

Documentação

\_\_\_\_\_

**Criadores**

**-** Lucas Gomes Cecchini

- Gabriel Henrique Pereira

**Visão Geral**

Este documento o ajudará a usar o Assets **Unity MMD Collection**.

Com ele você tem ferramentas que podem facilitar o uso de modelos **MMD** na **Unity**, além de **Shaders** customizados para isso.

Os **Shaders** foram feitos com base em estudos no pacote [MMD4Mecanim](https://stereoarts.jp/), sendo recriados e aprimorados em duas ferramentas. O [Shader Graph](https://unity.com/pt/features/shader-graph) e [Amplify Shader Editor](https://assetstore.unity.com/packages/tools/visual-scripting/amplify-shader-editor-68570) devido às capacidades e aplicações de cada recurso.

Ele também possui **Scripts** que moldam a interface da Unity para fazê-lo ficar semelhante às ferramentas do **MMD** e tornar mais prático e rápido a utilização. Além de outras coisas que permitem acelerar o desenvolvimento.

**Instruções**

Você pode obter mais informações na Playlist no YouTube:

<https://youtube.com/playlist?list=PL5hnfx09yM4IWSWveW0NKCfX1Anec4dw7&si=UIqEtAoFcXZH7WFM>

**Compatibilidade**

Este pacote é compatível com:

**- Unity 6000.0.24f1**

**- Amplify Shader Editor v 1.9.7**

**- MMD4Mecanim 2020-01-05**

**Explicações do Shader**



|  |
| --- |
| **MMD Shader** |
| **Propriedades do Shader**  **==Propriedades de Sistemas==**  - **Show Default Systems** = Mostra às propriedades padrões ou customizadas.  - **Mat-Name (JP)** = Nome do material em Japonês.  - **Mat-Name (EN)** = Nome do material em Inglês.  - **Memo** = Cabeçalho de texto para anotações.  **==Material Color==**  - **Diffuse** = Cor do material.  - **\_Color ("Diffuse", Color)**: Define a cor principal do material quando iluminado diretamente.  - **Specular** = Cor do reflexo da luz/brilho.  - **\_Specular ("Specular", Color)**: Define a cor dos reflexos especulares no material.  - **Ambient** = Afeta a cor e a iluminação do modelo, similar a um raycast.  - **\_Ambient ("Ambient", Color)**: Define a cor com luz ambiente, contribuindo para a iluminação geral.  - **Opaque** = Valor alfa (opacidade)  - **\_Opaque ("Opaque", Range(0, 1))**: Controla a opacidade do material.  - **Reflection** = Valor de reflexão  - **\_Shininess ("Reflection", Float)**: Define a intensidade do brilho especular (reflexão) do material.  **==Rendering==**  - **2-SIDE** = Renderiza ambos os lados da malha **[Equivalente a Render Face/\_Cull]**  - **\_Cull ("Render Face", Float)**: Controla quais faces da malha (frente, verso ou ambas) devem ser renderizadas.  - **G-SHAD** = Sombra no chão **[Equivalente a Cast Shadows/ShaderPass"SHADOWCASTER"]**  - **\_CastShadows ("Cast Shadows", Float)**: Define se o material projeta sombras no chão.  - **S-MAP** = Sombra na malha (incluindo a própria malha) **[Equivalente a Receive Shadows/\_ReceiveShadows/Keyword"\_RECEIVE\_SHADOWS\_OFF"]**  - **\_ReceiveShadows ("Receive Shadows", Keyword)**: Controla se a malha recebe sombras de outras fontes.  - **S-SHAD** = Recebe sombra apenas de si mesmo  - **\_SShad ("S-SHAD", Float)**: Controla a sombra que a malha projeta apenas sobre si mesma.  **==Edge (Outline)==**  - **On** = Ativa o contorno  - **\_On ("On", Float)**: Controla a ativação do contorno (borda).  - **Color** = Cor do contorno, incluindo transparências  - **\_OutlineColor ("Color", Color)**: Define a cor do contorno ao redor da malha.  - **Size** = Tamanho/distância do contorno  - **\_EdgeSize ("Size", Float)**: Controla a espessura do contorno.  **==Texture/Memo==**  - **Texture** = Textura do material  - **\_MainTex ("Texture", 2D)**: Textura principal aplicada ao material.  - **Toon** = Complementa o shading do objeto  - **\_ToonTex ("Toon", 2D)**: Textura toon, usada para shading estilo cartoon.  - **UV Layer** = Camada de UV a ser usada  - **\_UVLayer ("UV Layer", Float)**: Seleciona qual camada de UV será aplicada à textura.  - **SPH** = Reflexão artificial para complementar o Specular  - **\_SphereCube ("SPH", CUBE)**: Textura cúbica usada para reflexões artificiais.  **==Effects==**  - **Disabled** = Não faz nada  - **\_EFFECTS ("Effects", Float)**: Desativa os efeitos.  - **Multi-Sphere** = Multiplica um mapa de esfera brilhante (reflexão metálica)  - **\_EFFECTS ("Effects", Float)**: Ativa o efeito de múltiplas esferas brilhantes.  - **Add-Sphere** = Cria um mapa de esfera brilhante (reflexão prática)  - **\_EFFECTS ("Effects", Float)**: Ativa o efeito de adição de esferas brilhantes.  - **Sub-Tex** = Adiciona uma camada de textura UV extra para efeitos mais complexos  - **\_EFFECTS ("Effects", Float)**: Troca o uso da textura cúbica para usar uma subtextura, aplicando-a a outra camada UV.  **==Configurações de Efeitos Personalizados==**  - **Specular Intensity** = Define a intensidade do brilho especular  - **SpecularIntensity ("Specular Intensity", Range(0, 1))**: Intensidade do brilho especular.  - **SPH Opacity** = Define a opacidade da Textura cúbica  - **SPHOpacity ("SPH Opacity", Range(0, 1))**: Opacidade da textura cúbica.  - **Shadow Luminescence** = Intensidade da sombra  - **\_ShadowLum ("Shadow Luminescence", Range(0, 10))**: Luminescência das sombras.  - **HDR** = Faz o objeto brilhar no escuro  - **\_HDR ("HDR", Range(1, 1000))**: Controla o mapeamento HDR.  - **Toon Tone** = Ajusta o tom da sombra Toon  - **\_ToonTone ("Toon Tone", Vector)**: Define o tom do shading toon.  - **Multiple Lights** = Permite o objeto receber múltiplas luzes  - **\_MultipleLights ("Multiple Lights", Float)**: Ativa ou desativa o suporte para múltiplas luzes.  - **Fog** = Liga e desliga a neblina  - **\_Fog ("Fog", Float)**: Ativa ou desativa o suporte para neblina.  O **"Surface Options"** e **"Advanced Options"** são padrões da Unity que exigem um conhecimento prévio de suas funcionalidades. |
| **Bugs conhecidos**  **Decal não aparece na Build:** Para resolver isso, adicione o módulo de Decal a todos os 'Universal Renderer Data' no seu projeto.  **Sombras distorcidas ao usar Morph:** Para resolver isso, habilite a opção 'Legacy Blend Shape Normals' no **.FBX** (ou modelo 3D) ao importar do Unity.  **Luz atravessando objetos:** Adicione um 'Directional Light' com pelo menos 0,001 de Intensity para atualizar o Dynamic Lightmap |

