ViewModel模块报告

付之笑 3160102285 混合班1601

## 1.分工解决方案

本模块需要在view层发生各类事件时，接收事件的指令，根据指令对相应model层数据做出更改并反馈给view层。View层的事件分为点击事件和更新事件，根据区域的不同而有所差别。为解决此问题，为每一个可能会被点击的物件，viewmodel层提供了一个相应的点击函数来执行对应的操作。而对于更新事件，由于是每一帧进行更新，viewmodel层也准备更新函数来获取当前状态，更改为下一状态。游戏总共有三个主要场景，因此这些函数分属于共三个该层的子类

## 2.设计思路

### 2.1按钮物件

对于按钮型物件，由于物件确定，其功能也确定，因此只要直接执行相应操作如退出或更换场景即可。一个例子如下：

void StartSceneActions::startOnClick() {

View::MainGameRunner\* mgr = View::MainGameRunner::getMainGameRunner();

mgr->setScene<View::GameScene>();//更换场景为游戏场景

}

### 2.2点击事件

对于飞机物件，我们将其点击事件和更新事件分开为两个函数。飞机的点击事件即为判断是否选中以及需要为飞机物件添加用于反馈选中的圆圈物件。当点击事件发生时，根据module层的选中状态和选中的指针这两个成员决定更换选中飞机以及圆圈还是直接新建。为view层添加相应的圆圈物件。代码实现如下：

void GameSceneActions::planeClick(const PlaneObject\*planeObject) {

using namespace View;

using namespace Model;

if (ChosenPlane::chosenPlane != nullptr) {//已有选中飞机

if (ChosenPlane::chosenPlane != planeObject->logicPlane)

noChoose();//与当前选择的不一样则取消选中

else {

return;//一致则无需再做操作

}

}

ChosenPlane::isFirstTime = true;//设置为第一次点击

ChosenPlane::chosenPlane =static\_cast<Plane\*>(planeObject->logicPlane); static\_cast<CircleObject\*>(ChosenPlane::chosenPlane->circle.UICircle)->radius = Plane::ordinaryRadius;//保存选中飞机的指针并设置该飞机的选中圆圈

return;

}

对于线物件，其点击事件发生在飞机以外的区域，本层提供一个非飞机区域的点击函数。调用model层数据判断情况为飞机第一条路线或是非第一条路线，以及分为左右键点击情况。更改model层和viewmodel层的和线有关的数据。代码实现如下：

void GameSceneActions::backgroundClick(const Point&point,bool isLeft) {

using Model::ChosenPlane;

using Model::Plane;

Plane\*plane = ChosenPlane::chosenPlane;

if (plane != nullptr) {//存在选中飞机

if (isLeft) {//左键点击

using namespace Model;

using namespace View;

if (ChosenPlane::isFirstTime) {//选中先清空可能的之前路线

ChosenPlane::isFirstTime = false;

for (list<Line\*>::iterator it = plane->lines.begin(); it != plane->lines.end(); ++it) {

runner->getScene()->removeObject(static\_cast<Object\*>((\*it)->UILine));

delete (\*it)->UILine;

delete \*it;

}

plane->lines.clear();

}

Line\*line = new Line();//创建新路线终点为鼠标点击点

line->to = point;

for (list<Airport>::iterator it = Airport::airports.begin(); it != Airport::airports.end(); ++it) {//判断终点是否落入某个机场

if (it->checkIn(point)) {

line->isAttachingAirport = true;

break;

}

}

if (plane->lines.empty()) {//第一条线则起点设为飞机中心 line->from = plane->position;

plane->rotation = line->to - line->from;// 方向设为线方向

}

else

line->from = plane->lines.back()->to;//否则起点为前一条线终点

LineObject\*UILine = new LineObject(

lineDepth++,

line->from,

line->to,

line->isAttachingAirport ? Line::attachingColor : Line::normalColor

);

line->UILine = UILine;

plane->lines.push\_back(line);

runner->getScene()->addObject(UILine);//线对象加入场景中

}

else

noChoose();//右键点击取消选中

}

}

### 2.3更新事件

游戏进程首先需要确定当前游戏状态为碰撞停止还是进行中，如碰撞停止则需要展示一段时间最终状态后跳转到结束界面，代码实现如下：

if (!goUpdate) {

if (clock() >= lastTime + 3000) {//展示了3000click则清空并跳转到结束

for (list\_P::iterator it = Plane::planes.begin(); it != Plane::planes.end(); ++it)

removePlane(\*it, scene);

