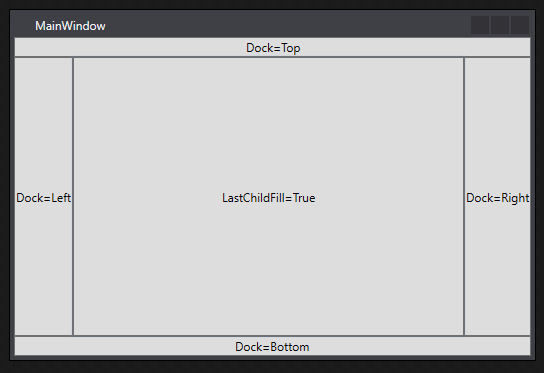
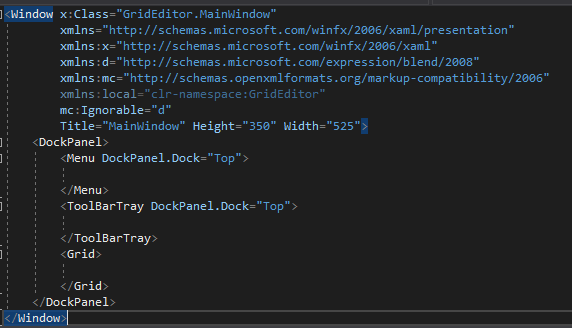
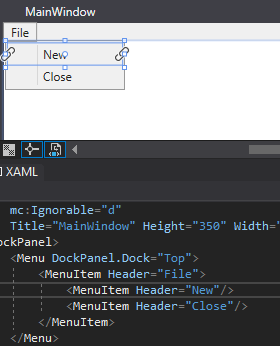
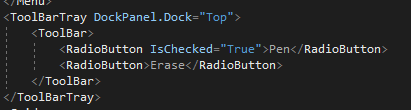
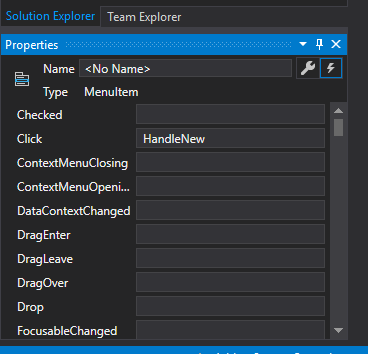
**Grid Editor**

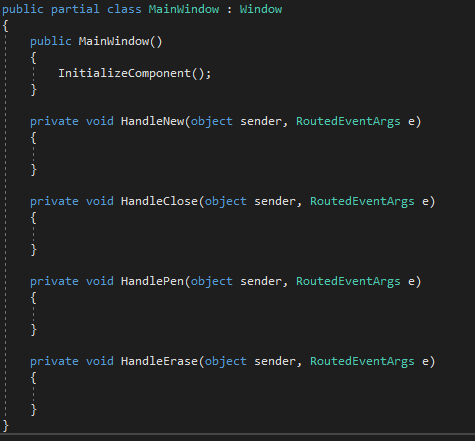
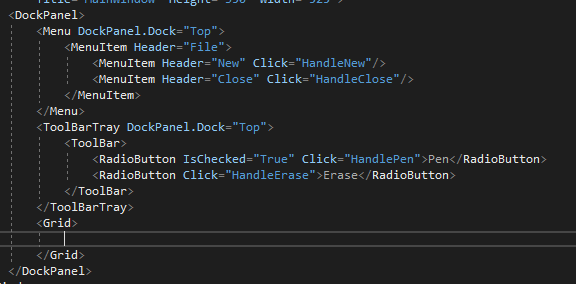
1. Create a new “WPF App” in Visual Studio and call it GridEditor

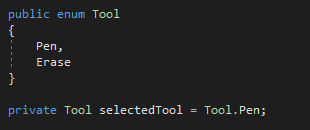
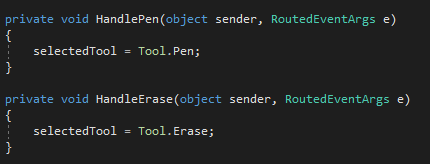
In our XAML, we should start with a <Window>, and within that it should contain a <Grid>

