EXTENDING THE EDITOR WITH UI TOOLKIT

Arto Koistinen / April 2023 Unity Meetup

WHO AM I?

- Using Unity since early 2009
- Won 2010 Best Mobile Game Unity Award for Rimelands: The Hammer of Thor
- First extension I wrote was a simple level editor for the above game
- Available for consulting!

ABOUT THIS TALK

- I'm not going to teach you how to program or even how to use UI Toolkit on a concrete level
  - There are better resources for that
- Overview of UI Toolkit
- My experiences on working with UI Toolkit
- Some concepts I felt were not adequately explained in Unity's own videos
- Real-life examples of using UI Toolkit and editor extensions

THE CASE FOR EDITOR EXTENSIONS

- One of the most powerful features of Unity since the beginning
- Quick way to automate project-specific tasks
  - Generate ID numbers
  - Batch edit assets
  - Collect data
- Created within Unity with the same language and tools as the runtime code
- Consider cost:
  - Time spend developing vs time saved *vs nerves saved*

TYPES OF EDITOR SCRIPTS 1/2

- Custom Inspectors
  - The most commonly used
  - Makes editing complex component much easier
- Scene View extensions
  - Add extra functionality to the scene view
  - Often a functionality of a Custom Editor
  - E.g. Level editor
- Property Drawers
  - Instead of a whole component, custom editor for a type
  - Can be used to provide some sanity checks
  - Can only be used with UI Toolkit with some extra boiler-plate

TYPES OF EDITOR SCRIPTS 2/2

- Menu Items
  - The fastest one to create
  - Create an object or component with specific values
  - Open an editor window
- Editor Windows
  - A completely custom editor
  - E.g. dialogue graph
- Wizards
  - A subtype of an editor window
  - Create objects with user-selected values

WHAT IS UI TOOLKIT?

- Unity's new(ish) way to draw UI in the editor
- Can be created either in code or through XML
  - … but you don't really need to touch the XML if you just use the UI Builder
- Layout is done via CSS-like stylesheets, which can also be edited from code

HOW UI TOOLKIT DIFFERS FROM IMGUI/ONGUI

- Not immediate
- Slightly more verbose
- Always automatic layouting
- Override different methods depending on the context
- Uses events extensively

GETTING STARTED

- Requires 2021 LTS
- No packages necessary
- Create a script in a folder called Editor
- Extend the correct class:
  - Editor (custom inspectors)
  - EditorWindow (custom editor windows)
  - PropertyDrawer

OVERRIDING METHODS

- The actual UI Toolkit code goes into one of the following methods
  - void CreateGUI()
    - EditorWindow class
  - VisualElement CreateInspectorGUI()
    - Editor class
  - VisualElement CreatePropertyGUI(SerializedProperty property)
    - PropertyDrawer class
- Two last ones return VisualElement that contains the UI

CREATING THE LAYOUT

- Everything is a VisualElement
- VisualElements can be nested
  - Each element determines the layout rules for its children
  - Layout of children is determined by FlexDirection
  - Children inherit the properties of their parents
- VisualElement has .style property which can be editor runtime
  - E.g. modify the size of the element

COMMON COMPONENTS

- Button
  - Constructor or RegisterCallback to add the functionality
- IntField / FloatField
  - Int or float value
- Label
  - Any uneditable text

```
var generateButton = new Button();

generateButton.text = "Generate Map";

generateButton.RegisterCallback<ClickEvent>(ev :ClickEvent  =>
{
    GenerateMap(width.value, height.value, minSize.value);
});
```

