Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Proposal

on

“Bouncing Ball and Pendulum Simulation”

[Code No: COMP 342]

Submitted by

Abiral Adhikari (02)

Mahip Adhikari (03)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 15/06/2024

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Project Background

Our project addresses the challenge of visualizing complex Newtonian physics concepts by creating interactive simulations of bouncing balls and pendulums. These simulations will allow us to observe and analyze collisions, trajectories, and oscillatory motions in ways that are impossible to achieve practically. The bouncing ball simulation will facilitate detailed observations of elastic and frictionless collisions involving multiple balls of varying sizes and masses within a confined space. Similarly, the pendulum simulation will enable us to explore oscillatory motion under different conditions, such as varying lengths, masses, and gravitational forces, even allowing us to simulate a pendulum on Mars.

By developing these simulations, we aim to bridge the gap between theoretical physics and practical visualization, enhancing our understanding of these classic mechanics concepts. This project not only deepens our knowledge of physics but also provides us with hands-on experience in computer graphics, aligning with the objectives of our course 342. Through this endeavor, we transcend the limitations of textbook learning and gain a dynamic, interactive perspective on the fascinating world of physics.

1.2 Objectives

- To simulate and observe the nature of collision of the balls bouncing in a closed space
- To simulate and observe the motion and time taken by a pendulum of different length and mass in different gravity.
- To learn and explore the concepts of computer graphics using OpenGL graphics

1.3 Expected Outcomes

- A visual demonstration of pendulum's oscillatory motion for varying mass, length, and gravity, with the ability to simulate multiple pendulums simultaneously.
- A visual demonstration of ball collision simulation in a closed space, allowing user control over the number, speed, and elasticity of the balls.
- The simulation will show a real-time graph of displacement vs. time for the pendulum(s) during the simulation.
- It will serve as a valuable instructional tool for college students, educators, and professionals in physics, engineering, and related fields.
- The project will provide a basis for future enhancements, including integration with educational curricula and expansion into other mathematical concepts.

Chapter 2 Library and Language used.

2.1 OpenGL

OpenGL serves as a powerful tool for creating dynamic and visually appealing simulations, such as pendulum swings and bouncing ball animations. Known for its proficiency in rendering both 2D and 3D graphics, OpenGL provides a robust framework for generating complex visual representations. Utilizing OpenGL enables animators to achieve smooth motion, realistic rendering, and interactive elements, enhancing the overall user experience of simulations like pendulum swings and bouncing ball dynamics. Developing realistic simulations of these concepts with OpenGL requires a solid grasp of computer graphics principles and OpenGL programming skills.

2.2 Python:

Python is a versatile and efficient programming language renowned for its ability to build a wide array of applications. It provides a flexible framework with extensive libraries and modules tailored to diverse use cases. Python supports object-oriented programming with features like classes, instance variables, methods, operator overloading, and function overloading, facilitating data abstraction and efficient memory management. Known for its dynamic typing and high-level constructs, Python is highly favored for developing applications ranging from web development and data analysis to scripting and automation tasks. Its simplicity and readability further contribute to its popularity among developers.

Chapter 3 Methodology

In this project the pendulum simulation aims to visually demonstrate the physics of pendulum motion through interactive animation approach while the bouncing ball simulation aims to visually depict the dynamics of a bouncing ball through interactive animation. The methodology involves the following steps:

3.1 Physics and Mathematical Modeling:

3.1.1 Pendulum Oscillatory Motion Modeling:

A simple pendulum consists of a mass (bob) suspended by a string or rod, experiencing gravity and tension. The simulation assumes small-angle oscillations, where the pendulum's motion is described by the differential equation:

$$d^2\theta/dt^2 + (g/L) \sin(\theta) = 0$$

where g represents gravitational acceleration and L represents the pendulum's length. For small angles, $\sin(\theta)$ can be approximated as θ, simplifying the equation to:

$$d^2\theta/dt^2 + (g/L) \theta = 0$$

The solution to this equation is $\theta(t) = \theta_0 \cos(\omega t)$, where θ_0 is the initial angular displacement and $\omega = \sqrt{(g/L)}$ is the angular frequency. This model accurately captures the pendulum's harmonic motion, visualized through real-time animation within the simulation.

