《软件工程与技术》团队作业

题 目 自选题—坦克大战

学 院

专 业 计算机技术

学 号 2017540192

队 员 曹融 张婷 代锦施 薛呈

1. **课程题目**

* **坦克大战（Java版本）**

实现一个java版本的坦克大战游戏

【功能提示】

1. 游戏要有图形用户界面，界面能够反映游戏所有的细节。
2. 界面中要有坦克，墙，树林，河流。
3. 界面中要有一个“家”,“家”被攻击中则输了游戏。
4. 坦克分两种，敌方和我方。
5. 墙也分为两种，可以穿透的普通墙，不可以穿透的铁墙。
6. 树和河流均只有一种，树林坦克可以通过。
7. 坦克可以发射子弹，敌我方的子弹属性一样。
8. 我方子弹可以杀死敌方坦克，敌方子弹也可以杀死我方坦克，但需要多枪子弹才可以杀死。
9. 敌方之间不可以杀死对方。
10. 子弹击中坦克要有爆炸，但击中墙不能爆炸。
11. 我方可以吃血包增加生命。
12. 游戏可以暂停，重新开始，游戏帮助等功能。具体如下图：
13. 
14. **题目分析与设计**

* **需求分析**

1. 设计一个图形用户界面，将所有游戏的元素都能在此用于界面上表现出来，界面能够接受用户的操作，具有人机交互功能。用户可以选择重新开始游戏，退出游戏，暂停游戏和查找游戏帮助。
2. 界面中包含坦克，树，河流，阻挡墙和游戏用户所要保护的“家”。
3. 坦克：坦克分为两种，敌方坦克和用户控制的坦克。用户方和敌方坦克均可以发射子弹，可以改变路径的行走，且在行走过程中遇到墙等阻碍物和游戏边界时要改变方向，而不能一直顶着障碍物不放。坦克之间不能穿越，碰撞到后自动调换方向。
4. 树:界面中要包括树林，作为遮掩物体和修饰物体，以便增加游戏的元素，使得游戏更加贴切人性化。树的数目不限，以游戏整体界面清晰美观来决定数目。
5. 河流：界面中要包括河流，河流作用与树林相同，同样河流的数目不限，以游戏界面整体美观度来决定数目。
6. 墙：墙体分为两种，普通的墙和铁墙，普通的墙在受到子弹袭击时会损坏，而铁墙可以阻碍子弹的穿越。两方的坦克均不能穿越两种墙体，但都可以摧毁普通墙体。普通墙体的数目配合其他元素合理安排，铁墙则不宜过多，不然游戏难度太低甚至因为坦克不能穿越铁墙而使得游戏无法进行下去。
7. “家”：用户方除了有一辆自己的坦克外还有一个要保护的家，家由普通墙体包围，家受到子弹攻击后游戏立即结束，用户方则输了本局游戏。
8. 子弹：子弹可以由敌方和用户方发射，且发射出去的子弹可以直线移动，直到碰到障碍物就消失，子弹可以穿越树林和河流。敌方坦克受到一颗子弹攻击会爆炸从而导致死亡，死亡后坦克消失。用户方受到子弹攻击后会减少寿命，且受到四次攻击则死亡，此时如还有敌方坦克存在，则用户方输掉本次游戏。
9. 爆炸：当子弹射击到坦克身上时，要产生爆炸效果。
10. 方向：坦克和子弹都有方向，可以选择上下左右四个方向，且子弹的方向从属于坦克的方向。

* **具体设计**

以下的论述均根据上面需求分析的顺序来安排：

1. 图形用户界面

图形用户界面要用到Java课程里的第六章的知识，需要用到抽象窗口工具集即AWT和Swing来设计实现，由于游戏需要有功能按键来控制游戏的重新开始，退出，暂停，帮助等，所以在实现过程中，设计菜单项。

坦克，河流，树，家，墙等都是调用各自类里面的draw函数来画出来，最后new一个图形面板出来，把所有的元素都add进面板就可以了。界面的构造在程序中由TankClient类的构造方法来完成，即没构造一个用户类都会产生这样一个包含各种游戏元素在内的界面。由于画面是静止的，而我们游戏的画面要求是动态的，游戏中的坦克和子弹都是在不断的动，可穿透的普通墙体也在“动”，所以对于画面，必须要不断的更新，重画才能产生动态的效果。

