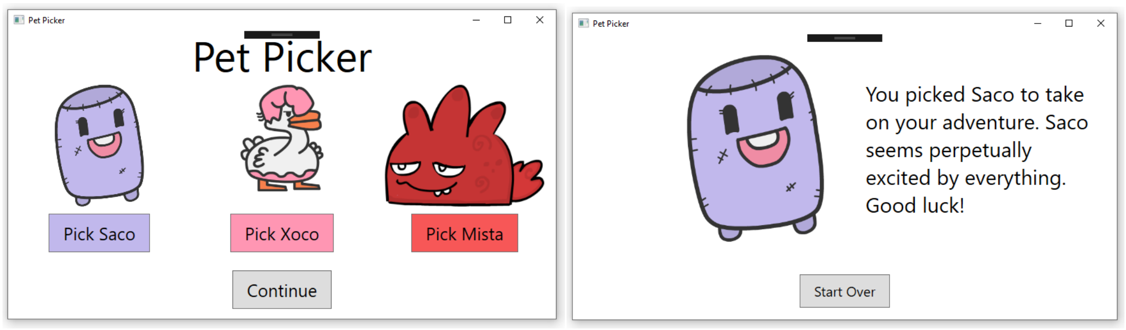
Pet Picker v2 – WPF Navigation

So far, we know how to put content on a single page within WPF – but any complex application is usually composed of multiple pages (e.g. Spotify has a playlist page, an artist page, an album page, etc.).

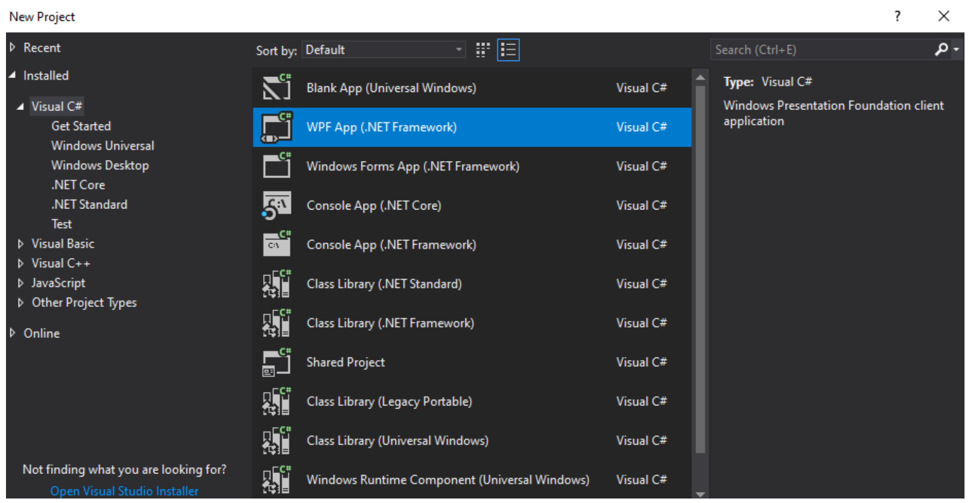
We're going to be refactoring our Pet picker to use a Frame element, which is an element that can load Pages dynamically.