**Explicações dos Scripts**

|  |
| --- |
| **Free Camera** |
| Este script define uma classe **`FreeCamera`** em C# para ser usada em um projeto Unity. Ele permite que a câmera se mova livremente no espaço 3D, respondendo a entradas do teclado e do mouse. A seguir está uma explicação detalhada do script:  **Declarações e Atributos da Classe**  **1. Dependências e Configurações Iniciais**:    - **`RequireComponent(typeof(Camera))`**: Garante que o objeto ao qual este script é anexado tenha um componente **`Camera`**.  - **`AddComponentMenu("MMD Collection/Free Camera")`:** Adiciona este script ao menu de componentes, sob o caminho especificado.  **2. Variáveis de Configuração da Câmera:**    - **`movementSpeed`**: Velocidade padrão de movimento da câmera.  - **`fastMovementSpeed`**: Velocidade aumentada de movimento quando o modo rápido está ativado.  - `**sensitivity`**: Sensibilidade do mouse para rotação.  - **`zoomSensitivity`** e **`fastZoomSensitivity`**: Sensibilidade para o zoom (normal e rápido).  **3. Configurações de Teclas:**    - Define teclas para mover a câmera em diferentes direções e alternar modos.  **4. Configurações de Eixos:**    - Define os eixos do mouse para zoom e rotação.  **5. Variável Privada:**    - **`looking`**: Indica se a câmera está em modo de observação (rotação livre com o mouse).  **Método `Update`**  **6. Movimento da Câmera:**    - Verifica se o modo rápido está ativado e ajusta a velocidade de movimento.  - Calcula a direção do movimento baseado nas teclas pressionadas.  - Atualiza a posição da câmera.  **7. Rotação da Câmera:**    - Ajusta a rotação da câmera com base no movimento do mouse se o modo de observação estiver ativo.  **8. Zoom da Câmera:**    - Ajusta o zoom da câmera baseado na rolagem do mouse.  **9. Modo de Observação:**    **Métodos Auxiliares**  **10. Método `OnDisable`:**    - Garante que o modo de observação seja desativado quando o script é desabilitado.  **11. Métodos para Iniciar e Parar o Modo de Observação:**    - **`StartLooking`**: Ativa o modo de observação, escondendo e travando o cursor.  - `**StopLooking`**: Desativa o modo de observação, mostrando e liberando o cursor.  **Resumo**  Este script fornece um controle detalhado e personalizável para movimentação e rotação de uma câmera em um ambiente 3D, usando entradas do teclado e do mouse. Ele permite um movimento suave e intuitivo, semelhante ao de um jogo FPS, e inclui funcionalidades para acelerar o movimento e o zoom, bem como alternar entre modos de observação e movimento normal. |

|  |
| --- |
| **Copy Animation** |
| Este script Unity, chamado CopyAnimation, é um componente responsável por copiar rotações de um conjunto de ossos fonte para um conjunto de ossos alvo em tempo real. Abaixo está uma explicação detalhada de cada parte do script:  **Definições de Componentes e Variáveis**  **1. Imports e Atributos de Classe**    - **`using UnityEngine;`**: Importa o namespace UnityEngine, necessário para usar as classes e métodos do Unity.  - **`[AddComponentMenu("MMD Collection/Copy Animation")]`**: Adiciona este script ao menu de componentes do Unity sob a categoria "MMD Collection".  **2. Variáveis Públicas e Privadas**    - **`public bool update`**: Controla se as rotações devem ser atualizadas ou não.  - **`[SerializeField] private Transform[] copyBone`**: Array de ossos fonte dos quais as rotações serão copiadas.  - **`[SerializeField] private Transform[] pasteBone`**: Array de ossos alvo para onde as rotações serão coladas.  **Método `LateUpdate`**  **3. Método `LateUpdate`**    - **`if (!update) return;`**: Se **`update`** for **`false`**, o método retorna e não faz nada.  - Verifica se os arrays **`copyBone`** e **`pasteBone`** têm o mesmo comprimento. Se não, exibe um erro no console e retorna.  - Itera por cada par de ossos, copiando a rotação do osso fonte para o osso alvo.  **Considerações Finais**  - Este script é útil em animações, especialmente em sistemas de rigging e animações esqueléticas, onde a rotação de certos ossos precisa ser sincronizada ou limitada.  - O uso de arrays para **`copyBone`** e **`pasteBone`** permite a cópia de múltiplos ossos de uma só vez, garantindo que a hierarquia esquelética mantenha as rotações corretas. |

|  |
| --- |
| **Draw Mesh Instanced** |
| Este script é um componente do Unity que desenha instâncias de malhas (meshes) usando a técnica de instanciamento gráfico.  **Imports e Declaração da Classe**    - O script começa importando namespaces necessários: **`System.Collections.Generic`** para manipulação de listas e **`UnityEngine`** para acesso à API do Unity.  - A classe **`DrawMeshInstanced`** herda de **`MonoBehaviour`**, permitindo que seja anexada a um GameObject no Unity.  - O atributo **`[AddComponentMenu("MMD Collection/Draw Mesh Instanced")]`** adiciona essa classe ao menu de componentes no editor do Unity.  **Variáveis de Configuração**    - **`OnDrawSelected`**: Se verdadeiro, desenha as instâncias da malha apenas quando o objeto está selecionado no editor.  - **`reuseMaterials`**: Se verdadeiro, reutiliza os materiais para submeshes.  - **`drawMeshInstancedLists`**: Uma lista de parâmetros para instanciar as malhas.  **Método `Update`**    - O método **`Update`** é chamado a cada frame. Ele chama o método **`DrawMesh`** para desenhar as instâncias da malha.  **Método `DrawMesh`**    - Este método itera através de **`drawMeshInstancedLists`**.  - Para cada item na lista, verifica se a malha e os materiais são válidos.  - Para cada submalha (**`subMesh`**), seleciona o material apropriado.  - Se **`reuseMaterials`** for falso e todos os materiais já tiverem sido usados, a iteração é interrompida.  - Cria uma matriz de transformação e chama **`Graphics.DrawMeshInstanced`** para desenhar a instância da malha.  **Método `OnDrawGizmosSelected`**    - Este método é chamado quando o objeto é selecionado no editor do Unity.  - Se **`OnDrawSelected`** for verdadeiro, chama **`DrawMesh`**.  - Desenha esferas verdes na posição do objeto e esferas vermelhas nas posições dos **`Transform`** de `drawMeshInstancedLists`.  **Classe `DrawMeshInstancedList`**    - Esta classe armazena os parâmetros necessários para instanciar uma malha.  - **`transform`**: Define a posição, rotação e escala das instâncias.  - **`mesh`**: A malha a ser instanciada.  - **`materials`**: Os materiais aplicados às instâncias. Os materiais devem suportar o recurso "Enable GPU Instancing".  **Resumo**  O script **`DrawMeshInstanced`** permite desenhar múltiplas instâncias de uma malha com diferentes materiais e posições definidas por transformações. Ele também fornece a funcionalidade de desenhar essas instâncias apenas quando o objeto é selecionado no editor do Unity. A classe auxiliar **`DrawMeshInstancedList`** facilita a configuração desses parâmetros no editor. |