3.1.2 Bouncing Ball Elastic Collision Modeling:

- **Motion Equations**

The motion of each ball in the simulation is governed by Newton's second law of motion. This law states that the sum of forces acting on an object is equal to its mass times acceleration ($F = ma$).

For a ball of mass m moving in two dimensions with velocity $v = (v_x, v_y)$, subjected to gravitational acceleration g, the equations of motion are:

$d^2x/dt^2 = 0$

$d^2y/dt^2 = -g$

These equations tell us:

Horizontal Motion: The horizontal acceleration ($d^2x/dt^2 = 0$) is zero, implying constant velocity in the x-direction for the ball in the absence of external forces.

Vertical Motion: The vertical acceleration ($d^2y/dt^2 = -g$) reflects the gravitational force pulling the ball downward with an acceleration of g.

- **Elastic Collisions with Wall:**

When a ball collides with a vertical wall at its boundaries (x = 0 or x = display width), its x-component of velocity reverses after the collision:

$vx' = -vx$

$vy' = vy$

Here:

vx and vy are the components of the ball's velocity before collision.

vx' and vy' are the components after collision.

- **Collision between Two Balls:**

When two balls collide, their velocities after collision (v_1' and v_2') are calculated using these equations:

v1' = (m1 - m2) v1 + 2 m2 v2 / (m1 + m2)

v2' = (m2 - m1) v2 + 2 m1 v1 / (m1 + m2)

Here:

m1 and m2 are the masses of the two balls.

v1 and v2 are their velocities before collision.

v1' and v2' are their velocities after collision.

3.2 Implementation:

3.2.1 Pendulum Simulation Function Implementation:

The pendulum simulation for the project is done by implementing the physics and mathematics for the pendulum. The pendulum is represented by drawing it on the window using the details stored in the class Pendulum. Pendulum Class represents each pendulum with attributes such as length, mass, initial angle, gravity, and color. The class includes methods to update the pendulum's angle over time (update), calculate its position (get_position), and store data for plotting the angle versus time graph.

The pendulum is then drawn in window for a instance of the pendulum at a instance of timeusing the draw_pendulum(pendulum) function. draw_pendulum(pendulum) draws the pendulum's rod and bob using OpenGL primitives (GL_LINES and GL_POLYGON), with color specified by the pendulum's blob_color.

The graph of the pendulum simulation is drawn by passing all the instance of pendulums to draw_graph(pendulums) function. draw_graph(pendulums)plots the angle versus time graph for all pendulums using GL_LINE_STRIP, visualizing the harmonic motion.

pendulumsimulation() function is the main function that initializes Pygame and OpenGL, sets up the simulation environment (projection and model view matrices), and continuously updates and renders the pendulum animations and graphs based on user-defined parameter.

3.2.2 Pendulum Simulation Program Flow:

The program begins by initializing Pygame and configuring an OpenGL display window for interactive simulation. Users are prompted to input the number of pendulums along with their physical attributes such as length, mass, initial angle, gravity, and chosen color. Inside the main simulation loop (while True), Pygame manages user interactions, ensuring smooth operation including handling events like program termination requests. OpenGL is utilized to clear the display screen and render visual representations of the pendulums, each depicted with its own angle versus time graph displayed in separate viewports. The Pendulum class governs the dynamics of each pendulum, updating its angular position using the principles of harmonic motion derived from the simple pendulum equation. Drawing functions leverage OpenGL primitives to illustrate the pendulum's bob and rod, ensuring accurate graphical depiction based on the specified parameters. This interactive simulation loop persists until the user opts to exit the program, offering a dynamic and educational showcase of pendulum dynamics grounded in mathematical modeling principles.

