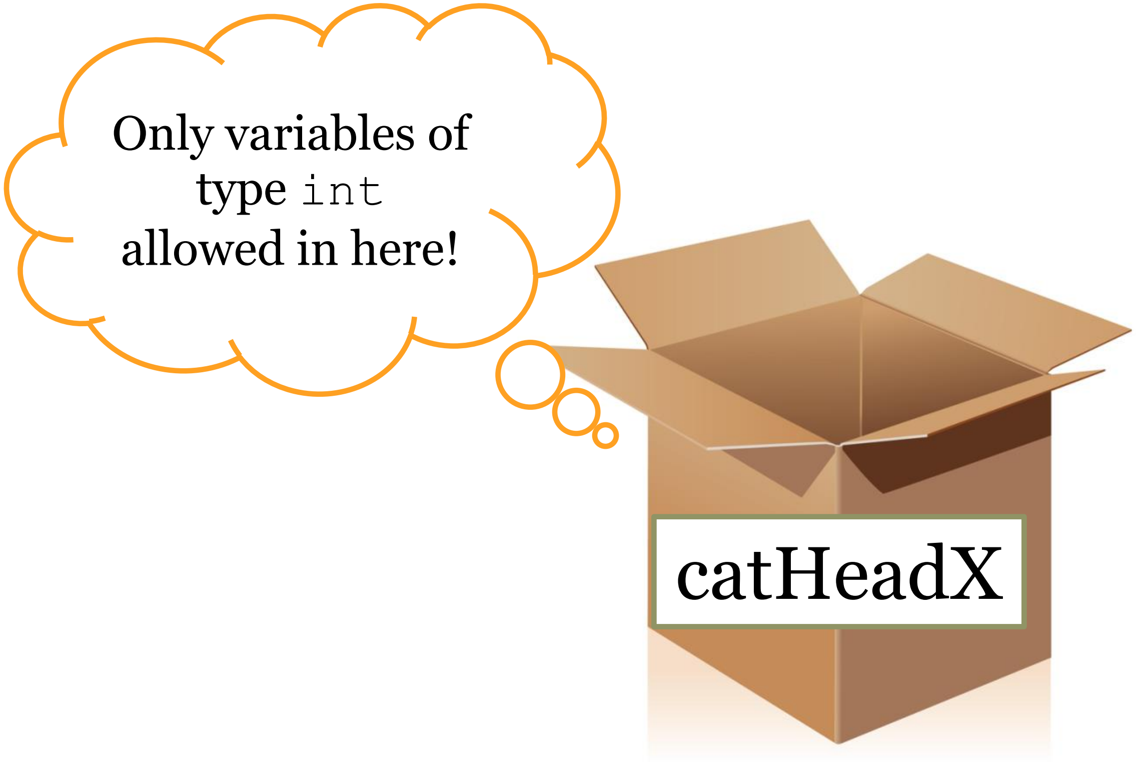
A Simple Character AI

type casting, while loops, arrays, modulus operator, operator precedence

Data Type	Values
boolean	true/false
byte	generic 8 bits of data
char	character (‘a’, ‘b’, …)
color	a grayscale or RGB color
double	floating point with double precision
float	floating point (number with a decimal point)
int	integer (whole number)
long	really big integer

Data Type	Values
boolean	`false` or `true`
byte	whole number from -128 to 127
char	a single letter with single quotes, e.g. `‘a’`
color	`color(45, 67, 34)`
double	`5.0,10.3,4.565456`
float	`5.0f,10.3f,4.565456f`
int	`-2,147,483,648` to `2,147,483,647`
long	`-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807`



Only variables of type `int` allowed in here!

Variable type → `int catHeadX;`

Type Conversion

```
int numInt = 10;
float numFloat = numInt;
```

Type Conversion

```
int numInt = 10;
float numFloat = numInt;
```

this works!

Type Conversion

```
float numFloat = 10.5f;
int numInt = numFloat;
```

Type Conversion

```
float numFloat = 10.5f;
int numInt = numFloat;
```

this doesn’t ☹

Type Conversion

```
float numFloat = 10.5f;
int numInt = (int)numFloat;
```

convert (“cast”)
the int to a float

Type Conversion: No Casting

From	To
boolean	(not applicable)
byte	short, int, long, float, or double
char	int, long, float, or double
color	(not applicable)
double	(none)
float	double
int	long, float, or double
long	float or double

Type Conversion: Yes Casting

From	To
boolean	(not applicable)
byte	(none)
char	byte or short
color	(not applicable)
double	byte, short, char, int, long, or float
float	byte, short, char, int, or long
int	byte, short, or char
long	byte, short, char, or int

Breaking the problem down…

1. Create a character that moves toward the mouse.
2. Give character three states of behavior.
3. Draw one instance of the colored rings.
4. Animate the colored rings so they change colors.

Step 1

Create a character that moves toward the mouse.

Step 1a:

Trigonometry to move forward in current direction

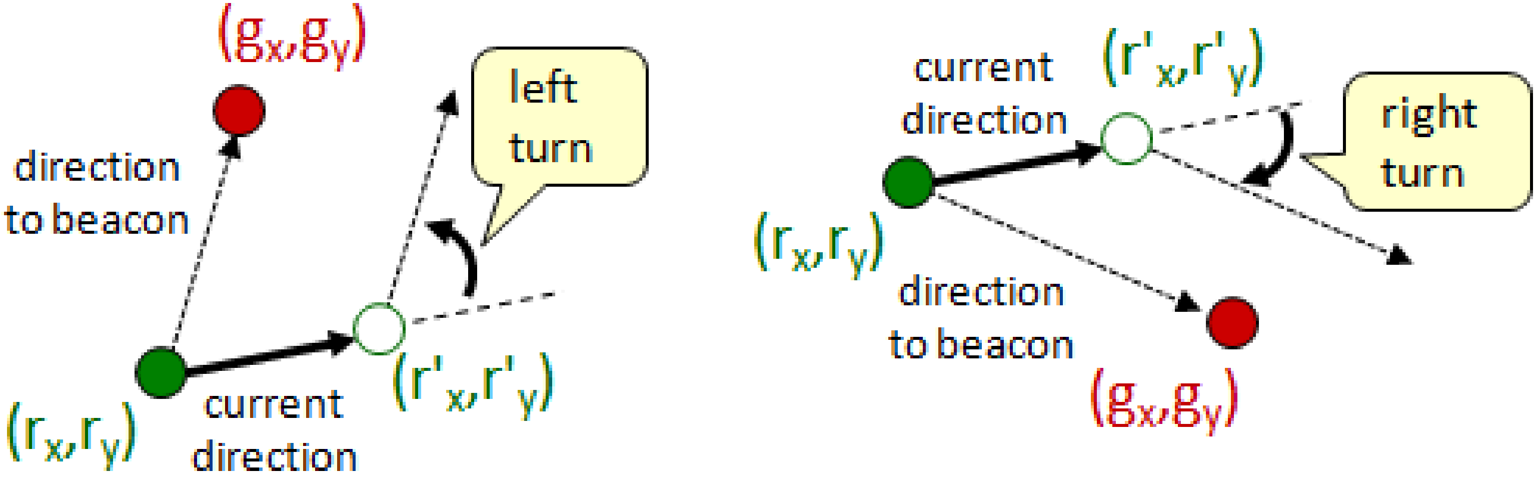
$$x = x + \text{speed} * \cos(\text{direction})$$

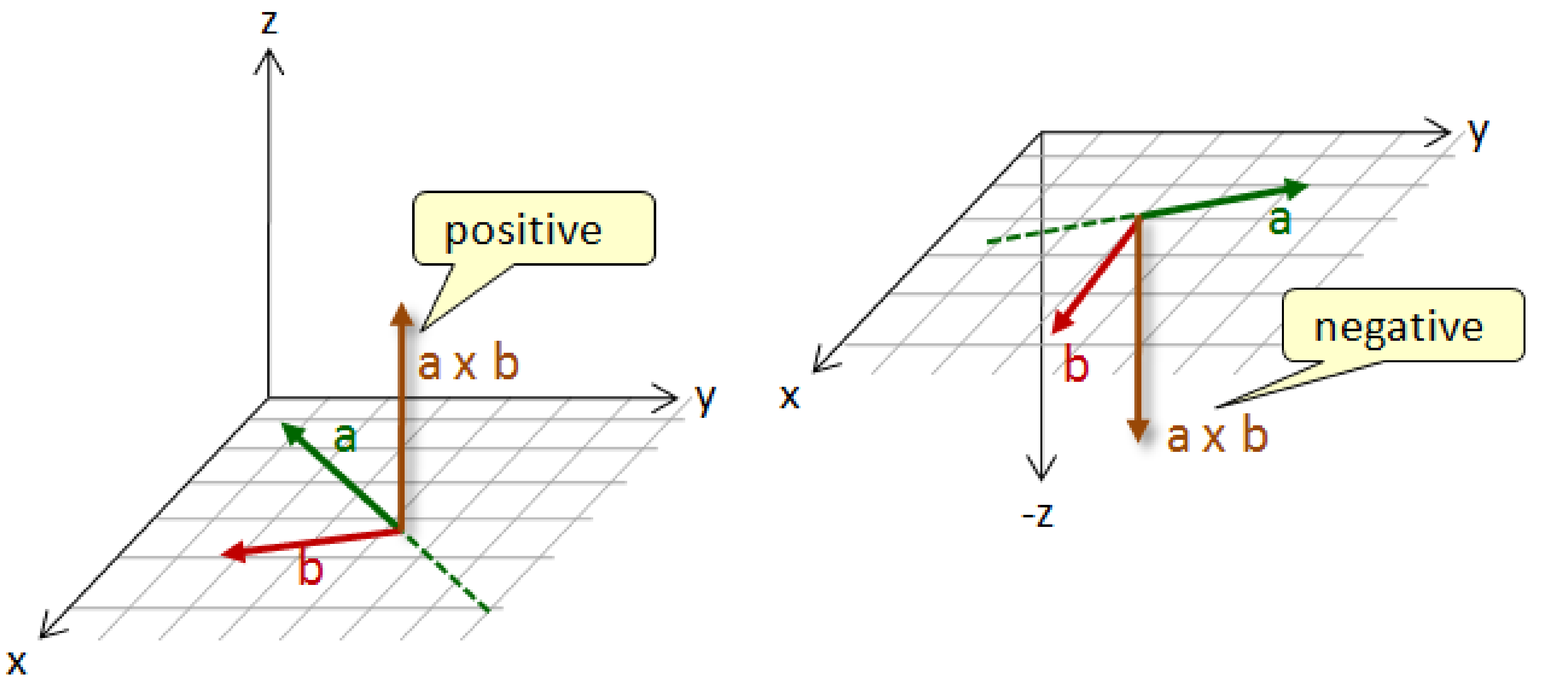
$$y = y + \text{speed} * \sin(\text{direction})$$

Step 1a:
Trigonometry to move forward in current direction

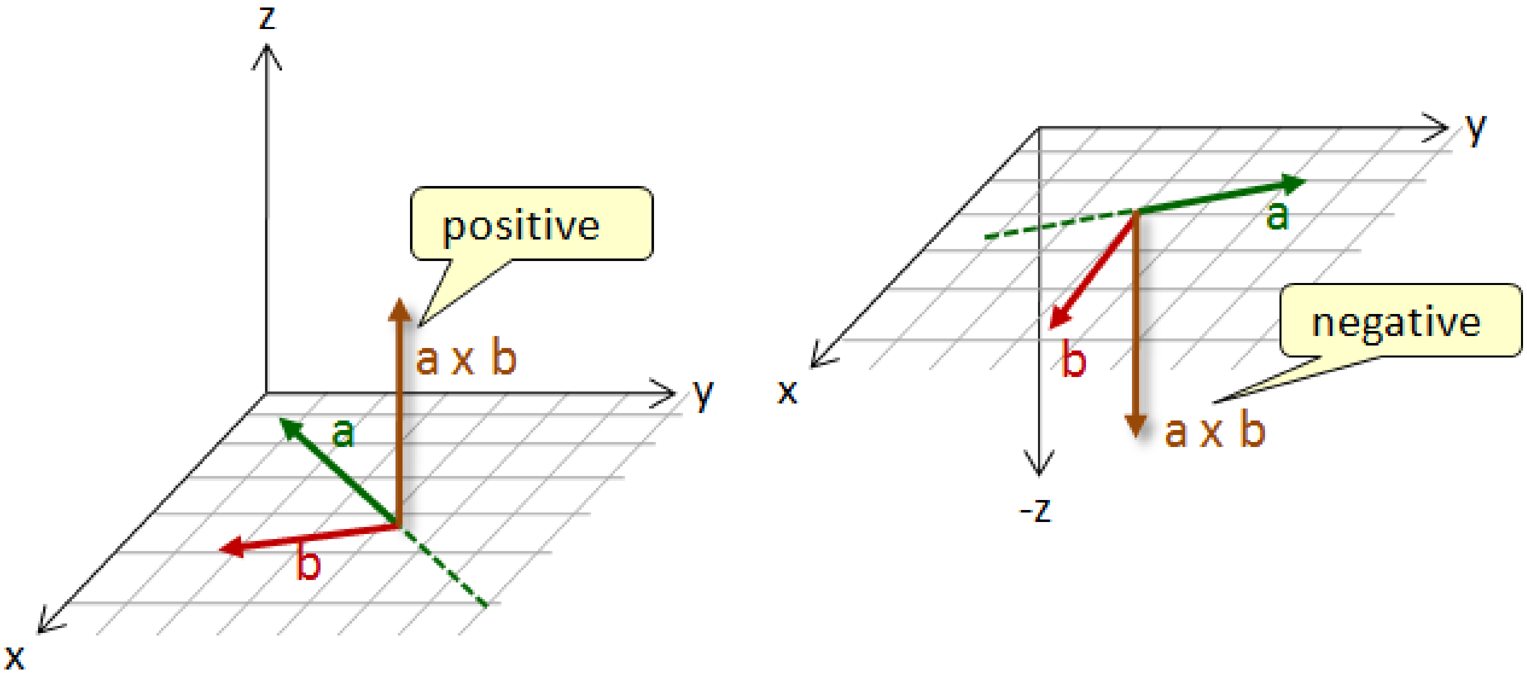
```
int nextX = sheepX + int(sheepSpeed * cos(sheepDirection));
int nextY = sheepY + int(sheepSpeed * sin(sheepDirection));
```

Step 1b:
Change direction to head for beacon (mouse location)

