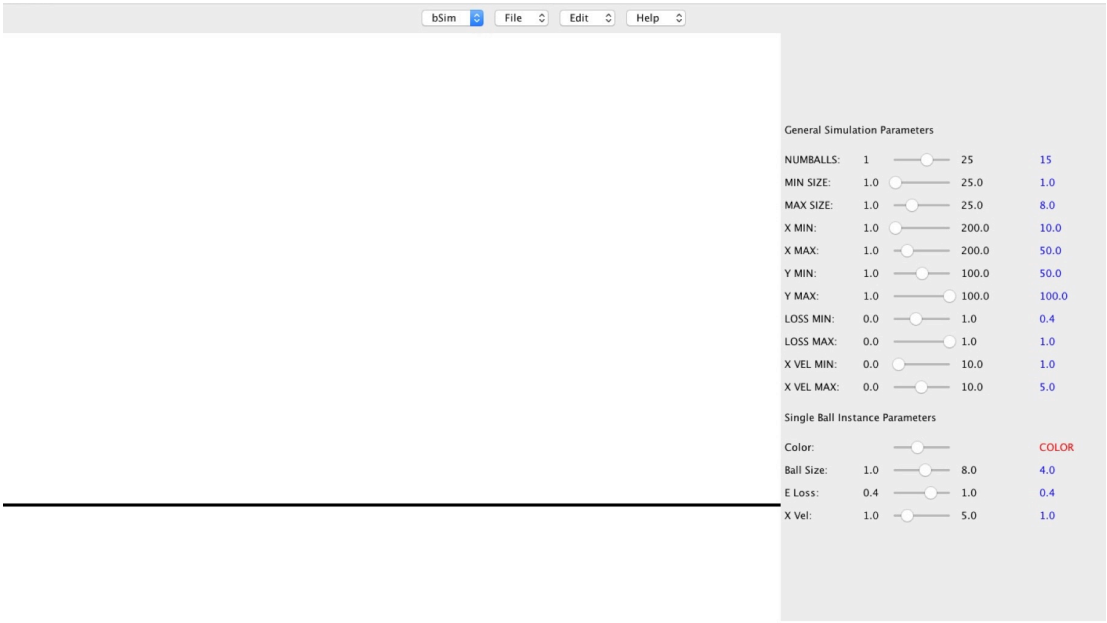
ECSE 202 – Introduction to Software Development
Assignment 4
Update

Background

After talking to many of you and answering questions about Assignment 4, we've decided to issue this update to clarify a number of points that you have raised, as well as to re-scope the assignment to be feasible for students without prior coding experience.

Requirements (100 points)



Your application should present something similar to the screenshot shown above. There should be a set of sliders that control the parameters of the simulation that were fixed in Assigment 3. In the screenshot, these are listed under General Simulation Parameters. There should be at least one JcomboBox for selecting the operations that can be performed and should include Run, to start a new simulation, and Quit, to exit the program. Selecting Run should start a new simulation that erases the previous output and begins a new simulation using the parameters selected using using the sliders. If you wish, you can include an explicit Clear option to force the clear at the discretion of the user (e.g. to abort the currently running simulation). In effect, A4 adds a graphical user interface to A3, permitting the user to interact with the program.

Bonus (25 points)

The bonus part of the question takes user interaction one step further. At any time the user should be able to select any ball, moving or not, and drag it to a new position on the screen. When released, the ball's parameters change to values provided by a second set of sliders, the Single Ball Instance Parameters, shown in the figure above. When released, the ball simulation runs forward from that point in time until it runs out of energy and halts. If the selected ball is released before the main simulation has completed, it should appear in the sort in proper order, otherwise, it simply runs independently until it stops. This mode of operation continues until the simulation is re-started.

Hints

1. Consider writing a sliderBox class that assembles a JLabel identifying the box next to a JLabel showing the minimum value, followed by a JSlider, followed by another JLabel to identify the max value, and finally a final JLabel to display the selected value. As the slider moves, the readout should change accordingly as shown below.

To perform this assembly, you can create a JPanel, which operates exactly the same way as the display canvas. You can also tell JPanel which layout manager to use so that you can control alignment more precisely. Here is partial code for a sliderBox class.

```
public sliderBox(String name, Integer min, Integer dValue,
Integer max) {        //  Integer values
      myPanel = new JPanel();
      nameLabel = new JLabel(name);
      minLabel = new JLabel(min.toString());
      maxLabel = new JLabel(max.toString());
      mySlider = new JSlider(min,max,dValue);
      sReadout = new JLabel(dValue.toString());
      sReadout.setForeground(Color.blue);
      myPanel.setLayout(new TableLayout(1,5));
```

```
        myPanel.add(nameLabel,"width=100");
        myPanel.add(minLabel,"width=25");
        myPanel.add(mySlider,"width=100");
        myPanel.add(maxLabel,"width=100");
        myPanel.add(sReadout,"width=80");
        imin=min;
        imax=max;
}
```

This code corresponds to the constructor for this class (which does most of the heavy lifting). Since you'll need to handle both integer and double values, you will need a second constructor to handle doubles. Finally you will have to write get and set methods for both cases inorder to read back values and update the display.

Note that you can use the same approach to put multiple slider boxes inside a JPanel, all precisely aligned using the TableLayout manager, and then add that panel to the default canvas.

2. Once you have your set of slider boxes, you will need to set up listeners,

```
MaxSize.mySlider.addChangeListener((ChangeListener) this);
XMin.mySlider.addChangeListener((ChangeListener) this);
XMax.mySlider.addChangeListener((ChangeListener) this);
etc…
```

   and then implement the stateChanged method to catch events generated by the sliders,

```
public void stateChanged(ChangeEvent e) {
JSlider source = (JSlider)e.getSource();

      if (source==NumBalls.mySlider) {
            PS_NumBalls=NumBalls.getISlider();
            NumBalls.setISlider(PS_NumBalls);
      }
      else if (source==MinSize.mySlider) {
            PS_MinSize=MinSize.getFSlider();
            MinSize.setFSlider(PS_MinSize);
      }
      etc…
}
```

3. The other interface element that you will need is a JComboBox to implement the chooser menus (by the way, it's OK to use a JComboBox to select the colors if you wish – I choose not to for aesthetic reasons).

```
void setChoosers() {
      bSimC = new JComboBox<String>();
      bSimC.addItem("bSim");
      bSimC.addItem("Run");
      bSimC.addItem("Clear");
      bSimC.addItem("Stop");
      bSimC.addItem("Quit");
      add(bSimC,NORTH);
}
```

You don't have to follow the example literally – this is just to show you how to use this element. Next, you need to set up an item change listener,

```
void addJComboListeners() {
      bSimC.addItemListener((ItemListener)this);
      etc…
}
```

and then implement the item change listener method,

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

      if (source==bSimC) {
            if (bSimC.getSelectedIndex()==1) {
                  System.out.println("Starting simulation");
                  this.SimRunning=true;
            }
      etc…
      }
```

4. Implementing the Bonus will require some additional features for the bTree class, namely methods for i) searching for the gBall instance that corresponds to a specific GOval returned on a muse click, ii) re-sorting the bTree in the event that the size parameters of one or more gBalls have changed. The first is a straightforward traversal comparing an input GOval reference with those at each node of the tree. The second is easily done by writing a copy function that traverses the bTree to be re-sorted, using the addNode() method to build aa new tree in the process. Once this traversal is complete, the reference to the new bTree is copied over to the old.
5. When a particular GOval instance is selected, it is often easier to kill its corresponding thread and create a new gBall instance (that is then attached to the GOval), rather than modifying the gBall class.

Final Points

The program is being judged on functionality, so as long as it performs the functions outlined above, all is good. However, there is one exception. Since this is supposed to be built on A1, A2, and A3, you may not change the underlying storage method , i.e., you may not use arrays to store gBalls. If you do, there will be a penalty. Have a look at the posted solutions for A3.

Fpf/November 1, 2018