Plane::planes.clear();

runner->setScene<EndScene>();

}

return;

}

随后进行所有飞机状态的更新，获得所有的飞机物件指针后，对每一个进行相同的更新。判断飞机目前是否走到了一段路线的终点，终点为机场则进行渐变退出，还有路线则删去前一条路线改为这一条。接着判断飞机旋转角度和路线方向是否一致，不一致则进行旋转操作。代码实现如下：

for (list\_P::iterator it = Plane::planes.begin(); it != Plane::planes.end(); ) {

if ((\*it)->overFlag) {

if ((\*it)->overFlag + Plane::delayTime <= clock()) {

list\_P::iterator next = it++;

removePlane(\*next, scene);

Plane::planes.erase(next);

}

else

++it;

continue;

}

using Tools::square;

if (!(\*it)->lines.empty()) {

using Tools::square;

Line\*line = (\*it)->lines.front();

if (square(Plane::moveVelocity) > square((\*it)->position.x - line->to.x) + square((\*it)->position.y - line->to.y)) {//几乎抵达终点

if (line->isAttachingAirport) {//终点为机场

if (ChosenPlane::chosenPlane == (\*it))

noChoose();

static\_cast<PlaneObject\*>((\*it)->UIPicture)->alpha = 0.95;

scene->setScore(ScoreManager::addPlaneReward());

(\*it)->overFlag = clock();

//计分，设置结束状态

continue;

}//终点后还有路线

scene->removeObject(static\_cast<Object\*>(line->UILine));

delete line->UILine;

delete line;//删除现在路线，更换为下一条

(\*it)->lines.pop\_front();

if ((\*it)->lines.empty())

goto out;

else

line = (\*it)->lines.front();

}

static\_cast<LineObject\*>(line->UILine)->position = line->from = (\*it)->position;

(\*it)->rotation = line->to - line->from;

}

out://飞机继续按线速度移动

double scale = Plane::moveVelocity / sqrt(square((\*it)->rotation.x) + square((\*it)->rotation.y));

double deltaX = (\*it)->rotation.x\*scale;

double deltaY = (\*it)->rotation.y\*scale;

(\*it)->position.x += deltaX;

(\*it)->position.y += deltaY; static\_cast<PlaneObject\*>((\*it)->UIPicture)->setMidPosition((\*it)->position); static\_cast<CircleObject\*>((\*it)->circle.UICircle)->setPosition((\*it)->position);

//更改角度

double arc = acos((\*it)->rotation.x / sqrt(square((\*it)->rotation.x) + square((\*it)->rotation.y)));

if ((\*it)->rotation.y > 0)

arc = 1.5\*CV\_PI - arc;

else

arc = arc - 0.5\*CV\_PI;

double r = arc \* 180 / CV\_PI - UIPlane->getRotation();

while (r > 180.0)

r -= 360.0;

while (r < -180.0)

r += 360.0;

if (r > Plane::rotateVelocity)

UIPlane->setRotation(UIPlane->getRotation() + Plane::rotateVelocity);

if (r < -Plane::rotateVelocity)

UIPlane->setRotation(UIPlane->getRotation() - Plane::rotateVelocity);

++it;

}

所有飞机更新完之后需要判断是否产生了碰撞或越界，代码实现如下：

for (list\_P::iterator it = Plane::planes.begin(); it != Plane::planes.end(); ++it) {

if ((\*it)->overFlag)

continue;

if ((\*it)->position.y<PlaneCreator::certain[0] ||

(\*it)->position.x<PlaneCreator::certain[1] ||

(\*it)->position.y>PlaneCreator::certain[2] ||

(\*it)->position.x>PlaneCreator::certain[3]) {

static\_cast<CircleObject\*>((\*it)->circle.UICircle)->radius = Plane::ordinaryRadius\*3;//越界检测

lastTime = clock();

goUpdate = false;

return;

