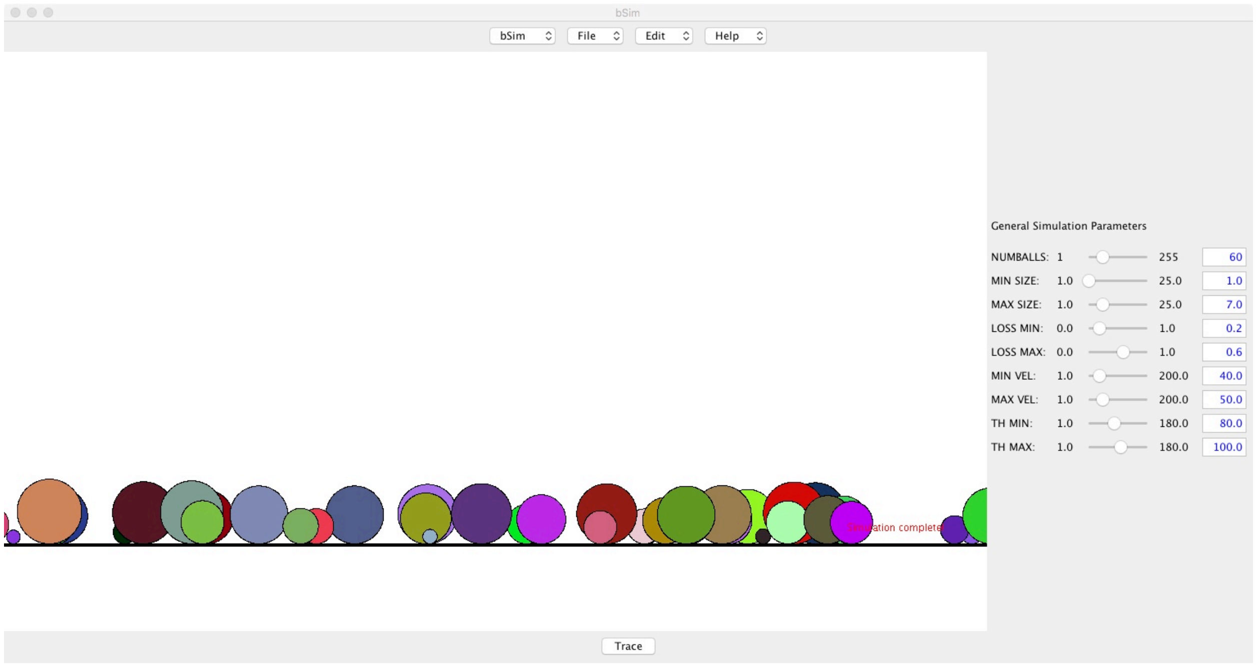
ECSE 202 – Introduction to Software Development October 27, 2019
Assignment 4
Due November 11, 2019 at 5:00 pm

Background

The simulation in Assignment 3 has no user interface, it simply runs according to a set of hard-wired parameters. In this assignment you will add a simple user interface as depicted below in Figure 1.



General Simulation Parameters

```
NUMBALLS:  1     255     60
MIN SIZE:  1.0   25.0    1.0
MAX SIZE:  1.0   25.0    7.0
LOSS MIN:  0.0   1.0     0.2
LOSS MAX:  0.0   1.0     0.6
MIN VEL:   1.0   200.0   40.0
MAX VEL:   1.0   200.0   50.0
TH MIN:    1.0   180.0   80.0
TH MAX:    1.0   180.0   100.0
```

Figure 1

As can be seen in the figure, the simulation incorporates a JPanel that includes a means of entering simulation parameters. Here sliders are being used in conjunction with IntFields and DoubleFields (similar to the Temperaure Conversion example in the notes). When the user moves a slider, the corresponding value is echoed in the corresponding Int/Double field. Similarly, when the user enters a number in the Int/Double field, the slider moves to a corresponding position. Across the top a set of 4 JComboBoxes are implemented to minic a familiar user interface, but only the first is actually used by the program. These allow the the user to select different features of the program (more below in Requirements). Finally, there is a single JToggleButton at the bottom. As its label implies, if pushed, the trace of each ball is plotted on the screen in the same color as the ball. Together these features allow considerable flexibility over the program in Assignment 3.

Requirements

1. Functionality

Your program must allow the user to perform the following functions. How you implement the User Interface is entirely up to you (e.g. JComboBox, push-buttons, etc.). In other words, you do not have to replicate the example in Figure 1.

