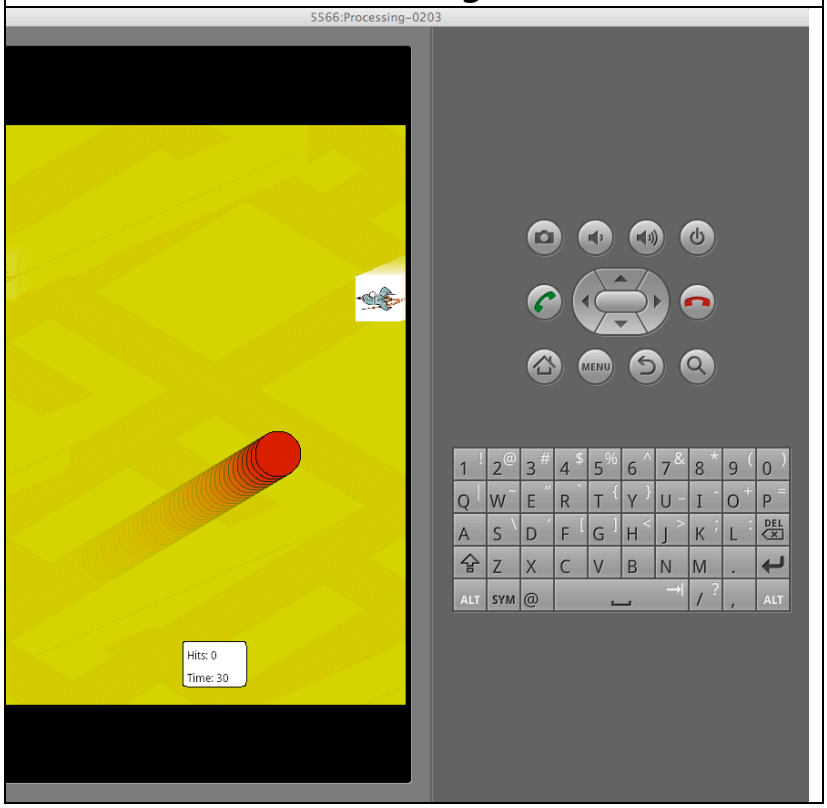
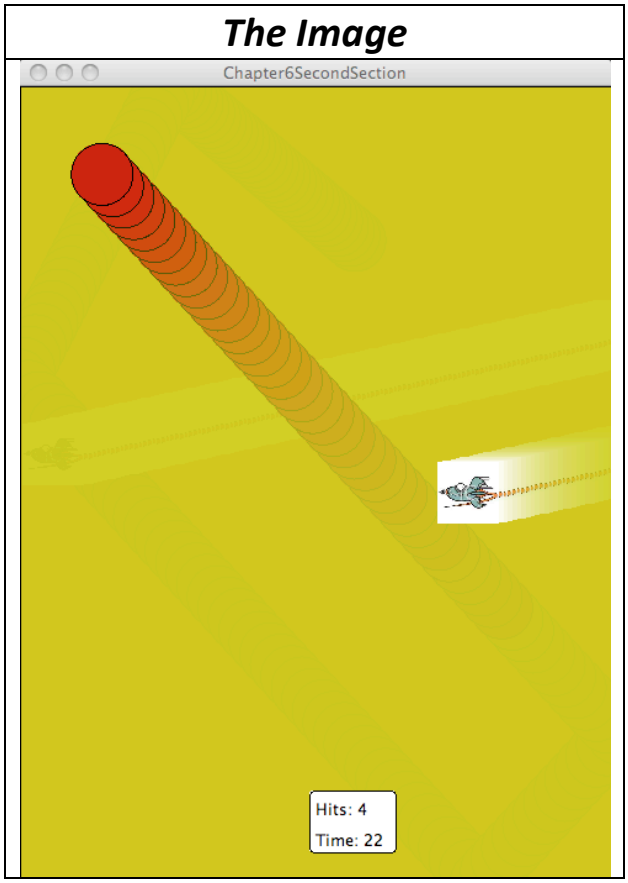
# *The Junior Woodchuck Manuel of Processing Programming for Android Devices*

