



坦克大战项目 技术分析

主讲人：高洪涛
2018-03-29

- 1 地图设计
- 2 坦克运动
- 3 碰撞检测:
- 4 两个按键同时控制
- 5 自动寻路A*算法

地图设计 1.1 如何把实际地图抽象化成数据？

采用二维数组表示地图。地图大小20x20格，每个元素占用一格，每格大小30x30像素点，总共地图大小600x600像素点。

```
int map[20][20];
```

元素定义：

0（空白）、1（水）、
2（树林）、3（砖块）
4（铁块）、5（家）



1.2 如何表示多关卡的地图？

一个二维数组表示一关地图，那么多个二维数组就表示多关地图，因此定义成3维数组就行了。

// 定义3关地图

```
public int[][][] map =  
    new int[LEVEL][ROW][COL];
```

1.3 如何显示地图？

采用二重循环，依次读取map数组，分别绘制每一个元素。

在游戏面板的paintComponent中，应该先绘制绘制地图，然后依次绘制坦克、子弹、道具等对象。

1.4 如何摧毁地图？

设计成2部分地图数据：

(1) 不可变的地图数据，就是每关地图数据。这部分数据是永不会改变的，只能读取。

(2) 可变的地图数据，定义游戏运行地图

```
public int[][] gamemap = new int[ROW][COL];
```

然后把当前关卡数据拷贝到gamemap中。

```
for (int i = 0; i < ROW; i++) {  
    for (int j = 0; j < COL; j++) {  
        gamemap[i][j] = map[newLevel][i][j];  
    }  
}
```

