CSCI 2081 Introduction to Software Development - Fall 2024
Homework 01 - Particle Simulation

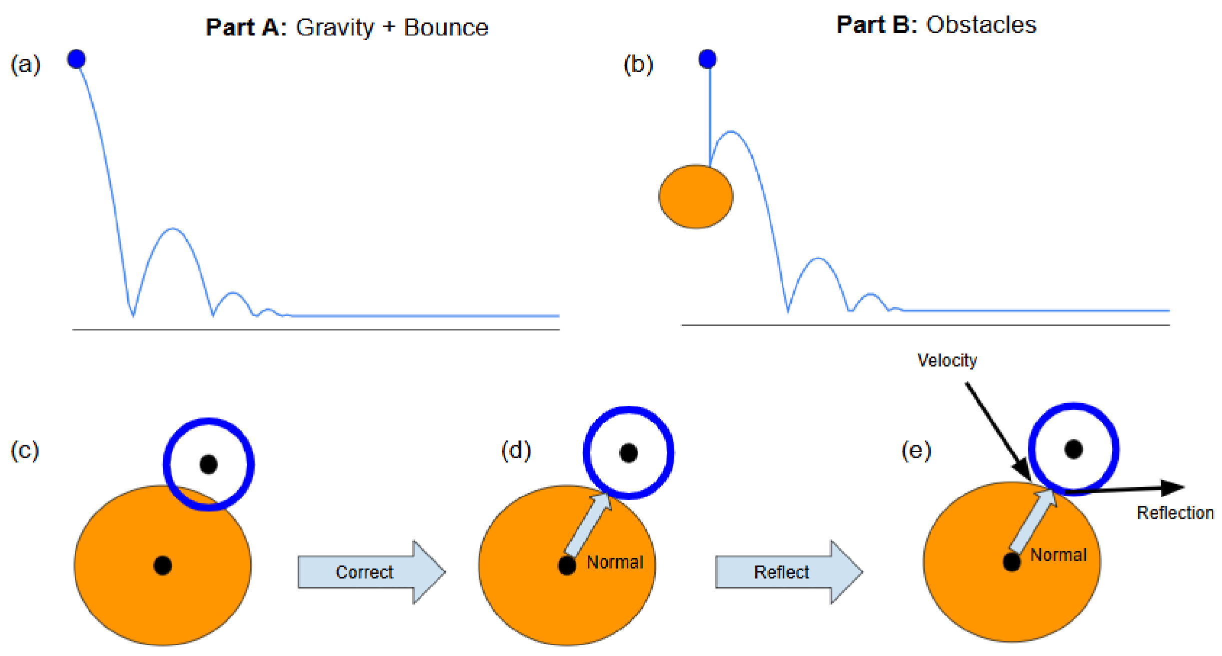

Figure 1 - Illustration of the development tasks for this assignment.

Due Date: Sunday, September 22, 2024 @ 11:59pm

Instructions: This homework assignment is a tutorial on how to write classes and methods for creating simulations. Specifically, this assignment implements the particle system below featured in Figure 1. **Part A**, featured in Figure 1(a) implements a particle that is affected by gravity and bounces against the ground. In **Part B**, represented by Figure 1(b), we will bounce the particle against an obstacle. Your task is to complete Part A and Part B by following the video tutorial here:

- **Video Tutorial:** https://umn.zoom.us/rec/share/7PiGynewueP3qVEZcsfuppqT7GZrSZpAlfQy7Apa9xkwx9iIhgl89pmaeXRbtht9.VF_GfHqwWUeGSXpB

Submission: Once complete, submit all your java files for your solution to Canvas. It is okay if your code looks exactly like the code from the tutorial.

Goal: Learn how to use object oriented programming to create a useful particle simulation that we can use for future data analysis assignments. Practice coding in Java to build programs.

