**坦克大战**

1. **环境**
2. 语言：c++
3. 编译器版本号：MSVC Version: 1929
4. 应用程序开发框架：Qt version 5.9.1
5. **游戏内容：**
6. **游戏规则：**

本游戏为玩家操纵黄色坦克守卫堡垒，获胜条件为消灭四个绿色的敌方坦克，一旦玩家自己操纵的坦克被子弹击中或者堡垒被子弹击中则游戏失败。

玩家坦克由键盘WASD控制， J发射子弹，敌方坦克由电脑控制

1. **代码实现：**

**（1）地图设置：**

创建ground头文件及ground.cpp文件，用来初始化地图设置map二维数组用来表示地图，其中，‘1’代表可通行的道路，‘2代表’可被子弹击毁的墙，‘3’代表子弹可穿过坦克不可穿过的河流，‘4’代表子弹不可击毁的墙。

**（2）地形设置：**

创建QPixmap类的对象wall、river、hardwall用于表示墙、河流、硬墙，并利用scaled函数统一大小

1. QPixmap wall;
2. QPixmap river;
3. QPixmap hardwall;
4. QPixmap camp;
5. wall.load((rootdir + "wall.gif").c\_str());
6. wall.scaled(50, 50, Qt::AspectRatioMode::IgnoreAspectRatio, Qt::TransformationMode::SmoothTransfor mation);

（river、hardwall类似）

**（3）坦克、子弹、堡垒创建：**

首先创建Bullet类，创建如下成员变量及成员函数：

1. **int** speed;
2. **bool** flying;//是否在飞行
3. Face face;
4. QPixmap upimg;
5. QPixmap downimg;
6. QPixmap leftimg;
7. QPixmap rightimg;
8. QRect bullrect;//子弹本身的矩形
9. **void** setbullet();
10. **void** setstate(**bool**);//设置子弹飞行状态
11. **bool** getstate();
12. **void** move();
13. **void** Faceto(Face);
14. **void** paintbullet(QPainter&);//画子弹
15. **bool** ifreach(**int**, **int**, **int**,**int**);//同坦克
16. Bullet();

创建QRect类成员变量bullrect，以作为子弹在游戏中的实际存在，其左上角坐标/50，作为子弹在map数组中的位置，QPixmap类成员变量upimg、downimg、leftimg、rightimg来绘制子弹各个方向的图像，以flying确定子弹状态、setstate改变子弹状态、getstate获得子弹状态、move控制子弹移动，faceto设置子弹移动方向，ifreach判断子弹能否继续运动

创建Tank基类，在Tank基类中创建如下成员变量及成员函数

1. **int** speed;
2. QPixmap upimg;//各个方向的坦克的图像
3. QPixmap downimg;
4. QPixmap rightimg;
5. QPixmap leftimg;
6. QRect rect;//坦克本身在窗口中占的大小
7. Face face;//方向
8. Bullet bullet;
9. **void** shot();
10. **virtual** **void** move();//坦克移动
11. **virtual** **void** Faceto(Face f);//确定坦克朝向
12. **virtual** **void** paintTank(QPainter& p);//绘制坦克
13. **bool** ifreach(**int**, **int**, Face);//判断是否越界

创建QRect类成员变量rect，以作为坦克在游戏中的实际存在，其左上角坐标/50，作为坦克在map数组中的位置，设置shot类控制子弹发射、创建虚函数move、faceto、painttank、isreach作用如注释

同时创建玩家坦克类和敌方坦克类，二者均继承Tank基类，并在构造函数中完成对speed、ismove等成员变量的定义以用于控制坦克移动，同时均设置成员函数settankdir以设置坦克四个方向的图像（upimg等），设置faceto设置坦克运动方向

最后设置QRect类成员campRect（在Widget类中创建），用以创建堡垒矩形，并设置QPixmap类用以绘制

对子弹坦克类创建完成之后，在widget类中创建QList类enemies，

enemies包含四个enemytank类成员，用enemynum确定，用以完成对敌方坦克的创建，创建player类，作为玩家坦克。

1. QList<EnemyTank>enemies;
2. QPainter painter;
3. Mytank player;
4. **int** enemynum;
5. **窗口设置：**

将窗口大小设置为600\*600，由于map数组为12\*12二维数组，则每个单位在窗口中占据大小为50\*50。

创建Widget类，继承自QWidget类，

1. **public**:
2. Widget(QWidget\* parent = nullptr);
3. **void** buildMyTank();
4. **void** buildEnemyTank();
5. **void** ifcollision();
6. **private**:
7. **void** build();//构建地图
8. **void** paintEvent(QPaintEvent\*) override;
9. **void** Gameover();

