Layout Management in Various Libraries

1. Qt

These are some of the layout management classes provided by Qt to arrange various UI widgets.

QBoxLayout	Lines up child widgets horizontally or vertically
QButtonGroup	Container to organize groups of button widgets
QFormLayout	Manages forms of input widgets and their associated labels
QGraphicsAnchor	Represents an anchor between two items in a QGraphicsAnchorLayout
QGraphicsAnchorLayout	Layout where one can anchor widgets together in Graphics View
QGridLayout	Lays out widgets in a grid
QGroupBox	Group box frame with a title
QHBoxLayout	Lines up widgets horizontally
QLayout	The base class of geometry managers
QLayoutItem	Abstract item that a QLayout manipulates
QSizePolicy	Layout attribute describing horizontal and vertical resizing policy

QSpacerItem	Blank space in a layout
QStackedLayout	Stack of widgets where only one widget is visible at a time
QStackedWidget	Stack of widgets where only one widget is visible at a time
QVBoxLayout	Lines up widgets vertically
QWidgetItem	Layout item that represents a widget

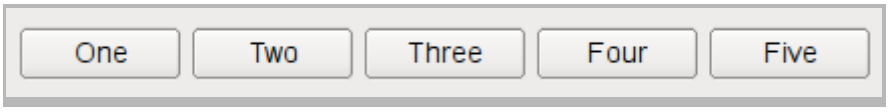
Code Snippets for Qt layout

The following code snippet shows how to stack different UI widgets horizontally:

```
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton("One");
QPushButton *button2 = new QPushButton("Two");
QPushButton *button3 = new QPushButton("Three");
QPushButton *button4 = new QPushButton("Four");
QPushButton *button5 = new QPushButton("Five");

QHBoxLayout *layout = new QHBoxLayout(window);
layout->addWidget(button1);
layout->addWidget(button2);
layout->addWidget(button3);
layout->addWidget(button4);
layout->addWidget(button5);

window->show();
```



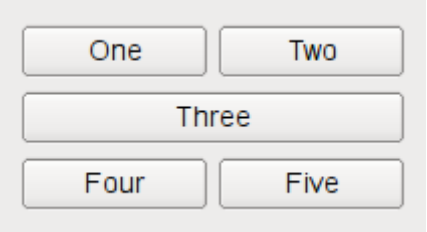
Horizontally stacked UI widgets

The following code snippet shows how to place the widgets in a grid fashion:

```
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton("One");
QPushButton *button2 = new QPushButton("Two");
QPushButton *button3 = new QPushButton("Three");
QPushButton *button4 = new QPushButton("Four");
QPushButton *button5 = new QPushButton("Five");

QGridLayout *layout = new QGridLayout(window);
layout->addWidget(button1, 0, 0);
layout->addWidget(button2, 0, 1);
layout->addWidget(button3, 1, 0, 1, 2);
layout->addWidget(button4, 2, 0);
layout->addWidget(button5, 2, 1);

window->show();
```



Widgets placed in a grid

Links to the official documentation:-

- Examples for layouts

2. imgui:

These are the classes provided by imgui to implement grid like layout:-

1. Columns:

Columns can be used to add widgets horizontally till a set number of column as soon as the last column is reached it resets it to first column, thus creating a grid like pattern
Code snippet for column:-

```
ImGui::Columns(2, nullptr, false); // You set 2 columns
                                ImGui::SetColumnOffset(1, 200);
                                {
                                        ImGui::Separator();
                                        ImGui::Text("Visuals");
                                        ImGui::Separator();

                                        ImGui::CheckboxText("Enable Visuals");
                                        ImGui::CheckboxText("Chams");
                                        ImGui::CheckboxText("Skeleton");
                                        ImGui::CheckboxText("Box");

                                        ImGui::NextColumn(); // You go into 2nd column

                                        ImGui::Separator();
                                        ImGui::Text("Removals");
                                        ImGui::Separator();

                                        ImGui::CheckboxText("No Hands");
                                        ImGui::CheckboxText("No Smoke");
                                        ImGui::CheckboxText("No Flash");
                                        ImGui::CheckboxText("No Sky");

                                        ImGui::NextColumn(); // You put yourself back in the first
column
                                }
```

Live showcase of imgui widgets can be found here.

Imgui also has a method to group different UI widgets together. Below is the code snippet from grouping widgets together.

```
imgui.begin_group()
imgui.text("First group (buttons):")
imgui.button("Button A")
```

```
imgui.button("Button B")
imgui.end_group()

imgui.same_line(spacing=50)

imgui.begin_group()
imgui.text("Second group (text and bullet
texts):")
imgui.bullet_text("Bullet A")
imgui.bullet_text("Bullet B")
imgui.end_group()
imgui.end()
```

Links to official documentation:-

- Grouping UI widgets
- Implementation of grouping
- Interesting discussion on grid

