SKILL\_MESSAGE\_H\_

#define \_SKILL\_MESSAGE\_H\_

#include <string>

using namespace std;

class SkillMessage{

public:

int id;<span style="white-space:pre"> </span>//唯一id标识

string name;<span style="white-space:pre"> </span>//技能的名称

string desc;<span style="white-space:pre"> </span>//技能描述

string icon;<span style="white-space:pre"> </span>//技能的图标

int begin\_e;<span style="white-space:pre"> </span>//技能使用时的效果

int state\_e;<span style="white-space:pre"> </span>//技能持续中的特效

int hit\_e;<span style="white-space:pre"> </span>//打中别人之后，别人身上跑的效果

int lastTime;<span style="white-space:pre"> </span>//技能状态持续时间

int cd;<span style="white-space:pre"> </span>//冷却时间

string skill\_bullet\_img;<span style="white-space:pre"> </span>//当在技能状态时，子弹的图片

};

#endif

class EffectUtil{

public:

static EffectUtil\* getInstance();

Animate\* getSkillEffectById(int id,int loop=1);

private:

static EffectUtil\* m\_instance;

};//获取动画

Animate\* EffectUtil::getSkillEffectById(int id,int loop){

SpriteFrameCache\* cache = SpriteFrameCache::getInstance();

cache->addSpriteFramesWithFile(String::createWithFormat("Skill/Effect/%d.plist",id)->getCString(),

String::createWithFormat("Skill/Effect/%d.png",id)->getCString());

SpriteFrame\* temp;

Vector<SpriteFrame\*> v;

int index = 1;

do{

CCLOG("INDEX = %d",index);

temp = cache->getSpriteFrameByName(String::createWithFormat("%d\_%d.png",id,index)->getCString());

index++;

if(temp == nullptr){

break;

}else{

v.pushBack(temp);

}

}while(true);

Animation\* animation = Animation::createWithSpriteFrames(v);

animation->setLoops(loop);

animation->setDelayPerUnit(0.1f);

Animate\* ret = Animate::create(animation);

return ret;

}

class Skill : public Node{

public:

bool init(Hero\* h);

static Skill\* create(Hero\* h);

void onTouchDown(Ref\* pSender,ui::TouchEventType type);

Hero\* getHero();

private:

ui::Button\* skillIcon;

Hero\* m\_hero;

};//添加技能框 skillIcon是技能图标

Skill\* Skill::create(Hero\* h){

Skill\* skill = new Skill();

if(skill && skill->init(h)){

skill->autorelease();

return skill;

}else{

CC\_SAFE\_DELETE(skill);

return nullptr;

}

}

bool Skill::init(Hero\* h){

m\_hero = h;

skillIcon = ui::Button::create("Skill/" + h->getSkill().icon);

skillIcon->addTouchEventListener(this,toucheventselector(Skill::onTouchDown));

this->addChild(skillIcon);

return true;

}//需要的英雄类参数

void Skill::onTouchDown(Ref\* pSender,ui::TouchEventType type){

if(type == TouchEventType::TOUCH\_EVENT\_BEGAN){

<span style="white-space:pre"> </span>m\_hero->skill();

}

}//调用英雄接口

bool FlightLayer::init(){

this->scheduleUpdate();

initListener();

m\_cur\_controlPtr = nullptr;

m\_cur\_control = nullptr;

m\_skill = nullptr;

Size visibleSize = Director::getInstance()->getVisibleSize();

Sprite\* BG = Sprite::create("flightBG.jpg");

BG->setAnchorPoint(Point(0.5f,0.5f));

BG->setPosition(visibleSize.width/2,visibleSize.height/2);

this->addChild(BG);

//m\_skill就是技能框，一开始没有选中任何英雄，所以技能框是null

m\_skill = nullptr;

return true;

}//在update函数里面时刻检测是否有选择英雄

void FlightLayer::updateSkill(){

if(m\_cur\_control==nullptr){<span style="white-space:pre"> </span>//如果没有选中的英雄

if(m\_skill){<span style="white-space:pre"> </span>//技能框不为空

m\_skill->removeFromParentAndCleanup(true);//清除技能框

m\_skill = nullptr;

