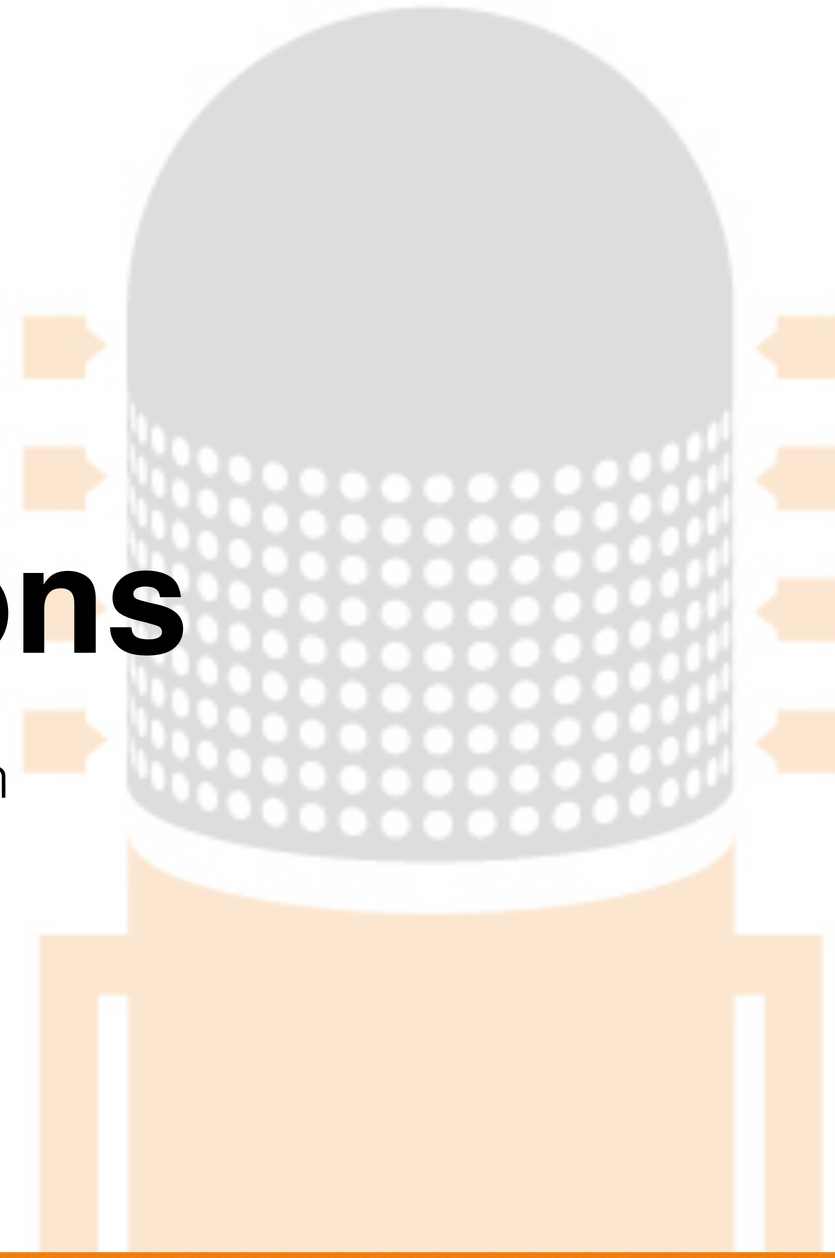


Editor Extensions

Allowing devs to let their hair down

Who is this guy?

Ronan Tumelty

Education:
- B.A. (Mod) Computer Science - Trinity College Dublin
- HND Digital Game Design - Ballyfermot College of Further Education

Corporate experience:
- Software Tester - State Street International Ireland
- Developer, PS3 Development Kit - Sony Computer Entertainment International Ireland

Start-ups:
- Partner and Co-Founder of Raptor Games / The Mighty Kraken
- Freelance Designer and Developer
- Front-end Developer at SoapBox Labs

Who are SoapBox Labs ?

