IAT 265 Lecture 4

Mouse interaction & Collision detection

Topics

- Array
- Mouse interactions
- Polygons
- Nested loops
- Collision detection
- Case study: Collision among Beatles

Arrays

- An **array** is a contiguous collection of data items of one type
  - An array stores a list of values
  - Values are accessed by index numbers

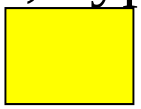
Review: creating a variable of int

Code

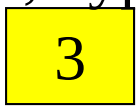
```
// Single int
int anInt;

// Put a value in the int
anInt = 3;

// Type error!
anInt = “hello”;
```

Effect

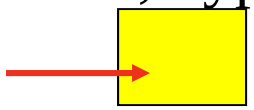
Name: anInt, Type: int



Name: anInt, Type: int



Name: anInt, Type: int

“hello” 

Can’t shove “hello” into an int

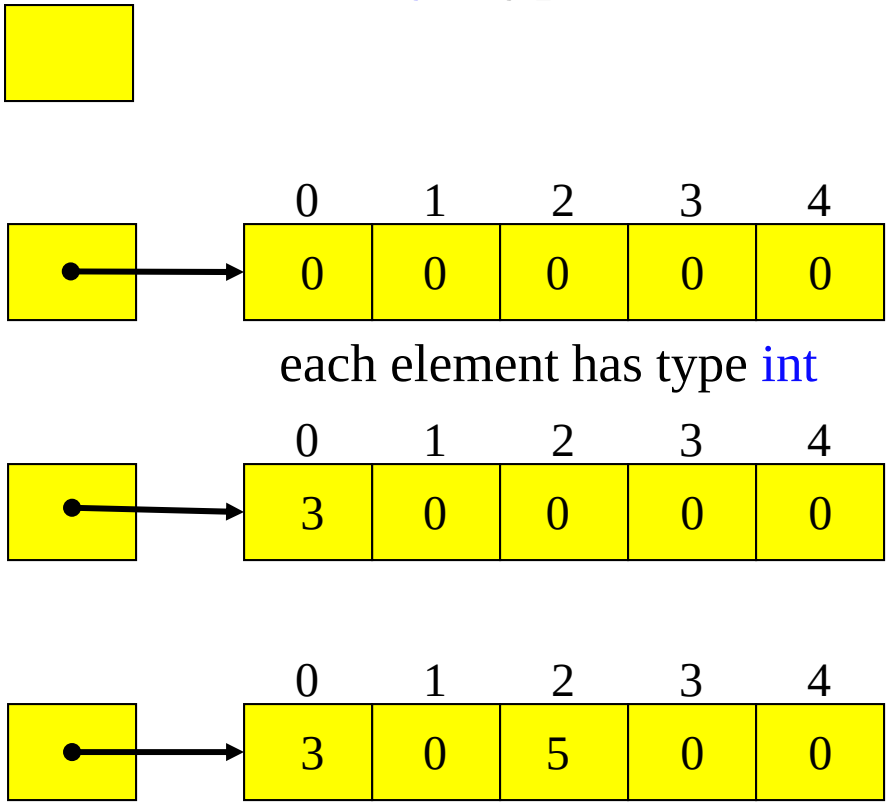
Creating an array of ints

Code

```
// declare int array
int[] intArray;


// allocate memory space
// for the array
intArray = new int[5];


// set first element
intArray[0] = 3;


// set third element
intArray[2] = 5;
```

Effect

Name: intArray, Type: int[]



A Shortcut to Create Arrays

- You can also create an array with a list of initial values (if you know beforehand) as follows:

```
int[] intArray = {3, 2, 5, 4, 1, 6, 9, 8, 7};
```

  - By doing this you save the steps to claim memory space using *new* operator, and set values

- You can get the length of an array by: *arrayName.length*
  - `intArray.length` → **9**

Use loops to process arrays

- Calculate the sum of an array of ints:

```
int[] intArray = {3, 2, 5, 4, 1, 6, 9, 8, 7};
int sum=0;
for(int i = 0; i < intArray.length; i++){
  sum += intArray[i];
}
println("Num counts: "+ intArray.length);
println("sum = " + sum);
```

Mouse variables

- **`mouseX`** and **`mouseY`** – variables that automatically contain the current mouse location
  - **`pmouseX`** and **`pmouseY`** hold the previous location

- **`mousePressed`** – boolean variable that is true if a mouse button is down
  - **`mouseButton`** – value is **`LEFT`**, **`RIGHT`** or **`CENTER`** depending on which button is held down

Example: Mouse variables

```
void draw() {
  line(pmouseX, pmouseY, mouseX, mouseY);
  if(mousePressed) {
   noLoop();
  }
}
```

Mouse callback methods

- There are several built-in methods you can fill in to respond to mouse *events*

```
mousePressed()    mouseReleased()
mouseMoved()      mouseDragged()
```

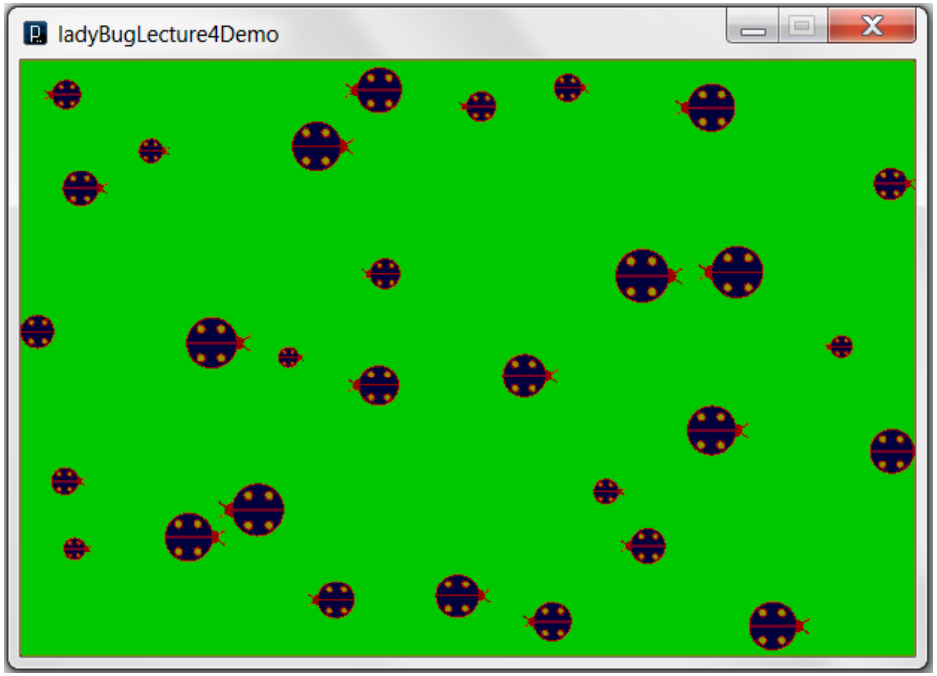
Example:

```
void mousePressed()
{
  if( mouseButton == LEFT ){
    println( “Left Mouse Button was pressed” );
    loop();  // activate drawing again
  }
}
```

