**《介绍》**

c++学习笔记

**《环境》**

cpp版本：不同的cpp版本的语言规范不一样，要额外注意。vs2022中右键项目名称->常规->c++语言标准

**vs2022**

专业的cpp开发软件，支持虚幻5等游戏开发，自带cpp编译环境方便使用

安装：教程网上搜索，安装社区版，选择需要安装的组件，可以安装到非C盘，如果安装时不能更改盘，通常是之前安装过vs，需要搜教程把原来的残留物清理干净

创建项目：在vs2022中新建cpp项目后，文件夹中会自动生成.vs、项目名.sln、项目名.vcxproj、项目名.vcxproj.filters、项目名.vcxproj.user配置文件，.sln是vs2022的打开入口。vs2022中会有引用、外部依赖项、头文件、源文件、资源文件5个内容，在源文件中写项目代码

快捷键修改：[工具]->[选项]->[键盘]，输入快捷键功能查找并修改，格式化的功能名称为‘编辑.设置文档的格式’

**vscode**

使用vscode的优势在于vscode体量小、c++和python代码可以一起开发

vscode下载：https://code.visualstudio.com/

环境配置：下载mingw，在vscode插件中下载c++插件，搜教程配置编译环境。配置时注意使用g++替代gcc

MinGW：c++开发工具，包含GCC编译器，安装c++环境需要下载MinGW

MinGW下载地址：https://github.com/niXman/mingw-builds-binaries/releases

c\_cpp\_properties.json：配置好c++插件后会自动在.vscode下生成的环境配置文件

tasks.json：编译时的配置文件。自动生成：[终端菜单]->[配置文件]->[选择MinGW环境]。文件中的args参数的-g为debug编译(默认)，改为-02为release编译

编译：[终端菜单]->[运行生成任务]->[选择MinGW环境]。会生成.exe文件

运行：在下方终端中输入[.\文件名.exe]运行

vscode运行：点击[运行和调试]后会自动编译和运行，并且如果有断点会直接调试。快捷键[ctrl+f5]

新项目：创建新项目时，可以复制原来的.vscode配置文件

多文件编译：多文件编译(不同文件间函数调用)时要改造tasks.json

格式化：c++代码按ctrl+s保存时会自动格式化

Chinese插件：中文翻译插件

Remote-SSH插件：ssh远程连接服务器插件。安装后需要配置一下服务器的信息

PyCharm Theme 插件：将vscode风格换成pycharm

JetBrains IDE Keymap插件：将vscode快捷键换成pycharm

**其他**

cmake：开源、跨平台的项目构建系统，支持多种操作系统和编译器。比如从github下载和安装库，可以通过cmake的方式构建来避免环境影响

**《c++基础》**

**基础知识**

c++标准：c++语言规范，国际标准化组织制定，c++标准和编译器都在不断更新

c++库安装：由于c++的编译器没有统一的标准，导致很多库不通用，没有像python的conda一样的库管理工具，要手动下载和管理c++的库

编译器：c++语言需要编译器转换为对应系统的可执行文件，比如windows上编译后为.exe文件。常见的编译器有GCC、Clang、MSVS等。常用的编译器为GCC，支持多种系统和最新c++标准。windows上可以用vs2022(自带环境)或下载MinGW开发工具(包含GCC编译器)来配置c++环境，linux上通常自带GCC编译环境

GCC：GCC编译器有gcc、g++两种，gcc针对c，g++针对c++

编译模式：编译器会将代码编译为可执行语言，有debug编译和release编译两种方式。前者包含调试信息，后者是经过大量优化的最终版本

字符串：c++中要使用双引号

静态库：.a或.lib。程序使用静态库时，会复制静态库的内容到系统上。优点是效率高，跨系统时不依赖环境；缺点是占用空间大，每次调试代码时要重新编译静态库

动态库：.so或.dll。动态库存在于程序外部，供程序调用

cpp中的函数、条件语句的{}后面可以不加;

注释符：//。注释符前后不需要加空格，但习惯[2空格+//+1空格+内容]

赋值内存：cpp中的基本类型(包括数组)在赋值auto A=B;时会单独创建内存，只是少数的对象和类示例在修改后才会共用内存

结束符：cpp中的函数、条件语句的{}后面可以不加结束符;

**项目结构**

一个项目只有一个入口，编译后只有一个.eve，即只有一个含主函数main的cpp文件。如果只对想某个模块进行测试，要新建项目或修改main函数位置或使用单元测试

导入库：比如#include <iostream>，导入外部库时需要设置好系统路径等才能找到

代码结构：比如有文件main.cpp、block.h、block.cpp。main中存放main函数，写入#include "block.h"；block.h写入#pragma once(防止重复包含)、函数声明、类(类中函数只写声明)；block.cpp写入#include "block.h"、函数、类函数的实现方式