3. Pyimgui

Pyimgui provides python bindings for the imgui c++ library. All the core implementations remain the same. Here's a quick example of columns implemented under pyimgui:-

```
imgui.begin("Example: Columns -
File list")
imgui.columns(4, 'filelist')
imgui.separator()
imgui.text("ID")
imgui.next_column()
imgui.text("File")
```

```
imgui.next_column()
imgui.text("Size")
imgui.next_column()
imgui.text("Last Modified")
imgui.next_column()
imgui.separator()
imgui.set_column_offset(1, 40)
```

Links to the official documentation:-

- Implementation/Example for columns

4. wxPython:

wxPython also supports various layouts in which widgets can be placed. Although, we can also opt in for absolute placements i.e. place the widgets as per specified coordinates. Here are some classes:-

1. wx.BoxSizer
2. wx.StaticBoxSizer
3. wx.GridSizer
4. wx.FlexGridSizer
5. wx.GridBagSizer

Below is a code snippet for wx.BoxSizer:

```
import wx


class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title)

        self.InitUI()
        self.Centre()

    def InitUI(self):

        panel = wx.Panel(self)

        panel.SetBackgroundColour('#4f5049')
        vbox = wx.BoxSizer(wx.VERTICAL)

        midPan = wx.Panel(panel)
        midPan.SetBackgroundColour('#ededed')

        vbox.Add(midPan, wx.ID_ANY, wx.EXPAND | wx.ALL, 20)
        panel.SetSizer(vbox)


def main():

    app = wx.App()
    ex = Example(None, title='Border')
    ex.Show()
    app.MainLoop()


if __name__ == '__main__':
    main()
```

wxPython also supports nested grids i.e. one can pass one grid layout to another grid layout which enables the users to create very complex UI's

Links to official documentation:-

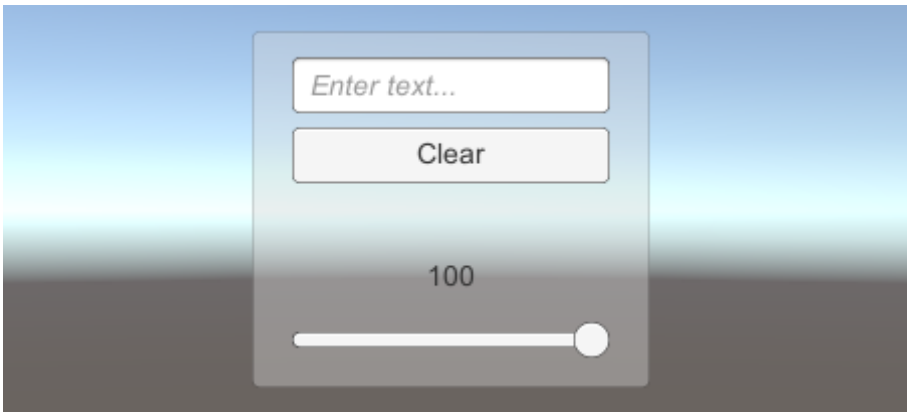
- Examples for layouts

Layout Engine In Unity

The layout engine is a part of unity's UI toolkit. It helps in placing visual elements based on layout and styling properties that are loosely based on CSS properties.

The layout engine is based on Yoga Layout which is an open source layout engine.

Unity's main UI is GameObject oriented UI which essentially means that the UI can be dragged and dropped into the scene and there is no code that initializes the UI. Below is an example:-



On top of the main UI, unity also supports **Immediate Mode GUI (IMGUI).** Which means all the UI is driven from code. Below is an example:-


```
void OnGUI() {
      if (GUILayout.Button("Press Me"))
          Debug.Log("Hello!");
  }
```

The layout engine only works with unity's main UI components and not the IMGUI. **Below are some behaviours of Unity's layout engine:-**

- A container distributes its children vertically.
- The position of a container rectangle includes its children's rectangles. This behaviour can be restricted by other layout properties.
- A visual element with text uses the text size in its size calculations. This behaviour can be restricted by other layout properties.

These changeable/fluid properties can be modified by a Style Sheet specific to Unity i.e. USS (Unity Style Sheets). These sheets are similar to CSS and almost all properties remain the same with some minor differences.

The following list provides tips on how to use the layout engine:

- Set the `width` and `height` to define the size of an element.
- Use the `flexGrow` property (in USS: `flex-grow: <value>;`) to assign a flexible size to an element. The value of the `flexGrow` property acts as weighting when the size of an element is determined by its siblings.
- Set the `flexDirection` property to row (in USS: `flex-direction: row;`) to switch to a horizontal layout.
- Use relative positioning to offset an element based on its original layout position.
- Set the `position` property to `absolute` to place an element relative to its parent position rectangle. In this case, it does not affect the layout of its siblings or parents.

