

# 第二次课程设计——坦克大战

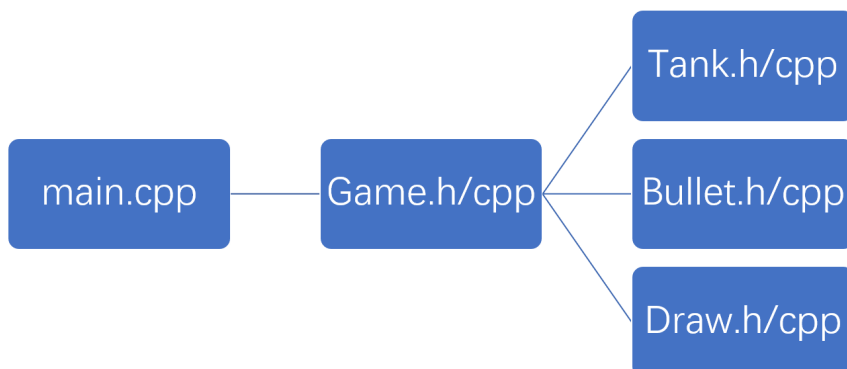
学号：191220088 姓名：秦嘉余

## 1.课程内容

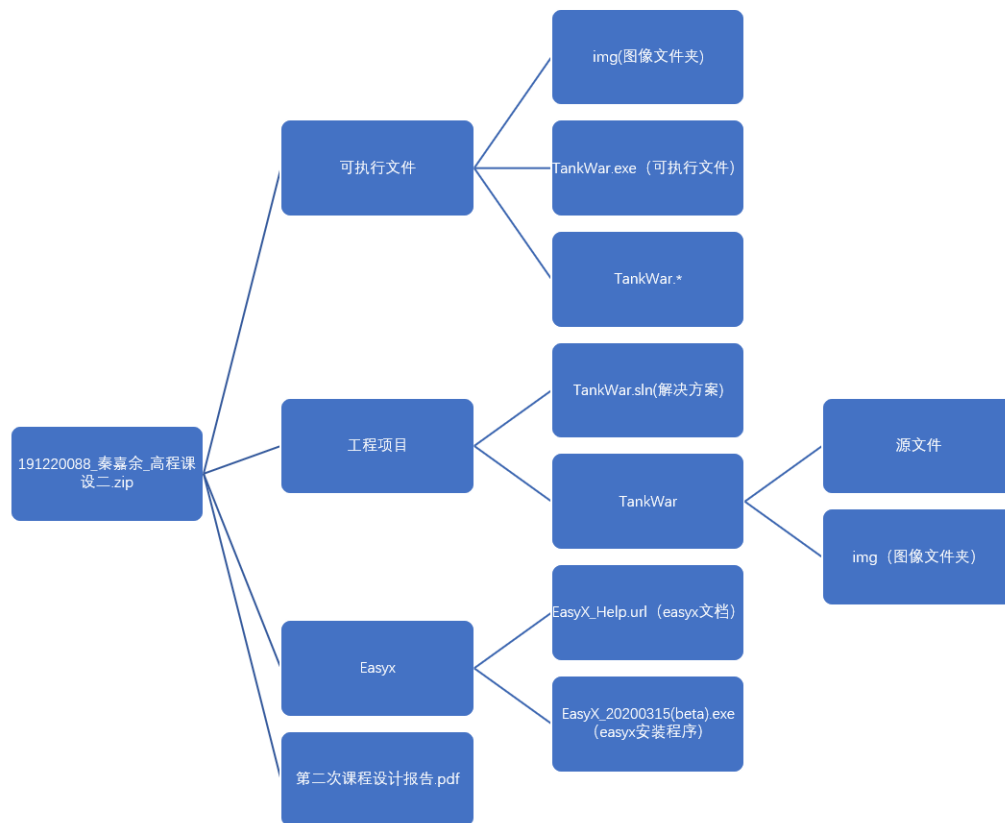
本次课程设计实验，利用面向对象的方式，完成了经典坦克大战的命令行版本。玩家控制自己的坦克'w', 'd', 'a', 's'控制自己坦克的移动，利用 'j' 发射子弹来击毁敌方坦克，地图中会出现各种地形，同时也有一定几率刷出道具，玩家利用这些来保卫自己的基地。当击毁的坦克数量到达一定数量时游戏胜利，当自己的基地被摧毁或者自己的坦克被击毁时游戏失败。

