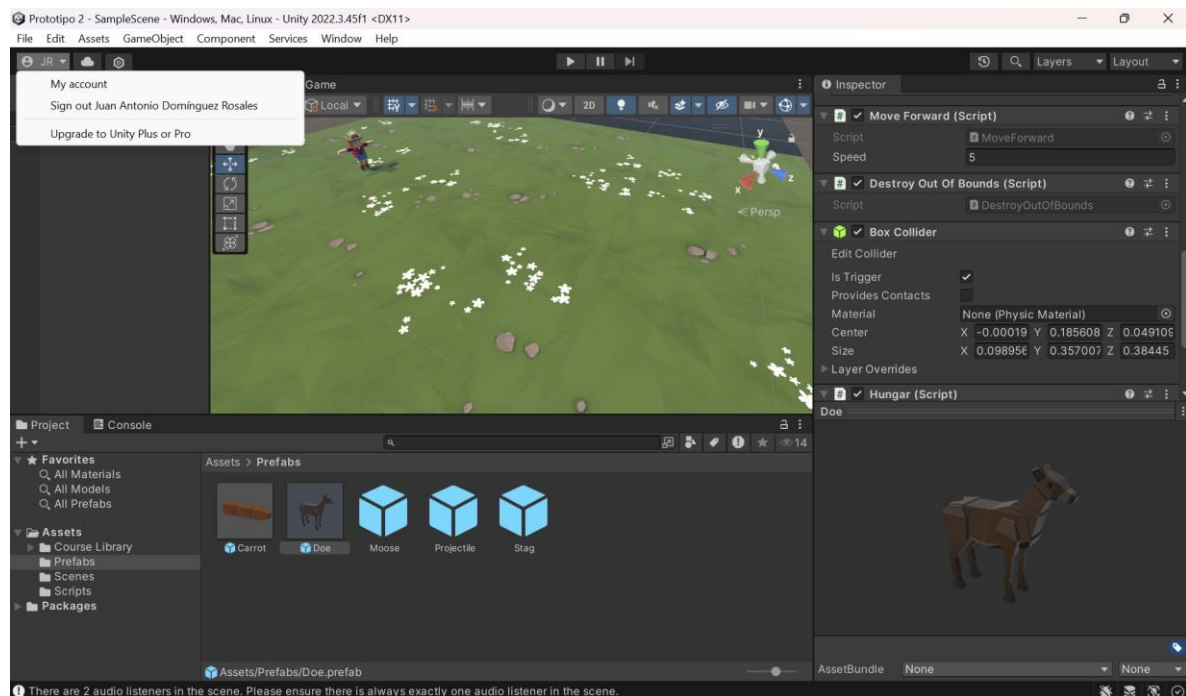
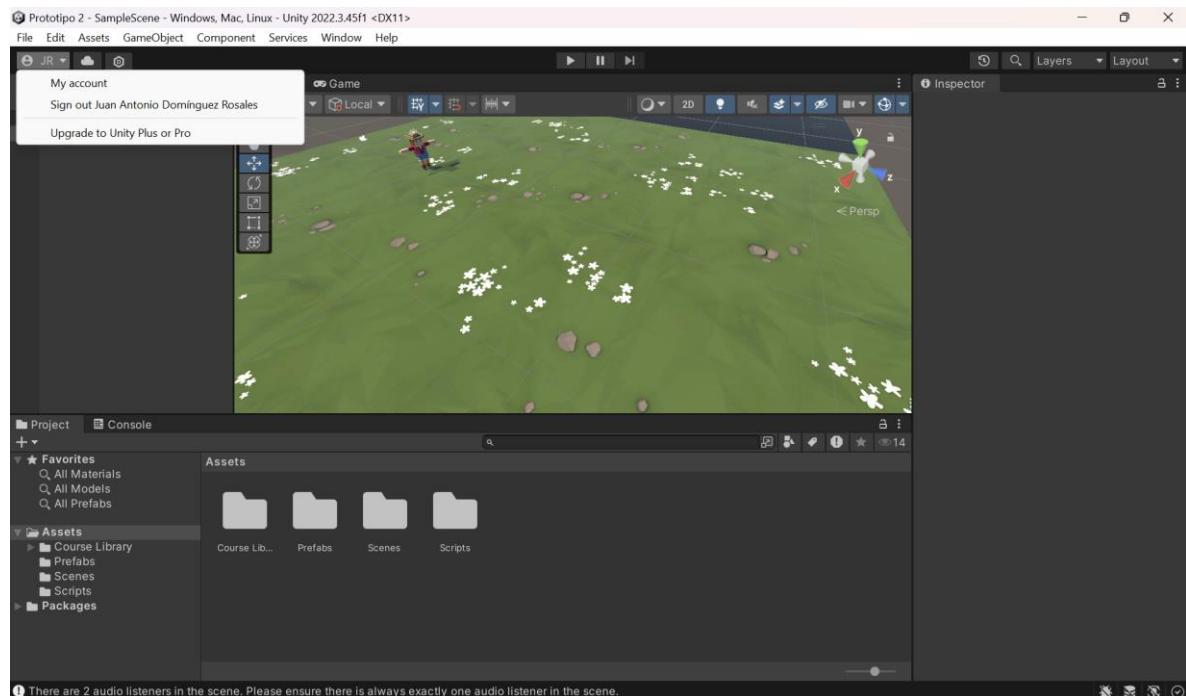