|  |
| --- |
| **Custom MMD Data** |
| **Visão Geral**  Ele define um **`ScriptableObject`** chamado **`CustomMMDData`** para armazenar dados personalizados relacionados ao MMD (MikuMikuDance), que é um software de animação 3D popular no Japão.  **Detalhamento do Código**  **Namespace e Bibliotecas**    - **`using System.Collections.Generic;`**: Importa a biblioteca para utilizar coleções genéricas, como listas.  - **`using UnityEngine;`**: Importa a biblioteca principal do Unity que contém funcionalidades essenciais para o desenvolvimento de jogos.  **Definição do ScriptableObject**    - **`public class CustomMMDData : ScriptableObject`**: Define uma classe que herda de **`ScriptableObject`**. **`ScriptableObject`** é uma maneira conveniente de armazenar grandes conjuntos de dados que podem ser facilmente editados no editor do Unity e usados em diferentes partes de um jogo.  - **`[Header("MMD Material Settings")]`**: Adiciona um cabeçalho no Inspetor do Unity para organizar e identificar melhor os campos.  - **`public bool showSystemsDefault;`**: Um campo booleano que indica se deve mostrar mais sistemas de shaders (padrões do sistema).  - **`public List<MMDMaterialInfo> materialInfoList = new();`**: Uma lista para armazenar informações sobre materiais MMD. A lista é inicializada como uma nova lista vazia.  **Classe Serializable**    - **`[System.Serializable]`**: Permite que a classe seja serializável, ou seja, seus dados podem ser exibidos e editados no Inspetor do Unity.  - **`public class MMDMaterialInfo`**: Define uma classe para armazenar informações sobre um material MMD específico.  - **`public Material mmdMaterial;`**: Referência a um material do Unity.  - **`public string materialNameJP;`**: Nome do material em japonês.  - **`public string materialNameEN;`**: Nome do material em inglês.  - **`public string materialMeno;`**: Campo para notas adicionais sobre o material.  **Funcionamento Geral**  **1. CustomMMDData:**  - É um **`ScriptableObject`** que armazena uma lista de objetos **`MMDMaterialInfo`** e uma flag booleana (**`showSystemsDefault`**).  - Pode ser usado para organizar e gerenciar dados de materiais MMD no Unity.  **2. MMDMaterialInfo:**  - Armazena detalhes sobre um material específico, incluindo referências ao material no Unity e seus nomes em japonês e inglês, além de um campo para notas.  **Utilização no Unity**  - Esse **`ScriptableObject`** pode ser criado e editado diretamente no editor do Unity, permitindo fácil gerenciamento e organização dos dados de materiais MMD.  - Os desenvolvedores podem adicionar, remover e modificar entradas na lista **`materialInfoList`** através do Inspetor do Unity.  - A flag **`showSystemsDefault`** pode ser usada para alternar configurações específicas de exibição ou comportamento dentro do jogo ou ferramenta. |

|  |
| --- |
| **Custom MMD Data Utility Editor** |
| Esse script em C# é usado no Unity Editor para gerenciar assets do tipo **`CustomMMDData`**.  **Visão Geral**  Este script fornece uma classe utilitária estática chamada **`CustomMMDDataUtilityEditor`**, que inclui métodos para encontrar, criar e gerenciar assets **`CustomMMDData`** dentro do Unity Editor.  **Detalhamento do Código**  **Namespace e Bibliotecas**    - **`using System.Collections.Generic;`**: Importa a biblioteca para utilizar coleções genéricas, como listas.  - **`using System.IO;`**: Importa funcionalidades de entrada e saída, como manipulação de arquivos e diretórios.  - **`using UnityEditor;`**: Importa funcionalidades específicas do editor do Unity.  - **`using UnityEngine;`**: Importa a biblioteca principal do Unity.  **Classe Utilitária**    - **`public static class CustomMMDDataUtilityEditor`**: Define uma classe estática para métodos utilitários relacionados a **`CustomMMDData`**.  - **`public static CustomMMDData GetOrCreateCustomMMDData()`**: Método que recupera ou cria um asset **`CustomMMDData`**.  - Tenta encontrar um asset **`CustomMMDData`** existente chamando **`FindCustomMMDData()`**.  - Se não encontrar, cria um novo chamando **`CreateCustomMMDData()`**.  **Método para Encontrar um Asset Existente**    -**`private static CustomMMDData FindCustomMMDData()`**: Método que encontra um asset **`CustomMMDData`** existente no projeto.  - Utiliza **`AssetDatabase.FindAssets`** para procurar por assets do tipo **`CustomMMDData`**.  - Se encontrar, carrega o primeiro asset encontrado usando **`AssetDatabase.LoadAssetAtPath`**.  **Método para Criar um Novo Asset**    - **`private static CustomMMDData CreateCustomMMDData()`**: Método que cria um novo asset **`CustomMMDData`**.  - Define o caminho da pasta onde o asset será armazenado.  - Verifica se a pasta existe e, se não existir, cria a pasta.  - Gera um caminho único para o asset e cria uma nova instância de `**CustomMMDData`**.  - Cria o asset no caminho especificado, salva e atualiza o banco de dados de assets.  **Método para Remover Materiais Inválidos**    - **`public static void RemoveInvalidMaterials(CustomMMDData customMMDMaterialData)`**: Método que remove materiais inválidos do asset **`CustomMMDData`**.  - Verifica se o asset é nulo e sai do método se for.  - Cria uma nova lista para armazenar materiais válidos.  - Itera sobre cada **`MMDMaterialInfo`** na lista **`materialInfoList`** e adiciona à lista de materiais válidos apenas se a referência ao material não for nula.  - Substitui a lista **`materialInfoList`** com a lista de materiais válidos.  **Funcionamento Geral**  **1. Criação ou Recuperação de `CustomMMDData`:**  - O método **`GetOrCreateCustomMMDData`** tenta encontrar um asset **`CustomMMDData`** existente.  - Se não encontrar, cria um novo asset.  **2. Encontrar `CustomMMDData`:**  - O método **`FindCustomMMDData`** procura por assets do tipo **`CustomMMDData`** no projeto.  - Se encontrar, carrega e retorna o primeiro asset encontrado.  **3. Criar Novo `CustomMMDData`:**  - O método **`CreateCustomMMDData`** cria um novo asset **`CustomMMDData`** em uma pasta específica.  - Se a pasta não existir, cria a pasta e então cria o asset.  **4. Remover Materiais Inválidos:**  - O método **`RemoveInvalidMaterials`** remove entradas na lista **`materialInfoList`** que possuem referências a materiais nulas.  Esse script é muito útil para garantir que os assets **`CustomMMDData`** estejam sempre válidos e facilmente acessíveis dentro do Unity Editor, facilitando o gerenciamento de dados de materiais MMD no projeto. |

