坦克大战

一、游戏介绍

相信大部分同学都玩过或看过“坦克大战”这款经典游戏。现在，就由我们自己动手来开发它。只要大家具备了 C++语言和面向对象的基础知识，然后按照实验指南的指导一步一步进行下去，相信我们每个同学都能把这款经典游戏做出来。

二、实验目标

综合运用 C++及其面向对象的知识开发一款小游戏。

改为用 C 语言和 EasyX 等图形库等图形用户界面（GUI）开发一款小游戏。

三、实验内容

在一个战场上，玩家控制坦克，消灭敌方坦克，并防止敌方坦克摧毁我方基地。游戏的具体要求如下：

1、游戏有一个初始页面，如下图。屏幕上最内部的黑色区域为玩家坦克的活动区域，左上角坐标为(-26,-22)，右下角坐标为(26,22)。当坦克运动到该区域边界时，坦克不能继续前进。

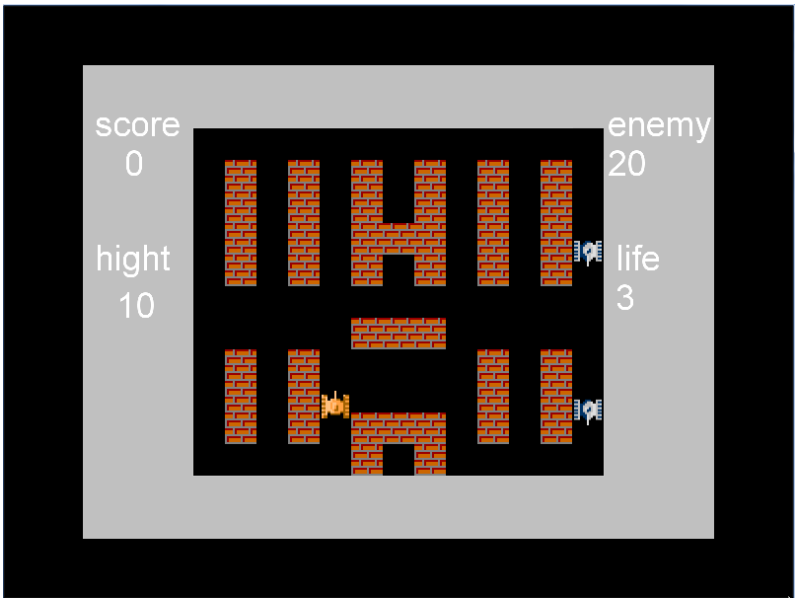
2、按下任意键开始游戏，玩家控制坦克在战场上穿梭，碰到墙时，不能通过。

3、敌方坦克自由移动，每隔 2 秒改变一个方向，每隔 3 秒发射一发子弹。

4、敌方坦克每隔 5 秒出现一辆，从屏幕上方的左、中、右三个位置依次出现。

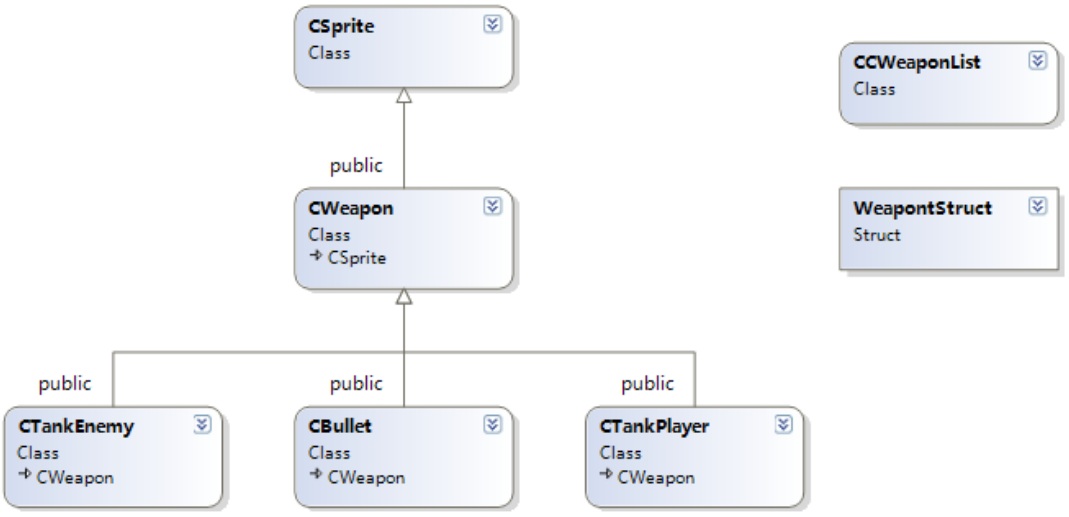
5、当玩家被消灭或者我方基地被摧毁或者游戏时间大于 30 秒的时候，游戏结束。

游戏开始前



进入游戏

四、游戏的整体框架



五、实验指南

实验准备

打开 FunCode，创建一个新的 C++项目。注意：项目名称必须为英文和数字，且不能有空格。

点击“项目”→“导入地图模板”，从对话框中选取名称为 TankWar 的模板导入。导入成功后，界面如下：

实验一 游戏开始

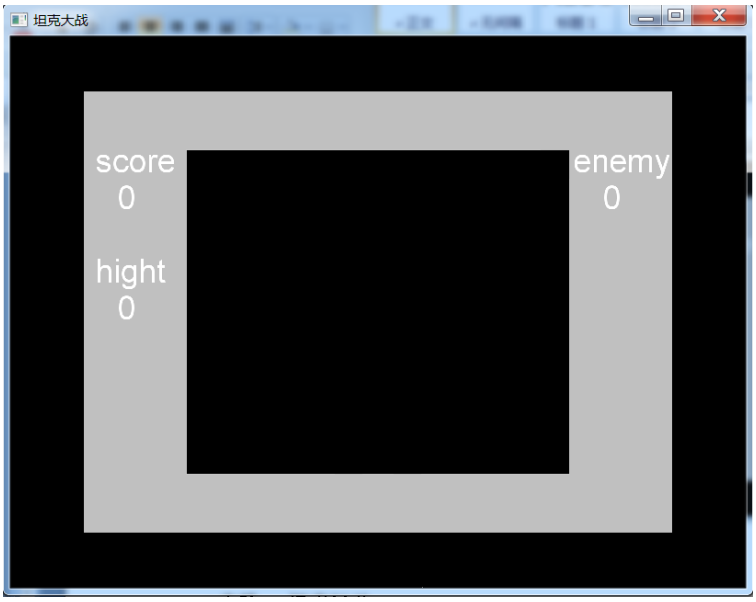
【实验内容】

1、设置游戏标题

2、按空格键,提示图片消失,游戏进入开始状态.

【实验运行结果】

游戏开始前



按下空格键后

【实验思路】

要处理 FunCode 中的图片，我们需要声明 CSprite 类型对象来指向相应图片，然后调用精灵类的相应函数进行处理。

按下空格键是键盘按下事件，系统调用 CSystem::OnKeyDown 函数进行响应。该函数中调用 CGameMain::OnKeyDown 函数。因此我们可以在 CGameMain 类的 OnKeyDown 函数完成相应代码。

按下键盘后，需要改变游戏的状态，游戏从未开始进入开始状态。成员变量 m_iGameState 用来表示游戏状态。

【实验指导】

1、C++程序的执行入口是主函数。FunCode 的主函数名称叫 WinMain，写在 Main.cpp 文件中。CSystem::SetWindowTitle 是设置程序运行窗口标题的函数，修改如下：

```
CSystem::SetWindowTitle("坦克大战");
```

2、FunCode 程序运行时，当发生键盘按下事件，程序首先调用并执行 CSystem::OnKeyDown 函数，然后由 CSystem::OnKeyDown 函数调用并执行 CGameMain::OnKeyDown 函数。因此，键盘按下事件的响应代码我们在 CGameMain::OnKeyDown 函数中编写即可。

3、我们要处理的两个精灵如下图，它们的名称分别是 splash 和 start。我们需要创建两个 CSprite 类对象与这两个精灵绑定。

4、在 CGameMain 类中声明两个 CSprite*类型，根据面向对象的封装性原理，成员变量的访问权限应该是 private。代码应该写 LessonX.h 文件中。

```
CSprite*  m_pSplash;
CSprite*  m_pStart;
```

