红色字体：重要信息

黄色背景字体：写的时候忘记了

蓝色背景字体：我采用了不一样的写法

绿色字体：表示已经完成

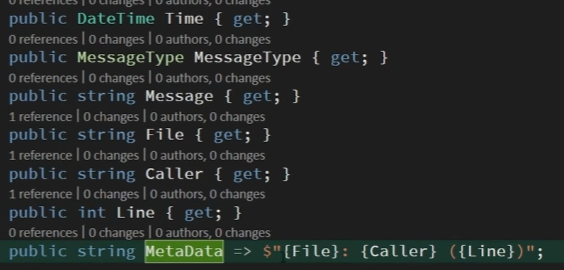
蓝色字体：我的回答和思路

# Game Engine Programming 006 - Implement game entity and components in Primal editor| C++ Game Engine

1. 添加一个SaveCommand，Ctrl+s保存；
2. 把Keybinding添加到WorldEditorControl上，你需要保证主窗口初始化后，WorldEditorControl默认为Focus的；
3. 测试一下保存项目文件的功能；
4. 修复执行UndoRedo后，WorldEditorControl lose focus的问题，我们的策略是，如果检测到Project.UndoRedoManager.UndoList发生了任何改变，就要重新设置Focus；
5. 创建Components/GameEntity.cs,Componet.cs,Transform.cs，一个Scene有一个GameEntity的Collection，一个GameEntity有一个Component集合；
6. 添加一个按钮用于向Scene添加一个GameEntity，注意按钮的ICommand的初始化逻辑，想一想应该写在哪里，这个Command需要一个GameEntity类型的参数；
7. 用ListBox来展示一个Scene中的所有GameEntity，只有Scene为Active才能添加GameEntity，ListBox才展示；
8. 我们希望能在Scene中多选GameEntity；
9. .primal文件中没有GameEntities字段，你需要为某个项目写入GameEntities列表字段，然后更新所有模板里的.primal文件；
10. 接下来为GameEntity添加一个Transformer组件，保证每一个GameEntity在被创建后都有这个组件；
11. 多态的serialization和deserialization需要特殊处理，需要让Component里的Collection知道有哪些type，我们需要KownType属性；
12. 创建GameEntityControl（UserControl），它的DataContext是GameEntity，随着用户的选择变化的，所以暂时为null。这个Control提供单例的接口，因为我们将在ProjectLayoutControl中访问它；
13. 当ProjectControl中用户选择的GameEntity改变，那么GameEntityControl的DataContext随之改变；
14. 在GameEntityControl中显示GameEntity的名字，以及一些ItemsPanel，他们是StackPanel；
15. 将GameEntityControl放到WorldEditorControl里面去；

# Game Engine Programming 007 - Undoable selections and a message log | C++ Game Engine

1. 在选择Entity这个过程，我们也希望能加上撤回重做的功能，使得我们能回退到上一个选择。你需要添加UndoRedoAction，每当ProjectLayoutControl中的GameEntity ListBox选择发生变化，我们记录下这个UndoRedoAction即可。特别注意，当你Redo selection的时候，GameEntity ListBox选择发生变化，记录了一个不该记录的UndoRedoAction。这种现象其实就是当我们在UndoRedo的时候记录了UndoRedo操作，这不是我们想要的现象。这里的解决方案就是通过一个bool字段\_enableAdd实现，显示地写出在RedoUndo的时候\_enableAdd为false，其他时候要为true；
2. GameEntity添加IsEnable属性；
3. 给GameEntityControl的界面显示出GameEntity的名字（TextBox）和是否启用（CheckBox）；
4. GameEntityControl中添加一个能添加Component的按钮，这个按钮有些讲究，它是一个ToggleButton（不知道为什么），它的Content是一个DockPanel（不知道为什么），我们需要在DockPanel中写出按钮名字，和一个用Path画出来的倒三角”M0,0 L0,2 L4,6 L8,2 L8,0 L4,4 z”；
5. 做一个小功能，当我们打开PrimalEditor，如果没有最近打开的项目，那么直接跳转到Create Project界面。这个功能注册到OnLoaded事件；
6. 优化GameEntity的命名。我们希望命名可以撤回重做，为此需要RenameCommand，命令参数为新名字。输入名字的时候可以按回车确认，按ESC返回；
7. 我们还希望GameEntity的IsEnable的设置也可以撤回重做，为此需要EnableCommand，命令参数为新Enable值；
8. 可以看到这两个操作实际上仅仅只是新值代替旧值，后续会有很多类似这样的操作。为了方便实现这种操作的撤回重做功能，我们改善UndoRedoAction类，通过添加一个构造函数，写好\_undoAction和\_redoAction，分别用于给一个对象instance的某个属性property设置新值redoValue和旧值undoValue，你可以思考一下怎么写；
9. 创建了一个ResourcesDictionary，Dictionaries/ControlTemplates.xaml，你需要手动添加ControlTemplates.xaml.cs，ControlTemplates类继承自ResourceDictionary，并且是partial class。为了使ControlTemplates.xaml是ControlTemplates类的实例，你为ControlTemplates.xaml添加x:Class；
10. 先写一个空的TextBoxStyle作为TextBox控件的基础样式，再为TextBox控件添加一个默认样式（想一想默认样式要怎么写），默认样式继承自TextBoxStyle样式，为默认样式添加KeyDown事件和处理函数；
11. 上述事件按键的事件处理函数中，我们将实现上述新值代替旧值的撤回重做逻辑。注意，只有当该TextBox的Text属性有绑定时（想一想怎么判断），我们才需要处理新旧值替换的撤回重做逻辑，否则对于按键操作我们不需要做任何事情。如果有BindingExpression，你来完成Enter键和ESC键的逻辑。看视频36:36来检查你的代码是否正确；
12. 每个TextBox应该有一个属于它的Command，但是TextBox没有这个属性，我们暂时用Tag属性实现；
13. 在App.xaml中，我们使用这个ControlTemplates，通过MergedDictionary来获取resource dictionary，记住写法即可，设置sources = “pack://application:,,,(Path to your resource dictionary file)”；
14. 接下来我们完善GameEntityControl，整个界面通过一个ScrollViewer来实现，并且当DataConext为一个有效的GameEntity才显示，这个功能通过定义ScrollViewer.Style的Trigger触发器来实现，触发器中需要指定当哪个属性变化为什么值的时候触发，设置Setter来定义当触发时设置什么值为什么，通过这些提示，你来完成这个功能，跳转到40:23来检查你的代码；
15. 为GameEntityControl的Enable检查框也完成Command的绑定。值得注意的是IsChecked属性会使得用户操作（目标域）改变GameEntity的Enable属性（源域），同时又触发Command，这会带来一些不确定性，我们只希望用户操作触发Command，你可以想一想怎么做（用mode=oneway）。第二点值得注意的xaml中CommandParameter的写法，Binding并不是DataContext的属性(用RelativeSource定位到CheckBox自己)，看42:56来检查你的代码；
16. 接下来完成Logging功能，创建Utilities/Logger.cs，里面有一个enum MessageType，class LoggerMessage包含如下属性，构造函数你自己完成，；



