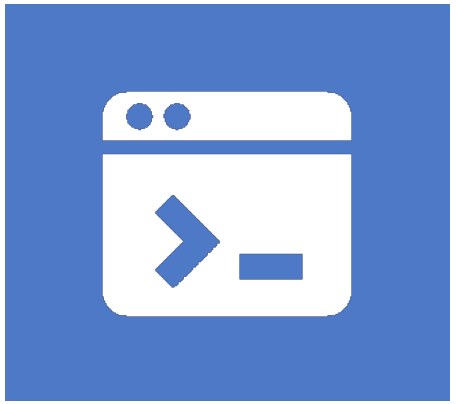
Advanced Game Programming



Week 8

Homework Review

Recursion and Optimization

Assets in Builds

Scriptable Objects

What is a Scriptable Object?

- ScriptableObject is a serializable Unity class
- Allows you to store large quantities of shared data independent from script instances.
- Can be easily edited in editor

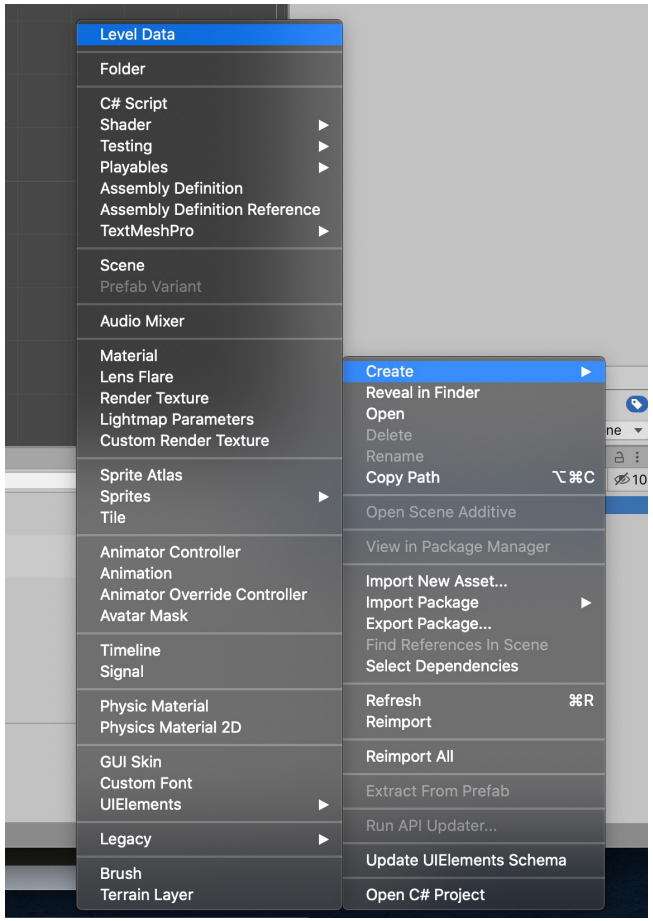
When to Use Scriptable Objects for Data

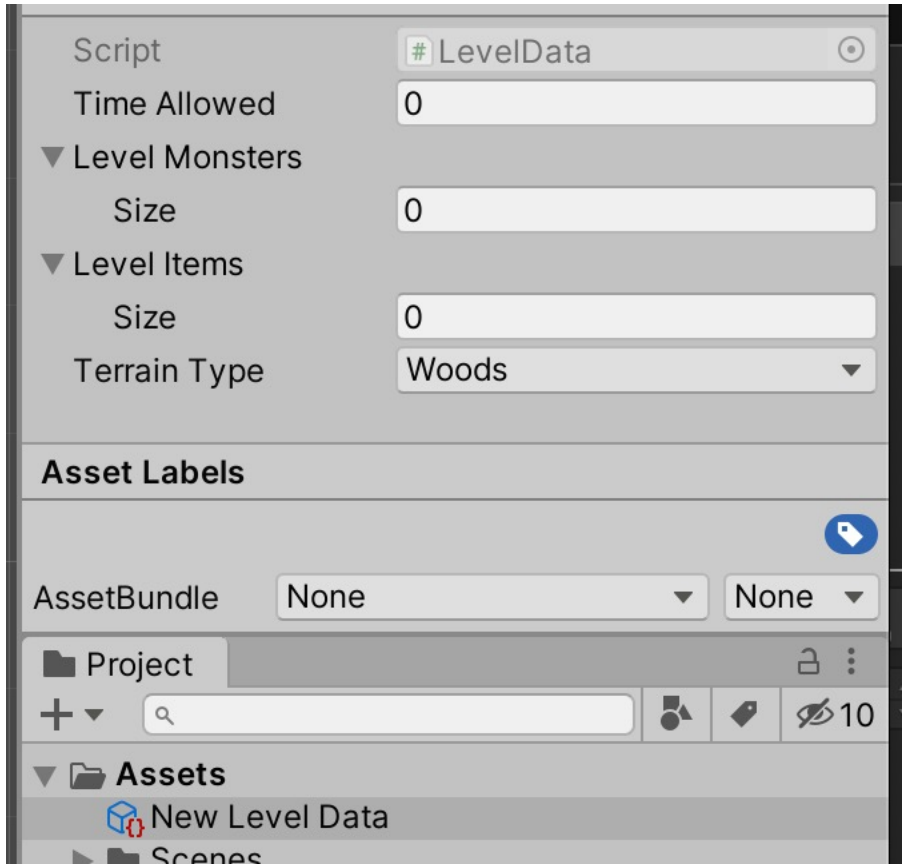
- Structured data you want to edit in editor
- Configuration Data
  - Levels
  - Color Palettes
  - Audio Palettes
- When you want to load data into memory at once
  - Prefab Database
  - Audio SFX Clip Library

Scriptable Object for Data

```
[CreateAssetMenu(menuName = "Level Data")]
public class LevelData : ScriptableObject
{
    public float _timeAllowed;
    public Monster[] levelMonsters;
    public Item[] levelItems;
    public TerrainType terrainType;
}
```

Creating Scriptable Object





Cons

- If you have large scriptable objects, every time you get any piece of data it must load entire scriptable object each time
  - Same issue as any large struct or class
- Editable in Editor

When Not To Use Scriptable Objects For Data

- The size of the scriptable object will quickly grow large
  - All the music in a music heavy game
  - All of the dialogue in a game
- The structure of the scriptable object will quickly devolve
  - All the configuration data in your game in one scriptable object
- There’s no reusable structure
  - Settings for only one type of one NPC

Advanced Scriptable Objects

- Scriptable Objects don’t have to be just data.