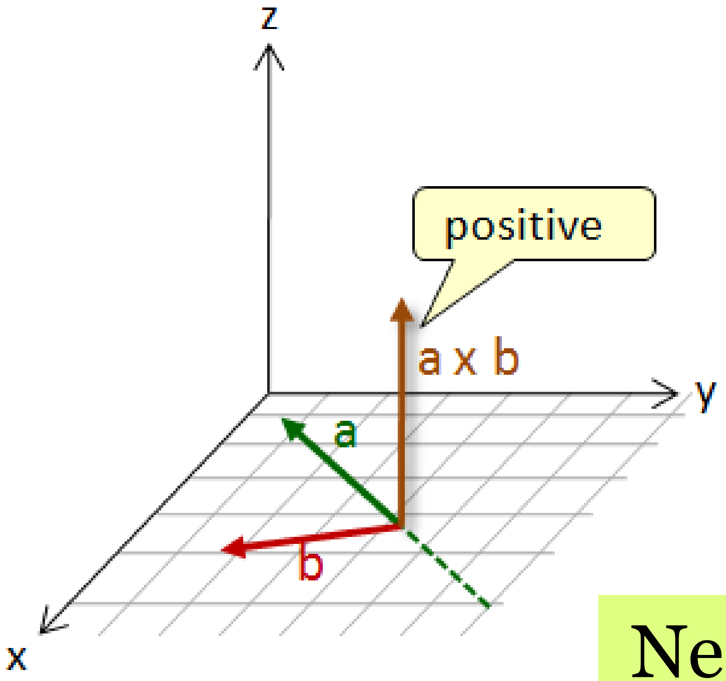


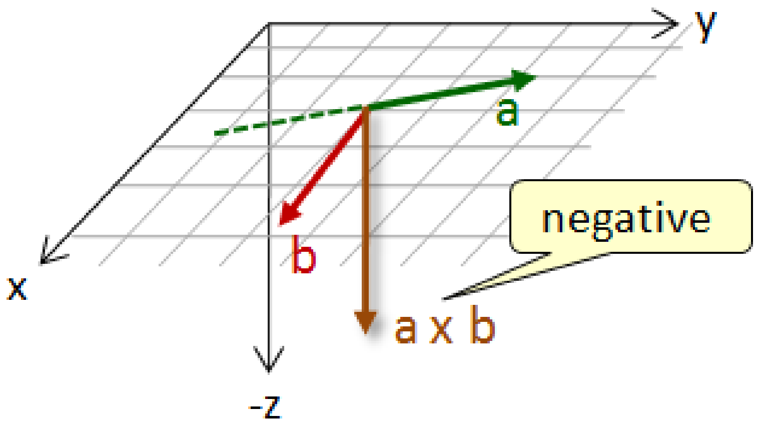


$$\text{crossProduct} = (\mathbf{r'}_\mathbf{x} - \mathbf{r}_\mathbf{x})(\mathbf{g}_\mathbf{y} - \mathbf{r}_\mathbf{y}) - (\mathbf{r'}_\mathbf{y} - \mathbf{r}_\mathbf{y})(\mathbf{g}_\mathbf{x} - \mathbf{r}_\mathbf{x})$$



```
int crossProduct =
    (nextX - sheepX)*(mouseY - sheepY) -
    (nextY - sheepY)*(mouseX - sheepX)
```



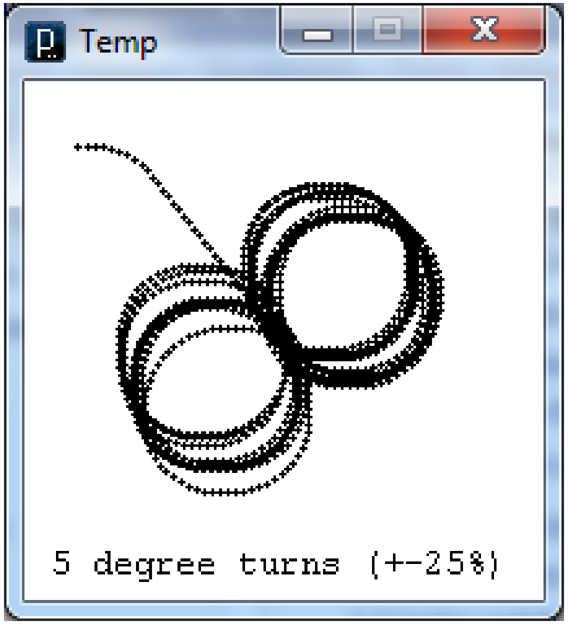


Negative result: turn right
Positive result: turn left

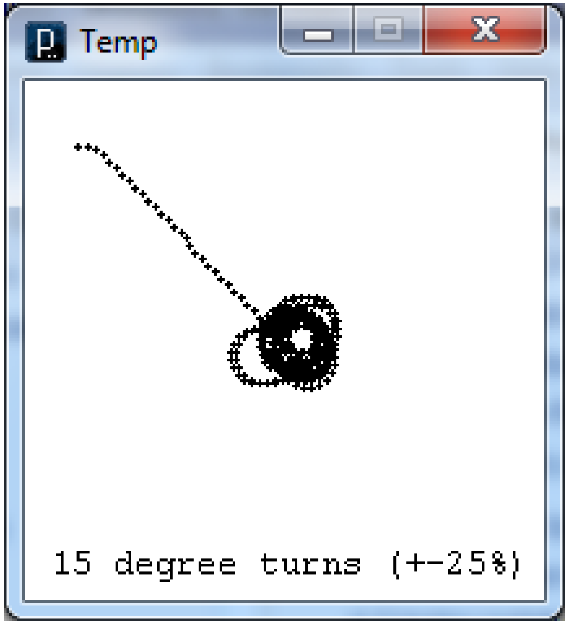
```
int crossProduct =
    (nextX - sheepX)*(mouseY - sheepY) -
    (nextY - sheepY)*(mouseX - sheepX)
```

Step 1c:
Add random error for realism

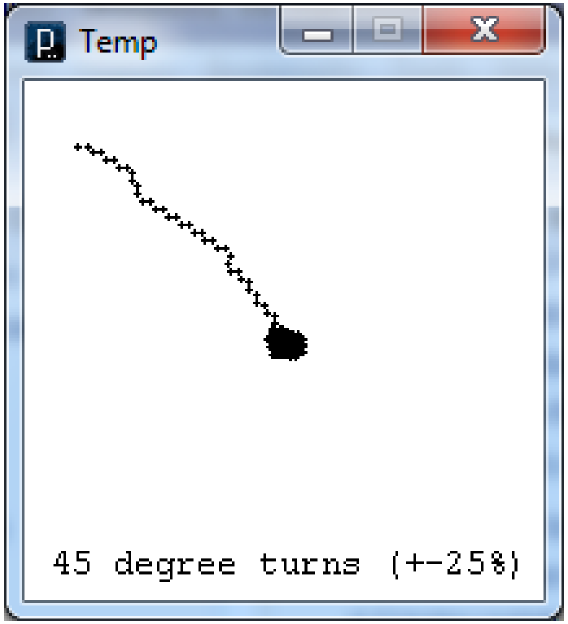
amountToTurn =
θ + random(θ/4) - (θ/4)/2



5 degree turns (+-25%)



15 degree turns (+-25%)



45 degree turns (+-25%)

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
else // turn left
{
  sheepDirection += radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
```

This is theta, the base angle to turn, in degrees

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
else // turn left
{
  sheepDirection += radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
```

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);

                          s(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
```

Subtract 30 degrees plus or minus a small random amount

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
```

Subtract 30 degrees plus or minus a small random amount

(We subtract to turn right since positive y is down!)

```
                          s(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
```

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
else // turn left
{
  sheepDirection += radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
```

We add up to 7.5 degrees...

```
final int angleToTurn = 30;
if (crossProduct < 0) // turn right
{
  sheepDirection -= radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
else // turn left
{
  sheepDirection += radians(angleToTurn
                            + random(angleToTurn/4)
                            - angleToTurn/8);
}
```

…then subtract half of 7.5 to shift the range.

Step 1d:
Only turn 5% of the time

