Simulating Solar System and Gravitational Sling Shot in Python.

R. Virinchi I20PH008
NIT Surat

May 2023

1 Introduction

Python has become a go-to programming language for astronomical simulations, playing a pivotal role in modeling and understanding complex celestial phenomena. Astronomers utilize Python for a variety of simulation tasks, ranging from the dynamics of celestial bodies to the evolution of galaxies. The simplicity and readability of Python code, along with its extensive ecosystem of libraries, make it an ideal choice for researchers. Astropy, a core package for astronomy in Python, provides functionalities for celestial coordinate transformations, time and date handling, and physical constants, making it essential for accurate simulations. Additionally, packages like AstroML offer machine learning tools that can be applied to analyze simulated data and extract meaningful patterns. Furthermore, Python is employed in numerical simulations through libraries like NumPy and SciPy, allowing astronomers to solve complex mathematical equations involved in simulating gravitational interactions, stellar evolution, and fluid dynamics within cosmic structures. Visualization tools such as Matplotlib aid in creating clear and insightful plots to interpret simulation results.

2 pygame

Pygame serves as a robust and user-friendly open-source library for the Python programming language, specifically designed for game development and multimedia applications. Built on top of the Simple DirectMedia Layer (SDL), Pygame provides a simplified interface, making it accessible to both beginners and experienced developers. With its comprehensive set of modules, Pygame facilitates the creation of 2D games, simulations, and interactive graphical programs. Offering functionalities for handling graphics, sound, input devices, and more, Pygame empowers developers to bring their creative visions to life by seamlessly integrating multimedia elements into their Python projects. Whether aspiring to craft classic arcade-style games or engaging educational simulations, Pygame stands out as a versatile and widely-used framework within the Python programming ecosystem.