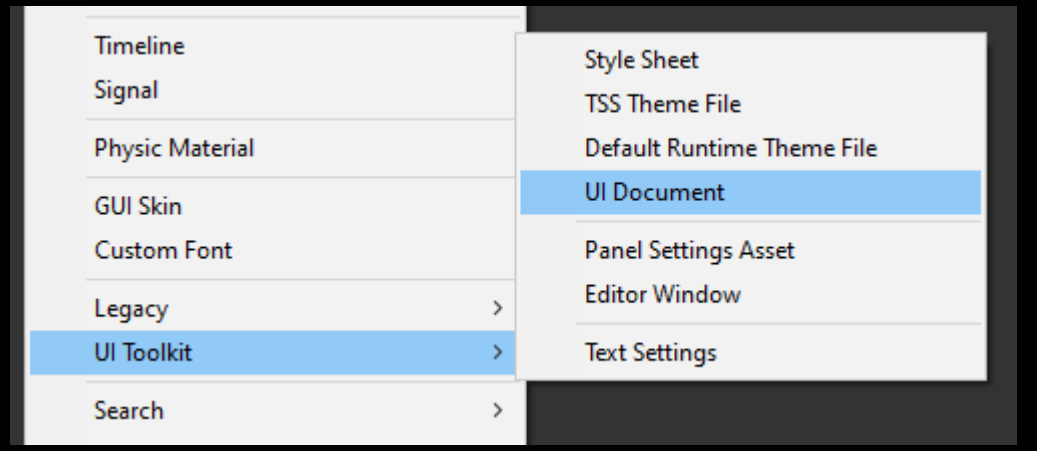
```
_objectField = root.Q<ObjectField>();

_objectField.RegisterValueChangedCallback(new EventCallback<ChangeEvent<Object>>((evt) =>
{
    UpdateImagePreview();
}));
```

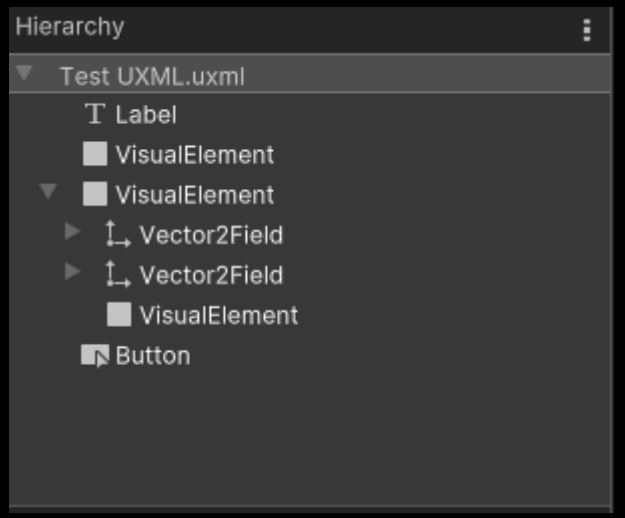
- ObjectField
  - Reference field for Unity Objects

