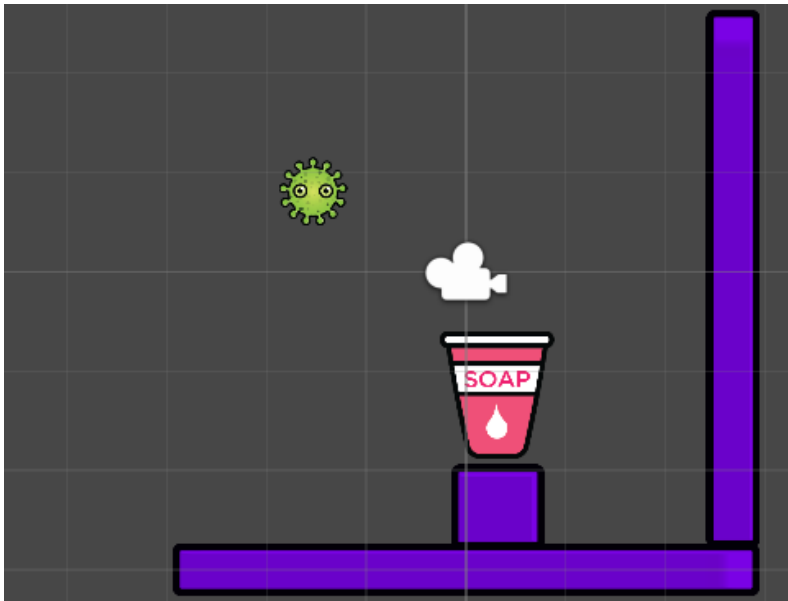
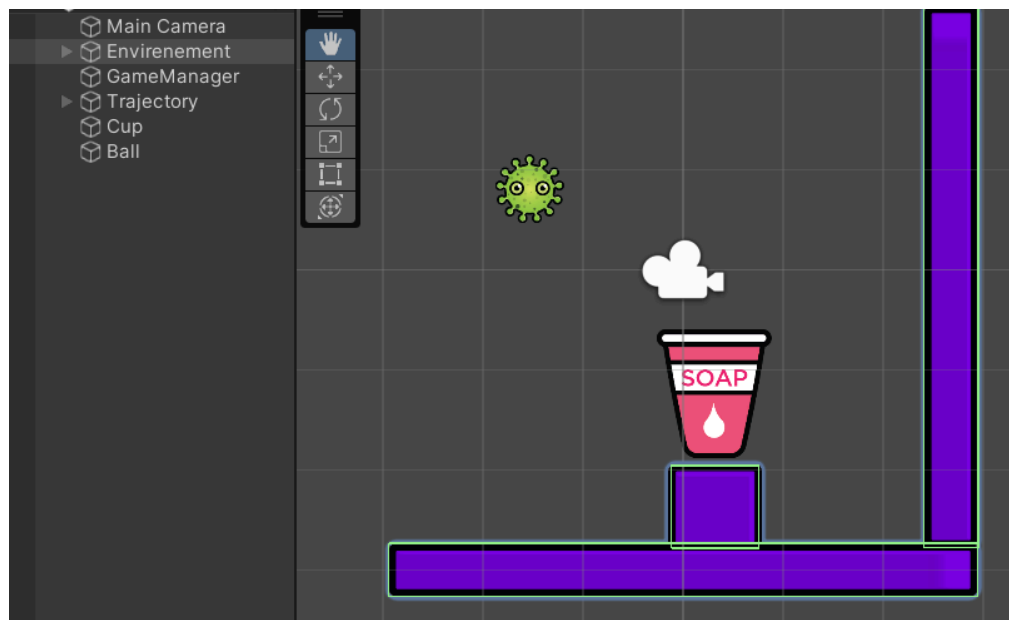


INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Desarrollo de videojuegos				
TITULO DE LA PRÁCTICA:	Laboratorio 04				
NÚMERO DE PRÁCTICA:	04	AÑO LECTIVO:	2024	NRO. SEMESTRE:	
FECHA DE PRESENTACIÓN	10/24	HORA DE PRESENTACIÓN			
INTEGRANTE (s)	David Flores Silva			NOTA (0-20)	
DOCENTE(s):					

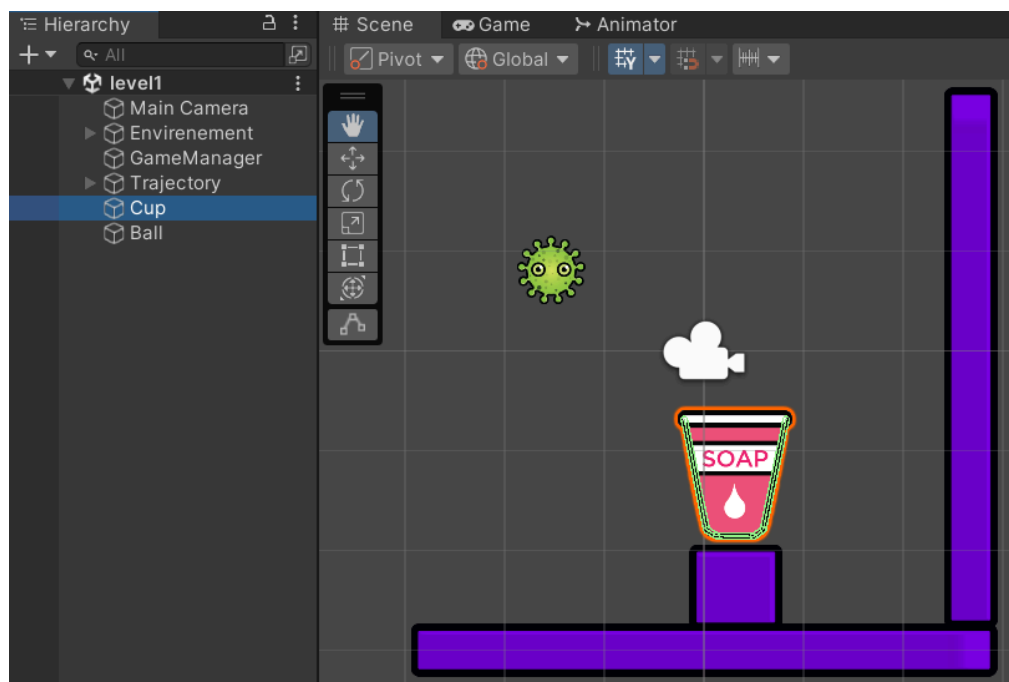
RESULTADOS Y PRUEBAS
<p>I. EJERCICIOS RESUELTOS:</p> <p>1. Explique el funcionamiento de los métodos de la estructura de clases del videojuego.</p> <p>Escenario:</p> <div data-bbox="145 1236 935 1834">  </div> <p>Como podemos ver en la imagen, este es nuestro escenario inicial del juego lo que nosotros buscamos o el objetivo del juego es que la pelotita podamos insertarla al vaso, como si fuera un juego de basketball, manejamos la misma idea para el juego. Lo que nosotros buscamos es aplicar físicas de trayectoria, la de caída libre y movimiento parabólico mediante la fórmula aplicada en un script.</p>