Case study: handle multiple bugs

- Create a list of bugs of different sizes that move randomly

- Involve *arrays*, *loops*, *random*, *translate*, *scale*, *pushMatrix* & *popMatrix*



Step 1: Create & initialize arrays for bug positions

```
float[] bugPosesX = new float[30];//array for x coordinates
float[] bugPosesY = new float[30];//array for y coordinates

void setup() {
  size(600, 400, OPENGL);
  smooth();
  //initialize positions  with some random nums
  for(int i=0; i<bugPosesX.length; i++) {
    bugPosesX[i] = random(bugX, gardenW);
    bugPosesY[i] = random(bugY, gardenH);
  }
}
```

About random() function

- **random()**: Generates random numbers within the specified range
- Syntaxes:
  - random(high); // return a **float** between 0 and // the **high** parameter
  - random(low, high); // return a **float** with a value // between the parameters
- Examples
  - random(5) returns values between 0 and 5 (but not including 5)
  - random(-5, 10.2) returns values starting at -5 up to (but not including) 10.2

Step 2: Loop through the array to draw those bugs

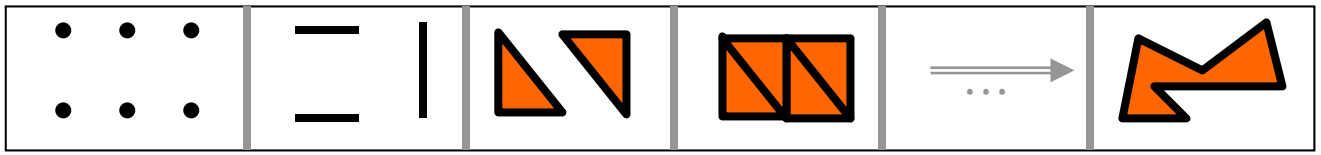
```
void draw() {
//whatever done before
for(int i=0; i<bugPosesX.length; i++) {
    pushMatrix();   //save the current coordinate system
    translate(bugPosesX[i], bugPosesY[i]);
    scale((i+1)*0.4);

    //Reset the bug's position to its starting point when it hits the walls
    if((bugPosesX[i]+bugW+9) > (gardenX+gardenW)) {
      bugPosesX[i]=gardenX+50;
    }
     if((bugPosesY[i]+bugH) > (gardenY+gardenH) ||bugPosesY[i] < gardenY) {
       bugPosesY[i]=gardenY+50;
    }
```

Step 2: Loop through the array to draw those bugs (1)

```
    //change its position with randomized speed
    bugPosesX[i] = bugPosesX[i]+changeX*random(0,2);
    bugPosesY[i] = bugPosesY[i]+changeY*random(0,1);

     //The rest is roughly the same as the base program except that
     //bSize is now replaced by bSize[i]
     ...

      popMatrix();   //restore the coordinate system to previous state
  }
}
```

Building Special Shapes

- The **`beginShape()`** and **`endShape()`** functions allow us to draw irregular shapes from any number of points we define
- **`beginShape(MODE)`** accepts the following arguments for MODE:
  - `POINTS, LINES, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUADS, QUAD_STRIP, POLYGON`



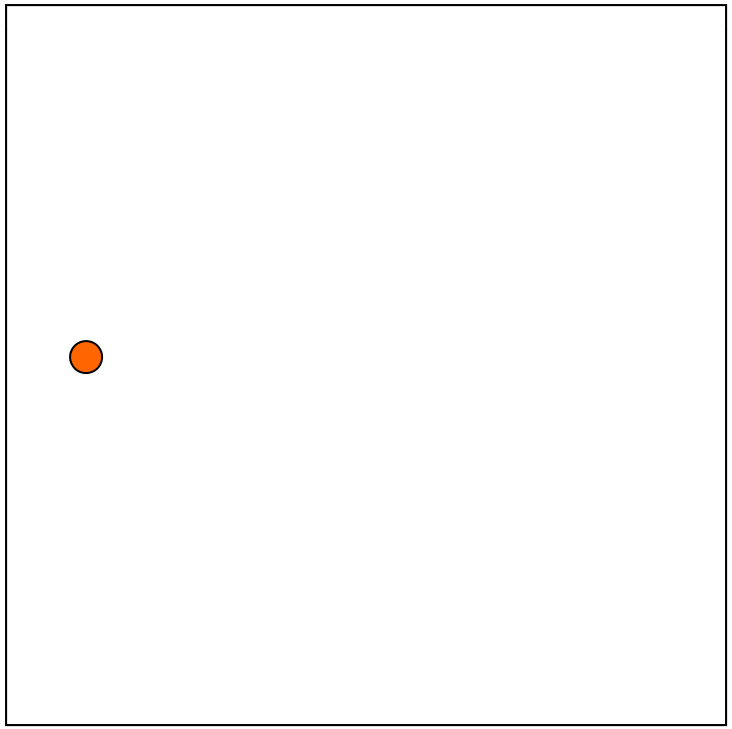
Building Polygons

- **`beginShape( POLYGON );`**
  - Tells the program to start the polygon
- **`vertex(x, y);`**
  - Make as many calls to this as you have vertices in your polygon.
- **`endShape( CLOSE );`**
  - Finishes the shape, connecting the last vertex to the first vertex to close the polygon, then colors it with the current `fill()` color