- Core systems that persist throughout runtime can often be Scriptable Objects
  - Don’t have Transforms
  - Won’t get Update functions
  - Will maintain state between scene loads without any special initialization
  - Can be hotswapped with any scriptable object

- EX: An Inventory system that is a Scriptable Object
  - Can swap in a test inventory or tutorial inventory
  - Can view contents in Editor easily

Advanced Scriptable Objects (cont.)

- Scriptable objects keep state after runtime
  - Can use for developing AI or any system where play over sessions is important.
  - *Don’t save during runtime once built, only in editor*

Prevent Serializing Runtime Changes

```
[CreateAssetMenu(menuName = "Example Scriptable Object")]
public class ExampleScriptableObject : ScriptableObject
{

    [SerializeField]
    private float _storedValue;

    [NonSerialized]
    public float RuntimeValue;

    public void OnAfterDeserialize() {
        RuntimeValue = _storedValue;
    }

    public void OnBeforeSerialize() { }
}
```

AssetBundles

Common Types of Assets

- Image files
  - Unity supports most common image file types, such as BMP, TIF, TGA, JPG, and PSD.
  - If you save your layered Photoshop (.psd) files into your Assets folder, Unity imports them as flattened images.
- FBX and Model files
  - You can save 3D files from most common 3D modeling software in their native format (for example, .max, .blend, .mb, .ma).
  - When Unity finds them in your Assets folder, it imports them by calling back to your 3D modeling software's FBX export plug-in
- Meshes and animations
- Audio files
- Other Asset types

AssetBundles Have Been Depreciated!