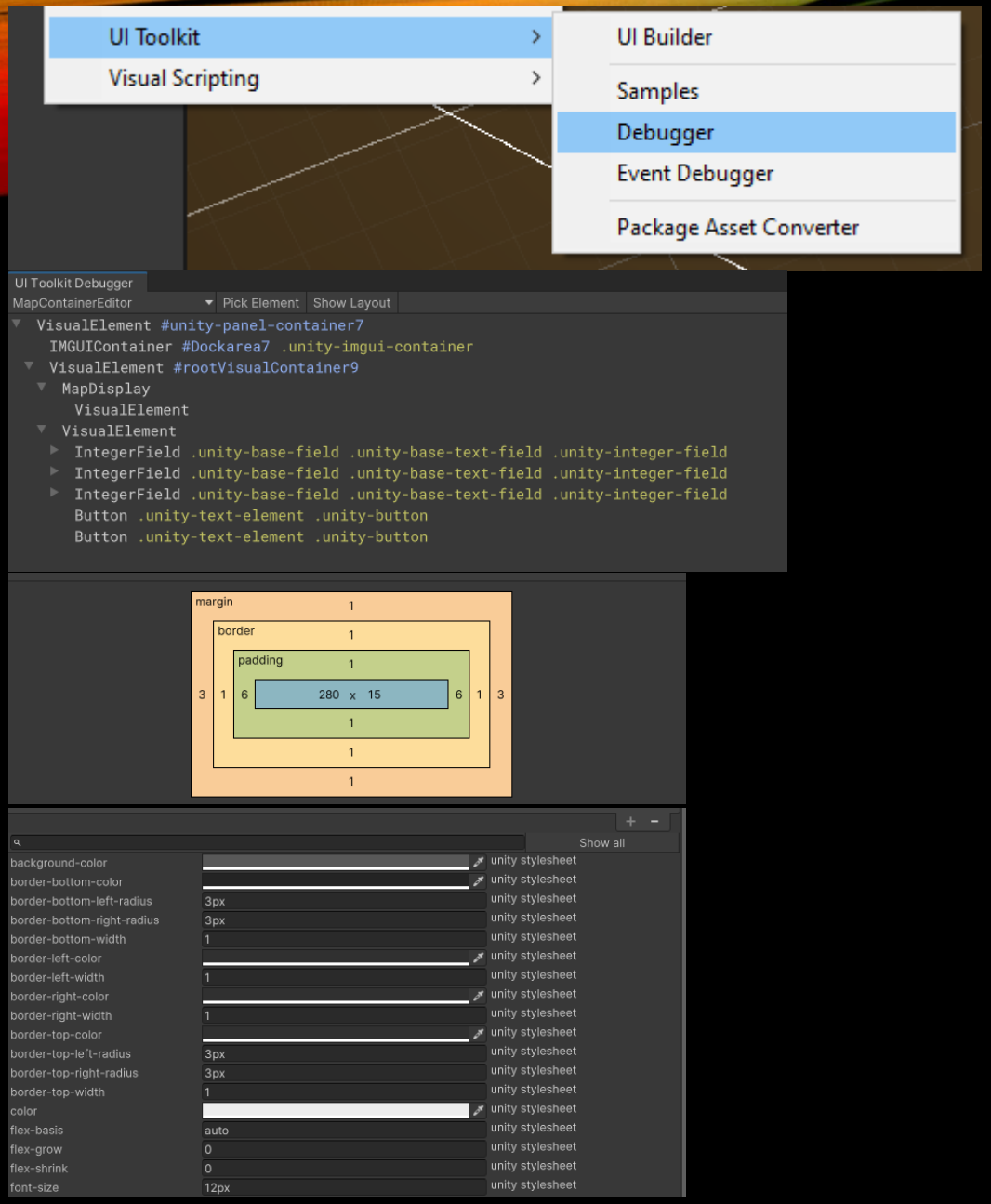
UI BUILDER

- Use for more complex static UIs, a couple of buttons or fields are probably faster to do with code
- Double-click on a UXML file to open:

- Bind data via paths







UI TOOLKIT DEBUGGER