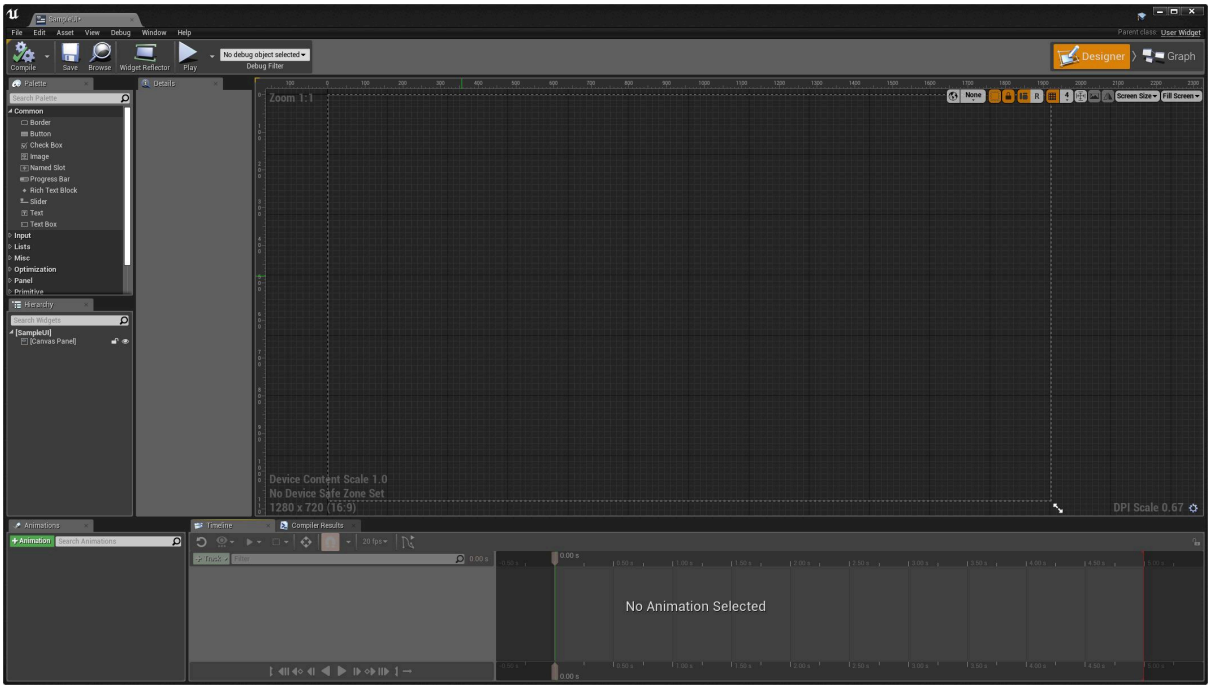
These Style Sheets are parsed and applied to a particular UI element by a custom parser implemented under Unity's UI toolkit.

Links to official documentations:-

- Unity UI
- Yoga Layout Docs
- Creating UI in Unity
- UI Toolkit
- Layout Engine
- USS (Unity Style Sheets)
- Immediate Mode GUI (IMGUI)

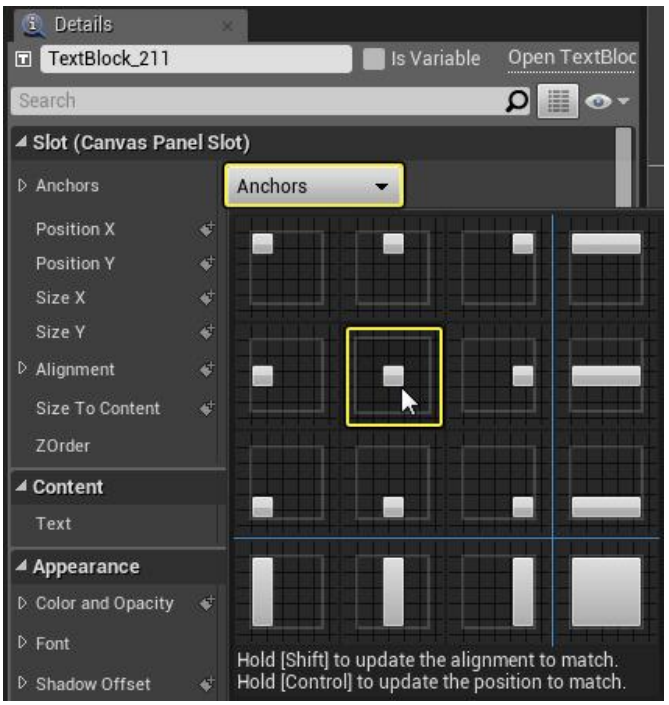
UI/Layout in Unreal Engine

Unreal Engine has a totally different solution to UI and layout. Like everything else in the engine, UI can be created from a widget blueprint. After creating and opening the UI blueprint, this menu opens:-



Here, different sub-widgets are present under the Common label. These can be dragged and dropped into the editor.

To actually have some layout in the UI widgets, UE provides anchors for the sub-widgets. The sub-widgets then maintain their positions in the widget w.r.t the anchor. This is how it looks:-



The UI widgets can also be made responsive i.e. react and adapt to window size. This is done checking the respective option in the editor. After checking the option, the UI responds to window resizing.

Links to official docs/tutorials:-

[Tutorial to create custom UI]

[Scale UI w.r.t screen size]