- Speech technology start-up
- Speech verification for early literacy learning

- Semi-finalists in the Alpha Pitch competition at Web Summit
- Finalists in the ESB Spark of Genius competition (also Web Summit)

- Using Unity for audio gathering and educational apps

What's that title about?

- What's an editor?
- What do you mean, extending?
- Why and how would I do that?
- Was the pun necessary?

Why Extend the Editor?

- Improve work flow
- Limit potential errors

- Allow non-coders to contribute
- Easier for new team members
- Save your developer's time (and sanity)

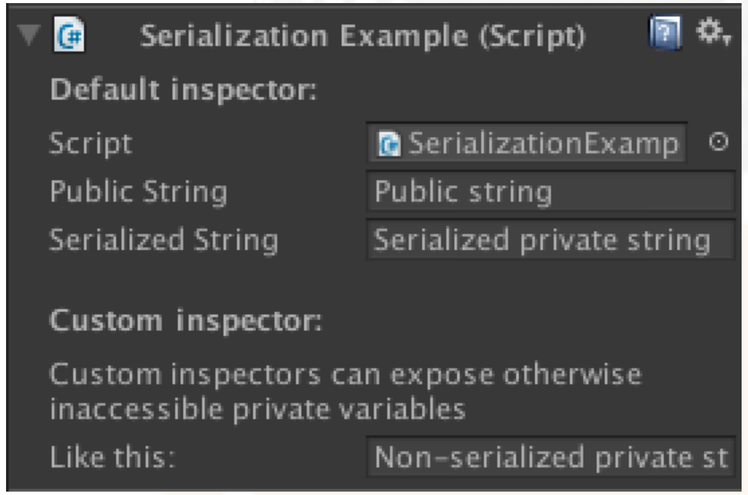
- Essential for sale on Asset Store / other 3rd party distribution

Easy Mode: Serialisation

```
8  public class SerializationExample : CustomMonoBehaviour {
9
0      private string nonSerialized = "Non-serialized private string";
1      public string publicString = "Public string";
2      [SerializeField]
3      private string serializedString = "Serialized private string";
```

- public variables automatically serialised
- [SerializedField] exposes hidden properties (e.g. private/protected)
- [System.Serializable] exposes whole structs/classes

- Exposes properties to Editor
- Simple, quick results, but a lot of depth.



GUI(Layout) and EditorGUI(Layout)

- Provide functions for displaying and editing common data types
  - Primitives (int, float, bool, string, etc)
  - Complex objects (GameObjects, Transforms, AnimationCurves…)

- And for positioning and presentation
  - Labels, Buttons, Foldouts, Popups…

- GUI and EditorGUI - explicitly laid out
- GUILayout and EditorGUILayout - laid out automatically

Window Dressing

- Can build custom windows
  - **Essential** for third-party tools

- Acts like native Unity windows
  - Can dock, resize, maximise via familiar controls

- Can specify menu item to draw window

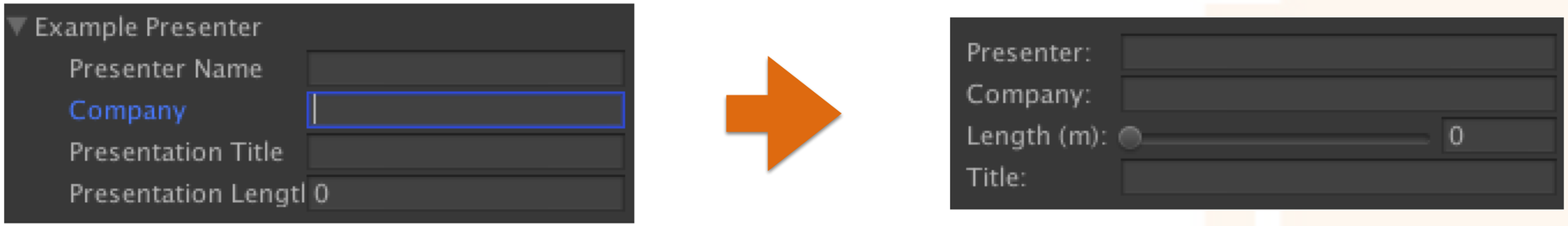

```
[MenuItem("DUUG/Event Manager")]
public static void ShowWindow() {
    eventList = AssetDatabase.LoadAssetAtPath(path, typeof(DUUGEventList)) as DUUGEventList;
    EditorWindow.GetWindow(typeof(DUUGEventManager));
}
```

