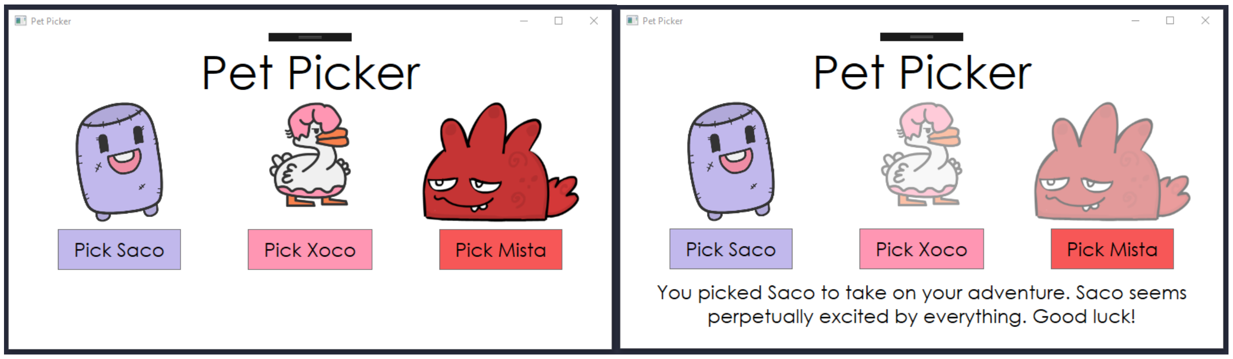
# Pet Picker – WPF Layouts

We're going to be designing a simple pet picker, where you can pick one of three options:

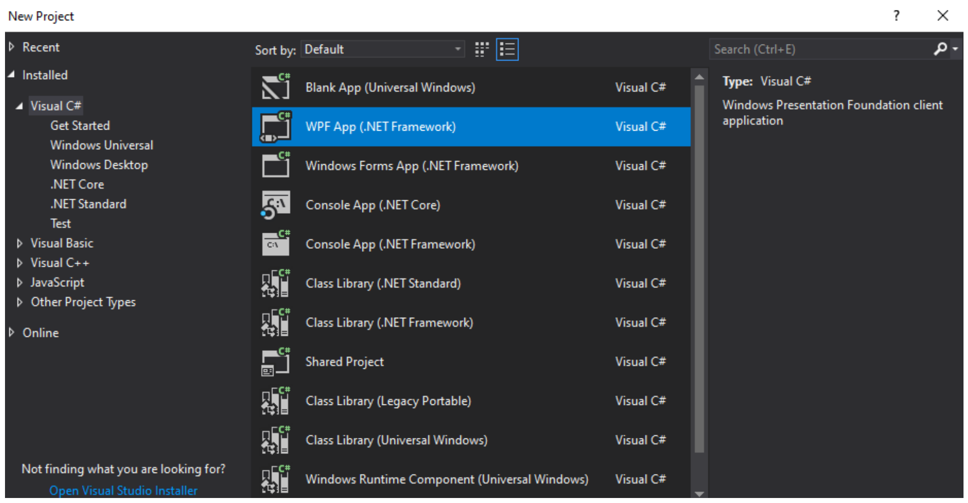


You picked Saco to take on your adventure. Saco seems perpetually excited by everything. Good luck!

When you click on one of the buttons, a description of your chosen pet pops up on screen and the other pets are grayed out slightly.

## Step 0: New Project

- Create a new WPF C# application called PetPicker.
- Set the title of your Window to "Pet Picker."
- Set the window width and height to a size you like.



New Project

- Recent
- Installed
  - Visual C#
    - Get Started
    - Windows Universal
    - Windows Desktop
    - .NET Core
    - .NET Standard
    - Test
  - Visual Basic
  - Visual C++
  - JavaScript
  - Other Project Types
- Online