## 2.注意事项

- 源文件结构：



- 提交文件结构：






- 本次实验采用easyx库完成图形的绘制，即使得画面形象也保证了命令程序的要求，打开源文件需要安装easyx库，easyx库安装包附在压缩文件中，打开解决方案运行需要安装easyx库

### 3.设计方案

#### • 地图：

- 命令行的地图区域被分为26\*26的方格，每个方格的像素大小为30\*30
- 每个地形占用一个方格，每个坦克占用四个方格，每个子弹占用两个方格，每个道具占用两个方格
- 有地形的地方坦克不能移动
- 实现三种地形（不包含空地）：

- 木墙 
- 铁墙 
- 海洋 

- 木墙时最脆弱的墙，可以被任意坦克的子弹摧毁
- 铁墙只能被第三类坦克（装甲坦克摧）摧毁
- 海洋为禁区，不能移动至此，也不会消失

#### • 坦克：

- 每个坦克占用四个方格
- 坦克不能移动到有地形的地方，也不能移动到有基地和其他坦克的地方

- 最多存在三个敌方坦克

## ○ 己方坦克：



- 己方坦克为黄色，有四种方向的形态
- 可以发射子弹：●
- 'w'控制坦克向上移动，'s'控制坦克向下移动，'a'控制坦克向左移动，'d'控制坦克向右移动，'j'控制坦克发射子弹
- 上一颗子弹存在时不能发射下一颗子弹
- 己方坦克的默认血量为：10，所有坦克移动速度默认为：1，己方坦克发射的子弹默认造成伤害：1，子弹的默认速度速度：2（1为坦克的移动速度，子弹速度应该大于坦克速度）
- 己方的坦克子弹攻击力为1，默认情况下只能击毁木墙

## ○ 敌方坦克：

- 敌方坦克1（普通坦克）：



- 敌方坦克1（普通坦克）为白色，有四种方向的形态
- 可以发射子弹：●
- 坦克自动移动，当碰撞墙壁或者移动了一定的步数后转化方向，移动的同时一直发射子弹
- 上一颗子弹存在时不能发射下一颗子弹
- 敌方坦克1（普通坦克）的默认血量为：1，所有坦克移动速度默认为：1，敌方坦克1（普通坦克）发射的子弹默认造成伤害：1，子弹的默认速度速度：2（1为坦克的移动速度，子弹速度应该大于坦克速度）
- 敌方坦克1（普通坦克）子弹攻击力为1，默认情况下只能击毁木墙
- 敌方坦克2（速击坦克）：





- 敌方坦克2（速击坦克）为绿色，有四种方向的形态
- 可以发射子弹●
- 坦克自动移动，当碰撞墙壁或者移动了一定的步数后转化方向，移动的同时一直发射子弹
- 上一颗子弹存在时不能发射下一颗子弹
- 敌方坦克2（速击坦克）的默认血量为：2，所有坦克移动速度默认为：1，敌方坦克1（普通坦克）发射的子弹默认造成伤害：1，子弹的默认速度速度：5（1为坦克的移动速度，子弹速度应该大于坦克速度）
- 敌方坦克2（速击坦克）子弹攻击力为1，默认情况下只能击毁木墙
- 敌方坦克3（装甲坦克）：




- 敌方坦克3（装甲坦克）为粉色，有四种方向的形态
- 可以发射子弹●

- 坦克自动移动，当碰撞墙壁或者移动了一定的步数后转化方向，移动的同时一直发射子弹
- 上一颗子弹存在时不能发射下一颗子弹
- 敌方坦克3（装甲坦克）的默认血量为：3，所有坦克移动速度默认为：1，敌方坦克3（装甲坦克）发射的子弹默认造成伤害：1，子弹的默认速度为：2（1为坦克的移动速度，子弹速度应该大于坦克速度）
- 敌方坦克3（装甲坦克）子弹攻击力为1，但可以摧毁铁墙和木墙





## • 子弹

- 子弹分为两种：
  - 己方坦克发射的子弹：
  - 敌方坦克发射的子弹：
- 两种子弹在本质上无差别
- 任何两个子弹在碰撞时可以相互抵消
- 子弹击中坦克，坦克的生命值减少
  - 一般子弹减少1点生命
  - 在己方坦克吃掉星星道具后，坦克的攻击力变为2，可以对敌方坦克造成2点伤害
- 子弹击中基地，基地的生命减少1
- 子弹也有速度之分，默认情况下：
  - 敌方坦克2（速击坦克）的子弹速度为2
  - 其余坦克的速度为1