Public Property

```
6   [CustomPropertyDrawer(typeof(DUUGPresenter), true)]
7   public class DUUGPresenterDrawer : PropertyDrawer {
8       public override void OnGUI(Rect position, SerializedProperty property, GUIContent label) {
```

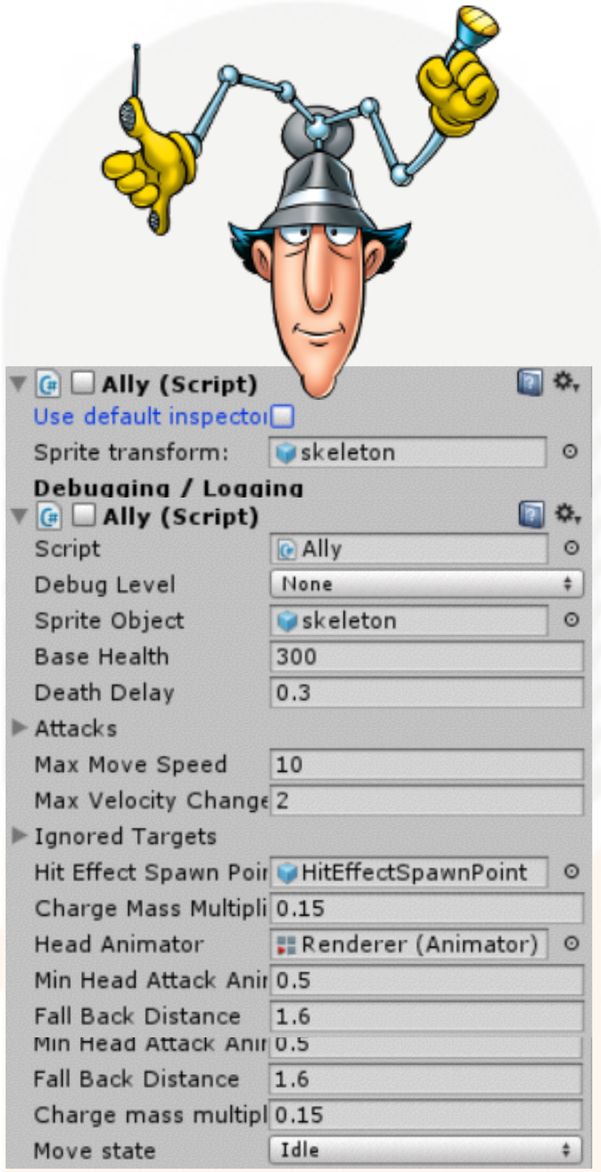
- Property Drawers - customise how objects are displayed
- Requires objects and relevant fields to be serialised

- New interface drawn by default inspectors automatically
- Drawn with PropertyField() method in EditorGUI and EditorGUILayout

