Name ____ Role __________

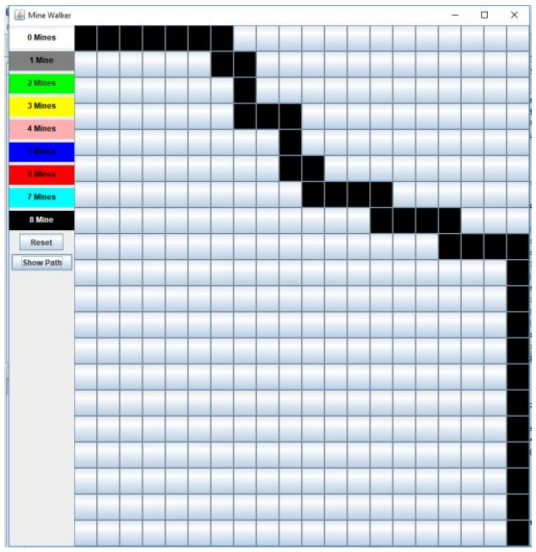
Mine Hunter Path

Your Tasks (Mark these off as you go)

- ☐ Review the MineHunterPath class
- ☐ Initialize an ArrayList of Points
- ☐ Write the getNext method
- ☐ Write the step method
- ☐ Write the isDone method
- ☐ Write the reset method
- ☐ Write the showPath method
- ☐ Receive credit for this lab guide

☐ Review the MineHunterPath class

In this lab you will build upon the Mine Hunter program you wrote previously. The goal of this lab is to write a class that when implemented will find a path through the mines on the grid like the picture shown below.



Below is the MineHunterPath you will complete. Notice we use and ArrayList to store the points.

MineHunterPath.java

```java
public class MineHunterPath {

    private int gridDimensions;
    private Point start;
    private Point end;
    private Point current;
    private ArrayList<Point> path = new ArrayList<Point>();
    private boolean done;
    private boolean[][] mines;
    private int x, y;
```

```
        public MineHunterPath(int gridDimensions, boolean[][] mines){

            this.mines = mines;
            this.gridDimensions = gridDimensions;
            this.start = new Point(0,0);
            this.end = new Point(gridDimensions-1, gridDimensions-1);
            this.current = start;
            this.done = false;
            path.add(start);

        }

        /**
         * Adds the next available tile to the path ArrayList
         * step continues until isDone is true
         * If no tile is available, the grid will reset
         */
        private void step(){}
        /**
         * sets the grid back to the starting point and
         * clears the path ArrayList
         */
        private void reset(){}

        /**
         * Randomly locates the next available tile in the right
         * or down position.
         * Tiles that contain Mines are not available
         * @return the next tile as a Point on the grid
         */

        private Point getNext(){return new Point(x,y);}

        /**
         * returns the path of points
         * @return
         */
        public ArrayList<Point> getPath(){
             step();
             return path;
        }

        /**
         * checks if the end of the grid has been reached
         * @return whether or not the end is reached
         */
        private boolean isDone(){return false;}
}
```

☐ Initialize an ArrayList of Points

To create the path, you can assume that the path begins at coordinates x = 0 and y = 0 and ends at x = gridDimensions -1 and y = gridDimensions – 1. The path your program creates will be random, that is, you will generate a random number and depending on the number, the path will proceed forward or down. But the path can NEVER hit a mine. Because we do not know how long the path will be, we will store the coordinates of the path in an ArrayList. The ArrayList you create will hold Point values. Point values are convenient because we can use the points to represent x, y coordinates on the grid.