- Asset Bundle were external collections of assets.
- These files exist outside of the built Unity player, usually sitting on a web server for end-users to access dynamically.
- To build an Asset Bundle, you call BuildPipeline.BuildAssetBundles() from inside an Editor script.
  - In the arguments, you specify an array of **Objects** to be included in the built file, along with some other options.
- This will build a file that you can later load dynamically in the runtime by using AssetBundle.LoadAsset().
- You can unload resources of an AssetBundle by calling AssetBundle.Unload().
  - If you pass **true** for the **unloadAllLoadedObjects** parameter, both the objects held internally by the AssetBundle and the ones loaded from the AssetBundle using AssetBundle.LoadAsset() will be destroyed and memory used by the bundle will be released.

Addressables

Unity Addressable Asset System

- Load assets by “address”
- Handles content pack creation and deployment.
- Uses asynchronous loading

What is an Addressable Asset

- Making an asset "Addressable" allows you to use that asset's unique address to call it from anywhere.
  - In local application
  - On a content delivery network,
- You can load a single Addressable Asset via its address, or load many Addressable Assets using a custom group label that you define.

Why use Addressable Assets?

- **Iteration time**
  - Optimizations to content no longer require changes to code
- **Dependency management**
  - All meshes, shaders, animations, and other dependencies load before returning content
- **Memory management**:
  - System unloads assets as well as loading them
  - Counts references automatically
  - Providing a robust profiler to help spot memory issues
- **Content packing**
  - System maps and understands complex dependency chains
  - Efficient packing of bundles, even when moving or renaming assets.
  - Easily prepare assets for both local and remote deployment
  - Supports downloadable content and reduced application size
- **Profiles**:
  - Create string variables to change how content is put into bundles without modifying settings in multiple places.

