课程设计3

@author 吕晟 181180083

设计整体介绍

在本次设计中，我们完成的功能有

- 使用Easy-X作为图形界面的设计工具继续完善了游戏的GUI界面设计
- 设计了两种游戏模式：标准模式和调试模式
- 标准模式中僵尸的出现是随机的
- 新增了铁桶僵尸和门框僵尸的设计
- 完善了小车的设计，使其可以移动并杀死路上的僵尸
- 完善了大蒜的设计，僵尸遇到大蒜可以改变行
- 增加了开场的动画和结束时输赢的画面
- 增加了背景音效和植物种植时的音效
- 增加了暂停功能
- 对冷却时间进行了重新设计，更符合原版游戏的设定
- 增加了首页游戏积分排行榜的制作

至此，我们已经完成的功能有

- 完成了`Object`类，`Animation`类和`Movable`类三个基类的编写
- 完成了游戏图形界面的制作，具有一定的可玩性
- 僵尸：普通僵尸、投石僵尸、铁桶僵尸、小丑僵尸、路障僵尸、读报僵尸、门框僵尸、撑杆僵尸
- 植物：向日葵、豌豆射手、双发射手、南瓜、窝瓜、大蒜、樱桃炸弹、坚果、高坚果
- 功能：Image类，Sound类，File-Operations
- 场景类：进场界面，模式选择界面，胜利界面，失败界面

具体类的介绍

1. 图形界面的绘制

   图形界面的绘制功能主要包括：背景图片，僵尸图片，僵尸动画，植物图片，植物动画，植物选择框，鼠标操作。

   - `Image`类

```
#include "Image.h"

Image::Image()
{
    pimage = new IMAGE();
```

```
}

Image::Image(const std::string& name):imageName(name){
    pimage = new IMAGE();
    load();
}

Image::Image(const char name[]):imageName(name)
{
    pimage = new IMAGE();
    load();
}

Image::Image(const Image& other)
{
    pimage = new IMAGE();
    if (this == &other || other.imageName == "\0") {
        return;
    }
    imageName = other.imageName;
    load();
}

Image::~Image()
{
    delete pimage;
}

void Image::put(int x, int y) const
{
    if (type == "jpg")
        putimage(x, y, pimage);
    if (type == "png")
        transparentimage(NULL, x, y, pimage);
}

void Image::put(int x, int y, const DWORD& t) const
{
    putimage(x, y, pimage, t);
}

void Image::setName(const std::string& name)
{
    imageName = name;
    load();
}

void Image::load()
{
    type = imageName.substr(imageName.length() - 3, 3);
    if (!loadFlag) {
        loadimage(pimage, stringToTCHAR("images\\" + imageName));
        loadFlag = 1;
    }
}

Image& Image::operator=(const Image& other)
{
    if (this == &other || other.imageName == "\0") {
        return *this;
    }
```

```
    imageName = other.imageName;
    load();
    return *this;
}

std::string Image::getType() const
{
    return type;
}

void Image::transparentimage(IMAGE* dstimg, int x, int y, IMAGE* srcimg) const
{
    // 变量初始化
    DWORD* dst = GetImageBuffer(dstimg);
    DWORD* src = GetImageBuffer(srcimg);
    int src_width = srcimg->getwidth();
    int src_height = srcimg->getheight();
    int dst_width = (dstimg == NULL ? getwidth() : dstimg->getwidth());
    int dst_height = (dstimg == NULL ? getheight() : dstimg->getheight());

    // 计算贴图的实际长宽
    int iwidth = (x + src_width > dst_width) ? dst_width - x : src_width;
// 处理超出右边界
    int iheight = (y + src_height > dst_height) ? dst_height - y : src_height;
// 处理超出下边界
    if (x < 0) { src += -x;             iwidth -= -x;   x = 0; }
// 处理超出左边界
    if (y < 0) { src += src_width * -y; iheight -= -y;  y = 0; }
// 处理超出上边界

    // 修正贴图起始位置
    dst += dst_width * y + x;

    // 实现透明贴图
    for (int iy = 0; iy < iheight; iy++)
    {
        for (int ix = 0; ix < iwidth; ix++)
        {
            int sa = ((src[ix] & 0xff000000) >> 24);
            int sr = ((src[ix] & 0xff0000) >> 16);  // 源值已经乘过了透明系数
            int sg = ((src[ix] & 0xff00) >> 8);     // 源值已经乘过了透明系数
            int sb = src[ix] & 0xff;                // 源值已经乘过了透明系数
            int dr = ((dst[ix] & 0xff0000) >> 16);
            int dg = ((dst[ix] & 0xff00) >> 8);

            int db = dst[ix] & 0xff;

            dst[ix] = ((sr + dr * (255 - sa) / 255) << 16)
                | ((sg + dg * (255 - sa) / 255) << 8)
                | (sb + db * (255 - sa) / 255);
        }
        dst += dst_width;
        src += src_width;
    }
}
```

