算法分析与设计期末小程序开发文档

复仇者联盟游戏（低仿泡泡堂）开发文档

专业	金融工程
姓名	郑宇浩
学号	41621101
联系电话	18903041915
邮箱	yuba316@hotmail.com
指导教师	王征

复仇者联盟游戏（低仿泡泡堂）开发文档

目录

一、 开发目的.....3
二、 总体设计.....3
三、 详细设计.....4
3.1 地图的设计.....4
3.2 敌人的设计.....5
3.3 主角的设计.....6
3.4 炸弹的设计.....6
3.5 游戏输赢判断的设计.....7
3.6 游戏的初始化与 UI 设计.....7
四、 关键代码说明.....7
4.1 Barrier 类.....7
4.2 Enemy 类.....9
4.3 Character 类.....10
4.4 Bomb 类.....11
4.5 WinOrLose 类.....12
4.6 Framework 类.....14
五、 后期优化.....15
六、 参考资料.....16

一、 开发目的

随着《复仇者联盟 4：终局之战》（以下简称为“复联 4”）从电影院下线，影迷们都为不能够再在电影院里一睹这一部漫威影业十年来的的收官旷世巨作而感到伤心。在过去上映的一个月以来，复联 4 在全球斩获了数十亿美元的票房，迅速登上了全球累计票房排行榜的第二位，其市场之大可想而知。如此庞大的粉丝数量必定为其官方周边等复联 4 的衍生产品的销售带来了有所保障的消费者基数。因此，趁着粉丝们对复联 4 还意犹未尽，加上近日漫威影业公布要在全球重映复联 4 加长版的消息，借着这一阵轩然大波，我们开发一款与复联 4 相关的游戏必定能够吸引到无数粉丝的追捧。

同样也是在不久之前，腾讯的泡泡堂因在线游戏人数不足而被迫关闭服务器，有不少网友纷纷抱怨道自己其实是很希望能够再次玩到这款经典的街机游戏的，哪怕只是单机模式也可以。这一股复古游戏风随着网友们在微博上的转发、评论与点赞愈演愈热，故此时将游戏开发的方向转到传统怀旧的风格上，或许能够引来不少老玩家为了重温经典而下载游戏。

基于对上述两点的分析，为了能够同时吸引到漫威迷和老玩家的目光，我们将开发一款以复仇者联盟为主题的泡泡堂类型单机游戏，势必会在两个市场上都大获丰收。

二、 总体设计

为实现一款类似于泡泡堂的游戏，我们需要完成游戏中的四个主要元素的构建：地图中的障碍物、自动移动的敌人、可供玩家操控的主角以及可被主角安放的炸弹。上述四个主要元素将分别在程序的四个类中得以实现，而每个类的结尾都会附有初始化函数，供最终整合四个类的初始化类调用以生成最终的游戏界面。该初始化类还必须带有 UI 设计的功能，以美化游戏的菜单与主界面。同时，我们额外增加一个输赢判断类，用于实现敌人被杀光或者主角撞到敌人后的胜负条件判断，以及游戏结束后胜负窗口的生成。综上所述，该程序大致需要五个类来实现游戏的功能。

首先，地图中的障碍物主要分为三种：外围边界，用以限制游戏的活动范围；可摧毁的墙壁以及不可摧毁的墙壁。其中，墙壁可以随机地分布在地图的任一点，但是不能够将主角或者敌人围住，否则游戏将无法进行。同时，为了使得游戏具有耐玩性，地图中障碍物的摆设位置最好每一次都是不一样的。

地图中的敌人要能够实现随机的自动移动，且移动速度不能过快，要尽可能地平稳。敌人碰到炸弹时会死亡，且当地图上所有的敌人都被炸弹杀死后，游戏胜利。

玩家操控的主角要能够通过上下左右键以及空格键实现移动以及炸弹的安放，同时还要保证主角移动的流畅性。在碰到炸弹或者敌人的时候，主角就会死亡，则游戏结束。

炸弹爆炸后要有一定的火花持续范围与时间，在此期间内不论是敌人还是主角，在接触到火花后都会死亡。同时为了平衡游戏的难度，每次通关后炸弹的威力要上升一级，即爆炸的范围更为广阔，但根据泡泡堂的游戏规则，炸弹只能是横向或纵向爆炸，而不能朝斜方向释放火花。

