《工程与科研中的c++》个人报告

刘秉贤 3160105106 计科1601

**1.** 分工任务的解决方案  
 我的任务是完成view层的设计。

功能上，view层进行交互。负责通过调用opencv自带的api，来实现对贴图、简单几何图形、文字的显示效果。同时，接受玩家的键盘按键输入与鼠标输入来发送消息到app层。

业务上，view层的输出只限于图片、几何元素、文字，输入中只有鼠标事件。要求支持鼠标移动导致的菜单高亮，左右键点击导致的点击事件。

因为opencv没有自带的迭代更新机制，只能手动设置对图片进行显示与修改，故首先需要实现一个完整的迭代机制以保证图片可以动态刷新。

故设计一个APP类，在main中调用其单例。该单例将进行不间断的循环迭代以更新游戏界面与实现动画效果。

再将各个场景的展示形式各用一个类实现，用共同的基类统一接口，各个图层再用共同的基类统一接口，令场景类各自存储各自图层。在每一帧，APP单例将对对应的场景进行迭代。该场景将对其存储的所有图层进行更新与迭代，由此完成view层的显示功能。

在鼠标事件发生后将其传递到view层判断该事件的归属图层，并据此传递给逻辑模块。

**2.设计思路**

2.1.MainGAmeRunner

在这里考虑用一个MainGameRunner类完成循环迭代，同时它将承担类似APP层的功能，将各层进行一定的联系与使用基础。该类负责设置循环的帧数，同时在规定的时间点调用各层的更新函数，此处只调用当前场景的迭代更新函数。

另外，该类被设计为单例，以方便使用，以下为其包含的成员的声明。

**class** **MainGameRunner** {

**private**:

**static** MainGameRunner\*runner;

**static** void mouseEvent(int event, int x, int y, int flags, void\* param);

**const** **static** int fps = 60;

Scene\*currentScene, \*nextScene;

MainGameRunner();

**public**:

**const** char\*gameName;

**static** MainGameRunner\* getMainGameRunner();

Scene\*getScene();

void run();

**template**<**typename** T>

void setScene() {

nextScene = **new** T();

}

};

其中场景的切换需要在一帧的结束时进行，故在迭代各个元素时，若有元素发出请求更新场景，则将其记录于nextScene指针，待两帧的间隙进行处理。(int类型的const static变量将被优化为常数，故可置于.h)

run()成员函数运行加载的场景，内置一个无限循环，在每一帧结束时检查是否需要切换场景。每一帧等待计算出的特定时间段即可使游戏的帧率稳定更新。

void MainGameRunner::run() {

time\_t now=clock(),target;

time\_t delta = 1000 / fps;

*//do the update in each frame.*

**for** (;;) {

**if** (nextScene != **nullptr**) {

**delete** currentScene;

currentScene = nextScene;

nextScene = **nullptr**;

currentScene->update();

cvSetMouseCallback(gameName, mouseEvent, **nullptr**);

}

**else**

currentScene->update();

target = clock();

**if**(target<=now+delta)

cv::waitKey(now + delta - target);

now = clock();

}

}

}

2.2.Scene

接下来根据前述，游戏被划分为三个界面：初始界面，游戏界面，结束界面。

设置Scene类以实现对各个界面的显示用图片的管理。Scene类负责存储各个界面的元素，在每一帧对界面内所有元素进行更新与显示，对MainGameRunner提供接口，所有场景类将由该类派生。各个显示元素由Object类派生，提供接口，最重要的是有一个显示先后顺序，也即图层的深度。Scene类将根据各类的深度决定显示的先后顺序。为方便与快捷，Scene将直接使用STL中的set，依据各元素的深度来按顺序存储Object元素。

**class** **Scene** {

**protected**:

std::set<Object\*, Object::Compare> images;

std::set<Object\*, Object::InverseCompare> c\_images;

std::set<Object\*, Object::InverseCompare> t\_images;

Mat canvasToShow;

IplImage img;

bool doUpdate;

**public**:

Point currentMousePoint;

Scene();

**virtual** ~Scene();

**virtual** void update();

void mouseEvent(int event, int x, int y, int flags, void\* param);

void addObject(Object\*);

void removeObject(Object\*);

void pause();

void goOn();

bool isPaused();

};

在该场景下的鼠标事件由mouseEvent成员函数接管。在场景中可动态插入Object与删除Object。每一帧其update函数将被调用，而update将依次调用set中依深度存储的Object，进行各个元素的更新与绘制。其中的img成员是用于为画布成员canvasToShow更新的一个变量。Object::Compare将使Object从深到浅排列，绘制时便优先绘制较深的图层，再绘制更浅的图层。而Object::InverseCompare则将其从浅到深排列，用于鼠标事件响应的优先权比较。越浅的图层越先接受鼠标的点击事件与移动事件。t\_images存储鼠标经过将闪烁的物件，c\_images将存储能够点击的物件。

