# Section 1: Setting up the Entry Point

1. Create a new project called Minesweeper. You do not need to select any additional libraries, but you should tick the “Create project from template” box to make it a command line app.
2. If you successfully ticked the “Create project from template” button in the first step you should have a project with a Main.java already created for you. If you do not have this, you should create a new Java file called Main.java. Inside this file you will have a public class Main. You can place a new line between the { and } then type main and press tab. This will auto-generate a main method. The code should look similar to what you see below.

public class Main {

public static void main(String[] args) {

}

}

1. Now create a second Java file called Game.java. This is where the main game logic will be in future sections. The next few steps relate to adding code to this file.
2. Create a private instance variable called scan of type Scanner. This will be used to read in all typed input.
3. Create a default constructor for the Game class. This constructor should initialise the scan variable by assigning it a new Scanner object with System.in as the input source.
4. Create a method called startGame() that returns nothing (void) and takes no parameters.
5. Inside the startGame() method write code to complete the following steps.
   1. Declare two int type variables. You can call these num1 and num2.
   2. Print out the message: "Enter space separated X then Y coordinate: ".
   3. Set num1 to the value returned by scan.nextInt().
   4. Do the same for num2.
   5. Print out a message showing "You entered: num1 num2" where num1 and num2 are the values you read in.
6. Return to Main.java and add the following two lines to the main method.

Game game = new Game();

game.startGame();

1. Run the program from Main.java and test that your code works so far. Try entering 1 2 and pressing enter. You should see the following. (1 2 are bold to show they are input).

Enter space separated X then Y coordinate:

**1 2**

You entered: 1 2

# Section 2: Creating a Board with Cells

1. Create a new class called Cell.java. This will represent the state of a single cell on the board. In later sections the full state information will be set up, but for now cells will simply have a state that is either revealed or not revealed.
2. Declare a private instance variable of type boolean called isRevealed.
3. Create a default constructor that takes no parameters. Inside this method set the value of isRevealed to false.
4. Create a public method called reveal that takes no parameters and returns nothing. This method should set isRevealed to true.
5. Create a public method called getIsRevealed that takes no parameters and returns a boolean value. Use this method to return the value stored in isRevealed.
6. Create a public toString method that takes no parameters and returns a String. Inside this method you should write code to do the following. This method will be used to determine what character to show representing the state of the cell.
   1. Check if isRevealed is true, and if it is then return “0” as a String.
   2. Otherwise return “\*”.
7. Create a new class called Board.java. This will manage a 2D array of cells.
8. Declare a private instance variable of type Cell as a 2D array (by writing [][]) called cells. This will be the grid of all cells we create and manage.
9. Declare a private int called width, and a private int called height. These will be used to remember the dimensions stored inside the array.
10. Create a constructor for Board that takes two parameters. An int width, and an int height. Follow the steps below to initialise this class.
    1. Assign the value from the width and height parameters into their respective instance variables. You can do this by writing for example: this.width = width;
    2. Initialise the cells array to be a new Cell array with width and height by writing:

cells = new Cell[width][height];

* 1. Write a for loop that initialises a variable y to 0, loops while y is less than height, and increments by 1.
  2. Inside this loop write another loop that initialises a variable x to 0, loops while x is less than width, and increments by 1.
  3. Inside the inner loop set the value of cells[x][y] to be a new Cell object.

1. Create a public method called printBoard that takes no parameters and returns nothing. This will print out the contents of the board. The following instructions explain the content of this method.
   1. Copy the nested for loop with y and x that you wrote in steps 19 c to d and paste it into printBoard.
   2. Instead of creating new cells you will need to print out the current value of the cell. We declared a toString method inside the Cell so it can be printed out by referencing the cell. Write a print (instead of println) method call that prints out cells[x][y] followed by a space (in quotes).
   3. Just after the inner for loop ends, but before the outer loop does you should add a System.out.println(); call by itself. This will add the newlines after each row.
2. Create a public method called revealCell that takes two int type variables called x and y and returns nothing. Inside this method just write the line. This will reveal a cell we specify.

cells[x][y].reveal();

1. Open Game.java again and the remaining steps of this section will relate to this file.
2. Declare a private instance variable of type Board called board.
3. Add a statement to the default constructor to initialise the board variable. Initialise board to a new Board type object and pass 10 and 10 as the width and height.
4. In the startGame method complete the following steps.
   1. At the start of the method add a call to board.printBoard();
   2. At the end of the method add a line calling the revealCell method in board passing num1 as the first parameter and num2 as the second parameter.
   3. Then on the line after you should call board.printBoard(); again.
5. Compile and run your program and again test with an input of 1 2. You should see output similar to the following.