(3) 当子弹摧毁地图元素时，改变游戏运行地图gamemap中元素。

(4) 当重新开始时，只要重新把当前关卡数据拷贝到gamemap中，这样游戏界面就恢复过来了。

(5) 下一关时，重新设定gamemap数据即可。

坦克运动的步进值该如何选取？

坦克运动步幅决定了速度，因为地图上最小一格是 30×30 ，坦克的步进值必须能被30整除，这样才能使得运动坐标刚好是30的整数倍。

如果不能整除，就会导致坦克坐标不是30的整数倍，坦克无法穿过一些小格子缺口，界面上表现是本方坦克总也对不准缺口。

30x30格子可选取的步进值是 1/2/3/5/6/10/15。

碰撞检测分成3种，

第一是检测是否与地图碰撞，

第二是检测是否与坦克碰撞，

第三是检测是否与敌方子弹碰撞。

地图中树林元素是可穿过的，因此不会碰撞。其它元素都不能穿越，必须进行识别检测。

一般方法：

(1) 把当前坐标+前进方向的步进值得到下一步的新坐标值。

(2) 把新坐标值转换成在地图数组上的行列数，

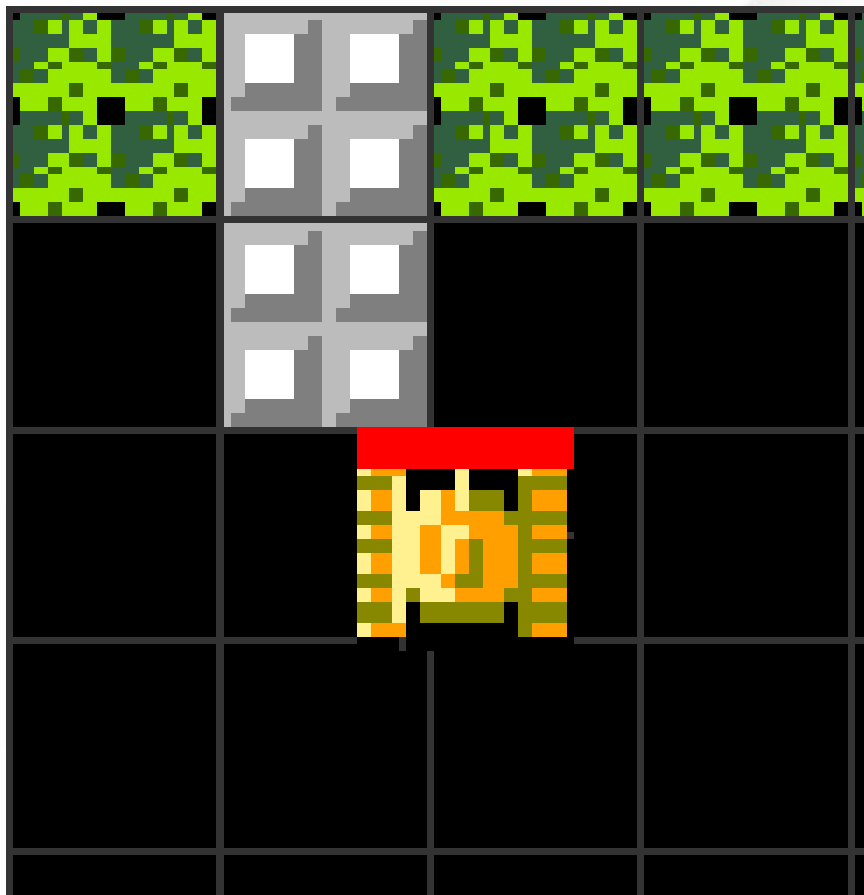
(3) 判断这个行列数对应的地图元素是否是可穿越的类型。

如果可穿越，就修改坦克坐标为新坐标值。

如果不可穿越，就随机生成新的运动方向，但是坦克坐标值保持不变。

上述方法中会出现各种异常情况，原因是坦克在地图上运动是按照像素点前进的，会出现跨越2个格子的现象，这样单纯靠一个坐标点检查就不行了，必须检查运动方向上的两个顶点。

3.1 与地图碰撞



坦克在地图上运动是按照像素点前进的，会出现跨越2个格子的现象，因此不能用简单的判断每个格子上是有坦克的方法确定碰撞。采用坦克的矩形区域是否相交来计算。

```
// 获得坦克对象自己的矩形
```

```
Rectangle thisRec= new Rectangle(newX, newY,  
Data.TANK_SIZE, Data.TANK_SIZE);
```

```
// 从链表中取出其他坦克的矩形
```

```
Rectangle tankRec = new Rectangle(tank.x,  
tank.y, Data.TANK_SIZE, Data.TANK_SIZE);
```

```
// 比较两个矩形是否有交集
```

```
if (thisRec.intersects(tankRec))
```

```
    flag = true;// 不可穿越
```

```
    return flag;
```