Not finding what you are looking for?
Open Visual Studio Installer

Sort by: Default

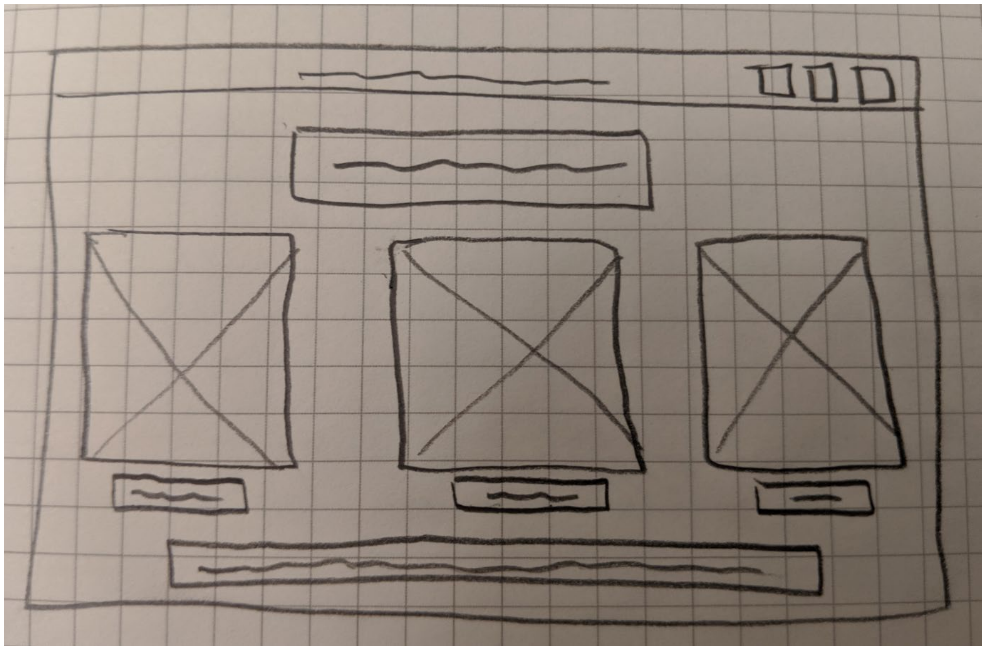
| | |
|---|---|
| Blank App (Universal Windows) | Visual C# |
| WPF App (.NET Framework) | Visual C# |
| Windows Forms App (.NET Framework) | Visual C# |
| Console App (.NET Core) | Visual C# |
| Console App (.NET Framework) | Visual C# |
| Class Library (.NET Standard) | Visual C# |
| Class Library (.NET Framework) | Visual C# |
| Shared Project | Visual C# |
| Class Library (Legacy Portable) | Visual C# |
| Class Library (Universal Windows) | Visual C# |
| Windows Runtime Component (Universal Windows) | Visual C# |

Search (Ctrl+E)

**Type:** Visual C#
Windows Presentation Foundation client application

## Step 1: Sketching and Wireframe

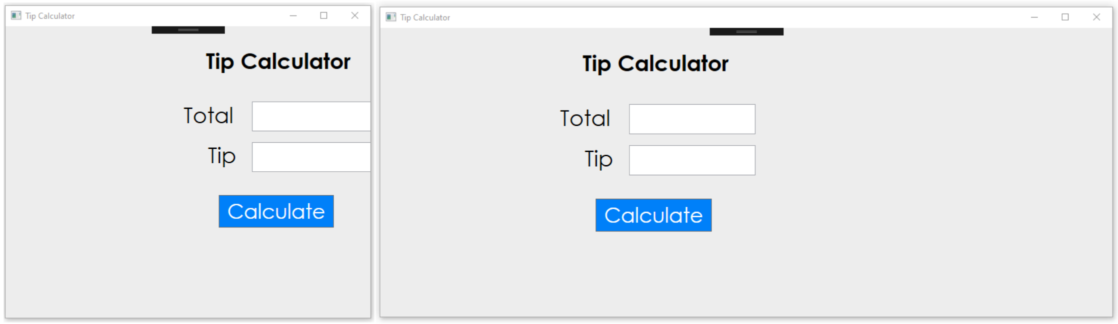
The first step in any user interface is to sketch it out on paper:



The idea is to plan out where you want your elements to go before you start placing them in XAML. The goal of a wireframe is not to create something pretty – it's to focus on layout. Some common conventions that I use – boxes with Xs for images, squiggles for text.

### Step 2: The Grid Layout

Last time, we placed things manually using the toolbox. That's a great way to get started with WPF, but it's not the best way to lay things out on the page. It places things in fixed positions (from the top left of the window) using margins, which means your app doesn't look great if the screen size changes:

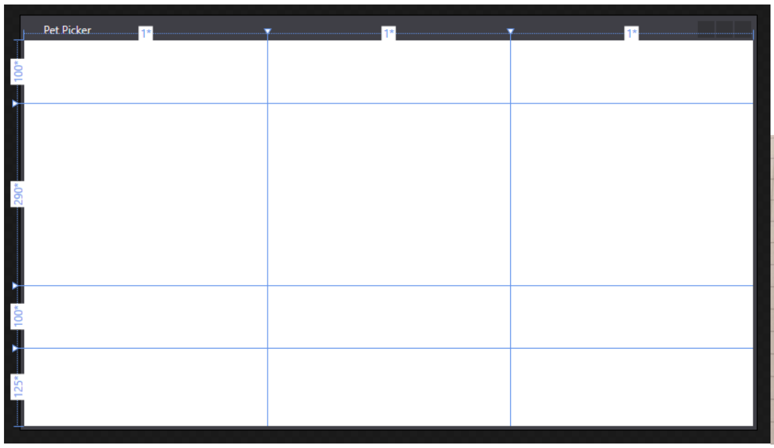


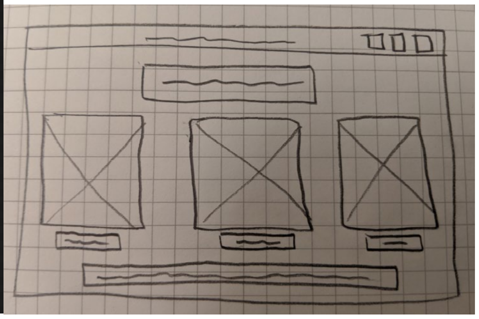
There are a few layout panels in WPF that more flexibly allow you to lay out elements on your page:

- Grid Panel
- Stack Panel
- Dock Panel

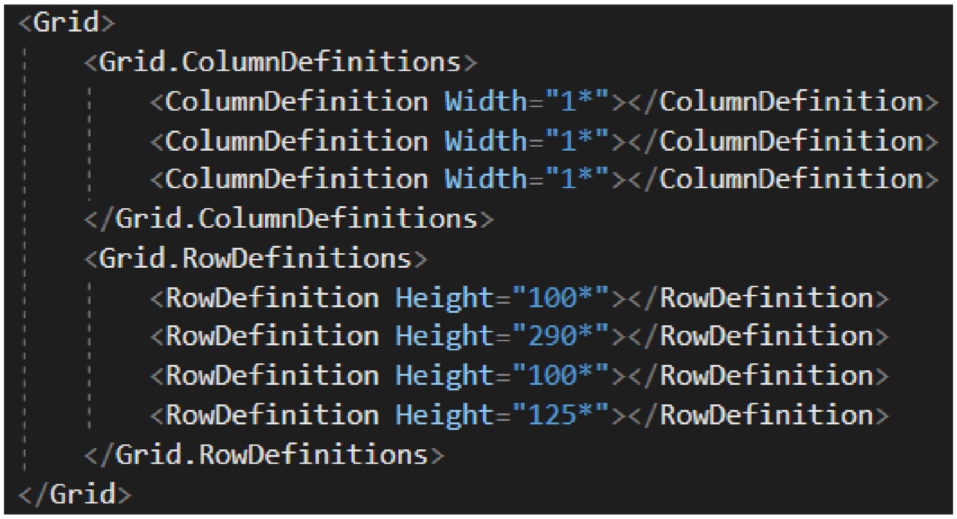
- Wrap Panel
- Canvas Panel

Read through those short tutorials so that you understand what each is. For this project, we're going to use a Grid, which sets up a flexible structure that allows us to place different elements in each cell of a grid. Here's the grid structure we're going to use to place elements in our app (app on the left, wireframe on the right):





