# c++课程设计报告

## 一、需求分析（游戏思路）

本次课程设计所要设计的内容是一种MUD游戏。我们经过交流后，想要进行开发一种带有地图的游戏，这样可玩性与交互性也也会更高。在我们实际设计过程中便是采用的这种思路。

然后有了地图之后，地图之间要有连通器进行地图之间的链接，这样玩家才能在不同地图间移动。不同的地图可以设计成不同的尺寸，这样在不同地图间移动时会更有真实感。地图里要有NPC，怪物，物品。NPC要有对话功能，商店功能可以设计为与NPC交谈进行时选择进行购买。而与怪物则可以进行打斗，打斗后可以有选择性地掉落东西，然后人物可以进行拾取。而物品有攻击武器，防御武器和药品，可以相应地提高人物的不同属性。然后要有一个主任务线，任务线还要有分支，这样故事才能比较有趣。

## 二、uml类图构建

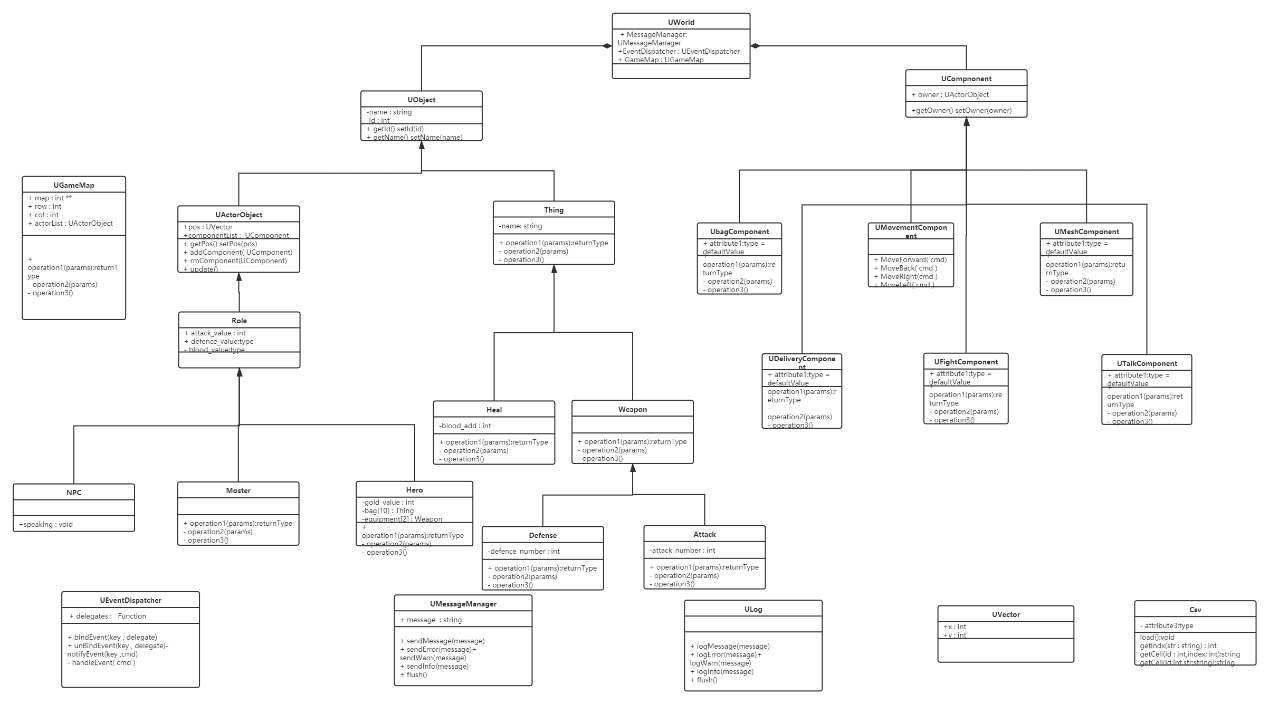
我们经过将近一周的讨论，确认了游戏的大致情节和功能之后，我们便开始了UML类图的绘制，其中人物方面，我们有玩家、NPC、怪物三种角色，所以我们将他们三个抽象出来，分别形成Hero、Monster和NPC三个类，又将这三个类抽象出来他们的基类Role玩家类，他们有姓名、id、血量、攻击值和防御值等共同属性。在游戏物品方面，我们主要有武器和药品这两种，其中武器有派生出攻击武器和防御武器两种，其中攻击武器有攻击值属性，防御武器有防御值属性，药品有增加血量属性，所以又找出了药品和武器的基类Thing类，其具有姓名、id属性，又由Role类和Thing类共同派生出基类UObject类，所以UObject是上面所有类的基类，其派生出Role类和Thing类，Role类派生出NPC、Moster和Hero类，Thing类派生出药品和武器类。