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

Step 0: New Project

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

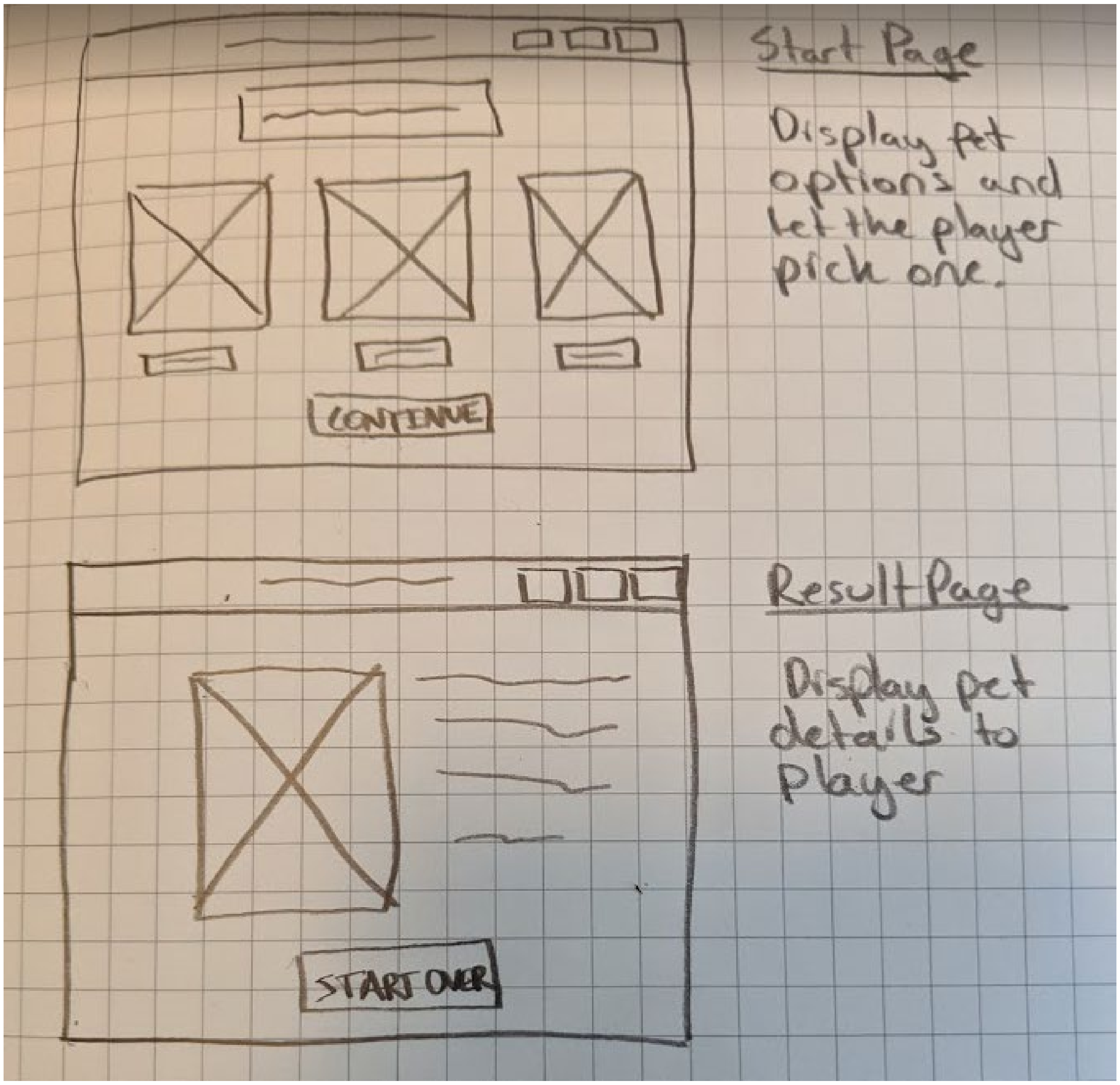

New Project
Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
.NET Core
.NET Standard
Test
Visual Basic
Visual C++
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Blank App (Universal Windows) Visual C#
WPF 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

Same as last time, the first step before touching code should be planning the layout with a wireframe. It's not supposed to be pretty or overly detailed! This helps us plan out the layout – can you picture the Grid structure we'll use on these two pages?


Start Page
Display pet options and let the player pick one.
CONTINUE
Result Page
Display pet details to player
START OVER

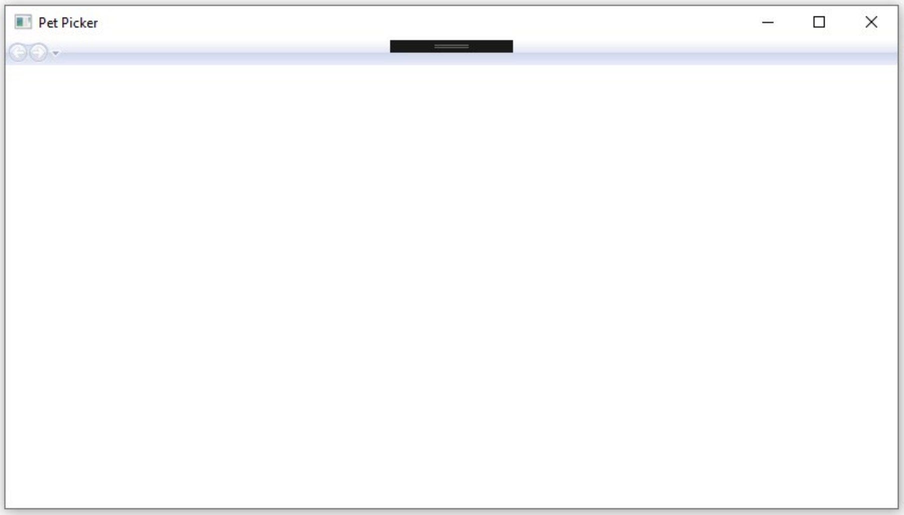
Step 2: Frames and Pages

There are a couple ways to create a navigation system in WPF. We're going to use a Frame, an element that dynamically loads content. Also, see the Microsoft tutorial on navigation.

Inside of our MainWindow.xaml, we'll create a Frame element. This will host our content. Then we'll create two new pages – Start.xaml and Results.xaml. We'll set up our logic so that the Frame displays either Start.xaml or Results.xaml depending on where we are in the flow of the application. It will show Start.xaml initially, and then when the player chooses a pet and hits the continue button, we'll show Results.xaml.