在 CGameMain 类的构造函数中，对上面两个指针变量进行初始化。通过调用 CSprite 的构造函数，将精灵和精力对象绑定在一起。

```
m_pSplash= new CSprite("splash");
m_pStart= new CSprite("start");
```

5、最后，我们在 CGameMain::OnKeyDown 函数中处理空格键按下事件。

```
if( 0 ==GetGameState() )
{
        if(iKey ==KEY_SPACE)
        {
        m_iGameState = 1;
        }
}
```

KEY_SPACE 是枚举变量 KeyCodes 的成员，表示空格键。KeyCodes 定义在 CommonClass.h 文件中。该枚举变量定义了全部的键盘值。注意：**必须将 m_iGameState 的值改为 1。**当 GameMainLoop 函数再次执行时，根据 m_iGameState 的值调用并执行 GameInit 函数，游戏进入开始状态。

进入游戏开始状态时，调用 CSprite 类的 SetSpriteVisible 函数将精灵图片设置为不可见。

在 GameInit 函数中添加代码：

```
        m_pSplash->SetSpriteVisible(false);
        m_pStart->SetSpriteVisible(false);
```

6、程序执行完 CGameMain::GameInit 函数后，执行 CGameMain:: SetGameState(2)，将 m_iGameState 的值改为 2，游戏进入运行状态。

7、在 CGameMain 类的析构函数中 delete 本实验创建的指针对象，以释放分配的内存。

```
CGameMain::~CGameMain()
```

```
{
        delete m_pSplash;
        delete m_pStart;
}
```

8、编译并运行程序，然后按下空格键，看看运行效果。

实验二 坦克运动

【实验内容】

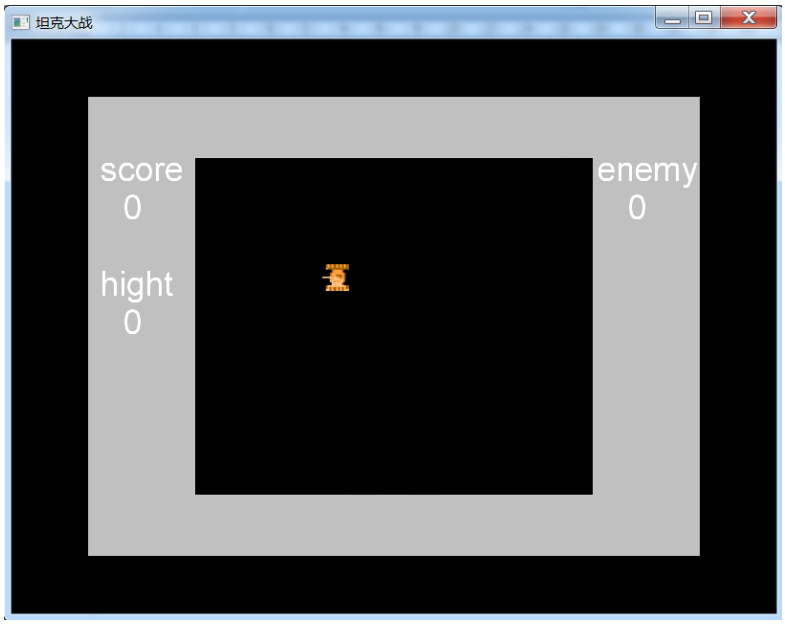
1、创建坦克类 CTankPlayer；

2、游戏开始时，将坦克放置于(0,0)的坐标上；

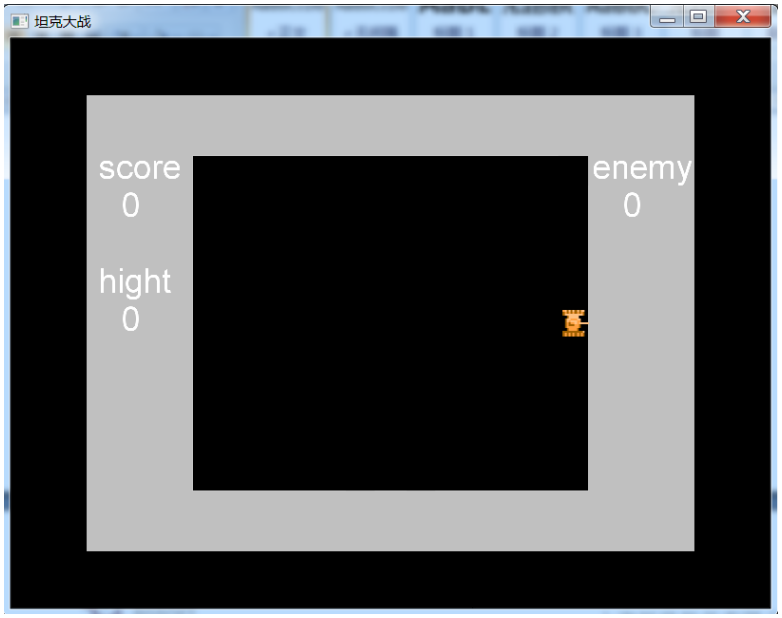
3、通过按键 WSAD 控制坦克上下左右运动；

4、坦克运行到黑色区域边界时不能继续前进。

【实验运行结果】



在区域内运动



运动到区域边界

【实验思路】

坦克也是精灵，但是它具备一些自己独特的功能，比如通过键盘控制运动、开炮等。因此我们可以创建一个新的类 CTankPlayer 类，并让它继承 CSprite 类。

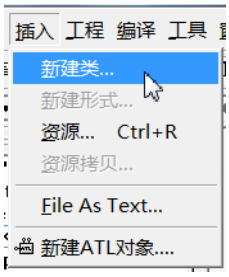
通过 WASD 键，控制坦克做相应的的上左下右运动。按下某个键，给坦克设置相应方向的运动速度；松开时，将该方向的速度设为 0，表示停止运动。