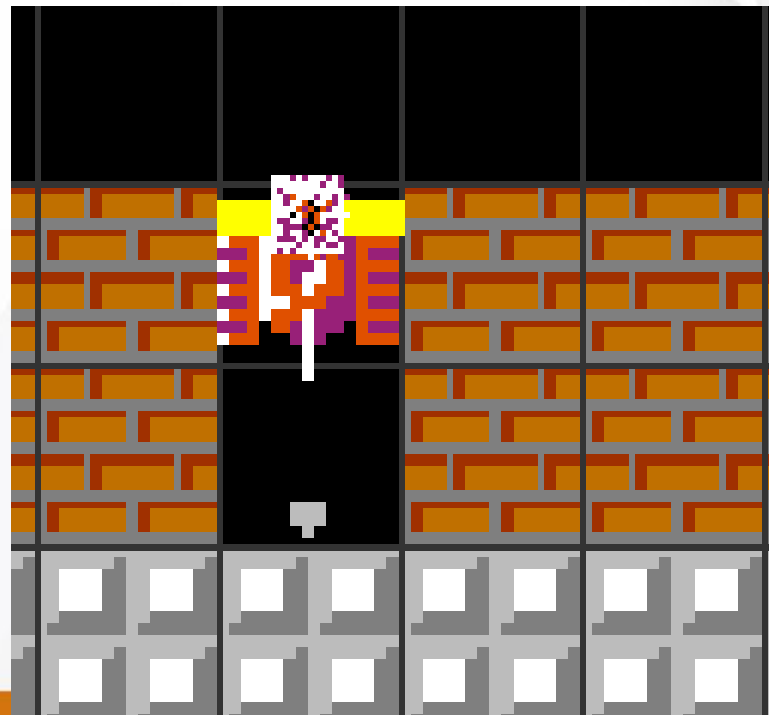
```
}
```

3.3 与子弹碰撞

坦克与子弹碰撞不影响运动，因此在坦克前进中不检测子弹碰撞。

在子弹的运动过程中会被坦克阻挡，因此应该在子弹的检测过程中进行判断。

原理方法和坦克碰撞检测相同，只需要循环判断敌方坦克的外包矩形区域是否包含子弹的中心点坐标 (centerx, centery)。



3. 4坦克运动碰撞检测流程

本方坦克运动过程中必须依次检测这些碰撞事件：

`intersectionMap()` //检测与地图的碰撞，如果有阻碍就返回true

`intersectionTank()` //检测与坦克的碰撞，如果有阻碍就返回true

`intersectionProperty()` //检测与道具的碰撞，如果有阻碍就返回道具对象

如果有碰撞地图或坦克就不更新当前坐标。

如果碰到道具就吃掉道具`eatProperty(p)`，然后更改坐标。

如果没有碰撞就更改当前坐标。

敌方坦克的碰撞检测流程

`intersectionMap()` //检测与地图的碰撞，如果有阻碍就返回true

`intersectionTank()` //检测与坦克的碰撞，如果有阻碍就返回true

如果有碰撞就随机改变方向`changeDirection(1)`;

如果没碰撞就更新坐标。

4 两个按键同时控制

一般控制坦克运动是靠键盘监听器，最普通的方法是在KeyListener监听器中接收按键事件：

```
public void keyTyped(KeyEvent e) {  
    // TODO Auto-generated method stub  
    char ch = e.getKeyChar();  
    System.out.println("ch=" + ch);  
    switch (ch) {  
        case 'a':  
            player1.x -= 5;  
            if (player1.x < 0)  
                player1.x = 0;  
            break;
```

这种方法的原理是按一下走一下，持续按下按键后利用操作系统自动触发多次按键事件进行移动。

弊端1：：自动重复触发需要一段时间延时。这个延时时间可以在操作系统中设定，但是对我们操作有影响。

弊端2：按下方向键a时又按下开火键k，这时检测到的按键是k，造成一发射子弹坦克就停止运动了。

解决方法是采用keyPressed、keyReleased检测多个按键被按下弹起事件。具体做法如下：

(1) 本方坦克增加属性运动状态runState:

```
public char direction; // 上 -U 下 -D 左-L 右-R  
public int runState; // 运动状态, 1: 运动, 0: 停止  
public static final int STATE_MOVING = 1; // 运动中的坦克  
public static final int STATE_STOP = 0; // 停止不动的坦克
```

(2) 当按下方向键asdw时设置为运动态，弹起时设置为停止态：

```
int key = e.getKeyCode();  
switch (key) {  
case KeyEvent.VK_W: // 向上  
    // System.out.println(" press up");  
    game.gamePanel.heroTank.setDirection('U');  
    game.gamePanel.heroTank.runState = Tank.STATE_MOVING;  
    break;
```

(3) 使用定时器驱动坦克运动。 坦克Move函数中根据自身的运动或停止状态决定。

```
public ActionListener taskTankMove = new  
ActionListener() {  
    public void actionPerformed(ActionEvent evt) {  
        for (int i = 0; i < gc.scene.tankList.size();  
i++) {  
            Tank t = gc.scene.tankList.get(i);  
            //System.out.println(t+ "t.type="+t.type);  
            t.move();  
        }  
  
        gc.game.gamePanel.repaint();  
    }  
};
```