Let's create the frame first. Inside of MainWindow.xaml:

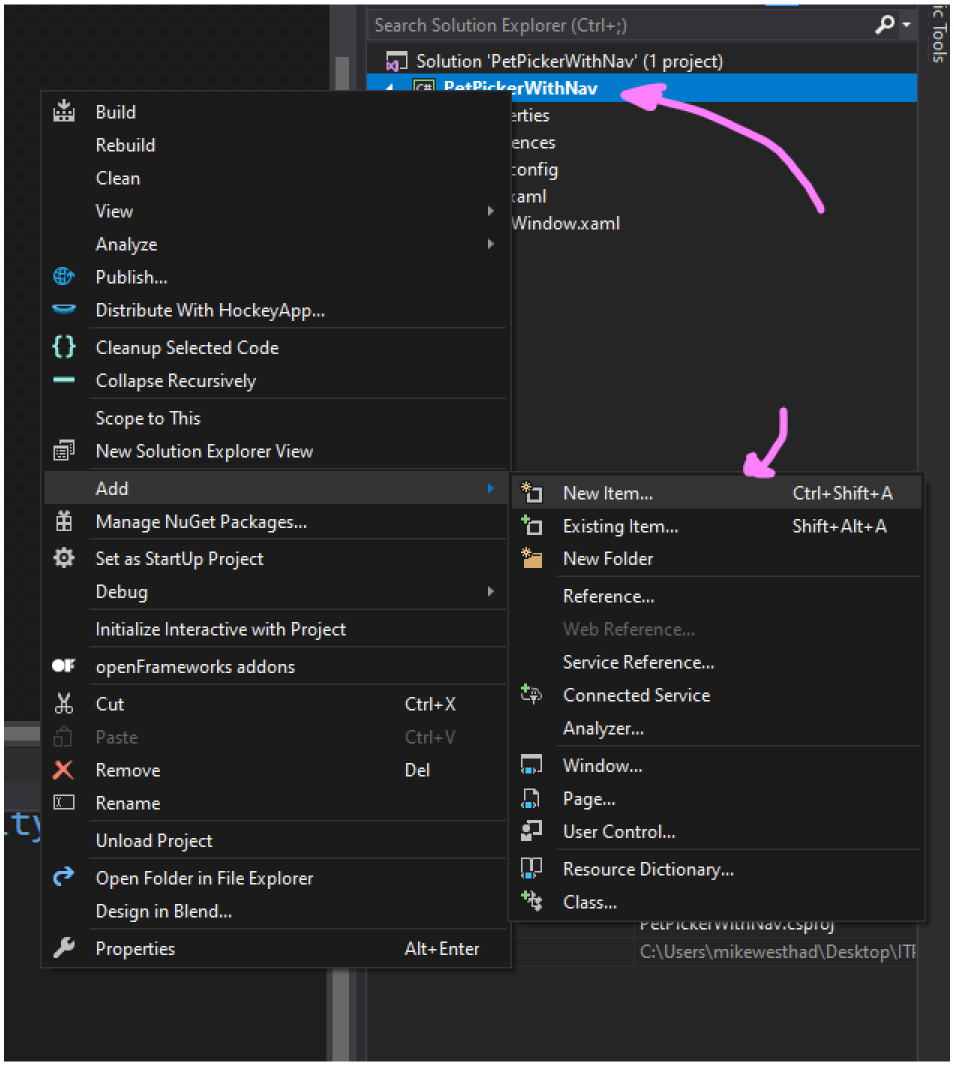
```
<Grid>
    <Frame Name="Navigation" NavigationUIVisibility="Visible"></Frame>
</Grid>
```



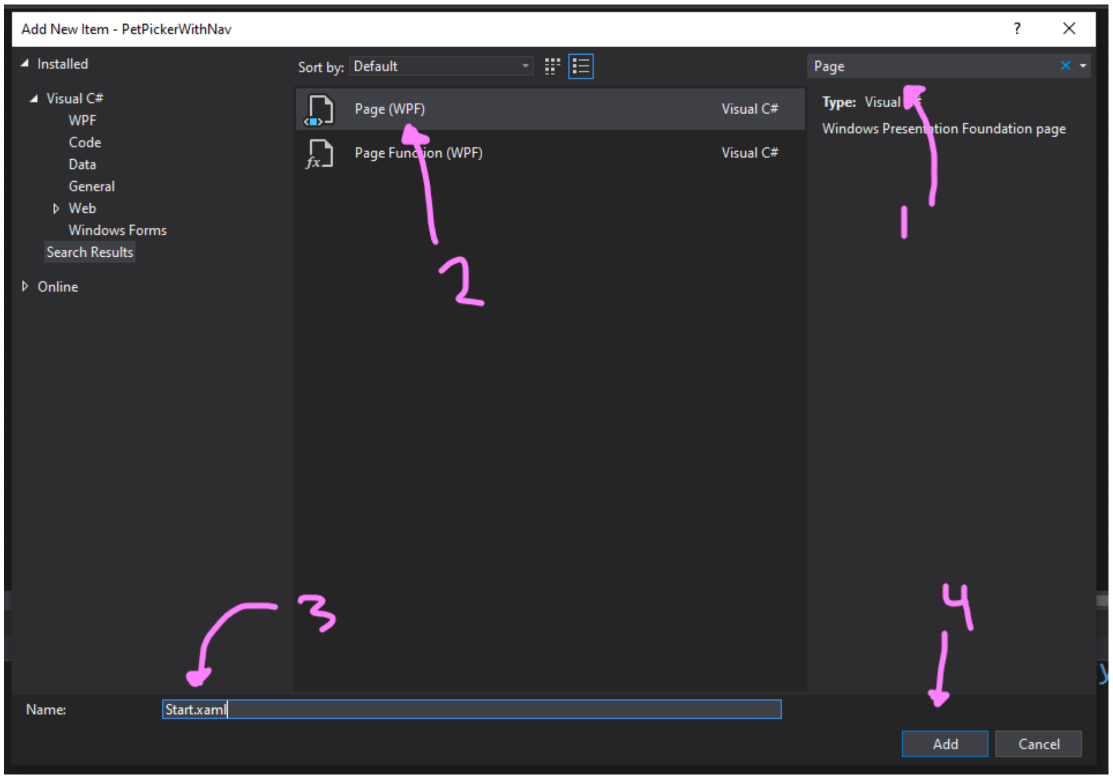
This turns our window into a mini Internet Explorer window with a pretty-dated looking top bar with forward and backward buttons. Luckily, we can turn that off by changing NavigationUIVisibility to "Hidden" – go ahead and do that now.