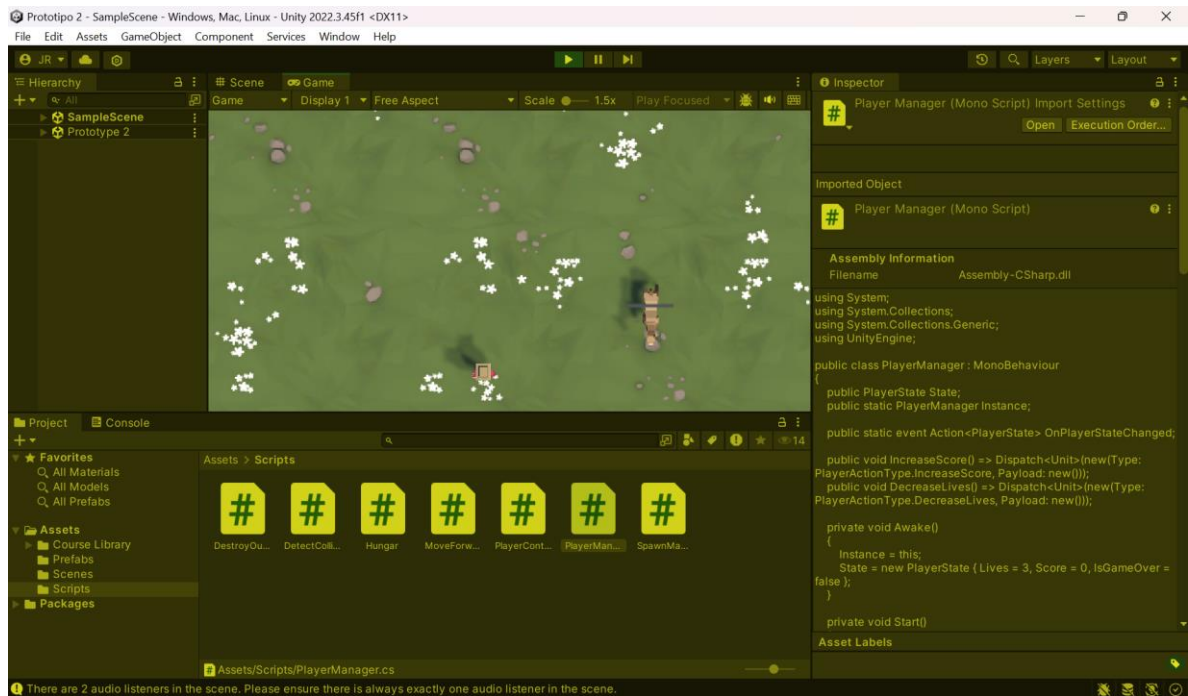
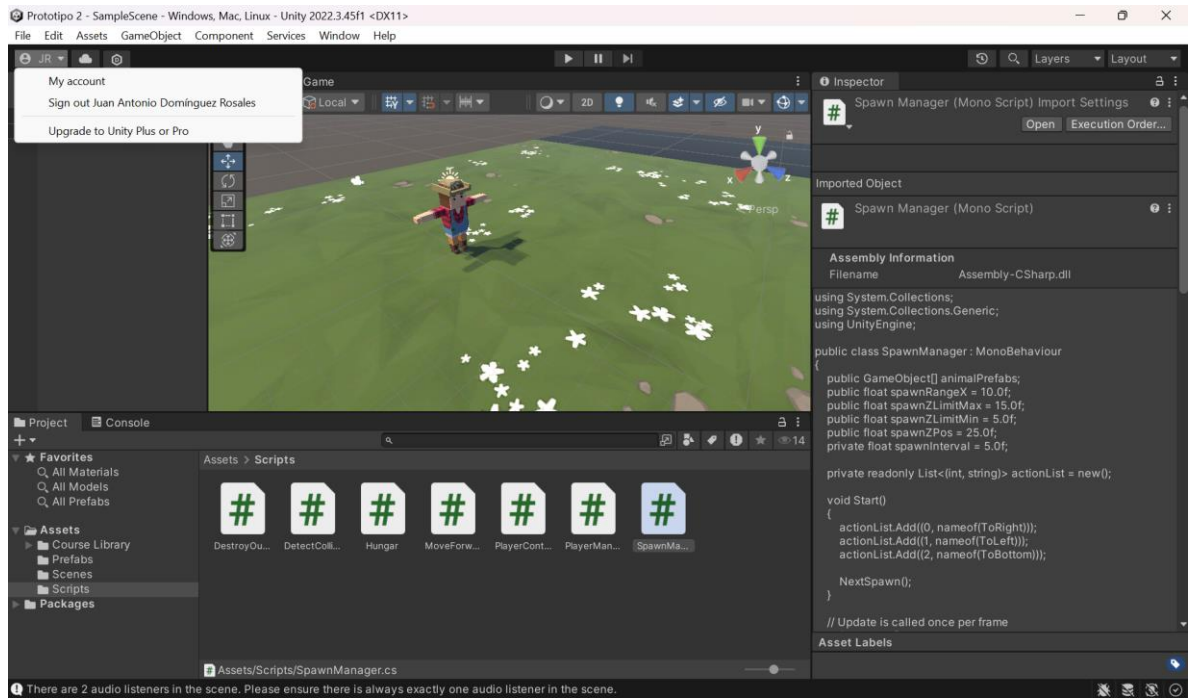


