One Way of Implementing Coloured Grids in WPF

The following guide is not the definitive way to implement a DataGrid with coloured cells, but it is one way that is likely to be reasonably reusable and not too difficult.

What C# classes you will need

- **A data generator:** One or more classes capable of generating the activity grid or heat map information (e.g., teaching–consulting–free, or clash–no clash, or value + colour intensity). This is left up to you. Implement it/them however you wish. Part of your code will be capable for transforming your data into eight instances of the following:
- **Row data:** A view model class that represents the data for **one row of the table**. It could have 10 main properties (5 for day values [text or numbers] and 5 for colours) or 2 main properties (an array of 5 strings and an array of 5 colours). The colours will need to be defined as type `SolidColorBrush`, which can be created from an actual `Color` object if needed (for generating the heat map, for instance). It will likely need at least one more property to hold the time of day.
- **Custom grid control:** A UserControl holding a `DataGrid` (more details below) that becomes your reusable coloured grid. It should expose some properties for setting the data it is to display.

At design time

1. Design your user control and implement any required code behind (none of which is complicated as it's for display, not data generation).
2. Place your custom grid control in all the places you need it.

At runtime

When one of these coloured grids needs to be displayed,

1. the appropriate **data generator** will retrieve or process all necessary data and construct its own internal model of colours (type `Color` or `SolidColorBrush`) and any data it needs displayed;
2. the **data generator** produces a `List` or other ordered collection of **row data** objects based on its internal model; and
3. the relevant controller passes that list to the **custom grid control**, which sets the `ItemsSource` property of its `DataGrid` accordingly. The `DataGrid` will then display each **row data** object in each visual row in turn.

What XAML will the UserControl have?

The key to having the cells coloured appropriately is some additional XAML in the definition for each DataGrid column that sets the Background property of the TextBlock (that displays the cell's value) to a `Brush` object obtained from the **row data** object being displayed.

Here is a sample that requires just a little more refinement to be useful in your application. Note that this example assumes the **row data** object has an array- or List-valued property called `Values` (which are the values to display in each cell of the row) and another array- or List-valued column called `Colours` that contains `Brush` objects, and so can be bound the the Background property of the TextBlock that occupies each cell.

```
<DataGrid x:Name="tblMap" AutoGenerateColumns="False" IsReadOnly="True"
CanUserReorderColumns="False">
    <DataGrid.Columns>
        <!-- There are also ways to define row labels and have them bind to a
property (such as the time of day)
             This would be neater visually, but is not shown here. -->
        <DataGridTextColumn Header="Not a time cell" Binding="{Binding
TimeOrSimilar}" />

        <DataGridTextColumn Header="A" Binding="{Binding Values[0]}">
            <DataGridTextColumn.ElementStyle>
                <Style TargetType="{x:Type TextBlock}">
                    <Setter Property="Background" Value="{Binding Colours[0]}"/>
```

```
                </Style>
            </DataGridTextColumn.ElementStyle>
        </DataGridTextColumn>

        <DataGridTextColumn Header="B" Binding="{Binding Values[1]}">
            <DataGridTextColumn.ElementStyle>
                <Style TargetType="{x:Type TextBlock}">
                    <Setter Property="Background" Value="{Binding Colours[1]"/>
                </Style>
            </DataGridTextColumn.ElementStyle>
        </DataGridTextColumn>
    </DataGrid.Columns>

    <!-- And three more like that... -->
</DataGrid>
```

That's obviously quite verbose, although the five repeated DataGridTextColumns are very similar, so the amount of effort is not great. By defining that inside a `UserControl` you can reuse it without repeating all the XAML needed to define the columns and where they'll get their contents and background colours from.

Best of luck with it, and remember that it's an extension task, only required for the highest levels of achievement.