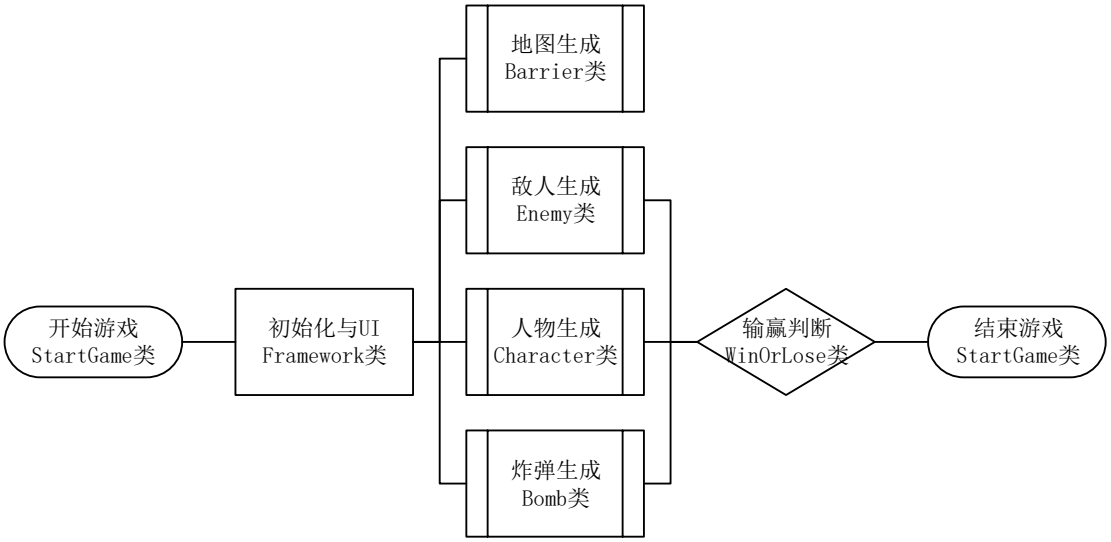
最后，根据游戏的胜负条件来判断游戏的结束与否，并跳出对应的窗口供玩家选择进入下一关或重新开始。而每次通关后或者重新开始时的游戏初始化都要使得地图上的障碍物全部重新生成，敌人和主角能够得以重生至各自的出生点，且炸弹处于未安放状态。

游戏的界面有背景图与主菜单，菜单中包括了“开始游戏”、“重新开始”和“退出游戏”

三个选项，点击开始即可进入游戏主界面开始畅玩。

各模块实现的功能以及两两之间的关系如下图 1 所示：

【图 1】总体设计流程图



三、 详细设计

3.1 地图的设计

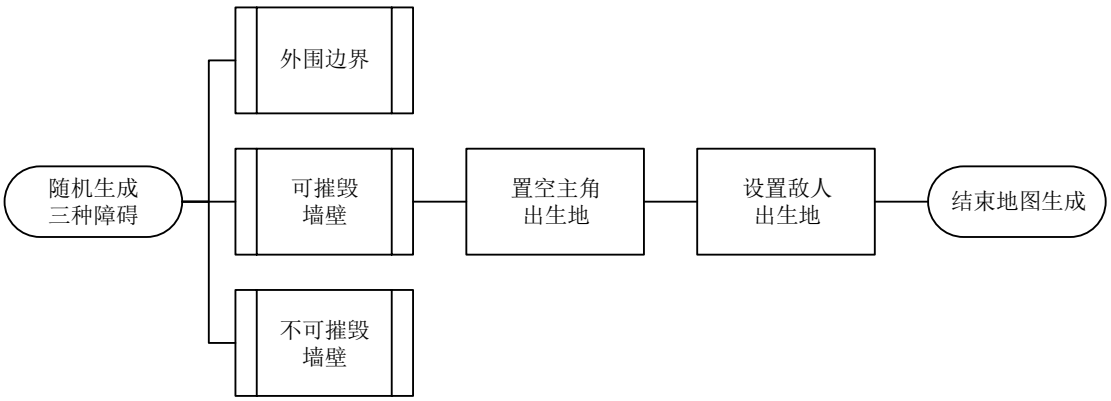
为了提高游戏的可玩性，本程序将在地图上设置两种不同的墙壁，一种为可摧毁的墙壁，即可一次就被炸弹摧毁为空地，而另一种则是不可摧毁的墙壁，需要受到两次炸弹攻击才能被摧毁。其中，实现两种障碍物间的属性区别，即可否被炸弹摧毁，需要依赖于炸弹爆炸后的火花与不同障碍物间的交互效果，故其实质上的区别将在定义炸弹的类中得以体现，在地图上呈现出来的只是外观上的不同罢了。