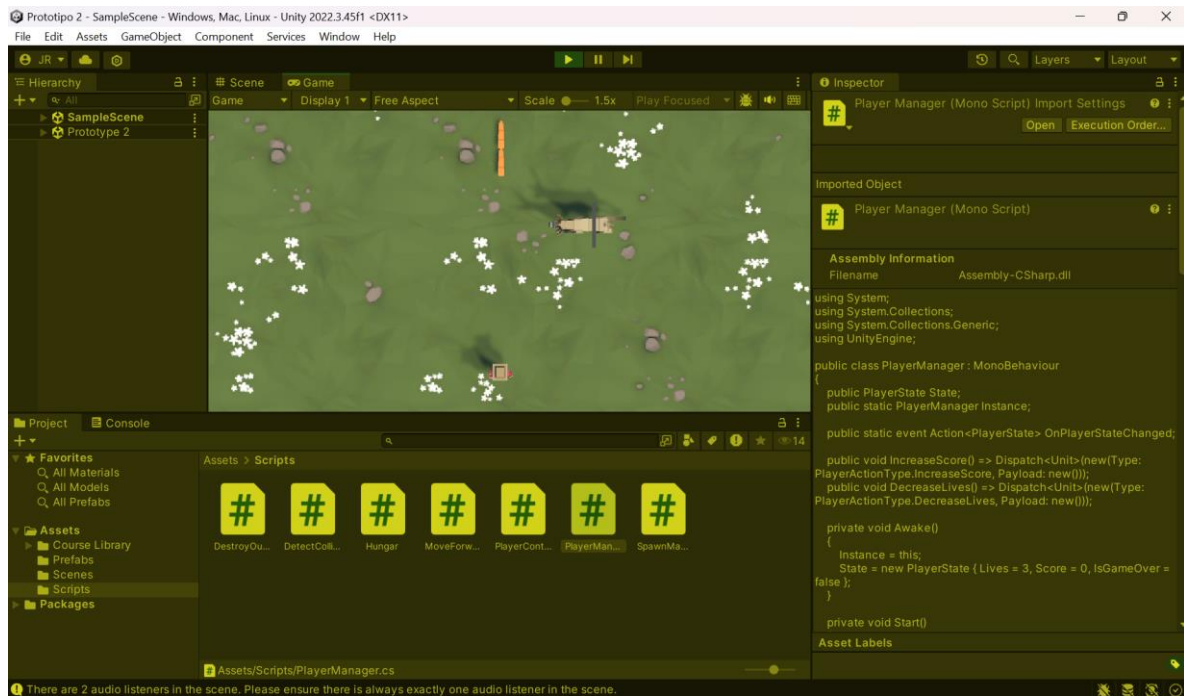
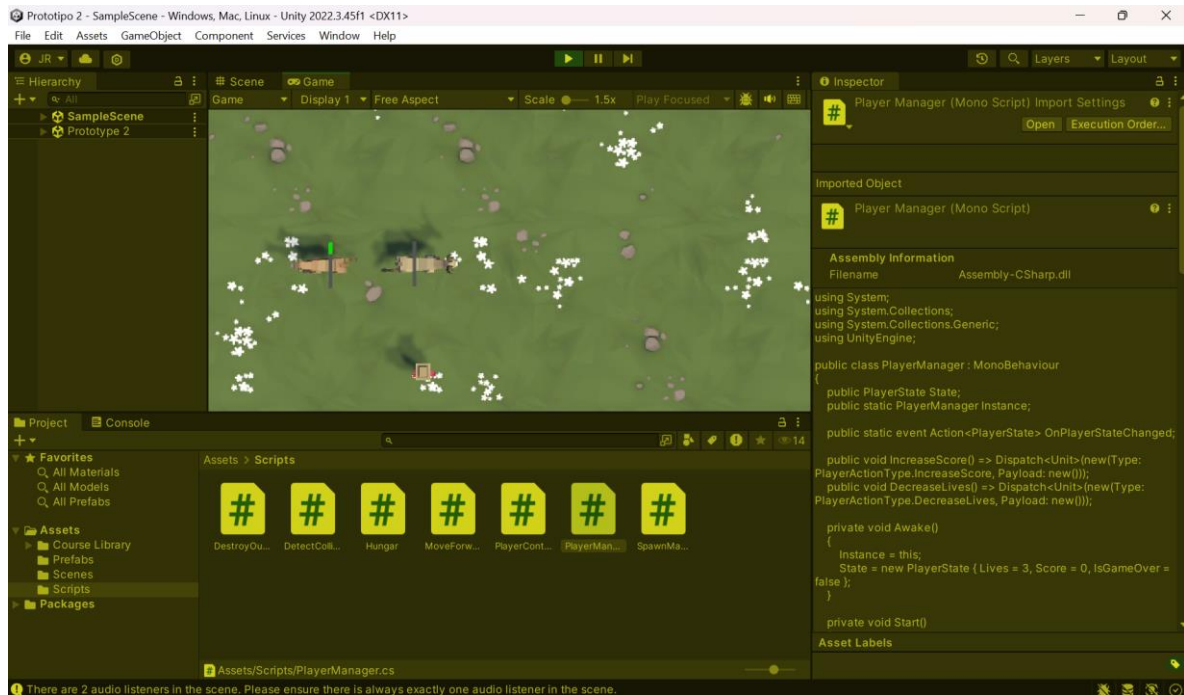
Prototipo 2

