GameEngine7

2024年10月7日 16:21

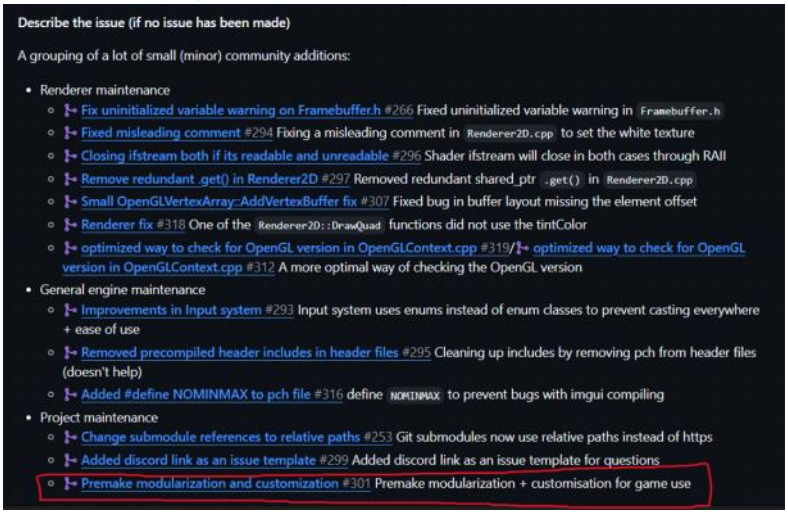
----------Saving & Loading scene----------

》》》》更改 Premake 文件构架

这一集中 Cherno 对 premake 文件进行了操作，不过此时 Premake 文件的构架发生了改变（现在每个项目的 premake 被放置在项目的文件夹下，而不是集中放置在 Nut 根目录下的 Premake 文件中），这是因为之前的一次 pull request。

本来准备先完善引擎 UI，后面集中对引擎进行维护，现在看来就先提交一下这个更改吧。

具体可以参考：（ https://github.com/TheCherno/Hazel/pull/320 ）



Describe the issue (if no issue has been made)

A grouping of a lot of small (minor) community additions:

- Renderer maintenance
  - Fix uninitialized variable warning on Framebuffer.h #266 Fixed uninitialized variable warning in `Framebuffer.h`
  - Fixed misleading comment #294 Fixing a misleading comment in `Renderer2D.cpp` to set the white texture
  - Closing ifstream both if its readable and unreadable #296 Shader ifstream will close in both cases through RAII
  - Remove redundant .get() in Renderer2D #297 Removed redundant shared_ptr `.get()` in `Renderer2D.cpp`
  - Small OpenGLVertexArray::AddVertexBuffer fix #307 Fixed bug in buffer layout missing the element offset
  - Renderer fix #318 One of the `Renderer2D::DrawQuad` functions did not use the tintColor
  - optimized way to check for OpenGL version in OpenGLContext.cpp #319/optimized way to check for OpenGL version in OpenGLContext.cpp #312 A more optimal way of checking the OpenGL version
- General engine maintenance
  - Improvements in Input system #293 Input system uses enums instead of enum classes to prevent casting everywhere + ease of use
  - Removed precompiled header includes in header files #295 Cleaning up includes by removing pch from header files (doesn't help)
  - Added #define NOMINMAX to pch file #316 define `NOMINMAX` to prevent bugs with imgui compiling
- Project maintenance
  - Change submodule references to relative paths #253 Git submodules now use relative paths instead of https
  - Added discord link as an issue template #299 Added discord link as an issue template for questions
  - Premake modularization and customization #301 Premake modularization + customisation for game use

》》》》什么是 .editorconfig 文件？有什么作用?

问题引入：在深入研究这次提交时，一个以 .editorconfig 署名的文件映入眼帘，这是什么文件？

文件介绍：

EditorConfig helps maintain consistent coding styles for multiple developers working on the same project across various editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles. EditorConfig files are easily readable and they work nicely with version control systems. 来自 <https://editorconfig.org/>	**翻译：** EditorConfig 可帮助多个开发人员在不同的编辑器或 IDE 上维护同一个项目的编码风格，使其保持一致。EditorConfig 项目包含一个用于定义编码风格的文件格式和一组文本编辑器插件，这些插件可让编辑器读取文件格式并遵循定义的风格。EditorConfig 文件易于阅读，并且可与版本控制系统完美配合。

作用：

通过使用 EditorConfig 文件，团队中的每个成员可以确保他们的代码遵循相同的格式，降低因代码风格不一致而引起的问题。许多现代代码编辑器和 IDE（如 Visual Studio Code、Atom、JetBrains 系列等）都支持 EditorConfig，可以自动读取这些规则并应用到打开的文件中。