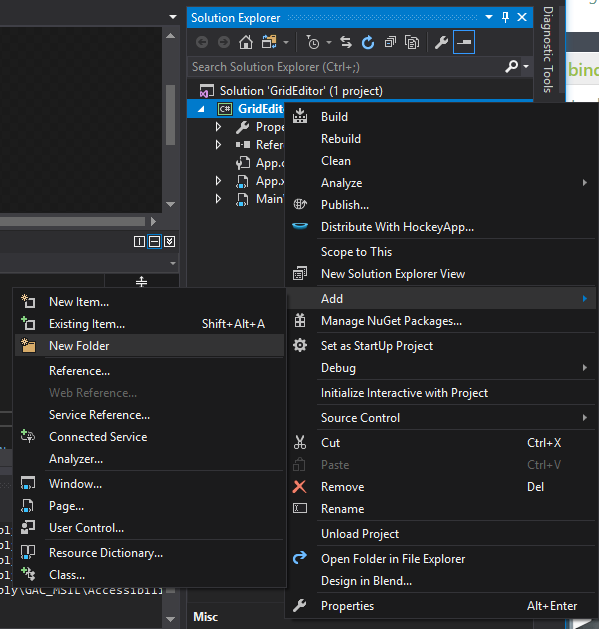
1. Let’s change our layout from ‘Grid’ to a component called ‘DockPanel’:  
   DockPanel, helps align it’s children to different sides of the window (or it’s parent). Finally by default the last child fills the inner most content.
2. This is very useful for having bars on top and bottom.
3. We are also going to be creating a ‘Menu’ and ‘Toolbar’. The menu will contain File->New and File->Close and the Toolbar will contain our painting tools.
4. In XAML Remove your grid, Create a DockPanel, And within that put in a menu (aligned to the top) and a toolbar tray (also aligned to the top)  
     
   Notice that in XAML, sometime you can add properties to the layout of the children by saying: ParentClassName.Property=”Value”  
   In this case, we are giving out Menu and Toolbar tray some layout settings that it’s parent can read (DockPanel).  
   You will see that later with Grid.Row and Grid.Column
5. Let’s add menu items to our menu. A menu item can be nested to add more child nodes. So we want to create a File->New and File->Close,. They have a property called ‘Header’ which you can type in it’s name
6. 
7. Now let’s add some tools to our toolbar tray. A tray contains and can dock many toolbars. Lets just add a single toolbar with the two RadioButtons ‘Pen’ and ‘Erase’. The first radiobutton we will mark as checked. RadioButtons can only check one in it’s group (like only selectingone tool)

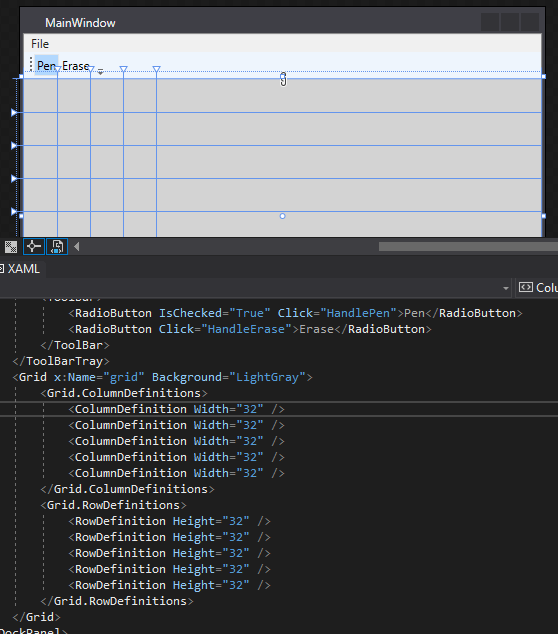
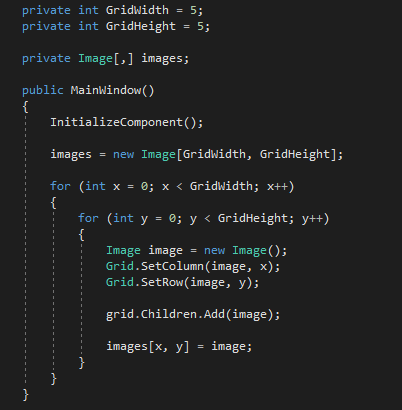
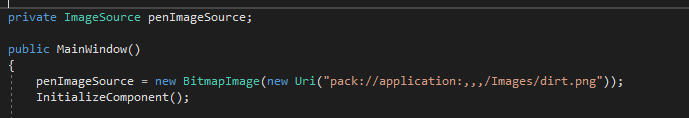
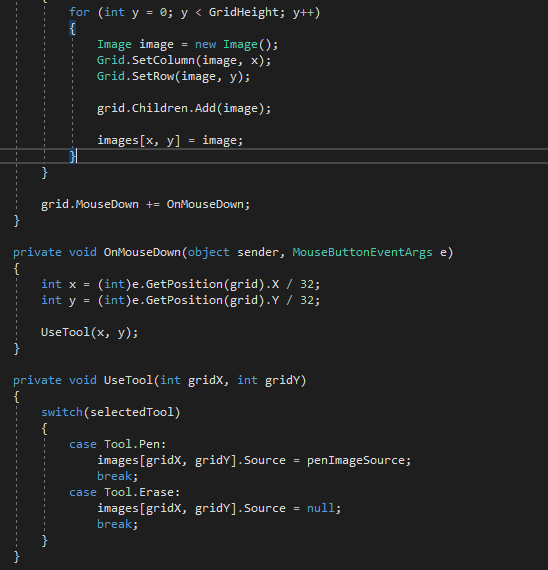


1. Great! You can compile and run this, but it doesn’t do much. Let’s link these buttons and menu items to Click events so we can add some code.  
   Add the Click events to the two Menu Items (New and Open), and another two to the buttons (Pen and Erase).  
   You can do this easily by selecting the node in the XAML, then clicking the Lighting Bolt (events) icon in the Properties Panel. And typing in a function name inside ‘Click’ and hitting the return key.  
     