And that's all we have to do to the Window! Now, let's create two pages that we'll be able to dynamically insert into that Frame.

Right click on your project in the Solution Explorer and click "New Item" (or Ctrl + Shift + A):

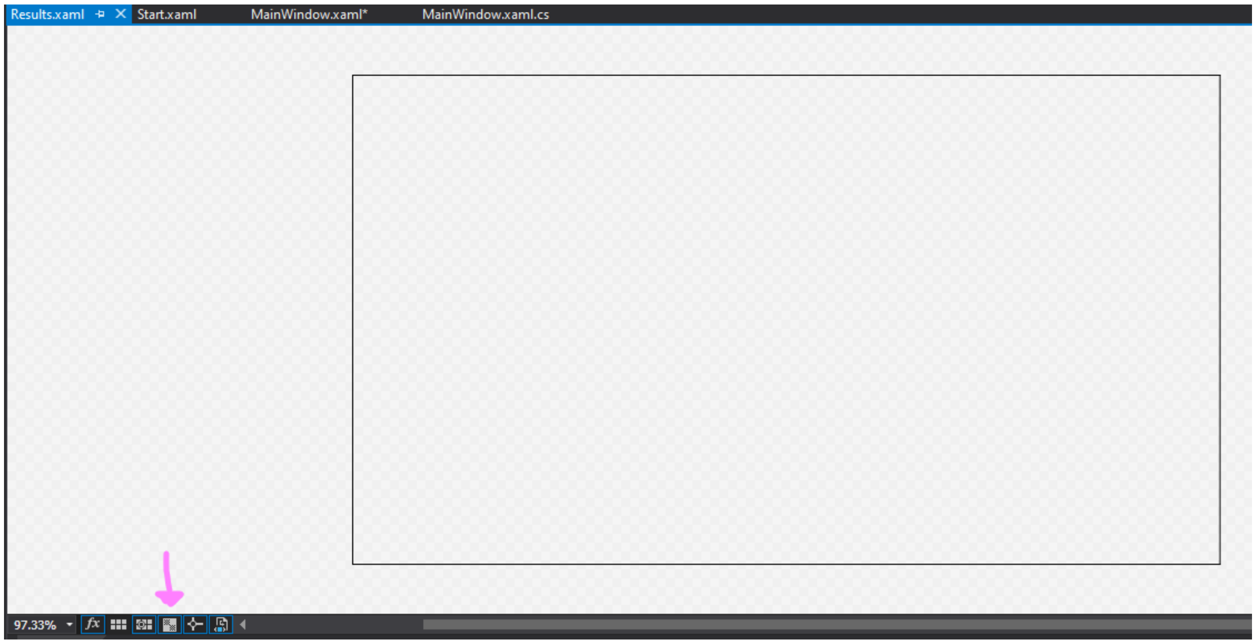


Then search for the "Page (WPF)" item, name it Start.xaml and click Add:



Repeat that process to create a "Results.xaml" page. Now you can click on your Start or Results page in your Solution Explorer to open it up.