关键的update函数如下：

void Scene::update() {

**if** (!doUpdate)

**return**;

**using** **namespace** cv;

cvSet(&img, Scalar(255,255,255), **nullptr**);

**for** (set\_O::iterator it = images.begin(); it != images.end(); ++it)

(\*it)->print(canvasToShow);

imshow(MainGameRunner::getMainGameRunner()->gameName, canvasToShow);

}

首先将画布设为白色，再按深度依次调用图层的画图接口，令其在白色画布上作图。再将绘制完毕的画布显示出来。

另一个关键的鼠标事件传递函数mouseEvent如下：

void Scene::mouseEvent(int event, int x, int y, int flags, void\* param) {

**if** (event == CV\_EVENT\_MOUSEMOVE) {

**for** (set\_in\_O::iterator it = t\_images.begin(); it != t\_images.end(); ++it) {

bool&flag = **dynamic\_cast**<ToggleObject\*>(\*it)->flag;

**if** ((\*it)->mouseIn(Point(x, y)))

flag = true;

**else** if (flag == true)

flag = false;

}

}

**else** **if** (event == CV\_EVENT\_LBUTTONDOWN || event == CV\_EVENT\_RBUTTONDOWN) {

currentMousePoint.x = x;

currentMousePoint.y = y;

**for** (set\_in\_O::iterator it = c\_images.begin(); it != c\_images.end(); ++it) {

**if** ((\*it)->mouseIn(Point(x, y))) {

**dynamic\_cast**<PictureObject\*>(\*it)->onClick(event == CV\_EVENT\_LBUTTONDOWN);

**return**;

}

}

}

}

在移动鼠标事件下，进入闪烁响应列表里，查找所有图层，将其显示开关依照鼠标是否在内而进行相应设置。此处原本应当编写成员函数来设置，但鉴于该属性公开后不会导致逻辑错误，且全部view层由一个人编写，故直接修改成员变量。

在点击事件时从浅至深查看所有可相应图层，将第一个包括了鼠标位置的图层的点击事件调用，传入左键或者右键点击的信息。

鉴于物件中只有图片类物件接受点击事件，在物件的类的设计时，将可点击的特性唯一地赋予了PictureObject类，而没有单独分出一个ClickableObject类。

2.3.Object

Object是显示的图层的接口类，向Scene暴露了可显示的接口，其深度属性暴露给自带的一个用于set中比较大小的类。

对于所有图层，分为几何图形类，图片类，文字类。原本可以用贴图代替简单几何图形，但据测试opencv自带的简单几何图形绘制的速度要快于贴图叠加，则简单几何图形使用opencv自带的几何绘制功能。其中文字类用于显示动态的分数，故略去其点击响应功能。而图片类用于飞机贴图，菜单，静态图片，故一部分要求能进行鼠标事件的相应。

关于Object类的设计如下：

**class** **Object** {

**protected**:

int attribute;*//click+toggle*

int depth;

Object(int depth,**const** Point&position);

**public**:

Point position;

**virtual** ~Object();

**virtual** void print(**const** cv::Mat&canvas) = 0;

*//only used by scene*

int getAttribute();

*//only used by scene*

**virtual** bool mouseIn(**const** Point&point) = 0;

**class** **Compare** {

**public**:

**typedef** Object\* Object\_ptr;

bool operator()(**const** Object\_ptr&, **const** Object\_ptr&)**const**;

};

**class** **InverseCompare** {

**public**:

**typedef** Object\* Object\_ptr;

bool operator()(**const** Object\_ptr&, **const** Object\_ptr&)**const**;

};

};

除了位置信息深度信息等必要属性外，还有一个attribute信息，其二进制表示代表了该物件是否拥有点击属性与闪烁属性。其第二位是是否拥有点击属性，第一位是是否拥有闪烁属性。

对该类进行继承以满足接口要求。其子类具体所有元素的继承解决方案过于冗长且逻辑简单，见类图框架。

其中需要特殊说明的有ToggleObject，与PlaneObject。

ToggleObject具有闪烁特性，其类内存储有两张图片，是鼠标经过与没有经过时的图片，由前述的开关控制。

PlaneObject中飞机飞行时需要旋转，存储有图片的旋转角度，原图，旋转后图片。声明如下：

**class** **PictureObject**:**public** Object {

**private**:

**protected**:

**public**:

Mat picture;

PictureObject(int depth,**const** Point&point,**const** char\*pictureName);

**virtual** void print(**const** cv::Mat&canvas);

**virtual** bool mouseIn(**const** Point&point);

**virtual** void onClick(bool isLeft) = 0;

};

**class** **PlaneObject** :**public** PictureObject {

**private**:

**static** Mat white;

double rotate;

Mat rotatedPic;

**public**:

double alpha;

void\*logicPlane;

PlaneObject(int depth, **const** Point&point);

void print(**const** Mat&canvas);

void setRotation(double rotate);

double getRotation();

void setPosition(**const** Point&point);

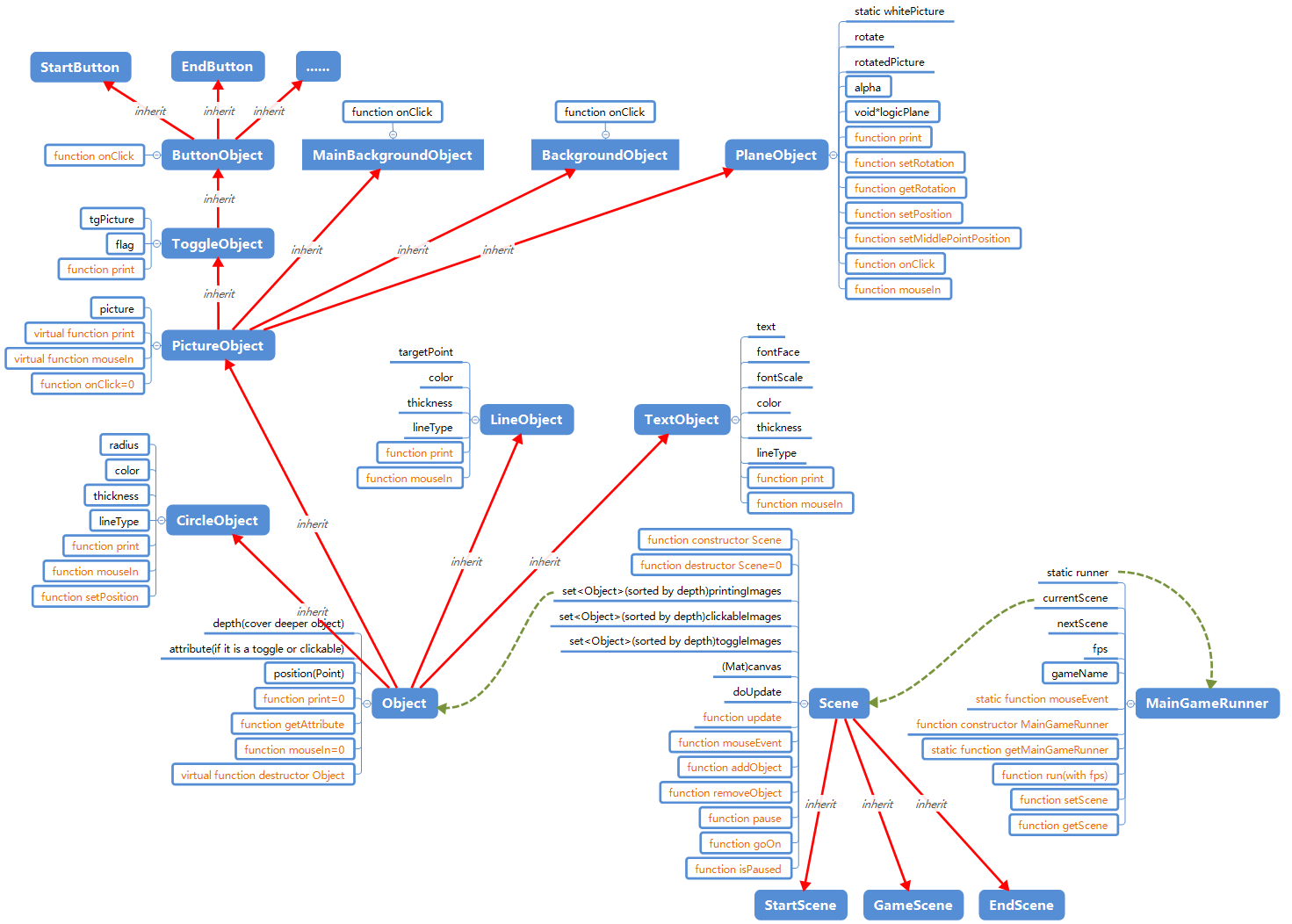
void setMidPosition(**const** Point&point);

void onClick(bool isLeft);

bool mouseIn(**const** Point&point);

};

该类继承自PictureObject类，带有原图指针picture在父类PictureObject中。若将一张图反复旋转将失真，所以该类必须一直持有一个原图的指针。同时为了避免每一帧重绘旋转后的图片，将旋转后的图片绘制在一张画布上并保留指针为rotatedPic。每次图片的旋转角度rotate改变都将重绘rotatedPic，故修改该属性必须经过成员函数setRotation，由其负责将旋转后的图片随旋转值一同更新。另外，为了逻辑上的方便，这里增加一个修改图片中点位置的接口setMidPosition，调用该函数将使飞机的中心点的坐标修改（原本的显示坐标依opencv的显示接口，将图片的左上点作为锚点）。

**3.类图框架**

该图浅显易懂，尽量列举出各主要类的成员函数与变量，详细介绍略去。

**4.运行效果图**

只用该模块无法显示整个程序的运行效果，只能通过本地简单的测试粗略检验程序。在对接以后通过全面测试查找问题。暂无运行效果。

**5.本课程心得体会**

在这门课程中，我在实践里进行了工程的需求分析与c++下的架构搭建。利用c++的面向对象特性进行类的剖分与设计，对接口和多态的运用。同时学习到一系列的工具链的使用，例如云端集成自动化测试，cmake使用，git使用与版本划分，单元测试的使用，任务划分及合作编写等等。其中也接触到工程界前沿的工程模型MVVM模型，对现行的工程界常见的架构进行学习与接触，在编写过程与同老师的交流过程中更加体会这一架构的设计思想。

**6.对课程的改进意见**

本门课程极具实用价值，是难得的实践为主的课程。唯一的意见是希望老师能在课堂上用更详细的例子来说明一下MVVM模型的具体实现，可以减少课后资料查找的时间。

**附录：View层相关源码（MainGameRunner类在内）**

MainGameRunner.h

#pragma once

#include*"Scene.h"*

**namespace** View {

**class** **MainGameRunner** {

**private**:

**static** MainGameRunner\*runner;

**static** void mouseEvent(int event, int x, int y, int flags, void\* param);

**const** **static** int fps = 60;

Scene\*currentScene, \*nextScene;

*//this class must have only one instance.*

MainGameRunner();

**public**:

**const** char\*gameName;

*//~MainGameRunner();*

*//too promise it.*

**static** MainGameRunner\* getMainGameRunner();

Scene\*getScene();

void run();

*//set new scene*

**template**<**typename** T>

void setScene() {

nextScene = **new** T();

}

};

}

MainGameRunner.cpp

#include*"MainGameRunner.h"*

#include*<ctime>*

**namespace** View {

MainGameRunner\* MainGameRunner::runner = **nullptr**;

*//initialize*

MainGameRunner::MainGameRunner() {

currentScene = **nullptr**;

nextScene = **nullptr**;

}

*//set the mouse evnet entry with the scene's mouse event*

void MainGameRunner::mouseEvent(int event, int x, int y, int flags, void\* param) {

runner->currentScene->mouseEvent(event, x, y, flags, param);

}

*//get the single instance*

MainGameRunner\* MainGameRunner::getMainGameRunner() {

**if** (runner == **nullptr**)

runner = **new** MainGameRunner();

**return** runner;

}

*//~MainGameRunner();*

void MainGameRunner::run() {

*//cvSetMouseCallback(gameName, mouseEvent, nullptr);*

time\_t now=clock(),target;

time\_t delta = 1000 / fps;

*//do the update in each frame.*

**for** (;;) {

**if** (nextScene != **nullptr**) {

**delete** currentScene;

currentScene = nextScene;

nextScene = **nullptr**;

currentScene->update();

cvSetMouseCallback(gameName, mouseEvent, **nullptr**);

}

**else**

currentScene->update();

target = clock();

#ifdef DEBUG

cv::waitKey(delta);

#else

*//while debugging, this will never do waitKay()*

**if**(target<=now+delta)

cv::waitKey(now + delta - target);

#endif *// DEBUG*

now = clock();

}

}

Scene\*MainGameRunner::getScene() {

**return** currentScene;

}

}

Scene.h

#pragma once

#include*"Object.h"*

*//#include <opencv2/core/core.hpp>*

*//#include <opencv2/highgui/highgui.hpp>*

#include*<set>*

**namespace** View {

**using** cv::Mat;

**typedef** std::set<Object\*, Object::Compare> set\_O;

**typedef** std::set<Object\*, Object::InverseCompare> set\_in\_O;

**class** **Scene** {

**protected**:

set\_O images;

*//able to be clicked*

set\_in\_O c\_images;

*//toggles*

set\_in\_O t\_images;

Mat canvasToShow;

IplImage img;

bool doUpdate;

**public**:

Point currentMousePoint;

Scene();

**virtual** ~Scene();

**virtual** void update();

void mouseEvent(int event, int x, int y, int flags, void\* param);

void addObject(Object\*);

void removeObject(Object\*);

void pause();

void goOn();

bool isPaused();

};

}

Scene.cpp

#include*"Scene.h"*

#include*"ToggleObject.h"*

#include*"Tools.h"*

#include*"MainGameRunner.h"*

#include *<opencv2/core/core.hpp>*

#include *<opencv2/highgui/highgui.hpp>*

**namespace** View{

**using** Tools::clamp;

**using** cv::Scalar;

Scene::Scene() {

doUpdate = true;

canvasToShow = Mat(640, 800, CV\_8UC3, Scalar(255,255,255));

img = IplImage(canvasToShow);

