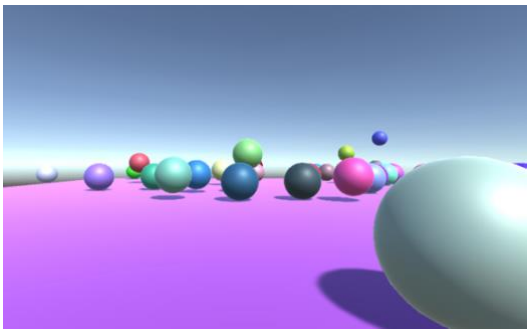
Documentación de animaciones segunda semana
Diógenes Grajales Corona A01653251

Spawner de esferas

Se utilizó un script capaz de crear un determinado número de esferas con una posición y color random. Se coloca un plano con un material físico en una escena y un game object vacío con el script generador de esferas. Al correr la escena, se obtiene un resultado como el siguiente:



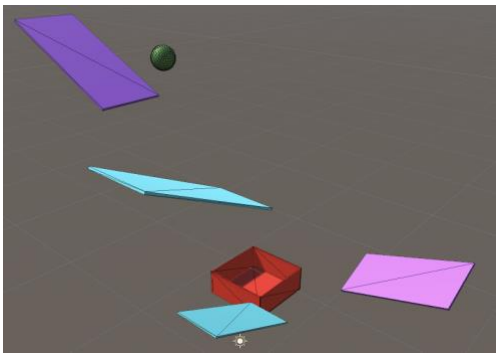
Código:

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

public class CreateSpheres : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        for (int i = 0; i < 100; i++) {

            float px = Random.Range(0.0f,10.0f);
            float py = Random.Range(0.0f, 10.0f);
            float pz = Random.Range(0.0f, 10.0f);
            Vector3 p = new Vector3(px,py,pz);


            float cR = Random.Range(0.0f,1.0f);
            float cG = Random.Range(0.0f, 1.0f);
            float cB = Random.Range(0.0f, 1.0f);
            Color c = new Color(cR,cG,cB);
            createSphere(p,c);


        }
```

```
    }

    GameObject createSphere(Vector3 position, Color color) {

        GameObject mySphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        mySphere.transform.localPosition = position;
        //mySphere.transform.localPosition = gameObject.transform.position;

        Renderer rend = mySphere.GetComponent<Renderer>();
        rend.material = new Material(Shader.Find("Standard"));
        rend.material.SetColor("_Color", color);


        Rigidbody rb = mySphere.AddComponent<Rigidbody>();
        rb.mass = 2;

        PhysicMaterial mat = new PhysicMaterial();
        mat.bounciness = 1.0f;
        Collider collider = mySphere.GetComponent<Collider>();
        collider.material = mat;

        return mySphere;

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Juego de la pelota

Utilizando el mismo principio que la escena anterior, se necesitan planos con un material físico con un coeficiente de rebote y estos serán colocados en un circuito capaz de llevar una pelota a su meta. La pelota será generada con un script muy similar al anterior, los únicos cambios serán que únicamente se generará una pelota y la posición ya no será random, sino que su posición será la misma del game object vacio al cual se le asigne el script, de este modo, controlaremos exactamente donde pondremos la pelota.



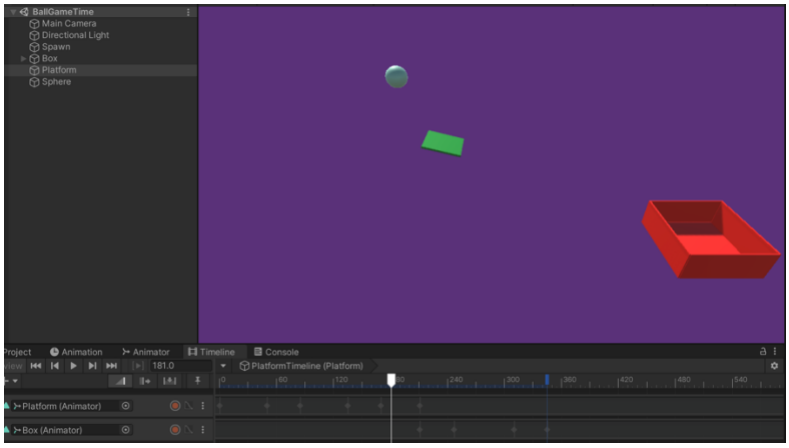
Código:

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

public class BallSpawn : MonoBehaviour
{
   // Start is called before the first frame update
   public int ballmass = 2;
   public float bounciness = 1.0f;

   void Start()
   {
      float cR = Random.Range(0.0f, 1.0f);
      float cG = Random.Range(0.0f, 1.0f);
      float cB = Random.Range(0.0f, 1.0f);
      Color c = new Color(cR, cG, cB);
      createSphere(c);
   }

   GameObject createSphere(Color color)
   {

      GameObject mySphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
      mySphere.transform.localPosition = gameObject.transform.position;

      Renderer rend = mySphere.GetComponent<Renderer>();
      rend.material = new Material(Shader.Find("Standard"));
      rend.material.SetColor("_Color", color);


      Rigidbody rb = mySphere.AddComponent<Rigidbody>();
      rb.mass = ballmass;

      PhysicMaterial mat = new PhysicMaterial();
```

```
        mat.bounciness = bounciness;
        Collider collider = mySphere.GetComponent<Collider>();
        collider.material = mat;

        return mySphere;

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

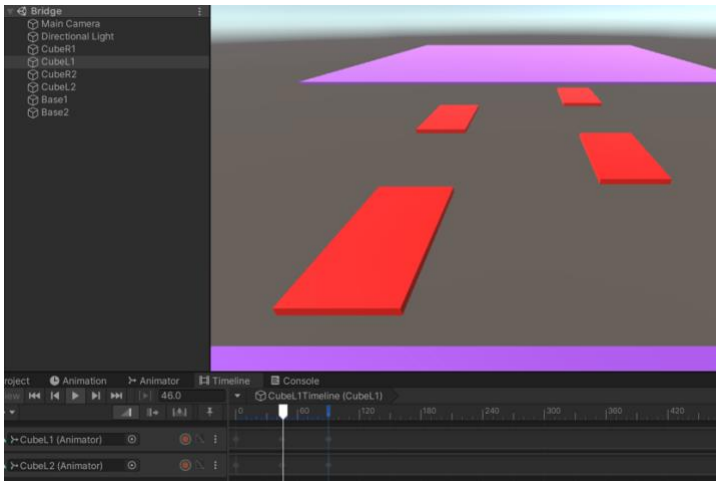
Juego de la pelota con Timeline

Será una escena y código prácticamente igual al anterior. La diferencia radica en el Timeline, este se puede obtener mediante la pestaña de Window en Unity y seleccionando Timeline. Se agrega un objeto de la escena al timeline para crear un controlador y se podrá modificar sus propiedades a lo largo de la duración de la animación. En este caso, una sola plataforma hará el trabajo de las 4 que se utilizaron en el anterior ejemplo, la diferencia es que esta cambiará su posición para rebotar la pelota hasta la meta



