Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 1
Introduction to Programming in Java

Introduction

This assignment is based on Chapter 2 of the textbook by Eric Roberts, "Programming by Example". Even if you have never written a computer program before, it is possible to write useful programs by learning a few basic "patterns" as discussed in the chapter. In preparation for this assignment, make sure that you've read and understand the examples in Chapter 2.

Problem Description

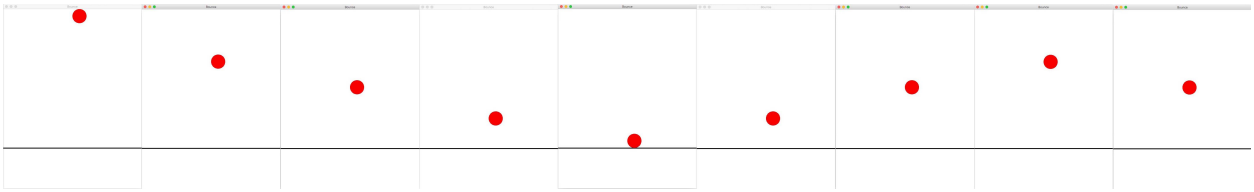

Figure 1: Snapshots from simulation of ball drop

Figure 1 shows a sequence of screen shots of the simulation of a 1 Kg ball dropped from a height of 10 meters, with the ball constrained to move in a vertical direction only. At each time step, the height of the ball is calculated and the position changed on the display. This can be done using simple Newtonian mechanics as follows:

If the ball starts off at height h_0 above the ground, then at each time instant, t, $h(t) = h_0 - 0.5\ g\ t^2$, where g is the gravitational constant, $9.8\ m/s^2$. Just before the ball collides with the ground, it attains a terminal velocity, v_t, which can be determined from conservation of energy, i.e., $0.5\ m\ v^2 = m\ g\ h_0$, so $v_t = (2\ g\ h_0)^{0.5}$. Assuming no loss of energy, height in the upward direction is given by $h(t) = v_t\ t - 0.5\ g\ t^2$. In practice, some fraction of the kinetic energy is lost, so v_t decreases over time. Assume that l represents the fraction of energy lost in the collision. Then $v_t = (2\ e_l\ g\ h_0)^{0.5}$, where $e_l = 1 - l$.

So, where does one begin? One can break the program down into the following steps:

1. Create a display window with an instance of a ball and a line representing the ground as shown in Figure 1.
2. Read simulation parameters from the user – the initial height, h_0, and the energy loss, e_l, represented as a number in the range [0,1].
3. Initialize the simulation: set $h(t) = h_0$, $time = 0$; $direction = 0$ (down); $v_t = (2\ g\ h_0)^{0.5}$
4. Simulation Loop:

```
If direction is down then {
    h(t) = h0 – 0.5 g t^2
    If h(t) <=ball_diameter {
        last_top = ball_diameter
        direction = 1
```

```
                time=0
                vt = vt* e_l
        }
}
else {
        h(t) = ball_diameter + v_t t - 0.5 g t^2
        if h(t) < last_top {
                if h(t) < ball_diameter   END SIMULATION
                direction = 0
                time = 0
        }
        last_top = height
}
```

Let's look at these steps in detail:

1. Setting up the program:

Chapter 2 provides several examples (i.e. code) for displaying simple graphical objects in a display window. Essentially you create a class that extends the acm class, GraphicsProgram, and provide a run() method (i.e. your code). There are a couple of items not described in Chapter 2 that will be useful here.

Parameters: It is useful to define parameters as shown below. It not only makes code easier to read, but it also makes changing program behavior less prone to error.

```
private static final int WIDTH = 600;
private static final double G = 9.8;
```

Display: When you create an instance of a graphics program, the display "canvas" is automatically created using default parameters. To create a window of a specific size, use the `resize` method as shown below:

```
public void run() {
     this.resize(WIDTH,HEIGHT);
```

where `WIDTH` and `HEIGHT` are the corresponding dimensions of the display canvas in pixels. Since we are doing a simulation of a physical system, it would be convenient to set up a coordinate system on the screen that mimics the layout of the simulation environment. This is shown schematically in Figure 2 on the following page.