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**1 2**

You entered: 1 2

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* 1 \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

# Section 3: Validating Input Part 1

1. Create a new Java class called Position.java. This class will be used to pass a coordinate with an x and y property around between different elements that need both pieces of information.
2. Inside the position class declare two public instance variables of type int called x and y.
3. Create a constructor for Position that takes two parameters of type int called x and y.
4. Inside the constructor store the value for x and y from the parameters into their respective instance variables. You can do this with this.x = x; for example.
5. Open Board.java.
6. Create a public method called validPosition that takes a single parameter of type Position called position and returns a boolean result.
   1. A valid position is one that matches with coordinates inside the bounds of the cells array. We already have a width and height variable, and arrays start at 0 so these can be used to check the validity.
   2. The method should return true if all the following conditions are true. If any are false, the result should be false.
      1. X coordinate is greater than or equal to 0.
      2. X coordinate is less than width.
      3. Y coordinate is greater than or equal to 0.
      4. Y coordinate is less than height.
7. Create a public method called isCellRevealed that takes one parameter of type Position called position and returns a boolean result. The method should simply return the result of cells[position.x][position.y].getIsRevealed().
8. Open Game.java. The rest of the work for this section will occur in this file.
9. Create a public method called isPositionInputValid that takes one parameter of type Position called position and returns a boolean result. The following steps should be completed inside this method.
   1. Write an if statement that checks if the position is not valid. You can use the ! character to get the not result and call the validPosition method in Board. The result of this if statement being true should be the text "Coordinate not inside the play space!" and then return false.
   2. Write another if statement after the previous one that checks using your board variable for the result of isCellRevealed. If this is true, you should print out the text "That cell is already revealed!" and then return false.
   3. Finally, if both the previous if statements were false, you should return true. This indicates the position is valid.
10. Create a public method called getPositionInput that returns an object of type Position. The following steps define what you should write for this method.
    1. Declare an object of type Position called input and initialise it with a new Position object set to 0,0. These numbers are just to initialise it and will be overwritten with user input.
    2. Write a do while loop as follows. This code will continue looping until a valid position has been entered.

do {

} while(!isPositionInputValid(input));

* 1. Inside the loop you should start by writing a statement to print out the text "Enter space separated X then Y coordinate: ".
  2. Set the value of input.x to scan.nextInt().
  3. Set the value of input.y to scan.nextInt().
  4. Finally, after the loop at the end of the method return the variable input.

1. As a last step before leaving the Board.java file. Go to your definition of revealCell and change the parameters from int x, int y to Position position. Then use position.x and position.y for the call to reveal.
2. Delete the contents of your startGame method as most of this is now handled inside the getPositionInput method.
3. Inside the startGame method add code to complete the following steps.
   1. Declare a variable of type Position called inputPosition.
   2. Write a do – while loop that has a loop condition set to true; written as while(true);
   3. Inside the loop you should call printBoard using your board object like you did previously.
   4. Then set the value of your inputPosition variable to the result of calling getPositionInput().
   5. Then call the revealCell method in your board object by passing inputPosition to it.
4. Compile and run your program. You should be able to continually enter pairs of x and y coordinates. If you enter numbers less than 0 or 10+ you should see your error message saying, “Coordinate not inside the play space!” and if you specify the same coordinate twice in a row it should output “That cell is already revealed!”. Test for both of these messages and that entering numbers between 0 and 9 for different positions on the grid make the correct cells reveal.

The following page shows some examples of output showing what you should expect to see with some example inputs.

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**-1 5**

Coordinate not inside the play space!

Enter space separated X then Y coordinate:

**5 5**

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**5 5**

That cell is already revealed!

Enter space separated X then Y coordinate:

**5 6**

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

# Section 4: Validating Input Part 2

1. Open Game.java. You may have tried to enter something that was not numbers in the previous testing and found it crashed your program. The changes in this section will make those crashes not possible, prepare for later steps where the user will be able to flag cells, and provide a way to quit at any time.
2. Create a public method called getStringOrQuit that takes no parameters and returns a String. The following steps should be completed inside this method.
   1. Declare a variable of type String called input and initialise it to scan.nextLine().trim(). The scan.nextLine() will retrieve the entire line of current input and trim() will remove any white space at the start/end of the line.
   2. Write an if statement that checks if input is the word “quit” you can do this by calling equalsIgnoreCase and supplying “quit” as the argument. This means it will be true regardless of how quit is capitalised.
   3. If the if statement is true, you should output a line saying “Thanks for playing! Goodbye!”. Then call System.exit(0) to terminate the application.
   4. After the if statement return input.
3. Modify the getPositionInput method with the following additional changes.
   1. After the line telling the user what to enter, but before the line setting input.x write an if statement that checks if scan.hasNextInt() is false. You can do this with a ! if you wish. If this if statement is true call getStringOrQuit(), output “Invalid X coordinate.”, and write continue; by itself on a line.

This will make sure that the line setting input.x will only be reached if the input has a valid int type value. And if it is not valid the input line is cleared with the call to getStringOrQuit() while at the same time checking if the input was just “quit”. Then the question is reasked to the user so they can enter a new coordinate because continue will return to the start of the loop.

* 1. Repeat the previous step between the lines that set input.x and input.y so that you can verify the input for the Y coordinate is a valid number. Of course, you will need to change the message to say “Invalid Y coordinate” for this, but it will otherwise be the same if statement block.
  2. After both input.x and input.y are set inside the loop add lines to subtract 1 from each of them. This means that users can input numbers from 1 to 10 instead of 0 to 9.
  3. In the next steps we will be making the numbers to input more obvious, so modify the previous message “Enter space separated X then Y coordinate: “ to say “Enter space separated X (bottom) then Y (right) coordinate: “.