|  |
| --- |
| **Custom Inspector Utility Editor** |
| **Explicação Detalhada do Script**  Este script é uma extensão personalizada para o editor de materiais no Unity, fornecendo funcionalidades avançadas para inspeção e modificação de materiais que utilizam shaders personalizados. Ele estende a classe **`ShaderGUI`**, permitindo controle detalhado sobre várias propriedades dos shaders e materiais diretamente no editor.  **Classe `CustomInspectorUtilityEditor`**  A classe **`CustomInspectorUtilityEditor`** herda de **`ShaderGUI`** e serve como a base para criar inspetores personalizados de shaders. Ela oferece várias funções estáticas e métodos para carregar, salvar e renderizar propriedades do material de forma dinâmica e interativa.  **Variáveis Principais**  - **`private Material currentMaterial`**: Armazena uma referência ao material atualmente sendo inspecionado.  - **`private CustomMMDMaterialData customMMDMaterialData`**: Um objeto que armazena dados específicos do material, como nomes e outros metadados.  - **`private bool showSystemsDefault`**: Um booleano que determina se as configurações padrão do sistema devem ser exibidas.  **Método `LoadData`**  Este método carrega os dados existentes do material atual (**`currentMaterial`**) e os armazena nas variáveis relevantes. Ele verifica se o objeto **`customMMDMaterialData`** existe e, em caso afirmativo, carrega informações como os nomes em japonês e inglês do material (**`materialNameJP`**, **`materialNameEN`**), a descrição (**`materialMemo`**), e a configuração sobre a exibição dos valores padrão do sistema (**`showSystemsDefault`**).  **Método `SaveData`**  Este método salva os dados do material atual. Ele utiliza o método **`DetectChanges`** para verificar se houve modificações nos dados do material. Caso haja mudanças, ele atualiza as informações e marca o objeto como "sujo" (**`SetDirty`**), indicando que há alterações a serem salvas. Finalmente, ele persiste essas mudanças no sistema de assets do Unity.  **Método `DetectChanges`**  Este método compara os valores atuais das propriedades do material com os valores armazenados anteriormente em **`customMMDMaterialData`**. Ele retorna **`true`** se houver alterações e **`false`** caso contrário, permitindo que o sistema saiba quando salvar as mudanças.  **Métodos de Renderização**  Estes métodos são responsáveis por renderizar as várias propriedades do material no inspetor do Unity. Eles permitem que o desenvolvedor visualize e modifique atributos específicos do shader diretamente na interface do Unity.  - **`RenderSurfaceOptions`**: Renderiza opções de superfície, como tipo de superfície, modo de blending, culling, entre outras.  - **`RenderLightmapFlags`**: Renderiza opções de lightmap, incluindo realtime, baked, e emissive.  - **`RenderColorProperty`**: Renderiza uma propriedade de cor com um rótulo associado.  - **`RenderSliderFloatProperty`**: Renderiza uma propriedade do tipo **`float`** como um slider.  - **`RenderDoubleSidedToggle`**: Renderiza um toggle para definir se a renderização será de dupla face.  - **`RenderShaderPassToggle`**: Renderiza um toggle para habilitar ou desabilitar um **`pass`** específico do shader.  - **`RenderKeywordToggle`**: Renderiza um toggle para ativar ou desativar uma **`keyword`** específica.  - **`RenderUIToggle`**: Renderiza um toggle para propriedades de UI.  - **`RenderUIColorProperty`**: Renderiza uma propriedade de cor específica para UI.  - **`RenderFloatProperty`**: Renderiza uma propriedade do tipo **`float`** com um rótulo.  - **`RenderVector4Property`**: Renderiza uma propriedade do tipo **`Vector4`**.  - **`RenderDropdownProperty`**: Renderiza uma propriedade do tipo dropdown, verificando se as opções e os valores correspondem. Caso contrário, lança uma exceção.  - **`RenderTextureProperty`**: Renderiza uma propriedade do tipo textura com um rótulo e exibe opcionalmente os campos de **Tiling** e **Offset**.  - **`RenderCubemapProperty`**: Renderiza uma propriedade do tipo cubemap com um rótulo.  - **`RenderVector3Property`**: Renderiza uma propriedade do tipo **`Vector3`**, ajustando cada componente separadamente.  - **`RenderDepthWriteDropdown`**: Renderiza um dropdown para opções de escrita de profundidade (Depth Write).  - **`RenderBlendingModeDropdown`**: Renderiza um dropdown para opções de modo de blending, atualizando automaticamente as propriedades de **`srcBlend`** e **`dstBlend`** com base na seleção.  - **`RenderSurfaceTypeDropdown`**: Renderiza um dropdown para opções de tipo de superfície (opaco ou transparente), configurando as tags de renderização do material adequadamente.  **Métodos Auxiliares**  - **`IsToggleUIPropertyEnabled`**: Verifica se uma propriedade de toggle de UI está habilitada, baseado no valor da propriedade de float (1.0 = habilitado).  - **`HasFloatPropertyValue`**: Verifica se uma propriedade **`float`** possui um valor específico, comparando diretamente o valor atual da propriedade.  - **`CheckBlendingMode`**: Verifica e ajusta o modo de blending com base nas propriedades atuais do material. Esta função garante que as propriedades de blending estejam configuradas corretamente, dependendo do modo de superfície.  **Conclusão**  Este script **`CustomInspectorUtilityEditor`** oferece uma interface robusta e intuitiva para personalizar e gerenciar materiais que utilizam shaders avançados no Unity. Ele permite que desenvolvedores e artistas tenham controle detalhado sobre como os materiais são renderizados, oferecendo uma experiência de edição rica e flexível diretamente no editor do Unity. Com ele, é possível configurar desde propriedades simples, como cores e texturas, até opções avançadas como blending e depth write. |