Muros



Los muros cumplen una función importante porque mediante estos se genera un efecto rebota en la pelota, lo que podemos utilizar para ayudarnos a cumplir el objetivo

Vaso



El vaso es nuestro objetivo del juego, cada vez que se meta la pelota dentro del vaso finaliza el juego

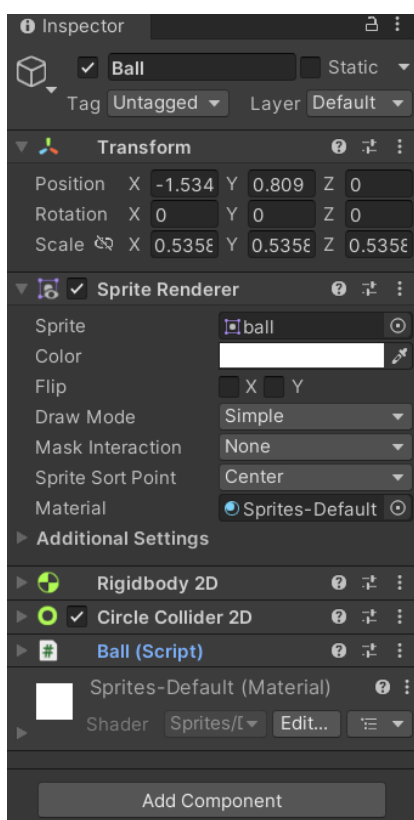


Por acá podemos ver las características que se le agregó al vaso, como se evidencia tiene un sprite renderer el cual le permite tener el material definido para dicho vaso. También el rigidbody de 2d para que tenga los efecto de gravedad y el edge collider 2d para que tenga los efectos de colisiones.

Pelota

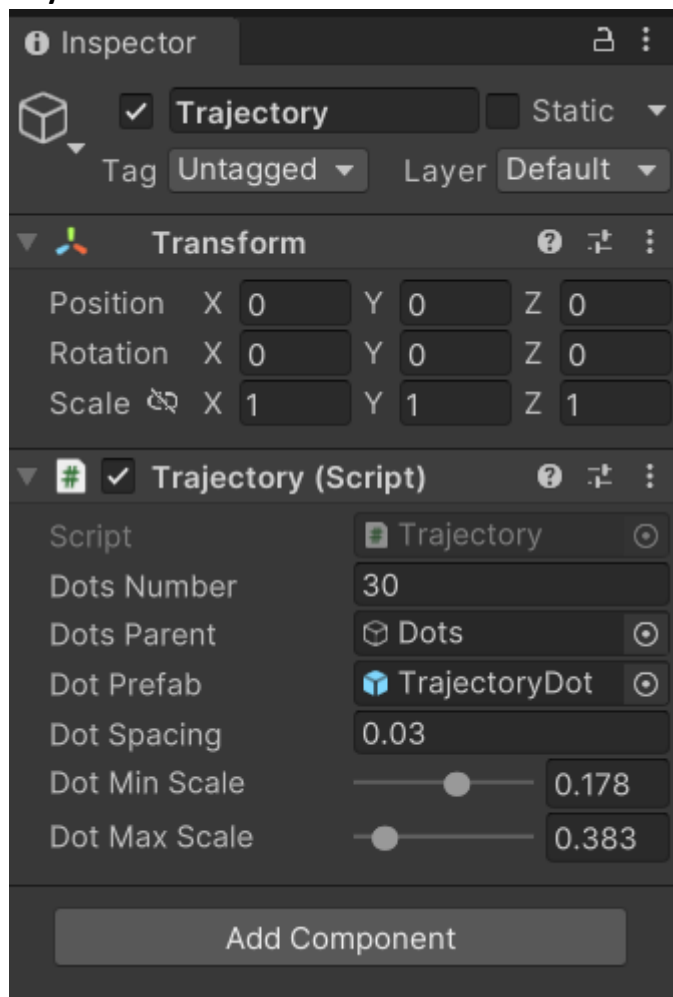


La pelota cumple una función especial ya que aquí es donde nosotros ponemos toda las físicas que se van a aplicar, como la trayectoria.



Similar a todos los componentes agregados al vaso, de la misma forma tenemos para la pelota aunq le agregamos un script donde están definidas las diferentes físicas

Trayectoria



La trayectoria es una de las físicas que acoplamos a la pelota, en este caso también agregamos un script para nuestra trayectoria con diferentes atributos los cuales nos permiten hacer esas físicas para el juego.

SCRIPT

#Game

```
GameManager.cs
1
2 using UnityEngine;
3
4 public class GameManager : MonoBehaviour
5 {
6     #region Singleton class: GameManager
7
8     public static GameManager Instance;
9
10    void Awake ()
11    {
12        if (Instance == null) {
13            Instance = this;
14        }
15    }
16
17    #endregion
18
19    Camera cam;
20
21    public Ball ball;
22    public Trajectory trajectory;
23    [SerializeField] float pushForce = 4f;
24
25    bool isDragging = false;
26
27    Vector2 startPoint;
28    Vector2 endPoint;
29    Vector2 direction;
30    Vector2 force;
31    float distance;
32
33    //-----
34    void Start ()
35    {
36        cam = Camera.main;
37        ball.DesactivateRb ();
```

```
38     }
39
40     void Update ()
41     {
42         if (Input.GetMouseButtonDown (0)) {
43             isDragging = true;
44             OnDragStart ();
45         }
46         if (Input.GetMouseButtonUp (0)) {
47             isDragging = false;
48             OnDragEnd ();
49         }
50
51         if (isDragging) {
52             OnDrag ();
53         }
54     }
55
56     //-Drag-----
57     void OnDragStart ()
58     {
59         ball.DesactivateRb ();
60         startPoint = cam.ScreenToWorldPoint (Input.mousePosition);
61
62         trajectory.Show ();
63     }
64
65     void OnDrag ()
66     {
67         endPoint = cam.ScreenToWorldPoint (Input.mousePosition);
68         distance = Vector2.Distance (startPoint, endPoint);
69         direction = (startPoint - endPoint).normalized;
70         force = direction * distance * pushForce;
71     }
```

```
72 //just for debug
73 Debug.DrawLine (startPoint, endPoint);
74
75
76 trajectory.UpdateDots (ball.pos, force);
77 }
78
79 void OnDragEnd ()
80 {
81 //push the ball
82 ball.ActivateRb ();
83
84 ball.Push (force);
85
86 trajectory.Hide ();
87 }
88
89 }
90
```

