Unity Tools Programming

Start Improving Your Workflow

About Me

@defuncart

in/jamesjleahy/

defuncart.com/games/

James Leahy

2D Mobile

Game Developer

What is This Talk?

- A general introduction on coding Editor Tools for Unity.
- A short discussion on how such tools have helped improved our workflow.

Who is This Talk For?

- Anyone using Unity!
- Developers: code samples are of an intermediate level.
- Non-Developers: learn how developers can help you!

GitHub

All code samples shown throughout this talk are available to download and use freely under an MIT licence.

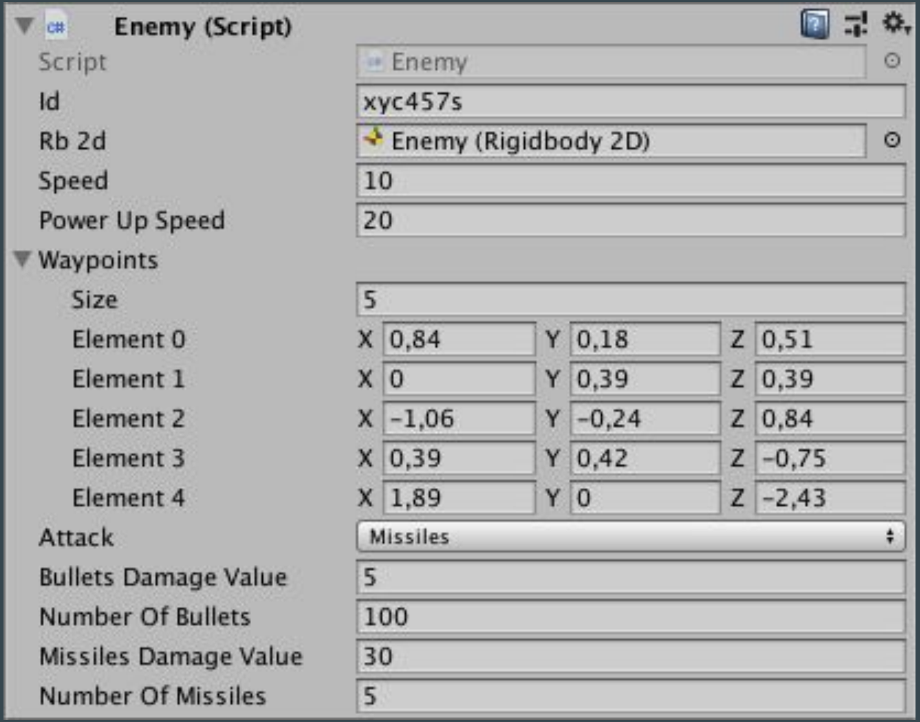
github.com/defuncart/
Talk-Unity-Tools-Programming

Unity Tools Programming

- A tool is something which aids the team in creating the game, it is not a part of the final product.
  - Artist → Pixel Art Editor.
  - Game Designer → Level Editor.
  - Sound Designer → Optimize music compression level.
  - Automated build system.
  - Localization database import.
  - Reskin the UI for a Holiday event.

1. Inspector

1. Inspector



Enemy (Script)

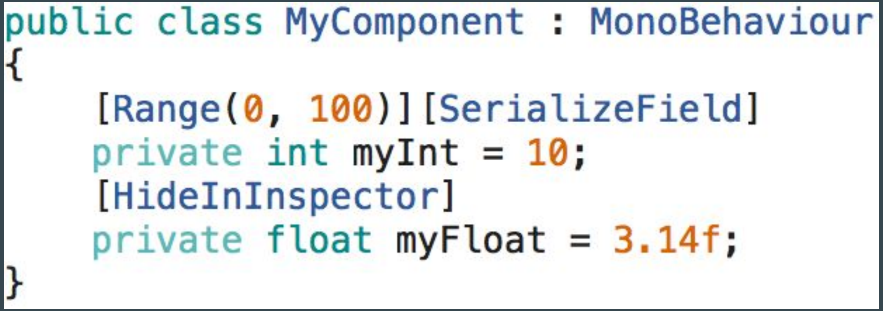
Script	Enemy
Id	xyc457s
Rb 2d	Enemy (Rigidbody 2D)
Speed	10
Power Up Speed	20
Waypoints	
Size	5
Element 0	X 0,84 Y 0,18 Z 0,51
Element 1	X 0 Y 0,39 Z 0,39
Element 2	X -1,06 Y -0,24 Z 0,84
Element 3	X 0,39 Y 0,42 Z -0,75
Element 4	X 1,89 Y 0 Z -2,43
Attack	Missiles
Bullets Damage Value	5
Number Of Bullets	100
Missiles Damage Value	30
Number Of Missiles	5

- The Inspector window displays detailed information about a component or an asset.
- When there are numerous properties, the inspector can seem cluttered.
- Often the person interacting with the inspector isn't the script author.
- Well laid-out inspectors are easier to understand and use.

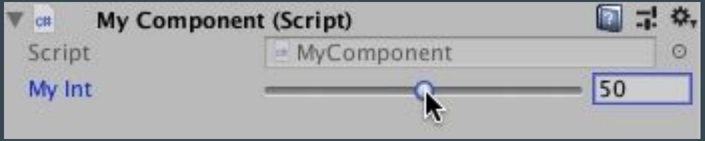
1.1 Pimp the Inspector

1.1 Pimp the Inspector : Attributes

- [Range] constrains a float or an int to a certain range.
- [SerializeField] exposes a private property in the Inspector.
- [HideInInspector] hides a serialized property in the Inspector.