屏幕上最内部的黑色区域为玩家坦克的活动区域，左上角坐标为(-26,-22)，右下角坐标为(26,22)。当坦克运动到该区域边界时，坦克不能继续前进。

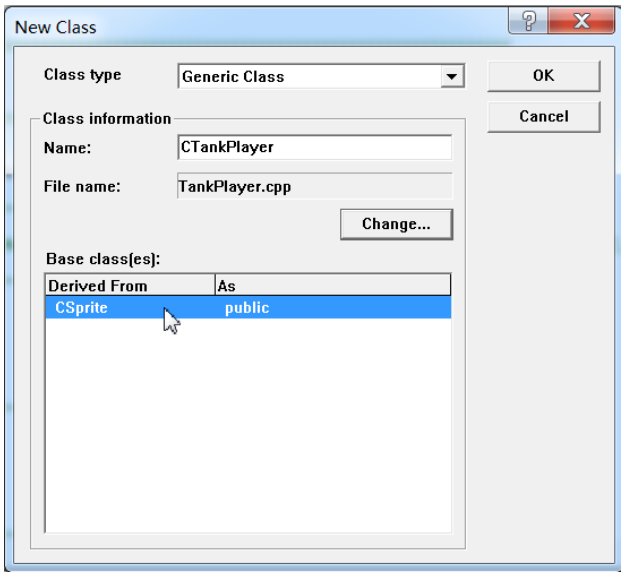
【实验指导】

1、通过类向导创建 CTankPlayer 类，其继承于 CSprite 类。以 VC++ 6.0 为例：

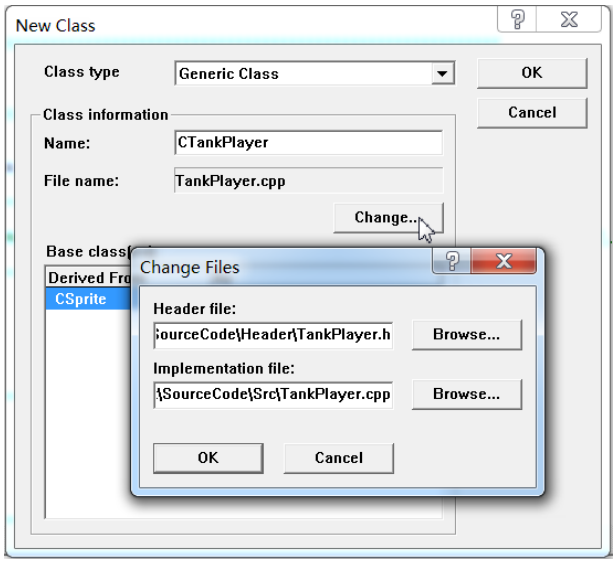
第一步、点击菜单“插入”-〉“新建类”。



第二步、在“New Class”对话框中输入类名和父类名。

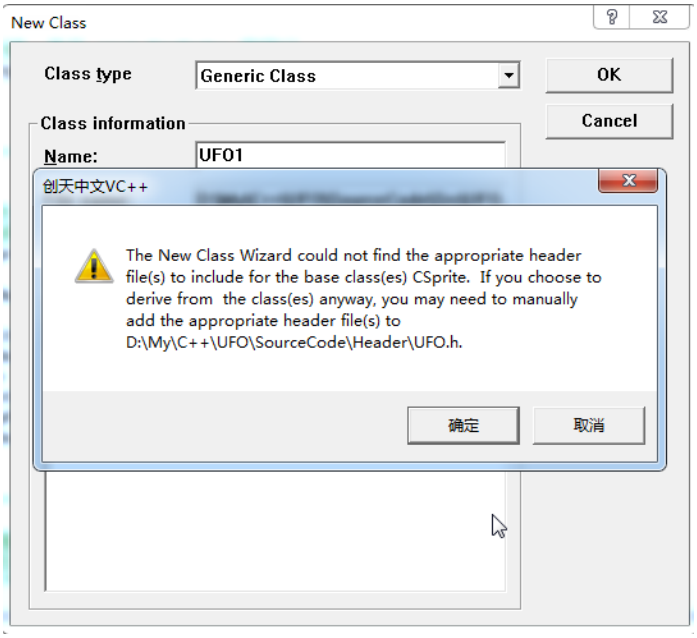


第三步、点击“Change”按钮，在新对话框中修改 CTankPlayer 类的头文件和 cpp 文件的路径。将头文件保存到项目文件夹的\SourceCode\Header 文件夹中，将 cpp 文件保存到项目文件夹下的\SourceCode\Src 文件夹中。



这里需要特别注意的是创建文件路径的问题，所有的.h 头文件应该在项目文件夹\SourceCode\Header 中，所有的.cpp 源文件应该放在项目文件夹下的\SourceCode\Src 文件夹中。这样我们在#include 的时候只需要写文件名就可以了。如果保存到其他路径下，需要写相对路径。

2、点击 OK，出现下面提示，不用管它，点击“确定”。



3、这是因为，我们创建的类集成了 CSprite 类，但是却没有把这个类的声明文件 include 进来。因此我们在新建类的头文件中，做如下操作：

```
#include "CommonClass.h"
```

4、编译程序，发现如下提示：

```
error C2512: 'CSprite' : no appropriate default constructor available
```

打开 CommonClass.h 文件，找到 CSprite 类的定义，可以发现该类没有缺省构造函数，只有一个带参数的构造函数。因此，作为它的子类，构造函数也必须带参数，并且将参数传递给父类。构造函数的声明和定义修改如下：

```
CTankPlayer(const char* szName);
CTankPlayer::CTankPlayer(const char* szName):CSprite(szName)
{
}
```

5、为 CTankPlayer 类添加 m_iDir,m_fSpeedX,m_fSpeedY，m_iHp 四个成员变量（成员变量。用来表示坦克在 X 轴和 Y 轴方向上的速度以及运行的方向，并且在构造函数中初始化为 0。这里规定，m_iDir 的值为 0、1、2、3，分别表示上、右、下、左。成员函数的声明和定义如何添加，访问权限是什么，可参考实验一，下文不再继续提示），分别表示运动方向、X 轴、Y 轴速度以及血量值。**本文档的命名采用匈牙利命名法，**m_表示类成员变量，i 表示整型，f 表示 float 型，sz 表示字符指针，g_表示全局变量等。

同时需要在 public 权限下定义获取和设置这些变量的 Get 和 Set 方法，可在 LessonX.h 文件中完成：

```
//set 方法
void SetHp(int hp)                  {m_iHp = hp;}
void SetDir(int dir)                {m_iDir = dir;}
void SetSpeedX(float speedX)        {m_fSpeedX = speedX;}
void SetSpeedY(float speedY)        {m_fSpeedY = speedY;}
//get 方法
```

```
int GetHp()                    {return m_iHp;}
int GetDir()                   {return m_iDir;}
float GetSpeedX()              {return m_fSpeedX;}
float GetSpeedY()              {return m_fSpeedY;}
```

6、在 CTankPlayer 类的构造函数完成上面四个成员变量的初始化。

```
CTankPlayer::CTankPlayer(const char* szName):CSprite(szName) //对构造函数进行实现
{
    m_iDir=0;
    m_fSpeedX=0.f;
    m_fSpeedY=0.f;
    m_iHp=2;
}
```

子类对象创建时，要先调用父类的构造函数完成父类部分的构造。如果父类没有默认构造函数，子类的构造函数必须显示调用父类的构造函数。CTankPlayer 构造函数调用 CSprite 类构造函数，并将参数 szName 的值传递给它，从而将名称为 szName 的精灵图片与 CTankPlayer 对象绑定起来。

7、为 CTankPlayer 类添加 Init 函数，该函数主要用来完成该类的初始化工作。这里，先调用 setHp 方法设置血量。然后调用父类的 SetSpritePosition 函数将坦克精灵设置在屏幕中央(0,0)处。接下来给坦克精灵设置时间边界的大小，与世界边界的碰撞模式为 WORLD_LIMIT_NULL，表示精灵与世界边界碰撞的响应由代码完成，同时设置坦克碰撞模式为发送碰撞和接收碰撞。

```
void CTankPlayer::Init()
{
    SetHp(2);
    SetSpritePosition(0.f,0.f);
    SetSpriteWorldLimit(WORLD_LIMIT_NULL, -26, -22, 26, 22);
    SetSpriteCollisionActive(1,1);//设置为可以接受和发生碰撞
    SetSpriteVisible(true);
}
```

8、完成 CTankPlayer 类相关定义后，在 CGameMain 类中增加一个私有成员变量 m_pTankplayer，类型为 CTankPlayer*。

注意在 LessonX.h 添加头文件：

```
#include"TankPlayer.h"
```

9、按下空格键后，程序会调用键盘按下事件响应函数，将 m_iGameState 的值改为 1，游戏进入开始状态（见实验一）。程序再次执行 CGameMain::GameMainLoop 函数时，根据 m_iGameState 的值调用并执行 CGameMain::GameInit 函数。因此，可以在该函数中创建我方坦克，并调用 CTankPlayer::Init 函数来完成该类对象的初始化工作。

```
m_pTankPlayer=new CTankPlayer("myPlayer");//新建一个名字是 myPlayer 的我方坦克对象
m_pTankPlayer->CloneSprite("player");//我方坦克克隆在 funcode 模板中存在的名字为 player 的坦克，表示新建的坦克对象有现在精灵的所有属性
m_pTankPlayer->Init();
```

10、接下来，我们为 CTankPlayer 类添加 OnMove 函数，参数 iKey、bPress 分别表示按下的是哪个按键和按键是否按下。首先声明该函数，访问权限为 public：

```
void OnMove(int iKey, bool bPress);
```