|  |
| --- |
| **MMD Material Custom Inspector (Amplify Shader Editor & Shader Graph)** |
| **Explicação Detalhada dos Scripts**  As duas classes, **`MMDMaterialCustomInspector\_AmplifyShaderEditor`** e **`MMDMaterialCustomInspector\_ShaderGraph`**, são scripts de inspetor customizado para materiais do MikuMikuDance (MMD) no Unity, utilizando o Amplify Shader Editor. Ele estende a interface do Unity Editor, permitindo que os usuários manipulem propriedades específicas dos materiais MMD com facilidade, oferecendo tanto opções padrão quanto customizadas.  **Principais Funcionalidades:**  **1. Variáveis de Controle:**  - Armazena nomes de materiais em japonês e inglês, além de um campo de memo para anotações.  - A variável **`showSystemsDefault`** controla a exibição do inspetor customizado ou do inspetor padrão do Unity.  - **`customMMDMaterialData`** é utilizado para armazenar e gerenciar dados específicos dos materiais MMD.  **2. Método `OnGUI`:**  - Inicializa o inspetor de materiais (**`materialInspector`**), propriedades do material (**`materialProperties`**), e o material atual sendo editado (**`currentMaterial`**).  - Apresenta uma opção de alternância para mostrar ou ocultar o inspetor padrão do Unity.  - Carrega os dados customizados do material MMD, se ainda não estiverem carregados.  - Se **`showSystemsDefault`** estiver desativado, renderiza o inspetor customizado, caso contrário, renderiza o inspetor padrão do Unity.  - Salva automaticamente as alterações feitas nos dados dos materiais MMD.  **3. Método `RenderCustomMaterialInspector`:**  - Oferece controles detalhados para o ajuste de propriedades dos materiais MMD, incluindo:  - Nomes de materiais (em japonês e inglês).  - Propriedades de cores como **`Diffuse`**, **`Specular`** e **`Ambient`**.  - Configurações de opacidade e reflexo.  - Opções de renderização avançadas (como faces duplas, sombreador de sombras, recepção de sombras).  - Configurações de borda (contorno), incluindo tamanho e cor.  - Configurações de texturas (principal, toon e efeitos esféricos como **`SPH`**).  - Um campo de memo para notas ou informações adicionais.  - Ajustes de efeitos customizados, como intensidade especular, opacidade de sombras, HDR, entre outros.  - Opções avançadas, incluindo fila de renderização, instanciação de GPU e iluminação global de face dupla.  Ambos os scripts utilizam o utilitário personalizado **`CustomMMDDataUtilityEditor`** para carregar, salvar e renderizar propriedades e dados customizados dos materiais MMD. Esse utilitário é essencial para manter o código organizado e reutilizável, além de facilitar a adição de novas funcionalidades no futuro.  Também tem uma variante do script chamado **`MMDTessellationMaterialCustomInspector`** para adicionar **`Edge Length`**, **`Phong Tess Strength`** e **`Extrusion Amount`** que são fusões de Tessellation que é uma técnica usada em shaders para subdividir polígonos de uma malha em triângulos menores em tempo de execução. Isso permite adicionar mais detalhes geométricos a objetos. |

|  |
| --- |
| **Create Prefab From Model** |
| Este script é uma ferramenta personalizada para o Unity Editor, projetada para criar prefabs a partir de modelos selecionados. Vamos detalhar cada parte do script:  **Importações e Definições Iniciais**    Essas linhas importam os namespaces necessários:  - **`UnityEngine`**: Para acessar as funcionalidades principais do Unity.  - **`UnityEditor`**: Para criar ferramentas personalizadas dentro do Unity Editor.  - **`System.Collections.Generic`**: Para usar listas genéricas.  - **`System.IO`**: Para manipular caminhos de arquivos e diretórios.  **Classe Principal**    Define uma classe chamada **`CreatePrefabFromModel`**, que herda de **`EditorWindow`**, permitindo criar janelas personalizadas no Unity Editor.  **Método Estático para Criar Prefabs**    Este método cria uma entrada de menu no Unity Editor sob "Assets/MMD Collection/Create Prefabs From Selected Model". Quando essa entrada é clicada, o método **`CreatePrefabsFromModel`** é executado.  **Seleção e Validação de Objetos**    - **`Selection.objects`**: Obtém os objetos selecionados no Unity Editor.  - **`List<GameObject> models`**: Lista para armazenar modelos válidos.  - Itera pelos objetos selecionados:  - Verifica se o objeto é um **`GameObject`** e se é um modelo (**`PrefabAssetType.Model`**).  - Obtém o caminho do ativo e carrega o modelo.  - Adiciona o modelo à lista de modelos válidos, se carregado com sucesso.  - Exibe um aviso se nenhum modelo válido for encontrado.  **Conversão de Modelos em Prefabs**    Para cada modelo válido, chama o método **`ModelConverter`**.  **Método `ModelConverter`**    - Cria uma lista para armazenar objetos criados.  - Obtém todos os componentes `**SkinnedMeshRenderer`** e **`MeshFilter`** do modelo.  - Se não houver componentes válidos, exibe um erro e retorna.  - Converte os **`SkinnedMeshRenderer`** e **`MeshFilter`** em objetos separados.  - Se objetos válidos forem criados, chama o método **`CreatePrefab`**.  **Conversão de `SkinnedMeshRenderer` e `MeshFilter`**    - **`SkinnedMeshRendererConverter`**: Converte componentes **`SkinnedMeshRenderer`** em novos objetos.  - **`MeshFilterConverter`**: Converte componentes **`MeshFilter`** em novos objetos, verificando se possuem um **`MeshRenderer`**.  **Criação de GameObject com Mesh e Materiais**    Cria um novo **`GameObject`** com componentes **`MeshFilter`** e **`MeshRenderer`**, atribuindo a ele a malha e os materiais fornecidos.  **Criação de Prefab**    - Verifica se há objetos válidos para criar o prefab.  - Se houver apenas um objeto, usa-o diretamente; caso contrário, cria um novo **`GameObject`** para agrupar todos os objetos criados.  - Obtém o caminho do diretório do modelo e constrói o caminho do prefab.  - Tenta salvar o objeto criado como um prefab.  - Exibe mensagens de log para sucesso ou erro na criação do prefab.  - Destrói o objeto criado após salvar o prefab para evitar resíduos na cena.  **Resumo**  Este script automatiza o processo de criação de prefabs a partir de modelos 3D selecionados no Unity Editor. Ele valida os modelos, converte componentes de malha em objetos separados e salva esses objetos como prefabs. |