## • 基地

- 基地的始终位于地图的横向正中央，纵向最下方，有一圈木墙保护
- 基地的形态：
- 基地的大小为4个像素块
- 基地默认生命值为：5
- 基地的生命为0时，游戏失败

## • 道具

- 道具的大小为1个像素块
- 道具在到达一定分数时随机出现在随机位置
- 炸弹道具：
  - 己方坦克吃掉后清空全屏坦克
- 时停道具：
  - 己方坦克吃掉后使得敌方坦克停止移动一定时间
- 迷你坦克道具：
  - 己方坦克吃掉后增加己方坦克3点生命
- 星星道具：
  - 己方坦克吃掉后增加己方坦克生命力为2，且为永久

## • 状态栏

- 状态栏显示：
  - 击败的每种坦克的数量
  - 距离游戏胜利还需要的坦克数量
  - 己方坦克的生命值
  - 基地的生命值
  - 己方坦克的攻击力

## • 游戏过程

- 进入游戏后先进入欢迎界面：



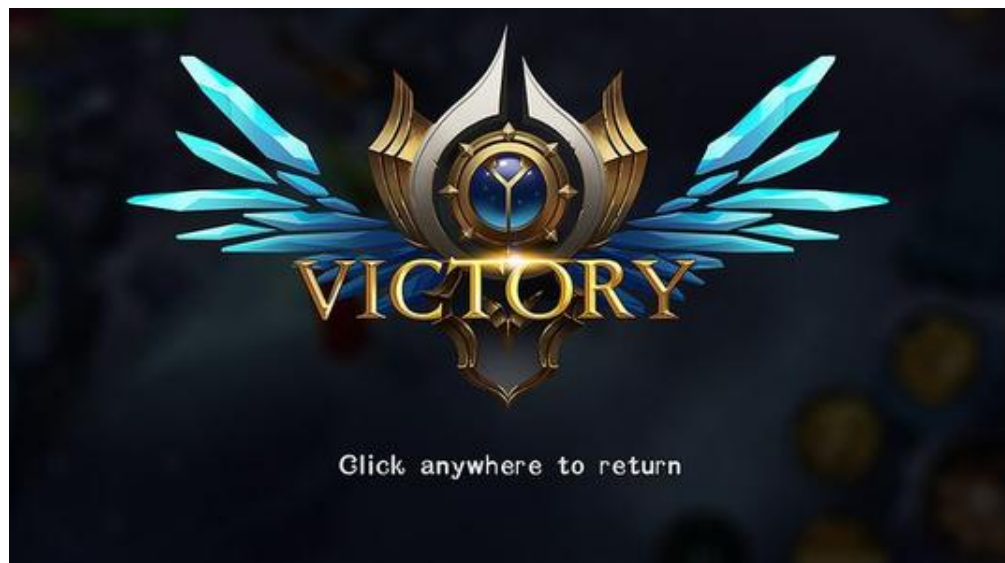
- 点击PLAYER进入游戏
  - 点击上一关，下一关选择关卡
  - 右方有控制提示
- 进入游戏后进入主界面：



- 游戏失败进入失败提示:

■ **GAME  
OVER**

- 游戏胜利进入胜利提示:



## 4.类的设计

### 坦克类

坦克类的共同抽象基类 (Tank) :

```
#define S 30          //有效单元格缩放比例
#define W 26          //宽
```

```

#define H 26          //高

enum DIR
{
    UP=0, DOWN, LEFT, RIGHT
};

class Tank
{
    //坐标均为坦克左上角的左边
protected:
    int lx;           //存储移动前一步的横坐标
    int ly;           //存储移动前一步的纵坐标
    int x;            //横坐标
    int y;            //纵坐标
    DIR d;            //方向
    Bullet* e;        //子弹的指针
    int speed;         //速度，默认为30，移动一个单元格
    int bulletspeed;   //子弹的速度，默认为2
    int attack;        //攻击力，默认为1
    int l;            //标识符，0为自己，1为敌方
    int HP;            //生命值
    bool live;        //是否存活
public:
    Tank(int _x, int _y); //构造函数
    void mkpos()           //使得坦克所在单元格的数据标记为-1
    void rmpos()           //使得坦克所在单元格的数据标记为0
    virtual void Draw() = 0; //纯虚函数，画出坦克
    virtual void move() = 0; //纯虚函数，移动坦克（更改横纵坐标）
    virtual void shoot() = 0; //纯虚函数，发射子弹
    Bullet* getbullet();    //子弹get方法
    void setbullet(Bullet* b); //子弹的set方法
    int getHp();            //HP的get方法
    void reduceHP();        //减少1点生命
    void addHP(int x);      //增加x点生命
    bool is_live();         //判断是否存活，同时更新live
    int getl();             //标识符的get方法
    int getx();             //x的get方法
    int gety();             //y的get方法
    bool is_be_hit(int _x, int _y); //判断是否被（_x, _y）坐标的物体碰撞
    int getbulletspeed();   //子弹速度的get方法
    int getattack();        //攻击力的attack方法
    void addattack();       //攻击力为1时增加一点攻击力
};