Qt的QWidget类提供了绘制窗口的虚函数paintEvent，直接重载就能实现窗口的绘制，当界面更新时自动调用该函数。

诸如buildMyTank等函数用以创建坦克、地图等图形，Gameover函数用以游戏结束对话框显示，在**build**函数中调用buildmytank、buildenemytank函数，对坦克坐标、大小、状态、其子弹状态进行设置

**（5）按键设置**

在widget类中重载如下按键函数

1. **void** keyPressEvent(QKeyEvent\*) override;//按键事件
2. **void** keyReleaseEvent(QKeyEvent\* event) override;//松开按键

利用上述两个函数，按下wasdj等键以移动、发射子弹，控制玩家坦克状态。

**（6）有关坦克子弹移动**

首先创建枚举类型Face{up0,down0,left0,right0}作为方向变量

我们已经在Tank类中创建成员函数move，利用设置好的方向变量与speed变量改变坐标位置。

1. **int** x = rect.x(), y = rect.y();
2. **if** (face == Face::up0)
3. {
4. y -= speed;
5. }
6. **else** **if** (face == Face::down0)
7. {
8. y += speed;
9. }
10. **else** **if** (face == Face::left0)
11. {
12. x -= speed;
13. }
14. **else** **if** (face == Face::right0)
15. {
16. x += speed;
17. }
18. **if** (ifreach(x, y, **this**->face))
19. {
20. rect.moveTo(x, y);
21. }

方向变量的设置：

①子弹：

Tank类中创建了成员变量bullet，成员函数shot

1. **void** Tank::shot()
2. {
3. //子弹已经射出就 return
4. **if** (bullet.getstate())
5. **return**;
6. bullet.setstate(**true**);
7. bullet.Faceto(face);
8. bullet.bullrect.setRect(rect.x() , rect.y(), 50, 50);
9. }

shot函数将坦克的方向变量一并赋给子弹。

对于玩家坦克：当玩家坦克接受到keyPressEvent函数的指令同时玩家坦克的子弹状态为未飞行则会调用player.shot，

1. **else** **if** (event->key() == Qt::Key::Key\_J)
2. {
3. **if** (!player.bullet.getstate())
4. {
5. player.shot();
6. }
7. }

对于敌方坦克：shot的调用与定时器有关，放在后面讨论。

②玩家坦克：

玩家坦克接受keyPressEvent函数的指令，通过按键确定玩家坦克的方向，

1. **if** (event->key() == Qt::Key::Key\_W)
2. {
3. player.Faceto(Face::up0);
4. }
5. **else** **if**(event->key() == Qt::Key::Key\_A)
6. {
7. player.Faceto(Face::left0);
8. }
9. **else** **if** (event->key() == Qt::Key::Key\_S)
10. {
11. player.Faceto(Face::down0);
12. }
13. **else** **if**(event->key() == Qt::Key::Key\_D)
14. {
15. player.Faceto(Face::right0);
16. }

③敌方坦克（BFS算法求无权图最短路径）：

利用BFS算法，为敌方坦克设置最短路径，即转化为无权图最短路径问题。代码实现如下：

1. **int** dx[4] = { 1,0,0,-1 };
2. **int** dy[4] = { 0,1,-1,0 };
3. std::vector<**int**>path(**int** x, **int** y, **int** u, **int** v)
4. {
5. **bool** visited[**sizeof**(map)]; **int** d[**sizeof**(map)]; **int** path[**sizeof**(map)];
6. std::queue<**int**> empty;
7. std::queue<**int**>q;
8. **for** (**int** i = 0; i < **sizeof**(map); ++i)
9. {
10. d[i] = **false**; //单源到各点路径长度的最短路径,先初始化,false代表不可到达
11. path[i] = -1;  //最短路径从哪个顶点过来,先初始化
12. visited[i] = **false**;
13. }
14. **int** p = y \* 12 + x;
15. **int** end = v \* 12 + u;
16. d[p] = 0; q.push(p);//顶点p入队列
17. visited[p] = **true**; //标记顶点p已被标记
18. **while** (!q.empty())//主过程
19. {
20. **int** k = q.front(); q.pop();
21. **for** (**int** i = 0; i <= 4; i++)
22. {   //遍历当前出队列的元素的所有邻接顶点,第一次为遍历顶点u的所有邻接顶点
23. //当前出队列的元素即跳出for循环之后,再进入for循环时,本例中,u即为1号元 素
24. **int** kx = k % 12; **int** ky = k / 12;
25. **int** wx = kx + dx[i]; **int** wy = ky + dy[i];
26. **int** w = wy \* 12 + wx;
27. **if** (wx >= 0 && wx <= 11 && wy >= 0 && wy <= 11&& !visited[w]&& map[wy][wx] != '3' && map[wy][wx] != '4') //w为u为尚未访问的邻接顶点
28. {
29. d[w] = d[k] + 1;//路径长度加1
30. path[w] = k; //最短路径为k到w
31. visited[w] = **true**;//标记顶点w已被标记
32. **if** (w == end)//如果w到达终点
33. {
34. swap(empty, q);
35. **break**;
36. }
37. **else**
38. q.push(w);//顶点w入队列
39. }
40. }
41. }
42. std::vector<**int**>Path;
43. **int** size = empty.size(); **int** k = end;
44. **while** (k != y \* 12 + x)
45. {
46. Path.push\_back(k);
47. k = path[k];
48. }
49. std::reverse(Path.begin(), Path.end());
50. **return** Path;
51. }