```
if (random(1) < 0.05)
{
   // turning code
}
```

Step 2

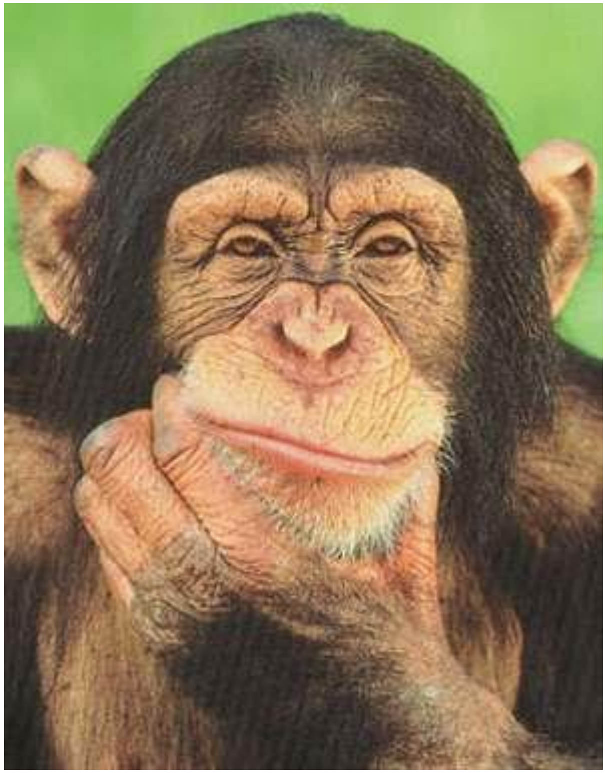
Give character three states of behavior.

In artificial intelligence…

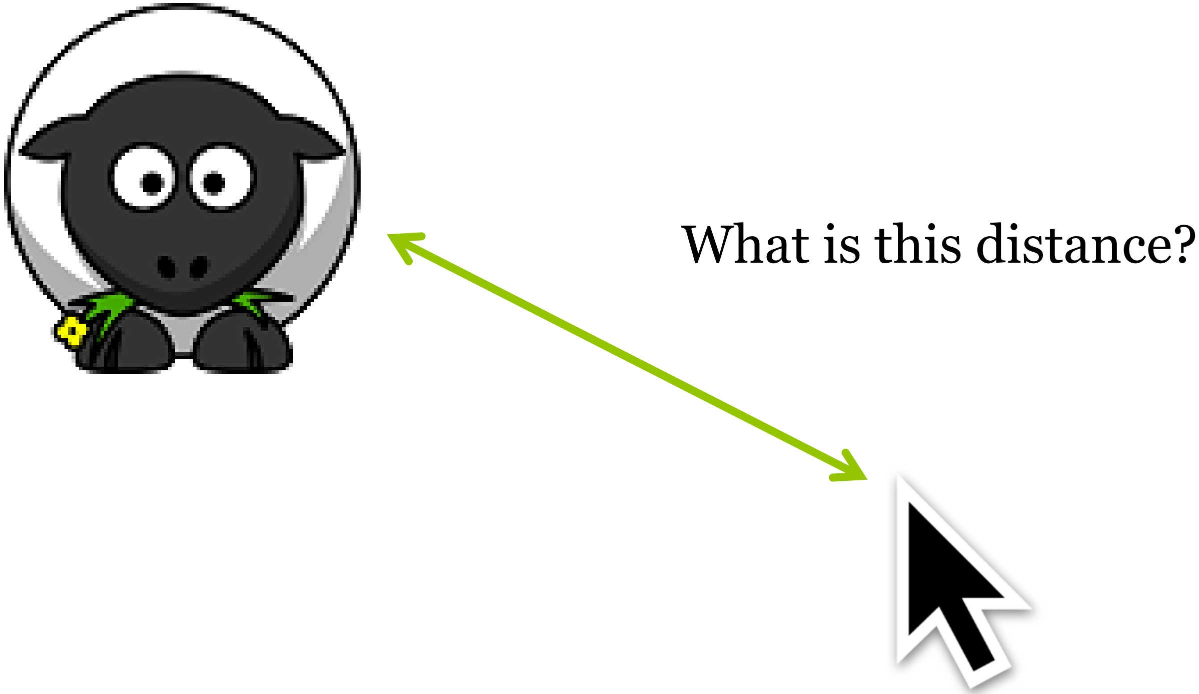
Sense what’s happening

Think about what should be done

Take action

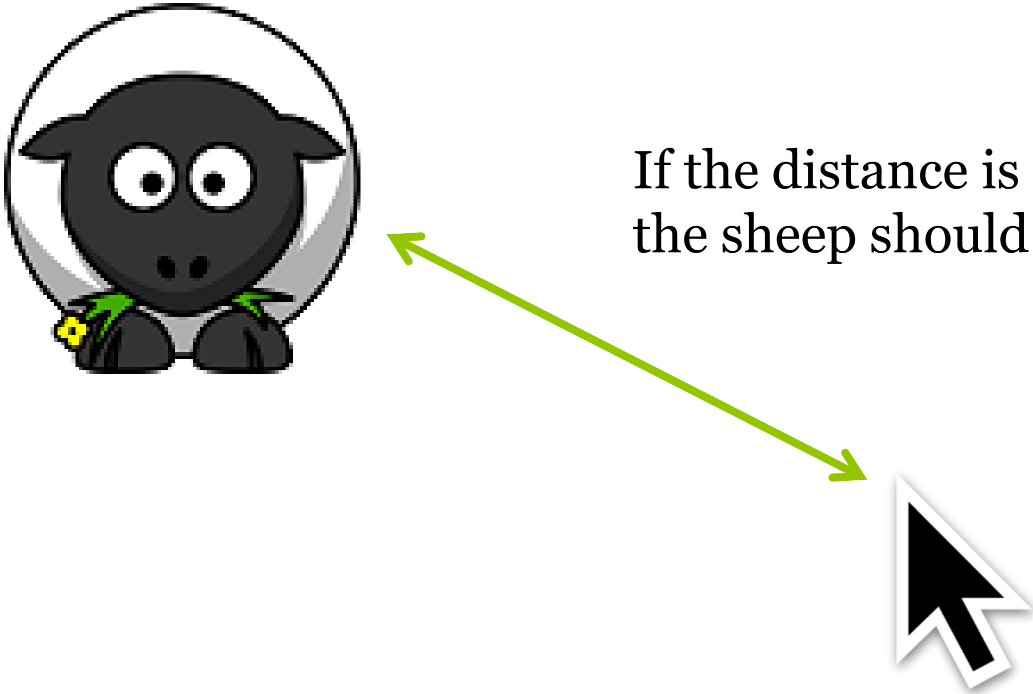


Sense what’s happening…



What is this distance?

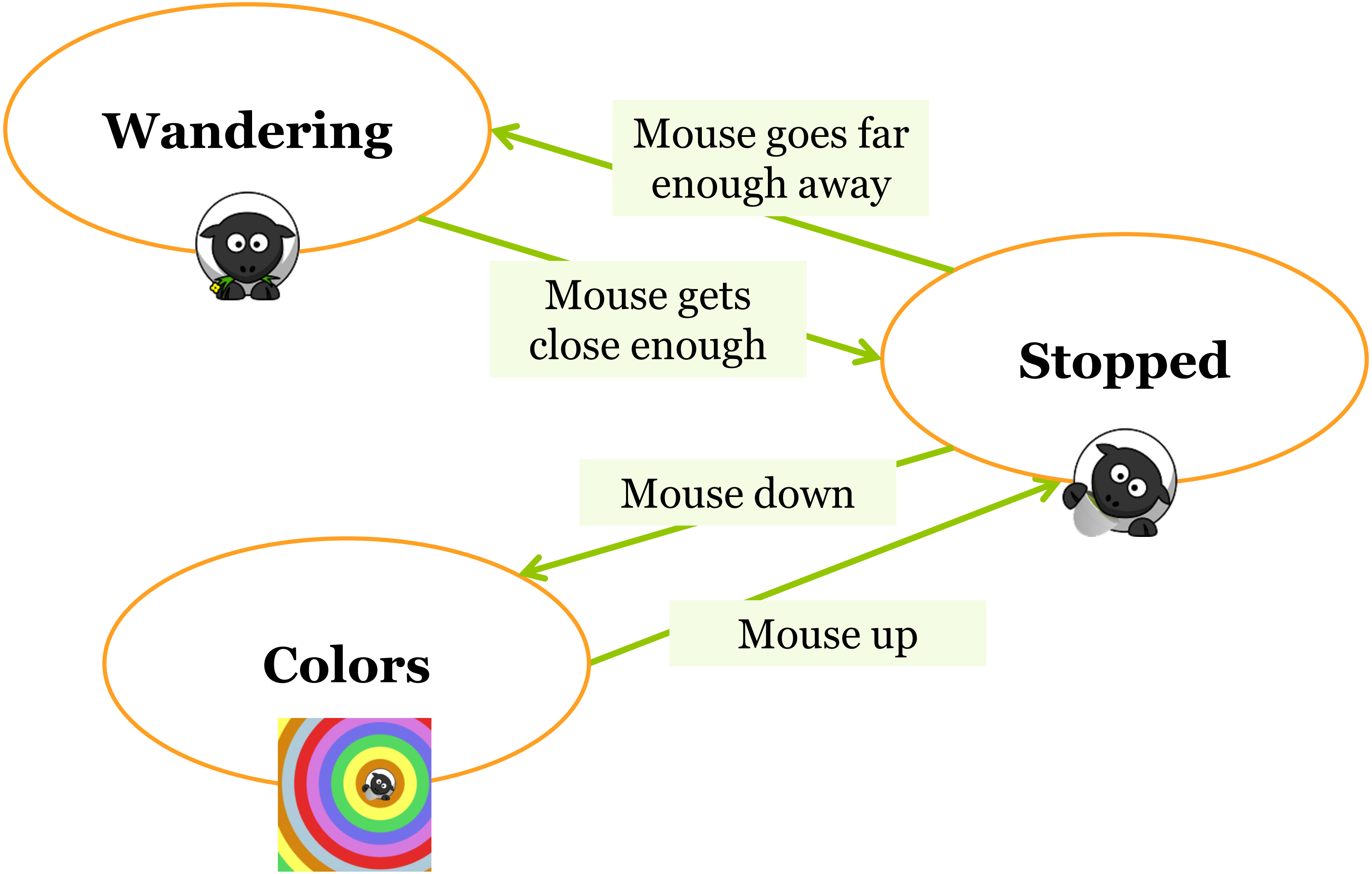
Think about what should be done...



If the distance is small enough,
the sheep should start drinking tea.

Take action!

State Machine



Wandering

Stopped

Colors

Mouse goes far enough away

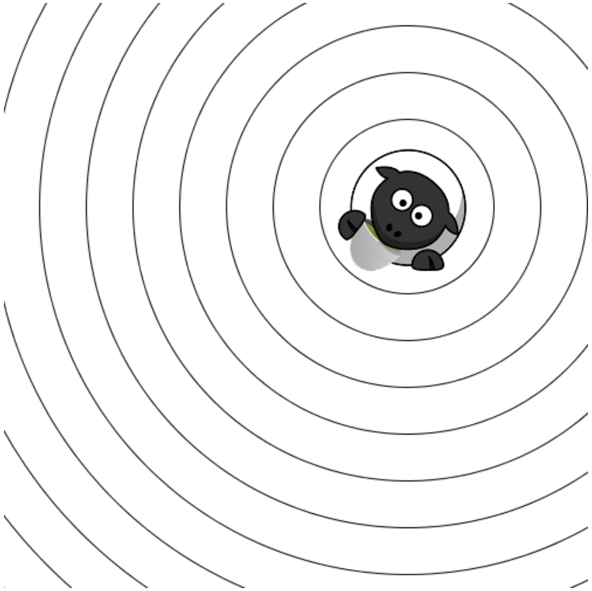
Mouse gets close enough

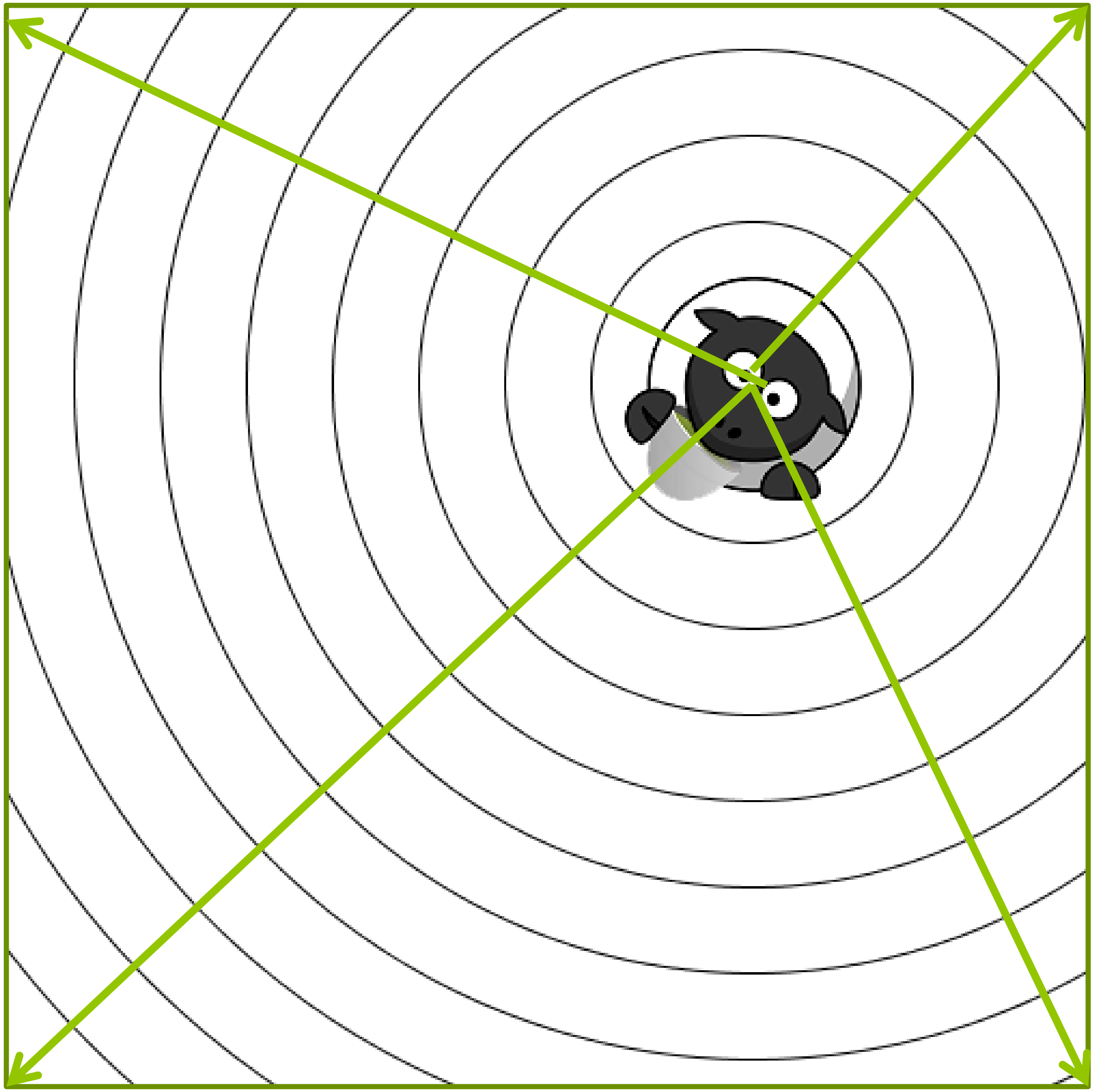
Mouse down

Mouse up

Step 3

Draw one instance of the colored rings. (Simplify: start with plain circles.)

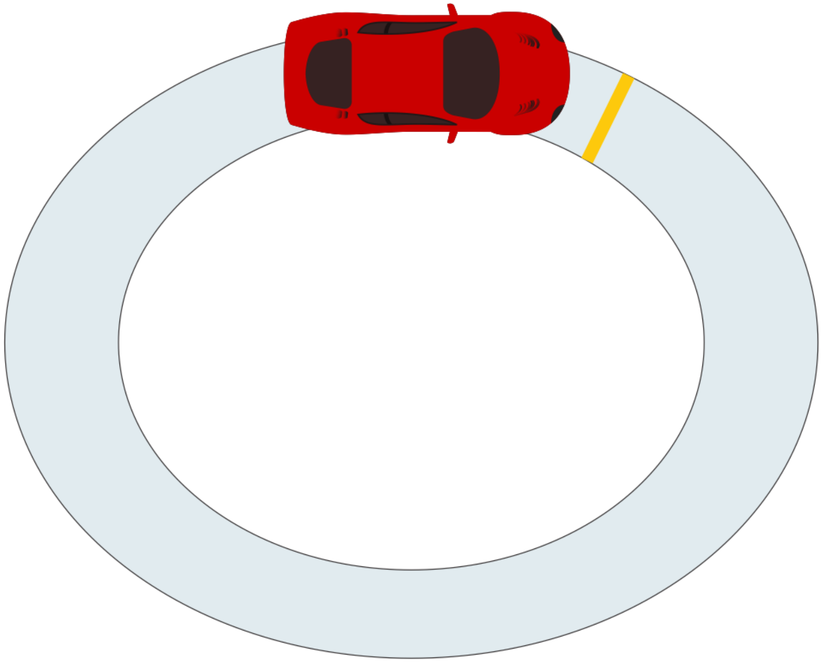




How can we draw an unknown number of circles?

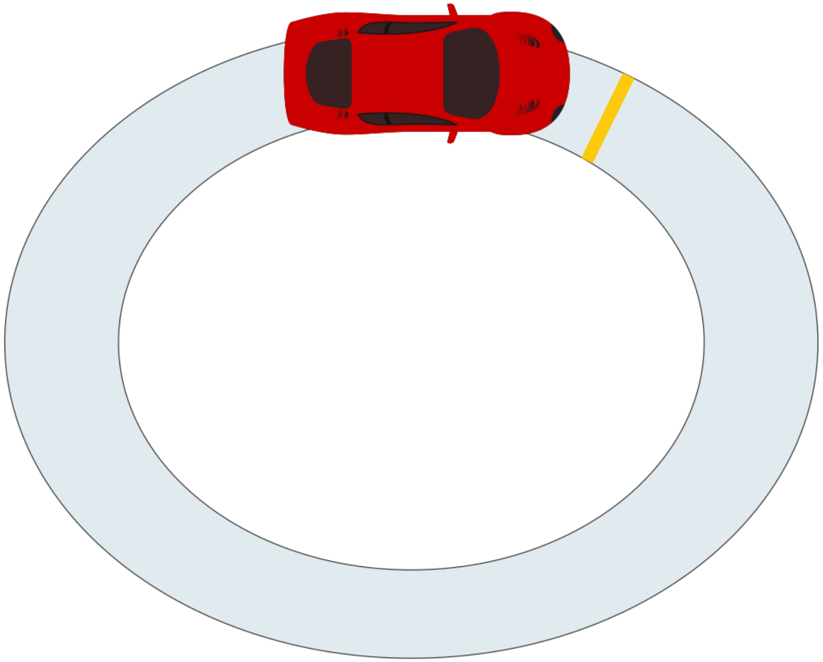
We need a “while loop”!

Loops



Drive the same track multiple times

while loop



Drive the track while the race is not over

```
WHILE you are not dizzy
{
    Spin around quickly
}
Throw up or fall down (or both)
```

```
float radius = 0;

while (radius < maxDistance)
{
  ellipse(x, y, 2*radius, 2*radius);
  radius += radiusChange;
}
```

```
float radius = 0;

while (radius < maxDistance)
{
  ellipse(x, y, 2*radius, 2*radius);
  radius += radiusChange;
}
```

while loop

Boolean expression

```
float radius = 0;

while (radius < maxDistance)
{
  ellipse(x, y, 2*radius, 2*radius);
  radius += radiusChange;
}
```

```
float radius = 0;

while (radius < maxDistance)
{
  ellipse(x, y, 2*radius, 2*radius);
  radius += radiusChange;
}
```

loop body

check this again

```
float radius = 0;

while (radius < maxDistance)
{
  ellipse(x, y, 2*radius, 2*radius);
  radius += radiusChange;
}
```

Exercise

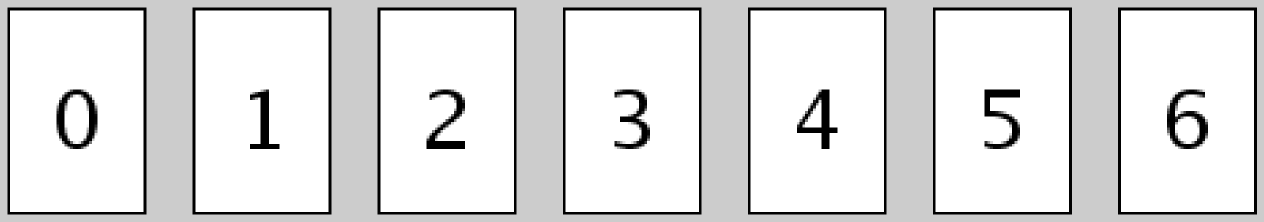
What will the following code output?

