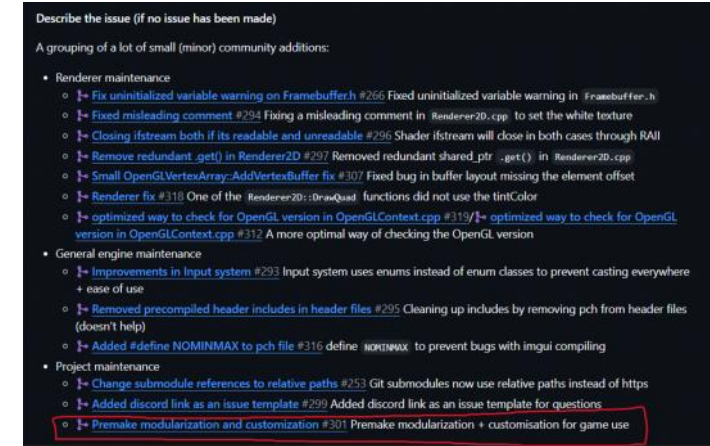


-----Saving & Loading scene-----

》》》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>）

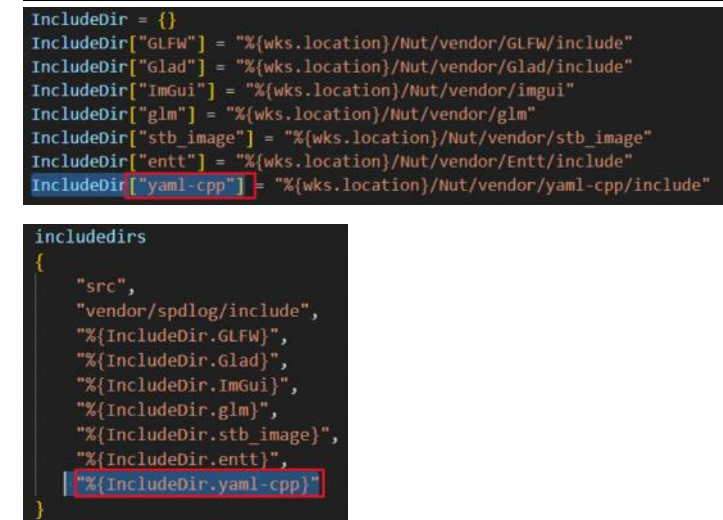


》》》一个问题：关于 premake 文件中的命名

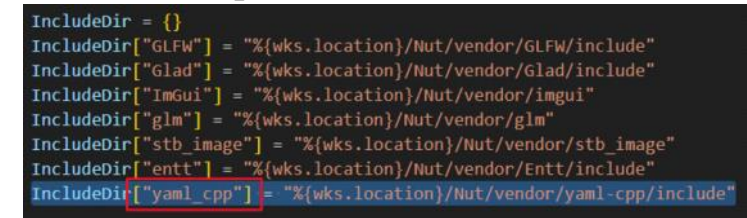
当我将 yaml-cpp 作为键（Key），并以此来索引存储的值（Value），此时会出现一个错误：

Error: [string "return IncludeDir.yaml-cpp"]:1: attempt to perform arithmetic on a nil value (field 'yaml') in token: IncludeDir.yaml-cpp	（错误：[string "return IncludeDir.yaml-cpp"] : 1: 尝试对令牌中的零值（字段 "yaml"）执行算术运算：IncludeDir.yaml-cpp）
---	--

编译器似乎将 '-' 识别为算术运算符，而不是文本符号，这导致他尝试进行算术运算操作。



但是当我将 '-' 更改为 '_' 时，这样的问题便消失了。



```

includedirs
{
    "src",
    "vendor/spdlog/include",
    "%{IncludeDir.GLFW}",
    "%{IncludeDir.Glad}",
    "%{IncludeDir.ImGui}",
    "%{IncludeDir.glm}",
    "%{IncludeDir.stb_image}",
    "%{IncludeDir.entt}",
    "%{IncludeDir}.yaml_cpp"
}

```

》》》关于最新的 YAML 导致链接错误的解决方案

编译器疑似在以动态库的方式尝试运行 yaml-cpp 库，并发出了很多警告

```

yaml-cpp\include\yaml-cpp\parser.h(65,28) warning C4251: "YAML::Parser::m_scanner" : class "std::unique_ptr<YAML::Scanner,std::default_delete<YAML::Scanner>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\parser.h(66,31) warning C4251: "YAML::Parser::m_directives" : class "std::unique_ptr<YAML::Directives,std::default_delete<YAML::Directives>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\parser.h(66,31) warning C4251: "YAML::Parser::m_directives" : class "std::unique_ptr<YAML::Directives,std::default_delete<YAML::Directives>>" 需要有 dll 接口由 class "YAML::Parser" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\binary.h(65,30) warning C4251: "YAML::Binary::m_data" : class "std::vector<unsigned char,std::allocator<unsigned char>>" 需要有 dll 接口由 class "YAML::Binary" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\binary.h(18) message : 参见 "std::vector<unsigned char,std::allocator<unsigned char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\ostream_wrapper.h(48,29) warning C4251: "YAML::ostream_wrapper::m_buffer" : class "std::vector<char,std::allocator<char>>" 需要有 dll 接口由 class "YAML::ostream_wrapper" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\ostream_wrapper.h(49) message : 参见 "std::vector<char,std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\emitter.h(132,33) warning C4251: "YAML::Emitter::m_state" : class "std::unique_ptr<YAML::EmitterState,std::default_delete<YAML::EmitterState>>" 需要有 dll 接口由 class "YAML::Emitter" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\emitter.h(132) message : 参见 "std::unique_ptr<YAML::EmitterState,std::default_delete<YAML::EmitterState>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(155,58) warning C4275: 非 dll 接口 class "std::runtime_error" 用作 dll 接口 class "YAML::Exception" 的基 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
36) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\stdexcept(101) message : 参见 "std::runtime_error" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(155) message : 参见 "YAML::Exception" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\exceptions.h(164,15) warning C4251: "YAML::Exception::msg" : class "std::basic_string<char,std::char_traits<char>,std::allocator<char>>" 需要有 dll 接口由 class "YAML::Exception" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
38) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\wstring(4871) message : 参见 "std::basic_string<char,std::char_traits<char>,std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\node.h(135,15) warning C4251: "YAML::Node::m_invalid_key" : class "std::basic_string<char,std::char_traits<char>,std::allocator<char>>" 需要有 dll 接口由 class "YAML::Node" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
38) Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\wstring(4871) message : 参见 "std::basic_string<char,std::char_traits<char>,std::allocator<char>>" 的声明 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)
yaml-cpp\include\yaml-cpp\node.h(136,40) warning C4251: "YAML::Node::m_memory" : class "std::shared_ptr<YAML::detail::memory_holder>" 需要有 dll 接口由 class "YAML::Node" 的客户端使用 (编译源文件 src\Wut\Scene\SceneSerializer.cpp)

```

初步解决方案:

@mjthebest7294 2年前

I had to define "YAML_CPP_STATIC_DEFINE" in the premake for the newer version of YAML, otherwise it will try to compile as a DLL.

我必须在新版本 YAML 的预编译中定义 "YAML_CPP_STATIC_DEFINE", 否则它将会尝试编译为 DLL

12 回复

6 条回复

@p3rk4n27 2年前

It now build yaml project but cant link it to engine... there are errors like unresolved external dllimport...

它现在构建 yaml 项目, 但无法将其链接到引擎... 存在诸如未解析的外部 dllimport 之类的错误...

2 回复

@mjthebest7294 2年前

@p3rk4n27 maybe the engine compiles as a .dll instead of a static .lib

@p3rk4n27 也许引擎编译为 .dll 而不是静态 .lib

》》AND..

@rio9415 1年前

With the new version of yaml-cpp, you need to change staticruntime to "on" in premake file of yaml-cpp project

使用新版本的yaml-cpp, 需要在yaml-cpp项目的premake文件中将staticruntime更改为"on"

首先我已经在 yaml-cpp 的 premake 文件中声明了 "YAML_CPP_STATIC_DEFINE", 并且打开了 staticruntime, 但我发现没有作用。

```

defines
{
    "YAML_CPP_STATIC_DEFINE"
}

filter "system:windows"
{
    systemversion "latest"
    cppdialect "C++17"
    staticruntime "on"
}

```

接着解决:

问题是, 你还需要在你所使用项目的 premake 文件中再次声明 "YAML_CPP_STATIC_DEFINE"

```
project "Nut-Editor"
objdir {"${wks.location}/bin-int/" .. outputdir .. "/"${prj.name}"}

files
{
    "src/**/*.h",
    "src/**/*.cpp"
}

defines
{
    "YAML_CPP_STATIC_DEFINE"
}

includedirs
{
    "${wks.location}/Nut/vendor/spdlog/include",
    "${wks.location}/Nut/src",
    "${wks.location}/Nut/vendor",
    "${IncludeDir.glm}",
    "${IncludeDir.entt}",
    "${IncludeDir.yaml_cpp}"
}
```

@kingofspades9720 2周前

If you are having issues using YAML, one fix might be adding `#define YAML_CPP_STATIC_DEFINE` inside of the Hazel premake file not just inside of the yaml-cpp premake file.

如果您在使用 YAML 时遇到问题，一种解决方法可能是在 Hazel 预编译文件内添加 `#define YAML_CPP_STATIC_DEFINE`，而不仅仅是在 yaml-cpp 预编译文件内。

👍 2 🗨 回复

@yu_a_v4427 11个月前

for new version of yaml-cpp just add a `#define YAML_CPP_STATIC_DEFINE` before including `<yaml-cpp/yaml.h>` in any file,


and turn on `staticruntime` in premakefile of yaml-cpp

对于新版本的 yaml-cpp，只需在任何文件中包含 `<yaml-cpp/yaml.h>` 之前添加 `#define YAML_CPP_STATIC_DEFINE`，

并在 yaml-cpp 的 premakefile 中打开 `staticruntime`

👍 6 🗨 回复

总结：



1分钟前

As of October 2024, you can correctly run yaml-cpp with the following requirements:

1. Define `YAML_CPP_STATIC_DEFINE` in the premake file of yaml-cpp, and define `YAML_CPP_STATIC_DEFINE` in the project configuration file that uses yaml-cpp (such as premake)
2. Define `staticruntime 'on'` in the yaml-cpp premake file

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

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

Table of Contents

- [EditorConfig Specification](#)
 - [Introduction \(informative\)](#)
 - [Terminology](#)
 - [File Format](#)
 - [No inline comments](#)
 - [Parts of an EditorConfig file](#)
 - [Glob Expressions](#)
 - [File Processing](#)
 - [Supported Pairs](#)

代码理解：

```
...  ...  @@ -0,0 +1,8 @@

1  + # top-most EditorConfig file
2  + root = true
3  +
4  + # Unix-style newlines with a newline ending every file
5  + [*]
6  + end_of_line = lf
7  + insert_final_newline = true
8  + indent_style = tab
```

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。

》》》Y A M L U know what I'm saying

》》》YAML YAML YAML

》》》关于这次 premake 构架的维护，我只上传了一部分，剩下的留到之后维护时再做。现在我去了解一下 YAML。

》》》YAML, What is yaml ? What we can do by yaml ?