1. Open Board.java.
2. Modify your printBoard method with the following changes.
   1. In the loop that prints out your board add an additional space into the loop to keep spacing working when numbering shows a 10 later on.
   2. Inside your outer loop you had a println() method call that had nothing in it. Modify this line so that it prints out “ |” followed by the value (y+1). Note that there is a space before the | character to give a space after the end of the row. This will show numbers down the right side so users can see the row number for the y coordinate.
   3. At the end of the method outside the loop write a new for loop. This for loop should loop for a value of x starting at 0, while x is less than width, and increment by 1. Inside the loop simply use print (not println) to print “\_ “ a \_ followed by two spaces. This will create a line of spaced-out dashes.
   4. After this for loop use a println call by itself to move to the next line.
   5. Write another for loop like the last one to start x at 0, loop while x is less than width, and increment by 1. Inside the loop you should print (x+1) followed by a space. Then an if statement that checks if x+1 is less than 10. If this is true, you should print out an additional space. This will keep the spacing the same for single and double digits.
   6. After the final loop write another println statement to move to the next line.
3. Compile and test your program. You should test entering letters for each of the X and Y coordinates one at a time. Test normal input still works. And finally test that quit works for one or more of the scenarios where it will. The output shown below is an example of what you should expect to see at this point.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**a 1**

Invalid X coordinate.

1 \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**1 a**

Invalid Y coordinate.

1 1 \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**7 7**

1 1 \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* 1 \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**quit**

Invalid X coordinate.

That cell is already revealed!

Enter space separated X (bottom) then Y (right) coordinate:

Thanks for playing!

# Section 5: Adding Bombs and Neighbour Counts

1. Open Cell.java. The next steps will prepare the Cell to handle the state of being a bomb or adjacent to a bomb.
2. Declare a private instance variable of type boolean called isBomb.
3. Declare a private instance variable of type int called neighbours.
4. Create a method called resetCell with no parameters that returns nothing. Move the line setting isRevealed to false into this method from the default constructor.
5. Still inside this method you should also set isBomb to false and neighbours to 0.
6. Inside the default constructor call resetCell() to have this method apply the default settings.
7. Create a method called setAsBomb with no parameters that returns nothing. Inside this method simply set isBomb to be true.
8. Create a method called getIsBomb with no parameters that returns a boolean value. Inside this method return the value stored in isBomb.
9. Create a method called addNeighbour with no parameters that returns nothing. The method should increase the value in neighbours by 1.
10. Create a method called getNeighbours with no parameters that returns an int value. The method should return the value stored in neighbours.
11. Modify your toString method with the following change.
    1. Inside the part of your if statement when isRevealed is true nest another if statement to check isBomb. If isBomb is true you should return a “B”. Otherwise, the value of neighbours added to an empty String. The empty String will convert the number to a String.
12. Open Board.java.
13. Create a method called isCellBomb that takes one parameter of type Position called position and returns a boolean value. This method should return the result of calling getIsRevealed in cells at position.x and position.y.
14. Create a method called addBomb that takes one parameter of type Position called position and returns a boolean. The following steps define what you need to include for this method.
    1. Write an if statement that calls isCellBomb passing the position as a parameter. If this is true, return false from this method. This is known as a method guard. You do not need to put the rest of the method in an else. This just prevents the method adding a bomb somewhere that already has a bomb.
    2. The next step is to calculate the minimum and maximum bounds to loop through for incrementing neighbour counts. If the position is at the edge the positions iterated over should not include invalid positions. These can be calculated using the Math.min and Math.max methods.
    3. Declare a variable of type int called minX and assign it the result of calling Math.max(0, position.x-1). This will make sure that the minimum value is larger than or equal to 0. Where the X coordinate is to the left of the position.
    4. Declare a variable of type int called maxX and assign it the result of calling Math.min(width-1, position.x+1). This either sets the position to the right of the original position or to the maximum possible value.
    5. Declare a variable of type int called minY and assign it the result of calling Math.max(0,position.y-1).
    6. Declare a variable of type int called maxY and assign it the result of calling Math.min(height-1, position.y+1).
    7. Now that the min and max values have been calculated create a for loop that sets a variable y to minY, loops while y is less than or equal to maxY and increments by 1.
    8. Inside this loop create another loop with a variable x that is set to minX, and loops while x is less than or equal to maxX and increments by 1.
    9. Inside this inner loop access the cell at [x][y] and call addNeighbour(). It does not matter that this also increases the count on the bomb because the toString method will always be showing either a \* or a B instead of the number once it is a bomb.
    10. After the loops access cells at position.x and position.y to call the setAsBomb method.
    11. Finally return true to indicate the bomb was created successfully.
15. Create a method called spawnBombs that takes one parameter of type int called maxBombs that returns nothing. The following steps walk through what you should do in this method.
    1. Declare a variable of type Random called rand and assign it a new Random() object.
    2. Create a for loop with a variable i that starts at 0 and loops while i is less than maxBombs and increments by 1.
    3. Inside the loop call addBomb and pass it a new Position object with parameters set to rand.nextInt(width) and rand.nextInt(height).
16. Create a method called revealAll that takes no parameters and returns nothing. This will allow easy testing to see the neighbour numbers are working properly and will be used to reveal the game when it ends. The following steps define what the method should do.
    1. Create a for loop that defines a variable y starting at 0, looping while less than height and incrementing by 1.
    2. Inside this for loop create another for loop that defines a variable x starting at 0, looping while less than width and incrementing by 1.
    3. Inside the inner loop you should access cells at [x][y] and call reveal.
17. Open Game.java.
18. In the constructor after the board has been initialised add a call to spawnBombs with a parameter 10.
19. Just after this line add another line to call revealAll. This will reveal all the cells so you can see if numbers appear correctly around each of the bombs.
20. Compile and run your program and you should see something similar to the below, except of course your bombs will be differently placed. You can see in the output below that all the cells that just have one B next to them have a 1 in them, those with two Bs next to them have a 2, and those with no Bs next to them have a 0. You may see higher numbers, and you could force this by spawning more bombs if you want to test easily.