同时为了能够让游戏在每次开始之后都呈现不同的障碍物布局，使得游戏具有耐玩性，本程序将让两种障碍物随机地在地图上的任一点上生成。但为了保证主角不会在一开始就被障碍物给困住，我们只能强制限定了主角的出生地，并设置出生地周围至少有一个拐角的空地（有拐角主角即可安放炸弹并躲避爆炸，继续开辟道路），如下图 2 所示：

【图 2】主角出生地设置示意图

实现地图障碍物设计的类为 Barrier 类，其具体的功能结构图如下图 3 所示：

【图 3】地图障碍设计流程图

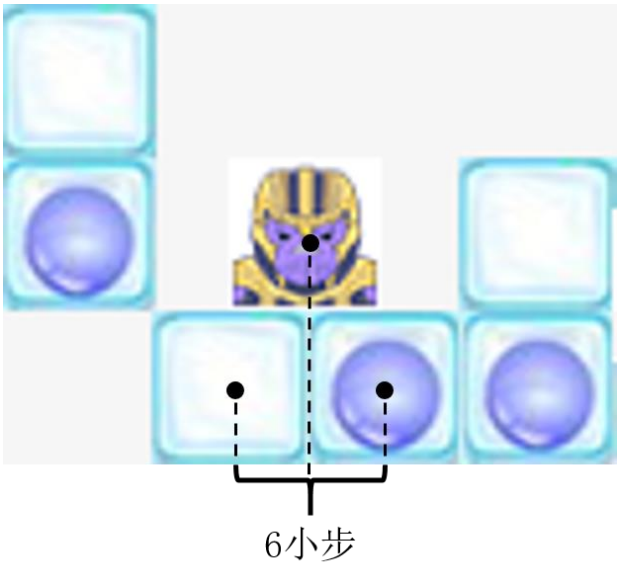


3.2 敌人的设计

同地图上的障碍物一样，敌人也是随机地在地图上的任一点上生成，但我们也要保证敌人不会一开始就被障碍物给困住，虽然这对游戏的进行影响不大，但会降低其可玩性。于是，本程序在已生成好障碍物的地图上，将敌人出生点及其周围的四个坐标点（除外围边界）都强制设为空地，保证敌人在一开始时至少可以上下左右移动。但即便如此，敌人也很有可能被困在一个狭小的空间里，这就需要玩家操控主角安放炸弹将障碍物摧毁，才能攻击到敌人。关于这一问题，我们也会在后期优化中再做详细的讨论，探讨其他更优的解决方案，防止游戏的可玩度下降。

除此之外，敌人移动的流畅性也是至关重要的，但程序能够做到的只是将敌人的图像从一个坐标点移动到另一个坐标点。于是，我们将原游戏地图的坐标点间距放大了 6 倍，而敌人的移动仍然是每次一步，这样，从一个坐标点到另一个坐标点间的移动，敌人实质上是移动了 6 小步，而每两步之间又有一个小小的停顿，加上敌人的图像足够大到可以覆盖一整个原坐标点，所以看起来就像是敌人在缓慢移动，如下图 4 所示：

【图 4】 敌人移动示意图

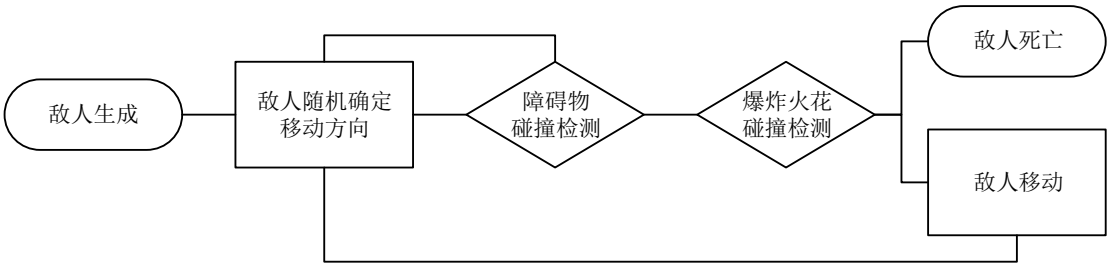


敌人在碰到障碍物后会止步不前，继续随机地更换移动方向直到可以前进为止。当敌人遇到炸弹时会被杀死，并被自动地从存储运行敌人线程的数组里删除。在敌人撞上主角时，游戏自动结束，这将会在游戏输赢判断的类中进行定义。上述所有的碰撞检测都将利用判断