介绍：

YAML is a human-readable data serialization language that is often used for writing configuration files. Depending on whom you ask, YAML stands for yet another markup language or YAML ain't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents. 来自 < https://www.redhat.com/en/topics/automation/what-is-yaml >	YAML 是一种人类可读的数据序列化语言，通常用于编写配置文件。根据使用的对象，YAML 可以代表另一种标记语言或者说 YAML 根本不是标记语言（递归缩写），这强调了 YAML 用于数据，而不是文档。
---	---

理解：

在程序中，我们可以使用 yaml 对文件进行两种操作：序列化和反序列化（Serialize & Deserialize）。
序列化意味着我们可以将复杂的数据转变为字节流，进而可以将其轻易保存到文件或数据库中。
反序列化则意味着我们可以对已经序列化的数据进行逆处理，进而将数据转换回原始的数据结构或对象状态。

基础：

基本结构

映射（Map）：键值对的集合。	key: value
序列（Sequence）：有序的元素列表。	- item1 - item2

	- item3
--	---------

2. 嵌套结构

YAML 支持嵌套映射和序列，可以组合使用：	person: name: John Doe age: 30 hobbies: - reading - cycling
------------------------	--

3. 数据类型

YAML 支持多种数据类型，包括：字符串，数字，布尔值，Null 值。	例如： string: "Hello, World!" number: 42 boolean: true null_value: null
-------------------------------------	---

》》》yaml-cpp 的使用（详情请阅览：<https://github.com/ibeder/yaml-cpp/blob/master/docs/Tutorial.md>）

在 C++ 中使用 yaml-cpp 库，可以方便地处理 YAML 数据的读取和写入。（以下是读取 Yaml 文件和写入 Yaml 文件的示例）

读取 YAML	<pre>#include <iostream> #include <yaml-cpp/yaml.h> int main() { YAML::Node config = YAML::LoadFile("config.yaml"); std::string name = config["person"]["name"].as<std::string>(); std::cout << "Name: " << name << std::endl; return 0; }</pre>
写入 YAML： 使用 YAML::Emitter 可以生成 YAML 文件	<pre>#include <iostream> #include <yaml-cpp/yaml.h> int main() { YAML::Emitter out; out << YAML::BeginMap; out << YAML::Key << "name" << YAML::Value << "John Doe"; out << YAML::Key << "age" << YAML::Value << 30; out << YAML::EndMap; std::cout << out.str() << std::endl; // 输出生成的 YAML return 0; }</pre>

YAML::Node

定义：YAML::Node 是 YAML-CPP 中的一个核心类，表示 YAML 文档中的一个节点。一个节点可以是标量（单个值）、序列（列表）或映射（键值对）。通过 YAML::Node，你可以以编程方式访问和操作 YAML 数据结构。	创建和使用 YAML::Node： Eg. <pre>#include <yaml-cpp/yaml.h> YAML::Node node = YAML::Load("key: value"); std::string value = node["key"].as<std::string>();</pre>
--	--

Sequences 和 Maps

Sequences （序列）是一个有序列表，表示一组无命名的值。它们在 YAML 中用短横线表示：	fruits: - Apple - Banana - Cherry
在 YAML-CPP 中，你可以这样处理序列：	Eg. <pre>YAML::Node sequence = YAML::Load("[Apple, Banana, Cherry]"); for (const auto& item : sequence) { std::cout << item.as<std::string>() << std::endl; }</pre>
Maps （映射）是一组键值对，表示命名的值。它们在 YAML 中用冒号分隔表示：	person: name: John Doe age: 30
在 YAML-CPP 中，你可以这样处理映射：	Eg. <pre>YAML::Node map = YAML::Load("name: John Doe\nage: 30"); std::string name = map["name"].as<std::string>(); int age = map["age"].as<int>();</pre>

Sequences 和 Maps 的不同之处

序列和映射都是 YAML::Node 的一种。你可以在一个映射中嵌套序列，反之亦然。

不同之处：

序列：	没有键，每个项都有顺序。
映射：	每个项都有唯一的键，顺序不重要。

Converting To/From Native Data Types

YAML-CPP 提供了方便的方法来将 YAML::Node 转换为 C++ 的原生数据类型。你可以使用 as<T>() 方法进行转换。

示例：从 YAML::Node 转换到原生数据类型	<pre>YAML::Node node = YAML::Load("name: John Doe\nage: 30"); std::string name = node["name"].as<std::string>(); int age = node["age"].as<int>();</pre>
示例：从原生数据类型转换到 YAML::Node	<pre>YAML::Node newNode; newNode["name"] = "Jane Doe"; newNode["age"] = 28; // 序列 YAML::Node fruits; fruits.push_back("Apple"); fruits.push_back("Banana"); newNode["fruits"] = fruits; // 输出为 YAML 格式 std::cout << newNode << std::endl;</pre>

》》由此引出两个疑惑:

问题一：

查阅文档时，我发现当插入的索引超出当前序列的范围时，YAML-CPP 会将节点视为映射，而不是继续保持序列

Indexing a sequence node by an index that's not in its range will *usually* turn it into a map, but if the index is one past the end of the sequence, then the sequence will grow by one to accommodate it. (That's the only exception to this rule.) For example,

添加新元素:

node[3] = 4; 试图在索引 3 的位置插入 4。由于索引 3 在当前序列中不存在，所以 node 仍然被认为是序列，结果是 [1, 2, 3, 4]。

```
YAML::Node node = YAML::Load("[1, 2, 3]");
node[3] = 4; // still a sequence, [1, 2, 3, 4]
node[10] = 10; // now it's a map! {0: 1, 1: 2, 2: 3, 3: 4, 10: 10}
```

插入非连续索引:

node[10] = 10; 尝试将值 10 插入到索引 10 的位置，因为 10 远大于当前序列的最大索引 (3)，这导致 node 变成了一个映射。最终结果是 {0: 1, 1: 2, 2: 3, 3: 4, 10: 10}。

结论：动态类型：YAML::Node 的类型是动态的，可以在运行时根据操作的不同而变化。当你使用整数索引时，它保持序列。当你使用非连续的索引或字符串键时，它会转变为映射。

问题二：如何为Node添加一个映射？

在 YAML::Node node = YAML::Load("[1, 2, 3]"); 的情况下，使用 node[1] = 5 是不合适的。

如果你想让 node[1] 表示一个映射，node[1] = 5 会将序列中索引为 1 的元素（即第二个元素）设置为整数 5，而不是将其更改为一个映射。

如果你想在当前位置设置一个映射，你可以这样做：	<pre>YAML::Node node = YAML::Load("[1, 2, 3]"); node[1] = YAML::Node(YAML::NodeType::Map); // 创建一个新的映射 node[1]["key"] = "value"; // 向映射中添加键值对</pre>
结构：	<pre>- 1 - key: value - 3</pre>
或者：	<pre>YAML::Node node = YAML::Load("[1, 2, 3]"); // 将 node[1] 设置为一个新的映射 node[1] = YAML::Node(YAML::NodeType::Map); // 设置键为原来的值 2，并赋值为 5 node[1][2] = 5; // 这里的 2 是之前 index 1 的值</pre>
结构：	<pre>- 1 - 2: 5 - 3</pre>

注意：

如果你使用了 node["1"] = 5，由于 "1" 是一个字符串键，而不是数字索引，这将使程序尝试在 node 中以 "1" 为键插入值 5。node 原本是一个序列，但它会因此转变为一个映射。	最终结果会是 { 0: 1, 1: 2, 2: 3, "1": 5}, 其中 "1" 是一个新的字符串键。
---	---

》》》ifstream 和 ofstream 之间的关系

二者定义在 <fstream> 头文件中，管理文件流。

std::ifstream:	用于从文件中读取数据（输入文件流）。
Std::ofstream:	用于向文件中写入数据（输出文件流）。

易混淆: <iostream>和文件流没有关系，<iostream> 是提供输入或输出流的标准库，主要包括 std::cin, std::cout, std::cerr 等。

》》ofstream 的使用: std::ofstream 用于创建和写入文件

```
// Send file-stream
std::ofstream fout(filepath); // Create and open a file from the filepath
fout << out.c_str();          // Writing data
```

》》ifstream 的使用: std::ifstream 用于读入文件，进而对读入的文件进行一些处理。

(图例: 逐行读取文件内容到字符串中)

```
std::ifstream file(filepath);
std::string line;
// 逐行读取文件内容
while (std::getline(file, line)) {
    std::cout << line << std::endl;
}
```

或者 (比上述方法更加高效, 迅捷)

```
std::ifstream file(filepath);
std::stringstream fileContent;
fileContent << file.rdbuf();
```

《《《什么是 rdbuf();

在 C++ 中, rd 通常是 "read" 的简写, 意味着与读取操作相关的函数。

rdbuf()

