**Simulation of Solar System**

Author: Ian Chang, Yi-Jhan Huang

**Abstract** This is a simulation and analysis project based on python and pygame for the purpose of a visual representation of the solar system.

**1. Introduction**

Our research project is based on exploring the solar system in the way of using python and other few packages (numpy, pygame, matplotlib) to simulate and analyze the orbital period, path, and other related details about how each planet react with the sun if different attributes have changed.

**2. Data and Methods**

Programming Language: Python

Packages required: Numpy, Pygame, Matplotlib

We create our self-defined class “Planet” as our main class for simulation. With the combination of visual representation using pygame and graphic analysis using matplotlib, we get all of our result based on different physics equation and the actual mass, distance to sun for each planet to complete all the simulation.

**Class Planet** (Complete code in appendix 1)

- Constructor:

def\_\_init\_\_('name', distance to sun(astronomical unit),

radius (self-define for graphic simulation),

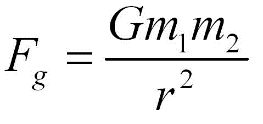
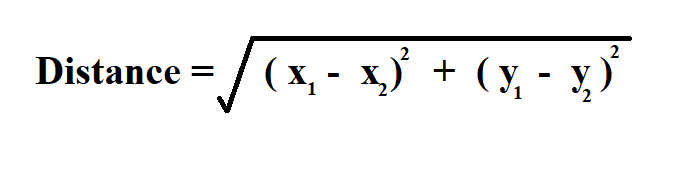
presenting color (RGB code),

mass of planet (Kilogram)):

- Attraction Function: (use as the function to calculate attraction force between two [class planet] objects)

def attraction(self, other):

return force\_x, force\_y

Physics Equation relates to the function

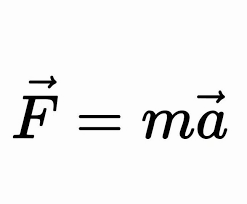
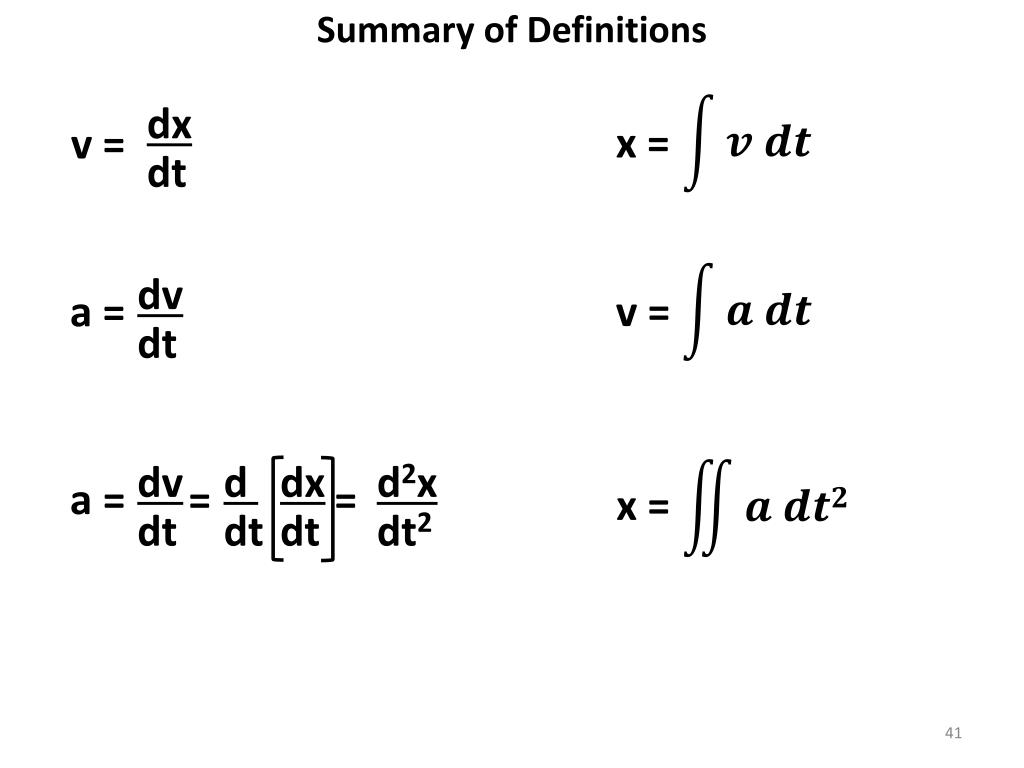
First equation (left): two-dimension distance formula