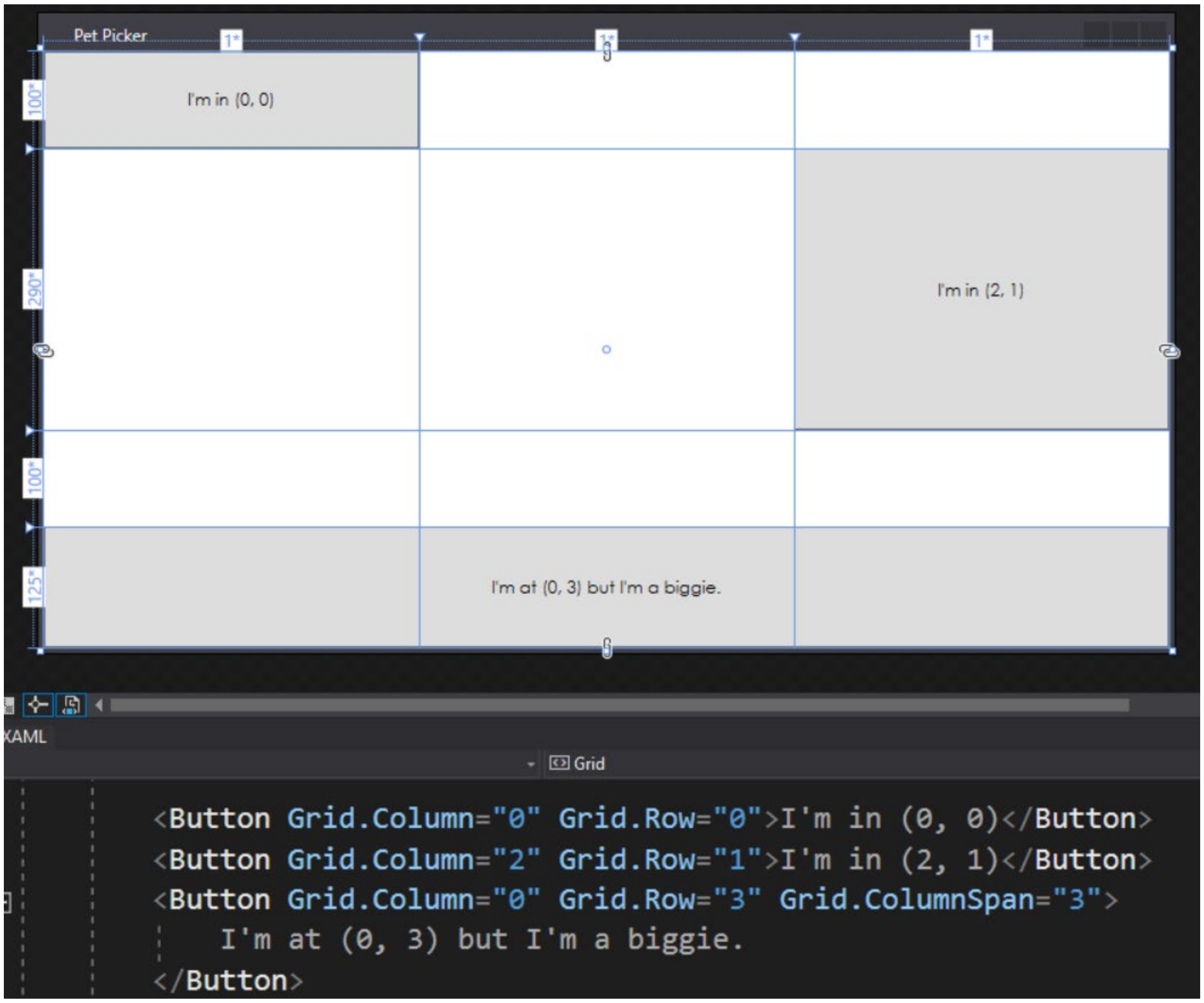
Read the first three sections from the Grid Panel tutorial. To set up the above grid, we need to define the rows and columns. Inside of our MainWindow.xaml, we can add Grid.ColumnDefinitions and Grid.RowDefinitions elements to set up the structure of our grid. Add this to your XAML:



```xml
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*"></ColumnDefinition>
        <ColumnDefinition Width="1*"></ColumnDefinition>
        <ColumnDefinition Width="1*"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="100*"></RowDefinition>
        <RowDefinition Height="290*"></RowDefinition>
        <RowDefinition Height="100*"></RowDefinition>
        <RowDefinition Height="125*"></RowDefinition>
    </Grid.RowDefinitions>
</Grid>
```

Note: if you hover over the dividing lines in the Visual Designer window within Visual Studio (the window with the preview of your app), you can drag and drop to resize the rows/columns, which will change the XAML.

We can then control how we place elements using the Grid.Column and Grid.Row attributes. We can control how many columns/rows they take up with Grid.ColumnSpan and Grid.RowSpan. You can test this out with the XAML below. (If you put it in your project, don't forget to comment out or remove the buttons before moving on.)

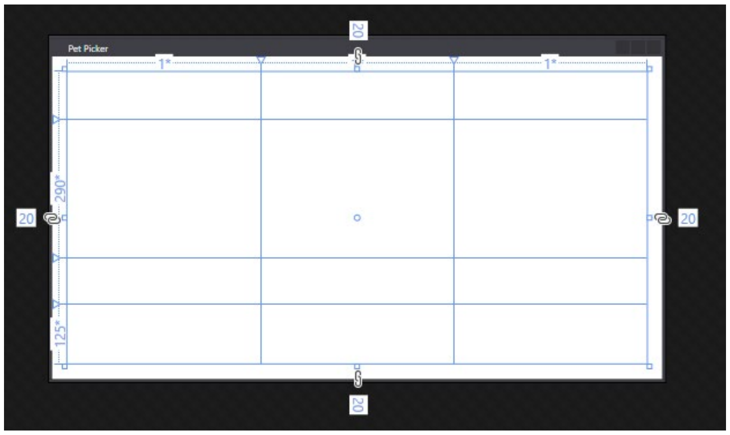


```
<Button Grid.Column="0" Grid.Row="0">I'm in (0, 0)</Button>
<Button Grid.Column="2" Grid.Row="1">I'm in (2, 1)</Button>
<Button Grid.Column="0" Grid.Row="3" Grid.ColumnSpan="3">
    I'm at (0, 3) but I'm a biggie.
</Button>
```

