Unity Tools 3

Advanced Scripting

Index

- Handles and gizmos
- Unity reserved editor methods
  - Exercise 1: Turret Tool
    - Turret movement behavior
    - Turret developer interaction (gizmo + action buttons)
- Unity editor windows
- Unity editor styles
  - Exercise 2: Tile generator
    - Place prefabs along grid positions
    - Update materials from selected game objects.
  - Exercise 3: Play save mode
- Editor toolbar customization

Index

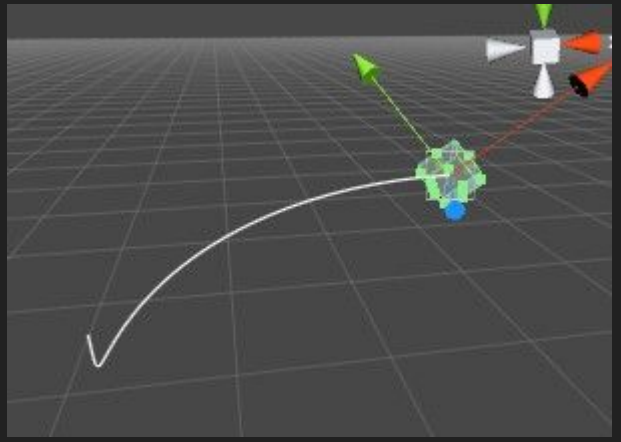
- MVD Editor window
  - Utility tools
  - Placement tool
- Spline tool
  - Move objects along spline
- Final deliver

Unity Tools: Methods list

- Editor reserved method names, used for different purposes.

Methods	Description
OnDrawGizmos	Draw gizmo on scene
OnDrawGizmoSelected	Draw gizmo when object selected on scene
OnInspectorGUI	Draws custom component properties
OnSceneGUI	Draws GUI on scene
OnGUI	Draw custom window GUI
OnValidate	When a property of a component is changed.
DrawDefaultInspector	Draws default inspector GUI
Repaint	Used to repaint the inspector GUI