Second equation (right): Newton’s law of universal gravitation

- Update Position Function: (use as the function to update position of the planet. Calculated from a list of Planet objects)

def update\_pos(self, planets):

return None

Physics Equation relates to the function

First equation (left): Newton's Second Law of Motion

Second equation (right): instantaneous acceleration formula

- Move function: (use as the function to re-scale the point within the orbit position list to fit in the screen of the simulation, returning a list of orbit points and final x, y position)

def move(self):

return updated\_points, (x, y)

**Simulation** (Complete code in appendix 2)

def simulation(planets, limit=-1, delay=True):

pygame.init()

...

while run and count != limit:

...

pygame.quit()

Parameters for the function:

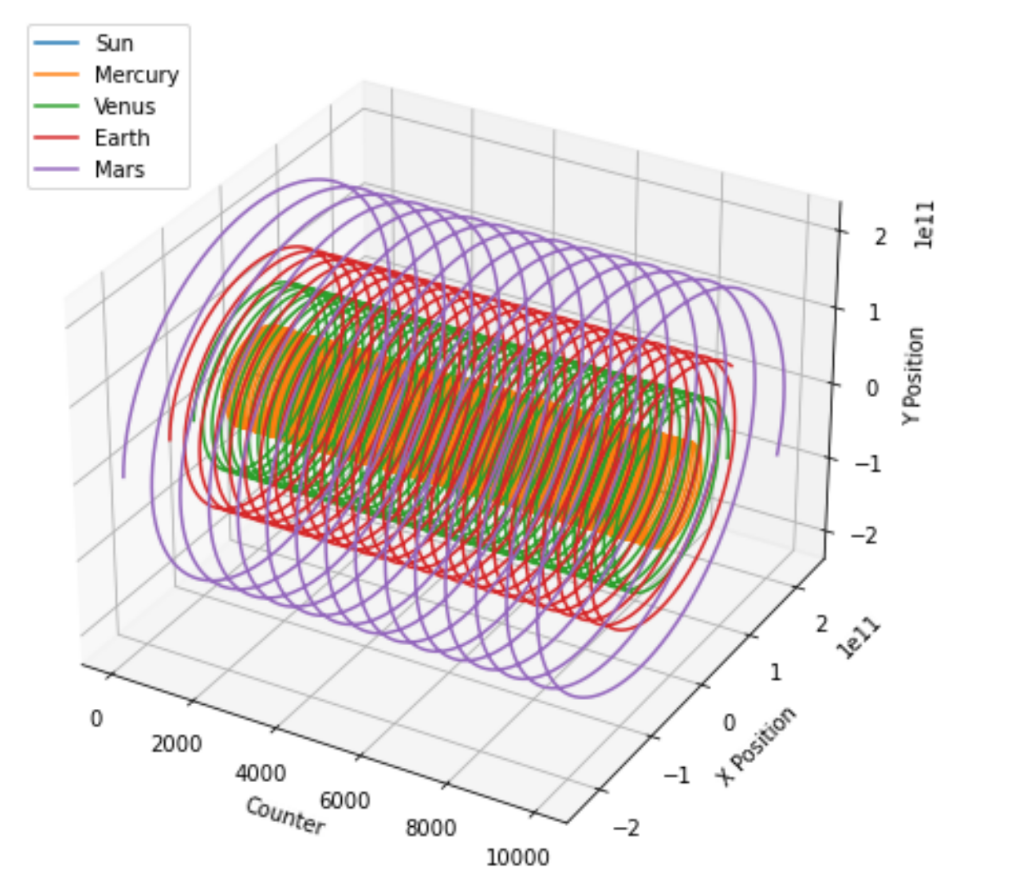
planets: list of planets in the system

limit: set limit for move steps

delay: (Boolean) if True, there will be delay in the update of the graph

**3. Results**

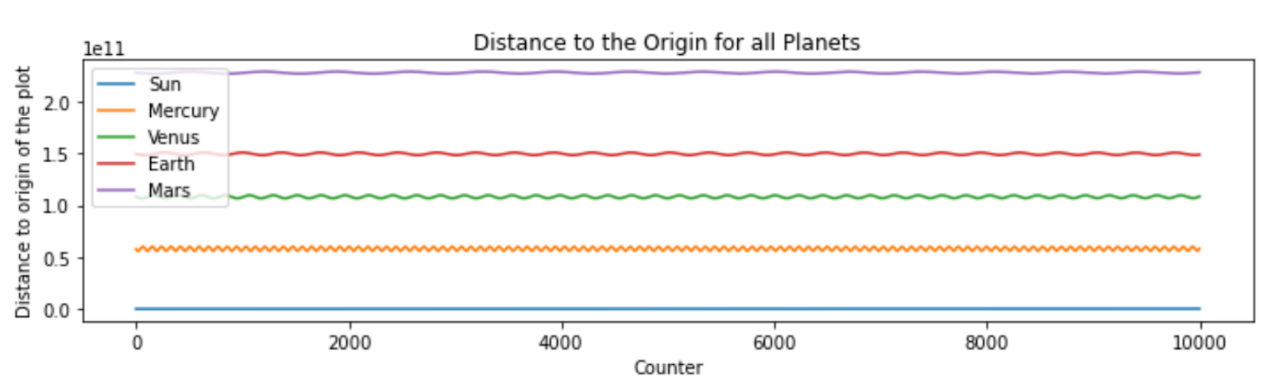
- Normal Case (The actual Solar system, mass of sun = 1.98892e30)

**Resulting Path, 3d representation**

Analyze:

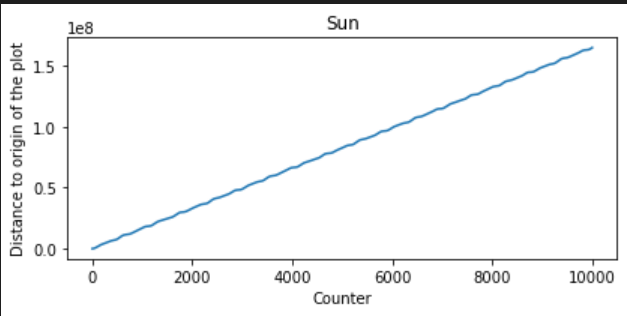
1. No clear change of the orbital path
2. We can see that the planet further from the sun has a longer orbit period