- Animation类

```
#include "Animation.h"
#include <ctime>
```

```cpp
Animation::Animation(int s, bool png):animationStatus(s),is_png(png)
{
}

void Animation::animationUpdate()
{
    if (!is_loadItertor) {
        ImageIter = Images.begin();
        if (!is_png) BackImageIter = BackImages.begin();
        is_loadItertor = 1;
    }

    ImageIter++;
    if (!is_png) BackImageIter++;
    if (ImageIter == Images.end()) {
        ImageIter = Images.begin();
        BackImageIter = BackImages.begin();
    }
}

void Animation::play(int x, int y)//动画函数，即遍历图片
{
    if (!is_loadItertor) {
        ImageIter = Images.begin();
        if(!is_png) BackImageIter = BackImages.begin();
        is_loadItertor = 1;
    }

    if (!is_png){
        switch (animationStatus) {
        case 0:
            BackImages.begin()->put(x, y, NOTSRCERASE);
            Images.begin()->put(x, y, SRCINVERT);
            break;
        case 1:
            BackImageIter->put(x, y, NOTSRCERASE);
            ImageIter->put(x, y, SRCINVERT);
            break;
        case 2:
            break;
        }
    } else {
        switch (animationStatus) {
        case 0:
            Images.begin()->put(x, y);
            break;
        case 1:
            ImageIter->put(x, y);
            break;
        case 2:
            break;
        }
    }

}

// ! path是images文件夹下的相对路径
void Animation::loadAnimation(const std::string& path, int n, bool png)
{
    is_png = png;
```

```
    if (!is_loadAnimation) {
// 加载图片，只加载一次
        if (!is_png) {
            for (int i = 0; i < n; i++) {
                //自动填写图片名
                Images.push_back(Image(path + std::to_string(i + 1) + ".jpg"));
                BackImages.push_back(Image(path + std::to_string(i + 1) +
"_bk.jpg"));
            }
        } else {
            for (int i = 0; i < n; i++) {
                //自动填写图片名
                Images.push_back(Image(path + std::to_string(i + 1) + ".png"));
            }
        }
        is_loadAnimation = 1;
    }
}


int Animation::getAnimationStatus() const
{
    return animationStatus;
}

void Animation::setAnimationStatus(int animationStatus)
{
    this->animationStatus = animationStatus;
}

void Animation::randomAnimation()
{
    if (!is_loadItertor) {
        ImageIter = Images.begin();
        if (!is_png) BackImageIter = BackImages.begin();
        is_loadItertor = 1;
    }
    srand((unsigned)time(0));
    int r = rand() % 20;
    for (int i = 0; i < r; i++) {
        ImageIter++;
        if (!is_png) BackImageIter++;
        if (ImageIter == Images.end()) {
            ImageIter = Images.begin();
            BackImageIter = BackImages.begin();
        }
    }
}
```