3.2.3 Bouncing Ball Simulation Implementation:

The Ball is represented in the program with Ball class that defines the attributes and behaviors of a bouncing ball within the simulation. It initializes each ball with parameters such as position (x, y), velocity (vx, vy), radius (radius), color (color), and elasticity (elasticity). The move method updates the position based on velocity and handles collisions with the display boundaries, ensuring the ball bounces realistically. The draw method uses OpenGL to visually render the ball as a filled circle with specified color and radius.

The check_collision(ball1, ball2) function detects collisions between two balls (ball1 and ball2). It computes the distance between their centers and checks if it's less than the sum of their radii. Upon collision, it calculates new velocities based on conservation of momentum and adjusts positions to prevent overlap, ensuring realistic interaction.

The random_balls(num_balls, elasticity, speed_factor) function generates num_balls with random positions, velocities, sizes, and colors. Each ball's initial parameters are randomized within specified ranges, providing diverse initial conditions for dynamic simulation.

The manual_balls(num_balls, elasticity, speed_factor) function prompts users to input parameters (radius, launch location) for each ball. Based on the chosen launch location, it assigns initial positions and velocities. This allows users to customize the starting conditions of each ball within the simulation.

3.2.4 Bouncing Ball Simulation Program Flow:

The bouncing ball program implements a bouncing ball simulation using Pygame and OpenGL. Upon execution, it initializes a Pygame window with specified dimensions and sets up an OpenGL orthographic projection to match the window size. The user is prompted to choose between manual or random initialization of multiple bouncing balls, each characterized by attributes such as radius, initial velocity, color, elasticity, and speed factor.

In the main simulation loop, the program continuously updates and renders the balls' positions and velocities. It detects collisions between balls using geometric calculations based on their radii and distances, adjusting velocities according to the principles of elastic collisions.Users can exit the program by entering '3' when prompted for the simulation mode. This structure ensures interactive and visually appealing simulation of bouncing balls governed by realistic physics within the defined Pygame and OpenGL environment.

3.2.5 Full Project Program Flow:

When the program runs the user is prompted to choose one of the simulation and after simulation is done they can choose to conduct the simulation again for another simulation with different variables.

3.3 Results and Output

The screenshots from the instances of simulation are provided as reference to successful execution of simulation.