Juan Antonio Domínguez Rosales – GIDS4102

10/10/2024







This screenshot shows the initial setup of the `SpawnManager` script in Visual Studio Code. The Explorer panel on the left shows the project structure with `Assets > Scripts > C# > SpawnManager.cs` selected. The main editor displays the following code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 references
6
7 public class SpawnManager : MonoBehaviour
8
9 9 references
10 public GameObject[] animalPrefabs;
11 2 references
12 public float spawnRangeX = 10.0f;
13 2 references
14 public float spawnZLimitMax = 15.0f;
15 2 references
16 public float spawnZLimitMin = 5.0f;
17 1 reference
18 public float spawnZPos = 25.0f;
19 1 reference
20 private float spawnInterval = 5.0f;
21
22 5 references
23 private readonly List<(int, string)> actionList = new();
24
25 0 references
26 void Start()
27 {
28     actionList.Add(0, nameof(ToRight));
29     actionList.Add(1, nameof(ToLeft));
30     actionList.Add(2, nameof(ToBottom));
31
32     NextSpawn();
33 }
34
35 // Update is called once per frame
36 0 references
37 void Update()
38 {
39 }
```

The status bar at the bottom indicates the project is `Assembly-CSharp` and is being debugged on `Any CPU`. The cursor is at line 11, column 36.

This screenshot shows the implementation of the `SpawnManager` script in Visual Studio Code. The Explorer panel on the left shows the project structure with `Assets > Scripts > C# > SpawnManager.cs` selected. The main editor displays the following code:

```
5 public class SpawnManager : MonoBehaviour
6
7 // Update is called once per frame
8 0 references
9 void Update()
10 {
11 }
12
13 1 reference
14 void ToRight()
15 {
16     int animalIndex = Random.Range(0, animalPrefabs.Length);
17     Vector3 spawnPos = new(-20, 0, Random.Range(spawnZLimitMin,
18 spawnZLimitMax));
19
20     GameObject animalPrefab = animalPrefabs[animalIndex];
21     animalPrefab.tag = "Left";
22
23     Instantiate(animalPrefab,
24 spawnPos,
25 animalPrefabs[animalIndex].transform.rotation * Quaternion.Euler(new
26 (0, -90, 0));
27
28     NextSpawn();
29 }
30
31 1 reference
32 void ToLeft()
33 {
34     int animalIndex = Random.Range(0, animalPrefabs.Length);
35     Vector3 spawnPos = new(20, 0, Random.Range(spawnZLimitMin,
36 spawnZLimitMax));
37
38     GameObject animalPrefab = animalPrefabs[animalIndex];
39     animalPrefab.tag = "Right";
40 }
```

The status bar at the bottom indicates the project is `Assembly-CSharp` and is being debugged on `Any CPU`. The cursor is at line 11, column 36.

This screenshot shows the Visual Studio Code editor with the `SpawnManager.cs` script open. The Explorer panel on the left shows the project structure, including `Assets`, `Scripts`, and `Logs`. The `SpawnManager.cs` file is selected in the Explorer. The script is a C# class that inherits from `MonoBehaviour`. It contains two methods: `ToLeft()` and `ToBottom()`. Both methods use `Random.Range` to select an animal index and generate a spawn position. The `ToLeft()` method sets the spawn position to `(20, 0, Random.Range(spawnZLimitMin, spawnZLimitMax))` and the animal tag to `"Right"`. The `ToBottom()` method sets the spawn position to `(Random.Range(-spawnRangeX, spawnRangeX), 0, spawnZPos)` and the animal tag to `"Top"`. Both methods instantiate the selected animal prefab at the specified position and rotation, and then call `NextSpawn()`.

```
File Edit Selection View Go ... ← → Prototipo 2

EXPLORER
  PROTOTIPO 2
    Assets
    Scripts
      DestroyOutOfRo...
      DetectCollisions...
      Hungar.cs
      MoveForward.cs
      PlayerController...
      PlayerManager.cs
      SpawnManager.cs
    Logs
    Packages
    UserSettings
    Assembly-CSharp.c...
    Prototipo 2.sln

Assets > Scripts > C# > SpawnManager.cs > 14 SpawnManager > 15 spawnZPos
5 public class SpawnManager : MonoBehaviour
6
7 1 reference
8 void ToLeft()
9 {
10     int animalIndex = Random.Range(0, animalPrefabs.Length);
11     Vector3 spawnPos = new(20, 0, Random.Range(spawnZLimitMin,
12 spawnZLimitMax));
13
14     GameObject animalPrefab = animalPrefabs[animalIndex];
15     animalPrefab.tag = "Right";
16
17     Instantiate(animalPrefab,
18 spawnPos,
19 animalPrefabs[animalIndex].transform.rotation * Quaternion.Euler(new
20 (0, 90, 0));
21
22     NextSpawn();
23 }
24
25 1 reference
26 void ToBottom()
27 {
28     int animalIndex = Random.Range(0, animalPrefabs.Length);
29     Vector3 spawnPos = new(Random.Range(-spawnRangeX, spawnRangeX), 0,
30 spawnZPos);
31
32     GameObject animalPrefab = animalPrefabs[animalIndex];
33     animalPrefab.tag = "Top";
34
35     Instantiate(animalPrefab,
36 spawnPos,
37 animalPrefabs[animalIndex].transform.rotation);
38
39     NextSpawn();
40 }
41
42 74
43
44 0 0 0 0 Projects: Assembly-CSharp Debug Any CPU Ln 11, Col 36 Spaces: 4 UTF-8 LF C# Go Live Prettier
```

This screenshot shows the Visual Studio Code editor with the `SpawnManager.cs` script open. The Explorer panel on the left shows the project structure, including `Assets`, `Scripts`, and `Logs`. The `SpawnManager.cs` file is selected in the Explorer. The script is a C# class that inherits from `MonoBehaviour`. It contains two methods: `ToBottom()` and `NextSpawn()`. The `ToBottom()` method uses `Random.Range` to select an animal index and generate a spawn position. The `NextSpawn()` method uses `Random.Range` to select a random index from the `actionList` and then calls `Invoke` with the selected action and the `spawnInterval`.

