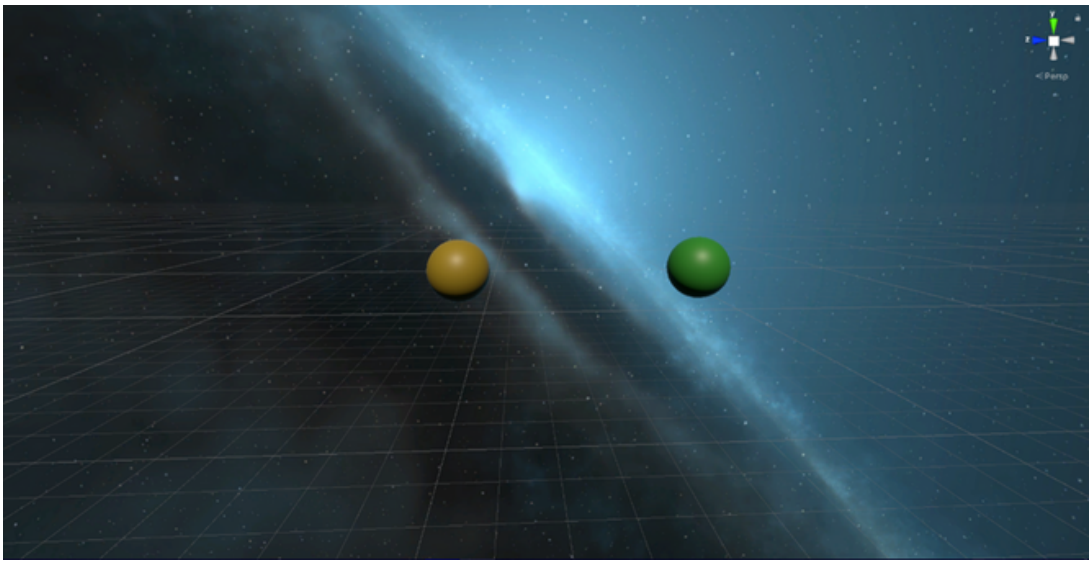
Ley de gravitación universal

Empecé este a desarrollar esta simulación a partir de mi proyecto en equipo que era tratar de imitar las físicas de Mario Galaxy en ese entonces no tenia los conceptos para desarrollar algo semejante a la simulación de la gravedad de una manera mas realista así que me di a la tarea después de la clase de gravitación a llegar a simular la ley de gravitación, me tope con varios problemas y cuestiones en esta simulación, empecemos a decir que esto es un tema aparte de mi proyecto en equipo, ya que una cosa es imitar las físicas de un juego y otra tratar de simular las interacciones de los cuerpos celestes basándonos en cantidades reales con una ley real de la física.
Empecé creando de la misma manera que en las otras escenas objetos lo mas similares a los reales para que la simulación tomara color y realismo, entonces creé dos esferas las cuales solo a una de ellas le agregue Rigidbody, y las escale lo suficiente para que se vieran mas grandes, agregue así un Skybox de espacio(es un fondo no afecta en anda), para que se viera como si estuviéramos en el espacio…



Y después empecé a crear el script básico que nos permitiera el movimiento de los cuerpos.

Era lo mismo que el de Mario Galaxy, pero esto no iba a funcionar ya que todos los planetas tenían que ejercer una gravedad, y si recordamos en el script de Mario Galaxy lo que hacia era sobre escribir en cada frame la dirección de la gravedad que viene en unity, pero esto no nos sirve entonces, lo que hice en primer lugar para simular una ingravidez fue ir a las preferencias del proyecto y en la parte de físicas colocar en el vector de la gravedad (0,0,0), de esta manera prácticamente no hay gravedad, y todos los objetos que se instancien quietos, permanecerán quietos hasta que se les ejerza un fuerza. Ahora y el core del scrip de como comencé a simular algo similar a el resultado final fue, primero el agregar como hijo de nuestro objeto otro objeto el cual tuviera una zona(Trigger) para que si otro planeta entrara y permaneciera en este se le ejerciera una fuerza tipo aceleración, porque en unity al ejercer una fuerza hay cuatro tipos…

