Lab 7 Due next lab period – week of June 6-9

Purpose

1. Create a memory game using C#
2. Further intro to .NET Framework for C#

What to submit

1. Demo will be done in lab, in your lab section.
2. Upload your project compressed into a .zip file to the Dropbox on D2L.

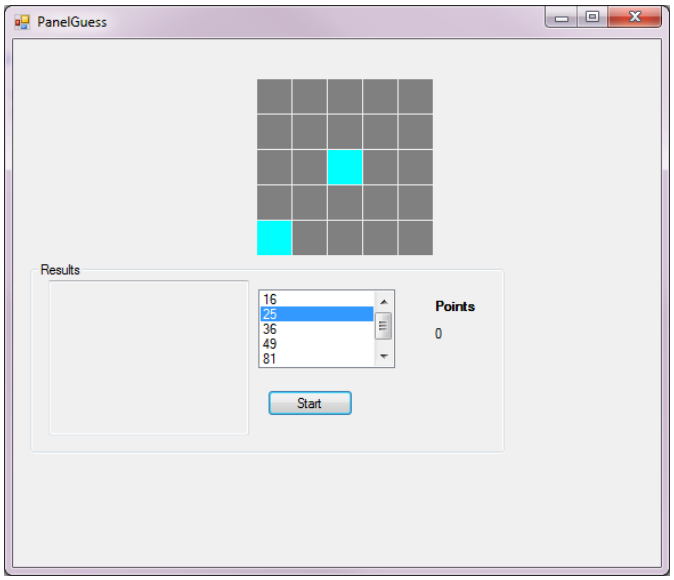
Evaluation (24 marks):

1. Be prepared to demonstrate your solution at the start of next lab. If you are not ready at the start of lab or are not present, a mark of 0 will be given.
2. Your solution will be assessed based on (3 marks each):
    - Proper form layout, adhering to principles of good design
    - Panels drawn correctly on form, based on size selected from listbox control
    - Panels all handle click events properly, and share a single event procedure
    - Selected panels are randomly generated
    - Game end programmed correctly
    - Easy/Hard mode works correctly
    - C# code style – variables named appropriately, comments used in the code where needed, code is readable and understandable.
    - Correctly answering questions about your solution.
3. Bonus: Add sound or other special effects to the game (up to 2 marks).
4. **All work must be individual.** Any plagiarism will result in a mark of 0 and possibly additional penalties.

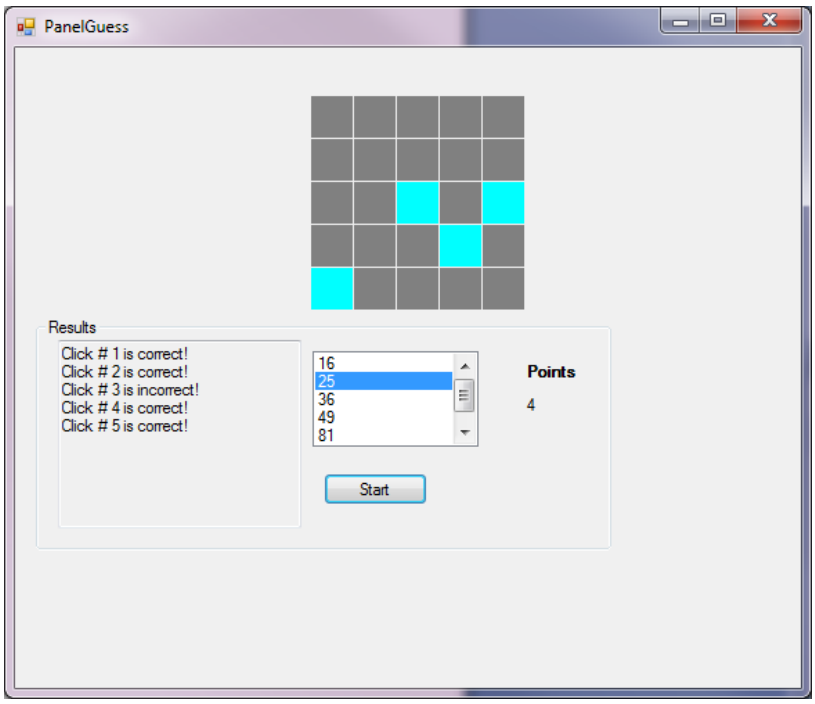
Description

In this lab you will use the C# Visual Studio template to build a simple prototype memory game using coloured panels as blocks.

A grid of blocks coloured gray appears and when the user clicks start, a pattern of six randomly selected blocks are highlighted as light blue. After a second, the selected blocks revert back to their original gray colour and the user must now click on those blocks which were highlighted. The user may click on the correct blocks in any order – the sequence does not matter. A point is awarded for each correctly identified block. If the user clicks on an incorrect block, a message appears in the Results textbox. Once all six blocks are identified by the user, additional points are awarded – six minus the number of incorrectly clicked blocks up to a maximum of six. The listbox shows the various grid sizes available: 16, 25, 36, 49, 81, and 100. The form should automatically resize itself based on the grid size with the grid roughly centered within the form. Not shown is the menu option to permit a selection between “easy mode” and “challenge mode”. When the easy mode option is selected, the user is presented with a message box after the six highlighted blocks are shown. The message box provides extra time for the user to visually remember the locations of the blocks. The default is “challenge mode” where the user is not presented with a message box but instead is given only 1 second to preview the highlighted block locations in the grid before starting. Also, once a block has been clicked, it should not be counted again as an incorrect or correct block.



First two of six blocks are highlighted.



User has found four of the six blocks.

As this is a prototype, the points calculation is kept simple. There is no provision for adjusting the score for the larger sized grids, which might be more difficult.

