MULTIMEDIA PROJECT DOCUMENTATION:

TOPIC:-

PHYSICS SIMULATION:-

SOLAR SYSTEM SIMULATION USING PYGAME.

SUBMITTED BY:-

Chetanya Agarwal (211141)N - Documentation

SUBMITTED TO:-

Mr. Ayush Sharma

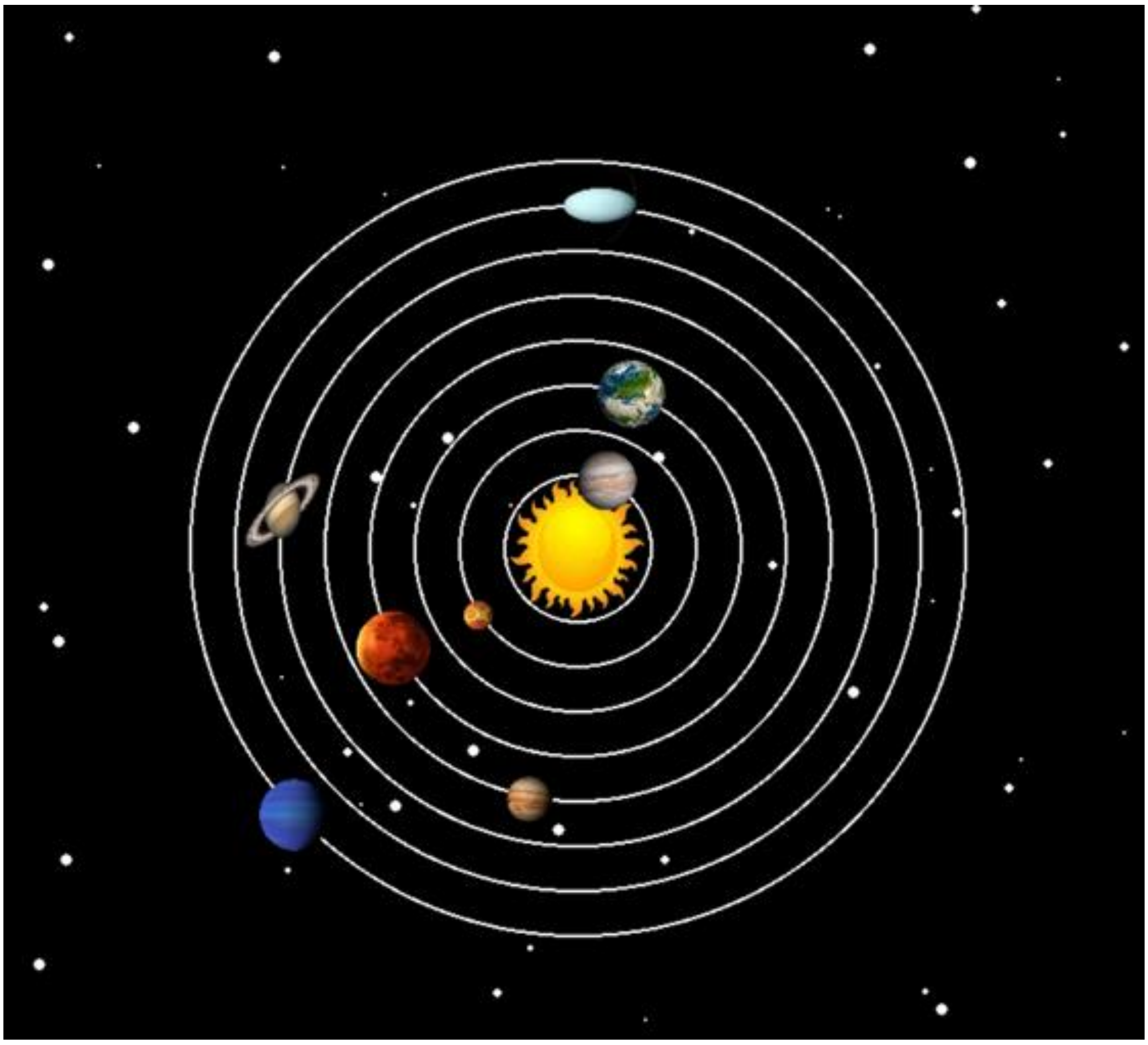
INDEX

ACKNOWLEDGEMENTS:

While crafting this Solar System Simulation project using Pygame, I extend my heartfelt appreciation to my mentor and my respected teacher Mr. Ayush Sharma for their unwavering guidance and support. Their insightful inputs played a pivotal role in shaping the trajectory of our simulation, turning it into a comprehensive and engaging endeavour. A warm shoutout goes to my friends, whose collaborative spirit and shared enthusiasm added a layer of joy to the entire process. From intense brainstorming sessions to marathon debugging, their contributions went beyond technical support, making this journey a memorable and enjoyable one. Additionally, I express my gratitude to other supporters who helped, inspiration, or simply a friendly ear during the highs and lows of this project. This endeavour has been a collective effort, and I am immensely thankful for the shared moments of coding and problem-solving that brought the solar system to vibrant life on the screen. Cheers to the collaborative spirit that made this project a success.

INTRODUCTION:

We're delving into the creation of a Solar System Simulation using Pygame. This project takes us on a journey through space right from our computer screens. Throughout this report, we'll walk you through the steps and intricacies of bringing the solar system to life with code. So, let's dive in and explore the universe in a whole new light.



ABSTRACT:

The code is a Python program that uses the pygame library to create a simple solar system simulation. It generates stars, displays the sun, and animates the planets in their respective orbits. Each planet has its image, and their movement is based on trigonometric functions.

LIBRARIES USED:

pygame: Used for creating the graphical interface, handling events, and displaying images.

random: Used to generate random coordinates for stars.

math: Used for trigonometric calculations to simulate the planets' circular motion.

SOURCE CODE:

```
import random
import pygame
from pygame.locals import *
import math

pygame.init()

# Set up the Pygame window
width = 800
height = 800
WHITE = [255, 255, 255]
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Solar System")

# Initialize stars
stars = []
```

```
for i in range(60):
    x = random.randrange(0, width)
    y = random.randrange(0, height)
    stars.append([x, y])

# Initialize planet orbit angles
angle1 = 0
angle2 = 0
angle3 = 0
angle4 = 0
angle5 = 0
angle6 = 0
angle7 = 0
angle8 = 0

# Load planet images and set initial positions
sun =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\sun.png")
sun = pygame.transform.scale(sun, (100, 100))

mercury =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\P3_Multimedia\\Pla
net_Images\\mercury.png")
mercury = pygame.transform.scale(mercury, (40, 40))
mRect = mercury.get_rect()

venus =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\venus.png")
venus = pygame.transform.scale(venus, (20, 20))
vRect = venus.get_rect()

earth =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\earth.png")
earth = pygame.transform.scale(earth, (50, 50))
eRect = earth.get_rect()
```

```
mars =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\mars.png")
mars = pygame.transform.scale(mars, (50, 50))
msRect = mars.get_rect()

jupiter =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\jupiter.png")
jupiter = pygame.transform.scale(jupiter, (50, 50))
jRect = jupiter.get_rect()

saturn =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\Pla
net_Images\\saturn.png")
saturn = pygame.transform.scale(saturn, (50, 50))
sRect = saturn.get_rect()

uranus =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\uranus.png")
uranus = pygame.transform.scale(uranus, (50, 50))
uRect = uranus.get_rect()

neptune =
pygame.image.load("C:\\Users\\agarw\\OneDrive\\Desktop\\P3_Multimedia\\Pl
anet_Images\\neptune.png")
neptune = pygame.transform.scale(neptune, (50, 50))
nRect = neptune.get_rect()

clock = pygame.time.Clock()
done = False

# Main game loop
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    # Draw background stars
    screen.fill((0, 0, 0))
    for i in range(len(stars)):
        pygame.draw.circle(screen, (255, 255, 255), stars[i], random.randrange(1,
5))
        stars[i][1] += 1
        if stars[i][1] > height:
            y = random.randrange(-50, -10)
            stars[i][1] = y
            x = random.randrange(0, 800)
            stars[i][0] = x

    # Draw planetary orbits
    p1 = pygame.draw.circle(screen, WHITE, (width // 2, height // 2), 260, 2)
    # ... (similar drawing for other orbits)

    # Draw and update the position of each planet
    screen.blit(sun, (350, 350))
    mRect.center = (50 * math.cos(angle1) + 400, 50 * math.sin(angle1) + 400)
    screen.blit(mercury, mRect)
    vRect.center = (80 * math.cos(angle2) + 400, 80 * math.sin(angle2) + 400)
    screen.blit(venus, vRect)
    eRect.center = (110 * math.cos(angle3) + 400, 110 * math.sin(angle3) + 400)
    screen.blit(earth, eRect)
    msRect.center = (140 * math.cos(angle4) + 400, 140 * math.sin(angle4) +
400)
    screen.blit(mars, msRect)
    jRect.center = (170 * math.cos(angle5) + 400, 170 * math.sin(angle5) + 400)
    screen.blit(jupiter, jRect)
    sRect.center = (200 * math.cos(angle6) + 400, 200 * math.sin(angle6) + 400)
    screen.blit(saturn, sRect)
    uRect.center = (230 * math.cos(angle7) + 400, 230 * math.sin(angle7) + 400)
    screen.blit(uranus, uRect)
    nRect.center = (260 * math.cos(angle8) + 400, 260 * math.sin(angle8) + 400)
    screen.blit(neptune, nRect)

    # Increment angles for the next frame
    angle1 += 0.06
    angle2 += 0.03
    angle3 += 0.01
    angle4 += 0.009
    angle5 += 0.006
    angle6 += 0.004
    angle7 += 0.002
    angle8 += 0.001

    # Control the frame rate
    clock.tick(60)
    pygame.display.update()

# Quit Pygame when the loop is done
pygame.quit()
```