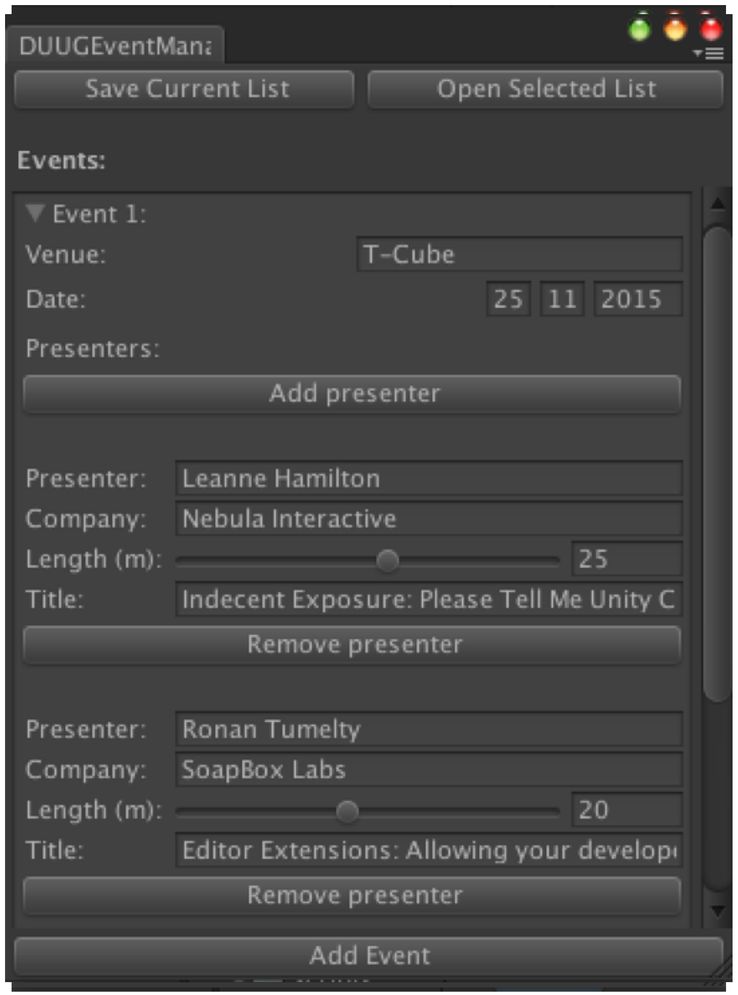


Inspector Gadgets!

```
5   // Instructs engine to use this class over default inspector
6   [CustomEditor(typeof(CustomMonoBehaviour), true)]
7   // Allows the editing of multiple instances of an object at once
8   [CanEditMultipleObjects]
9   public class CustomMonoBehaviourInspector : Editor {
10
11      public override void OnInspectorGUI() {
12          CustomMonoBehaviour customBehaviour = target as CustomMonoBehaviour;
13          customBehaviour.ShowInspectorGUI(this);
14      }
15  }
```



- Custom Inspectors replace default inspector
- Display script properties in more meaningful fashion
- DrawDefaultInspector… draws default inspector



Doing It With Style

- GUISkin - used to define custom style options
- EditorStyles - provides access to UI style used by the Editor

- GUILayoutOptions - controls use of space
  - Found in GUILayout class

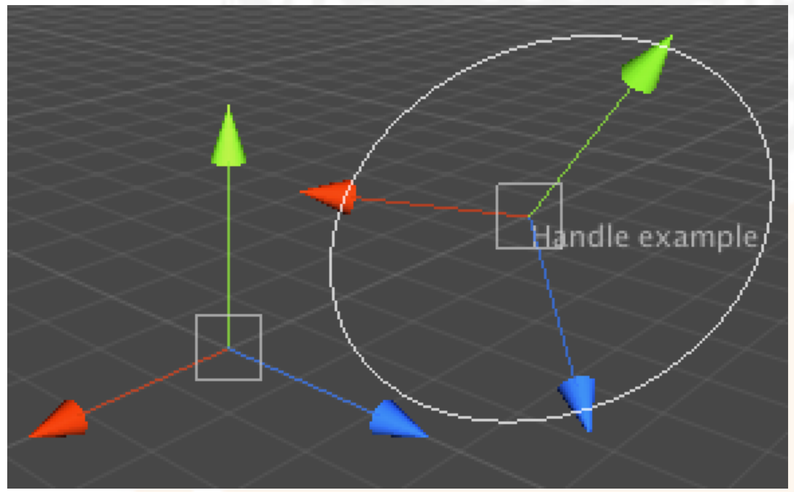
```
GUILayout.Label("Select EventList asset in Project view, or create a new EventList", EditorStyles.wordWrappedLabel);
path = EditorGUILayout.TextField("File path:", path, GUILayout.ExpandWidth(true));
```