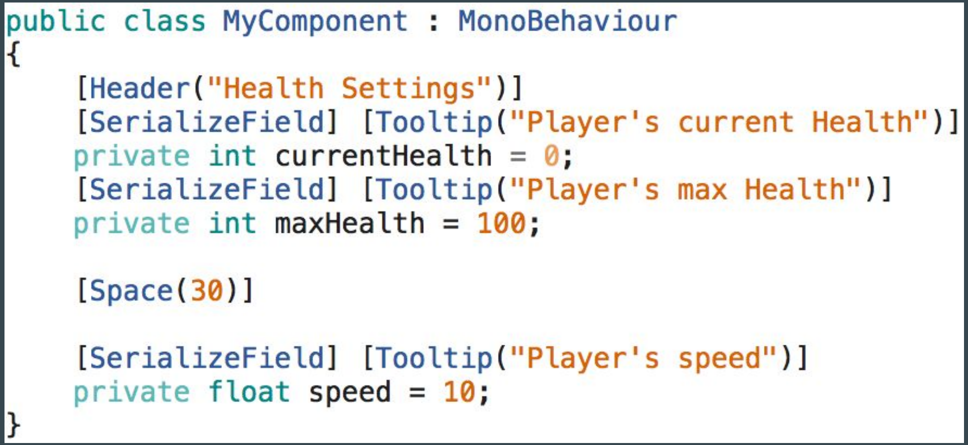
```
public class MyComponent : MonoBehaviour
{
    [Range(0, 100)][SerializeField]
    private int myInt = 10;
    [HideInInspector]
    private float myFloat = 3.14f;
}
```



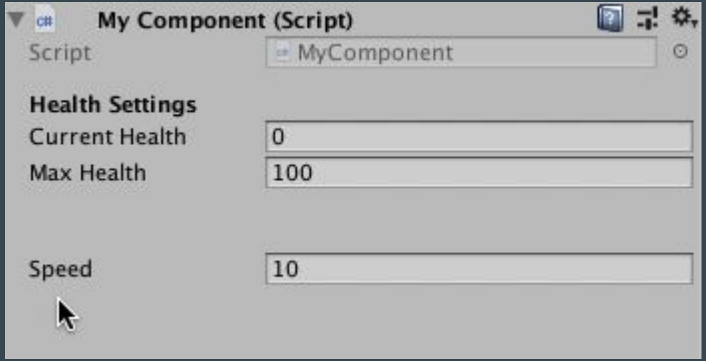
My Component (Script)
Script: MyComponent
My Int: 50

1.1 Pimp the Inspector : Attributes

- **[Header]** adds a header above fields in the Inspector.
- **[Space]** adds a space between fields in the Inspector.
- **[Tooltip]** adds a tooltip for a field. This is visible when the mouse hovers over the field in the inspector.

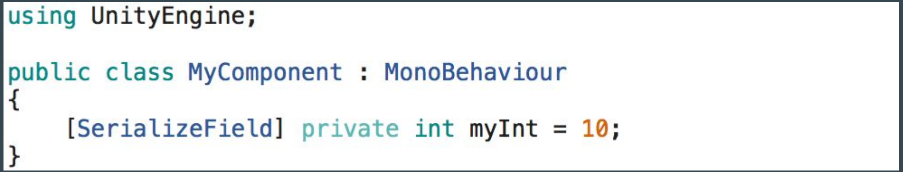


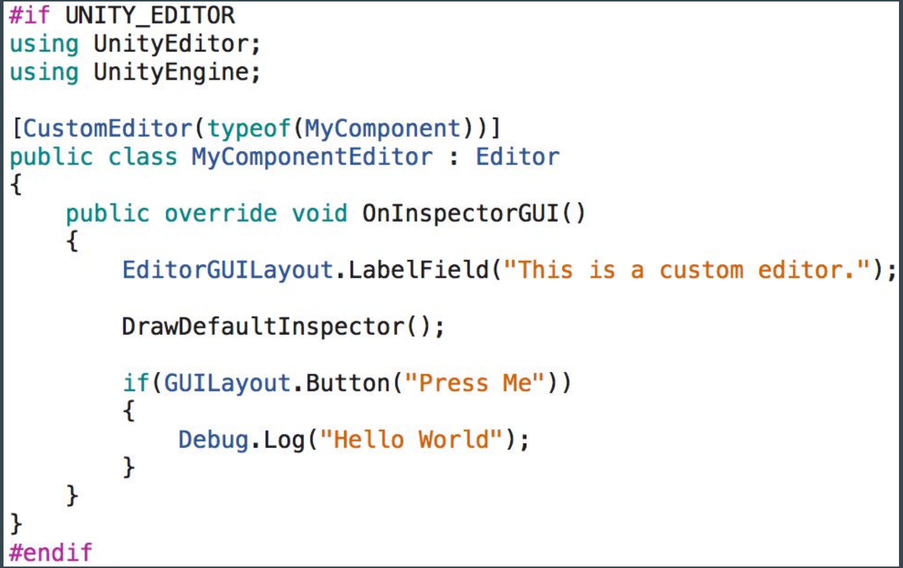
```
public class MyComponent : MonoBehaviour
{
    [Header("Health Settings")]
    [SerializeField] [Tooltip("Player's current Health")]
    private int currentHealth = 0;
    [SerializeField] [Tooltip("Player's max Health")]
    private int maxHealth = 100;

    [Space(30)]

    [SerializeField] [Tooltip("Player's speed")]
    private float speed = 10;
}
```