2 B 1 0 0 1 1 1 1 B |1

B 2 1 0 0 1 B 1 1 1 |2

1 1 0 1 1 2 1 1 0 0 |3

0 0 0 1 B 1 0 0 1 1 |4

1 1 1 1 1 1 0 1 2 B |5

1 B 1 0 0 0 0 1 B 2 |6

1 1 1 0 0 0 1 2 2 1 |7

0 0 0 0 0 0 1 B 1 0 |8

0 0 1 1 1 0 1 1 1 0 |9

0 0 1 B 1 0 0 0 0 0 |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

1. Now remove the line you added in step 65 to revealAll. Compile and run your program again and test revealing a few cells to see if you can find a bomb.

# Section 6: Flagging Cells

1. Open Cell.java. (Make sure you didn’t forget to remove the call to revealAll in step 67).
2. Declare a private instance variable of type boolean called isFlagged.
3. In your resetCell() method assign the value false to isFlagged.
4. Create a public method called getIsFlagged with no parameters that returns a boolean value. In this method return the value stored in isFlagged.
5. Create a public method called toggleIsFlagged with no parameters that returns nothing. In this method set isFlagged to !isFlagged. This will swap the value between true and false depending on its current value.
6. Modify your toString method by adding an else if before the else containing the return “\*”. You need to check if isFlagged is true. If it is, you should return “F”. This value for “F” will appear if the cell is not revealed and is currently flagged.
7. Open Board.java.
8. Create a public method called isCellFlagged that has one parameter of type Position called position and returns a boolean value. The method should return a call to getIsFlagged from cells at [position.x][position.y].
9. Create a public method called flagCell with one parameter of type Position called position that returns nothing. Inside the method call toggleIsFlagged() referencing cells at [position.x][position.y].
10. Open Game.java.
11. Make the following changes to the startGame() method.
    1. At the start of the method declare a boolean called isFlagging.
    2. After the line where you store the value into inputPosition, add a line that stores the result of calling getStringOrQuit().equalsIgnoreCase(“flag”) into the isFlagging variable. This will set the isFlagging variable to true when the user enters something like “1 2 flag”.
    3. Write an if statement that checks if isFlagging is true. If it is true, call flagCell on your board object passing inputPosition as a parameter.
    4. Write an else if that checks the result of calling isCellFlagged in the board object using inputPosition as a parameter. If this is true, you should print out the message “You need to un-flag that cell first.”. This will prevent accidentally trying to reveal a cell that is currently flagged.
    5. Write an else to capture any other case. Move the call to board.revealCell(inputPosition) inside this else block.
12. Compile and run the program. You should test that the flagging mechanics are working. Test that cells can be flagged, un-flagged, when attempting to reveal a flagged cell it shows the error message, and that cells can still be revealed. The following couple of pages shows an example of expected output.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

5 5

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* 1 \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

7 7 flag

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* 1 \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* F \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

7 7

You need to un-flag that cell first.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* 1 \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* F \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

7 7 flag

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* 1 \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

# Section 7: Endgame Conditions

1. Open Board.java.
2. Declare a private instance variable of type int called bombCount.
3. Declare a private instance variable of type int called revealedTotal.
4. In the constructor initialise both bombCount and revealedTotal to 0.
5. In the addBomb method just before the return true at the end add a line that increases bombCount by 1.
6. In revealCell() before the line revealing the single cell increase revealedTotal by 1.
7. Create a public method called isWon that takes no parameters and returns a boolean result. In this method return the result of adding revealedTotal to bombCount and checking if the sum is equal to width\*height. This defines a game as being won when all cells have been revealed except for bombs.
8. Create a public method called printStatus that takes no parameters and returns nothing. This method should print out a line with a message showing “revealedTotal revealed of totalCells with bombCount bombs!”. The revealedTotal and bombCount are the variables in your class and totalCells can be calculated using width\*height.
9. Open Game.java.
10. Modify the startGame() method with the following changes.
    1. Just after the call to printBoard() add a call to printStatus() using the board object.
    2. Change the true in the while loop condition to the following.