*//canvas.cols = 800;*

*//canvas.rows = 640;//could use file reading*

}

Scene::~Scene() {

*//who new,who delete. but it doesn't work here*

**for** (set\_O::iterator it = images.begin(); it != images.end(); ) {

set\_O::iterator tt = it++;

**delete** \*tt;

}

}

*//do update,then print.*

void Scene::update() {

**if** (!doUpdate)

**return**;

**using** **namespace** cv;

cvSet(&img, Scalar(255,255,255), **nullptr**);

*//(\*images.begin())->print(canvasToShow);*

**for** (set\_O::iterator it = images.begin(); it != images.end(); ++it)

(\*it)->print(canvasToShow);

imshow(MainGameRunner::getMainGameRunner()->gameName, canvasToShow);

}

void Scene::mouseEvent(int event, int x, int y, int flags, void\* param) {

**if** (event == CV\_EVENT\_MOUSEMOVE) {

**for** (set\_in\_O::iterator it = t\_images.begin(); it != t\_images.end(); ++it) {

bool&flag = **dynamic\_cast**<ToggleObject\*>(\*it)->flag;

**if** ((\*it)->mouseIn(Point(x, y)))

flag = true;

**else** if (flag == true)

flag = false;

}

}

**else** **if** (event == CV\_EVENT\_LBUTTONDOWN || event == CV\_EVENT\_RBUTTONDOWN) {

currentMousePoint.x = x;

currentMousePoint.y = y;

**for** (set\_in\_O::iterator it = c\_images.begin(); it != c\_images.end(); ++it) {

**if** ((\*it)->mouseIn(Point(x, y))) {

**dynamic\_cast**<PictureObject\*>(\*it)->onClick(event == CV\_EVENT\_LBUTTONDOWN);

**return**;

}

}

}

}

void Scene::addObject(Object\*object) {

images.insert(object);

int attr = object->getAttribute();

**if** (attr & 10)

c\_images.insert(object);

**if** (attr & 1)

t\_images.insert(object);

}

*//warning, the function wouldn't free the memory of object*

void Scene::removeObject(Object\*object) {

images.erase(object);

int attr = object->getAttribute();

**if** (attr & 10)

c\_images.erase(object);

**if** (attr & 1)

t\_images.erase(object);

}

void Scene::pause() {

doUpdate = false;

}

void Scene::goOn() {

doUpdate = true;

}

bool Scene::isPaused() {

**return** !doUpdate;

}

}

EndScene.h

#pragma once

#include*"Scene.h"*

#include*"TextObject.h"*

**namespace** View {

**class** **EndScene** :**public** Scene {

**private**:

TextObject \* text;

**protected**:

**public**:

EndScene();

void setScore(int);

};

}

EndScene.cpp

#include*"EndScene.h"*

#include*"StartButtonObject.h"*

#include*"EndButtonObject.h"*

#include*"BackgroundObject.h"*

#include*"TextObject.h"*

#include*"EndSceneActions.h"*

**namespace** View {

EndScene::EndScene() {

BackgroundObject\*bg = **new** BackgroundObject(4, Point(0, 0), "Start.png");

StartButtonObject\*restart = **new** StartButtonObject(

1,

Point(150, 100),

"restart1.png",

"restart2.png"

);

EndButtonObject\*end = **new** EndButtonObject(

2,

Point(450, 100),

"endgame1.png",

"endgame2.png"

);

TextObject\*text = **new** TextObject(-12234, Point(150, 600));

**this**->text = text;

addObject(bg);

addObject(restart);

addObject(end);

addObject(text);

ViewModel::EndSceneActions::startEndScene(**this**);

}

void EndScene::setScore(int t) {

char cache[20];

sprintf\_s(cache, "%d", t);

text->text = "Score:" + (String(cache));

}

}

GameScene.h

#pragma once

#include*"Scene.h"*

#include*"TextObject.h"*

**namespace** View {

**class** **GameScene** :**public** Scene {

**private**:

TextObject\*score;

time\_t lastTime;

**protected**:

**public**:

GameScene();

~GameScene();

void addAndPrintObject(Object\*);

void update();

void setScore(int);

};

}

GameScene.cpp

#include*"GameScene.h"*

#include*"TextObject.h"*

#include*"MainGameRunner.h"*

#include*"PlaneObject.h"*

#include*"GameSceneActions.h"*

#include*<ctime>*

#include*"MainBackgroundObject.h"*

**namespace** View {

GameScene::GameScene() {

MainBackgroundObject\*background = **new** MainBackgroundObject(0x7fffffff, Point(0, 0), "HiresScreenshot.png");

addObject(background);

TextObject\*score = **new** TextObject(

0x7ffffffe,

Point(0, 630),

CV\_FONT\_HERSHEY\_SIMPLEX,

1.0,

Scalar(255,255,255)

);

**this**->score = score;

addObject(score);

srand(time(0));

ViewModel::GameSceneActions::startGameScene(**this**);

}

GameScene::~GameScene() {

}

*//do update,then print.*

void GameScene::update() {

**using** **namespace** cv;

cvSet(&img, Scalar(255, 255, 255), **nullptr**);

*//(\*images.begin())->print(canvasToShow);*

**if** (doUpdate) {

ViewModel::GameSceneActions::sceneUpdate(**this**);

**for** (set\_O::iterator it = images.begin(); it != images.end(); ++it) {

(\*it)->print(canvasToShow);

}

imshow(MainGameRunner::getMainGameRunner()->gameName, canvasToShow);

}

}

void GameScene::addAndPrintObject(Object\*object) {

images.insert(object);

int attr = object->getAttribute();

object->print(canvasToShow);

**if** (attr & 10)

c\_images.insert(object);

**if** (attr & 1)

t\_images.insert(object);

}

void GameScene ::setScore(int t) {

char cache[20];

sprintf\_s(cache, "%d", t);

score->text = "Score:"+(String(cache));

}

}

StartScene.h

#pragma once

#include*"Scene.h"*

**namespace** View {

**class** **StartScene**:**public** Scene {

**private**:

**public**:

StartScene();

};

}

StartScene.cpp

#include*"StartScene.h"*

#include*"StartButtonObject.h"*

#include*"EndButtonObject.h"*

#include*"BackgroundObject.h"*

**namespace** View {

StartScene::StartScene() {

BackgroundObject\*bg = **new** BackgroundObject(4, Point(0, 0), "smallStart3.png");

StartButtonObject\*start = **new** StartButtonObject(

1,

Point(150, 100),

"startbutton1.png",

"startbutton2.png"

);

EndButtonObject\*end = **new** EndButtonObject(

2,

Point(450, 100),

"exitgame1.png",

"exitgame2.png"

);

addObject(bg);

addObject(start);

addObject(end);

}

}

Object.h

#pragma once

#include *<opencv2/core/core.hpp>*

#include *<opencv2/highgui/highgui.hpp>*

**namespace** View {

**using** cv::Point;

**class** **Object** {

**protected**:

int attribute;*//click+toggle*

int depth;

Object(int depth,**const** Point&position);

**public**:

Point position;

**virtual** ~Object();

**virtual** void print(**const** cv::Mat&canvas) = 0;

*//only used by scene*

int getAttribute();

*//only used by scene*

**virtual** bool mouseIn(**const** Point&point) = 0;

**class** **Compare** {

**public**:

**typedef** Object\* Object\_ptr;

bool operator()(**const** Object\_ptr&, **const** Object\_ptr&)**const**;

};

**class** **InverseCompare** {

**public**:

**typedef** Object\* Object\_ptr;

bool operator()(**const** Object\_ptr&, **const** Object\_ptr&)**const**;

};

};

}

Object.cpp

#include*"Object.h"*

**namespace** View {

Object::Object(int depth, **const** Point&position):depth(depth),position(position) {}

Object::~Object(){}

bool Object::Compare::**operator**()(**const** Object\_ptr&a, **const** Object\_ptr&b)**const** {

**return** a->depth > b->depth;

}

bool Object::InverseCompare::**operator**()(**const** Object\_ptr&a, **const** Object\_ptr&b)**const** {

**return** a->depth < b->depth;

}

int Object::getAttribute() {

**return** attribute;

}

}

CircleObject.h

#pragma once

#include*"Object.h"*

**namespace** View {

**using** cv::Mat;

**using** cv::Scalar;

**class** **CircleObject** :**public** Object {

**private**:

**protected**:

**public**:

int radius;

Scalar color;

int thickness;

int lineType;

CircleObject(

int depth,

**const** Point&position,

int radius=50,

**const** Scalar&scalar=cvScalarAll(0),

int thickness=1,

int lineType=16);

void print(**const** Mat&canvas);

bool mouseIn(**const** Point&point);

void setPosition(**const** Point&point);

};

}

CircleObject.cpp

#include*"CircleObject.h"*

#include*<opencv2\imgproc.hpp>*

#include*"Tools.h"*

**namespace** View {

CircleObject::CircleObject(

int depth,

**const** Point&position,

int radius,

**const** Scalar& color,

int thickness,

int lineType):

Object(depth,position),radius(radius),color(color),thickness(thickness),lineType(lineType) {

attribute = 0;*//0+0*

}

void CircleObject::print(**const** Mat&canvas) {

circle(canvas, position, radius, color, thickness, lineType);

}

bool CircleObject::mouseIn(**const** Point&point) {

**return** true;

*// using Tools::square;*

*// return square(radius) >= square(position.x - point.x) + square(position.y - point.y);*

}

void CircleObject::setPosition(**const** Point&point) {

position = point;

}

}

LineObject.h

#pragma once

#include*"Object.h"*

**namespace** View {

**using** cv::Scalar;

**class** **LineObject**:**public** Object {

**private**:

**protected**:

**public**:

Point targetPoint;