```
int x = 6;
while (x > 4)
{
  println(x);
  x = x - 1;
}
```

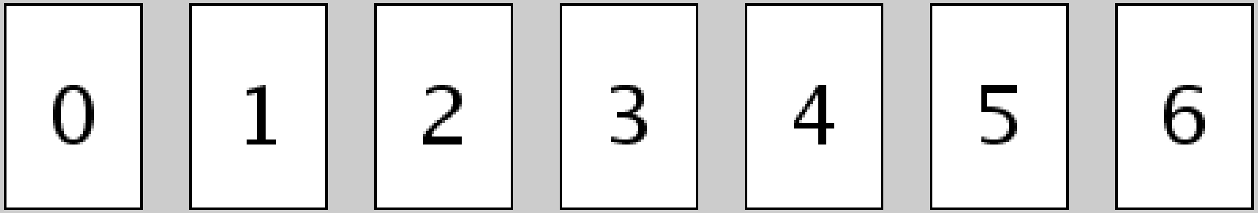
Exercise

What will the following code output?

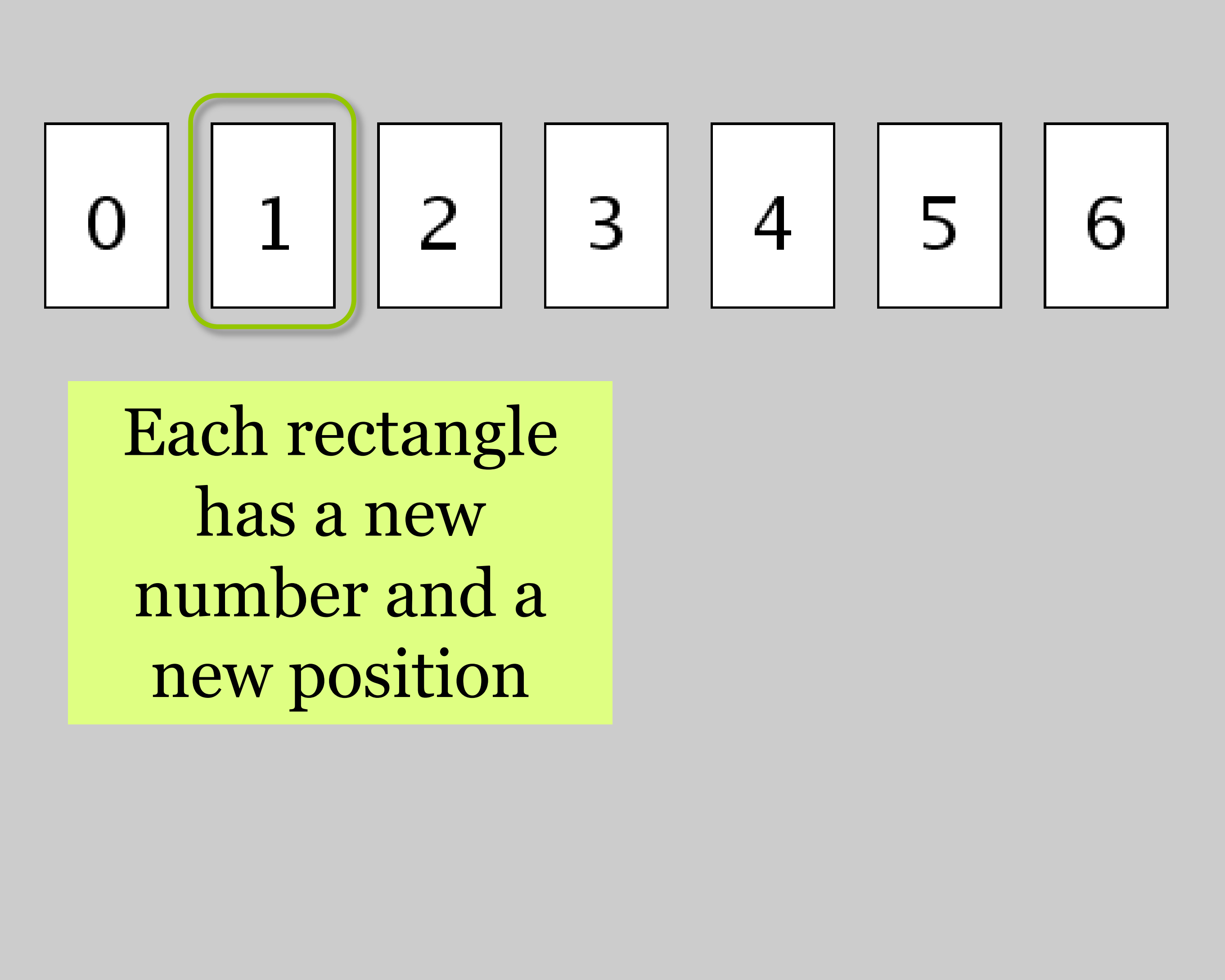
```
int n = 3;
while (n > 0)
{
  if (n == 5)
  {
    n = -99;
  }
  println(n);
  n = n + 1;
}
```



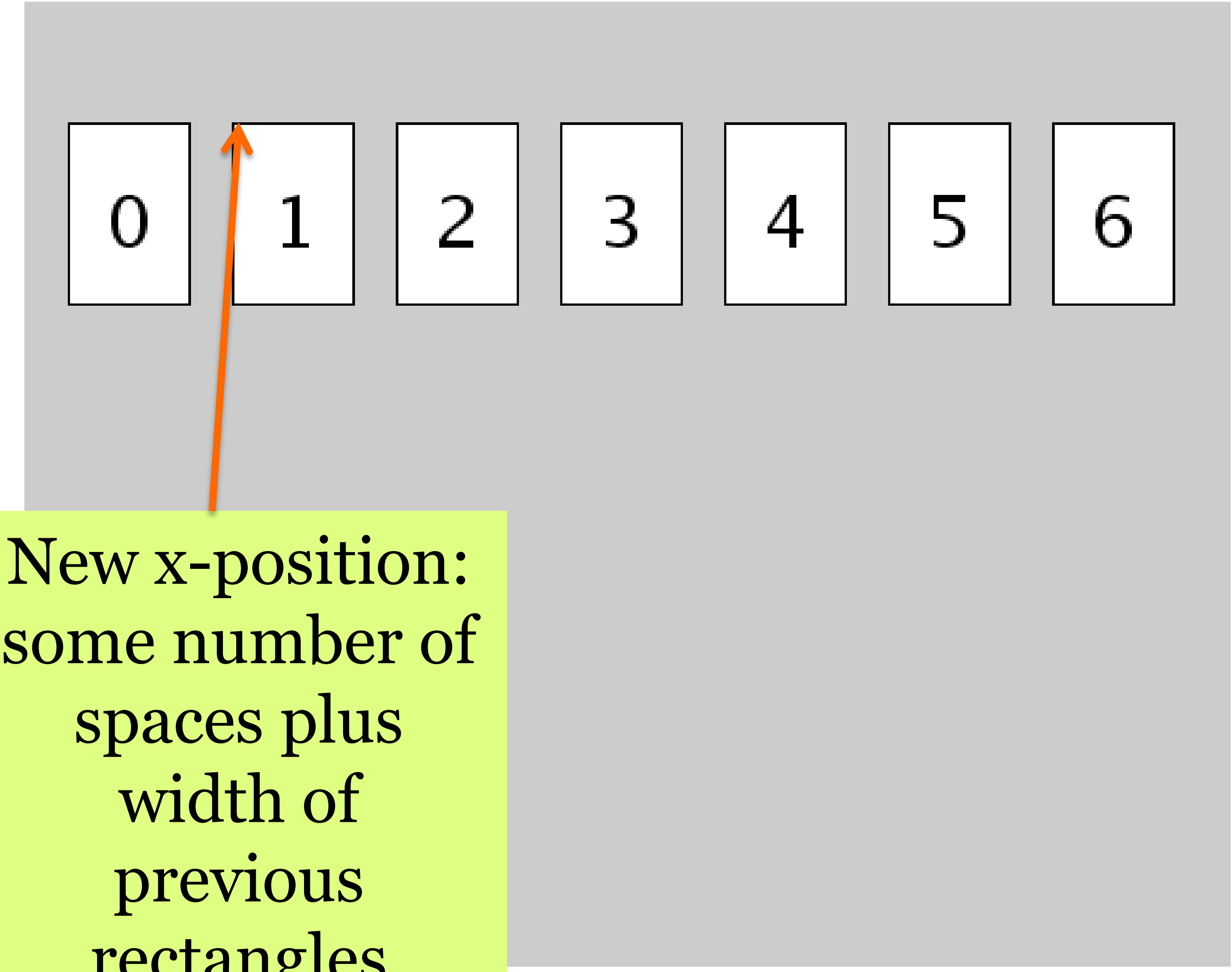
How is this drawn?



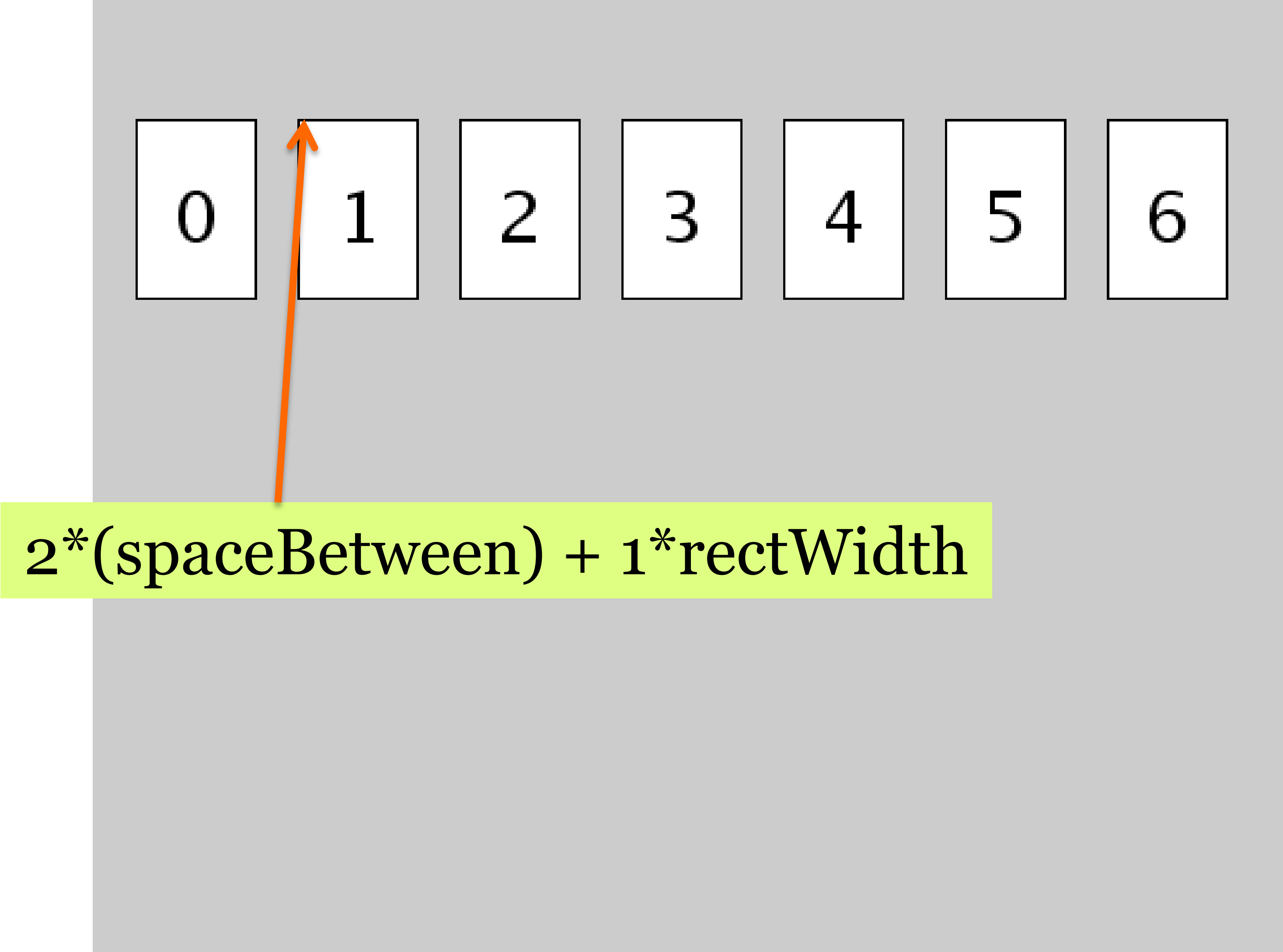
Ask yourself: What value changes over time?



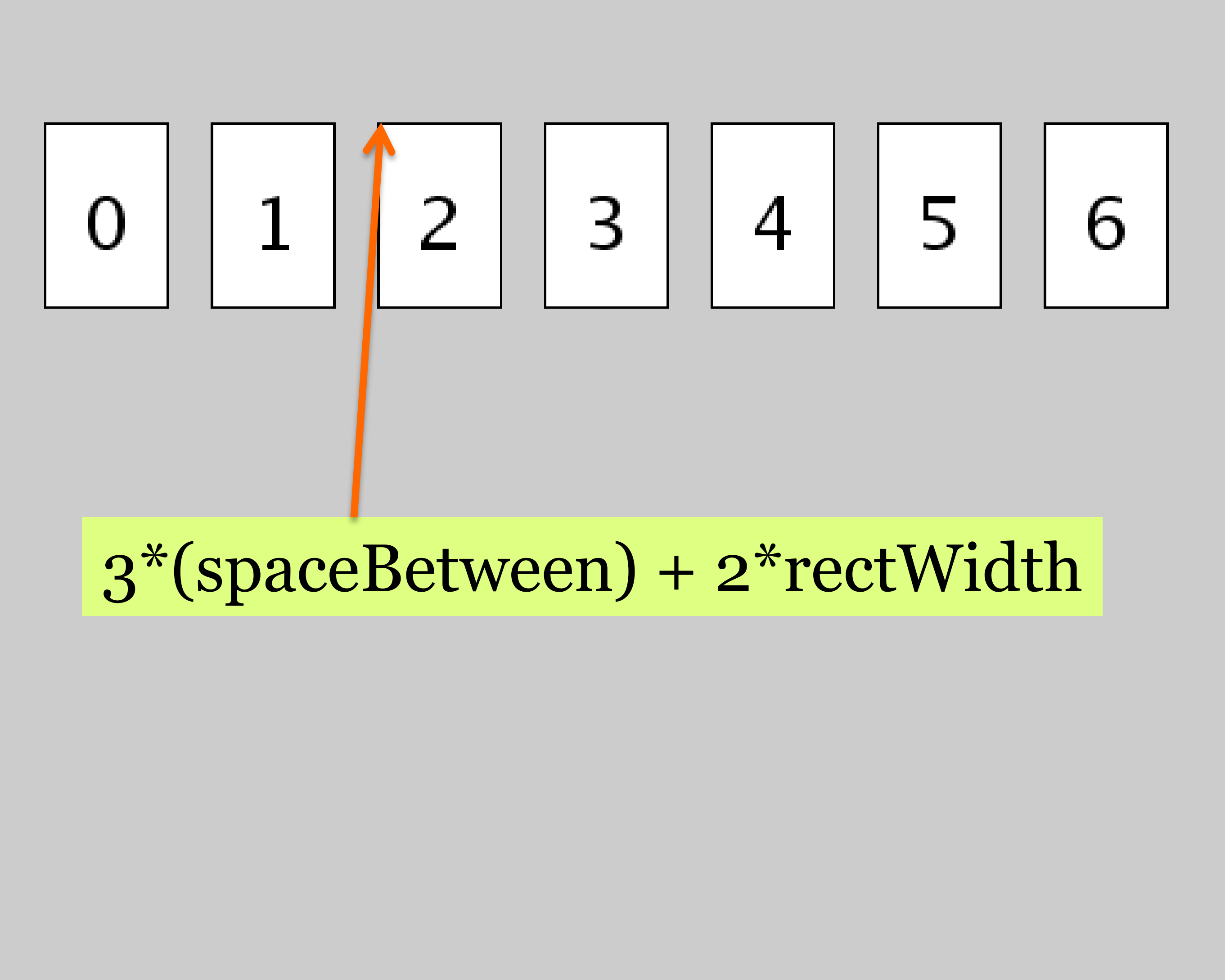
Each rectangle has a new number and a new position



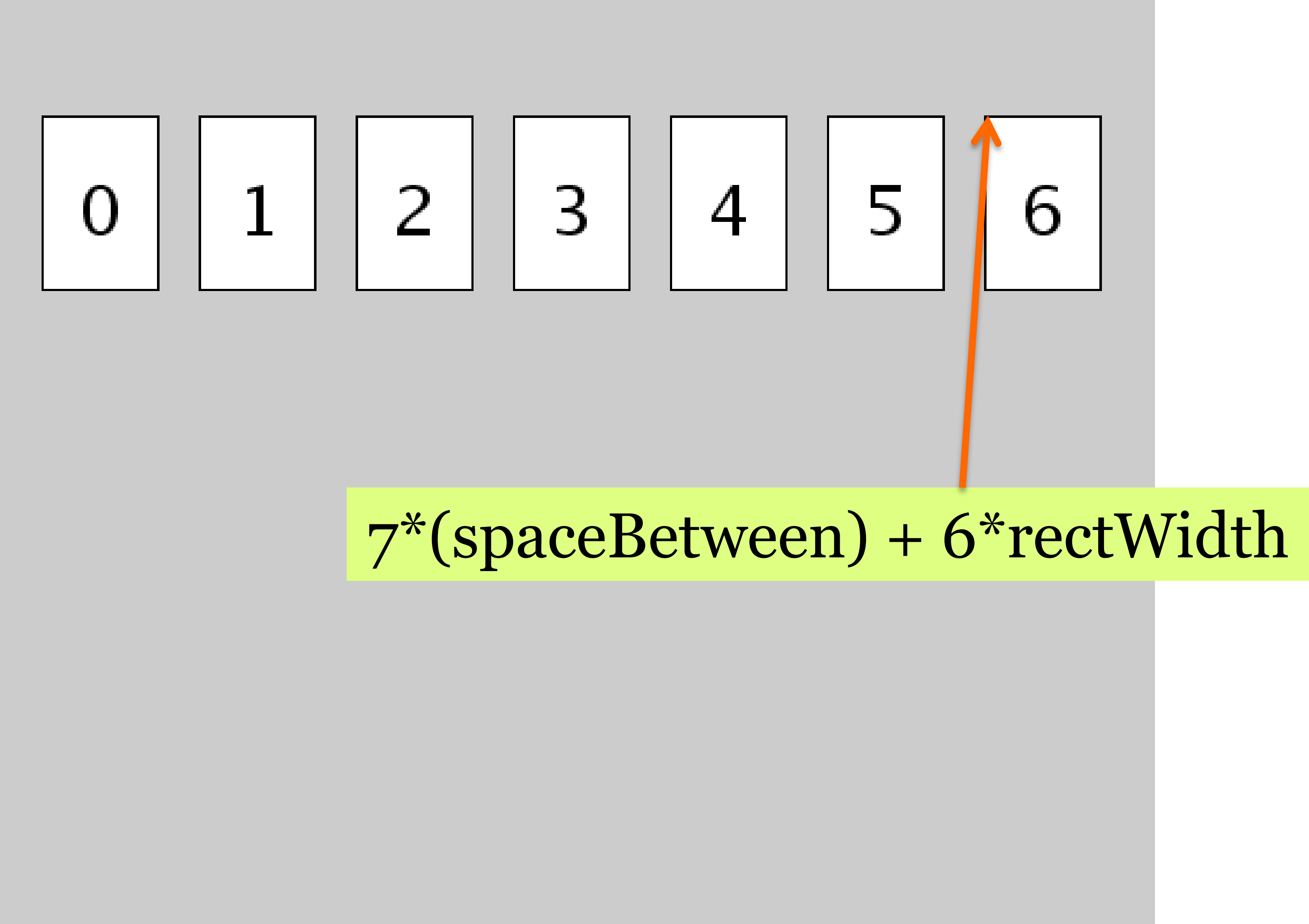
New x-position: some number of spaces plus width of previous rectangles



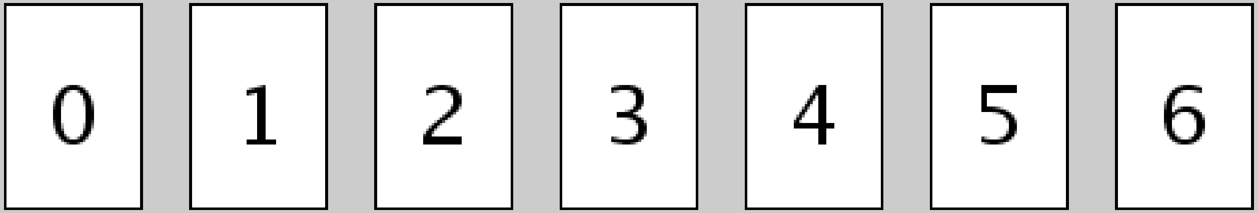
2*(spaceBetween) + 1*rectWidth



3*(spaceBetween) + 2*rectWidth



7*(spaceBetween) + 6*rectWidth



(n+1)*(spaceBetween) +
(n)*(rectWidth)

(n starts at zero)

