Jerry Cain
CS 106AJ

Handout #24
October 22, 2018

Section Handout #4

Portions of this handout by Eric Roberts, Patrick Young, Jeremy Keeshin, Mehran Sahami, Nick Troccoli, and Kat Gregory

Problem 1: Interactive, Animated Random Circles

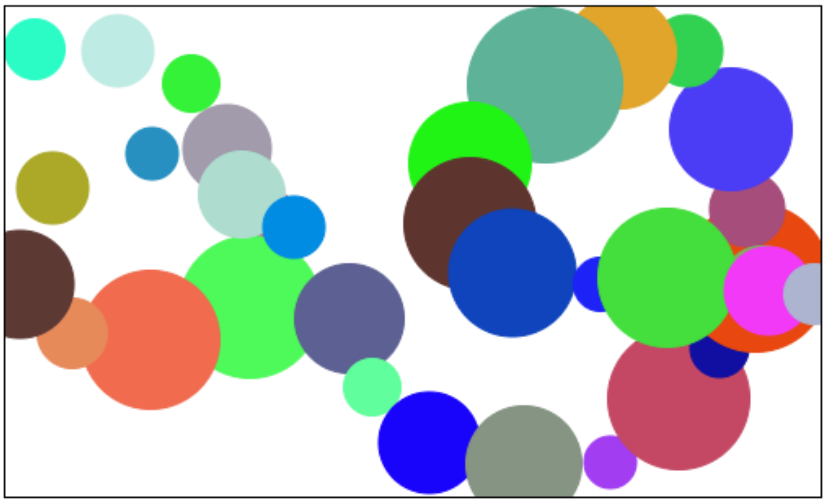
Remember the Random Circles exercise from last week's section? We wrote a **`GraphicsProgram`** that drew a set of ten circles with different sizes, positions, and colors. Let's make it a bit more interesting. In particular, we'll add interaction and animation.

Write an interactive **`GraphicsProgram`** that grows a random circle in every place the user clicks. Each circle should be centered on the user's mouse click and have a randomly chosen color. It should also have a randomly chosen radius between 5 and 50 pixels, but you should animate the circle growing to this size over time. Note that your program should be able to handle multiple circles growing at once if the user clicks in quick succession.

```
/* Constants */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;
const MIN_RADIUS = 15;
const MAX_RADIUS = 50;
const TIME_STEP = 20;
const DELTA_SIZE = 1;
```

```
/* Useful functions to look into */
GWindow.addEventListener("click", action);
setInterval(action, time_interval);
clearInterval(timer);
GOval.setBounds(x, y, width, height);
```

The following sample run shows one possible outcome if the user clicks along a loop-de-loop trajectory:



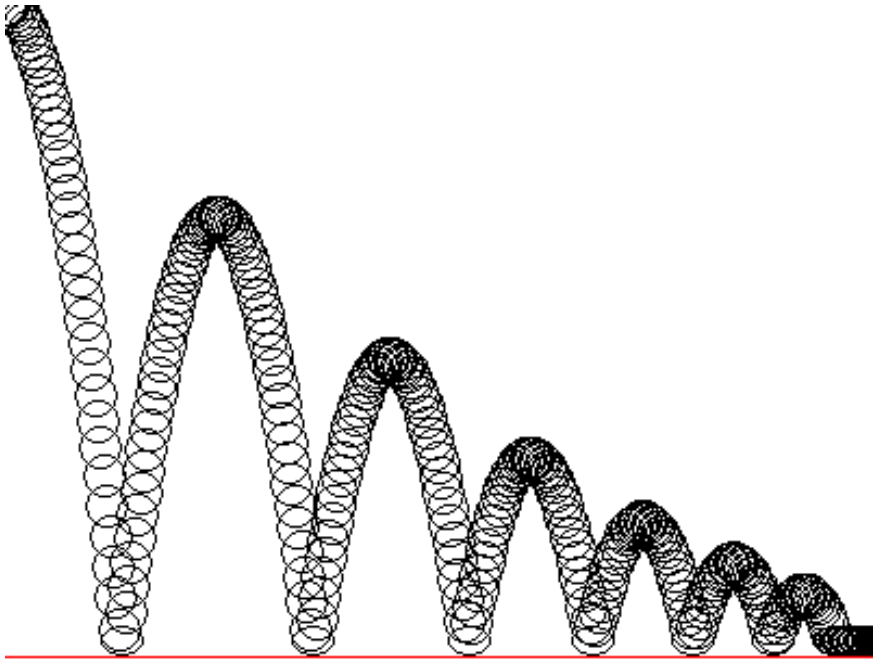
Problem 2: Simulating Physics with Bouncing Balls

Write an interactive **`GraphicsProgram`** that simulates the motion of balls bouncing under the influence of gravitational force.

Each time the user clicks, drop a randomly-colored ball from the top left corner of the window. The ball should begin with a random velocity in the X direction and no velocity in the Y direction. It will fall, accelerating towards the bottom of the screen, until it hits the bottom of the window. As it bounces, the ball will reverse direction and begin decelerating upwards, but it will also lose some energy from the collision. It should continue bouncing until it rolls off the right hand end of the window. Like the previous exercise, your program should be able to handle multiple balls bouncing at once if the user clicks in quick succession.

```
/* Constants */
const GWINDOW_WIDTH = 800;
const GWINDOW_HEIGHT = 300;
const DIAMETER = 20;
const MIN_X_VEL = 3;
const MAX_X_VEL = 15;
const TIME_STEP = 20;
const GRAVITY = 3; // amount Y velocity is increased each cycle
const BOUNCE_REDUCE = 0.75; // amount Y velocity is reduced during bounce
```

The diagram below illustrates the trajectory of the dropped ball:



Problem 3: Adding commas to numeric strings

When large numbers are written out on paper, it is traditional—at least in the United States—to use commas to separate the digits into groups of three. For example, the number one million is usually written in the following form:

```
1,000,000
```

To make it easier for programmers to display numbers in this fashion, implement a function

```
function addCommasToNumericString(digits)
```

that takes a string of decimal digits representing a number and returns the string formed by inserting commas at every third position, starting on the right. For example, the code below should produce the following outputs.

```
addCommasToNumericString("17")            returns 17
addCommasToNumericString("1001")          returns 1,001
addCommasToNumericString("12345678")      returns 12,345,678
addCommasToNumericString("999999999")     returns 999,999,999
```

Toolbox (more documentation can be found on page 200 of the course reader):

```
str.length
str.charAt(i)
str.substring(start)
str.substring(start, end)
str.indexOf(pattern)
str.toLowerCase()
str.toUpperCase()
str.startsWith(prefix)
str1 + str2
str1 === str2
```