1. Logger类较为晦涩，它的思路就是创建一个LoggerMessage的列表（ObservableCollection）\_messages，通过Log函数来添加消息，显示在UI界面上。关键在于，项目在后续可能会采用多线程，其他线程也需要提交错误信息到message，从而UI界面更新显示消息，但是WPF需要保证对UI界面的更新在主线程上完成，不能再其他线程去更新界面，因此这里需要用Application.Current.Dispatcher来获得主线程的调度器，通过BeginInvoke往UI线程的队列中添加一个Action，这个Action会在UI线程空闲时自动异步执行。由于BeginInvoke被标记为awaitable，是一个异步函数，我们需要添加await关键字来等待这个异步操作的完成，如此一来，Log函数也得是异步函数；
2. 添加Clear异步函数，清除消息列表，逻辑同上；
3. 为了实现对\_messages列表的过虑，筛选出我们想要的类型的消息，我们需要一个CollectionViewSource，在Logger的静态构造函数中，我们设置好过滤函数添加到Filter事件中，你来实现这个功能，看56:54检查你的代码；
4. 创建一个Utilities/LoggerControl.xaml，作为消息的展示界面，注意ItemSource的设置绑定的写法，我们绑定的是一个静态成员，而且是CollectionViewSource，你需要会写；
5. 同TabControl把LoggerControl和UndoRedoControl写在一起；
6. 为Project.Unload函数添加清除UndoList和RedoList的简易代码；
7. 为所有需要打印Logger写上对应需要打印的信息；
8. Catch中重新抛出异常，一个throw关键字即可；

# Game Engine Programming 008 - Multiselection game entities | C++ Game Engine

1. 完善一下LoggerControl的UI界面，我们采用ScrollViewer，对于每一条消息，我们用DockPanel，通过设置Style实现如下功能：显示提示信息，即MetaData；背景颜色为透明，说说用什么实现（Setter）；当鼠标放在消息上，背景颜色为灰色，说说用什么实现（Style.Triggers和Setter）；消息的显示我们用一个TextBlock显示时间，注意时间显示格式我们用（StringFormat）实现；时间需要通过颜色区分消息类型，我们通过（DataTrigger和Setter）实现；然后再用一个TextBlock显示消息信息，保证自动换行即可；
2. 添加三个按钮用于过滤三种类型的消息，为了添加三个ToggleButton，我们把LoggerControl用DockPanel展示，把三个按钮放在顶部，用Border包起来。再加一个Button用于清空消息列表。把他们全部摆在右上角；
3. 完成三个ToggleButton的Click事件函数，在这个函数中，我们根据哪些按钮被按下了来设置消息的过滤；
4. 接下来完成GameEntity的多选多修改功能，为此我们创建ViewModel的抽象基类MSEntity，类MSGameEntity继承自该类，MSEntity中有和GameEntity一样的属性，只不过注意他们得是nullable的，因为用户多选的GameEntity在某个属性上可能不一致，我们用null来表示这种不一致。此外MSEntity还有一个列表，表明用户选中了哪些GameEntity；
5. 我们还需要抽象基类MSComponent，我们需要一个模板参数T来表示这个MSComponent是哪个类型的Component，为了使得我们能创建MSComponent的列表或集合，我们需要用（Interface）来实现。另外类Component也得是抽象的，因为我们不创建任何Component实例，只创建继承自该类的实例；
6. 完成MSEntity的构造函数，构造函数通过一个GameEntity的列表来初始化。值得注意的是，我们希望修改了MSEntity的任意属性，我们同步修改所有被选中的GameEntity的对应属性，为了完成这个功能，我们通过（PropertyChanged事件）实现；
7. 通过一个函数来完成对所有GameEntity对应属性的修改，考虑到子类也有可能会修改他自己的属性，这个函数需要被关键字（virtual）修饰，通过返回值来通知是否找到并完成了所有GameEntity对应属性的修改；

我没有按照视频方法用switch case来写，用了PropertyInfo，所以也不需要把这个函数标记为虚函数

1. 通过一个函数完成将所有的GameEntity对应属性同步到MSEntity自己的属性上，类似地这个函数也得是（virtual）的，你可以思考一下这个函数怎么写最方便；

我也没有按照视频方法，也是用了PropertyInfo，并针对每一个MSEntity需要更新的属性都更新，这里还是需要virtual

1. 创建Utilities/MathUtil.cs来帮助完成上述内容，包括浮点数的比较函数IsTheSame，通过this关键字实现添加到类的方法；

由于我之前没有switch case来逐个属性比较，这里也不需要这类函数

1. 上述更新ameEntity属性方法和更新MSEntity属性方法的过程有一个细节可能会出错，当我们某次想调用更新MSEntity的属性的方法时，由于属性改变，PropertyChanged事件被触发，这会导致所有GameEntity的对应属性被统一更新，比如某MSEntity属性被更新为null，那么这时所有GameEntity的对应属性也将被设为null，这是不好的。解决方法是通过一个bool \_enableUpdate来实现，确保\_enableUpdate为true时才能更新所有GameEntity，并且当我们调用更新MSEntity属性方法时enableUpdate为false，禁用对所有GameEntity的更新；
2. MSGameEntity就是一个简单的对MSEntity的继承，构造函数里面更新属性即可，不知道为什么这样设计；
3. 接下来，每次当选择的GameEntity发生改变时，我们创建MSGameEntity，把这个MSGameEntity作为GameEntityView的DataContext（以前是GameEntity）；
4. 既然更换了DataContext，我们就需要重新在MSEntity中写RenameCommand和EnableCommand的逻辑（实际上不需要Command），GameEntity中的这两个Command和其相关内容就删掉。对于重命名的撤回重做功能，视频中通过GetKeyBoardFocus和LostKeyBoardFocus事件来完成重命名的撤回重做逻辑，见51:57，我认为略显复杂且不妥。我的想法依然是用ICommand和ResouceDictionary来完成，类似之前写过的RenameCommand，给MSEntity写一个RenameCommand，放在TextBox的Tag上，在KeyDown事件中，当按下Enter，我们执行RenameCommand，当按下ESC，直接UpdateTarget，同时都要让焦点聚焦到WorldEditorControl，离开TextBox。LostFocus事件处理也类似，和按下Enter的处理逻辑一致。你来把这里好好写一写，和视频不一样；

我用RenameCommand和EnableCommand完成了上述和下述功能，不同于视频内容

1. MSEntity的EnableCommand也类似；

# Game Engine Programming 009 - Identifiers with index and generation parts | C++ Game Engine

1. 切换到Engine solution，添加Common/CommonHeaders.h，包含stdint.h和assert.h；
2. 创建Common/PrimitiveTypes.h，为uintxx\_t、intxx\_t、float重命名。定义常量uxx\_invalid\_id = …。然后再CommonHeaders.h中包含这个头文件；
3. 引擎中我们实现Entity Component系统用的是Data Oriented，接下来我们将实现这个技术。创建Common/Id.h，namespace是primal::id，在里面定义一个id\_type类型，我们用u32。定义一个常量generation\_bit表示有多少位表示代系，我们用8位。定义一个常量index\_bit表示有多少位是索引部分，我们用id\_type的位数减去generation part的位数。定义一个index\_mask，和一个generation\_mask（注意视频中generation的1位在低位），和一个id\_mask。定义一个generation\_type，这个类型用于表示generation\_bit，根据其位数来选择generation\_type的类型（暂时不知道这个类型何时使用）。然后做一些静态断言，检查generation\_type类型的位数是否合理（通过了两个方面来检查，你想一想）。见50:01检查你的代码；
4. 写一个内联函数检查id是否是合法id。写一个内联函数获得id的索引部分。写一个内联函数获得id的代系部分。写一个函数获得代系增加一的type\_id，这个函数中我们确保代系不要超过最大值（教程中写的是255，或许我可以显示写出来是pow(2, generation\_bit) – 1）。看53:21检查你的代码；
5. 开始实现Entity Component。创建Components/Entity.cpp，Components/Entity.h，Componets/ComponentsCommon.h，其中Componets/ComponentsCommon.h用于添加一些本文件夹下的cpp文件常用的头文件，注意，你需要修改项目设置，添加Common文件夹为包含的头文件目录；
6. 为了在Debug模式下区分对于Entity和Component的索引，虽然他们都是id\_type类型，我们需要在Common/id.h中定义一个结构体internal::id\_base，注意构造函数你可以写一个有参的和一个无参的，还需要一个类型转换函数转换到id\_type（你需要自己完成），这些构造函数和类型转换成员函数需要被适当的标注为常成员、不抛出异常、变量尽量为编译时常量、等等。然后，通过\_Debug宏区区分debug和release模式，均通过定义宏DEFINE\_TYPE\_ID(name)来为我们想要的索引类型进行定义。这一步非常妙，因为在debug模式下我们通过继承来完成不同类型宏定义。见1:03:03检查你的代码；
7. 在Components/ComponetsCommon.h我们通过刚才写的宏来定义类型，我们需要定义primal::game\_entity::entity\_id类型；
8. 在Components/Entity.h中，我们需要primal::game\_entity命名空间下的声明创建entity和删除entity的函数，注意创建entity的函数需要entity\_info，你来完成。还有一个检查entity是否存在的函数，注意参数和返回值类型要用我们宏定义中写的；
9. 创建Components/Transform.h中，需要primal::transform命名空间下的transform\_id类型，需要创建transform的函数，注意创建transform的函数需要init\_info，你来完成。还需要一个移除transform的函数；
10. 注意类型的forward declare，这里用宏定义来方便完成对于每一种component的init\_info的forward declare，我在我的cpp项目中经常遇到这个问题，所以不在话下；

