**Minefield – Task 1**

The first part of this task is to construct the game space and the infrastructure on which it runs. In other words, set up the Form, along with the GUI objects that need to appear upon it. I decided this game will use label objects as its main building blocks, upon which the placement and movement of a sprite, the locations of bombs, and the depiction of areas that have already been traversed, are all presented. The labels must be positioned carefully so that they form a grid. Each grid cell is represented by a single label object.

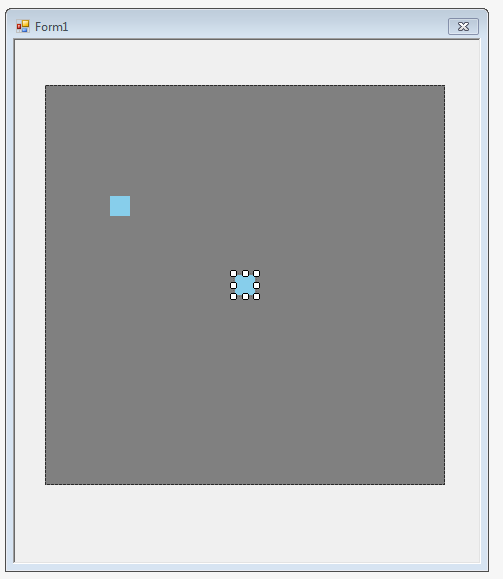
Immediately, some key design decisions must be made; such as:

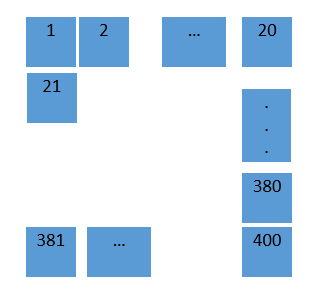
1. The dimensions of the grid – how many rows and columns should the grid have?

I decided to set this to 20 rows by 20 columns. I will follow the C# convention of zero-indexing, calling the topmost row, Row 0, the next down, Row 1, and so on. Likewise, the leftmost column is Column 0, and the rightmost is at Column 19.

1. We also need to set a specific size for each label (grid cell).

Given the grid size defined above, a label size of (20, 20) pixels seems good choice. This creates a total playing space of 400x400 pixels. You must choose a backcolor for the labels: pick anything that takes your fancy.

1. In total 400 labels must be created and precisely positioned to create the grid. This could be done manually, but it would be painfully slow and laborious, so we don’t want to do that. We need to find a way to avoid having to do this activity by hand.
2. While thinking through the design I decided to use a panel object to hold all the labels. This makes the coordinates of their locations much easier to manage. The coordinate of the top-right corner of the panel is (0,0) no matter whereabouts the panel itself is positioned on the form. Furthermore, if I need to reposition the grid on the form, I can move the panel to adjust the position of all labels in one go.
3. One solution to avoiding the manual creation and placement of 400 labels is to write a support program. This program will generate the C# code that needs to appear in Form1\_Designer.CS to define all 20 x 20 = 400 label objects in their correct positions. The labels relationship to the grid is shown below. Label1 represents the top-left cell of the grid (or Row 0, Column 0), Label 20 is the top-right cell. Label 21 is the one at Row 1, Column 0, and Label400 is the bottom-left most cell.



To create the code needed to construct these labels requires some understanding of the contents of the Designer.cs file, or at least the bits that currently matter to us. To investigate further, you should…

* Created a Form and resized it so it is big enough to hold all the labels and still provide space  
   around the edges for the presentation of other control buttons, labels, and so on.
* Added a panel object to the form, positioned it accordingly, and set its size to be precisely 400,400 pixels. I modified its Backcolor just so that I it can see it clearly on the Form.
* Next, create a single label object. Leave it named as Label1, but set up its size and backcolor properties. It can be placed anywhere on the panel for now, as in the example above. Copy and paste this label so that you have two of them, again as shown above.

Now open the Designer.cs file associated with the Form and study it carefully, after expanding the region of code that defines the labels and other GUI elements



Careful study should determine that there are four key areas of interest, each illustrated below…

(1)

private void InitializeComponent()

{

this.panel1 = new System.Windows.Forms.Panel();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

(2)

//

// panel1

//

this.panel1.Controls.Add(this.label2);

this.panel1.Controls.Add(this.label1);

(3)

//

// label1

//

this.label1.BackColor = System.Drawing.Color.SkyBlue;

this.label1.Location = new System.Drawing.Point(77, 231);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(20, 20);

this.label1.TabIndex = 0;

(4)

private System.Windows.Forms.Label label1;  
private System.Windows.Forms.Label label2;

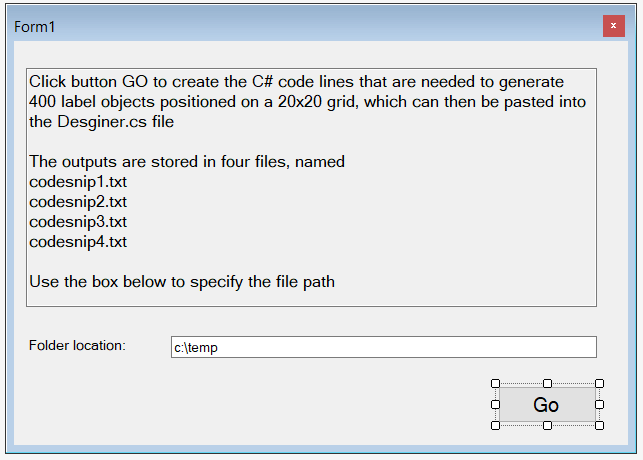
Study the precise nature of the code that defines the two label objects. Notice most of it is common to both. You do not need to understand every nuance of it, but should be able to appreciate how to replicate these code lines in order to generate additional label objects, using the current code as a template to guide you.