3 Solar System Simulation

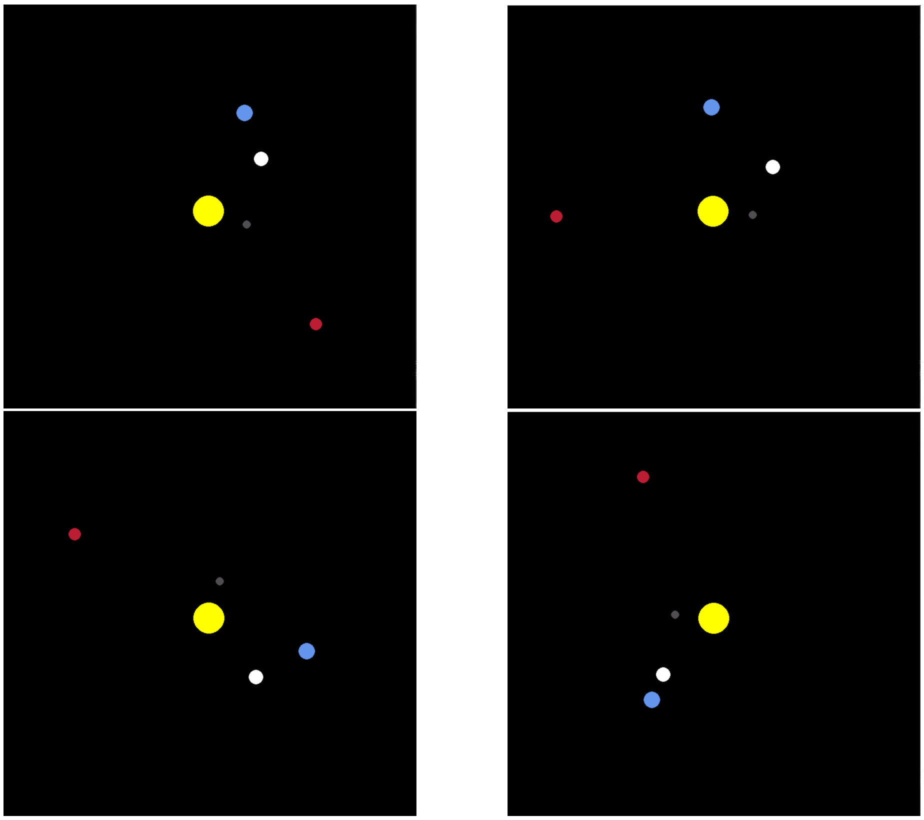
3.1 Python Code

```
import pygame
import math
pygame.init()

WIDTH, HEIGHT=800,800

WIN=pygame.display.set_mode((WIDTH,HEIGHT))
pygame.display.set_caption('solar system simulation')

WHITE=(255,255,255)
YELLOW = (255,255,0)
BLUE=(100,149,237)
RED=(188,29,50)
DARK_GREY=(80,78,81)
BROWN=(139,69,19)

class Planet:
    AU=149.6e6*1000
    G=6.67428e-11
    SCALE=200/AU # 1AU = 100 pix
    TIMESTEP = 3600*24

    def __init__(self,x,y,radius,color,mass):
        self.x=x
        self.y=y
        self.radius=radius
        self.color=color
        self.mass=mass

        self.sun=False
        self.dist_to_sun=0
        self.orbit=[]

        self.x_vel=0
        self.y_vel=0

    def draw(self,win):
        x=self.x*self.SCALE+WIDTH/2
```

```
        y=self.y*self.SCALE+HEIGHT/2
        if len(self.orbit)>2:
            updated_points=[]
            for point in self.orbit:
                x,y=point
                x=x*self.SCALE+WIDTH/2
                y=y*self.SCALE+HEIGHT/2
                updated_points.append((x,y))


        #pygame.draw.lines(win,WHITE,False,updated_points, 1)
        pygame.draw.circle(win, self.color, (x,y),
        self.radius)

    def attraction(self,other):
        other_x,other_y=other.x,other.y
        distance_x=other_x-self.x
        distance_y=other_y-self.y
        distance = math.sqrt(distance_x**2+distance_y**2)

        if other.sun:
            self.dist_to_sun=distance

        force=self.G*self.mass*other.mass/distance**2
        theta = math.atan2(distance_y,distance_x)
        force_x=math.cos(theta)*force
        force_y=math.sin(theta)*force
        return force_x,force_y

    def update_pos(self,planets):
        total_fx=total_fy=0
        for planet in planets:
            if self==planet:
                continue
            fx,fy=self.attraction(planet)
            total_fx+=fx
            total_fy+=fy

        self.x_vel+=total_fx/self.mass*self.TIMESTEP
        self.y_vel+=total_fy/self.mass*self.TIMESTEP

        self.x+=self.x_vel*self.TIMESTEP
        self.y+=self.y_vel*self.TIMESTEP
        self.orbit.append((self.x,self.y))

def main():
```

```
    run =True
    clock=pygame.time.Clock()

    sun=Planet(0,0,30,YELLOW,1.98892*10**30)
    sun.sun=True

    earth=Planet(-1*Planet.AU, 0, 16, BLUE, 5.9742*10**24)
    earth.y_vel=29.783*1000
    mars=Planet(-1.524 * Planet.AU, 0, 12, RED,
    6.39*10**23)
    mars.y_vel=24.077*1000
    mercury=Planet(0.387*Planet.AU, 0, 8,DARK_GREY,
    3.30*10**23)
    mercury.y_vel=-47.4*1000
    venus=Planet(0.723*Planet.AU, 0, 14,WHITE,4.86*10**24)
    venus.y_vel=-35.02*1000
    planets=[sun,earth,mars,venus,mercury]

    while run:
        clock.tick(60)
        WIN.fill((0,0,0))

        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                run=False

        for planet in planets:
            planet.update_pos(planets)
            planet.draw(WIN)

        pygame.display.update()

    pygame.quit()

main()
```