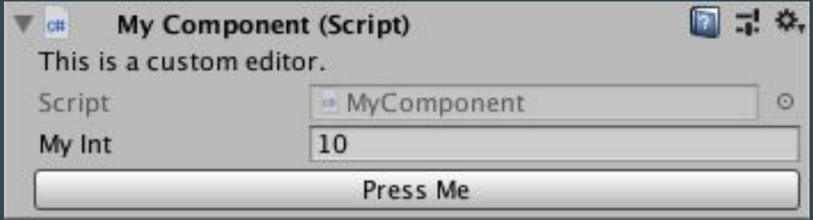
```
My Component (Script)
Script            MyComponent

Health Settings
Current Health    0
Max Health        100


Speed             10
```

1.2 Pimp the Inspector : Custom Inspectors

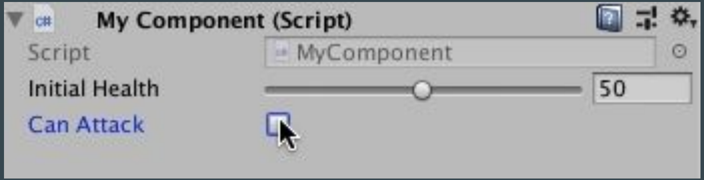

```
using UnityEngine;

public class MyComponent : MonoBehaviour
{
    [SerializeField] private int myInt = 10;
}
```


```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

[CustomEditor(typeof(MyComponent))]
public class MyComponentEditor : Editor
{
    public override void OnInspectorGUI()
    {
        EditorGUILayout.LabelField("This is a custom editor.");

        DrawDefaultInspector();

        if(GUILayout.Button("Press Me"))
        {
            Debug.Log("Hello World");
        }
    }
}
#endif
```

By writing a script extending from Editor and overriding the callback OnInspectorGUI, we have complete control over the inspector's layout.


My Component (Script)
This is a custom editor.
Script MyComponent
My Int 10
Press Me

1.2 Pimp the Inspector : Custom Inspectors

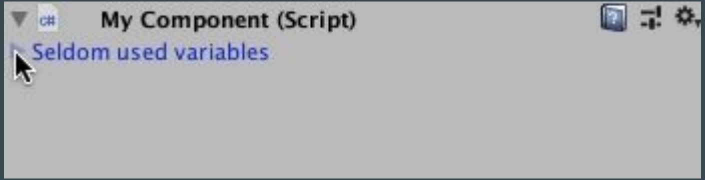

My Component (Script)

Script: MyComponent

Initial Health: 50

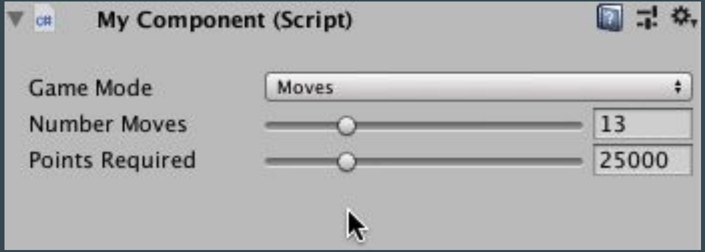
Can Attack

A boolean used to display additional settings.


My Component (Script)

Seldom used variables

A fold-out group of seldom used settings.


My Component (Script)

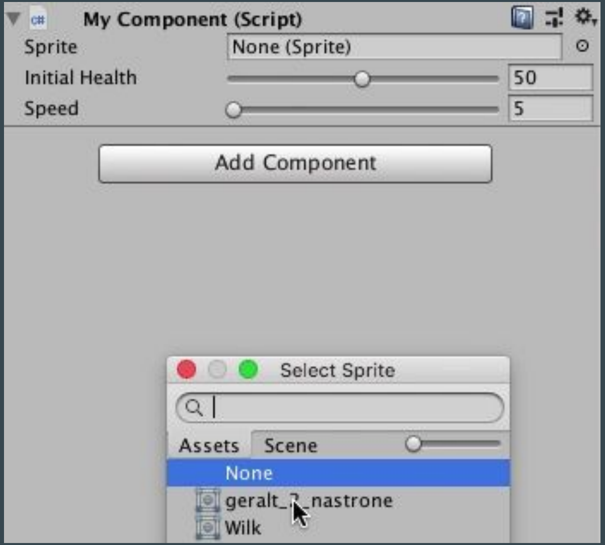
Game Mode: Moves

Number Moves: 13

Points Required: 25000

An enum used to display relevant settings for a game mode.

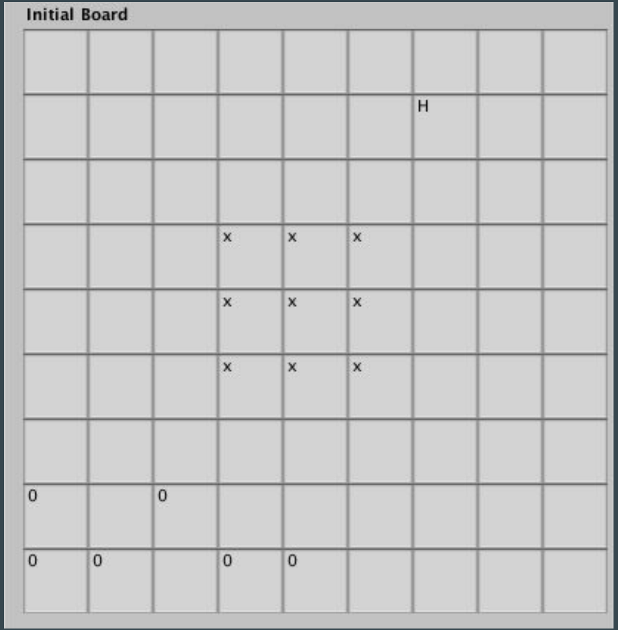
1.2 Pimp the Inspector : Custom Inspectors



My Component (Script)

Sprite	None (Sprite)
Initial Health	50
Speed	5

Drawing a sprite property inside the inspector.