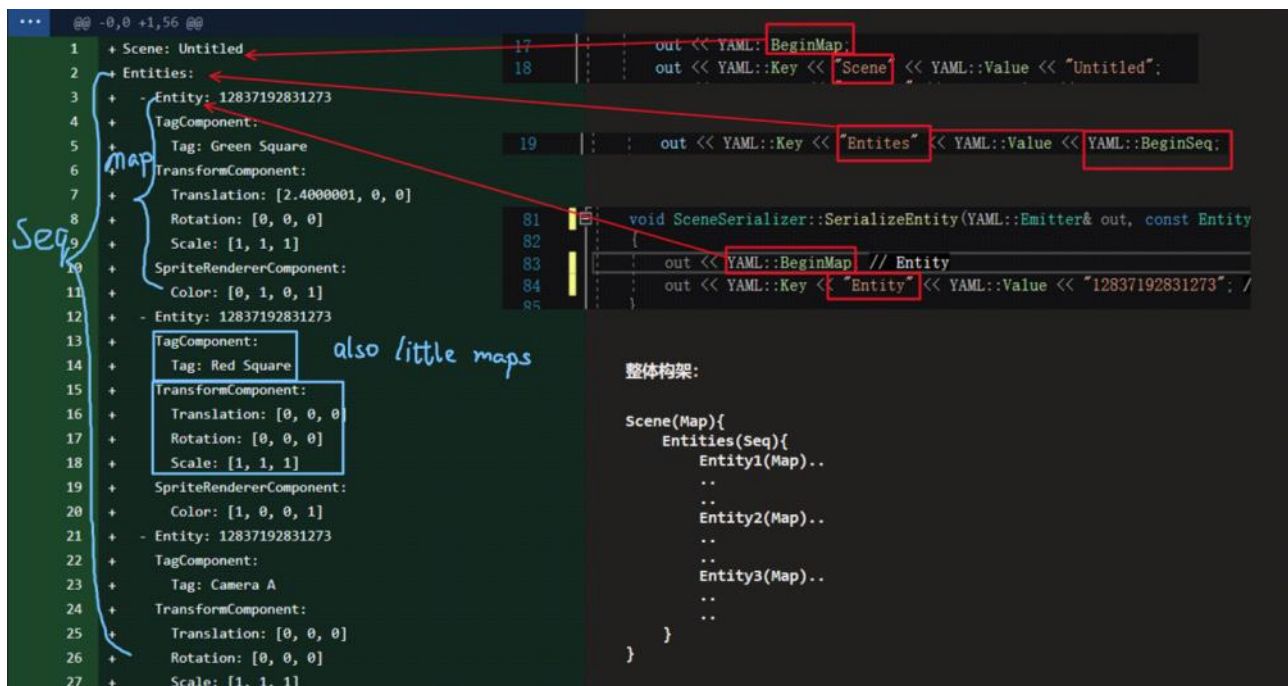
释义:	rdbuf() 是 C++ 中的一个成员函数, 可以直接访问流的底层缓冲区。它通常用于与输入输出流 (如 std::ifstream, std::ofstream, std::iostream 等) 交互。
返回类型:	std::streambuf* 返回指向与流关联的 std::streambuf 对象的指针。该指针可以用于直接进行低级别的输入输出操作。
优点: 直接访问缓冲区	rdbuf() 返回一个指向当前流缓冲区的指针 (即 std::streambuf 对象), 允许你直接从流中读取或写入数据。 这意味着, 你可以将整个文件的内容一次性读入, 而不需要逐行或逐字符地读取, 从而提高了效率。 当处理大型文件时, 逐行读取会涉及多次 I/O 操作, 这可能导致性能瓶颈。而使用 rdbuf() 可以减少这些 I/O 操作, 因为它一次性读取整个缓冲区的数据。

类似的 rd 开头的函数还有 rdstate()	含义: rdstate() 是一个成员函数, 用于获取流的状态标志。它返回一个整数, 表示流的当前状态, 包括是否已达到文件结束、是否发生了错误等。
--------------------------	--

》》》FIEL STRUCTURE U know what I'm saying

》》》YAML YAML YAML

》》》YAML 文件构架, YAML 文件设置思路



因此我们也可解释 Deserialize() 函数中做出的操作: 从 data(map) 中取出序列 entities(seq), 然后通过 For 循环对序列中的 entity(map) 进行读取, 随后根据读取的数据去复现场景。
 需要提醒的是: Map 中的元素不能重复, Seq 中的元素可以重复。

```
// According to data, we reproduce the scene
auto entities = data["Entity"];
if(entities)
{
    for (auto entity : entities)
    {
        // reproduce codes
    }
}

return true;
```

》》》》YAML::Emitter out << YAML::Flow;

概念: out << YAML::Flow 是 C++ 中使用 YAML 库 (如 yaml-cpp) 时的一种语法, 它用来设置 YAML 输出的格式为“流式” (flow style)。

详解: YAML::Flow 是一个常量, 指示输出的 YAML 数据应采用流式表示形式。

流式表示形式将集合 (如数组和映射) 以更紧凑的方式表示, 例如使用方括号 [] 表示数组, 使用花括号 {} 表示映射。

使用场景:

当你想要以更紧凑的格式输出数据时, 可以使用流式表示形式。相比于块状表示 (block style), 流式表示在视觉上更简洁, 适用于小型数据结构或在单行内表示数据。

假设你有一个简单的 YAML 数据结构, 如果不使用 YAML::Flow, 输出可能是这样的 (块状表示):	items: - item1 - item2
而如果使用 YAML::Flow, 此时, 输出将是:	items: [item1, item2]

示例代码:

```
#include <yaml-cpp/yaml.h>
#include <iostream>
int main() {
    YAML::Emitter out;

    out << YAML::Flow; // 设置为流式格式
    out << YAML::BeginMap
    << YAML::Key << "items" << YAML::Value
    << YAML::BeginSeq
    << "item1" << "item2"
    << YAML::EndSeq
    << YAML::EndMap;
    std::cout << out.str() << std::endl;
}
```

》》》》设计上的理解

1.Serialize


```

if (entity.HasComponent<CameraComponent>())
{
    out << YAML::Key << "CameraComponent";
    out << YAML::BeginMap;

    auto& cc = entity.GetComponent<CameraComponent>();
    auto& camera = cc.Camera;

    out << YAML::Key << "Camera" << YAML::Value;
    out << YAML::BeginMap;{
        out << YAML::Key << "ProjectionType" << YAML::Value << (int)camera.GetProjectionType();
        out << YAML::Key << "PerspectiveFOV" << YAML::Value << camera.GetPerspectiveVerticalFOV();
        out << YAML::Key << "PerspectiveNear" << YAML::Value << camera.GetPerspectiveNearClip();
        out << YAML::Key << "PerspectiveFar" << YAML::Value << camera.GetPerspectiveFarClip();
        out << YAML::Key << "OrthographicSize" << YAML::Value << camera.GetOrthographicSize();
        out << YAML::Key << "OrthographicNear" << YAML::Value << camera.GetOrthographicNearClip();
        out << YAML::Key << "OrthographicFar" << YAML::Value << camera.GetOrthographicFarClip();
    }
    out << YAML::EndMap;

    out << YAML::Key << "Primary" << YAML::Value << cc.Primary;
    out << YAML::Key << "Fixed Aspect Ratio" << YAML::Value << cc.FixedAspectRatio;

    out << YAML::EndMap;
}

```

2.Yaml data

```

Scene: Untitled
Entities:
- Entity: 256257383941
  TagComponent:
    Tag: Clip-Camera
  TransformComponent:
    Translation: [0, 0, 0]
    Rotation: [0, 0, 0]
    Scale: [1, 1, 1]
  CameraComponent:
    Camera:
      ProjectionType: 1
      PerspectiveFOV: 0.785398185
      PerspectiveNear: 0.00999999978
      PerspectiveFar: 1000
      OrthographicSize: 5
      OrthographicNear: -1
      OrthographicFar: 1
    Primary: false
    Fixed Aspect Ratio: false

```

3.Deserialize

```

auto cameraComponent = data["CameraComponent"];
if(cameraComponent)
{
    auto& cameraProps = cameraComponent["Camera"];
    auto& cc = entity.AddComponent<CameraComponent>();

    int projectionType = cameraProps["ProjectionType"].as<int>();
    cc.Camera.SetProjectionType((SceneCamera::ProjectionType)projectionType);
    cc.Camera.SetPerspectiveVerticalFOV(cameraProps["PerspectiveFOV"].as<float>());
    cc.Camera.SetPerspectiveNearClip(cameraProps["PerspectiveNear"].as<float>());
    cc.Camera.SetPerspectiveFarClip(cameraProps["PerspectiveFar"].as<float>());
    cc.Camera.SetOrthographicSize(cameraProps["OrthographicSize"].as<float>());
    cc.Camera.SetOrthographicNearClip(cameraProps["OrthographicNear"].as<float>());
    cc.Camera.SetOrthographicFarClip(cameraProps["OrthographicFar"].as<float>());
    // Unlike Camera, Primary is a separate key-value mapping,
    // while Camera is a map that requires further access.
    cc.Primary = cameraComponent["Primary"].as<bool>();
    cc.FixedAspectRatio = cameraComponent["Fixed Aspect Ratio"].as<bool>();
}

```

》》 There are few issues you need to know:

- 1.字符匹配: 查找value所用的key需要正确无误, 比如你想查找yaml文件中的 Fixed Aspect Ratio, 你就必须用 Fixed Aspect Ratio 作为索引, 而不是 FixedAspectRatio.
- 2.Map访问: 如果你在map中查找其中存储的元素, 你只能访问顶层的元素, 而不能访问靠底层的元素。比如 CameraComponent

```

CameraComponent: ← map
  Camera: ← element 1 (also a map)
    ProjectionType: 1
    PerspectiveFOV: 0.785398185
    PerspectiveNear: 0.00999999978
    PerspectiveFar: 1000
    OrthographicSize: 5
    OrthographicNear: -1
    OrthographicFar: 1
  Primary: false ← element 2 (key-value)

```

图例此时处于 cameraComponent, 你通过这两个 key 访问其中的 value: Camera 和 Primary (比如调用 cameraComponent["Camera"])。可是如果你想通过 CameraComponent 访问

projectionType, 就不能使用 cameraComponent["ProjectionType"] 这样的语句, 而必须使用 cameraComponent["Camera"]["ProjectionType"], 否则会报错。

》》》Cherno 将文件保存在 .hazel 后缀的文件中, 这可以吗? 为什么? 只有Hazel 才能处理这种文件吗? ?

```
+         if (ImGui::MenuItem("Serialize"))
+         {
+             SceneSerializer serializer(m_ActiveScene);
+
+             serializer.Serialize("assets/scenes/Example.hazel");
+         }
+
+         if (ImGui::MenuItem("Deserialize"))
+         {
+             SceneSerializer serializer(m_ActiveScene);
+
+             serializer.Deserialize("assets/scenes/Example.hazel");
+         }
```

这取决于你的打开方式, 现在 Hazel (或者 Nut) 有能力接受这种文件, 通过我们定义的函数, Hazel (或者 Nut) 通过文件流合理读入文件, 然后识别文件内容并且做出了相应操作。如果你使用 word 打开这种文件, 应用程序就会通过文本格式打开这个文件, 这样也是被允许的, 因为我们就是以文本的形式去设置了 yaml 文件。不过使用其他打开方式可能就会出错。

》》》》注意事项/可改进事项

@KarimInordinata 3年前

Is there any reason you've implemented this without reflection? Or is it just simplicity? For my engine I've added simple metaprogramming to allow for reflection, which means I don't have to write deserializers for every member variable, or write code to show it in the editor.

您是否有任何理由不加反思就实施了这一点? 或者只是简单? 对于我的引擎, 我添加了简单的元编程以允许反射, 这意味着我不必为每个成员变量编写反序列化器。或编写代码以在编辑器中显示它。

25 0 回复

7 条回复

@qx-jd9mh 3年前

@mattmurphy7030 "game devs" are allergic to template metaprogramming

@mattmurphy7030 "game devs"对模板元编程过敏

2 0 回复

@ipatrick6686 3年前

how?