Building Polygons

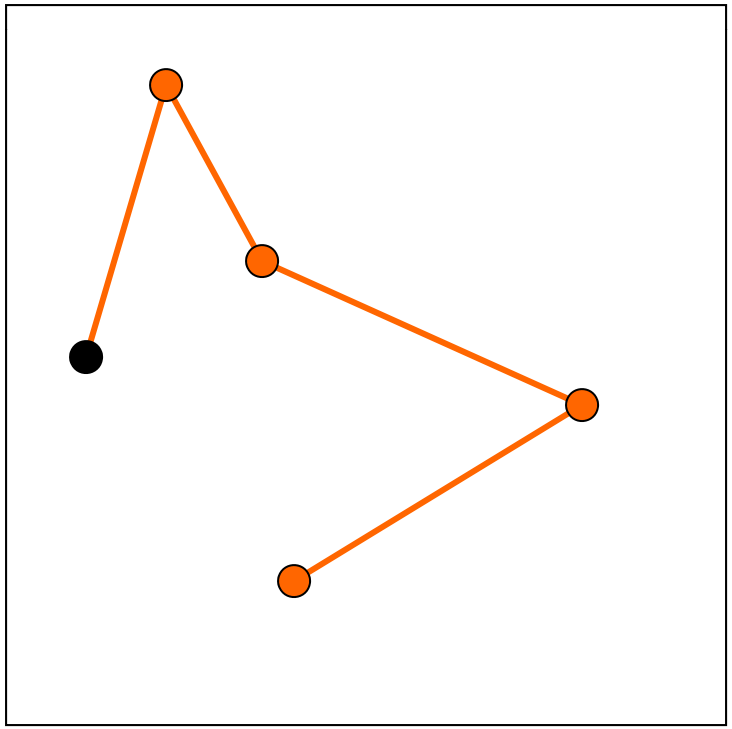
```
beginShape();
vertex(10, 50);
```

(starts a new polygon, and begins at point (10, 50).)



Building Polygons

```
vertex(20, 10);
vertex(30, 40);
vertex(80, 60);
vertex(40, 80);
```

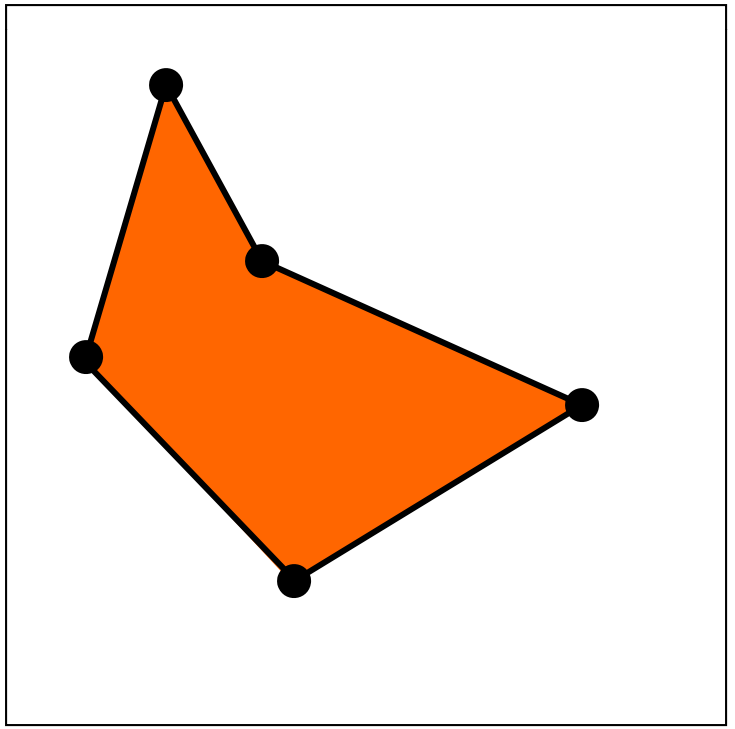


(adds 4 more points to the polygon, and connects them in the order they are called.)

Building Polygons

```
endShape(CLOSE);
```

(connects the last point to the first point, and fills the polygon.)



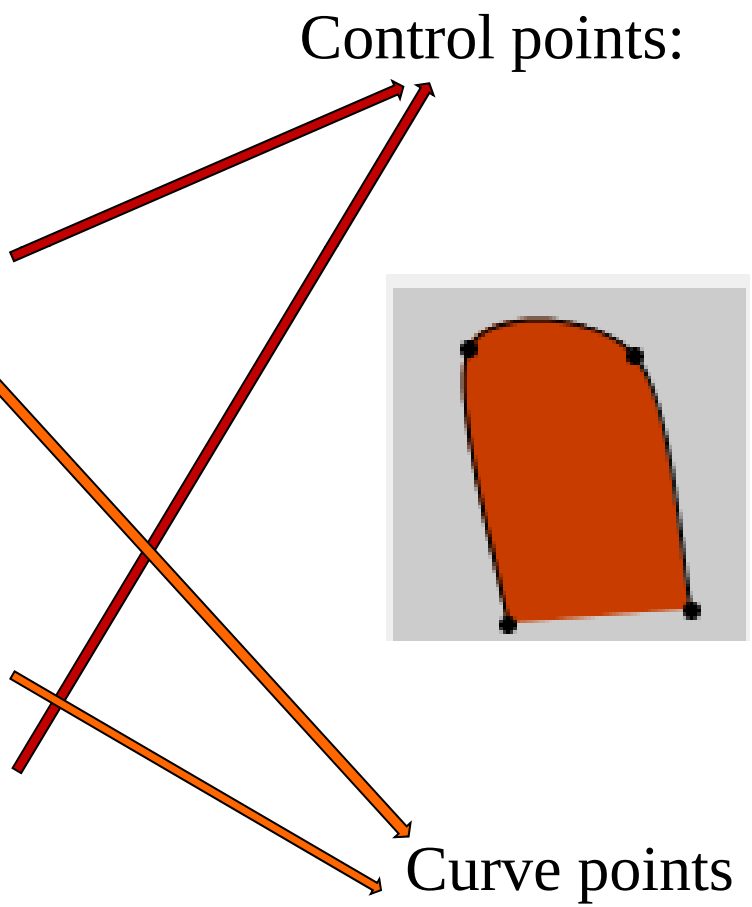
Let's Use Arrays

- Let's store the points that we're drawing.

```
int[] xvals = {10, 20, 30, 80, 40};
int[] yvals = {50, 10, 40, 60, 80};

beginShape();
for(int i = 0; i < xvals.length; i++) {
  vertex( xvals[i], yvals[i] );
}
endShape(CLOSE);
```