Initial Board

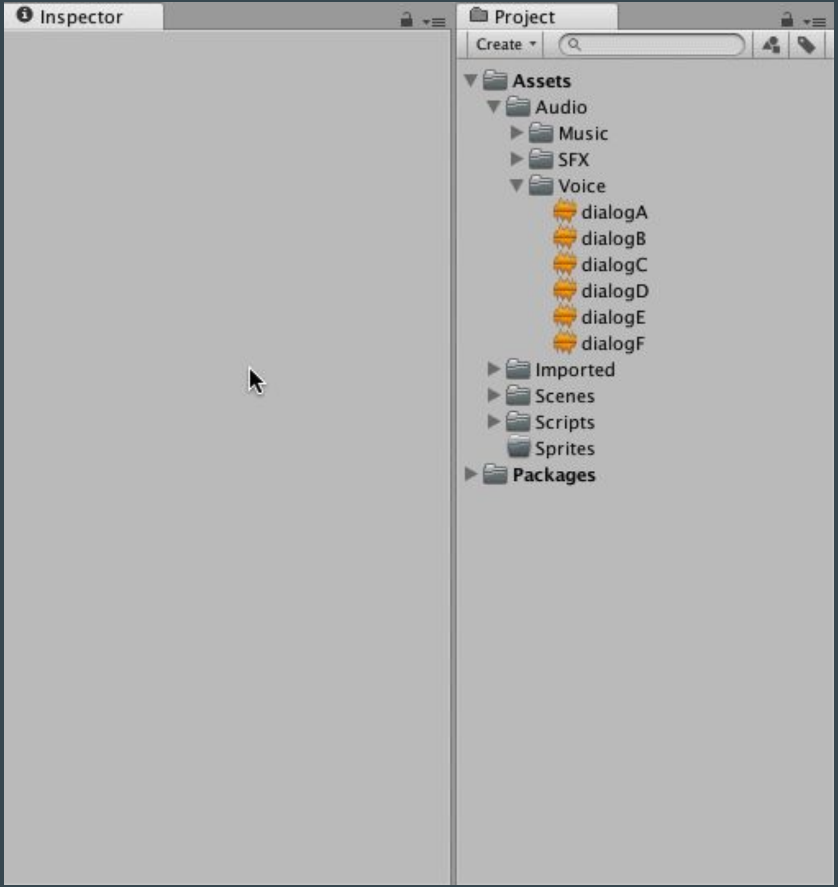
						H		
			x	x	x			
			x	x	x			
			x	x	x			
0		0						
0	0		0	0				

Visualizing a 2d array

2. Asset Import

2. Asset Import

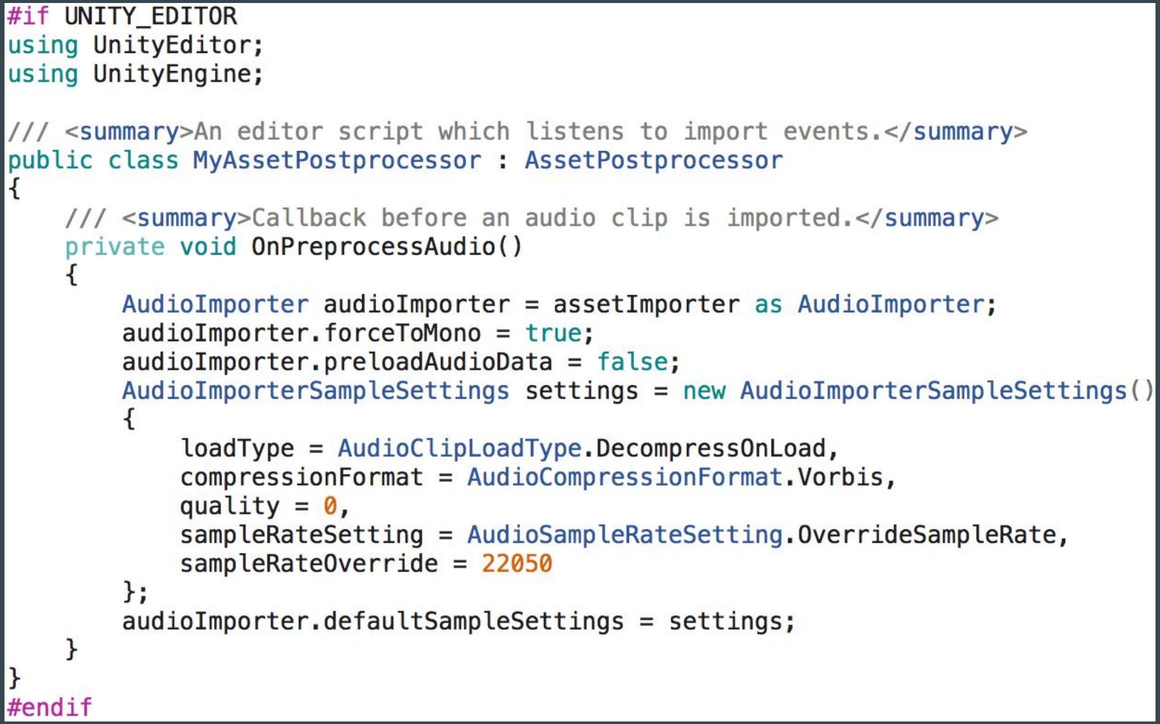
- By group selecting files, settings for multiple files of the same type can be simultaneously updated.
- However, this is
  - repetitive,
  - error-prone,
  - easy to forget.
- Wouldn't it be great to be able to automate import settings?