则调用该函数我们可以得到储存两坐标间的最短路径vector数组。

对于四辆敌方坦克，我们令QList中的第一位敌方坦克（命名为killer）始终追踪玩家坦克，则调用函数时将两者的坐标作为参数，得到最短路径数组，方向便由数组第0位的坐标与killer坐标确定，并在利用定时器在killer函数中不断更新两者坐标，不断改变Killer移动方向。令其他三辆坦克的目标为camp堡垒，则调用函数以堡垒与敌方坦克的坐标作为参数即可，并再assault函数中不断更新改变移动方向。

1. **void** Widget:: killer()//四个坦克中的第一个追踪玩家坦克
2. {
3. EnemyTank&killer = enemies[0];
4. std::vector<**int**>Path = path(killer.rect.x() / 50, killer.rect.y() / 50, player.rect.x() / 50, player.rect.y() / 50);
5. **if** (Path.size() > 0)
6. {
7. **int** x = Path[0] % 12; **int** y = Path[0] / 12;
8. **if** (x == killer.rect.x() / 50)
9. {
10. **if** (y > killer.rect.y() / 50)killer.Faceto(Face::down0);
11. **else** killer.Faceto(Face::up0);
12. }
13. **else**
14. {
15. **if** (x > killer.rect.x() / 50)killer.Faceto(Face::right0);
16. **else** killer.Faceto(Face::left0);
17. }
18. killer.move();
19. }
20. **else**//距离等于零了还没被打死，，，，那就随便往哪走一步吧
21. {
22. **static** **int** d;
23. srand((unsigned)time(NULL));
24. //随机方向移动
25. d = rand() % 4;
26. **if** (d == 0)
27. {
28. killer.Faceto(Face::left0);
29. }
30. **else** **if** (d == 1)
31. {
32. killer.Faceto(Face::up0);
33. }
34. **else** **if** (d == 2)
35. {
36. killer.Faceto(Face::right0);
37. }
38. **else** **if** (d == 3)
39. {
40. killer.Faceto(Face::down0);
41. }
42. killer.move();

45. }
46. }
48. **void** Widget::assault()
49. {
50. **for** (**int** i = 1;i<enemies.size();i++)//如果只剩一个对方坦克，对方坦克是去打我的
51. {
52. std::vector<**int**>Path = path(enemies[i].rect.x() / 50, enemies[i].rect.y() / 50, campRect.x() / 50, campRect.y() / 50);
54. **if** (Path.size() > 0)
55. {
56. //qDebug() << Path[i] / 12 << " " << Path[i] % 12;
57. **int** x = Path[0] % 12; **int** y = Path[0] / 12;
58. **if** (x == enemies[i].rect.x() / 50)
59. {
60. **if** (y > enemies[i].rect.y() / 50)enemies[i].Faceto(Face::down0);
61. **else** enemies[i].Faceto(Face::up0);
62. }
63. **else**
64. {
65. **if** (x > enemies[i].rect.x() / 50)enemies[i].Faceto(Face::right0);
66. **else** enemies[i].Faceto(Face::left0);
67. }
68. enemies[i].move();
69. }
70. **else**//距离等于零了还没被输，，，，那就随便往哪走一步吧
71. {
72. **static** **int** d;
73. srand((unsigned)time(NULL));
74. //随机方向移动
75. d = rand() % 4;
76. **if** (d == 0)
77. {
78. enemies[i].Faceto(Face::left0);
79. }
80. **else** **if** (d == 1)
81. {
82. enemies[i].Faceto(Face::up0);
83. }
84. **else** **if** (d == 2)
85. {
86. enemies[i].Faceto(Face::right0);
87. }
88. **else** **if** (d == 3)
89. {
90. enemies[i].Faceto(Face::down0);
91. }
92. enemies[i].move();