两个矩形间是否相交的方法来实现，因为实质上障碍物、炸弹、主角和敌人都是一个矩形图像，则用此方法判断两两之间是否发生碰撞最为直接且简单。

实现敌人设计的类为 Enemy 类，其具体的功能结构图如下图 5 所示：

【图 5】敌人生成设计流程图

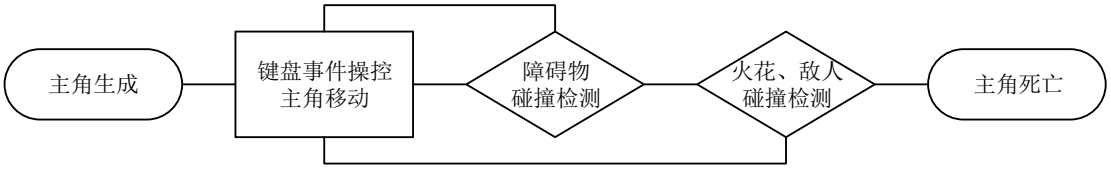


3.3 主角的设计

主角的移动与炸弹的安放将交由玩家来操控，通过添加上下左右键和空格键的键盘敲击事件可以实现这一功能。而实现主角的移动与敌人流畅移动的实现方法相类似，也是通过放大坐标间距得以近似模拟。不同的是，主角的两次移动间将不设置睡眠时间，这样才能够保证玩家操纵主角移动的流畅性。

实现主角设计的类为 Character 类，其具体的功能结构图如下图 6 所示：

【图 6】主角生成设计流程图



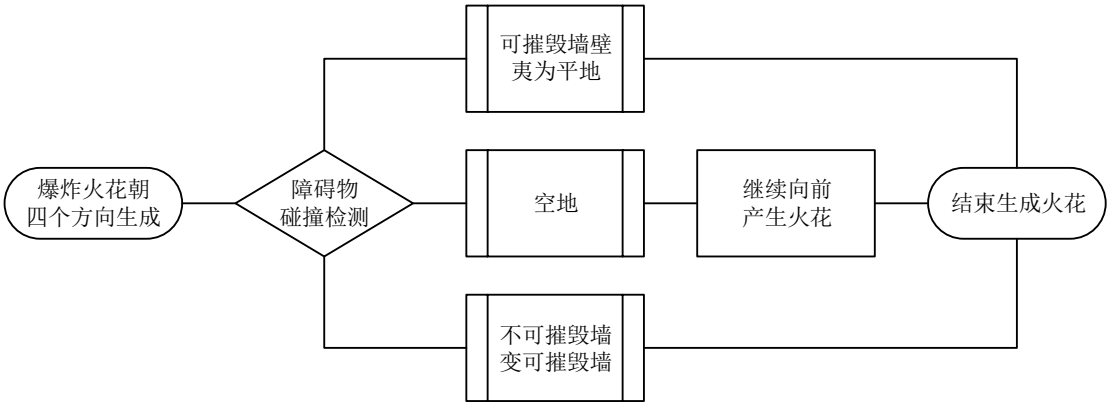
3.4 炸弹的设计

炸弹爆炸后会产生上下左右四个方向的火花特效，但当火花攻击到障碍物时，它将摧毁障碍物或者改变其属性，即从不可摧毁的墙壁变为可摧毁的墙壁，而不继续向前。这就需要在每次生成火花特效时都判断下一步坐标是否为障碍物，并对不同的障碍物做出不同的反应，同时阻止火花特效的继续向前产生。