2. Asset Import

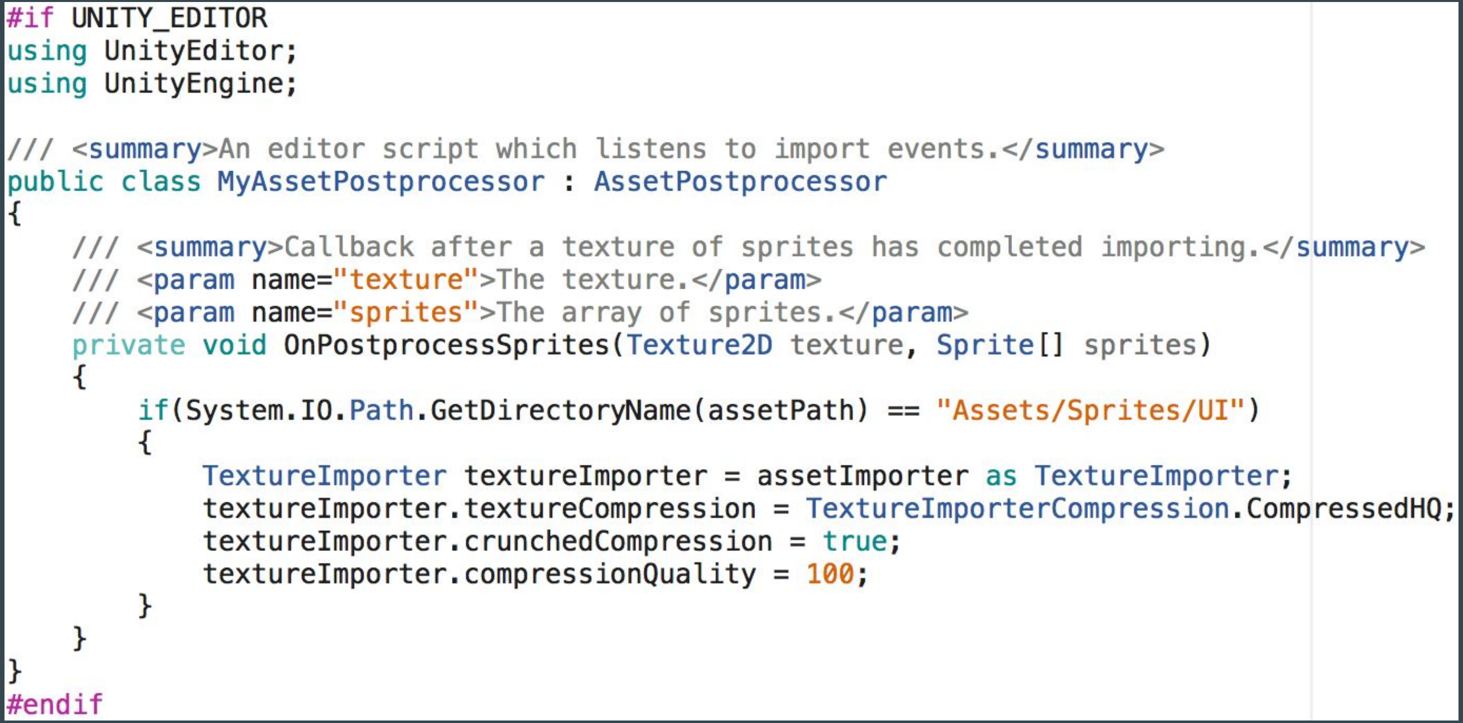
- AssetPostprocessor is an Editor class which allows access to the import pipeline and the ability to run scripts before or after importing assets.
- Thus import settings can be specified in code and run on all assets of a certain type (i.e. audio) or those in a specific folder (i.e. Assets/Sprites/UI).

2.1 Asset Import : Audio Preprocess



```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

/// <summary>An editor script which listens to import events.</summary>
public class MyAssetPostprocessor : AssetPostprocessor
{
    /// <summary>Callback before an audio clip is imported.</summary>
    private void OnPreprocessAudio()
    {
        AudioImporter audioImporter = assetImporter as AudioImporter;
        audioImporter.forceToMono = true;
        audioImporter.preloadAudioData = false;
        AudioImporterSampleSettings settings = new AudioImporterSampleSettings()
        {
            loadType = AudioClipLoadType.DecompressOnLoad,
            compressionFormat = AudioCompressionFormat.Vorbis,
            quality = 0,
            sampleRateSetting = AudioSampleRateSetting.OverrideSampleRate,
            sampleRateOverride = 22050
        };
        audioImporter.defaultSampleSettings = settings;
    }
}
#endif
```