***The Image***



# *Chapter 7*
# *Be It Ever So Humble . . .*

***The Image***




## *In our last exciting chapter our hero was. . .*

***Sorry… That was what we call a "senior moment". . .***

***In this last chapter we will try to do a lot –probably too much so you may have some work you can finish on your own.***

***There will be no "formal" review in this chapter. You need to go back over the previous chapters and look at our code and your code to be sure you understand as much as possible.***

# *Combining the Code*

***If you have a computer at home that has Processing installed, you can get a head start by combining two programs you wrote earlier. Here is what we need:***

- ***the code from the program that bounces the circle off of the walls – just the variables and functions that bounce the circle – not the rectangle.***
- ***the code from the program that uses the mouse to control the movement of the rectangle – the variables and functions.***

***These two sets of code need to be copied and pasted into a single new program that has a bouncing circle and a mouse controlled movable rectangle.***

***That is our starting point.***

***The next two pages show our code that does this. Here is the color key to reading it:***

- ***The red code is required for the bouncing circle.***
- ***The blue code is for the movable rectangle.***
- ***The black code is needed by both.***

***You can copy and paste this code into a Processing program. There may be a few copy/paste errors for some of you. Using our code is not ideal because you may not understand what some of the code does. It is much better if you use your own code with your own variable names for this.***

```
float circleX ,circleY, yChange, xChange,
      circleLeft, circleRight, circleUp,
      circleDown;

float rectX,rectY, rectHorizontalSpeed,
      rectVerticalSpeed, changeAmount;

void setup( )
{
    size( 480, 640 );
    noFill( );

    circleX = 400;
    circleY = 100;
    circleLeft = -3;
    circleRight = 3;
    circleUp = -5;
    circleDown = 5;
    yChange = circleLeft;
    xChange = circleDown;

    rectX = 100;
    rectY = 100;
    rectHorizontalSpeed = 0;
    rectVerticalSpeed = 0;
    changeAmount = 0.1;
}
void draw( )
{
   prepareScreen( );

   circleFigure( );
   moveCircleFigure( );
   checkCircleFigure( );

   rectFigure( );
   moveRectFigure( );
   checkRectFigure( );
}

void prepareScreen( )
{
   fill( 200, 200, 0, 5 );
   rect( 0, 0, width, height );
}
```

```
void circleFigure( )
{
   fill( 200, 0, 0 );
   ellipse( circleX, circleY,  20, 20 );
}

void moveCircleFigure( )
{
    circleY = circleY + yChange;
    circleX = circleX + xChange;
}

void checkCircleFigure( )
{
    if ( circleY < 0 )
    {
       yChange = circleRight;
    }
    if( circleY > height)
    {
       yChange = circleLeft;
    }
    if ( circleX < 0 )
    {
       xChange = circleDown;
    }
    if( circleX > width)
    {
       xChange = circleUp;
    }
}
```

```
void rectFigure( )
{
   fill( 0, 0, 200 );
   rect( rectX, rectY,  20, 20 );
}

void moveRectFigure( )
{
    rectY = rectY + rectVerticalSpeed;
    rectX = rectX + rectHorizontalSpeed;
}

void checkRectFigure( )
{
    if ( rectY < 0 )
    {
       rectY = height;
    }
    if( rectY > height)
    {
       rectY = 0;
    }
    if ( rectX < 0 )
    {
       rectX = width;
    }
    if( rectX > width)
    {
       rectX = 0;
    }
}
```

```
void mouseDragged ( )
{
   if ( mouseX > pmouseX)
   {
     rectHorizontalSpeed =
      rectHorizontalSpeed + changeAmount;
   }
   if ( mouseX < pmouseX)
   {
     rectHorizontalSpeed =
      rectHorizontalSpeed - changeAmount;
   }

   if ( mouseY > pmouseY)
   {
     rectVerticalSpeed =
       rectVerticalSpeed + changeAmount;
   }
   if ( mouseY < pmouseY)
   {
     rectVerticalSpeed =
       rectVerticalSpeed - changeAmount;
   }
}
```