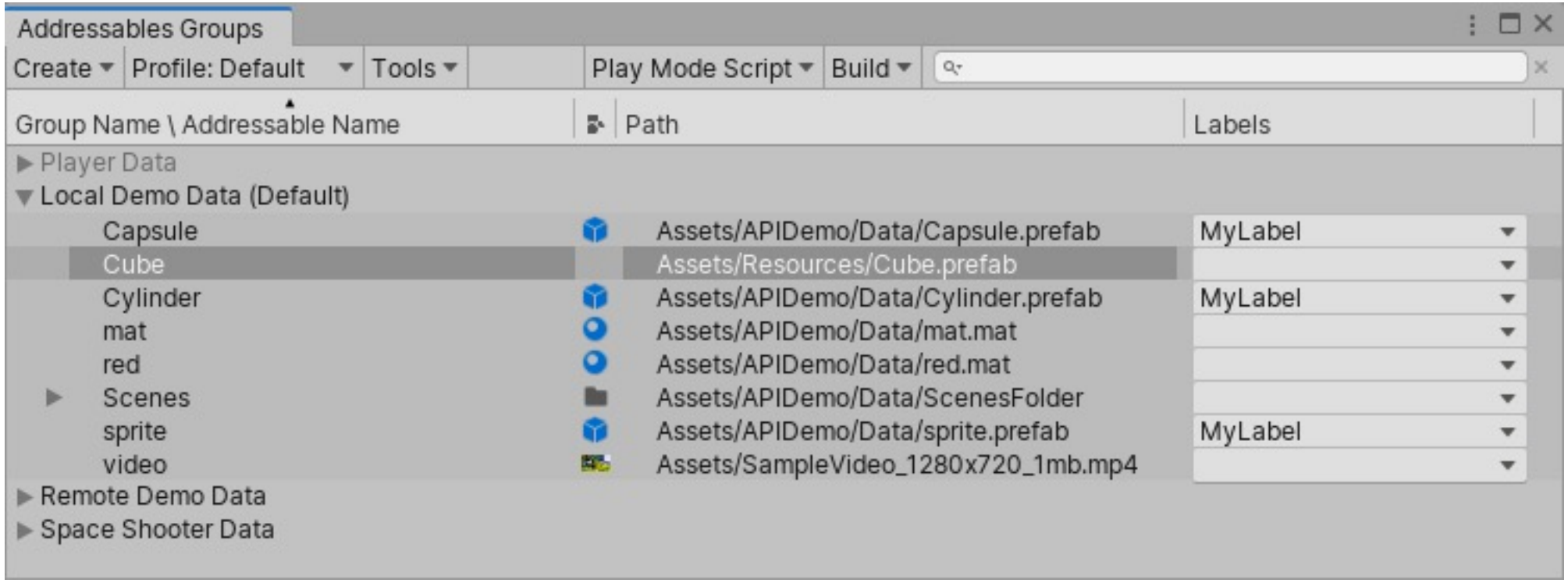
Upgrading to Addressables

- Direct References:
  - Add assets directly into components or Scenes, which the application loads automatically.
- Resource Folders:
  - Add assets to your Resource folder and load them by filename.
- Asset Bundles:
  - Add assets to asset bundles, then load them with their dependencies by file path.

Migrating Direct References

- Replace your direct references to objects with asset references
  - `public GameObject directRefMember;`
  - `public AssetReference AssetRefMember;`
- Drag assets onto the appropriate component’s Inspector, as you would for a direct reference.
- If you'd like to load an asset based on an object rather than a string name, instantiate it directly from the AssetReference object
  - `AssetRefMember.LoadAssetAsync<GameObject>();`
  - `AssetRefMember.InstantiateAsync(pos, rot);`

Marking Asset as Addressable



Asynchronous Loading

- The Addressable Asset system loads assets asynchronously.
- When you update your direct references to asset references, you must also update your code to operate asynchronously.

Asynchronous Loading

- Addressables provides a multicast you can subscribe to:

```
public class AddressablesExample : MonoBehaviour {

    GameObject myGameObject;

        ...
        Addressables.LoadAssetAsync<GameObject>("AssetAddress").Completed += OnLoadDone;
    }

    private void OnLoadDone(UnityEngine.ResourceManagement.AsyncOperations.AsyncOperationHandle<GameObject> obj)
    {
        // In a production environment, you should add exception handling to catch scenarios such as a null result.
        myGameObject = obj.Result;
    }
}
```

Migrating Resource Folders

- Mark asset in a *Resources* folder as Addressable
- Unity will automatically move the asset from the *Resources* folder to a new folder in your Project named *Resources_moved*.
- The default address for a moved asset is the old path, omitting the folder name.

```
Resources.LoadAsync<GameObject>("desert/tank.prefab");

Addressables.LoadAssetAsync<GameObject>("desert/tank.prefab");
```

When to Use Addressables System

- Your project has *a lot* of assets.
- Your project is targeting a system with limited memory.
- You’re having issues w/long load times, or spikes from Garbage Collection.

Tool Development

"Remember, tools live longer than games do."

- John Romero (link)

"I think every company needs to make the decision to either invest significantly more in tools, or just not bother.”

- Tim Sweeny (link)

"In the metaphorical game development gold rush, there are the devs out hunting for gold, and there are the people making shovels for them."

- John Harris (link)

What is Tool Development?

- Tool development is creating the scripts, applications, servers, and environment surrounding the development of a game.
- It’s a role usually reserved for AAA studios
- At larger indie studios, there’s sometimes an expectation that programmers will be able to do this kind of work.
- Indies have to be their own tool developers.
- Almost all game development tools are developed by the developer custom for one game, or by a console manufacturer (such as Nintendo or Microsoft) as part of a game development kit.

Tool Developer

- a.k.a. **Game Tools Programmer, Engine Tools Engineer**
- Programmer dedicated to producing tools and utilities to increase the productivity of fellow game developers.
- Works directly with artists and designers to develop workflows and pipelines for efficient content production.
- Help create tools for other programmers on the game development team (game logic, engine, server and graphics programmers).

