Modeling in Processing Term 4 Exam

Monday, Feb. 21, 2022

Bouncing Among Obstacles — 25 pts — each part is 4 pts

Keep all the code for this problem in a single file. You will be emailing me the code when you are done. Do not bother going on to (b) until you have (a) working. Do not bother going on to (c) until you have (b) working. Etc., etc. Do not turn in something that doesn't run and work. If you only got (b) to run and work, then send that code to me. If (b) doesn't run and work turn in what you did for (a). Whatever code you turn in should be indented and spaced according to standard Java conventions. When you are done, email me your code with subject "Exam Problem."

Part (a) Start by making a single object sitting at a random point on a white canvas.

Make the class of this object be `Obstacle`. The class will need two instance variables: a `location` that is a `PVector`, and a `length` that is a `float`. It will have one method called `display()`. An `Obstacle` object should display as a solid black square with sides that are `length` long. Declare one `Obstacle` object like this after the `Obstacle` class:

```
Obstacle o;
```

In the `setup()` function initialize this object with a random location and length. The `Obstacle` object is going to stay fixed, so it does not need an `update()` method. Display the `Obstacle` object in the `draw()` function.

Part (b) Continue by making a simple class called `Bouncer`.

The class needs three instance variables: a `location` that is a `PVector`, a `velocity` that is a `PVector`, and a `radius` that is a `float`. It does not need acceleration! It will have a method called `display()`. A `Bouncer` object should display as a circle whose diameter is twice `radius`. Display it with a black stroke and no fill. It will also need an `update()` method in which it moves by `DELTA_T` times its velocity. Declare one `Bouncer` object like this:

```
Bouncer b;
```

In the `setup()` function, initialize this object with a random position, a random velocity, and a random radius. Call the `update()` and `display()` methods on the `Bouncer` object in the `draw()` function.

So far nothing interesting should happen. The bouncer just moves off the canvas with whatever velocity it had and never comes back.

Part (c) Add a `checkEdges()` method to the `Bouncer` object. In this method, if the `Bouncer` object is within `radius` of a wall, have it reverse direction. In the `draw()` function, call the `checkEdges()` method right after calling the `update()` method.

So at least the bouncer now stays on the canvas.

Part (d) Add a `bounce()` method to the `Bouncer` object that takes one argument: an `Obstacle` object. Call the `bounce()` method like this just before `b.checkEdges()`:

```
b.bounce(o);
```

Add a `boolean` instance variable called `consumed` to the `Bouncer` object. Initialize it to `false` in the `Bouncer` constructor.

If the bouncer hits the obstacle, change `consumed` to `true`. HINT: The hit test is going to have a lot of conditional tests that start like:

```
if (location.x + radius > obstacleLocation.x - obstacleLength / 2)
```

In the `display()` method for `Bouncer`, only display the bouncer if `consumed` is `false`.

Part (e) Upgrade the code to have many `Obstacle` objects. The declaration that was `Obstacle o;` will be changed to be for an `Obstacle` array:

```
Obstacle[] obstacles = new Obstacle[5];
```

Upgrade the code as needed. For example, the code that looked like `b.bounce(o);` will become a loop:

```
for (int i = 0; i < obstacles.length; ++i) {
    b.bounce(obstacles[i]);
}
```

Part (f) Upgrade the code again to have many `Bouncer` objects. The declaration that was `Bouncer b;` will be changed to be for a `Bouncer` array:

```
Bouncer bouncers[] = new Bouncer[50];
```

Again, upgrade the code as needed. For example, it will need to loop over all the Bouncer objects in draw(), something like this:

```
for (int j = 0; j < bouncers.length; ++j) {
 for (int i = 0; i < obstacles.length; ++i) {
```

Part (g) Instead of making the bouncers be consumed when they hit an obstacle, make the component of their velocity reverse realistically (the reversal will depend on whatever side of the obstacle it has hit).

Hey, the last part is pretty hard. If you've had enough already, congratulate yourself that you got to (f) and email me that. If you complete everything including Part (g) perfectly, you actually end up with 3 points of extra credit :)