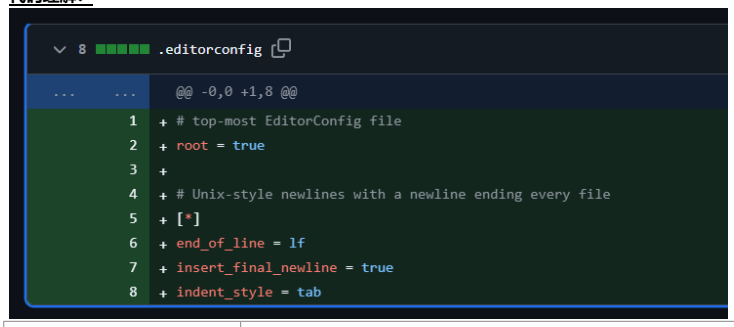
使用规范：

文件名：	文件名为 .editorconfig，通常放在项目根目录。

键值对格式：	使用 key = value 的形式定义规则，每条规则占一行。 空行和以 # 开始的行会被视为注释。
范围选择器：	使用 [*] 表示应用于所有文件，也可以使用其他模式如 *.js 或 *.py 来指定特定文件类型。
支持的属性：（支持的键值对）	常用属性包括： root：指示是否为顶层文件。 end_of_line：指定行结束符（如 lf, crlf, cr）。 insert_final_newline：是否在文件末尾插入换行符。 indent_style：设置缩进样式（如 tab 或 space）。 indent_size：指定缩进的大小，可以是数字或 tab。 charset：文件字符集（如 utf-8, latin1 等）。 trim_trailing_whitespace：是否修剪行尾空白。

详情参考文档：（ https://spec.editorconfig.org/ ）

代码理解：



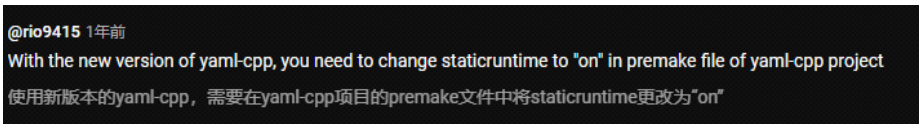
root = true:	指示这是一个顶层的 EditorConfig 文件，编辑器在找到此文件后不会再向上查找其他 EditorConfig 文件。
[*]:	表示应用于所有文件类型的规则。
end_of_line = lf:	指定行结束符为 Unix 风格的换行符（LF，Line Feed）。这通常在类 Unix 系统（如 Linux 和 macOS）中使用。
insert_final_newline = true:	指定在每个文件的末尾插入一个换行符。这是一种良好的编码习惯，许多项目标准要求这样做。
indent_style = tab:	指定缩进样式为制表符（tab），而不是空格。这会影响代码的缩进方式。

《《《《拓展：什么是 Hard tabs？什么是 Soft tabs?

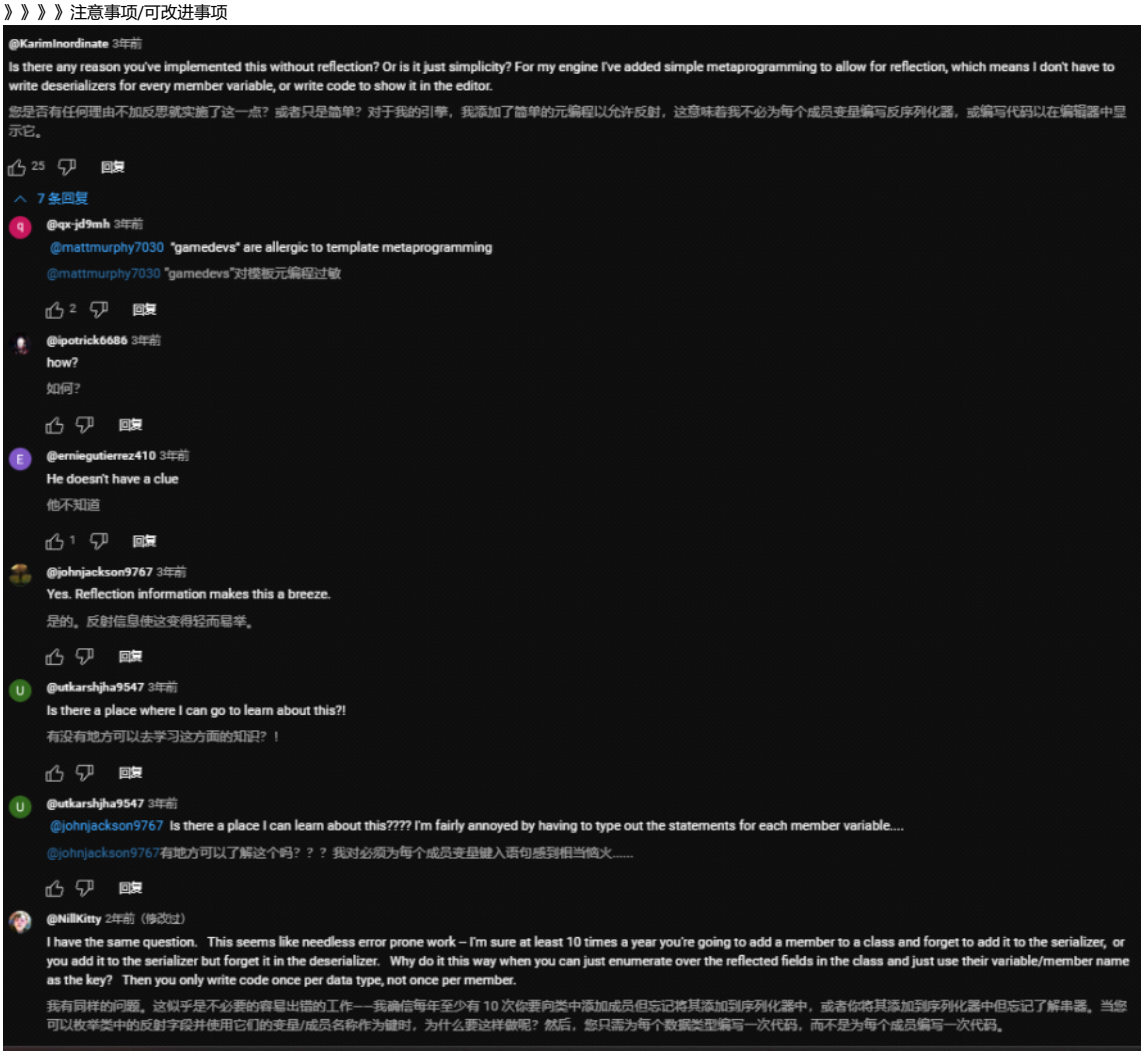
Hard Tabs	是使用制表符进行缩进，具有灵活性但可能导致跨环境的不一致。
Soft Tabs	是使用空格进行缩进，保证了一致性但文件体积可能更大。