# Game Engine Programming 010 - Entity and component implementation in C++ | C++ Game Engine（A bit hard）

1. 创建Utilities/Utilities.h，给出两个宏定义分别表示是否使用stl的vector或者deque，通过宏的值来定义vector和deque，如果使用stl的，那么就简单进行一个对std::vector和std::deque的重命名，放在命名空间primal::util下，否则，用自己写的class vector和class deque。然后这个头文件要放在Common/CommonHeaders.h中；
2. 完善一下命名空间，需要Id.h中的某些变量放在primal::id::internal命名空间下，表示这些变量不希望被外部代码访问（你来决定是那些变量），过于底层；
3. 在取出id索引部分的函数中，加一个断言确保索引部分是有效索引；
4. 在取出id代系部分的函数中，也加一个断言确保代系是有效的，即小于最大值，注意这个最大值的写法；
5. 在添加id代系的函数中，同理，修改你的最大值写法；

我直接用internal::generation\_mask作为最大值，不同于视频id\_type(1) << internal::generation\_bit – 1的写法

1. 在id\_base和其子类中，构造函数默认参数改为0，这是为了保证在debug和release下，’{component\_name}\_id()’这种写法创建了一个值0的索引（之前，’{component\_name}\_id()’在debug下创建了一个无效索引，而在release下创建了一个0索引）；
2. 创建EngineAPI/GameEntity.h，这个文件夹下的是API，可以给游戏引擎代码使用的。首先我们需要一个class primal::game\_entity::entity，目前这个类只有一个id\_type类型的成员变量，表示游戏中的一个entity。你来完成这个类，运用好常量表达式关键字（constexpr和const）并保证类必须显示构造（explicit），写出你能想到的一些可能在后续用到的成员函数；
3. 完成后，你可能需要整理代码，比如移动一下你的定义entity\_id类型的代码段(后面的transform\_id也是类似，不再提醒)，我的代码获取可以直接放在ComponentCommon.h中，自己考虑；
4. 开始写Entity.cpp。首先需要一个匿名空间，表示所有entity的代系的vector容器，这个容器按索引顺序存储了当前内存中所有曾经创建过entity的位置的代系值，不管这个entity是否被移除，这个容器还暗示了我们至今为止在多少内存上创建过了entity。还有一个表示当前内存中哪些索引的entity是已经被移除过且空闲的一个deque容器，从旧到新的顺序；
5. 在Common/Id.h中，再定义一个常量，表示内存中最少删除了多少个entity后，再添加entity就会开始使用这些空闲的槽，我们的项目中设为1024；
6. 然后开始完成创建entity的函数，由于这个函数需要给游戏引擎代码使用，返回值更换为之前写的entity类（而不是直接返回一个entity\_id），其他的删除entity和判断entity是否存活的函数也同理。创建entity的函数首先要保证指定了transform控件。然后根据当前是否有足够多的空闲entity槽来完成创建entity的功能，创建entity需要：
7. 创建transform组件，同时更新transfom组件索引容器；
8. 获得当前entity的id
9. 同步更新代系容器；
10. 利用id最后构造好entity，返回；
11. 完成删除entity的函数，想一想你要做什么事，代系容器要做什么（什么都不做）？空闲容器要做什么（在队尾添加当前索引）？transform组件容器又要做什么事（清理为无效索引）？；
12. 完成判断一个entity是否存活的函数，如果一个合法的entity所在的槽是否已经被替换了（对比代系值），或者entity确实已经被移除但尚未被替换（通过transform组件是否还有效判断），那么这个entity就没有存活，反之；
13. 一个提示，使用#pragma warning(disable: 4530)可以关闭有关异常的编译警告信息；
14. 创建一个Engine/TransformComponent.h，在其中定义primal::transform::component类，其实现方式于entity类类似，你自己完成；
15. 然后在Entity.h中，在添加一个vector容器，表示当前内存中所有创建的transform组件，它是与entity同步创建的，因为每一个entity都有一个transform组件；
16. entity类需要一个获得transform\_id（或者获得transform即可）的成员函数，你来实现；
17. 关于上述以及后续这些函数的实现，你都注意一下写好断言，做一些有必要的检查，包括检查传入的id是否合法，检查entity的索引是否不超过代系容器的长度，等等你能想到的；
18. 如果在windows 64位编译环境下，我们需要DirectXMath这个数学库，你需要包含其头文件；
19. 创建Utilities/MathType.h，在这里我们在primal::math命名空间下定义一些数学常量如pi，epsilon，在windows 64位编译环境下重新命名DirectXMath中的向量和矩阵，重新命名为vx，vxa，u32vx、s32vx、m3x3、m4x4、m4x4a；
20. 开始写Transform.cpp，首先，和entity.cpp类似的，你需要一个匿名空间，定义位置容器，旋转容器和缩放容器，表示内存中创建过的entity的transform组件；
21. 完成创建transform组件的函数，这个函数中，你需要判断是在创建空闲槽的transform组件还是创建新的transform组件（直接用位置容器的大小和entity的index比较即可），然后更新位置容器，旋转容器和缩放容器。最后我们返回transform组件的id，注意返回值类型（transform::component），这个id和entity的id一致即可；
22. 完成删除transform组件的函数；
23. 完成primal::transform::component中获取roration，position和scale的成员函数；、