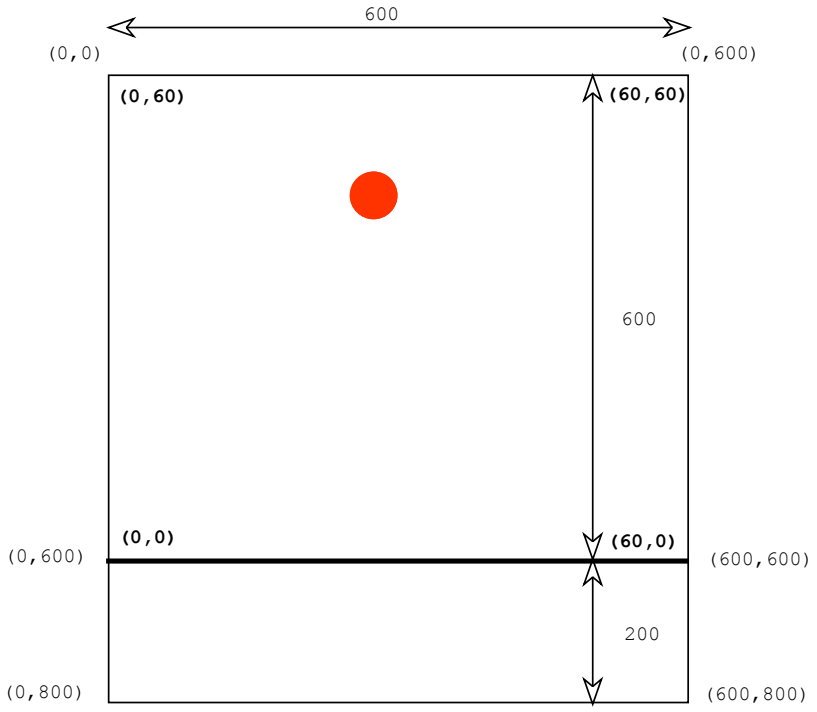

Figure 2

In the example shown in Figure 2, WIDTH=600 and HEIGHT=600+200. Let (x,y) represent a point in pixel coordinates and (X,Y) the corresponding point in simulation coordinates. The ground plane is represented by a filled rectangle beginning at (0,600) with width=600 and height=3 pixels. The upper left corner of this rectangle (0,600) corresponds to (0,0) in simulation coordinates. It is easy to infer the following relations between pixel and simulation coordinates: $x = X * 6$, $y = 600 - Y * 6$. The number 6 is the *scale factor* between pixel and simulation units. Since the y axis is in a direction opposite to the Y axis, we need to invert the scale by subtracting the scaled Y value from HEIGHT less the offset of the ground plane, 200.

2. Getting input from the user:

If you've programmed in Java before, you know how to obtain input from the user using the `Scanner` class. If this is your first time, then it is suggested that you include the acm classes in your program. How to do so will be explained in the tutorial sessions (you could also consult the course text). In this program you will need to input the initial drop height and energy loss parameters, both of which are represented as type `double` (computer approximation of real numbers using floating point representation). Chapter 2 shows how to read integer values using the `readInt` method; `readDouble` is the analogous method for real numbers, and has a similar form: `double value = readDouble("Enter value: ");`

3. Initialization:

Good software design practice usually entails thinking about the representation for a problem before any code is written. Each of the variables within the Simulation Loop needs to be explicitly represented as a Java datatype, in this case `double` or `int`. Consider, for example,

terminal velocity *vt*. In a Java program (class to be more specific), *vt* can be declared and initialized in a single line of code:

```
double vt = Math.sqrt(2*g*h0);
```

Before the simulation begins, each variable should be initialized to a known state. Pay attention to order. The above expression cannot be evaluated until *h0* is read from the user input.

4. Program structure – the simulation loop.

The template for the simulation program has the following form:

```
Public class Bounce extends GraphicsProgram {

     private static final int WIDTH = 600;
     .
     .

     public void run() {
          this.resize(WIDTH,HEIGHT);

     // Code to set up the Display shown in Figure 2.
     // (follow the examples in Chapter 2)
     .
     .
     .

     // Code to read simulation parameters from user.

     double h0 = readDouble ("Enter height the height of
     the ball in meters [0,60]: ");
     .
     .

     // Initialize variables

     double vt = Math.sqrt(2*g*h0);
     .
     .

     // Simulation loop

     while (true) {
          if (dir == 0) {
               height = last_top - 0.5*g*time*time;
               if (height <= ball_dia) {
                         dir=1;
```

```
        .
        .

        }
    }
```

You can find most of what you need to code this assignment in the examples in Chapter 2. The only detail that has not been covered at this point is the `while` loop. As the name implies, code within the loop is executed repeatedly until the looping condition is no longer satisfied or a `break` statement is executed within the loop. The general form of a while loop is

```
while (logical condition) {
     <code>
}
```

In the example above, the logical condition is set to true, which means the loop executes until the program is terminated or a `break` executed as shown below:

```
while (true) {
     <code>
     if (height < BALL_DIAMETER) break;
}
```

In the above example, the simulation will run until the height of the return bounce falls below a threshold value.

Instructions

1. Write a Java class, `Bounce.java`, that implements the simulation outlined above. Do this within the Eclipse environment so that it can be readily tested by the course graders. For your own benefit, you should get in the habit of naming your Eclipse projects so that they can easily be identified, e.g., ECSE-202_A1.
2. Test your program for various drop heights and collision losses. Make sure that the simulation produces correct results relative to the simple model used.
3. Bonus. Once you have your program working in the vertical direction, it is not difficult to incorporate motion along the horizontal axis (i.e. $x = v_x T$). Note: be careful here – for calculating height, t is reset to zero at each change of direction (time relative to last direction change). The variable T used here refers to time since the beginning of simulation. It is also useful to provide a *trace* of the ball trajectory as shown below in Figure 3.

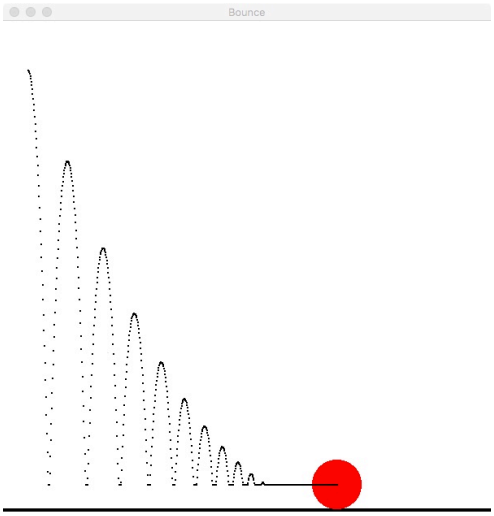

Figure 3

To get the full 10 marks for the Bonus, your simulation should replicate the output of Figure 3. Hand in a screen capture corresponding to Figure 3 for your simulation.

To Hand In:

1. The source file, Bounce.java.
2. A screen capture file for the Bonus.

All assignments are to be submitted using myCourses (see myCourses page for ECSE 202 for details).

About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS.

https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf/July 6, 2018