|  |
| --- |
| **Paste As Child Multiple** |
| Esse script é um editor customizado para o Unity, que adiciona a funcionalidade de colar múltiplas instâncias de um prefab ou objeto selecionado como filhos de objetos selecionados na hierarquia. Vou explicar cada parte do script detalhadamente:  **Imports e Declarações**    Esses são os namespaces necessários:  - **`UnityEngine`** e **`UnityEditor`** são essenciais para qualquer script que interaja com o Unity e seu editor.  - **`System.Collections.Generic`** e **`System`** fornecem estruturas de dados e funções básicas.  **Classe Principal**    - **`PasteAsChildMultiple`** herda de **`EditorWindow`**, permitindo criar uma janela customizada no editor do Unity.  - **`objectToCopy`** armazena o objeto que será copiado.  - **`enumerate`** define se os novos objetos receberão nomes enumerados.  - **`newObjects`** mantém uma lista dos novos objetos criados.  **Inicialização da Janela**    - **`MenuItem`** define onde a nova opção de menu será exibida no editor do Unity.  - **`Init`** cria e exibe a janela customizada com título e dimensões fixas.  **Layout da Janela**    - **`OnGUI`** define a interface gráfica da janela.  - Cria um título com estilo negrito.  - Campo para selecionar o objeto a ser copiado.  - Toggle para definir se os novos objetos terão nomes enumerados.  - Botão "Paste As Child", que chama o método **`PasteAsChild`** ao ser pressionado.  **Método para Colar o Objeto**    - **`PasteAsChild`** é o método principal que lida com a colagem dos objetos.  - Verifica se há objetos selecionados e um objeto a ser copiado.  - Itera sobre os objetos selecionados, instanciando o prefab e configurando-o como filho de cada objeto selecionado.  - Ajusta o nome do novo objeto se **`enumerate`** estiver habilitado.  - Registra a operação de desfazer para cada novo objeto criado.  - Expande a hierarquia para mostrar os novos filhos.  - Seleciona os novos objetos e fecha a janela.  **Método para Expandir a Hierarquia**    - **`ExpandHierarchy`** usa reflexão para acessar e invocar métodos da **`SceneHierarchyWindow`**, expandindo a hierarquia para mostrar os novos objetos.  **Considerações Finais**  Este script oferece uma ferramenta útil para duplicar e organizar objetos na hierarquia do Unity, facilitando o processo de desenvolvimento e prototipagem. A interface é simples, mas funcional, permitindo que os desenvolvedores rapidamente instanciem múltiplos objetos e configurem suas propriedades de forma eficiente. |

|  |
| --- |
| **Material Property Cleaner** |
| Este script é um editor de Unity que limpa propriedades inválidas de materiais selecionados. Vou explicar cada parte em detalhes.  **Imports**    Esses são os namespaces importados:  - **`UnityEngine`**: A principal biblioteca do Unity.  - **`UnityEditor`**: Fornece funcionalidades específicas do editor do Unity.  - **`System.Collections.Generic`**: Permite o uso de coleções genéricas como **`HashSet`** e **`List`**.  **Declaração da Classe**    **`MaterialPropertyCleaner`** é uma classe pública que herda de **`MonoBehaviour`**, mas neste contexto, ela é usada principalmente para acessar funcionalidades do editor.  **Menu Item e Método Principal**    Este atributo adiciona um item de menu no editor do Unity. Quando clicado, ele chama o método **`CleanMaterialProperties`**.  **Janela de Confirmação**    Mostra uma janela de diálogo de confirmação ao usuário. Se o usuário clicar "No", o método é encerrado.  **Loop para Objetos Selecionados**    Para cada objeto selecionado no editor:  - Verifica se o objeto é um **`Material`**.  - Obtém o shader do material e, se não houver shader, registra um erro e continua para o próximo objeto.  **Propriedades Válidas do Shader**    Cria um conjunto de propriedades válidas, iterando pelas propriedades do shader e adicionando seus nomes ao conjunto **`validProperties`**.  **Propriedades Salvas do Material**    Obtém um objeto serializado do material e acessa as propriedades salvas.  **Remoção de Propriedades Inválidas**    Chama o método **`RemoveInvalidProperties`** para diferentes tipos de propriedades (**`m\_TexEnvs`**, **`m\_Ints`**, **`m\_Floats`**, **`m\_Colors`**) e aplica as modificações.  **Verificação de Objetos Não Materiais**    Se o objeto selecionado não for um material, registra um aviso.  **Método de Remoção de Propriedades Inválidas**    Este método remove propriedades inválidas:  - Intera pelas propriedades de trás para frente.  - Se a propriedade não estiver em **`validProperties`**, ela é removida.  **Resumo**  Este script limpa propriedades inválidas de materiais selecionados no Unity Editor, garantindo que apenas propriedades válidas definidas pelo shader do material sejam mantidas. Isso ajuda a evitar dados redundantes ou inválidos nos materiais, potencialmente melhorando o desempenho e a organização dos projetos no Unity. |

