ONTARIO TECH UNIVERSITY
FACULTY OF SCIENCE, COMPUTER SCIENCE

Project Group: 15 April 6, 2024

Simulation and Modeling: Gravitational Pull of Heavenly Bodies Simulation

by

Johnny Liang

Abstract:

How would an object react to the gravitational pull from a massive heavenly body? That is what we are simulating through our project. We are placing massive bodies at different locations and having them to interact with each other. Also we will be able to control a spaceship to visualize the pull at different locations.
Github Hyperlink

1 Introduction

Pygame simulation that explores how Celestial Bodies interact with each other, in addition, a Spaceship, controlled by the user, is included to experiment the pull of the different Celestial Bodies at different locations.

2 Body

2.1 Methodology

The simulation contains two classes, `CelestialBody` and `Spaceship`.
The `CelestialBody` and `Spaceship` classes contain the variable `state[4]` which contains position in the x and y axis and velocity in the x and y axis. The variables and their differential equation are as follows:

$$\{x, y, vx, vy\} \rightarrow \{vx, vy, ax, ay\}$$

where v is the velocity in the respective axis and a is the acceleration in the respective axis.
Both classes have a function $f()$ as input for the ode solver and they are defined as follows:

```
def f(self,t,state):
    vx=self.state[2]
    vy=self.state[3]
    return [vx,vy,0,0]
```

Note that instead of calculating the acceleration here, the a values are kept 0. This implementation allows the `Spaceship` to move indefinitely as long as there is no force to stop its movement, same as in space in the real world. So, in order to accelerate and move, when a directional key is pressed, the spaceship will add an arbitrary amount (input variable - `ACCELERATOR`) to the velocities variables in the corresponding direction.

The other way objects accelerate in the simulation is through the Gravitational Force:

$$F = \frac{GMm}{r^2}$$

Where G is the gravitational constant, M is the mass of an object, m is the mass of the other object, and r is the distance between them. For each step in the simulation, for each `CelestialBody` in the simulation, this Force will be calculated. By:

1. Calculating the distance between objects.
2. Computing the Unit Vector
3. Computing the Force
4. Multiplying the Unit Vector with Force to get the Force in each axis.

Then through the equation:

$$ma = \frac{GMm}{r^2}$$

we can get the acceleration and add it to the `Spaceship`'s velocity variables.

2.2 Experiments

In this simulation, if the `Spaceship` and `CelestialBody` collide, the `Spaceship` will explode and the simulation pauses. If two `CelestialBody` collide, they will fuse and become one. For collision detection, each object is treated as a circle. If the distances of a pair of object is less than the sum of their radii, a collision is detected. The simulation incorporates the Sweep and Prune algorithm which abuses the transitive property the objects have.

For all elements a, b, c in X, if a is related to b, a is also related to c.

In our context, if a does not overlap b then a does not overlap c as well, and that is also true for the rest of objects in the array. //

Another addition for this simulation is that the gravitational pull from each celestial body can be localized so that only those objects within an area will be affected. This can be toggled through the variable `ring_influence` within the `main` function. If toggled on, the area of the ring will be visualized and if the spaceship or another `CelestialBody` is inside the ring, the Gravitational Forces will be accounted for them.

3 Conclusion

The simulation mimics the interaction of Celestial Bodies one with another through the Gravitational Force they exert on one another. It incorporates collision optimization, RK4 iterative methods, and Pygame library.

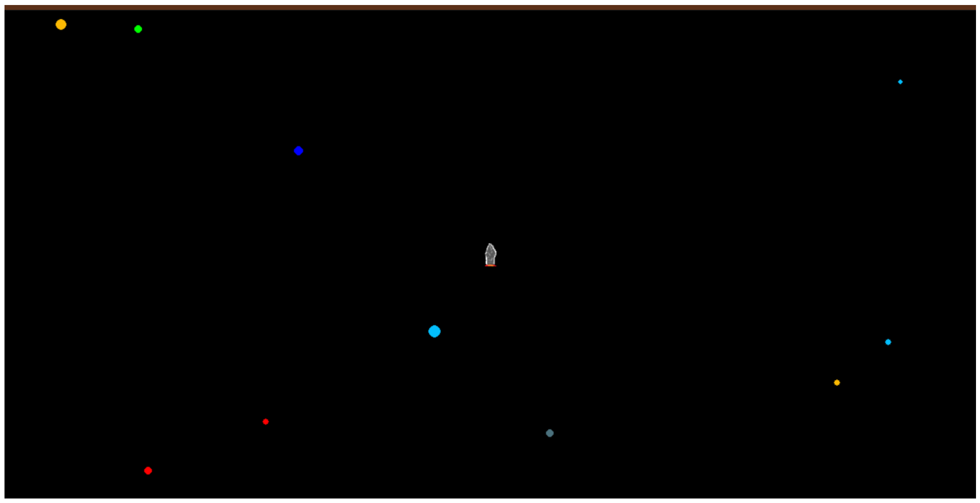
Figure 1: Demo of Simulation

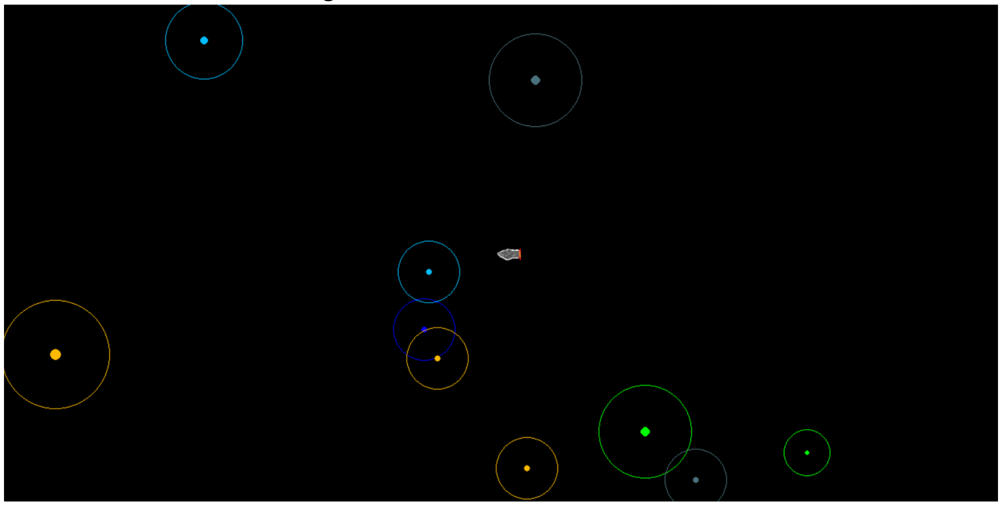
Figure 2: Demo of Simulation with Rings Toggled.

This simulation is limited in the following ways:

1. Slow when it contains many objects (10+) even with Sweep and Prune optimization
2. Not to scale. Size and speed of Spaceship is not to scale with the simulation, hence less realistic.
3. Very simplistic and rough graphic work.
4. Fusing of two Celestial Bodies is oversimplified and does not reflect a real life scenario.

Some improvements:

1. Further optimize with QuadTree structure
2. Implement a dynamic camera, this will allow more screen space, thus, allowing the creation of bigger and more realistic Celestial Bodies.
3. Through dynamic camera implementation, add more details to the Celestial Bodies and Spaceship.
4. Investigate more and implement a more realistic outcome for Ceelstial Bodies colliding.