Areas of Tool Development Covered

- Asset Authorship
- Asset Manipulation
- Generation
- Analysis / Modeling

Other Areas w/in “Tool Development”

- Source Control
- Mastering final game for consoles
- Build Systems / Continuous Integration
- Applications for back-end servers
- Pipeline
- Modifying existing tools
  - Writing plugins for content creation tools

Glossary

- DCC Tools
  - Blender, Maya, Zbrush, 3DS Max, Photoshop
- Pipeline
  - a.k.a. Content Pipeline, Production Pipeline, Asset Build Pipeline, Asset Pipeline, Build Pipeline, Asset Conditioning Pipeline, ACP, Tool Chain, Resource Conditioning Pipeline, RCP, Build
  - Process that transforms asset from idea to a part of the game binary.
  - This process lets the artists use the DCC tools that they like while meeting the needs of the development team.

Assets

Authorship and Manipulation

Overview

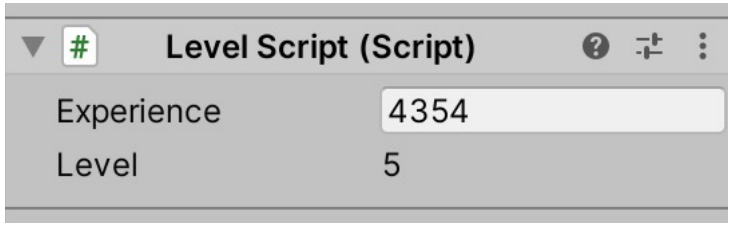
- Asset Authorship
  - Unity Editor Scripting
  - Creating external authorship tools
    - Dialogue writing systems
    - Level/Sound/Graphics Editors
- Asset Manipulation
  - Processing assets into a game engine or other content creation tool
    - Import
    - Export
  - Processing assets for build(s)
  - Localization

Level/Audio/Graphics Editors

- Editors are like very simple games, that write out or read in serialized data.
- If packaged with game or used as a standalone application, need to be able to be built outside of Unity
- If used inside Unity, may need to be a Unity Editor Script

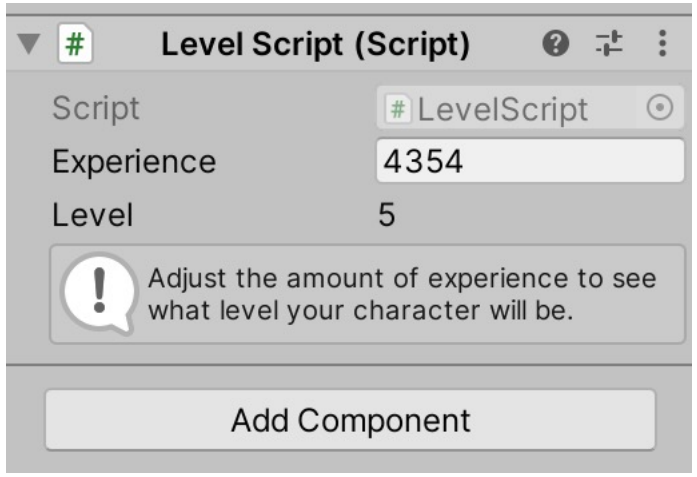
Unity Editor Scripting

```
using UnityEditor;
using UnityEngine;

public class LevelScript : MonoBehaviour
{
    public int experience;
    public int Level => experience / 750;
}

[CustomEditor(typeof(LevelScript))]
public class LevelScriptEditor : Editor
{
    public override void OnInspectorGUI()
    {
        var myTarget = (LevelScript) target;

        myTarget.experience = EditorGUILayout.IntField("Experience", myTarget.experience);
        EditorGUILayout.LabelField("Level", myTarget.Level.ToString());
    }
}
```