**Step 3: Filling out the Grid**

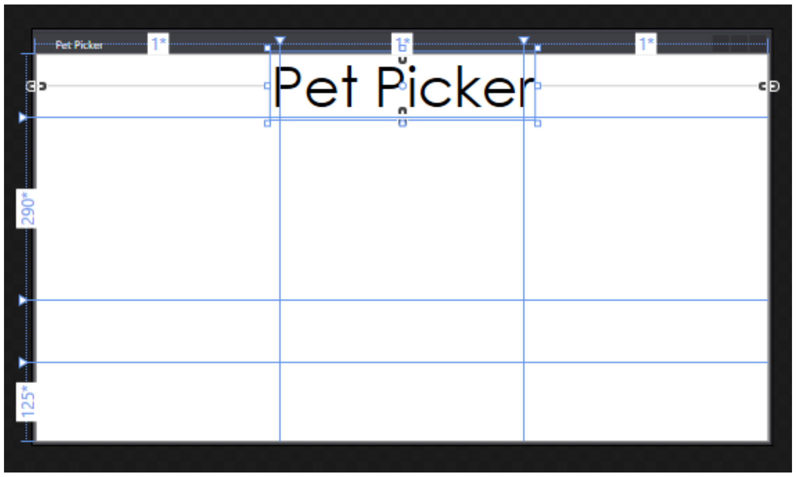
One thing I like to do, before adding anything, is add Margin to the Grid to keep the elements from touching the edge of the screen. (You can also add Margin to most other elements in WPF too.)

```
<Grid Margin="20">
```



First things first, we want out title in the first row:

```
<TextBlock Grid.ColumnSpan="3" Grid.Column="0" Grid.Row="0" HorizontalAlignment="Center"
  VerticalAlignment="Center" FontSize="60">Pet Picker</TextBlock>
```



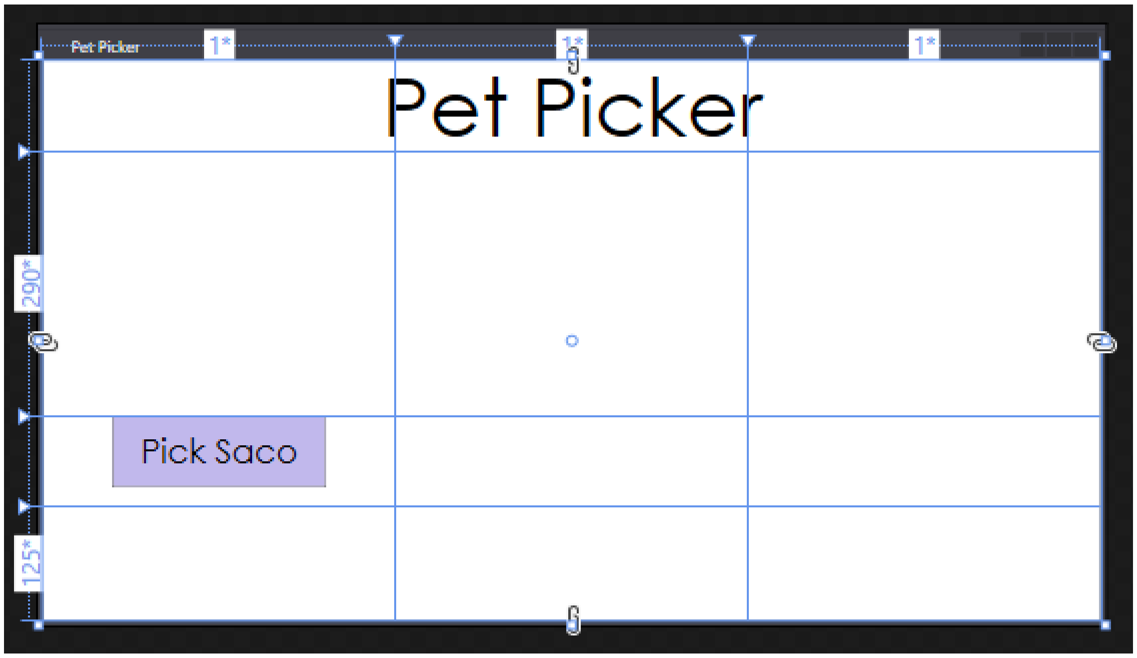
It’s placed at (0, 0) in the grid and spans 3 columns, so it takes up the first row. The HorizontalAlignment and VerticalAlignment attributes control where the element gets placed within its grid location. Try the other possible values (they pop up in intellisense after you type “HorizontalAlignment=”). What do they do? See the docs.

**Important Note!** As much as possible for this tutorial, write out the XAML when creating elements. Dragging elements from the toolbox into your app will create Margin, Width and Height properties that you’ll have to go back and delete.