Write code that could be used to declare an ArrayList called path which stores Point type objects.
`ArrayList<Point> path = new ArrayList<Point>();`

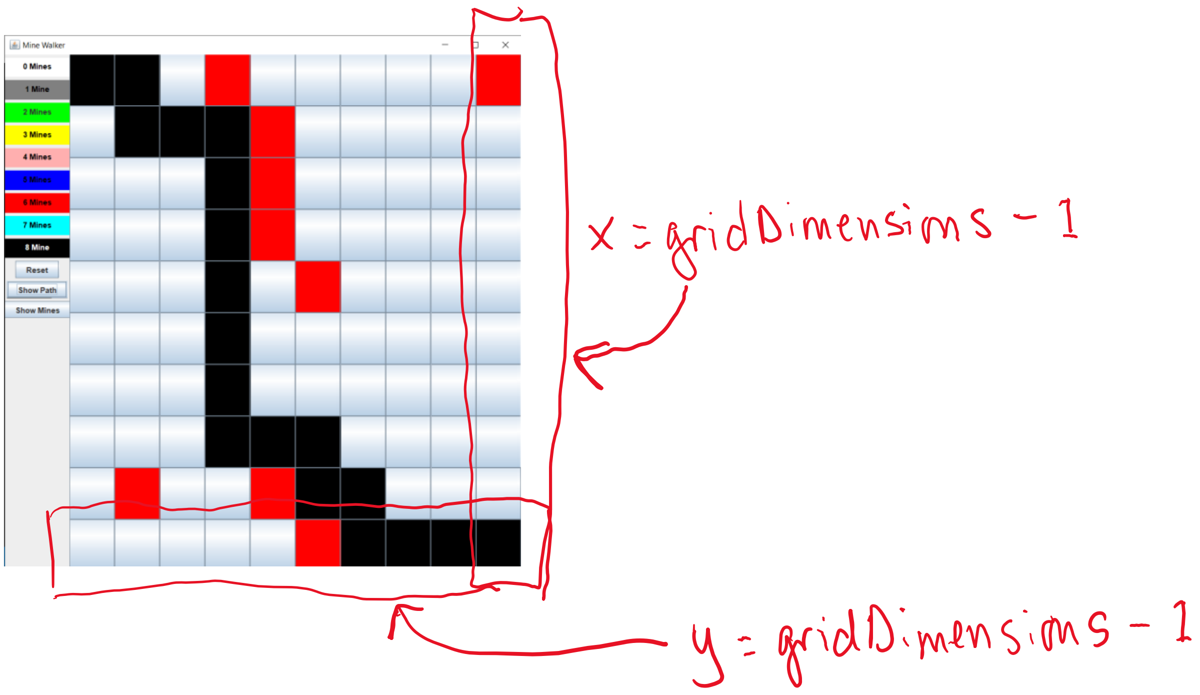
☐ Write the getNext method

The getNext() method randomly locates and returns the next available Point in the right or down position. Tiles that contain a mine are not available. That is, locations where mines[row][col] == true are not available.

The signature for the getNext method is shown below,

```
private Point getNext(){
     return new Point(x,y);
}
```

When attempting to get the next location, you should also be mindful of boundaries for example,



If for example, x = gridDimensions – 1, you can only go down. If y = gridDimensions – 1 you can only go forward, etc.

Your current location on the grid is stored in the ArrayList you declared previously. It is the last location that was added to the ArrayList,

```
current = path.get(path.size()-1);
x = current.x;
y = current.y;
```

Write the getNext method below. The getNext method does the following (Not necessarily in the order written)

(1) Checks to see if there is an available move in the right or down position, that is mines[x][y] for the next move is false.
(2) If there is no available move, reset() is called. You will write the reset method below.
(3) If there is an available move, getNext() generates a random number to determine whether the next move should be in the right or down direction.
(4) Next, getNext should check to see if the next location contains a mine. If the next location contains a mine, step (3) continues
(5) Once the next location is found, it is returned as a point