DrawDefaultInspector method

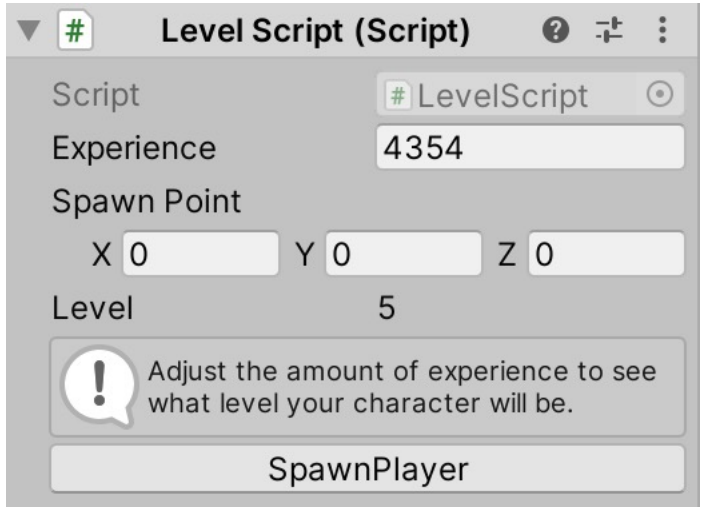
```
public override void OnInspectorGUI()
{
    var myTarget = (LevelScript) target;

    DrawDefaultInspector();
    EditorGUILayout.LabelField("Level", myTarget.Level.ToString());
    EditorGUILayout.HelpBox("Adjust the amount of experience to see what level your character will be.", MessageType.Info);

}
```



Buttons

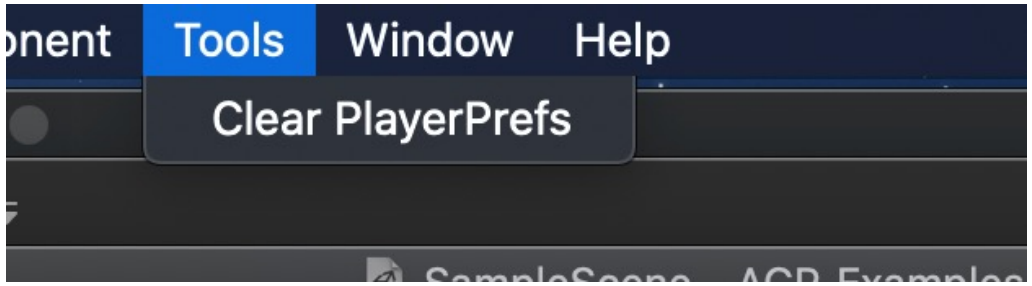
```
public void SpawnPlayer()
{
    Instantiate( original: Resources.Load<GameObject>( path: "Prefabs/Example"),
        spawnPoint, Quaternion.identity);
}
```

```
if(GUILayout.Button( text: "SpawnPlayer"))
{
    myTarget.SpawnPlayer();
}
```

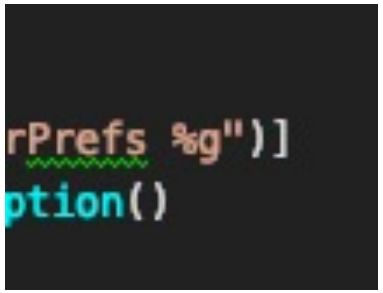


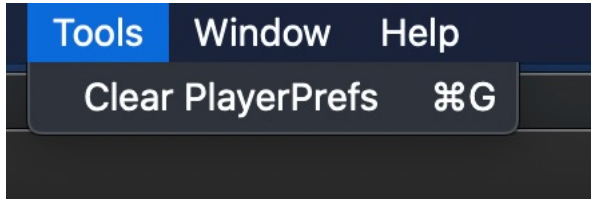
Adding MenuItems

```
public class MenuItems
{
    [MenuItem("Tools/Clear PlayerPrefs")]
    private static void NewMenuOption()
    {
        PlayerPrefs.DeleteAll();
        PlayerPrefs.Save();
    }
}
```