```
int rectNum = 0;
while (rectNum < numRectangles)
{
  int rectX = spaceBetween*(rectNum+1) + rectWidth*rectNum;
  int rectY = 50;

  fill(255);
  rect(rectX, 50, rectWidth, rectHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

Change rectNum ('n') over time since it's used to compute the rest

```
int rectNum = 0;
while (rectNum < nu
{
  int rectX = spaceBetween*(rectNum+1) + rectWidth*rectNum;
  int rectY = 50;

  fill(255);
  rect(rectX, 50, rectWidth, rectHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

Use rectNum to compute x-position

```
int rectNum = 0;
while (rectNum < numRectangles)
{
  int rectX = spaceBetween*(rectNum+1) + rectWidth*rectNum;
  int rectY = 50;

  fill(255);
  rect(rectX, 50, rectWidth, rectHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

```
int rectNum = 0;
while (rectNum < numRectangles)
{
  int rectX = spaceBetween*(rectNum+1) + rectWidth*rectNum;
  int rectY

  fill(255)
  rect(rect                       tHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

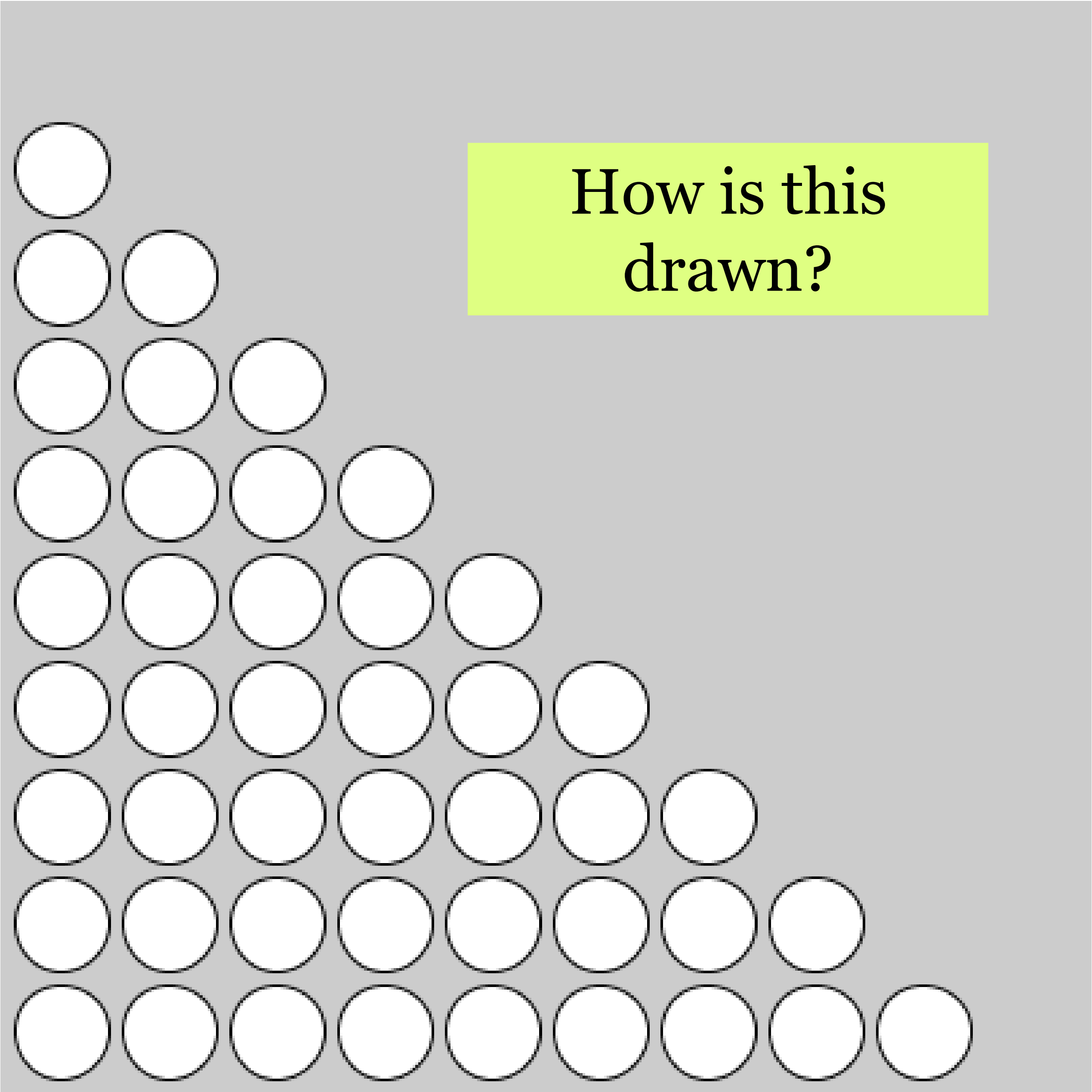
Also use rectNum to draw number

```
int rectNum = 0;
while (rectNum < numRectangles)
{
  int rectX = spaceBetween*(rectNum+1) + rectWidth*rectNum;
  int rectY = 50;

  fill(255);
  rect(rectX, 50, rectWidth, rectHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

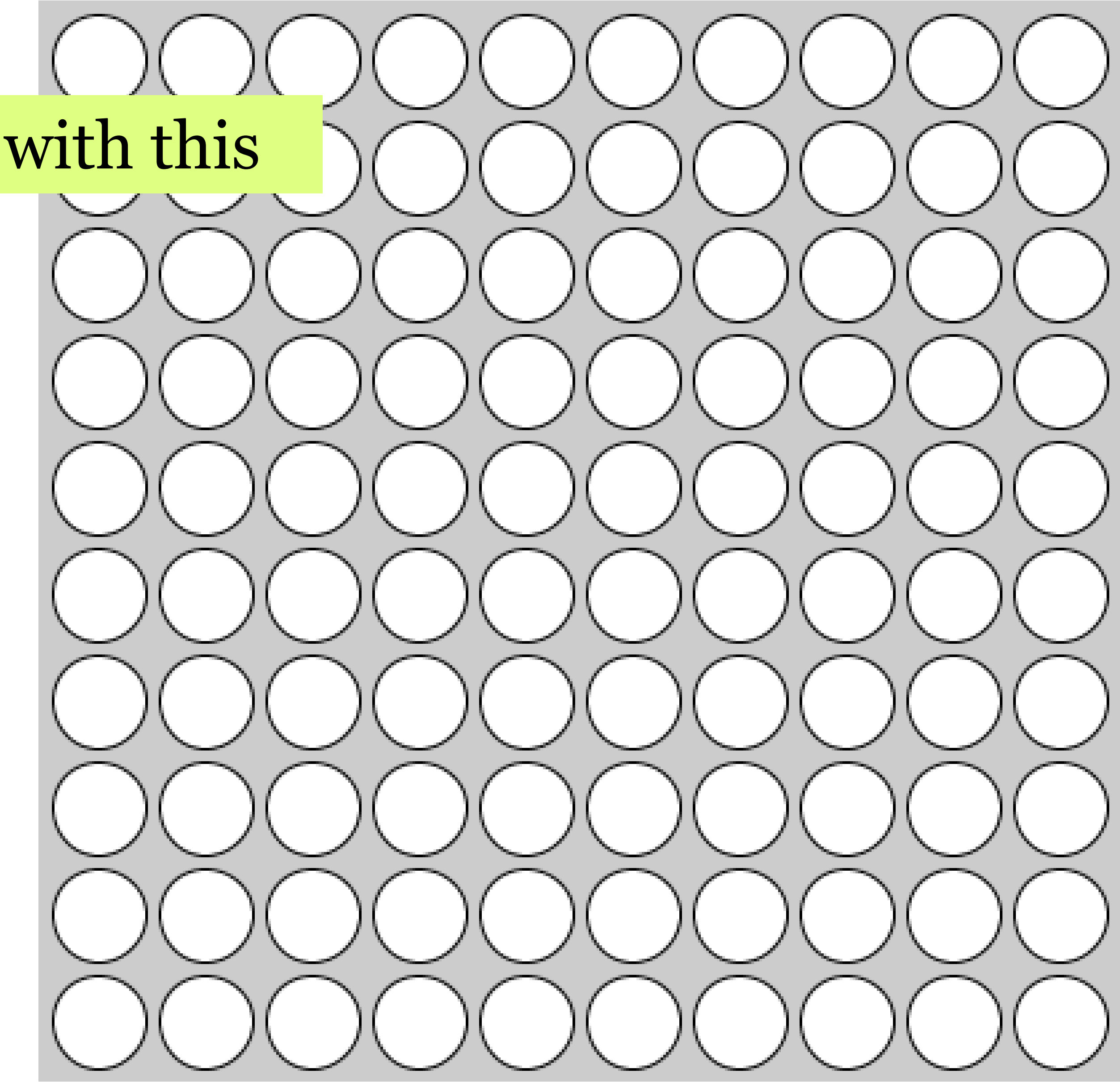
Increase rectNum

```
int rectNum = 0;
while (rectNum < numRectangles)
{
  int rectX = spaceBetween*(rectN                    tNum;
  int rectY = 50;

  fill(255);
  rect(rectX, 50, rectWidth, rectHeight);

  fill(0);
  text(rectNum, rectX + rectWidth/2, rectY + rectHeight/2);

  rectNum++;
}
```

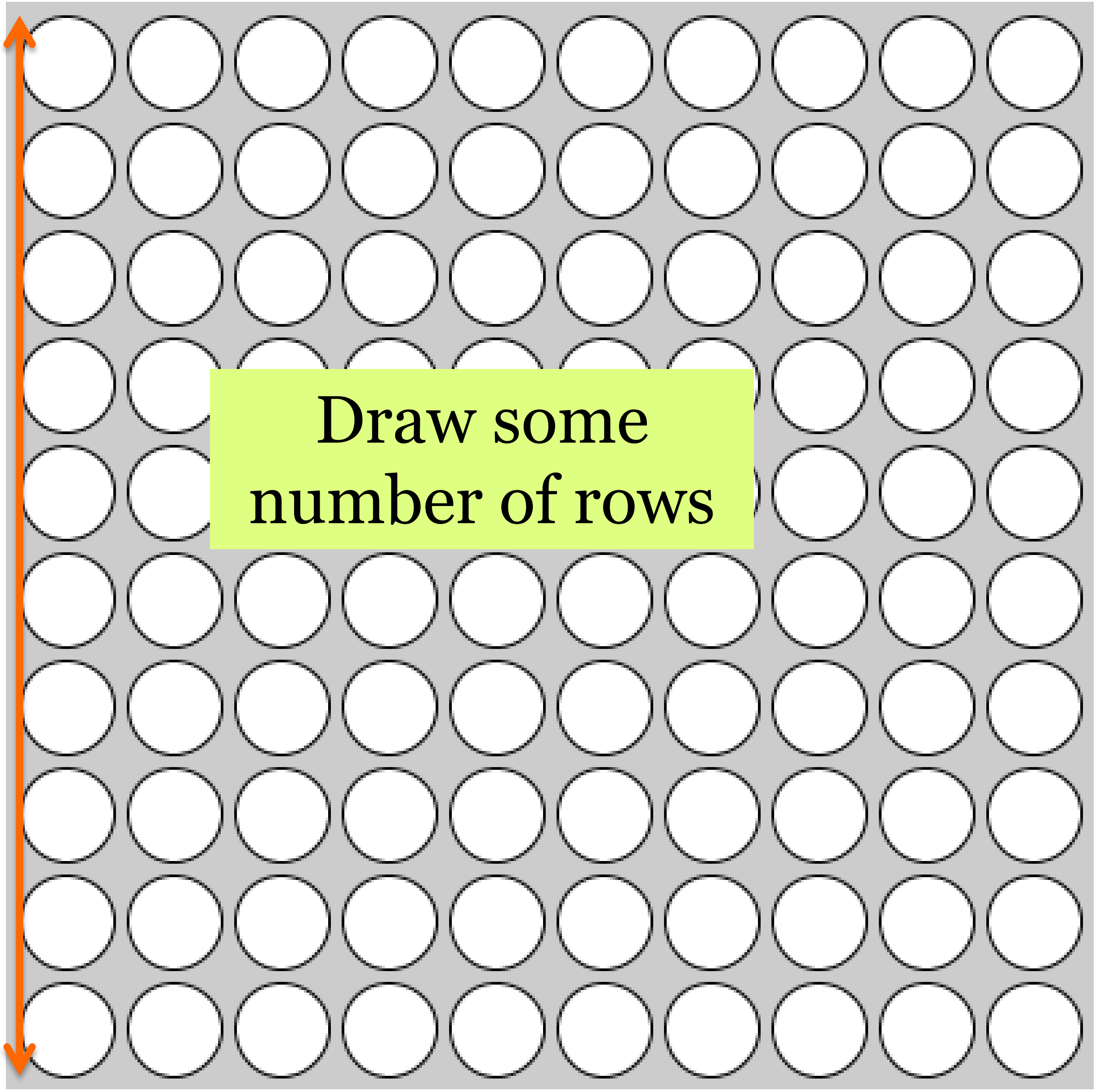
Repeat until rectNum becomes 7 (we don't draw a rectangle with 7 on it)