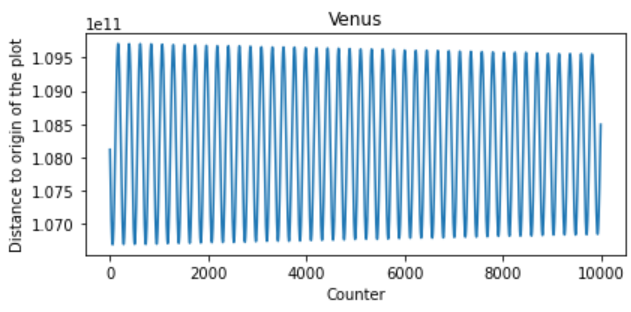
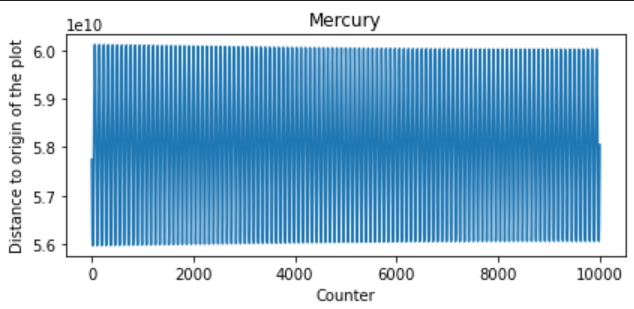
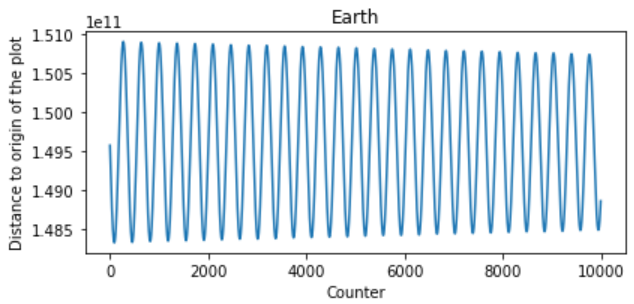
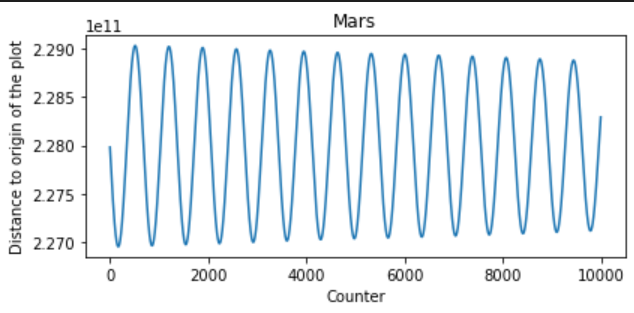
**Resulting Path, 2d representation**

Analyze:

1. Not changing the average distance from the sun
2. The distance change in wave, indicates the orbit path is ellipse instead of circle

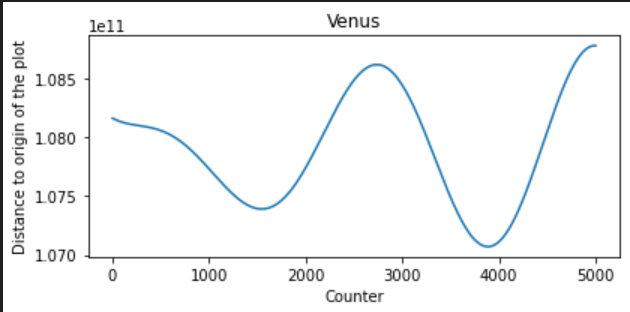
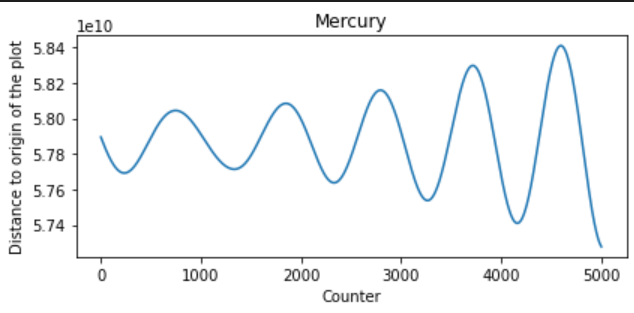
**Resulting Path, 2d representation**

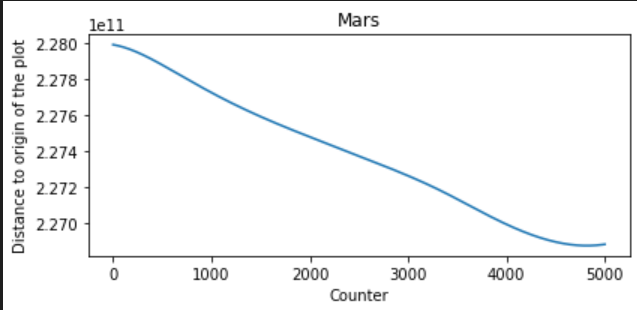
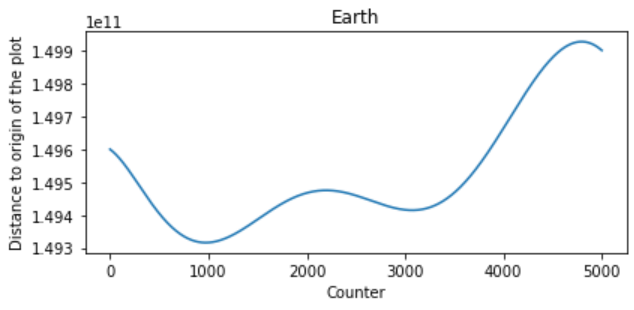