2.2 Asset Import : Sprite Postprocess

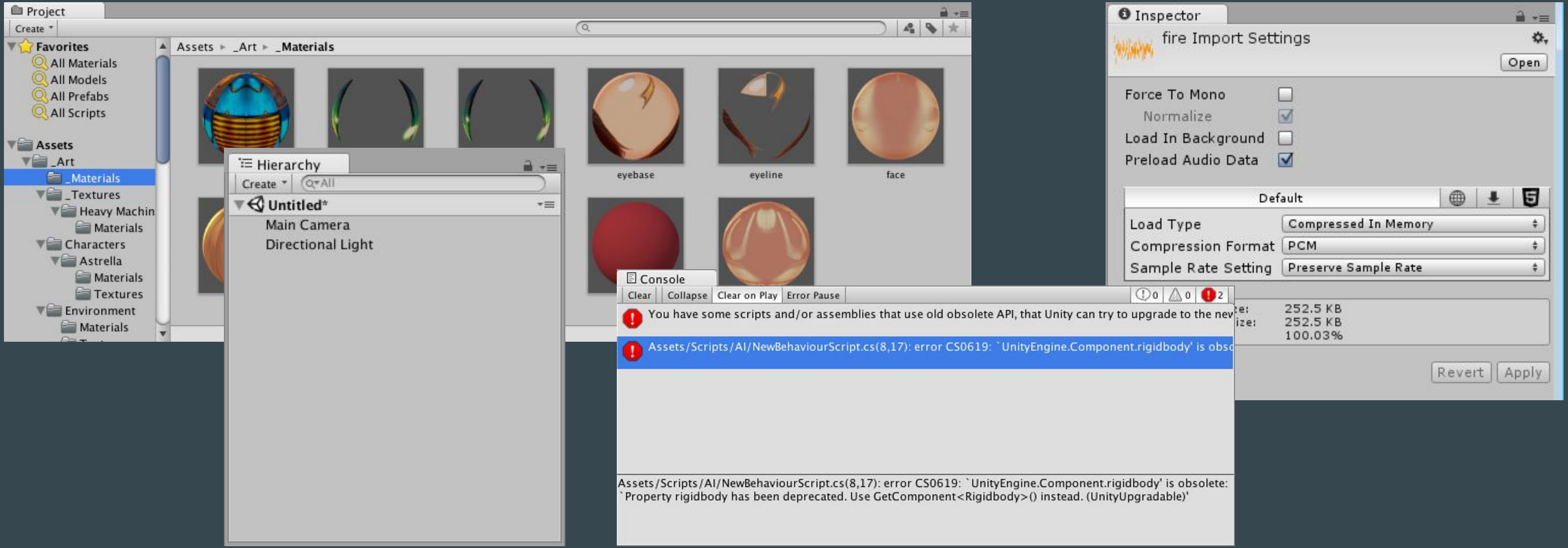


```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

/// <summary>An editor script which listens to import events.</summary>
public class MyAssetPostprocessor : AssetPostprocessor
{
    /// <summary>Callback after a texture of sprites has completed importing.</summary>
    /// <param name="texture">The texture.</param>
    /// <param name="sprites">The array of sprites.</param>
    private void OnPostprocessSprites(Texture2D texture, Sprite[] sprites)
    {
        if(System.IO.Path.GetDirectoryName(assetPath) == "Assets/Sprites/UI")
        {
            TextureImporter textureImporter = assetImporter as TextureImporter;
            textureImporter.textureCompression = TextureImporterCompression.CompressedHQ;
            textureImporter.crunchedCompression = true;
            textureImporter.compressionQuality = 100;
        }
    }
}
#endif
```

3. Custom Windows and Custom Menus

3.1 Custom Windows

The Unity Editor is a collection of windows.



You have some scripts and/or assemblies that use old obsolete API, that Unity can try to upgrade to the new

Assets/Scripts/AI/NewBehaviourScript.cs(8,17): error CS0619: `UnityEngine.Component.rigidbody' is obso

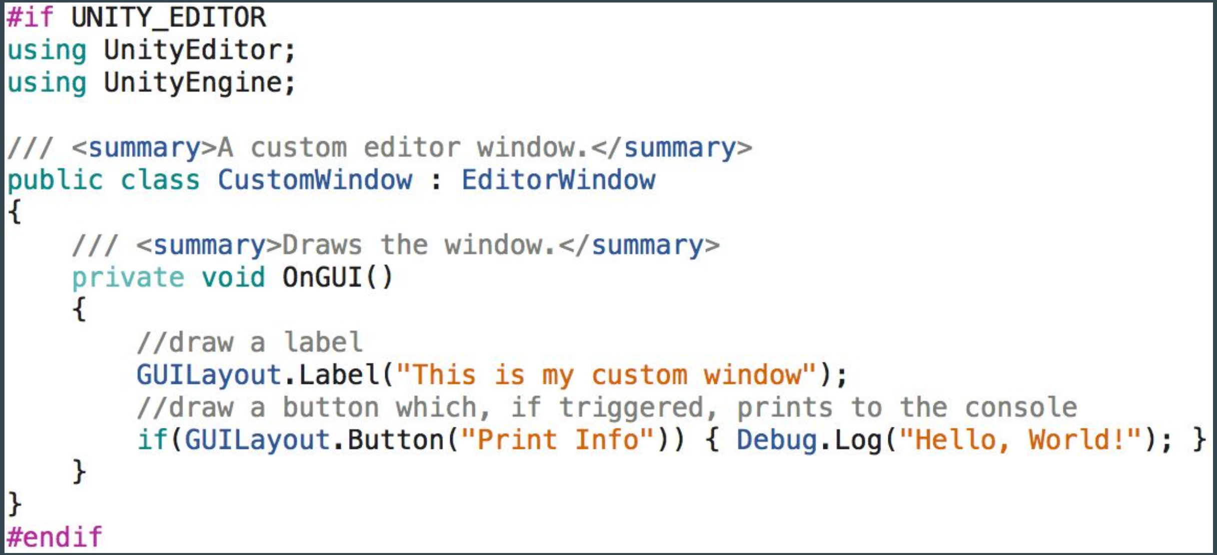
```
Assets/Scripts/AI/NewBehaviourScript.cs(8,17): error CS0619: `UnityEngine.Component.rigidbody' is obsolete:
`Property rigidbody has been deprecated. Use GetComponent<Rigidbody>() instead. (UnityUpgradable)'
```