游戏中的功能主要是通过组件实现，其中组件有共同的基类UComponent类，它可以获取玩家的指针，进而获得玩家的信息，通过它派生出的UBagComponent类、UDeliveryComponent类、UMovementComponent类、UFightComponent类、UMeshComponent类、UTalkComponent类，分别实现背包功能、传送功能、移动功能、攻击功能、拾取物品功能和交流功能。

除了上的角色、物品、功能基类之外，为了实现人物和功能的连接，我们有编写了UWorld类，为了帮助UWorld类实现其连接的功能，我们还编写了UGameMap类，将人物、物品等放在地图类UGameMap中。通过对地图建模实现不同地图之间的差异性。

除此之外，我们编写了时间分配类UEventDispatcher实现事件的分配、交流管理类UTalkManager类完成任务之间的对话、读取表格类UCsvManager实现读取表格中的信息并进行处理，诸如此类还有其他类实现预想的功能，不再一一列举。

游戏的类图如下：



## 三、代码实现

### 3.1设计思路：

以“万物皆为对象”的思想进行编程。从对象基类开始，向下派生出所有物品，基类对象可以添加到地图中，这样保证所有的对象都能够添加到地图中。以地图为核心开始进行编码，地图主要完成的功能为，添加对象，删除对象，更新对象，渲染地图。这样以地图为核心，可以渲染出来一个房间。

每一个房间都可以渲染出来之后，开始构建地图，也就是渲染“世界”，“世界”主要功能为，渲染整个地图，将每一个房间渲染出来，更新“世界”当前显示房间以及房间内的对象。

当整个地图的渲染工作完成之后，就需要对各种人物进行构建。人物类和物品类都继承于Object基类，因为Object基类可以被添加到地图中，所以人物类和物品类都能够被添加到地图中。人物类继续可以派生出来NPC，怪物，英雄；物品类可以继续派生出回复血量的药品，武器等等。

角色构建完成之后，要继续添加角色具有的功能。对功能进行抽象，抽象为组件类，如果角色拥有了这个组件，改角色就有了这个组件所具有的功能。

组件由基类派生，可以派生出移动组件，背包组件，传送组件，攻击组件，交谈组件等等。

程序开始，先对主界面进行渲染，然后运行。主界面的角色可以在主界面进行自由的移动，当移动到传送组件的周围，传送组件将角色移动到该组件指向的下一个房间。当角色移动到物品周围，可以进行对物品的拾取。当移动到npc周围，可以和npc进行交谈。移动到怪物周围，可以对怪物进行攻击。