# Game Engine Programming 011 - Testing entities system and creating the engine DLL | C++ Game Engine

1. 创建一个C++空项目用于测试我们刚才的代码，命名为EngineTest，测试创建和移除entity的代码是否可行；
2. 创建main.cpp，
3. 创建Test.h，定义一个抽象Test类，包括initialize，run和shutdown三个抽象成员函数
4. 借助这个Test类，在TestEntityComponent.h中完成你的测试代码，写在类engine\_test中，然后再main.cpp中使用，你需要借助宏定义的方法来切换测试代码；
5. 在写engine\_test这个类的时候，你需要来自Engine项目的一些头文件，所以为EngineTest项目添加额外的include directory。这个项目还需要Engine项目生成的静态库，所以你也要添加库目录；
6. 为了测试添加和删除entity的功能，首先定义一些成员变量，包括记录创建了哪些entity的容器，一个u32表示添加了多少个entity，一个u32表示删除了多少个entity，一个u32表示当前一共有多少entity；
7. 首先完成engine\_test的initialize函数，这个函数用于初始化，这个测试中，我们需要随机地添加和删除些许entity，因此首先用（srand((u32)time(nullptr))）代码来生成随机种子。
8. 完成engine\_test的run函数，每当用户输入非q键位跑一次测试。首先我们随机添加0-20个entity（如果内存中没有entity就直接加1000个），然后再随机地删除0-20个entity（如果内存中不足1000个entity就不删），以上内容每次测试做10000遍；
9. 写一个engine\_test::print\_result来输出我们的执行结果信息；
10. main.cpp中，我们可以直接在源代码中链接到lib库，通过（#pragma comment(lib, "Engine.lib")）语句实现；
11. main.cpp中，通过（\_CrtSetDbgFlag）来设置在debug模式下编译器追踪堆内存分配和释放（\_CRTDBG\_ALLOC\_MEM\_DF），并在程序结束时检查并输出内存泄漏情况（\_CRTDBG\_LEAK\_CHECK\_DF）；
12. 执行测试，看看效果，看看哪里有bug，主要关注id\_type的代系部分会不会到达最大值；
13. 把generation\_bit设为10，看看id\_type的代系部分是不是会更晚达到最大值；
14. 开始为Engine项目导出DLL，创建一个EngineDLL项目，只留下dllmain.cpp文件。
15. 该项目还不应该使用预编译头文件，到项目属性里设置一下；
16. 类似EngineTest项目，EngineDLL项目也需要Engine项目编译生成的静态库，还有头文件，因此在项目设置里添加库目录和包含目录。还要注意C++版本设置；
17. 在dllmain.cpp中，保证dll文件被加载时，启用堆内存分配和释放的跟踪，以及在程序结束时检查内存泄漏；
18. dllmain.cpp需要包含一些必要的头文件，你自己完成；
19. 创建EngineAPI.cpp，在这其中完成你的给Editor使用的函数。首先你需要一个宏定义EDITOR\_INTERFACE专门用于修饰函数，其作用是：
20. 保证使用C语言的链接规范来实现函数调用，也即禁用C++的函数名称修饰，使得C++函数可以被其他编程语言链接，利用（extern "C"）实现；
21. 指定一个函数或变量需要被导出到DLL中，通过（\_declspec(dllexport)）实现；
22. 在PrimalEditor中，我们开始修正GameEntity类，为了能关联到引擎代码中entity，添加一个EntityId属性，由于在引擎代码中，我们的id\_type固定为32位无符号整数，而且不会去修改它，所以在Editor代码中，我们可以用4字节的int类型来表示EntityId。为了能在Editor中向引擎添加和删除entity，再添加另一个属性IsActive，在set中完成在引擎中添加和删除entity的功能。为了表示Id，需要引入静态类ID来写出INVALID\_ID等和ID相关的函数，你来完成；
23. 然后写第一个需要被导入到DLL的函数CreateGameEntity，注意返回值类型的写法，函数参数是一个descriptor指针（对应到C#就是引用类型的类），我们称为game\_entity\_descriptor，它对应的就是Editor中的GameEntity类，因此你需要完成这个结构体。这个结构体的内容就是来自Editor中的GameEntity的，你的任务就是从这个结构体的内容转换到用于创建entity的entity\_info。为此你需要首先转换来自编辑器的transform组件的信息到transform\_info，这个过程关键要把欧拉角转换到四元数，借助DirectX的数学函数完成。注意：
24. 所有的向量类型我们用的都是16字节对齐的，这是因为在C#中，Vector这个结构体保证了按照16字节对齐，也就是说这个结构体起始地址和占用大小都是16字节的倍数，即使它的字段float x,y,z都是4字节的（按照通常，那就是按4字节对齐，因为最大的字段是4字节）。
25. C#中引用类型占用的内存是由托管堆管理的，所以我们如果写了一个嵌套的引用类型，比如引用类型A嵌套了一个引用类型B，A的引用类型变量a指向托管堆的空间memory(a)，memory(a)中就存放了B的内存memory(b)而非B的引用类型变量b。所以你要注意C++中game\_entity\_descriptor结构体的写法！；

见视频1:00:39检查你的代码；

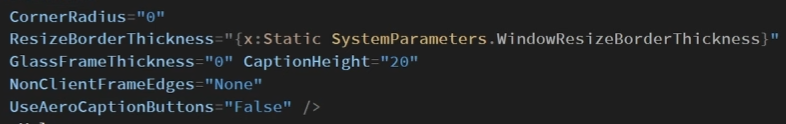
1. 完成EngineAPI.cpp中的移除Entity的函数，参数就是一个id，它对应GameEntity中的EntityID属性。见1;01:41检查你的代码；
2. 开始在Editor中导入这个DLL，创建PrimalEditor/DllWrapper/EngineAPI.cs，这个类是静态类，你在完成时需要注意以下几个点：
3. 通过（[DllImport(dllName: \_dll\_name)]）属性来标记函数是从DLL中导入的；
4. 创建Entity的函数CreateGameEntity，函数参数是一个类GameEntityDescriptor，对应到EngineAPI.cpp中的game\_entity\_descriptor，你也需要在C#中完成这个类。这个类可能需要Attribute（[StructLayout(LayoutKind.Sequential)]）来修饰，保证内存布局是按字段的声明顺序排列的。这个函数有两个，一个private一个public，一个作为内部的的，一个作为直接传入GameEntity的；
5. 为了从一个Editor中的GameEntity获取到Transform组件，我们需要给GameEntity添加一个成员函数，专门用于获得Components容器中的指定类型的实例，可以写为模板函数直接进行类型转换，这个技巧很不错；
6. 移除Entity的函数也是类似，你来完成；
7. 既然我们用GameEntity的IsActive属性的设置来完成引擎中Entity的创建和移除，在编辑器中，我们需要进行设置。在Scene相关代码文件中，修改你的添加和删除GameEntity的逻辑，尤其注意这部分的撤回重做代码。另一个需要做的地方就是在通过读取项目文件初始化场景时，我们需要把读进来的GameEntity逐个在引擎中创建；
8. 接下来你需要在Editor中测试你的添加删除GameEntity的代码，启用Editor项目设置中的Enable native code debug可以帮助我们从C#代码跳转到C++代码进行调试；

# Game Engine Programming 012 - Dark UI theme and animations | C++ Game Engine

1. 添加名为EditorColors的Resource Dictionary，在这里面，定义好颜色和对应的笔刷，颜色和名称如下。然后你需要把这个字典放到App.xaml中。特别需要注意，我们用ResourceDictionary.MergedDictionaries时也要注意字典的引用顺序！；

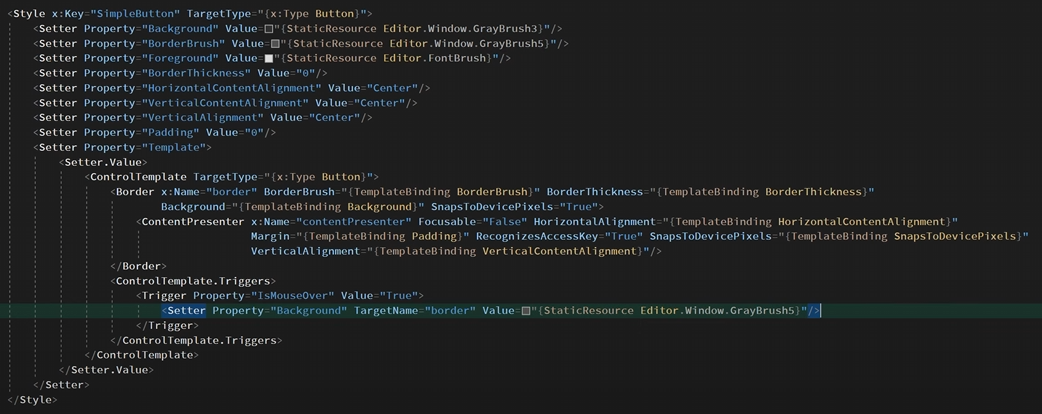


1. 然后我们给主窗口设置一下颜色，先在ControlTemplate.xaml中写一个style，为了设置主窗口，设置TargetType属性为（x:Type Window），然后我们主要设置窗口的背景色为GrayColor1对应的笔刷，设置边框为BackgroundColor对应的笔刷（目前，设置Windows控件的边框颜色并没有效果）；
2. 接下来修改主窗口的tiltle bar，使用Setter设置WindowChrome.WindowChrome属性，涉及到如下属性，你可以分别调整看看他们有什么作用。见10：10看代码；