分离编译：编译时.cpp 文件会独立编译成.obj 文件，最后链接成可执行文件

头文件(.h)：文件对外暴露的接口，需要写入函数声明、类结构以供main.cpp识别链接

源文件(.cpp)：写入函数具体执行的代码，以供头文件使用

**数据类型**

不同编译器下的数据大小可能有区别，这里以常见的为例

字节：1个字节有8位二进制空间

short：短整型。2字节，-32768到32767

unsigned short：无符号短整型。2字节，0到65535

int 整型。4字节，-20多亿到20多亿

unsigned short：无符号整型。4字节，0到40多亿

long：长整型。4/8字节，64位linux为8，其他为4

unsigned long：无符号长整型。4/8字节，64位linux为8，其他为4

long long：长长整型。8字节

unsigned long long：无符号长长整型。8字节

float：单精度浮点数。4字节，7位有效数字

double：双精度浮点数。8字节，15位有效数字

long double：扩展精度浮点数。16字节，19位有效数字

Void：无类型。0字节

Bool：布尔值。1字节

指针(\*)：4/8字节。32位系统为4，64位系统为8

枚举：变量默认4字节，可以定义数据类型

联合体(union)：所有成员共用一个内存

引用(&b=a)：引用成员的数值和内存地址都是同一个

**基本操作**

int a=1;：定义整型。也可以int a;//a=1;

char A="1";：定义字符串

&a：a的地址。所有的数据都有值和地址两个属性

const int a=1;：定义常量，之后a的值不能改变

constexpr int a=1：编译时常量。在编译时会直接将a替换为1，性能比const高，但由于在编译时优化，不能包含函数调用和对象构造

auto 变量=A：定义变量时替换原类型，会自动推导类型，不影响效率。auto全部功能要在c++20以上使用。通常在A为一个复杂的结构时使用。使用lambda表达式时可以用auto定义。可以定义函数，定义函数参数要c++20以上

namespace 命名空间{各类函数}：将各类函数放到命名空间下，可以嵌套。比如std::cout为使用std中的cout

namespace name=命名空间：给命名空间起别名。通常为了避免命名空间冲突，命名空间名称会很长，使用时再用别名来简化

a++：自加1

a--：自减1

abs(a)：取绝对值

sizeof(A)：A的字节大小

swap(A,B)：交换A,B两个数的值。A和B要为同一数据类型

floor(a)：向下取整，a可以为整数或浮点数

ceil(a)：向下取整，a可以为整数或浮点数

static：定义静态方法。加在全局变量前时，外部程序无法调用；加在非类函数中时，变量只会定义一次，函数结束时也会一直存在，下次函数调用时会用上一次的值；加在类的变量前时，所有类的对象都共用一个变量值，会同时改变；加在类的函数前时，该函数为静态函数，不需要初始化类可以直接调用：类::函数

**指针**

指针类型：不同指针类型的区别在于解引用\*p的处理方式不同，转换时只改变处理方式。void\*无类型指针由于不知道解引用的方式，因此不进行解引用和算术运算

int\* p=&a;：定义指针。注意定义后的指针是p不是\*p，指针p有自己的地址，值为其他数据的地址，\*p为a的值。即：&p=未知，p=&a，\*p=a。p+1指向a地址傍的下一个地址

const int\* p=&a;：指针常量。指针的对象可以改变，对象的值不能改变

int\* const p=&a;：常量指针。指针的对象不能改变，对象的值可以改变

const int\* const p=&a;：常指针常量。指针的对象和的值都不能改变

int\* p=&array[0]/array;：定义1维数组行指针。p+1=&array[1]

int(\*p)[i]=&array;：定义1维数组指针。p+1=&array+1，p[0]+1=p[1]=array[1]

int(\*p)[i]=&array[0]/array;：定义2维数组行指针。p+1=p[0+1]=&array[1][0]，p[0]+1=&array[0][1]

int(\*p)[i][j]=&array;：定义2维数组指针。p[0][0]+1=p[0][0+1]=&array[0][1]

**智能指针**

智能指针和new在编译优化后的汇编代码是一样的，因此性能相同

auto\* p=std::make\_unique<类型>(值);：堆上创建数据

auto\* p\_str=std::make\_unique<std::string>("text");：堆上创建string数据

auto\* p\_array=std::make\_unique<int[]>(3,{1,2,3});：堆上创建并初始化数组

auto\* p\_class=std::make\_unique<a\_class>();：堆上创建实例化类

p\_int.reset();：释放数据

**数组**

cpp内置数组无越界检查和更多功能，通常静态数组会用std::array，动态数组会用std::vector。数组的内存连续，高维数组中低维度的内存连续，比如array[2][3]中，&array[0][2]的下一个地址为&array[1][0]

int array[i];：定义静态数组。数组名&array[0]=array=&array，array+1=&array[0]+1，&array+1=整个数组的后一个地址。sizeof(array)=sizeof(&array[0])=4

int array[i][j];：2维静态数组。数组名&array[0]=array=&array=&array[0][0]，&array[0][0]+1=下一个元素地址，array+1=&array[0]+1=下一行首地址，&array+1=整个数组的后一个地址

int array[3]={1,2,3};：定义并初始化

int array[]={1,2,3};：定义时会自动推断大小

**无类型**

void\* p：无类型指针。无型指针可以随意赋值，比如int\* p1，p=p1成立，p1=p不行。按照c++标准无类型指针不要进行p++等运算操作，要先转换类型

void 函数名：无类型函数，不需要return返回值

void\* 函数名：函数的返回结果为无类型指针

**堆**

通常将大数据和不确定大小的动态数据放在堆上。为了方便内存的释放，优先使用智能指针std::make\_unique和std::make\_shared。动态数据优先使用std::vector