### 3.2具体实现：

#### 3.2.1类图总表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 文件 | 基类 | 类名 |
| 1 | Component | 无 | UComponent |
| 2 | Component | UComponent | UDeliveryComponent |
| 3 | Component | UComponent | UFightComponent |
| 4 | Component | UComponent | UMeshComponent |
| 5 | Component | UComponent | UMovementComponent |
| 6 | Component | UComponent | UStoryComponent |
| 7 | Component | UComponent | UTalkComponent |
| 8 | Csv | 无 | Csv |
| 9 | Csv | 无 | Quest |
| 10 | Csv | Csv | RoleCsv |
| 11 | Engine | 无 | UActorObject |
| 12 | Engine | 无 | UEventDispatcher |
| 13 | Engine | 无 | UGameMa |
| 14 | Engine | 无 | UMessage |
| 15 | Engine | 无 | UObject |
| 16 | Engine | 无 | UVector |
| 17 | Role | UActorObject | Role |
| 18 | Role | Role | Hero |
| 19 | Role | Role | NPC |
| 20 | Role | Role | Monster |
| 21 | Thing | 无 | Thing |
| 22 | Thing | Thing | Heal |
| 23 | Thing | Thing | Weapon |
| 24 | System | 无 | UCsvManager |
| 25 | System | 无 | UIManager |
| 26 | System | 无 | UMessageManager |
| 27 | System | 无 | UTalkManager |
| 28 | UI | 无 | UI |
| 29 | MudGame | 无 | UWorld |

#### 3.2.2组件类

为了实现游戏中人物的能力，我们编写了游戏组件这一大基类，其实现代码如下：

void UComponent::setOwner(UActorObject \* \_owner)

{

owner = \_owner;

}

void UComponent::init()

{

}

UActorObject \* UComponent::getOwner()

{

return owner;

}

void UComponent::update()

{

}

void UComponent::destroy()

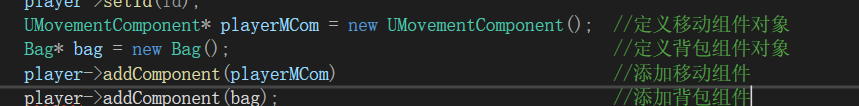
{

}

其中组件基类主要有setOwner和getOwner两大函数，其功能分别为设置组件的拥有者和获取组件的拥有者指针，通过这两个函数实现获取人物角色的信息和对人物角色信息的更改，通过组件基类派生出其他功能组件的派生类，接下来对几个功能派生类进行具体讲解。