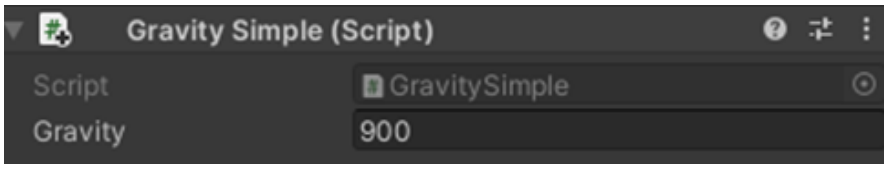
- Impulso.
- Aceleración.
- Fuerza.
- Cambio de velocidad.

¿Cual es el tipo de fuerza de la gravedad? Es de tipo aceleración, entonces cree el primer prototipo de script, como siempre, como estos scripts son reutilizables declaro una variable la cual puedo editar desde el editor y dependiendo de los planetas le asigno un valor u otro, y es sencillo, imaginemos que nosotros somos el planeta entonces si algún objeto entra en nuestra zona de gravedad, se le debe de aplicar una aceleración hacia mi. Sabiendo esto usaremos tres métodos de la api de unity, Awake, OnTriggerStay y OnCollisionEnter, los que explicare el OnTriggerStay ya que es el mas importante, si el objeto se mantiene adentro de mi campo, a este objeto se le aplicara la fuerza.
A continuación anexo el script…

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

public class GravitySimple : MonoBehaviour
```

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

public class GravitySimple : MonoBehaviour
{
// para establecer desde el editor cuanta fuerza ejercer en el campo
public float gravity;
// para cachear y no consumir tanta memoria buscando el componente a cada frame
private Transform _transform;

private void Awake()
{
// obtenermos la componente transform
_transform = gameObject.transform.parent.GetComponent<Transform>();
// agrandamos el radio del trigger para que tengamos buen rango del campo

GetComponent<SphereCollider>().radius = _transform.localScale.x*1000;

}

private void OnTriggerStay(Collider other)
{
// aqui si un obbjeto entra en el campo le sacamos la direccion a donde tiene que ir
Vector3 direction = (_transform.position - other.transform.position).normalized;
// y me marcaba un error de que no encontraba el rigidbody entonces cree la condicion para
que no llorar unity
    if(other.GetComponent<Rigidbody>()){
        /* y le aplicamos la fuerza en la direccion del planeta que esta ejerciendo la
aceleracion
la multiplicamos por el tiempo para que independiente de la maquino se le ejerza la misma
aceleracion
         y le decimos que es de tipo aceleración.
```

```
aqui obtengo en cada frame que el objeto que este dentro porque van a existir muchos
objetos dentro
        entonces no puedo cachear los rigidbodys, bueno solo uno no, pero esta bien de esta
manera.
        */
other.GetComponent<Rigidbody>().AddForce(direction*( gravity*
Time.deltaTime),ForceMode.Acceleration);
}
}

private void OnCollisionEnter(Collision collision)
{
// Para ver en consola el nombre del objeto
Debug.Log(collision.transform.name);
}




}
```



La variable gravedad como vemos la verdad no estoy muy segura si se le aplica en newtons en el caso de la fuerza tipo aceleración entonces con 900 en la primera prueba se me hizo bien, porque se ve rápido y podemos ver el potencial de nuestro script, teniendo esta versión muy básica nuestro objeto que se le ejerce una fuerza puede llegar a orbitar si lo tomamos con el ratón y lo movemos desde el editor de unity.

Este primer prototipo de la funcionalidad fue un gran avance, pero yo quería tener algo similar a la realidad, entonces investigando di con la ley universal de gravitación, pero me surgieron varias dudas, entonces mejor espere a la semana donde se explicaba este tema para poder preguntarle mis dudas a usted(si es que lo esta leyendo), a mi profe de Mecánica, ya tenia en mente mas o menos cual tenia que ser la manera de construcción de los objetos y también del script que tenían que tener, además de esto ya tenia el script core de como tenia que funcionar, entonces me tope con el primer problema o pensamiento que fue mi primer obstáculo.