11、接着，完成 OnMove 方法。

```
void CTankPlayer::OnMove(int iKey, bool bPress)
{
    if(bPress)
    {
        switch (iKey)
        {
            case KEY_W:
            SetDir(0);
            SetSpeedX(0);
            SetSpeedY(-8);
            break;
            case KEY_D:
            SetDir(1);
            SetSpeedX(8);
            SetSpeedY(0);
            break;
            case KEY_S:
            SetDir(2);
            SetSpeedX(0);
            SetSpeedY(8);
            break;
            case KEY_A:
            SetDir(3);
            SetSpeedX(-8);
            SetSpeedY(0);
            break;
        }
        SetSpriteRotation(float(90*GetDir())); //用方向值乘于 90 得到精灵旋转度数
        SetSpriteLinearVelocity(GetSpeedX(),GetSpeedY());
    }
    else
    {
       if(iKey == KEY_W || iKey == KEY_D || iKey == KEY_S || iKey == KEY_A)
       {
            SetSpeedX(0);
            SetSpeedY(0);
            SetSpriteLinearVelocity(GetSpeedX(),GetSpeedY());
       }
     }
}
```

用参数 bPress 来判断键盘是按下还是弹起。如果 bPress 为 false，表示键盘弹起，且弹起的键是 WASD 键时，设置坦克的 X 轴和 Y 轴移动速度都为 0。如果 bPress 为

true，表示键盘按下，根据按下的键，为 m_iDir,m_fSpeedX,m_fSpeedY 赋予相应的值。SetSpriteRotation 和 SetSpriteLinearVelocity 函数用来设置精灵的角度和线性速度，具体含义参考 CommonClass.h 文件。

12、在 CGameMain 类的 OnKeyDown 函数中，通过调用 CTankPlayer 类的 OnMove 方法，根据按下的键，控制坦克朝指定方向运动。只有游戏进入开始状态，按键才有效。注意：OnMove 参数 bPress 的值为 true。在 OnKeyDown 函数中添加代码如下：

```
if(m_iGameState == 2)
{
   m_pTankPlayer->OnMove(iKey, true);
}
```

13、编译并运行程序。按空格键开始，按 WASD 键，坦克会向规定的方向运动。但是松开按键后，坦克继续前进，不会停止下来。

14、在 CGameMain::OnKeyUp 函数中调用 CTankPlayer 的 OnMove 方法，参数 bPress 的值为 false。

```
void CGameMain::OnKeyUp(const int iKey)
{
    if(m_iGameState == 2)
    {
        m_pTankPlayer->OnMove(iKey, false);
    }
}
```

15、编译并运行程序，按下 WASD 键，坦克运行；松开按键，坦克停止。接下来，我们来实现坦克运行到黑色区域边界停止运动的效果。

16、当坦克精灵碰到边界时，将精灵的速度都设为 0，精灵停止运动。首先，我们需要给该精灵设置世界边界的碰撞模式。可以在 CTankPlayer::GameInit()中完成。

17、用字符串处理函数 strstr()来判断碰到世界边界的精灵是否为我方坦克。当碰到世界边界的精灵名字中含有”player”的时候，strstr 会返回一个非空值。因此代码如下：

```
void CGameMain::OnSpriteColWorldLimit( const char *szName, const int iColSide )
{
    if(strstr(szName,"myPlayer") != NULL) //判断碰到世界边界的坦克是否为我方坦克
    {
       m_pTankPlayer->SetSpriteLinearVelocity(0,0);
    }
}
```

实验三 坦克开炮

【实验内容】

1、创建子弹类 CBullet；

2、通过按键 J 控制坦克开炮；

【实验运行结果】

【实验思路】

创建子弹类，该类应具备子弹的运动方向、起始位置、X 轴速度、Y 轴速度等属性，其中运动方向和起始位置由坦克的方向和位置决定。

通过按键控制坦克发射子弹，因此在 CGameMain 类的 OnKeyDown 方法中实现该需求。此时我们为 CTankPlayer 类添加发射子弹的方法 OnFire。当按下 J 键后，坦克发射子弹，在 OnFire 方法中，我们需要告诉子弹的方向和初始位置。而创建子弹我们由 CGameMain 类来完成。这样减少了类之间的耦合。

【实验指导】

1、通过类向导创建 CBullet 类，其继承于 CSprite 类，具体做法参考实验一；

2、为 CBullet 类添加 m_iDir,m_fSpeedX,m_fSpeedY,m_iHp，m_iOwner 五个成员，分别表示方向、X 轴、Y 轴速度、子弹血量以及发射子弹归属坦克，0 表示地方坦克，1 表示我方坦克，并参考实验二添加变量的 get 和 set 方法。

3、为 CBullet 类添加构造函数和析构函数。参照实验二，把构造函数中将所有变量进行初始化。

4、为 CBullet 类添加 OnMove 方法，参数为 iDir 表示子弹的运动方向。

5、完成 OnMove 方法。根据方向 m_iDir，首先设置 m_fSpeedX,m_fSpeedY 的值，然后设置根据方向设置旋转角度，最后设置子弹的运动速度。

```
void CBullet::OnMove(int iDir)
{
    SetDir(iDir);
    switch(GetDir())
    {
        case 0:
        SetSpeedX(0);
        SetSpeedY(-10);
        break;
        case 1:
        SetSpeedX(10);
        SetSpeedY(0);
```

```
            break;
            case 2:
            SetSpeedX(0);
            SetSpeedY(10);
            break;
            case 3:
            SetSpeedX(-10);
            SetSpeedY(0);
            break;
        }
        SetSpriteRotation(90*GetDir());
        SetSpriteLinearVelocity(GetSpeedX(),GetSpeedY());
    }
```

6、在 CGameMain 类中添加表示子弹数目的成员变量 m_iBulletNum(注意初始化为 0)。然后添加 AddBullet 方法。方法中有 iDir、fPosX、fPosY、iOwner 四个参数分别表示子弹方向、子弹初始位置的 X 轴 Y 轴坐标以及子弹所属坦克。由于地图上只有一个子弹模板，所以我们需要先复制这个模板，然后设置该精灵的世界边界。

```
void CGameMain::AddBullet( int iDir,float fPosX,float fPosY ,int iOwner)
{
    char* szName = CSystem::MakeSpriteName("bullet",m_iBulletNum);//创建坦克名字
    CBullet* pBullet = new CBullet(szName);
    pBullet->CloneSprite("bullet");
    pBullet->SetSpriteWorldLimit(WORLD_LIMIT_NULL,-26, -22, 26, 22); //设置世界边界
    pBullet->SetSpritePosition(fPosX,fPosY);
    pBullet->SetSpriteCollisionSend(true); //设置接收碰撞
    pBullet->OnMove(iDir);
    m_iBulletNum++; //子弹个数加 1
    if(iOwner == 1)
    {
        pBullet->SetOwner(1);//1 表示我方坦克发射的子弹
    }
    else
    {
        pBullet->SetOwner(0); //0 表示地方坦克发射的子弹
    }
}
```

注意：这里用到了 CBullet 类型，因此需要 include 相应的头文件。

7、为 CTankPlayer 类增加 OnFire 方法，实现发射子弹的功能。在根据坦克的运动状态，得到子弹的相关属性后，通过 AddBullet 方法在游戏中增加一发子弹。

```
void CTankPlayer::OnFire()
{
    float x,y;
    x = GetSpritePositionX();
    y = GetSpritePositionY();
```

```
switch(GetDir())
{
    case 0:
    y=y-GetSpriteHeight()/2-1;
    break;
    case 1:
    x=x+GetSpriteWidth()/2+1;
    break;
    case 2:
    y=y+GetSpriteHeight()/2+1;
    break;
    case 3:
    x=x-GetSpriteWidth()/2-1;
    break;
}
g_GameMain.AddBullet(GetDir(),x,y,1);
}
```

因为用到 g_GameMain 这个全局对象，所以需要在 CTankPlayer.cpp 中声明头文件：

```
#include"LessonX.h"
```

8、按 J 键发射子弹。在 CGameMain 类的 OnKeyDown 函数 if(m_iGameState == 2)下添加

```
if(iKey == KEY_J)//判断按下键是够为 J 键
{
   m_pTankPlayer->OnFire();
}
```

