
Pig.cs Monobehaviour NEW



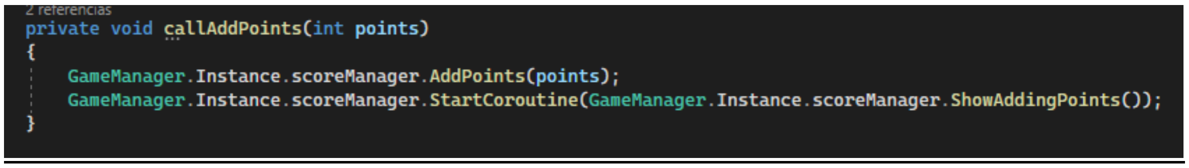
```
//Added PIG configuration
[SerializeField] private EnemySriptableObject pigSO;

//Change order of variables and declarations, using SO configuration
private float _health = 150f;
private int _pointsToGiveWhenHit = 10;
private int _pointsToGiveWhenDestroyed = 20;
private float _receivedDamage = 0f;
private float _explotionDamage = 0f;
```



```
void OnCollisionEnter2D(Collision2D col)
{
    //Used declared game object instead of constant calling of Collision
    GameObject collisioner = col.gameObject;

    if (collisioner.GetComponent<Rigidbody2D>() == null) return;
    if (collisioner.CompareTag("Bird"))
    {
        float damage = collisioner.GetComponent<Rigidbody2D>().velocity.magnitude * 10;
        _receivedDamage = damage;
        CheckHealth(_receivedDamage);
        if (collisioner.GetComponent<Bird>().birdSO.explosiveBird)
        {
            var explotion = Instantiate(collisioner.GetComponent<Bird>().birdSO.explotion, collisioner.transform.position, Quaternion.identity);
            //Turn off gameobject visually
            collisioner.GetComponent<SpriteRenderer>().enabled = false;

            //Turn off collider so only hits once
            if (collisioner.GetComponent<CircleCollider2D>())
            {
                collisioner.GetComponent<CircleCollider2D>().enabled = false;
            }
            Destroy(explotion, 1f);
        }
        //Destroy Bird Gameobject
        Destroy(collisioner, 1.5f);
    }

    //ExplotionCollision For Exploding Bird
    if (collisioner.CompareTag("Explotion"))
    {
        _receivedDamage = _explotionDamage;
        CheckHealth(_receivedDamage);
    }
```

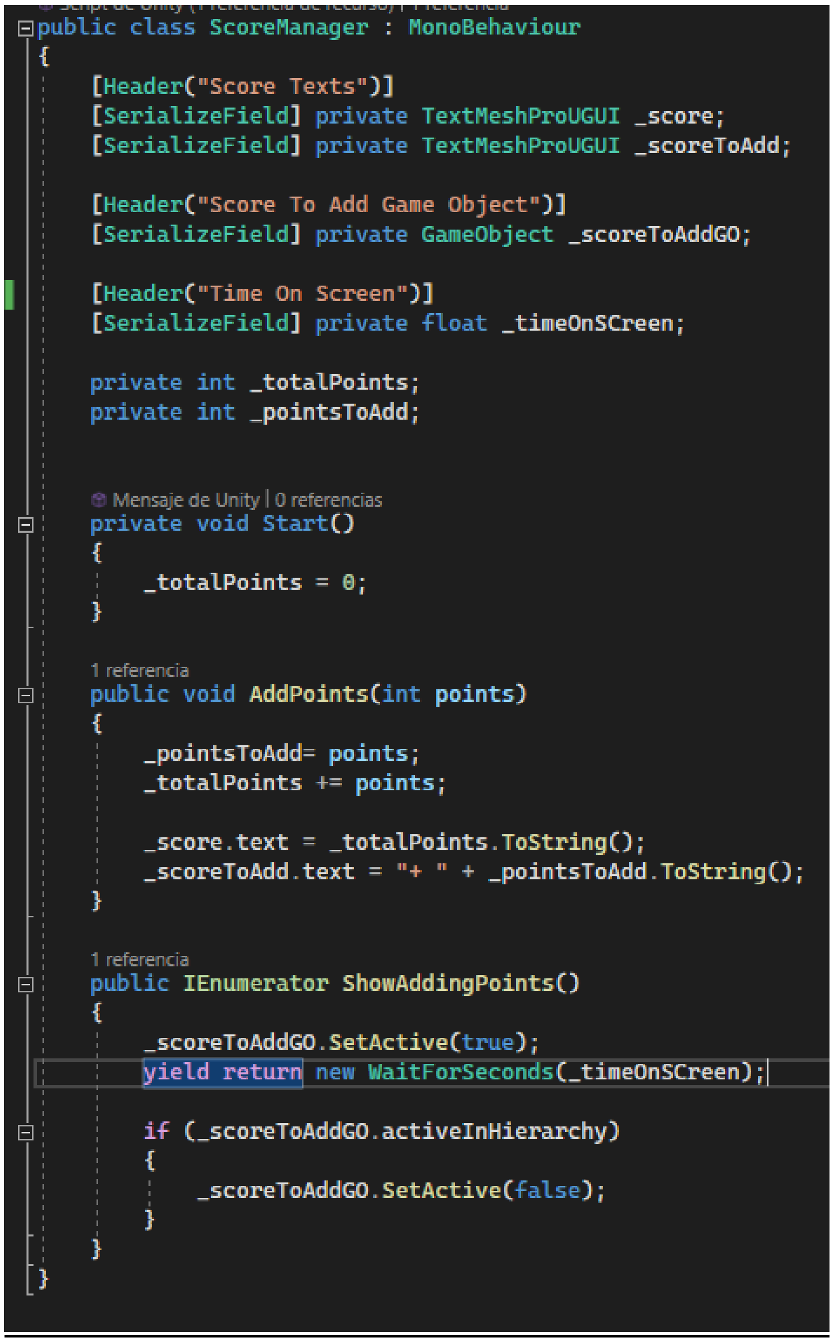


```
2 referencias
private void CheckHealth(float damage)
{
    if (damage >= 10)
        GetComponent<AudioSource>().Play();
    _health -= damage;
    if (_health <= 0)
    {
        callAddPoints(_pointsToGiveWhenDestroyed);
        GetComponent<AudioSource>().Play();
        Destroy(gameObject);
    }
    else
    {
        callAddPoints(_pointsToGiveWhenHit);
    }
}
```



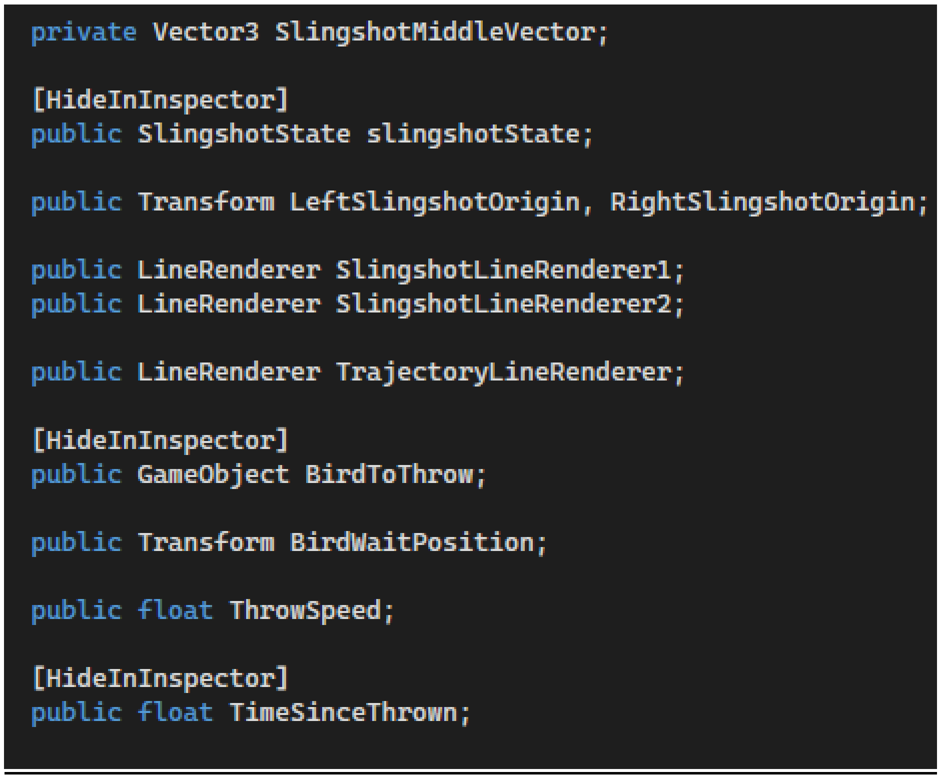
```
2 referencias
private void callAddPoints(int points)
{
    GameManager.Instance.scoreManager.AddPoints(points);
    GameManager.Instance.scoreManager.StartCoroutine(GameManager.Instance.scoreManager.ShowAddingPoints());
}
```