如何?

1 0 回复

@erniegutierrez410 3年前

He doesn't have a clue

他不知道

1 0 回复

@johnjackson9767 3年前

Yes. Reflection information makes this a breeze.

是的。反射信息使这变得轻而易举。

1 0 回复

@utkarshja9547 3年前

Is there a place where I can go to learn about this?!

有没有地方可以去学习这方面的知识? !

1 0 回复

@utkarshja9547 3年前

@johnjackson9767 Is there a place I can learn about this???? I'm fairly annoyed by having to type out the statements for each member variable....

@johnjackson9767有地方可以了解这个吗? ? 我对必须为每个成员变量键入语句感到相当恼火.....

1 0 回复

@NikiGity 2年前 (修改过)

I have the same question. This seems like needless error prone work - I'm sure at least 10 times a year you're going to add a member to a class and forget to add it to the serializer, or you add it to the serializer but forget it in the deserializer. Why do it this way when you can just enumerate over the reflected fields in the class and just use their variable/member name as the key? Then you only write code once per data type, not once per member.

我有同样的问题。这似乎是不必要的容易出错的工作——我确信每年至少有 10 次你要向类中添加成员但忘记将其添加到序列化器中, 或者你将其添加到序列化器中但忘记了解串器。当您以枚举类中的反射字段并使用它们的变量/成员名称作为键时, 为什么要这样做呢? 然后, 您只需为每个数据类型编写一次代码, 而不是为每个成员编写一次代码。

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"];

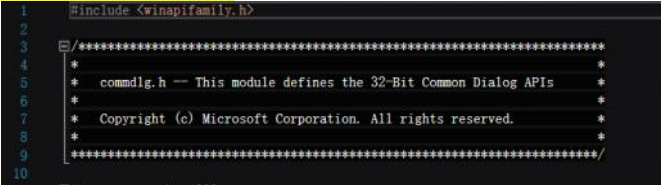
```
// 反转它以按实体的顺序添加实体
// 最初添加的
std::vector<YAML::Node> EntityRev(entitiesNode.begin(),
    实体节点.end());

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

玩得开心！

----- Save/Open file dialog -----

》》》什么是 commdlg 库？



概念： commdlg.h 是 Windows API 中的一个头文件，它用于实现标准对话框功能，如文件打开和文件保存对话框。这个头文件定义了与这些对话框相关的结构、常量和函数，使开发者能够方便地在应用程序中集成文件选择功能。

在使用 commdlg.h 时，通常会涉及到以下几个函数：

GetOpenFileName:	用于显示“打开文件”对话框。
GetSaveFileName:	用于显示“保存文件”对话框。

》》》 glfw3.h 和 glfw3native.h 这两个库之间的不同

不同： glfw3.h 和 glfw3native.h 之间的区别在于： glfw3.h 用于创建 glfw 类型的窗口， glfw3native.h 用于获取原生操作系统中的窗口的句柄，以此来进行更底层的操作。

<p>1. <GLFW/glfw3.h></p> <p>目的： 这是 GLFW 的主要头文件，提供了创建窗口、处理输入、管理 OpenGL 上下文、以及其他与窗口和输入相关的功能。</p> <p>内容： 包含了 GLFW 的所有核心功能，比如：创建和管理窗口和上下文、处理键盘、鼠标等输入事件、管理 OpenGL 扩展、定时器等功能</p>	<p>2. <GLFW/glfw3native.h></p> <p>目的： 这个头文件提供了平台特定的功能，通常用于访问底层原生窗口句柄或其他系统级别的功能。</p> <p>内容： 包含了一些函数，这些函数允许你获取与操作系统相关的窗口句柄。例如，在 Windows 系统上，你可以通过它获得 HWND 句柄；在 X11 上，你可以获得相应的 X11 窗口 ID。</p> <p>使用场景： 如果你需要与操作系统的原生 API 进行交互（例如，在窗口中嵌入第三方控件，或与其他库集成），则可能需要使用这个头文件。</p>
--	---

》》》关于 glfw3native.h 和 #define GLFW_EXPOSE_NATIVE_WIN32

```

161  B/*****
162      * Functions
163      *****/
164
165  B#if defined(GLFW_EXPOSE_NATIVE_WIN32) 非活动预处理器块
166  #endif
167
168
169
170
171
172  /*! @brief Returns the `HWND` of the specified window.
173   *
174   * @return The `HWND` of the specified window, or `NULL` if an
175   * [error](#error_handling) occurred.
176   *
177   * @errors Possible errors include @ref GLFW_NOT_INITIALIZED and @ref
178   * GLFW_PLATFORM_UNAVAILABLE.
179   *
180   * @remark The `HDC` associated with the window can be queried with the
181   * [GetDC](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getdc)
182   * function.
183   * @code
184   * HDC dc = GetDC glfwGetWin32Window(window);
185   * @endcode
186   * This DC is private and does not need to be released.
187   *
188   * @thread_safety This function may be called from any thread. Access is not
189   * synchronized.
190   *
191   * @since Added in version 3.0.
192   *
193   * @ingroup native
194   */
195  GLFWAPI HWND glfwGetWin32Window(GLFWwindow* window);
196  #endif
197

```

```
std::string FileDialogs::openFile(const char* filter)
{
    OPENFILENAMEA ofn;
    CHAR szFile[260] = { 0 };
    ZeroMemory(&ofn, sizeof(OPENFILENAME));
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = glfwGetWin32Window((GLFWwindow*)Application::Get().GetWindow().GetNativeWindow());
    ofn.lpstrFile = szFile;
    ofn.nMaxFile = sizeof(szFile);
    ofn.lpstrFilter = filter;
    ofn.nFilterIndex = 1;
    ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_NOCHANGEDIR;
    if (GetOpenFileNameA(&ofn) == TRUE)
    {
        return ofn.lpstrFile;
    }
    return std::string();
}
```

下面逐句解释代码的功能：（标蓝的函数/类型名/变量有额外笔记）

分区 GameEngine 的第 12 页

	• OFN_NOCHANGEDIR: 打开对话框时, 不更改当前工作目录。
if (GetOpenFileNameA(&ofn) == TRUE)	调用 Windows API 函数 GetOpenFileNameA 显示文件对话框。如果用户选择了一个文件, 返回值为 TRUE。
{ return ofn.lpstrFile; }	如果文件选择成功, 返回 ofn.lpstrFile 中存储的文件路径。
return std::string(); }	如果用户取消了操作或发生错误, 返回一个空的字符串。

《《关于 OPENFILENAMEA 的定义

```
typedef struct tagOFNA {  
    DWORD      lStructSize;  
    HWND       hwndOwner;  
    HINSTANCE  hInstance;  
    LPCSTR      lpstrFilter;  
    LPSTR       lpstrCustomFilter;  
    DWORD      nMaxCustFilter;  
    DWORD      nFilterIndex;  
    LPSTR       lpstrFile;  
    DWORD      nMaxFile;  
    LPSTR       lpstrFileName;  
    DWORD      nMaxFileName;  
    LPCSTR      lpstrInitialDir;  
    LPCSTR      lpstrTitle;  
    DWORD      Flags;  
    WORD       nFileOffset;  
    WORD       nFileExtension;  
    LPCSTR      lpstrDefExt;  
    LPARAM      lCustData;  
    LPOFNHOOKPROC lpfnHook;  
    LPCSTR      lpTemplateName;  
#ifdef _MAC  
    LPEDITMENU lpEditInfo;  
    LPCSTR      lpstrPrompt;  
#endif  
#if (_WIN32_WINNT >= 0x0500)  
    void *      pvReserved;  
    DWORD      dwReserved;  
    DWORD      FlagsEx;  
#endif // (_WIN32_WINNT >= 0x0500)  
} OPENFILENAMEA, *LPOPENFILENAMEA;
```

《《关于 ZeroMemory 函数的定义

在 minwinbase.h 函数中:

```
39  |#define ZeroMemory RtlZeroMemory  
20266 |define RtlZeroMemory(Destination, Length) memset((Destination), 0, (Length))
```

《《ofn.lpstrFilter = filter; 之中, filter 为什么是 const char* ? 填入的时候有什么规范?

lpstrFilter 格式

示例:

```
const char *filter = "Hazel Files (*.hazel)\0*.hazel\0All Files (*.*)\0*.*\0\0";
```

格式:

每一组文件类型描述由两部分组成 -> 描述字符串和扩展名字符串:

描述字符串是用户在对话框中看到的文件类型名称, 扩展名字符串指定了可以被选择的文件扩展名。各组之间用 \0 分隔, 最后以两个 \0 结束。

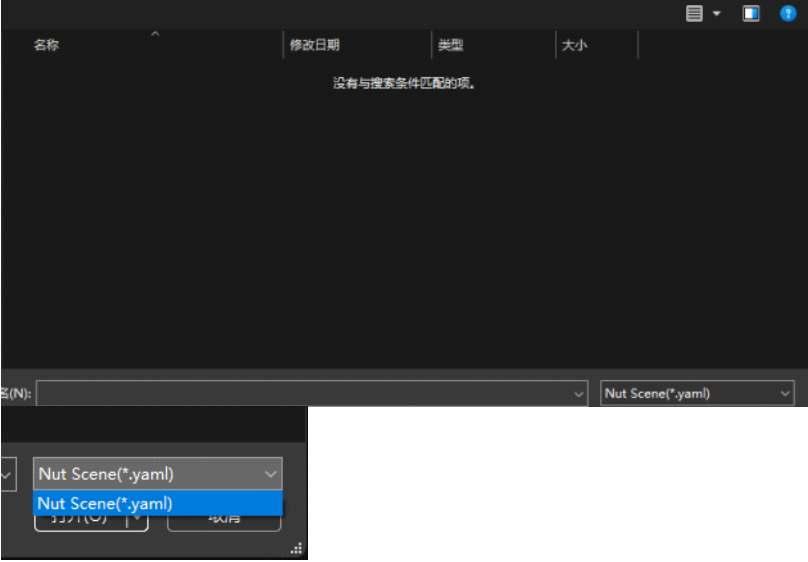
对话框如何识别和表示:

Hazel Files (*.hazel)\0*.hazel\0 -> 在文件对话框中, 用户会看到 "Hazel Files (*.hazel)" 作为文件类型的选项。当选择这个选项后, 对话框会过滤出所有以 .hazel 结尾的文件。

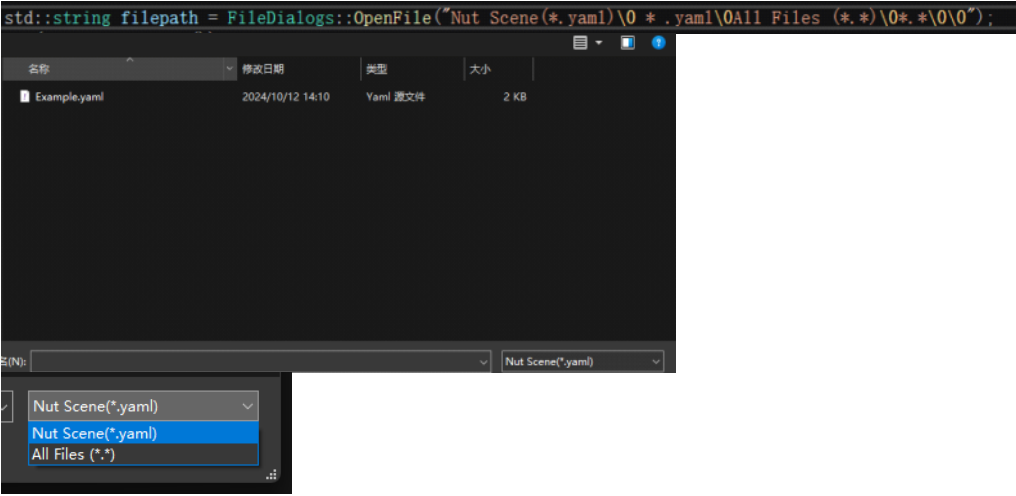
All Files (*.*)\0*.*\0 -> 如果用户选择 "所有文件 (.)", 则会显示所有文件, 包括 .hazel 文件。

例如:

```
std::string filepath = FileDialogs::OpenFile("Nut Scene(*.yaml)\0*.*\0*.hazel\0");
```

Nut Scene(*.yaml) 为显示的文本提示，通常设置为你可以选择的过滤器，通过文本提示，代码会索引到合适的过滤器。hazel 则会根据你选择的文本索引到你设置的过滤器，然后对所有文件进行过滤，如果符合 .hazel 后缀，便显示在对话框窗口中。



《《 GetOpenFileNameA 函数的作用：

GetOpenFileNameA 是 Windows API 中的一个函数，用于显示一个标准的“打开文件”对话框，让用户选择一个文件。

函数原型	BOOL GetOpenFileNameA(LPOPENFILENAMEA lpofn);
参数	lpofn: 指向 OPENFILENAMEA 结构体的指针，该结构体包含了对对话框的配置信息和用户选择的文件路径。
返回值	如果用户成功选择了一个文件并点击“确定”，函数返回非零值（通常是 TRUE）。 如果用户取消对话框或发生错误，返回值为零（FALSE）。可以通过调用 GetLastError 来获取更多错误信息。

《《 Flags 详细解释：

在 OPENFILENAME 结构体中，可以设置多个标志（Flags）来控制对话框的行为。

OFN_PATHMUSTEXIST:	含义：用户输入的路径必须存在。如果用户在对对话框中输入了一个路径（而不是从浏览器中选择），这个路径必须是有效的。 触发情况：当用户输入一个不存在的路径并尝试打开文件时，会出现错误提示，说明路径无效。
OFN_FILEMUSTEXIST:	含义：用户选择的文件必须存在。即使用户在对对话框中选择了文件，如果该文件不存在，就会阻止选择。 触发情况：当用户选择的文件实际上在磁盘上不存在时，系统会显示错误提示，说明所选文件不存在。
OFN_NOCHANGEDIR:	含义：打开对话框时不改变当前工作目录。默认情况下，打开文件对话框可能会改变程序的当前工作目录以便于访问文件。 触发情况：这个标志的作用是确保选择文件后，程序的当前工作目录保持不变，即使用户选择了不同的文件路径。

《关于 OFN_NOCHANGEDIR 的详细说明

背景：在 Windows 应用程序中，当前工作目录是指程序在文件系统中默认访问的位置。当应用程序启动时，它会有一个初始的工作目录，通常是可执行文件所在的位置。

使用场景：

当用户打开文件对话框并选择一个文件时，默认情况下，Windows 会将当前工作目录更改为用户选择的文件所在的路径。这意味着如果用户选择了一个不同位置的文件，之后程序的所有文件访问操作都会基于这个新的工作目录。

然而，在某些情况下，这种行为可能会导致问题。例如：

- 依赖于相对路径：如果程序中的文件操作使用相对路径，工作目录的变化可能会导致文件访问失败。
- 多次调用：如果你的应用程序需要频繁打开文件，改变工作目录可能会使管理变得复杂，特别是在需要回到原始路径时。

假设你有一个文本编辑器应用程序，用户可以打开和编辑多个文件。在这个程序中，用户最开始可能在 C:\Documents 中打开一个文件。
OPENFILENAME ofn; // common dialog box structure

```

char szFile[260];      // buffer for file name

// Initialize OPENFILENAME
ZeroMemory(&ofn, sizeof(ofn));
ofn.lStructSize = sizeof(ofn);
ofn.hwndOwner = hwnd;
ofn.lpstrFile = szFile;
ofn.lpstrFile[0] = '\\0';
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFilter = "Text Files\\0*.TXT\\0All Files\\0*.*\\0";
ofn.nFilterIndex = 1;
ofn.lpstrFileTitle = NULL;
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.lpstrTitle = "Open File";
ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_NOCHANGEDIR;

// Display the Open dialog box
if (GetOpenFileName(&ofn)) {
    // 用户选择了文件
    // 这里可以进行文件读取等操作
}

```

在此示例中:

如果没有设置 `OFN_NOCHANGEDIR`, 选择一个位于 `D:\OtherFiles` 的文件可能会把当前工作目录从 `C:\Documents` 改为 `D:\OtherFiles`, 这意味着接下来如果程序尝试以相对路径访问文件 (例如, 读取 `data.txt`), 它会在 `D:\OtherFiles` 查找, 而不是在用户原本的路径 `C:\Documents`.

加入 `OFN_NOCHANGEDIR` 后的效果:

通过加入 `OFN_NOCHANGEDIR`, 即使用户选择了位于不同文件夹的文件, 程序的当前工作目录仍然保持在 `C:\Documents`。这样, 任何依赖于该目录的文件操作都不会受到影响。

》》》问题: 触发断点 -> 为什么将 `CreateRef` 改为 `Ref` 时会触发断点异常?

```

void EditorLayer::OpenScene()
{
    std::string filepath = FileDialogs::OpenFile("Nut Scene(*.yaml)\\0 *.yaml\\0All Files (*.*)\\0*.*\\0\\0");
    if (!filepath.empty())
    {
        m_ActiveScene = CreateRef<Scene>();
        m_ActiveScene->OnViewportResize((uint32_t)m_ViewportSize.x, (uint32_t)m_ViewportSize.y); // We use
        m_SceneHierarchyPanel.SetContext(m_ActiveScene); // We use

        SceneSerializer serializer(m_ActiveScene);
        serializer.Deserialize(filepath);
    }
}

```

因为 `Ref (std::shared_ptr<T>())` 创建了一个智能指针, 但未分配内存;

而 `CreateRef (std::make_shared<T>())` 创建并初始化一个智能指针, 同时分配内存并构造对象。

》》》关于快捷键触发这个事件函数的设计:

```

bool EditorLayer::OnKeyPressed(KeyPressedEvent& event)
{
    // You can triggering this event only once,
    // because if you try to trigger this event again,
    // the event.GetRepeatCount will bigger than 0.
    // And function will return false
    if (event.GetRepeatCount() > 0)
        return false;

    bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
    bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
    switch (event.GetKeyCode())
    {
        case NUT_KEY_N: {
            if (ctrl)
                NewScene();

            break;
        }
        case NUT_KEY_O: {
            if (ctrl)
                OpenScene();

            break;
        }
        case NUT_KEY_S: {
            if (ctrl && shift)
                SaveSceneAs();

            break;
        }
    }
}

```

1. GetRepeat() 在这里的作用：防止处理重复按键事件。

e.GetRepeatCount() 函数用于获取按键的重复次数。当一个按键被按下并保持时（比如长按），操作系统会不断生成按键按下事件，这样会导致该事件被多次触发。所以当检测到重复按键时，函数直接返回 false，意味着后续的代码（如处理快捷键的逻辑）将不会被执行。这可以防止同一个快捷键被多次触发，从而避免造成意外的重复操作。

2. 这个函数为什么只在事件重复时返回一个布尔量，其他条件下不返回值呢。为什么 GetRepeat() 需要设置在函数最前面？

在事件触发之后，我们运行实际逻辑代码（OpenScene/NewScene），但是我们不能返回 true，因为 true 不会阻止事件运行，这将会导致事件会被连续触发。其次，如果在逻辑代码之后返回 false，而不是在函数开头返回 false，都会导致事件被触发第二次，因为在判断是否重复触发时依旧会先运行逻辑代码，随后判断出来结果是 false（应该阻塞），不过此时已经没有用了。

```

bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
switch (event.GetKeyCode())
{
    case NUT_KEY_N: {
        if (ctrl)
            NewScene();

        break;
    }
    case NUT_KEY_O: {
        if (ctrl)
            OpenScene();

        break;
    }
    case NUT_KEY_S: {
        if (ctrl && shift)
            SaveSceneAs();

        break;
    }
}

if (event.GetRepeatCount() > 0)
    return false;
return true;

```

第一次触发事件

不满足，故返回 true，不阻塞