}

}

if(m\_skill == nullptr){<span style="white-space:pre"> </span>//如果技能框为空

if(m\_cur\_control){<span style="white-space:pre"> </span>//但是有选中的英雄

Hero\* h = dynamic\_cast<Hero\*>(m\_cur\_control);//添加技能框

m\_skill = Skill::create(h);

m\_skill->setPosition(50,650);

addChild(m\_skill,4);

}

}else{<span style="white-space:pre"> </span>//如果技能框不为空

if(m\_cur\_control){<span style="white-space:pre"> </span>//但是有选择的英雄

if(m\_cur\_control != m\_skill->getHero()){

m\_skill->removeFromParentAndCleanup(true);//如果不同的英雄，删除技能框

m\_skill = nullptr;

}

}

}

}

void Hero::skill(){

SkillImpl impl;

impl.runSkill(m\_skill.id,this);

}//skill接口的实现

#ifndef \_SKILL\_IMPL\_

#define \_SKILL\_IMPL\_

class Hero;

class SkillImpl{

public:

void runSkill(int id,Hero\* hero);

};

#endif// SkillImp是什么

void SkillImpl::runSkill(int id,Hero\* hero){

switch(id){

case 4001://skill\_kb

{

hero->setHp(hero->getHp()/2);

hero->setSpeed(hero->getSpeed()\*2);

hero->setAtkSpeed(hero->getAtkSpeed()\*2);

hero->setAtkHateValue(hero->getAtkHateValue()\*3);

break;

}

case 4002://skill\_zl

{

Role\_Ptr initTarget = hero->getAttackTarget();

std::list<Role\_Ptr> list = hero->getLayer()->getRolesArray();

for(auto it = list.begin();it!=list.end();it++){

if((\*\*it)->getRoleType() == Role::ROLE\_TYPE\_HERO){

hero->setAttackTarget(\*it);

hero->sendBullet();

}

}

hero->setAttackTarget(initTarget);

break;

}

case 4003://skill\_ld

{

hero->setAtk(hero->getAtk()\*2);

hero->setBulletSpeed(hero->getBulletSpeed()\*2);

break;

}

default:

break;

}

}//通过switch..case就可以修改每个英雄的属性

void Role::runSkillEffect(int id,int loop){

Sprite\* sp = Sprite::create("Skill/null.png");

sp->setAnchorPoint(Point(0.5f,0));

sp->setPosition(0,0);

this->addChild(sp,10);

Animate\* animate = EffectUtil::getInstance()->getSkillEffectById(id,loop);

CallFunc\* call = CallFunc::create([=](){sp->removeFromParentAndCleanup(true);});

Sequence\* action = Sequence::create(animate,call,NULL);

sp->runAction(action);

}//技能特效 Hero类型里的函数

void Hero::skill(){

runSkillEffect(m\_skill.begin\_e);

runStateEffect(m\_skill.state\_e);

SkillImpl impl;

impl.runSkill(m\_skill.id,this);

}//skill接口的实现

void Role::injured(int effect,int damage){

runSkillEffect(effect);

}//击中特效

void Hero::update\_state(float dt){

m\_state\_lastTime += dt;

if(m\_state\_lastTime > m\_skill.lastTime){

if(skilling){

m\_state\_sprite->removeFromParentAndCleanup(true);

m\_state\_sprite = nullptr;

recover();

}

skilling = false;

m\_state\_lastTime = 0;

}

}//持续时间的实现，时间到了属性回到初始

void updateCD(float delta);

Label\* cdTime;

bool Skill::init(Hero\* h){

m\_hero = h;

skillIcon = ui::Button::create("Skill/" + h->getSkill().icon);

skillIcon->addTouchEventListener(this,toucheventselector(Skill::onTouchDown));

this->addChild(skillIcon);

int cd = m\_hero->skillCD;

cdTime = Label::create(\_\_String::createWithFormat("%d",cd)->getCString(),"Arial",30);

this->addChild(cdTime);

if(cd == 0){

cdTime->setVisible(false);

}

this->schedule(schedule\_selector(Skill::updateCD),1.0f);

return true;

}//技能剩下多长时间

void Hero::update\_skill\_cd(float dt){

if(skillCD <= 0){ canUseSkill = true; skillCD = 0;

}

else{

skillCD -= dt; }

}//Hero类里面也增加一个表示冷却时间的属性skillCD并且在Hero类的update函数时刻更新