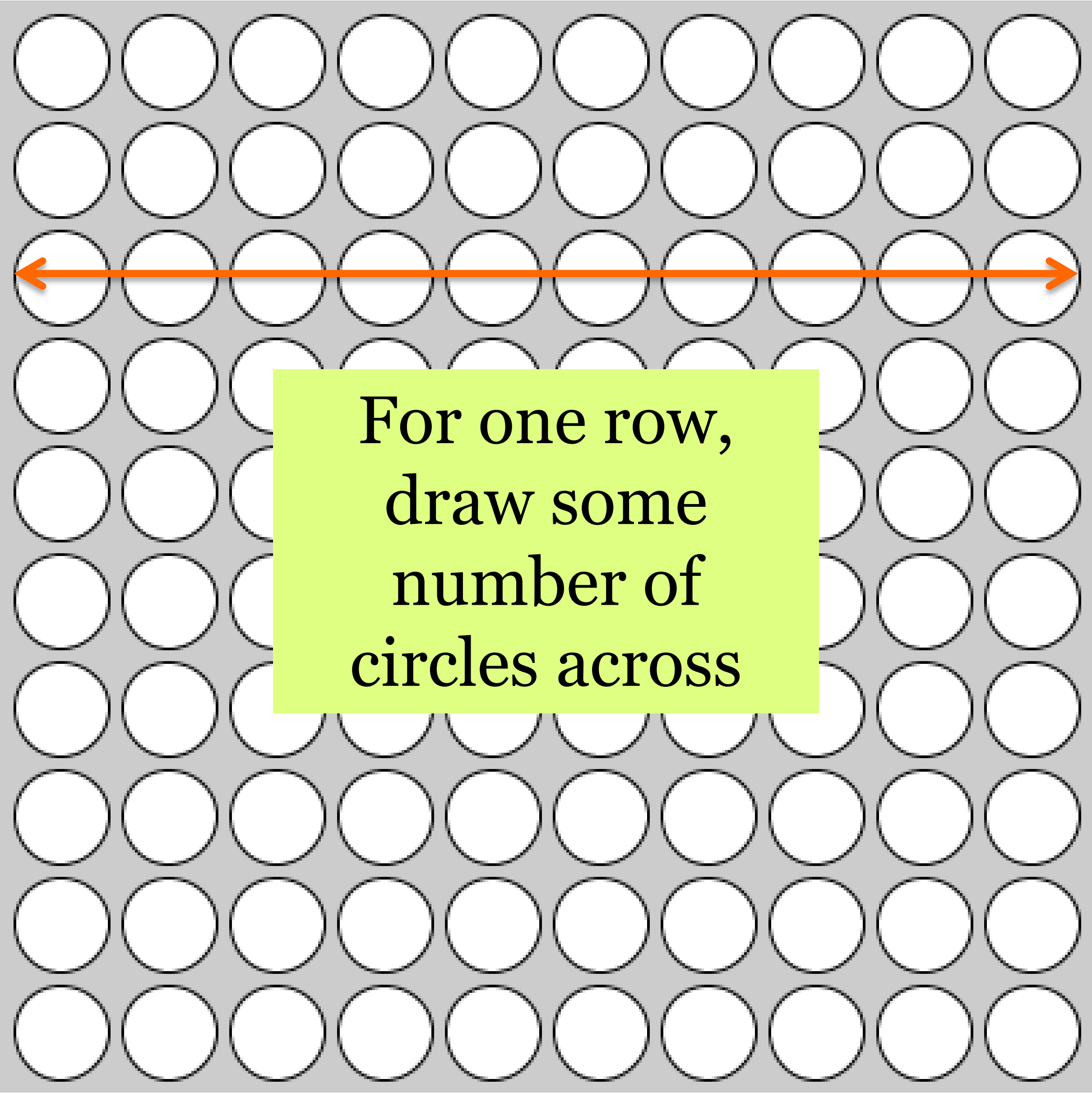
How is this drawn?



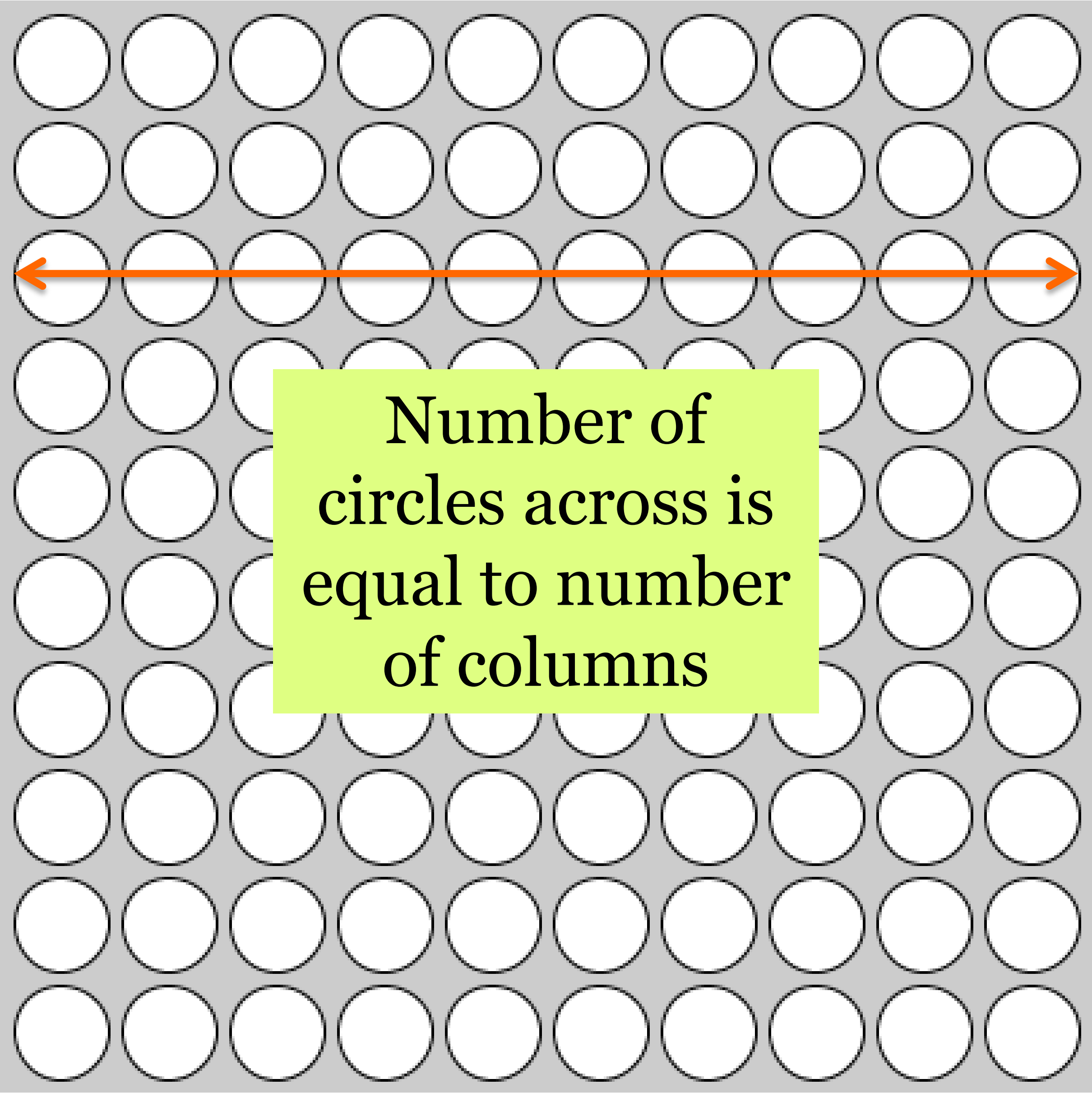
Start with this



Draw some number of rows



For one row, draw some number of circles across



Number of circles across is equal to number of columns

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

    ellipse(ellipseX, ellipseY, diameter, diameter);

    circleColNum++;
  }

  circleRowNum++;
}
```

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

    ellipse(ellipseX, ellipseY, diameter, diameter);

    circleColNum++;
  }

  circleRowNum++;
}
```

Change which row we are drawing until we have drawn enough rows

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetw                     diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetw                     diameter;

    ellipse(ellipseX, ellipseY,

    circleColNum++;
  }

  circleRowNum++;
}
```

For each row, start at the first column, and draw one circle for each column

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*                                  eter;

    ellipse(ellipseX, e

    circleColNum++;
  }

  circleRowNum++;
}
```

x-position similar to numbered boxes example

(uses column number rather than rectNum, uses CORNER mode for ellipses)

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

    ellipse(ellipseX, ellipseY, diameter, diameter);

    circleColNum++;
  }

  circleRowNum++;
}
```

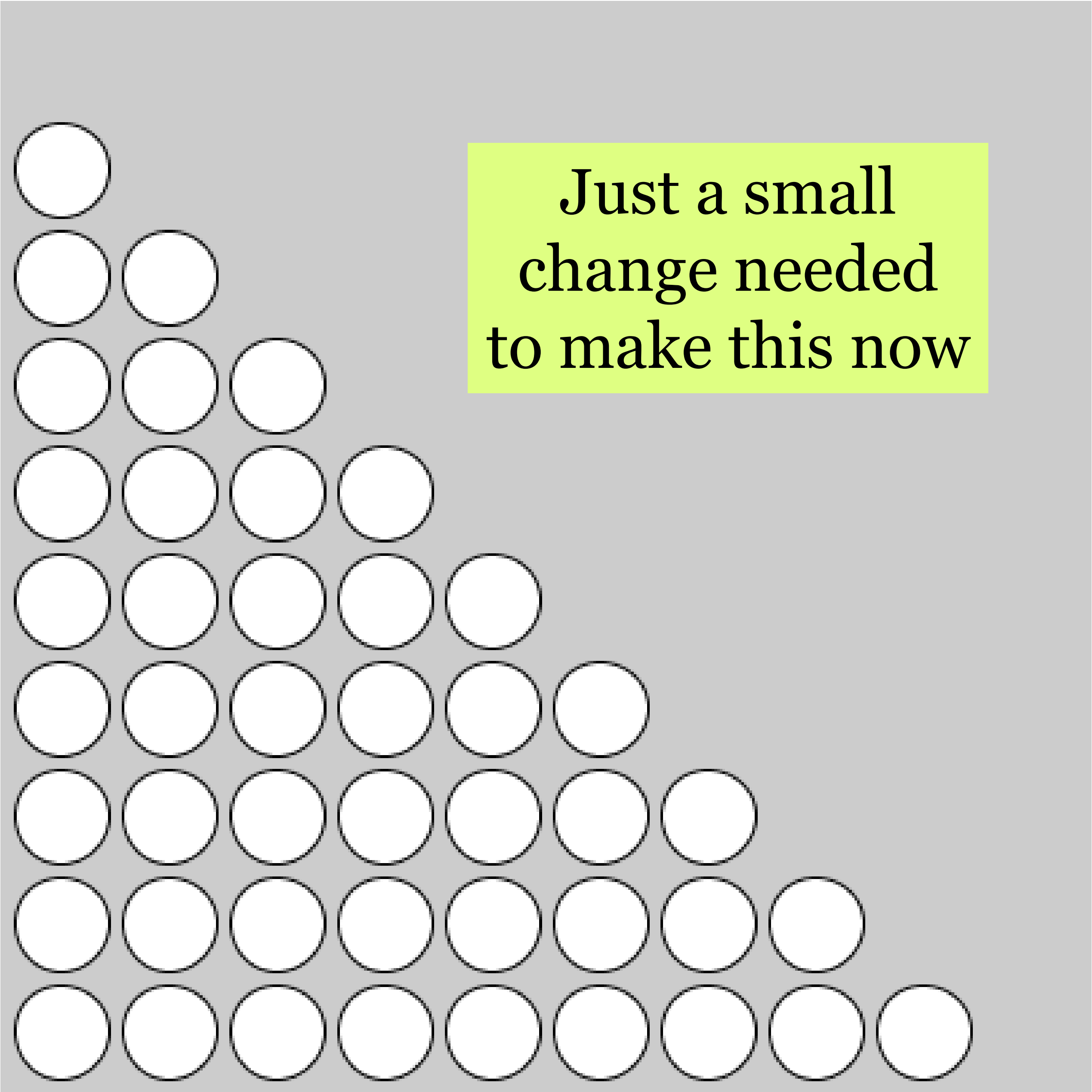
Same idea for y-position, except vertical; use row number

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

    ellipse(ellipseX, ellipseY, diameter, diameter);

    circleColNum++;
  }

  circleRowNum++;
}
```

