Model Audit Tool - Documentation

FR : Model Audit Tool est un outil Unity Editor permettant de scanner tous les prefabs 3D de votre projet
et d'en afficher les métriques agrégées : nombre de sommets, triangles, matériaux, LOD, colliders, taille, etc.

Il est conçu pour les artistes techniques, développeurs et producteurs afin de visualiser rapidement
la complexité et la structure des modèles utilisés dans la production du jeu.

EN : Model Audit Tool is a Unity Editor window that scans all 3D prefabs in your project
and displays aggregated metrics: vertex count, triangle count, materials, LODs, colliders, size, etc.

It is designed for technical artists, developers and producers to quickly visualize the complexity
and structure of the models used in game production.

Sommaire / Table of Contents

1. Vue d'ensemble / Overview

Model Audit Tool est un outil Unity Editor permettant de scanner tous les prefabs 3D de votre projet
et d'en afficher les métriques agrégées : nombre de sommets, triangles, matériaux, LOD, colliders, taille, etc.

Il est conçu pour les artistes techniques, développeurs et producteurs afin de visualiser rapidement
la complexité et la structure des modèles utilisés dans la production du jeu.

Model Audit Tool is a Unity Editor window that scans all 3D prefabs in your project
and displays aggregated metrics: vertex count, triangle count, materials, LODs, colliders, size, etc.

It is designed for technical artists, developers and producers to quickly visualize the complexity
and structure of the models used in game production.

2. Fonctionnalités / Features

FR:

- Scan de tous les prefabs 3D
- Affichage des sommets, triangles, matériaux, LOD, colliders, etc.
- Tri, recherche, colonnes personnalisables
- Bouton d'aide intégré

EN:

- Scan all 3D prefabs
- Show vertex, triangle, material count, LODs, colliders, etc.
- Sort, search, toggle columns
- Built-in Help button

3. Utilisation / Usage

FR:

- Menu : Tools > Model Audit
- Bouton 'Scan All Prefabs' pour lancer l'analyse
- Cliquez sur un nom pour ouvrir le prefab
- Tri par dropdown, filtrage, colonnes activables

EN:

- Menu: Tools > Model Audit
- 'Scan All Prefabs' button to launch analysis
- Click a name to ping the prefab
- Dropdown sorting, filtering, toggleable columns

4. Script complet commente / Full commented script

```
using UnityEditor;
using UnityEngine;
using UnityEngine.Rendering;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using UnityEditor.AddressableAssets;

namespace JulienNoe.Tools.ModelAuditTool
{
    /// <summary>
    /// Editor window that audits all 3D model prefabs and displays aggregated metrics
    /// (vertices, triangles, materials, LOD levels, colliders, etc.) per prefab.
    /// </summary>
    public class ModelAuditTool : EditorWindow
    {
        // --- UI State ---
        private Vector2 scrollPos;
        private Vector2 toggleScroll;
        private string searchString = string.Empty;
        private int sortOption = 0;
        private readonly string[] sortOptions = {
            "Name", "Vertices", "Triangles", "Materials", "LOD Levels",
            "Collider", "Static", "Addressable"
        };

        // Column toggles (persisted via EditorPrefs)
        private bool showVertices    = true;
        private bool showTriangles   = true;
        private bool showMaterials   = true;
        private bool showLOD         = true;
        private bool showCollider    = true;
        private bool showStatic      = true;
        private bool showAddressable = true;
        private bool showFileSize    = true;
        private bool showMemory      = true;
        private bool showPath        = true;
```

```
// Scan results
private List<ModelData> results = new List<ModelData>();

[MenuItem("Tools/Model Audit")]
public static void ShowWindow() =>
    GetWindow<ModelAuditTool>("Model Audit");

private void OnGUI()
{
    DrawHeader();
    DrawControls();
    if (GUILayout.Button("Scan All Prefabs", GUILayout.Height(30)))
        ScanPrefabs();
    if (results.Count > 0)
        DrawResultsTable();
}

private void DrawHeader()
{
    EditorGUILayout.BeginHorizontal();
    EditorGUILayout.LabelField("Model Audit Tool (Aggregated by Prefab)", EditorStyles.boldLabel);
    var prevColor = GUI.backgroundColor;
    GUI.backgroundColor = Color.cyan;
    if (GUILayout.Button("Help ?", GUILayout.Width(60)))
        ShowHelpPopup();
    GUI.backgroundColor = prevColor;
    EditorGUILayout.EndHorizontal();
}

private void DrawControls()
{
    EditorGUILayout.BeginHorizontal("box");
    searchString = EditorGUILayout.TextField(searchString, GUILayout.ExpandWidth(true));
    EditorGUILayout.EndHorizontal();

    sortOption = EditorGUILayout.Popup("Sort by", sortOption, sortOptions);

    toggleScroll = EditorGUILayout.BeginScrollView(toggleScroll, false, false,
        GUILayout.Height(EditorGUIUtility.singleLineHeight + 6));
    EditorGUILayout.BeginHorizontal("box");
```

```
    showVertices    = GUILayout.Toggle(showVertices,    "Vertices");
    showTriangles   = GUILayout.Toggle(showTriangles,   "Triangles");
    showMaterials   = GUILayout.Toggle(showMaterials,   "Materials");
    showLOD         = GUILayout.Toggle(showLOD,         "LOD Levels");
    showCollider    = GUILayout.Toggle(showCollider,    "Collider");
    showStatic      = GUILayout.Toggle(showStatic,      "Static");
    showAddressable = GUILayout.Toggle(showAddressable, "Addressable");
    showFileSize    = GUILayout.Toggle(showFileSize,    "File Size");
    showMemory      = GUILayout.Toggle(showMemory,      "Memory");
    showPath        = GUILayout.Toggle(showPath,        "Path");
    EditorGUILayout.EndHorizontal();
    EditorGUILayout.EndScrollView();
}

private void ScanPrefabs()
{
    results.Clear();
    var guids = AssetDatabase.FindAssets("t:Prefab", new[] { "Assets" });
    foreach (var guid in guids)
    {
        var path = AssetDatabase.GUIDToAssetPath(guid);
        var prefab = AssetDatabase.LoadAssetAtPath<GameObject>(path);
        if (prefab == null) continue;

        // Aggregate metrics
        var meshFilters = prefab.GetComponentsInChildren<MeshFilter>(true)
            .Where(m => m.sharedMesh != null);
        var skinned = prefab.GetComponentsInChildren<SkinnedMeshRenderer>(true)
            .Where(s => s.sharedMesh != null);

        int totalVerts = meshFilters.Sum(m => m.sharedMesh.vertexCount)
                       + skinned.Sum(s => s.sharedMesh.vertexCount);
        int totalTris  = meshFilters.Sum(m => m.sharedMesh.triangles.Length / 3)
                       + skinned.Sum(s => s.sharedMesh.triangles.Length / 3);

        var data = new ModelData
        {
            prefabName  = prefab.name,
            path        = path,
            vertices    = totalVerts,
```

```
            triangles   = totalTris,
            materials   = GetMaterialCount(prefab),
            lodLevels   = GetLODCount(prefab),
            hasCollider = HasCollider(prefab),
            // Check static flags: any static flag means true
            isStatic    = GameObjectUtility.GetStaticEditorFlags(prefab) != 0,
            addressable = AddressableAssetSettingsDefaultObject.Settings?
                          .FindAssetEntry(guid) != null,
            fileSize    = new FileInfo(Path.GetFullPath(path)).Length,
            memory      = EstimateMemory(totalVerts)
        };
        results.Add(data);
    }
}

private int GetMaterialCount(GameObject prefab)
{
    var allMaterials = prefab.GetComponentsInChildren<Renderer>(true)
        .SelectMany(r => r.sharedMaterials)
        .Where(mat => mat != null);
    return allMaterials.Distinct().Count();
}

private int GetLODCount(GameObject prefab)
{
    var group = prefab.GetComponentInChildren<LODGroup>(true);
    return group != null ? group.GetLODs().Length : 0;
}

private bool HasCollider(GameObject prefab)
{
    return prefab.GetComponentInChildren<Collider>(true) != null;
}

private long EstimateMemory(int totalVertices)
{
    return totalVertices * 12L;
}

private void DrawResultsTable()
```

```
{
    scrollPos = EditorGUILayout.BeginScrollView(scrollPos);
    var filtered = results
        .Where(d => string.IsNullOrEmpty(searchString)
                || d.prefabName.IndexOf(searchString,
                    System.StringComparison.OrdinalIgnoreCase) >= 0)
        .ToList();

    // Apply sorting
    switch (sortOption)
    {
        case 1: filtered = filtered.OrderByDescending(d => d.vertices).ToList(); break;
        case 2: filtered = filtered.OrderByDescending(d => d.triangles).ToList(); break;
        case 3: filtered = filtered.OrderByDescending(d => d.materials).ToList(); break;
        case 4: filtered = filtered.OrderByDescending(d => d.lodLevels).ToList(); break;
        case 5: filtered = filtered.OrderByDescending(d => d.hasCollider).ToList(); break;
        case 6: filtered = filtered.OrderByDescending(d => d.isStatic).ToList(); break;
        case 7: filtered = filtered.OrderByDescending(d => d.addressable).ToList(); break;
        default: filtered = filtered.OrderBy(d => d.prefabName).ToList(); break;
    }

    // Header row
    EditorGUILayout.BeginHorizontal();
    EditorGUILayout.LabelField("Prefab", GUILayout.Width(150));
    if (showVertices)    EditorGUILayout.LabelField("Verts",    GUILayout.Width(60));
    if (showTriangles)   EditorGUILayout.LabelField("Tris",     GUILayout.Width(60));
    if (showMaterials)   EditorGUILayout.LabelField("Mats",     GUILayout.Width(60));
    if (showLOD)         EditorGUILayout.LabelField("LOD",      GUILayout.Width(60));
    if (showCollider)    EditorGUILayout.LabelField("Coll",     GUILayout.Width(60));
    if (showStatic)      EditorGUILayout.LabelField("Static",   GUILayout.Width(60));
    if (showAddressable) EditorGUILayout.LabelField("Addr.",    GUILayout.Width(60));
    if (showFileSize)    EditorGUILayout.LabelField("Size",     GUILayout.Width(80));
    if (showMemory)      EditorGUILayout.LabelField("Mem",      GUILayout.Width(80));
    if (showPath)        EditorGUILayout.LabelField(
                                TruncatePath(d.path, 50), GUILayout.Width(300));
    EditorGUILayout.EndHorizontal();

    // Data rows
    foreach (var d in filtered)
    {
```

```
        EditorGUILayout.BeginHorizontal();
        if (GUILayout.Button(d.prefabName, GUILayout.Width(150)))
            EditorGUIUtility.PingObject(
                AssetDatabase.LoadAssetAtPath<UnityEngine.Object>(d.path));
        if (showVertices)    EditorGUILayout.LabelField(d.vertices.ToString(),    GUILayout.Width(60));
        if (showTriangles)   EditorGUILayout.LabelField(d.triangles.ToString(),   GUILayout.Width(60));
        if (showMaterials)   EditorGUILayout.LabelField(d.materials.ToString(),   GUILayout.Width(60));
        if (showLOD)         EditorGUILayout.LabelField(d.lodLevels.ToString(),   GUILayout.Width(60));
        if (showCollider)    EditorGUILayout.Toggle(d.hasCollider,                GUILayout.Width(60));
        if (showStatic)      EditorGUILayout.Toggle(d.isStatic,                   GUILayout.Width(60));
        if (showAddressable) EditorGUILayout.Toggle(d.addressable,                GUILayout.Width(60));
        if (showFileSize)    EditorGUILayout.LabelField($"{d.fileSize/1024f:F2} KB",  GUILayout.Width(80));
        if (showMemory)      EditorGUILayout.LabelField($"{d.memory/1024f:F1} KB", GUILayout.Width(80));
        if (showPath)        EditorGUILayout.LabelField(
                                 TruncatePath(d.path, 50), GUILayout.Width(300));
        EditorGUILayout.EndHorizontal();
    }
    EditorGUILayout.EndScrollView();
}

private void ShowHelpPopup()
{
    EditorUtility.DisplayDialog(
        "Model Audit Tool Help",
        "? Click **Scan All Prefabs** to gather aggregated model metrics.\n" +
        "? Filter by name, sort and toggle columns.\n" +
        "? Click on a prefab name to ping it in the Project window.",
        "OK"
    );
}

private string TruncatePath(string path, int maxLength)
{
    if (path.Length <= maxLength)
        return path;
    int part = (maxLength - 3) / 2;
    return path.Substring(0, part) + "..." + path.Substring(path.Length - part);
}

private class ModelData
```

```
        {
            public string prefabName, path;
            public int vertices, triangles, materials, lodLevels;
            public bool hasCollider, isStatic, addressable;
            public long fileSize, memory;
        }
    }
}
```