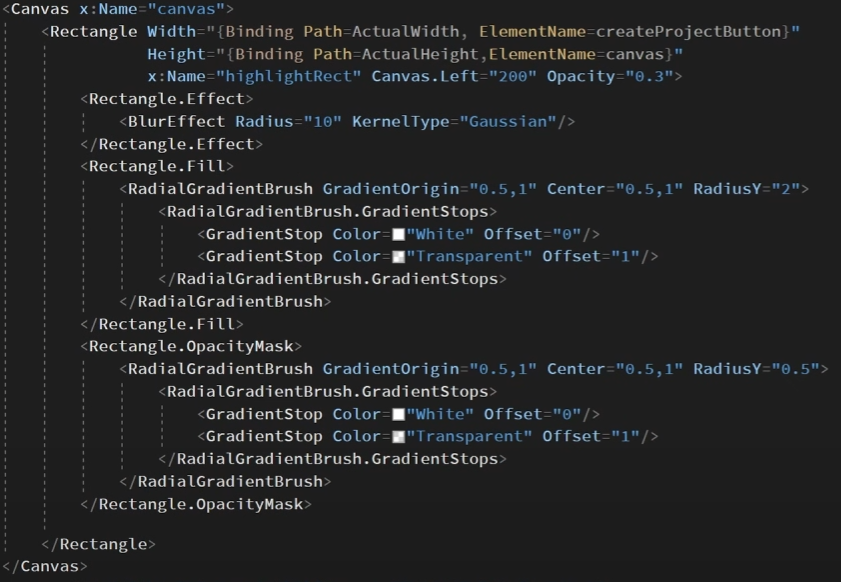


1. 还是修改Windows控件的Style，首先通过（ControlTemplate）来为Window控件设置模板样式，我们用一个Border控件来展示主窗口，设置边框的厚度为1，背景颜色和Window保持一致，边框颜色也和Window的边框颜色保持一致。当窗口最大化时，设置窗口边框厚度为调整窗口大小的可交互边框的厚度（SystemParameters.WindowResizeBorderThickness）。用（ContentPresenter）和一个装饰器（AdornerDecorator）来进行嵌套表示窗口主要内容（Window.Content）,并通过设置Margin来控制窗口主要内容和标题栏不重叠。然后用DockPanel来展示title bar，一个textblock放在左上角表示标题（文本绑定到windows的标题）。右上角放一个最小化按钮，最大化按钮和关闭窗口的按钮，通过Content属性和（Path）控件来设置按钮里面的图案，并添加click事件函数，其中通过（）属性来获得主窗口Window的示例。为了统一三个按钮的样式，我们用DockPanel.Resources来设置Path和Button的默认样式，重要的是通过属性（）实现让这三个按钮能被WindowsChrome点击相应。
2. 给ControlTemplate.xaml再写一个Button的样式，设计如下内容，包括设置背景色，前景色，边框厚度（使用了Border），按钮内容的对齐方式，按钮本身的对齐方式以及按钮控件内容的Padding。还有设置Button的模板，我们通过Border来包含按钮的内容，保证Border的一些颜色属性和Button的一致。对于Button的内容，我们保证居中对齐，启用快速访问键，不可聚焦，设置好Margin，保证内容精确吸附到像素无模糊。当鼠标移动到按钮上我们设置某个颜色即可。然后，窗口右上角的三个按钮的样式继承这个样式。注意触发器要写再在ControlTemplate上而不是Border上，似乎Border并没有IsMouseOver属性；

我还学习了使用IValueConvert来通过设置Button的Tag属性实现不同的Button在鼠标悬停时有不同的背景颜色；



1. 对话框的样式同上述，不过右上角只有一个关闭对话框的按钮；
2. 接下来我们开始优化ProjectBrowserDlg的外观，从ToggleButton开始。我们的做法是把ToggleButton的样式提取到ProjectBrowserDlg.xaml中，而不是从头开始编辑。然后删除一些你不需要的样式属性，设置背景为透明，设置好前景色，字体大小大一些，选一个你觉得合适的字体，内容全部居中对齐，设置边框透明和边框厚度。在ControlTemplate中我们只需要写一个TextBlock即可，用Effect属性做一些美观设置，再通过Trigger和MultiTrigger进行一些颜色的设置；
3. 在ControlTemplates.xaml中，写一个TextBlock的Style，设置好前景色和对齐即可；
4. 我们修改NewProjectControl，用Border来包裹所有的控件内容，设置好合适的margin和背景色（比Window的亮一些），运用刚刚写的TextBlock的style；
5. 再修改ControlTemplates中的TextBox的Style，包括前景色，背景色，选中颜色，边框颜色，对齐和填充；
6. 同样地修改OpenProjectControl，也用Border来包裹所有的控件内容；
7. 在ProjectBrowserDlg中，给这两个Control的StackPannel也添加上DropShadow；
8. 接下来完成OpenProject和NewProject的面板切换动画。首先再OpenProjectControl或NewProjectControl的上方添加一根好看的线（Rectangle+LinearGradientBrush+GradientStops）。然后给ToggleButton的下面添加一个Spot light的效果，通过如下图所示的Canvas和Rectangle实现，你可以自己调整看看每个参数是什么含义；



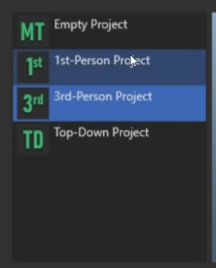
1. 然后我们做切换动画，当我们点击OpenProject或者NewProject，完成两个动画。第一个是SpotLight的动画，作用于Rectangle的左边界属性，用（DoubleAnimation）完成，第二个是BrowserContent的动画，作用于Margin，用（ThicknessAnimation）完成。另外，我们切换完成后要设置他们的Enable属性；
2. 优化动画效果，我们用CubicEase作为这两个动画的EasingFunction；
3. 让OpenProjectControl和CreateProjectControl间隔远一些；

# Game Engine Programming 013 - Finishing the dark UI theme | C++ Game Engine

1. 开始为Button写Style。我们的做法是随便先找一个项目中的Button，提取Style，然后开始编辑，完成后拷贝到ControlTemplate.xaml中。我们需要修改的属性包括：被键盘聚焦后的边框颜色，按钮背景色设置为一个从上到下的渐变，按钮边框色是从上到下渐变但颜色稍不一样，前景色，按钮内容对齐，填充，按钮有一点圆弧拐角，当按钮为默认时设置背景为蓝色且从上到下有一个透明度的渐变，当按钮按下或者鼠标停留设置背景也为蓝色且从上到下（或按下时从下到上）有一个透明度的渐变且字体更亮，当按钮被Disable是分别设置字体颜色背景色和边框颜色；

我按照自己的想法大致完成；

1. 接下来为ListBox和ListBoxItem写Style。你自己来写吧，写成大概下面这个效果。重要的是额外的给OpenProjectControl的面板中的项目添加一些项目信息如路径和日期；



1. 修改WorldEditorControl的界面；
2. 修改LoggerView的三个ToggleButton，让他们变成红黄绿三种颜色。美化一下他的界面，字体颜色；