#Ball


```
1
2  using UnityEngine;
3
4  public class Ball : MonoBehaviour
5  {
6      [HideInInspector] public Rigidbody2D rb;
7      [HideInInspector] public CircleCollider2D col;
8
9      [HideInInspector] public Vector3 pos { get { return transform.position; } }
10
11     void Awake ()
12     {
13         rb = GetComponent<Rigidbody2D> ();
14         col = GetComponent<CircleCollider2D> ();
15     }
16
17     public void Push (Vector2 force)
18     {
19         rb.AddForce (force, ForceMode2D.Impulse);
20     }
21
22     public void ActivateRb ()
23     {
24         rb.isKinematic = false;
25     }
26
27     public void DesactivateRb ()
28     {
29         rb.velocity = Vector3.zero;
30         rb.angularVelocity = 0f;
31         rb.isKinematic = true;
32     }
33 }
34
```

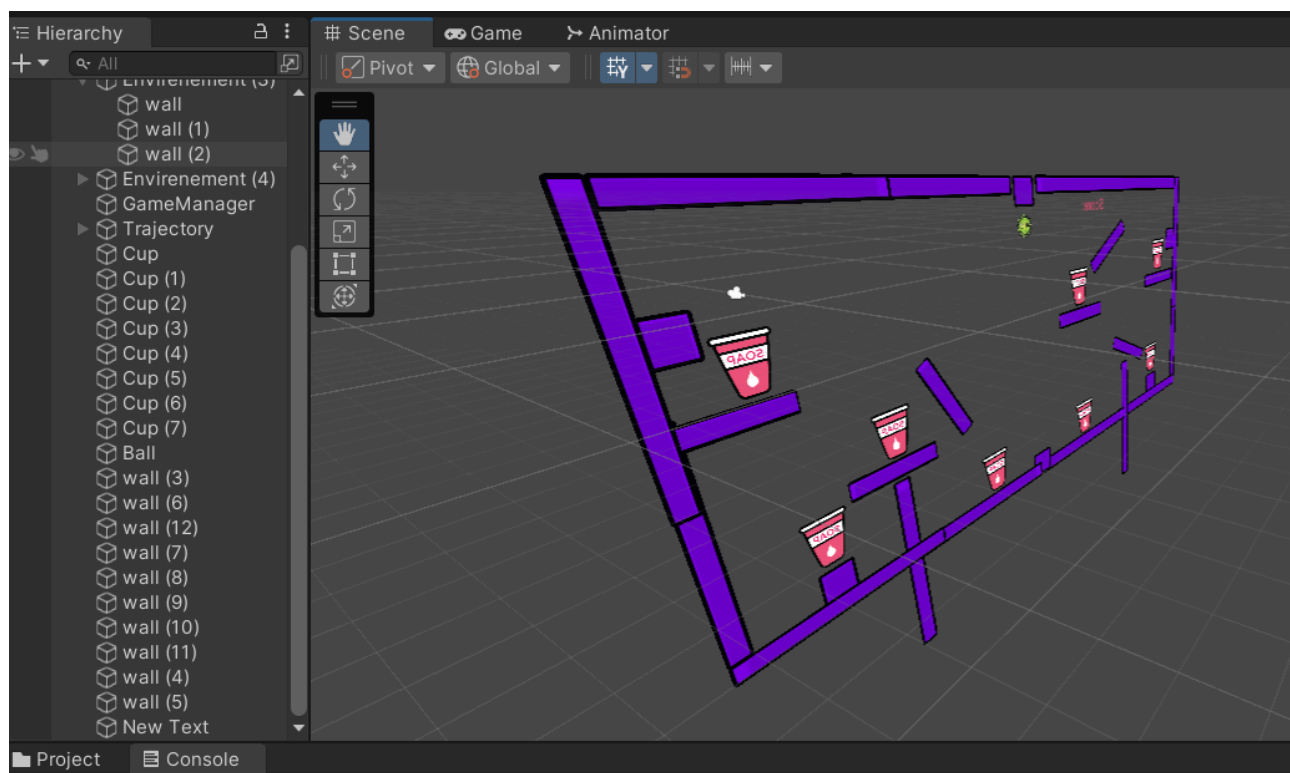
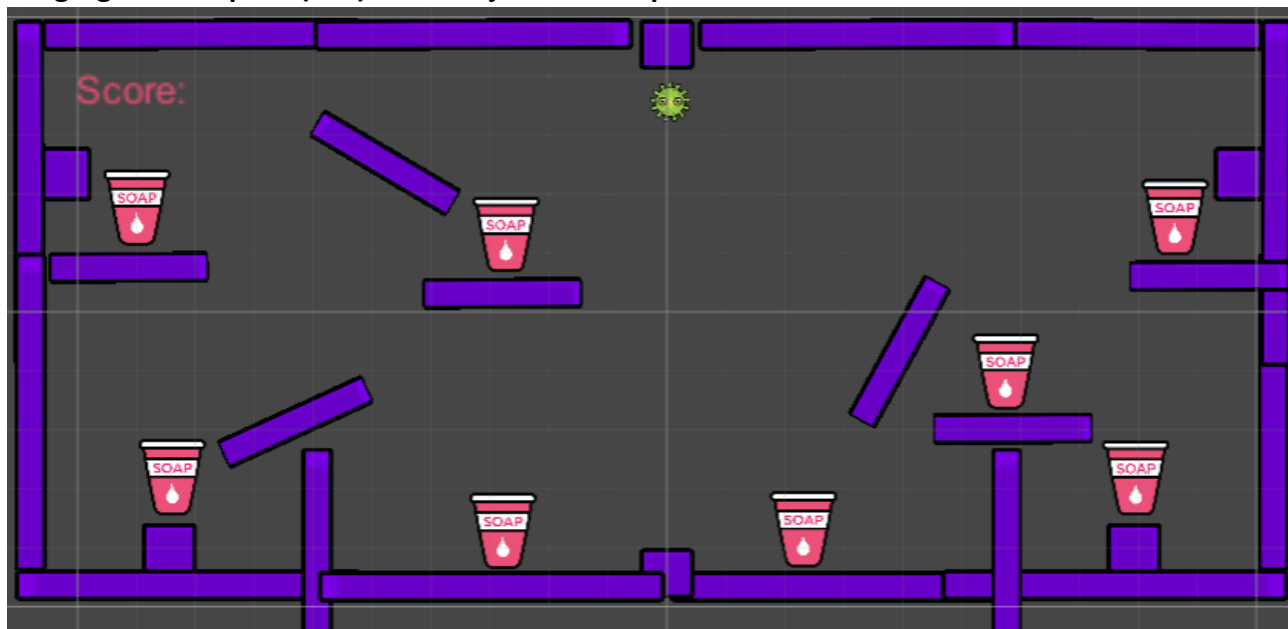
#Trayectoria

