Reversing the Polarity of a Number

When we want to reverse the polarity of a number, we mean that we want a positive number to become negative and a negative number to become positive. This is achieved by multiplying by –1. Remind yourself of the following:

- `-5 * -1 =  5`
- `-5 * -1 = -5`
- `-1 *  1 = -1`
- `-1 * -1 =  1`

Running the sketch, we now have a circle that turns around when it reaches the right-most edge, but runs off the left-most edge of the screen. We'll need to revise the conditional slightly.

If the ball goes off either the right or left edge, turn the ball around.

Or more formally. . .

If x is greater than width or if x is less than zero, reverse speed.

```
if ((x > width) || (x < 0)) {
  speed = speed * -1;
}
```

Remember, || means "or".

Example 5-6 puts it all together.

Example 5-6: Bouncing ball

```
int x = 0;
int speed = 1;

void setup() {
  size(200,200);
  smooth();
}

void draw() {
  background(255);

  x = x + speed;

  if ((x > width) || (x < 0)){
    speed = speed * -1;
  }

  // Display circle at x location
  stroke(0);
  fill(175);
  ellipse(x,100,32,32);
}
```

Add the current speed to the *x* location.

If the object reaches either edge, multiply speed by −1 to turn it around.

Exercise 5-9: Rewrite Example 5-6 so that the ball not only moves horizontally, but vertically as well. Can you implement additional features, such as changing the size or color of the ball based on certain conditions? Can you make the ball speed up or slow down in addition to changing direction?

The "bouncing ball" logic of incrementing and decrementing a variable can be applied in many ways beyond the motion of shapes onscreen. For example, just as a square moves from left to right, a color can go from less red to more red. Example 5-7 takes the same bouncing ball algorithm and applies it to changing color.

Example 5-7: "Bouncing" color

```
float c1 = 0;
float c2 = 255;

float c1dir = 0.1;
float c2dir = -0.1;

void setup() {
  size(200,200);
}

void draw() {
  noStroke();

  // Draw rectangle on left
  fill(c1,0,c2);
  rect(0,0,100,200);

  // Draw rectangle on right
  fill(c2,0,c1);
  rect(100,0,100,200);

  // Adjust color values
  c1 = c1 + c1dir;
  c2 = c2 + c2dir;

  // Reverse direction of color change
  if (c1 < 0 || c1 > 255) {
    c1dir *=  -1;
  }

  if (c2 < 0 || c2 > 255) {
    c2dir *=  -1;
  }
}
```

Two variables for color.

Start by incrementing c1.
Start by decrementing c2.

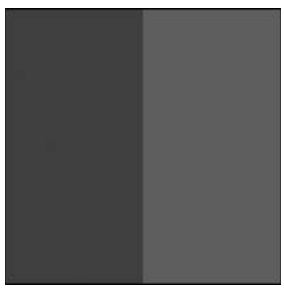

fig. 5.9

Instead of reaching the edge of a window, these variables reach the "edge" of color: 0 for no color and 255 for full color. When this happens, just like with the bouncing ball, the direction is reversed.

Having the conditional statement in our collection of programming tools allows for more complex motion. For example, consider a rectangle that follows the edges of a window.

One way to solve this problem is to think of the rectangle's motion as having four possible states, numbered 0 through 3. See Figure 5.10.

- State #0: left to right.
- State #1: top to bottom.
- State #2: right to left.
- State #3: bottom to top.

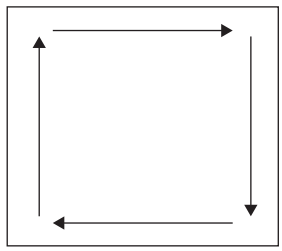

fig. 5.10

We can use a variable to keep track of the state number and adjust the *x*, *y* coordinate of the rectangle according to the state. For example: "If the state is 2, *x* equals *x* minus 1."

Once the rectangle reaches the endpoint for that state, we can change the state variable. "If the state is 2: (a) *x* equals *x* minus 1. (b) if *x* less than zero, the state equals 3."

The following example implements this logic.