3.2 Output



3.3 GitHub Link

https://github.com/RVirinchi/Solar-System-Simulation.git

4 Gravitational Sling Shot Simulation

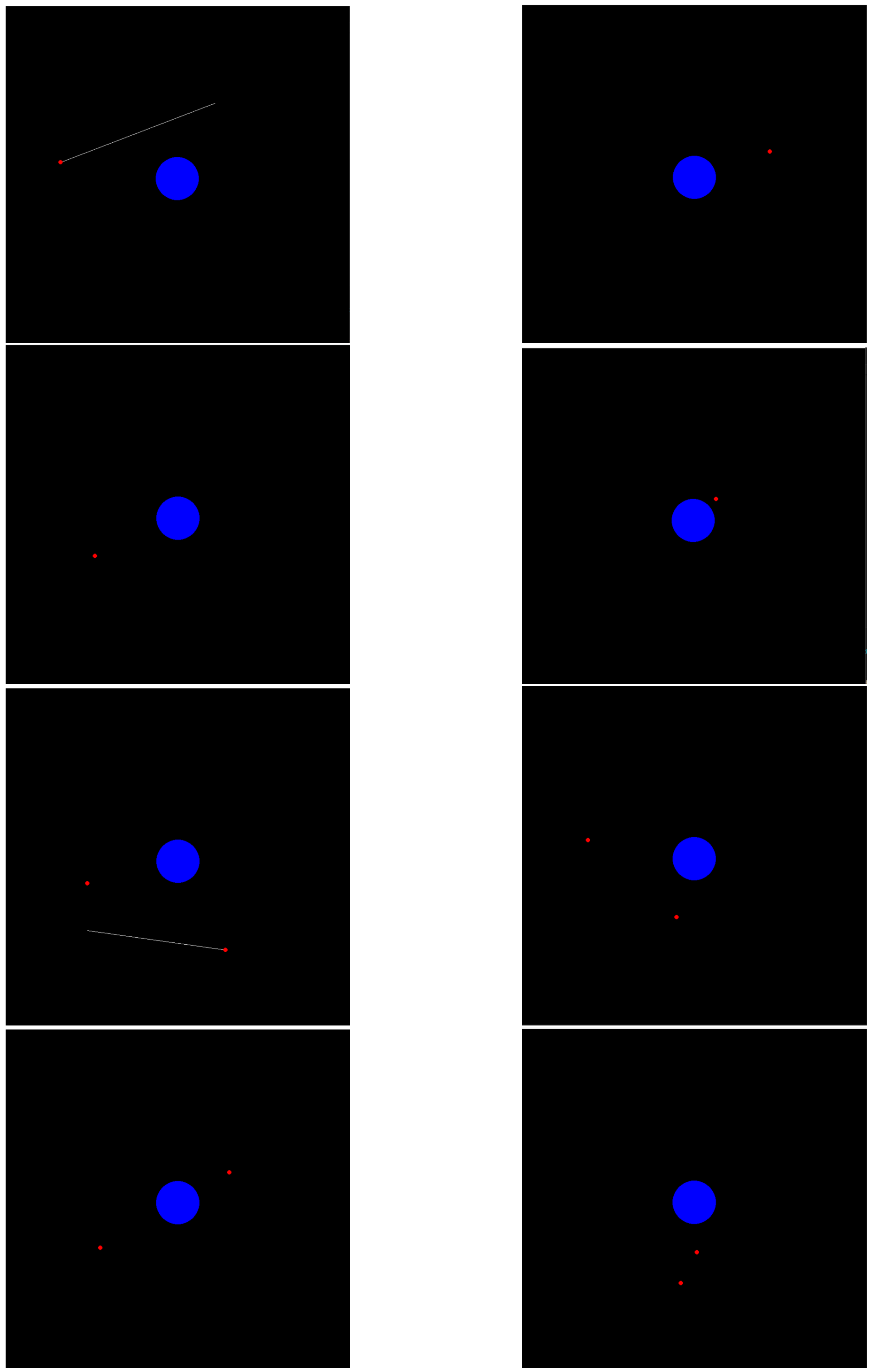
4.1 Python code

```
import pygame
import math

pygame.init()

WIDTH, HEIGHT = 800,800
win = pygame.display.set_mode((WIDTH,HEIGHT))
pygame.display.set_caption('gravitational sligshot
simulation')

PLANET_MASS=100
SHIP_MASS=2
G=10
FPS = 60
PLANET_SIZE=50
OBJ_SIZE=5
VEL_SCALE=100
WHITE = (255,255,255)
RED = (255,0,0)
BLUE=(0,0,255)

class Planet:
    def __init__(self,x ,y ,mass):
        self.x=x
        self.y=y
        self.mass=mass

    def draw(self):
        pygame.draw.circle(win, BLUE, (self.x, self.y),
        PLANET_SIZE)

class Spacecraft:
    def __init__(self, x, y, vel_x, vel_y, mass):
        self.x=x
        self.y=y
        self.vel_x=vel_x
        self.vel_y=vel_y
        self.mass=mass
```

```
    def draw(self):
        pygame.draw.circle(win, RED, (int(self.x),
        int(self.y)), OBJ_SIZE)

    def move(self,planet=None):
        distance = math.sqrt((self.x-planet.x)**2+(self.y-
        planet.y)**2)
        force = (G*self.mass * planet.mass)/distance**2
        acceleration = force/self.mass
        angle=math.atan2(planet.y-self.y,planet.x-self.x)

        acceleration_x=acceleration*math.cos(angle)
        acceleration_y=acceleration*math.sin(angle)

        self.vel_x+=acceleration_x
        self.vel_y+=acceleration_y

        self.x+=self.vel_x
        self.y+=self.vel_y



def create_ship(Location, mouse):
    t_x, t_y = Location
    m_x, m_y = mouse
    vel_x=(m_x-t_x)/VEL_SCALE
    vel_y = (m_y-t_y)/VEL_SCALE
    obj = Spacecraft(t_x, t_y, vel_x, vel_y, SHIP_MASS)
    return obj