C# Trajectory.cs

```
1
2 using UnityEngine;
3
4 public class Trajectory : MonoBehaviour
5 {
6     [SerializeField] int dotsNumber;
7     [SerializeField] GameObject dotsParent;
8     [SerializeField] GameObject dotPrefab;
9     [SerializeField] float dotSpacing;
10    [SerializeField] [Range (0.01f, 0.3f)] float dotMinScale;
11    [SerializeField] [Range (0.3f, 1f)] float dotMaxScale;
12
13    Transform[] dotsList;
14
15    Vector2 pos;
16    //dot pos
17    float timeStamp;
18
19    //-----
20    void Start ()
21    {
22        //hide trajectory in the start
23        Hide ();
24        //prepare dots
25        PrepareDots ();
26    }
27
28    void PrepareDots ()
29    {
30        dotsList = new Transform[dotsNumber];
31        dotPrefab.transform.localScale = Vector3.one * dotMaxScale;
32
33        float scale = dotMaxScale;
34        float scaleFactor = scale / dotsNumber;
35
36        for (int i = 0; i < dotsNumber; i++) {
```

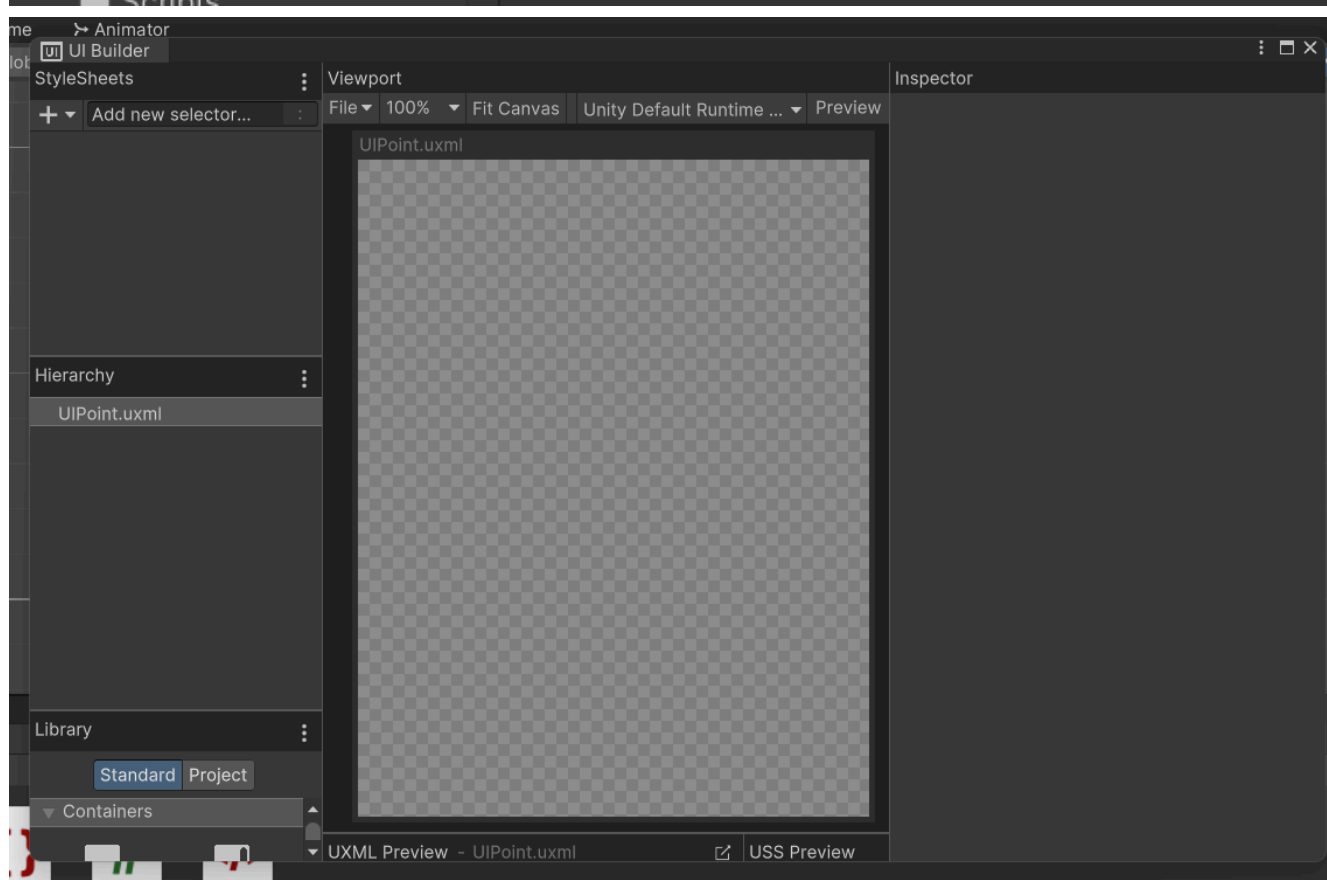
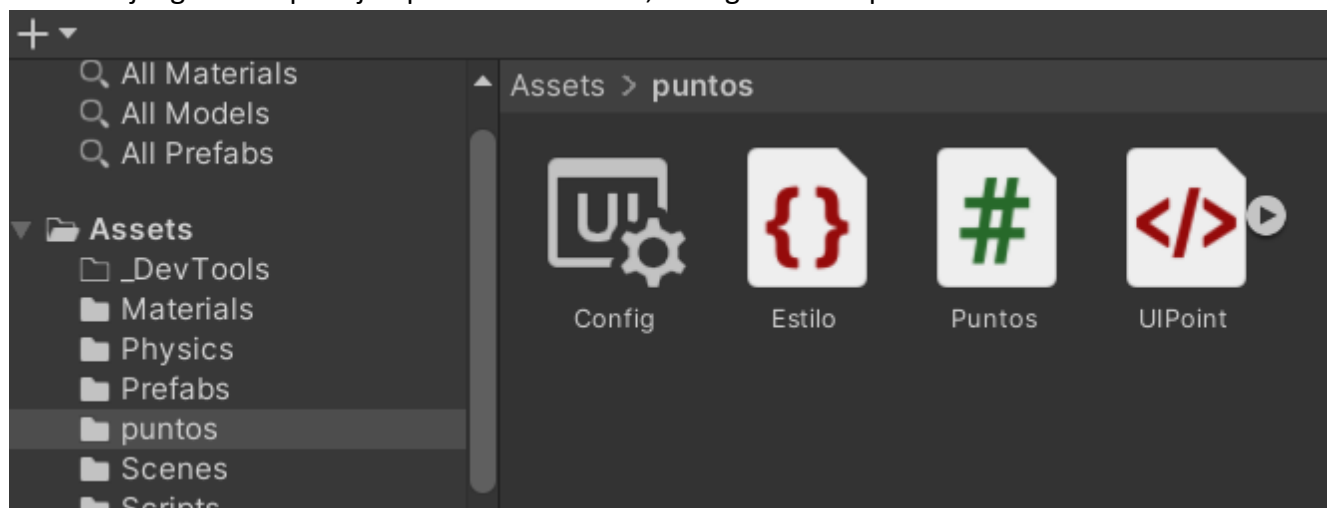
```
35
36     for (int i = 0; i < dotsNumber; i++) {
37         dotsList [i] = Instantiate (dotPrefab, null).transform;
38         dotsList [i].parent = dotsParent.transform;
39
40         dotsList [i].localScale = Vector3.one * scale;
41         if (scale > dotMinScale)
42             scale -= scaleFactor;
43     }
44 }
45
46 public void UpdateDots (Vector3 ballPos, Vector2 forceApplied)
47 {
48     timeStamp = dotSpacing;
49     for (int i = 0; i < dotsNumber; i++) {
50         pos.x = (ballPos.x + forceApplied.x * timeStamp);
51         pos.y = (ballPos.y + forceApplied.y * timeStamp) - (Physics2D.gravity.magnitude * timeStamp * timeStamp);
52
53         //you can simplify this 2 lines at the top by:
54         //pos = (ballPos+force*time)-((-Physics2D.gravity*time*time)/2f);
55         //
56         //but make sure to turn "pos" in Ball.cs to Vector2 instead of Vector3
57
58         dotsList [i].position = pos;
59         timeStamp += dotSpacing;
60     }
61 }
62
63 public void Show ()
64 {
65     dotsParent.SetActive (true);
66 }
67
68 public void Hide ()
69 {
70     dotsParent.SetActive (false);
71 }
72 }
73
```