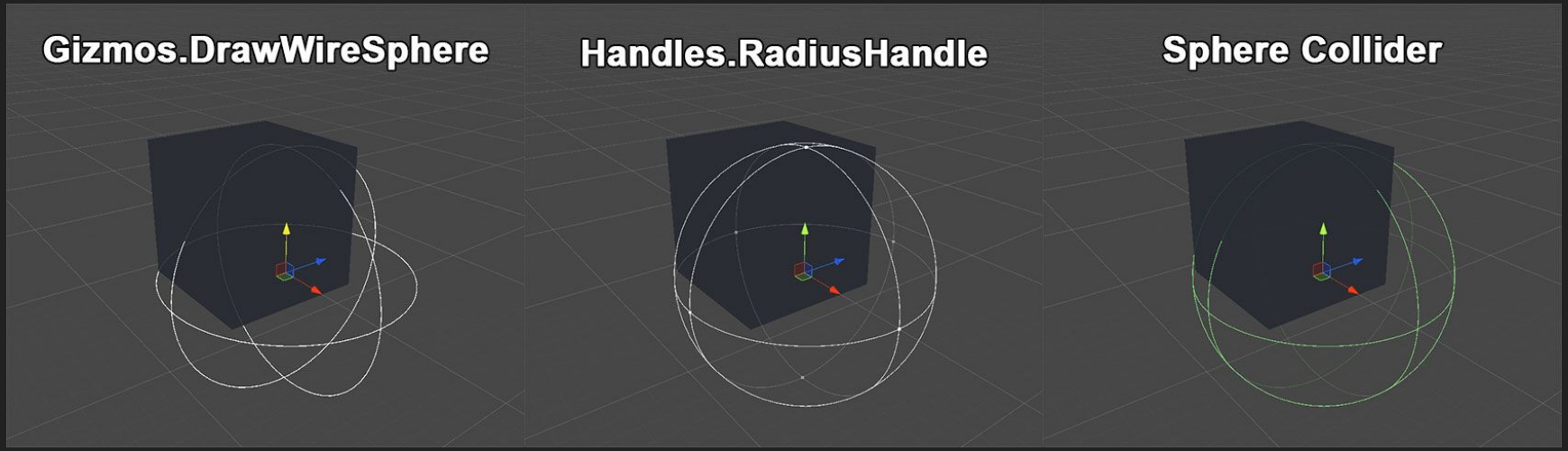
Unity Tools: Handles & gizmos



- Handles
  - Custom 3D gui controls in the scene view
  - You can define your own custom handles
  - Very useful to realtime control groups and entities on the scene view.
  - Focused in object manipulation
  - Scripting API, many mathematical functions already in.
  - Reference: https://docs.unity3d.com/ScriptReference/Handles.html

- Gizmos
  - Gizmos are collection of built-in handles
  - These are normally primitive shapes to manipulate objects
  - Reference: https://docs.unity3d.com/ScriptReference/Gizmos.html

Unity Tools: Inspector tools



Unity Tools: Examples

- Exercise 1: Turret Tool (Player shooting)
  - Projectile Editor
  - Launcher Editor
  - Debug launcher on player.
- Exercise 2: ...
- Exercise 3: ...

Unity Tools: Exercise 1

- Code requirements
  - Create a projectile script
  - Create a launcher script
  - Create a projectile editor script
  - Create a launcher editor script
  - Correctly set gameobject layers

- Functionality requirements
  - Define an offset position as firing point
  - Define two button actions to aim and fire independently
  - Define a polyline to describe bullets trajectory.
  - Projectile launcher and settings must be able to be changed.

Classes provided

Projectile.cs
Launcher.cs
ProjectileEditor.cs
LauncherEditor.cs

Unity Tools: Editor scripting 1

- How to expose methods in the Inspector
- How to use Handles to create custom Gizmos
- How to use field attributes to customise the Inspector

```
1.  public class Launcher : MonoBehaviour
2.  {
3.    public Rigidbody projectile;
4.    public Vector3 offset = Vector3.forward;
5.    [Range(0, 100)] public float velocity = 10;
6.
7.    [ContextMenu("Fire")]
8.    public void Fire()
9.    {
10.     var body = Instantiate(
11.       projectile,
12.       transform.TransformPoint(offset),
13.       transform.rotation);
14.     body.velocity = Vector3.forward * velocity;
15.   }
16. }
```

```
1.  using UnityEditor;

1.  [CustomEditor(typeof(Launcher))]
2.  public class LauncherEditor : Editor
3.  {
4.      void OnSceneGUI()
5.      {
6.          var launcher = target as Launcher;
7.          var transform = launcher.transform;
8.          launcher.offset = transform.InverseTransformPoint(
9.              Handles.PositionHandle(
10.                 transform.TransformPoint(launcher.offset),
11.                 transform.rotation));
12.     }
13. }
```

Unity Tools: Editor scripting 2

- Widgets in the scene view
- We can edit the methods OnSceneGUI

```
using UnityEditor;
```

```
[CustomEditor(typeof(Launcher))]
public class LauncherEditor : Editor
{
    void OnSceneGUI()
    {
        var launcher = target as Launcher;
        var transform = launcher.transform;
        launcher.offset = transform.InverseTransformPoint(
            Handles.PositionHandle(
                transform.TransformPoint(launcher.offset),
                transform.rotation));
        Handles.BeginGUI();
        var rectMin = Camera.current.WorldToScreenPoint(
            launcher.transform.position +
            launcher.offset);

```

```
var rect = new Rect();
        rect.xMin = rectMin.x;
        rect.yMin =
SceneView.currentDrawingSceneView.position.height -
            rectMin.y;
        rect.width = 64;
        rect.height = 18;
        GUILayout.BeginArea(rect);
        using (new
EditorGUI.DisabledGroupScope(!Application.isPlaying))
        {
            if (GUILayout.Button("Fire"))