- Movable类

```
#include "Movable.h"

Movable::Movable(int s, int vx, int vy):moveStatus(s), speed_x(vx),
speed_y(vy)
{
}

Movable::~Movable()
```

```
{
}

void Movable::move(int &x, int &y)
{
    switch (moveStatus) {
    case 0:
        break;
    case 1:
        x += speed_x;
        y += speed_y;
        break;
    case 2://鼠标
        x = m.x - 30;
        y = m.y - 30;
        break;
    }
}
int Movable::getmoveStatus()
{
    return moveStatus;
}
void Movable::setMoveStatus(int s)
{
    moveStatus = s;
}

void Movable::setSpeed_x(int v)
{
    speed_x = v;
}
```

- Easy-X中相关的重要接口
  - `HWND initgraph（int width, int height, int flag = NULL)`
    - 返回值：绘图窗口的句柄
    - `flag`：绘图窗口样式，`EW_SHOWCONSOLE`显示控制台窗口
  - `HWND GetHWnd()`
    - 获取绘图窗口句柄
  - `void BeginBatchDraw()`
    - 用于开始批量绘图。执行后，任何绘图操作都将暂时不输出到绘图窗口上，直到执行 `FlushBatchDraw` 或 `EndBatchDraw` 才将之前的绘图输出。
  - `void FlushBatchDraw()`
    - 用于执行未完成的绘制任务
  - `MOUSEMSG GetMouseMsg()`
    - 用于获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待

```
struct MOUSEMSG
{
    UINT uMsg;          // 当前鼠标消息
    bool mkCtrl;        // Ctrl 键是否按下
    bool mkShift;       // Shift 键是否按下
    bool mkLButton;     // 鼠标左键是否按下
    bool mkMButton;     // 鼠标中键是否按下
    bool mkRButton;     // 鼠标右键是否按下
    int x;              // 当前鼠标 x 坐标（物理坐标）
    int y;              // 当前鼠标 y 坐标（物理坐标）
    int wheel;          // 鼠标滚轮滚动值
};
```

- `void outtextxy(int x, int y, TCHAR c)`
  - 用于在指定位置输出字符串
  - c为待输出的字符，注意要进行TCHAR转换
- `void putimage(`

```
int dstX,               // 绘制位置的 x 坐标
  int dstY,             // 绘制位置的 y 坐标
  IMAGE *pSrcImg,           // 要绘制的 IMAGE 对象指针
  DWORD dwRop = SRCCOPY // 三元光栅操作码
);
```

2. 声音类的制作

声音类的制作主要调用链Windows系统API：`mciSendString`

```
MCIERROR mciSendString(
          LPCTSTR lpszCommand,      //MCI命令字符串
          LPTSTR  lpszReturnString, //存放反馈信息的缓冲区
          UINT    cchReturn,        //缓冲区的长度
          HANDLE  hwndCallback      //回调窗口的句柄，一般为NULL
       );
```

具体类定义与实现如下：

```
#pragma once
#include<conio.h>
#include <graphics.h>
#include <string>
#pragma comment(lib,"Winmm.lib")

// 声音类，用声音文件名初始化，封装了声音的打开、播放、循环播放、关闭功能

extern const TCHAR* stringToTCHAR(const std::string& str);

class Sound
{
public:
    Sound(const std::string &name);
    void play() const;
    void playRepeatedly() const;
    void close() const;
private:
    std::string soundName;
};
```

```
#include "Sound.h"
```

```
Sound::Sound(const std::string& name):soundName(name)
{

}

void Sound::play() const
{
    mciSendString(stringToTCHAR(("open musics/" + soundName)), NULL, 0, 0); //注意声音文件放在musics目录下
    mciSendString(stringToTCHAR("play musics/" + soundName), NULL, 0, 0);
}

void Sound::playRepeatedly() const
{
    mciSendString(stringToTCHAR("open musics/" + soundName), NULL, 0, 0); //注意声音文件放在musics目录下
    mciSendString(stringToTCHAR("play musics/" + soundName + " repeat"), NULL, 0, 0);
}

void Sound::close() const
{
    mciSendString(stringToTCHAR("stop musics/" + soundName), NULL, 0, 0);
    mciSendString(stringToTCHAR("close musics/" + soundName), NULL, 0, 0);
}
```