Increase the column number inside its loop

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < numCols)
  {
    int ellipseX =
      (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

    int ellipseY =
      (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

    ellipse(ellipseX, ellipseY, diameter, diameter);

    circleColNum++;
  }
  circleRowNum++;
}
```

Increase the row number only after all the column circles are done



Just a small change needed to make this now

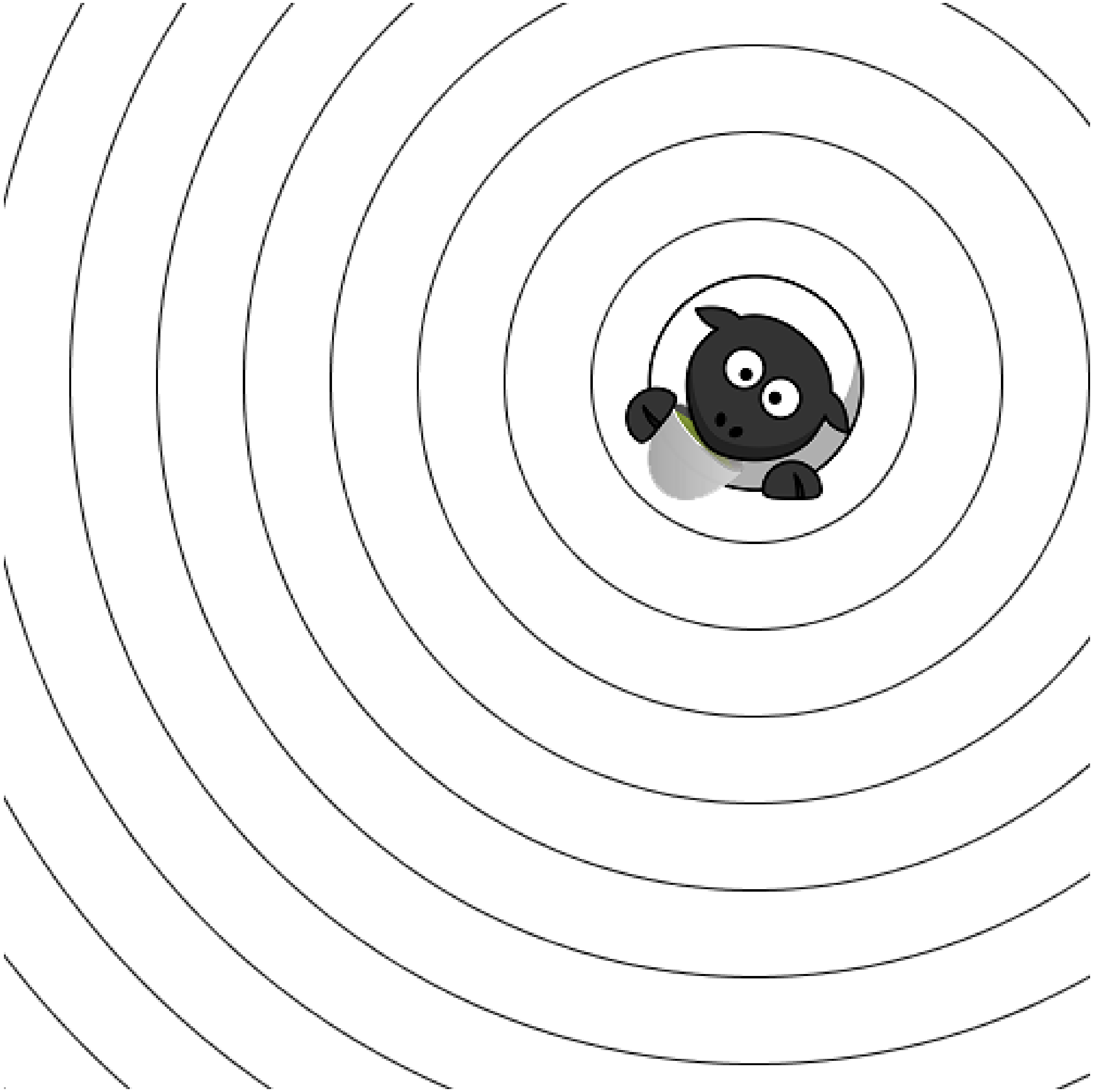
```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
 int circleColNum = 0;
 while (circleColNum < circleRowNum)
 {
  int ellipseX =
   (circleColNum+1)*spaceBetween + (circleColNum)*diameter;

  int ellipseY =
   (circleRowNum+1)*spaceBetween + (circleRowNum)*diameter;

  ellipse(ellipseX, ellipseY, diameter, diameter);

  circleColNum++;
 }

 circleRowNum++;
}
```

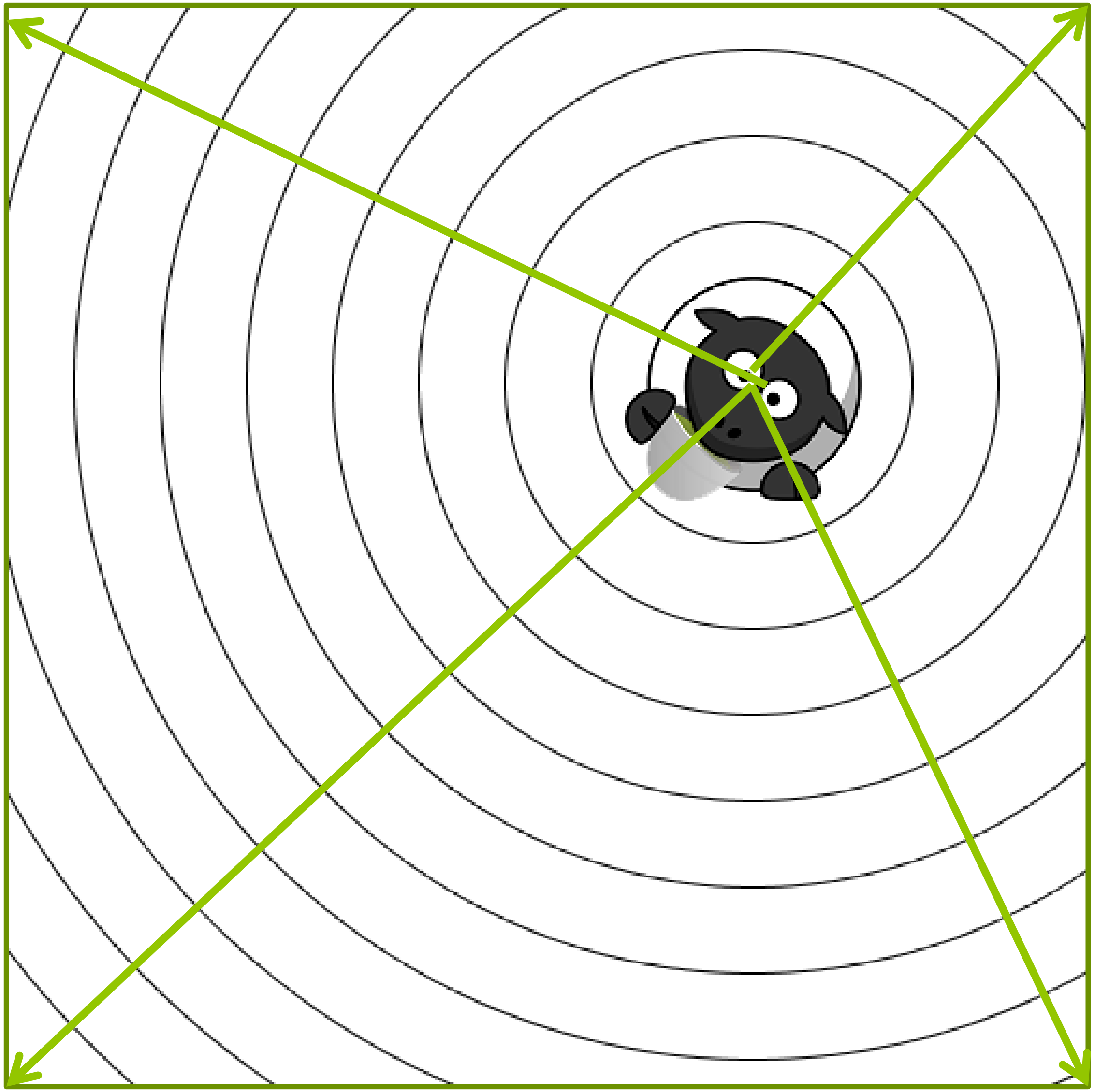
```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
   int circleColNum = 0;
   while (circleColNum < circleRowNum)
   {
      int ellipseX =
         (circleColN                                 Num)*diameter;

      int ellipseY
         (circleRowN                                 Num)*diameter;

      ellipse(ellipseX, ellipseY, diameter, diameter);

      circleColNum++;
   }

   circleRowNum++;
}
```

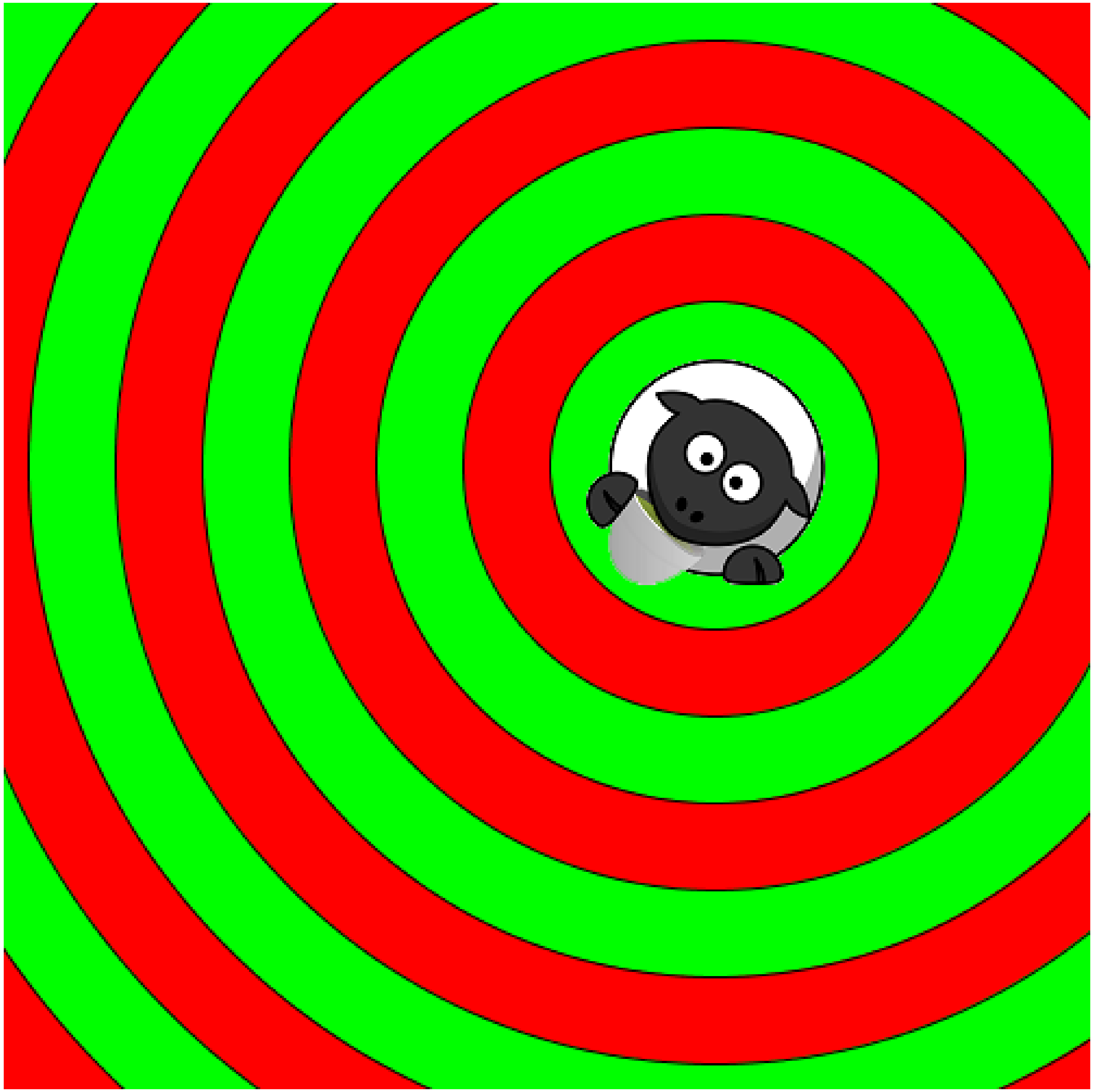
Draw one less column circle as there are rows so far

```
int circleRowNum = 0;
while (circleRowNum < numRows)
{
  int circleColNum = 0;
  while (circleColNum < circleRowNum)
  {
    int ellipseX =
      (circleColN                                      eter;

    int ellipseY
      (circleRowN                                      eter;

    ellipse(ellip

    circleColNum-
  }

  circleRowNum++;
}
```

circleRowNum	circleColNum
0	<none>
1	0
2	0, 1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4







```
float radius = maxDistance;
boolean fillRed = true;
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Start with the biggest circle so we don't draw on top of smaller ones

```
float radius = maxDistance;
boolean fillRed = tru
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Stop when the circles get too small

```
float radius = maxDistance;
boolean fillRed = true;
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2
  radius -= radiusChange;
}
```

Decrease the radius each time rather than increase it

```
float radius = maxDistanc
boolean fillRed = true;
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Two states to track: red and not red

```
float radius = maxDistance;
boolean fillRed = true;
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Once red is drawn,
flip to not red

```
float radius = maxDistance;
boolean fillRed = true;
while (radius > 0)
{
  if (fillRed)
  {
    fill(255,0,0);
    fillRed = false;
  }
  else
  {
    fill(0,255,0);
    fillRed = true;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Once ‘not red’ (i.e. green) is drawn, flip back to red

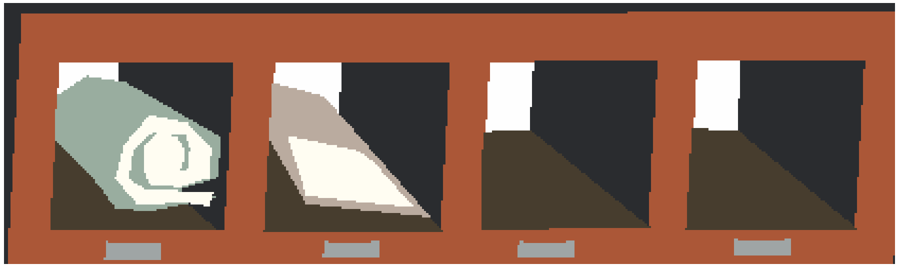
How do we get lots of colors?

arrayOfColors

```
color[] arrayOfColors = new color[4];
```

Declare and initialize an array that holds four things

`arrayOfColors`



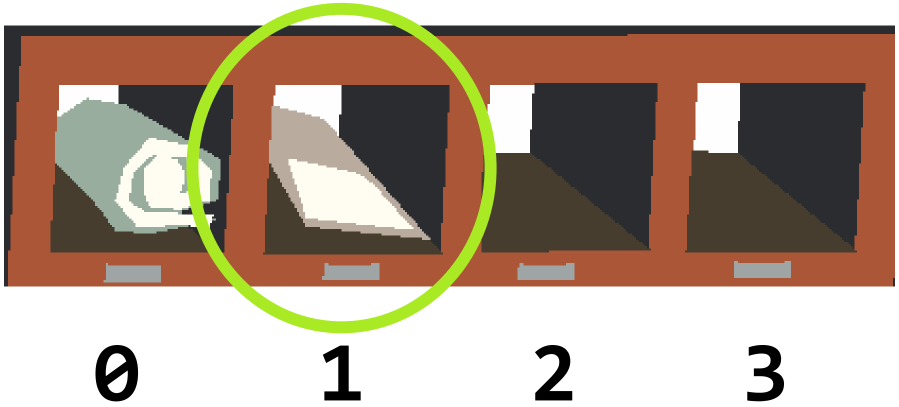
0 1 2 3

Assign to the slot at index 0

```
arrayOfColors[0] = color(0, 45, 200);
arrayOfColors[1] = color(24, 45, 10);
```

Assign values to the slots of the array

arrayOfColors



0 1 2 3

```
color secondColor = arrayOfColors[1];
```

Access the value in one slot of the array

```
final color[] listOfColors =
{
  color(227, 41, 41),    // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),    // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

Shortcut: initialize array with values already in it

Exercise

What is the output of the following code?