# *The Idea Behind the Game*

***What we want to do is make a game using this code. The idea is somewhat simple – the user will "drive" the rectangle around the screen with the mouse or their finger and try to hit the bouncing circle. Hits will be counted and "dramatically" displayed – don't get too excited. . .***

***We begin by asking question – not by coding.***

***Before looking at the next page, think about what functions you might need to add to the program to do this. Do not think about variables or how to code the functions, just think about what functions must be added to this program to make the game. Write your ideas down.***

*Here is what your teacher came up with. The changes are in green.*

```
void draw( )
{
   prepareScreen( );

   circleFigure( );
   moveCircleFigure( );
   checkCircleFigure( );

   rectFigure( );
   moveRectFigure( );
   checkRectFigure( );

   checkCollisions( );
   showStats( );
}
```

*Nothing major here. This is all we think we need. After we move , draw, and check the circle and move, draw, and check the rectangle, we will check for a hit and show the stats.*

*How are the hits and the stats related?*
*We want to count the hits.*

*Do we need any new variables? Again, think and write before looking at the next page.*

*Here are your teacher's thoughts:*

```
float circleX ,circleY, yChange, xChange,
      circleLeft, circleRight, circleUp,
      circleDown;

float rectX,rectY, rectHorizontalSpeed,
      rectVerticalSpeed, changeAmount;

int hitCount;
```

*The word,* int *is new for us. An* int *is a number without a decimal Your teacher did not like displaying 3.000 hits. To get rid of the decimal point, he decided to use* int *. You can use float or* int *– it is up to you.*

*We need to initialize the new variable in setup( ):*

```
void setup( )
{
    size( 480, 640 );
    noFill( );

    circleX = 400;
    circleY = 100;
    circleLeft = -3;
    circleRight = 3;
    circleUp = -5;
    circleDown = 5;
    yChange = circleLeft;
    xChange = circleDown;

    rectX = 100;
    rectY = 100;
    rectHorizontalSpeed = 0;
    rectVerticalSpeed = 0;
    changeAmount = 0.1;

    hitCount = 0;
}
```

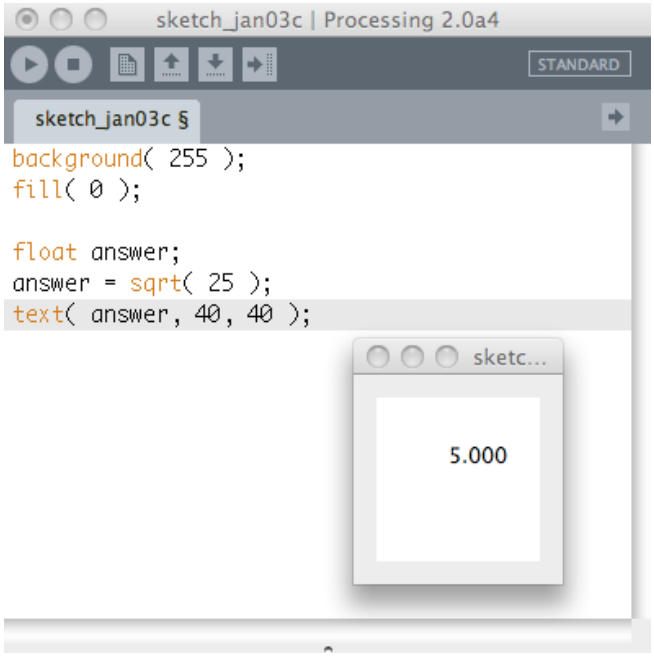
# *Did the Hit?*

***Now for the collisions. In order to do this we need to look at some new stuff. Hey! – this is school… and doing "new stuff" is part of the deal…***