```

- 每个成员变量及函数的意义已在注释中，不再赘述

## 己方坦克类（OwnTank）：

```

class OwnTank :public Tank
{
private:
    vector<IMAGE> image;           //存储己方坦克的图像
public:
    OwnTank(int _x, int _y);       //构造函数
    void Draw();                   //具体实现了Draw函数
    void move();                   //具体实现了move函数
    void shoot();                  //具体实现了shoot函数
};

```

- 己方坦克类 (OwnTank) 为Tank基类的子类
- move函数:

```

void OwnTank::move()
{
    lx = x;
    ly = y;
    char ch;
    if (_kbhit())
    {
        ch = _getch();
        switch (ch)
        {
            case 'w':
                if (d == UP) { if (y > 0 && access_move_y(x, y - 1)) { rmpos();
y--; mkpos(); } }
                else { d = UP; }
                break;
            case 's':
                if (d == DOWN) { if (y < H - 2 && access_move_y(x, y + 2)) {
rmpos(); y++; mkpos(); } }
                else { d = DOWN; }
                break;
            case 'a':
                if (d == LEFT) { if (x > 0 && access_move_x(x - 1, y)) {
rmpos(); x--; mkpos(); } }
                else { d = LEFT; }
                break;
            case 'd':
                if (d == RIGHT) { if (x < W - 2 && access_move_x(x + 2, y)) {
rmpos(); x++; mkpos(); } }
                else { d = RIGHT; }
                break;
            case 'j':
                shoot();
                break;
            default:
                break;
        }
    }
}

```



- \_kbhit函数非阻塞输入：
  - w, s, a, d控制上下左右，利用两个函数access\_move判断，如果可以移动就移动坐标，同时更新地图中的标识
  - j发射子弹，调用shoot函数
- shoot函数：

```
void OwnTank::shoot()
{
    //已有子弹
    if (e != NULL)
        return;
    //所在位置不能发子弹
    if (d == UP && y == 0) { return; }
    if (d == DOWN && y == H - 2) { return; }
    if (d == LEFT && x == 0) { return; }
    if (d == RIGHT && x == W - 2) { return; }
    switch (d)
    {
    case 0:
        e = new Bullet(x, y-1, UP, 0);
        break;
    case 1:
        e = new Bullet(x, y+1, DOWN, 0);
        break;
    case 2:
        e = new Bullet(x-1, y, LEFT, 0);
        break;
    case 3:
        e = new Bullet(x+1, y, RIGHT, 0);
        break;
    }
}
```

- 无子弹存在时，new创建新的子弹对象

## 敌方坦克类 (EnemyTank)：

```
class EnemyTank :public Tank
{
protected:
    vector<IMAGE> image;
    int num;          //移动步数记录
public:
    EnemyTank(int _x, int _y);
    void move();
    void shoot();
    void Draw();
};
```

- image载入12个图像（4个方向\*3种）
- num记录移动步数
- 成员函数功能与OwnTank类似，仅说明不同点
  - move函数自动移动，根据num到达一定步数时改变移动方向（随机改变），不可移动时改变移动方向

- Draw函数根据生命值画出不同图像（速击坦克HP=1时形态变为普通坦克，装甲坦克类似）

## 敌方坦克类1 (Enemy1) :

```
class Enemy1 :public EnemyTank
{
public:
    Enemy1(int _x, int _y, DIR _d); //构造函数
};
```

- 构造函数:

```
Enemy1::Enemy1(int _x, int _y, DIR _d):EnemyTank(_x,_y)
{
    l = 1;
    d = _d;
}
```