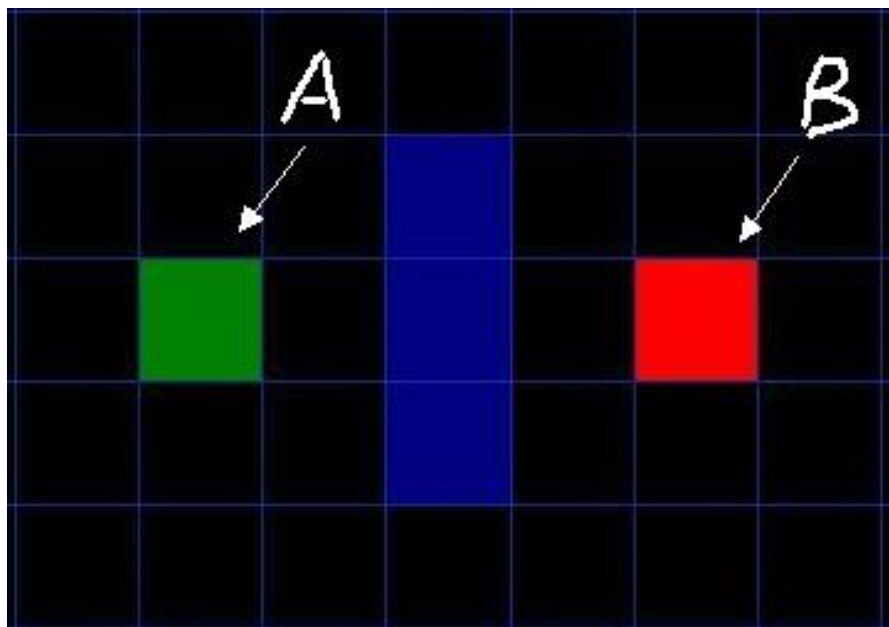
一般游戏中鼠标在地图中点击后角色就会自动走到那个位置，如果中途有障碍会自动绕开，这是怎么做到的呢？