Scalar color;

int thickness;

int lineType;

LineObject(

int depth,

**const** Point&position,

**const** Point&target,

**const** Scalar&color=cvScalarAll(0),

int thickness=1,

int lineType=CV\_AA);

void print(**const** cv::Mat&canvas);

bool mouseIn(**const** Point&point);

};

}

LineObject.cpp

#include*"LineObject.h"*

#include*<opencv2\imgproc.hpp>*

**namespace** View {

LineObject::LineObject(

int depth,

**const** Point&position,

**const** Point&target,

**const** Scalar&color,

int thickness,

int lineType) :

Object(depth, position),

targetPoint(target),

color(color),

thickness(thickness),

lineType(lineType) {

attribute = 0;*//0+0*

}

void LineObject::print(**const** cv::Mat&canvas) {

cv::line(canvas, position, targetPoint, color, thickness, lineType);

}

bool LineObject::mouseIn(**const** Point&point) {

**return** true;

}

}

TextObject.h

#pragma once

#include*"Object.h"*

**namespace** View {

**using** cv::String;

**using** cv::Scalar;

**class** **TextObject**:**public** Object {

**private**:

**protected**:

**public**:

String text;

int fontFace;

double fontScale;

Scalar color;

int thickness;

int lineType;

TextObject(int depth,

**const** Point&position,

int fontFace=CV\_FONT\_HERSHEY\_SIMPLEX,

double fontScale=1.0,

**const** Scalar&color=Scalar(0,0,0),

int thickness=1,

int lineType=8);

void print(**const** cv::Mat&canvas);

bool mouseIn(**const** Point&point);

};

}

TextObect.cpp

#include*"TextObject.h"*

#include*<opencv2/imgproc.hpp>*

**namespace** View {

TextObject::TextObject(int depth,

**const** Point&position,

int fontFace,

double fontScale,

**const** Scalar&color,

int thickness,

int lineType):

Object(depth,position),

fontFace(fontFace),

fontScale(fontScale),

color(color),

thickness(thickness),

lineType(lineType){

attribute = 0;*//0+0*

}

void TextObject::print(**const** cv::Mat&canvas) {

cv::putText(canvas, text, position, fontFace, fontScale, color, thickness, lineType);

}

bool TextObject::mouseIn(**const** Point&point) {

**return** true;

}

}

PictureObject.h

#pragma once

#include*"Object.h"*

**namespace** View {

**using** cv::Mat;

**class** **PictureObject**:**public** Object {

**private**:

**protected**:

**public**:

Mat picture;

PictureObject(int depth,**const** Point&point,**const** char\*pictureName);

**virtual** void print(**const** cv::Mat&canvas);

*//only used by scene*

**virtual** bool mouseIn(**const** Point&point);

**virtual** void onClick(bool isLeft) = 0;*//must be defined in son class*

};

}

PictureObject.cpp

#include*"PictureObject.h"*

#include*"Tools.h"*

**namespace** View {

**using** cv::imread;

PictureObject::PictureObject(int depth, **const** Point&point,**const** char\*pictureName):

Object(depth,point),picture(imread(pictureName,-1)) {}

void PictureObject::print(**const** cv::Mat&canvas) {

SHOW\_PICTURE(picture);

}

bool PictureObject::mouseIn(**const** Point&point) {

**return** position.x <= point.x&&point.x <= position.x + picture.cols&&

position.y <= point.y&&point.y <= position.y + picture.rows;

}

*//void PictureObject::onClick() {*

*// //to be continued;*

*//}*

}

BackgroundObject.h

#pragma once

#include*"PictureObject.h"*

**namespace** View {

**class** **BackgroundObject**:**public** PictureObject {

**private**:

**protected**:

**public**:

BackgroundObject(int depth, **const** Point&point, **const** char\*name);

void onClick(bool isLeft);

};

}

BackgroundObject.cpp

#include*"BackgroundObject.h"*

**namespace** View {

BackgroundObject::BackgroundObject(int depth, **const** Point&point, **const** char\*name) :

PictureObject(depth, point, name) {

attribute = 0;*//0+0*

}

void BackgroundObject::onClick(bool isLeft) {

*//do nothing*

}

}

MainBackgroundObject.h

#pragma once

#include*"PictureObject.h"*

**namespace** View {

**class** **MainBackgroundObject** :**public** PictureObject {

**private**:

**protected**:

**public**:

MainBackgroundObject(int depth, **const** Point&point, **const** char\*name);

void onClick(bool isLeft);

};

}

MainGameBackground.cpp

#include*"MainBackgroundObject.h"*

#include*"GameSceneActions.h"*

**namespace** View {

MainBackgroundObject::MainBackgroundObject(int depth, **const** Point&point, **const** char\*name) :

