Make Class Tile {

Use encapsulation for the following variables:

tileColor and tileType of type String.

tileColumn and tileRow of type Integer.

Constructor/Initializer: Tile (Col and Row of type Integer) {

Assign the value in Col to tileColumn.

Assign the value from row to tileRow.

Choose a random color for tileColor.

Choose which type of tile is tileType, with tileType having a high chance of “normal” and low chance of “bomb”.

}

Getters and setters for variables.

}

Make class Board {

declare variable: minNumOfMatches of type Integer.

Declare a two-dimensional array named boardTiles of type Tile.

Declare a Hashmap named tileMatches.

Constructor/Initializer: Board (boardSize\_Row, boardSize\_Column and minMatches of type Integer.) {

Initialize the size of the array boardTiles using both boardSize\_Row and boardSize\_Column.

Assign the value in minMatches to minNumOfMatches.

Use a while loop to call the method createBoard as many times until isMatchAvilable method results in True.

}

Method/Function: createBoard () {

Use a nested loop to initialize tiles by column and row. Use the first loop to represent the columns and the second loop for rows.

}

Boolean Method/Function: isMatchAvilable () {

Declare variable: match of type integer.

Loop though Tiles with comparing the tiles by color and store the biggest number of matches into match.

If (match is bigger or equal to minNumOfMatches)

Return True.

Else

Return False.

}

Method/Function: updateBoard () {

Loop through each column of the boardTiles array and push down all tiles that has no tiles under them.

Using a nested loop for all empty slots in boardTiles array fill them with new Tile objects.

}

Method/Function: checkClickedTile () {

Use recursion to check for matches for the tile clicked and add those matches to tileMatches.

If (tileMatches has matches) {

Delete those tiles.

Call method updateBoard ().

}

}