2. Agregue otro sprite (cub) como objetivo de disparo.

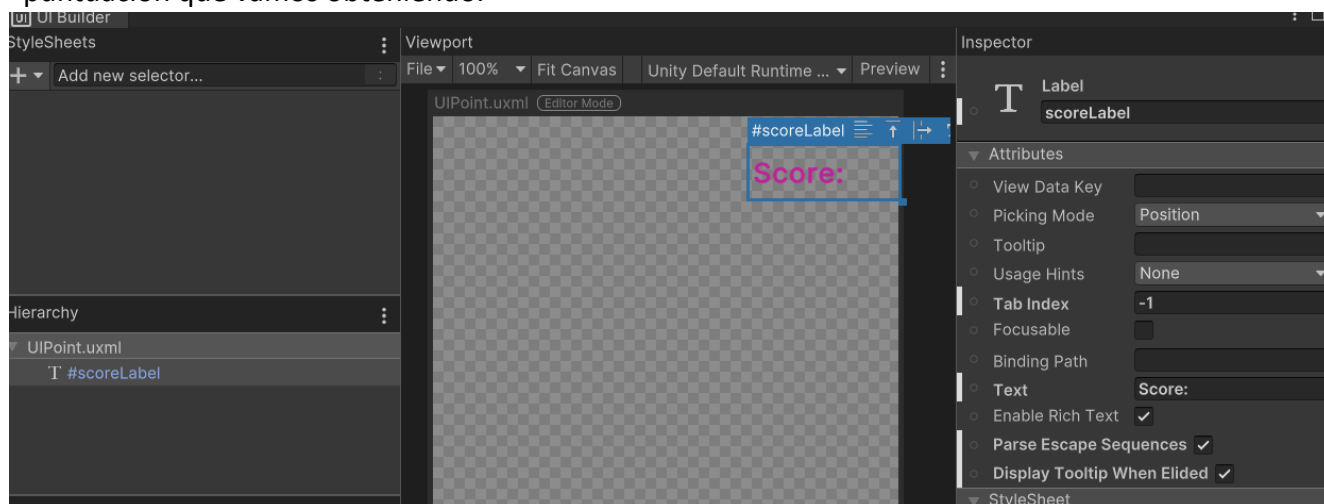


3. Agregue el conteo de puntuación cada vez que da al objetivo.

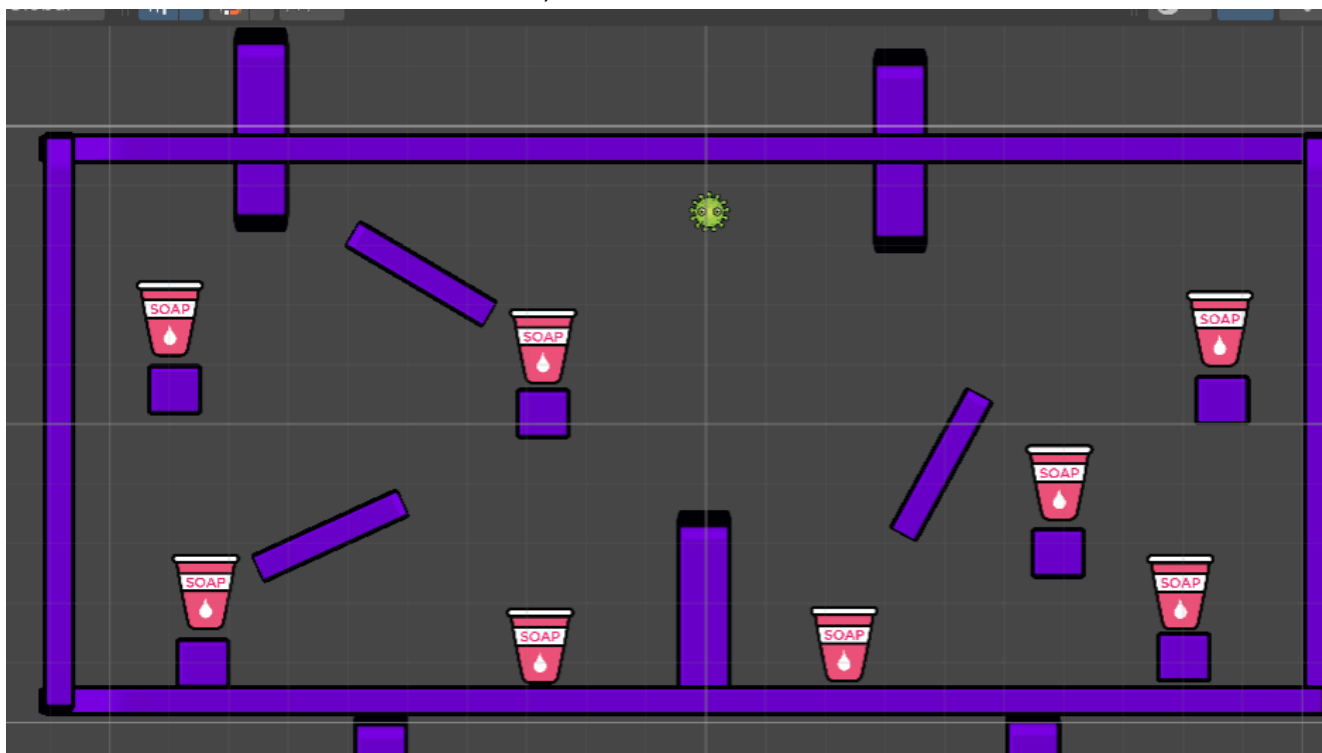
Lo primero que hacemos es crear un UIDocument esto nos permite editar interfaces para asignar a nuestro juego como por ejemplo menú de inicio, configuración o puntuaciones como nuestro caso.

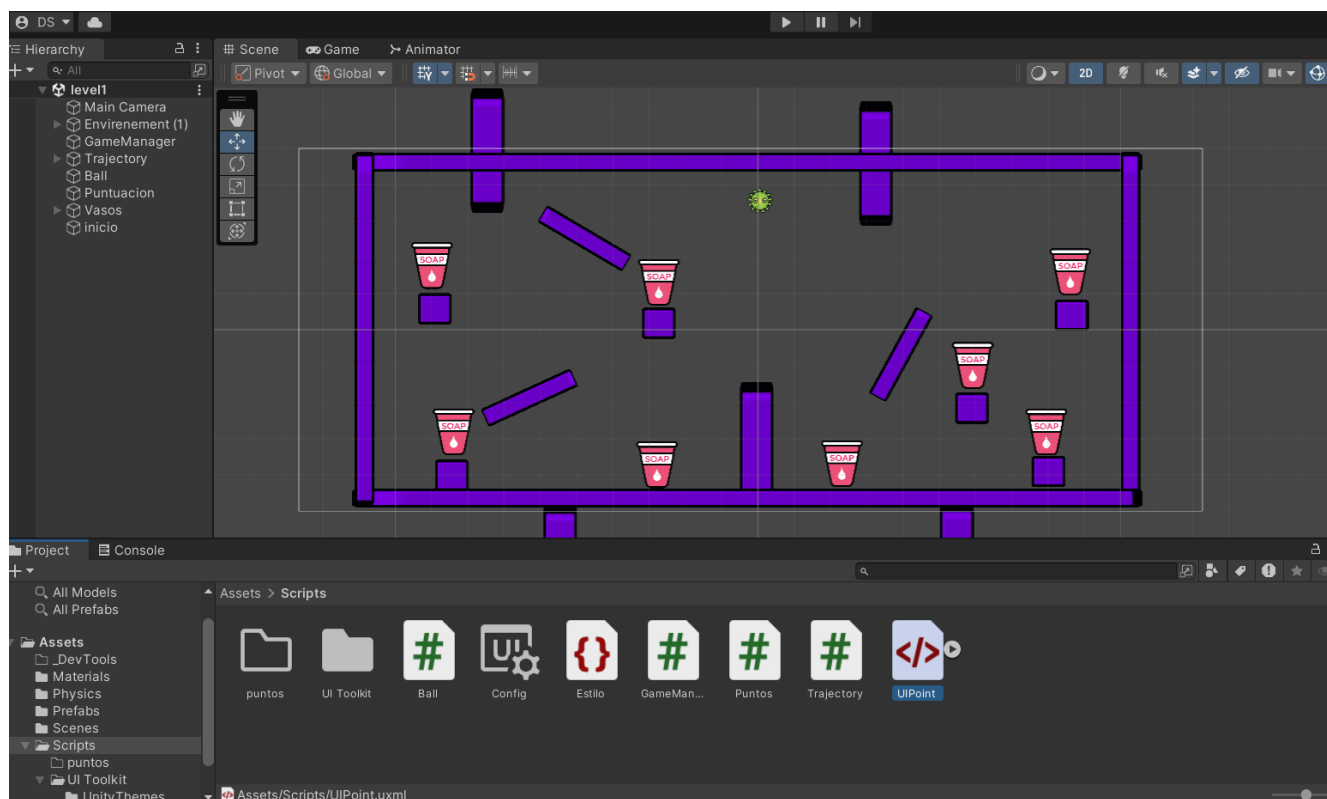


Aquí adentro vamos a colocar un text label el cual toma la descripción de score el cual informara la puntuación que vamos obteniendo.



Ahora configuramos nuestro escenario con una interacción de más vasos y paredes que nos permita simular diferentes físicas como el rebote, o la fuerza.





También hemos creado algún script importantes como la notificación del script de ball y la creación del script score.