实验四 敌方坦克

【实验内容】

1、创建 CTankEnemy 类；

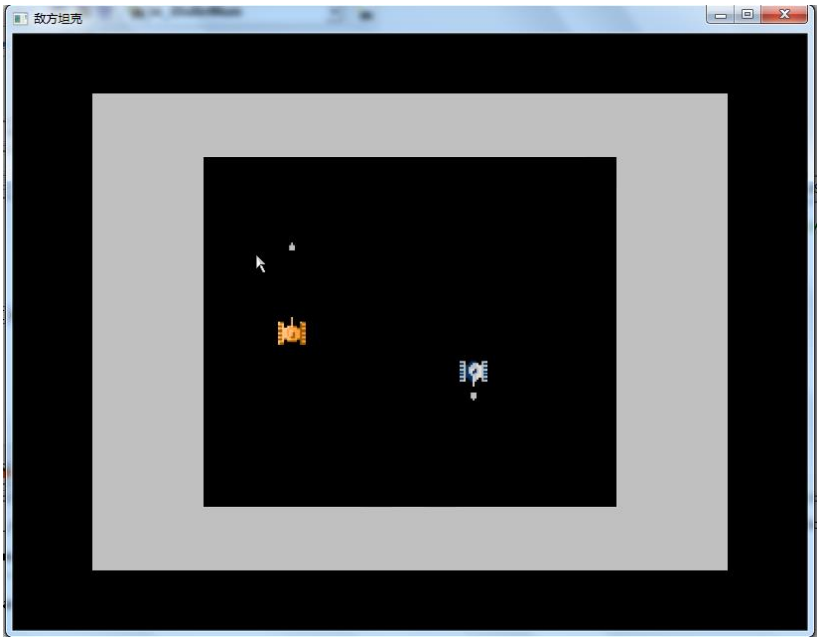
2、创建一个敌方坦克实例，实现坦克从上方左中右三个位置随机出现。

3、实现坦克隔 2 秒，随机改变一个方向。

4、实现坦克隔 5 秒发射一发子弹；

5、当坦克与世界边界碰撞时，改变方向；

【实验运行结果】



【实验思路】

同我方坦克一样，敌方坦克运动，也需要有运动方向、X 轴 Y 轴速度三个基本属性。为了实现坦克能自由运动，即坦克运动一段时间后改变方向，需增加一个改变方向的时间间隔。同时为了实现坦克自动发射子弹，需增加一个发射子弹的时间间隔。

然后为敌方坦克增加移动和发射子弹的方法。

最后我们在 CGameMain 类的 GameRun 方法中调用敌方坦克移动和发射子弹的方法。

【实验指导】

1、通过类向导创建 CTankEnemy 类；

2、参照前面的实验，为 CTankEnemy 类添加构造函数和析构函数。

3、为类 CTankEnemy 添加 m_iDir,m_fSpeedX,m_fSpeedY,m_iHp 四个成员，分别表示方向、X 轴、Y 轴速度和血量；

4、在构造函数中对成员变量进行初始化。

5、为类 CTankEnemy 添加 Init 方法初始化敌方坦克。设置血量为 2，位置坐标为上方左中右的随机位置，然后函数中给坦克精灵设置世界边界的大小，与世界边界的碰撞模式为 WORLD_LIMIT_NULL，表示精灵与世界边界碰撞的响应由代码完成。最后设置方向和速度代码如下：

```
int iPos = CSystem::RandomRange(0,2);
float fPosX;
SetDir(2);
SetHp(2);
switch (iPos)
{
    case 0:
        fPosX = -24.f;
        break;
    case 1:
```

```
            fPosX = 0.f ;
            break;
        case 2:
            fPosX = 24.f;
            break;
        default:
            break;
    }
    SetSpritePosition(fPosX,-20.f);
    SetSpriteLinearVelocity(0.f,8.f);
    SetSpriteCollisionActive(1,1); //设置可以接受和发送碰撞
    SetSpriteRotation(float(90*GetDir()));
    SetSpriteWorldLimit(WORLD_LIMIT_NULL,-26, -22, 26, 22);
```

注意：敌方坦克的位置，是根据黑色屏幕空间以及敌方坦克大小计算出来，这样敌方坦克会偏离边界一点点。

6、为类 CTankEnemy 添加不带参数的 OnMove 方法，实现坦克随机旋转 90 度运动。然后**根据方向设置子弹的运动速度 m_fSpeedX,m_fSpeedY 的值**。与 CTankPlayer 类的 OnMove 方法类似。

设置速度方向与原来方向旋转 90 度的代码如下：

```
int iDir=0;
iDir = CSystem::RandomRange(0,3);
    switch (iDir)
    {
        case 0:
            SetDir(0);
            SetSpeedX(0);
            SetSpeedY(-8);
        break;
        case 1:
            SetDir(1);
            SetSpeedX(8);
            SetSpeedY(0);
        break;
        case 2:
            SetDir(2);
            SetSpeedX(0);
            SetSpeedY(8);
        break;
        case 3:
            SetDir(3);
            SetSpeedX(-8);
            SetSpeedY(0);
        break;
}
```

```
SetSpriteRotation(float(90*GetDir())); //用方向值乘于 90 得到精灵旋转度数
SetSpriteLinearVelocity(GetSpeedX(),GetSpeedY());
```

通过方向设置速度参考前边实验。

7、为类 CTankEnemy 添加带参的 OnMove 方法，参数 fDeltaTime，为游戏的时间差，实现隔一段时间随机改变方向。在 CTankEnemy 类中添加成员变量 m_fChangeDirTime 在构造函数中初始化为 0，并添加 get 和 set 方法。

```
void CTankEnemy::OnMove(float fDeltaTime)
{
    m_fChangeDirTime+=fDeltaTime;
    if(m_fChangeDirTime>2.0f)
    {
        OnMove();
        m_fChangeDirTime = 0.f;
    }
}
```

8、为类 CTankEnemy 添加 OnFire 方法，参数也为 fDeltaTime，实现隔一段时间自动发射子弹。在 CTankEnemy 类中添加成员 m_fBulletCreateTime 变量并在**构造函数中初始化为 0**。

```
void CTankEnemy::OnFire(float fDeltaTime)
{
    m_fBulletCreateTime+=fDeltaTime;
    if(m_fBulletCreateTime>3.0f)
    {
        m_fBulletCreateTime = 0.0f;
        float x,y;
        x = GetSpritePositionX();
        y = GetSpritePositionY();
        switch(GetDir())
        {
            case 0:
                y=y-GetSpriteHeight()/2-1;
                break;
            case 1:
                x=x+GetSpriteWidth()/2+1;
                break;
            case 2:
                y=y+GetSpriteHeight()/2+1;
                break;
            case 3:
                x=x-GetSpriteWidth()/2-1;
                break;
        }
        g_GameMain.AddBullet(GetDir(),x,y,0);
```

```
        }
    }
```

因为用到全局对象 G_GameMain，所以需要在 TankEnemy.cpp 中声明头文件：

```
#include"LessonX.h"
```

9、在 CGameMain 类中添加 CTankEnemy 类指针变量 m_pTankEnemy。注意在 LessonX.h 中声明头文件：

```
#include"TankEnemy.h"
```

然后在 GameInit 中创建并初始化。

```
m_pTankEnemy = new CTankEnemy("enemy");
m_pTankEnemy->Init();
```

10、在 CGameMain 类的 GameRun 方法中实现敌方坦克自由移动和发射子弹。

```
void CGameMain::GameRun( float fDeltaTime )
{
   if(m_pTankEnemy)
   {
      m_pTankEnemy->OnMove(fDeltaTime);
      m_pTankEnemy->OnFire(fDeltaTime);
   }
}
```

11、在 CGameMain 类的 OnSpriteColWorldLimit 方法中添加敌方坦克与世界边界碰撞的检测。

```
if(m_pTankEnemy&&strcmp(m_pTankEnemy->GetName(),szName)==0)
{
   m_pTankEnemy->OnMove();
}
```