Quick note: a page doesn't have a background color by default, so you can toggle between previewing your page on a white background or on a black background with the following button:

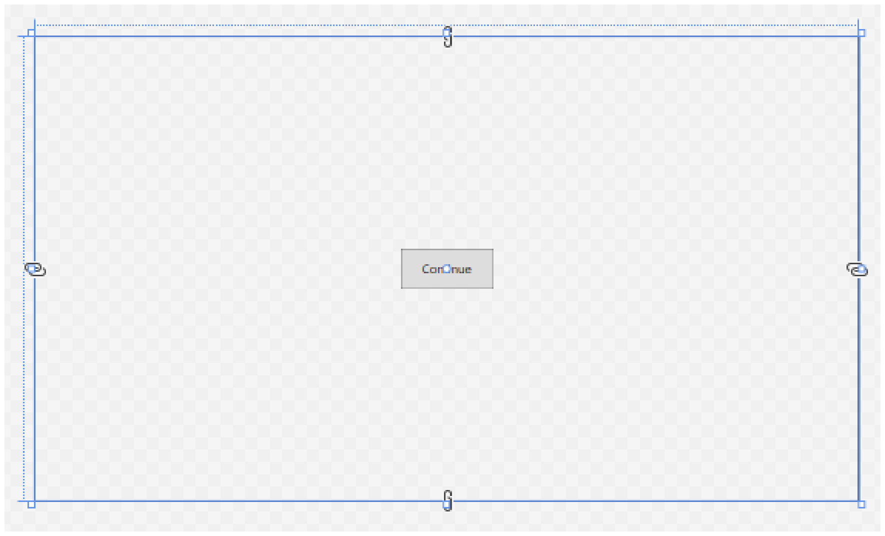


Step 3: Testing out the Pages

Run your app. Nothing shows up right? That Frame within MainWindow is blank currently. We're going to fill in some skeleton content on our pages and get navigation between pages working so that we can actually see something within the Frame.

Open up Start.xaml (double click in Solution Explorer) and add a button (by typing it in the XAML!). This will be the pet picker page where the player picks a pet and then hits a continue button to go to the next page. It should look something like:

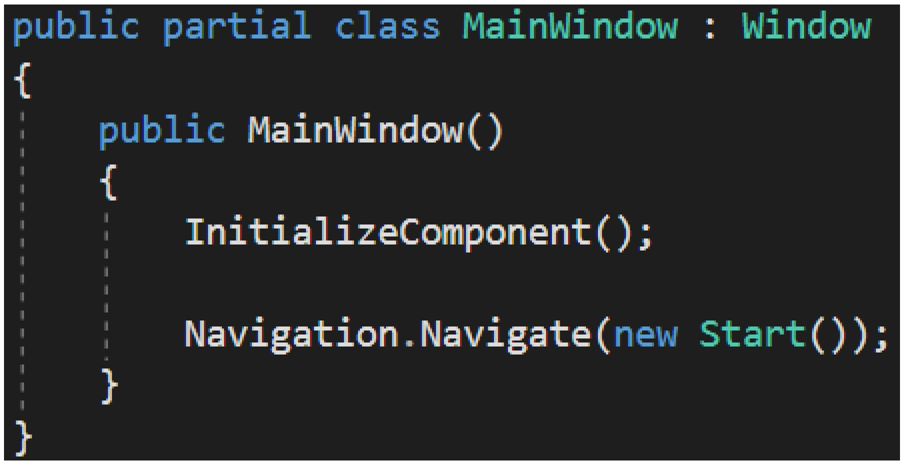
```
<Grid>
    <Button Name="ContinueButton" HorizontalAlignment="Center" VerticalAlignment="Center" Padding="20 10">
        Continue
    </Button>
</Grid>
```



Make sure you give the button a descriptive name. Check your knowledge – what are the alignment and padding attributes doing? Change their values and see what happens.

It may not look like much, but it's enough for now. Open up Results.xaml and repeat the processing by adding a start over button. This page will display the player's chosen pet and some more detailed about it. The start over button will take the player back to Start.

With those two pages roughly set up, we can now hook up the navigation. Head to the MainWindow.xaml.cs code behind file and after InitializeComponent, use the following code to instantiate a page and navigate to it:



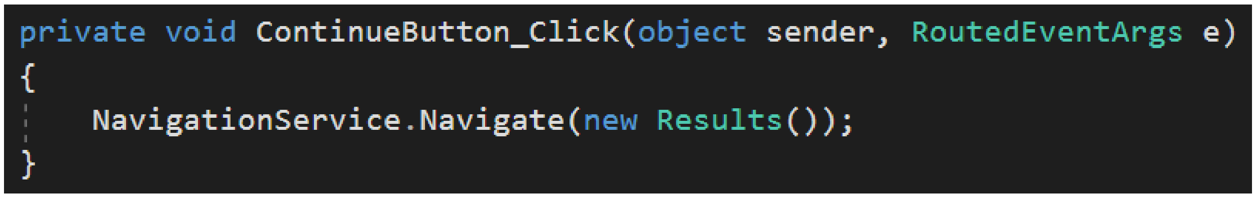
```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        Navigation.Navigate(new Start());
    }
}
```

Note: “Navigation” is the name of our frame within the MainWindow, so we can use the variable “Navigation” in our code behind. Navigation is a variable of type System.Windows.Controls.Frame, which has a Navigate method.

Run your app, you should see the continue button pop up on screen.