     
   After this your code should a bit like this:



1. Now let’s create an Enumeration (enum) for easy switching detection for our tools (in case we create more later).  
   Define an Enum called ‘Tool’, and the values of ‘Pen’ and ‘Erase’. Then create this enum with it’s default value to ‘Pen’ (as that is currently selected in our XAML button at launch).  
   
2. Now let’s actually change this value when we click our buttons in out events.
3. Lets add some more assets to our project, Right Click The Project (not the solution) and Add->New Folder. Call this ‘Images’



1. Right click the folder, and add ‘Existing Item’ and change the filetype to ‘Any’. This should enable you to add single 32x32 image sprites to your image folder. Add an image to this folder. I’m using ‘SingleTileSamples/dirt.png’ from Atlas.zip
2. Let’s setup the grid on XAML for now but adding rows and columns for our tiles to go inside and a reference name so we can access it from our code later.  
     
   **Xaml Feature**  
   For any property in a Xaml Node. You can also use the dot syntax with a new inner node.  
   For example, if you have the header in the following syntax:  
     
   <MenuItem Header=”My Header”/>   
     
   You can use the dot syntax to set any property like so:  
     
   <MenuItem>  
    <MenuItem.Header>My Header</MenuItem.Header?  
   </MenuItem>  
     
   This is useful for where we have objects that are bigger/more complex than simple strings. Like in the following ‘Grid.ColumnDefinitions’  
     
   Setup your grid to have 5 Columns and 5 Rows, each with a Width/Height of 32 pixels. We do this by setting the array ‘ColumnDefinitions’ with new classes called ‘ColumnDefinition’. We also do the same for RowDefinitions.  
   Note this starts to slice our grid in the MainWindow which highlighting the ColumnDefinitions or RowDefinitions.  
     
   Finally, add x:Name=”grid” to our grid. This way we can access it by it’s variable name in our .cs file.  
   And add the property ‘Background’ to ‘LightGray’ or a color of your choice *(this is important for our grid to be clickable with no content).*
3. Let’s add images to every cell, ready for an ImageSource. It’s best to do this in code with loops.  
   Here we make a nested array like a grid. Which can store based 2 indexes. This is perfect for our scenario.  
   Here want to add images to each cell by looping through the grid. Create an image, assign it’s cell. Add it to the grid, and finally save it in the nested array for us to modify later.  
   
4. Lets make a ‘ImageSource’ for our Pen to draw with.  
   Define an ‘ImageSource’ property. Then load it with a bitmap image from a Uri.  
   Notice, we have to do a strange syntax. This is because our image is embedded inside our application as a ‘resource’. Using this syntax it will be able to load from code just like a regular file (**Note:** You don’t have to do it this way in Xaml).  
   With this property, later we can load it into the ‘Image’ cells.
5. Finally we need to add the MouseDown listener on grid, detect it’s position and then change an image. Here I’ve added a listener in code to a new function called ‘OnMouseDown’ by typing ‘grid.MouseDown +=’ and hitting Tab to autocomplete and writing my function.  
   Whenever a mouse down is detected on our grid. We can intercept this and do something (like add tiles).  
    In mouse down I have calculated the tile position index given the sprite size of 32. (e.GetPosition, returns the mouse position relative to a view in pixels).  
   And then called a function called UseTool. Which will check the current tool, change the ImageSource of one of our Images in the right cell of our grid.  
   
6. Now when running this, you should be able to click the top left corner and add sprites. Click the eraser and erase srpties. Note there is a bug when clicking out of range when using the tool

Extra Tasks:

1. Use an if statement to fix the error in ‘UseTool’ when clicking outside of our 5x5 grid.
2. Add the line: Application.Current.Shutdown(); to Close the application
3. A lot like ‘MouseDown’ we have ‘MouseUp’, ‘MouseMove’ and ‘MouseLeave’ events. Add these functions, set a ‘penIsDown’ Boolean to true on ‘MouseDown’ and false on ‘MouseUp’ and ‘MouseLeave’. Then do ‘UseTool’ in the MouseMove but only when the penIsDown is set to true.
4. Try and implement the ‘New’ button
5. As ‘MainWindow’ is getting pretty big and messy. Go ahead and create a new class called “TileGrid” that extends Grid.  
   Take all the image code, and put it in this class.  
   To declare it in Xaml in your MainWindow, add the local namespace:  
   <local:TileGrid x:Name=”grid”/>   
   Take all the image code, and put in in your class with two helpful function “void SetTile(int x, int y, ImageSource source)” and “ImageSource GetTile(int x, int y)”  
   You will also need to set the ColumnDefinition and RowDefintion in code which will make it more flexbile.
6. Add as many features as you’d like to (more sprites, fill, load sprite) etc.