!board.isWon() && (isFlagging || !boardisCellBomb(inputPosition))

This will keep the game looping when the win condition has not been met and the position was either not a bomb, or the position was being used as part of a flagging action.

* 1. After the end of the while loop add a call to revealAll() in the board object.
  2. Then call printBoard in the board object.
  3. Write an if statement that checks using the board object if isWon() is true.
  4. If it is true print out “Victory! You revealed all the non-bombs!”.
  5. Otherwise, print out “Boom! You hit a bomb! :(“.

1. Compile and run the game. You should test out these changes to make sure the game ends when you hit a bomb. At this point with revealing only one cell at a time it is probably not worth trying to test for a game win yet. Make sure that the status text is correctly printing with the number of revealed cells. The following page shows an example of some output.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

0 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

**5 5**

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* 1 \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

1 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

**6 6**

0 0 0 0 1 1 1 0 0 0 |1

1 2 2 1 1 B 1 0 0 0 |2

1 B B 1 1 1 1 0 0 0 |3

3 4 3 1 0 0 0 0 0 0 |4

B B 1 0 1 1 1 0 0 0 |5

2 2 1 0 2 B 2 0 0 0 |6

0 0 0 0 3 B 3 0 0 0 |7

1 1 0 0 2 B 2 0 0 0 |8

B 1 0 0 1 1 1 0 1 1 |9

1 1 0 0 0 0 0 0 1 B |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Boom! You hit a bomb! :(

# Section 8: Revealing Around a Point

1. Create a private method called revealAllAroundPoint with one parameter of type Position called position that returns nothing. Follow the steps below to complete the method.
   1. Start by going to your addBomb method and copying the code for the loop. Including the creation of the minX, maxX, minY, maxY, and nested loop.
   2. Paste the code into the revealAllAroundPoint method.
   3. Delete the line calling addNeighbour().
   4. Inside the inner loop write an if statement that checks if the cell at position [x][y] is not revealed. You can call getIsRevealed() and use a ! to invert the result. This condition prevents cells for being revealed that were already revealed
   5. If this if statement is true, call the reveal() method for cells at position [x][y] and increase revealedTotal by 1.
2. Modify the revealCell method with the following changes.
   1. At the start of the method write an if statement that checks the getNeighbours() for cells at [position.x][position.y] is not equal to 0.
   2. If this is true you should call the existing line to reveal the single cell and increase revealedTotal by 1.
   3. Write an else that calls revealAllAroundPoint() passing position as the parameter.
3. Compile and test the game. Now that you can reveal multiple points, try to win the game to test it can be won. The points should reveal all adjacent cells when you reveal a cell with 0 neighbouring bombs. The following example output shows the last steps to testing a victory.

80 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

8 2

0 1 1 1 0 0 0 0 0 \* |1

1 2 F 1 0 0 0 0 0 \* |2

F 2 1 1 0 0 1 1 2 \* |3

2 2 0 0 0 1 2 \* \* \* |4

F 2 1 0 0 1 F 2 2 1 |5

2 F 1 0 0 1 1 1 0 0 |6

1 1 1 0 0 1 1 1 0 0 |7

0 0 0 0 1 2 F 1 0 0 |8

0 1 1 1 1 F 2 1 0 0 |9

0 1 F 1 1 1 1 0 0 0 |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

86 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

10 2

0 1 1 1 0 0 0 0 0 0 |1

1 2 F 1 0 0 0 0 0 0 |2

F 2 1 1 0 0 1 1 2 1 |3

2 2 0 0 0 1 2 \* \* \* |4

F 2 1 0 0 1 F 2 2 1 |5

2 F 1 0 0 1 1 1 0 0 |6

1 1 1 0 0 1 1 1 0 0 |7

0 0 0 0 1 2 F 1 0 0 |8

0 1 1 1 1 F 2 1 0 0 |9

0 1 F 1 1 1 1 0 0 0 |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

89 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

8 4 flag

0 1 1 1 0 0 0 0 0 0 |1

1 2 F 1 0 0 0 0 0 0 |2

F 2 1 1 0 0 1 1 2 1 |3

2 2 0 0 0 1 2 F \* \* |4

F 2 1 0 0 1 F 2 2 1 |5

2 F 1 0 0 1 1 1 0 0 |6

1 1 1 0 0 1 1 1 0 0 |7

0 0 0 0 1 2 F 1 0 0 |8

0 1 1 1 1 F 2 1 0 0 |9

0 1 F 1 1 1 1 0 0 0 |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

89 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

9 4

0 1 1 1 0 0 0 0 0 0 |1

1 2 B 1 0 0 0 0 0 0 |2

B 2 1 1 0 0 1 1 2 1 |3

2 2 0 0 0 1 2 B 2 B |4

B 2 1 0 0 1 B 2 2 1 |5

2 B 1 0 0 1 1 1 0 0 |6

1 1 1 0 0 1 1 1 0 0 |7

0 0 0 0 1 2 B 1 0 0 |8

0 1 1 1 1 B 2 1 0 0 |9

0 1 B 1 1 1 1 0 0 0 |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Victory! You revealed all the non-bombs!