Open up your code behind C# file for your Start page. You’ll notice that this file contains a class that inherits from Page (unlike your MainWindow, which inherits from Window). We can’t use “Navigation” here to refer to the Frame element. Our pages don’t have direct access to the named elements within the MainWindow. Instead, if you look through the docs for Page, you’ll see that it has a NavigationService property, which has a similar Navigate method. We’re going to use that.

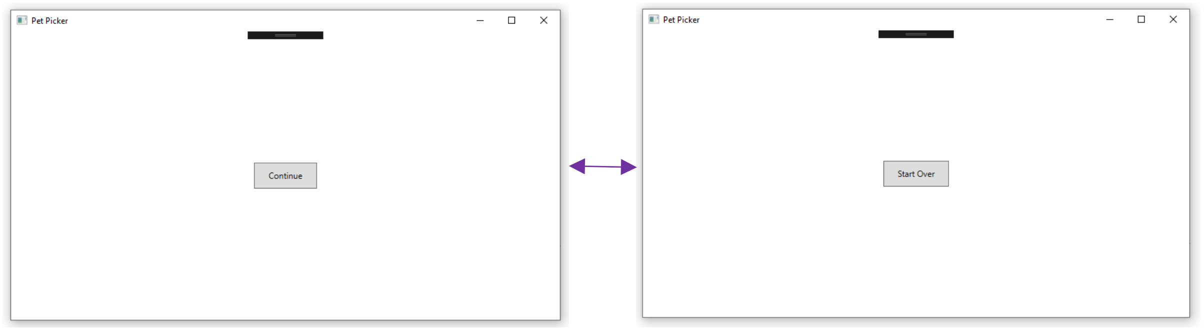
Add a Click event handler to your continue button on the Start page (remember the method name in your C# needs to match the Click attribute in your XAML). Add the following to your click handler:



```
private void ContinueButton_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new Results());
}
```

Run your app now, you should be able to navigate from your start page to your results page now. But the start over button doesn’t work yet!

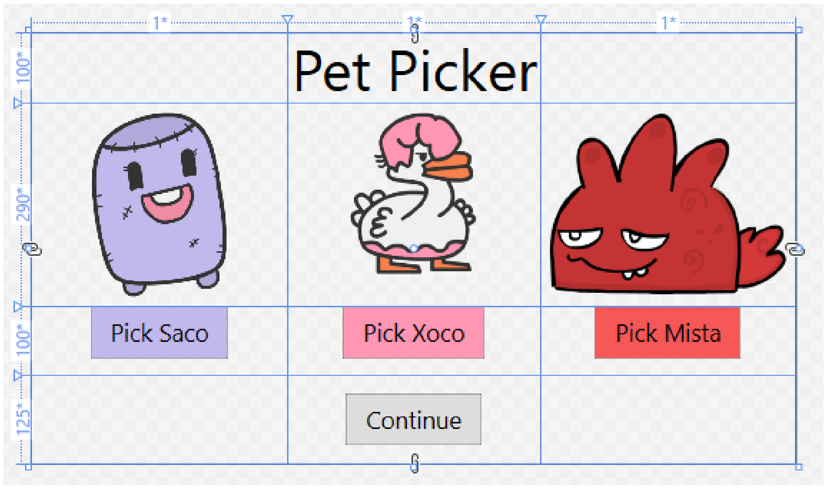
Repeat this process for your Results page. Add a click handler to your start over button that navigates back to the start page. You should have two pages that you can endlessly cycle between:



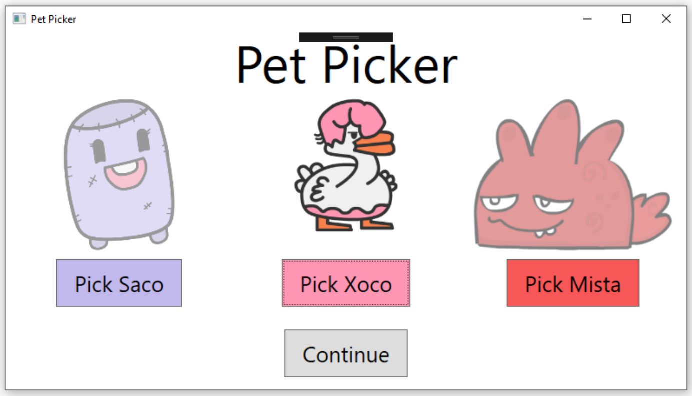
Test and run! It may not look like much yet but having multiple pages is really powerful. With this knowledge along – laying out pages with images/buttons on separate pages and navigating between them – you could rebuild a lot of your COYA game in a graphical form.