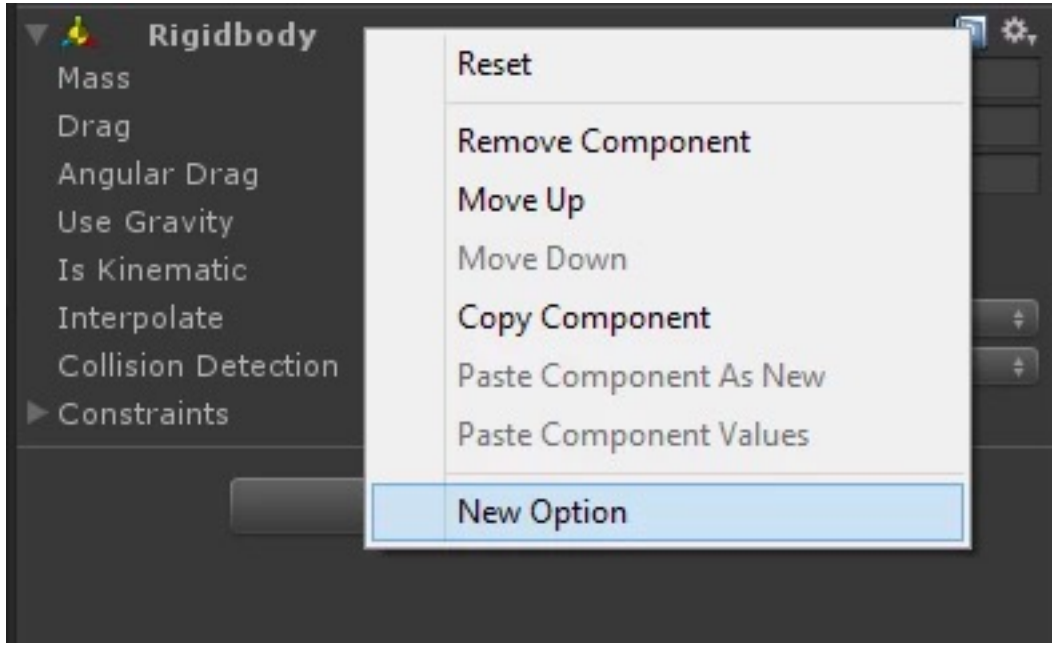
Adding Keyboard Shortcuts to MenuItems





- % – CTRL on Windows / CMD on OSX
- # – Shift
- & – Alt
- LEFT/RIGHT/UP/DOWN – Arrow keys
- F1…F2 – F keys
- HOME, END, PGUP, PGDN
- Character keys not part of a key-sequence are added by adding an underscore prefix to them (e.g: _g for shortcut key “G”).

Special Paths for MenuItems



- Assets
  - Available under the “Assets” menu,
  - Right-click inside the project view.
- Assets/Create
  - “Create” button in the project view
- CONTEXT/ComponentName
  - right-clicking inside the inspector of the given component.

Other Attributes

- [RequireComponent(typeof(Rigidbody))]
- Reset method
- [HideInInspector]
- [Range(0, 100)]
- [ContextMenu(“Fire”)]

```
[RequireComponent(typeof(Rigidbody))]
No asset usages
public class Projectile : MonoBehaviour
{
    [HideInInspector] new public Rigidbody rigidbody;   Unchanged
    [Range(0, 100)] public float velocity = 10;   Unchanged

    [ContextMenu( itemName: "Fire")]
    void Fire()
    {
        // instantiate and fire w/velocity
    }
    Event function
    void Reset()
    {
        rigidbody = GetComponent<Rigidbody>();
    }
}
```

Processing Assets in/out of DCC/Engines

- Into or out of DCC or Engines
  - Convert filetypes
  - Compress or uncompress files
- Processing Assets for Builds

Localization

Need systems for changing out not just words, but behavior:

- Different languages have different separators (sentence enders, paragraph enders)
- Different languages have different read directions
  - left -> right vs right -> left
  - Horizontal vs. vertical
- Different languages a character is a part of a sound, a complete sound, or a word.
- Different languages use or do not use spaces.