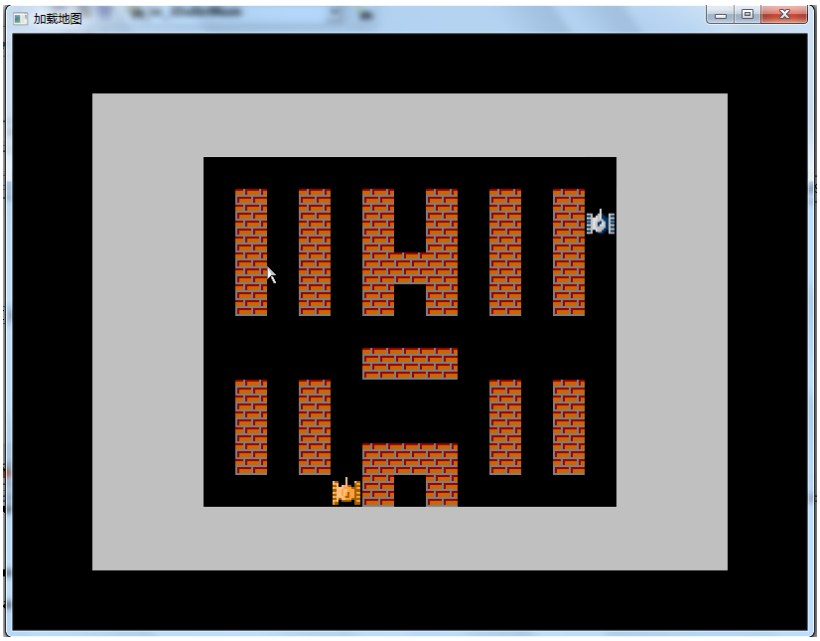
注意：有时候敌方坦克碰到世界边界会一直打转而不能离开，这是因为坦克不管怎么转向，始终都跟世界边界碰撞。处理办法之一，根据坦克转向，将坦克的位置稍微内移，确保不会始终碰到世界边界。

实验五 加载地图

【实验内容】

1、加载游戏地图；

【实验运行结果】



【实验思路】

在 CGameMain 类中添加方法 LoadMap 实现地图加载。地图数据是已知的，数据中的 0 表示此处为空；1 表示的是地图中此处为墙；2 表示此处为玩家指挥部。每块墙大小为 4*4。

【实验指导】

1、首先在 LessonX.cpp 中定义一个表示地图的二维数组来表示地图，其中 0 表示没有墙块，1 表示有墙块：

```
int g_iMap[11][13]=
{
    {0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,1,0,0,0,1,0,1,0,0,0,1,0},
    {1,1,1,1,1,1,1,1,1,1,1,1,1},
    {0,0,0,1,0,1,0,1,0,1,0,0,0},
    {0,0,0,1,0,0,0,0,0,1,0,0,0},
    {0,0,0,1,0,1,0,1,0,1,0,0,0},
    {1,1,1,1,1,1,1,1,1,1,1,1,1},
    {0,1,0,1,0,0,0,0,0,1,0,1,0},
    {0,0,0,0,0,1,1,1,0,0,0,0,0},
    {0,0,0,0,0,1,0,1,0,0,0,0,0}
};
```

2、在 CGameMain 类中添加 LoadMap 方法，实现加载地图。在方法中用 for 循环遍历地图二维数组，当二维数组中的某个值为 1 时表示该处是一堵墙。

```
void CGameMain::LoadMap()
{
        char* szName;
        int i,j;
```

```
float x,y;
for(i=0;i<11;i++)
{
    for(j=0;j<13;j++)
    {
        if(g_iMap[i][j]==1)
        {
            szName = CSystem::MakeSpriteName("wall",j+i*13+i);//重新起名
            CSprite* pWall = new CSprite(szName); //新建对象
            pWall->CloneSprite("wall"); //克隆墙块
            pWall->SetSpriteCollisionActive(0,1); //设置为接受碰撞
            pWall->SetSpriteCollisionResponse(COL_RESPONSE_CUSTOM);
            x =float(-24+4*j);
            y =float(-20+4*i);
            pWall->SetSpritePosition(x,y);
        }
    }
}
}
```

3、 在 GameInit 中调用加载地图的函数：

```
LoadMap();
```

实验六 创建父类-CWeapon 类

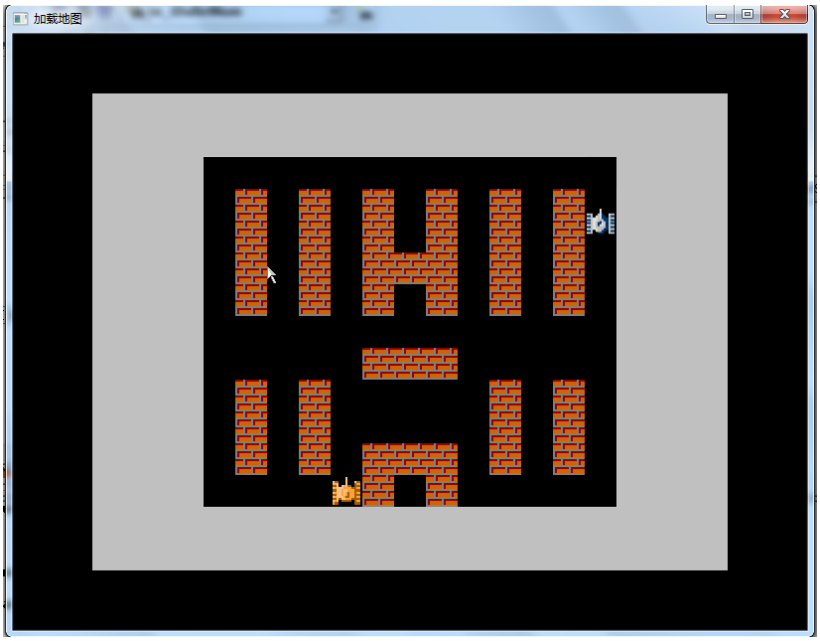
【实验内容】

1、添加 CWeapon 类，使 CBullet、CTankPlayer、CTankEnemy 类继承 CWeapon 类。

2、将三个子类 OnFire 和 OnMove 方法中相同的功能抽取出来，在父类中创建相应方法。子类调用父类方法进行改进。

3、创建一个容器，用来存放所有精灵。

【实验运行结果】



【实验思路】

1、CBullet、CTankPlayer、CTankEnemy 类中有许多共同的属性和方法，或者为了方便管理，我们添加父类 CWeapon。

2、把所有创建的精灵存入到容器中，增加从容器中查找和删除精灵的函数。在此基础上，对整个项目进行大改。

注意此步骤对于下方实验非常重要，需要理解继承的用法，根据自己的理解把该删除和更改的地方做好。此实验会创建一个包含内容为 CWeapon* 的容器，把创建的我方坦克，敌方坦克，墙块，子弹，全部存入到这个容器中进行管理。

【实验指导】

1、通过类向导添加 CWeapon 类，其继承于 CSprite 类。

2、根据分析为 CWeapon 类添加表示方向、血量、X 轴 Y 轴速度、四个变量。

```
int     m_iDir;
int     m_iHp;
float m_fSpeedX;
float m_fSpeedY;
```

添加 get 和 set 方法；

3、仿照前面的实验添加构造函数，在构造函数中对这些数值初始化，其中 m_iHp 初始化为 2，其他初始化为 0；

4、添加函数以及虚函数：

```
bool    IsDead();     //判断精灵是否死亡
virtual    void Init(){}; //初始化函数
virtual    void OnMove(float fDeltaTime){};    //敌方坦克移动函数
virtual    void OnMove(){};
virtual    void OnFire(float deltaTime){};    //发射子弹函数
virtual    void OnSpriteColSprite(CWeapon* pSprite){}; //精灵与精灵碰撞时处理函数
```

在 CWeapon 类中实现 IsDead()函数，此函数的作用是判断精灵是否死亡，在游戏运

行时，会根据这个函数的返回值来确定是否此精灵：

```
boolCWeapon::IsDead()
{
    if(m_iHp == 0)
    {
        return true;
    }
    return false;
}
```

5、将 CBullet、CTankPlayer、CTankEnemy 类的父类改为 CWeapon。

例如在 TankPlayer.h 中把 class CTankPlayer :public CSprite

改为 class CTankPlayer :public CWeapon

注意需要 Bullet.h、TankPlayer.h、TankEnemy.h 包含头文件:

```
#include"Weapon.h"
```

6、注意同时修改构造函数。把继承类的名称由 CSprite 改为 Cweapon。**删除与 CWeapon 定义重复的变量声明，原来的 get 和 set 方法，以及在构造函数中对这些变量赋初值的代码。特别主要要设置 Hp=2。**