#### 3.2.3背包类

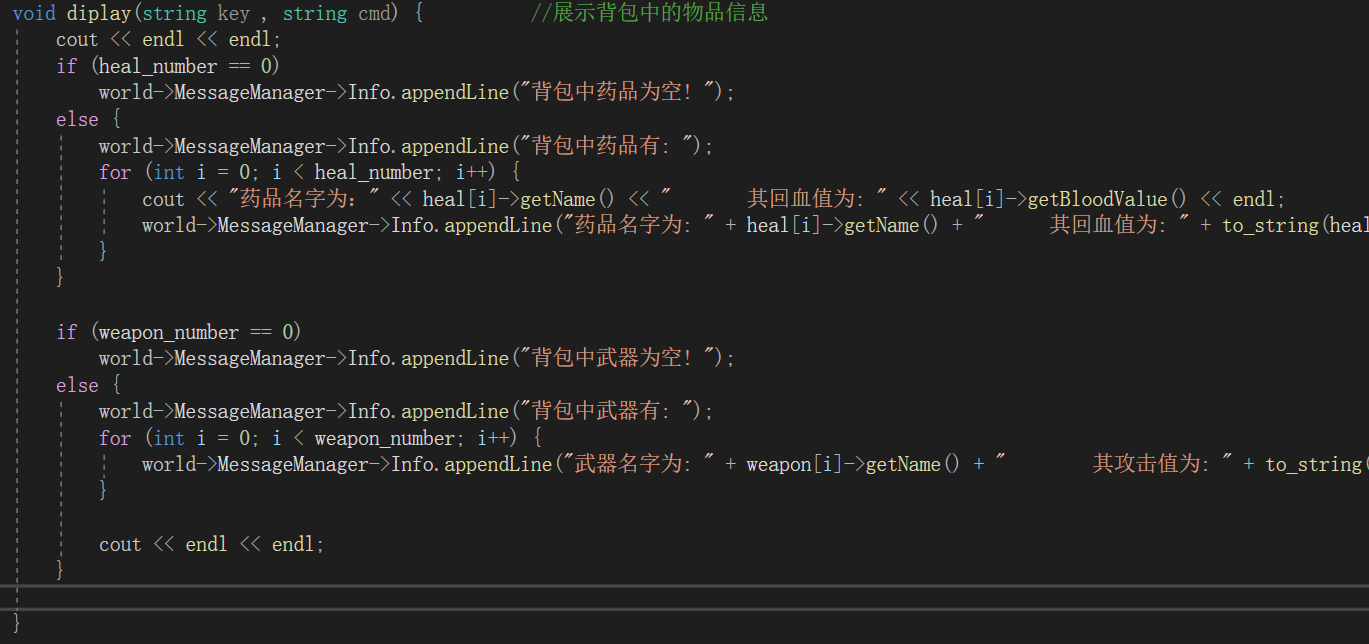
下面讲解背包组件，首先程序通过下面的代码将背包或其他组件添加到人物玩家身上：



对于背包组件的内部实现通过“world->MessageManager”代码实现获取玩家信息指针功能，通过该指针对玩家信息进行更改，下图为拾取药品、武器函数：



展示背包中物品信息：



#### 3.2.4移动组件：

class UMovementComponent :public UComponent {

public:

void init();

void MoveForward(string key, string cmd);

void MoveBack(string key, string cmd);

void MoveLeft(string key, string cmd);

void MoveRight(string key, string cmd);

void update();

void destroy();

};

void UMovementComponent::MoveForward(string key , string cmd) {

UVector tempPos = this->getOwner()->getPos();

int step = 1;

if (cmd != "")

step = safeStoi(cmd);

tempPos.x-= step;

tempPos.y;

if (world->GameMap->checkMap(tempPos.x, tempPos.y))

this->getOwner()->setPos(tempPos.x, tempPos.y);

else

world->MessageManager->Error.appendLine("超出地图范围,无法移动");

}

移动组件类的具体功能为使其拥有者具有移动的属性。

移动组件有具有四个功能，分别是前后左右四个功能。由于四个功能仅有细微的差距，这里仅解释前进功能。

首先移动组件得到他的拥有者在地图中的位置，然后修改在地图中的位置，紧接着重新设置拥有者的位置。加上容错功能之后，就成为了上面的代码。

这里有一点特别的地方。由于处理输入之后，虽然移动组件将拥有者的位置修改了，但是地图还是不知道拥有者修改之后的位置，我们也就不能看到对象在地图中移动了，所以想要看到对象实时移动的效果，我们也要在地图的更新（update（））函数中，进行修改。地图应当将其内的对象全部进行更新，然后将地图进行清空，然后将地图内的对象重新加入，这样就能做到，对象进行移动的效果。

#### 3.2.5 地图类

class UGameMap {

private:

int\*\* map;

int row;

int col;

list<UActorObject \*>\* actorList = new list<UActorObject\*>();

list<UActorObject\*>\* removeList = new list<UActorObject\*>();

public:

UGameMap(int \_row, int \_col);

virtual bool checkMap(int x, int y);

virtual void update();

virtual void addActor(UActorObject\* actor);

virtual UActorObject \* searchActor(int x, int y);

virtual void removeActor(UActorObject\* actor);

virtual bool isEmpty(int x, int y);

virtual void render();

virtual void destroy();

};

地图类可以实现不同的地图，不同的地图有不同的大小，里面有物品，NPC以及野怪等等，玩家可以在地图内移动，并在靠近NPC时可以与其对话，在靠近野怪时可以选择是否与其战斗，在靠近物品时可以选择是否拾取。并且在靠近地图之间的连接点时，可以从一个地图传送到另一个地图。玩家的一切活动都是在地图中进行的。在角色进行移动后，会调用void UGameMap::update()函数进行地图显示的更新。

#### 3.2.6世界类：

class UWorld {

public:

UMessageManager\* MessageManager = nullptr;

UIManager\* UIManager = nullptr;

UTalkManager\* TalkManager = nullptr;

UCsvManager\* CsvManager = nullptr;

UEventDispatcher\* EventDispatcher = nullptr;

UGameMap\* GameMap = nullptr;