int\* p=new int：创建堆指针并在堆上创建int数据。堆上内容需要手动释放：delete p

int\* p\_array=new int[i]：堆上创建数组。释放：delete[] p

a\_class\* p\_class=new a\_class(...)：堆上创建实例化类。释放delete p\_class

**条件语句**

if (条件){内容}else if (条件){内容}else{内容}：条件判断语句

for (int i=0;i<a;i++){内容}：for循环

while (条件){内容}：while循环

**字符串**

string定义的字符串是一个封装的对象，可以执行更高级的操作，比char更好用

string str\_;：定义空字符串

str2=str1+"...";：合并字符串

str\_.insert(a,"A");：在下标为a的字符前插入字符A

str\_.replace(a,b,"A");：从下标a开始往后共b个字符替换为A

str\_.substr(a,b);：选取下标a开始往后共b个字符

str\_.erase(a,b);：删除下标a开始往后共b个字符

str\_.find("A");：查找是否含有A，存在返回第一个字符的下标，不存在返回-1

**位运算符**

优先级：算术运算符>关系运算符>位运算符>逻辑运算符

a<<n：二进制左位移n次。等于乘以2的n次方，运算效率更高

a>>n：二进制右位移n次。等于除以2的n次方，运算效率更高

&：二进制与

|：二进制或

^：二进制异或

~：二进制非

**逻辑运算符**

&&：与

||：或

!：异或

**函数**

类型 函数名(类型 参数=默认值){...}：定义函数。参数和传入参数的类型要一致

void 函数名：无类型函数，不需要return返回值

int/float 函数名：整型/浮点型，定义了return返回的数据类型

static类型 函数名：定义静态函数。类中的静态函数不需要初始化类也可以调用

inline void 函数名：内联函数。在执行代码时会将函数中的内容作为一个单独的模块，如果展开后与调用代码合在一起编译会更简洁、提高运行速度，但有for循环等会导致代码体积膨胀，导致编译后文件变大，因此对于复杂的函数不使用。相比类似展开的宏定义，inline有类型检查更安全

函数重载：允许有相同名称的函数，但它们的参数类型要不同。比如void 函数(int a)和void 函数(float a)，当传入参数不同时会调用不同的函数

**结构体**

结构体中可以定义函数，结构体和类基本功能一致。通常只有变量时使用类

struct structure{public:int a...;float b...; };：定义结构体。结构体中有公有函数public、保护函数protected、私有函数private，不写时默认公有。public就是正常函数；protected只供函数内部和子类使用，private只供函数内部使用

struct structure:public 父结构体{int a...;float b...; };：结构体继承

structure A;：结构体实例化。A={...}可以依次赋值

A.a：使用结构体中的元素

**类**

class 类名{public:...}：定义类。类中有公有函数public、保护函数protected、私有函数private，不写时默认私有。public就是正常函数；protected只供函数内部和子类使用，private只供函数内部使用

类名 示例名;：实例化类，不加括号。当有构造函数且有参数时为：类名 示例名(参数);

类名.元素：使用结构体中的元素

类型 函数(...){...}：类中定义函数。可以只写一个声明，在类外部再定义：类型 类::函数(...){...};。在类中定义的函数都为内联函数，在外部定义需要加inline

类型 函数(...) const{...}：如果函数不会改变类中的变量，按规范要在后面加const。同时const修饰函数中不能调用非const修饰函数

构造函数：即初始化函数。不需要定义类型，写在public中任意位置，实例化时会最后执行。类中可以有多个构造函数，但只根据函数重载使用一个。如果父类和继承类都有，则会先使用父类的再使用子类的。无参数传入时：int 参数; 类名():参数1(初始值),参数2(初始值){...}。有参数时：int 参数; 类名(int 参数):参数(初始值){...}。父类有参数时：int 参数; 类名(int 参数):父类名(父类参数),参数(初始值){...}

析构函数：用于自动销毁内容：~类名(){销毁内容}。当实例类的生命周期结束时(比如类不再使用且遇到了后大括号})会自动运行析构函数。如果父类和继承类都有，则会先使用子类的再使用父类的

class 类名:public 父类1, public 父类2{...}：定义继承类。public表示继承父类的public中内容；如果是protected表示继承public和protected中的内容；如果是private表示继承所有内容，但父类private的内容不可访问。子类的子类会继承之前的所有父类元素。子类元素和父类同名时，父类元素会隐藏

父类::重写函数(...);：子类重写函数时，在内容前加上表示先执行父类的同名函数

**虚函数**

多态：允许使用基类指针或引用来调用子类的重写函数，通过基类来管理子类，方便类指针的定义。通过virtual实现多态。没有多态时，{父类\* p=&子示例化类;p->函数;}使用的是父类函数，要改为{子类\* p=&子示例化类;}，但当子类较多时扩展性不好；有多态时，前者使用的是子类函数

virtual 类型名 函数(...){...}：定义虚函数。父类有虚函数时，子类重写实现多态

virtual 类型名 函数()=0：定义纯虚函数。类中有纯需函数时无法实例化，只能作为父类，而且子类中必须重写该函数，纯虚函数通常用来限定子类的内容规范

类型名 函数(...)override{...}：重写函数标志。不加override也可以，但不方便可视化维护，同时如果重写函数名写错时不加会当做正常函数，而加了必须为重写，防止出错

virtual 类型名 函数(...)override{...}：既重写父类，也让子类继续重写

**《c++常用库》**

**<stdio.h>**

c语言的标准输入输出库，有printf、scanf,、puts,、gets等函数，c++也可以使用，但不支持面向对象编程、不提供类型安全。c++有自己的<iostream>库，提供更丰富的功能