PictureObject(depth, point, name) {

attribute = 2;*//1+0*

}

void MainBackgroundObject::onClick(bool isLeft) {

ViewModel::GameSceneActions::backgroundClick(

MainGameRunner::getMainGameRunner()->getScene()->currentMousePoint,

isLeft

);

}

}

PlaneObject.h

#pragma once

#include*"PictureObject.h"*

**namespace** View {

**class** **PlaneObject** :**public** PictureObject {

**private**:

**static** Mat white;

double rotate;

Mat rotatedPic;

**public**:

double alpha;

void\*logicPlane;

*//the position must be mid point position!!*

PlaneObject(int depth, **const** Point&point);

void print(**const** Mat&canvas);

void setRotation(double rotate);

double getRotation();

void setPosition(**const** Point&point);

void setMidPosition(**const** Point&point);

void onClick(bool isLeft);

bool mouseIn(**const** Point&point);

*//const Mat&getPicToShow();*

};

}

PlaneObject.cpp

#pragma once

#include*"PictureObject.h"*

**namespace** View {

**class** **PlaneObject** :**public** PictureObject {

**private**:

**static** Mat white;

double rotate;

Mat rotatedPic;

**public**:

double alpha;

void\*logicPlane;

*//the position must be mid point position!!*

PlaneObject(int depth, **const** Point&point);

void print(**const** Mat&canvas);

void setRotation(double rotate);

double getRotation();

void setPosition(**const** Point&point);

void setMidPosition(**const** Point&point);

void onClick(bool isLeft);

bool mouseIn(**const** Point&point);

*//const Mat&getPicToShow();*

};

}

ToggleObject.h

#pragma once

#include*"PictureObject.h"*

**namespace** View {

**class** **ToggleObject** :**public** PictureObject {

**private**:

**protected**:

**public**:

Mat tgPicture;

bool flag;

ToggleObject(int depth, **const** Point&point, **const** char\*picture, **const** char\*tgPicture);

void print(**const** cv::Mat&canvas);

*//virtual void onClick();//to be continued*

};

}

ToggleObject.cpp

#include*"ToggleObject.h"*

#include*"Tools.h"*

**namespace** View {

**using** cv::imread;

ToggleObject::ToggleObject(int depth, **const** Point&point, **const** char\*picture, **const** char\*tgPicture):

PictureObject(depth,point,picture),tgPicture(imread(tgPicture,-1)),flag(false){

attribute = 3;*//1+1*

}

void ToggleObject::print(**const** cv::Mat&canvas) {

**if** (flag)

SHOW\_PICTURE(tgPicture)

**else**

SHOW\_PICTURE(picture)

}

}

EndButtonObject.h

#pragma once

#include*"ToggleObject.h"*

**namespace** View {

**class** **EndButtonObject** :**public** ToggleObject {

**private**:

**protected**:

**public**:

EndButtonObject(int depth, **const** Point&point, **const** char\*P1name, **const** char\*P2name);

void onClick(bool isLeft);

};

}

EndButtonObject.cpp

#include*"EndButtonObject.h"*

#include*"StartSceneActions.h"*

**namespace** View {

EndButtonObject::EndButtonObject(int depth, **const** Point&point, **const** char\*P1name, **const** char\*P2name) :

ToggleObject(depth, point, P1name, P2name) {

attribute = 3;*//1+0*

}

void EndButtonObject::onClick(bool isLeft) {

ViewModel::StartSceneActions::endOnClick();

}

}

PauseButtonObject.h

#pragma once

#include*"ToggleObject.h"*

**namespace** View {

**class** **PauseButtonObject** :**public** ToggleObject {

**private**:

**protected**:

**public**:

PauseButtonObject(int depth, **const** Point&point, **const** char\*P1name, **const** char\*P2name);

void onClick(bool isLeft);

};

}

PauseButtonObject.cpp

#include*"PauseButtonObject.h"*

**namespace** View {

PauseButtonObject::PauseButtonObject(int depth, **const** Point&point, **const** char\*P1name, **const** char\*P2name) :

ToggleObject(depth, point, P1name, P2name) {

attribute = 3;*//1+0*

}

void PauseButtonObject::onClick(bool isLeft) {

*//do something*

}

}

StartButtonObject.h

#pragma once

#include*"ToggleObject.h"*

**namespace** View {

**class** **StartButtonObject** :**public** ToggleObject {

**private**:

**protected**:

**public**:

StartButtonObject(int depth, **const** Point&point, **const** char\*P1name,**const** char\*P2name);

void onClick(bool isLeft);

};

}

StartButtonObject.cpp

#include*"StartButtonObject.h"*

#include*"StartSceneActions.h"*

**namespace** View {

StartButtonObject::StartButtonObject(int depth, **const** Point&point, **const** char\*P1name, **const** char\*P2name) :

ToggleObject(depth, point, P1name,P2name) {

attribute = 3;*//1+0*

}

void StartButtonObject::onClick(bool isLeft) {

ViewModel::StartSceneActions::startOnClick();

}

}