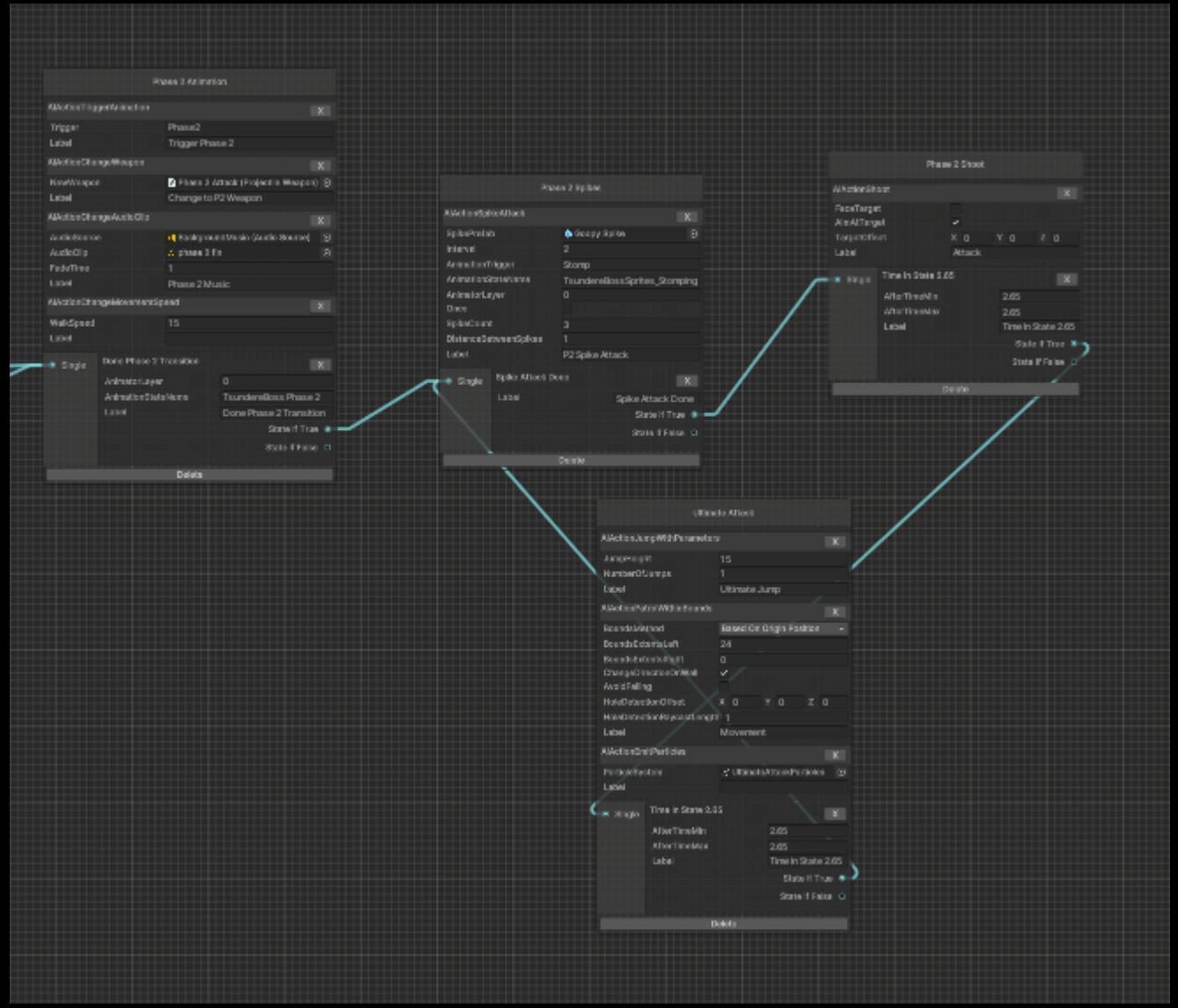
- This can be a life-saver!
- Show the hierarchy of any window (whether created in UI Builder or in code
- Similar to Developer Tools in Chrome
- Let's you edit values on the fly and see what works (edits are not saved, however)