**<iostream>**

#include <iostream> // using namespace std;

c++的标准输入输出库，早期版本使用<iostream.h>

std::cin>>x;：从键盘输入数据。x为提前定义的数据类型

std::cout<<x<<std::endl;：打印输出x，<<endl为换行符可不加

**字符串**

std::string为处理字符串的库，不用关心内存分配和\0

std::string str\_;：定义空字符串

getline(cin,str\_);：输出带空格的字符串

str3=str1+“ “+str2;——字符串合并，中间可加空格

str\_.insert(a,”A”);——在str\_下标为a的字符前插入字符A

str\_.replace(a,b,”A”);——从下标a开始往后共b个字符替换为A

str\_.substr(a,b);——选取下标a开始往后共b个字符

str\_.erase(a,b);——删除下标a开始往后共b个字符

str\_.find(“A”);——查找str1中是否含有A，存在返回A第一个字符的下标，不存在返回-1

**<map>**

std::map<std::string,int> dict{{"key",value}};：创建有序字典。按大小排序

std::map<std::string,std::map<std::string,int>> dict{{"key",value}};：嵌套

dict["key"]=value;：增加键值对

dict.empty()：判断字典是否为空，空为1

dict.size()：返回字典的键值对数量

**<unordered\_map>**

std::unordered\_map<std::string,int> dict{{"key",value}};：创建无序字典

**<array>**

静态数组，相比内置数组有更多扩展。std::array和结构体的效率基本一样快，内存连续，大小一样，可视化时用结构体更好

std::array<int,3> array={1,2,3};：定义静态数组

std::array<std::array<int,3>,2> array={{{1,2,3},{4,5,6}}};：定义2维静态数组，注意要多加一个{}

array[i]：获取元素

array.at(i)：获取元素并进行越界检查

array.front()：第1个元素

array.back()：最后1个元素

array.size()：数组的元素个数

array.fill(i);：用指定值填充数组

**<vector>**

动态数组，相比列表效率。更高、空间更小。相比std::array的效率略慢，内存连续，大小略大，支持动态扩容，但不要频繁扩容

std::vector<int> vector={...};：创建动态数组

std::vector<std::vector<int>> vector={{...},{...}};：2层嵌套动态数组

vector.reserve(100);：提前预留扩容空间

vector.capacity();：当前容量，没有预留时为创建时的大小

vector.size();：列表大小

vector.empty();：判断是否为空。空返回1

vector.clear();：清空列表

vector.back(A);：最后面的元素

vector.front(A);：最前面的元素

vector.push\_back(A);：在最后面添加元素

vector.push\_front(A);：在最前面添加元素

vector.pop\_back(A);：删除最后面的元素，没有返回值

vector.pop\_front(A);：删除最前面的元素，没有返回值

vector.remove(A);：删除所有值为A的元素

auto i=vector.begin();：获取迭代器指向列表的第一个元素

**<list>**

双向链表，元素在内存中非连续存储，通过指针连接。考虑到效率通常用vector更好，对list的操作函数基本与vector一致

std::list<int> list={...};：创建列表。不可以嵌套

std::list<std::list<int>> list={{...},{...}};：2层嵌套列表

**<queue>**

std::priority\_queue<int> queue;：最大堆优先队列

std::priority\_queue<int,std::vector<int>,std::greater<int>> queue;：最小堆

queue.top()：顶部元素

queue.push(A)：添加元素

queue.pop()：删除顶部元素，没有返回值

**嵌套**

struct node {int distance,...; };：定义结构体嵌套

auto lambda=[](const node& node\_old,const node& node\_new){return node\_old.distance>node\_new.distance;};：定义队列的比较方式，return为1时放到堆顶

std::priority\_queue<node,std::vector<node>,decltype(lambda)> queue(lambda);：创建嵌套优先队列并应用lambda关系式。node为队列的元素类型；std::vector为队列的类型；decltype自动获取lambda的类型；queue(lambda)为队列应用lambda表达式

**<Eigen/Dense>**

#include <Eigen/Dense> // using namespace Eigen;

vscode安装教程：<https://blog.csdn.net/qq_45948208/article/details/132434560>

Eigen是数学计算库。c++的eigen、numcpp和python的numpy库计算效率相差不大，单纯矩阵计算eigen>numcpp>numpy，动态矩阵计算(矩阵大小经常改变)numpy>numcpp>eigen

**<opencv2>**

c++版本的opencv函数效率和python版本的基本一致，但环境安装等比较麻烦，因此可以用python写相关函数，然后用c++调用python代码

**<python>**

#include <Python.h>

vscode安装：先安装anaconda或miniconda环境，下载python插件

vscode配置：找到python环境的include位置，在配置文件c\_cpp\_properties.json的includePath中添加"D:\\anaconda3\\include"，然后在tasks.json的args中添加"-I"和"D:\\anaconda3\\include"、"-L"和"D:\\anaconda3\\libs\\\*"

Py\_SetPythonHome(L"D:\\anaconda3");：设置python库根目录

putenv("PYTHONIOENCODING=utf-8");：执行python程序用utf-8编码，放在初始化前

Py\_Initialize();：初始化

PyRun\_SimpleString("import py库");：执行python命令

**《ue5》**

**基础知识**

安装：先下载epic games平台，在epic games平台中可以创建和下载不同版本的虚幻5引擎，初次安装时不需要选目标平台(文件很大)

UE官方：<https://dev.epicgames.com/documentation/en-us/unreal-engine>

功能搜索：可以使用中文搜索相关功能，然后找到相关蓝图和cpp函数的使用方法。在事件图表中双击蓝图可以查看cpp函数。右键组件打开头文件可以查看cpp函数

功能实现：UE中实现同一功能通常有多种方法。比如视角功能写在玩家空间控制器或pawn中都可以实现，获取玩家输入也有多种方式，需要根据复用性、维护性等自行决定

游戏组成：虚幻5中一个游戏由多个关卡组成，开发者先制作每个关卡然后拼到一起

关卡(地图、世界)：游戏世界的基本单元，独立的场景容器，每个关卡可以包含地形、光照、人物、UI、逻辑等元素，所有物体称为演员(actor)。比如主菜单可为一个纯UI关卡

