A tale of three UI frameworks

A decade of evolving developer and designer workflows in a game engine
Who are we?

Adam Mechtley
Lead Developer
DOTS Physics & Rigging

Damian Campeanu
Developer
Editor & UI
What are we going to talk about?

- Brief overview of Unity Architecture
- IMGUI
- uGUI
- UI Toolkit
- Q&A
What is Unity and how do people make stuff with it?

* Oversimplified version
More than just a game engine!

- Run-Time
  - Built-in libraries (input, animation, physics, UI, rendering, etc.)
  - Compatible with much of .NET ecosystem
- Editor
- Services
  - Analytics, live ops, etc.
- Asset Store
- Millions of users, hundreds of thousands active monthly
How do users make stuff with Unity?

- Create **GameObjects** and add **Components** to produce behavior
  - Create new Components via C#
- Create reusable **Prefabs** from GameObjects
  - Prefabs can be nested in each other with overrides
using UnityEngine;

class Rotator : MonoBehaviour
{

    public Vector3 Axis = Vector3.forward;
    public float AngularSpeed = 45f;

    void Update()
    {
        transform.Rotate(Axis, AngularSpeed * Time.deltaTime);
    }
}

How do users create content for it?

- Save source file in project folder
- Built-in or custom asset importer generates artifact(s)
How do users create UI with it?

- **Unity 1.0**
  - GUITexture, GUIText, GUILayer, TextMesh

- **Unity 2.0**
  - IMGUI run-time

- **Unity 2.5**
  - Customize editor with IMGUI

- **Unity 2019.1**
  - First public release of UI Toolkit (formerly UIElements)

- **2005**
  - Unity 1.0

- **2007**
  - Unity 2.0

- **2009**
  - Unity 2.5

- **2014**
  - Unity 4.6

- **2019**
  - Unity 2019.1

First release of uGUI
IMGUI
A simple UI framework for an ever-changing world

- **Unity 1.0**: GUITexture, GUIText, GUILayer, TextMesh
- **2005**: Unity 1.0
- **2007**: Unity 2.0, IMGUI run-time
- **2009**: Unity 2.5, Customize editor with IMGUI
- **2014**: Unity 4.6, First release of uGUI
- **2019**: Unity 2019.1, First public release of UI Toolkit (formerly UIElements)
Design considerations

- Unity needs a UI framework! (both run-time and editor)
- Most Unity projects...
  - ...are small web player experiences
  - ...are created by small teams with few/broad role specializations
- Most game UI...
  - ...communicates frequently updating values
  - ...is non-diegetic overlays
IMGUI API

- **OnGUI()** callback
  - Event loop with `Event.current`
  - Call order determines event handling priority
- Library of static methods in **GUI** class for common functionality
  - **GUILayout** variants to assist with `Rect` calculations
  - Both run-time and editor-only variants for most types
- **GUIStyle** class
- **GUISkin** asset
API and Workflow Demo
IMGUI advantages and disadvantages

+ Gathering and responding to input is trivial
+ Fast for programmers to prototype with
+ Works well for property grids
+ Simple API organization
+ Predictable performance

— But not very great performance
— Limited designer workflows
— No control over rendering pipeline
— Only supports non-diegetic UI
— Lots of manual work making new controls
uGUI
A framework for to make UI feel more like the rest of Unity
Design considerations

- Unity needs needs to empower designers to be productive more independently

- Most Unity projects...
  - ...are created by teams with clearer role specializations
  - ...run on mobile platforms where draw calls are expensive and display specifications vary wildly

- Most game UI...
  - ...contains diegetic/spatial and non-diegetic elements
  - ...is richly animated with effects
### Fagerholt & Lorentzon (2009)

<table>
<thead>
<tr>
<th>Exists in game fiction?</th>
<th>Visualized in 3D world?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Non-diegetic</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Spatial</td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td>Meta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Diegetic</td>
</tr>
</tbody>
</table>
uGUI API

- **UIBehaviour** base class inherits **MonoBehaviour**
  - Selectable, Graphic, etc. sub-classes
- **Canvas** and **CanvasScalar** control rendering of hierarchies of elements
  - **RectTransform** (inherits **Transform**) use for layout
  - Most components draw **Sprite** assets
  - Set geometry, materials, etc. on child **CanvasRenderer**
- **StandaloneInputModule** and **EventSystem** gather and delegate input events
  - **BaseRaycaster** of some kind finds event handlers
  - **IPointerDownHandler, IPointerUpHandler, IDragHandler**, etc.
API and Workflow Demo
uGUI advantages and disadvantages

+ Designer workflows that fit with other Unity features (prefabs, animation, etc.)
+ Serializable event handlers
+ Automatic atlassing and scaling based on physical size, DPI, etc.
+ Diegetic UI
+ Common rendering pathway with everything else

— Performance overhead from GameObjects and Components
— Authoring data format hard to read and debug at a glance
— No centralized styling
— Canvases require specialized knowledge to optimize
UI Toolkit

A framework to make Unity feel more like the rest of the world

Unity 1.0
GUITexture, GUIText, GUILayer, TextMesh

Unity 2.0
IMGUI run-time

Unity 2.5
Customize editor with IMGUI

Unity 4.6
First release of uGUI

Unity 2019.1
First public release of UI Toolkit (formerly UIElements)
Design considerations

👩‍💻 Collaboration
Different team members can work on different parts of the same UI.