CREATION: SCOREMANAGER.CS MONOBEHAVIOUR

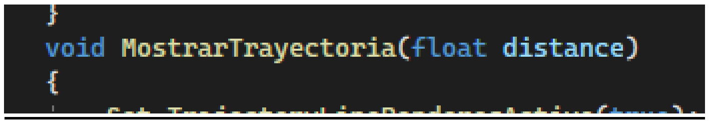


```
public class ScoreManager : MonoBehaviour
{
    [Header("Score Texts")]
    [SerializeField] private TextMeshProUGUI _score;
    [SerializeField] private TextMeshProUGUI _scoreToAdd;

    [Header("Score To Add Game Object")]
    [SerializeField] private GameObject _scoreToAddGO;

    [Header("Time On Screen")]
    [SerializeField] private float _timeOnSCreen;

    private int _totalPoints;
    private int _pointsToAdd;


    Mensaje de Unity | 0 referencias
    private void Start()
    {
        _totalPoints = 0;
    }

    1 referencia
    public void AddPoints(int points)
    {
        _pointsToAdd= points;
        _totalPoints += points;

        _score.text = _totalPoints.ToString();
        _scoreToAdd.text = "+ " + _pointsToAdd.ToString();
    }

    1 referencia
    public IEnumerator ShowAddingPoints()
    {
        _scoreToAddGO.SetActive(true);
        yield return new WaitForSeconds(_timeOnSCreen);

        if (_scoreToAddGO.activeInHierarchy)
        {
            _scoreToAddGO.SetActive(false);
        }
    }
}
```

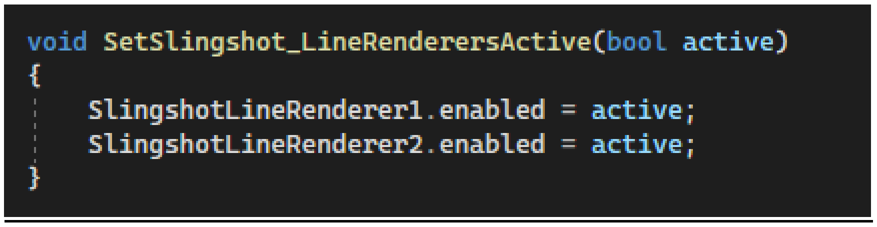
Slingshot.cs Monobehaviour OLD



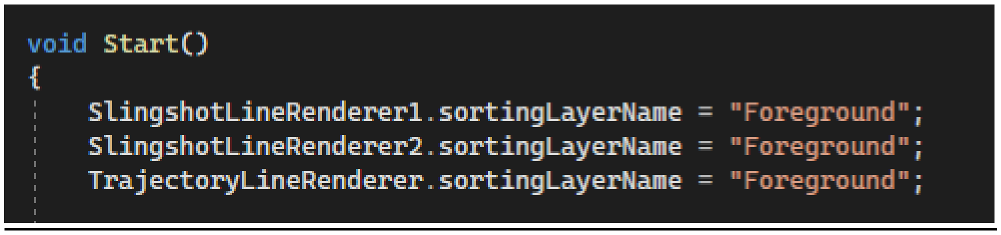
```
private Vector3 SlingshotMiddleVector;

[HideInInspector]
public SlingshotState slingshotState;

public Transform LeftSlingshotOrigin, RightSlingshotOrigin;

public LineRenderer SlingshotLineRenderer1;
public LineRenderer SlingshotLineRenderer2;

public LineRenderer TrajectoryLineRenderer;

[HideInInspector]
public GameObject BirdToThrow;

public Transform BirdWaitPosition;

public float ThrowSpeed;

[HideInInspector]
public float TimeSinceThrown;
```



```
 }
 void MostrarTrayectoria(float distance)
 {
```



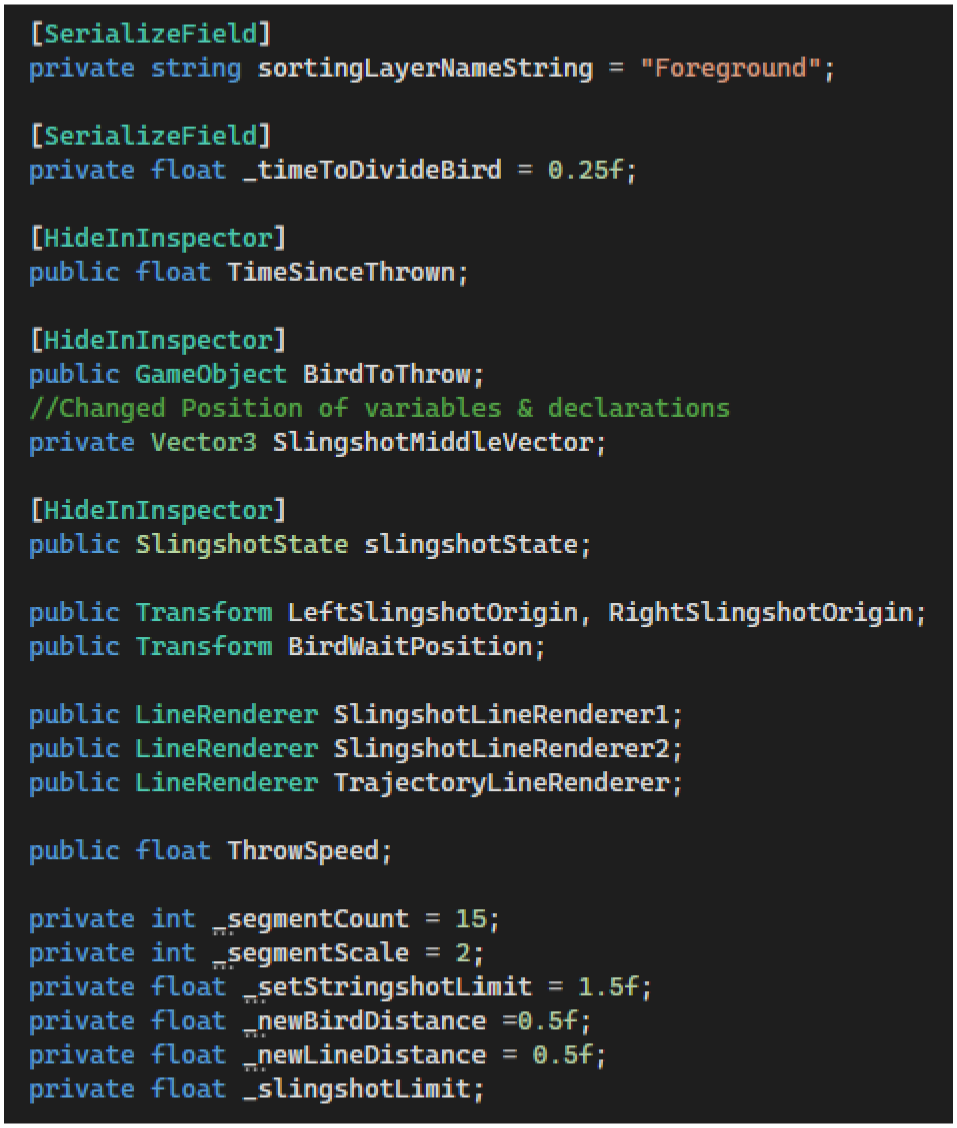
```
void SetSlingshot_LineRenderersActive(bool active)
{
    SlingshotLineRenderer1.enabled = active;
    SlingshotLineRenderer2.enabled = active;
}
```



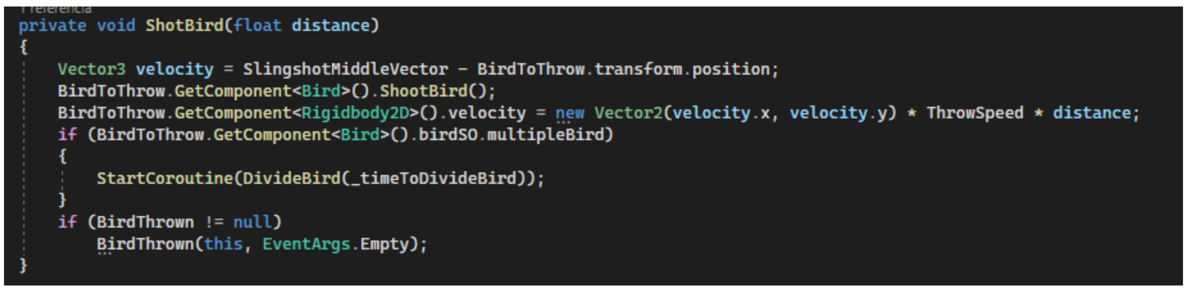
```
void Start()
{
    SlingshotLineRenderer1.sortingLayerName = "Foreground";
    SlingshotLineRenderer2.sortingLayerName = "Foreground";
    TrajectoryLineRenderer.sortingLayerName = "Foreground";
```