演员(actor)：由多个组件组成，在细节中可以看到是某个cpp类的实例

组件：组件可以为静态网格组件、点光源组件、自定义组件(蓝图类或cpp类)

静态网格组件：网格模型，可以加上材质，材质中包含贴图和渲染设置

事件图表：通过事件图表将组件、演员、关卡等联系起来，组成完整的游戏逻辑

蓝图：可视化编辑，蓝图和蓝图类的概念有重合，与之对应的是cpp类。蓝图类由蓝图虚拟机读取并转为字节码逐条解释执行。可以用cpp类替代蓝图类来提高效率

子蓝图类：蓝图类右键可以创建子蓝图类，继承蓝图类的功能，可以修改定义的参数

编辑：蓝图编辑后要点击左上的编译才会生效

坐标轴：X轴是红色，Y轴是绿色，Z轴是蓝色

坐标单位：坐标单位是cm，地形中一个方块为1平方米

文件移动和删除：虚幻5中只会对部分文件进行操作，如果操作后原来的文件还存在，是因为有zip等文件(但虚幻中看不到)，需要进入电脑文件夹后删除(右键在浏览器中显示)

虚幻商城：有免费的素材资源可以下载

cpp版本：UE5使用c++17版本，但通常尽量使用UE5提供的工具链

**蓝图类型**

actor、pawn、character三者为继承关系，理想情况下尽量使用最少的功能

演员(actor)：继承Object，场景对象的基类，只有静态网格等少量组件，常用于场景中静态物体。防御塔等建筑物可以在其基础上实现。创建：项目文件右键->蓝图类->actor

pawn：继承actor，增加可操作的组件，常用于玩家视角控制。创建：项目文件右键->蓝图类->pawn

角色(character)：继承pawn，增加复杂移动、导航网格寻路、多人游戏同步等组件。创建：项目文件右键->蓝图类->角色

HUD蓝图：屏幕上叠加显示的非游戏世界元素，比如玩家生命中和小地图。创建：项目文件右键->蓝图类->搜索HUD

蓝图接口：用于蓝图之间的通讯。使用类型转换(cast to)会加载蓝图，消耗资源。创建：项目文件右键->蓝图->蓝图接口

**制作流程**

游戏设计：确定游戏玩法规则

资源规划：制作或购买单位模型、材质、音效、动画

技术选型：用蓝图制作、c++编写寻路等高性能模块、mass ai框架管理大规模单位

玩家控制：玩家镜头视角、选中单位、下达指令

UI系统：主菜单、资源面板、单位面板、小地图。通过UMG控件蓝图和逻辑制作

地图设计：绘制地形、植被、遮挡碰撞体

单位：导航网格控制行走区域，行为树和黑板实现行为逻辑

关卡连接：将主菜单、关卡之间连接起来，保存玩家进度，跨关卡传递数据

优化测试：nanite处理高模地形和建筑，lumen光照设置，HLOD(分层细节优化)优化远距离单位渲染，测试单位寻路和多人同步

打包发布：打包游戏并发布到目标平台(手机或电脑)

**快捷键**

视角移动：按住鼠标左/右键进行前后和旋转，此时使用wasd和eq可以快速移动，按住鼠标的同时滑动滚轮可以修改移动速度，视角移动速度可以在视口的右上角看到

物体调整：q选择；W移动；e旋转；r缩放。移动物体时按住shift保持视角跟随

F：视角跳转到选择的物体

Delete：删除

FN+END：物体对齐到地面上。地面不平整或有其他物体时，会对齐到最高点

CTRL：按住CTRL拖动物体时，物体会吸附在表面上

Ctrl+D：在旁边复制一个相同的物体

Alt：按住Alt拖动物体时会复制并拖动新的物体。细节中选物体(默认)时复制的是演员，选组件时复制的是组件

鼠标中键：单个或多个物体进行操作时，对坐标轴按住鼠标中键可以临时移动坐标轴点

**性能**

蓝图与c++：每帧都进行的高频计算、寻路算法用c++重写可能会提高10倍以上速度

启用垂直同步(VSync)：[编辑]->[编辑器偏好设置]->[性能]

垂直同步(VSync)：使游戏画面与显示器刷新率一致，限制GPU性能，防止在复杂场景高速移动时画面撕裂。可能会增加输入延迟，在竞技类游戏中不使用

绘制调用(Draw Call)：ue5中每个物体渲染时CPU会调用图像程序(可能为GPU)，程序会进行切换渲染状态(材质、着色器、贴图)、加载顶点数据(位置、UV、法线)、顶点着色器和像素着色器计算、输出像素到屏幕。将多个物体合并为同一个物体会提高效率，但如果各物体的材质不同且复杂、单个物体顶点太多(30-50万以上)则反而会降低效率