Steps

1. Start a new C# Windows Form application and name it PanelGuess.
2. Open the source file in the project Form1.cs.
3. Edit the Form1.cs[Design] page to match the above image, adding a GroupBox control and inside it add a TextBox, ListBox, Button, two Labels, and a Timer. Change the text of the GroupBox to “Results”. Make the TextBox multi-line and readonly, give it the name txtMessage. Name the ListBox lstGridSize. Name the button btnStart with the text “Start”. The labels should be stacked vertically. The top label should have the text “Points” in bold font. The bottom label should have the name lblPoints. (for your final solution you may improve this interface, if you like)

4. Do not edit the Program.cs file. This is the main class file for the application.
5. Edit the file Form1.cs.
    a. Below the line that reads

```
public partial class Form1 : Form
{
```

Declare the following class variables:

- rndSpot - of type Random that will be used to select a spot at random from the gird
- Variables of type int:
    - intNumSpots – the total highlighted spots in the grid
    - intCount - the count of randomly selected spots
    - intClickCountCorrect - the number or correctly identified selected panels
    - intClickCountIncorrect – the number of incorrectly identified selected panels
    - intPanelCount – the total number of panels
    - intPoints – the total number of points
- Boolean variables:
    - blnReadyToClick – set to false initially to prevent the user from clicking panels while the spots are being randomly selected
    - blnEasyMode – indicating whether test is in easy mode
- intSpots - an array of integers of size 100, that contains the indices of the randomly marked panels
- lstPanels, a list of the panels in the form

Declare the following constants:

- `private const int intPanelWidth = 30;`
- `private const int intPanelHeight = 30;`
- `private const int intPanelSepDistance = 1;`

b. The Form1() constructor in C# has the `InitializeComponent();` statement only. Just below that add `StartPanelGuess();` which will be the procedure that sets up the UI.

c. The `StartPanelGuess` procedure must initialize key elements of the form such as the points and make the 5 x 5 grid the default.

i. Set the Text property of the panel to “PanelGuess”
ii. Initialize the Text property of the points label to 0
iii. Initialize the tableau to a 5 x 5 grid
iv. Instantiate the rndSpot object
v. Initially start with 6 spots
vi. Establish the possible tableau sizes from easy (16-25) to hard (81-100).

```
lstGridSize.Items.Add(16);
// ... student adds in the rest...


// Start with the 5 x 5 grid.
// ... student makes the 5 x 5 the default selection
```

vii. Set the groupBox1 Text property to “Results”
viii. Instantiate the lstPanels list

```
lstPanels = new List<Panel>();
```

d. Setup the Timer control to show the selected panels:

i. Add using System.Threading; to the top of the source file

The Timer control will be set up so that a highlighted block appears every half second until all six are selected by the program. Once all six blocks are shown, the Timer is set to disabled and "hide" the selected blocks.

The variable intCount keeps track of how many blocks have been randomly selected by the program so far. The variable intNumSpots is going to be six (the total number of selected blocks).

The blocks are represented by Panel controls. The grid is a List of Panels called lstPanels. So we can refer to individual Panels within the List as lstPanels[n] for the *n*th Panel.

ii. For the Timer's Tick event write the following in C# code:

```
If the number of selected panels (intCount) is less than the number we need to select (intNumSpots)
    Get the location of the selected panel from the array intSpots
    Set the BackColor property of the selected panel to Aqua
    Increment intCount
Else
    Disable the timer
    If player is in easy mode
            Show the message box until the user presses OK
            After the message box is closed, hide the selected panels
    After 1000 milliseconds
            Hide the selected panels
```

e. Complete the procedure `DisplayPanels` to cause the grid of panels to appear on the form.
   i. initially set all the elements in the `intSpots` array to -1. Then as a random `Panel` is selected, the corresponding entry in `intSpots` is made. A Boolean function named `SpotAvailable` will return true if the chosen panel at start up has not already been selected. Most of the code in `DisplayPanels` is the calculations for the position of the grid and the `GroupBox` within the form. Since the `Panels` are created dynamically as the program is running, their positions can change depending on the selected grid size. The larger grids will require the `GroupBox` to be shown further down on the form. If you wish, you may use the code found in DisplayPanels.txt on D2L
f. Write the SpotAvailable function:
   i. The `SpotAvailable` Boolean function accepts a single integer parameter and determines if that value is found within the elements of the `intSpots` array. If so, a true value is returned; else, false.
g. Write the Click event for the Start button:
   i. The Click event for the Start button indicates to the program that the user wants to begin a new round. All the existing panels in the List `lstPanels` need to be removed. The `intPanelCount` is determined based on the current `lstGridSize` selection. Then the `intPanelCount` is checked to find the proper `Height` and `Width` dimensions of the form. A call to the procedure `DisplayPanels` is made and the Boolean flag `blnReadyToClick` is made false. To determine the form's dimensions there is a utility in Microsoft Visual Studio Tools called Spy++ which will show you various details of the open windows.
   ii. The code in StartButton.txt on D2L will get you started, but note that it is not complete!
h. We have the C# code to handle the Timer events, code to display the grid, code to initialize the UI but we need the last piece, C# code to handle the Panel Click events. Rather than have a separate

event handler procedure for each panel, we will use one procedure for all the panels. The code will pick up the Name property of the clicked Panel and that will reveal which one (e.g. Panel6 is the sixth panel). Then we need to determine if that Panel is one of the selected Panels and score accordingly. The Panel1_Click.txt code outline can be found on D2L.

i. The flag `blnReadyToClick` needs to be set to true once the user is ready to start clicking on Panels. This assignment needs to be done somewhere in the `Timer Tick` event.

j. Add in a `MenuStrip` control which will allow the user to toggle between "Easy Mode" and "Challenge Mode".