Role\* Player = nullptr;

string nextGameMap = "";

string gameMapName = "";

string gameTitle = "";

string uiName = "";

string levels = "levels : ";

bool Running = false;

bool preEnd = false;

void render();

void update();

void run();

void init();

void iexit(string key ,string cmd);

void ijump(string key, string cmd);

void htrans(string key, string cmd);

map<string, IDelegate\*> SceneMap;

template<class T>

T\* findComponent(UActorObject\* actor)

{

auto cls = T::StaticClass();

auto componentList = actor->componentList;

for (auto it = componentList->begin(); it != componentList->end(); it++) {

auto name = (\*it)->getClass();

if (name == cls)

return dynamic\_cast<T\*>(\*it);

}

return nullptr;

}

template<class T>

void addScene(string key, T\* \_object, void (T::\* \_method)(string, string)) {

auto it = SceneMap.find(key);

if (it == SceneMap.end()) {

IDelegate\* handle = newDelegate(\_object, \_method);

SceneMap.insert(pair<string, IDelegate\*>(key, handle));

levels += key + " , ";

}

}

void addScene(string key, void (\*\_func)(string, string)) {

auto it = SceneMap.find(key);

if (it == SceneMap.end()) {

IDelegate\* handle = newDelegate(\_func);

SceneMap.insert(pair<string, IDelegate\*>(key, handle));

levels += key + " , ";

}

}

void makeScene(string key) {

auto it = SceneMap.find(key);

if (it != SceneMap.end()) {

(\*it).second->invoke("", "");

}

else if(key != "main"){

makeScene("main");

}

}

};

extern UWorld\* world;

世界类主要对地图之间的切换，每一个地图，等信息进行维护。

在对象中实例化“信息管理器”，“界面管理器”，“交谈管理器”，“时间分配器”，“当前地图”，“当前角色”，“下一个地图”，“地图名字”，“界面名字”。

地图对象具有的功能有，渲染，更新，初始化，退出游戏，跳转位置，寻找组件，添加房间，建模房间。

实例化一个世界，然后进行extern拓展到全局，这样保证全局都能够拿到world对象，并且对其进行修改。

#### 3.2.7事件分配器：

class UEventDispatcher {

private:

map<string, CMultiDelegate \*> delegateMap;

public:

template<class T>

void bindEvent(string key, T\* \_object, void (T::\* \_method)(string, string)) {

IDelegate\* handle = newDelegate(\_object, \_method);

auto it = delegateMap.find(key);

if (it == delegateMap.end()) {

CMultiDelegate \*delegate = new CMultiDelegate();

\*delegate += handle;

delegateMap.insert(pair<string, CMultiDelegate\*>(key, delegate));

}

else {

(\*(\*it).second) += handle;

}

};

template<class T>

void unBindEvent(string key, T\* \_object, void (T::\* \_method)(string, string)) {

IDelegate\* handle = newDelegate(\_object, \_method);

auto it = delegateMap.find(key);

if (it != delegateMap.end()) {

(\*(\*it).second) -= handle;

}

};

void dispatcherEvent(string key , string cmd);

void executeInput(string uiName , string cmd);

};

事件分配器，主要进行的功能为对键盘按键和对应的事件进行绑定。

用户输入按键，然后对按键进行处理，处理的时候将界面名字和输入共同作为参数，调用分配事件函数dispatcherEvent(uiName + key, cmd.substr(1, cmd.length() - 1 ));，分配事件函数从map中进行查找，如果没有查找到对应的委托，则创建一个新的委托auto it = delegateMap.find(key); if (it != delegateMap.end()) { (\*(\*it).second)(key, cmd); }，如果找到该委托，则调用该委托对应的函数。

事件管理器两外两个函数为，绑定函数void bindEvent(string key, T\* \_object, void (T::\* \_method)(string, string))，void unBindEvent(string key, T\* \_object, void (T::\* \_method)(string, string)) ，这两个函数，就是对按键进行绑定，和解绑，我们设置可以做出来一个接口，让用户去自主修改功能对应的按键，提高用户的可操作性。