```
File Edit Selection View Go ... ← → Prototipo 2

EXPLORER
  PROTOTIPO 2
    Assets
    Scripts
      DestroyOutOfRo...
      DetectCollisions...
      Hungar.cs
      MoveForward.cs
      PlayerController...
      PlayerManager.cs
      SpawnManager.cs
    Logs
    Packages
    UserSettings
    Assembly-CSharp.c...
    Prototipo 2.sln

Assets > Scripts > C# > SpawnManager.cs > 14 SpawnManager > 15 spawnZPos
5 public class SpawnManager : MonoBehaviour
6
7 1 reference
8 void ToBottom()
9 {
10     int animalIndex = Random.Range(0, animalPrefabs.Length);
11     Vector3 spawnPos = new(Random.Range(-spawnRangeX, spawnRangeX), 0,
12 spawnZPos);
13
14     GameObject animalPrefab = animalPrefabs[animalIndex];
15     animalPrefab.tag = "Top";
16
17     Instantiate(animalPrefab,
18 spawnPos,
19 animalPrefabs[animalIndex].transform.rotation);
20
21     NextSpawn();
22 }
23
24 4 references
25 void NextSpawn()
26 {
27     int randomIndex = Random.Range(0, actionList.Count);
28     (int, string) tuple = actionList.Find(tuple => tuple.Item1 ==
29 randomIndex);
30     if (tuple != (null, null))
31         Invoke(tuple.Item2, spawnInterval);
32 }
33
34 74
35
36 0 0 0 0 Projects: Assembly-CSharp Debug Any CPU Ln 11, Col 36 Spaces: 4 UTF-8 LF C# Go Live Prettier
```



```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class PlayerManager : MonoBehaviour
7 {
8     public PlayerState State;
9     public static PlayerManager Instance;
10
11     public static event Action<PlayerState> OnPlayerStateChanged;
12
13     public void IncreaseScore() => Dispatch<Unit>(new(Type: PlayerActionType.
14     IncreaseScore, Payload: new()));
15
16     public void DecreaseLives() => Dispatch<Unit>(new(Type: PlayerActionType.
17     DecreaseLives, Payload: new()));
18
19     private void Awake()
20     {
21         Instance = this;
22         State = new PlayerState { Lives = 3, Score = 0, IsGameOver = false };
23     }
24
25     private void Start()
26     {
27         Debug.Log("Lives = " + State.Lives);
28         Debug.Log("Score = " + State.Score);
29     }
30 }
```

```
31
32
33     private void Start()
34     {
35         Debug.Log("Lives = " + State.Lives);
36         Debug.Log("Score = " + State.Score);
37     }
38
39     private void Dispatch<T>(PlayerAction<T> action)
40     {
41         switch (action.Type)
42         {
43             case PlayerActionType.IncreaseScore:
44                 HandleIncreaseScore();
45                 break;
46             case PlayerActionType.DecreaseLives:
47                 HandleDecreaseLives();
48                 break;
49             default:
50                 throw new ArgumentOutOfRangeException(nameof(action), action, null);
51         }
52         OnPlayerStateChanged?.Invoke(State);
53     }
54
55     private void HandleIncreaseScore()
56     {
57         State = State with { Score = State.Score + 1 };
58     }
59
60     private void HandleDecreaseLives()
61     {
62     }
```

```
File Edit Selection View Go ... Prototipo 2

EXPLORER
  PROTOTIPO
    Assets
      Scripts
        PlayerManager.cs
        SpawnManager.cs
    Packages
    UserSettings
    Assembly-CSharp
    Prototipo 2.sln

  OUTLINE
  TIMELINE
  SOLUTION EXPLORER

  0 0 0 0 Projects: Assembly-CSharp Debug Any CPU

  Ln 1, Col 1 Spaces: 4 UTF-8 LF C# Go Live Prettier
```

```
6 public class PlayerManager : MonoBehaviour
7 {
8     private void HandleDecreaseLives()
9     {
10         if (State.Lives > 0)
11             State = State with { Lives = State.Lives - 1 };
12         if (State.Lives == 0 && !State.IsGameOver)
13             State = State with { IsGameOver = true };
14     }
15 }
16
17 1 reference
18 interface IPlayerAction
19 {
20     3 references
21     PlayerActionType Type { get; }
22 }
23
24 1 reference
25 interface IPlayerAction<T> : IPlayerAction where T : notnull
26 {
27     2 references
28     T Payload { get; }
29 }
30
31 4 references
32 class PlayerAction<T> : IPlayerAction<T>
33 {
34     3 references
35     public PlayerActionType Type { get; }
36     2 references
37     public T Payload { get; }
38
39     2 references
40     public PlayerAction(PlayerActionType Type, T Payload)
41     {
42         this.Type = Type;
43     }
44 }
```