  Analyze:

1. Sun is actually slowly moving away from the origin due to attraction force from other planets in the solar system
2. The distance change in wave, indicates the orbit path is ellipse instead of circle
3. The wave length is smaller for planet closer to the sun, indicates that the orbit speed is faster

- What if Case (Mass of the sun / 100, mass of sun = 1.98892e28)

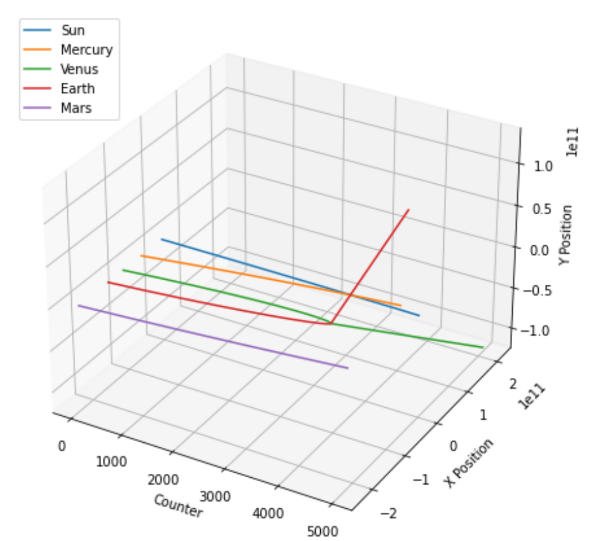
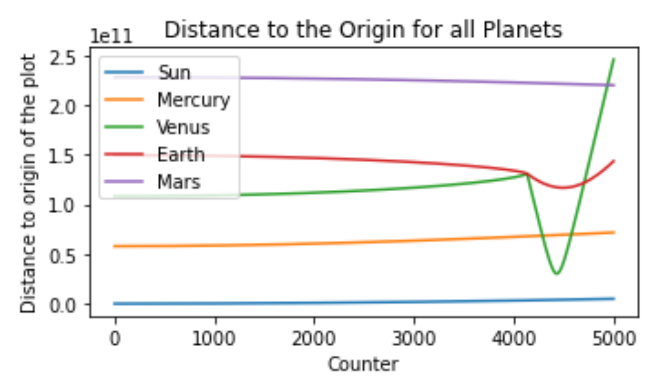




Analyze:

1. Mercury, Venus, Earth is slowly leaving the origin
2. Mars is slowly moving closer to the origin

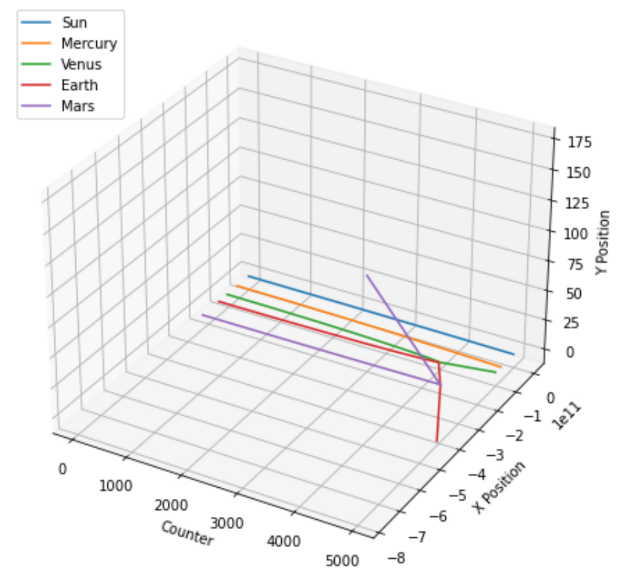
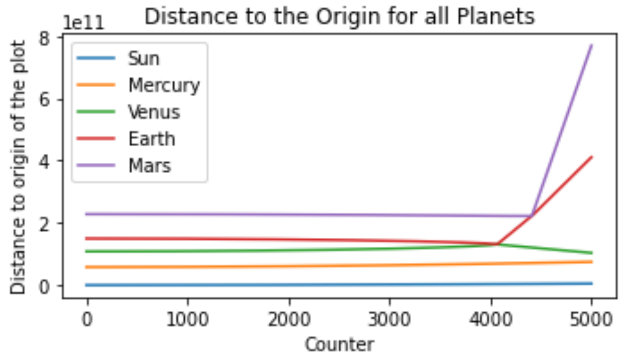
- What if Case (Mass of the Sun = Earth, mass of sun = 5.972e24)