Step 4: Filling out Start and Storing Data

Now it's time to fill out our start page. We did most of this already on the last version of the pet picker, so bring over your grid and elements. You don't need the Results TextBlock we created in that last version – instead your Continue button goes in your last row. It should look like:

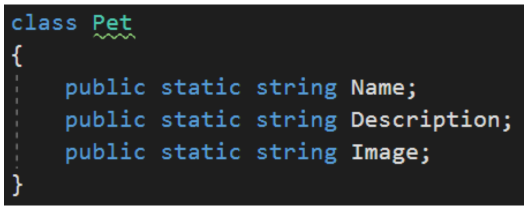


Bring over your logic from last time too – when a player clicks on any of the pet buttons, the images should still change opacity:



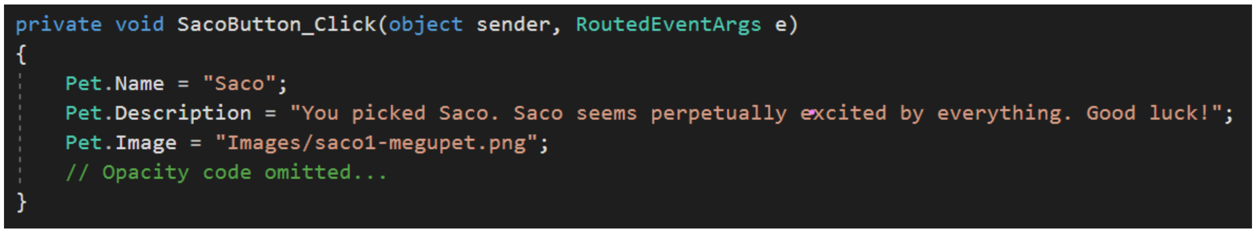
Now the challenge is to figure out how to store which pet has been selected so that when we get to the results page, we have a pet whose detailed information we can display.

There are a number of ways to do that, but one simple way is for us to create a static class called Pet in our project to hold on to the selected pet's info:



```
class Pet
{
    public static string Name;
    public static string Description;
    public static string Image;
}
```

Since Name, Description and Image are all public and static, we can access them from anywhere – including our start and results page. So for instance, we can update our click handler for Saco to look like:



```
private void SacoButton_Click(object sender, RoutedEventArgs e)
{
    Pet.Name = "Saco";
    Pet.Description = "You picked Saco. Saco seems perpetually excited by everything. Good luck!";
    Pet.Image = "Images/saco1-megupet.png";
    // Opacity code omitted...
}
```

The name, description and image path should match whatever you want your pet options to be in the app. Note: you'll have to add your images again since this is a new project.

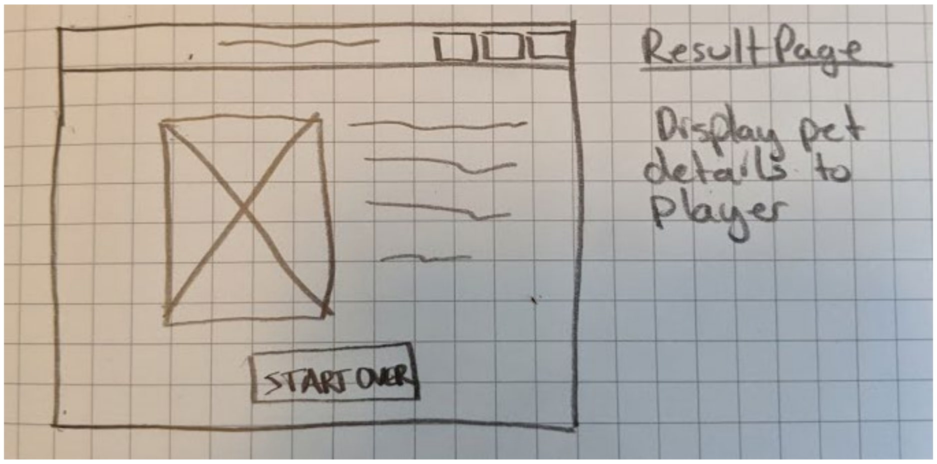
Repeat this process for your other two pet buttons. Inside of your continue button click handler, you should be able to print out the pet name and description now.

Step 5: Filling out Results

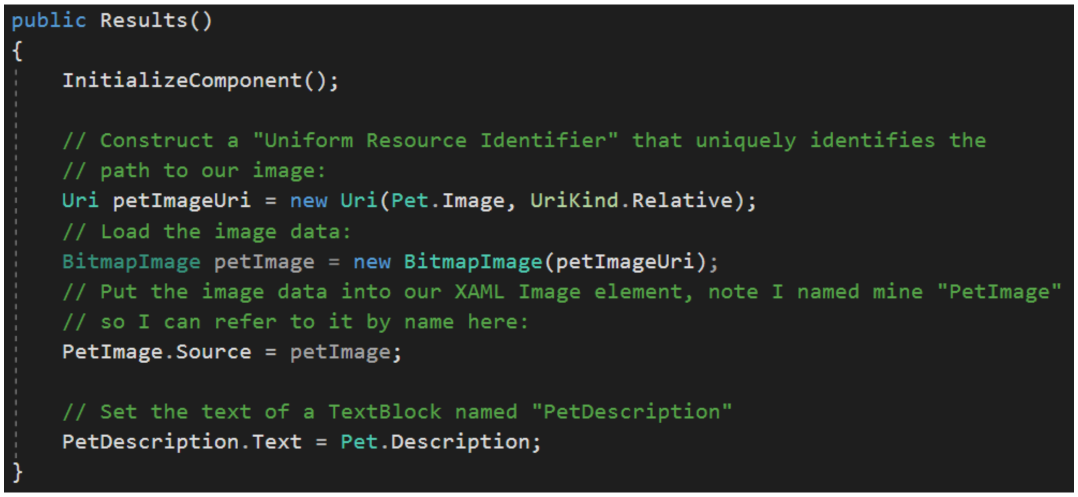
Now it's your turn to lay out the Results page. We've got the Name, Description and Image URI saved, so you'll at least want one TextBlock and an Image on the page.