Getting a Handle on Things

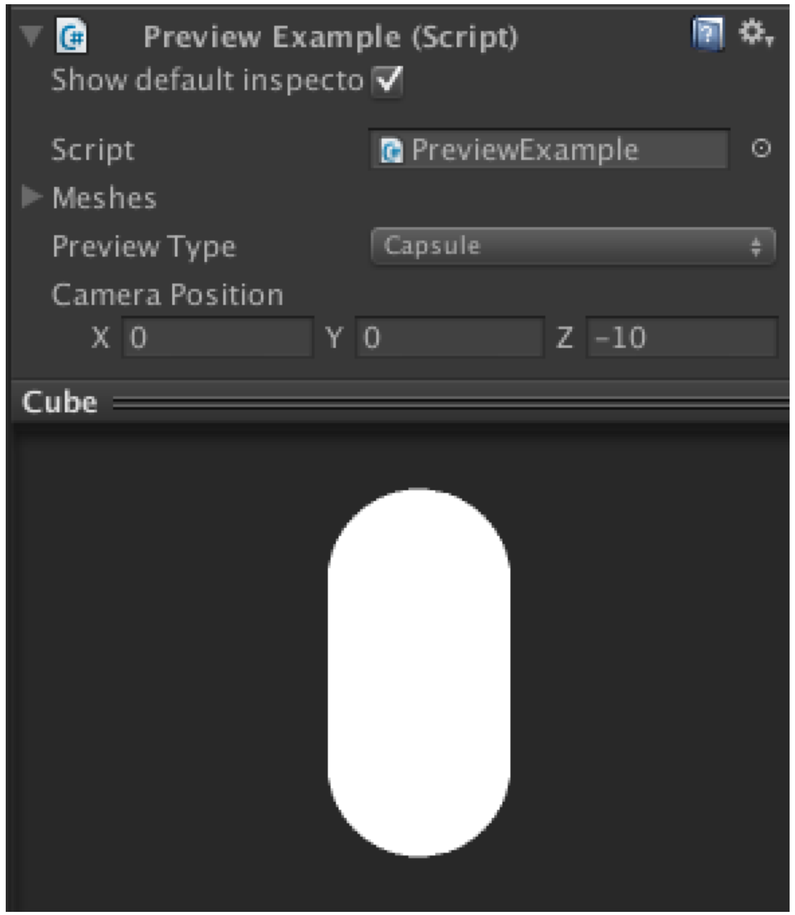
```
12  #if UNITY_EDITOR
13      public override void ShowSceneGUI() {
14          handlePosition = Handles.PositionHandle(handlePosition, handleRotation);
15          handleRotation = Handles.Disc(handleRotation, handlePosition, new Vector3(1, 1, 0), 1, false, 0);
16      }
17  #endif
```

- Handle class draws control gizmos in Scene View
  - Call in OnSceneGUI()

- Modify things visually and responsively



Using Inspector Previews




```
40      public override bool HasPreviewGUI()...
48
49      public override void ShowPreviewGUI(Rect r, GUIStyle background)...
```

- Inspector preview panel can be customised
- Can draw 2D sprites, 3D objects, graphs
- Full control of preview camera

Stumbling Blocks

- Overkill for smaller projects
- Inflating code base
  - Need to manage codebase carefully
- UX (user experience) is important
  - Especially if selling tools

Useful Hacks and Tips!

- Reduce number of inspector scripts via inheritance
  - Create parent class with custom inspector
  - Derive subclasses from parent
  - Use preprocessor commands to include editor code in runtime scripts

- Use Unity's documentation and tutorials

- Slides and example project on DUUG's Github
- Video to be uploaded to DUUG's YouTube channel

Cheatsheet

- EditorGUI
- GUI
- Editor
- PropertyDrawer
- EditorGUILayout
- GUILayout
- EditorWindow
- Handles
- PreviewRenderUtility

Any Questions?