Cantidades reales dentro de la simulación

Yo tenia entendido que las cantidades que se manejan tanto de masa, diámetro y distancias son demasiadas grandes para meterlas en una simulación, entonces no quería probar esto ya que confió en unity pero no lo suficiente para poner cantidades reales, entonces trate de poner a escala la fórmula de gravitación universal la cual es…

$$F = G\frac{m_1 m_2}{d^2}$$

daré una pequeña introducción a la ley.

La ley de la gravitación universal, o simplemente, ley de la gravedad, establece la fuerza con la que se atraen dos cuerpos por el simple hecho de tener masa. Esta ley fue desarrollada por Sir Isaac Newton en el tercer libro de su obra Principios matemáticos de filosofía natural, titulado Sobre el sistema del mundo.
Yo quería que al crear los planetas, asignando el script y jugando con la escala de estos, mas aparte con la masa del rigidbody, estos ejercieran una aceleración a escala jugando con estos parameros, pero me tope que implementando esta formula y jugando con los valores dichos, mas aparte poniendo a escala la constante de gravitación la cual es…

$$G = 6.67 \cdot 10^{-11}\ \frac{Nm^2}{kg^2}$$

Tenia comportamientos raros entonces entendí que no tenia que calcular la fuerza de atracción entre estos dos objetos, tenia que encontrar cuanta gravedad estaban ejerciendo base a su peso y diámetro, entonces encontré esta formula…

$$F = G\frac{m}{d^2}$$

Donde:
G es igual a la constante.
m es la masa del planeta.
d2 es el diámetro al cuadrado del planeta.

Pero recordé que dependiendo la distancia afecta mas o menos la gravedad del cuerpo celeste, entonces busque la ecuación la cual me ayudara a resolver este problema entonces busque, y encontré que es la mismo formula, pero en el diámetro le tenia que sumar la altura a la que se encontraba el objeto, entonces la formula queda así…

$$F = G\frac{m}{(d+h)^2}$$

Con esto ya tenemos la formula para programar la funcionalidad, y lo probaremos con cantidades reales.
Entonces busque la aceleración de nuestra estrella mas cercana, que es el sol y es de

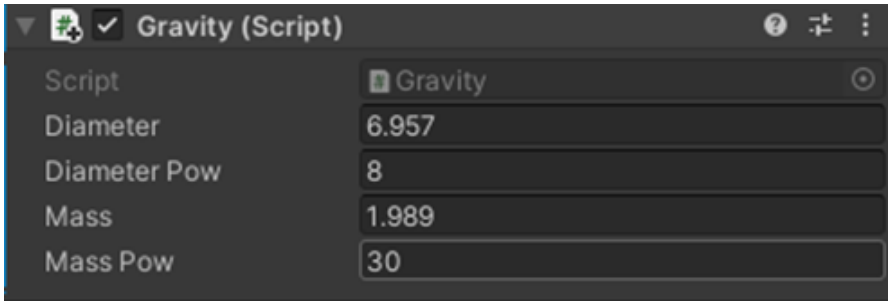
$$274\frac{m}{s^2}$$

entonces me propuse que con mi script me diera ese numero, y me di a la tarea.
Les dejo el script a continuación…