- Se crearon variables para evitar el uso de números y strings mágicos
- Se cambió el nombre de funciones español->inglés
- Se creo Corutina para división de ave

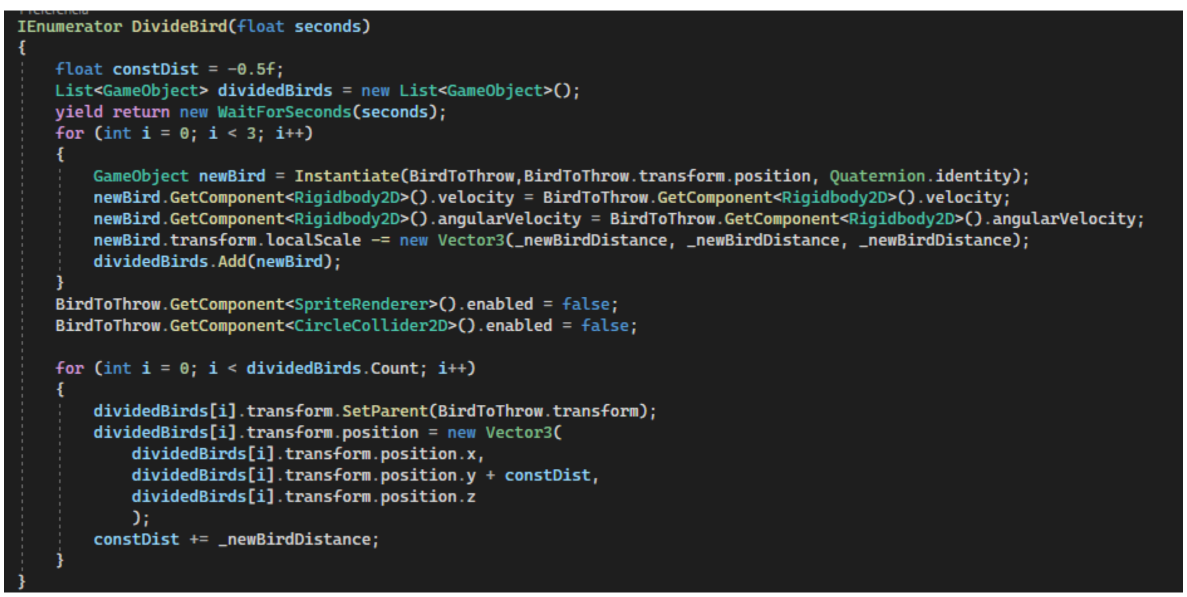
Slingshot.cs Monobehaviour NEW



```
[SerializeField]
private string sortingLayerNameString = "Foreground";

[SerializeField]
private float _timeToDivideBird = 0.25f;

[HideInInspector]
public float TimeSinceThrown;

[HideInInspector]
public GameObject BirdToThrow;
//Changed Position of variables & declarations
private Vector3 SlingshotMiddleVector;

[HideInInspector]
public SlingshotState slingshotState;

public Transform LeftSlingshotOrigin, RightSlingshotOrigin;
public Transform BirdWaitPosition;

public LineRenderer SlingshotLineRenderer1;
public LineRenderer SlingshotLineRenderer2;
public LineRenderer TrajectoryLineRenderer;

public float ThrowSpeed;

private int _segmentCount = 15;
private int _segmentScale = 2;
private float _setStringshotLimit = 1.5f;
private float _newBirdDistance =0.5f;
private float _newLineDistance = 0.5f;
private float _slingshotLimit;
```



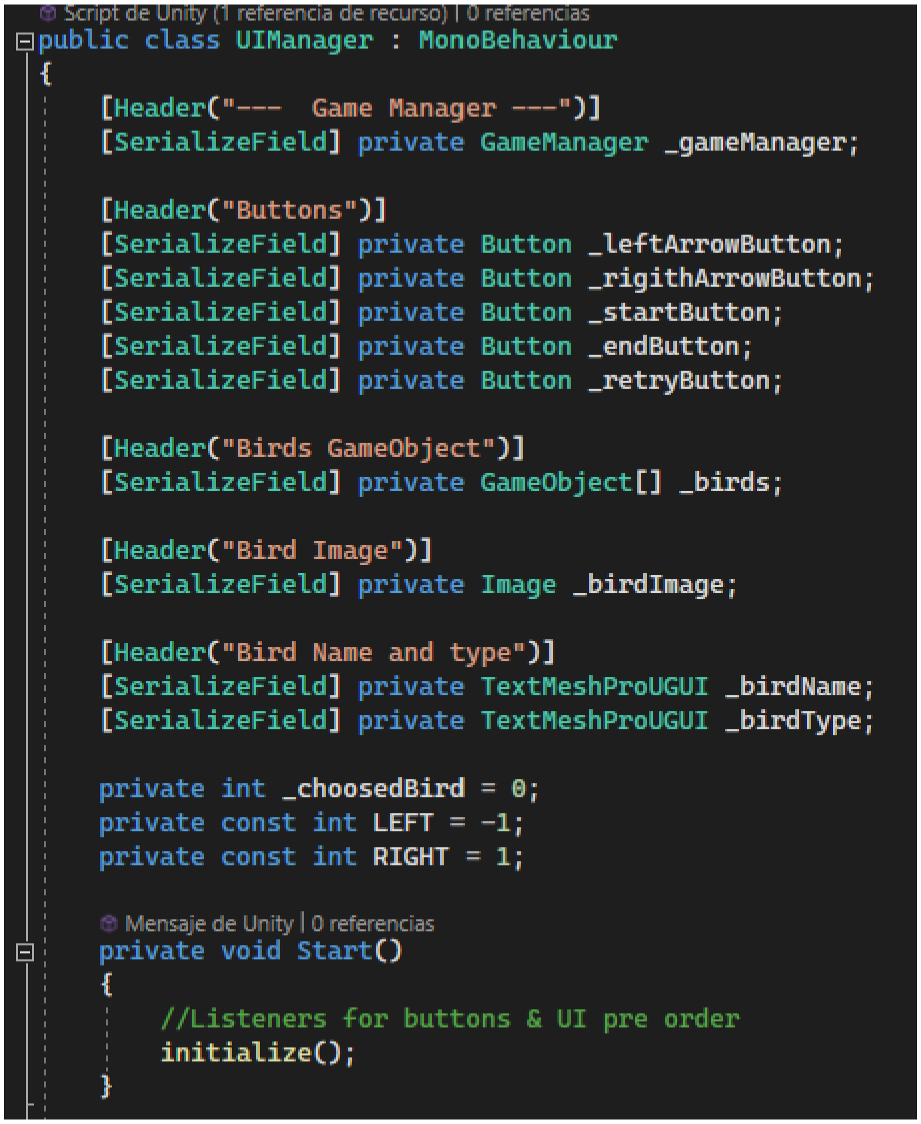
```
private void ShotBird(float distance)
{
    Vector3 velocity = SlingshotMiddleVector - BirdToThrow.transform.position;
    BirdToThrow.GetComponent<Bird>().ShootBird();
    BirdToThrow.GetComponent<Rigidbody2D>().velocity = new Vector2(velocity.x, velocity.y) * ThrowSpeed * distance;
    if (BirdToThrow.GetComponent<Bird>().birdSO.multipleBird)
    {
        StartCoroutine(DivideBird(_timeToDivideBird));
    }
    if (BirdThrown != null)
        BirdThrown(this, EventArgs.Empty);
}
```