7、在 CGameMain 类中添加一个成员变量 m_vWeapon，用来存储所有精灵：

```
vector<CWeapon*>     m_vWeapon;
```

注意用到容器，添加头文件声明和命名空间：

```
#include”Weapon.h”
#include<vector>
using namespace std;
```

8、然后声明和定义查找和删除容器成员的两个函数 FindWeaponByName 和 DeleteWeaponByName：

```
CWeapon* CGameMain::FindWeaponByName(const char* szName)//根据名字查找到对象
{
    for(int i=0; i<m_vWeapon.size(); i++)
    {
        if(strcmp(szName,m_vWeapon[i]->GetName()) == 0)
        {
            return m_vWeapon[i];
        }
    }
}

void CGameMain::DeleteWeaponByName(const char* szName)//根据名字把精灵从容器中删除
{
    for(vector<CWeapon*>::iterator it=m_vWeapon.begin();it!=m_vWeapon.end();)
    {
        CWeapon* cw =*it;
        if(strcmp(szName,cw->GetName()) == 0)
```

```
            {
                m_vWeapon.erase(it);
                cw->DeleteSprite();
                delete cw;
                break;
            }
            else
            {
                it++;
            }
        }
    }
```

9、在前边创建**子弹、我方坦克、以及地图**的地方用 m_vWeapon.push_back()函数把精灵添加到容器中。

1）将我方精灵添加容器中。之前我们在GameInit()函数中创建我方坦克和敌方坦克对象。在这个函数末尾增加下方代码：

下边添加如下代码：

```
m_vWeapon.push_back(m_pTankPlayer);
```

注意：下一个实验中，我们会创建多个敌方坦克对象，所以这里暂时不把敌方坦克放入容器中。

2）子弹添加到容器，在AddBullet函数最后添加一行代码：

```
m_vWeapon.push_back(pBullet);
```

3）地图添加容器，在LoadMap()函数中，把CSprite* pWall = new CSprite(szName);改为：

```
CWeapon* pWall = new CWeapon(szName);
```

然后在pWall->SetSpritePosition(x,y);下添加：

```
m_vWeapon.push_back(pWall);
```

实验七 游戏整体正常运行

【实验内容】

1、创建敌方坦克每隔 5 秒出现一辆，从屏幕上方的左、中、右三个位置随机出现。

2、当子弹与游戏中的任何东西碰撞时，双方都消失；

3、当我方坦克与敌方坦克碰撞时，我方坦克静止不动，敌方坦克改变方向；

4、当我方坦克被消灭或者我方军营被击中时，游戏结束。

【实验运行结果】

【实验思路】

3、创建一个函数生成坦克的函数产生坦克。

4、在 OnSpriteColSprite 方法中添加碰撞检测。

5、更改在 OnSpriteColWorldLimit 中与世界边界的碰撞检测。

【实验指导】

1、注释掉实验四中，在CGameMain类中添加的声明和创建一个敌方坦克对象的代码。

LessonX.h中：

```
//CTankEnemy* m_pTankEnemy;
```

在 Lesson.cpp 文件 GameInit 函数：

```
//m_pTankEnemy = new CTankEnemy("enemy");
//m_pTankEnemy->Init();
```

同时注释在 GameRun 函数中添加的使敌方坦克运动的代码。

```
/*if(m_pTankEnemy)
{
m_pTankEnemy->OnMove(fDeltaTime);
m_pTankEnemy->OnFire(fDeltaTime);
}*/
```

以及注释在OnSpriteColWorldLimit函数中添加的使敌方坦克运动的代码。

```
/**if(m_pTankEnemy&&strcmp(m_pTankEnemy->GetName(),szName)==0)
{
m_pTankEnemy->OnMove();

}**/
```

2、在CGameMain中创建一个不断产生地方坦克的函数AddTankEnemy。

先在CGameMain中添加一个成员变量m_fTankEnemyTime和m_iTankEnemyNumber分别代表创建坦克时间，以及坦克数量(**注意将两数进行初始化**)。当时间

m_fTankEnemyTime大于5时，创建敌方坦克，设置属性，并进行调用初始化函数init来设置敌方坦克的初始位置以及碰撞模式等内容。

```
voidCGameMain::AddTankEnemy(float fDeltaTime)
{
    m_fTankEnemyTime += fDeltaTime;
    if(m_fTankEnemyTime > 5)
    {
        char* szName = CSystem::MakeSpriteName("enemy",m_iTankEnemyNumber);
        CTankEnemy* m_pTankEnemy = new CTankEnemy(szName);
        m_pTankEnemy->CloneSprite("enemy");
        m_pTankEnemy->Init();
        m_iTankEnemyNumber++;
        m_vWeapon.push_back(m_pTankEnemy);   //把创建的敌方坦克插入到容器中
        m_fTankEnemyTime=0.f;
    }
}
```

在 CGameMain 类的 GameRun 函数中调用增加敌方坦克的函数：

```
AddTankEnemy(fDeltaTime);
```

此时运行游戏可以看到，每五秒有一辆敌方坦克从下方随机生成并向下运动。

3、生成敌方坦克精灵以后，需要让它们运动起来，并不时发射子弹。在 GameRun 中添加如下代码：

```
for(int i=0;i<m_vWeapon.size();i++)
{
    m_vWeapon[i]->OnMove(fDeltaTime);
    m_vWeapon[i]->OnFire(fDeltaTime);
}
```

注意：我们直接把容器中对象调用出来，然后调用运动和开火的函数。为什么可以这样？这就是多态的好处。大家思考一下原因是什么。

4、接下来判断精灵是否死亡，把已死亡的精灵进行删除，使游戏运行正常。

```
for(int i=0;i<m_vWeapon.size();i++)
{
    if(!m_vWeapon[i]->IsDead())
    {

        m_vWeapon[i]->OnMove(fDeltaTime);
        m_vWeapon[i]->OnFire(fDeltaTime);
    }
    else
    {
        DeleteWeaponByName(m_vWeapon[i]->GetName());
    }
}
```

5、但是我们看到上面步骤实际并没有启到预想的效果。这是因为我们还没有处理精灵与精灵的碰撞。CWeapon 中有个 OnSpriteColSprite 的虚函数，具体实现我们在各个子类中完

成。

6、在 CTankPlayer 类中添加我方坦克与其他精灵碰撞的处理函数，使我方坦克碰到子弹后，血量设为 0，碰到墙或敌方坦克后，设置速度为 0：

```
void CTankPlayer::OnSpriteColSprite(CWeapon* pSprite)
{
	if(pSprite == NULL)
	{
		return;
	}
	else if(strstr(pSprite->GetName(),"bullet") != NULL)
	{
		SetHp(0);
	}
	else if(strstr(pSprite->GetName(),"wall")!= NULL|| strstr(pSprite->GetName(),"enemy") != NULL)
	{
		SetSpeedX(0);
		SetSpeedY(0);
		SetSpriteLinearVelocity(GetSpeedX(),GetSpeedY());
	}
}
```

7、在CBullet类中添加子弹与其他精灵碰撞后的判断代码，判断子弹与其他精灵碰撞发生的反应，当我方子弹与军营发生碰撞，地方坦克发射的子弹与敌方坦克发送碰撞时，只设置子弹血量为0，其他不做处理。其他情况，设置被碰撞精灵血量为0：

现在 Bullet.h 中添加函数的声明：

```
void OnSpriteColSprite(CWeapon* pSprite);
```

再在 Bullet.cpp 中添加函数定义：

```
void CBullet::OnSpriteColSprite(CWeapon* pSprite)
{
	if(pSprite == NULL)
	{
		return;
	}
	SetHp(0);
	if(GetOwner() == 1 && strstr(pSprite->GetName(),"aim_nor") != NULL)    //我方坦克子弹与军营发生碰撞
	{
		return;
	}
	if(GetOwner() == 0 && strstr(pSprite->GetName(),"enemy") != NULL) //敌方坦克子弹打中地方坦克
	{
		return;
```

```
        }
        pSprite->SetHp(0);
    }
```

