Planetary Motion with Three.js

Isaac Naupa & Hussain Raza

(OOP)s I did it again!

- Object Oriented Programming (OOP) was the main priority.
  - We made our project useable for more than just our solar system.
  - Our code can be used on multiple planetary systems with our approach using classes.
- Each planet class extends from *Planet.js.* This allows each planet to have its own functionality
  - (i.e. Moons, Rings, Rotation about its Axis, etc.).
- Our stars for these systems extends the *Star.js* file.
  - This allows each star to have its own functionality (i.e. Brightness, Rotation about its Axis, etc.).

A snapshot of our Planet constructor:

```
class Planet {
    constructor(mass, perDist, apDist, vel, size, texture) {
        this.mass = mass;
        this.apDist = apDist;
        this.perDist = perDist;
        this.vy = vel;
        this.vx = 0;
        this.x = (perDist+apDist)/2;
        this.y = 0;
        this.size = size;
        this.texture = texture;
        this.ax = 0;
        this.ay = 0;
    }
```

More OOP

- The class Planet.js has certain behaviors such as the ability to be accelerated, or being able to compare distances between other planet objects.
- This again facilitates the process by only having to write the acceleration definitions once.
- Imagine having to write this function over and over again for each planet, keeping on your indexes in check.
- This would be hard to read, difficult to debug, bad code in general.

```
distBTWN(ox, oy) {
    let distBTWNX = this.x - ox;
    let distBTWNY = this.y - oy;
    let distBTWNR = Math.sqrt(distBTWNX**2 + distBTWNY**2);
    return distBTWNR**3;
}
accelerationX(p1,p2,p3,p4,p5,p6,p7,p8,p9) {
    return (G*p1.mass*(this.posNegX(p1.x))*this.x/(this.distBTWN(p1.x,p1.y))) + (G*p2.mass*(this.posNegX(p2.x))*this.x/(this.distBTWN(p2.x,p2.y)))
    + (G*p3.mass*(this.posNegX(p3.x))*this.x/(this.distBTWN(p3.x,p3.y))) + (G*p4.mass*(this.posNegX(p4.x))*this.x/(this.distBTWN(p4.x,p4.y)))
    + (G*p5.mass*(this.posNegX(p5.x))*this.x/(this.distBTWN(p5.x,p5.y))) + (G*p6.mass*(this.posNegX(p6.x))*this.x/(this.distBTWN(p6.x,p6.y)))
    + (G*p7.mass*(this.posNegX(p7.x))*this.x/(this.distBTWN(p7.x,p7.y))) + (G*p8.mass*(this.posNegX(p8.x))*this.x/(this.distBTWN(p8.x,p8.y)))
    + (G*p9.mass*(this.posNegX(p9.x))*this.x/(this.distBTWN(p9.x,p9.y)));
}
```

The Gravitational Force:

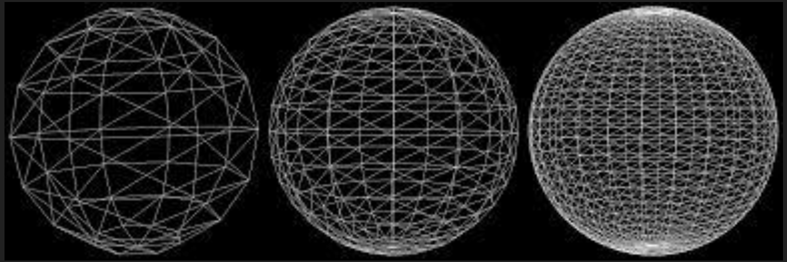
- We implemented verlet in order to march the time.
- Euler leads to error aggregation
  - This gives us wacky dynamics like Mercury being engulfed by the sun.
- Verlet gives us the ability to work with higher *dt* as well as conserve energy.

$$m\frac{d^2r}{dt} = -GMm\,\hat{r}/r^2$$

```
sun.ax = accX(sun);
sun.ay = accY(sun);

sun.x += sun.vx*dt + sun.ax*.5*dt*dt;
sun.y += sun.vy*dt + sun.ay*.5*dt*dt;

sun.vx += .5*(sun.ax + accX(sun))*dt;
sun.vy += .5*(sun.ay + accY(sun))*dt;
```

Three.js

- Allows us to use WebGL as our main graphics power house.
- Provides a high level of Abstraction
- Higher level user immersion with Camera and Light tools.

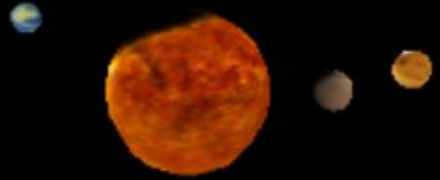


```
renderer = new THREE.WebGLRenderer();
```

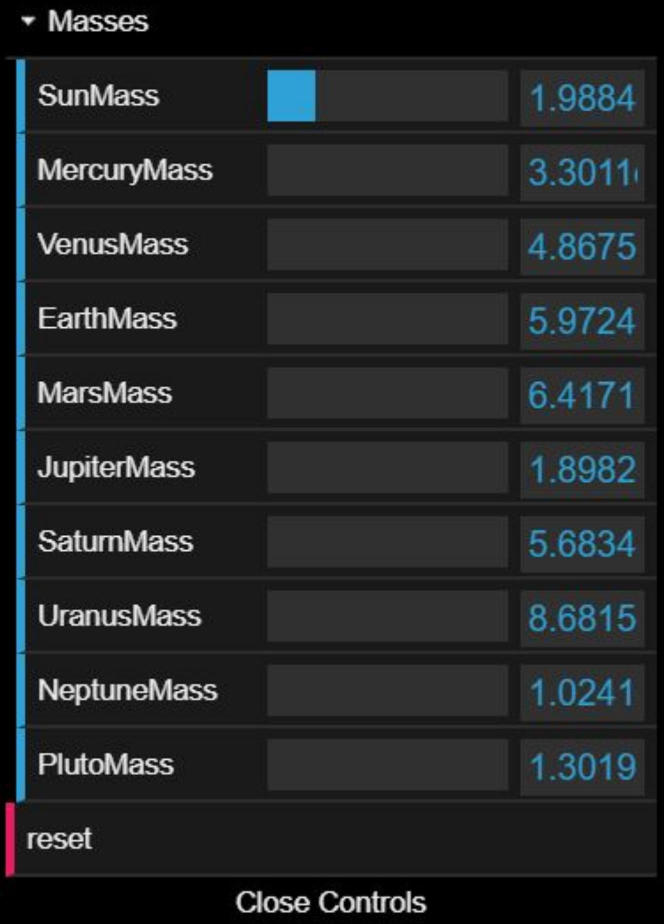
```
scene = new THREE.Scene();

camera = new THREE.PerspectiveCamera(
    75,
    window.innerWidth / window.innerHeight,
    0.1,
    1000
);
```

Features:

Camera Controls:
Left click: look around
Right click : move around
Zoom in/out w/wheel or finger gestures



Change dynamics real-time w/ gui



Improvements/ Further Work

- Plotting the trajectories, user events when hovering over certain planets,
- Exploring Light features in three.js (making each sun have its own lighting)
- Including the asteroid belt
- Adding planet specific functionality

Acknowledgements

Special Thanks to Flavio and Abouzar for helping us with this project along the way, and for a great semester.