Analyze:

1. Mercury, Venus, Earth, Mars are not orbiting around the sun.
2. Gravitational force slowly pulls each planet together

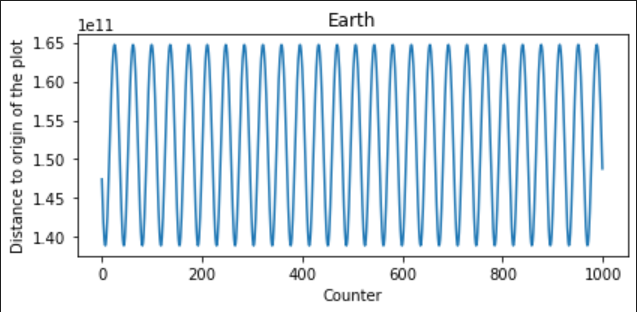
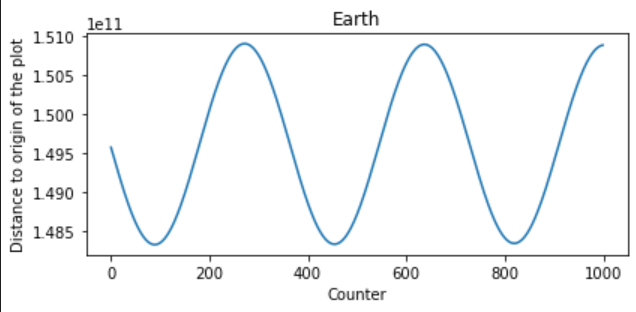
- What if Case (Mass of the Sun = 1, mass of sun = 1)



Analyze:

1. Mercury, Venus, Earth, Mars are not affected by sun.
2. Gravitational force slowly pulls each planet together stronger

- What if Case (Mass of the Sun \* 100, mass of sun = 1.98892e32)



Original Mass (left), \*100 Mass (right)

Analyze:

1. Earth is orbiting closer and at a faster speed.

**4. Conclusion**

We successfully simulate the solar system using the pygame and numpy. As for the analysis part, the result suggests that the orbit path of the planets is actually in ellipse instead of circle. Also, from all the what if case of the changing of mass of the sun, we can see that the orbiting of the planet is strongly based on the large mass of the sun and the attraction force creates from the extremely large mass of the sun.

**5. References**

- https://towardsdatascience.com/simulate-a-tiny-solar-system-with-python-fbbb68d8207b

- https://thepythoncodingbook.com/2021/12/11/simulating-3d-solar-system-python-matplotlib/

- https://towardsdatascience.com/simulate-a-tiny-solar-system-with-python-fbbb68d8207b

- <https://github.com/xhinker/orbit/blob/main/solar_orbit_3d_plt.py>

**Appendix 1**

class Planet:

    AU = 149.6e6 \* 1000             # distance from earth to sun

    G = 6.67428e-11                 # gravitational constant

    SCALE = 200 / AU                # small value to scale down the solar system to fit it on the screen