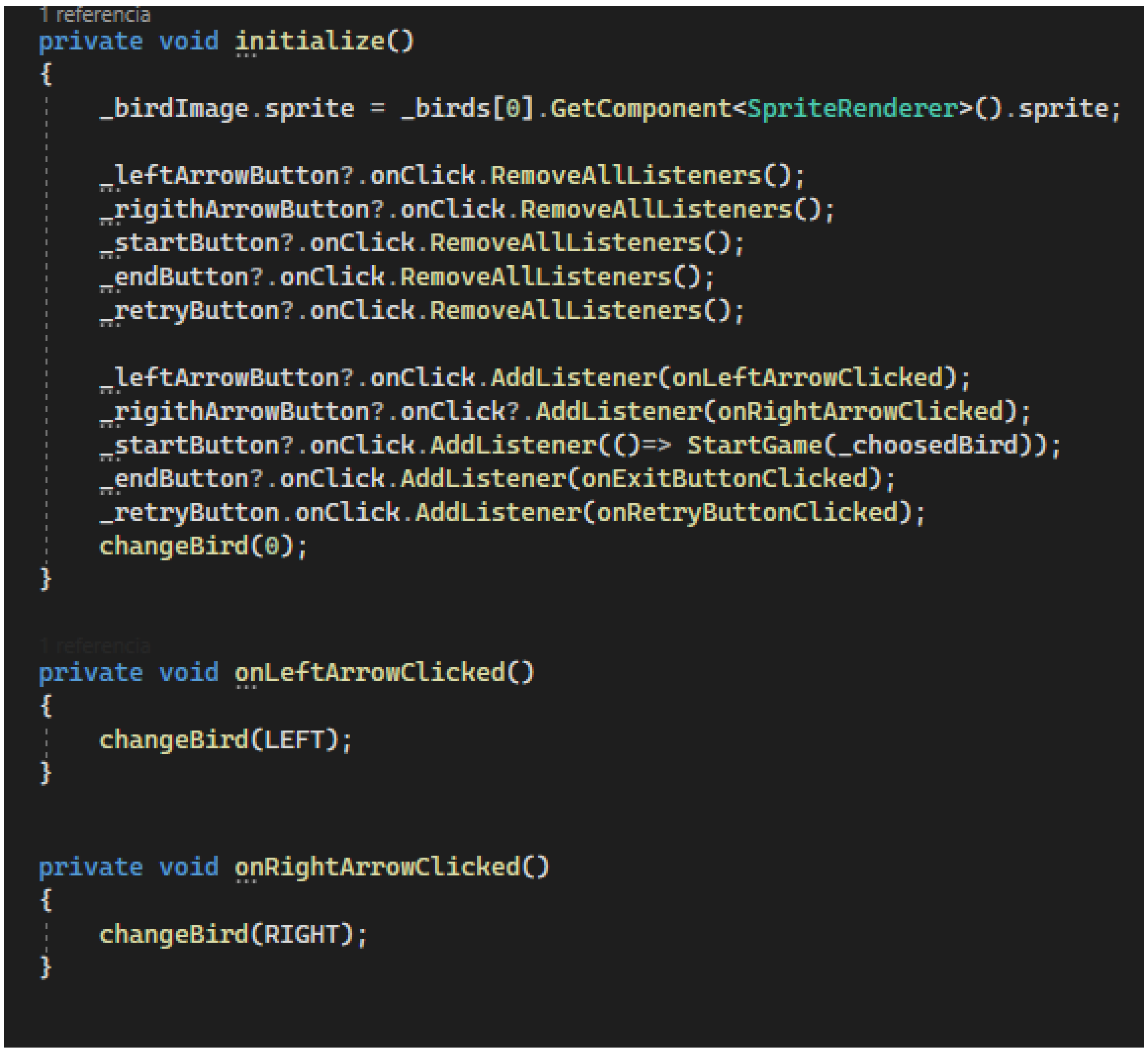
```csharp
IEnumerator DivideBird(float seconds)
{
    float constDist = -0.5f;
    List<GameObject> dividedBirds = new List<GameObject>();
    yield return new WaitForSeconds(seconds);
    for (int i = 0; i < 3; i++)
    {
        GameObject newBird = Instantiate(BirdToThrow,BirdToThrow.transform.position, Quaternion.identity);
        newBird.GetComponent<Rigidbody2D>().velocity = BirdToThrow.GetComponent<Rigidbody2D>().velocity;
        newBird.GetComponent<Rigidbody2D>().angularVelocity = BirdToThrow.GetComponent<Rigidbody2D>().angularVelocity;
        newBird.transform.localScale -= new Vector3(_newBirdDistance, _newBirdDistance, _newBirdDistance);
        dividedBirds.Add(newBird);
    }
    BirdToThrow.GetComponent<SpriteRenderer>().enabled = false;
    BirdToThrow.GetComponent<CircleCollider2D>().enabled = false;

    for (int i = 0; i < dividedBirds.Count; i++)
    {
        dividedBirds[i].transform.SetParent(BirdToThrow.transform);
        dividedBirds[i].transform.position = new Vector3(
            dividedBirds[i].transform.position.x,
            dividedBirds[i].transform.position.y + constDist,
            dividedBirds[i].transform.position.z
            );
        constDist += _newBirdDistance;
    }
}
```

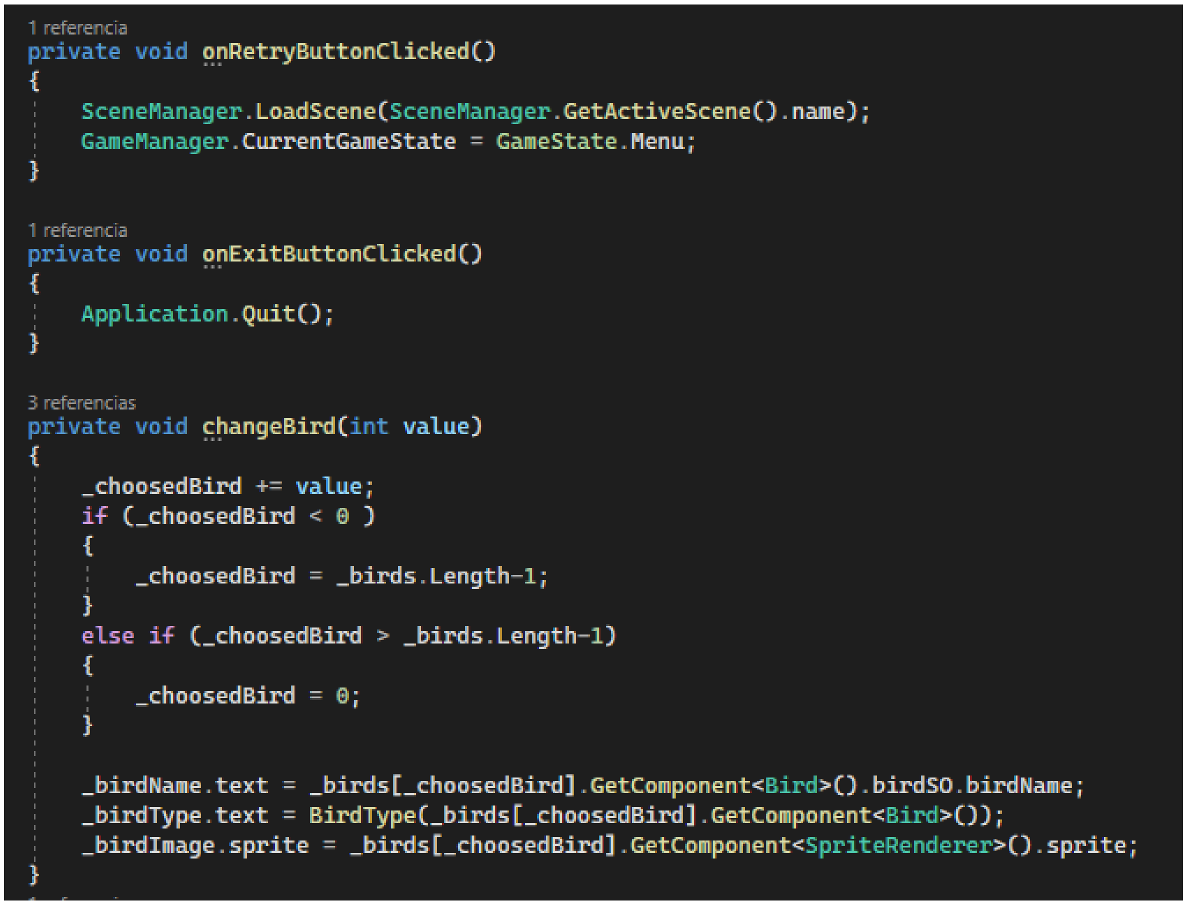
CREATION: UIMANAGER.CS MONOBEHAVIOUR

- Referencias de botones:
  - START GAME
  - SELECT BIRD LEFT / RIGHT
  - RETRY
  - QUIT GAME
- Listeners
- Referencias a aves para elección (GameObject.sprite)