Make sure you give each element a unique Name since we'll be changing the content dynamically via the C# code behind.

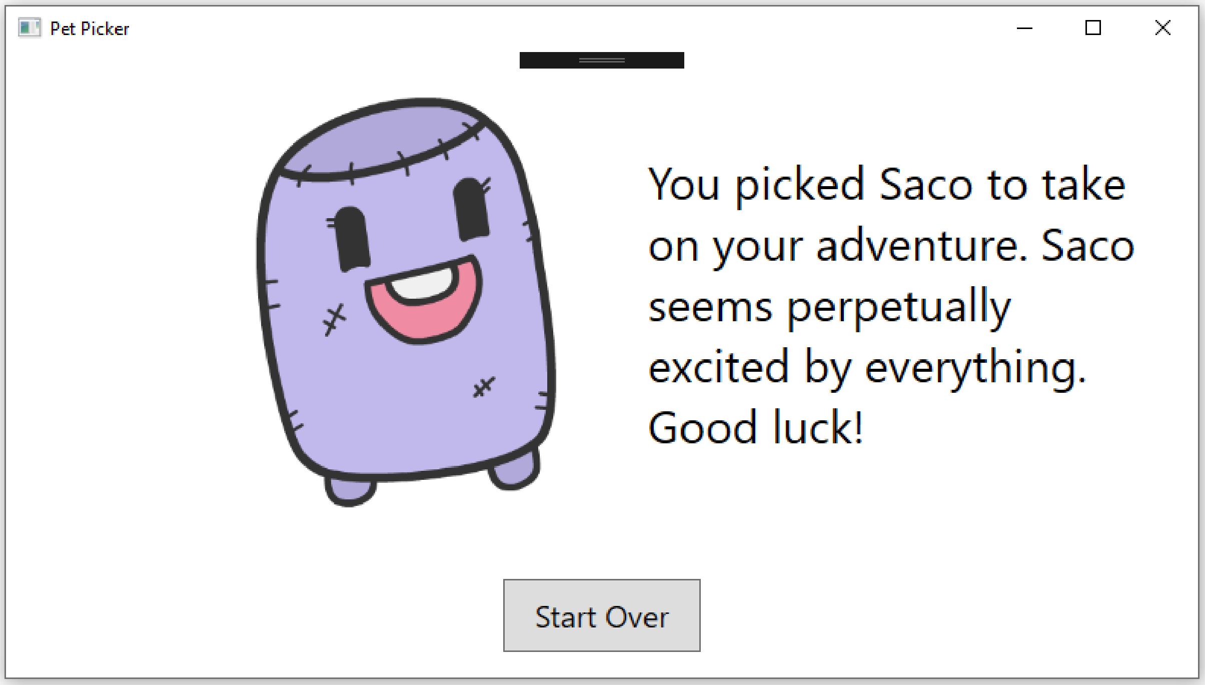
You can use this wireframe as a guide. The only requirement is that you don't place things by dragging & dropping from the toolbox!



Inside of the constructor on the Results page, you should be able to do something like this. Read the comments to follow what each line does:



```
public Results()
{
    InitializeComponent();

    // Construct a "Uniform Resource Identifier" that uniquely identifies the
    // path to our image:
    Uri petImageUri = new Uri(Pet.Image, UriKind.Relative);
    // Load the image data:
    BitmapImage petImage = new BitmapImage(petImageUri);
    // Put the image data into our XAML Image element, note I named mine "PetImage"
    // so I can refer to it by name here:
    PetImage.Source = petImage;

    // Set the text of a TextBlock named "PetDescription"
    PetDescription.Text = Pet.Description;
}
```



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

Step 7: Extensions (Optional)

Pat yourself on the back, you've made your first multi-page application. Now it's time to customize it and make it your own:

- What happens if someone doesn't pick a pet before hitting continue on the start page? Don't let them do that.
- Add a page in-between the Start and Results page where you can give your pet a nickname. (Think back to the tip calculator – how did you get input from the user?) Display the nickname on the final page.
- It's tedious to have to style your buttons with font size and padding each time you add a new button. Read through this tutorial on styling and use the application-wide method to style your buttons.