在程序中，此部分功能也是由TankClient类里的update函数和framPaint函数共同实现，将“动”的元素进行重画，而不动的元素如菜单项就不用重画，所以这也是为什么把菜单项设计进TankClient类的构造方法中。当然，图形用户界面中也包括对各种界面元素的大小，位置和颜色等设计，细节问题这里就不赘述。

1. 界面中的元素

界面中的元素都是在构造界面的时候用界面实例对象的add方法加入进去的，且“动”的元素要不断更新重画。

1. 坦克

坦克主要由Tank类来实现。坦克的属性：速度（有X轴和Y轴速度之分），坦克大小，坦克所在新的位置（X坐标和Y坐标），坦克是否还活着，行动方向等。所有的这些属性都有一个初始化值，从而在游戏一开始就可以运行。

在设计过程中，坦克出现的位置由坐标而定。用户方之后的位置由键盘监听方向，按指定方向以恒定的速度前进——此速度为全局静态变量，当没有接受到键盘的控制时，则保持静止。敌方的坦克则是根据随机数来控制随机的方向和路径的，当敌方坦克撞到阻碍物时，会转回到前一步的位置，从而解决了坦克撞到阻碍物不回头的问题，这个功能由Tank类里的changToOldDir（）方法实现。

坦克的方向和子弹发射以及游戏的重新开始都是由键盘来控制的，所以在Tank类里面必须要实现这些功能。

Tank类的keyPressed()方法用于接受键盘的按键监听，接收到相应的键盘信息后，如接收到F，则表示发射子弹，所以此时要调用Tank类里的fire()方法，fire()方法不带传递参数，因为子弹的方向总是从属于坦克的方向和位置。

由于坦克在碰到墙，界面边界和“家”等阻碍物时要改变方向，所以在Tank里面必须对于每一种阻碍物要设计一个方法来作为应对策略，当然解决方法是把下一步的位置调整到上一步的位置。

用户方Tank还可以吃红心来增加自己的生命值，得到一个红心，增加100生命值。所以在程序中坦克必须要有一个方法来判断当坦克接触到红心时，生命值增加的方法。程序中用eat()方法来实现“吃”红心并且增加生命值，当然，这其中要使得坦克的生命值不会超过自己生命的极限值200，所以判断的时候，当生命小于等于100时，直接加100生命值，但是当生命值大于100时，就只能使生命值加满到极限值200.当然，说到生命值，一定要在图形用户界面中显示出来，所以在Tank类里面一定要设计一个方法，来画出生命的增减过程，在Tank类里面是用DrawBloodbBar()来刻画。

1. 树林

树林主要是用来做修饰物体的，具有不透明性。由一个Tree类来描述，Tree有两个属性，位置和长宽。树的类里有一个自己的构造方法，当然还有一个必不可少的draw()方法。

1. 河流

河流的作用与树林类似，这里不再赘述。

1. 墙体

墙体分为普通墙和铁墙。普通墙可以被子弹损坏，而铁墙则不会。

* 普通墙

普通墙有以下几个属性：墙的固定长度和宽度，墙的位置坐标，墙体是由图片加入到图形用户界面来表示的。程序中用CommonWall类来描述普通墙，此类里有一个构造方法，用于传递参数，还有一个draw()方法，来画指定位置的墙，另外还有个getRect()方法来构造长方形实例。

* 金属墙

金属墙的参数和普通墙完全类似，只不过金属墙不能被子弹穿破，但这个属性在接下来讨论的子弹的属性里面，由于上面已经给出里普通墙的属性，所以这里将不再赘述金属墙的属性了。

1. 家：

家是由Home类来抽象的，具体的属性有:家的大小，家的位置，家的存活状态，与树林和河流一样，家还有draw（）和自己的构造方法，作用也和前面的一样，但是家还有一些新的方法，因为可以在游戏中重新开始游戏，所以要有设置方法让家重新“活”起来，当然，有时候要判断家现在的存活状态，所以就必须要isLive()和setLive()两个方法了。除此之外，还要有一个游戏结束界面的清理工作和提示工作，gameover()方法就是解决这个问题的。

1. 子弹：

子弹的属性：子弹前进的X轴和Y轴的速度——初始速度都为10，子弹的长度和宽度——初始的长宽都为10，子弹的位置，子弹的方向，子弹是否live等。由于不同方向的子弹其实是图片，所以这里要考虑不同图片的选择用对应的方向来指定，所以要建立Map键值对，用String属性的方向来指定不同的图片。如：*imgs*.put("L", *bulletImages*[0]);