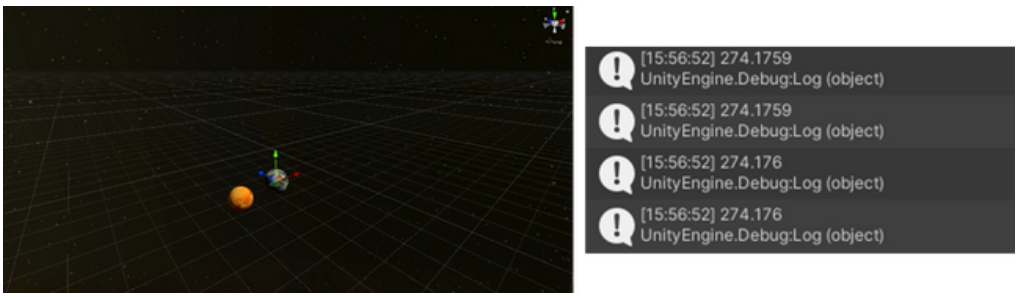
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Gravity : MonoBehaviour
{
//declaro la variable para obtener mi rigidbody
private Rigidbody rb;
// obtengo mi transform para agrandar por script nuestro campo
private Transform _transform;
// declaro la constante de la ley de gravitacion
private double G = 6.67428 * Math.Pow(10, -11);
// declaro nuestro diametro
  public float diameter;
// y a la potencia que va a ir nuestro diametro con notacion cientifica
public int diameterPow;
```

```
// lo mismo para la masa
public float mass;
public int massPow;
/*
* Cabe decir que se tuvo que hacer de esta manera, ya que no podemos escalar las esferas
* demasido grandes ya que no caben en pantalla, ademas de que a gran escala se deforman
* e igual con la masa, desde script tomo los datos hago el calculo y se lo asigno al rigidbody
*/
private void Awake()
{
// obtenemos nuestro rigidbody
rb = GetComponentInParent<Rigidbody>();
// obtenemos nuestro transform
_transform = gameObject.transform.parent.GetComponent<Transform>();
// agrandamos nuestro campo en el cual detectamos los objetos cerca de nosotros
GetComponent<SphereCollider>().radius = _transform.localScale.x*1000;

}
void Update()
{
// this is not the way
    //_direction = (sun.transform.position - transform.position).normalized;
}

private void OnTriggerStay(Collider other)
{
// obtenemos la direccion hacia donde se le tiene que aplicar la fuerza
Vector3 direction = (_transform.position - other.transform.position).normalized;
// esto es nuevo, ya que en funcion a la distancia la aceleracion disminuye
// necesitamos sabes que tan lejos esta el objetos de nosotros, entonces este metodo
// de la clase Vector3 nos dice a que distancia esta en metros
// como nota, podemos multiplicar este numero por 10 o por 100 dependiendo de nuestra
simulacion
float distance =
Vector3.Distance(transform.parent.transform.position,other.transform.position);
// comprobamos que exista el rigidbody del objeto para que no marque errores
```

```
if(other.GetComponent<Rigidbody>()){
        // le aplicamos la fuerza de tipo aceleracion
        // en la direccion correcta(hacia nosotros)
other.GetComponent<Rigidbody>().AddForce(direction*( GravityAcceleration(mass,
massPow, diameter, diameterPow,distance*1000)*
Time.deltaTime),ForceMode.Acceleration);
}
}
// le declaramos las variables que va a recibir para hacer los calculos
public float GravityAcceleration(float mass,int massPow,float diameter, int diameterPow,
float distance=0)
{
/*
* masa, la potencia de la masa, el diametro, la potencia del diametro, y la distancia
* la distancia por default es 0 por si no se le manda este no afecta en los calculos y no se
vera
* afectada la operacion.
* Esto quiere decir que podemos hacer que tome la distancia o no depende de nuestra
simulacion.
*/
// calculo por parte para que no se vea feo y se pueda leer por el ojo humano