Force To Mono	☐
Normalize	☑
Load In Background	☐
Preload Audio Data	☑
Load Type	Compressed In Memory
Compression Format	PCM
Sample Rate Setting	Preserve Sample Rate

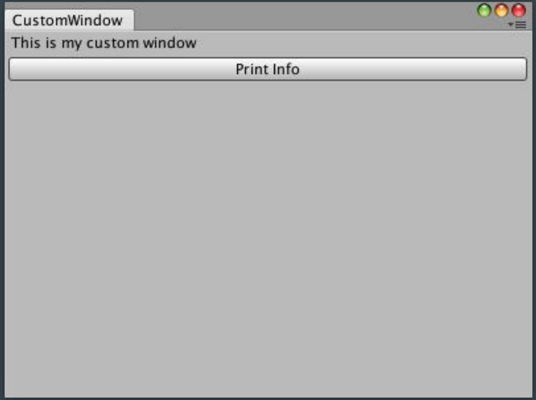
…e:	252.5 KB
…ize:	252.5 KB
	100.03%

3.1 Custom Windows

Custom windows can easily be created by extending a new script from EditorWindow and implementing OnGUI.

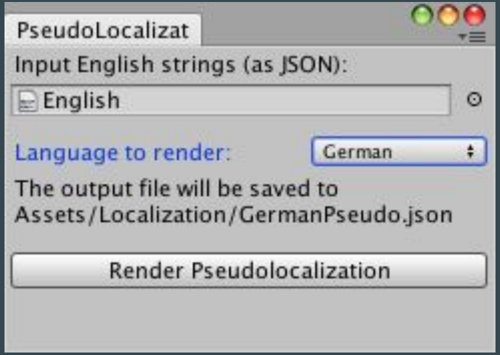


```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

/// <summary>A custom editor window.</summary>
public class CustomWindow : EditorWindow
{
    /// <summary>Draws the window.</summary>
    private void OnGUI()
    {
        //draw a label
        GUILayout.Label("This is my custom window");
        //draw a button which, if triggered, prints to the console
        if(GUILayout.Button("Print Info")) { Debug.Log("Hello, World!"); }
    }
}
#endif
```



This is my custom window

3.1 Custom Windows


PseudoLocalizat

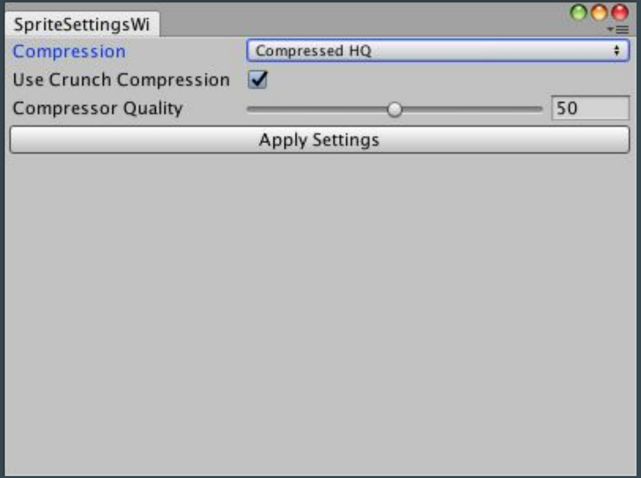
Input English strings (as JSON):

English

Language to render: German

The output file will be saved to
Assets/Localization/GermanPseudo.json

A window which renders a Pseudolocalization for a target language.