3. 界面的制作

基类Scene

```
class Scene
{
    public:
        Scene(const std::string& bk, const std::string& mc);
        virtual ~Scene()=0;
        virtual void RunScene() const = 0;
    protected:
        Image background;                       //背景图片
        const Sound BGM;                              //BGM
};
```

初始界面

```
#include "StartScene.h"

const StartScene* StartScene::getStartScene() {
    static const StartScene instance;
    return &instance;
}

StartScene::StartScene():Scene("startScene.jpg", "StartMusic.mp3"){
}

void StartScene::RunScene() const {
    // 显示欢迎界面图片
    background.put(0, 0);
    outtextxy(30, 65, _T("排行榜"));
    for (int i = 0; i < 10; i++) {
```

```
        outtextxy(30, 65+(i+1)*20, stringToTCHAR(std::to_string(rankingList[i])));
    }
    FlushBatchDraw();

    // 播放背景音乐
    BGM.playRepeatedly();

    do {
        m = GetMouseMsg();
    } while (m.uMsg != WM_LBUTTONDOWN);      //检测鼠标左键点击

    BGM.close();                             //关闭BGM
}

StartScene::~StartScene()
{
}
```

模式选择界面

```
#include "ModeScene.h"
#include"globalvariables.h"

int mode;
const ModeScene* ModeScene::getModeScene() {
    static const ModeScene instance;
    return &instance;
}

ModeScene::ModeScene() :Scene("ModeScene.jpg", "StartMusic.mp3") {
}

void ModeScene::RunScene() const {
    // 显示欢迎界面图片
    background.put(0, 0);
    FlushBatchDraw();

    // 播放背景音乐
    BGM.playRepeatedly();

    do
    {
        m = GetMouseMsg();//捕捉鼠标信息
        if (m.x >=500 && m.x<=780 && m.y>=100 && m.y <= 200)
        {
            mode = 1;
        }
        if (m.x >= 490 && m.x <= 770 && m.y >= 250 && m.y <= 350)
        {
            mode = 2;
        }
    } while (m.uMsg != WM_LBUTTONDOWN);

    BGM.close();                             //关闭BGM
}

ModeScene::~ModeScene()
{
}
```

成功界面

```
#include "WinScene.h"

const WinScene* WinScene::getWinScene() {
    static const WinScene instance;
    return &instance;
}

WinScene::WinScene() :Scene("win.jpg", "win.mp3") {
}

void WinScene::RunScene() const {
    background.put(0, 0);
    outtextxy(300, 300, _T("得分上榜，恭喜！"));
    BGM.play();
    FlushBatchDraw();
    system("pause");
}

WinScene::~WinScene()
{
}
```

失败界面

```
#include "LoseScene.h"

const LoseScene* LoseScene::getLoseScene() {
    static const LoseScene instance;
    return &instance;
}

LoseScene::LoseScene() :Scene("lose.jpg", "lose.mp3") {
}

void LoseScene::RunScene() const {
    background.put(0, 0);
    outtextxy(300, 300, _T("僵尸吃掉了你的大脑！"));
    BGM.play();
    FlushBatchDraw();
    system("pause");
}

LoseScene::~LoseScene()
{
}
```

4. 游戏结束标志

```
// 判定游戏失败
for (std::vector<Zombie*>::iterator iter = zombie.begin(); iter != zombie.end(); )
{
    if ((*iter)->getPozition_x() < 10) {        // 遍历僵尸，若有僵尸越过防线，僵尸消失，基地血量减一
        std::vector<Car*>::iterator iterC;
        for (iterC = car.begin(); iterC != car.end(); ++iterC)
        {
            if (is_same_row((*iter)->getPozition_y(),(*iterC)->getPozition_y()))
```

```
            {
                (*iterC)->setAnimationStatus(1);
                (*iterC)->setMoveStatus(1);
                (*iterC)->setSpeed_x(10);
                break;
            }
        }
        if (iterC == car.end())
        {
            gameFlag = -1;
            return;
        }
        Zombie::killNum++;
        iter = zombie.erase(iter);
        Car::BaseHp--;
    } else {
        ++iter;
    }
```