# Game Engine Programming 014 - NumberBox, ScalarBox, VectorBox | C++ Game Engine

1. 创建Utilities/Control/NumberBox.cs，我们开始自己写一个Control。这个Control中，我们需要一个string类型的属性，作为数值，通过（public static readonly DependencyProperty）实现。他需要作为默认样式，通过（DefaultStyleKey或DefaultStyleKeyProperty）实现。我们需要一个TextBlock显示数字和一个TextBox来输入数字，因此他们需要作为这个控件的（TemplatePart）。当ControlTemplate应用在控件上，我们需要获取ControlTemplate中指定的控件，作为这个Control的属性。写出textBlock的鼠标左键按下，鼠标左键抬起，鼠标移动的事件函数，我们的目的就是点住TextBlock并拖动，可以改变value的值，你来完成，注意的细节包括：
2. 这些事件不要路由传播了；
3. 拖动有一个值变化的最小阈值（用SystemParamter中的）；
4. 数值字符串保留五位小数即可；
5. 鼠标抬起时需要判断一下我们是直接改变数值，还是启用textBox；
6. 在Dictionaries/ControlTemplates.xaml中写出NumberBox的默认样式，为此你需要做一下工作：
7. 我们首先需要一个特殊的textBox样式，这个样式用于类似重命名的功能，当按下回车或esc，或者TextBox失焦，这个TextBox就会消失（设置Visibility属性），你来写一个；
8. 此外，稍微修改一个之前的逻辑，那就是textBox按下回车或ESC后，焦点切换到上一个控件，而不是切换到worldEditor；
9. 在NumberBox的ControlTemplate中，我们需要一个TextBlock，设置当鼠标停留是更换鼠标指针为SizeWE；
10. 还需要一个TextBox，它默认是不可见的，注意设置UpdateSourceTrigger的方式，水平对齐属性；
11. 我们需要实现当TextBox可见时则TextBlock不可见；
12. 当Value为Null时，设置一下TextBlock的背景色和ToolTip即可；
13. 为NumberBox添加一个功能，当鼠标点击并拖动时，按住shift数值走的更快，按住ctrl数值走的更慢；
14. 为NumberBox添加一个Multiplier属性，用于完成上述功能；
15. 完成一个VecterBox控件，添加到Utilities/Controls/VectorBox.cs。
16. 添加四个属性X、Y、Z、W，还有一个Multiplier属性；
17. 在VecterBox中应用四个NumberBox，写好对应的toolTip，对于其中的NumberBox，可以通过Style设置Multiplier属性，设置ToolTip和它的显示时间延迟，间隔显示时间延迟；
18. 我们需要一个Enum表示使用2、3和4个维度的向量，然后通过一个属性来表示这个Vector是几维的；
19. 添加一个表示方向的属性，表示这些数字是竖着排的还是横着排的；
20. 通过DataTrigger完成，根据方向属性设置四个NumberBox的排列；
21. 通过DataTrigger完成，根据向量枚举类型设置显示几个NumberBox；
22. 设置当鼠标停留的效果，边界线更亮，当Disable，整个控件透明度减半；

最后VectorBox长这样：



1. 写一个ScalerBox控件，本质上就是一个NumberBox，但是有VectorBox的样式，他直接继承自NumberBox即可，然后你来写Style，最后长这样：



# Game Engine Programming 015 - Transform component | C++ Game Engine

1. 首先修正一下ProjectLayout的选中项目的背景颜色；
2. 之前在我们的项目中，当UndoList发生了元素的改变，我们设置focus为WorldEditorControler，这样做可能有些问题，先去掉这个逻辑；
3. 接下来我们完成MSComponent这个类，你需要完成如下内容：
4. 首先需要一个列表属性，表示这个MSComponent对应到哪些实际的Component；
5. 构造函数中，给出一个MSEntity，我们的任务是完成上述属性的填充，我们要从所有选中的GameEntity中获得到对应的组件；
6. 需要两个抽象函数，类似于MSGameEntity，一个就是通过MSComponent的属性来更新所有它的对应的Component，另一个是通过所有它对应的所有Component来同步到当前MSComponent上；
7. 类似的，需要一个功能，当MSComponent的任意一个属性发生变化，把这个属性同步到所有Component上，这个功能需要通过（PropertyChanged事件）实现，注意“循环更新”的问题，参考MSGameEntity即可；
8. 创建MSTransform类，显然它继承自MSComponent，你需要完成如下内容：
9. 写出所有属性，注意这里全用float而不是Vector3，因为Vector3作为ViewModel的属性难以在WPF中表示，可能并不是WPF的基本数据类型，此外注意浮点数的比较问题；
10. 参考MSGameEntity，完成MSTransform属性的更新同步到所有Transform的属性的函数，完成所有Transform的属性同步到MSTransform属性的函数，你可能可以直接在MSComponent中完成？你自己考量（答：这不好）。此外，这个过程可能需要用到MSGameEntity的GetMixedValue函数，你可能需要修改这个函数，或者可以写得更简洁一些，提示用.Any；
11. 每当创建一个MSTransform的时候，更新一下它的所有属性；
12. 在MSEntity中我们有一个MSComponent的列表L，现在写一个函数来完成从所有选中的GameEntity中来创建L，其逻辑就是查找所有选中的GameEntity的共同组件，如果所有选中GameEntity都有这个类型的组件，则创建对应的MSComponent，添加到L中。其中你可能需要一个函数，给出一个未知类型的组件，你需要从一个MSEntity中获取到指定这个组件类型的MSComponent，我认为可以通过Component的静态函数完成（不对，这里还是通过成员函数完成）；
13. 接下来完成展示各种MSComponent的User Control，首先创建Editor/WorldEditor/ComponentControl这样一个User Control，它就是包含一个Expander。需要注意的是，这个Expander的Header和Content是可以设置的，我们的做法是为ComponentControl添加（DependencyProperty），这样我们就可以在xaml中写可配置的Header和Content了，因为有Content，所以我们需要给ComponentControl指定一个（ContentProperty）属性。为了使得Expander的Header内容和可以对应到这个ComponentControl的（DependencyProperty），我们要用到Binding；
14. 接下来创建TransformControl，给Transform组件写一个界面，这需要我们用到刚才创建的ComponentControl；
15. 为了能够在GameEntityControl中展示出各种ComponentControl，我们这里用的是ItemControl，ItemSource中指定的集合有不同的MSComponent，为了先能够显示出MSTransformComponent，我们需要写一个（DataTemplate）来实现；
16. 测试一下Transform组件是否运作正常；
17. 你可以按照你的想法给ComponentControl添加一些样式，比如鼠标移上去，亮一些；
18. 接下来为Transform组件的更改添加撤回重做功能。
19. 为TransformControl的三个VectorBox都添加鼠标按下事件、鼠标弹起事件和失去焦点事件；
20. 在鼠标按下时，我们记住当前MSTransform对应的所有Transform组件及其Position/Rotation/Scale属性，并写好Undo和Action。关于UndoAction，我们设置会之前记住的每个Transform的Position/Rotation/Scale属性为之前记住的值，然后更新MSEntity的MSTransform的所有属性。注意，视频中用MSEntity来找到MSTransform，为什么我不能直接用TransformControl的DataContext（MSTransform）来更新？我自己摸索一下；
21. 当鼠标抬起时，首先检查MSTransform中是否有属性被更改了，这个检查很巧妙，通过给（MSTransform注册新的PropertyChanged事件）实现的。当有属性改变，则为UndoRedoManager添加UndoAction和RedoAction，RedoAction的实现类似于UndoAction，你来完成；
22. 当VectorBox失焦，这可能是输入数字然确定后导致的，所以这里的逻辑就是直接调用上述鼠标抬起事件的函数；
23. 你可以考虑简化代码，旨在减少代码重复；