能够在坦克项目中应用呢？

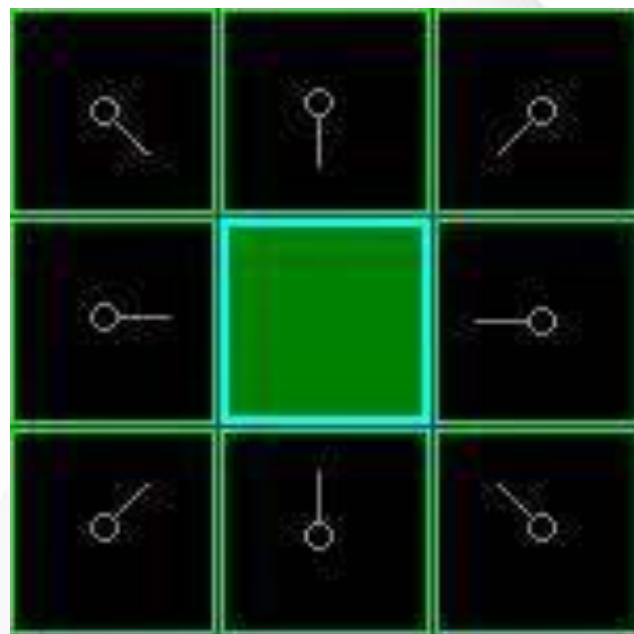
例如： 敌方坦克会自动朝着玩家家园前进，自动开炮摧毁，这就智能化了。

5.1 A*算法基础

简易地图 A起点，B终点，
蓝色障碍物

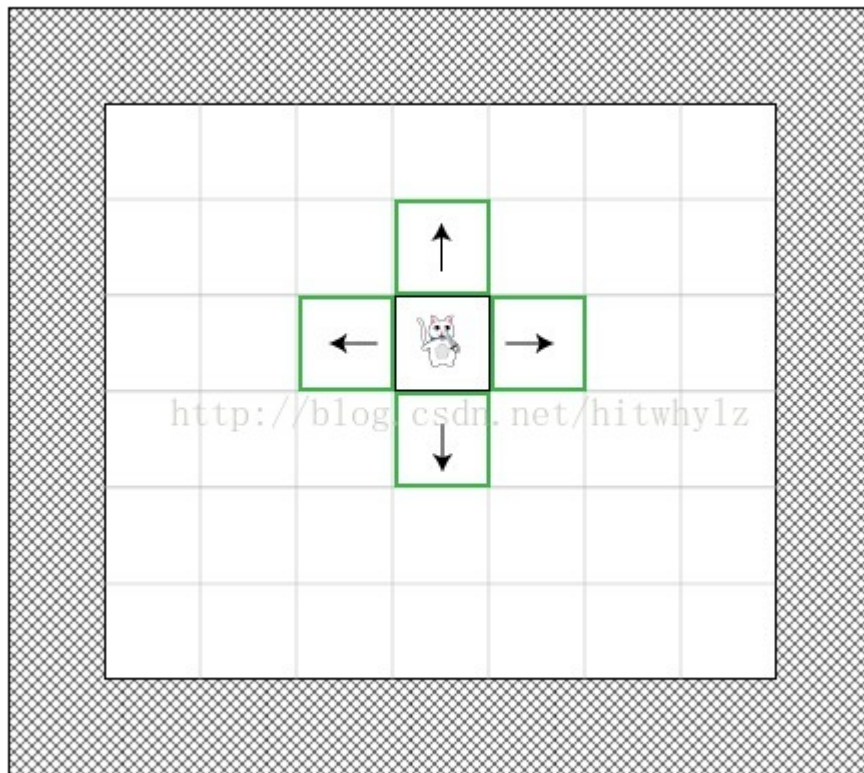


相邻方块 取上下左右4个



5.2 Open列表和Close列表

- » 它需要两个列表:
- » 一个记录下所有被考虑来寻找最短路径的方块 (称为open列表)
- » 一个记录下不会再被考虑的方块 (成为closed列表)



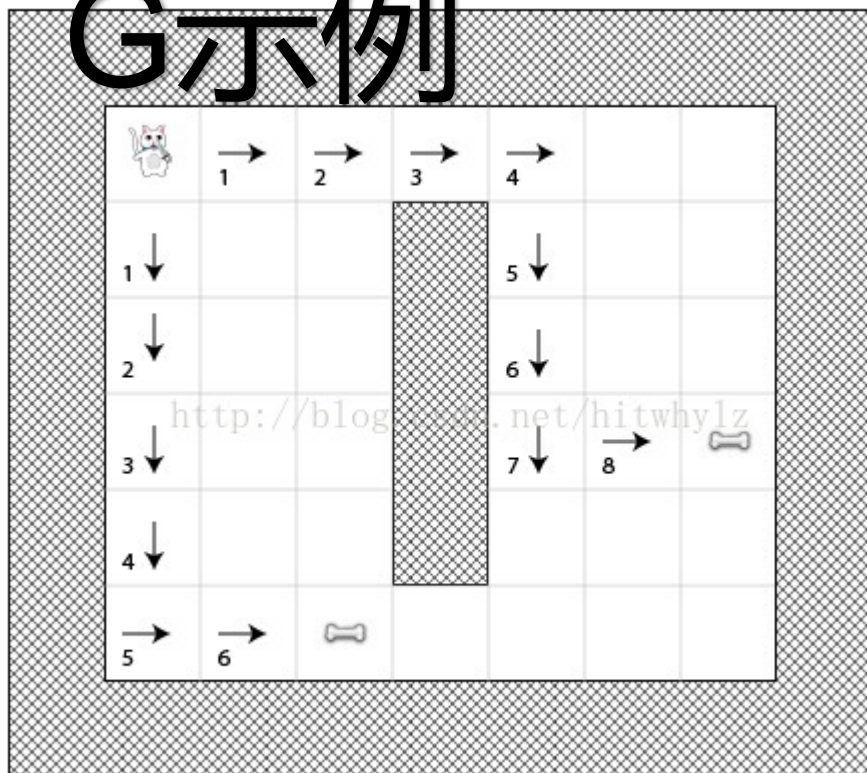
给每个方块一个 $F=G+H$

G: 是从开始点A到当前方块的移动量(可沿斜方向移动)

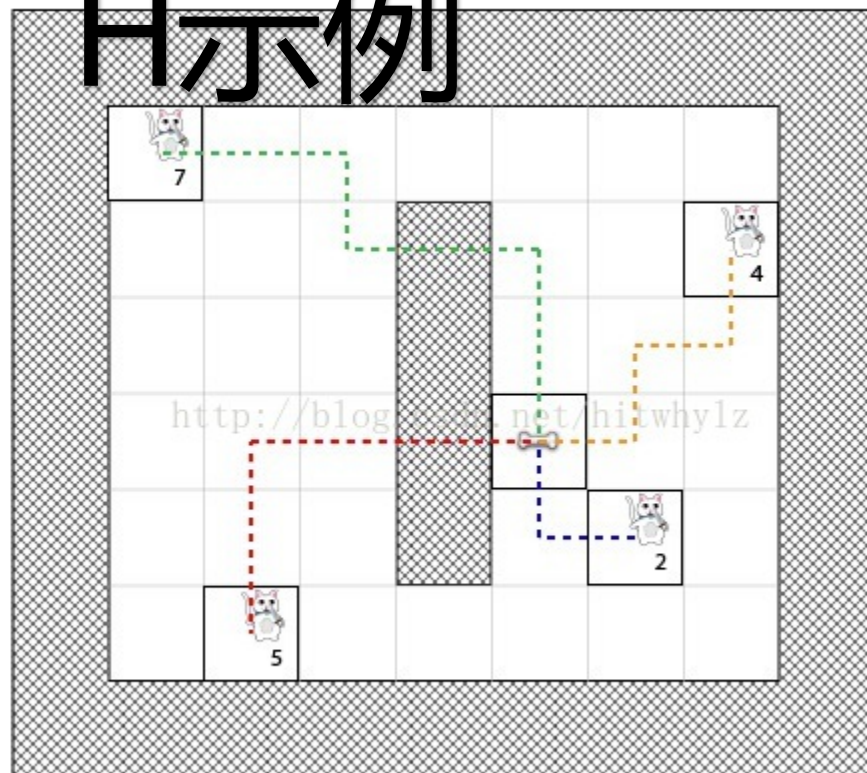
H: 是从当前方块到目标点的移动量估算值 (有多种估值方法)

曼哈顿距离、或者“城市街区距离”

G示例



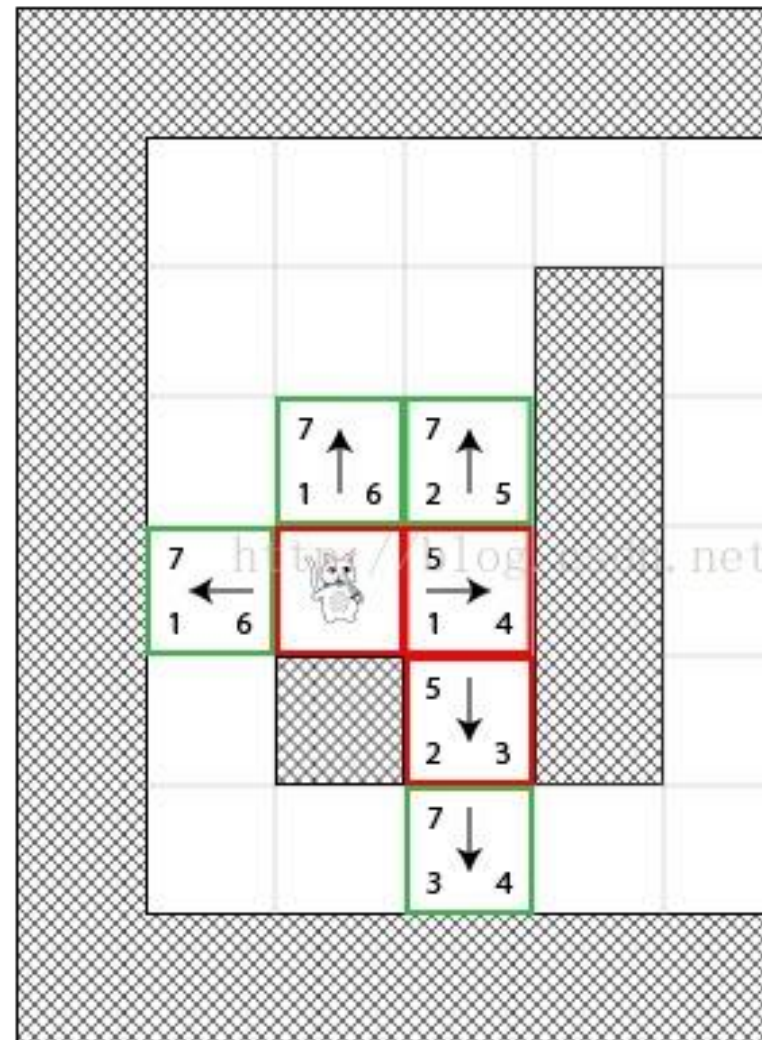
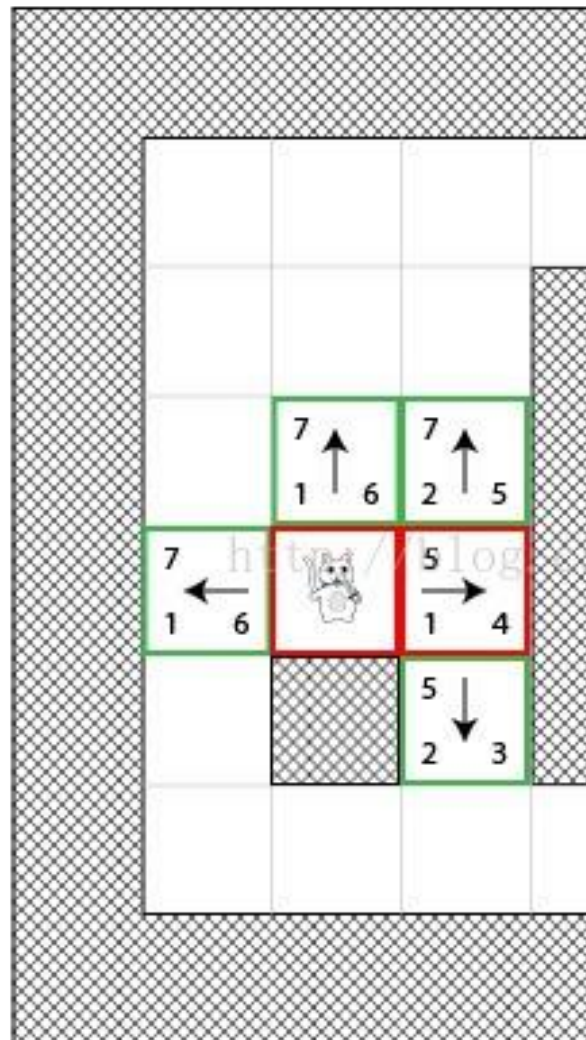
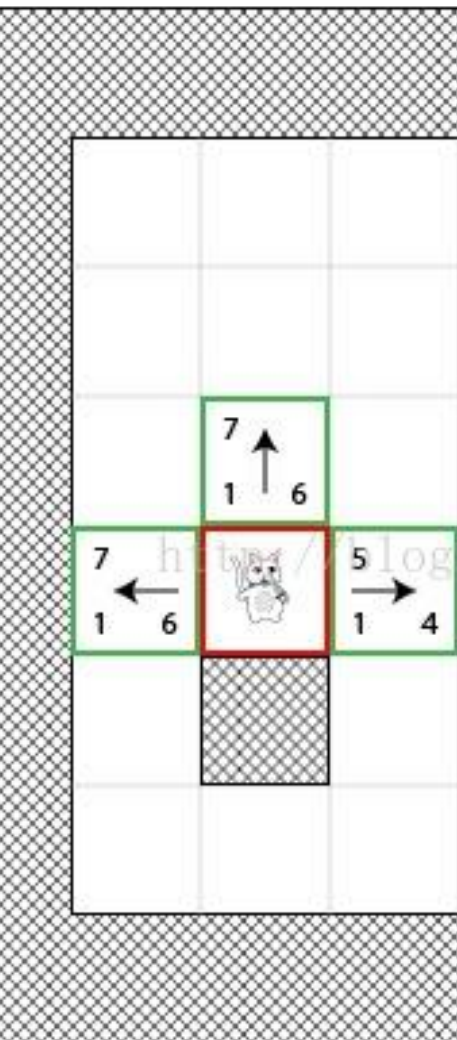
H示例




1. 从起点A开始, 把它作为待处理的方格存入一个open列表, open列表就是一个等待检查方格的列表.
2. 寻找起点A周围可以到达的方格, **判断合格才将它们放入"开启列表"**, 并设置它们的"父方格"为A.
3. 从open列表中删除起点 A, 并将起点 A 加入close列表, close列表中存放的都是不需要再次检查的方格。
- 4 寻找open列表中F值最低的格子, 我们称它为当前格, 重新开始循环
- 5 结束条件, 当前点是终点。

对相邻方格的判断：

- 1 如果是障碍物，不管它。
- 2 如果T在closed列表中：不管它。
- 3 如果T不在open列表中：添加它然后计算出它的和值，设置T的父对象是当前块。
- 4 如果T已经在open列表中：当我们使用当前生成的路径到达那里时，检查F 值是否更小。如果是，更新它的F值和它的父对象。






坦克大战
— □ ×

24	24	24		24	24	24	24	24	24									
7	17	8	16	9	15	11	13	12	13	11	14	10	15	9	16	8		
22	22	22		22	22	22	22	22	22	22	22	22	22	22	22	22		
6	16	7	15	8	14	10	12	11	11	12	10	13	9	14	8	15	7	
20	20	20	20	20	20			22	22	22	22	22	22	22	22	22	22	
5	15	6	14	7	13	8	12	9	11		13	9	14	8	15	7	16	6
18			18			18					22	22	22	22	22	22	22	22
4	14		6	12		8	10				15	7	16	6	17	5		
16			16			16	16				22	22	22	22	22	22	22	22
3	13		5	11		7	9	8	8		16	6	17	5	18	4		
14	14	14	14	14	14	14	14	14	14		22	22	22	22	22	22	22	22
2	12	3	11	4	10	5	9	6	8	7	7		17	5	18	4	19	3
12	12										22							
1	11	2	10								18	4						
10	10									24	22							
0	10	1	9							20	4	19	3					
10										24	22		22	22	22	22	22	22
1	9									21	3	20	2	21	1	22	0	
12										24	24							
2	10									21	3	22	2					

☒ 步进方式
 ☐ 完全计算

下一步

开始搜索

重新开始

显示搜索路径

显示Open列表

显示Close列表

5.6 项目中应用示例