# Section 9: Flood Fill Revealing

1. The current method of revealing only a square around the chosen point is not how the traditional game works. When clicking a cell with no neighbouring bombs all connected cells with no neighbours should be revealed and a border of all cells around those cells. This is going to be accomplished in two passes. First by using what is called a flood fill algorithm to reveal all the connected 0s while remembering a list of all the cells that were revealed. Then iterating over every point and revealing all the points around using the method from the previous section.
2. Create a private method called checkFloodFillToCell that takes three parameters. A Position object called position, a boolean[][] array called vis, and a Queue<Position> object called positionQueue. The position is the position in cells to evaluate as a possible floodFill target. The vis array is a visited matrix that will contain false for positions not yet visited and false for positions that have been visited. The positionQueue is a queue (first in, first out data structure) that is used to step over all the positions when performing a floodFill. The following describes that you need to include in this method.
   1. Write an if statement that uses the validPosition method to check if position is valid.
   2. If this is true, write another if statement that checks the following conditions:
      1. vis[position.x][position.y] is false.
      2. AND isCellRevealed() is false using position as a parameter.
      3. AND cells[position.x][position.y].getNeightbours() is equal to 0.
   3. If this is true, call positionQueue.add(position).
   4. After the inner if statement set vis at [position.x][position.y] to true.
3. Create a private method called floodFillReveal that takes one parameter of type Position called position and returns a List<Position> as the result. The following steps define what the method should do.
   1. Declare a variable of type boolean[][] called vis and assign it a new boolean array with dimensions [width][height].
   2. Write a for loop that iterates over every position in the vis array. You need to write a loop with x starting at 0, looping while x is less than width, incrementing by 1. Then a nested for loop that uses y starting at 0, looping while y is less than height, incrementing by 1. Inside the inner loop set the value of vis at [x][y] to false.
   3. Declare a variable of type List<Position> called changedPoints and initialise it to a new ArrayList<>().
   4. Declare a variable of type Queue<Position> called positionQueue and initialise it to a new LinkedList<>().
   5. Add the first position to the queue by writing positionQueue.add(position).
   6. Set vis at [position.x][position.y] to true.
   7. Write a while loop that loops while positionQueue.isEmpty() is false (you can use the ! symbol to flip the result).
   8. Declare a variable of type Position called positionToReveal and assign it the result of calling positionQueue.remove(). This will take the first element of the queue and remove it storing the result into positionToReveal.
   9. Call reveal() using cells at [positionToReveal.x][positionToReveal.y].
   10. Increase the value of revealedTotal by 1.
   11. Add the position to the changedPoints list by writing changedPoints.add(positionToReveal).
   12. The next four lines will be very similar but are important to write correctly. You need to use the checkFloodFillToCell method to validate whether the specified positions should be added to the queue. The structure of the method calls will appear as:

checkFloodFillToCell(position\_value\_here, vis, positionQueue);

You will need to replace position\_value\_here with creation of a new Position object where the positions are defined as:

* + 1. Up: positionToReveal.x+1, positionToReveal.y
    2. Down: positionToReveal.x-1, positionToReveal.y
    3. Right: positionToReveal.x, positionToReveal.y+1
    4. Left: positionToReveal.x, positionToReveal.y-1
  1. Finally, after the loop return changedPoints.

1. Modify the revealCell method with the following changes.
   1. In the else, declare a variable of type List<Position> called revealedCells and assign it the result of calling floodFillReveal(position).
   2. Write a for each loop as follows. This will loop for each value in revealedCells and each iteration of the loop the value of p will be updated to the next element.

for(Position p : revealedCells) {

}

* 1. Inside the for each loop put the revealAllAroundPoint method call but change the parameter to p representing the point from revealedCells.

1. Compile and run the game and test the flood fill algorithm is working properly. If you find it is not working properly go back through and make sure the variables are all correct. It is easy to accidentally write position instead of positionToReveal or check for a true instead of a false for example. The following example output over page shows an example of what should happen when flood filling correctly.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

0 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

5 5

\* 1 0 1 \* 1 0 0 0 0 |1

\* 2 0 1 \* 1 0 0 0 0 |2

\* 2 0 1 1 1 0 0 0 0 |3

1 1 0 0 0 0 0 0 0 0 |4

0 0 0 0 0 0 0 0 0 0 |5

0 0 0 0 1 1 1 0 0 0 |6

0 0 0 0 2 \* 2 0 1 1 |7

1 2 1 1 2 \* 3 1 1 \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

72 revealed of 100 with 10 bombs!