95. }
96. }
97. }

由此，可确定敌方坦克的移动方向。

确定了方向变量之后，利用bullet和tank类中的move函数便可令子弹与坦克移动。

**（7）碰撞检测与子弹或坦克的移动限制：**

①碰撞检测：

主要是用来检测子弹和坦克、大本营之间是否发生碰撞用以刷新各对象的执行状态。

Qt的QRect类提供了现成的碰撞检测函数intersects下用来判断两个矩形框是否相交。利用该函数，我们可对子弹与子弹、敌方子弹与玩家坦克、玩家子弹与敌方坦克进行碰撞检测。

对于子弹与子弹：一旦碰撞检测检测到两者相撞，则改变两者的飞行状态，一旦检测到子弹与堡垒相撞，则游戏结束

1. **if** (player.bullet.bullrect.intersects(enemy.bullet.bullrect))
2. {
3. player.bullet.setstate(**false**);
4. enemy.bullet.setstate(**false**);
5. **break**;
6. }
7. **else** **if** (campRect.intersects(enemy.bullet.bullrect) || campRect.intersects(player.bullet.bullrect))
8. {
10. camp.load((rootdir + "camp1.gif").c\_str());
11. camp.scaled(50, 50, Qt::AspectRatioMode::IgnoreAspectRatio, Qt::TransformationMode::SmoothTransformation);
12. update();
13. Gameover();
14. }

对于子弹与坦克，一旦检测到玩家子弹与坦克相撞，则敌方坦克数减一，为零游戏结束，一旦检测到敌方子弹与玩家坦克相撞，则游戏结束

1. **for** (auto enemy = enemies.begin(); enemy != enemies.end(); enemy++)
2. {
3. **if** (player.bullet.bullrect.intersects(enemy->rect))
4. {
5. enemies.erase(enemy);
6. enemynum--;
7. **if** (enemynum == 0)
8. {
9. **return**;
10. }
11. player.bullet.setstate(**false**);
12. **break**;
13. }
14. }
15. //玩家和敌方子弹碰撞检测
16. **for** (auto& enemy : enemies)
17. {
18. **if** (player.rect.intersects(enemy.bullet.bullrect))
19. {
20. enemy.bullet.setstate(**false**);
22. Gameover();
24. **break**;
25. }
26. }

②移动限制ifreach函数：

判断按照给定方向，物体能否继续向前移动，以及遇到障碍物的应对措施，下给出坦克类的ifreach函数

1. **bool** Tank::ifreach(**int** x, **int** y, Face dir)
2. {
3. x /= 50;
4. y /= 50;
6. //判断是否越界
7. **if** (x < 0 || x>11 || y < 0 || y>11)
8. {
9. **return** **false**;
10. }
11. //    判断是否有障碍物
12. **else** **if** (map[y][x] == '3'||map[y][x]=='2'||map[y][x]=='4')
13. {
14. **return** **false**;
15. }
16. **else**
17. {
18. **return** **true**;
19. }
20. }

（8）定时器的使用：

设置6个定时器，与widget函数中的六个槽函数绑定，定时对坦克、子弹移动以及窗口进行更新。

1. timer1 = **new** QTimer(**this**);
2. timer2 = **new** QTimer(**this**);
3. timer3 = **new** QTimer(**this**);
4. timer4 = **new** QTimer(**this**);
5. timer5 = **new** QTimer(**this**);
6. timer6 = **new** QTimer(**this**);
7. timer1->start(120);
8. timer2->start(600);
9. timer3->start(600);
10. timer4->start(500);
11. timer5->start(120);
12. timer6->start(30);
13. connect(timer1, &QTimer::timeout, **this**, &Widget::operation);
14. connect(timer2, &QTimer::timeout, **this**, &Widget::killer);
15. connect(timer3, &QTimer::timeout, **this**, &Widget::assault);
16. connect(timer4, &QTimer::timeout, **this**, &Widget::Enemyshot);
17. connect(timer5, &QTimer::timeout, **this**, &Widget::Bulletmove);//玩家不控制子弹移动，子弹自动发射
18. connect(timer6, &QTimer::timeout, **this**, &Widget::repaint);

最后在widget的析构函数中delete六个定时器

**三、遇到的问题**

在对二维数组map的初始化中，对如下map数组

2 2 2 2 2 2 2 2 2 2 2 2

2 6 1 1 1 6 1 1 1 1 6 2

2 1 2 2 2 1 2 2 2 2 1 2

2 2 2 2 2 1 2 2 2 2 2 2

2 2 2 2 1 6 1 2 2 2 2 2

2 2 2 3 3 1 3 3 2 2 2 2

2 2 2 2 3 1 3 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 3 3 3 2 2 2 2 2

2 2 2 2 2 5 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 0 2 2 2 2 2 2

竖方向为x，横方向为y；然而对于窗口，横方向为x，竖方向为y,

这对于path函数确定路径、eventpaint函数绘制地图等情况下，需要将横纵坐标反转。