***You have used lots of functions the past five weeks. Processing wrote some of them and you wrote some of them and they all "did" something. There is another type of function – a function that does some arithmetic and computes an answer that we can use in our code. Programmers say that a function like this "returns" a value.***

***if you have used a calculator, you have entered a number and pressed some other button like the + button or may a sqrt button. The calculator takes your number ( the input ) and does something with it ( add it if you pressed the + or computes the square root if you pressed the sqrt button) and displays the answer (the output or the return value). The calculator "returned" the result of the addition or the square root.***

***Programming languages like Processing have functions that can take arguments, compute something using them, and then return a value to our code so we can use it. Here is a simple Processing program that uses Processing's sqrt( ) function and the output it makes:***



*We will talk about the text( ) function a bit later. For now look how this code uses the value that the sqrt( ) function computes and returns. Notice that the argument is 25.*

*In order to see out if the user managed to drive the rectangle and hit the circle we have to use a Processing function. We could to this with the sqrt( ) function. In fact, a few years ago we had to use the sqrt( ) function to do this. But the Processing programmers added a new function that is very helpful. This function is the* dist( ) *function.*

*The* dist( ) *function takes four arguments:*

- *x coordinate of one of the figures*
- *y coordinate of the same one of the figures*
- *x coordinate of the other figure*
- *y coordinate of the same other figure.*

*and computes and returns the distance between the two figures.*

*We have two figures: the circle and the rectangle so we can use their coordinates as the arguments.*

```
dist( circleX, circleY, rectX, rectY );
```

*and it will return the distance between the circle and rectangle.*

*We have to store the value that is returned by the dist( ) function so we need a new variable such as distance. Once we have added the distance variable to our program:*

```
float distance:
```

*we can use it like this:*

```
void checkCollisions( )
{
    distance = dist( circleX, circleY, rectX, rectY );
```

*Now we know how far apart the circle and rectangle are.*
*What do we need to ask?*
*We need to ask if they hit.*

*How do we do that?*
*We can ask if the distance is smaller than some number of pixels?*

***What number do we use?***

***That is up to you, the programmer. You can use a magic number like 42 or the diameter of the circle or the length of the rectangle – it is up to you.***

***How do we ask a question?***

***Think back -- we used the if( ) to ask questions. Remember this code:***

```
if ( circleY < 0 )
{
    yChange = circleRight;
}
```

***and this code:***

```
if ( rectY < 0 )
{
   rectY = height;
}
```

***We need to something similar… and we leave that code to you?***

***Next part of the collision stuff – what do we do when there is a collision?***

***When your code answers true to the question "did they hit?" we should do several things inside the { } of the if( ):***

- ***add one to the variable that is remembering how many hits and***
- ***"dramatically" demonstrate the hit in some visual form so the user knows she/he was successful.***

***You should be able to figure out the code for adding one to the hit variable.***

***The "dramatic" demonstration of the hit is also up to you. Your teacher just draws a huge red rectangle – boring…***

***You should try to get this working now. We will do the stats stuff in a bit.***

# *Displaying the Stats*

***The user of your game will want to know how good they are so we need to show some stats on the screen. We can show the number of hits. And maybe how long they have been playing.***

***More new Stuff Alert…***

***If you go back to the code picture that was used to show the sqrt( ) function you will see a new function named text( ). The API tells us that the*** `text( )` ***function takes three arguments:***

1. ***what to display***
2. ***x coordinate of the text***
3. ***y coordinate of the text***

***That's it. But there is more new stuff. It turns out that we can combine words and numbers using the + operator. Here is your teacher's display again:***


***He combined the word "Hits: " with the number of hits.***
***This is pretty easy to do. He just wrote this in his program:***

```
text( "Hits: " + hitCount, width/2, height-50 );
```

***He used the + sign to combine the stuff inside the " " with the name of the variable that is remembering the number of hits. Processing looked up the value that the variable hitCount was remembering and pasted it next to the stuff inside the " ". Programmers call the stuff inside the " " a string as in a "string of letters". Notice that there is a space after the : in the string. Without this space, the number 4 would be jammed next to the : -- remember this if you want your program to look good and to be easy to read.***