Enter space separated X (bottom) then Y (right) coordinate:

# Section 10: Adding Some Colour

1. Open Cell.java and add the following instance variable definitions. These are parts you can include as part of Strings to add colour to everything after the code appears up until a reset code is received. The next step will define a way to combine these easily with Strings and then we will create a modified method to return a coloured String.

private static final String ANSI\_RESET = "\u001B[0m";

private static final String ANSI\_RED = "\u001B[31m";

private static final String ANSI\_GREEN = "\u001B[32m";

private static final String ANSI\_YELLOW = "\u001B[33m";

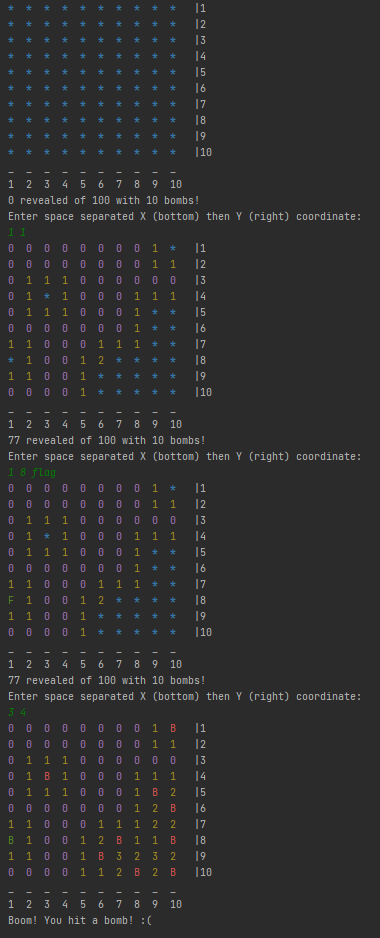
private static final String ANSI\_BLUE = "\u001B[34m";

private static final String ANSI\_PURPLE = "\u001B[35m";

1. Create a private method called colourString that takes two parameters, a String called str, and a String called colourFlag. The method should return a String value. You should include the following return line inside the method. This will create a String by combining the colour code with the preferred String and then resetting back to the default colour.

return colourFlag + str + ANSI\_RESET;

1. Create a public method called getColouredString that takes no parameters and returns a String value. The following steps define what this method should do to use the correct colours.
   1. Declare a variable of type String called str and set it equal to the result of calling toString().
   2. Write an if statement that is true when isFlagged is true. When true, return the result of calling colourString by passing str and ANSI\_GREEN to it.
   3. Write an else if statement that is true when isRevealed is true. When this is true you should check each of the following inside it.
      1. If isBomb is true, return the result of calling colourString with str and ANSI\_RED.
      2. Else if neighbours is equal to 0, return the result of calling colourString with str and ANSI\_PURPLE.
      3. Else return the result of calling colourString with str and ANSI\_YELLOW.
   4. Write a final else statement at the end of the method that returns the result of calling colourString with str and ANSI\_BLUE.
2. Open Board.java.
3. Modify the printBoard method where you print out cells[x][y] by making it call cells[x][y].getColouredString().
4. Compile and run the program. You should test that all the colours are correctly appearing as expected. You can see an example of output showing the different colours you should expect to see over the page. Anything not yet revealed is green if flagged or blue if not. Anything revealed is purple if it is a 0, yellow if it is any other number, red if it is a bomb, unless that bomb was flagged before revealing (then it is still shown as green).



# Section 11: Initial Setup for a GUI

1. Open Board.java.
2. Create a public method called getWidth with no parameters that returns an int value. Return the value stored in width.
3. Create a public method called getHeight with no parameters that returns an int value. Return the value stored in height.
4. Create a new Java file called BoardPanel.java.
5. After “public class BoardPanel” add “extends JPanel”. This will turn this class into something that can be used as part of a GUI.
6. Declare a private instance variable of type int called CELL\_SIZE and initialise it to 32.
7. Declare a private instance variable of type Board called board.
8. Create a constructor with two parameters. A variable of type int called boardWidth, and a variable of type int called boardHeight. Add the following lines to the method.
   1. Initialise the variable board by creating a new Board object by passing boardWidth and boardHeight.
   2. Using the board object call spawnBombs() with a parameter of 10.
   3. Call setPreferredSize() by writing the following line.

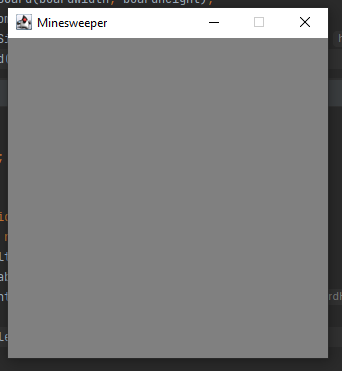
setPreferredSize(new Dimension( CELL\_SIZE \* boardWidth, CELL\_SIZE \* boardHeight));

* 1. Call setBackground() and pass Color.gray as the parameter.

1. Open Main.java.
2. In a later section there will be an option popup to choose between running the text based (CLI) version of the game that has just been completed and the GUI version that is starting now.

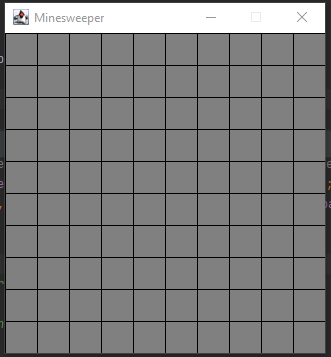
For now, comment out the lines that create a Game object and start the game.