Draw Curves using curveVertex()

```
fill(200, 60, 0);
beginShape();
curveVertex(84,  91);
curveVertex(84,  91);
curveVertex(68,  19);
curveVertex(21,  17);
curveVertex(32, 95);
curveVertex(32, 95);
endShape();
```



Nested Loops

- Nesting of loops refers to putting one loop inside the braces of another

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

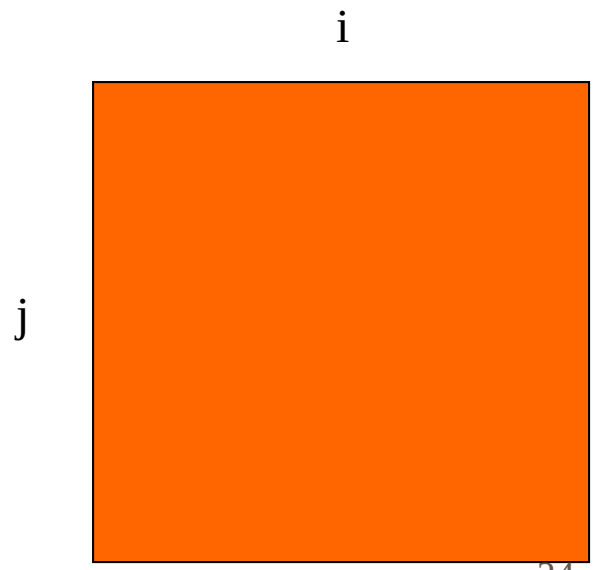
- For each **i**, the inside **j** loop will run through;
- then **i** will increment;
- then the inside **j** loop will run through again…

Nested Loops

- Let's look at this thing closer.

Sets i to 10. First step.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

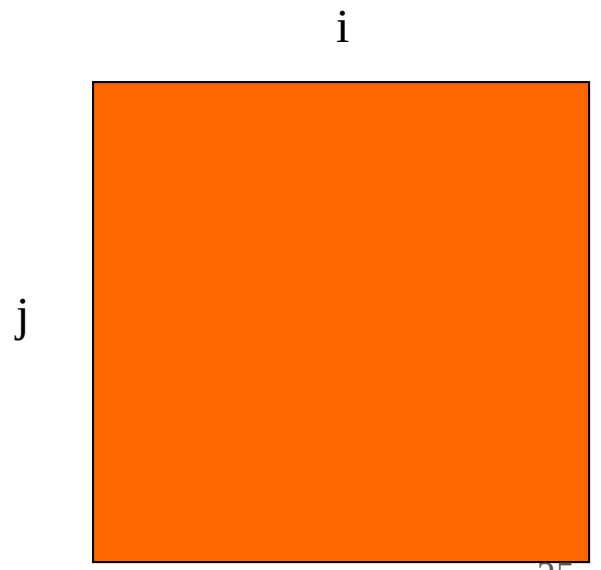


Nested Loops

- Let's look at this thing closer.

Sets j to 10. That's all for now.