}

for (list\_P::iterator it2 = Plane::planes.begin(); it2 != Plane::planes.end(); ++it2) {//碰撞检测

if ((\*it2)->overFlag)

continue;

if (\*it == \*it2)

continue;

using Tools::square;

if (square(Plane::ordinaryRadius\*2) >= square((\*it)->position.x - (\*it2)->position.x) + square((\*it)->position.y - (\*it2)->position.y)) {

static\_cast<CircleObject\*>((\*it)->circle.UICircle)->radius = Plane::ordinaryRadius;

static\_cast<CircleObject\*>((\*it2)->circle.UICircle)->radius = Plane::ordinaryRadius;//绘制反馈的碰撞圆圈

lastTime = clock();

goUpdate = false;

return;

}

}

}

最后根据游戏时间判断是否需要产生新飞机。

### 2.4其他函数

startGameScene函数用于初始化各个机场位置以及分数置零，记录主场景开始时间以供使用。

generatePositionAndVelocity函数用于生成飞机随机的初始位置以及方向。

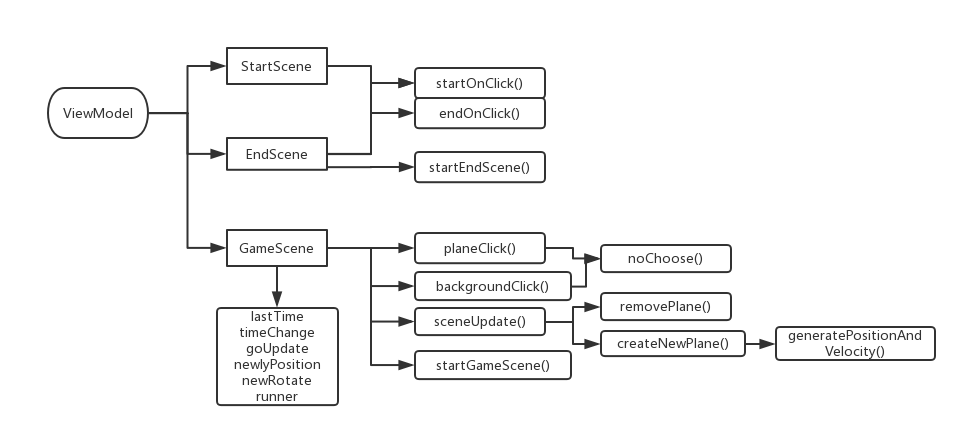
noChoose函数实现取消选中。

createNewPlane函数产生新的飞机。

removePlane函数移除特定的飞机。

这些函数的实现主要为对各层存储的各类数据的初始化或修改，因此不在此赘述。

## 3.viewmodel层数据和函数图



## 4.心得体会

本次工程的合作中，初次接触到MVVM模型，对模型的理解不够深刻，因此在划分模块时仍然有耦合的地方。由于事先商定了各个模块之间的调用关系和接口，所以仍然能同步进行工作，但实际上无法对模块功能直接进行测试，属于有瑕疵的模型，今后在工程协作中应当注意这一点，实现各模块间即使不知道对方的存在也仍然可以正常进行功能。

实现过程中，图像的旋转遇到了一些困难，由于图像的坐标中y轴向下为正，导致计算角度时需要打破以往y轴向上为正的观念。而飞机图像初始为竖直的即已经旋转九十度，因此在设置新的旋转方向时需要考虑这一点。为使飞机头最终对准线条方向，经过了较长时间的思考和尝试。此外，飞机的移动也有难点，因为像素点坐标是int类型，而实际根据角度移动时计算三角函数的结果是double类型，为实现按方向移动，最终在飞机中存储了double类型的数据，计算完成后转为int类型。

## 5.课程建议

在课程的学习过程中，对最新的mvvm框架理解一直处于模糊不清的状态，尽管有同学展示作为例子，但同学的讲解也太过迅速。希望老师在讲解框架时能放慢速度讲得更详细一些，再提供一个简单的例子来演示各个模块之间和内部如何实现，更方便理解。此外，课程教师的网络较差，并不适合小组的共同工作，希望老师往后能事先考察申请良好网络的教室。