```

Resource: https://docs.unity3d.com/ScriptReference/Editor.OnSceneGUI.html

Unity Tools: GUI Layout

- Scope Areas
  - Areas of code, where UI elements are affected by the settings provided.

```
using UnityEngine;
using System.Collections;
using UnityEditor;

[CustomEditor(typeof(LevelScript))]
public class LevelScriptEditor : Editor
{
    public override void OnInspectorGUI()
    {
        using (new EditorGUI.IndentLevelScope())
        {
            EditorGUILayout.LabelField("X:", obj.rotation.x.ToString());
            EditorGUILayout.LabelField("Y:", obj.rotation.y.ToString());
            EditorGUILayout.LabelField("Z:", obj.rotation.z.ToString());
            EditorGUILayout.LabelField("W:", obj.rotation.w.ToString());
        }
    }
}
```

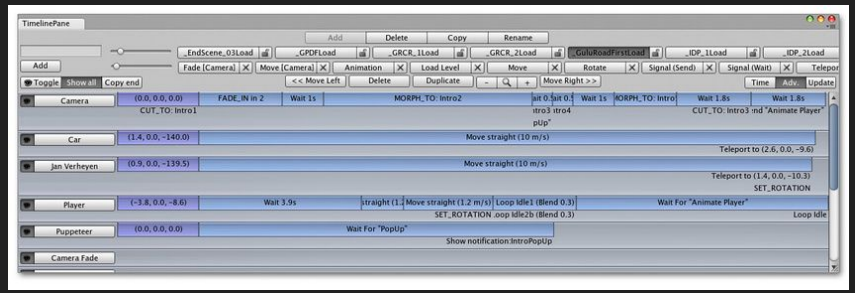
```
using UnityEngine;
using System.Collections;
using UnityEditor;

[CustomEditor(typeof(LevelScript))]
public class LevelScriptEditor : Editor
{
    public override void OnInspectorGUI()
    {
        Rect r = EditorGUILayout.BeginHorizontal("Button");
        if (GUI.Button(r, GUIContent.none))
            Debug.Log("Go here");
        GUILayout.Label("I'm inside the button");
        GUILayout.Label("So am I");
        EditorGUILayout.EndHorizontal();

    }
}
```

Resource: https://docs.unity3d.com/ScriptReference/GUILayout.html

Unity Tools: Extending the editor

- You can create any number of custom windows in your app.
- These behave just like the **Inspector**, **Scene** or any other built-in ones.
- To build a Unity Editor Window we need:
  - Create class deriving from EditorWindow
  - Use the property [MenuItem (“Window/My Window”)]
  - Use the method ShowWindow to display the window at the editor (Windows are recycled)
  - Implement functionality using OnGui method.

- Useful classes:
  - Editor Application
  - Editor Utility



Resource: https://docs.unity3d.com/ScriptReference/EditorUtility.html

Unity Tools: GUI Layout

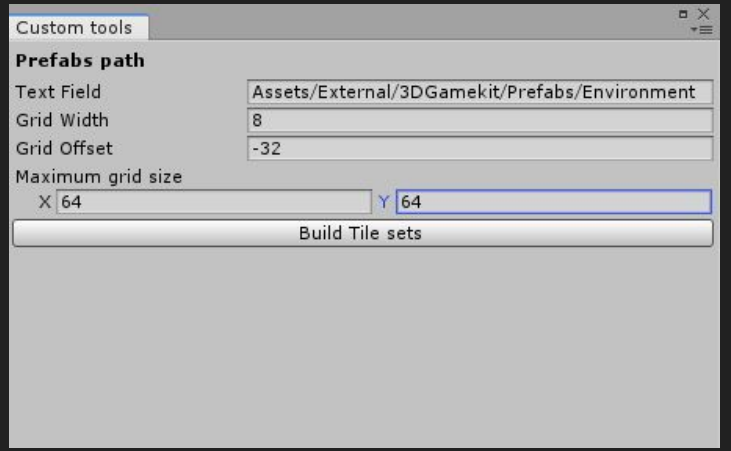
- GUI Edition is key on editor scripting
- GUI Edition works by scope areas
- Editor tools MUST NOT over complicate the work process. UI follow KISS principle