3.3.1 Pendulum Simulation:

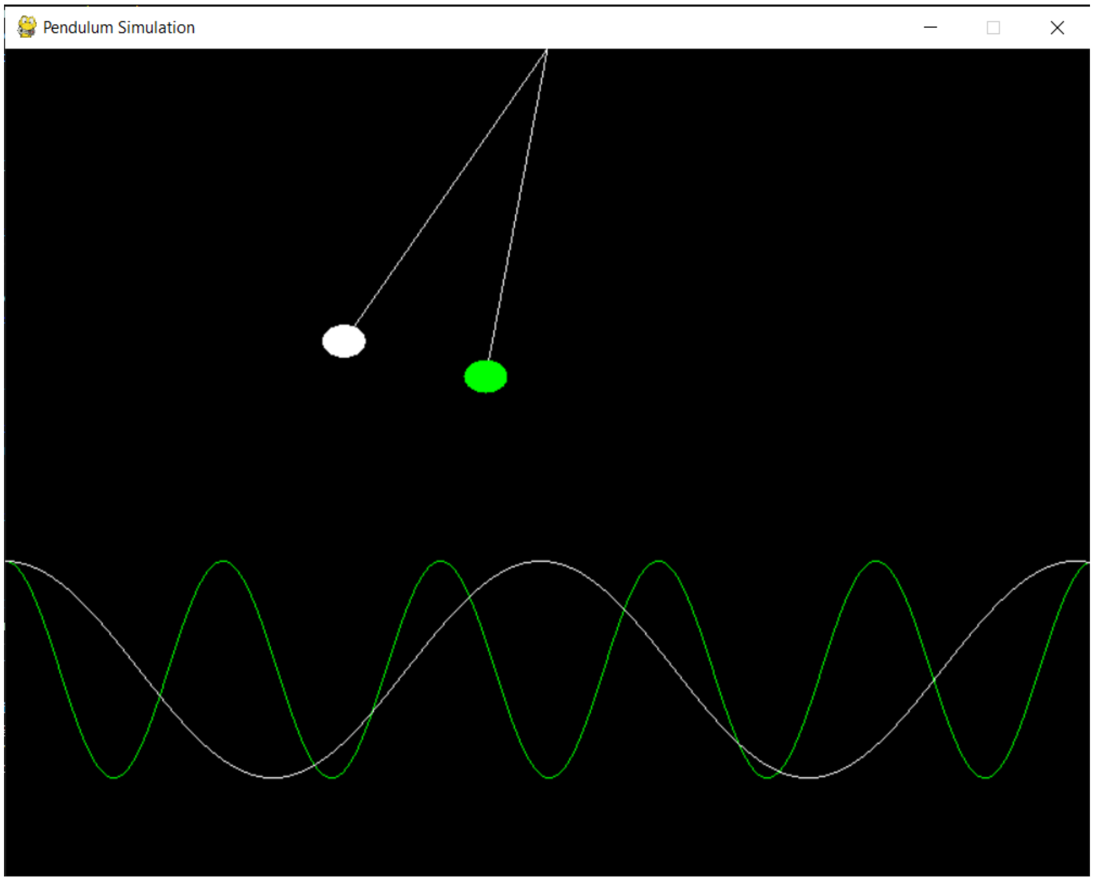


Figure 0.1-1 Pendulum Simulation for gravity at earth(green) and moon(white) for pendulum of same

