Assignment 0312

This assignment seeks to reinforce what you have learned so far without requiring [a lot of] new language knowledge. Instead, we aim the same knowledge at a new kind of program: discrete event simulation.

Outcomes

This assignment will affect your proficiency measures for outcomes 1a–1c, 2a–2c, and 3a–3f.

Not for Submission

The “Classes and Objects” lesson from the Java tutorials website continues to be front and center for this assignment:

http://docs.oracle.com/javase/tutorial/java/javaOO

If you haven’t read through this material yet, it is now high time that you do.

For Submission

Your work will simulate two circular projectiles, also known as balls, launched on a 2D field. The balls are affected by gravity, and the simulation ends when they both reach the ground. Your simulation reports the status of each ball at each “tick” of a simulated clock. When the simulation completes (i.e., both balls have reached the ground), the simulation should then report whether the projectiles hit each other in flight, and when.

To assist your simulation, a display program called AngryBalls2D has been written for you. Pay close attention in class for instructions on how to use this program alongside your simulation.

Specifications

Your simulation shall model two balls, each of which is initialized with a radius, initial location, and initial velocity. The simulation itself shall also be given a time slice or grain, indicating the amount of real time that should pass at each “frame” of the simulation.

The display program defaults to a field that is 1280 meters across and 720 meters high, with (0, 0) on the lower-left corner of the display. Thus, although technically the balls can start anywhere and travel at any velocity, in order to see the action you should set up your simulation runs to fit this area.

Note that if you start with a ball on either end of the field (i.e., \( x = 0 \) and \( x = 1280 \), respectively), the ball on the right side should have an initial velocity whose \( x \)-component is negative, because it should be traveling toward the left.

The balls shall be affected by gravity, at a rate of 9.8 \( \text{m/s}^2 \). Because gravity goes downward, the simulation will actually use a \( y \)-component of \(-9.8\) to update the balls’ velocities. Standard Newtonian mechanics shall be used to compute each ball’s new location and velocity per time slice; the calculations will be explained in class if you are not familiar with them. When a ball hits the ground (i.e., \( y \leq 0 \)—for simplicity we define the “ground” relative to the center of the ball), its velocity should be set to zero (i.e., it should stop moving).

Call the simulation AngryBallsSimulation; to run it, have it accept ten required arguments with an eleventh optional one:

• The radius, initial location of its center, and initial velocity of the first ball (which the display program will render in red)
• The radius, initial location of its center, and initial velocity of the second ball (which the display program will render in blue)
• Optionally, the time slice or grain to use, in seconds (or fractions thereof if less than one)—if not supplied, use a default of one second

Thus, an invocation of AngryBallsSimulation would look like this:

```
java AngryBallsSimulation 10 0 1 20 50 20 1280 1 -30 40
```

This creates a simulation where the red ball has a radius of 10, starting at \((0, 1)\) for its center with an initial velocity of \((20, 50)\), and the blue ball as a radius of 20 starting at \((1280, 1)\) for its center with an initial velocity of \((-30, 40)\)—the \(x\)-component is negative because that ball is traveling to the left. The time slice defaults to one second; if an 11th argument is provided, then that time slice is used.
Usage: java AngryBallsSimulation <red radius> <red x> <red y> <red velocity x> <red velocity y> <blue radius> <blue x> <blue y> <blue velocity x> <blue velocity y> [ <grain> ]

All sizes are in meters and the grain is in seconds. The grain is optional and defaults to 1 second if not supplied.

The arguments supplied do not match what AngryBallsSimulation expects.

Usage: java AngryBallsSimulation <red radius> <red x> <red y> <red velocity x> <red velocity y> <blue radius> <blue x> <blue y> <blue velocity x> <blue velocity y> [ <grain> ]

All sizes are in meters and the grain is in seconds. The grain is optional and defaults to 1 second if not supplied.

Yes, upon an error, you are supposed to print the error messages then display the same usage message. Make sure that your implementation does not do so redundantly.

For output, the simulation shall display one line for every simulated time slice, with each line displaying the radius, x-coordinate, and y-coordinate of each ball, separated by a single space. Once the simulation ends (i.e., both balls hit the ground), print the single word end on its own line. After that, display this message with italicized values to be supplied by the simulation:

The balls collided at timestamp \( t \) with the red ball at \((x, y)\) and the blue ball at \((x, y)\).

If the balls did not collide, display this:

The balls did not collide.

The class briefing will provide some sample output. Note that the output is particularly important here, because AngryBalls2D relies on this format to display the simulation results.

Design

New Java knowledge is not expected for this assignment, but to kick things up a notch, minimal starter code is provided. Beyond the already-discussed AngryBallsSimulation class, you will need at least the following two more:

- The Ball class should retain the state of a single ball. It should maintain a radius, a location, and a velocity. It also updates its location based on its current velocity, and can change its velocity according to a given acceleration. Both of these updates are based on a time slice or grain.
- The Vector class encapsulates computations on locations, velocities, and accelerations. It maintains an \((x, y)\) ordered pair, and can add itself to another Vector or scale itself according to a magnitude. Both of these operations create a new Vector.

Stubs for these classes are provided on the course website. You are free to implement more classes and or methods as needed. However, you may not change the stubs that have been given—this will ensure that test harnesses have a consistent baseline on which to perform tests.

Deliverables

The expected work consists of complete versions of the three aforementioned classes as well as any other classes you might have deemed necessary. In addition, create an AngryBallsTestHarness class, patterned after the test harness classes that have been given previously. Yes, for this assignment you are now expected to provide unit tests for your own code.

How to Turn It In

Upload your code to your GitHub repository. Don’t forget to commit as you go.

Due to the number of source files involved in this assignment, you should place your code in a separate angryballs folder. Designating folders via the GitHub web interface will be demonstrated during the class briefing for this assignment.

Potential Enhancements

Depending on how quickly you are able to finish the required work, consider the following enhancements to the simulation:

- Make the balls bounce off the ground, gradually losing lift with each bounce
- Give the field walls and a roof
- Make the balls bounce (reflect) off each other if they collide