// calculamos la masa
double massCalcule =mass * Math.Pow( 10, massPow);
// y se la asignamos a nuestro planeta, esto para que tenga el peso real y no solo
// calcule la aceleracion
rb.mass = Convert.ToSingle(massCalcule);
// igual para el diametro, lo calculammos, en este caso solo el calculo para la operacion.
double diameterCalcule = diameter * Math.Pow(10, diameterPow);
// esta es la operacion de la ley de gravitacion...
double calcule = G * (massCalcule/ Math.Pow(diameterCalcule+distance, 2));
// lo convertimos a flotante ya que nos arroja un numero tipo double por los decimales
float gravityForce = Convert.ToSingle(calcule);
//la retornamos
return gravityForce;
}
}
```



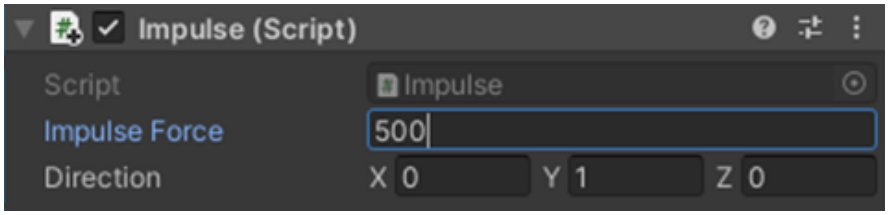
Así se ve nuestro scrpit en el editor, y este es el resultado de la aceleración de nuestro sol…



¡LO LOGRE! Logré simular la aceleración que ejerce el sol en los demás cuerpos celestes.
Ya lo tenemos todo, pero, si pongo dos planetas chocan entre ellos, normal ¿no?, pues si pero los planetas orbitan, entonces ¿cómo hacemos que orbiten?.
El que orbiten es "simple" lo trataremos de esa manera, de manera simple. El hacer que orbite ya esta resuelto, si recordamos que ya lo hacia si lo empujamos manualmente, y ¿cómo no lo hacemos manualmente? eso es sencillo, con un script, todo es fácil si sabes programar, entonces lo único que necesitamos es un script que en el COMIENZO le aplique a nuestro cuerpo una fuerza de impulso en alguno de los tres ejes, o en dos de estos.
Les dejo el script que realice resolviendo esta sencilla cuestión…

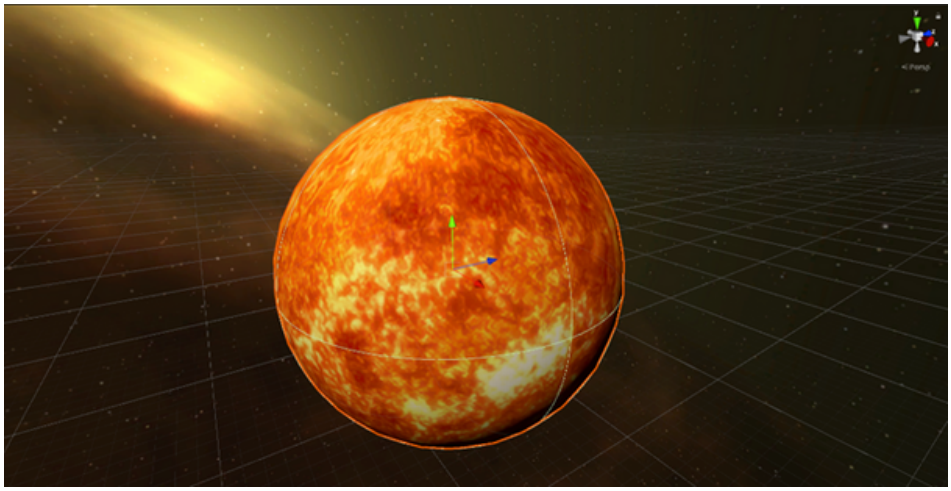
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Impulse : MonoBehaviour
{
// fuerza en newtons que se le aplicara en el comienzo de la simulacion
public float impulseForce=0;
// nuestro rigidbody que sirve para ejercer la fuerza
private Rigidbody rb;
```

```
// la direccion en la que ira nuestro impulso
public Vector3 direction;

private void Awake()
{
// obtenemos nuestro rigidbody
rb = GetComponent<Rigidbody>();
}

void Start()
{
// le damos el impulso en la direccion deseada justo en el inicio de la simulacion
rb.AddForce(direction*impulseForce,ForceMode.Impulse);
}

}
```



Ya tenemos todo para simular nuestro universo, con cantidades reales, incluso ya orbitan, y es escalable, podemos poner mas planetas con estos comportamientos gracias a que nuestros scripts son reutilizables, pero no tiene una rotación, y los planetas rotan, bueno para esto ya es otro tema, pero hagamos que roten solo por estética.