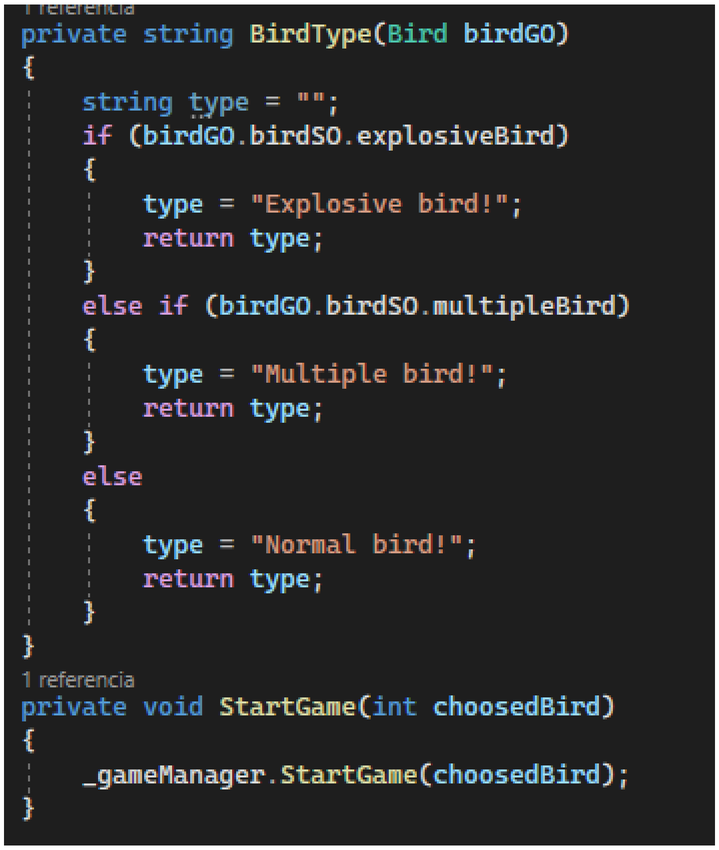


```
Script de Unity (1 referencia de recurso) | 0 referencias
public class UIManager : MonoBehaviour
{
    [Header("---  Game Manager ---")]
    [SerializeField] private GameManager _gameManager;

    [Header("Buttons")]
    [SerializeField] private Button _leftArrowButton;
    [SerializeField] private Button _rigithArrowButton;
    [SerializeField] private Button _startButton;
    [SerializeField] private Button _endButton;
    [SerializeField] private Button _retryButton;

    [Header("Birds GameObject")]
    [SerializeField] private GameObject[] _birds;

    [Header("Bird Image")]
    [SerializeField] private Image _birdImage;

    [Header("Bird Name and type")]
    [SerializeField] private TextMeshProUGUI _birdName;
    [SerializeField] private TextMeshProUGUI _birdType;

    private int _choosedBird = 0;
    private const int LEFT = -1;
    private const int RIGHT = 1;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        //Listeners for buttons & UI pre order
        initialize();
    }
```


```
1 referencia
private void initialize()
{
    _birdImage.sprite = _birds[0].GetComponent<SpriteRenderer>().sprite;

    _leftArrowButton?.onClick.RemoveAllListeners();
    _rigithArrowButton?.onClick.RemoveAllListeners();
    _startButton?.onClick.RemoveAllListeners();
    _endButton?.onClick.RemoveAllListeners();
    _retryButton?.onClick.RemoveAllListeners();

    _leftArrowButton?.onClick.AddListener(onLeftArrowClicked);
    _rigithArrowButton?.onClick?.AddListener(onRightArrowClicked);
    _startButton?.onClick.AddListener(()=> StartGame(_choosedBird));
    _endButton?.onClick.AddListener(onExitButtonClicked);
    _retryButton.onClick.AddListener(onRetryButtonClicked);
    changeBird(0);
}

1 referencia
private void onLeftArrowClicked()
{
    changeBird(LEFT);
}


private void onRightArrowClicked()
{
    changeBird(RIGHT);
}
```

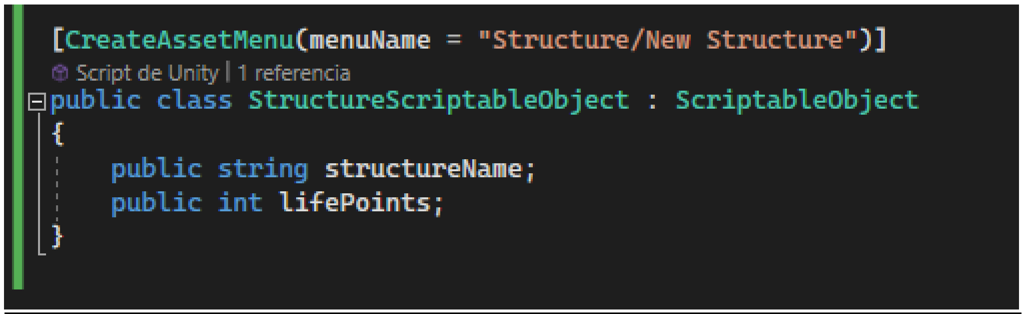


```
1 referencia
private void onRetryButtonClicked()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    GameManager.CurrentGameState = GameState.Menu;
}

1 referencia
private void onExitButtonClicked()
{
    Application.Quit();
}

3 referencias
private void changeBird(int value)
{
    _choosedBird += value;
    if (_choosedBird < 0 )
    {
        _choosedBird = _birds.Length-1;
    }
    else if (_choosedBird > _birds.Length-1)
    {
        _choosedBird = 0;
    }

    _birdName.text = _birds[_choosedBird].GetComponent<Bird>().birdSO.birdName;
    _birdType.text = BirdType(_birds[_choosedBird].GetComponent<Bird>());
    _birdImage.sprite = _birds[_choosedBird].GetComponent<SpriteRenderer>().sprite;
}
```

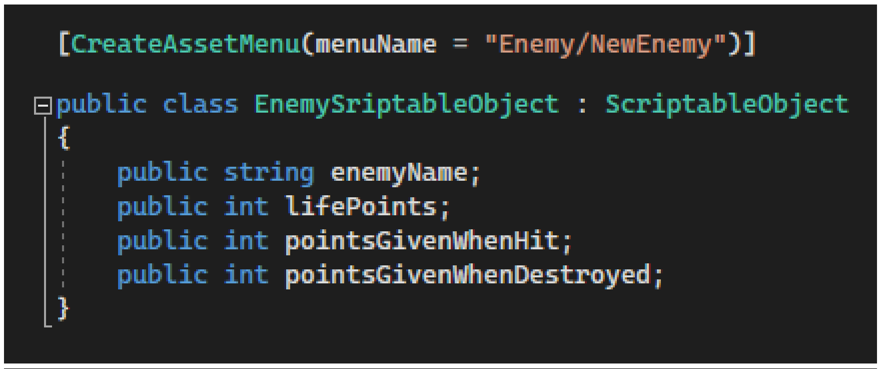


```
1 referencia
private string BirdType(Bird birdGO)
{
    string type = "";
    if (birdGO.birdSO.explosiveBird)
    {
        type = "Explosive bird!";
        return type;
    }
    else if (birdGO.birdSO.multipleBird)
    {
        type = "Multiple bird!";
        return type;
    }
    else
    {
        type = "Normal bird!";
        return type;
    }
}
1 referencia
private void StartGame(int choosedBird)
{
    _gameManager.StartGame(choosedBird);
}
```

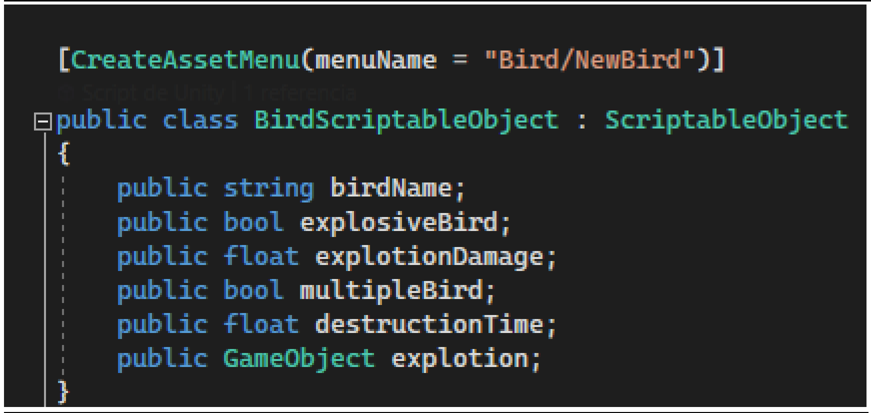
Scriptable Objects:



```
[CreateAssetMenu(menuName = "Structure/New Structure")]
Script de Unity | 1 referencia
public class StructureScriptableObject : ScriptableObject
{
    public string structureName;
    public int lifePoints;
}
```