|  |
| --- |
| **Material Shader Converter** |
| **Finalidade do script**  Este script simplifica a conversão de shaders de material para uso no Universal Render Pipeline (URP) do Unity. Ele foi projetado para shaders MikuMikuDance (MMD), permitindo uma transição mais suave das propriedades do shader e garantindo compatibilidade com o URP.  **Visão geral de variáveis ​​e métodos**  **1. `ShaderModel` Enum**  Define os tipos de shaders para os quais o script pode converter:  - **`Default`**: shader básico.  - **`Tessellation`**: shader com detalhes de superfície mais finos.  - **`Empty`**: shader mínimo sem efeitos adicionais.  - **`FourLayers`** e **`EightLayers`**: shaders que suportam várias camadas de contorno.  - **`NoShadow`** e **`NoShadowAndTessellation`**: desabilita sombras, com uma opção para tessellation.  **2. Método ConvertShader()**  - Este é o ponto de entrada principal, permitindo que um usuário selecione materiais no editor Unity para serem convertidos.  - Ele itera sobre cada material selecionado e, com base no nome do shader, seleciona a conversão apropriada chamando **`ChangeShader()`** com parâmetros específicos.  **3. Método ChangeShader()**  - Ajusta as propriedades do shader para o material selecionado, registrando-o para possíveis ações de desfazer.  - Define parâmetros básicos de renderização (**`instancing`**, configurações **`GI`**).  - Chama métodos auxiliares (**`ApplyStandard`**, **`ApplyEmpty`**, **`ApplyMultiplePass`**) com base no tipo de shader.  - Limpa propriedades incompatíveis usando **`CleanMaterialProperties()`**.  **4. Método ApplyStandard()**  - Configura a cor primária, a transparência e o tamanho da borda de um material.  - Ajusta as propriedades de sombra e contorno com base no **`ShaderModel`** fornecido.  - Lida com as configurações de tesselação para shaders que precisam de detalhes de superfície mais finos.  **5. Método ApplyEmpty()**  - Aplica um shader mínimo, limpando propriedades extras e definindo transparência.  - Desativa a projeção de sombras e ajusta a fila de renderização para exibição transparente.  **6. Método ApplyMultiplePass()**  - Configura shaders que exigem renderização multipassagem (por exemplo, vários contornos).  - Mantém as configurações de cor do contorno e tamanho da borda para continuidade.  **7. Método RenderQueueToTransparent()**  - Ajusta a fila de renderização do material se a transparência estiver habilitada, colocando-a no intervalo correto para renderização transparente.  **8. Método CleanMaterialProperties()**  - Remove propriedades do material que não são necessárias para o novo shader.  - Usa **`RemoveInvalidProperties`** e **`CleanInvalidKeywords`** para limpeza precisa.  **9. Método RemoveInvalidProperties()**  - Remove propriedades inválidas do material com base nas suportadas pelo novo shader, o que garante transições limpas entre shaders.  **10. Método CleanInvalidKeywords()**  - Desativa quaisquer palavras-chave de shader não suportadas pelo novo shader, garantindo compatibilidade.  **Uso**  **1. Selecionando materiais no Unity Editor**  Selecione os materiais que deseja converter no editor Unity e execute **`ConvertShader()`** no menu em **`Assets > MMD Collection > Convert Material Shader (MMD4Mecanim)`**.  **2. Conversão de shader**  O script detectará o nome do shader de cada material, selecionará um shader compatível com URP apropriado e aplicará as configurações com base no tipo de shader.  **3. Capacidade de desfazer**  Como as alterações são registradas, os usuários podem desfazer a conversão do shader no editor.  **4. Otimização de renderização**  Por meio das funções auxiliares, o script garante que apenas as propriedades necessárias permaneçam no material, o que otimiza o desempenho da renderização.  Esta configuração automatiza a conversão de shader URP, garantindo que os shaders MMD permaneçam visualmente consistentes em projetos URP. |

|  |
| --- |
| **Manage Objects** |
| Este script é um editor personalizado para Unity. O script tem como objetivo gerenciar a visibilidade de uma lista de GameObjects no editor do Unity. Vou dividir a explicação em duas partes: o script principal (**`ManageObjects`**) e o editor personalizado (**`ManageObjectsEditor`**).  **Script Principal: `ManageObjects`**  O script **`ManageObjects`** é um componente MonoBehaviour que pode ser adicionado a um GameObject na cena. Ele fornece funcionalidades para gerenciar a visibilidade de uma lista de GameObjects.  **1. Definições e Variáveis**  - **`public ManageObjectsList[] manageObjects`**: Um array de **`ManageObjectsList`**, que é uma classe serializável (definida ao final do script). Cada item no array contém uma referência a um GameObject e seu estado de visibilidade.  - **`[HideInInspector] public bool state`**: Variável que controla o estado global de visibilidade para todos os objetos na lista. Se **`true`**, todos os objetos são visíveis; se **`false`**, todos são invisíveis.  - **`[HideInInspector] public bool hide`**: Variável que controla se os objetos serão ocultados no inspetor.  - **`[HideInInspector] public bool hideInspector`**: Variável que controla se o inspetor padrão do componente será ocultado.  **2. Métodos**  - **`public void Toggle(int i)`**: Alterna a visibilidade do GameObject na posição **`i`** do array **`manageObjects`**. O método registra a ação para permitir o desfazer (undo) e altera o estado de ativação do GameObject.  - **`public void ToggleAll()`**: Alterna a visibilidade de todos os GameObjects na lista, de acordo com o estado global **`state`**. Também registra a ação para desfazer e aplica o estado a todos os GameObjects.  - **`public void RemoveItem(int i)`**: Remove o GameObject na posição **`i`** do array **`manageObjects`**, registrando a ação para desfazer.  - ‘**Destruição Condicional’**: Fora de builds de desenvolvimento, o componente ‘**ManageObjects**’ é automaticamente destruído no método Start() para otimizar a performance em builds de produção.  **Editor Personalizado: `ManageObjectsEditor`**  O **`ManageObjectsEditor`** é uma classe que estende **`Editor`** e é responsável por criar uma interface personalizada no inspetor do Unity para o componente **`ManageObjects`**.  **1. Método `OnInspectorGUI()`**  Este método é chamado para desenhar a interface do inspetor personalizado:  - **Botão "Toggle All"**: Alterna a visibilidade de todos os objetos na lista e exibe o estado global.  - **`DrawObjectBox(script)`**: Desenha uma área onde você pode arrastar e soltar GameObjects para adicioná-los à lista. O método lida com eventos de arrastar e soltar e adiciona os objetos ao array se não estiverem já presentes.  - **`EditorGUILayout.Toggle()`**: Adiciona toggles para esconder objetos e o inspetor padrão.  - **`DrawDisplayButtons(script)`**: Desenha botões para alternar a visibilidade e remover cada GameObject da lista, se os objetos não estiverem escondidos.  - **`DrawDefaultInspector()`**: Desenha o inspetor padrão do componente, se não estiver escondido.  **2. Métodos Auxiliares**  - **`DrawObjectBox(ManageObjects script)`**: Desenha a área de arrastar e soltar para adicionar novos GameObjects à lista. Verifica os eventos de arrastar e soltar e adiciona os GameObjects ao array **`manageObjects`** se não estiverem já presentes.  - **`IsObjectInList(ManageObjects script, GameObject gameObject)`**: Verifica se um GameObject já está na lista **`manageObjects`**.  - `**DrawDisplayButtons(ManageObjects script)`**: Desenha botões para alternar a visibilidade e remover objetos da lista, além de exibir mensagens de aviso se não houver objetos.  **Classe `ManageObjectsList`**  - **`public GameObject gameObjects`**: Referência ao GameObject que será gerenciado.  - **`public bool objectState`**: Estado de visibilidade do GameObject.  **Resumo**  O script fornece uma maneira prática de gerenciar a visibilidade de múltiplos GameObjects a partir de um editor personalizado no Unity. O componente **`ManageObjects`** gerencia a lista de GameObjects e seu estado de visibilidade, enquanto o **`ManageObjectsEditor`** oferece uma interface de usuário no inspetor para adicionar, remover e alternar a visibilidade dos objetos. Em builds de produção, o componente é removido automaticamente para evitar impacto na performance. |