爆炸火花的范围将由炸弹的威力来决定，威力越强，则循环向前生成火花特效的次数就越多。

实现炸弹设计的类为 Bomb 类，其具体的功能结构图如下图 7 所示：

【图 7】炸弹生成设计流程图

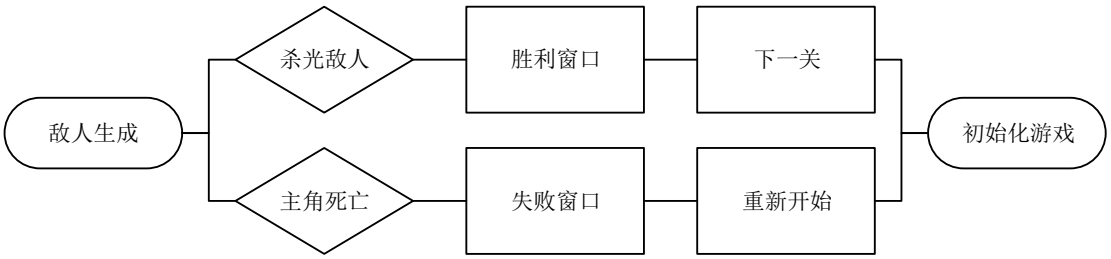


3.5 游戏输赢判断的设计

游戏的输赢判断实现了敌人的生成以及敌人是否被杀光或主角是否撞到敌人的判断。本程序将敌人的碰撞检测写在了游戏的输赢判断类里，是为了方便程序直接对存储运行敌人线程的数组进行敌人的添加与删除。当敌人数组为空时，游戏获胜，跳出胜利窗口，可进入下一关，失败则同理跳出失败窗口。

实现游戏输赢判断设计的类为 WinOrLose 类，其具体的功能结构图如下图 8 所示：

【图 8】游戏输赢判断设计流程图



3.6 游戏的初始化与 UI 设计

游戏的初始化将调用上述五个类进行障碍物、敌人、主角以及炸弹的生成，同时设计有通关后炸弹威力和敌人数目提升的功能。主菜单中含有“开始游戏”、“重新开始”和“退出游戏”三个选项，点击“开始游戏”或“重新开始”都将对游戏进行一次初始化。游戏界面中还需要有计分板来显示玩家目前的得分情况。实现游戏初始化与 UI 设计的类为 Framework 类。

四、 关键代码说明

4.1 Barrier 类

4.1.1 类的功能：

该类主要实现游戏地图的生成，具体包括三种障碍物：游戏地图的外围边界、可被炸弹摧毁的墙壁和不可摧毁的墙壁，以及敌人与炸弹爆炸后的火花特效。

4.1.2 实现思路：

游戏地图的各个坐标点利用二维数组来存储，对每一个坐标点进行从 0 到 3 的随机编号，即可把游戏地图中的每一个坐标点分为空地和三种障碍物的生成点。

之后只需要在非 0 的坐标点上生成矩形障碍物即可。至于不同障碍物的特性，即可否被炸弹摧毁，都将在 Bomb 类中得以体现，在这里将不作赘述。

在生成障碍物之后，只需将固定个数的敌人的坐标点随机地覆盖在原坐标点上即可，即把部分坐标值设为 4 以标记敌人的出生点。

4.1.2 具体代码：

利用 random 函数乘以 4 可实现 0 到 3 的随机数生成，即可对坐标点进行随机分类，具体的使用方法如下：

```
for(int i=0;i<20;i++){ // 游戏地图的所有坐标点用20X20的二维数组来存储
    for(int j=0;j<20;j++){
        if(i==0||i==19|j==0|j==19){ // 将外围边界设为1
            Barrier.place[i][j]=1;
            continue;
        }
        // 利用随机函数实现任一点4种障碍物的随机生成
        int setBarrier=(int)(4*Math.random());
        if(setBarrier==1){
            Barrier.place[i][j]=0; // 若随机数为1，也设置为0，即为空地
            continue;
        }
        // 随机数为2或3则生成对应的障碍物
        Barrier.place[i][j]=setBarrier;
    }
}
```

BarrierRect=new ArrayList<Rectangle>()声明了一个存储矩形类型[1]的ArrayList，将之后生成的障碍物矩形都用add函数添加进去，具体的使用方法如下：

```
for(int i=0;i<20;i++){
    for(int j=0;j<20;j++){
        if(Barrier.place[i][j]==1||Barrier.place[i][j]==2||Barrier.p
lace[i][j]==3){ // 如果是外围边界或者障碍物2和3，则生成障碍物矩形
            BarrierRect.add(new Rectangle(j*30, i*30, 30, 30));
        } // 7为炸弹爆炸后的火花效果，将在Bomb类中进行详细阐述
        if(Barrier.place[i][j]==7){
            firework.add(new Rectangle(j*30, i*30, 30, 30));
        }
```

```
            }
        }
```