类里面惯例有个构造方法，用来默认传递位置和方向，当然这里用到了同构，另外一个构造函数来取得子弹的状态和界面。darw()方法和move()方法来控制画子弹和移动子弹

接下来就是要考虑到子弹打到个=各元素上的反应，敌方的子弹打到敌方时，不会杀死自己人，所以这里只要直接return true就可以了，不用移除子弹碰到的坦克，更不用调用爆炸来显示爆炸效果。

具体代码：

**public** **boolean** hitTanks(List<Tank> tanks) {//当子弹打到坦克时

**for** (**int** i = 0; i < tanks.size(); i++) {

**if** (hitTank(tanks.get(i))) { //对每一个坦克，调用hitTank

**return** **true**;

}

}

**return** **false**;

}

子弹打到其他坦克上时用hitTank(Tank t) 方法，子弹打到墙上时用hitWall(CommonWall w)方法，打到金属墙上hitWall(MetalWall w)，打到家hitHome()，当然在这些方法里，都要作相应的操作来表示接受到子弹的后果，普通墙要移除对应的位置，所以在方法里面又要调用remove()方法来移除，如打到家上，就要吧home的生命设置为false，从而结束游戏。实现如下：

**public** **boolean** hitHome() { //当子弹打到家时

**if** (**this**.live && **this**.getRect().intersects(tc.home.getRect())) {

**this**.live = **false**;

**this**.tc.home.setLive(**false**); //当家接受一枪时就死亡

**return** **true**;

}

**return** **false**;

}

还有，当子弹射击到对方（用户方对敌方）时，当敌方射击到用户方，用户方要减少生命值，没接受一枪，就执行 t.setLife(t.getLife() - 50); // 受一粒子弹寿命减少50，接受4枪就死,总生命值200，当然方法里面还要做其他很多的判断，如生命值是否小于0，如果是的话就执行t.setLive(**false**);从而结束游戏。

1. 爆炸：

坦克的爆炸效果独立出来用一个类来描述，爆炸的属性：位置和存活状态，另外，画爆炸效果的时候要取得用户界面控制，所以要定义**private** **static** Toolkit *tk* = Toolkit.*getDefaultToolkit*();其他的方法只能的大同小异。

1. 菜单功能

菜单功能包括游戏的重新开始，暂停，退出等。这些功能的用户界面放在了TankClient的构造方法里，用到的是面板类和相应的监听功能，实现出来很简单。

* **类的介绍**

1. **Direction** 是一个枚举类 有L, U, R, D, STOP表示左，上，右， 下和静止五个状态
2. **River**  描述河流的类

**属性：**

|  |  |
| --- | --- |
| *riverWidth* | 河流的宽度 |
| riverLength | 河流的长度 |
| x | X轴坐标 |
| y | Y轴坐标 |
| TankClient tc | TankClient对象 |
| *riverImags* | 河流的图片 |
| River(int x, int y, TankClient tc) | 河流的构造方法 |
| draw(Graphics g) | 画河流的方法 |
| Rectangle getRect() | 长方形实例 |
| int getX() | 取得x坐标 |
| void setX(int x) | 设置x坐标 |
| int getY() | 取得y坐标 |
| void setY(int y) | 设置y坐标 |
| int getRiverWidth() | 取得河流的宽度 |
| int getRiverLength() | 取得河流的长度 |

1. **Home** 表示用户方的家的类

**属性：**

|  |  |
| --- | --- |
| *width=30* | 家的静态全局宽度 |
| *length=30* | 家的静态全局长度 |
| x | X轴坐标 |
| y | Y轴坐标 |
| TankClient tc | TankClient对象 |
| *homeImags* | 家的图片 |
| *boolean live* | 初始存活状态为true 表示家的存活状态 |
| Home(int x, int y, TankClient tc) | 家的构造方法 |
| void draw(Graphics g) | 画出家的方法 |
| Rectangle getRect() | 长方形实例 |
| void gameOver  (Graphics g) | 游戏结束方法，做清理工作和相应的提示 |
| boolean isLive() | 判断是否存活 |
| void setLive(boolean live) | 设置存活状态，以便从新开始游戏 |