いると® Iteration Speed
Quickly develop and validate UI for different contexts.

🪤 Familiarity
UI authoring tools and workflows are familiar and easy to learn.

🔄 Reusability
Share styles and templates within or across projects.

🌟 Extensibility
Customize and extend existing styles and templates or build custom ones.

🔍 Rich Content
Build engaging UI that performs well as it scales.
UI Toolkit API

```csharp
void OnEnable() {
    var a = AssetDatabase.LoadAssetAtPath("Assets/hierarchy.uxml");
    VisualElement row = a.CloneTree();
    var label = row.Q<Label>("random-explosion");
    label.RegisterCallback<MouseUpEvent>(
        evt => evt.StopPropagation());
    rootVisualElement.Add(row);
}
```

```html
<!-- hierarchy.uxml -->
<UXML xmlns="UnityEngine.UIElements">
    <VisualElement class="container">
        <Style src="styles.uss" />
        <Label
            name="random-explosion"
            text="UIElements!" />
    </VisualElement>
</UXML>
```
UI Toolkit API

```csharp
// MyWindow.cs
void OnEnable() {
    var a = AssetDatabase.LoadAssetAtPath(
        "Assets/hierarchy.uxml"
    );
    VisualElement row = a.CloneTree();
    var label = 
        row.Q<Label>("random-explosion");
    label.RegisterCallback<MouseUpEvent>(
        evt => evt.StopPropagation());
    rootVisualElement.Add(row);
}
```

```css
/* styles.uss */
.container { font-size: 40px; }
#random-explosion { color: blue; }

<!-- hierarchy.uxml -->
<UXML xmlns="UnityEngine.UIElements">
    <VisualElement class="container">
        <Style src="styles.uss" />
        <Label
            name="random-explosion"
            text="UIElements!" />
    </VisualElement>
</UXML>
```
UI Toolkit API

```csharp
void OnEnable()
{
    var a = AssetDatabase.LoadAssetAtPath("Assets/hierarchy.uxml");
    VisualElement row = a.CloneTree();
    var label = row.Q<Label>("random-explosion");
    label.RegisterCallback<MouseUpEvent>(
        evt => evt.StopPropagation();
    );
    rootVisualElement.Add(row);
}
```

// MyWindow.cs

```html
<!-- hierarchy.uxml -->
<UXML xmlns="UnityEngine.UIElements">
    <VisualElement class="container">
        <Style src="styles.uss" />
        <Label
            name="random-explosion"
            text="UIElements!" />
    </VisualElement>
</UXML>
```
UI Toolkit API

/* styles.uss */
.container { font-size: 40px; }
#random-explosion { color: blue; }

<!-- hierarchy.uxml -->
<UXML xmlns="UnityEngine.UIElements">
  <VisualElement class="container">
    <Style src="styles.uss" />
    <Label
      name="random-explosion"
      text="UIElements!" />
  </VisualElement>
</UXML>

// MyWindow.cs
void OnEnable() {
  var a = AssetDatabase.LoadAssetAtPath("Assets/hierarchy.uxml");
  VisualElement row = a.CloneTree();
  var label = row.Q<Label>("random-explosion");
  label.RegisterCallback<MouseUpEvent>(
    evt => evt.StopPropagation());
  rootVisualElement.Add(row);
}

// MyWindow.cs

UIElements!
API and Workflow Demo
UI Toolkit advantages and disadvantages

+ Great performance for most use cases
+ Powerful automatic layouting via Flexbox
+ Centralized styling using standard paradigms (CSS)
+ Visual authoring without writing code
+ One API for both Editor and Runtime

— Name-based handles can easily break
— Inefficient when lots of things are changing at once
— More complicated Event-based value bindings
— More complicated bindings to Unity objects and gameplay
Cool story. What should I tell my friends I learned today?
Final thoughts

- Immediate-mode and retained-mode GUI each have strengths and disadvantages in different situations
- As the rest of the world evolves, so, too, must your API
  - Everything comes with a maintenance cost
- Reasonableness of API design decisions is very contextual
  - Aesthetic tastes of the historical moment
  - Technical requirements of target hardware
  - Tools ecosystem
- Design influences users’ expressive capabilities
Thank you.

#unity3d