Processing Notes

Chapter 12: Functions Returning Booleans and Collision Detection

So far we have not done much with booleans explicitly. Again, a boolean is a variable or expression that takes on exactly one of two values: true or false. For example, in the statement:

```
if ( currentX < width)
```

the expression in parenthesis evaluates to a boolean. We can write functions that return a boolean. Here are three simple functions that return a boolean:

```
boolean isOdd( int inValue)
{
  if ((inValue % 2) == 1)
    return (true) ;
  else
    return (false) ;
}

boolean firstIsGreaterThanSecond(int inFirst, int inSecond)
{
  if (inFirst > inSecond)
    return(true) ;
  else
    return(false) ;
}

boolean isLessThanFifty( int inNum)
{
  if (inNum < 50)
    return(true) ;
  else
    return(false) ;
}
```

And here are examples of calling the functions:

```
void setup()
{
  int num1 = 3 ;
  int num2 = 4 ;
  int num3 = 200 ;

  if ( isLessThanFifty(num1) )
    System.out.println(num1 + " is less than fifty") ;
  if ( isOdd(num2) )
    System.out.println(num2 + " is odd") ;
  if ( isOdd(num1) )
    System.out.println(num1 + " is odd") ;
  if ( firstIsGreaterThanSecond( 3, 4) )
    System.out.print("First argument is greater than second") ;
}
```

The output is two lines of text:

```
3 is less than fifity
3 is odd
```

These are rather contrived examples, but they illustrate the main parts of boolean functions: they return true of false and a call to them can be place in any expression that is supposed to evaluate to a boolean.

Lets consider some less contrived examples.

```
float rectX = 20 ;
float rectY = 20 ;
float rectWidth = 30 ;
float rectHeight = 50 ;

boolean MouseInRect(float inRectX, float inRectY, float inRectWidth, float
inRectHeight)
{
  if ( (mouseX > inRectX) && (mouseX < (inRectX + inRectWidth) )  &&
       (mouseY > inRectY) && (mouseY < (inRectY + inRectHeight) ) )
       return (true) ;
  else
       return(false) ;
}

void setup()
{
  size(400,400) ;
  fill(0) ;
}

void draw()
{
  background(100) ;
  rect(rectX,rectY,rectWidth,rectHeight) ;

}

void mousePressed()
{
  if ( MouseInRect( rectX, rectY, rectWidth, rectHeight ) )
    fill(255,0,0);
  else
    fill(0) ;
}
```

In the above example there is a function called MouseInRect( ). The function takes as parameters the x, y, width, and height values of a rectangle. It then returns true if the mouse is currently located inside the rectangle, otherwise it returns false. The code creates a rectangle and keeps drawing it so we have a stationary rectangle. The mousePressed( ) function calls MouseInRect( ) with the coordinates/width/height of the rectangle to see if the user just clicked on the rectangle. If so, the fill is changed to red and the rectangle remains red until the user clicks outside of the rectangle.

The MouseInRect( ) function tests to check a geometric condition: in this case if a point is inside a rectangle. The point is the mouse coordinates, the rectangle is passed in via the function call.