1. **Tree** 描述树林的类

**属性：**

|  |  |
| --- | --- |
| *width=30* | 树林的静态全局宽度 |
| *length=30* | 树林的静态全局长度 |
| x | 树林在界面中X轴坐标 |
| y | 树林在界面中Y轴坐标 |
| TankClient tc | TankClient对象 |
| Toolkit *tk* | 取得界面的控制 |
| *treeImags* | 树林的图片 |
| Tree(**int** x, **int** y, TankClient tc) | 家的构造方法 |
| void draw(Graphics g) | 画出树林的方法 |

1. **Bullets** 描述子弹的类

**属性：**

|  |  |
| --- | --- |
| *speedX = 10* | 子弹的全局静态速度 x轴方向的 |
| *speedY = 10* | 子弹的全局静态速度 y轴方向的 |
| *width* = 10 | 子弹的全局静态宽度 |
| *length* = 10 | 子弹的全局静态长度 |
| Direction diretion | 子弹的方向 |
| x | X轴坐标 |
| y | Y轴坐标 |
| TankClient tc | TankClient对象 |
| *bulletImages* | 子弹的图片 |
| *boolean live* | 初始存活状态为true 表示家的存活状态 |
| Bullets(**int** x, **int** y, Direction dir) | 子弹的构造方法，带三个参数 |
| Bullets(**int** x, **int** y, **boolean** good, Direction dir, TankClient tc) | 子弹的构造方法，带五个参数的 |
| void draw(Graphics g) | 画出子弹的方法 |
| move() | 刻画子弹运动的方法 |
| Rectangle getRect() | 长方形实例 |
| void gameOver  (Graphics g) | 游戏结束方法，做清理工作和相应的提示的方法 |
| boolean isLive() | 判断是否存活 |
| **boolean** hitTanks(List<Tank> tanks)· | 子弹打到地方自己的时候的方法 |
| **boolean** hitTank(Tank t) | 子弹打到对方的方法 |
| **boolean** hitWall(CommonWall w) | 子弹打到普通墙的方法 |
| **boolean** hitWall(MetalWall w) | 子弹打到金属墙的方法 |
| **boolean** hitHome() | 子弹打到家的方法 |

1. **GetBlood** 血包描述类

**属性**：

|  |  |
| --- | --- |
| *width=36* | 血包的静态全局宽度 |
| *length=36* | 血包的静态全局长度 |
| x | X轴坐标 |
| y | Y轴坐标 |
| TankClient tc | TankClient类实例化对象 |
| Random *r* | 用于产生随机数 |
| Toolkit *tk* | tk获得用户界面 |
| *bloodImags* | 家的图片 |
| *poition* | 位置数组 |
| *boolean live* | 初始存活状态为false 表示初始化时无血包 |
| void draw(Graphics g) | 画出血包的方法 |
| **void** move() | 血包移动的方法 |
| Rectangle getRect() | 长方形实例 |
| boolean isLive() | 判断是否存活，是否有血包 |
| void setLive(boolean live) | 设置存活状态，设置血包的存在 |

1. **CommonWall** 普通墙的类

**属性：**

|  |  |
| --- | --- |
| *width=20* | 普通墙的静态全局宽度 |
| *length=20* | 普通墙的静态全局长度 |
| x | 普通墙在界面中X轴坐标 |
| y | 普通墙在界面中Y轴坐标 |
| TankClient tc | TankClient类实例化对象，因为画墙要取得界面的控制权 |
| Toolkit *tk* | tk获得用户界面 |
| *wallImags* | 普通的图片 |
| *CommonWall(****int*** *x,* ***int*** *y, TankClient tc)* | 普通墙的构造方法，用于传递墙的位置参数和TankClient |
| void draw(Graphics g) | 画出普通墙的的方法 |
| Rectangle getRect() | 长方形实例 |

1. **MetalWall** 金属墙的类

**属性：**

|  |  |
| --- | --- |
| *width=30* | 金属墙的静态全局宽度 |
| *length=30* | 金属墙的静态全局长度 |
| x | 金属墙在界面中X轴坐标 |
| y | 金属墙在界面中Y轴坐标 |
| TankClient tc | TankClient类实例化对象 |
| Toolkit *tk* | tk获得用户界面 |
| *wallImags* | 金属墙的图片 |
| *MetalWall(****int*** *x,* ***int*** *y, TankClient tc)* | 金属墙的构造方法，用于传递墙的位置参数和TankClient |
| void draw(Graphics g) | 画出金属墙的的方法 |
| Rectangle getRect() | 长方形实例 |