UML Diagram of our Design

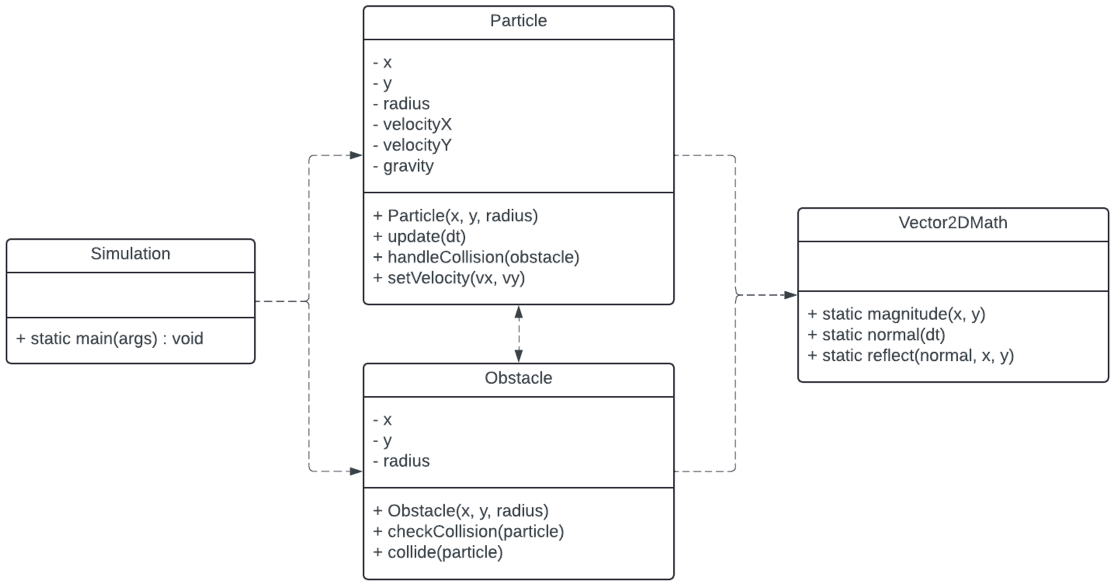

Figure 2 - UML Class Diagram showing our proposed solution. The Simulation creates a Particle and an Obstacle and runs the simulation. Vector2DMath has common math functions used throughout the program.

Part A: Gravity + Bounce

Task: Your first task is to create a particle that falls due to gravity and bounces against the ground. Figure 1(a) illustrates this task.

1. Open up the video tutorial above. Follow along and implement the solution on your machine (you will turn in your solution).

2. First create all the classes, member variables, and the methods in the UML Diagram.
    - Download the Vector2DMath.java file from the Canvas site.

3. Create getters and setters for the private member variables.

4. Implement the constructors by setting the private member variables using the `this` keyword. Also set the gravity to -1.0.

5. Implement the following:
    - Particle.setVelocity(vx,vy) - set the private velocity variables
    - Particle.update(dt)
        i. Update the downward velocity (velocityX += gravity*dt).
        ii. Update the particle's position by the velocity (Euler Integration)
            1. x += velocityX*dt
            2. y += velocityY*dt
        iii. Check to see if the particle hits the ground (y <= 0.0). If so, move it above ground and bounce: y *= -vy*(0.6). The 0.6 represents some loss of energy.

6. Update the Simulation:
    - Create a new particle.
    - Set the velocity of the particle to be a positive x value so the particle moves left to right.
    - Loop 100 times using a for loop. Inside the loop, call update(dt), where dt is some small timestep. Then print the x and y values for the particle.

7. Compile all classes and run the simulation. Copy and paste the x and y values and show the particle path over time. You should see a path as represented in Figure 1(a).

Part B: Obstacles

Task: Your task is to create an obstacle for the particle to bounce off of (see Figure 1b). The math in this section is a bit more complicated, but following the tutorial will help you here.

1. Implement the following methods:
    - **Obstacle.checkCollision(particle) -** See if the distance between the particle and the obstacle are less than the sum of the two radius values. See Figure 1(c) for an illustration.
    - **Obstacle.collide(particle) -** We need to correct the position of the particle since it should not be inside the obstacle (See Figure 1(d)). We do that by moving the particle to the edge of the obstacle using the calculated "normal" direction. We then return the normal to be used in the reflection calculation.
    - **Particle.handleCollision(obstacle) -** First check to see if there is a collision. If there is a collision, calculate the reflected vector (inside Vector2DMath) using the collision normal and the velocity vector (velocityX, velocityY). Update the velocity to be the reflected vector. If the velocity is too small, make it bounce by updating the velocity to be some scaler of the normal. See Figure 1(e) for an example.

2. In the Simulation:
    - Create an obstacle.
    - Inside the loop, call particle.handleCollistion(obstacle).

3. Compile, run, and visualize the simulation. Enjoy!