物体合并：若要进一步提高效率，可以去除重叠看不到的面、简化所有面的顶点数

**页面**

项目文件：左下角[内容侧滑菜单]。可以查看项目中所有保存的关卡和资产

顶部选项：[文件]为保存相关操作；[编辑]对整个项目进行设置；[窗口]打开窗口

工具栏：对当前的关卡进行修改

视口：屏幕中间的画面窗口为一个视口。在上方[窗口]->[视口]中打开。关卡画面

大纲：右上窗口。在上方[窗口]->[大纲]中打开。关卡中所有的内容

细节：右下窗口。在上方[窗口]->[细节]中打开。关卡中选中对象的属性编辑

世界场景设置：默认没有，可以从右上[设置]中添加。设置当前关卡

**项目文件:**

新创建或复制的内容会带上\*号，此时未保存，保存所有后会消除\*号

关卡：一个关卡为一张地图，地图保存后会得到关卡文件和HLOD0层文件

**工具栏:**

有保存、选项模式、添加图标、蓝图图标、导演图标、运行关卡、平台等选项

**视口:**

帧数：FPS为每秒画面数，ms为FPS的倒数。FPS以30为基础(单机游戏)、60为流畅(3A游戏)、90以上为VR游戏、120以上为电竞级别，通常单机游戏不到30帧也能玩，会受电脑性能、复杂场景、优化和渲染技术(nanite,lumen)、垂直同步(VSync)

距离测量：[透视]->[顶视图]，按住鼠标中键测量距离(cm)

表面对齐：右上角[表面对齐]，开启后拖动物体和另外一个物体碰撞时会附着在表面，但最好是精确计算坐标来对齐

对齐网格：右上角[自动对齐网格]开启后，会以固定距离(cm)移动物体。还有固定角度旋转、固定缩放距离等

多视图：右上角开启多视角窗口，可以看到不同视角或维度的窗口

**大纲:**

HLOD0文件夹：包含HLOD(分层细节优化)的配置

Lighting文件夹：包含基本的光线、天气等。在环境光源混合器中统一编辑环境效果

Landspace：包含所有的地形区块

PlayerStart：开始关卡后玩家的起始点

WorldDataLayers-1：世界场景内容管理的配置

WorldPartitionMiniMap：世界分区缩略图的配置

**地形**

管理：创建新的地形。分段大小是每个区块的大小，组件数量是创建的区块数

雕刻：绘制地形的形状。平整时设置扁平目标为0可以压平地面

材质：在项目文件中可以创建材质，在物体细节中可以应用

自由绘制图层：使用材质中的LandscapeLayerBlend节点制作材质。<参考该节点资料>

**光源**

在添加图标中添加光源。添加光源后可以在大纲中选择、在细节中修改效果

环境光源混合器：在窗口中选择，在混合器中可以同时编辑光照、大气、云等各类环境效果，初始项目大纲中的Lighting文件夹中为默认的环境效果

自动曝光：项目设置中搜索自动曝光，默认开启。自动曝光可以模拟人眼的自适应，比如玩家从亮处进入暗处后，先是很暗，然后慢慢变亮

**基础设置**

创建项目：创建项目时可以选蓝图或者c++，c++需要安装编程软件和环境。创建项目后会生成一张空的关卡。每次进入项目后会生成空的关卡，点击项目中保存的关卡进行切换

蓝图项目：包含Config、Content、项目名.uproject、其他临时文件

cpp项目：在蓝图项目的基础上增加.vs、.vsconfig、Source、项目名.sln。当蓝图项目中创建c++类后会自动转为cpp项目，.vs和.vsconfig为配置文件，项目名.sln为项目从cpp软件打开的入口，Source存放创建的类(还有一些模板文件)

全中文开发：UE5对中文的支持很好，项目中的文件名和蓝图可以使用全中文，不影响打包发布，只有少数的插件不支持中文。在搜索功能组件时，由于很多组件是显示中文不准确，导致搜不到，可以右键查看实际的英文来搜索

缓存路径：编辑->编辑器偏好设置->全局->全局本地DDC路径。所有项目的缓存路径

语言设置：编辑->编辑器偏好设置->区域和语言

默认地图：编辑->项目设置->地图和模式。可以设置默认打开的地图和游戏实例等

运行窗口：在运行游戏的右侧的设置中，选择新建编辑器窗口。运行后会有单独的窗口

隐藏选项卡：每个窗口顶部的选项卡可以右键选择隐藏

窗口合并：新打开的窗口可以拖动到主页面中合并

碰撞预设：编辑->项目设置->碰撞。可以新增碰撞类型和设置碰撞预设

轴映射：项目设置->输入->轴映射。轴映射的作用是将按键输入接入事件图表中。比如向前移动设置键盘为w、缩放为1，在事件图表中可以搜索到该轴映射

**材质**

官方文档：创建节点后，右键[查看文档]可以查看详细用法

材质(material)：右键空白处选择材质。双击进入材质图表。在图表中右键可以新建节点，基础着色器为最终的输出节点。基础着色器有输入时，可以在左侧勾选使用材质属性，会将输入自动拆解到相应的通常上

材质实例：如果需要将材质应用到多个物体但只改变颜色等参数时，复制多份会占内存，创建材质实例的方式可以共用大部分相同的内容、更高效。如果在材质节点中定义了变量，则材质实例中可以修改相关的变量

注释：选择多个相连的节点右键注释，可以加框并添加注释，可以改颜色

**快捷键**

Ctrl：按住Ctrl移动连好的线

1+左键：常量

2+左键：创2值常量

3+左键：3值常量

4+左键：4值常量

**材质节点**

MakeMaterialAttributes：基础节点，材质效果通过此节点输出

Constant3Vector：常量节点，可以搜常量，或按住3+点击鼠标左键。xyz代表rgb通道，当值在0-1时表示颜色，大于1时会自发光

TextureSample：贴图节点，连在材质节点上。拖入基础贴图、法线贴图可以直接创建。基础贴图连接到basecolor，法线贴图连接到normal

TextureCoordinate：控制贴图效果。连在所有贴图节点的UV上，通常会乘一个比例

Multiply：乘法。将两个输入相乘后输出

ScalarParameter：变量参数。定义变量后，在材质实例中可以修改

LandscapeLayerBlend：地形材质混合，创建自由绘制的图层。将多个材质节点连接到混合器，在左侧的图层中可以编辑连接数量，注意新建的层需要命名。地形材质混合后的材质颜色是黑色的，需要通过地形绘制使用：[细节]->[地形]导入材质，[地形模式]->[绘制]->[层]点击[从指定的材质创建层]，对每个图层创建[权重混合层(法线)]，保存地形图层到本地，此时选择图层可以自由绘制地形的颜色，绘制后会改变本地的地形图层文件