1. **BombTank** 坦克爆炸类

**属性：**

|  |  |
| --- | --- |
| x | 坦克爆炸效果在界面中X轴坐标 |
| y | 坦克爆炸效果在界面中Y轴坐标 |
| TankClient tc | TankClient类实例化对象 |
| Toolkit *tk* | tk获得用户界面 |
| *Image[] imgs* | 爆炸效果的图片 |
| *BombTank(****int*** *x,* ***int*** *y, TankClient tc)* | 爆炸效果的构造方法，用于传递爆炸效果的位置参数和TankClient |
| void draw(Graphics g) | 画出爆炸效果的的方法 |

1. **Tank** 描述Tank的类

**属性：**

|  |  |
| --- | --- |
| *speedX = 6* | 坦克的全局静态速度 x轴方向的 |
| *speedY = 6* | 坦克的全局静态速度 y轴方向的 |
| *width* = 35 | 坦克的全局静态宽度 |
| *length* = 35 | 坦克的全局静态长度 |
| Direction diretion | 坦克的状态 ，初始化为静止 |
| Direction Kdirection | 坦克的方向，初始化为向上 |
| Int x | X轴坐标 |
| Int y | Y轴坐标 |
| int oldX | 旧状态的X轴坐标 |
| int oldY | 旧状态的Y轴坐标 |
| boolean live | 坦克是否存活，初始化为活着的 |
| TankClient tc | TankClient对象 |
| int life | 初始化生命值为200 |
| Random *r* | 随机数，用于控制坦克的方向 |
| int step | 路径 |
| Image[] *tankImags* | 各个方向坦克的图片数组集合 |
| Tank(int x, int y, boolean good) | 坦克的构造方法1 ，带三个参数 |
| Tank(int x, int y, boolean good, Direction dir, TankClient tc) | 坦克的构造方法2 ，带五个参数的 |
| void draw(Graphics g) | 画出坦克的方法 |
| move() | 描述坦克运动方向的方法 |
| void changToOldDir() | 转换到旧方向， |
| void keyPressed(KeyEvent e) | 键盘监听方法 |
| void decideDirection() | 决定移动方向方法 |
| void keyReleased(KeyEvent e) | 监听键盘释放方法 |
| Bullets fire() | 坦克开火方法 |
| Rectangle getRect() | 长方形实例的方法 |
| boolean isLive() | 判断是否存活的方法 |
| void setLive(boolean live) | 设置坦克存活方法 |
| boolean isGood() | 用于返回是否是good状态 |
| boolean collideWithWall(CommonWall w) | 坦克碰到普通墙的解决方法 |
| boolean collideWithWall(MetalWall w) | 坦克碰到金属墙的解决方法 |
| boolean collideRiver(River r) | 坦克碰到河流的解决方法 |
| boolean collideHome(Home h) | 坦克碰到家的解决方法 |
| Boolean collideWithTanks(java.util.List  <Tank> tanks) | 坦克撞到坦克的解决方法 |
| int getLife() | 获得坦克的生命状态的方法 |
| void setLife(int life) | 设置坦克的生命状态的方法 |
| boolean eat(GetBlood b) | 坦克吃红心血包的方法 |
| int getX() | 确定X的位置的get方法 |
| int getY() | 确定Y的位置的get方法 |

1. **TankClient** 坦克用户类，继承Frame，实现ActionListener接口

实例化各个类的容器：

List<River> theRiver = **new** ArrayList<River>();

List<Tank> tanks = **new** ArrayList<Tank>();

List<BombTank> bombTanks = **new** ArrayList<BombTank>();

List<Bullets> bullets = **new** ArrayList<Bullets>();

List<Tree> trees = **new** ArrayList<Tree>();

List<CommonWall> homeWall = **new** ArrayList<CommonWall>(); // 实例化对象容器

List<CommonWall> otherWall = **new** ArrayList<CommonWall>();

List<MetalWall> metalWall = **new** ArrayList<MetalWall>();

下面列表只列出主要方法和变量

**属性：**

|  |  |
| --- | --- |
| ***int*** *Fram\_width* | 游戏用户界面的全局静态变量宽度，初始化为800 |
| ***int*** *Fram\_length* | 游戏用户界面的全局静态变量长度，初始化为600 |
| ***void*** *update(Graphics g)* | 图形用户界面的更新方法 |
| ***void*** *framPaint(Graphics g)* | 画图形用户界面的方法，需要重画的界面元素都在这个方法里面 |
| **void** actionPerformed(ActionEvent e) | 监听菜单项的方法 |
| **static** **void** main(String[] args) | 主函数，启动整个程序 |

用户类主要是综合调用各个类来协作完成完成游戏的设计。

* **程序流程逻辑图**

下图是程序运行的逻辑流程图，只是表示逻辑，并没有具体到每一个细节：

**流程图：**

****

* **开发环境：**Myeclipse

1. **测试分析**

* **阐述扩展包**

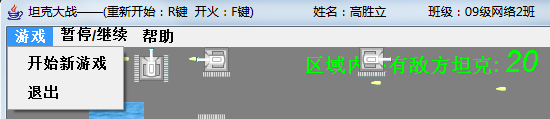
本程序没有用到SUM的扩展包，也未涉及到数据库的建立

* **程序效果图测试**

1. **菜单项测试**

菜单项可以选择开始新游戏，退出，暂停，继续，帮助功能，对于开始新游戏，退出两个功能，要给出提示，用户是否真的想继续操作！

1. 菜单效果图：



1. 开始新游戏功能测试：



当点击确认后开始新游戏，当点击取消后恢复到原来的页面。这里不再附图给出点击确定和取消后的效果。

1. 暂停功能测试：



1. 帮助模块测试图：



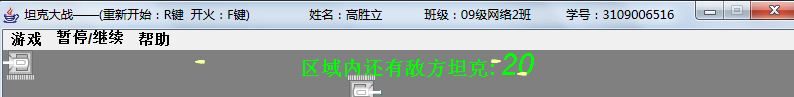
1. 退出功能测试：



点击确认后将退出游戏，点击取消后不会退出游戏。

1. **游戏信息测试：**

游戏标题栏给出游戏名称和制作人相关信息

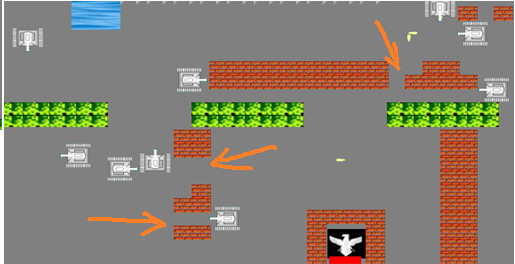


1. **子弹射击到对方坦克后爆炸效果测试：**

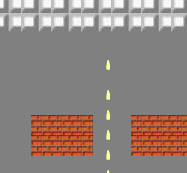
****

图中的红色类似爆炸状的图像是用户方击中地方坦克后的爆炸效果显示

1. **子弹撞击到普通墙后，会使得墙消失一块的测试：**

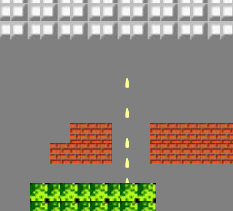
****

图中画红箭头的地方时受到子弹攻击后的效果。墙体被破坏。



1. **金属墙受到子弹攻击的测试**

如图为子弹攻击金属墙体前：

如右图为子弹攻击金属墙后的效果图**：**

**综上两个可以看出，金属墙都到子弹的攻击后不会被损毁，但是当在把程序中子弹碰撞金属墙的语句改变后，会损毁金属墙！如下：**

原来的代码段： public boolean hitWall(MetalWall w) { //子弹打到金属墙上

if (this.live && this.getRect().intersects(w.getRect())) {

this.live = false;

return true;

}

return false;

}

修改后的代码段：public boolean hitWall(MetalWall w) { //子弹打到金属墙上

if (this.live && this.getRect().intersects(w.getRect())) {

this.live = false;

**this.tc.metalWall.remove(w); //子弹可以穿越金属墙了**

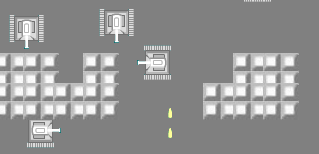
return true;

}

return false;

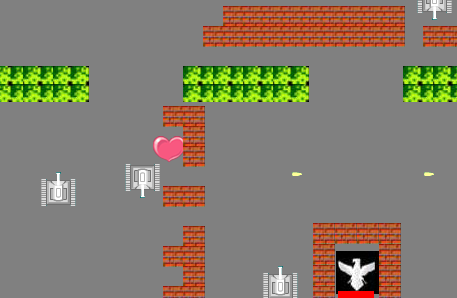
}

**修改后的效果：**从下图可以看出，修改后金属墙也可以被摧毁！

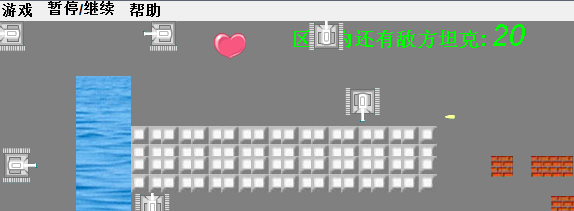
****

1. **血包效果图：**血包会随机的移动，以下是血包所在的两次不同位置：

所在位置1：

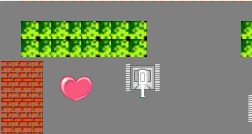
****

所在位置2**：**



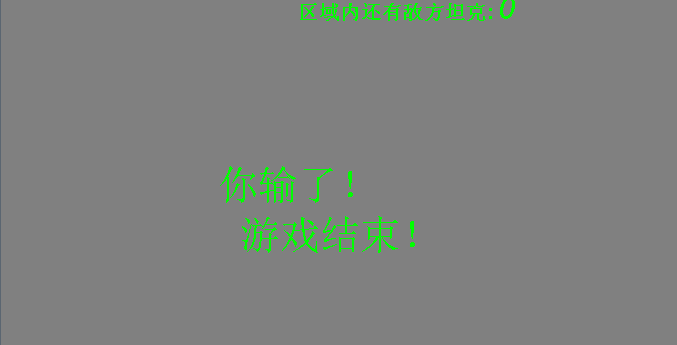
1. **坦克吃血包增加生命值测试；**

没有吃血包前**：**可以看出血包在用户方附近，当用户方去接近血包时，就可以增加生命值。

****

吃完血包后加满生命值：

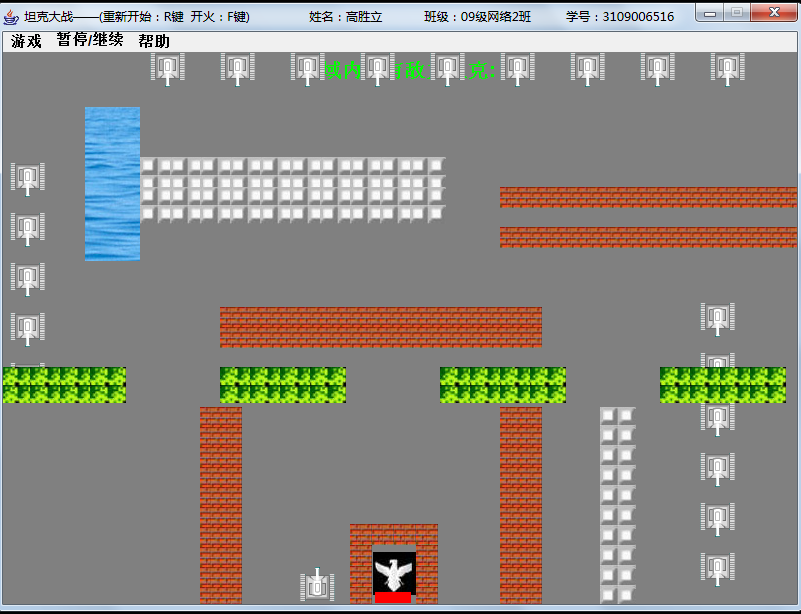
1. **当家被攻击后结束游戏：**

****

游戏结束是调用gameover()方法，在方法里面会清除所有的页面元素。

1. **快捷键的测试：**

R键为重置游戏：如下为重新开始后的页面：



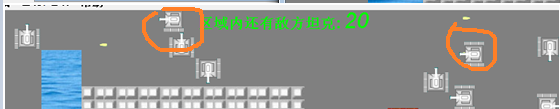
提示：由于上下左右和开火等快捷方式键都是动态的效果，所以这里不方便给出，用户可以自行测试！

1. **坦克撞击边缘的效果：**

撞击前：

****

撞击后：

****

提示：这里只给出撞击边缘的效果，其他坦克撞击坦克，家，普通墙，金属墙等效果用户可以自行测试！

1. **用户赢得游戏后的效果测试：**

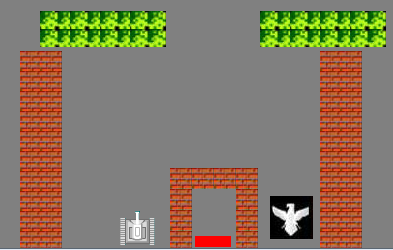
****

注释：由于坦克大战的画面都是动态的，所以不可能全部截图来测试每一个功能，如有需要，用户可以自行测试。

**下面改变程序中的一些典型的数据，看修改后的效果：**

* **典型数据测试：**

1. 家的位置测试：由于程序中的界面位置是严格嵌套设计的，即家是刚好落在它外面的一层保护墙能，当修改家原来的位置参数Home home = **new** Home(373, 545, **this**);后，会走出达不到预期的效果：如下图：



图示为把373改为了450后的效果！

1. 坦克全局静态速度的改变：

坦克的全局速度原来是x轴方向为6，y轴方向为6，修改后，坦克的前进速度会加快，当然这样一来可以加大游戏的难度，可以作为游戏的扩展功能来实现，由于坦克的速度是动态的，这里无法测试，用户可以在Tank类里修改speedX和speedY来测试。

子弹的速度的性质同坦克相同，这里不再赘述。

1. 墙体的部分测试：

改变墙体的界面布局:如改变普通墙的界面布局，是界面中没有普通墙，只要将程序中的如下代码屏蔽掉就可以了：

**for** (**int** i = 0; i < 32; i++) {

**if** (i < 16) {

otherWall.add(**new** CommonWall(220 + 20 \* i, 300, **this**)); // 普通墙布局

otherWall.add(**new** CommonWall(500 + 20 \* i, 180, **this**));

otherWall.add(**new** CommonWall(200, 400 + 20 \* i, **this**));

otherWall.add(**new** CommonWall(500, 400 + 20 \* i, **this**));

} **else** **if** (i < 32) {

otherWall.add(**new** CommonWall(220 + 20 \* (i - 16), 320, **this**));

otherWall.add(**new** CommonWall(500 + 20 \* (i - 16), 220, **this**));

otherWall.add(**new** CommonWall(220, 400 + 20 \* (i - 16), **this**));

otherWall.add(**new** CommonWall(520, 400 + 20 \* (i - 16), **this**));

}

}

屏蔽后效果如下：



从图中可以看出，没了界面中的除开家以外的普通墙体了，其他的金属墙体，树林，河流等也是一样的道理.

1. 子弹撞击其他物体如坦克的改变：当改变子弹射击坦克的方法时，如将子弹射击到坦克，不将坦克的生命结束，则游戏将永远打下去，谁都打不死谁.改变Bullets类里的hittank方法里的t.setLive(**false**);语句，只要将这个语句中的false改为true就可以了，当然对于其他的hit\*方法，也只要作相应的修改就可以了。

* **完善功能：**
* 已完善的功能：

1. 血包可以让我们知道还剩下多少生命值，但是有时候不是很直观，所以在界面中加入一行信息来实时更新用户坦克的生命值，让用户有直观的感觉。修改后的效果如下图：



1. 游戏级别的设置

游戏的级别只做了简单的级别设置，可以设置四个级别，级别1，级别2，级别3和级别4，默认进入游戏是级别1，当然可以在游戏的过程中切换级别。

级别的设置主要是把前面的Tank类和Bullets类的全局静态最终变量改成全局静态变量，利用监听来设置坦克的速度和子弹的速度来设置游戏的难度，从而形成了级别的设置。

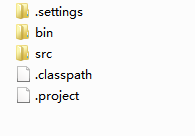
修改后的画面效果为：

* **还可以进一步完善的功能：**

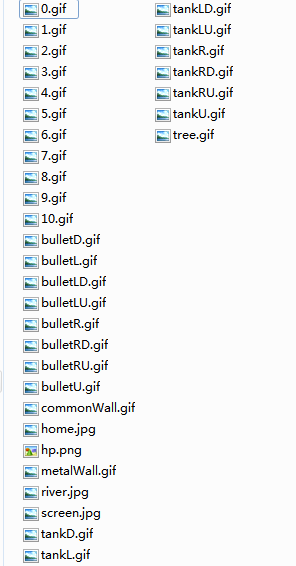
1. **可以为游戏加入声音，由于在尝试的时候失败了，所以程序中就没有来实现这些功能。**
2. **存储游戏结果。**
3. **联网功能等**
4. **附录：**

**项目名称：TankWar**

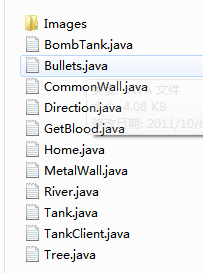
**TankWar下的包和文件：**

****

**Images下的文件：**

****

**Src包下的文件：**

****