Now we can add our first Pet’s image and button. Let’s start with the button.

```
<Button Name="SacoButton" Grid.Column="0" Grid.Row="2" HorizontalAlignment="Center"
  VerticalAlignment="Top" FontSize="25" Padding="20 10" Background="#FFC1B8EC">Pick Saco</Button>
```

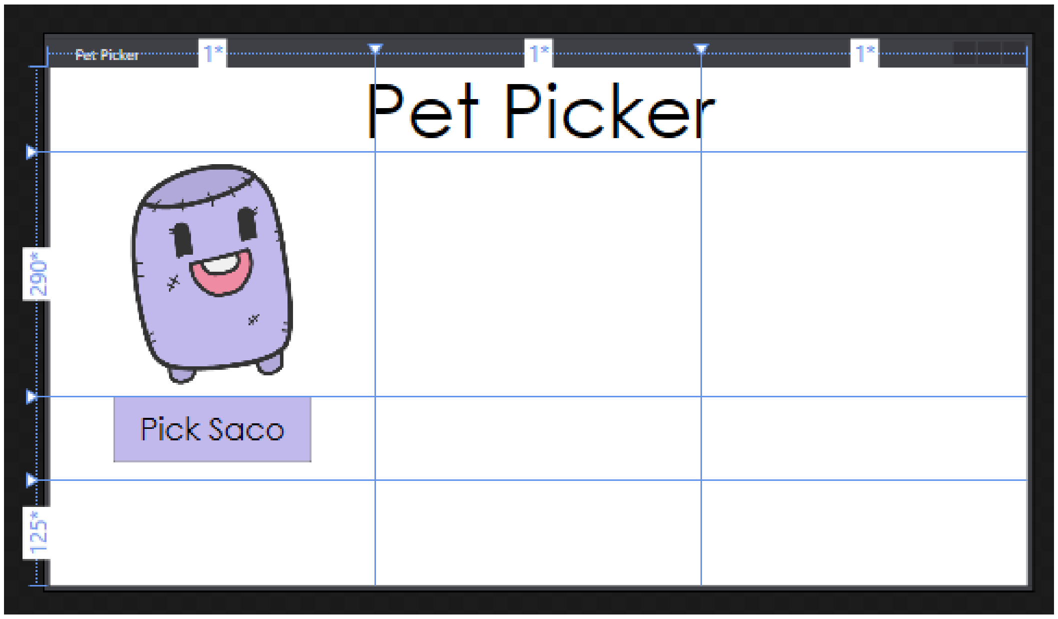
Which looks like:



There are a couple of important properties on there. First up – Padding. That adds space between the content ("Pick Saco") and the edges of the control (the button). Next up, the HorizontalAlignment and VerticalAlignment are used to push the button to the top and center of the grid cell. Last is Background, which is the color of the button. Unlike the padding and alignment attributes, Background is an attribute I'll set via the properties panel instead of coding it out manually.

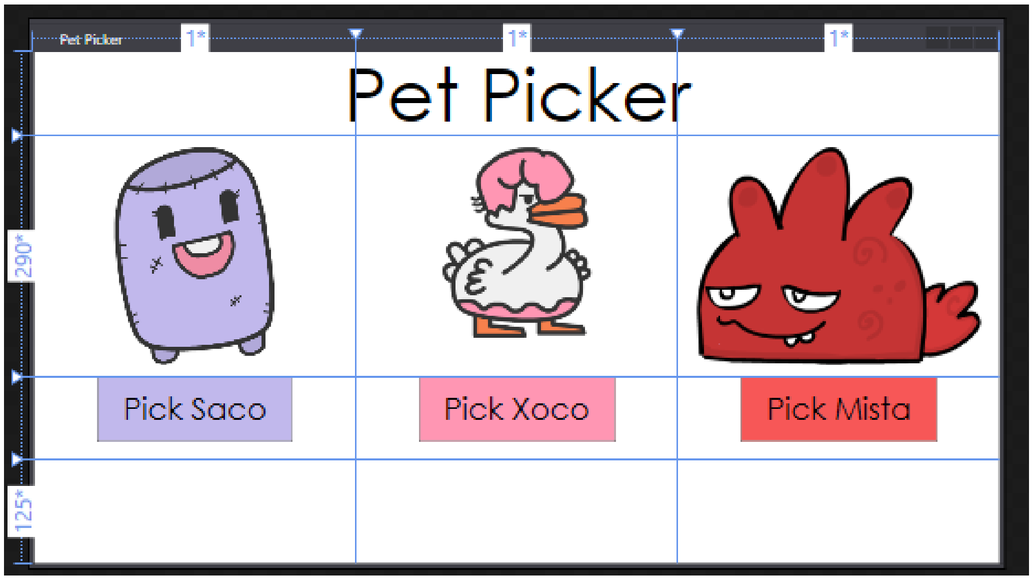
Now for an image, which we can type out in XAML below the existing button:

```
<Image Name="SacoImage" Margin="10" Grid.Column="0" Grid.Row="1"
       Source="Images/saco1-megupet.png"></Image>
```



You can create an image in a lot of different ways (see tutorial), but for this app, we just need to point the Source attribute to a location within our project. Here, I downloaded an image (from megupets) and added it to my project under an Images folder. Note: the forward slash in a path represents a folder.