Run:	Starts a new simulation with parameters chosen from the current parameter set. You should be able to run multiple simulations with new balls added to the display at each Run instance.
Stack:	Stacks **all** the balls created to date. For example, assume that you have run a simulation and stacked the balls. Now you run another simulation and stack again. This time all the balls in the previous stack are included in the current one. As long as you have not created a new bTree, all balls will be included when stacked.
Clear:	Clears the current display and kills the simulation attached to each ball. If the simulation is run again with new balls created, balls created previously are effectively eliminated.
Stop:	Stops the current simulation, freezing the current display.
Quit:	Exits the program.

In addition, your program must include a Trace mode that can be enabled or disabled. When enabled, the output resembles the output of Assignment 1 where a marker is placed at location traversed by the ball. In this implementation the color of the "dot" should match the color of the ball it represents. The display should be set up using with control functions laid across the top row, parameters along the right side of the display, and the Trace mode button at the bottom of the display. This is most easily accomplished using the BorderLayout manager. However you choose to implement selection, it must be event-driven so that the program reacts accordingly.

2. Parameter Entry

The key requirements are that i) all the parameters inputs should appear within a single JPanel that gets added to the display as shown, and ii) the user must be able to enter numerical parameters using the keyboard. The example in Figure 1 is quite complex to implement as the program must respond to multiple event types. The point here is to demonstrate what is possible. You can do this quite simply using Int/Double field boxes that generate ActionEvents (see the Temperature Conversion example in the notes). By the way, you can also do this within GraphicsProgram, but a workaround is needed for it to work correctly (see below).

Implementation

Most of the work in this assignment is related to modifying bSim to incorporate user interface elements as well as re-packaging the simulation code and stacking code as methods that can be called from the user interface. In particular, you should create a doSim() method that generates a set of balls, attaches them to the screen, and waits for the simulation to complete. Rather than immediately proceeding to ball stacking on completion, doSim() should return. Stacking functionality should be packaged as a method, doStack(), which traverses myTree and generates the appropriate display. Unfortunately, if you try to call doSim() on an event, it will run, but not update the display. This has to do with how GraphicsProgram is implemented and is beyond the scope of this course. If you're experienced writing Java code, then you can (if you haven't already in Assignment 3) use a different framework outside of acm. Otherwise, a workaround is shown below:

The usual way of handling a user selection would look something like the following:

1. Restructure bSim so that the simulation component (generate the set of bBalls) is set up as a method that can be called more than once, e.g., doSim();
2. Set up an entry for doSim() in a JComboBox or push button.
3. Dispatch to doSim() when the corresponding entry is selected as shown below:

```
public void itemStateChanged(ItemEvent e) {
                JComboBox source = (JComboBox)e.getSource();

                if (source==bSimC) {            \
                        if (bSimC.getSelectedIndex()==1) {
                                System.out.println("Starting simulation");
                                doSim();
                        }
                        else if (bSimC.getSelectedIndex()==2) {
                                System.out.println("Histogramming balls");
                                doHist();
                        }
                        etc…
                }
        }
```

Unfortunately, do to the way in which acm is implemented, your simulation will run, but not update the screen. An inelegant solution to this problem is as follows:

```
public void init() {
        simEnable=false;                                        // this is an instance variable

        …other code…

                while(true) {
                        pause(200);
                        if (simEnable) {                        // Run once, then stop
                                doSim();
                                bSimC.setSelectedIndex(0);
                                simEnable=false;
                        }
                }
}
```

The idea is to run the simulation from within the init() method rather than call it from itemStateChanged(). Variable simEnable allows the simulation to run exactly once if simEnable=true.

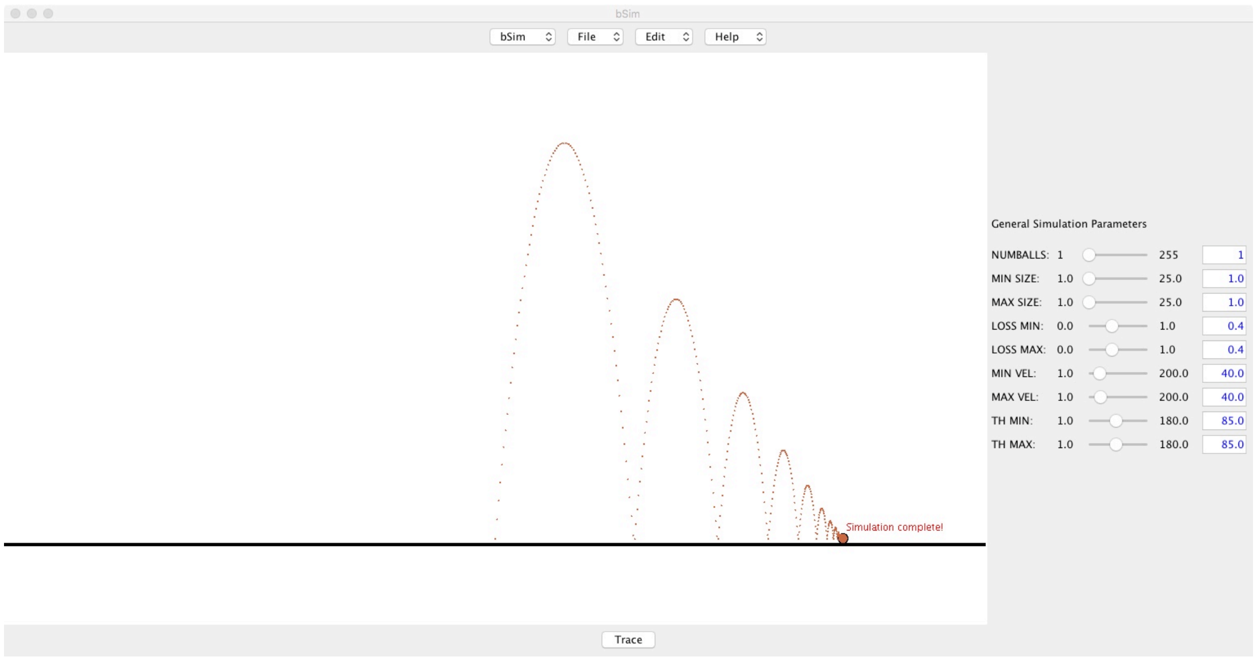
Back at itemStateChanged(), instead of calling doSim(), we simply change this to simEnable=true. This method is referred to as *polling* – you will encounter it again in ECSE 211, Design Principles and Methods.

Instructions

Modify the bSim class to implement the specified user interface. Take note that the display area shown in Figure 1 must match the display area in Assignment 3. In other words, you need to add additional space to incorporate the parameter panel. In A3 WIDTH corresponded to the entire display. Now it corresponds to the width of the display less the width of the parameter panel. You might also need to modify aBall to incorporate the trace feature.

To Hand In

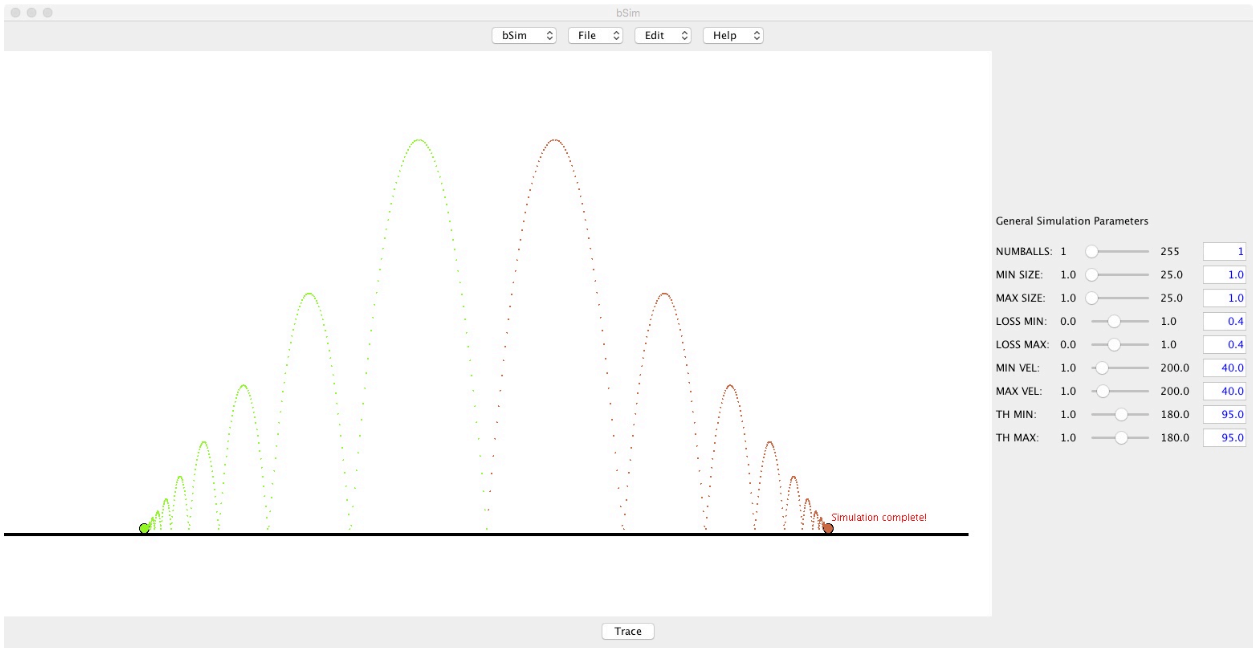
1. The source java files. Note – use the default package. Make sure that the random number generator is seeded with long integer value = 424242; this will ensure that your output matches ours if your simulations are correct.
2. Run a simulation with a single ball, starting at the center of the display field, with ball size = 1.0, loss = 0.40, velocity = 40.0, angle = 85.0 (you do this by making the min and max parameters equal to the specified value). Enable trace. This should replicate the output of Assignment 3, Part 1 as shown below. Take a screenshot of the display and hand it in as a pdf file named A4-1.pdf.



General Simulation Parameters

NUMBALLS:	1	255	1
MIN SIZE:	1.0	25.0	1.0
MAX SIZE:	1.0	25.0	1.0
LOSS MIN:	0.0	1.0	0.4
LOSS MAX:	0.0	1.0	0.4
MIN VEL:	1.0	200.0	40.0
MAX VEL:	1.0	200.0	40.0
TH MIN:	1.0	180.0	85.0
TH MAX:	1.0	180.0	85.0

3. Without altering the display from Part (2), change the angle parameter from 85° to 95° and run the simulation again. Take a screenshot of the display and save as a pdf file named A4-2.pdf. It should produce the output shown below.

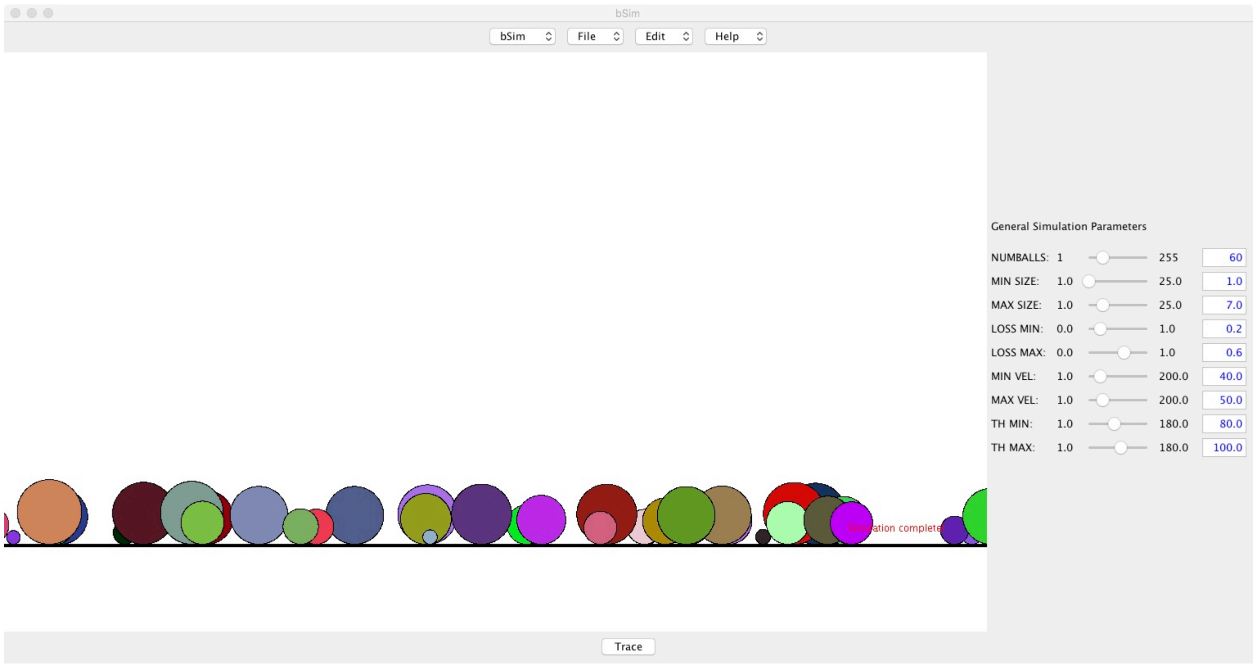


Simulation complete!

General Simulation Parameters

NUMBALLS:	1	255	1
MIN SIZE:	1.0	25.0	1.0
MAX SIZE:	1.0	25.0	1.0
LOSS MIN:	0.0	1.0	0.4
LOSS MAX:	0.0	1.0	0.4
MIN VEL:	1.0	200.0	40.0
MAX VEL:	1.0	200.0	40.0
TH MIN:	1.0	180.0	95.0
TH MAX:	1.0	180.0	95.0

4. Without leaving the program, enter a clear command to remove the current bTree and erase the display (same conditions as when the program is started from scratch. Run a simulation with the exact parameters as the last run from Assignment 3. On completion, take a screenshot of the display and save as a pdf file named A4-3.pdf. It should produce the output shown below.

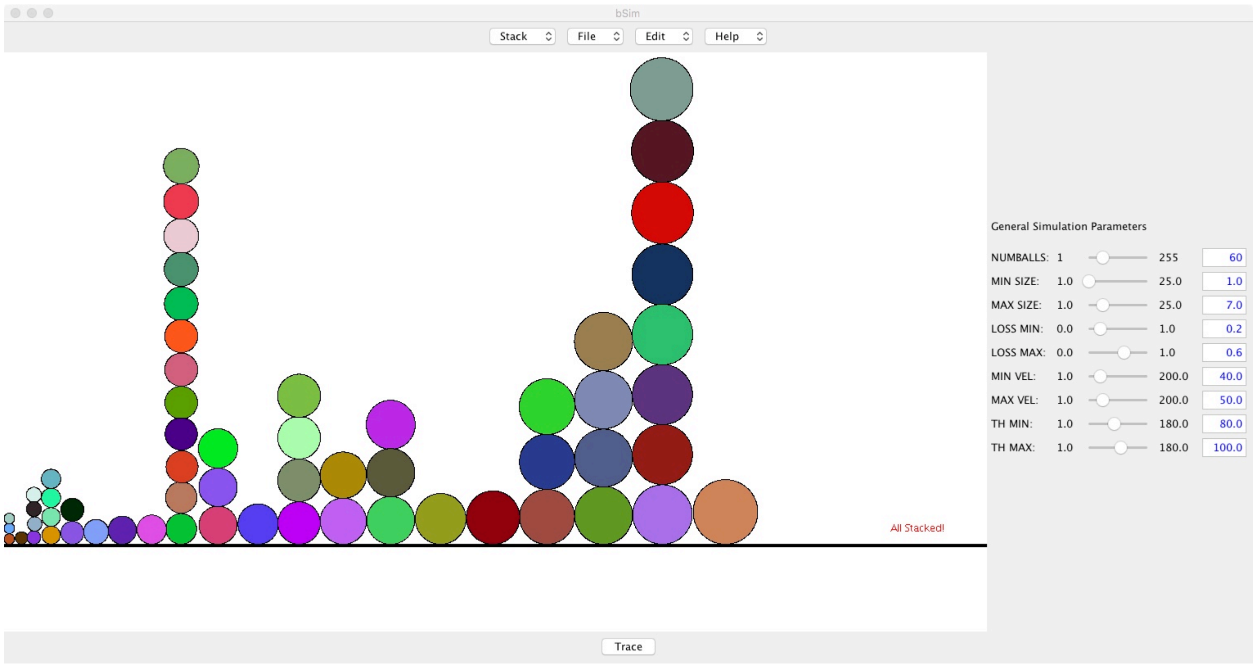


ation complete

General Simulation Parameters

NUMBALLS:	1	255	60
MIN SIZE:	1.0	25.0	1.0
MAX SIZE:	1.0	25.0	7.0
LOSS MIN:	0.0	1.0	0.2
LOSS MAX:	0.0	1.0	0.6
MIN VEL:	1.0	200.0	40.0
MAX VEL:	1.0	200.0	50.0
TH MIN:	1.0	180.0	80.0
TH MAX:	1.0	180.0	100.0

5. Finally, with the display as shown above, issue a stack command. Take a screenshot of the display and save as a pdf file named A4-4.pdf. It should produce the output shown below.



General Simulation Parameters

NUMBALLS:	1	255	60
MIN SIZE:	1.0	25.0	1.0
MAX SIZE:	1.0	25.0	7.0
LOSS MIN:	0.0	1.0	0.2
LOSS MAX:	0.0	1.0	0.6
MIN VEL:	1.0	200.0	40.0
MAX VEL:	1.0	200.0	50.0
TH MIN:	1.0	180.0	80.0
TH MAX:	1.0	180.0	100.0

Notes:

Since this is the last Java assignment, we will pay particular attention to the documentation of your code. We know that assignments from previous years are floating around on the Web, and naturally we will run comparisons of your code against them. The point of these assignments is for you to develop the requisite skills that will carry you through the rest of the program.

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, especially when the assignment is a variation from the previous semester.

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 October 27, 2019