def main():
    running = True
    clock = pygame.time.Clock()

    objects=[]
    temp_obj_pos = None

    while running:
        clock.tick(FPS)
        win.fill((0,0,0))

        planet = Planet(WIDTH//2, HEIGHT//2, PLANET_MASS)

        mouse_pos = pygame.mouse.get_pos()
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
```

```
                running=False

            if event.type==pygame.MOUSEBUTTONDOWN:
                if temp_obj_pos:

                    obj=create_ship(temp_obj_pos,
                    mouse_pos)
                    objects.append(obj)
                    temp_obj_pos = None
                else:
                    temp_obj_pos = mouse_pos

        if temp_obj_pos:
            pygame.draw.line(win, WHITE, temp_obj_pos,
            mouse_pos, 1)
            pygame.draw.circle(win, RED, temp_obj_pos,
            OBJ_SIZE)

        for obj in objects[:]:
            obj.draw()
            obj.move(planet)
            off_screen = obj.x < 0 or obj.x >WIDTH or
            obj.y<0 or obj.y>HEIGHT
            collided = math.sqrt((obj.x - planet.x)**2+
            (obj.y-planet.y)**2) <= PLANET_SIZE
            if off_screen or collided:
                objects.remove(obj)
        planet.draw()
        pygame.display.update()
    pygame.quit()

if __name__ == '__main__':
    main()
```

4.2 Output



4.3 GitHub Link

https://github.com/RVirinchi/Gravitational-Slingshot-Simulation.git

5 Summary

Python's versatility, ease of use, and the availability of specialized libraries make it a powerful tool for astronomers engaged in simulating various astronomical processes, contributing significantly to our understanding of the dynamic and evolving nature of the universe.