- 构造函数包含\_d参数，随机生成方向
- 构造函数改变了一些成员变量的值为敌方坦克1所需要的属性

## 敌方坦克类2 (Enemy2) :

```
class Enemy2 :public EnemyTank
{
public:
    Enemy2(int _x, int _y, DIR _d);
};
```

- 构造函数:

```
Enemy2::Enemy2(int _x, int _y, DIR _d):EnemyTank(_x,_y)
{
    l = 1;
    HP = 2;
    d = _d;
    bulletspeed = 5;
}
```

- 构造函数包含\_d参数，随机生成方向
- 构造函数改变了一些成员变量的值为敌方坦克2所需要的属性

## 敌方坦克类3 (Enemy3) :

```
class Enemy3 :public EnemyTank
{
public:
    Enemy3(int _x, int _y, DIR _d);
};
```

- 构造函数:

```

Enemy3::Enemy3(int _x, int _y, DIR _d) :EnemyTank(_x,_y)
{
    l = 1;
    HP = 3;
    d = _d;
}

```

- 构造函数包含\_d参数，随机生成方向
- 构造函数改变了一些成员变量的值为敌方坦克3所需要的属性

## 子弹类

### 子弹类 (Bullet) :

```

class Bullet
{
private:
    int lx;        //存储移动后以前的横坐标
    int ly;        //存储移动后以前的纵坐标
    int x;         //横坐标
    int y;         //纵坐标
    DIR d;         //若炮弹为横向，占据60*30，为纵向，占据30*60
    int l;         //标识符，0为自己的炮弹，1为敌方的炮弹
    bool live;     //是否存活
    vector<IMAGE> image; //子弹的图像

public:
    Bullet(int _x,int _y,DIR d,int _l); //构造函数
    void Draw(); //画出子弹
    void move(); //移动子弹
    DIR getdir(); //dir的get方法
    bool is_live(); //判断是否存活，同时改变live
    int getl(); //l的get方法
    int getx(); //x的get方法
    int gety(); //y的get方法
};

```

- 每个成员变量及函数的意义已在注释中，大部分不再赘述
- Draw函数根据l标识符画出对应的坦克
- 构造函数经过坦克类的shoot函数调用
  - 根据坦克生成x, y坐标
  - 根据坦克方向生成d (子弹方向)
  - l标记子弹的类型
- move函数改变坐标：

```

void Bullet::move()
{
    lx = x;
    ly = y;
    //判断子弹是否撞墙
    if (d == 0 && y == 0) { live = false; return; }
    if (d == 1 && y == H - 1) { live = false; return; }
    if (d == 2 && x == 0) { live = false; return; }
}

```

```

    if (d == 3 && x == w - 1) { live = false; return; }

    if (!live) { return; }

    switch (d)
    {
    case 0: if(y>0)y--; break;
    case 1: if(y<H-1)y++; break;
    case 2: if(x>0)x--; break;
    case 3: if(x<W-1)x++; break;
    }

    //判断子弹是否击中建筑
    if (d == 0 || d == 1)
    {
        if (Draw::map[x][y] == 1) { Draw::map[x][y] = 0; live = false; }
        if (Draw::map[x + 1][y] == 1) { Draw::map[x + 1][y] = 0; live =
false; }
        if (Draw::map[x][y] == 4) { live = false; }
        if (Draw::map[x + 1][y] == 4) { live = false; }
        //if (Draw::map[x][y] == 2) { live = false; }
        //if (Draw::map[x + 1][y] == 2) { live = false; }
    }
    else
    {
        if (Draw::map[x][y] == 1) { Draw::map[x][y] = 0; live = false; }
        if (Draw::map[x][y + 1] == 1) { Draw::map[x][y + 1] = 0; live =
false; }
        if (Draw::map[x][y] == 4) { live = false; }
        if (Draw::map[x][y + 1] == 4) { live = false; }
        //if (Draw::map[x][y] == 2) { live = false; }
        //if (Draw::map[x][y + 1] == 2) { live = false; }
    }
    if (!live) { return; }
}

```

- 同时判断子弹是否击中墙壁或者建筑

## 画图类

### 画图类 (Draw) :

```

class Draw
{
public:
    //0为无建筑, 1为木墙, 2为铁墙, 3为基地, 4为海洋
    static int map[W][H];
    //1为炸弹道具, 2为定时道具, 3为小坦克 (增加生命), 4为星星 (永久增加攻击力1点)
    static int pro[W][H];
private:
    IMAGE wall;
    IMAGE steel;
    IMAGE base;
    IMAGE bomb;
    IMAGE timer;
    IMAGE water;
    IMAGE mintank;
}