Example 5-8: Square following edge, uses a "state" variable

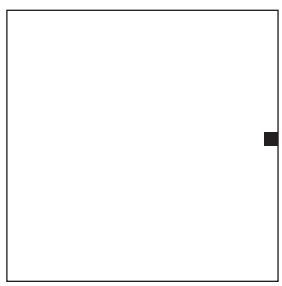

fig. 5.11

```
int x = 0; // x location of square
int y = 0; // y location of square

int speed = 5; // speed of square

int state = 0;

void setup() {
  size(200,200);
}

void draw(){
  background(100);

  // Display the square
  noStroke();
  fill(255);
  rect(x,y,10,10);
  if (state == 0) {
    x = x + speed;
    if (x > width-10) {
      x = width-10;
      state = 1;
    }
  } else if (state == 1) {
    y = y + speed;
    if (y > height-10) {
      y = height-10;
      state = 2;
    }
```

A variable to keep track of the square's "state." Depending on the value of its state, it will either move right, down, left, or up.

If the state is 0, move to the right.

If, while the state is 0, it reaches the right side of the window, change the state to 1.

Repeat this same logic for all states!

```
  } else if (state == 2) {
    x = x - speed;
    if (x < 0) {
      x = 0;
      state = 3;
    }
  } else if (state == 3) {
    y = y - speed;
    if (y < 0) {
      y = 0;
      state = 0;
    }
  }
}
```

5.8 Physics 101

For me, one of the happiest moments of my programming life was the moment I realized I could code gravity. And in fact, armed with variables and conditionals, you are now ready for this moment.

The bouncing ball sketch taught us that an object moves by altering its location according to speed.

location = *location* + *speed*

Gravity is a force of attraction between all masses. When you drop a pen, the force of gravity from the earth (which is overwhelmingly larger than the pen) causes the pen to accelerate toward the ground. What we must add to our bouncing ball is the concept of "acceleration" (which is caused by gravity, but could be caused by any number of forces). Acceleration increases (or decreases) speed. In other words, acceleration is the rate of change of speed. And speed is the rate of change of location. So we just need another line of code:

speed = *speed* + *acceleration*

And now we have a simple gravity simulation.

Example 5-9: Simple gravity

```
float x = 100; // x location of square
float y = 0;   // y location of square

float speed = 0; // speed of square
float gravity = 0.1;

void setup() {
  size(200,200);

}

void draw() {
  background(255);
```

A new variable, for gravity (i.e., acceleration). We use a relatively small number (0.1) because this acceleration accumulates over time, increasing the speed. Try changing this number to 2.0 and see what happens.

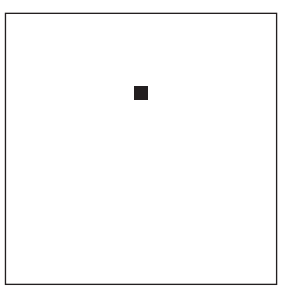

fig. 5.12

```
  // Display the square
  fill(0);
  noStroke();
  rectMode(CENTER);
  rect(x,y,10, 10);

  y = y + speed;

  speed = speed + gravity;

  // If square reaches the bottom
  // Reverse speed
  if (y > height) {
    speed = speed * -0.95;
  }
}
```

Add speed to location.

Add gravity to speed.

Multiplying by −0.95 instead of −1 slows the square down each time it bounces (by decreasing speed). This is known as a "dampening" effect and is a more realistic simulation of the real world (without it, a ball would bounce forever).

Exercise 5-10: Continue with your design and add some of the functionality demonstrated in this chapter. Some options:

- Make parts of your design rollovers that change color when the mouse is over certain areas.
- Move it around the screen. Can you make it bounce off all edges of the window?
- Fade colors in and out.

Here is a simple version with Zoog.

Example 5-10: Zoog and conditionals

```
float x = 100;
float y = 100;
float w = 60;
float h = 60;
float eyeSize = 16;

float xspeed = 3;
float yspeed = 1;

void setup() {
  size(200,200);
  smooth();
}

void draw() {
  // Change the location of Zoog by speed
  x = x + xspeed;
  y = y + yspeed;
```

Zoog has variables for speed in the horizontal and vertical direction.