Open Methods	Close Methods	
BeginArea, BeginHorizontal, Begin...	EndArea, EndHorizontal….	To determine blocks of areas with align
BeginScrollView	EndScrollView	Scroll list, useful to handle assets
Button, Lebel, Toggle	None, done with scopes.	User action elements
TextArea, TextField	None	To fill properties with strings
Flexible space, height, width	None	To handle alignment and sizes
Handles, Gizmos	None	Visual debug elements

Resource: https://docs.unity3d.com/ScriptReference/GUILayout.html

Exercise: Tile Set Loader

- Build an editor window
- Define number of prefabs per row
- Define prefab grid size
- Define offset per prefab



Bonus: Material update

Code provided: EditorPrefabLoader.cs

Unity Tools: Editor styles

- Used to alter editor window appearance (skins)
- Allow us to customize
  - Text size, font…
  - Button, toolbar etc.. interaction items look.
  - Window color and documentation
  - Reference: https://game-loop.com/style-unity-editor-window-toolbar-buttons-with-guiskin/
- We can setup custom images
  - Source: https://gist.github.com/MattRix/c1f7840ae2419d8eb2ec0695448d4321

```
using UnityEditor;


[CustomEditor(typeof(Launcher))]
public class LauncherEditor : Editor
{
    void OnEditorStyles()
    {
            prefabElementStyle = new GUIStyle(GUI.skin.FindStyle("ShurikenModuleTitle"));
            prefabElementStyle.imagePosition = ImagePosition.ImageOnly;
            prefabElementStyle.margin = new RectOffset(0, 0, 0, 0);
            prefabElementStyle.padding = new RectOffset(0, 0, 0, 0);
            prefabElementStyle.clipping = TextClipping.Clip;
            prefabElementStyle.contentOffset = Vector2.zero;
            prefabElementStyle.padding = new RectOffset(4, 4, 4, 4);
            prefabElementStyle.alignment = TextAnchor.MiddleCenter;
            prefabElementStyle.fixedHeight = 0;
            prefabElementStyle.border = new RectOffset(4, 4, 4, 4);
    }
}
```

Unity Tools: Play mode save!

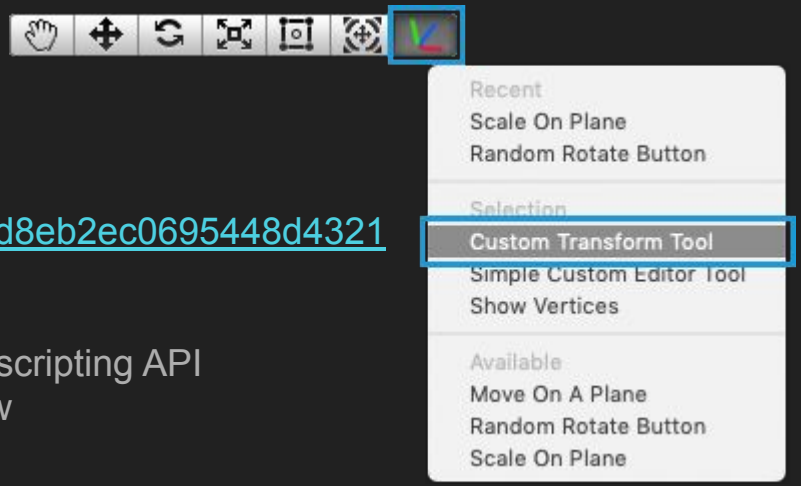
- First, remember to use color tints in play mode!!
  - Edit at preferences! https://unity3d.com/es/learn/tutorials/topics/tips/play-mode-editor-tint
  - You can also manually copy paste components

- Deep copy of the component in play mode
- Three different ways of doing it.
  - Prefab copy method
  - Component copy method
  - JSON Serialize method

- Our method: No support to update prefabs,
  - Hint: copy data into the original prefab, and reuse it.