```
File Edit Selection View Go ... Prototipo 2

EXPLORER
  PROTOTIPO
    Assets
      Scripts
        PlayerManager.cs
        SpawnManager.cs
    Packages
    UserSettings
    Assembly-CSharp
    Prototipo 2.sln

  OUTLINE
  TIMELINE
  SOLUTION EXPLORER

  0 0 0 0 Projects: Assembly-CSharp Debug Any CPU

  Ln 1, Col 1 Spaces: 4 UTF-8 LF C# Go Live Prettier
```

```
69 class PlayerAction<T> : IPlayerAction<T>
70 {
71     public PlayerAction(PlayerActionType Type, T Payload)
72     {
73         this.Type = Type;
74         this.Payload = Payload;
75     }
76 }
77
78 7 references
79 enum PlayerActionType
80 {
81     2 references
82     IncreaseScore,
83     2 references
84     DecreaseLives
85 }
86
87 4 references
88 public record PlayerState
89 {
90     8 references
91     public int Lives { get; set; }
92     5 references
93     public int Score { get; set; }
94     4 references
95     public bool IsGameOver { get; set; }
96 };
97
98 namespace System.Runtime.CompilerServices
99 {
100     0 references
101     internal static class IsExternalInit { }
102 }
103
104 4 references
105 public class Unit { }
```

This screenshot shows the initial setup of the `PlayerController.cs` script in Visual Studio Code. The Explorer panel on the left shows the project structure with `PlayerController.cs` selected. The script contains the following code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 references
6 {
7     2 references
8     public float horizontalInput;
9     2 references
10    public float verticalInput;
11    2 references
12    public float speed = 10.0f;
13    4 references
14    public float xRange = 10.0f;
15    2 references
16    public float zLimitMin = 0.0f;
17    2 references
18    public float zLimitMax = 14.0f;
19
20    2 references
21    private bool IsShowedGameOver = false;
22
23    2 references
24    public GameObject projectilePrefab;
25
26    0 references
27    private void Awake()
28    {
29        PlayerManager.OnPlayerStateChanged +=
30        PlayerManager_OnPlayerStateChanged;
31    }
32
33    // Update is called once per frame
34    0 references
35    void Update()
36    {
37    }
```

This screenshot shows the `PlayerController.cs` script with movement and shooting logic implemented. The script contains the following code:

```
5 public class PlayerController : MonoBehaviour
6 {
7     // Update is called once per frame
8     0 references
9     void Update()
10    {
11        horizontalInput = Input.GetAxis("Horizontal");
12        transform.Translate(horizontalInput * speed * Time.deltaTime * Vector3.
13        right);
14
15        verticalInput = Input.GetAxis("Vertical");
16        transform.Translate(verticalInput * speed * Time.deltaTime * Vector3.
17        forward);
18
19        if (transform.position.x < -xRange)
20            transform.position = new(-xRange, transform.position.y, transform.
21            position.z);
22
23        if (transform.position.x > xRange)
24            transform.position = new(xRange, transform.position.y, transform.
25            position.z);
26
27        if (transform.position.z < zLimitMin)
28            transform.position = new(transform.position.x, transform.position.
29            y, zLimitMin);
30
31        if (transform.position.z > zLimitMax)
32            transform.position = new(transform.position.x, transform.position.
33            y, zLimitMax);
34
35        if (Input.GetKeyDown(KeyCode.Space))
36            Instantiate(projectilePrefab, transform.position + Vector3.forward
37            * 1.2f, projectilePrefab.transform.rotation);
38    }
39 }
```


This screenshot shows the Visual Studio Code editor with the `PlayerController.cs` script open. The Explorer panel on the left shows the project structure, with `Scripts` expanded and `PlayerController.cs` selected. The script is a C# class that inherits from `MonoBehaviour`. It includes an `Update` method, an `OnTriggerEnter` method that checks for collisions with objects tagged "Projectile" and decreases lives, and an `OnDestroy` method that updates the player state. It also has a reference to `PlayerManager` and a method `PlayerManager_OnPlayerStateChanged` that checks for game over conditions.

```
5 public class PlayerController : MonoBehaviour
24 void Update()
25 {
46 }
47
0 references
48 private void OnTriggerEnter(Collider other)
49 {
50     if (!other.CompareTag("Projectile"))
51     {
52         PlayerManager.Instance.DecreaseLives();
53         Debug.Log("Lives = " + PlayerManager.Instance.State.Lives);
54     }
55 }
56
0 references
57 private void OnDestroy()
58 {
59     PlayerManager.OnPlayerStateChanged -=
60     PlayerManager_OnPlayerStateChanged;
61 }
62
2 references
63 private void PlayerManager_OnPlayerStateChanged(PlayerState state)
64 {
65     if (state.IsGameOver && !IsShownGameOver)
66     {
67         IsShownGameOver = true;
68         Debug.Log("Game Over!");
69     }
70 }
71 }
```