    TIMESTEP = 60 \* 60 \* 24         # 1 day

    MASS\_OF\_SUN = 1.98892 \* 10 \*\* 30 # mass of sun

    WIDTH, HEIGHT = 800, 800

    # Constructor

    def \_\_init\_\_(self, name, dis\_to\_sun, radius, color, mass):

        self.name = name

        self.x = -dis\_to\_sun \* Planet.AU           # x position

        self.y = 0                              # y position

        self.radius = radius                    # radius

        self.color = color                      # color

        self.mass = mass                        # mass

        self.sun = (dis\_to\_sun == 0)           # if it's the sun

        self.dis\_to\_sun = dis\_to\_sun \* Planet.AU  # distance from planet to sun

        if (self.sun): Planet.MASS\_OF\_SUN = mass

        self.orbit = []

        self.x\_speed = 0 # x speed

        self.y\_speed = 0 if (self.sun) else math.sqrt((Planet.G \* Planet.MASS\_OF\_SUN) / self.dis\_to\_sun)     # y speed

    # Draw line function for the updated\_points

    def move(self):

        # divide by two because pygame (0, 0) is the top left corner

        x = self.x \* self.SCALE + self.WIDTH / 2     # x position on screen

        y = self.y \* self.SCALE + self.HEIGHT / 2    # y position on screen

        updated\_points = []

        for point in self.orbit:

            x, y = point

            x = x \* self.SCALE + self.WIDTH / 2

            y = y \* self.SCALE + self.HEIGHT / 2

            updated\_points.append((x, y))

        return updated\_points, (x, y)

    def attraction(self, other):

        distance\_x = other.x - self.x # distance between x positions

        distance\_y = other.y - self.y # distance between y positions

        distance = math.hypot(distance\_x, distance\_y) # distance between planets

        force = self.G \* self.mass \* other.mass / distance \*\* 2 # force between the planets

        theta = math.atan2(distance\_y, distance\_x) # angle between the planets x and y positions

        force\_x = math.cos(theta) \* force # force in the x direction

        force\_y = math.sin(theta) \* force # force in the y direction

        return force\_x, force\_y

    def update\_pos(self, planets):

        total\_fx = total\_fy = 0 # initialize x and y force to zero

        for planet in planets:

            if self == planet: # if the planet is itself

                continue

            fx, fy = self.attraction(planet)

            total\_fx += fx

            total\_fy += fy

        self.x\_speed += total\_fx / self.mass \* self.TIMESTEP

        self.y\_speed += total\_fy / self.mass \* self.TIMESTEP

        self.x += self.x\_speed \* self.TIMESTEP

        self.y += self.y\_speed \* self.TIMESTEP

        self.orbit.append((self.x, self.y))

    def \_\_str\_\_(self):

        return f"""Planet(name={self.name}, radius={self.radius}, color={self.color}, mass={self.mass}, dis\_to\_sun={self.dis\_to\_sun},

        x={self.x}, y={self.y}, sun={self.sun}, orbit\_list={self.orbit[0:5]}, x\_speed={self.x\_speed}, y\_speed={self.y\_speed})

        """

**Appendix 2**

def simulation(planets, limit=-1, delay=True):

    pygame.init()

    # set pop up screen size

    SCREEN = pygame.display.set\_mode((Planet.WIDTH, Planet.HEIGHT))

    FONT = pygame.font.SysFont("garamond", 16)

    # set title of window

    pygame.display.set\_caption("Solar System")

    run = True

    clock = pygame.time.Clock()

    count = 0

    while run and count != limit:

        SCREEN.fill((0, 0, 0)) # fill the screen black every frame, or else the old frames will still be on the screen

        # Quit loop when receive quit signal

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

        for planet in planets:

            planet.update\_pos(planets) # update the position of the planets using the forces of attraction

            updated\_points, (x, y) = planet.move() # draw the planets

            if len(updated\_points) >= 2: pygame.draw.lines(SCREEN, planet.color, False, updated\_points)

            if not planet.sun:

                planet\_name\_text = FONT.render(f"{planet.name}", True, WHITE)

                SCREEN.blit(planet\_name\_text, \

                    (x - planet\_name\_text.get\_width() / 2, y - planet\_name\_text.get\_height() / 2 - 20))

            pygame.draw.circle(SCREEN, planet.color, (x, y), planet.radius) # draw the planet

        pygame.display.update() # update the display

        if (delay): clock.tick(60) # Set up update delay

        count += 1

    pygame.quit()