```

```

    IMAGE star;
    int checkpoint;           //关卡
public:
    Draw(int x);              //构造函数
    void init_image();         //载入图像
    void draw();               //画图函数
};

```

- map和pro数组记录地图信息和道具信息，设计为全局变便于其他类使用
  - map数组中：0为无建筑，1为木墙，2为铁墙，3为基地，4为海洋
  - pro数组中：1为炸弹道具,2为时停道具,3为小坦克（增加生命）,4为星星（永久增加攻击力1点）
- checkpoint为关卡信息
- 构造函数的参数赋值为checkpoint，为关卡信息，同时为map和pro初始化
- init\_image载入IMAGE类型成员变量的图像
- draw函数根据map和pro画出图像
- 再Draw.cpp中，定义了三个全局变量：

```

int data1[W][H];
int data2[W][H];
int data3[W][H];

```

记录三个内置的地图信息

## 游戏类

### 游戏类（Game）：

```

class Game
{
private:
    vector<Tank*> t;           //坦克动态数组，t[0]为己方坦克
    vector<IMAGE> image;      //载入必要的图像
    int HP;                   //己方坦克生命，同步于t[0]
    int record[3];            //分别为击败的各种坦克数量
    bool suspended;           //为时停道具计时用
    int suspendednum;         //为时停道具计时用
    int baseHP;               //基地生命
    bool p[3];                //避免道具重复生成使用
    int checkpoint;           //关卡信息
public:
    Game();                   //构造函数
    void exec();              //游戏主执行函数
    void movetank();           //移动所有坦克
    void movebullet(int i);    //移动所有子弹
    void is_bullet_tank();      //判断子弹是否击中坦克
    void killbullet();         //将生命结束的子弹置为NULL
    void killtank();           //去除t中生命结束的坦克
    void drawtank();           //画出所有坦克
    void drawbullet();         //画出所有坦克
    void is_bullet_bullet();    //判断子弹是否互相碰撞
    void is_bullet_steel();     //判断子弹是否击中铁墙
    void drawbar();            //画出状态栏

```

```

void welcome();           //欢迎界面
bool is_tank_pro();       //判断坦克是否吃到道具
bool have_pro();          //判断地图中是否有道具
void moveowntank();       //只移动自己的坦克，在时停期间使用
void gameover();          //游戏结束判断
void isdead();            //判断是否己方坦克死亡或者基地生命为0
void vectory();           //判断是否胜利
};

```

- 构造函数:

```

Game::Game() :suspended(false), suspendednum(0),baseHP(5),checkpoint(1)
{
    p[0] = false;
    p[1] = false;
    p[2] = false;
    initgraph(780+180, 780);
    record[0] = 0;
    record[1] = 0;
    record[2] = 0;
    IMAGE i;
    loadimage(&i, _T("img//enemy1U.gif"));
    image.push_back(i);
    loadimage(&i, _T("img//enemy2U.gif"));
    image.push_back(i);
    loadimage(&i, _T("img//enemy3U.gif"));
    image.push_back(i);
}

```

- 初始化一系列成员变量
  - 初始化easyx 的画布
  - IMAGE载入，为画状态栏使用
- main函数中只调用welcome函数作为开始的接口:

```

#include"Game.h"

int main()
{
    Game g;
    g.welcome();
    return 0;
}

```

- welcome函数载入初始界面:

```

oid Game::welcome()
{
    cleardevice();
    IMAGE welcome,tip1;
    loadimage(&welcome, _T("img//welcome.png"));
    putimage(90, 200, &welcome);
    loadimage(&tip1, _T("img//tip1.png"));
    putimage(780 + 25, 200, &tip1);
}