SCRIPT BALL.CS

```
Scripts > Ball.cs > ...
1  using UnityEngine;
2
3  0 references
4  public class Ball : MonoBehaviour
5  {
6      8 references
7      [HideInInspector] public Rigidbody2D rb;
8      2 references
9      [HideInInspector] public CircleCollider2D col;
10
11     0 references
12     [HideInInspector] public Vector3 pos { get { return transform.position; } }
13
14     2 references
15     public Puntos puntos; // Referencia a tu script Puntos
16
17     3 references
18     public Transform centerPosition; // La posición inicial o centro
19 }
```

```
14  ✓ void Awake()
15      {
16          // Inicializar componentes
17          rb = GetComponent<Rigidbody2D>();
18          col = GetComponent<CircleCollider2D>();
19
20          // Validar que los componentes existen
21  ✓   if (rb == null)
22      {
23          Debug.LogError("No se encontró Rigidbody2D en la pelota.");
24      }
25
26  ✓   if (col == null)
27      {
28          Debug.LogError("No se encontró CircleCollider2D en la pelota.");
29      }
30
31      // Validar que la posición central no sea nula
32  ✓   if (centerPosition == null)
33      {
34          Debug.LogError("No se ha asignado 'centerPosition' en el inspector.");
35      }
36
37      // Validar que el script Puntos no sea nulo
38  ✓   if (puntos == null)
39      {
40          Debug.LogError("No se ha asignado el script 'Puntos' en el inspector.");
41      }
42  }
43
44  0 references
45  ✓ void Start()
46      {
47          // Coloca la pelota en el centro al inicio
48          ResetPosition();
49      }
```



```
3 public class Ball : MonoBehaviour
50 void OnTriggerEnter2D(Collider2D other)
51 {
52     // Verificar si el objeto con el que colisionó tiene el tag "Vaso"
53     if (other.CompareTag("Vaso"))
54     {
55         // Incrementar el puntaje utilizando el script Puntos
56         puntos.IncreaseScore();
57
58         // Reiniciar la posición de la pelota al centro
59         ResetPosition();
60     }
61 }
62
63 2 references
64 void ResetPosition()
65 {
66     // Verificar que la posición central no sea nula
67     if (centerPosition == null) return;
68
69     // Desactivar la física momentáneamente
70     DesactivateRb();
71
72     // Coloca la pelota en la posición definida como el centro
73     transform.position = centerPosition.position;
74
75     // Reactivar la física (si es necesario)
76     ActivateRb();
77 }
78
79 0 references
80 public void Push(Vector2 force)
81 {
82     if (rb.isKinematic == false)
83     {
84         rb.AddForce(force, ForceMode2D.Impulse);
85     }
86 }
```

```
3    public class Ball : MonoBehaviour
78    public void Push(Vector2 force)
83    {
84    else
85    {
86        Debug.LogWarning("El Rigidbody está en modo Kinematic.
87    }
88    }
89
90    1 reference
90    public void ActivateRb()
91    {
92        rb.isKinematic = false;
93    }
94
95    1 reference
95    public void DesactivateRb()
96    {
97        rb.velocity = Vector3.zero;
98        rb.angularVelocity = 0f;
99        rb.isKinematic = true;
100    }
101 }
102
```

SCRIPT SCORE.CS

```
1  using UnityEngine;
2  using UnityEngine.UIElements;
3
4  0 references
5  public class Puntos : MonoBehaviour
6  {
7      3 references
8      private Label scoreLabel;
9      2 references
10     private int score = 0;
11
12     0 references
13     void OnEnable()
14     {
15         // Obtener el componente VisualElement del UI Document
16         var root = GetComponent<UIDocument>().rootVisualElement;
17         // Asignar el Label usando el nombre que le diste
18         scoreLabel = root.Q<Label>("scoreLabel");
19         // Actualizar el texto inicial
20         UpdateScore();
21     }
22
23     0 references
24     public void IncreaseScore()
25     {
26         score++; // Aumentar el puntaje
27         UpdateScore(); // Actualizar la UI
28     }
29
30     private void UpdateScore()
31     {
32         if (scoreLabel != null)
33         {
34             scoreLabel.text = "Score: " + score.ToString();
35         }
36         else
37         {
38             Debug.LogWarning("No se encontró el Label 'scoreLabel'.");
39         }
40     }
41 }
```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 20</p>

II. CUESTIONARIO:

1. ¿Cómo funciona el procedimiento de trayectoria de un proyectil?

La trayectoria de un proyectil es la curva que describe su movimiento en el espacio debido a las fuerzas que actúan sobre él, principalmente la gravedad. En un contexto físico básico, la trayectoria sigue las leyes de la **cinemática** y la **dinámica** en un sistema bidimensional o tridimensional.

Fórmula general de la trayectoria:

$$y(t) = y_0 + v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2}gt^2$$

2. ¿Qué elementos proporciona Unity para apoyar a la implementación de la trayectoria de proyectiles en videojuegos?

Rigidbody y su Configuración:

El componente Rigidbody en Unity permite a los objetos interactuar con la física, como la gravedad y las fuerzas, lo cual es esencial para proyectiles. Algunas propiedades relevantes:

- Mass: La masa del objeto, que afecta cómo responde a las fuerzas.
- Drag: Resistencia al movimiento lineal.
- Angular Drag: Resistencia al movimiento angular.
- Gravity: Activar o desactivar el efecto de la gravedad en el objeto.
- Use Gravity: Si se marca, el objeto será afectado por la gravedad.

Scripts de Movimiento y Componentes Cinemáticos:

Unity permite un control completo de los proyectiles mediante scripts personalizados. Puedes usar un script para aplicar física, controlar velocidades, manejar colisiones, y actualizar la posición y rotación del objeto.

Métodos para Aplicar Fuerzas:

- AddForce: Aplica una fuerza continua al objeto, puede ser ForceMode.Impulse para un impulso instantáneo o ForceMode.Force para aplicar una fuerza constante.
- AddTorque: Aplica una fuerza de rotación (torque) al objeto.