CODE STRUCTURE:

Initialization:
- Pygame.init(): Initializes the pygame library.
- Set up the window with dimensions and a caption.

Images:
- Load images for the sun and each planet. Scale them down to appropriate sizes.

Stars Initialization:
- Generate random coordinates for stars and store them in a list.

Main Loop:
- Handle pygame events, such as quitting the program.
- Update star positions, simulate them falling and reset when they go off-screen.

Planetary Orbits:
- Use trigonometric functions to calculate the positions of planets in their orbits.
- Update planet positions in the main loop.

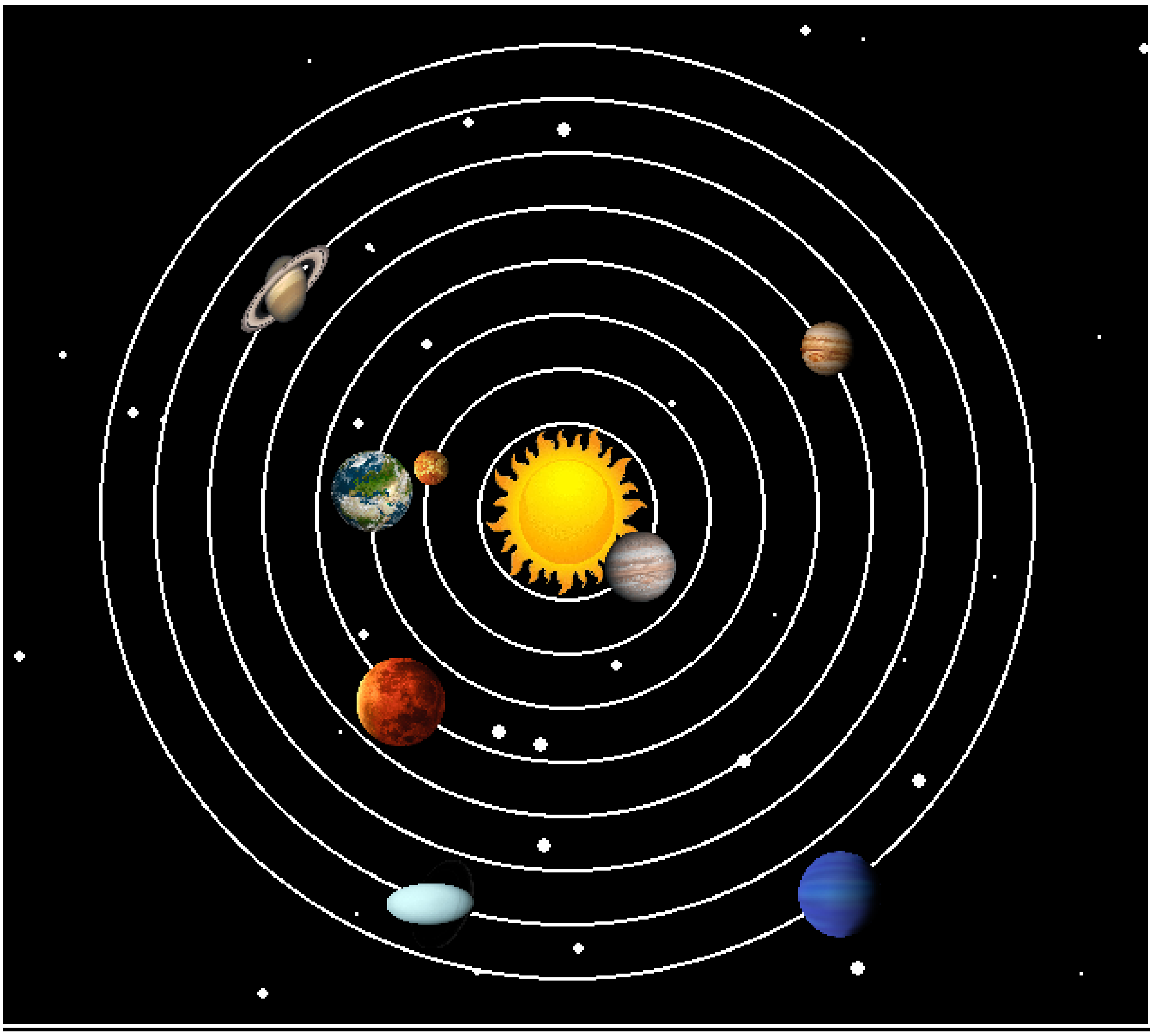
Drawing:
- Draw stars, planetary orbits, and the sun.
- Blit each planet's image at its calculated position.

Animation:
- Increment angles for each planet to create a sense of motion.

Clock and Display Update:
- Use pygame.time.Clock() to control the frame rate.
- Update the display within the main loop.

Quitting:
- Properly quit pygame when the main loop is exited.

OUTPUT:



REPORT:

Functionality: The code successfully simulates a basic solar system with stars, planets, and a moving sun.

Structure: The code is well-organized, separating initialization, image loading, main loop, and quitting sections.

Performance: The use of trigonometric functions might lead to performance issues with a larger number of planets or increased complexity.

Improvements: Consider encapsulating the functionality into functions or classes for better modularity.

SOFTWARE REQUIREMENTS SPECIFICATION (SRS):

Objective: Simulate a solar system with stars and planets.

Input: None (interactive, event-driven).

Output: Visual representation of a solar system.

Constraints: Dependent on pygame and random library. Requires appropriate image files for the sun and planets.

COMPILER AND IDE:

Compiler: Python is an interpreted language; no compilation is needed.

IDE: The code appears to be written in a code editor (like VS Code).

LOW-LEVEL DESIGN (LLD) FOR THE SOLAR SYSTEM SIMULATION:

Modules/Components:

Initialization Module:
- Responsible for initializing pygame and setting up the window.

Image Handling Module:
- Loads and scales images for the sun and planets.

Starry Backdrop Module:
- Generates and updates the positions of stars.

Main Loop Module:
- Handles pygame events and orchestrates the overall flow of the simulation.

Planetary Motion Module:
- Manages the trigonometric calculations for planetary motion.