Constant：常量

Constant2vector：2值常量。有3个接口，接口1是2个参数都传入，接口2/3对应x/y

Constant3vector：3值常量。有4个接口，可以设置为RGB颜色

Constant4vector：3个常量。有5个接口，可以设置为RGBA颜色

**地形材质**

MakeMaterialAttributes、TextureSample、TextureCoordinate、Multiply、ScalarParameter、LandscapeLayerBlend

**事件图表**

事件图表：通过事件图表将组件、演员、关卡等联系起来，组成完整的游戏逻辑

事件图表有很多类型，比如游戏实例、游戏模式、演员、组件等都有自己的事件图表

添加节点：部分节点需要拖动组件的引脚才可以添加，还有的节点需要拖动执行引脚

变量：事件图表中非固定的值可以提取为变量，变量会在演员的细节中显示，方便修改。变量中可以自定义类别，在细节中分类显示，方便修改

函数：框选编辑好的功能，右键可以折叠到函数，方便管理和复用。在函数中，拖动节点的变量引脚到函数入口，可以变为传入的变量，可以设置默认值

可视化运行：运行游戏时切换到事件图表，可以看到执行的逻辑线路

Ctrl：按住Ctrl移动连好的线

**基础节点**

三角形：节点中的三角形为执行接口，其他接口为输入和输出参数

数据类型转换：连接蓝图时，如果数据类型不一致会自动添加转换节点

EventBeginPlay：运行蓝图后执行该接口。如果事件图表在演员中，创建演员后会执行

EventTick：关卡中每一帧率都执行一次。其中Delta Seconds输出1/帧率(双精度)

PrintString：打印字符串，可以选择输出到屏幕或日志。仅限开发使用

键盘事件：按下固定键后激活事件

从类生成Actor：生成一个演员，需要生成坐标

**自身节点**

self：当前演员，拖动self后的引脚可以或者当前演员的属性

获取Actor：获取当前演员的位置、向前向量、旋转等信息

**时间节点**

延迟：时间延迟

可再触发延迟：连续激活时可以重置延迟时间

**游戏实例**

游戏实例：运行游戏后会一直执行，在其中管理游戏模式，完成跨关卡的数据传输等。创建：项目文件右键->蓝图类->所有类中搜索游戏实例(GameInstance)，不要选择平台游戏实例。应用：项目设置->地图和模式->游戏实例类

**游戏模式**

游戏模式：每个关卡都有一个游戏模式，打开关卡后会执行对应的游戏模式，多人联机中只有主机/服务器执行，在其中管理玩家控制器、游戏状态、关卡蓝图。创建：项目文件右键->蓝图类->游戏模式基础(GameModeBase)。默认应用：项目设置->地图和模式->默认游戏模式。关卡应用：世界场景设置->游戏模式重载

类设置：细节中设置类为创建的类

**RTS视角**

**pawn控制器：**

pawn控制器：控制玩家的视角。创建：项目文件右键->蓝图类->pawn

场景根组件：摄像机瞄准的目标，默认有。

弹簧臂组件：控制摄像头到地面距离，放到场景根组件中。细节中目标臂长度控制距离；旋转角度为俯视角；细节中pawn设置为玩家0

摄像机组件：玩家视角。放在弹簧臂组件下

浮动pawn移动：添加该组件后视角才可以移动

事件图表：可以将视角移动的逻辑写到pawn中，但更建议写到玩家控制器中

初始化蓝图：BeginPlay+延迟0.1(等初始化完成)+获取屏幕大小+存储变量

实时蓝图：Tick+获取鼠标位置+判断是否在屏幕边缘+添加移动输入+移动pawn

**玩家控制器**

玩家控制器：执行完游戏模式后会执行玩家控制器，多人游戏时每个玩家都有一个玩家控制器，在其中管理角色和单位。创建：项目文件右键->蓝图类->玩家控制器

**玩家输入**

输入操作：项目文件右键->输入->输入操作

**游戏状态**

游戏状态：同步关卡内的状态，由游戏模式管理

**关卡蓝图**

关卡蓝图：当其他蓝图都初始化后才会执行，每个关卡只有一个，多人联机中所有玩家都会执行。创建：在运行关卡左边的蓝图图标中打开关卡蓝图。关卡蓝图和游戏模式都与关卡一一对应，关卡蓝图通常只负责特定的剧情，由游戏模式管理