Now it's just a matter of repeating the process for our next two pets. Go ahead and write out the XAML.



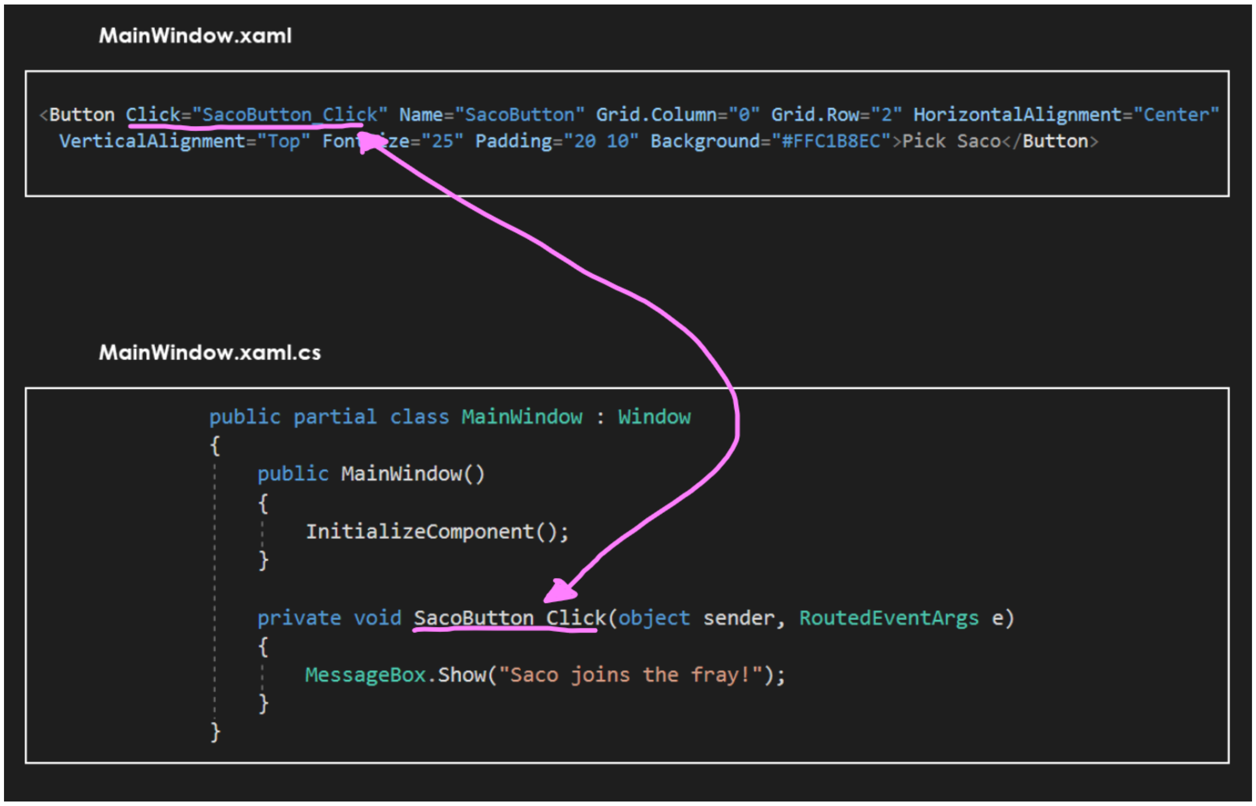
The last element in the grid is a TextBlock element that will take up the last row. This is where we'll display a little message to the player about the pet that they picked.

```
<TextBlock Name="ResultText" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="3"
  Grid.Column="0" Grid.ColumnSpan="3" TextAlignment="Center" FontSize="25" TextWrapping="Wrap">
</TextBlock>
```

**Step 4: Hooking the XAML to Code Behind**

All the important elements in your XAML should have unique Name attributes – the buttons, the images and the TextBlock. Remember, creating a Name attribute allows you to refer to the elements in your "code-behind" C#.