CÓDIGO**Ball.cs**

```
using UnityEngine;

public class Ball : MonoBehaviour
{
    [HideInInspector] public Rigidbody2D rb;
    [HideInInspector] public CircleCollider2D col;

    [HideInInspector] public Vector3 pos { get { return transform.position; } }

    public Puntos puntos; // Referencia a tu script Puntos

    public Transform centerPosition; // La posición inicial o centro

    void Awake()
    {
        // Inicializar componentes
        rb = GetComponent<Rigidbody2D>();
        col = GetComponent<CircleCollider2D>();

        // Validar que los componentes existen
        if (rb == null)
        {
            Debug.LogError("No se encontró Rigidbody2D en la pelota.");
        }

        if (col == null)
        {
            Debug.LogError("No se encontró CircleCollider2D en la pelota.");
        }

        // Validar que la posición central no sea nula
        if (centerPosition == null)
        {
            Debug.LogError("No se ha asignado 'centerPosition' en el inspector.");
        }
    }
}
```

```
}

// Validar que el script Puntos no sea nulo
if (puntos == null)
{
    Debug.LogError("No se ha asignado el script 'Puntos' en el
inspector.");
}
}

void Start()
{
    // Coloca la pelota en el centro al inicio
    ResetPosition();
}

void OnTriggerEnter2D(Collider2D other)
{
    // Verificar si el objeto con el que colisionó tiene el tag "Vaso"
    if (other.CompareTag("Vaso"))
    {
        // Incrementar el puntaje utilizando el script Puntos
        puntos.IncreaseScore();

        // Reiniciar la posición de la pelota al centro
        ResetPosition();
    }
}

void ResetPosition()
{
    // Verificar que la posición central no sea nula
    if (centerPosition == null) return;

    // Desactivar la física momentáneamente
    DeactivateRb();

    // Coloca la pelota en la posición definida como el centro
```

```
transform.position = centerPosition.position;

// Reactivar la física (si es necesario)
ActivateRb();
}

public void Push(Vector2 force)
{
    if (rb.isKinematic == false)
    {
        rb.AddForce(force, ForceMode2D.Impulse);
    }
    else
    {
        Debug.LogWarning("El Rigidbody está en modo Kinematic. No se puede
aplicar la fuerza.");
    }
}

public void ActivateRb()
{
    rb.isKinematic = false;
}

public void DesactivateRb()
{
    rb.velocity = Vector3.zero;
    rb.angularVelocity = 0f;
    rb.isKinematic = true;
}
}
```

Puntos.cs

```
using UnityEngine;
using UnityEngine.UIElements;

public class Puntos : MonoBehaviour
{
    private Label scoreLabel;
    private int score = 0;

    void OnEnable()
    {
        // Obtener el componente VisualElement del UI Document
        var root = GetComponent<UIDocument>().rootVisualElement;
        // Asignar el Label usando el nombre que le diste
        scoreLabel = root.Q<Label>("scoreLabel");
        // Actualizar el texto inicial
        UpdateScore();
    }

    public void IncreaseScore()
    {
        score++; // Aumentar el puntaje
        UpdateScore(); // Actualizar la UI
    }

    private void UpdateScore()
    {
        if (scoreLabel != null)
        {
            scoreLabel.text = "Score: " + score.ToString();
        }
        else
        {
            Debug.LogWarning("No se encontró el Label 'scoreLabel'. Asegúrate de que el nombre esté bien escrito.");
        }
    }
}
```


GameManager.cs

```
using UnityEngine;

public class GameManager : MonoBehaviour
{
    #region Singleton class: GameManager

    public static GameManager Instance;

    void Awake ()
    {
        if (Instance == null) {
            Instance = this;
        }
    }

    #endregion

    Camera cam;

    public Ball ball;
    public Trajectory trajectory;
    [SerializeField] float pushForce = 4f;

    bool isDragging = false;

    Vector2 startPoint;
    Vector2 endPoint;
    Vector2 direction;
    Vector2 force;
    float distance;

    //-----
    void Start ()
    {
        cam = Camera.main;
        ball.DesactivateRb ();
    }
}
```

```
}

void Update ()
{
    if (Input.GetMouseButtonDown (0)) {
        isDragging = true;
        OnDragStart ();
    }
    if (Input.GetMouseButtonUp (0)) {
        isDragging = false;
        OnDragEnd ();
    }

    if (isDragging) {
        OnDrag ();
    }
}

// - Drag -----
void OnDragStart ()
{
    ball.DesactivateRb ();
    startPoint = cam.ScreenToWorldPoint (Input.mousePosition);

    trajectory.Show ();
}

void OnDrag ()
{
    endPoint = cam.ScreenToWorldPoint (Input.mousePosition);
    distance = Vector2.Distance (startPoint, endPoint);
    direction = (startPoint - endPoint).normalized;
    force = direction * distance * pushForce;

    // just for debug
    Debug.DrawLine (startPoint, endPoint);
}
```

```
trajectory.UpdateDots (ball.pos, force);  
}  
  
void OnDragEnd ()  
{  
    //push the ball  
    ball.ActivateRb ();  
  
    ball.Push (force);  
  
    trajectory.Hide ();  
}  
}
```

Trayectory.cs

```
using UnityEngine;  
  
public class Trajectory : MonoBehaviour  
{  
    [SerializeField] int dotsNumber;  
    [SerializeField] GameObject dotsParent;  
    [SerializeField] GameObject dotPrefab;  
    [SerializeField] float dotSpacing;  
    [SerializeField] [Range (0.01f, 0.3f)] float dotMinScale;  
    [SerializeField] [Range (0.3f, 1f)] float dotMaxScale;  
  
    Transform[] dotsList;  
  
    Vector2 pos;  
    //dot pos  
    float timeStamp;  
  
    //-----  
    void Start ()
```

```
{
    //hide trajectory in the start
    Hide ();
    //prepare dots
    PrepareDots ();
}

void PrepareDots ()
{
    dotsList = new Transform[dotsNumber];
    dotPrefab.transform.localScale = Vector3.one * dotMaxScale;

    float scale = dotMaxScale;
    float scaleFactor = scale / dotsNumber;

    for (int i = 0; i < dotsNumber; i++) {
        dotsList [i] = Instantiate (dotPrefab, null).transform;
        dotsList [i].parent = dotsParent.transform;

        dotsList [i].localScale = Vector3.one * scale;
        if (scale > dotMinScale)
            scale -= scaleFactor;
    }
}

public void UpdateDots (Vector3 ballPos, Vector2 forceApplied)
{
    timeStamp = dotSpacing;
    for (int i = 0; i < dotsNumber; i++) {
        pos.x = (ballPos.x + forceApplied.x * timeStamp);
        pos.y = (ballPos.y + forceApplied.y * timeStamp) -
(Physics2D.gravity.magnitude * timeStamp * timeStamp) / 2f;

        //you can simplify this 2 lines at the top by:
//pos = (ballPos+force*time)-((-Physics2D.gravity*time*time)/2f);
//
//but make sure to turn "pos" in Ball.cs to Vector2 instead of Vector3
```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 29</p>

```

dotsList [i].position = pos;
timeStamp += dotSpacing;
}
}

public void Show ()
{
dotsParent.SetActive (true);
}

public void Hide ()
{
dotsParent.SetActive (false);
}
}

```

REFERENCIAS Y BIBLIOGRAFÍA

- Millington, I., & Funge, J. (2019). *Artificial intelligence for games. Third Ed.* CRC Press.
- <https://learn.unity.com/tutorial/calculatingtrajectories?uv=2019.4&courseId=5dd851beedbc2a1bf7b72bed&projectId=5df2611eedbc2a0020d90217#5e0a7bcb edbc2a317c8c91af>
- GIT HUB: