

1 Bounds for Balls into Bins

Consider the following scenario: You are given n bins in which you randomly throw balls. The probability that you hit a specific bin is $\frac{1}{n}$. But what if you throw n balls? Let $C_i(j)$ denote the indicator function that bin i was chosen for ball number j . The expectation for n balls is

$$\sum_{j=0}^n C_i(j) = n \frac{1}{n} = 1$$

So we can expect that every bin contains one ball on average. What is the expected maximum number per bin? With probability $1 - \frac{1}{n}$, no bin receives more than $\Theta\left(\frac{\log n}{\log \log n}\right)$ balls.

But on to the simulation: We have n bins, which will get indices, so let us use an array

⟨bin contents⟩ \equiv

```
val binContents = new Array[Int](n)
```

Throwing a ball into a bin amounts to choosing a bin at random and incrementing its count:

⟨throwing a ball⟩ \equiv

```
val rnd = new java.util.Random
def throwBall: Int = {
  val i = rnd.nextInt(binContents.size)
  binContents(i) = binContents(i) + 1
  i
}
```

1.1 Visualizing the ball throwing

In the end, we will want to simulate everything inside a JPanel, so let us subclass it:

⟨bin simulation⟩ \equiv

```
class BinSimulation(n: Int) extends JPanel {  
    <bin contents>  
    <throwing a ball>  
    <representing the balls>  
}
```

1.1.1 Representing a ball

A ball will be drawn as a filled circle. Given a graphics context, it will be easy to draw. We'll also have to remember the offset of this ball (if we have thrown 2 balls into the same bin, the second one has to start a little later). The time delay is here as we do not want all bins to fall at the same time.

<ball simulation> ≡

```
import java.awt._  
class Ball(var x: int, var y: Int, offset: Int, timeDelay: Int) {  
    val r = 10  
    def paint(g: java.awt.Graphics): Unit = {  
        g.setColor(Color.red)  
        g.fillOval(x - r, y - r, r * 2, r * 2)  
    }  
    <move the ball>  
}
```

If a ball should not yet appear, it has a delay before it will begin falling:

<move the ball> ≡

```
val dy = 20
val totalHeight = 200
var off = timeDelay * 6
def move: Unit = {
  if( off > 0 )
    off = off - 1
  else
    if( y < totalHeight - offset * dy)
      y += dy
    else
      y = totalHeight - offset * dy
}
```

Now for the actual representation of balls: We'll first throw them into bins (precalculation), and then according to that we'll create them. Note the rather contorted way of specifying delays: We'll store them in a map so that we know them during generation.

<representing the balls> ≡

```

def getBalls: List[Ball] = {
  1 to binContents.size foreach {
    i => binContents(i - 1) = 0
  }
  val timeDelays =
    new scala.collection.mutable.HashMap[Int,List[Int]]
  var timeDelay = 0
  1 to n foreach {
    x => {
      timeDelay = timeDelay + 1
      val bin = throwBall
      timeDelays(bin) = timeDelays.getOrElse(bin,Nil) ::: List(timeDelay)
    }
  }
  List.range(0,n) flatMap {
    x => List.range(0,binContents(x)) map {
      y => val indx = timeDelays(x).head
          timeDelays(x) = timeDelays(x).tail
          new Ball(x * 20 + 50, -20, y, indx)
    }
  }
}
var balls = getBalls

```

Note that we made this a variable, so that we can reset the experiment:

<representing the balls> + ≡

```

def reset: Unit = {
  balls = getBalls
}

```

For the animation, we'll have to move all balls.

<representing the balls> + ≡

```

def evolve: Unit =
  balls foreach { ball => ball.move }

```

Also, it would be nice to know how many balls got at maximum into a bin:

⟨representing the balls⟩ + ≡

```
def maximum: Int =
  (binContents foldLeft 0) {
    (max: Int, el: Int) ⇒ if( el > max ) el
                        else max
  }
override def paint(g: java.awt.Graphics) {
  balls foreach { ball ⇒ ball.paint(g) }
}
```

1.2 The applet

Applets in Scala use the same API as Java applets do, they inherit from `JApplet`. Also the same methods have to be implemented, namely `init`.

For the animation, we create a new runnable that just evolves the simulation.

⟨⟩* ≡

```

import javax.swing._
import java.awt.event._

<bin simulation>
<ball simulation>
class BinApplet extends JApplet {
  val simulationPanel = new BinSimulation(10)
  val simulationButton = new JButton("Start simulation")
  val countLabel = new JLabel()

  override def init() {
    val t = this
    simulationButton.addActionListener(
      new java.awt.event.ActionListener() {
        def actionPerformed(e: java.awt.event.ActionEvent) = {
          simulationPanel.reset
          countLabel.setText("Maximum balls per bin: " +
                               simulationPanel.maximum)

          val runner = new Runnable() {
            override def run(): Unit = {
              1 to 120 foreach {
                x => simulationPanel.evolve
                t.repaint()
                Thread.sleep(50)
              }
            }
          }
          new Thread(runner).start
        }
      }
    )
    getContentPane().add("Center",simulationPanel)
    getContentPane().add("North",simulationButton)
    getContentPane().add("South",countLabel)
  }
}

```