This screenshot shows the Visual Studio Code editor with the `MoveForward.cs` script open. The Explorer panel on the left shows the project structure, with `Scripts` expanded and `MoveForward.cs` selected. The script is a C# class that inherits from `MonoBehaviour`. It includes using statements for `System.Collections`, `System.Collections.Generic`, and `UnityEngine`. It has a public float `speed` set to 40.0f and an `Update` method that moves the object forward by a distance equal to `speed * Time.deltaTime`.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
0 references
5 public class MoveForward : MonoBehaviour
6 {
7     1 reference
8     public float speed = 40.0f;
9
// Update is called once per frame
0 references
10 void Update()
11 {
12     transform.Translate(speed * Time.deltaTime * Vector3.forward);
13 }
14 }
15 }
```

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 references
6 public class Hungar : MonoBehaviour
7 {
8     1 reference
9     public GameObject foreground;
10    3 references
11    public int neededCarrots;
12    5 references
13    private int currentCarrots = 0;
14    3 references
15    private RectTransform rectTransform;
16
17    4 references
18    private float posX;
19
20    // Start is called before the first frame update
21    0 references
22    void Start()
23    {
24        rectTransform = foreground.GetComponent<RectTransform>();
25    }
26
27    // Update is called once per frame
28    0 references
29    void Update()
30    {
31        HandleHungarBarSize();
32    }
33
34    0 references
35    private void OnTriggerEnter(Collider other)
36    {
37        if (other.CompareTag("Projectile"))
```

```
38        if (other.CompareTag("Projectile"))
39        {
40            if (currentCarrots != neededCarrots)
41                currentCarrots++;
42            if (currentCarrots == neededCarrots)
43            {
44                Destroy(gameObject);
45            }
46        }
47    }
48
49    1 reference
50    void HandleHungarBarSize()
51    {
52        if (currentCarrots == 0)
53            posX = 0;
54        else
55            posX = (currentCarrots * 100) / (neededCarrots * 2);
56
57        rectTransform.anchoredPosition3D = new(posX, 0, 0);
58        rectTransform.sizeDelta = new(posX * 2, 100);
59    }
60
61 }
```

This screenshot shows the Visual Studio IDE with the 'DetectCollisions.cs' script open. The Explorer pane on the left shows the project structure for 'PROTOTIPO 2', with 'Scripts' > 'DetectCollisions.cs' selected. The script content is as follows:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 references
6 public class DetectCollisions : MonoBehaviour
7 {
8     0 references
9     private void OnTriggerEnter(Collider other)
10    {
11        if (!other.CompareTag("Player"))
12        {
13            PlayerManager.Instance.IncreaseScore();
14            Debug.Log("Score = " + PlayerManager.Instance.State.Score);
15            Destroy(gameObject);
16        }
17    }
```

The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'C#'. The 'Go Live' button is visible in the bottom right corner.

This screenshot shows the Visual Studio IDE with the 'DestroyOutOfBounds.cs' script open. The Explorer pane on the left shows the project structure for 'PROTOTIPO 2', with 'Scripts' > 'DestroyOutOfBounds.cs' selected. The script content is as follows:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 references
6 public class DestroyOutOfBounds : MonoBehaviour
7 {
8     1 reference
9     private readonly float topBound = 30.0f;
10    1 reference
11    private readonly float bottomBound = -8.0f;
12    1 reference
13    private readonly float leftBound = -23.0f;
14    1 reference
15    private readonly float rightBound = 23.0f;
16
17    // Update is called once per frame
18    0 references
19    void Update()
20    {
21        if (transform.position.z > topBound)
22            Destroy(gameObject);
23
24        if ((gameObject.CompareTag("Left") && transform.position.x >
25            rightBound) ||
26            (gameObject.CompareTag("Right") && transform.position.x <
27            leftBound) ||
28            (gameObject.CompareTag("Top") && transform.position.z < bottomBound))
29        {
30            PlayerManager.Instance.DecreaseLives();
31            Debug.Log("Lives = " + PlayerManager.Instance.State.Lives);
32            Destroy(gameObject);
33        }
34    }
35 }
```

The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'C#'. The 'Go Live' button is visible in the bottom right corner.