3.3.2 Bouncing Ball Simulation:

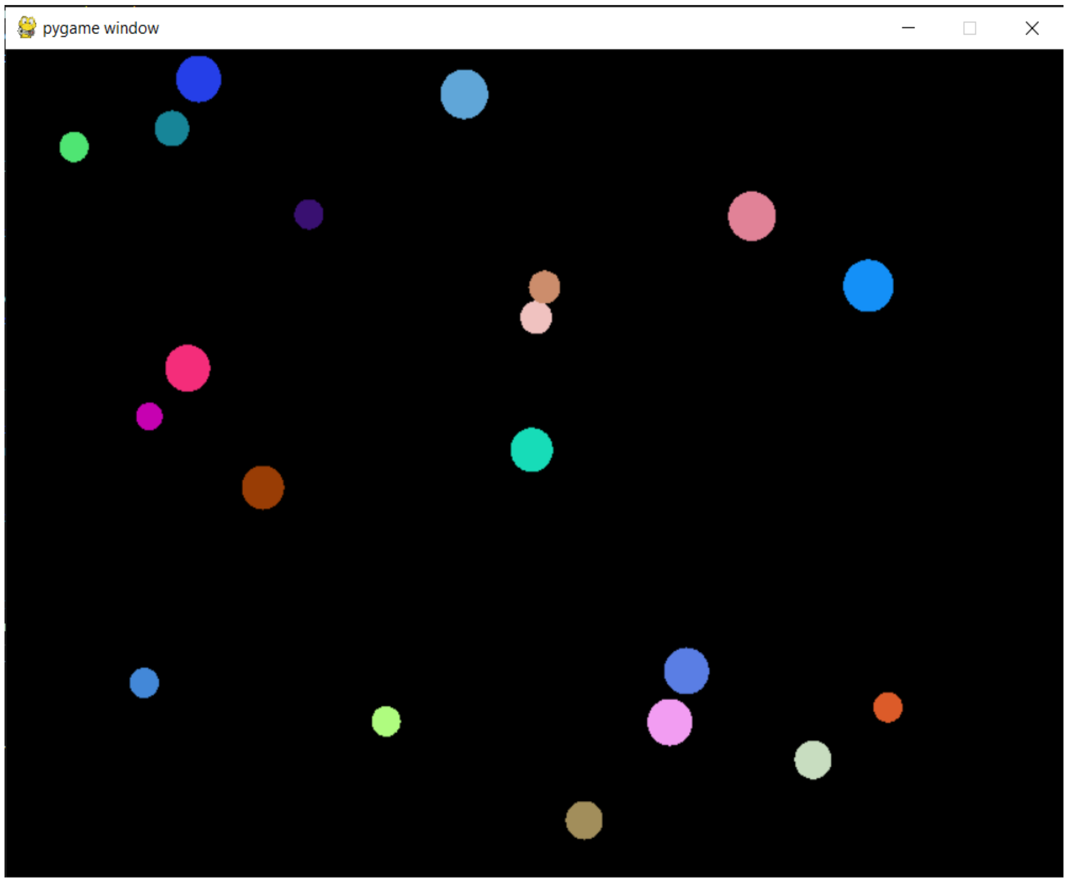


Figure 0.2-1 Bouncing Ball Simulation with 20 balls

Chapter 4 Conclusion:

In this way that graphics mini project was completed using PyOpengl showing the simulation of simple pendulums and bouncing balls elastic collision. The project code is available at https://github.com/abiral-adhikari/Pendulum-and-BouncingBall-Simulation.

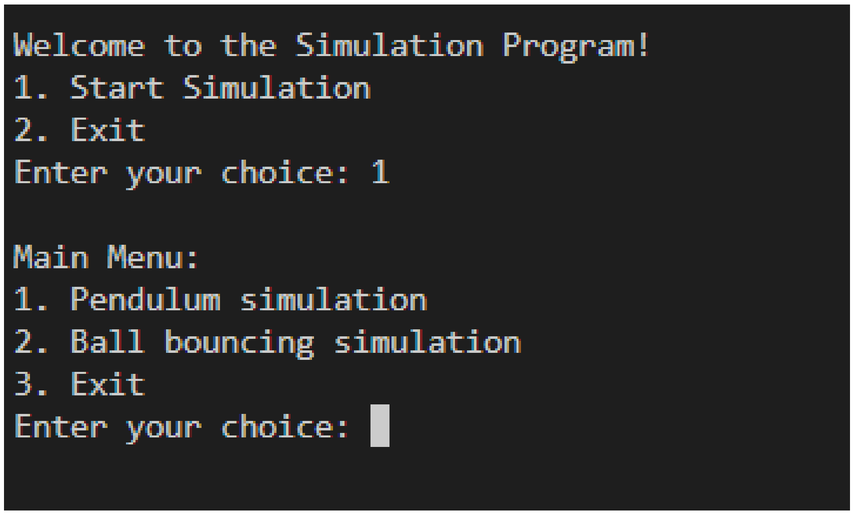
4.1 Limitations:

- In pendulum simulation the graph rendering is not seen when the graph leaves the right end of the window
- In bouncing ball simulation, the window size cannot accommodate large number of balls at same time
- Using many pendulums at same time is computationally heavy.
- All the user inputs are taken from terminal which is not optimal

4.2 Future Enhancement:

- Enable both a automatic default simulation and manual simulation for user.
- Put the input field and menu for user in the simulation window itself
- To convert the 2D simulation to 3D simulation