```

bool ctrl = Input::IsKeyPressed(NUT_KEY_LEFT_CONTROL) || Input::IsKeyPressed(NUT_KEY_RIGHT_CONTROL);
bool shift = Input::IsKeyPressed(NUT_KEY_LEFT_SHIFT) || Input::IsKeyPressed(NUT_KEY_RIGHT_SHIFT);
switch (event.GetKeyCode())
{
    case NUT_KEY_N: {
        if (ctrl)
            NewScene();

        break;
    }
    case NUT_KEY_O: {
        if (ctrl)
            OpenScene();

        break;
    }
    case NUT_KEY_S: {
        if (ctrl && shift)
            SaveSceneAs();

        break;
    }
}

if (event.GetRepeatCount() > 0)
    return false;
return true;

```

第二次触发事件（重复触发）

会再次处理

条件满足，返回 false 表示阻塞（但此时运行了两次业务）

3. GetKeyCode 和 GetRepeatCount 中返回的 keycode 和 count 是在哪里自动获取的，所以我们才能使用其返回值来进行判断

在之前设置的回调函数中：WindowsWindow.cpp 中

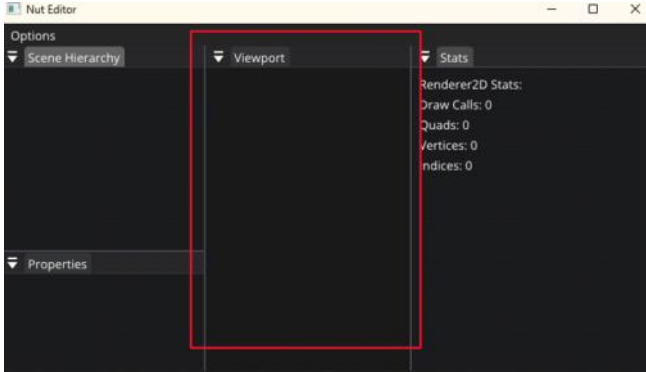
```

glfwSetKeyCallback(m_Window, [](GLFWwindow* window, int key, int scancode, int action, int mods)
{
    WindowData& data = *(WindowData*)glfwGetWindowUserPointer(window);

    switch (action)
    {
        case GLFW_PRESS: KeyPressedEvent event(key, 0); data.EventCallback(event); break; }
        case GLFW_RELEASE: KeyReleasedEvent event(key); data.EventCallback(event); break; }
        case GLFW_REPEAT: KeyPressedEvent event(key, 1); data.EventCallback(event); break; }
    }
});

```

4.为什么只有在鼠标单击了视口这个区域时（聚焦在 Viewport 窗口时），快捷键才能使用？



因为是在 EditorLayer::OnEvent() 函数中设置的事件分发。

```

void EditorLayer::OnEvent(Event& event)
{
    m_CameraController.OnEvent(event);

    EventDispatcher dispatcher(event);
    dispatcher.Dispatch<KeyPressedEvent>(NUT_BIND_EVENT_FN(EditorLayer::OnKeyPressed));
}

```

----- ImGuizmo -----

》》》这段 premake 代码什么意思？

```

57
58     filter "files:vendor/ImGuizmo/**/*.cpp"
59     flags {"NoPCH"}
60

```

》》》为什么Nut-editor 中没有包含 ImGui 库目录却可以使用 ImGui，但是没有包含 yaml-cpp 库目录的时候却不能使用 yaml-cpp？

```

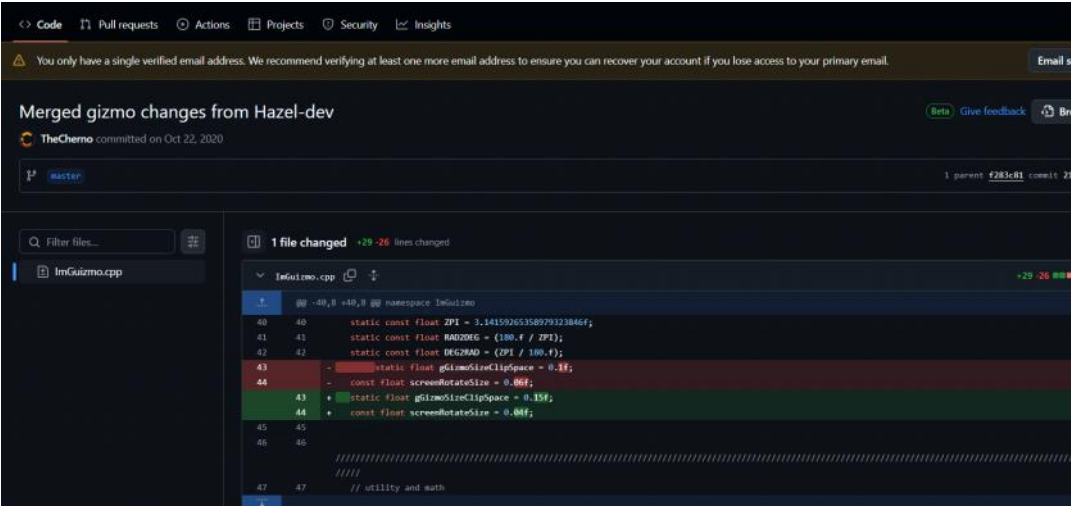
includedirs
{
    "%{wks.location}/Nut/vendor/spdlog/include",
    "%{wks.location}/Nut/src",
    "%{wks.location}/Nut/vendor",
    "%{IncludeDir.glm}",
    "%{IncludeDir.entt}",
    "%{IncludeDir.yaml_cpp}",
    "%{IncludeDir.ImGuizmo}"
}

```

因为实际上 ImGui 在 "%{wks.location}/Nut/vendor" 中已经被包含了。

》》》代码设计：gizmo库中对于 ImGuizmo.cpp 的更改：（<https://github.com/TheCherno/ImGuizmo/commit/218d60bde7d22061ac525d0d71e05360b4dcf978>）

我猜想Cherno做的是一些对于 ImGuizmo 样式的更改，并且由于时间原因，最新的 ImGuizmo.cpp 结构发生了很多改变，这让我们不容易同步这个更改。所以我决定先保持默认值，后续再查看是否需要同步此更改。



》》》什么是万向锁，什么情况下会触发万向锁？什么是四元数，为什么四元数可以解决万向锁？glm/gtx/quaternion.hpp 中处理四元数的函数怎么使用？

万向锁 (Gimbal Lock)

万向锁是一种在三维空间中旋转时遇到的现象，通常发生在使用欧拉角表示旋转的情况下。它指的是在某些特定的旋转配置下，系统的自由度减少，使得无法进行预期的旋转。

万向锁通常在以下情况下发生：

旋转轴对齐：	当一个旋转轴与另一个旋转轴对齐时，例如在俯仰角为 ±90° 时，两个旋转轴重合，这会导致系统失去一个自由度。
极限位置：	当物体达到某个极限位置（如直立或水平），就可能会导致无法实现某些方向的旋转。

万向锁的影响：

当发生万向锁时，物体可能会无法按预期方向旋转，或者某些旋转会变得不可用。这在动画、飞行控制和机器人运动等应用中会造成问题。

参考文档：（ <https://medium.com/@lalesena/euler-angles-rotations-and-gimbal-lock-brief-explanation-de1d4764170> ）

参考图片：

https://miro.medium.com/v2/resize:fit:498/format:webp/1*7JT7g5dLeZ-Q5uOYnX8hCw.gif
https://miro.medium.com/v2/resize:fit:400/format:webp/1*OckqKWmTtmDtzrukAX4hSA.gif

四元数 (Quaternion) 是一种扩展了复数的数学结构，通常用于表示三维空间中的旋转。它由一个实数部分和三个虚数部分组成，形式为：

$q = w + xi + yj + zk$

其中：w 是实数部分，x,y,z 是虚数部分（向量部分），i,j,k 是虚单位。

四元数通过以下方式解决万向锁问题：

四维表示：	四元数使用四个参数（一个实数和三个虚数）来表示旋转，而不是依赖于三个独立的角度（如俯仰、偏航和滚转）。这种表示方法避免了两条旋转轴重合的情况。
组合旋转：	四元数之间的乘法可以直接组合多个旋转，而不受单一旋转轴限制。这使得在任何方向上进行旋转都不会遭遇自由度的丧失。
插值平滑：	四元数支持球形线性插值 (Slerp)，这使得在动画中能够平滑地过渡旋转，从而避免因插值引起的万向锁问题。

参考文档---涉及到数理知识：（ <https://www.cnblogs.com/HDDDDDD/p/15067619.html> ）

》》》代码设计：TransformComponent 中的更改：

这可以防止通过 imGuizmo 调整物体旋转时图像极速发生变化的问题（通过 ImGuizmo 可视化工具轻轻更改欧拉角，角度却会极速增加，这会导致图像迅速旋转并闪烁）并且防止可能出现的万向锁问题（可以查看上述的：[万向锁 \(Gimbal Lock\)](#)）。