选择使用哪种方式通常取决于团队的编码标准或个人偏好。

》》》》关于最新的 YAML 导致链接错误的解决方案



》》》》注意事项/可改进事项



AND

@wakeupthesun3 1年前（修改过）

Another thing to note (I'm not sure if this is covered later) is the scene is being deserialized in inverse order in which the original entities were added to the scene. You can see this by the original scene has the red square on top, covering the green square. When deserialized, the green square is on top. You can either serialize your entities backwards, or deserialize them backwards. I think deserializing backwards is better, because then the serialized file will match the order of your hierarchy panel. To deserialize backwards, you can make a vector of the entity nodes and then get a reverse iterator to that vector:

```
auto entitiesNode = data["Entities"];

  // reverse it to add the entities in the order they were
  // originally added
  std::vector<YAML::Node> entitiesRev(entitiesNode.begin(),
                                      entitiesNode.end());

  for (auto it = entitiesRev.rbegin(); it != entitiesRev.rend(); ++it)
  {
    s_deserializeEntity(*it, mp_scene.get());
  }
```

Have fun!

另一件需要注意的事情（我不确定稍后是否会介绍这一点）是场景正在以与原始实体添加到场景相反的顺序进行反序列化。您可以通过原始场景看到顶部有红色方块，覆盖了绿色方块。反序列化时，绿色方块位于顶部。您可以向后序列化实体，也可以向后反序列化它们。我认为向后反序列化更好，因为这样序列化的文件将与层次结构面板的顺序匹配。要向后反序列化，您可以创建实体节点的向量，然后获取该向量的反向迭代器：

```
自动实体节点=数据["实体"];

// 反转它以按实体的顺序添加实体
// 最初添加的
std::vector<YAML::Node>EntityRev(entitiesNode.begin(),
实体节点.end());

for (auto it =EntityRev.rbegin();it!=entitiesRev.rend();++it)
  {
s_deserializeEntity(*it, mp_scene.get());
  }
```

玩得开心！

---------- Multiple Render Targets and Framebuffer refactor----------

》》》》gl_VertexID 在 GLSL 中关于顶点ID的一些细节：

gl_VertexID

gl_Position和gl_PointSize都是**输出变量**，因为它们的值是作为顶点着色器的输出被读取的。我们可以对它们进行写入，来改变结果。顶点着色器还为我们提供了一个有趣的**输入变量**，我们只能对它进行读取，它叫做gl_VertexID。

整型变量gl_VertexID储存了正在绘制顶点的当前ID。当（使用glDrawElements）进行索引渲染的时候，这个变量会存储正在绘制顶点的当前索引。当（使用glDrawArrays）不使用索引进行绘制的时候，这个变量会储存从渲染调用开始的已处理顶点数量。

虽然现在它没有什么具体的用途，但知道我们能够访问这个信息总是好的。