8、在 CTankEnemy 类中添加地方坦克与其他精灵碰撞的处理函数，设置速度为 0，并设置转换方向的时间 m_fChangeDirTime 为 1.8，在 CGameMain 类 GameRun 函数中，当 m_fChangeDirTime>2 时，敌方坦克转向。所以在 0.2 秒之后，这个坦克会自动转向。

```
void CTankEnemy::OnSpriteColSprite(CWeapon* pSprite)
{
    if(pSprite == NULL)
    {
        return;
    }
    SetSpriteLinearVelocity(0.f,0.f);
    m_fChangeDirTime = 1.8;
}
```

9、在 CGameMain 的 OnSpriteColSprite 中添加碰撞检测。进行的检测顺序是，首先判断发送碰撞方为谁，然后调用发送碰撞对象的碰撞处理函数，代码如下：

```
voidCGameMain::OnSpriteColSprite(const char *szSrcName, const char *szTarName)
{
    CWeapon* tarSprite = FindWeaponByName(szTarName);
    if(strstr(szSrcName,"bullet") != NULL)//发送碰撞为子弹
    {
        CBullet *tmpBullet = (CBullet*)FindWeaponByName(szSrcName);
        tmpBullet->OnSpriteColSprite(tarSprite);
        if( tmpBullet->GetOwner()==1 && strstr(szTarName,"enemy") != NULL)
        {
            m_iScore++;
            m_iEnemy--;
        }

    }
    else if(strcmp(szSrcName,"myPlayer")==0) //发送碰撞为我方坦克
    {
        m_pTankPlayer->OnSpriteColSprite(tarSprite);
    }
    else if(strstr(szSrcName,"enemy") != NULL)//发送碰撞为敌方坦克
    {
        CTankEnemy* tmpEnemy = (CTankEnemy*)FindWeaponByName(szSrcName);
        tmpEnemy->OnSpriteColSprite(tarSprite);
    }
}
```

现在运行程序，我方坦克或敌方坦克发射子弹都能摧毁目标；前进时遇到到障碍（墙或坦克）都要停止。

10、当敌方坦克碰到世界边界时，需要顺时针调转 90 度。子弹碰到世界边界时，则会被删除。在 CGameMain::OnSpriteColOnWorldLimit 中完成如下的代码：

```
else if(strstr(szName,"enemy") != NULL)
{
	CWeapon* pEnemy = FindWeaponByName(szName);
	pEnemy->SetSpriteLinearVelocity(0.f,0.f);

	switch(iColSide) {    //处理碰撞边界打转的问题
	case 0:
	pEnemy->SetSpritePosition(pEnemy->GetSpritePositionX() +
	0.5f,pEnemy->GetSpritePositionY());
	break;
	case 1:
	pEnemy->SetSpritePosition(pEnemy->GetSpritePositionX() -
	0.5f,pEnemy->GetSpritePositionY());
	break;
	case 2:
	pEnemy->SetSpritePosition(pEnemy->GetSpritePositionX(),pEnemy->GetSpritePositionY() +
	0.5);
	break;
	case 3:
	pEnemy->SetSpritePosition(pEnemy->GetSpritePositionX() ,pEnemy->GetSpritePositionY()-
	0.5f);
	break;
	default:
	break;
	}

	pEnemy->OnMove();
}
else if(strstr(szName,"bullet") != NULL)
{
	CWeapon* pBullet = FindWeaponByName(szName);
	pBullet->SetHp(0);
}
```

11、在 CGameMain 中添加对 CWeapon *类型对象 m_pAim_nor 表示我方军营。

在 GameInit 中创建我方军营精灵，把军营精灵插入到容器中，设置接收碰撞以及坐标。

```
m_pAim_nor = new CWeapon("myaim_nor");
m_pAim_nor->CloneSprite("aim_nor");
m_vWeapon.push_back(m_pAim_nor);
m_pAim_nor->SetSpriteCollisionReceive(true);
m_pAim_nor->SetSpritePosition(0.f,20.f);
```

实验八 显示游戏信息

【实验内容】

1、显示游戏的得分等信息。

2、完善游戏，使游戏能够循环运行。

3、记录游戏的分数，当分数高于上次的得分时，将此次得分写入文件。

【实验运行结果】

【实验思路】

在 OnSpriteColSprite 碰撞检测中，每消灭一辆地方坦克加 1 分，同时敌方坦克数量减 1。

完善 CGameMain 中的 GameInit，GameEnd 等，使程序能够循环运行。

在 CGameMain 类的 GameInit 方法中，读取游戏的最高分。在游戏结束 GameEnd 方法中，记录游戏的最高分。

【实验指导】

1、在 CGameMain 中添加以下成员变量：

```
CTextSprite*  m_pScore; //表示分数的文本精灵
 CTextSprite*  m_pHight; //表示最高分的文本精灵
CTextSprite*  m_pEnemy;//表示敌人数量的文本精灵
 int m_iScore;//分数
 int m_iEnemy;//敌人数量
 int m_iHight;//几局最高分
 float    m_fDeltaTime;//表示游戏时间
```

在构造函数中创建这三个文本精灵对象并初始化变量：

```
m_pScore = new CTextSprite("score");
m_pHight = new CTextSprite("hight");
m_pEnemy = new CTextSprite("enemyNum");
```

```
m_iScore=0;//分数
im_iEnemy=0;//敌人数量
m_iHight = 0;
m_fDeltaTime = 0.f;
```

2、在 AddTankEnemy()中的 if(m_fTankEnemyTime > 5)判断里添加代码使坦克数量增长。

```
m_iEnemy++;
```

3、在 OnSpriteColSprite 函数中，在判断发送碰撞为子弹的 if(strstr(szSrcName,"bullet") != NULL) 下添加代码：

```
if( tmpBullet->GetOwner()==1 && strstr(szTarName,"enemy") != NULL)
{
    m_iScore++;
    m_iEnemy--;
}
```

4、在 GameInit 方法初始化变量，并打开记录文件，如果文件不存在则创建该文件，如果存在，则读取数据。并显示在游戏中。

```
m_iBulletNum      =     0;
m_iTankEnemyNumber   =   0;
m_fTankEnemyTime = 4.f;
m_iScore = 0;
m_iHight = 0;
m_iEnemy = 0;
m_fDeltaTime = 0.f;

FILE * fp =fopen("save.dat","r+");
if(fp)
{
    fread(&m_iHight,sizeof(int),1,fp);
    fclose(fp);
}
m_pHight = new CTextSprite("hight");
m_pHight->SetTextValue(m_iHight);
```

5、表示游戏时间的成员变量 m_fDeltaTime 在 GameRun 自增：

```
m_fDeltaTime += fDeltaTime;
```

6、实时显示游戏数据，在 GameRun 函数中添加代码：

```
m_pScore->SetTextValue(m_iScore);
m_pHight->SetTextValue(m_iHight);
m_pEnemy->SetTextValue(m_iEnemy);
```

7、设置在我方坦克死亡，我方军营死亡，或者游戏时间>30 秒的条件下，游戏结束。

更改 CGameMain 类 GameMainLoop 函数中 case2 的判断条件：

将 if(true)改为

```
if(!m_pTankPlayer->IsDead() && !m_pAim_nor->IsDead() && m_fDeltaTime<30)
```

8、创建删除所有精灵的函数 DeleteAllSprite()：

```
voidCGameMain::DeleteAllSprite()
{
    int n=m_vWeapon.size();
    while(m_vWeapon.size()!=0)
    {
        vector<CWeapon*>::iterator itr=m_vWeapon.begin();
        CWeapon* cw = *itr;
        m_vWeapon.erase(itr);
        cw->DeleteSprite();
        delete cw;
    }
}
```

在 GameEnd 中调用此函数。

9、在 GameEnd 方法中，当游戏的得分大于最高分时，将游戏的得分写入文件中。

```
FILE * fp =fopen("save.dat","w+");
if(m_iScore>m_iHight)
fwrite(&m_iScore,sizeof(int),1,fp);
fclose(fp);
```

10、最后游戏结束后，设置游戏开始界面可见并游戏状态为 0。最后在 GameEnd 中添加代码：

```
m_pSplash->SetSpriteVisible(true);
m_pStart->SetSpriteVisible(true);
SetGameState(0);
```