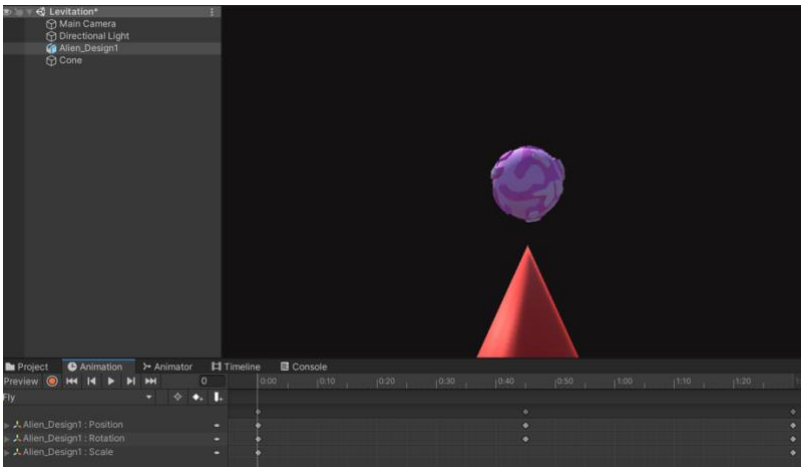
Puente

En este caso, el único objetivo es crear una escena con un puente movible. Para ello, solo necesitaremos 4 plataformas y una de ellas será seleccionada para hacer un controlador en el timeline. Las demás plataformas serán agregadas a la timeline como animation tracks y se podrá controlar cada una de ellas para hacer el efecto del puente movible



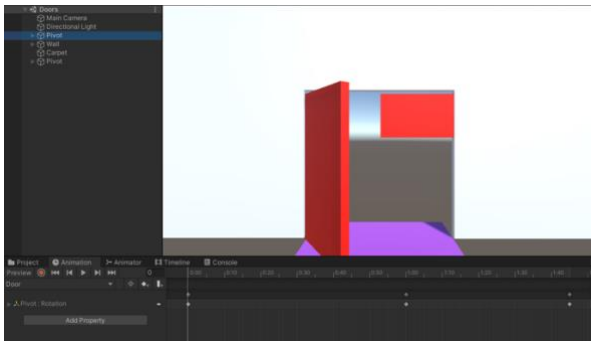
Levitación

Esta animación se creó con la herramienta Animation la cual puede ser obtenida desde la pestaña de Window en el Editor de Unity. Esta funciona prácticamente igual al Timeline con la diferencia de que controla únicamente un objeto por controlador. Igualmente, se pueden modificar todas las propiedades de dicho objeto a lo largo de la simulación y dar el efecto de levitación



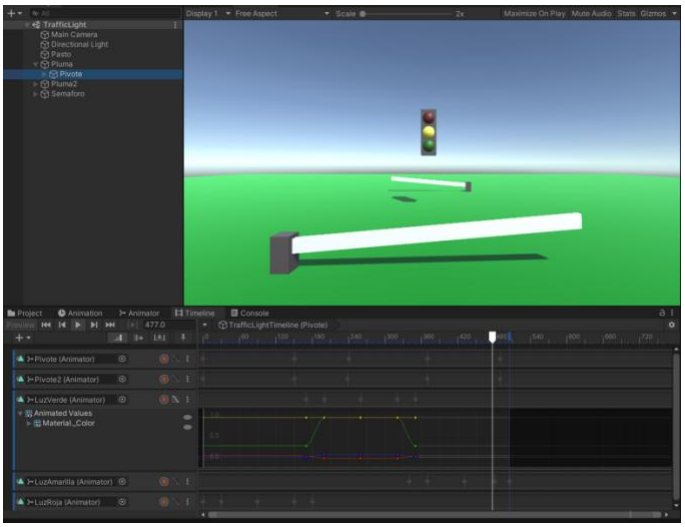
Puertas

La siguiente animación también se generó con la herramienta Animation. Se imita el comportamiento de un timeline al crear un controlador de animation para cada una de las puertas. Para generar el movimiento de una puerta, se tiene que asignar como hijo a un objeto vacío que será colocado en la posición sobre la cual rotará la puerta, es decir, será su pivote. Si no se hiciera esto y solo se rotará la puerta directamente, esta giraría sobre su propio eje. Al final, lo que rotará en la herramienta de Animation no será el objeto de la puerta, sino el pivote. De igual manera, se pueden controlar otras propiedades como la escala en esta herramienta.



