Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 3
Keeping Track of Objects

Introduction

This assignment is about data structures and objects, specifically the use of B-Trees to organize data in an explicit order. It extends the work done previously in Assignment 2 to include a mechanism for keeping track of all objects generated in order to i) determine when the entire set of balls has stopped moving and ii) to access each ball in order of size. You will use this new capability to generate the program described below. This assignment makes use of material from the Data Structures lectures.

Problem Description

Extend the program in Assignment 2 as follows. When the last ball stops moving, arrange each ball from left to right in size order. Figure 1a shows a configuration of 15 balls just as the last ball stops moving, and Figure 1b shows the result of arranging the balls in size order. Your program should operate as follows:

1. When launched, the simulation starts and runs until the last ball stops moving. At this point it should stop and prompt the user with the message "CR to continue" . The display should appear as shown in Figure 1a.
2. When the return key is pressed, the program proceeds with the ball sort and produces the display shown in Figure 1b. *Bonus:* Make the program fully interactive by using a GLabel to display "Click mouse to continue", the waitForClick() method for user input, and the setLabel() method to indicate "All Sorted!".

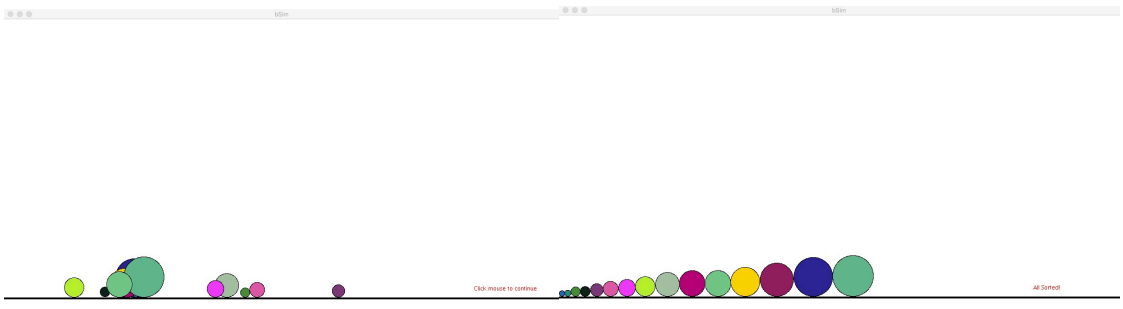

Figure 1a Figure 1b

For testing and submission of your program, please make sure to use the following parameters. Anything defined as a *double* corresponds to world (simulation) coordinates in units of meters, kilogams, seconds (MKS).

```
private static final int WIDTH = 1200;     // WIDTH, HEIGHT, and OFFSET
private static final int HEIGHT = 600;     // define the screen paramaters
private static final int OFFSET = 200;     // and represend PIXEL coordinates
private static final int NUMBALLS = 15;    // # balls to simulate
private static final double MINSIZE = 1;   // Min ball size (MKS from here down)
private static final double MAXSIZE = 8;   // Max ball size
private static final double XMIN = 10;     // Min X starting location
private static final double XMAX = 50;     // Max X starting location
private static final double YMIN = 50;     // Min Y starting location
private static final double YMAX = 100;    // Max Y starting location
private static final double EMIN = 0.4;    // Minimum loss coefficient
private static final double EMAX = 0.9;    // Maximum loss coefficient
private static final double VMIN = 0.5;    // Minimum X velocity
private static final double VMAX = 3.0;    // Maximum Y velocity
```

Design Approach

The first requirement is a data structure to hold the gBall objects generated and methods for adding new data to the B-Tree, and for tree traversal. Using the bTree class code posted on myCourses is a good starting point, but you will need to modify it to store gBall objects. Consequently in the run method of your bSim class, one of the things that you need to do is create an instance of the bTree class:

```
bTree myTree = new bTree();
```

You will have to modify the addNode method to accommodate the gBall object,

```
void addNode(gBall iBall);      // the argument is of type gBall
```

create a new method based on the in-order traversal routine that scans the B-Tree and checks the status of each gBall,

```
boolean isRunning();            // returns true if simulation still running
```

and finally a second new method based on the in-order traversal routine to move a ball to its sort order position instead of printing,

```
void moveSort(iBall);           // this will move all gBalls to their sorted position
```

Here is how this code might appear in your simulation class:

```
// Set up random number generator & B-Tree

    RandomGenerator rgen = RandomGenerator.getInstance();
    bTree myTree = new bTree();
```

```
// In the gBall generation loop

      gBall iBall = new gBall(Xi,Yi,iSize,iColor,iLoss,iVel);
      add(iBall.myBall);
      myTree.addNode(iBall);

// Following the gBall generation loop

      while (myTree.isRunning());              // Block until termination
      String CR = readLine("CR to continue");  // Prompt user
      myTree.moveSort();                       // Lay out balls in order
```

Modifications will also be needed to the gBall class. You will need to create an instance variable that indicates whether the while loop is active (that can be interrogated by the isRunning method), and a method to move a ball to a specified location, void moveTo(double x, double y), where (x,y) are in simulation coordinates.

Instructions

1. Modify the bTree class, `bTree.java`, to accommodate gBall objects and write the additional methods described earlier.
2. Modify the gBall class, `gBall.java`, to provide a method for returning run state and moving the ball.
3. Modify the bSim class, `bSim.java`, to complete the program design.
4. Before submitting your assignment, run your own tests to verify correct operation.

To Hand In:

1. The source java files. Note – use the default package.
2. A screenshot file (pdf) showing the state of the simulation at the user prompt as in Figure 1a.
3. A screenshot file showing the end of the program with the balls in sort order as in Figure 1b.

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

File Naming Conventions:

Fortunately myCourses segregates files according to student, so submit your .java files under the names that they are stored under in Eclipse, e.g., bSim.java, gBall.java, gUtil.java, bTree.java. We will build and test your code from here.

Your screenshot files should be named prompt.pdf and end.pdf respectively (again, this makes it possible for us to use scripts to manage your files automatically).

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 Oct 2018