[1410] WPF Version of Life

WPF only works on a Windows computer! Use the computer on the desks if you don't have Windows.

Let's create a WPF version of Life. Here's what it will look like:

Here are the steps:

1. Start up Visual Studo and create a new "WPF App (.NET Framework)" project named "WinLife". (You can find "WPF App" under "Installed / Visual C# / Windows Desktop"). DON'T FORGET TO CHANGE THE NAME FROM "WpfApp1" TO "WinLife"!

2. Open up the "Toolbox" on the left side of the screen and double-click the Button tool to add a button to the upper-left corner of the application window. In the Properties panel (lower right part of Visual Studio window) change the "Name" of the button to "RunStopButton" and change the "content" of the button to the "Run".

3. Click on the "Grid" tool in the Toolbox, then click and drag on the application window to add a grid that occupies the rest of the empy space in that window, then change the "Name" property of the grid to "LifeGrid"
4. Click on the top of the application window to select the entire window, then change its "Name" to "WinLifeWindow" and its "Title" to "WinLife".

5. With the application window still selected, click on the "Lightning Bolt" button in the upper right corner of the Properties area to open the Event Handlers panel.

6. Scroll down to the "Loaded" entry and double-click your mouse inside the textbox next to it. The code editor should open up with the "WinLifeWindow_Loaded" event handler added.

7. Add the following code just ABOVE the "MainWindow()" constructor (i.e. after line #22):

```
const int GridSize = 80;
const bool DEAD = false;
const bool ALIVE = true;
private Cell[,] cells = new Cell[GridSize, GridSize];
private System.Timers.Timer timer = new System.Timers.Timer(100);
```

8. Now add the following code inside of "WinLifeWindow_Loaded()":

```
for (int i = 0; i < GridSize; i++)
{
   LifeGrid.ColumnDefinitions.Add(new ColumnDefinition());
   LifeGrid.RowDefinitions.Add(new RowDefinition());
}
for (int i=0; i < GridSize; i++)
{
   for (int j=0; j < GridSize; j++)
   {
      cells[i, j] = new Cell(DEAD);
      Rectangle cellRect = new Rectangle();
      Grid.SetColumn(cellRect, j);
      Grid.SetRow(cellRect, i);
      cellRect.Fill = Brushes.Black;
      LifeGrid.Children.Add(cellRect);
   }
}
```

9. Next, we need to add the Cell class. In the "Solution Explorer" panel (upper right corner of the Visual Studio window), right-click on the "WinLife" node (it should be the second line from the top) and select "Add / Class..." from the menu that appears. Change the "Name" from "Class1.cs" to "Cell.cs" and press "Add"

10. Add the following code inside the Cell class template (i.e. after line #10):

```
  private bool state;

  public bool State {
    get { return state; }
    set { state = value; }
  }

  public Cell(bool state)
  {
    this.State = state;
  }
```

11. Compile and run the program. A big black square should appear where your grid is. Click Visual Studio's "Stop Debugging" button on the toolbar (i.e., the red square) to stop the program.

12. We're going to use WPF's "Data Binding" feature to update the cells. Start by updating the Cell class to be the following:

```
using System;
using System.ComponentModel;

namespace WinLife
{
    class Cell : INotifyPropertyChanged
    {
```

```
        private bool state;
        public event PropertyChangedEventHandler PropertyChanged;

        public bool State {
            get { return state; }
            set {
                state = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("State"));
            }
        }

        public Cell(bool state)
        {
            this.State = state;
        }

        public static implicit operator int(Cell c)
        {
            return Convert.ToInt32(c.State);
        }

    }
}
```

9. Now, back in "MainWindow.xaml.cs", update the WinLifeWindow_Loaded event handler so that it looks like this:

```
  for (int i = 0; i < GridSize; i++)
  {
    LifeGrid.ColumnDefinitions.Add(new ColumnDefinition());
    LifeGrid.RowDefinitions.Add(new RowDefinition());
  }

  for (int i=0; i < GridSize; i++)
  {
    for (int j=0; j < GridSize; j++)
    {
      cells[i, j] = new Cell(DEAD);
      Rectangle cellRect = new Rectangle();
      Grid.SetColumn(cellRect, j);
      Grid.SetRow(cellRect, i);
      cellRect.Fill = Brushes.Black;
      cellRect.DataContext = cells[i,j];
      cellRect.SetBinding(OpacityProperty, "State");
      LifeGrid.Children.Add(cellRect);
    }
  }
  timer.Elapsed += (x,y) => ComputeNextGeneration();
```

10. Click on the "MainWindow.xaml" tab and then click on the grid once to select it. Now, find the "MouseDown" event handler and double click the box next to it to add a template for that event to your code window.

11. Add the following code to the MouseDown event handler:

```
        private void LifeGrid_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.OriginalSource is Rectangle)
            {
                Rectangle rect = ((Rectangle)e.OriginalSource);
                Cell cell = (Cell)rect.DataContext;
                cell.State = !cell.State;
            }
        }
```

12. Now add the "ComputeNextGeneration()" method to perform the Life process:

```
        private void ComputeNextGeneration()
        {
            int[,] n = new int[GridSize, GridSize];

            for (int i = 1; i < GridSize - 1; i++)
            {
                for (int j = 1; j < GridSize - 1; j++)
                {
                    n[i, j] = cells[i - 1, j] + cells[i + 1, j] + cells[i, j - 1] + cells[i, j + 1]
                        + cells[i - 1, j - 1] + cells[i + 1, j - 1] + cells[i + 1, j + 1] + cells[i - 1, j +
1];
                }
            }

            for (int i = 1; i < GridSize - 1; i++)
            {
                for (int j = 1; j < GridSize - 1; j++)
                {
                    if (n[i, j] == 3) cells[i, j].State = ALIVE;
                    if (n[i, j] >= 4) cells[i, j].State = DEAD;
                    if (n[i, j] <= 1) cells[i, j].State = DEAD;
                }
            }
        }
```

13. One last thing - we need to enable the "Run" button. In "MainWindow.xaml", click on the Run button, then find the "Click" event handler in the Properties panel (click the lightning bolt if you don't see the event handlers). Double-click in the text box beside the "Click" entry and then enter the following code for the RunStopButton_Click() event handler:

```
        private void RunStopButton_Click(object sender, RoutedEventArgs e)
        {
            if (RunStopButton.Content.ToString() == "Run")
            {
                RunStopButton.Content = "Stop";
                timer.Start();
            }
            else
```

```
            {
                RunStopButton.Content = "Run";
                timer.Stop();
            }
        }
```

14. That's it! Run the program - draw an R-Pentomino and then click the "Run" button. Voila!