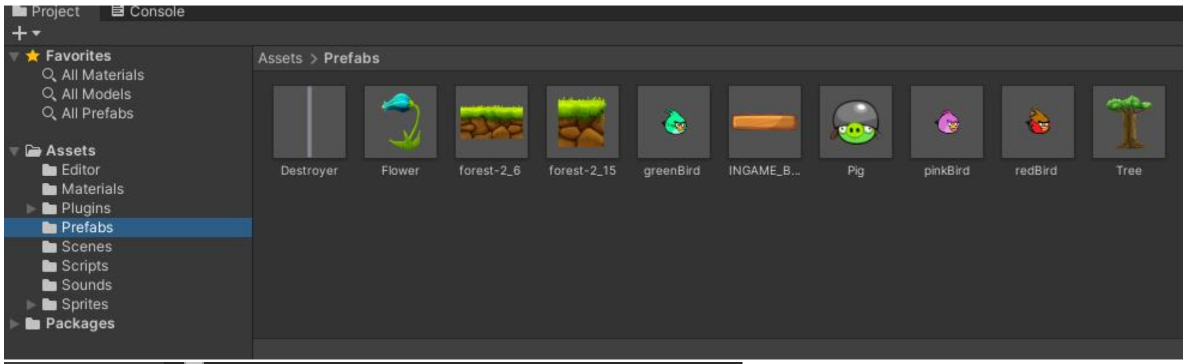
```csharp
[CreateAssetMenu(menuName = "Enemy/NewEnemy")]

public class EnemySriptableObject : ScriptableObject
{
    public string enemyName;
    public int lifePoints;
    public int pointsGivenWhenHit;
    public int pointsGivenWhenDestroyed;
}
```

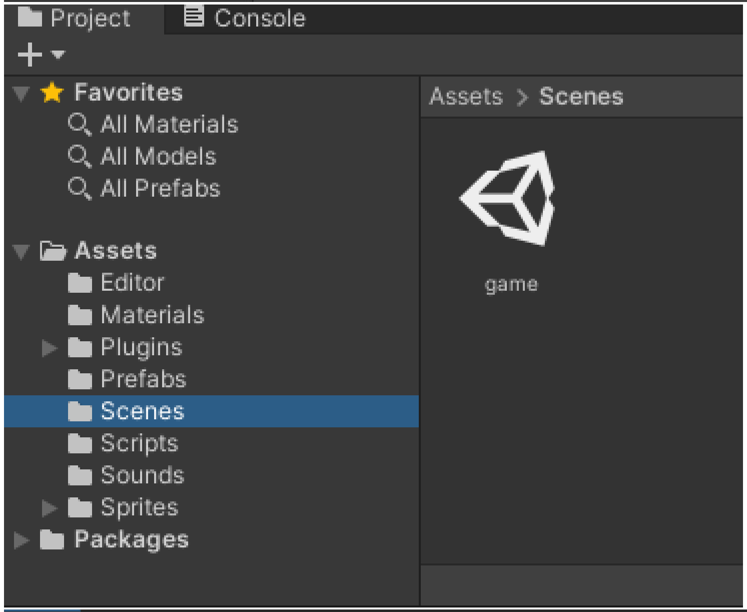


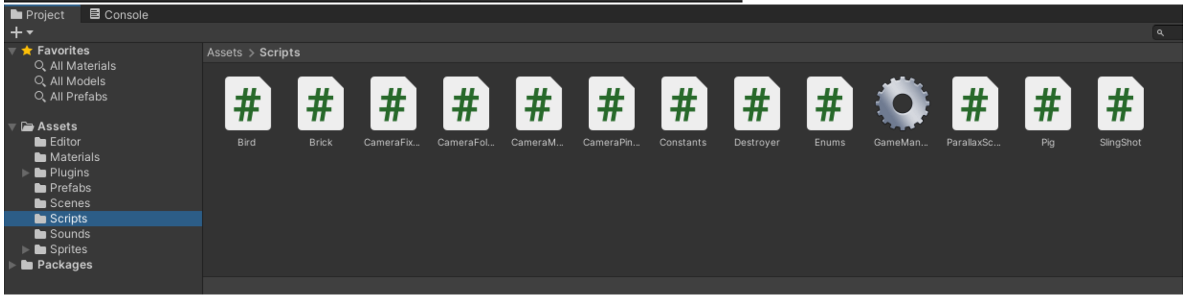
```csharp
[CreateAssetMenu(menuName = "Bird/NewBird")]
Script de Unity | 1 referencia
public class BirdScriptableObject : ScriptableObject
{
    public string birdName;
    public bool explosiveBird;
    public float explotionDamage;
    public bool multipleBird;
    public float destructionTime;
    public GameObject explotion;
}
```

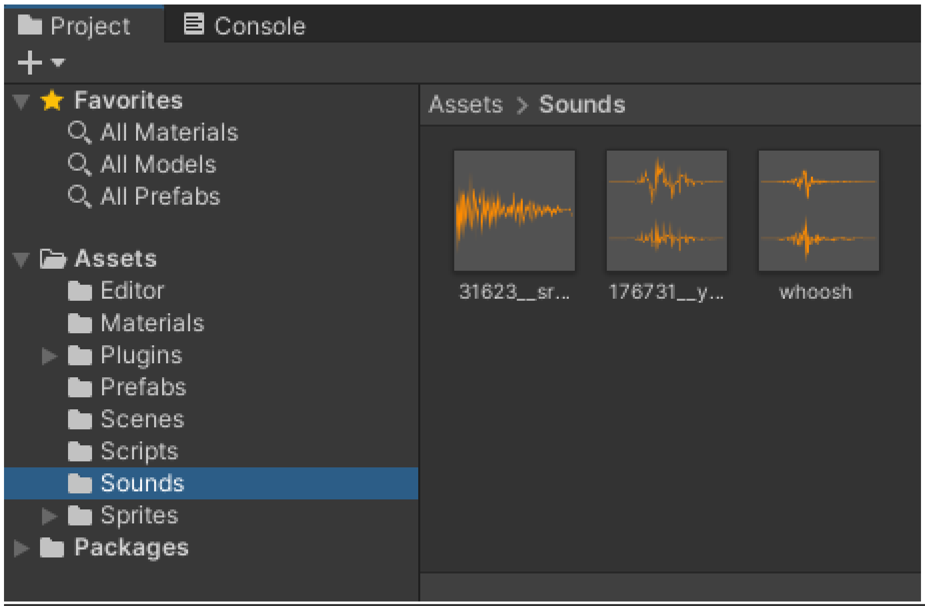
2. Estructura de Carpetas

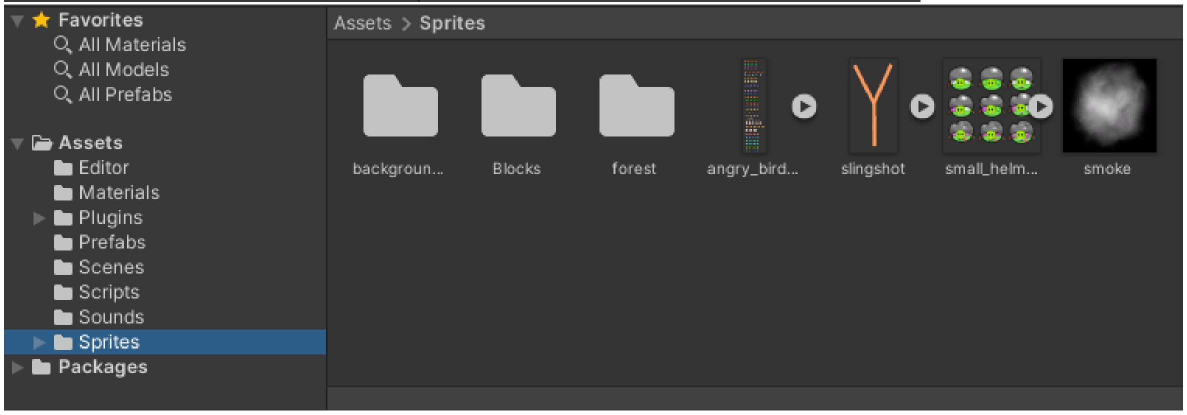
OLD:



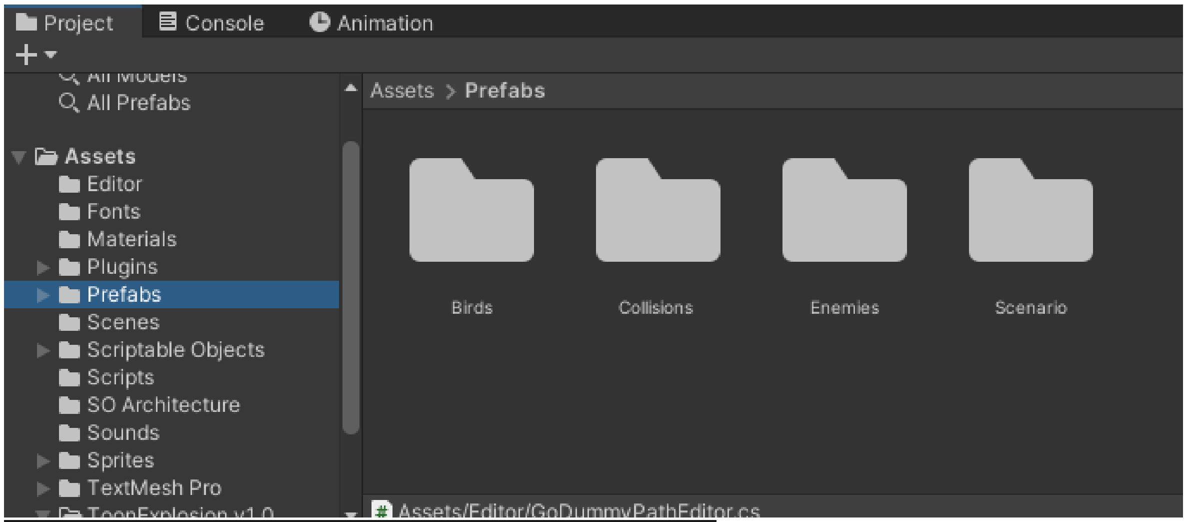


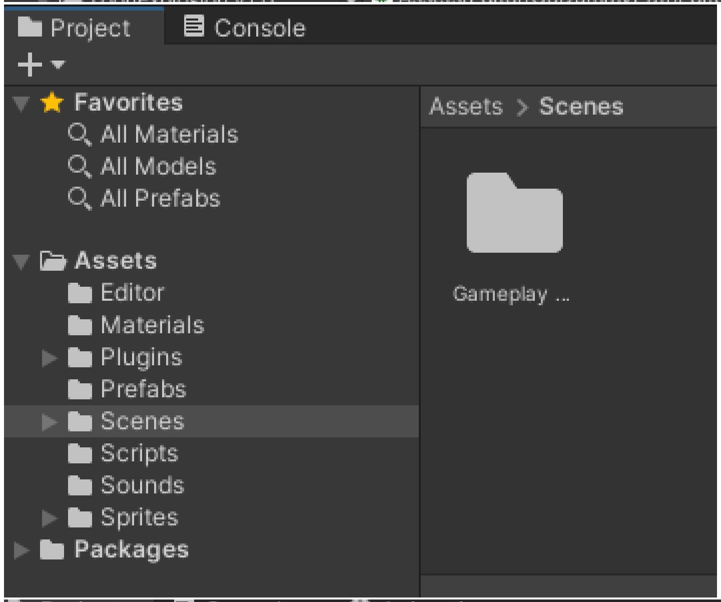


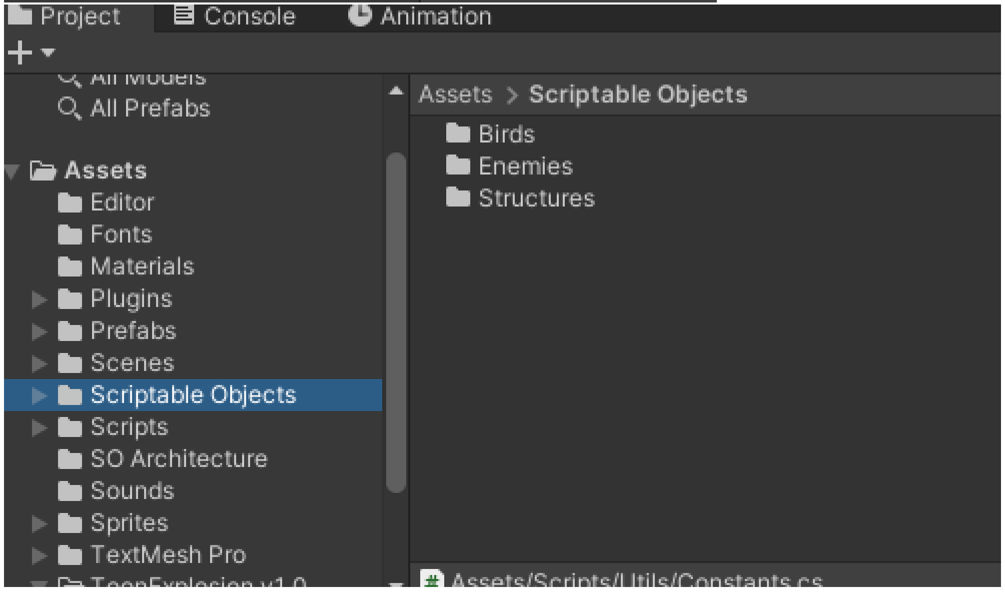

Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Editor
Materials
Plugins
Prefabs
Scenes
Scripts
Sounds
Sprites
Packages
Assets > Sounds
31623__sr...
176731__y...
whoosh

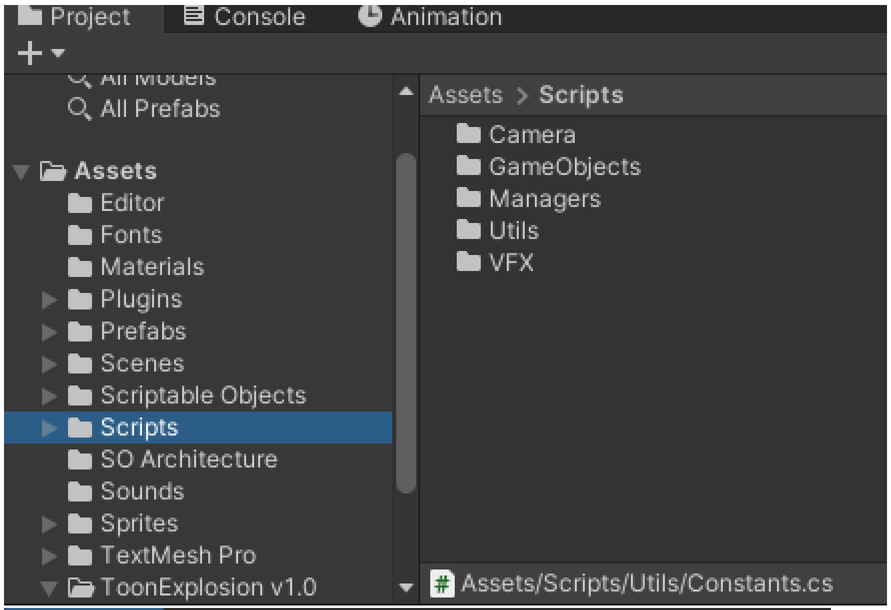

Favorites
All Materials
All Models
All Prefabs
Assets
Editor
Materials
Plugins
Prefabs
Scenes
Scripts
Sounds
Sprites
Packages
Assets > Sprites
backgroun...
Blocks
forest
angry_bird...
slingshot
small_helm...
smoke

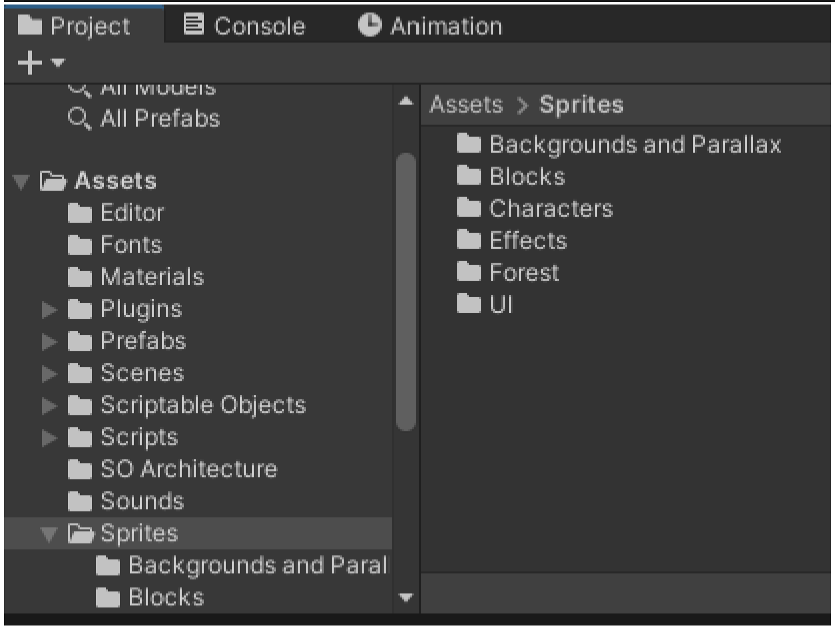
NEW:


Project
Console
Animation
All Prefabs
Assets
Editor
Fonts
Materials
Plugins
Prefabs
Scenes
Scriptable Objects
Scripts
SO Architecture
Sounds
Sprites
TextMesh Pro
Assets > Prefabs
Birds
Collisions
Enemies
Scenario
Assets/Editor/GoDummyPathEditor.cs

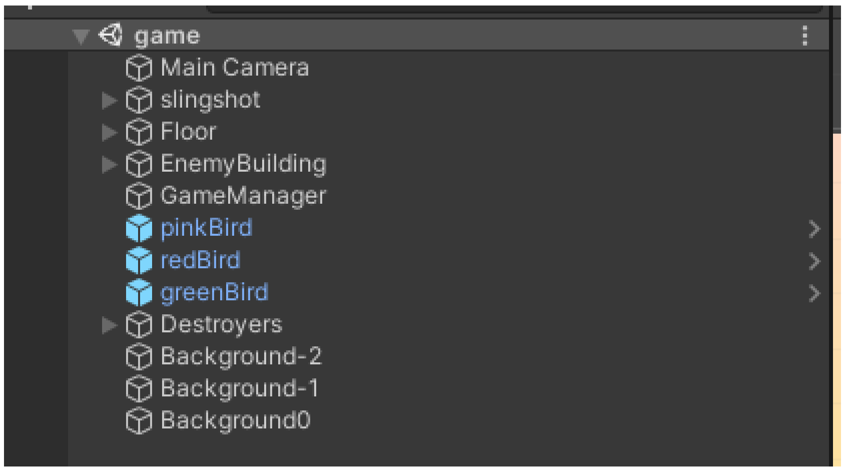

Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Editor
Materials
Plugins
Prefabs
Scenes
Scripts
Sounds
Sprites
Packages
Assets > Scenes
Gameplay ...


Project
Console
Animation
All Prefabs
Assets
Editor
Fonts
Materials
Plugins
Prefabs
Scenes
Scriptable Objects
Scripts
SO Architecture
Sounds
Sprites
TextMesh Pro
Assets > Scriptable Objects
Birds
Enemies
Structures
Assets/Scripts/Utils/Constants.cs


Project
Console
Animation
All Models
All Prefabs
Assets
Editor
Fonts
Materials
Plugins
Prefabs
Scenes
Scriptable Objects
Scripts
SO Architecture
Sounds
Sprites
TextMesh Pro
ToonExplosion v1.0
Assets > Scripts
Camera
GameObjects
Managers
Utils
VFX
Assets/Scripts/Utils/Constants.cs

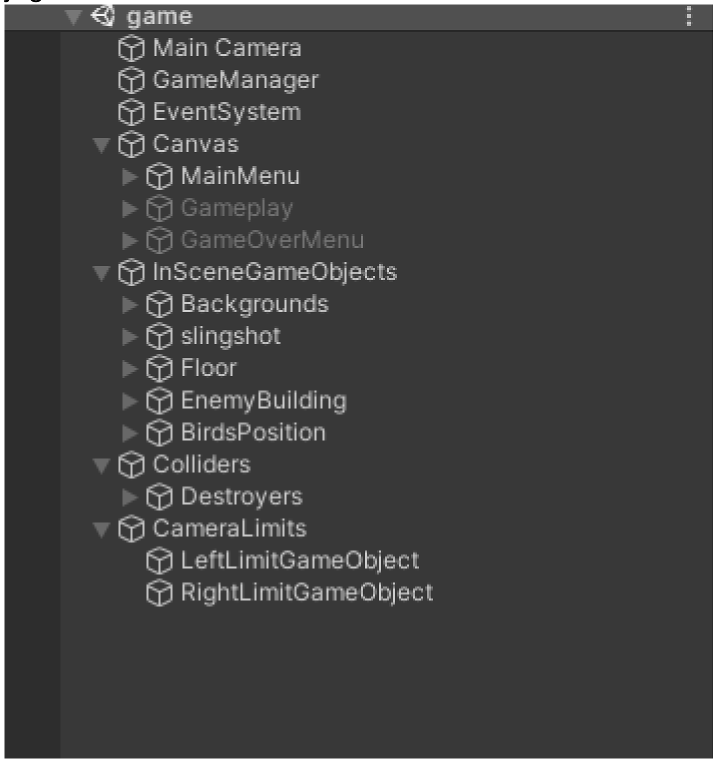

Project
Console
Animation
All Models
All Prefabs
Assets
Editor
Fonts
Materials
Plugins
Prefabs
Scenes
Scriptable Objects
Scripts
SO Architecture
Sounds
Sprites
Backgrounds and Paral
Blocks
Assets > Sprites
Backgrounds and Parallax
Blocks
Characters
Effects
Forest
UI

3. Jerarquía de Objetos y configuración de escena:

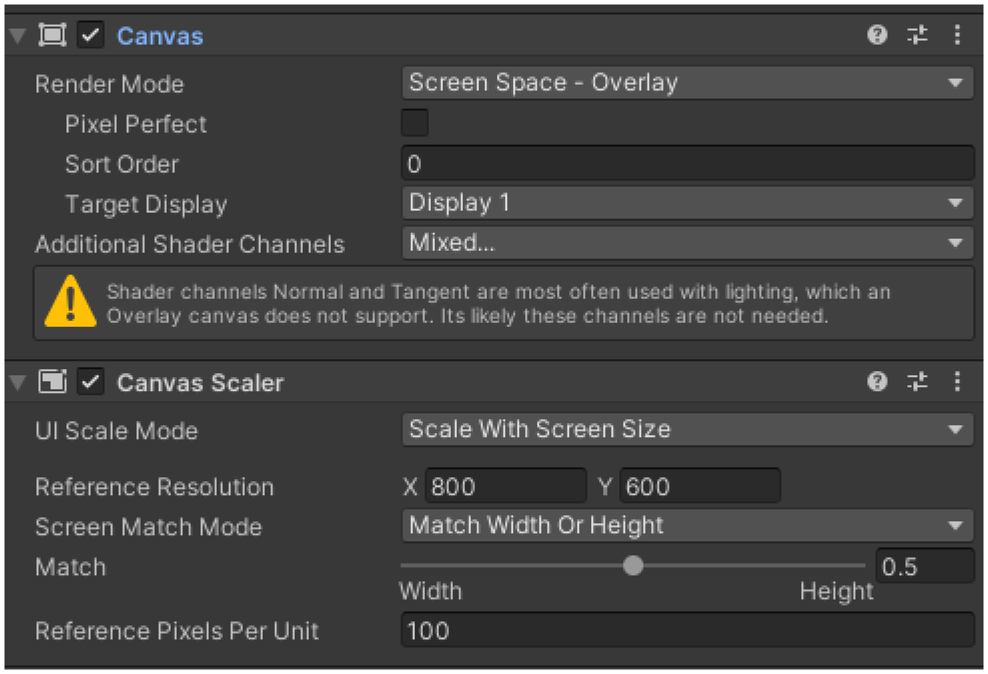


```
▼ game
    Main Camera
  ▶ slingshot
  ▶ Floor
  ▶ EnemyBuilding
    GameManager
    pinkBird
    redBird
    greenBird
  ▶ Destroyers
    Background-2
    Background-1
    Background0
```

- Poco orden de Gameobjects pre dispuestos en escena
- Sin jerarquía de Managers
- No hay Canvas para escalar UI
- Innecesaria instanciación de birds (Pink red Green) por configuración de GameManager
- No hay límite bien configurado para la cámara y no aplica si se intenta jugar en Portrait



```
▼ game
    Main Camera
    GameManager
    EventSystem
  ▼ Canvas
    ▶ MainMenu
    ▶ Gameplay
    ▶ GameOverMenu
  ▼ InSceneGameObjects
    ▶ Backgrounds
    ▶ slingshot
    ▶ Floor
    ▶ EnemyBuilding
    ▶ BirdsPosition
  ▼ Colliders
    ▶ Destroyers
  ▼ CameraLimits
      LeftLimitGameObject
      RightLimitGameObject
```



Canvas

Render Mode: Screen Space - Overlay

Pixel Perfect: ☐

Sort Order: 0

Target Display: Display 1

Additional Shader Channels: Mixed...

Shader channels Normal and Tangent are most often used with lighting, which an Overlay canvas does not support. Its likely these channels are not needed.

Canvas Scaler

UI Scale Mode: Scale With Screen Size

Reference Resolution: X 800 Y 600

Screen Match Mode: Match Width Or Height

Match: 0.5 (Width – Height)

Reference Pixels Per Unit: 100