Unity Tools: Editor toolbar

- Unity doesn't allow us to modify built-in windows
- Workarounds available
  - Custom third party toolbar edition
  - Scene window focus example
  - Source: https://gist.github.com/MattRix/c1f7840ae2419d8eb2ec0695448d4321



- Unity 2019 .1 supports toolbar customization
  - We can directly code toolbar within unity editor scripting API
  - Also we can edit any other built-in editor window

https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EditorTools.EditorTool.html
https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EditorTools.EditorTools.html
https://docs.unity3d.com/2019.1/Documentation/ScriptReference/EditorTools.EditorToolAttribute.html

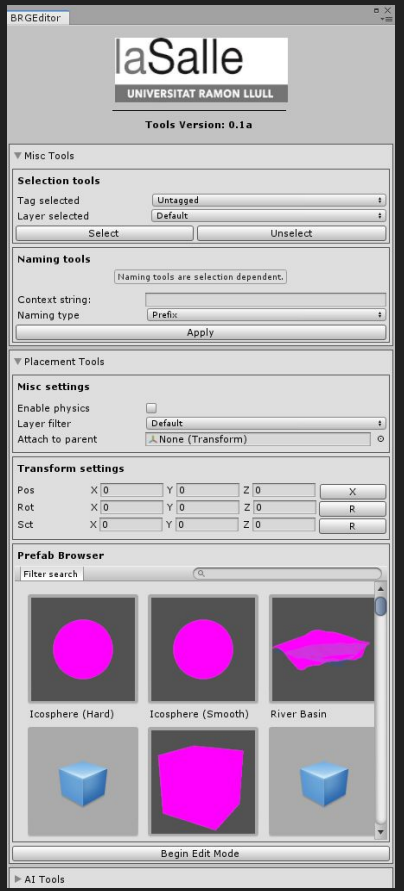
Unity Tools: Examples

- Exercise 1: Projectile Tool (Player shooting)
  - Projectile Editor
  - Launcher Editor
  - Debug launcher on player.
- Exercise 2: Raycast placing tool
  - Define a new editor window
  - Configure raycast tool to place prefabs
- Exercise 3: ...

Unity Tools: Raycast Placing Tool

- The raycast placing tool should be able to:
  - Place gameobjects in the scene raycasting through the scene
  - Raycast filtering by layer (hit only the given selected layer)
  - Gameobject placement transform should be able to be modified.
  - Gameobject placed must be a prefab from the prefab browser.
  - Gameobject placed must be attached to a given parent.
  - Labelfield to set prefab folder path.

- Bonus: Place decals with the tool instead of prefabs



Unity Tools: General Utilities

- Miscelaneous Exercise: Utility tools
  - Select gameobjects by tag
  - Select gameobjects by layer
  - Rename gameobjects selected
  - Component mass creator/update
    - Select component type
    - Edit component properties (DefualtDrawInspector)
  - Prefab randomizer generator
    - Randomize enemy generation

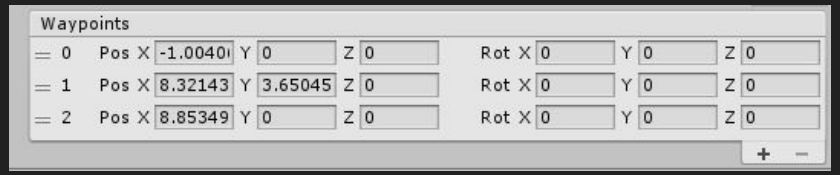
- MVD Tools window
  - Window bundle of tools
  - Best way to distribute a pack of tools for your artists/team