```

```

char s[20];
sprintf(s, "关卡 %d", checkpoint);
outtextxy(370, 500, s);
outtextxy(320, 530, "上一关");
outtextxy(400, 530, "下一关");
while (true)
{
    MOUSEMSG m;
    if (MouseHit())
    {
        m = GetMouseMsg();
        if (m.mkLButton)
        {
            if (m.x >= (90 + 198) && m.x <= (90 + 198 + 191) && m.y >=
(200 + 261) && m.y <= (200 + 261 + 25))
                exec();
            else if (m.x >= (320) && m.x <= (320 + 70) && m.y >= (530) && m.y <=
(530 + 30))
            {
                if (checkpoint > 1)
                    checkpoint--;
                sprintf(s, "关卡 %d", checkpoint);
                outtextxy(370, 500, s);
            }
            else if (m.x >= (400) && m.x <= (400 + 70) && m.y >= (530)
&& m.y <= (530 + 30))
            {
                if (checkpoint < 3)
                    checkpoint++;
                sprintf(s, "关卡 %d", checkpoint);
                outtextxy(370, 500, s);
            }
        }
    }
}
}

```

- 画出必要的元素：
  - 欢迎图片
  - 进入按钮
  - 上一关下一关按钮
  - 关卡信息
  - 操作提示
- 检测鼠标操作：
  - 鼠标单击下一关，关卡改变为下一关
  - 鼠标单击上一关，关卡改变为上一关
  - 鼠标单击开始，进入exec函数开始游戏
- exec函数执行游戏操作：

```

void Game::exec()
{
    struct _timeb timebuffer; // 定义一个结构
    _ftime(&timebuffer); // 初始化timebuffer
    srand(timebuffer.millitm); // timebuffer.millitm获得毫秒
}

```

```

Draw d(checkpoint);
OwnTank o(8, 24);
t.push_back(&o);
EnemyTank* e=NULL;
bool temp = true;
while (true)
{
    vectory();
    while (t.size() < 4)
    {

        int r = rand() % 10;
        int d = rand() % 3;
        DIR D;
        if (d == 0)D = UP;
        if (d == 1)D = DOWN;
        if (d == 2)D = LEFT;
        if (d == 3)D = RIGHT;
        int x = rand() % (W - 2);
        int y = rand() % (7);

        while ((Draw::map[x][y] != 0 || Draw::map[x + 1][y] != 0 ||
Draw::map[x][y + 1] != 0 || Draw::map[x + 1][y + 1] != 0))
        {
            x = rand() % (W - 2);
            y = rand() % (7);
        }
        if (r>=0&&r<=5)e = new Enemy1(x, y, D);
        if (r>=6&&r<=8)e = new Enemy2(x, y, D);
        if (r == 9)e = new Enemy3(x, y, D);
        t.push_back(e);
    }
    HP = t[0]->getHp();
    if ((record[0] + record[1] + record[2]) % 10 == 0&& (record[0] +
record[1] + record[2])!=0&&temp&& p[(record[0] + record[1] + record[2]) / 10
- 1] == false)
    {
        int x = rand() % 26;
        int y = rand() % 26;
        while (Draw::map[x][y] != 0)
        {
            x = rand() % 26;
            y = rand() % 26;
        }
        int z = rand() % 4;
        if(z==0)Draw::pro[x][y] = 1;
        if (z == 1)Draw::pro[x][y] = 2;
        if (z == 2)Draw::pro[x][y] = 3;
        if (z == 3)Draw::pro[x][y] = 4;
        temp = false;
        p[(record[0] + record[1] + record[2]) / 10 - 1] = true;
    }
    if (have_pro())temp = is_tank_pro();
    cleardevice();
    d.draw();
    drawbar();
}

```



```
if (suspendednum == 30)suspended = false;
//移动坦克
if (!suspended)
{
    movetank();
    suspendednum = 0;
}
else
{
    moveowntank();
    suspendednum++;
}
```

```
for(int i=0;i<5;i++)
{
    //移动子弹
    movebullet(i);
```

```
    //生命结束的子弹设为NULL
    killbullet();
```

```
    //子弹是否击中坦克
    is_bullet_tank();
```

```
    //生命结束的子弹设为NULL
    killbullet();
```

```
    //子弹是否击中子弹
    is_bullet_bullet();
```

```
    //生命结束的子弹设为NULL
    killbullet();
```

```
    //子弹是否击中铁墙
    is_bullet_steel();
```

```
    //生命结束的子弹设为NULL
    killbullet();
```

```
    //去除死亡的坦克
    killtank();
```

```
    //画出子弹
    drawbullet();
```

```
    //自己是否死亡
    isdead();
}
```

```
    //画出坦克
    drawtank();
```

```
    sleep(160);
```

```
}
```

```
}
```