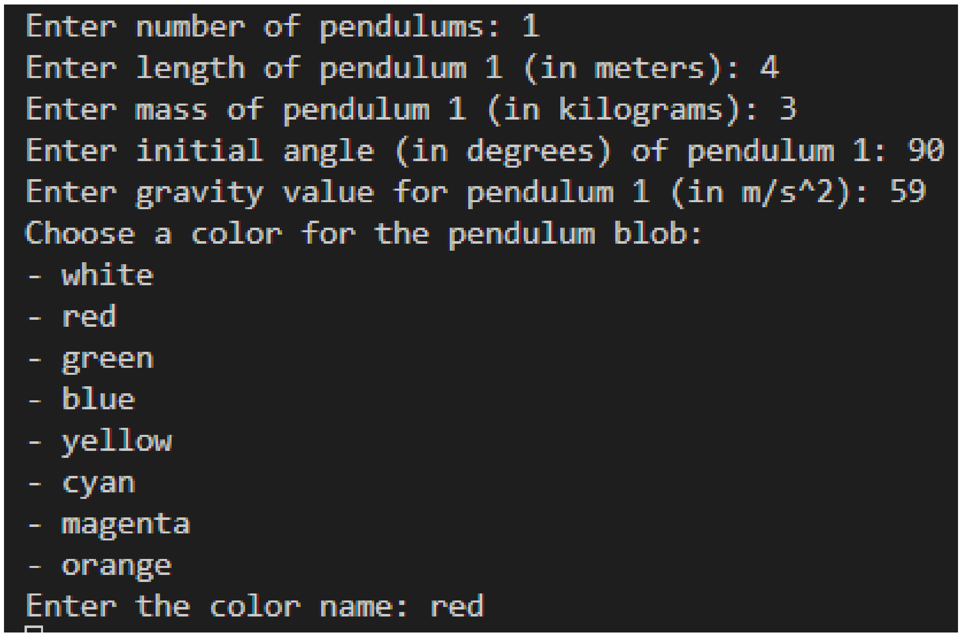
APPENDIX



```
Welcome to the Simulation Program!
1. Start Simulation
2. Exit
Enter your choice: 1

Main Menu:
1. Pendulum simulation
2. Ball bouncing simulation
3. Exit
Enter your choice:
```

Figure 0.1 Main Menu



```
Enter number of pendulums: 1
Enter length of pendulum 1 (in meters): 4
Enter mass of pendulum 1 (in kilograms): 3
Enter initial angle (in degrees) of pendulum 1: 90
Enter gravity value for pendulum 1 (in m/s^2): 59
Choose a color for the pendulum blob:
- white
- red
- green
- blue
- yellow
- cyan
- magenta
- orange
Enter the color name: red
```