Unity Tools: Examples

- Exercise 1: Projectile Tool (Player shooting)
  - Projectile Editor
  - Launcher Editor
  - Debug launcher on player.
- Exercise 2: Raycast placing tool
  - Define a new editor window
  - Configure raycast tool to place prefabs
- Exercise 3: Spline Tool
  - Build waypoints tool
  - Editable waypoints and customize
  - Connect to the AI

Unity Tools: Editor scripting 3

- ReorderableList: Used to display hierarchical data
- Based on list data structure, highly customizable
  - Nodes**:** List of elements that can be added to the list. Order can be altered
  - Callback based: List of elements is displayed by callbacks
    - They follow the UI principles as other standard UI elements
    - Must be added before any UI function is called.
- Objects linked to this list structure must be Scriptable Objects
  - Data is serialized, and not lost on scene reload.



Unity Tools: Spline Tool

- Spline tool, used to move elements along splines
- Splines are dynamic and new points can be added in real time
  - Points order can be altered.
  - Points transform can be altered (swap between modes).
  - The spline must be seen and selectable at the scene view.

- Finish the tool by adding movement to the elements in the spline
  - Elements must loop (back to the start point when they reach the end point)

Exercises:

- Finish the projectile tool.
  - Add damage when projectile hits the ground.
  - Add any other necessary control elements.
- Finish utility tools
  - Add and finish the prefab tile placing tool in the mvd tools window
  - Finish the material replace tool
  - Finish layer/tag selector and rename tools
- Finish the raycasting placing tool.
  - Apply Transform settings
  - Apply layer filter to the placing tool
  - Prefab folder selector (load only prefabs from the given folder)
- Finish the play save mode for prefabs
- Finish the spline tool
  - Add movable items to the spline
  - Items should move along the spline and loop back.

Deliver

- Description: Code must be fully functional without errors
- Distribution: Package for Unity 2018.3
- Date: To be determined.

Unity Tools: (Bonus) Node editor

```
using UnityEngine;
using UnityEditor;

public class NodeEditor: EditorWindow {

    Rect window1;
    Rect window2;

    [MenuItem("Window/Node editor")]
    static void ShowEditor() {
        NodeEditor editor = EditorWindow.GetWindow<NodeEditor>();
        editor.Init();
    }

    public void Init() {
        window1 = new Rect(10, 10, 100, 100);
        window2 = new Rect(210, 210, 100, 100);
    }

    void OnGUI() {
        DrawNodeCurve(window1, window2); // Here the curve is drawn under the windows

        BeginWindows();
        window1 = GUI.Window(1, window1, DrawNodeWindow, "Window 1");   // Updates the Rect's
        window2 = GUI.Window(2, window2, DrawNodeWindow, "Window 2");
        EndWindows();
    }
```

```
    void DrawNodeWindow(int id) {
        GUI.DragWindow();
    }

    void DrawNodeCurve(Rect start, Rect end) {
        Vector3 startPos = new Vector3(start.x + start.width, start.y + start.height / 2, 0);
        Vector3 endPos = new Vector3(end.x, end.y + end.height / 2, 0);
        Vector3 startTan = startPos + Vector3.right * 50;
        Vector3 endTan = endPos + Vector3.left * 50;
        Color shadowCol = new Color(0, 0, 0, 0.06f);
        for (int i = 0; i < 3; i++) // Draw a shadow
            Handles.DrawBezier(startPos, endPos, startTan, endTan, shadowCol, null, (i + 1) * 5);
        Handles.DrawBezier(startPos, endPos, startTan, endTan, Color.black, null, 1);
    }
}
```

Resources

- https://unity3d.com/es/learn/tutorials/topics/interface-essentials/building-custom-inspector?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/interface-essentials/drawdefaultinspector-function?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/interface-essentials/adding-buttons-custom-inspector?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/interface-essentials/unity-editor-extensions-menu-items?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/scripting/introduction-editor-scripting?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/scripting/creating-spline-tool?playlist=17117
- https://unity3d.com/es/learn/tutorials/topics/scripting/getting-started-ik?playlist=17117