```
Hazel/src/Hazel/Scene/Components.h
1  @@ -3,6 +3,9 @@
2
3  3  #include <glm/glm.hpp>
4  4  #include <glm/gtc/matrix_transform.hpp>
5  5
6  6  + #define GLM_ENABLE_EXPERIMENTAL
7  7  + #include <glm/gtx/quaternion.hpp>
8  8  +
9  9  #include "SceneCamera.h"
10 10 #include "ScriptableEntity.h"
11 11
12
13 @@ -31,9 +34,7 @@ namespace Hazel {
14
15 31 34
16 32 35         glm::mat4 GetTransform() const
17 33 36         {
18 34 -         glm::mat4 rotation = glm::rotate(glm::mat4(1.0f), Rotation.x, { 1, 0, 0 })
19 35 -         * glm::rotate(glm::mat4(1.0f), Rotation.y, { 0, 1, 0 })
20 36 -         * glm::rotate(glm::mat4(1.0f), Rotation.z, { 0, 0, 1 });
21 37 +         glm::mat4 rotation = glm::toMat4(glm::quat(Rotation));
22 37 38
23 38 39         return glm::translate(glm::mat4(1.0f), Translation)
24 39 40         * rotation
25 40
26 41
27 42
28 43
29 44
30 45
31 46
32 47
33 48
34 49
35 50
36 51
37 52
38 53
39 54
40 55
41 56
42 57
43 58
44 59
45 60
46 61
47 62
48 63
49 64
50 65
51 66
52 67
53 68
54 69
55 70
56 71
57 72
58 73
59 74
60 75
61 76
62 77
63 78
64 79
65 80
66 81
67 82
68 83
69 84
70 85
71 86
72 87
73 88
74 89
75 90
76 91
77 92
78 93
79 94
80 95
81 96
82 97
83 98
84 99
85 100
86 101
87 102
88 103
89 104
90 105
91 106
92 107
93 108
94 109
95 110
96 111
97 112
98 113
99 114
100 115
101 116
102 117
103 118
104 119
105 120
106 121
107 122
108 123
109 124
110 125
111 126
112 127
113 128
114 129
115 130
116 131
117 132
118 133
119 134
120 135
121 136
122 137
123 138
124 139
125 140
126 141
127 142
128 143
129 144
130 145
131 146
132 147
133 148
134 149
135 150
136 151
137 152
138 153
139 154
140 155
141 156
142 157
143 158
144 159
145 160
146 161
147 162
148 163
149 164
150 165
151 166
152 167
153 168
154 169
155 170
156 171
157 172
158 173
159 174
160 175
161 176
162 177
163 178
164 179
165 180
166 181
167 182
168 183
169 184
170 185
171 186
172 187
173 188
174 189
175 190
176 191
177 192
178 193
179 194
180 195
181 196
182 197
183 198
184 199
185 200
186 201
187 202
188 203
189 204
190 205
191 206
192 207
193 208
194 209
195 210
196 211
197 212
198 213
199 214
200 215
201 216
202 217
203 218
204 219
205 220
206 221
207 222
208 223
209 224
210 225
211 226
212 227
213 228
214 229
215 230
216 231
217 232
218 233
219 234
220 235
221 236
222 237
223 238
224 239
225 240
226 241
227 242
228 243
229 244
230 245
231 246
232 247
233 248
234 249
235 250
236 251
237 252
238 253
239 254
240 255
241 256
242 257
243 258
244 259
245 260
246 261
247 262
248 263
249 264
250 265
251 266
252 267
253 268
254 269
255 270
256 271
257 272
258 273
259 274
260 275
261 276
262 277
263 278
264 279
265 280
266 281
267 282
268 283
269 284
270 285
271 286
272 287
273 288
274 289
275 290
276 291
277 292
278 293
279 294
280 295
281 296
282 297
283 298
284 299
285 300
286 301
287 302
288 303
289 304
290 305
291 306
292 307
293 308
294 309
295 310
296 311
297 312
298 313
299 314
300 315
301 316
302 317
303 318
304 319
305 320
306 321
307 322
308 323
309 324
310 325
311 326
312 327
313 328
314 329
315 330
316 331
317 332
318 333
319 334
320 335
321 336
322 337
323 338
324 339
325 340
326 341
327 342
328 343
329 344
330 345
331 346
332 347
333 348
334 349
335 350
336 351
337 352
338 353
339 354
340 355
341 356
342 357
343 358
344 359
345 360
346 361
347 362
348 363
349 364
350 365
351 366
352 367
353 368
354 369
355 370
356 371
357 372
358 373
359 374
360 375
361 376
362 377
363 378
364 379
365 380
366 381
367 382
368 383
369 384
370 385
371 386
372 387
373 388
374 389
375 390
376 391
377 392
378 393
379 394
380 395
381 396
382 397
383 398
384 399
385 400
386 401
387 402
388 403
389 404
390 405
391 406
392 407
393 408
394 409
395 410
396 411
397 412
398 413
399 414
400 415
401 416
402 417
403 418
404 419
405 420
406 421
407 422
408 423
409 424
410 425
411 426
412 427
413 428
414 429
415 430
416 431
417 432
418 433
419 434
420 435
421 436
422 437
423 438
424 439
425 440
426 441
427 442
428 443
429 444
430 445
431 446
432 447
433 448
434 449
435 450
436 451
437 452
438 453
439 454
440 455
441 456
442 457
443 458
444 459
445 460
446 461
447 462
448 463
449 464
450 465
451 466
452 467
453 468
454 469
455 470
456 471
457 472
458 473
459 474
460 475
461 476
462 477
463 478
464 479
465 480
466 481
467 482
468 483
469 484
470 485
471 486
472 487
473 488
474 489
475 490
476 491
477 492
478 493
479 494
480 495
481 496
482 497
483 498
484 499
485 500
486 501
487 502
488 503
489 504
490 505
491 506
492 507
493 508
494 509
495 510
496 511
497 512
498 513
499 514
500 515
501 516
502 517
503 518
504 519
505 520
506 521
507 522
508 523
509 524
510 525
511 526
512 527
513 528
514 529
515 530
516 531
517 532
518 533
519 534
520 535
521 536
522 537
523 538
524 539
525 540
526 541
527 542
528 543
529 544
530 545
531 546
532 547
533 548
534 549
535 550
536 551
537 552
538 553
539 554
540 555
541 556
542 557
543 558
544 559
545 560
546 561
547 562
548 563
549 564
550 565
551 566
552 567
553 568
554 569
555 570
556 571
557 572
558 573
559 574
560 575
561 576
562 577
563 578
564 579
565 580
566 581
567 582
568 583
569 584
570 585
571 586
572 587
573 588
574 589
575 590
576 591
577 592
578 593
579 594
580 595
581 596
582 597
583 598
584 599
585 600
586 601
587 602
588 603
589 604
590 605
591 606
592 607
593 608
594 609
595 610
596 611
597 612
598 613
599 614
600 615
601 616
602 617
603 618
604 619
605 620
606 621
607 622
608 623
609 624
610 625
611 626
612 627
613 628
614 629
615 630
616 631
617 632
618 633
619 634
620 635
621 636
622 637
623 638
624 639
625 640
626 641
627 642
628 643
629 644
630 645
631 646
632 647
633 648
634 649
635 650
636 651
637 652
638 653
639 654
640 655
641 656
642 657
643 658
644 659
645 660
646 661
647 662
648 663
649 664
650 665
651 666
652 667
653 668
654 669
655 670
656 671
657 672
658 673
659 674
660 675
661 676
662 677
663 678
664 679
665 680
666 681
667 682
668 683
669 684
670 685
671 686
672 687
673 688
674 689
675 690
676 691
677 692
678 693
679 694
680 695
681 696
682 697
683 698
684 699
685 700
686 701
687 702
688 703
689 704
690 705
691 706
692 707
693 708
694 709
695 710
696 711
697 712
698 713
699 714
700 715
701 716
702 717
703 718
704 719
705 720
706 721
707 722
708 723
709 724
710 725
711 726
712 727
713 728
714 729
715 730
716 731
717 732
718 733
719 734
720 735
721 736
722 737
723 738
724 739
725 740
726 741
727 742
728 743
729 744
730 745
731 746
732 747
733 748
734 749
735 750
736 751
737 752
738 753
739 754
740 755
741 756
742 757
743 758
744 759
745 760
746 761
747 762
748 763
749 764
750 765
751 766
752 767
753 768
754 769
755 770
756 771
757 772
758 773
759 774
760 775
761 776
762 777
763 778
764 779
765 780
766 781
767 782
768 783
769 784
770 785
771 786
772 787
773 788
774 789
775 790
776 791
777 792
778 793
779 794
780 795
781 796
782 797
783 798
784 799
785 800
786 801
787 802
788 803
789 804
790 805
791 806
792 807
793 808
794 809
795 810
796 811
797 812
798 813
799 814
800 815
801 816
802 817
803 818
804 819
805 820
806 821
807 822
808 823
809 824
810 825
811 826
812 827
813 828
814 829
815 830
816 831
817 832
818 833
819 834
820 835
821 836
822 837
823 838
824 839
825 840
826 841
827 842
828 843
829 844
830 845
831 846
832 847
833 848
834 849
835 850
836 851
837 852
838 853
839 854
840 855
841 856
842 857
843 858
844 859
845 860
846 861
847 862
848 863
849 864
850 865
851 866
852 867
853 868
854 869
855 870
856 871
857 872
858 873
859 874
860 875
861 876
862 877
863 878
864 879
865 880
866 881
867 882
868 883
869 884
870 885
871 886
872 887
873 888
874 889
875 890
876 891
877 892
878 893
879 894
880 895
881 896
882 897
883 898
884 899
885 900
886 901
887 902
888 903
889 904
890 905
891 906
892 907
893 908
894 909
895 910
896 911
897 912
898 913
899 914
900 915
901 916
902 917
903 918
904 919
905 920
906 921
907 922
908 923
909 924
910 925
911 926
912 927
913 928
914 929
915 930
916 931
917 932
918 933
919 934
920 935
921 936
922 937
923 938
924 939
925 940
926 941
927 942
928 943
929 944
930 945
931 946
932 947
933 948
934 949
935 950
936 951
937 952
938 953
939 954
940 955
941 956
942 957
943 958
944 959
945 960
946 961
947 962
948 963
949 964
950 965
951 966
952 967
953 968
954 969
955 970
956 971
957 972
958 973
959 974
960 975
961 976
962 977
963 978
964 979
965 980
966 981
967 982
968 983
969 984
970 985
971 986
972 987
973 988
974 989
975 990
976 991
977 992
978 993
979 994
980 995
981 996
982 997
983 998
984 999
985 1000
986 1001
987 1002
988 1003
989 1004
990 1005
991 1006
992 1007
993 1008
994 1009
995 1010
996 1011
997 1012
998 1013
999 1014
1000 1015
1001 1016
1002 1017
1003 1018
1004 1019
1005 1020
1006 1021
1007 1022
1008 1023
1009 1024
1010 1025
1011 1026
1012 1027
1013 1028
1014 1029
1015 1030
1016 1031
1017 1032
1018 1033
1019 1034
1020 1035
1021 1036
1022 1037
1023 1038
1024 1039
1025 1040
1026 1041
1027 1042
1028 1043
1029 1044
1030 1045
1031 1046
1032 1047
1033 1048
1034 1049
1035 1050
1036 1051
1037 1052
1038 1053
1039 1054
1040 1055
1041 1056
1042 1057
1043 1058
1044 1059
1045 1060
1046 1061
1047 1062
1048 1063
1049 1064
1050 1065
1051 1066
1052 1067
1053 1068
1054 1069
1055 1070
1056 1071
1057 1072
1058 1073
1059 1074
1060 1075
1061 1076
1062 1077
1063 1078
1064 1079
1065 1080
1066 1081
1067 1082
1068 1083
1069 1084
1070 1085
1071 1086
1072 1087
1073 1088
1074 1089
1075 1090
1076 1091
1077 1092
1078 1093
1079 1094
1080 1095
1081 1096
1082 1097
1083 1098
1084 1099
1085 1100
1086 1101
1087 1102
1088 1103
1089 1104
1090 1105
1091 1106
1092 1107
1093 1108
1094 1109
1095 1110
1096 1111
1097 1112
1098 1113
1099 1114
1100 1115
1101 1116
1102 1117
1103 1118
1104 1119
1105 1120
1106 1121
1107 1122
1108 1123
1109 1124
1110 1125
1111 1126
1112 1127
1113 1128
1114 1129
1115 1130
1116 1131
1117 1132
1118 1133
1119 1134
1120 1135
1121 1136
1122 1137
1123 1138
1124 1139
1125 1140
1126 1141
1127 1142
1128 1143
1129 1144
1130 1145
1131 1146
1132 1147
1133 1148
1134 1149
1135 1150
1136 1151
1137 1152
1138 1153
1139 1154
1140 1155
1141 1156
1142 1157
1143 1158
1144 1159
1145 1160
1146 1161
1147 1162
1148 1163
1149 1164
1150 1165
1151 1166
1152 1167
1153 1168
1154 1169
1155 1170
1156 1171
1157 1172
1158 1173
1159 1174
1160 1175
1161 1176
1162 1177
1163 1178
1164 1179
1165 1180
1166 1181
1167 1182
1168 1183
1169 1184
1170 1185
1171 1186
1172 1187
1173 1188
1174 1189
1175 1190
1176 1191
1177 1192
1178 1193
1179 1194
1180 1195
1181 1196
1182 1197
1183 1198
1184 1199
1185 1200
1186 1201
1187 1202
1188 1203
1189 1204
1190 1205
1191 1206
1192 1207
1193 1208
1194 1209
1195 1210
1196 1211
1197 1212
1198 1213
1199 1214
1200 1215
1201 1216
1202 1217
1203 1218
1204 1219
1205 1220
1206 1221
1207 1222
1208 1223
1209 1224
1210 1225
1211 1226
1212 1227
1213 1228
1214 1229
1215 1230
1216 1231
1217 1232
1218 1233
1219 1234
1220 1235
1221 1236
1222 1237
1223 1238
1224 1239
1225 1240
1226 1241
1227 1242
1228 1243
1229 1244
1230 1245
1231 1246
1232 1247
1233 1248
1234 1249
1235 1250
1236 1251
1237 1252
1238 1253
1239 1254
1240 1255
1241 1256
1242 1257
1243 1258
1244 1259
1245 1260
1246 1261
1247 1262
1248 1263
1249 1264
1250 1265
1251 1266
1252 1267
1253 1268
1254 1269
1255 1270
1256 1271
1257 1272
1258 1273
1259 1274
1260 1275
1261 1276
1262 1277
1263 1278
1264 1279
1265 1280
1266 1281
1267 1282
1268 1283
1269 1284
1270 1285
1271 1286
1272 1287
1273 1288
1274 1289
1275 1290
1276 1291
1277 1292
1278 1293
1279 1294
1280 1295
1281 1296
1282 1297
1283 1298
1284 1299
1285 1300
1286 1301
1287 1302
1288 1303
1289 1304
1290 1305
1291 1306
1292 1307
1293 1308
1294 1309
1295 1310
1296 1311
1297 1312
1298 1313
1299 1314
1300 1315
1301 1316
1302 1317
1303 1318
1304 1319
1305 1320
1306 1321
1307 1322
1308 1323
1309 1324
1310 1325
1311 1326
1312 1327
1313 1328
1314 1329
1315 1330
1316 1331
1317 1332
1318 1333
1319 1334
1320 1335
1321 1336
1322 1337
1323 1338
1324 1339
1325 1340
1326 1341
1327 1342
1328 1343
1329 1344
1330 1345
1331 1346
1332 1347
1333 1348
1334 1349
1335 1350
1336 1351
1337 1352
1338 1353
1339 1354
1340 1355
1341 1356
1342 1357
1343 1358
1344 1359
1345 1360
1346 1361
1347 1362
1348 1363
1349 1364
1350 1365
1351 1366
1352 1367
1353 1368
1354 1369
1355 1370
1356 1371
1357 1372
1358 1373
1359 1374
1360 1375
1361 1376
1362 1377
1363 1378
1364 1379
1365 1380
1366 1381
1367 1382
1368 1383
1369 1384
1370 1385
1371 1386
1372 1387
1373 1388
1374 1389
1375 1390
1376 1391
1377 1392
1378 1393
1379 1394
1380 1395
1381 1396
1382 1397
1383 1398
1384 1399
1385 1400
1386 1401
1387 1402
1388 1403
1389 1404
1390 1405
1391 1406
1392 1407
1393 1408
1394 1409
1395 1410
1396 1411
1397 1412
1398 1413
1399 1414
1400 1415
1401 1416
1402 1417
1403 1418
1404 1419
1405 1420
1406 1421
1407 1422
1408 1423
1409 1424
1410 1425
1411 1426
1412 1427
1413 1428
1414 1429
1415 1430
1416 1431
1417 1432
1418 1433
1419 1434
1420 1435
1421 1436
1422 1437
1423 1438
1424 1439
1425 1440
1426 1441
1427 1442
1428 1443
```