其实你可以不这么做，你按照TextBox的写法来，用Command来做，自己想一想？（还是按照视频做法，跟直接一些）

1. 用GameEntityControl为GameEntity命名这个操作有可能有bug，你试一下撤回重做，看看有没有；√
2. 一些改善，当我们移除一个GameEntity，把Entity改为Invaild（检查一下你有没有做这个事，我做了），并且只能移除EntityId

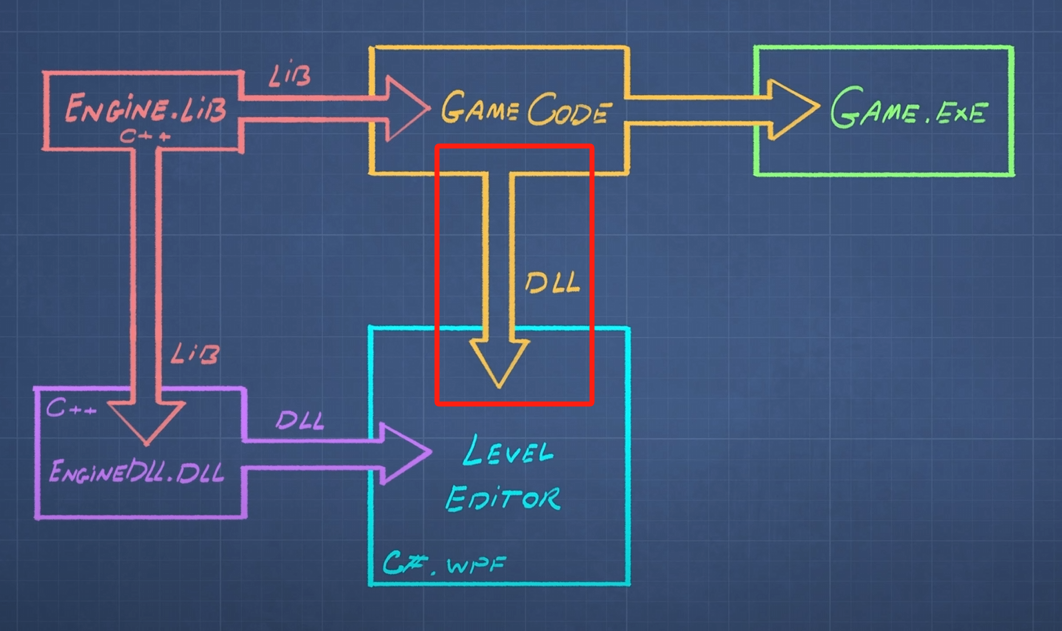
为valid的Entity；

# Game Engine Programming 016 - C++ scripting and script component | C++ Game Engine（A bit hard）

1. 创建primal::script::entity\_script类，其继承自primal::entity::game\_entity类，该类需要：
2. 一个begin\_play函数在游戏开始运行时被调用，即脚本开始运作，虚函数；
3. 一个update函数在游戏运行的每一帧被调用，一个float参数表示上一帧的运行时间，虚函数
4. 因为这个类型有虚函数了，可能发生多态，我们需要{将类的析构函数标记为虚析构函数}；
5. 该类作为基类使用，无需被实例化，所以设置{构造函数的访问级别为protected}，然后，你来完成构造函数；
6. 在Editor中，当一个GameEntity创建一个ScriptComponent，我们需需要一个函数方法来创建这个ScriptComponent对应的entity\_script。为此，在引擎代码中写一个函数create\_script，用于创建某种类型的entity\_script。因此，该函数参数为一个{primal::entity\_game\_entity}，返回值是一个指向某类型entity\_script的指针。你来完成这个函数的声明，并定义一个类型，表示指向某种entity\_script的指针{我们返回的时指向某类型的entity\_script的指针，返回类型不知道，所以这里要用函数模板}；
7. 使用using语法定义一个指向上述函数的指针类型，因为对于ScripComponent的创建，我们需要一个函数作为参数来创建entity\_script；
8. 上述内容并不需要被外部代码访问，可以放到primal::script::detail命名空间中；
9. 创建EngineAPI/ScriptComponent.h；
10. 创建Components/Script.cpp和Components/Script.h。接下来你要完成的任务就是，在创建一个entity的时候，同时也根据需要创建ScriptComponent，你需要修改多处代码，你来完成。你可能需要注意以下几点：
11. 实现primal::script::init\_info；
12. 实现primal::script::create函数，当可以填洞时怎么做，不填洞时怎么做；
13. 实现primal:: game\_entity::entity::get\_script成员函数;
14. 对于实际的primal::entity\_script的创建，我们要用double indexing的方法来完成，你首先需要理解这个方法，它需要一个容器entity\_scripts用来存放实际的primal::entity\_script，一个容器id\_mapping用来存放指向前述容器对应实际primal::entity\_script的索引。你需要声明所有相关容器放在匿名空间中；
15. 实现primal::script::exist（或primal::script::component::exist成员）函数；
16. 依据double indexing的实现原理，写一个primal::util::erase\_unorder函数，作用是先将vector容器中指定位置的元素和最后一个元素交换位置，然后删除最后一个元素；
17. 实现primal::script::remove（或primal::script::component::remove成员）函数，利用上述函数，然后注意正确操作id\_mapping，哪一个元素要设为invalid，哪一个元素要重新设置值。特别注意当entity\_scripts只有一个元素的边界情况！；
18. 你要注意需要在创建game\_entity的同时也完成判断创建script component的逻辑；我给忘了；
19. 删除game\_entity的方法中，也要记得判断并删除script component；我没忘了；
20. 完成后，build以下看看有没有编译错误；
21. 接下来，创建一个新的C++空项目FirstGameProject，类似于我们在引擎里面打开了一个项目，来写game code测试一下。你需要设置：
22. 项目平台位数为x64；
23. Additional Include Directory；
24. Additional Library Directory，用$(Platform)和$(Configuration)；
25. Additional Dependencies；
26. Forced Include File；
27. Calling Convention；
28. Language Standard C++17；
29. 写一个first\_game\_project::character\_script类，这个就是一个游戏实体的script脚本代码，C++语言；
30. 回到我们的Engine项目，创建Core/main.cpp，写main函数作为当前FirstGameProject项目的函数入口点。我这里Engine项目是vs中的Static Library项目，它本身并不需要主函数入口点；
31. Build一下，看看是不是生成了exe程序，那就是第一个游戏程序；
32. 我们在FirstGameProject里面写的character\_script类，Engine本身并不认识它，所以也没办法创建这个类的实例作为一个游戏实体的脚本组件。为此：
33. 写一个primal::script::detail::register\_script函数，这个函数做的事情就是将类名（这里利用hash转换为size\_t整数类型）和脚本创建函数（函数指针类型）作为键值对插入到一个unordered\_map当中去（命名为reg），注意这个reg要作为静态字段使用；
34. 为了实现在程序开始时，就完成这个脚本创建函数的注册，可以通过（为函数设置返回值，返回给一个全局变量）来实现。同时，register\_script并且简单返回一个u8值本质上只有0和1表示注册是否成功；
35. 特别注意，reg是一个静态字段，register\_script函数在程序启动时就被调用，reg也是在程序启动时就被初始化，我们无法保证他们的初始化顺序，为此，需要通过一个小技巧来声明和初始化reg变量；
36. 编写代码时，多使用一些using来为某些长名变量重命名；
37. 将上述注册脚本创建函数的代码内容放在Engine/Common/CommonHeaders.h中，利用宏来简化代码书写。这里你可能会用到字符串化操作符和连接符；
38. 为character\_script写构造函数，重写update函数，先什么都不做；