We want C# code to run when any of the buttons is clicked. In order to do that, we need to create some event handlers. The way event handlers work is that your C# method needs to exactly match the appropriate XAML attribute (in this case the "Click" attribute):



MainWindow.xaml

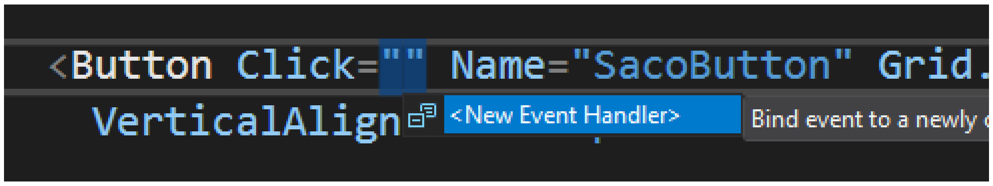
```
<Button Click="SacoButton_Click" Name="SacoButton" Grid.Column="0" Grid.Row="2" HorizontalAlignment="Center"
  VerticalAlignment="Top" FontSize="25" Padding="20 10" Background="#FFC1B8EC">Pick Saco</Button>
```

MainWindow.xaml.cs

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void SacoButton_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Saco joins the fray!");
    }
}
```

Visual Studio tries to be smart for us. If we start adding a Click attribute to one of our buttons, it will pop up an intellisense suggestion for generating a new method:



```
<Button Click="" Name="SacoButton" Grid.
  VerticalAlign
```

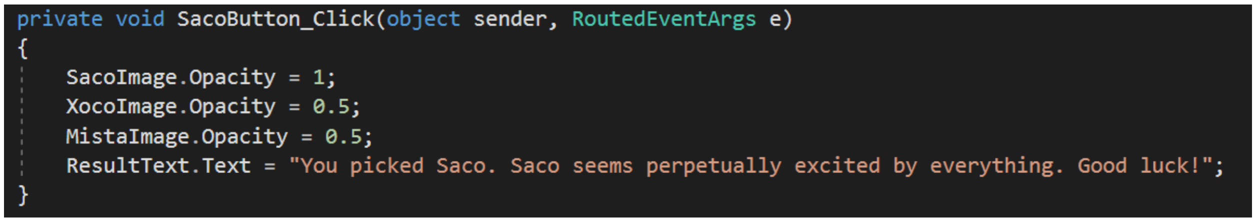
Click that, or press tab. Then flip over to your C# file to verify that it is working by displaying a MessageBox when the button is clicked. Run & test.

Once you are sure it is working, repeat the same process for each button. Each button should have a unique event handler method in MainWindow.xaml.cs when you are done.

## Step 5: Coding Time

Remember, the idea with XAML vs C# code behind is *separation of concerns*. We created our layout and style in XAML and now we can make it interactive in C#.

Inside of one of the buttons, we can do two things – "gray" out the non-selected pets and display a message about the chosen pet:



```
private void SacoButton_Click(object sender, RoutedEventArgs e)
{
    SacoImage.Opacity = 1;
    XocoImage.Opacity = 0.5;
    MistaImage.Opacity = 0.5;
    ResultText.Text = "You picked Saco. Saco seems perpetually excited by everything. Good luck!";
}
```

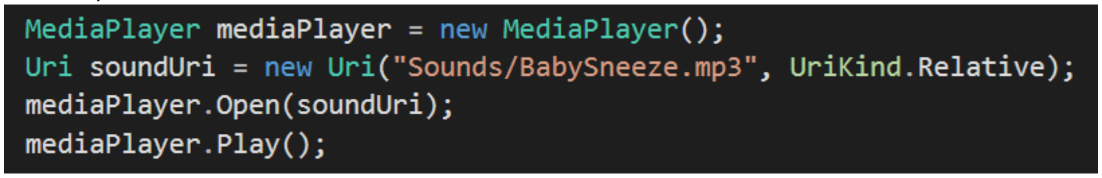
Opacity is a property that UIElements have. Setting it to 0 makes it fully transparent and setting it to 1 makes it fully opaque. Since this is the Saco button click event, we can make the other two pets half-transparent to "gray" them out.

Repeat the same process with the other two pets.

## Step 6: Extensions

Create the following extensions:

- Add a fourth pet option to the app. What do you need to change to do that?
- (Optional) Add a unique sound when each pet is picked. Below is a snippet to get you started. Unlike the SoundPlayer that we've used before which was limited to wav files, this one can handle mp3 files! See tutorial for more info. Note: you need to set the file to "Copy if Newer" in the properties window for the sound to play! You'll also want to create the MediaPlayer instance as a field of your MainWindow class – if you create it inside a click handler, it will get destroyed when the method ends.



```
MediaPlayer mediaPlayer = new MediaPlayer();
Uri soundUri = new Uri("Sounds/BabySneeze.mp3", UriKind.Relative);
mediaPlayer.Open(soundUri);
mediaPlayer.Play();
```