```
const float* snap = nullptr, const float* pivot = nullptr);
```

参数解析:

view:	相机的视图矩阵（cameraView），表示相机的位置 and 方向。
projection:	相机的投影矩阵（cameraProjection），用于确定场景中物体的透视效果。
operation:	变换操作类型（平移、旋转或缩放），通过 (ImGui::OPERATION)m_GizmoType 指定。
mode:	变换模式，通常是 ImGui::LOCAL（局部变换）或 ImGui::GLOBAL（全局变换）。
matrix:	物体的变换矩阵（transform），表示物体在场景中的位置、旋转和缩放。
deltaMatrix:	可选参数，表示变化的矩阵。
snap:	可选参数，指定对变换进行捕捉的值（例如，步长）。如果为 nullptr，则不进行捕捉。
pivot:	可选参数，指定旋转的中心点。

》》》ImGui::OPERATION 的定义

```
// call it when you want a gizmo
// Needs view and projection matrices.
// matrix parameter is the source matrix (where will be gizmo be drawn) and might be transformed by the function. Return deltaMatrix is optional
// translation is applied in world space
enum OPERATION
{
    TRANSLATE_X = (1u << 0),
    TRANSLATE_Y = (1u << 1),
    TRANSLATE_Z = (1u << 2),
    ROTATE_X    = (1u << 3),
    ROTATE_Y    = (1u << 4),
    ROTATE_Z    = (1u << 5),
    ROTATE_SCREEN = (1u << 6),
    SCALE_X     = (1u << 7),
    SCALE_Y     = (1u << 8),
    SCALE_Z     = (1u << 9),
    BOUNDS      = (1u << 10),
    SCALE_XU    = (1u << 11),
    SCALE_YU    = (1u << 12),
    SCALE_ZU    = (1u << 13),

    TRANSLATE = TRANSLATE_X | TRANSLATE_Y | TRANSLATE_Z,
    ROTATE    = ROTATE_X | ROTATE_Y | ROTATE_Z | ROTATE_SCREEN,
    SCALE     = SCALE_X | SCALE_Y | SCALE_Z,
    SCALEU    = SCALE_XU | SCALE_YU | SCALE_ZU, // universal
    UNIVERSAL = TRANSLATE | ROTATE | SCALEU
};
```

》》》代码设计：窗口响应

```
216 - Application::Get().GetImGuiLayer()->BlockEvents(!m_ViewportFocused || !m_ViewportHovered);
220 + Application::Get().GetImGuiLayer()->BlockEvents(!m_ViewportFocused && !m_ViewportHovered);
```

更改前：只要不满足 聚焦于窗口上/悬停在窗口上的任——一个条件，便会阻塞当前窗口中的事件（不再捕获鼠标或键盘的活动）

更改后：只有 窗口没有被聚焦且鼠标没有悬停在窗口上的时候，才会阻塞事件。

这让我们在结构面板中选中实体之后，只需要将鼠标悬停在 viewport 窗口上，便可以通过键盘调整/响应 Gizmo，这在实际使用中很舒服（本人亲测：>

》》》为什么这里需要使用 Const 标识?

```
Entity Scene::GetPrimaryCamera()
{
    auto view = m_Registry.view<CameraComponent>();
    for (auto entity : view)
    {
        const auto& cameraComponent = m_Registry.get<CameraComponent>(entity);
        if (cameraComponent.Primary == true)
            return Entity{ entity, this };
    }
    return {};
}
```

可能是因为没有需要对 cameraComponent 进行更改吧。

》》》这个函数的意义?

```
#pragma once

#include <glm/glm.hpp>

namespace Nut { namespace Math {
    bool DecomposeTransform(const glm::mat4& transform, glm::vec3& outTranslation, glm::vec3& outRotation, glm::vec3& outScale);
}}

```

》》》这段代码的意义？

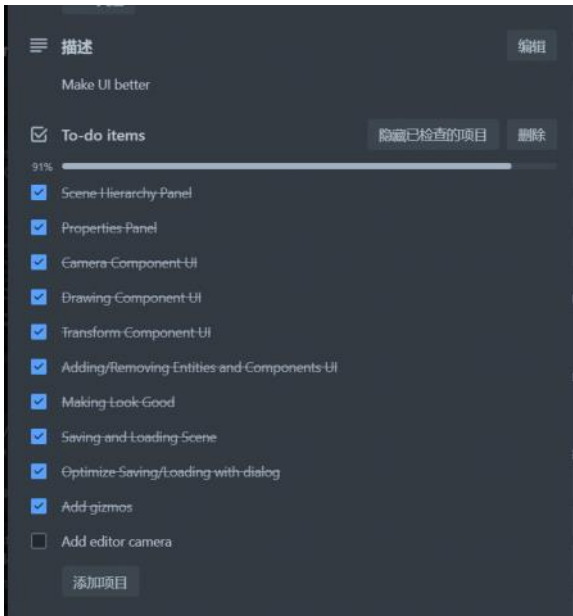
```
if (ImGui::IsUsing())
{
    glm::vec3 translation, rotation, scale;
    Math::DecomposeTransform(transform, translation, rotation, scale);
    glm::vec3 deltaRotation = rotation - tc.Rotation;
    tc.Translation = translation;
    tc.Rotation += deltaRotation;
    tc.Scale = scale;
}

```

----- Editor camera -----

》》》关于引擎的操作界面和简单操纵，我们已经完成了很多，Cherno 在接下来将开始实现鼠标选择实体，不过我可能会将之前遗漏的维护补充一下。

》》》总之先让我们开始这一集吧。



》》》这一集中，主要围绕 编辑时摄像机 和 运行时摄像机 两个概念来设计。

我目前是这样理解的：

```
void OnUpdate(Timestep ts);
void OnUpdateRuntime(Timestep ts);
void OnUpdateEditor(Timestep ts, EditorCamera& camera);
void OnViewportResize(uint32_t width, uint32_t height);

```

- 编辑时摄像机：用于在引擎中编辑物体，此摄像机为默认存在。只要你需要对物体进行编辑，那么直接建立一个 sprite 实体，在编辑器中便应该能够直接查看和编辑该实体，而不需要额外添加摄像机（运行时摄像机），这符合游戏引擎的使用逻辑。
- 运行时摄像机：用于在实际游戏程序运行时添加，此摄像机需要手动添加。游戏中不需要“上帝视角”的编辑时摄像机，而只需要用来观察物体的运行时摄像机，该摄像机不像编辑时摄像机一样默认存在，因为编辑器和游戏程序的逻辑是不同的。

这两个摄像机不是同时存在的：在编辑器中，底层默认存在一个编辑时摄像机，该相机只在编辑器运行时存在并更新；同样的，游戏程序中不使用编辑时摄像机，而是用运行时摄像机，该相机只在游戏程序运行时存在并更新。

由于我们现在在编辑器中设计程序，所以只需要对所有实体使用编辑时摄像机即可。对于游戏，我想我们应该需要在编辑器中添加一个游戏运行时摄像机，然后打开游戏程序，进行游玩。

----- 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。

----- Maintenace -----

》》》》 后续会整合 Jul/22 2022 之后的维护