***The second argument,*** `width/2` ***is the x coordinate of the lower left corner of the text,***

*The third argument,* `height-50` *is the y coordinate of the lower left corner of the text.*

*What about the time?*
*Well – that is another function called* `millis( )` *that stands for milliseconds or 1/1000 of a second. Your teacher wanted to display seconds instead of 1/1000 of a second so he used some arithmetic to convert the number of milliseconds to seconds.*

*How would you do this?*

*We will experiment with this in class. You can read the API for millis( ) to see if that is any help.*

*Also, look very carefully at the white background rectangle in your teacher's stats display.*


*It has rounded corners. This is something new in Processing and it may not be in the API by the time we do this in class. So, if you are interested, read on…*
*The API for the rect( ) function is this:*

**Syntax** `rect(x, y, width, height)`

*The Processing programmers added another way to use the rect( ) function. We can add a fifth argument that is the "roundedness' of the corners:*

**Syntax** `rect(x, y, width, height, roundedness)`

*Feel free to use this if you want to but beware – it does not work in the 1.5.1 version of Processing. It only works in the 2.0.4A version.*

# *Did You Happen to See a Small Bug?*

***What? Bug?? Not us!!!***

***Actually there is a small problem. If you run the program, you may notice that you get many hits counted for a single collision.***

***Why do you think this happens?***
***Think about this before going to the next page.***

# *Gotcha again…*
# *heh. heh.. heh…*

***The reason we can get many hits for a single collision is that the circle and square can remain close enough to each other for several frames and each frame adds one to the variable that is remembering the number of hits.***

***If you like this, fine..***

***If not, then how can we fix it?***
***Think about it before going on.***

## *Not this time…*

***Here is one idea – we can relocate the circle to a new place inside the if( ) that asks if they collide. Remember if the answer is true – they collided, we can do anything we want inside the { } of the if( ).***

***So, we can reposition the circle. We can even change is speed. This might make the game more exciting.***

***Let's see what we have to do to put the circle in a different place.***
***Here are the variables that the circle uses:***

```
float circleX ,circleY, yChange, xChange,
      circleLeft, circleRight, circleUp,
      circleDown;
```

***it looks like the two that remember where the circle is are:***

```
circleX ,circleY
```

***so we can just change them.***

***We could set them to magic numbers every time but let's try something else. Let's set them to random places.***

***Processing has a function that returns a random value between the two arguments type in the parentheses. It looks like this:***

```
circleX = random( 2, 10 );
```

***The random( ) function uses some kind of arithmetic to compute and return a number between 2 and 10. All we have to do is decide the two arguments.***

***Changing the speed is more involved. Here is how we use the speed variable:***

```
circleX = 400;
circleY = 100;
circleLeft = -3;
circleRight = 3;
circleUp = -5;
circleDown = 5;
yChange = circleLeft;
xChange = circleDown;
```

***The reason we have the variables circleRight and circleLeft is so the circle can bounce both left and right. The value of circleRight is always positive***

***and the value of circleLeft is always negative. The variable circleRight is 3 and the variable cirleLeft is -3. The circleUp and circleDown variables behave the same way.***

***If we want to change the horizontal speed, we have to do this with two steps:***

1. ***get a new random value for circleRight which will be positive, and***
2. ***give circleLeft the negative value of circleRight.***

***Think about how to code this – we will talk about it in class.***

***We have to do the same thing for circleDown and circleUp.***

# *Home At Last…*

***Well, that's it for us. We sincerely hope you have enjoyed this past seven week and learned something about programming.***

***We also hope that you continue to explore Processing and the API and continue to learn to program.***

***Remember that we did a very small part of programming with Processing. There is much more to learn and all of it is exciting.***

***You can program with pictures, video, sound, and create your own special effects. All it takes is time, practice and patience.***

***Good Programming…***