Willy Husted

2/28/13

Discrete Simulation: Playground

**A playground problem**

* Make a Java program, *PlaygroundSim*, that solves the following problem:

On a flat playground, there are a bunch of stationary flagpoles (of negligible diameter), and a bunch of playground balls (one foot in diameter); for reference, we'll assume the center of the playground is at the point ( x, y ) = ( 0, 0 ). At time ( hours, minutes, seconds) = ( 0, 0, 0.0 ) all the balls are kicked with different velocities. Via discrete simulation of the system, your program should output:

* + An initial report showing the locations of all objects, including the velocities of the balls.
  + After every time slice, a report showing the location of every ball.
  + **Either** a final report indicating the first ball-ball or ball-flagpole collision, giving the simulated time and the objects involved, **or** NO COLLISION IS POSSIBLE, giving the simulated time at which the program made that discovery.
* Data about each flagpole will be given to your program via three consecutive args, namely the letter **F**, followed by x- and y-coordinates, measured in feet.
* Data about each ball will be given to your program via five consecutive args, namely the letter **B**, followed by x- and y-coordinates for the ball's starting position, measured in feet, followed by the speeds in the x- and y-directions, measured in feet per second.
* The final arg, if present, specifies the time slice in seconds; if missing, your program should default it to sixty seconds.
* As always, your program should verify the validity of the args.
* A typical invocation of your program might look like this:

*java PlaygroundSim F 0 0 F -1.5 -2.5 B 300 300 -1 -2 B 5 10 3 6 F 14.3 -15 10.0*

which indicates three flagpoles at (0,0), (-1.5,-2.5), and (14.3,-15); one ball starting at (300,300) which moves at the rate of one foot west and two feet south per second; a second ball starting at (5,10) which moves at the rate of three feet east and six feet north per second; with the simulation to be carried out using a ten second time slice.

* Although you may not collaborate with classmates about the code itself, you are invited/encouraged to collaborate on the construction of test cases for the simulation.

**Major design issues:**

The main design issue that I had was writing methods that had to be used over and over again when dealing with an array of objects, rather than just one object as I had been previously accustomed to. Although I felt like I made considerable progress in conceptualizing this idea, in the end my program was only able to compare between two balls rather than an arbitrary number of balls. I struggled with “separating concerns”, a design principle that I understand to be very important, particularly in terms of readability. While I tried to separate my concerns, I often found myself trying to fit too much into a single method. For example, my isBallCollision method started off as a very complicated method from which I attempted to return a string of what balls collided at what point. However, I rethought it and decided isBallCollision should be a much simpler method that merely returns a boolean: did the balls collide or not.

The second major design issue I had was figuring out where to put my methods. It usually came down between putting them in the PlaygroundSim class or the Ball class. The move method was one in particular that I wasn’t sure if it was inherent to a ball itself (i.e. put it in the Ball class) or if that was a method specific to the discrete simulation. I ended up putting it in the PlaygroundSim class.

**Known Problems**:

I am aware of several problems with my program. The biggest problem is that it can only compare the distance and collision time of a maximum of two balls and one flagpole. If others are put into the command line, the first two are the only ones looked at. Furthermore, there are zero validity checks. If more than two numbers are inputted after an “F”, a flagpole is created with just the first two numbers and the rest of the numbers are ignored. The same occurs with more than four numbers after a “B”, indicating a new ball is to be made. If omitted, the slice is merely the last digit inputted; there is no default slice. In other words, the data has to be put in precisely, so the user is hung out to dry if he or she accidentally inputs bad data. Another known bug is that I have no way of checking if the time slice is too large and objects “pass through” each other.

Another flaw is the “No collision possible” final report. Instead of checking whether all the balls are getting farther and farther away from each other and the flagpoles, the program ends when my clock reaches an arbitrarily large time. I chose twelve hours.