1. Add the following to the main method.
   1. Declare a variable of type JFrame called gui and initialise it to a new JFrame object. Pass “Minesweeper” as the parameter to set the title of the frame.
   2. Using the gui object, call setDefaultCloseOperation() and pass JFrame.EXIT\_ON\_CLOSE as the parameter. This will make the program close when the window is closed.
   3. Using the gui object, call setResizable() and pass false as the parameter. This will lock the size of the frame.
   4. Using the gui object, call getContentPane().add(new BoardPanel(10,10)). This will create the BoardPanel with a 10 by 10 grid and add it to the JFrame.
   5. Using the gui object, call pack(). This will automatically size everything correctly based on the definition in BoardPanel for preferredSize.
   6. Using the gui object, call setVisible and pass true as the parameter. This will make the GUI visible and ready to use.
2. Compile and run the program. You should see a window appear with a grey background and nothing in it as seen below.



# Section 12: Drawing a Grid of Lines

1. Open BoardPanel.java.
2. Create a private method called drawGrid with one parameter of type Graphics called g that returns nothing. The following steps define what the method should do.
   1. Using the object g, call setColour and pass it a parameter Color.BLACK.
   2. Declare a variable of type int called y2 and initialise it to 0.
   3. Declare a variable of type int called y1 and initialise it to the result of calling getHeight() using the board object and multiplying it by CELL\_SIZE.
   4. Create a for loop that creates a variable x and initialises it to 0, loops while x is less than getWidth() from the board object and increments by 1.
   5. Inside the loop use the object g and call drawLine passing the parameters:
      1. x \* CELL\_SIZE
      2. y1
      3. x \* CELL\_SIZE
      4. y2
   6. After the loop, declare a variable of type int called x2 and initialise it to 0.
   7. Declare a variable called x1 and initialise it to the result of calling getWidth() using the board object and multiplying it by CELL\_SIZE.
   8. Create a loop that creates a variable y and initialises it to 0, loops while y is less than getHeight() from the board object and increments by 1.
   9. Inside the loop use the object g and call drawLine passing the parameters:
      1. x1
      2. y \* CELL\_SIZE
      3. x2
      4. y \* CELL\_SIZE
3. Create a public method called paint that takes one parameter of type Graphics called g that returns nothing. This will override the default implementation of drawing in the panel. Add the following statements to the paint method.
   1. Call super.paint(g) to make the background draw.
   2. Call drawGrid() passing g as a parameter.
4. Compile and run the program. You should see something like below with a grid of lines visible.



# Section 13: Drawing Elements on the Grid

1. Open Cell.java.
2. Create a private method called getColourForCell that takes no parameters and returns a Color object. This will be used to determine the colour of text drawn into the cells in a form that can be used for rendering. The following steps define what the method should do.
   1. Write if statements to complete this sequence of requirements with resulting return statements.
      1. If isBomb is true AND isFlagged is true, return Color.yellow.
      2. If isBomb is true AND isFlagged is false, return Color.red.
      3. If neighbours is 1, return Color.blue.
      4. If neighbours is 2, return Color.green.
      5. If neighbours is 3, return Color.red.
      6. If neighbours is 4, return new Color(128,0,128).
      7. If neighbours is 5, return new Color(128,0,0).
      8. If neighbours is 6, return new Color(0,0,128).
      9. If neighbours is 7, return Color.blue.
      10. Otherwise, return Color.black.
3. Create a public method called drawCell that takes four parameters. A Graphics object called g, an int called x, an int called y, and an int called cellSize. The x and y will be offsets to the top left corner of the cell, and cellSize is used to calculate some of the drawing. The method should follow the steps below.
   1. Write an if statement that checks if isRevealed is true. In the case it is true, execute the following statements.
      1. Write an if statement that checks if neighbours is 0. If it is true, write return with nothing after it to exit this method. Nothing should be drawn for empty cells that are revealed.
      2. After the above if statement, using the Graphics object g, call setColor() passing getColourForCell() as the parameter. This will set the colour to the correct value before drawing the text.
      3. Then also using the object g, call drawString. Passing the parameters as follow.

g.drawString(toString(), x+10, y+21);

* 1. Otherwise, if isRevealed is false (else) follow the statements below.
     1. Write an if statement to check if isFlagged is true. If it is, use the variable g to call setColor and pass Color.yellow as a parameter.
     2. Write an else that sets the colour to Color.darkGray instead (for when isFlagged is false).
     3. After the else write the following line of code to draw a rectangle that is a little smaller than the grid size.

g.fillRect(x+3, y+3, cellSize - 5, cellSize - 5);

1. Open Board.java.
2. Create a public method called drawboard that takes two parameters. A Graphics object called g, and an int variable called cellSize. Inside the method include the following.
   1. Write a for loop that starts with x initialised to 0, loops while x is less than width, and increments x by 1.
   2. Inside the loop write another for loop that starts with y initialised to 0, loops while y is less than height, and increments by 1.
   3. In the inner for loop call the drawCell method in cells at [x][y] with the parameters:
      1. g
      2. x \* cellSize
      3. y \* cellSize
      4. cellSize
3. Open BoardPanel.java.
4. Modify the paint() method. After the call to drawGrid(g), write another line that uses the board object to call drawBoard and pass g and CELL\_SIZE as the parameters.
5. Compile and run the program. At the moment you won’t yet be able to see most of the result of this section. After the next section these will all be testable. You should for now see something similar to below.