5. 游戏数据写入

```
#include <fstream>
#include <iostream>
#include <algorithm>
#include "globalvariables.h"

int rankingList[11];

void cleanRankingList() {
    using namespace std;
    ofstream out;
    out.open("rankingList.txt", ios::out | ios::binary);
    int x = 0;
    for (int i = 0; i < 10; i++)
        out.write((const char*)(&x), sizeof(int));
    out.close();
}

void loadRankingList() {
    using namespace std;
    ifstream in;
    ofstream out;
    in.open("rankingList.txt", ios::in | ios::binary);
    if (!in.is_open()) {
        cerr << "存档打开失败";
        exit(0);
    }
    for (int i = 0; i < 10; i++) {
        in.read((char*)(rankingList + i), sizeof(int));
        if (in.eof())
        {
            in.close();
            cleanRankingList();
            in.open("rankingList.txt", ios::in | ios::binary);
            break;
        }
    }
    in.close();
}
```

```
void writeRankingList(int score) {
    using namespace std;
    ofstream out;
    out.open("rankingList.txt", ios::out | ios::binary);
    rankingList[10] = score;
    sort(rankingList, rankingList + 11, greater<int>());
    for (int i = 0; i < 10; i++) {
        out.write((const char*)(rankingList + i), sizeof(int));
    }
    out.close();
}
```

6. 暂停功能设计

```
if ((m.x >= 700) && (m.x <= 800) && (m.y >= 15) && (m.y <= 65))
{
    has_stopped = !has_stopped;

    if (has_stopped)
    {
        has_stopped = true;
        stop();
        continue;
    }
    else
    {
        start();
    }
}

void stop()
{
    for (auto iter = zombie.begin(); iter != zombie.end(); ++iter)
    {
        (*iter)->setMoveStatus(0);
        (*iter)->setAnimationStatus(0);
    }
    for (auto iter = ball.begin(); iter != ball.end(); ++iter)
    {
        (*iter)->setMoveStatus(0);
    }
    for (auto iter = bullet.begin(); iter != bullet.end(); ++iter)
    {
        (*iter)->setMoveStatus(0);
    }
    for (auto iter = plant.begin(); iter != plant.end(); ++iter)
    {
        (*iter)->setAnimationStatus(0);
    }
}

void start()
{
    for (auto iter = zombie.begin(); iter != zombie.end(); ++iter)
    {
        (*iter)->setMoveStatus(1);
        (*iter)->setAnimationStatus(1);
    }
    for (auto iter = bullet.begin(); iter != bullet.end(); ++iter)
```

```
    {
        (*iter)->setMoveStatus(1);
    }
    for (auto iter = plant.begin(); iter != plant.end(); ++iter)
    {
        (*iter)->setAnimationStatus(1);
    }
}
```

程序亮点及运行方法

项目利用率面对对象的思想，利用继承多态等方法提高了程序的鲁棒性。在层次结构方面，由顶向下逐层细化，在逻辑上符合游戏的设计理念。我们在网络上寻找了大量素材并自己通过Photoshop制作了相关的僵尸和植物图片，使游戏的界面美观。我们从第一次设计开始就加入了鼠标操作，并一直沿用至今。

我们仿照原版游戏进行了相关功能的设计，设计完整并具有一定的可玩性。我们在本次设计中加入了初始界面和版本选择界面，与原版一致，并在最后加入了游戏成功和失败的判定，实现了积分榜在首页的显示。我们加入了游戏音乐，更符合游戏的场景。我们也加入了暂停功能。

总之，设计充分利用了C++的语言特性，并充分理解了OOP的设计理念，是一次较为成功的课程设计。

程序的不足

当然，项目还有较大的改进空间。原版游戏中还有金币装备、在游戏开始前选择植物，以及娱乐模式等，可以在今后的学习中不断改进。我们使用了容易上手的Easy-X作为图形界面的开发工具，简单实用，但功能单一。在今后的改进中，可以将其移植到以Qt为图形界面的程序中，这样可以使得图形界面更加美观，也可以加入按键操作。

当然，程序还有一些未被发现的功能错误，可以在不断调试中加以改进。