```
int[] myNumbers = {1, 2, 3};
myNumbers[0] = myNumbers[1];
myNumbers[1] = myNumbers[0];
println(myNumbers[1]);
```

Options:

0

1

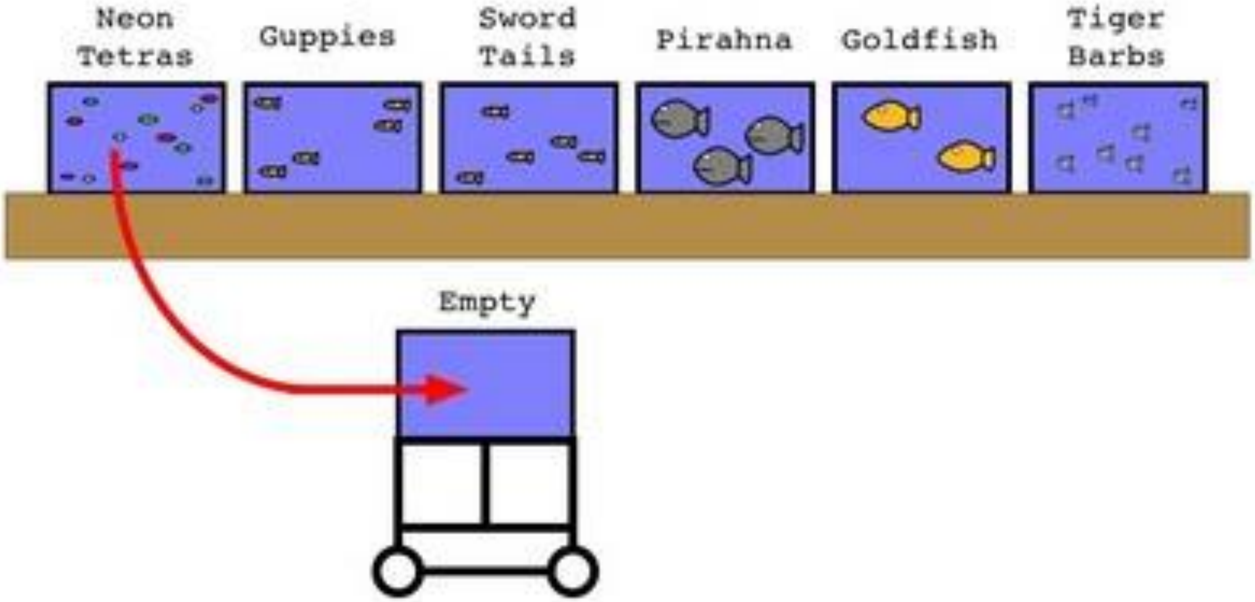
2

3

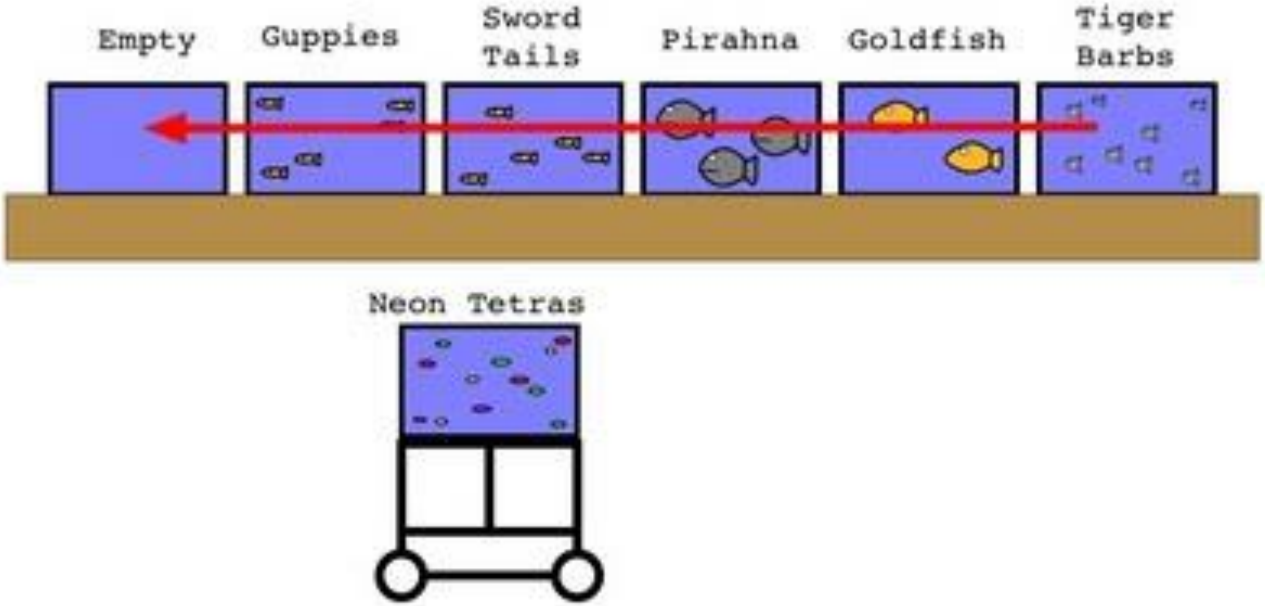
Swapping Values in an Array

http://computationaltales.blogspot.ca/2011/06/swapping-array-values-and-swimmy.html

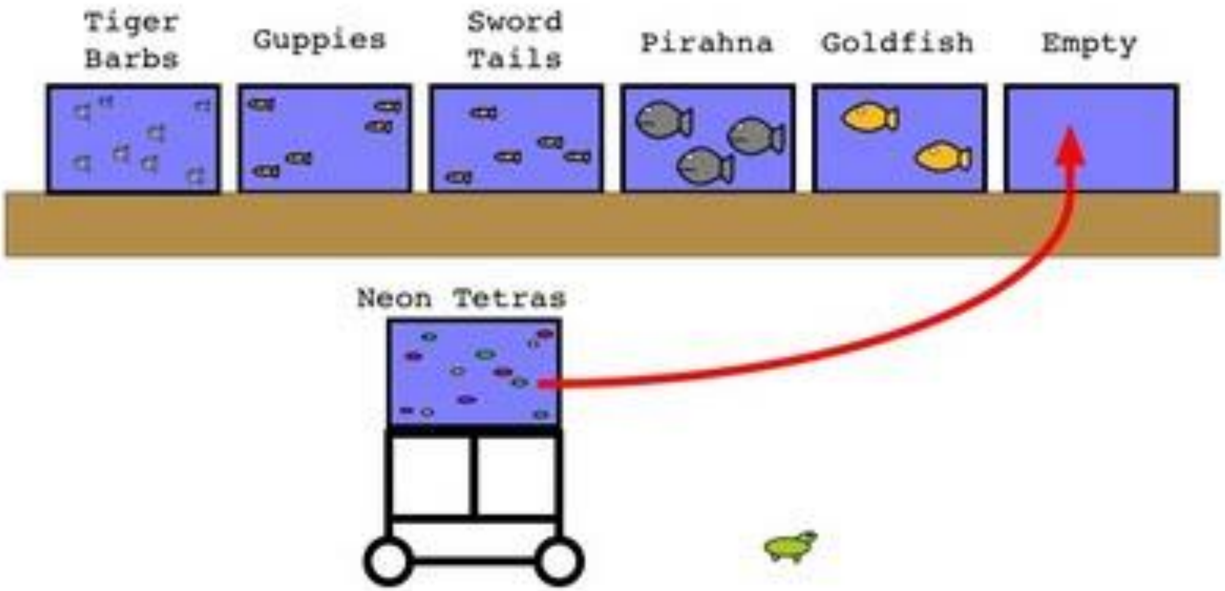
Swapping Values in an Array



Swapping Values in an Array



Swapping Values in an Array



Use an array to store all the colors.

```
final color[] listOfColors =
{
  color(227, 41, 41),   // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),   // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

```
final color[] listOfColors =
{
  color(227, 41, 41),    // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),    // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

listOfColors[0] -> red

```
final color[] listOfColors =
{
  color(227, 41, 41),   // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),   // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

```
listOfColors[6] -> blue
```

```
final color[] listOfColors =
{
  color(227, 41, 41),   // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),   // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

```
listOfColors[listOfColors.length-1] -> blue
```

```
final color[] listOfColors =
{
  color(227, 41, 41),   // red
  color(214, 122, 224), // purple
  color(115, 111, 234), // blue
  color(83, 216, 97),   // green
  color(252, 255, 95),  // yellow
  color(211, 133, 15),  // brown
  color(175, 202, 216), // blue
};
```

listOfColors[7] -> error!

```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorIndex]);
  colorIndex = (colorIndex + 1);
  if (colorIndex >= colors.length)
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorInde
  colorIndex = (colorIndex + 1);
  if (colorIndex >= colors.length)
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Radius is still the main value that is changing

```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorInd
  colorIndex = (colorI
  if (colorIndex >= co
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

Now also keep track of which color from the array to use for each circle

```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorIn
  colorIndex = (color
  if (colorIndex >= c
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

startIndex is given as a parameter to the function so it can change easily

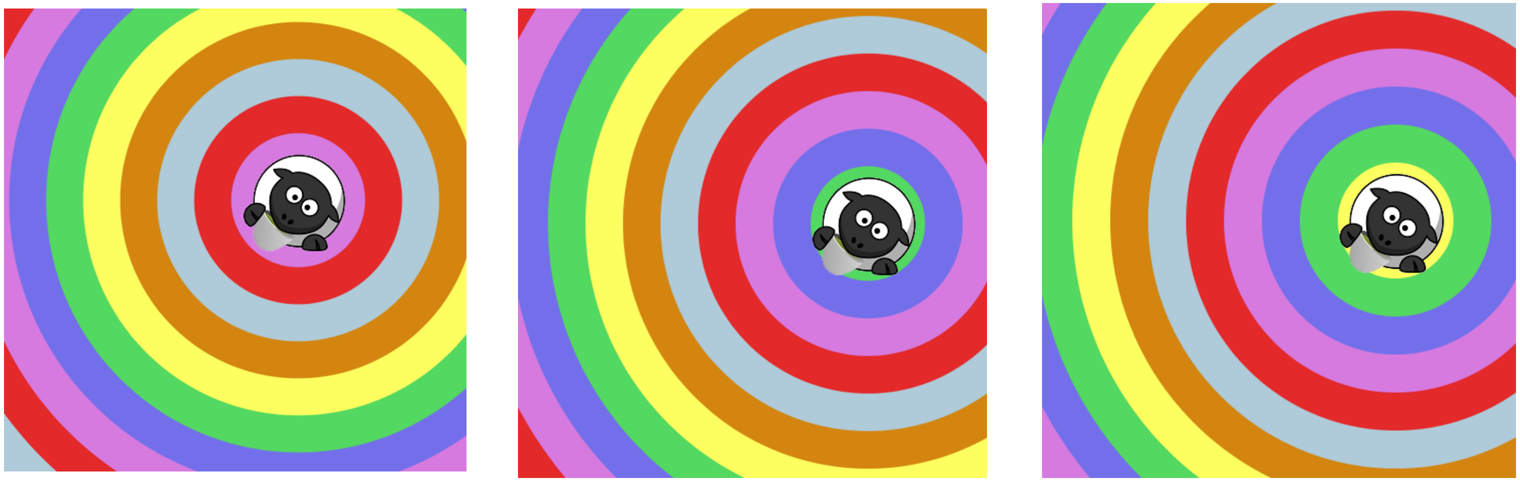
Set the fill of the next circle using the current color

```
float radius = max(corne
int colorIndex = startI
while (radius > 0)
{
  fill(colors[colorIndex]);
  colorIndex = (colorIndex + 1);
  if (colorIndex >= colors.length)
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

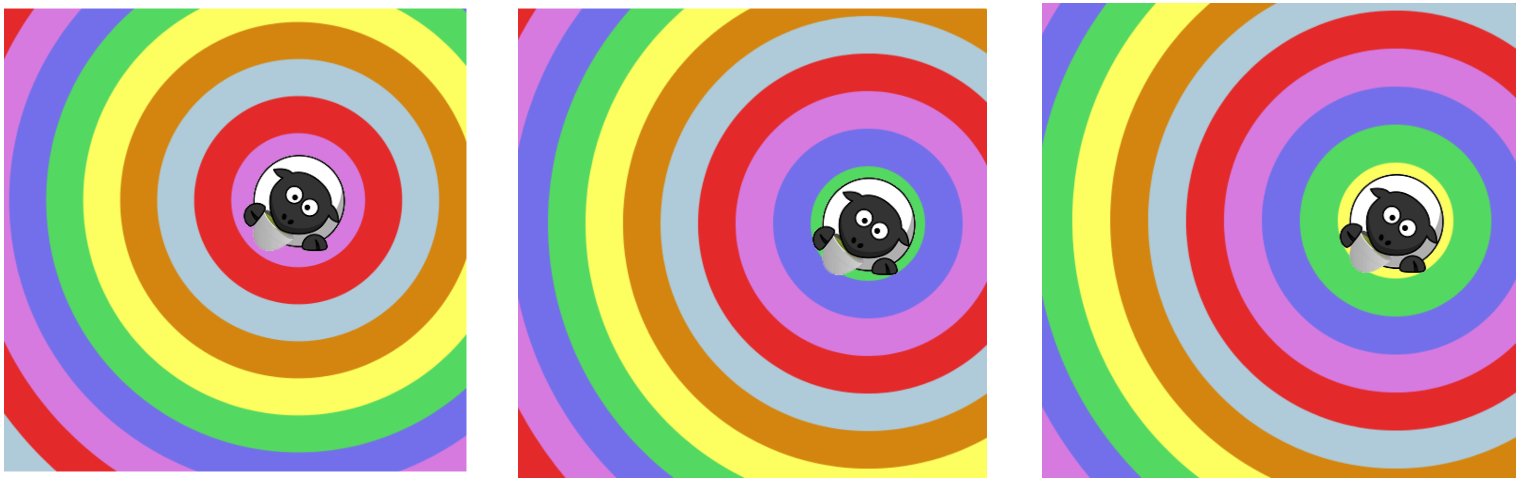
```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorIndex]);
  colorIndex = (colorIndex + 1);
  if (colorIndex >= colors.length)
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius
  radius -= radiusChange
}
```

Move to the next color, going back to 0 if the index gets too big

How can we animate the colors so they move outward?



Every 10 frames, start drawing the rings with a different color



Use `colorStartIndex` to track what color to start on when drawing rings

Use `numFramesToShiftColor` to tell us how often to switch to a new start color

Modulo Operator

What is the remainder
when number1 is evenly
divided by number2?

number1 % number2
=
remainder of number1 / number 2

Modulo Operator

$10 \% 2 = 0$

$10 \% 3 = 1$

$10 \% 4 = 2$

$10 \% 5 = 0$

$10 \% 6 = 4$

$1 \% 4 = 1$

$2 \% 4 = 2$

$3 \% 4 = 3$

Modulo Operator

Can be used to check for
every “nth” of something

```
if (frameCount % numFramesToShiftColor == 0)
{
  // move to the next colorStartIndex
}
```

Modulo Operator

Can be used to check for every “nth” of something

```
if (frameCount % numFramesToShiftColor == 0)
{
  // move to the next colorStartIndex
}
```

If numFramesToShiftColor is ten, then any time frameCount is a multiple of 10, this modulo will be 0

Operator Precedence

Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Exercise

What is the output of the following code?

```
int y = 10;
y -= 5 + y;
int x = 4 + y++ % 3 * y;
println(x);
```

Options:

4

8

12

15

…array examples…