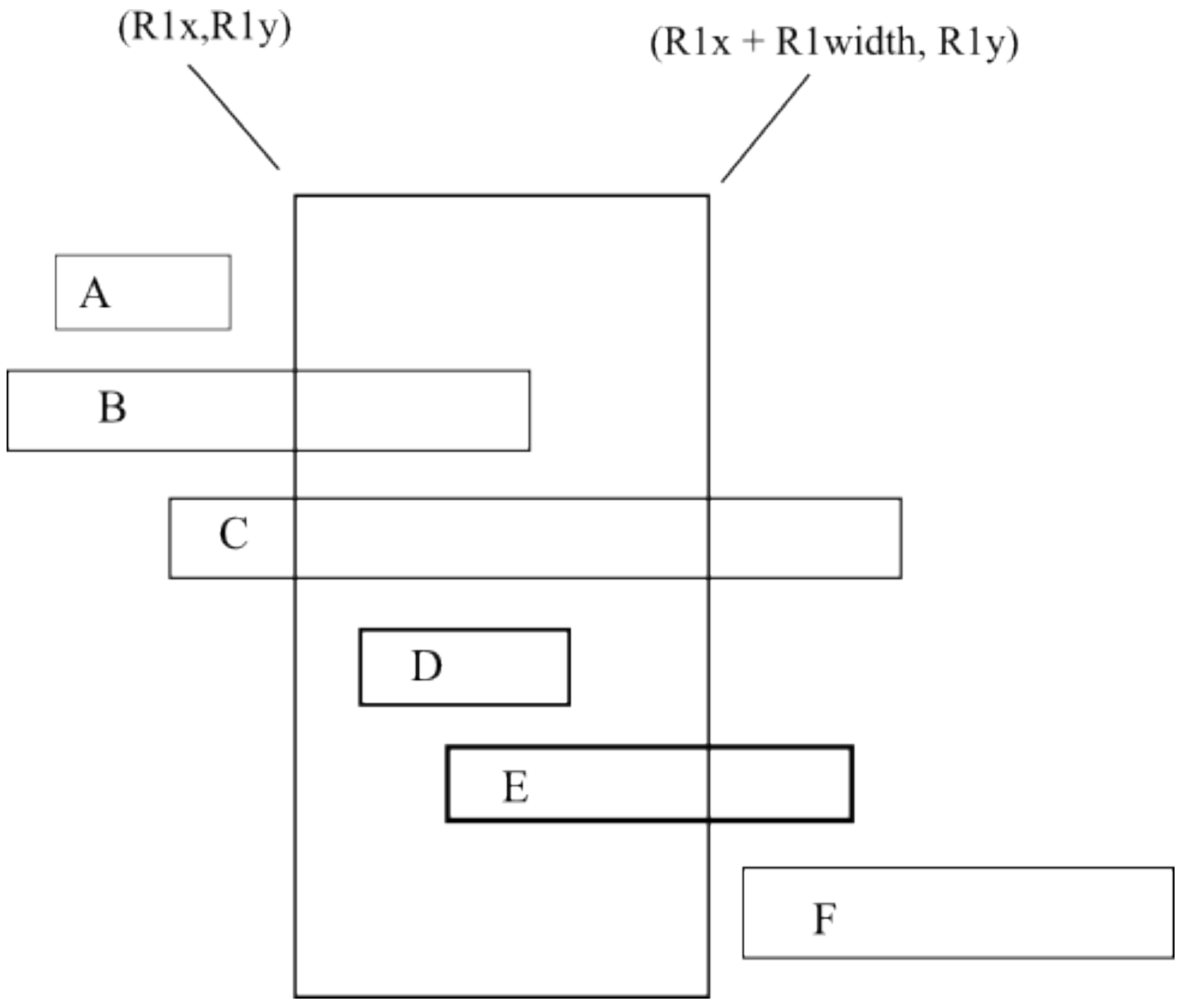
Other geometric tests can be written. Another simple on is detecting whether a point is contained in a circle. In this case we pass in a circle and see if the mouse is contained by that circle:

```
boolean MouseInCircle( float inCircleX, float inCircleY, float inCircleWidth)
{
  // the mouse is inside the cirle if the distance from the mouse to the
  // center of the circle is less than the radius

  float distanceToCenter ;
  float radius = inCircleWidth / 2 ;   // the radius is half the width
  float deltaX = mouseX - inCircleX ;    // distance along x-axis
  float deltaY = mouseY - inCircleY ;    // distance along y-axis
  distanceToCenter = sqrt( deltaX * deltaX + deltaY * deltaY) ;
  if (distanceToCenter < radius)
  {
     return(true) ;
  }
  else
    return(false) ;

}
```

The following code creates a circle and a rectangle and draws them on the screen. When the mouse is moved into the circle the circle turns yellow. When the mouse moves into the rectangle the rectangles turns red. When the mouse departs the circle or rectangle the circle/rectangle returns to black. The functions MouseInRect() and MouseInCircle() are not included below but are as above:

```
float rectX = 125 ;
float rectY = 125 ;
float rectWidth = 150 ;
float rectHeight = 150 ;
float circleX = 100 ;
float circleY = 100 ;
float circleWidth = 150 ;

void setup()
{
  size(400,400) ;
  fill(0) ;

}

void draw()
{
  background(100) ;

  if ( MouseInRect( rectX, rectY, rectWidth, rectHeight ) )
    fill(255,0,0);
  else
    fill(0) ;
  rect(rectX,rectY,rectWidth,rectHeight) ;

  if ( MouseInCircle( circleX, circleY, circleWidth) )
    fill(255,255,0) ;
  else
    fill(0) ;
  ellipse(circleX,circleY,circleWidth,circleWidth) ;

}
```

Another common geometric task, especially in games, is determining whether two rectangles intersect. Often in games a bounding box is fitted around a sprite and two sprites are said to collide if their bounding boxes intersect. This problem is also referred to as collision detection. Consider the following image:



In this figure we show six rectangles, A .. F and one big rectangle. We will refer to the big rectangle as rectangle R. We wish to write code to determine if a given rectangle, A..F, intersect with rectangle R. Rectanlges A..F represent the different possible cases along the x-extent. For rectangles A, B, and C the x-coordinate of the left side is smaller than the left side of rectangle R. Given the x-coordinate of the left hand side is smaller than the left hand side of R, then there are three possible cases: 1) the right hand side is smaller than the left side of R (as for rectangle A); 2) the right hand side is in the middle of rectanlge R (as for rectangle B); or 3) the x-coordinate of the right hand side is greater than the right hand side x-coordinate of rectangle R. If the left hand side of a rectangle is in the middle of x-extend of rectangle R there are two possible cases: 1) the right hand side of the rectangle is also contained in the x-extend of rectangle R (represented by rectangle D); or 2) the right hand side x-coordinate is greater than the right hand side x-coordinate of rectangle R (represented by rectangle E). The last possibility is that the left hand side of a rectangle is greater than the x-coordinate of the rectangle R (represented by rectangle F). We could write a separate boolean test for each of these size possibilities, but, the problem can be simplified to two tests:

a) the left hand side x-coordinate is smaller than the left hand side x-coordinate of rectangle R, in which case, to intersect, the right hand side x-coordinate must be greater than the left hand side x-coordinate of rectangle R.
b) the left hand side x-coordinate is greater than the left hand side x-coordinate of rectangle R and smaller than the right hand side x-coordinate of rectangle R. In this case it does not matter where the right hand side is, it is guaranteed to intersecte

The above test will determine if the two rectangles intersect along the x-extent, but, the intersect they must intersect on BOTH the x and y-extents.

