Creating User Interfaces with SplashKit

Introduction

The purpose of creating a user interface is to improve the usability of programs or applications by making them interactive and easy to use. With SplashKit, we can create various user interface elements such as buttons, textboxes and sliders that respond to user interactions. SplashKit provides event handling method that captures these interactions and triggers the appropriate operations.

User interface elements can be arranged in two main ways:
- fixed layout where elements are positioned at specific coordinates.
- custom layout where elements are dynamically positioned based on predefined rules such as using the "SplitIntoColumns" method.

In this guide, we will walk through the steps to build a user interface using both fixed and custom layout, along with its application in C# code.

Steps:

1. Set up the "Program" class

Create a "Program" class that initialises and runs the "InteractiveUI" class. This class will serve as the entry point for your application.

2. Design the "InteractiveUI" class

The "InteractiveUI" class manages the user interface components. Here is how to structure it:

2.1. Fields
- Define fields to store input values.

2.2. Constructor
- The constructor initialises the window, assigns input values and sets the user interface's background style.

2.3. "Run" method
- The "Run" method uses a while loop to continuously process events, create user interface elements, handle user interaction and refresh the screen. This ensures the interface continuous updates based on user inputs.

2.4. User interface elements
- Create user interface elements including buttons, textboxes and sliders to allow users to interact with the application.

2.5. Layout management

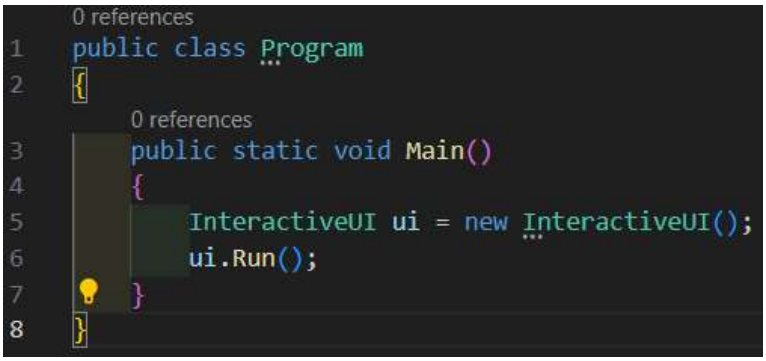
- The "Run" method arranges user interface elements inside a panel, a movable window where user interface components are placed. You can either use a fixed layout where elements are position at predefined location, or a custom layout where elements are automatically positioned based on rules. For example, we can use the "SplitIntoColumns" method to dynamically adjust the element positions.

Application

In this section, we will implement the steps discussed above in C#.

1. Creating "Program" class

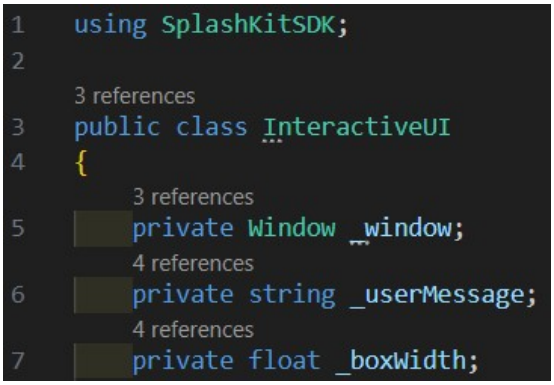
This class is the entry point of the application and is responsible for initialising and running the user interface.



2. Creating "InteractiveUI" class

2.1 Fields

Here we define the fields to store user input and the width of the textbox.



2.2 Constructor

The constructor initialises the window, values and sets the background style for the user interface.

```
1 reference
public InteractiveUI()
{
    _window = new Window("Interactive User Interface", 900, 700);
    _userMessage = "Type here...";
    _boxWidth = 165f;

    SplashKit.SetInterfaceStyle(InterfaceStyle.FlatDarkStyle);
}
```

2.3 “Run” method

The Run method contains the while loop that continuously processes events and updates the screen.

```
1 reference
public void Run()
{
    while (!_window.CloseRequested)
    {
        SplashKit.ProcessEvents();
        SplashKit.ClearScreen(Color.White);
```

2.4 User interface elements

We now create the buttons, a textbox and a slider inside the panel. For each user interface element, we define their coordinates, height, and width within the panel.

```
if (SplashKit.StartPanel("Panel 2", SplashKit.RectangleFrom(40, 40, 350, 250)))
{
    if (SplashKit.Button("Button1", SplashKit.RectangleFrom(20, 20, 165, 25)))
    {
        Console.WriteLine("Button1 is clicked");
    }
    if (SplashKit.Button("Button2", SplashKit.RectangleFrom(20, 60, 165, 25)))
    {
        Console.WriteLine(_userMessage);
    }

    _userMessage = SplashKit.TextBox(_userMessage,SplashKit.RectangleFrom(20, 100, _boxWidth, 25));

    _boxWidth = SplashKit.Slider(_boxWidth, 125, 300, SplashKit.RectangleFrom(20, 140, 165, 20));

    SplashKit.EndPanel("Panel 2");
}
SplashKit.DrawInterface();
SplashKit.RefreshScreen();
}
```

- Two buttons are created. When "Button1" is clicked, it displays "Button1 clicked" in the console. When "Button2" is clicked, it displays the message the user has typed in the textbox.
- The textbox allows the user to input a message. As the user types, the message dynamically updates the “_userMessage” field, which can be displayed in the console when "Button2" is clicked.
- The slider allows the user to adjust the width of the textbox. The width can be dynamically changed within a range of 125 to 300 pixels, providing flexible control on the textbox's size.

2.5 Layout Management

We can use a custom layout to dynamically arrange elements inside the panel.

```
28          if (SplashKit.StartPanel("Panel 1 Custom Layout", SplashKit.RectangleFrom(40, 40, 350, 250)))
29          {
30              SplashKit.StartCustomLayout();
31              SplashKit.SplitIntoColumns(2);
32              SplashKit.EnterColumn();
33              SplashKit.SetLayoutHeight(0);
34
35              if (SplashKit.Button("Button1"))
36              {
37                  Console.WriteLine("Button1 is clicked");
38              }
39              if (SplashKit.Button("Button2"))
40              {
41                  Console.WriteLine(_userMessage);
42              }
43
44              _userMessage = SplashKit.TextBox(_userMessage, SplashKit.RectangleFrom(0, 50, _boxWidth, 25));
45
46              _boxWidth = SplashKit.Slider("Slider", _boxWidth, 125, 300);
47
48              SplashKit.LeaveColumn();
49              SplashKit.ResetLayout();
50              SplashKit.EndPanel("Panel 1 Custom Layout");
51          }
52          SplashKit.DrawInterface();
53          SplashKit.RefreshScreen();
54      }
```

This shows how SplashKit arranges user interface elements using the “SplitIntoColumns(2)” method. The elements are automatically positioned into two columns inside the panel. As the panel size increases, the buttons will resize to occupy half the panel's width.

Running the program

Here is a screenshot of the program running.