Programming for Non-Programmers Java workshop 1

Workshop Description

Want to learn programming but do not know where to start from? In this workshop, we will be using **Java**, one of the popular computer programming languages, to build a simple but fun graphical application, a bomb sweeper game. Throughout writing your own program, we aim at you learning fundamental programming concepts, including data types, variables, functions, loops and conditionals, most of which are also applicable to many of other programming languages. (For the sake of simplicity on drawing graphical elements onto a screen, we will use a software called Processing on top of Java.)

Target Audiences

- People who are eager to learn programming (no prior experience is required)
- People who want to learn programming but do not know where to start from
- People who may be interested in taking Computer Science courses in near future

Agenda

6:00 - 6:10: [1] Introduction
6:10 - 6:30: [2] Development environment setup
6:30 - 7:00: [3] Game building 1 (Setup window, draw elements)
7:00 - 7:10: [4] Break
7:10 - 7:30: [5] Game building 2 (Mouse operation)
7:30 - 7:50: [6] Brush up your application (Programming challenges)
7:50 - 8:00: [7] Evaluation form and class photo

Evaluation Form

http://goo.gl/forms/O6t2FbFokZ

by Tufts CSLOL (Computer Science League of Learning)
on April 20, 2016 from 6-8pm
at Halligan Hall H112

Java workshop 1

---

[1] Introduction

1.1 Workshop Description

Want to learn programming but do not know where to start from? In this workshop, we will be using Java, one of the popular computer programming languages, to build a simple but fun graphical application, a bomb sweeper game. Throughout writing your own program, we aim at you learning fundamental programming concepts, including data types, variables, functions, loops and conditionals, most of which are also applicable to many of other programming languages. (For the sake of simplicity on drawing graphical elements onto a screen, we will use a software called Processing on top of Java.)

1.2 Target Audiences

- People who are eager to learn programming (no prior experience is required)
- People who want to learn programming but do not know where to start from
- People who may be interested in taking Computer Science courses in near future

1.3 Agenda

6:00 - 6:10: [1] Introduction
6:10 - 6:30: [2] Development environment setup
6:30 - 7:00: [3] Game building 1 (Setup window, draw elements)
7:00 - 7:10: [4] Break
7:10 - 7:30: [5] Game building 2 (Mouse operation)
7:30 - 7:50: [6] Brush up your application (Programming challenges)
7:50 - 8:00: [7] Evaluation form and class photo

[2] Development environment setup

To start writing program in Java, you first need to set up a so-called “development environment” on your computer. Generally speaking, we install a set of tools, 1) a text editor to express your idea as program source codes, 2) compiler to convert your source codes into an executable that a computer can understand and 3) runtime to execute the executable, your program. In this workshop, instead of installing the three individually, we install a software called Processing (https://processing.org/), which provides all of the three functioanlities for us within one softawre. (And also, the software helps us to draw graphical elements.)

2.1 Install Processing software

1. Go to the Processing webpage (https://processing.org/) and click “Download” from menus on the left side.
2. Download a file that matches to your computer’s OS, (Windows, Mac, Linux), and save it to wherever you prefer on your computer.
   (The latest ver. is 3.0.2 as of Feb. 15, 2016.)
3. Windows users:
   a. Double click on the downloaded file.
   b. Move the content (the folder of Processing) to i.e. your Desktop.

   Mac users:
   a. Double click on the downloaded file to unzip.
   b. Move the “Processing.app” to “/Applications” directory.

   Linux users:
   a. (Ask TAs if you need any helps.)

2.2 Start Processing software

1. Open the “Processing” application.
2. For the first time opening it, you will be prompted with a welcome message. Select the option “Create a new sketchbook folder for ....” and click “Get Started”.
3. Then, you will specify where to save source files that you will write in this workshop. In this document, we set it as:
   “/PATH/TO/YOUR/PREFERRED/DIRECTORY/cslol/”.
   i.e. “~/Desktop/cslol”

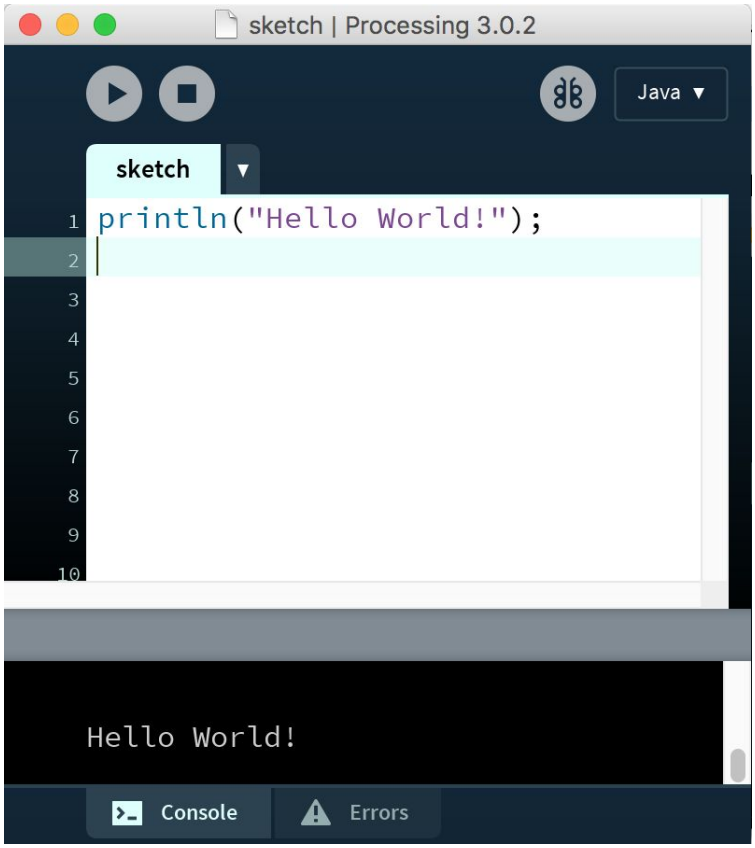
4. After the steps above, you will see a window having run/stop buttons on the top, a text editor on the middle and a console on the bottom.

2.3 Your first program!

1. Type the following in the text editor,

```
println(“Hello World!”);
```

2. Hit the “Run” button, then you will see “Hello World!” in the console.



Congratulations! You are now officially a programmer!
(Hitting the run button brings another window with a small gray square in it, but we can simply ignore it for now. We will use it later on.)

3. Hit the “Stop” button.

Getting excited to start learning programming?
If yes, great! The program you have written prints “Hello World!” in the console.

Curious what the program you have just written means?
If yes, great! What it meant was: call the function named “println” with the String argument “Hello World!”.

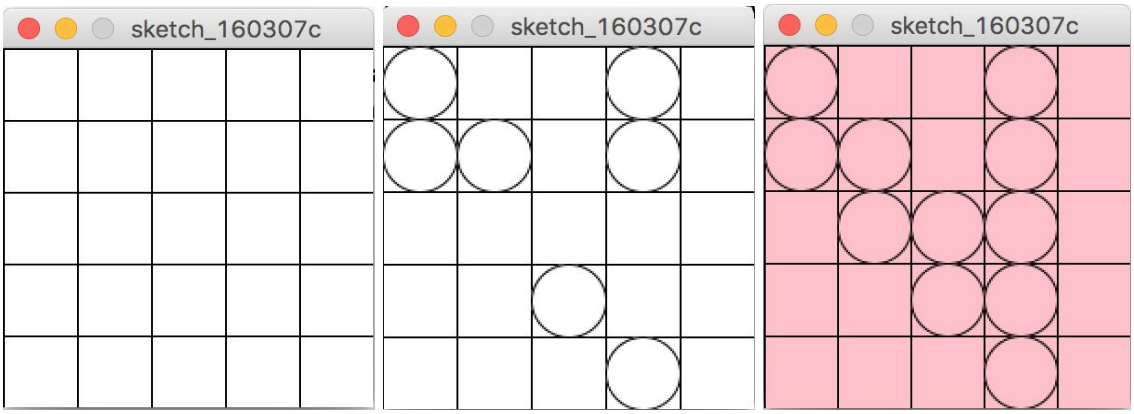
Complicated enough?
No worry! Let’s get started learning fundamental programming concepts together through building your own game application, and you will be able to find by the end what the sentence above meant.

[3] Game building 1 - Bomb Sweeper Game

3.1 Objective

You learn fundamental programming concepts, including data types, variables, functions, loops and conditionals, through building a bomb sweeper game.

3.2 Appearance of the game



(1) (2) (3)

3.3 The game rule and requirements

The rule of the game: To win the game, find and mark all of the cells, excepts cell(s) where a bomb is underneath it.
There are 3 requirements:

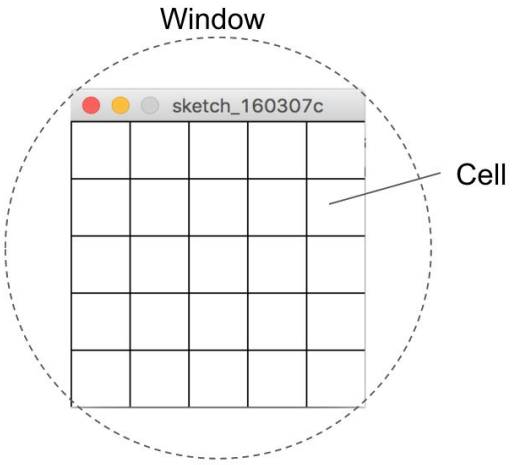
(1) When launching your program, a window with 5 by 5 grid shows up.
(2) When performing a left-click on a cell, if the clicked cell **does not have** a bomb, marking the cell with a circle.
(3) When performing a left-click on a cell, if the clicked cell **does have** a bomb, changing the widow color to red. (game over)

3.4 Let's get started to build your game!

1. Create a new file. (Menu bar -> "File" -> "New")
2. Rename the newly created file to "bomb_sweeper". (Click the triangle icon on the tab and next to "sketch_xxxxxxa" -> "Rename")

(1) Setup a window

To satisfy the requirement (1), let's take a look the appearance of the game one more time, and define two words, window and cell, as followings.



That is, the above figure shows a window containing 25 cells aligned as 5 by 5 grid. Back to Processing software, and hit the "run" button, then you will see a window. (Processing software makes it very easy for us to have a window.) Done...

but ... it is too small, so let's make it bigger. Type in the following:

```
size(200, 300);
```

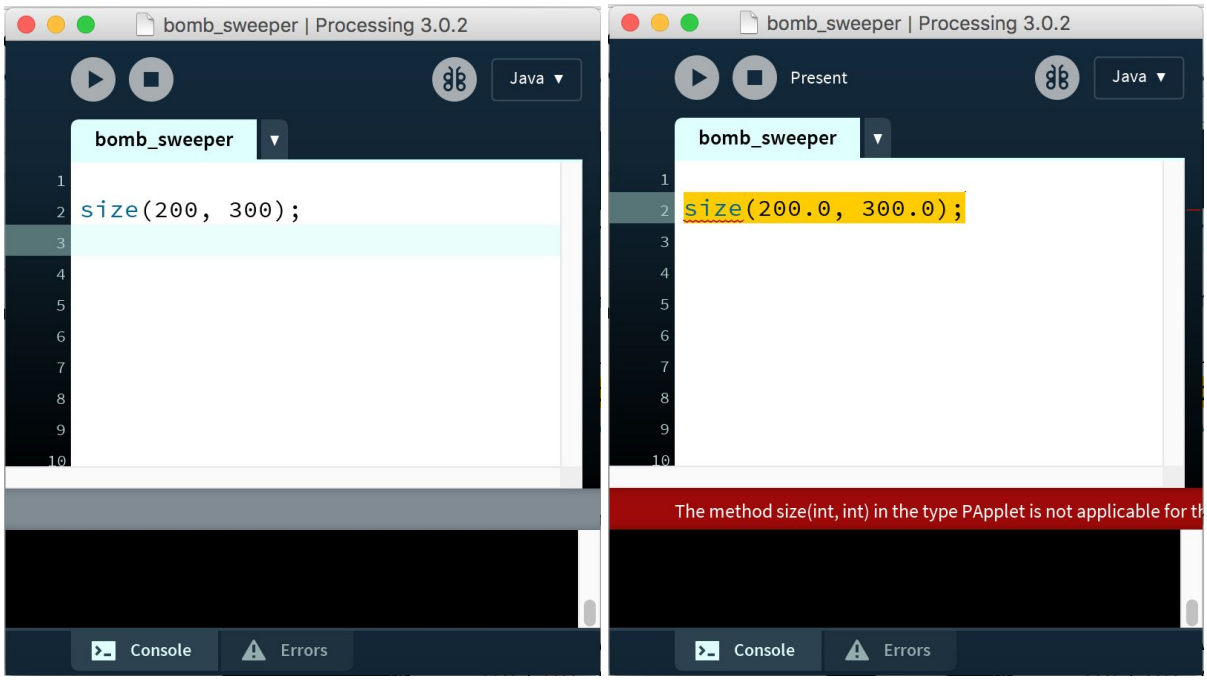
Then save the file and hit the "Run" button. You should see a bigger window now. What you did was you used a function named "size" that changes the window size to "200" (width) by "300" (height).

As we expect, if we can change "200" and "300" to any number to have a different size of window.
Well, how about changing the width and height from 200 x 300 to 200.0 x 300.0.

```
size(200.0, 300.0);
```

Save the file and hit the “Run” button.
Surprisingly, the compiler got angry (showed an error message), and we did not get a window this time.

Why is that? It’s time to introduce “data type” as one of the fundamental programming concepts.



Data Types

When hearing the word “data”, you could imagine that is something about a collection of numbers, texts, etc, and you know numbers and text strings are different things in some sense. You are actually close to find data types as a concept already; that is, data has a type. For example, 123 (data) is an integer (type), “candy” (data) is a String (type), etc.

Lesson: “Each data must come with a type” in Java.

Here is some primitive data types in Java.

`int`	Integer number
`float`	Real number (single-precision floating point)

`double`	Real number (double-precision floating point)
`char`	Character
`boolean`	`true` or `false`

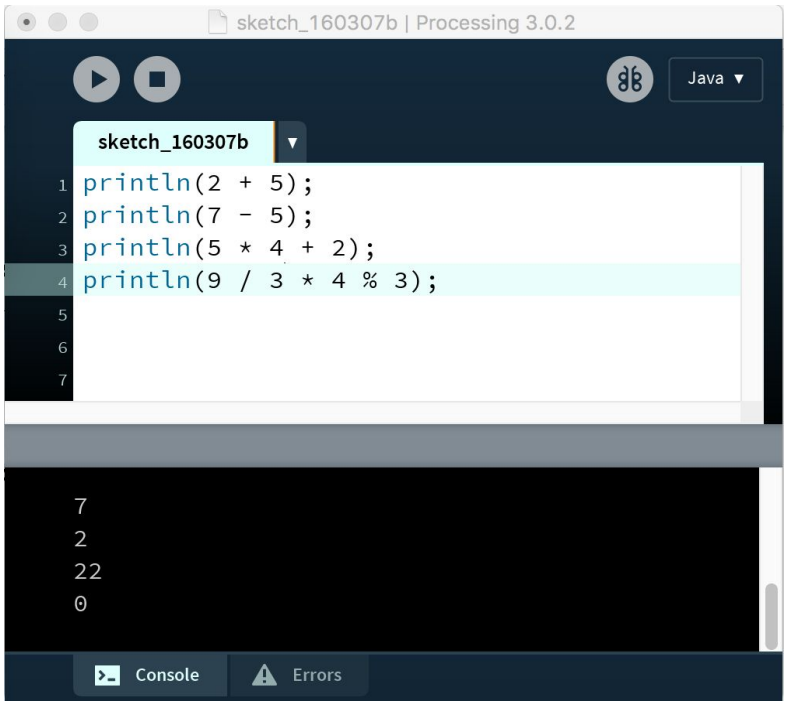
and (not exactly primitive)

`String`	Text string

To dive into the computer's world and to find why understanding data type is so important, let's take a look examples of arithmetic in Java.
Java allows us to perform operations such as addition (+), subtraction (-), multiplication(*), division(/) and remainder (%).

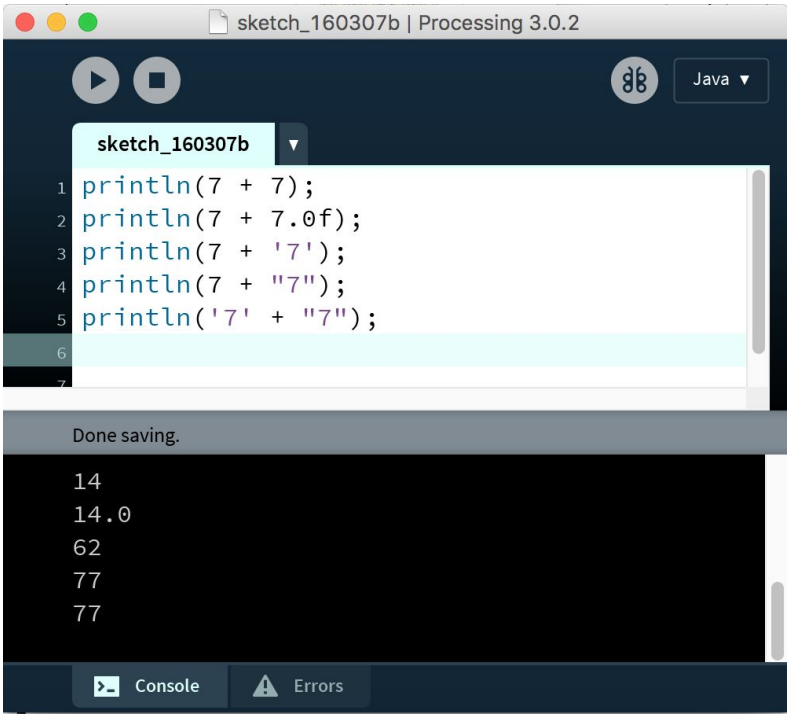
Try typing in the followings and hit the "run" button.
(println() is a function used to print out the result of the arithmetic operation written in the parentheses.)

```
println(2 + 5);
println(7 - 5);
println(5 * 4 + 2);
println(9 / 3 * 4 % 3);
```



Did you get the same results? How about the followings?

```
println(7 + 7);
println(7 + 7.0f);
println(7 + '7');
println(7 + "7");
println('7' + "7");
```



So that is, for example:

1) 7 as `int` and 7.0 as `float` are different in a computer program.
2) 7 as `int` and '7' as `char` are different in a computer program.
3) 7 as `int` and "7" as `String` are different in a computer program.
4) '7' as `char` and "7" as `String` are different in a computer program.
5) … Can you come up with other examples?

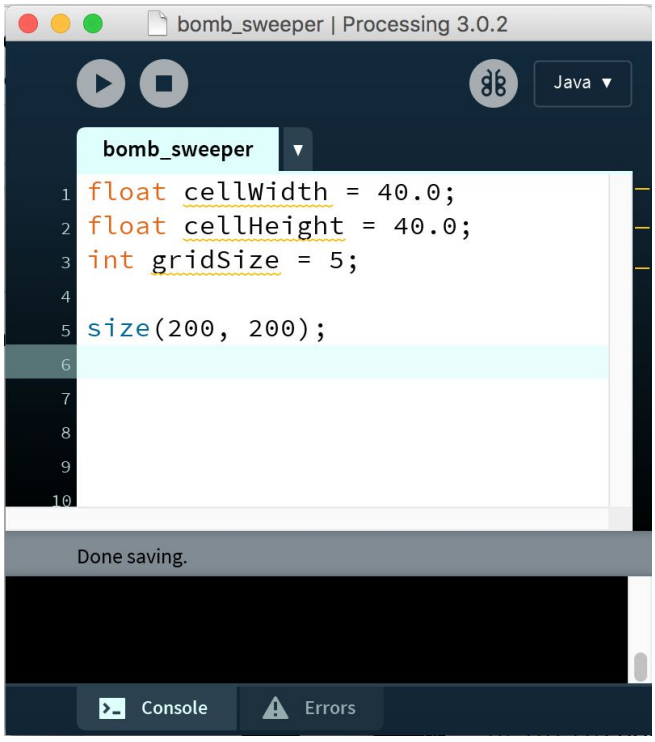
Back to the sizing a window, why the compiler got angry was we used `double` (real number) to specify its width and height while we needed to use `int` (integer number) for them.

(2) Cells

Let's first think about how to represent a cell in the window.
What kinds of properties do we need to put 25 cells (5 by 5 grid) in a window?

Here is an idea: we should know width and height of cell, how many cells in a row and column. Type in the followings:

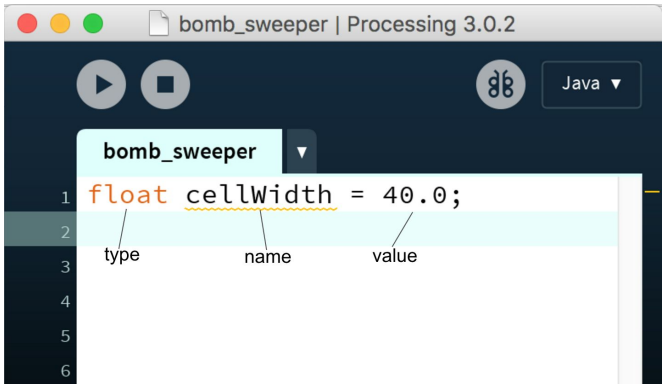
```
float cellWidth = 40.0;
float cellHeight = 40.0;
int gridSize = 5;
```



And, it's time to introduce "variables" as a concept.

Variables

A variable could be considered as a way to refer to value.



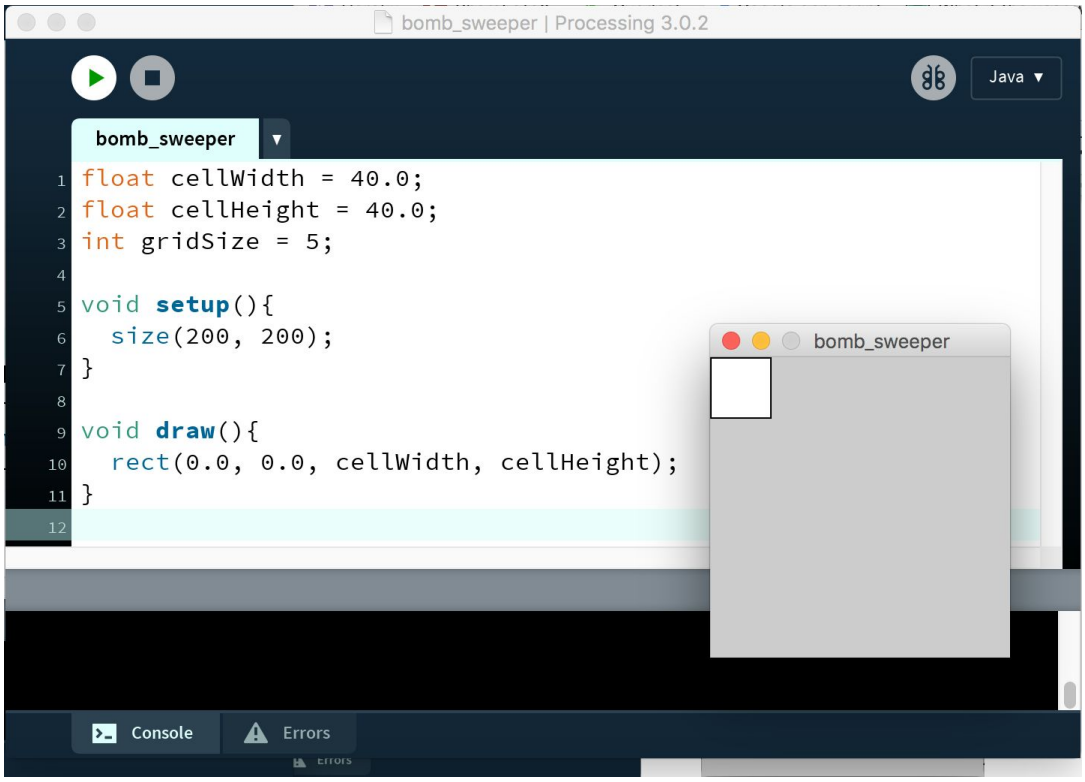
In the figure, the variable whose type is "float", name is "cellWidth" and value is "40.0" is to represent that the cell width is 40.0 (real number with single-precision floating point). The "=" sign is to assign a value to a variable.

Lesson: In Java, “each variable must come with a type for its content, and the content can be updated (excepts for constant).”

(3) Draw a cell

So far, we have a window, and defined cells, which means that we are ready to draw cells on the window! Type in the followings:

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

void draw(){
  rect(0.0, 0.0, cellWidth, cellHeight);
}
```



Hit the “Run” button, then you should see a (white) cell drawn in the window!
Let’s take a look the code you have just typed in, and it is time to introduce “functions” as a concept.

Functions

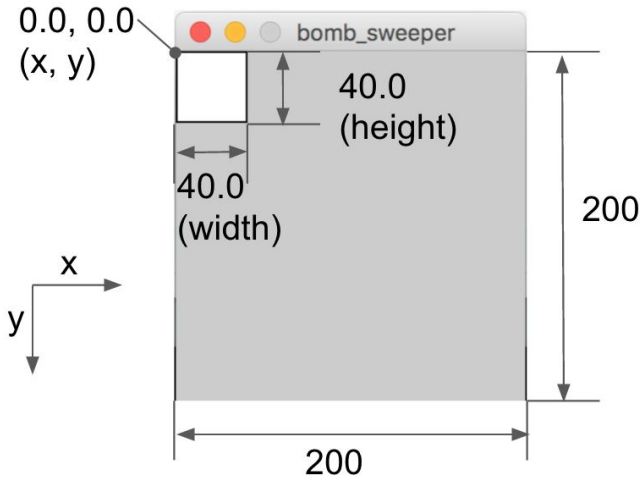
In the figure, believe or not, we are using 4 functions already, “setup”, “size”, “draw” and “rect”. In this case, all of the four are defined functions by Processing software; on the other hand, we can also define our own functions by ourselves. (One of the programming challenges in this workshop asks you to define and use one by yourself.)

A function could be described with 4 things, return type, function name, argument(s) and processes executed inside the function. Let’s take a look an example function below, which simply calculate an addition of two integers. “int” is the return type of the function. “add” is the function name. “int x and int y” are arguments. “int sum = x + y; return sum;” are the processes in the function.

```
int add(int x, int y){
  int sum = x + y;
  return sum;
}
```

Lesson: In Java, a function can group a sequence of processes.

Back to the game creation, “setup” is a function that is called once when you start program, so we put “size” function in it (because we want to call “size” function only once in this game creation.) “draw” is a function that is called many times to draw elements on the window. “rect” is a function taking 4 arguments, x, y, width and height as float, to draw a rectangle. (https://processing.org/reference/rect_.html) An important thing to know is that the origin is top-left corner.



(3) Draw 25 cells aligned as 5 by 5 grid

Let's draw more cells in the window! The goal is to draw 25 cells aligned as 5 by 5 grid. How could we achieve that? One simple way is to call the "rect" function 25 times by specifying x and y (and width and height) for each cells. Like,

```
void draw(){
  rect( 0.0,   0.0, cellWidth, cellHeight);
  rect( 0.0,  40.0, cellWidth, cellHeight);
  rect( 0.0,  80.0, cellWidth, cellHeight);
  rect( 0.0, 120.0, cellWidth, cellHeight);
  rect( 0.0, 160.0, cellWidth, cellHeight);
  rect(40.0,   0.0, cellWidth, cellHeight);
  rect(40.0,  40.0, cellWidth, cellHeight);
  rect(40.0,  80.0, cellWidth, cellHeight);
  // ... ... ... do like this for total 25 times ...
}
```

... But it seems too much to do by hand. (Think a case where the grid is 100 by 100 ...) So, it is time to introduce "loops" as a concept.

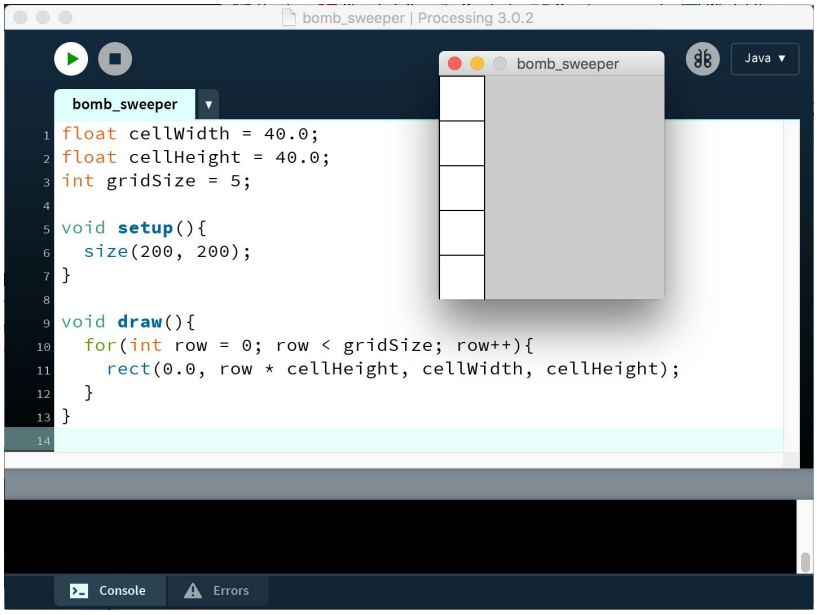
Loops

First, let's think to draw 5 cells (aligned as 5 by 1 grid). Type in the followings:

```
void draw(){
  rect(0.0, 0 * cellHeight, cellWidth, cellHeight);
  rect(0.0, 1 * cellHeight, cellWidth, cellHeight);
  rect(0.0, 2 * cellHeight, cellWidth, cellHeight);
  rect(0.0, 3 * cellHeight, cellWidth, cellHeight);
  rect(0.0, 4 * cellHeight, cellWidth, cellHeight);
}
```

Then, achieve the same thing by using a "for" loop. Type in the followings:

```
void draw(){
  for(int row = 0; row < gridSize; row++){
    rect(0.0, row * cellHeight, cellWidth, cellHeight);
  }
}
```



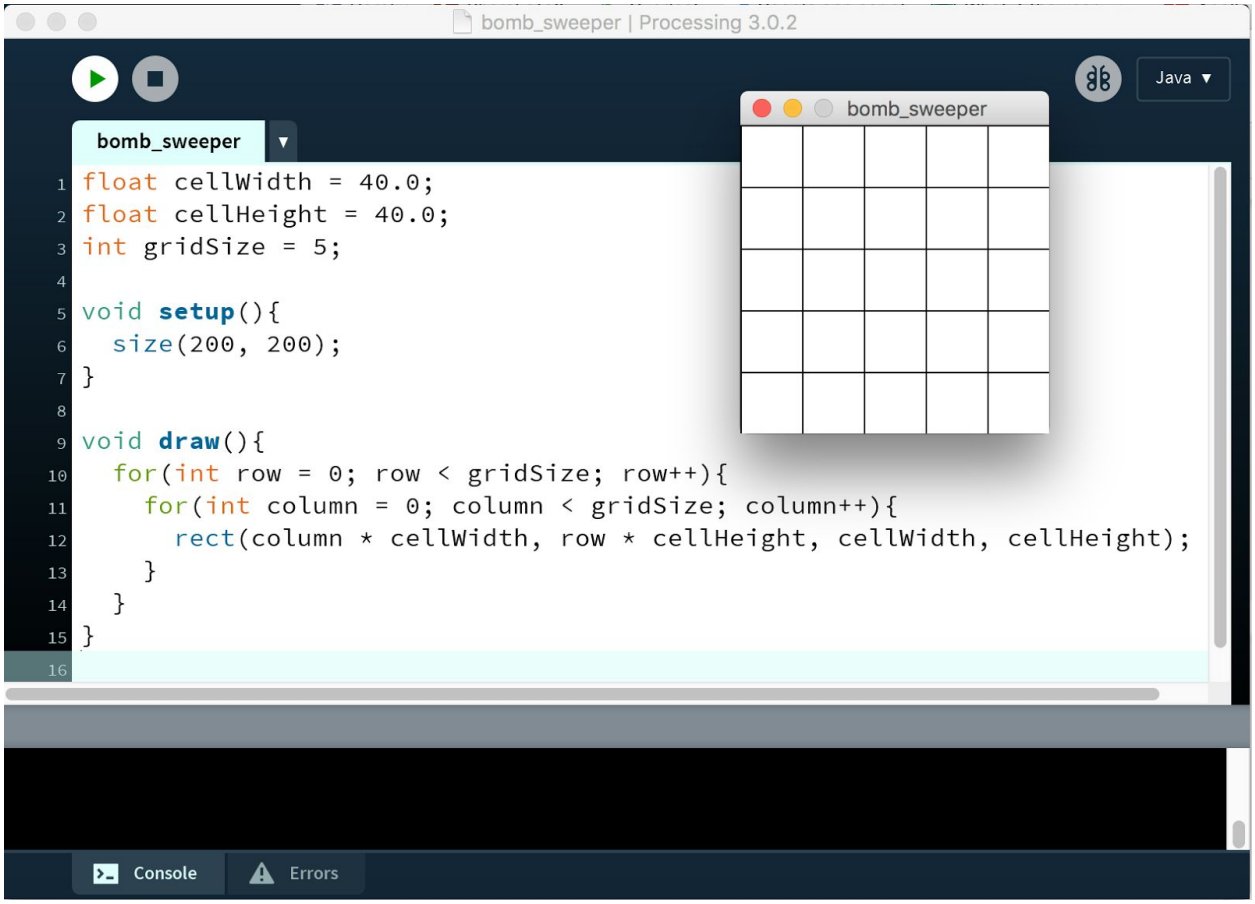
Hit the “Run” button, and then you should see the window with 5 cells.
Here, we used a “for” loop that is for iterating specified processes for a certain times. In the figure, we used the “for” loop to call the “rect” function 5 times.
How come 5? Because we wrote:

1) Initialize “row” with “0” (integer).
2) Repeat the processes inside the “for” loop, which is a “rect” function call, while “row” is smaller than “gridSize”, which was set to “5”.
3) Increment “row” by 1 when one loop is done.

Therefore, “row” changes 0 to 5, and also we can apply a bit math to calculate the y position of each cell in a column.

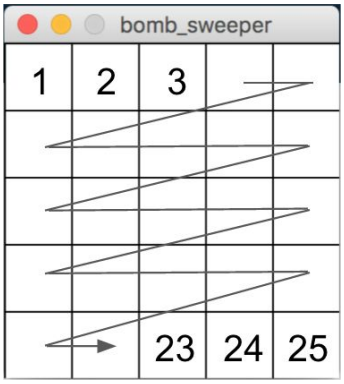
Now we are able to draw 5 cells, so let’s try to draw 25 cells aligned as 5 by 5 grid. Any idea? We so far manipulate the “row” number changed to 0 to 5 (when it is 5, 5 is not smaller than 5; therefore, no drawing happens), so how about to manipulate the “column” number with the same manner? Also, a fact is that we can nest “for” loops. Type in the followings:

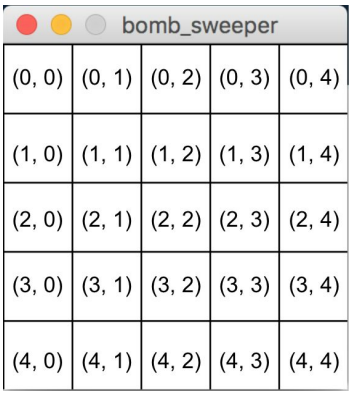
```
void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
    }
  }
}
```



Congratulation! Your program now satisfies the requirement (1).

By the way, can you tell in which order the cells are drawn?
The answer is ... Did you get right?





[4] Break

We have a short break here. If you have any questions so far, please feel free to talk with me, TAs and your peers.

Recap briefly:
So far, we have written about 15 lines of codes, which draw 25 cells as 5 by 5 grid of specified sizes on the window which also has its specified size.

Next, we will incorporate mouse actions, so that your program becomes playable (interactible). Ready?

[5] Game building 2 - Bomb Sweeper Game

(4) Detect left mouse clicks

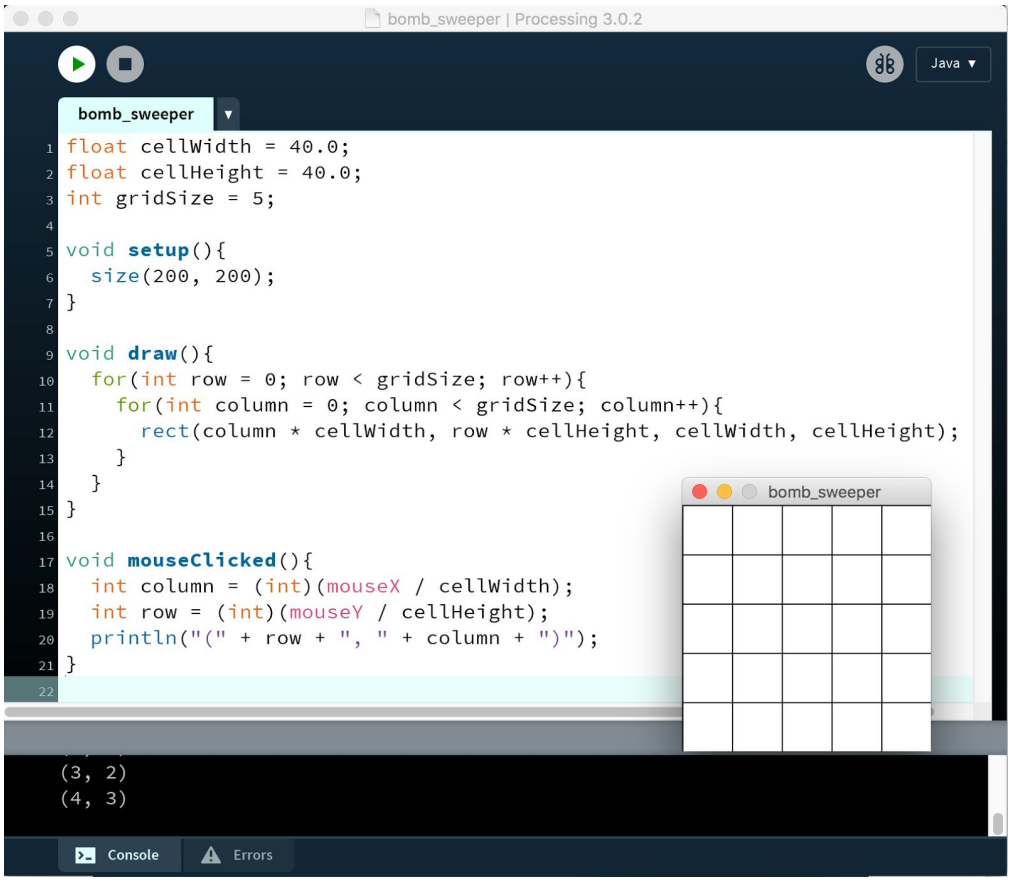
Next, we would like to handle (left) mouse clicks to mark cells on the window. Lucky, Processing software provides for us a function named “mouseClicked”, which is called when a mouse action occurred, and variables named “mouseX” and “mouseY”, which contains the current x, y position of the mouse cursor, respectively.

What we want to achieve here is that when a left click occurs, we want to translate the mouse x and y position into the row and column number, so that we can know which cell the mouse cursor clicked.

Recall that we used “row” and “column” of range from 0 to 4 to draw 5 by 5 grid cells. Now x and y can be from 0 (origin) to 200 (width and height of the window), and we know the cell width and cell height. Type in the following:

```
void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  println("(" + row + ", " + column + ")");
}
```

Hit the “Run” button, and make a left-click on a cell, and then you should see the row and column number of the clicked cell in the console.



(5) Define state of cells

So far we can detect which cell is clicked, so let's think to mark clicked cells to satisfy the requirement (2). The idea is we store the state (marked or not marked) of each cell. In order to store such, let's use a "data structure" called "array" (in our case, we actually use a 2D-array.)

An array can contain values that are a fixed size of the same data type. For example,
`int[] odds = {1, 3, 5, 7, 9};`

"odds" is the name of the array; "int" is the data type of contents ("[]" indicates it is an array); "{1, 3, 5, 7, 9}" are the contents of the array; and the array size is 5.

```
println(odds[0]); //1
println(odds[1]); //3
println(odds[4]); //9
```

For example,

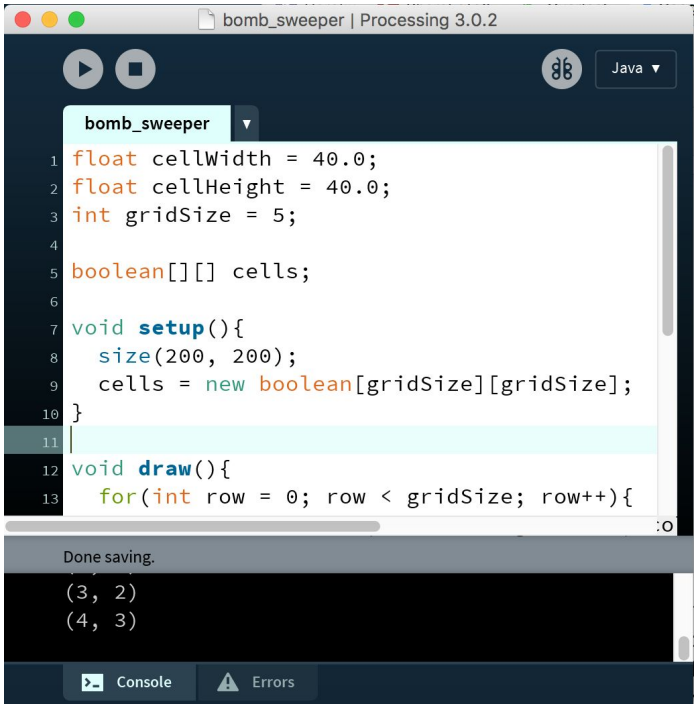
```
int[][] grids = {{ 0,  1,  2,  3,  4},
                 {10, 11, 12, 13, 14},
                 {20, 21, 22, 23, 24}};
```

"grids" is the name of the 2D-array; "int" is the data type of contents ("[][]" indicates it is a 2D-array); "{{0, 1, 2, 3, 4}, {10, 11, 12, 13, 14}, {20, 21, 22, 23, 24}}" are the contents of the 2D-array; and the 2D-array size is 3 (row) by 5 (column).

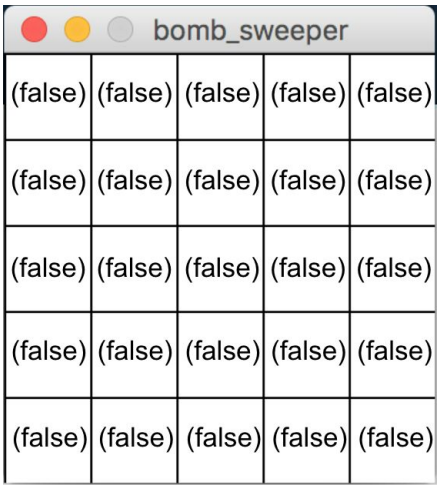
```
println(grids[0][0]); //0
println(grids[0][1]); //1
println(grids[0][4]); //4
println(grids[1][2]); //12
println(grids[2][3]); //23
println(grids[2][4]); //24
```

Back to the game creation, the 25 cells in the window look like exactly a grid, and the state of a cell will be either "marked" or "not marked", so let's represent those as a 2D-array of boolean data type by considering "marked" is "true" and "not marked" is "false". Type in the followings:

```
boolean[][] cells;

void setup(){
  size(200, 200);
  cells = new boolean[gridSize][gridSize];
}
```



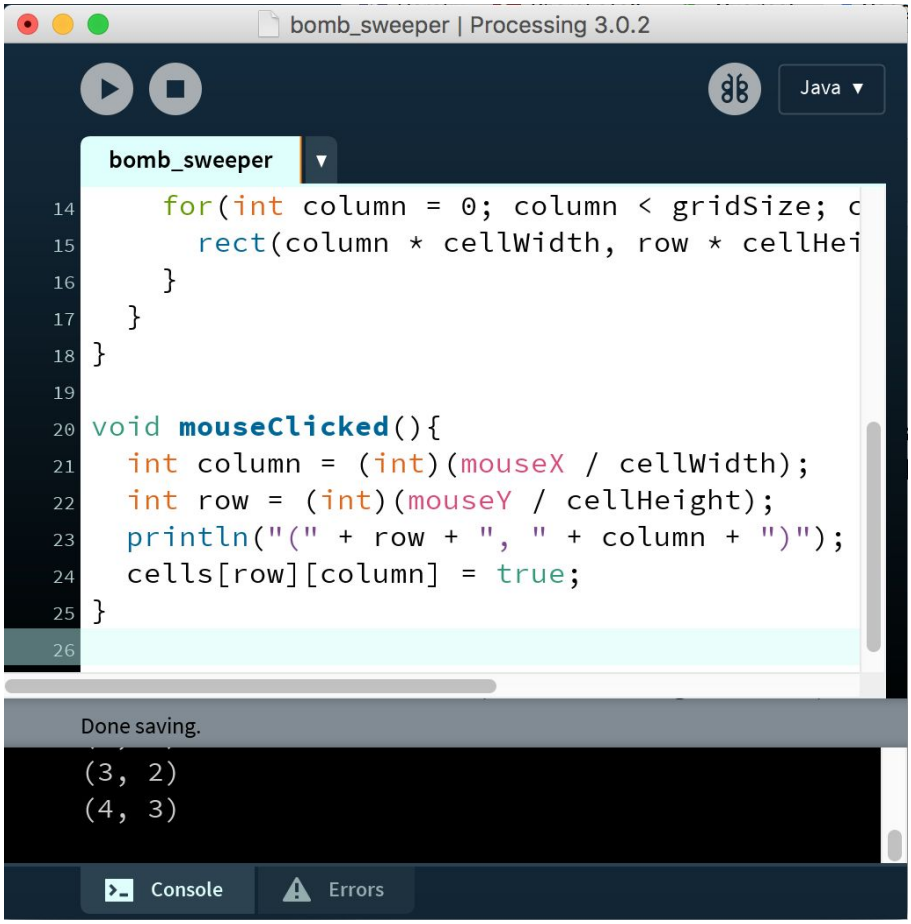
Here “boolean[][] cells;” part declare a “variable” named “cells” that is a “2D-array” of “boolean” data type. “cells = new boolean[gridSize][gridSize];” part specifies its size “gridSize” (5) by “gridSize” (5), and grabs a chunk of space on the computer memory, and also initialize each cell with “false” (default value). Therefore, the current state of each cell is all are “false” (all are not marked).



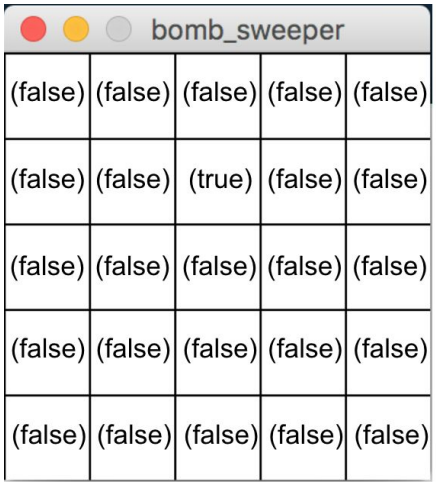
(6) Update the state of a cell by a left click

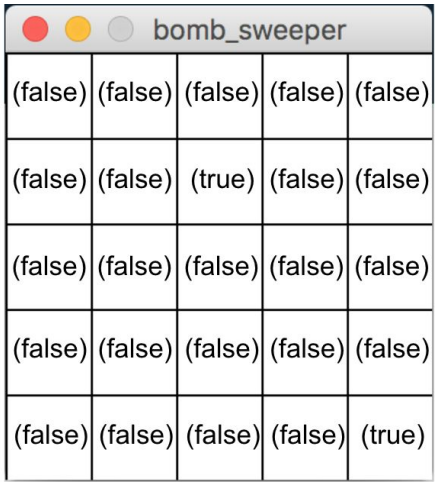
So far, we can detect which cell is clicked, and can store the state of each cell. So let's change the state of a cell by a mouse click. The idea is simple because we can know which row and column is clicked and can access the state of the cell by the clicked row and column. Type in the following:

```
void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  println("(" + row + ", " + column + ")");
  cells[row][column] = true;
}
```



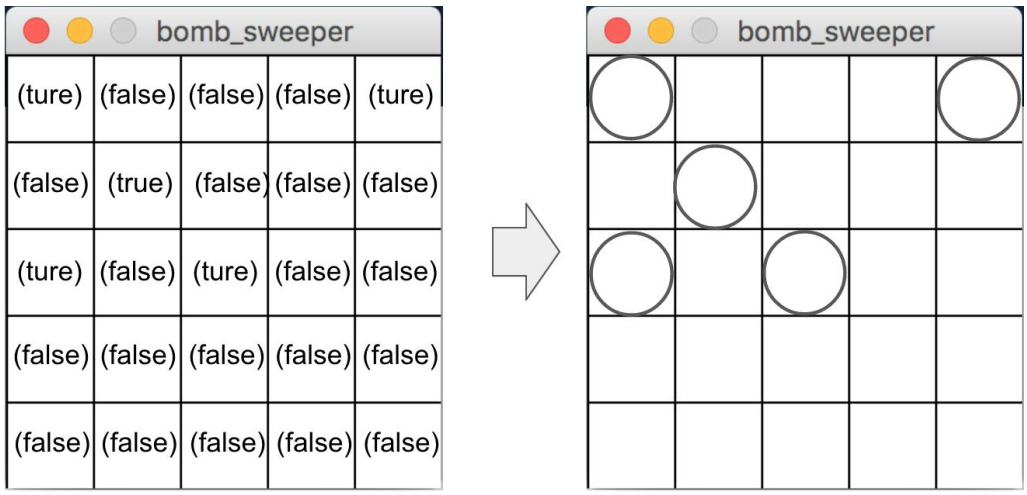
Here with "cells[row][column] = true;" part, we change the state of cell that has "row" and "column" to "true", which means the cell is marked.
For example, if you click (1, 2), and then do (4, 4), the states of the cells result in:





(7) Draw circle for marked cell

We are now ready to satisfy the requirement (2), to draw a circle when the cell is marked. The idea is to draw a circle if “cells[row][column]” is true. We can draw a circle by call a function named “ellipse”, provided by Processing software, by giving it x, y position, width and height of a circle. (https://processing.org/reference/ellipse_.html)



To represent “draw a circle if the state of a cell is “true”” in program, it is time to introduce “conditionals” as a concept.

Conditionals

A conditional statement could be used to select a block of codes to be executed by given conditions. You may be familiar with “if-then-rule” already, and “if” is one of the control statements.

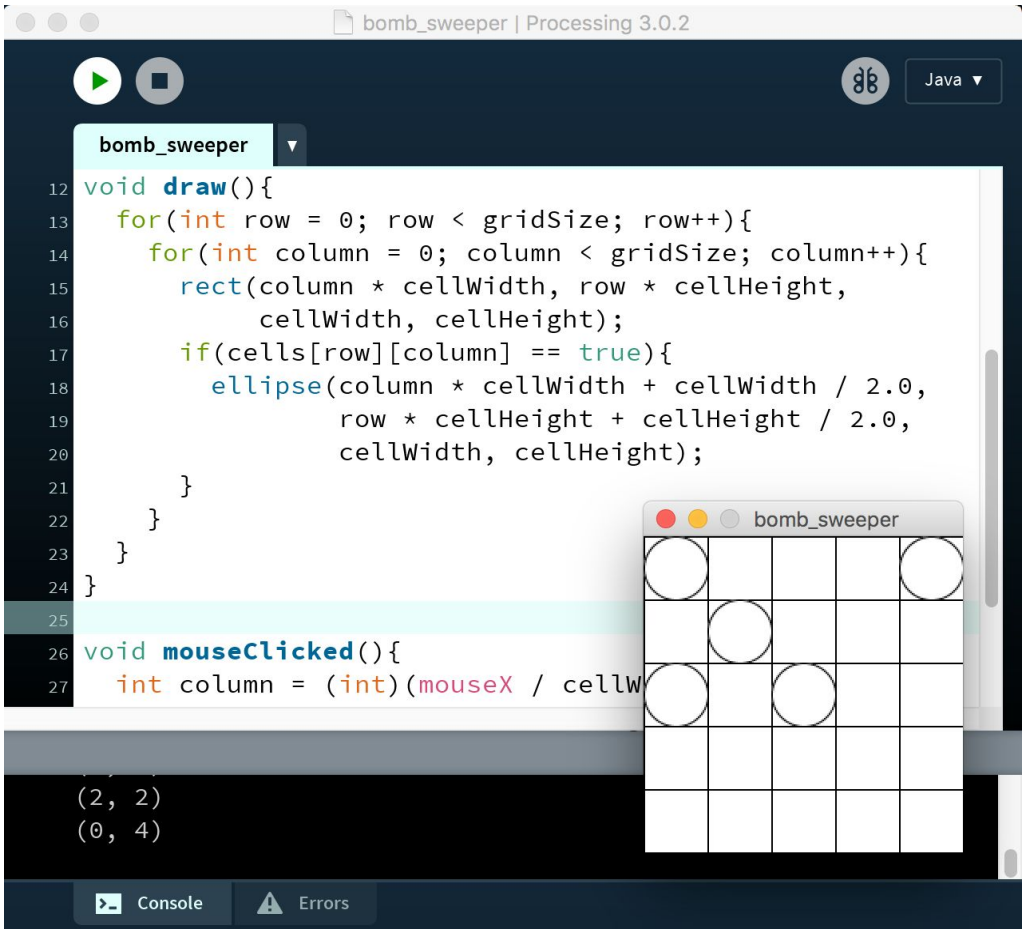
Here is an example:

```
int five = 5;

if(five == 5){
  println(“Yes”);
}else{
  println(“No”);
}

if(five < 0){
  println(“A”);
}else if(five == 0){
  println(“B”);
}else if(0 < five){
  println(“C”);
}else{
  println(“D”);
}

boolean isDone = false;

if(isDone != true){
  println(“AA”);
}else{
  println(“BB”);
}
```

Can you guess what the outputs printed in the console when you run the codes above?
“==” compares the left and right side, and becomes “true” when the two are the same; becomes “false” otherwise.
“!=” compares the left and right side, and becomes “true” when the two are not the same; becomes “false” otherwise.
We can also use things, such as “<”, “<=”, “>=”, “>”.
We can also use “&&” (AND) and “||” (OR) to deal with more than one conditions.
For example, “true && true” becomes “true”; “true && false” becomes “false”; “true || false” becomes “true”; “false || false” becomes “false”; with the example above “(five > 0) && (isDone)” becomes “false”; “(five > 0) && (!isDone)” becomes “true”; etc.

Back to the game creation, type in the followings:

```
void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
      if(cells[row][column] == true){
        ellipse(column * cellWidth + cellWidth / 2.0,
                row * cellHeight + cellHeight / 2.0,
                cellWidth, cellHeight);
      }
    }
  }
}
```

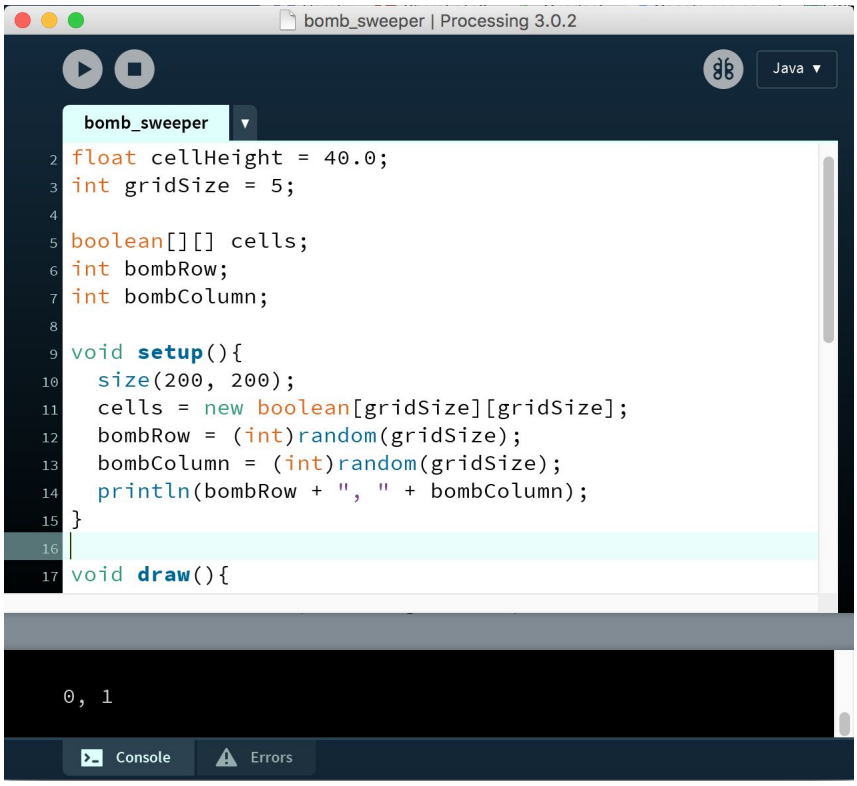


Hit the “Run” button, and click cells, and then you should see a circle for each clicked cell. Congraturation! Your program satisfy the requirement (2).

(8) Define a bomb

We have the last requirement (3). The first thing we need to do is to define a bomb. How can we achieve it? Let's think a bomb is at the specific row and column of the cells, "bombRow" and "bombColumn". We can simply say like "int bombRow = 2; int bombColumn = 3;", but in order to make the game more interesting, let the computer pick a row and column for the bomb. To do so, we can use a function "random" by giving it the highest number (which is exclusive), "gridSize". (https://processing.org/reference/random_.html)
Type in the followings:

```
int bombRow;
int bombColumn;

void setup(){
  size(200, 200);
  cells = new boolean[gridSize][gridSize];
  bombRow = (int)random(gridSize);
  bombColumn = (int)random(gridSize);
  println(bombRow + ", " + bombColumn);
}
```



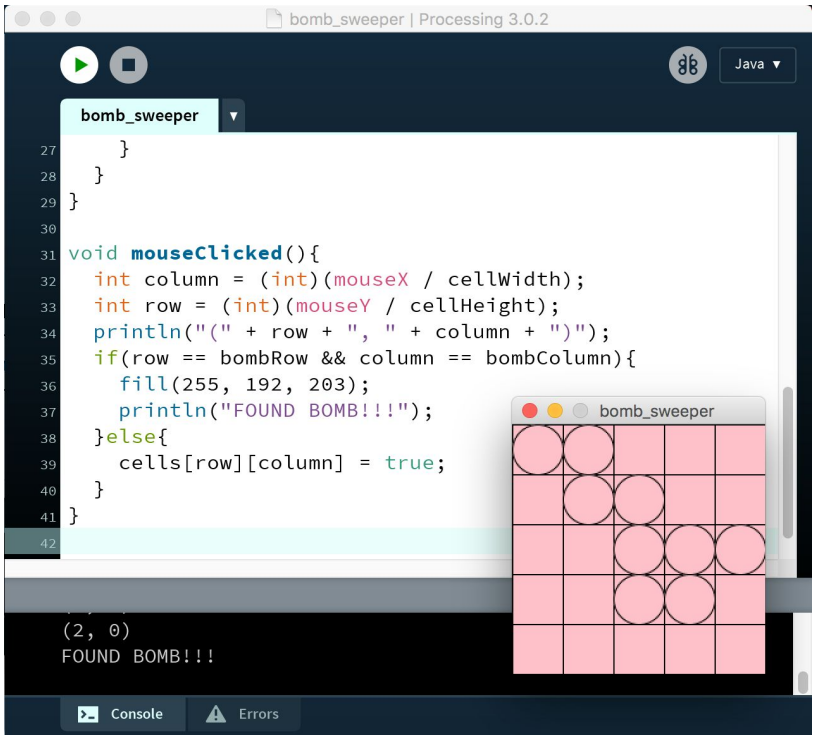
Because “gridSize” is set to 5, the “random” function chooses a number between 0 and 5 (5 is not included.)

(9) Change the window color to red when clicking a cell having the bomb

We are almost there. The rest for satisfying the requirement (3) is to change the window color to red when the bomb cell is clicked. We can use an “if” statement to check if the clicked cell is the bomb cell, and can use a function named “fill” by giving it a Red, Green, Blue value () to change a window color. (https://processing.org/reference/fill_.html)

Type in the followings:

```
void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  println("(" + row + ", " + column + ")");
  if(row == bombRow && column == bombColumn){
    fill(255, 192, 203);
    println("FOUND BOMB!!!");
  }else{
    cells[row][column] = true;
  }
}
```

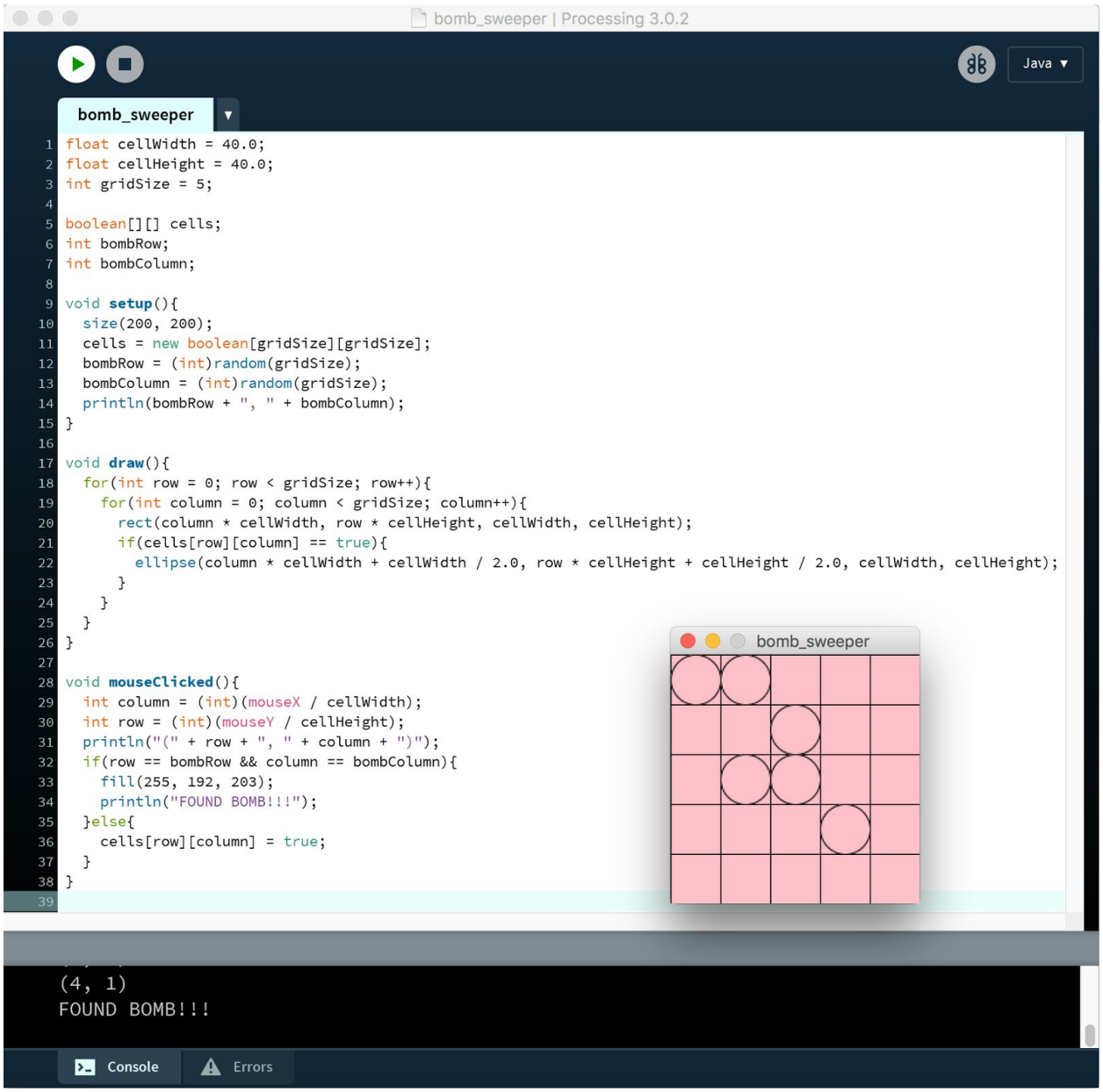


Congratulation! Your program now satisfies all of the three requirements. Play with your bomb sweeper game to make sure it works!

(9) Your program source code

```
float cellWidth = 40.0;
float cellHeight = 40.0;
int gridSize = 5;

boolean[][] cells;
int bombRow;
int bombColumn;

void setup(){
  size(200, 200);
  cells = new boolean[gridSize][gridSize];
  bombRow = (int)random(gridSize);
  bombColumn = (int)random(gridSize);
//  println(bombRow + ", " + bombColumn);
}

void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
      if(cells[row][column] == true){
        ellipse(column * cellWidth + cellWidth / 2.0,
                row * cellHeight + cellHeight / 2.0,
                cellWidth, cellHeight);
      }
    }
  }
}

void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
//  println("(" + row + ", " + column + ")");
```

```
  if(row == bombRow && column == bombColumn){
    fill(255, 192, 203);
    println("FOUND BOMB!!!");
  }else{
    cells[row][column] = true;
  }
}
```



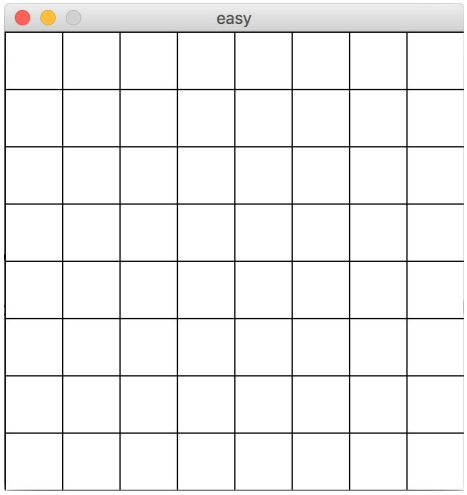
[6] Brush up your application (Programming challenges)

For the rest of the workshop, in order to make your game more sophisticated, we will introduce 3 programming challenges, each of them has a different challenge level. Feel free to talk with me, TAs or your peers to tackle the challenges. Are you ready?

[Easy]

Requirements:
Make your game to have 64 cells aligned as 8 by 8 grid where each cell size is 45.0 (width) and 45.0 (height).

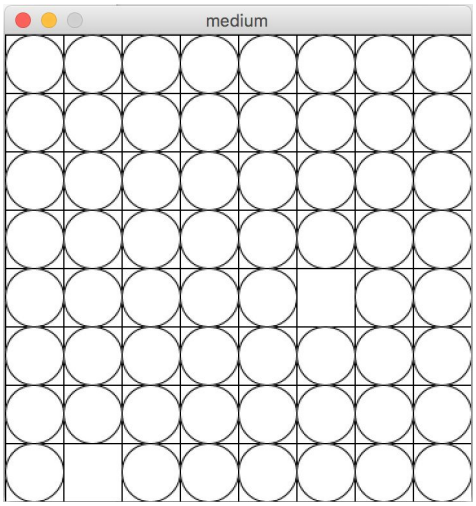
HInts:
Which variables do you need to change? If you change the number of and the size of cells, what do you also need to change to show off all of the cells?



[Medium]

Requirements:
Make your game to have 2 bombs. (It is ok that the program eventually sets the two bombs to the same cell if randomly picking up row and column numbers for the bombs.)

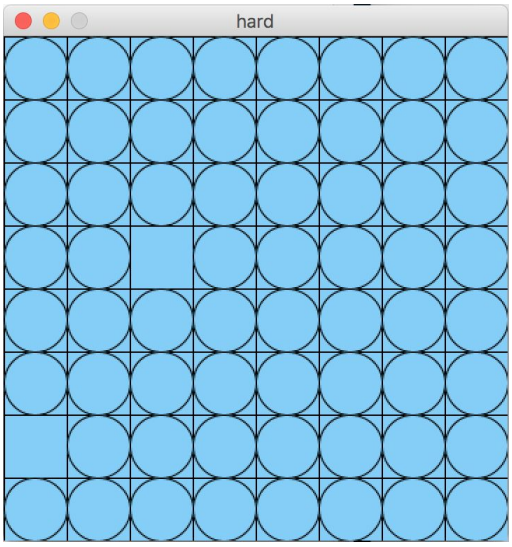
HInts:
What variable(s) do you need to add to represent the 2nd bomb? If we have 2 bombs, in which part of the codes do you also need to change to check whether a mouse click selected bomb1 or bomb2?



[Hard]

Requirements:
Make your game to change the window color to blue when all of the cells excepts ones having bombs are marked. (game clear)

Hint:
When can you say all of the cells excepts ones having bombs are marked? (You can check all values stored as “cells”, so create your own function to do so, and call it whenever you marked cells.)



[7] Evaluation form and class photo

Hope you find something fun in this workshop, or if you were able to sense what the programming looks like, it makes us feeling happy.

Lastly, we do appreciate your feedback for workshops by CSLOL. Please take a few minutes to fill out the evaluation form below.
http://goo.gl/forms/O6t2FbFokZ

Appendix - Source codes for the programming challenges

[Easy]

```
float cellWidth = 45.0;
float cellHeight = 45.0;
int gridSize = 8;

boolean[][] cells;
int bombRow;
int bombColumn;

void setup(){
  size(360, 360);
  cells = new boolean[gridSize][gridSize];
  bombRow = (int)random(gridSize);
  bombColumn = (int)random(gridSize);
  println(bombRow + ", " + bombColumn);
}

void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
      if(cells[row][column] == true)
```

```
        ellipse(column * cellWidth + cellWidth / 2.0,
                row * cellHeight + cellHeight / 2.0,
                cellWidth, cellHeight);
    }
  }
}

void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  if(row == bombRow && column == bombColumn){
    fill(255, 192, 203);
    println("FOUND BOMB!!!");
  }else{
    cells[row][column] = true;
  }
}
```

[Medium]

```
float cellWidth = 45.0;
float cellHeight = 45.0;
int gridSize = 8;

boolean[][] cells;
int bombRow1;
int bombColumn1;
int bombRow2;
int bombColumn2;

void setup(){
  size(360, 360);
  cells = new boolean[gridSize][gridSize];
  bombRow1 = (int)random(gridSize);
  bombColumn1 = (int)random(gridSize);
  println(bombRow1 + ", " + bombColumn1);
  bombRow2 = (int)random(gridSize);
  bombColumn2 = (int)random(gridSize);
  println(bombRow2 + ", " + bombColumn2);
```

```
}

void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
      if(cells[row][column] == true)
        ellipse(column * cellWidth + cellWidth / 2.0,
                row * cellHeight + cellHeight / 2.0,
                cellWidth, cellHeight);
    }
  }
}

void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  if((row == bombRow1 && column == bombColumn1) ||
     (row == bombRow2 && column == bombColumn2)){
    fill(255, 192, 203);
    println("FOUND BOMB!!!");
  }else{
    cells[row][column] = true;
  }
}
```

[Hard]

```
float cellWidth = 45.0;
float cellHeight = 45.0;
int gridSize = 8;

boolean[][] cells;
int bombRow1;
int bombColumn1;
int bombRow2;
int bombColumn2;
```

```
void setup(){
  size(360, 360);
  cells = new boolean[gridSize][gridSize];
  bombRow1 = (int)random(gridSize);
  bombColumn1 = (int)random(gridSize);
  println(bombRow1 + ", " + bombColumn1);
  bombRow2 = (int)random(gridSize);
  bombColumn2 = (int)random(gridSize);
  println(bombRow2 + ", " + bombColumn2);
}

void draw(){
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      rect(column * cellWidth, row * cellHeight,
           cellWidth, cellHeight);
      if(cells[row][column] == true)
        ellipse(column * cellWidth + cellWidth / 2.0,
                row * cellHeight + cellHeight / 2.0,
                cellWidth, cellHeight);
    }
  }
}

void mouseClicked(){
  int column = (int)(mouseX / cellWidth);
  int row = (int)(mouseY / cellHeight);
  if((row == bombRow1 && column == bombColumn1) ||
     (row == bombRow2 && column == bombColumn2)){
    fill(255, 192, 203);
    println("FOUND BOMB!!!");
  }else{
    cells[row][column] = true;
  }
  if(isCleared())
    fill(135, 206, 250);
}

boolean isCleared(){
```

```
  for(int row = 0; row < gridSize; row++){
    for(int column = 0; column < gridSize; column++){
      if(!(row == bombRow1 && column == bombColumn1) &&
         !(row == bombRow2 && column == bombColumn2)){
        if(cells[row][column] == false){
          return false;
        }
      }
    }
  }
  return true;
}
```