The purpose of Solar System Basic was to experiment with and learn Unity using physics. Unity's physics engine is very capable of simulating gravitational forces with rigidbodies.

See 'Credits.txt' in build folder for full acknowledgements. The process went through revisions. This methodology will only examine the final execution of the orbital physics.

C# was the only language used for the simulation. The scripts in the final program were as follows:

- CameraControl.cs
- CameraTrack.cs
- ExpandButton.cs
- Orbital.cs
- Quit.cs
- Speedometer.cs
- TimeController.cs
- TrailToggle.cs

The first step was creating gravitational attraction between GameObjects (henceforth 'bodies'). This was done using C# script (Orbital.cs) attached to each body.

Orbital.cs

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

public class Orbital : MonoBehaviour
{
	//To rotate to face an object
	public Transform parent;

	//Required for the physics to work
	public Rigidbody rb;
	public float gravConstant;
	public static List<Orbital> Attractors;
	private Vector3 velocity;

	//Initialise the behaviour of the object
	public Vector3 targetOrbitalVelocity;
	public Vector3 constantForce;
	public Vector3 startForce;
	public Vector3 rotateSpeed;

	//For gravity to pull on objects
	private float attractionForce;
	private Vector3 attraction;


	void Start ()
	{
		Velocity();
	}

	void FixedUpdate()
  {
	  foreach (Orbital attract in Attractors)
	  {
			if (attract != this)
				Attract(attract);
	  }
```

```
            //this applies a given force every fixed update
            rb.AddRelativeForce(constantForce);


            //Rotates at a given rate
            transform.Rotate (rotateSpeed);


            //Rotates to look at a given object
            transform.LookAt(parent);
    }

        //OnDisable and OnEnable are a more effecient way of finding objects in the scene to move towards
        void OnEnable ()
        {
                if (Attractors == null)
                        Attractors = new List<Orbital>();

                Attractors.Add(this);
        }

        void OnDisable ()
        {
                Attractors.Remove(this);
        }

        //This is responsible for all the physics enabled bodies attracting each other
    void Attract(Orbital objToAttract)
    {
        Rigidbody rbToAttract = objToAttract.rb;

        Vector3 direction = rb.position - rbToAttract.position;
        float distance = direction.magnitude;

                if (distance == 0f)
                {
                        return;
                }

        attractionForce = gravConstant * (rb.mass*rbToAttract.mass) / Mathf.Pow(distance, 2);
        attraction = direction.normalized*attractionForce;

        rbToAttract.AddForce(attraction);
    }

        //This sets the initial velocity of an object in UU / time
        void Velocity ()
        {
                //this converts the real world velocity and scales it down 510 times for this simulation
                velocity = targetOrbitalVelocity / 510.77586f;

                //This is the calculation used by the physics engine to apply velocity, scaling with gravConstant
                rb.AddRelativeForce(velocity * Mathf.Pow (gravConstant, .498f), ForceMode.VelocityChange);

                //If I want to add a random force at the start
                rb.AddRelativeForce(startForce);
        }

}
```

This script had several functionalities. The first was creating attraction between bodies. This was done using the Attract(), OnEnable(), OnDisable(), and FixedUpdate() methods. It uses Newton's law of universal gravitation.

Newton's law of universal gravitation

$F = G * (m_1 * m_2) / distance^2$

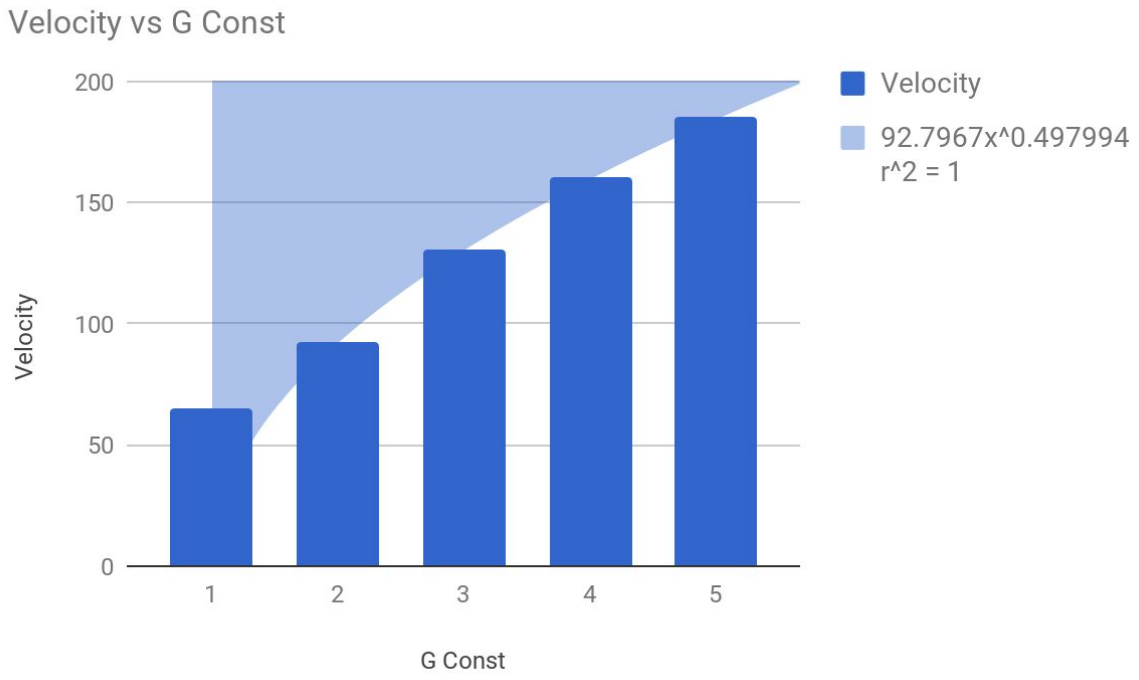
```
//attractionForce = gravConstant * (rb.mass*rbToAttract.mass) / Mathf.Pow(distance, 2);
```

It applies this equation to each bodies rigidbody component, and by searching the scene for every other rigidbody with this script attached. It repeats the process in FixedUpdate() - which is how Unity's physics engine works - operating at a fixed rate instead of every frame.

OnDisable() and OnEnable() were used to efficiently organise every body in the scene and automatically find new ones if they were added in.

The simulation is scaled down to each Unity Unit (UU) being equal to 1495978707 m (So that 1AU = 100UU). This is .000000066845871% of reality. It was essential on a cerebral level that the program could handle any gravitational constant (G) and scale the required velocity accordingly.

This was done by observing the required velocities for stable orbit on an incremental level relative to G.



This gave the following equation:

V = Required velocity
G = Given gravitational constant

V * $G^{.498}$ (Note the tiny decimal point.)

```
//rb.AddRelativeForce(velocity * Mathf.Pow (gravConstant, .498f), ForceMode.VelocityChange);
```

This meant that the program can handle any G while scaling the required velocity accordingly, maintaining stable orbit. The only noticeable difference would be the apparent speed of the simulation, a higher G means a faster velocity would be required. Velocity was handled by the Velocity() method which was called in Start().

Orbital.cs had a few functionalities which were not used in the final program, such as the ability for a force to be added along an axis of the body. The only other thing of import was the fact the script divided the required velocity by 510 in order to allow the real world value be added into the editor.