# Game Engine Programming 017 - Generating the Visual Studio solution for game code | C++ Game Engine

接下来几节的目标是实现图中架构的红色矩形内的部分；



1. 修改一下命名空间问题，primal::internal改为primal::detail，保证命名空间的一致性；
2. 上一节中，我们将将脚本类的注册函数写在头文件中，宏里面有匿名空间，那么包含这个头文件的任意文件都可以用这个匿名空间的变量了，这并不符合匿名空间的本意，我们应该将匿名空间放到cpp源文件中。为此，你移动一下宏的位置；
3. 我发现一个错误，请你修改一下Forced Include File，我选的头文件不对；
4. 为FirstGameProject项目添加两个Platform配置，称为DebugEditor和ReleaseEditor，记得设置copy setting；
5. 这两个Platform配置，都修改Configuration Type选项；
6. 这两个Platform配置，添加Preprocessor Definition，包括USE\_WITH\_EDITOR和\_WINDOWS（这个可以不需要，因为我们后面设置了subsystem为WINDOWS），还可以删除\_CONSOLE这个宏；
7. 接下来要做的事，就是利用我们的Editor来创建一个类似上述的FirstGameProject，为此，我们需要阅读一下VS的.sln文件，看看哪些内容是通用的，哪些内容是需要依据实际项目配置的。把文件内容进行string format，然后保存一份到PrimalEditor/ProjectTemplates/EmptyProject中，命名为MSVSSolution。为了完成这个任务，你需要了解一下msvc的sln文件结构；

你可以在此处记录一下需要format哪些内容：

{0}: Project GUID;

{1}: Project name;

{2}: Solution GUID;

1. 同样的，.vsxproj文件也需要进行string format，你也需要了解一下这个文件的结构。这其中你可能需要自定义UserMacros,你需要了解一下怎么在.vsxproj文件中自定义UserMacros。视频中添加了头文件包含目录宏和静态库（Engine.lib）包含目录宏这两个；

你可以在此处记录一下需要format哪些内容：

{0}: Project GUID

{1}: Project name

{2}: PrimalIncludePath

{3}: PrimalLinkPath

1. 记得设置一下subsystem为WINDOWS，而不是CONSOLE。这个选项的功能是让链接器选择正确的主函数入口点；
2. 视频中关闭了RunTimeTypeInfo，你也可以关闭；
3. 另存这个.vsxproj到PrimalEditor/ProjectTemplates/EmptyProject，命名为MSVSProj；
4. 接下来你要回到PrimalEditor项目中，实现功能：当打开一个项目，创建相应的GameProject（VS解决方案），做法是利用先前保存好的项目模板，填充好字符串（之前做了string format），保存到游戏项目的根目录的正确位置。这个过程中你可能会用到生成GUID的C#类，你自行检索；
5. 接下来做一个功能，当程序启动的时候，通过环境变量来查找Primal引擎的安装目录（我们一般认为，安装引擎的时候，就已经设置好了环境变量），如果没有找到，就通过弹对话框（新建WPF window）的方式来让用户输入目录地址，然后设置环境变量。你来完成对话框样式的设计，和projectBrowserDlg类似即可，很简单。另外，可以设置对话框不出现在任务栏，设置不能缩放，以及设置启动位置；

判断一个路径是否有效包括：路径设置了没有？路径合不合法？路径存不存在？

# Game Engine Programming 018 - Adding source files using Visual Studio automation | C++ Game Engine

1. 创建静态类GameDev/VisualStudio；
2. 为了能在代码中操作VisualStudio项目，我们为PrimalEditor项目添加COM Reference，添加EnvDTE和EnvDTE80（注意添加时找到正确的名称）；
3. 为静态类VisualStudio中写一个OpenVisualStudio函数。它的逻辑就是，首先检查Editor是否已经拥有了对于一个VS窗口的实例，如果没有，我们去查找任务栏，看看有没有已经在运行的VS窗口是打开的我们的游戏项目，有则获得其实例，如果也没找到，我们就创建一个VS实例。这个过程的代码可能需要你自寻检索；
4. 关于查找任务栏中有没有正确的VS窗口，需要用到IEnumMoniker（.Next方法遍历），IMoniker（.GetDisplayName()），IRunningObjectTable (.EnumRuning方法，GetObject方法)，IBindCtx，还有来自Windows的Ole32.dll文件中的GetRunningObjectTable函数和CreateBindCtx（你需要查阅一下这个函数）。对比找到的VS实例是不是打开的当前游戏项目，我们利用解决方法名称来判断。此外还需要调用释放这些COM对象；
5. 写一个CloseVisualStudio函数，关闭VS窗口。逻辑就是，如果VS窗口已经打开了解决方法，那么就调用ExecuteCommand保存，然后关闭解决方案，再Quit掉窗口；
6. 新建一个NewScriptDlg的WPF Windows，就是一个对话框，让用户输入Script name和Path，然后创建.h和.cpp文件。这个窗口的开发涉及：
7. 检查脚本名是否有效，要检查是否为空，是否有空格，是否包含非法字符（非法文件名字符）；
8. 检查路径是否有效，要检查是否为空，是否包含非法字符（非法路径字符），路径是否在./GameCode路径下（不一定需要），脚本文件是否已经存在；
9. 写好提示信息；
10. 创建按钮的相应函数写为异步的；
11. 在进行字符串格式化时，注意{}的使用；
12. 在脚本代码中，命名空间的名称和项目名称相同，但是要注意，项目名称是可以有空格的；
13. 我们认为输入框输入的路径是相对于当前项目路径的相对路径；我设计成是相对GameCode目录的相对路径
14. 向VS项目添加.h和.cpp文件的时候，VS可能因为繁忙而添加不上，我们项目中的做法是每一秒中尝试添加一次，一共尝试三次；
15. 添加.h和.cpp文件的操作，首先要在正确的路径上添加，然后还要添加到VS项目中去，这通过COM对象来完成，所以，要先打开VS才能完成添加，而且还要确保打开了当前解决方法，添加文件前最好先保存一下；
16. 解决方法可能有多个项目（用户可能自己添加了一些），我们要找到项目名称和当前项目名称相同的那一个（遍历\_vsInstance.Solution.Projects），然后为这个项目添加文件，添加后在用VS打开这个cpp文件，激活主窗口，设置Visible等（目前还不知道Activate和设置visible的效果是什么，可以测试一下）；
17. 在Project.Unload()函数中，同时调用刚刚完成的CloseVisualStudio函数；
18. 测试一下添加脚本的功能，通过添加按钮来完成；
19. 添加一个小功能，为NewScriptDlg添加一个BusyAnimation，当添加脚本的时候，会Disable整个界面，然后运行这个小动画，直到脚本加载完毕，关闭这个动画。启动动画和关闭动画的过程，通过给Opacity属性设置DoubleAnimation实现；

# Game Engine Programming 019 - Building and loading the game code DLL | C++ Game Engine

1. 添加一个按钮，叫Build，也就是为游戏项目进行构建；
2. 添加一个Combobox，表示构建的Configuration，有Debug和Release两种；
3. 对于这个Combobox的两种选项，SelectedIndex需要绑定到Project的一个属性中去，命名为BuildConfig；
4. 写一个BuildCommand，Command中执行的内容先完成编译游戏代码构建DLL的功能，其逻辑就是：先UnloadDll，把Eidtor中已经链接的DLL卸载，然后通过游戏代码又重新构建DLL，如果构建成功再LoadDll；
5. 在VisualStudio类中，我们完成构建游戏代码的流程，写函数BuildSolution。其逻辑是：
6. 首先确定一下VS是不是在做什么会影响到项目构建的工作，首先比如VS是否在调试中，我们会用\_vsInstance来检查（你来查询如何判断是否在调试中），我们的做法是轮询三次，每次等一秒，如果三次查询VS都在调试，那么就无法构建游戏代码了；
7. 如果上述检查通过了，VS没有在调试中，也就是说我们可以构建游戏项目。那么首先打开VisualStudio程序；