Drawing Module:
- Draws stars, planetary orbits, and blits planet images onto the screen.

Animation Module:
- Updates angles for planetary motion, creating the illusion of movement.

Timekeeping Module:
- Utilizes pygame's clock to maintain a consistent frame rate.

Conclusion Module:
- Ensures proper shutdown of pygame when the simulation ends.

Interactions/Flow:

Initialization Flow:
- Initialize pygame.
- Set up the window.

Image Handling Flow:
- Load and scale images for the sun and planets.

Starry Backdrop Flow:
- Generate random star coordinates.
- Update star positions, creating a falling effect.

Main Loop Flow:
- Handle pygame events, especially the quit event.
- Update star positions in the main loop.
- Update planetary positions through the Planetary Motion Module.
- Draw the cosmic scene using the Drawing Module.
- Add animation by updating angles with the Animation Module.
- Keep time with the Timekeeping Module.
- Continue looping until the quit event occurs.

Conclusion Flow:
- Properly shut down pygame.

Data Structures:

Stars List:
- List of (x, y) coordinates representing star positions.

Image Objects:
- Objects representing the loaded and scaled images for the sun and planets.

Angles:
- Variables storing the angles for each planet's motion.

Functions/Methods:

Initialization Methods:

- **initialize_pygame:** Initializes pygame.
- **setup_window:** Sets up the window.

Image Handling Methods:

load_and_scale_images: Loads and scales images for the sun and planets.

Starry Backdrop Methods:

- generate_stars: Generates initial star positions.
- update_stars: Updates star positions.

Main Loop Methods:

- **handle_events:** Handles pygame events.
- **main_loop:** Orchestrates the overall flow of the simulation.

Planetary Motion Methods:

calculate_planetary_positions: Calculates the positions of planets based on angles.

Drawing Methods:

draw_stars: Draws stars on the screen.
draw_planetary_orbits: Draws planetary orbits.
blit_planet_images: Blits planet images onto the screen.

Animation Methods:

update_angles: Updates angles for planetary motion.

Timekeeping Methods:

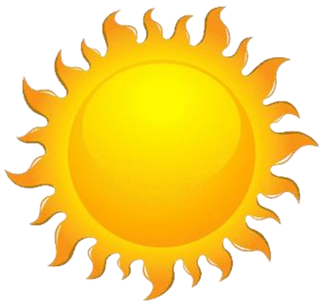
keep_time: Controls the frame rate using pygame's clock.

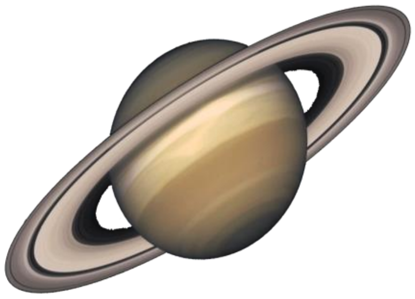
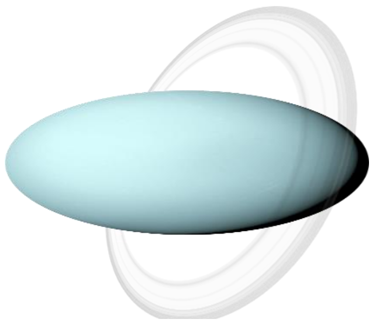
Conclusion Methods:

quit_pygame: Properly shuts down pygame.

This Low-Level Design outlines the different modules, flows, data structures, and methods that make up the solar system simulation code.

IMAGES USED IN CODE:



CONCLUSION:

The code provides a captivating visual representation of a solar system using pygame. It's a great starting point for further enhancements, such as adding more planets, moons, or interactive features.

REFERENCES:

- Pygame Documentation - Official documentation for Pygame
https://www.pygame.org/doc/

- NASA Solar System Exploration - Invaluable resource for accurate planetary data
https://solarsystem.nasa.gov/planets/

- "Invent Your Own Computer Games with Python" by Al Sweigart - Useful for game development with Python
https://inventwithpython.com/invent4thed/