Generation and Analysis

Overview

- Generation
  - Puzzle creation
  - Difficulty assessment
- Analysis / Modeling
  - Content verification
  - Visualization of Data
  - Writing analytics hooks
  - Managing analytics database

Puzzle Generation/Difficulty Assessment

- Puzzle game w/ anagrams
- Puzzle game w/ sums of numbers
- Maze

Python

Why Python?

- Simple syntax rules (easy to read/maintain/learn)
- Interactive shell and interpreted, not compiled
  - Easy to create command line applications with it.
- Strongly typed, and dynamically typed
- Multi-paradigm (OOP, functional, structured)
- Libraries
  - SciPy (scientific and technical computing)
  - NumPy (math and statistics)
  - Scikit-learn, TensorFlow, mlpy (machine learning)
  - Pandas (data analysis)
  - Mathplotlib (data visualization)

Why Python?

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	20.266% (-0.472%)	
2	Python	17.577% (-0.218%)	
3	Java	10.177% (+0.208%)	
4	Go	8.287% (+0.402%)	
5	C++	6.858% (-0.040%)	
6	Ruby	6.802% (+0.189%)	
7	TypeScript	6.249% (+1.222%)	^
8	PHP	5.279% (-0.684%)	v
9	C#	3.629% (+0.218%)	

Why not make everything in Unity?

- Unity is very heavy weight
  - Every project includes a lot of games-specific libraries
  - Every project must include a visual aspect
  - Loading and unloading assets is slow
- Knowing and working in more languages is very, very important
- Learning other programming paradigms is important
  - Functional programming
  - Command-line programming
- Unity may not be commonly used in the future

Shape of Python

- No semicolons
- Code blocks start with ‘:’
- Whitespace is important
- No braces or parentheses

```
def incrementX() :
    x = 1
    while x < 10 :
        print("x is " + x)
        x = x + 1
```

Conditional

```python
age_input = input('Please enter your age: ')
age = int(age_input)
if age >= 21:
    print("You can come in.")
    print('Drop by 123 Vine St at 8pm')
else:
    print('I\'m sorry, you can\'t come in.')
```

```python
def test() :
    if not isAvailable :
        print('Not available.')
    elif waiting and not loggedIn :
        print('You need to log in.')
    elif connected or waitingToConnect :
        print('You are waiting to connect.')
    elif key in ['example', 'key', 'list'] :
        print('Key is in the above list')
    else :
        print('Not connected to the internet.')
```

Loops

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while a < 10 :
    print(a)
    a, b = b, a+b


for i in range(0, 5) :
   print(i)


numbers = [1, 2, 3, 4, 5, 6]
for n in numbers :
    print(n, ',')
```

Why Not Use Python for Everything?

- It’s not as effective as something that gives you lower level access to the machine.
- It’s not as protective as a more restrictive language
  - More difficult to create large-scale systems with it.
- It’s not particularly specialized
- You need external libraries for creating visuals or adding physics

Example

```
book1Words = []

count = 0
with open('book1.txt') as book1:
    for line in book1.readlines():
        for word in line.split('\n'):
            if len(word) == 4 and word.isalpha() and word not in book1Words:
                book1Words.append(word.upper())
                count+=1

print('Total in book1:', count)
count = 0

with open('book2.txt') as book2:
    with open('output.txt', 'w') as filehandle:
        for line in book2.readlines():
            for word in line.split('\n'):
               if word.isalpha() and word in book1Words:
                   filehandle.write('%s\n' % word)
                   count+=1

print('Total in both:', count)
```

Tool Design Mistakes

- Design as you go
- The system model of design
- Leveraging the wrong technology
- Complicating the interface
- Extraneous features
- Designing for advanced users

Resources

Articles/Documents

- Toolsmiths (http://thetoolsmiths.org/)
- Tool Engineer Learning Path (link)
- Python Tutorial (link)

Videos