- 准备工作

- 为随机数播种
- 新建一个Draw对象，参数为关卡信息
- 新建一个己方坦克对象，放入t中

- 进入游戏的操作循环：

- vectory函数判断是否游戏成功

```
while (t.size() < 4)
{
    int r = rand() % 10;
    int d = rand() % 3;
    DIR D;
    if (d == 0)D = UP;
    if (d == 1)D = DOWN;
    if (d == 2)D = LEFT;
    if (d == 3)D = RIGHT;
    int x = rand() % (W - 2);
    int y = rand() % (7);

    while ((Draw::map[x][y] != 0 || Draw::map[x + 1][y] != 0 || Draw::map[x][y + 1] != 0 || Draw::map[x + 1][y + 1] != 0))
    {
        x = rand() % (W - 2);
        y = rand() % (7);
    }
    if (r>=0&&r<=5)e = new Enemy1(x, y, D);
    if (r>=6&&r<=8)e = new Enemy2(x, y, D);
    if (r == 9)e = new Enemy3(x, y, D);
    t.push_back(e);
}
```

- 敌方坦克不足时，补足敌方坦克
- 坦克坐标在一定范围内随机生成（地图上方，且为空地）
- 坦克方向随机生成

```
if ((record[0] + record[1] + record[2]) % 10 == 0&& (record[0] + record[1] + record[2])!=0&&temp&& p[(record[0] + record[1] + record[2]) / 10 - 1] == false)
{
    int x = rand() % 26;
    int y = rand() % 26;
    while (Draw::map[x][y] != 0)
    {
        x = rand() % 26;
        y = rand() % 26;
    }
    int z = rand() % 4;
    if(z==0)Draw::pro[x][y] = 1;
    if (z == 1)Draw::pro[x][y] = 2;
    if (z == 2)Draw::pro[x][y] = 3;
    if (z == 3)Draw::pro[x][y] = 4;
    temp = false;
    p[(record[0] + record[1] + record[2]) / 10 - 1] = true;
}
```

- 到达一定分数时生成道具
- 道具坐标随机生成（全图，且为空白处）
- 道具类型随机生成

```

■ if (suspendednum == 30)suspended = false;
    //移动坦克
    if (!suspended)
    {
        movetank();
        suspendednum = 0;
    }
    else
    {
        moveowntank();
        suspendednum++;
    }

```

- 移动坦克
- 在时停道具激活时只移动自身坦克
- 未激活时移动全部坦克
- 到底一定的时停循环次数，结束时停道具

```

■ for(int i=0;i<5;i++)
    {
        //移动子弹
        movebullet(i);

        //生命结束的子弹设为NULL
        killbullet();

        //子弹是否击中坦克
        is_bullet_tank();

        //生命结束的子弹设为NULL
        killbullet();

        //子弹是否击中子弹
        is_bullet_bullet();

        //生命结束的子弹设为NULL
        killbullet();

        //子弹是否击中铁墙
        is_bullet_steel();

        //生命结束的子弹设为NULL
        killbullet();

        //去除死亡的坦克
        killtank();

        //画出子弹
        drawbullet();

        //自己是否死亡
        isdead();
    }

```

- 子弹的操作，注释已说明，不再赘述
- ```
//画出坦克  
drawtank();  
  
sleep(160);
```
- 画出坦克
- 停止一段时间，保证图像可观测

## 5.遇到的问题

---

- 本次实验中，遇到的最大的问题就是各种数据之间信息互通的问题，例如在坦克类与子弹的交互中，光在坦克类中无法获得子弹的信息，为了解决这个问题，将一些有关的判断放在了Game类中，作为桥接坦克类与子弹类的桥梁
- 游戏分为26\*26块作为有效单元，在绘图时只能在这些单元中移动，不能有中间过渡，所以产生了帧率过低的感觉，为了解决这个问题，尝试在绘图时，补绘中间过渡图像，但是为了每个帧可观察必须利用Sleep是得其停止一段时间，这样更加增大了一个循环的时间，卡顿感更加明显，甚至无法正常游戏，随即只能放弃了这种方法
- 同时这个项目中仍有许多方面需要改进
  - 类的初始架构实现的不是很好，使得过多的功能放在的Game类中，有些功能从本质上应该归于坦克类
  - Game的主循环过于庞大，使得游戏运行时有很大的延迟感，游戏体验效果不是很好
  - 敌方坦克的三种派生类实际上并不需要，没有增加新功能，只改变了成员变量的值，故有些冗余