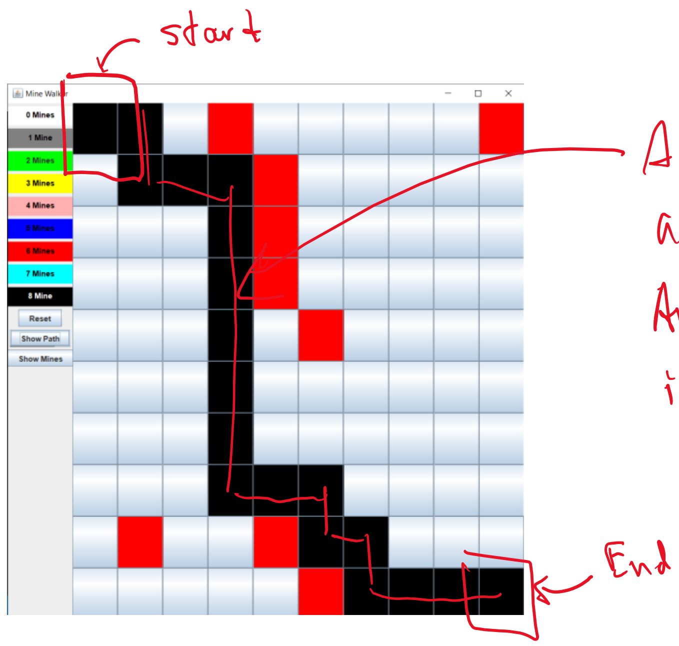
Be mindful of boundary conditions!

```
//top left corner
Point next = null;
if(x ==0 && y == 0){
       if(mines[x + 1][y] & mines[x][ y + 1]{
              reset();
       }
       if(mines[x + 1][y]){
           next.x = x;
           next.y = y + 1;
        }else if(minex[x][y + 1]){
           next.y = y;
           next.x = x + 1;
        }else{
             if(Math.random() < .5){
                    next.x = x;
                    next.y = y + 1;
             }else{
                    next.x = x + 1;
                    next.y = y;
              }
       }

}
return next;
```

☐ Write the step method

Step() is void type method, which adds a point to our array if we have a location to move to. It continues to add points until our destination is reached. This is illustrated below,



All points of the path are added to the path ArrayList until the end is reached

Write the step method. The step method does the following (not necessarily in the order written)

(1) Checks to see if the end is reached. That isDone() returns true. You will write the isDone() method below.
(2) If isDone() is false, getNext() is called.
(3) The point returned from getNext() is added to our path.

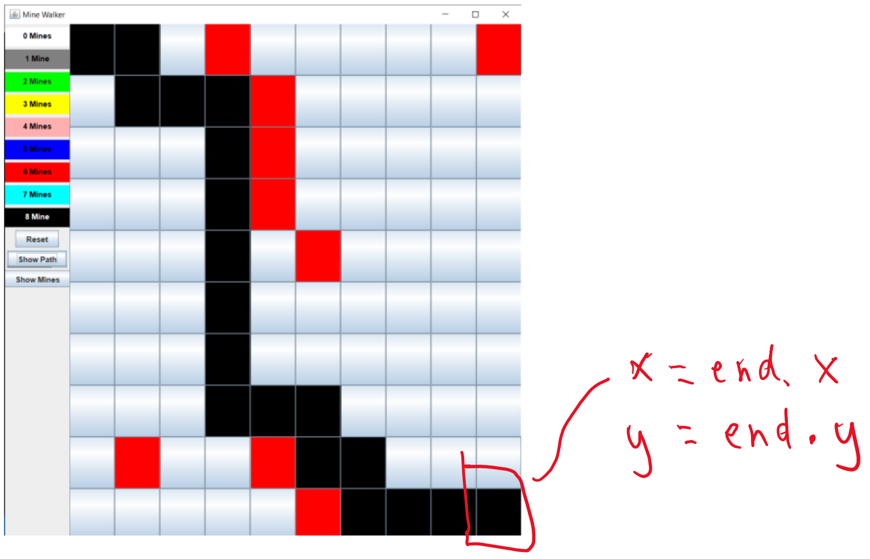
```
while(!isDone()){
        path.add(getNext());
 }
```

☐ Write the isDone method

The ending point on the grid is defined as follows.

```
end = new Point(gridDimensions-1, gridDimensions-1);
```

How to access values of x and y associated with this point is illustrated below.