SpriteSettingsWi

Compression: Compressed HQ

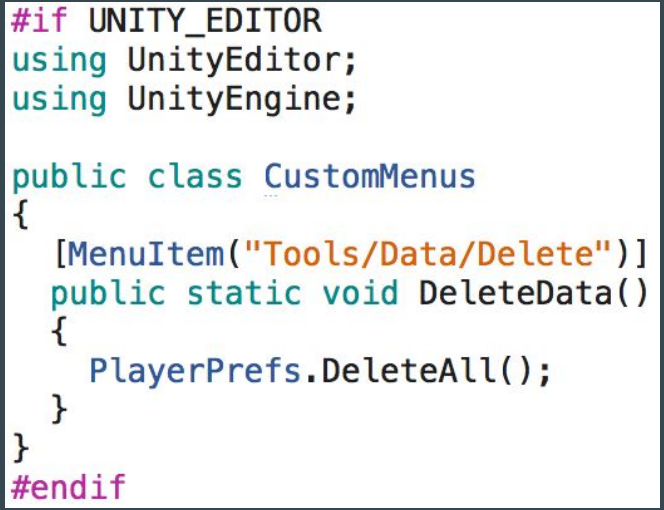
Use Crunch Compression: ✓

Compressor Quality: 50

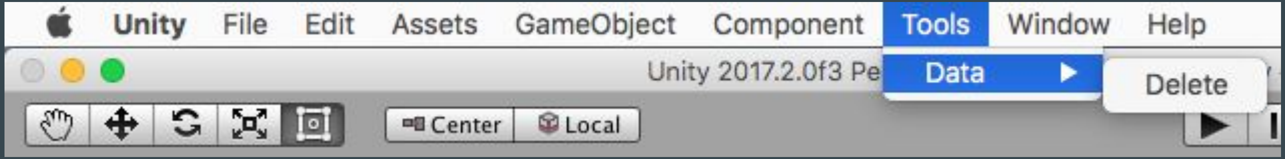
A window which updates compression settings for all sprites.

3.2.1 Custom Menus

The Unity Editor offers the addition of custom menus which look and behave like built-in menus.

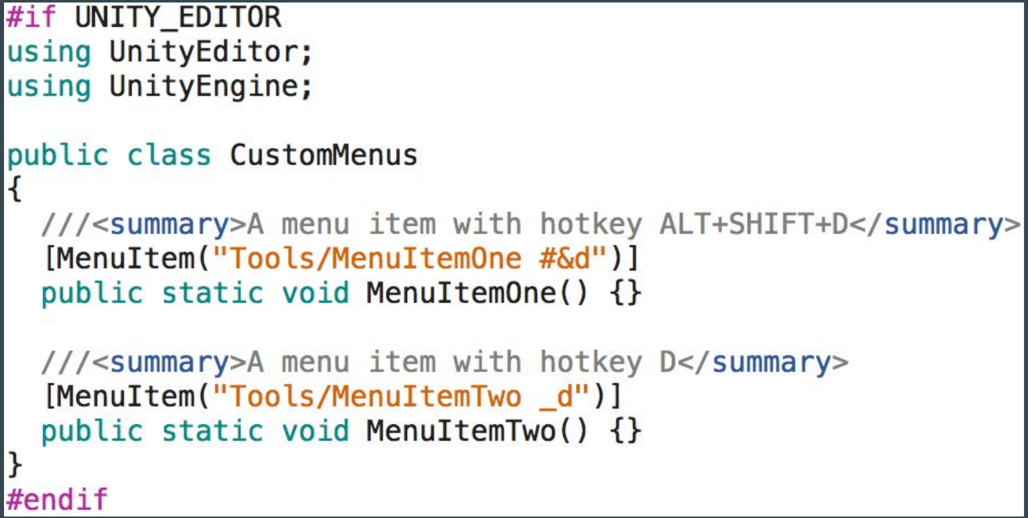


```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

public class CustomMenus
{
  [MenuItem("Tools/Data/Delete")]
  public static void DeleteData()
  {
    PlayerPrefs.DeleteAll();
  }
}
#endif
```

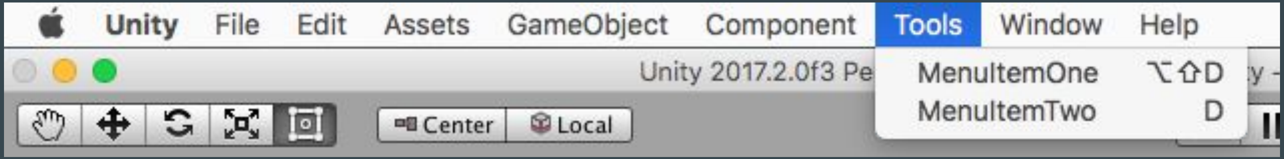


3.2.2 Custom Menus

These menu items can be assigned optional hotkey combinations (i.e. shortcuts) to automatically launch them.



```
#if UNITY_EDITOR
using UnityEditor;
using UnityEngine;

public class CustomMenus
{
  ///<summary>A menu item with hotkey ALT+SHIFT+D</summary>
  [MenuItem("Tools/MenuItemOne #&d")]
  public static void MenuItemOne() {}

  ///<summary>A menu item with hotkey D</summary>
  [MenuItem("Tools/MenuItemTwo _d")]
  public static void MenuItemTwo() {}
}
#endif
```



3.2.3 Custom Menus

These hotkey combinations can be very useful to trigger functionality that would generally only be possible via mouse interactions.

Shortcut	Action
ALT + C	Copy Transform
ALT + V	Paste Transform
ALT + UP	Move sibling up
ALT + DOWN	Move sibling down
ALT + L	Lock Inspector
ALT + K	Toggle Inspector Debug Mode
SHIFT + F4	Close Current Editor Window
SHIFT + ALT + C	Clear Console

Much, Much More!

- Gizmos
- ScriptableObjects
- Custom Attributes
- PropertyDrawer, DecoratorDrawer
- Editor Dialogs
- ContextMenu, ContextMenuItem
- GUIStyle, GUISkin
- Asset Store

Conclusion

- Well laid-out inspectors are easier to understand and use.
- Automate import settings.
- Use hotkey combinations instead of mouse clicking.
- A few minutes saved per person per day adds up to a very large number for a team over a project’s lifetime. Time saved = Money saved.
- Listen to your team’s needs.

Dziękuję!