The following function takes two rectangles as arguments and returns true if they intersect and false otherwise. Note, the code shows explicitly that two rectangles must intersect in the x-extent and in the y-extent, hence the and (&&) condition.

```
boolean TwoRectsIntersect(float inRX1, float inRY1, float inW1, float
inH1,
   float inRX2, float inRY2, float inW2, float inH2)
{
  if (
     ( ( (inRX1 < inRX2) && (inRX1+inW1 > inRX2) )  ||
     ( (inRX1 >  inRX2) && (inRX1 < inRX2+inW2) ) )
     &&
     ( ( (inRY1 < inRY2) && (inRY1+inH1 > inRY2) )  ||
     ( (inRY1 >  inRY2) && (inRY1 < inRY2+inH2) ) )
     )
     return (true) ;
  else
     return(false) ;
}
```

As in most non-trivial problems in computer programming, there are more than one possible solutions. Another approach would be to test if any of the corners in rectangle 1 are contained in rectangle 2, and if any of the rectangles in rectangle 2 are contained in rectangle 1. If any of them are, then the rectangle intersect. Further, we already have written a function that returns true if a point is contained in a rectangle, so, we can use that function in a rectangle intersect test as follows. First, the function to test if a point is contained in a rectangle:

```
boolean PointInRect(float inPX, float inPY, float inRectX, float inRectY, float
inRectWidth, float inRectHeight)
{
  if ( (inPX > inRectX) && (inPX < (inRectX + inRectWidth) )  &&
       (inPY > inRectY) && (inPY < (inRectY + inRectHeight) ) )
       return (true) ;
  else
       return(false) ;
}
```

Now, the function to see if two rectangle intersect using this approach:

```
boolean TwoRectsIntersectByPoints(float inRX1, float inRY1, float inW1, float
inH1,
   float inRX2, float inRY2, float inW2, float inH2)
{
  if ( PointInRect( inRX1, inRY1, inRX2, inRY2, inW2, inH2) )
    return(true) ;
  if ( PointInRect( inRX1, inRY1+inH1, inRX2, inRY2, inW2, inH2) )
    return(true) ;
  if ( PointInRect( inRX1+inW1, inRY1, inRX2, inRY2, inW2, inH2) )
    return(true) ;
  if ( PointInRect( inRX1+inW1, inRY1+inH1, inRX2, inRY2, inW2, inH2) )
    return(true) ;

  // comment out the next 8 lines and test, you will see they are necessary.  WHY?
  if ( PointInRect( inRX2, inRY2, inRX1, inRY1, inW1, inH1) )
    return(true) ;
  if ( PointInRect( inRX2, inRY2+inH2, inRX1, inRY1, inW1, inH1) )
    return(true) ;
  if ( PointInRect( inRX2+inW2, inRY2, inRX1, inRY1, inW1, inH1) )
    return(true) ;
  if ( PointInRect( inRX2+inW2, inRY2+inH2, inRX1, inRY1, inW1, inH1) )
    return(true) ;

  // if make it here, the rectangles do NOT intersect
  return(false) ;
}
```

And here is code that demonstrates the use of these functions. When you move the mouse around a rectangle follows it.

```
float rectX = 125 ;
float rectY = 125 ;
float rectWidth = 150 ;
float rectHeight = 150 ;

float rectWidth2 = 100 ;  // the width of the rectangle that follows the mouse
float rectHeight2 = 100 ;  // the height of the same rectangle


void setup()
{
  size(400,400) ;
  fill(0) ;
}

void draw()
{
  background(100) ;

  fill(0) ;
  rect(rectX,rectY,rectWidth,rectHeight) ;  // draw the fixed rectangle

  // if ( TwoRectsIntersectByPoints( rectX, rectY, rectWidth, rectHeight, mouseX,
mouseY, rectWidth2, rectHeight2) )
  if ( TwoRectsIntersect( rectX, rectY, rectWidth, rectHeight, mouseX, mouseY,
rectWidth2, rectHeight2) )
    fill(255,0,0) ;  // if they intersect set fill to red
  else
    fill(0) ;

  // draw a rectanlge at the mouse location
  rect(mouseX,mouseY,rectWidth2,rectHeight2) ;
}
```

EXERCISE 12A

Write a sketch that has a large rectangle that moves with the mouse, and 5 or more small rectangles that move around on their own (with individual x and y-velocities and x,y,width,heights). When these independent rectangles intersect they large mouse-controlled rectangle they turn red. When the independent rectangles move off the large rectangle the turn and remain black until they touch the large rectangle again.