Figure 0.2 Pendulum Simulation Menu

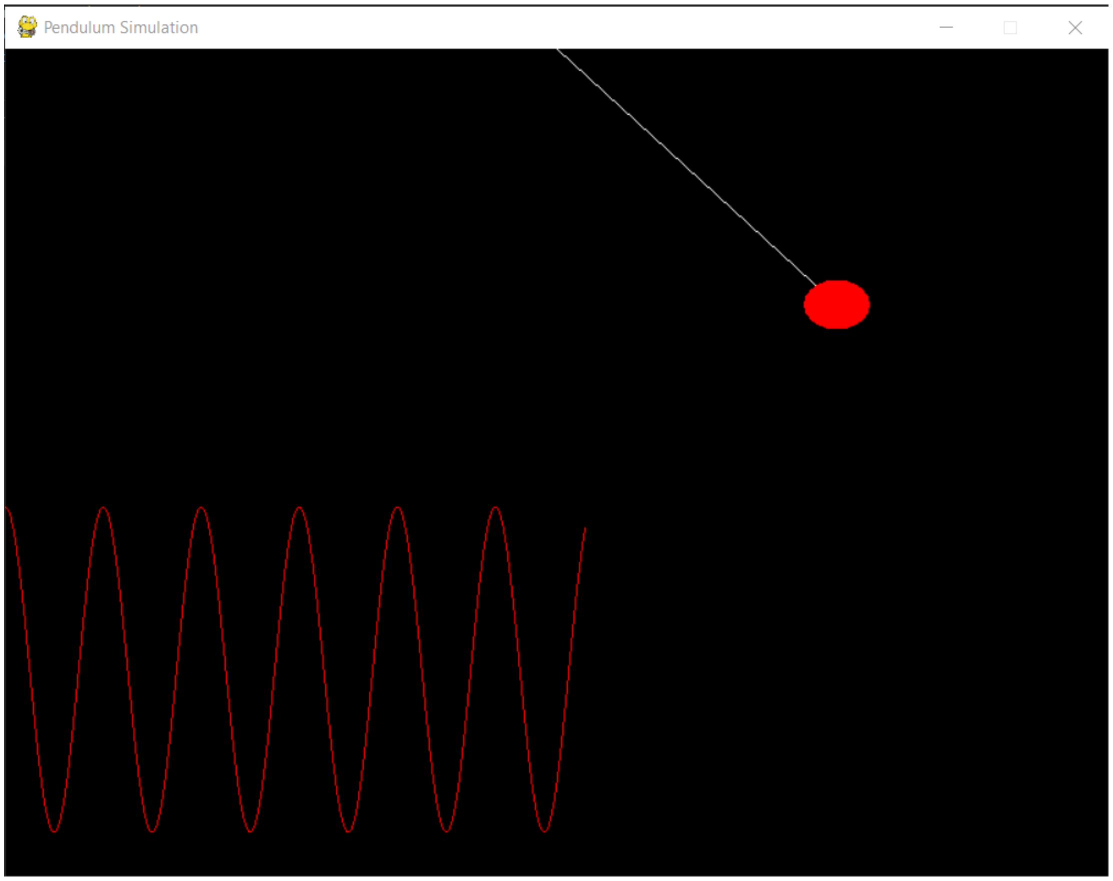
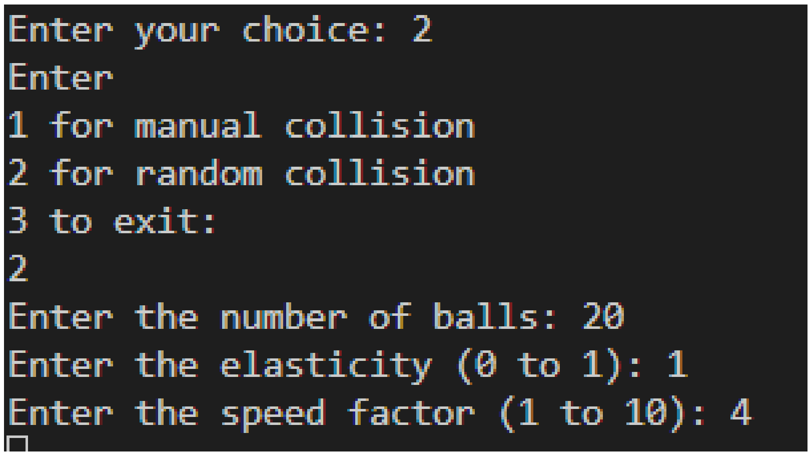


Figure 0.4 Pendulum Simulation



```
Enter your choice: 2
Enter
1 for manual collision
2 for random collision
3 to exit:
2
Enter the number of balls: 20
Enter the elasticity (0 to 1): 1
Enter the speed factor (1 to 10): 4
```