REAL-LIFE EXAMPLES FOR INSPIRATION

- AI State Editor with Graphview
- 2D Tile-based map display by extending a VisualElement
- Quick Shortcut window

AI STATE MACHINE EDITOR

- Based on the EXPERIMENTAL GraphView element from Unity, available as a Preview Package
- Best suited for situations where you read once and write once, e.g. export the data into a dialogue tree



AI STATE MACHINE EDITOR

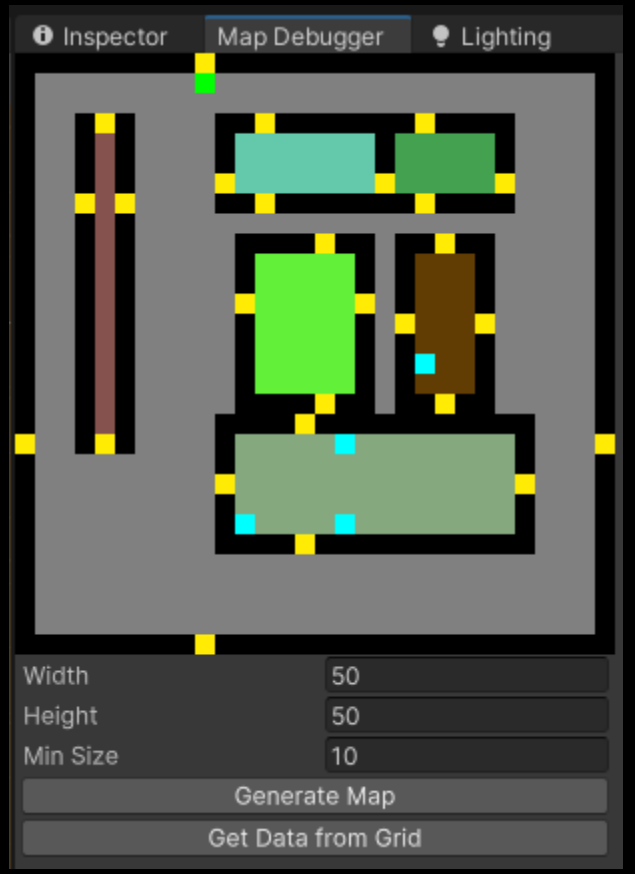
```
usages  overrides  Arto Koistinen  ext methods  exposing APIs
public StateMachineGraphView(StateMachineEditor editor)
{
    _editor = editor;

    styleSheets.Add( styleSheet: Resources.Load<StyleSheet>( path: "StateMachineGraph"));

    SetupZoom(ContentZoomer.DefaultMinScale, ContentZoomer.DefaultMaxScale);

    this.AddManipulator(new ContentDragger());
    this.AddManipulator(new SelectionDragger());
    this.AddManipulator(new RectangleSelector());

    graphViewChanged += OnGraphViewChanged;
    deleteSelection += (operationName :string , user) =>
        editor.ShowNotification(new GUIContent(
            text: "Deleting nodes is not supported this way, because GraphView is stupid. Use the button in the node."));


    var grid = new GridBackground();
    Insert( index: 0, grid);
    grid.StretchToParentSize();

}
```

```
usages  overrides  Arto Koistinen  ext methods  exposing APIs
private void GenerateInputPort()
{
    var port = InstantiatePort(Orientation.Horizontal, Direction.Input, Port.Capacity.Multi, typeof(float));

    inputContainer.Add(port);
}
```

```
usages  overrides  Arto Koistinen  ext methods  exposing APIs
public AIStateNode(StateMachineGraphView graphView, AIState state, AIBrain brain, Vector2 position)
{
    _state = state;
    _brain = brain;

    //title = state.StateName;

    titleBGColor = titleContainer.style.backgroundColor.value;

    titleContainer.style.justifyContent = (StyleEnum<Justify>)Justify.Center;

    titleContainer.Clear();

    SetNameEditable(string.IsNullOrWhiteSpace(state.StateName));

    int fields = 0;

    AddContextMenu( container: this, label: "Set As First State", condition: () => !IsFirstState(brain), action: (e :DropdownMenuAction ) => SetAsFirstState(brain));

    mainContainer.Add( child: new Button( clickEvent: () =>
    {...}) {text = "Delete"});


    GenerateInputPort();

    _outputPorts = new Dictionary<AITransition, OutputPortPair>();

    foreach (var transition in state.Transitions){...}

    SetPosition( newPos: new Rect( position.x, position.y, width: 100, height: 250 + fields * 30 ));

    GenerateActions();

    CheckAndAddEmptyLabels();

    AddContextMenu<AIAction>(_actionsPanel, brain.gameObject, contains: (t :AIAction ) => _state.Actions.Contains(t));
    AddContextMenu<AIDecision>(outputContainer, brain.gameObject, contains: (t :AIDecision ) =>
    {...});

    RefreshExpandedState();
    RefreshPorts();
}
```

MAP DEBUGGER

- Usage: Saves time debugging proc gen maps by not requiring Play Mode
- Takes map data and creates a simple visual representation by using coloured quads
- MapView Extends VisualElement and uses the generateVisualContent callback to generate the map mesh