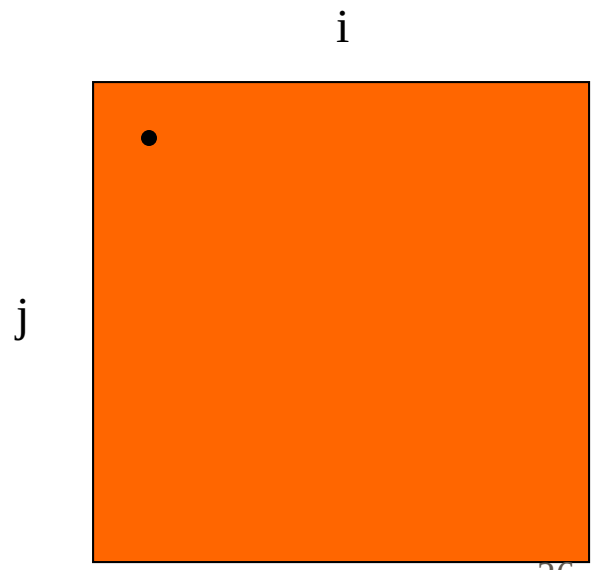
```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```



Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```



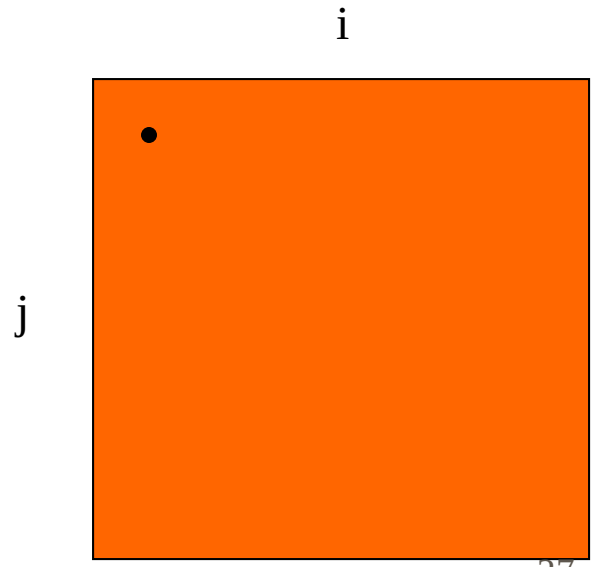
i = 10, j = 10. Draws a point at (10, 10).

Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

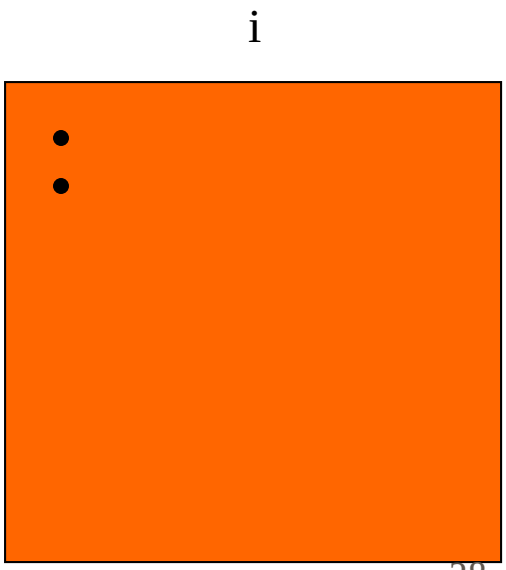
Now we jump to the final statement of inner loop, and increment j by 10, and then check if it's greater or equal to 100, which it's not, so we continue.



Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```



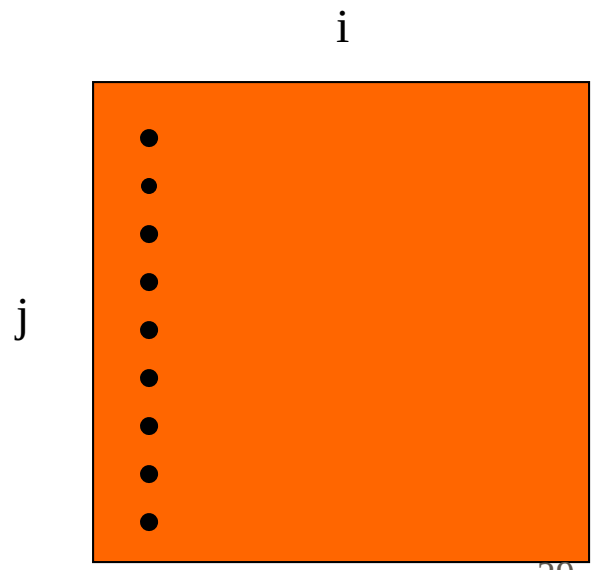
i is still 10, but j is now 20. We draw the new point, Then go to loop again.

Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

We keep looping in this j loop until it passes 100. Then what?

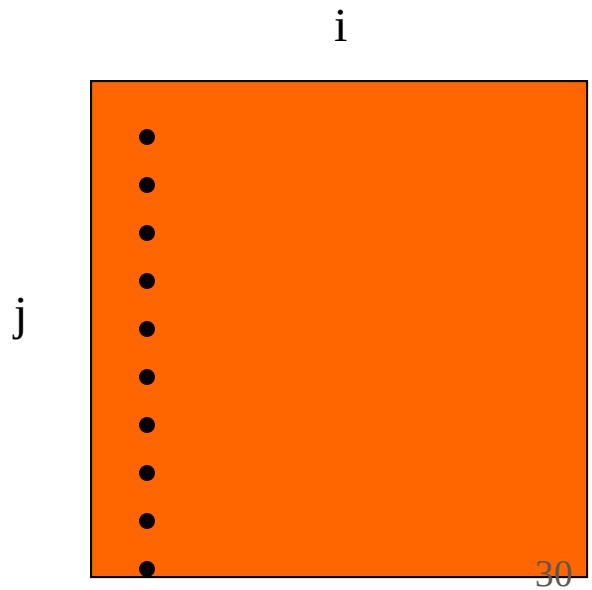


Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

Now that the program has exited from our j loop, It sees that it's still inside our i loop, and increments i by 10, and then check if i <=100, which it's not, so we continue …

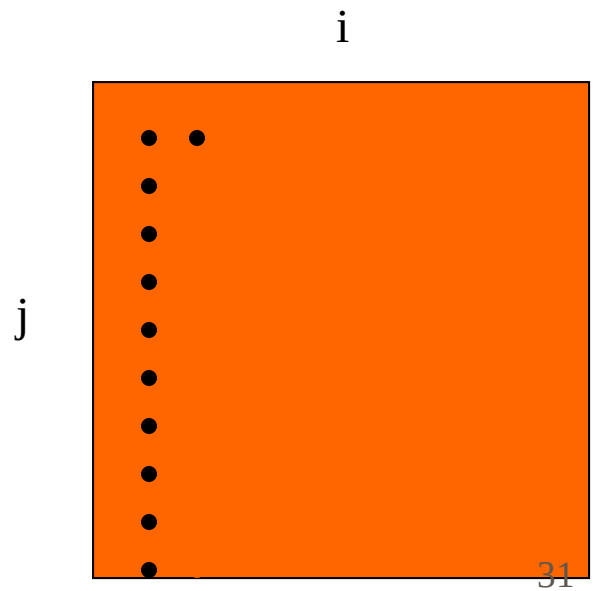


Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

The program comes to j loop again, so it loops All the way through drawing dots. Only this time, i is 20, so the dots are further to the right.

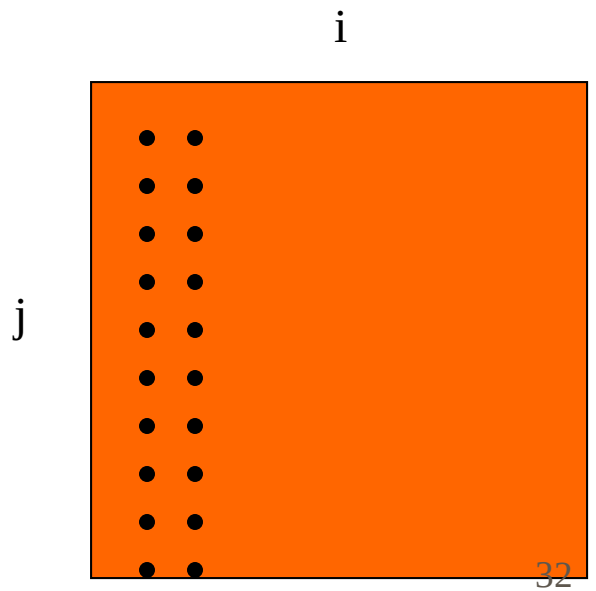


Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

Again, we keep looping in this j loop until it passes 100

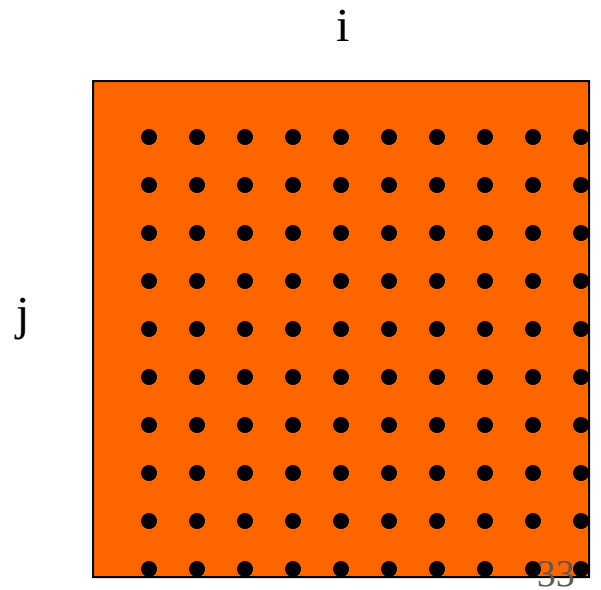


Nested Loops

- Let's look at this thing closer.