Figure 0.3 Ball Bouncing Simulation Menu

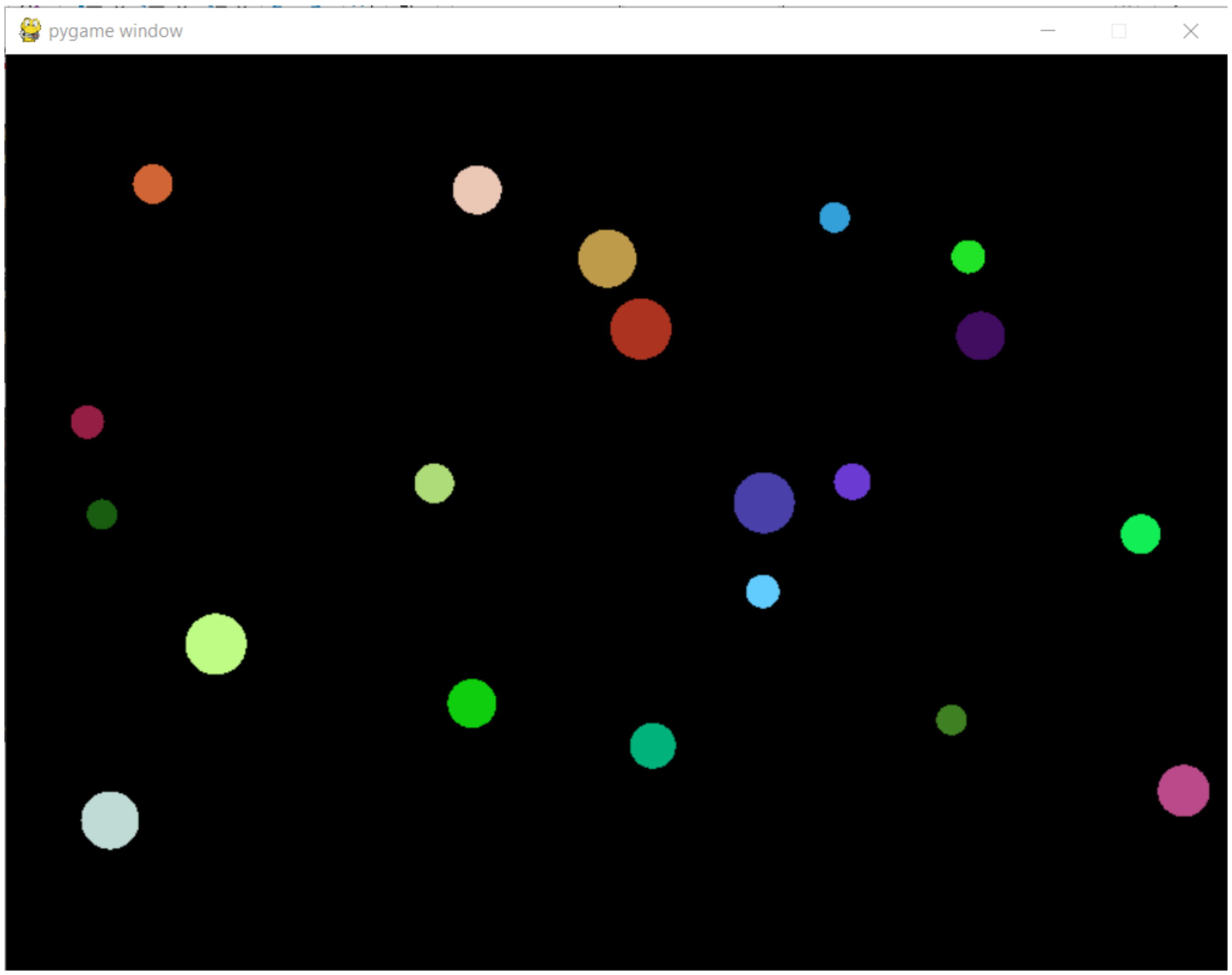
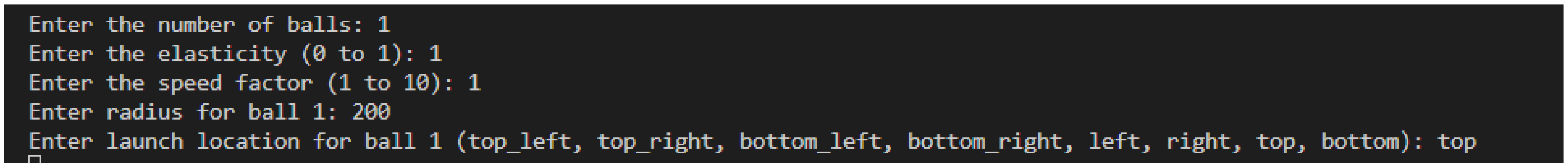


Figure 0.5 Random Bouncing Balls



```
Enter the number of balls: 1
Enter the elasticity (0 to 1): 1
Enter the speed factor (1 to 10): 1
Enter radius for ball 1: 200
Enter launch location for ball 1 (top_left, top_right, bottom_left, bottom_right, left, right, top, bottom): top
```

Figure 0.6 Manual Bouncing Ball Menu

Figure 0.7 Manual Bouncing Ball