#### 3.2.8 读取表格类

读取表格类用来实现读入主线剧情，读入NPC、野怪等信息。这些信息存储在csv文件中，读入NPC、野怪等信息后，便在相应地图中创建NPC及野怪，读入主线剧情后便可以进行玩家与NPC之间的对话。void Csv::load()该函数用来实现打开csv文件并进行判断是否能打开。

在主线剧情的csv文件中，第一列为语句的id，每个语句只有一个id，因而可以通过id来找到该语句。第二列text中为语句的具体内容。Choice列可以为玩家提供选择，不同的选择决定剧情的不同走向，根据选择choice中存储的id值从而可以找到下一条语句。第四列next中存储下一个语句的id，本语句输出后根据next中的id值找到下一个语句。在NPCid列中，可根据npcid确定具体是哪个npc所说的语句，在name中可以确定该npc的名字。最后的before和after列中可以存储在该npc的话语前或者后玩家要说的话。

#### 3.2.9 信息管理器

我们本着“万物皆对象”的原则，所以将主界面以及其他界面的标题、提示信息封装成UMessageManager类来实现，其主要代码如下：

此为信息管理器的构造函数，主要功能为设置字体的颜色。

此为展示信息函数，将需要展示的信息打印出来，并对输入的信息进行清空，之后展示的信息都是通过调用该函数进行实现。

通过调用该函数将界面中的标题信息、提示信息、错误信息等展示出来

#### 3.2.10主函数：

int main()

{

world->EventDispatcher = new UEventDispatcher();

world->MessageManager = new UMessageManager();

world->CsvManager = new UCsvManager();

world->TalkManager = new UTalkManager();

world->TalkManager->init();

world->gameTitle = "---------隐形守护者---------";

world->MessageManager->Tip.appendLine("欢迎进入游戏！！！");

world->addScene("main", makeMainScene);

world->addScene("level1", makeLevel1);

world->addScene("level2", makeLevel2);

world->addScene("copy", makeRandLevel);

world->addScene("story", makeStory);

world->makeScene("main");

world->init();

world->update();

world->run();

}

主函数，首先对world中的各种对象进行实例化，然后创建几个关键的地图（房间），然后开始运行world->run()。

void UWorld::run()

{

Running = true;

while (Running) {

system("cls");

if (preEnd)Running = false;

MessageManager->show();

render();

string cmd;getline(cin , cmd);

EventDispatcher->executeInput(uiName , cmd);

update();

}

}

这是world的运行函数，在运行的过程中，每次都进行清屏，信息管理器打印信息MessageManager->show();，渲染地图render()，获取用户的输入string cmd;getline(cin , cmd)，更新所有对象update();，然后再次运行，等待用户输入。在用户的输入和下一次输入的过程中产生的所有的信息，都依次打印出来。这样完成了游戏的运行。

## 四、优缺点

### 4.1优点

首先来说一说优点：

1.可扩展性强。移动、交流、物品拾取、攻击等功能全部是通过派生基类组件实现，之后若想增加其他功能直接增加该基类的派生类即可。

2.后期维护容易。例：玩家和NPC交流的信息全部存储在csv表格文件中，之后若想对游戏进行维护修改，对话方面只需要对表格信息进行修改即可，便于维护。

3.万物皆对象。编程过程中尽力避免面对过程编程思维定式，尽可能地通过创建类、对象等来实现。

### 4.2 缺点

1.功能完成不完整，如攻击功能，容错功能等。

2.程序中大量使用了指针，但是很多指针仅仅是使用，但是没有释放，造成内存空间浪费。

3.有一些变量名、函数名、类名没有按照统一的规范命名。

4.有部分字符串拼接使用了sstresm，但还是存在很多提示直接通过“+” 连接，开销大。

## 五、总结

通过本次实验课程的学习，我们学习并理解了在之前C++课堂上学习过的内容，并且在实践后加深了自己的记忆，在此之外，我们还自学了一些课程外的知识，为了更好地完成本次项目。在本次项目中，我们组内成员之间进行分工合作，每个人有自己负责的部分，并且对他人负责的部分也有着相应的理解，各自在自己进行学习实践的同时进行互相之间的交流，从而使得我们共同取得了进步。