**组件**

细节或蓝图类中可以看到物体的组件，组件就是某个类的实例，可以自定义组件蓝图类。组件可以在细节中修改值，细节中的值为默认值，在事件图表中有修改会覆盖

创建和复制：创建和复制新物体时，可以在大纲中创建，也可以在组件中创建，但组件的方式会归为一个物体，提高复用和效率，通常连接在一起的物体都放在一个组件里

静态网格体组件：包含演员的基本属性。可以替换静态网格体、材质。物体坠落：开启变换中的可移动性、物理中的模拟物理

重力：静态网格体中，可以启动或关闭重力

碰撞：静态网格体中，默认BlockAllDynamic有碰撞；NoCollision无碰撞；Custom自定义碰撞

发射物移动组件：发射子弹的效果，拖入到事件中可以接“设置velocity”设置速度

发射物重力范围：发射物移动组件中，也需要设置物体的重力

**用户界面**

创建：项目文件右键->用户界面->

**控件蓝图：**

制作UI界面，在事件图表中的“创建控件”中加载

画布面板：界面框架

文本：输入文字

水平框：水平框中的物体可以整齐排列

创建控件：外部加载UI界面。选择创建好的用户界面中的控件蓝图

添加到视口：外部应用UI界面控件

**建模**

静态网格体(Static Mesh)：固定大小，所有模型会共用一个本地文件中的网格体

动态网格体(Dynamic Mesh)：大小可以变化，在游戏中临时生成，不需要本地文件

建模模式：在选项模式中选择建模。修改和创建模型后会直接覆盖原模型

**快捷键**

ALT+C：查看网格的面

**基础操作**

修改坐标轴点：XForm->编辑枢轴点(Edit Pivot)

立方体网格：创建->CubeGrid。像我的世界一样快速搭建立方体框架

网格减面：网格体->Simplify。有多种简化算法

网格加面：网格体->Remesh。增加网格面数，增加模型细节

**组编辑**

组编辑：模型->polygroup。可以将多个面划分为一组，当做整体编辑

选择过滤器：可以选择点、线、面、循环边、循环边侧边

边框忽略遮挡：开启后，在框选时会选中看不到的遮挡目标

挤出：extrude。挤出时会自动对齐鼠标指向物体的高度

**架构**

DOD(Data Oriented Design)：面向数据设计，广泛应用于游戏架构中。通过数据局部性，批量处理，简化数据结构提高缓存命中率和内存访问效率。数据局部性：把相关的数据存在连续的内存中，提高缓存命中率；批量处理：用一个函数实现大量数据处理，减少函数调用开销；简化数据结构：使用简单、扁平的数据结构，不要使用图或树等指针串联地址的复杂数据结构

ECS(Entity-Component-System)：基于DOD实现的架构，将对象拆分为实体、组件和系统，简化开发过程和提高性能。实体：一组数据的索引，包含多个组件；组件：单一数据，比如只有一个变量的结构体；系统：批量处理组件的逻辑

MassEntity：UE5中基于ECS实现的框架，ECS中的实体、组件、系统对应MassEntity中的实体(entity)、片段(fragment)、处理器(processor)。其他概念：特质(trait)为一组片段；原型(archetype)为根据片段划分实体，提高内存存储。MassEntity为大规模的单位设计

实体(entity)：FMassEntityHandle结构体。有2个成员变量： Index用于索引数据；SerialNumber用于CRC校验确保映射有效性

片段(fragment)：FMassFragment结构体。只有单个成员变量的结构体

处理器(processor)：UMassProcessor类。核心函数：ConfigureQueries删选数据；Execute操作数据

特质(trait)：UMassEntityTraitBase类。核心函数：BuildTemplate定义需要的片段

原型(archetype)：FMassArchetypeData结构体。有很多变量和函数，通常不直接操作

**性能优化**

网格体：取消生成重叠事件。碰撞范围用盒体代替原模型。碰撞检测对象尽可能选忽略，碰撞已启用使用最少功能

纹理贴图：将图片尺寸做适当压缩，比如从4096变为2048，在虚幻5中可以只修改显示和打包的贴图大小，不影响真实大小，在文件中筛选所有纹理并批量编辑：全选->右键->资产操作->编辑属性矩阵->最大纹理尺寸

tick优化：降低不重要部分的tick频率

碰撞体优化：降低碰撞体复杂程度，比如球形碰撞

动画优化：在动画选项中提供了一些优化选项，更新速率优化比较重要

绘制调用(Draw Call)：stat EHI命令查看当前绘制调用数量。每一次绘制调用需要CPU和GPU的一次通讯，UE会将相同材质、渲染状态、顶点格式的数据合并为一个绘制调用(可能来自不同模型)，因此需要尽量合并模型

异步函数：动画逻辑复杂时，可以使用异步函数并行计算

InstanceStaticMesh组件：将相同的静态网格体合并来减少绘制调用、提高性能

Significance Manager插件：根据单位距离调整tick更新频率或切换为静态网格

Animation Sharing插件：通过共享动画来减少动画实例

LOD：远距离模型使用低面数模型，但给模型制作LOD需要时间，通常使用Nanite

Nanite：基于簇的虚拟网格渲染系统。远距离模型降低模型面数

虚拟纹理：当纹理贴图很大时加载贴图会消耗性能，虚拟纹理会根据物体在屏幕中的大小改变贴图的分辨率

静态光源：光照烘焙到纹理上，GPU不需实时计算光源

固定光源：综合静态光源和动态光照效果，效果最好，性能介于两者之间

光源半径：在满足效果时要尽量缩减光源半径

Lumen：实时动态全局光照系统。可以在Lumen的设置中调整性能以适应不同场景

**cpp类**

创建cpp类：项目文件->c++类->右键创建->选择继承的父类

结构：会创建.h和.cpp文件。.h文件为声明，内容写在.cpp文件中

编译：在UE界面的右下角

类中权限：public外界可访问；protected子类可访问；private只有类中可访问

BeginPlay()：类初始化时执行一次

Tick(float DeltaTime)：项目运行后每帧执行一次

**宏**

Super::重写函数名;：写在重写函数内容的最前面，表示先执行父类的同名函数再执行子类，通常BeginPlay、Tick等需要加上