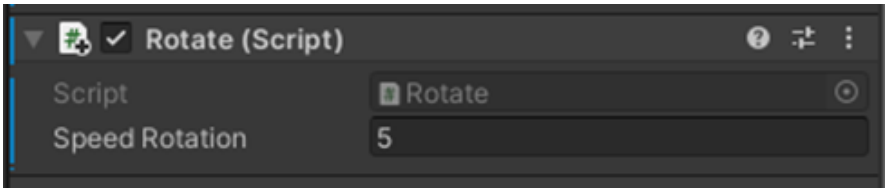
Esto es sencillo también, recordemos que todos los gameObjects tienen una componente que se llama transform, y en este tienen 3 cosas, posición, rotación y escala, solo tenemos que hacer es que por script aumentar el número de la rotación, en este caso en su eje y, pero ¿por que en el y? si vemos los objetos y estamos en el modo editor tienen estas líneas…



Bueno, esas líneas color amarillo, azul y roja son los ejes. En este caso el eje rojo es el x, el eje amarillo es el y, y por ultimo el azul es el z, es como si le atravesamos unos palos como si fuera pollo asado, y los rotamos en base esos palos.

Como el pollo asado de la imagen anterior, si tomamos 2 dos ejes, el x y el y, tiene el palo atravesado en el x, y si tomamos el palo y lo giramos el pollo gira en la dirección correspondiente al movimiento. Entonces sabiendo como funciona el rotar un objeto en unity lo tenemos todo para hacerlo rotar.
Y la solución es la siguiente…

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

public class Rotate : MonoBehaviour
{
// que tan rapido va a girar, se puede cambiar desde el editor
public float speedRotation = 0.1f;
void Update()
{
// por cada frame lo hacemos rotar
// multiplicamos el deltatime por la variable de velocidad y listo
transform.Rotate(0, Time.deltaTime * speedRotation, 0, Space.Self);
}
}
```



Ahora ya lo tenemos todo, ya rotan, ya orbitan, y lo mas importante estos planetas usan la ley de gravitación universal para calcular la aceleración y esta aceleración varia dependiendo a la distancia que se encuentren estos objetos.
Aun así, no me quede quiero, y durante unos tiempos libres, y porque es mi simulacion, hice algunos scripts extras los cuales no aportan anda para las físicas y simulaciones pero los anexo para que los vean…

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

public class Explote : MonoBehaviour
{
private ParticleSystem _ParticleSystem;

private MeshRenderer mshObject;
```

```
void Awake()
{
mshObject = GetComponent<MeshRenderer>();
_ParticleSystem = GetComponentInChildren<ParticleSystem>();
}

IEnumerator waiter()
{
mshObject.enabled = false;
_ParticleSystem.Play();
  yield return new WaitForSecondsRealtime(1.5f);
Destroy(gameObject);
}
private void OnCollisionEnter(Collision collision)
{
StartCoroutine(waiter());
}
}
```

Con el anterior script dispara unas partículas que simulan una explosión, esta explosión se activa cuando se detecta una colisión, después de la animación de explosión el planeta u objeto desaparece como si hubiera explotado realmente, esto lo hago en base a corrutinas. Les anexo el siguiente script…

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

public class DestroyOnClose : MonoBehaviour
{
public float grow;
private void OnTriggerStay(Collider other)
{
float distance =
Vector3.Distance(gameObject.transform.parent.transform.position,other.transform.position);
      if (distance < 40)
```

```
{

        Destroy(other.gameObject);
gameObject.transform.parent.transform.localScale += new Vector3(grow, grow, grow);
Debug.Log(gameObject.transform.parent.transform.localScale);
}
}
}
```

Este script sirve para destruir algo, lo que sea, que se le acerque a nuestro planeta a menos de 40 metros además le suma masa a nuestro objeto, y esto ¿para qué?, bueno si nos detenemos a pensar, tenemos planetas y estos ejercen una aceleración hacia ellos mismos, entonces si les agregamos este script a nuestros planetas y se le acercan cosas alrededor estos se destruirán y el planeta crecerá en escala, y si lo analizamos esto puede llegar hacer un tipo de agujero negro…