```
for(int i = 10; i <= 100; i += 10) {
  for(int j = 10; j <= 100; j += 10) {
    point(i, j);
  }
}
```

The i loop keeps incrementing in this way, drawing columns of dots, until i exceeds 100. The program then exits the i loop and therefore the nested loops



Nested Loops

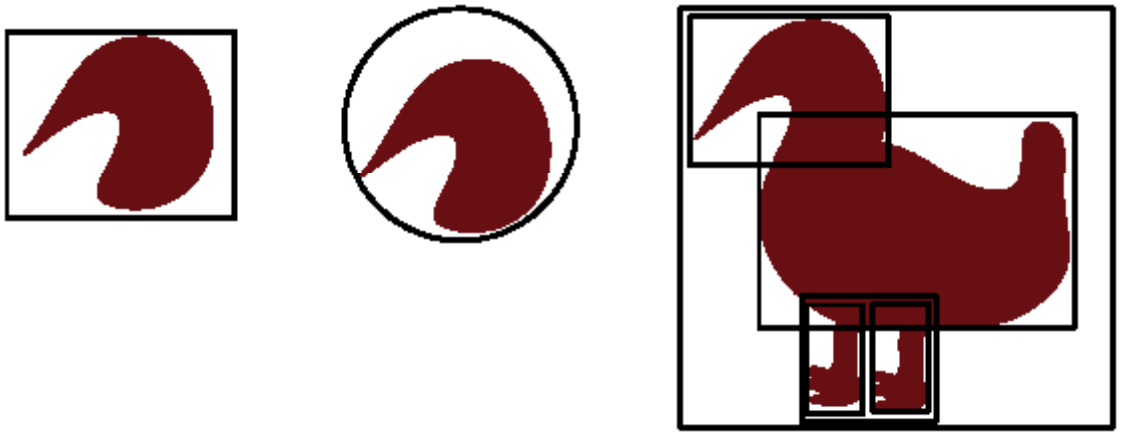
- Nested Loops are especially useful for
  - drawing in grid formations in a 2-D space
  - Running through a 2-D matrix to process data (e.g. image processing)
  - Testing among a list of objects for collision detection

Collision Detection

- Essential for many games
  - Shooting
  - Kicking, punching, beating, whacking, smashing, hitting, chopping, dicing, slicing, ...
  - Car crashes

Collision Detection for Complex Objects

- For each object ***i*** containing the set of polygons ***p***
  - Need to test for intersection with object ***j*** containing the set of polygons ***q***
    - Use bounding boxes/spheres
    - Hierarchies of bounding boxes/spheres



Distance calculation

$$d = sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2)$$

- Approximation: Manhattan distance - Shortest side/2

$$d = abs(x_1 - x_2) + abs(y_1 - y_2) - \min(abs(x_1 - x_2), abs(y_1 - y_2))/2$$

  - Example of approximation:

$$dx = 3, dy = 4 \longrightarrow d = 5$$

$$d' = 3 + 4 - 1.5 = 5.5$$

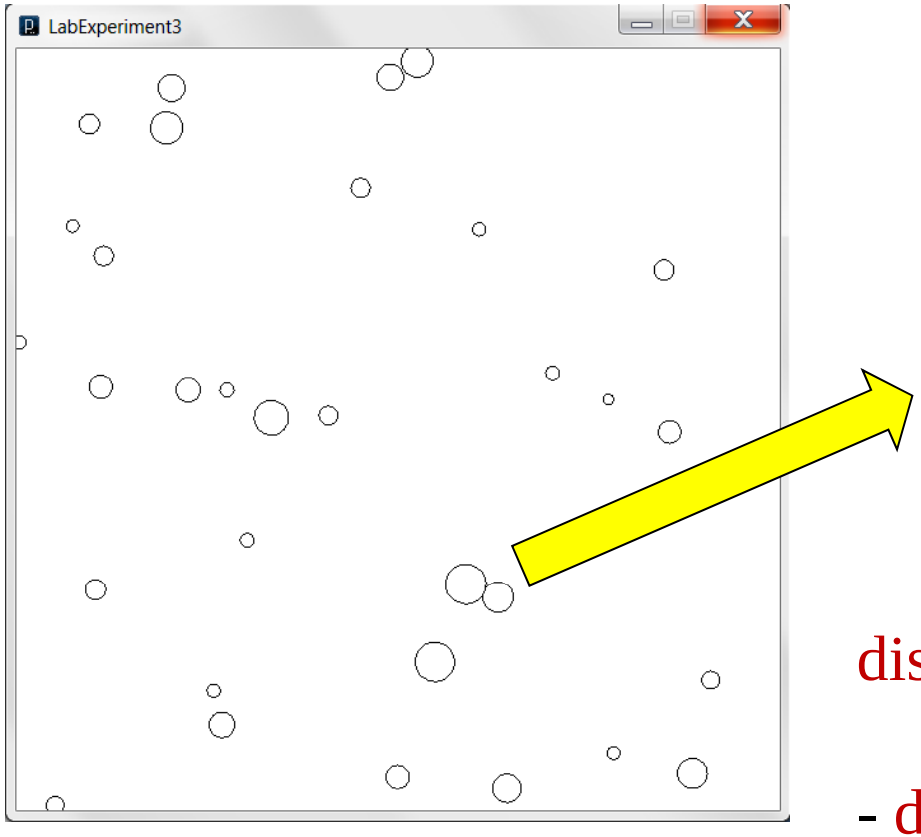
Processing: **dist()** method

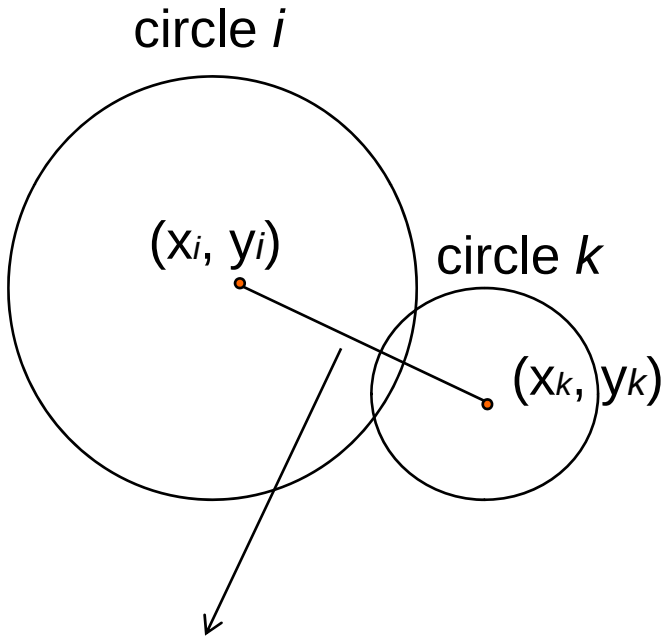
- dist(x1, y1, x2, y2)
  - Calculates the distance between two points
  - Example: draw a circle using the distance of the mouse from the origin

```
void draw() {
  background(255);
  float d = dist(0, 0, mouseX, mouseY);
  ellipse(50, 50, d, d);
  if(mousePressed) { noLoop(); }
}
```

Using dist() for collision detection

- Check the sketch ballCollisions.pde in WebCT:





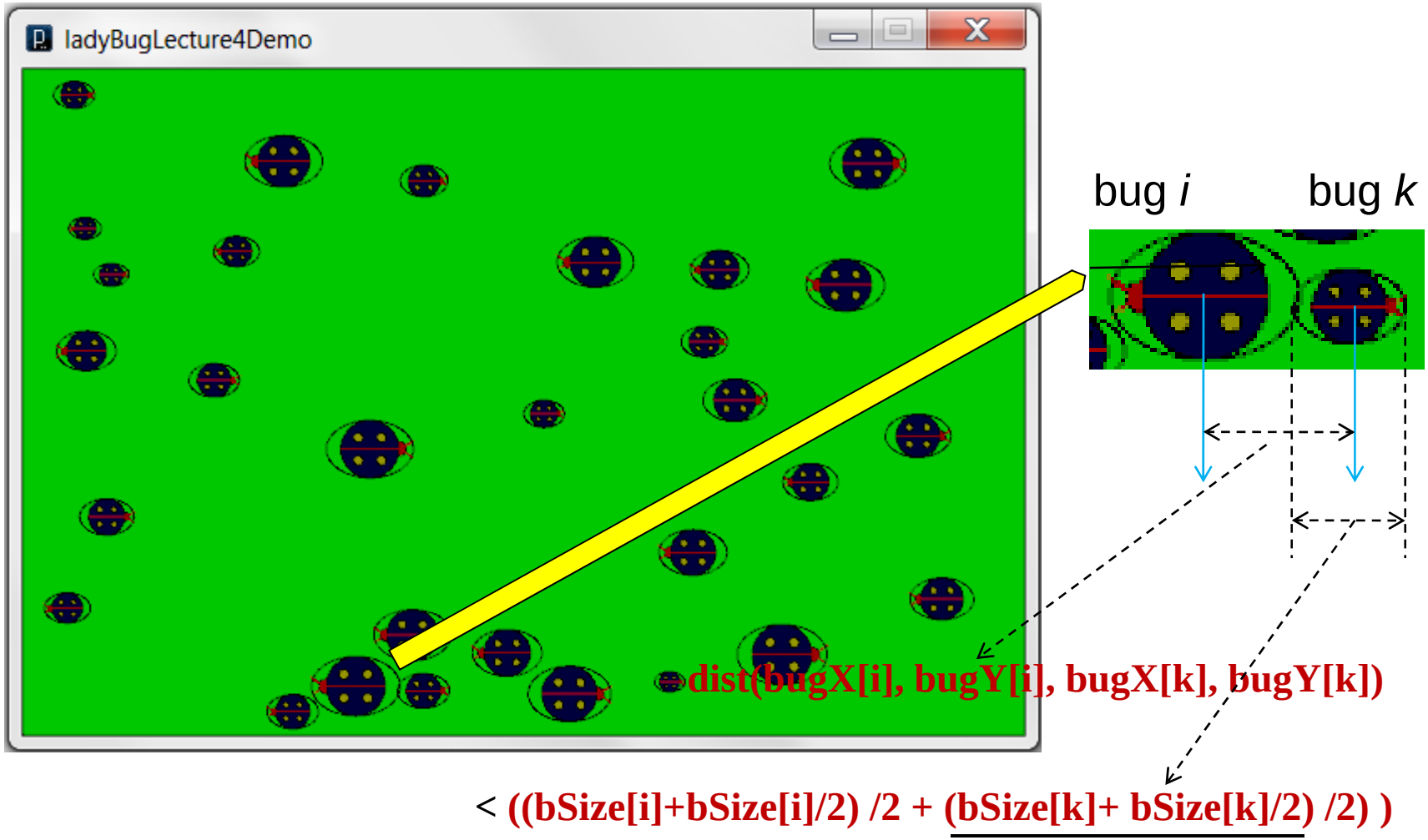
$dist(x_i, y_i, x_k, y_k) < d_i/2+d_k/2$

- d_i, d_k – diameters of the circles

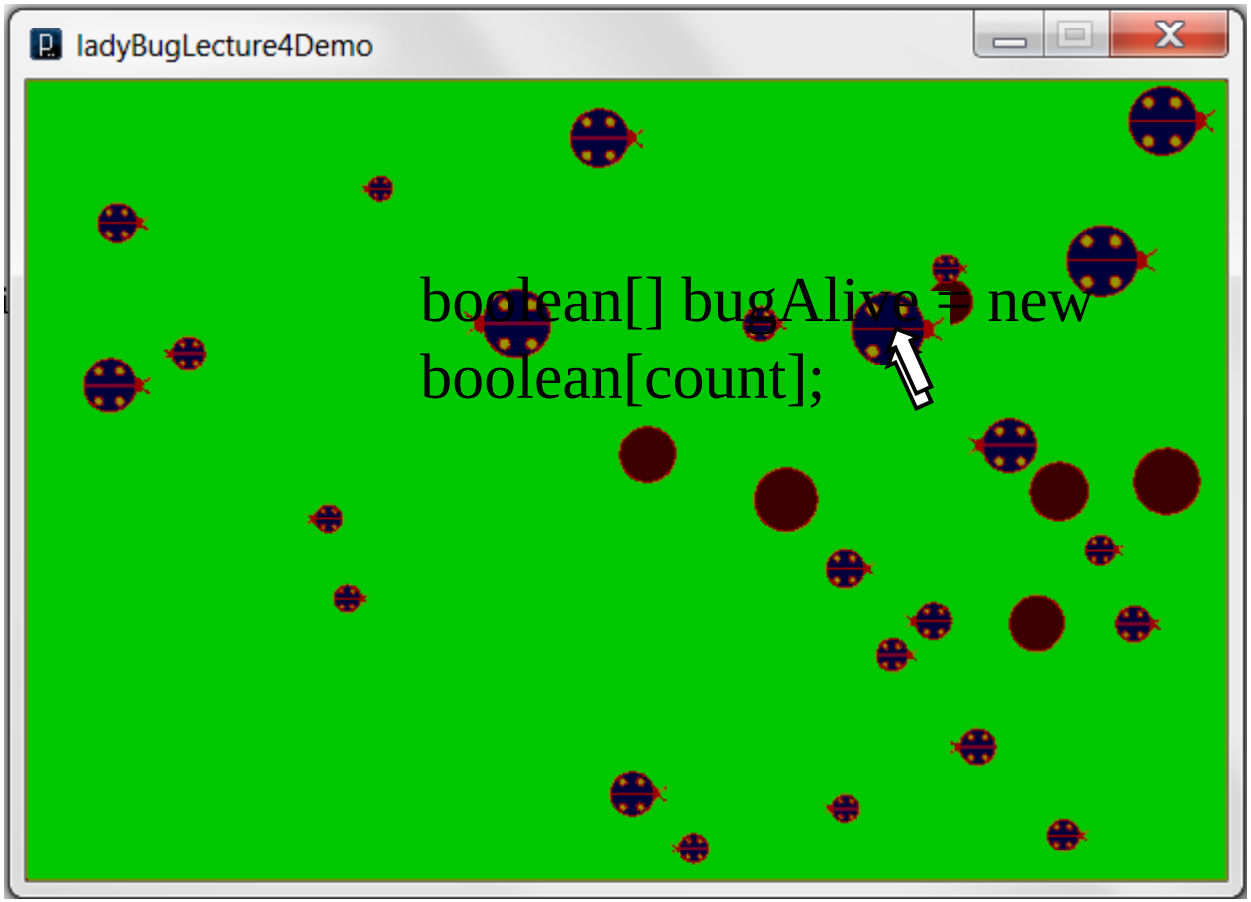
Collision detection among a list of balls

```
//This loop will loop over every ball so we can draw & move them around
for (int i = 0; i < amount; i++) {
    ...
    //This loop will loop over every ball to check for collisions with any other ball
   for (int k = 0; k < amount; k++) {
     if ( dist(x[i], y[i], x[k], y[k]) < (bSize[i]/2 + bSize[k]/2) && i != k) {
       //We get the angle of ball "k" facing ball "i"
       float angle = atan2( y[i] - y[k], x[i] - x[k]);
       //We send ball "i" away in the direction of our angle
       xVel[i] = 2 * cos(angle);
       yVel[i] = 2 * sin(angle);
       //and send ball "k" away in the opposite direction of our angle
       xVel[k] = 2 * cos(angle - PI);
       yVel[k] = 2 * sin(angle - PI);
     }
   }
```

Case study: Collision among Beatles



Kill beetles with mouse



Track on bugs' status of being alive (or killed)

```
boolean[] bugAlive = new boolean[count];
...
//in setup()
for(int i=0; i<bugX.length; i++) {
    ...
   bugAlive[i] = true;
 }
//in draw()
//Loop through the array to draw and move the bugs one by one
for(int i=0; i<bugX.length; i++) {
      if (mousePressed && dist(mouseX, mouseY, bugX[i], bugY[i])<bSize[i]) {
          changeX[i] = 0;
          changeY[i] = 0;
          bugAlive[i]=false;
        }
```

Use bugAlive for bugs' collision and drawing

```
    ...
    //This loop will loop over every bug to check for collision with other bug
   for (int k = 0; k < count; k++) {
    if(bugAlive[i] && bugAlive[k] && i != k) {
    if ( dist(bugX[i], bugY[i], bugX[k], bugY[k]) < ((bSize[i]+bSize[i]/2)/2 + (bSize[k]+
            bSize[k]/2)/2) ) {
            ...
    }
    ...
     //Drawing the bug 'i' based on its status (alive or not)
     if(bugAlive[i]) {    //if the bug is alive draw it as a normal bug
          ...
     } else {  //if the bug was killed draw it as a dark red circle
          fill(60, 0, 0);
          ellipse(0, 0, bSize[i], bSize[i]);
     }
    popMatrix();
} // end of for-loop
```

Summary

- Array
- Mouse interactions
- Polygons
- Nested loops
- Collision detection
- Case study: Collision among Beatles