MAP DEBUGGER

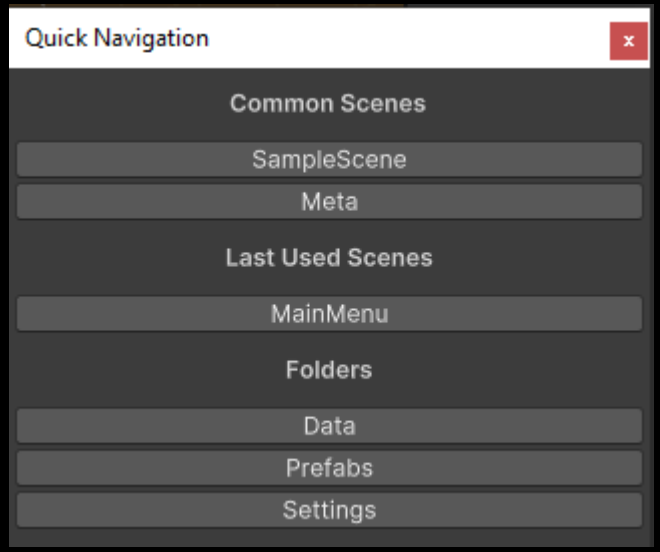
```
1 usage  Arto Koistinen
private void DrawCanvas(MeshGenerationContext context)
{
    Debug.Log( message: $"Rect {value.Width} | {value.Height}");

    if (value.Width == 0 || value.Height == 0)
        return;

    List<Tile> tiles = new List<Tile>();

    ushort[] tileIndices = { 0, 1, 2, 2, 3, 0 };

    for (int y = 0; y < _value.Height; y++)
    {
        for (int x = 0; x < _value.Width; x++){...}
    }

    List<Vertex> vertices = new List<Vertex>();
    List<ushort> indices = new List<ushort>();

    for(int i = 0 ; i < tiles.Count; i++)
    {
        vertices.AddRange(tiles[i].Vertices);

        for (int j = 0; j < tileIndices.Length; j++)
        {
            ushort s = (ushort) Convert.ToInt16(i * 4 + tileIndices[j]);

            indices.Add(s);
        }
    }

    var mwd :MeshWriteData = context.Allocate( vertexCount: vertices.Count, indexCount: indices.Count);

    mwd.SetAllVertices(vertices.ToArray());
    mwd.SetAllIndices(indices.ToArray());
}
```

```
4 usages  Arto Koistinen  ··· More
private class Tile
{
    private Vertex[] _vertices = new Vertex[4];

    1 usage  Arto Koistinen
    public Vertex[] Vertices => _vertices;

    2 usages  Arto Koistinen
    public Tile(Vector2Int position, float size, Color color)
    {
        for (int i = 0; i < _vertices.Length; i++)
        {
            _vertices[i].tint = (Color32)color;
        }

        _vertices[0].position = new Vector3( x: position.x * size, y: (position.y + 1) * size, Vertex.nearZ);
        _vertices[1].position = new Vector3( x: position.x * size, y: position.y * size, Vertex.nearZ);
        _vertices[2].position = new Vector3( x: (position.x + 1) * size, y: position.y * size, Vertex.nearZ);
        _vertices[3].position = new Vector3( x: (position.x + 1) * size, y: (position.y + 1) * size, Vertex.nearZ);
    }
```

```
_mapDisplay.generateVisualContent += DrawCanvas;
```

QUICK NAVIGATION

- Usage: Press ctrl+g to open a window with the most commonly used scenes and folders
- Note: Currently fails to set Project View as active



QUICK NAVIGATION

```
Event function
private void CreateGUI()
{
    AddLabel( text: "Common Scenes");

    foreach (var scene :string in EditorGlobal.CommonScenes)
    {
        AddSceneButton(scene);
    }

    AddLabel( text: "Last Used Scenes");

    var lastUsedScenes :List<string> = EditorGlobal.LastUsedScenes;

    int count = lastUsedScenes.Count > 5 ? 5 : lastUsedScenes.Count;

    for (int i = 0; i < count; i++)
    {
        AddSceneButton(lastUsedScenes[i]);
    }

    AddLabel( text: "Folders");

    foreach ( var key :string in EditorGlobal.Links )
    {
        AddLink( key );
    }
}
```

```
2 usages
private void AddSceneButton(string scene)
{
    var shortName :string = Path.GetFileNameWithoutExtension(scene);

    var button = new Button( clickEvent: () =>
    {
        EditorSceneManager.OpenScene(scene);

        Close();
    });

    button.text = shortName;

    rootVisualElement.Add(button);
}

3 usages
private void AddLabel(string text)
{
    var label = new Label(text);

    label.style.unityFontStyleAndWeight = (StyleEnum<FontStyle>)FontStyle.Bold;
    label.style.marginBottom = (StyleLength)10;
    label.style.marginTop = (StyleLength)10;
    label.style.alignSelf = (StyleEnum<Align>)Align.Center;

    rootVisualElement.Add(label);
}
```

WRAP-UP

- Editor extensions are a great tool for increasing productivity and decreasing frustration
- UI Toolkit is going to replace IMGUI in the long run, so using will result in more future-proof tools
- It's still not 100% there even for editor scripting (e.g. problems with the property drawers)

THANK YOU!

- arto.koistinen@randompotion.com
- Slides are up at www.randompotion.com
- @arzi@mastodon.gamedev.place

Random Potion

GAMES TO TELL STORIES ABOUT