Your first task is to write a program to generate the C# code lines needed to make all 400 label objects. This can be written out to a text file, or alternatively, into a textbox object. Once created, we will carefully copy and paste these lines into the correct places in our Designer.cs file, thereby avoiding creating all the labels manually!

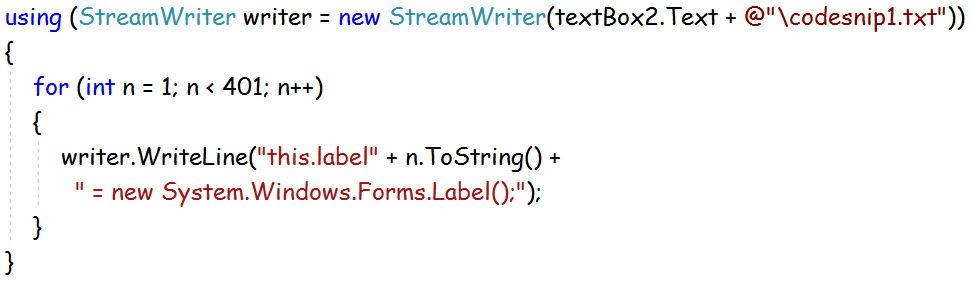
You may wish to revise of the following topics before you start

* + setting up loops in C#
  + writing text files in C#

Write simple Windows App (or Console program if you prefer) to create the code lines needed



**My Solution for the first section of code lines required:**



Have a go and see if you can work it out yourself first. If you can, well done! If you get stuck after making a valiant effort, you can expand the image of my code above to see how I have programmed this. Notes:

* I use the string concatenation operator (+) to join up various parts of the output text line
* I use a **for** loop with an integer control variable set to run from 1 to 400 (i.e. for each label in turn)
* I use the control variable (“n”) to construct names “label1”, “label2”, …through to “label400”
* I use a StreamWriter object, and its WriteLine method, to output the code lines to a text file

Very similar code is needed to generate the C# lines for sections 2 and 4 as identified above.   
The only real difference is each case in the precise details of the text created.

Using this example as a template, generate the necessary code lines for sections (3) and (4) of Designer.cs. The majority of code lines needed for each label object is the same in each instance, with only the specific name of the object changing in each instance.

Generating the code lines in section (3) is the most complex, because it requires a specific coordinate location for each label object to appear in the second line of generated code. I.e.

this.label1.Location = new System.Drawing.Point(0,20);

To position the labels in a tight grid pattern, these coordinates must be created as…

label1 (0,0) label2 (20,0) label3 (40,0) …etc label20 (0,380)

label21 (20,0) label22 (20,20) label23 (20, 40) … etc label 40 (20,380)

…and so on  
…until we get to  
 label400 (380,380)

With a bit of thought you should establish this requires…

* two variables named, say, X and Y, to hold the respective coordinates
* a loop to create the 400 label objects. At the end of each iteration of the loop, variable X will have the value 20 added to itself in order to position the next label correctly, as shown above
* if adding 20 to X takes it to the value 400, this signifies it is time to start a new row. So, reset the value of X to 0, and add 20 to the Y coordinate so that subsequent labels are positioned on the next row
* the label name itself appears inside double quotes; use the syntax \” to insert these inside a string

**My Solution:**

int x = 0;

int y = 0;

for (int n = 1; n < 401; n++)

{

writer.WriteLine("//");

writer.WriteLine("// label " + n.ToString());

writer.WriteLine("//");

writer.WriteLine("this.label" + n.ToString() + ".BackColor = System.Drawing.Color.SkyBlue; ");

writer.WriteLine("this.label" + n.ToString() + ".Location = new System.Drawing.Point(" +

x.ToString() + ", " + y.ToString() + ");");

writer.WriteLine("this.label" + n.ToString() + ".Name = \"label" + n.ToString() + "\";");

writer.WriteLine("this.label" + n.ToString() + ".Size = new System.Drawing.Size(20, 20);");

writer.WriteLine("this.label" + n.ToString() + ".TabIndex = " + n.ToString() + ";");

//update coordinates

x += 20;

if (x == 400)

{

x = 0;

y += 20;

}  
}

As before, try and develop the required code yourself first, but if you get stuck and need to move on select the lines presented above and reset the Font size so you can read them. You can even cut-and-paste these lines directly into your program to save further time and effort.

Once written and executed, copy-and-paste the outputs from your support program into the minefield Designer.cs file. Then test all is working as it should: in other words, your Solution should have 400 labels, named label1, lable2, up to to label400, all nicely positioned into a grid as previously discussed.

<commit1a>

<commit1b>