Semáforo

Para crear esta animación, se utilizó el mismo proceso que en la del puente. Todo está siendo controlado mediante el timeline y se utilizaron los principios del pivote de las puertas para levantar las plumas de la escena. Igualmente, se controlaron otras propiedades como el color de los materiales para lograr el efecto del semáforo. En la tiemline, se puede sincronizar todo lo que pasará en la animación



Animación de Cubo

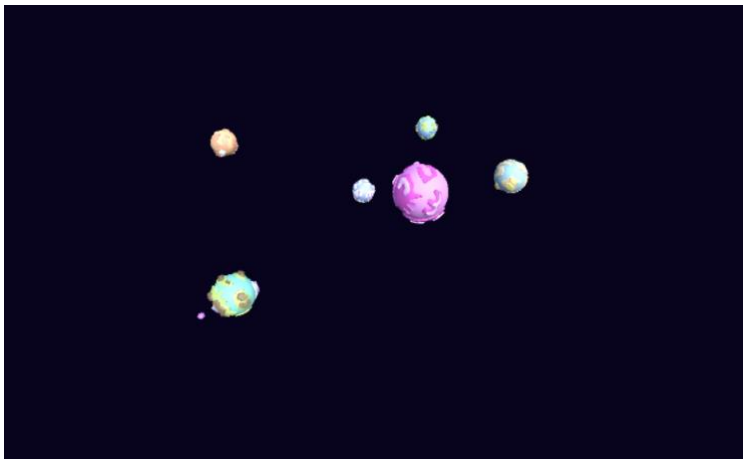
Esta animación es muy simple, se hizo con la herramienta de Animation y simplemente rota un Cubo en la escena. Lo interesante consiste en que se utiliza una función de Animation conocida como evento, esto hace que se pueda correr una función de algún script asociado al objeto de la animación en un punto específico. En este caso, se trata de un código que cambia el color del objeto a uno random. Esa parte es igual a las animaciones de pelotas presentadas anteriormente y esto logra que cada vez que el cubo de un par de vueltas, este cambie su color a uno completamente nuevo.

Código:

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

public class CambioColor : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        CambiaColor();
    }

    // Update is called once per frame
    void Update()
    {

    }

    void CambiaColor() {

        float cR = Random.Range(0.0f, 1.0f);
        float cG = Random.Range(0.0f, 1.0f);
        float cB = Random.Range(0.0f, 1.0f);
        Color c = new Color(cR, cG, cB);
        Renderer rend = gameObject.GetComponent<Renderer>();
        rend.material = new Material(Shader.Find("Standard"));
        rend.material.SetColor("_Color",c);

    }
}
```

Planetas

Esta animación fue creada mediante únicamente código, ni animation ni timeline fueron utilizadas. En esta, se utilizan conceptos como el pivote de las puertas para lograr que los planetas orbiten sobre el planeta central y mediante código, se puede utilizar la función RotateAround. Para dar el efecto de rotación de los planetas, se utiliza la función localRotation. Cabe mencionar que las funciones se llamaban en el Update del código para que la animación ocurriera a cada frame mientras la escena corria.



Código para rotar sobre su propio eje:

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

public class Rotation : MonoBehaviour
{
    float x;
    float y;
    float z;

    public float speed = 20;

    void mirotacion()
    {
        x += 20 * Time.deltaTime;
        y += speed * Time.deltaTime;
        z += 20 * Time.deltaTime;

        gameObject.transform.localRotation = Quaternion.Euler(0, -y, 0);
    }
    // Start is called before the first frame update
    void Start()
    {
```

```
    }

    // Update is called once per frame
    void Update()
    {
        mirotacion();
    }
}
```

Código para que roten alrededor de otro objeto:

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

public class RotateAround : MonoBehaviour
{
    public GameObject pivot;
    public float speed = 20;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Rotate(speed);
    }

    void Rotate(float speed) {

        gameObject.transform.RotateAround(pivot.transform.position, new Vector3(0, 1, 0),
speed * Time.deltaTime);
    }
}
```