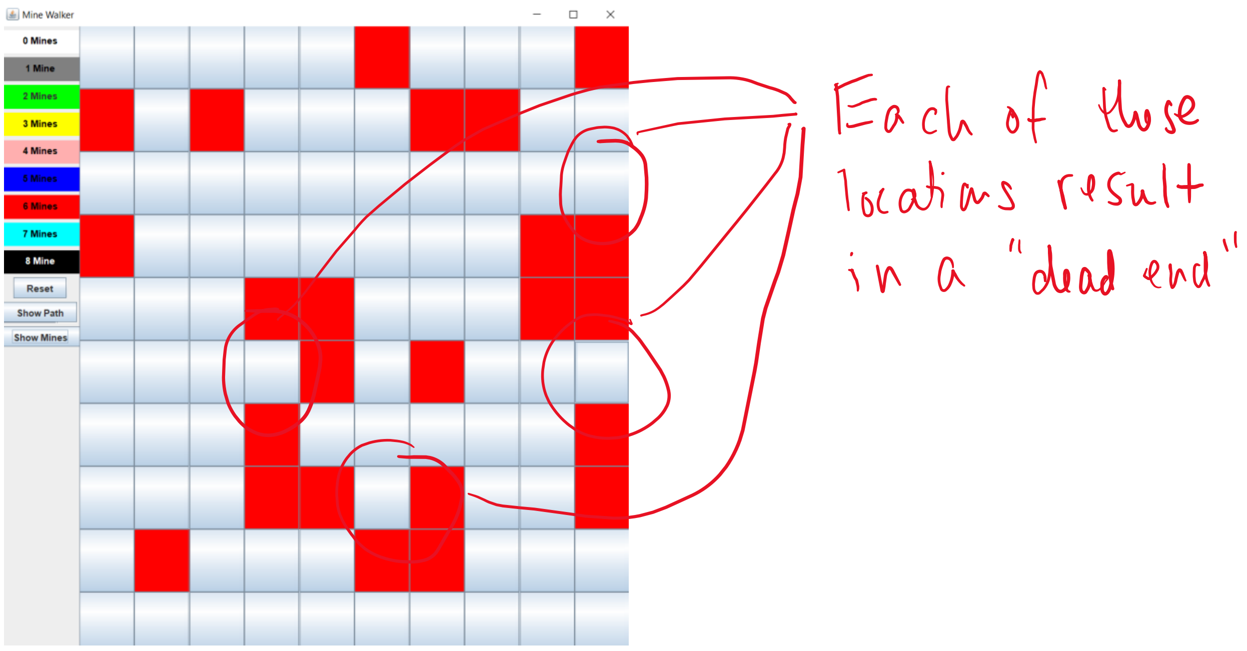


Write the isDone() method. isDone is a boolean type method that returns true if the conditions illustrated above are true, otherwise it returns false.

```
if(x = end.x && y == end.y){
    return true;
}
return false;
```

☐ Write the reset method

The reset method mentioned previously is called whenever the path reaches a dead end. Examples are illustrated below.



Write the reset() method. The reset() method does the following (not necessarily in the order written)

(1) Resets x and y to 0
(2) Clears the path
(3) Adds the start point. Note the start point is defined in the constructor as show below. start doesn't need to be redefined. Just added to the path.

```
        start = new Point(0,0);
```

```
x = 0;
y = 0;
path.clear();
path.add(start);
```

☐ Write the showPath method

The job of the showPath method is to display the path stored in the path ArrayList when the show path button is clicked.

Write the showPath() method below. The showPath() method does the following,

(1) Creates a new instance of the MineHunterPath,

```
MineWalkerPath newPath = new MineWalkerPath(gridDimensions, mines);
```

(2) Creates a path and assigns the new path to the path variable,

```
this.path = newPath.getPath();
```

(3) Iterate over the path above and access the tiles located at each x and y location. Color the background of tiles associated with each Point. How to access the x and y value of a Point in the path array is illustrated below,

```
Point somePoint = path.get(0);
int x = somePoint.x;
int y - somePoint.y;
```

```
MineHunterPath newPath = new
MineHunterPath(gridDimensions, mines);
   this.path = newPath.getPath();
   for(Point p : path){
     tiles[p.x][p.y].setBackground(Color.BLACK);
   }
```

☐ Receive Credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.