|  |
| --- |
| **Find Missing Scripts** |
| **Descrição Geral**  O objetivo deste script é encontrar todos os GameObjects na cena que possuem scripts faltando e selecioná-los no editor do Unity. Ele adiciona um item de menu para facilitar a execução dessa verificação.  **Estrutura do Script**  **1. Declarações de Namespace e Biblioteca**    - **`UnityEngine`**: Contém a API do Unity para manipulação de objetos de jogo.  - **`UnityEditor`**: Fornece classes e funções para criar ferramentas personalizadas no Editor do Unity.  - **`System.Collections.Generic`**: Inclui classes de coleções genéricas como **`List`**.  - **`System.Linq`**: Fornece funções de consulta para coleções, como **`Any`**.  **2. Classe `FindMissingScripts`**    - `**public class FindMissingScripts : MonoBehaviour`**: Define uma classe pública que herda de **`MonoBehaviour`**. Esta classe será anexada a um GameObject como um componente.  **3. Método `FindAllMissingScripts`**    - **`[MenuItem("GameObject/MMD Collection/Find Missing Scripts")]`**: Adiciona um item de menu no Unity Editor sob **`GameObject -> MMD Collection -> Find Missing Scripts`**. Este atributo faz com que o método **`FindAllMissingScripts`** seja executado quando o item de menu é selecionado.  - **`private static void FindAllMissingScripts()`**: Define um método estático privado que encontra todos os GameObjects com scripts faltando.  **4. Lista de GameObjects com Scripts Faltando**    - **`List<GameObject> objectsWithMissingScripts = new();`**: Cria uma lista para armazenar os GameObjects que possuem scripts faltando.  **5. Encontrar Todos os GameObjects**    - `**GameObject[] allObjects = Resources.FindObjectsOfTypeAll<GameObject>();`**: Obtém todos os GameObjects no projeto, incluindo aqueles que não estão carregados na cena atual.  **6. Iterar Sobre Todos os GameObjects**    - **`foreach (GameObject obj in allObjects)`**: Itera sobre todos os GameObjects encontrados.  - **`if (!obj.scene.IsValid() || !obj.scene.isLoaded || (obj.hideFlags & (HideFlags.NotEditable | HideFlags.HideAndDontSave)) != 0)`**: Verifica se o objeto pertence a uma cena válida, se a cena está carregada e se o objeto não está marcado como não editável ou não salvável. Se alguma dessas condições for verdadeira, o objeto é ignorado (**`continue`**).  - **`Component[] components = obj.GetComponents<Component>();`**: Obtém todos os componentes anexados ao GameObject atual.  - **`if (components == null) continue;`**: Pula para o próximo objeto se não houver componentes (o que não deveria ocorrer, mas é uma verificação de segurança).  - **`if (components.Any(component => component == null))`**: Verifica se algum dos componentes é nulo (indicando um script faltando).  - **`objectsWithMissingScripts.Add(obj);`**: Adiciona o GameObject à lista se possuir algum script faltando.  **7. Log e Seleção dos Objetos**    - **`Debug.Log($"Found {objectsWithMissingScripts.Count} objects with missing scripts.");`**: Exibe uma mensagem no console do Unity com o número de objetos encontrados com scripts faltando.  - **`Selection.objects = objectsWithMissingScripts.ToArray();`**: Seleciona os objetos com scripts faltando no Editor do Unity, facilitando a localização e correção.  **Conclusão**  Este script é útil para desenvolvedores que precisam garantir que não há scripts faltando nos GameObjects da cena, o que pode causar erros ou comportamentos inesperados. Ao adicionar um item de menu no editor, ele torna o processo de verificação rápido e acessível. |

|  |
| --- |
| **Shader Keyword Checker** |
| Este script foi criado para verificar palavras-chave de shader global em materiais selecionados no Unity Editor. Ele adiciona uma nova opção no menu de contexto Asset do Unity Editor, permitindo que os usuários acionem a verificação em materiais selecionados.  **Variáveis ​​e métodos**  1. **Atributo MenuItem**  Este atributo é usado para criar um novo item no menu Unity Editor. O caminho do menu é **`"Assets/MMD Collection/Check Shader Keywords"`**, e ele vincula ao método **`CheckShaderKeywords`**. Quando selecionado no menu, o método é executado.  **2. Método CheckShaderKeywords**  Este é o método principal acionado quando o usuário seleciona o item de menu. Seu objetivo principal é iterar por todos os objetos selecionados no Unity Editor e verificar se cada um é um material.  - **Selection.objects**: Esta é uma propriedade Unity que contém a lista de objetos selecionados no editor. O método a usa para acessar esses objetos.  - **Material material**: O script verifica se cada objeto selecionado é um **`Material`**. Se for, o **`Shader`** associado ao material é recuperado.  - **shader**: Esta variável representa o shader associado ao material. Se nenhum shader for atribuído ao material, uma mensagem de erro será registrada.  - Se o material tiver um shader, o script chamará o método **`CheckGlobalShaderKeywords`** para inspecionar as palavras-chave do shader global.  **3. Método CheckGlobalShaderKeywords**  Este método recebe um objeto **`Shader`** e verifica suas palavras-chave do shader global.  - **ShaderUtil**: O Unity tem uma classe interna chamada **`ShaderUtil`** que contém utilitários para operações relacionadas ao shader. No entanto, esta classe não é acessível publicamente.  - **Reflection (GetMethod)**: Como **`ShaderUtil`** é interno, o script usa reflection para acessar seu método oculto **`GetShaderGlobalKeywords`**. O Reflection permite acessar métodos e propriedades privadas ou internas em C#.  - **getKeywordsMethod**: Esta variável armazena o resultado da chamada da função **`GetMethod`**, que tenta encontrar o método **`GetShaderGlobalKeywords`** usando seu nome e sinalizadores de vinculação especiais (**`Static`** e **`NonPublic`**).  - Se o método for encontrado, ele será invocado para recuperar as palavras-chave do shader global, que são registradas no console do Unity. Se o método não for encontrado, um erro será registrado, indicando a falha.  **Uso**  1. Selecione materiais no Unity Editor.  2. Clique com o botão direito do mouse nos ativos selecionados e navegue até a opção **`"MMD Collection"`** no menu **`"Assets"`**.  3. Escolha **`"Check Shader Keywords"`**.  4. O script verificará se os ativos selecionados são materiais e recuperará as palavras-chave globais do shader. Essas palavras-chave serão registradas no console do Unity. |