敌人的数量将在Panel类中定义，此处利用循环体即可实现固定个数的敌人的随机生成。注意到敌人的出生地范围必须小于外围边界且不能够离主角的出生地太近，避免主角一开局就死亡。同时还要保证敌人起码可以上下左右移动一个坐标点，故强制将敌人出生地周围的障碍物设置为空地，虽然敌人也还是有可能被困住，这一点将在后期优化中再做具体阐述。具体的实现方法如下：

```
        for(int i=0;i<Panel.num_enemy;i++){
        // 保证敌人不会在边界上生成，且不会一开始就把主角给杀了
            int x=(int)(2+Math.random()*17);
            int y=(int)(2+Math.random()*17);
            Barrier.place[x][y]=4; // 将敌人设为4
            // 保证敌人可以左右移动:
            if(Barrier.place[x+1][y]!=1) { // 如果前方不为外围边界，则设为空地
                Barrier.place[x+1][y]=0;
            }
            if(Barrier.place[x-1][y]!=1) {
                Barrier.place[x-1][y]=0;
            }
            if(Barrier.place[x][y+1]!=1) {
                Barrier.place[x][y+1]=0;
            }
            if(Barrier.place[x][y-1]!=1) {
                Barrier.place[x][y-1]=0;
            }
        }
    }
```

4.2 Enemy 类

4.2.1 类的功能：

该类主要实现敌人的移动，以及与爆炸火花的碰撞检测，用于判断敌人的生死。

4.2.2 实现思路：

通过生成一个可移动的障碍物矩形来模拟敌人，为实现其移动，我们可以将原来的游戏地图坐标点放大6倍，而敌人矩形每次只移动一步，每两步之间设置一定的停留时间，就可以实现在原游戏地图上的坐标点之间的缓慢移动。

当敌人的下一步坐标点为参数0，即为空地时，则移动成功，反之移动不成功，即受到障碍物阻挡而原地不动。

若敌人落入到了炸弹爆炸后的火花范围内，则进行碰撞检测，用以判断敌人是否已被炸弹杀死。当然敌人与主角之间也会有碰撞检测，但由于敌人与主角的碰撞会直接导致游

戏的结束，故将其写在WinOrLose类中。

4.2.3 具体代码：

利用循环实现敌人从原游戏地图上的一个坐标点到另一个坐标点的6小步移动，每移动一小步都会进行一次与爆炸火花的碰撞检测用以判断敌人的生死，详见下方的getkill()函数。每两小步之间的移动都用sleep函数强制其休息一段时间，连续起来即可造成近似于缓慢移动的效果，具体实现方法如下：

```
for(int i=0;i<6;i++){ // 原坐标点被放大了6倍，而敌人每次只移动一小步
      setx(getx() - 1);
      getkill(); // 判断敌人是否被杀
      try {
          Thread.sleep(150); // 每次移动耗费150
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
```

同样利用random函数乘以4来随机生成敌人的移动方向，对每个方向的下一步坐标都判断是否为0，即是否为空地，具体实现方法如下：

```
boolean crash=true;
while(crash) {
    direction = (int)(Math.random()*4);
    if(direction==0){
        // 敌人下一步坐标，敌人为坐标点的6倍，故应该除以6得到其坐标
        int xEnemy=getx()/6-1;
        // 如果下一步为空地，则可以移动
        if(Barrier.place[xEnemy][gety()/6]==0){
            crash=false; // 不会撞
        }
     }
```

由于敌人只是用一个移动的矩形来实现，所以敌人与爆炸火花的碰撞检测就变成了两个矩形间的相交判断，可以用intersects函数实现[2]，具体的使用方法如下：

```
private boolean killing(Rectangle boom){
    return this.setEnemy(getx(),gety()).intersects(boom); // 相交为true
}
```

4.3 Character 类

4.3.1 类的功能：

该类主要实现主角的移动，以及与爆炸火花的碰撞检测，用以判断主角的生死。

4.3.2 实现思路：

与Enemy类的实现思路相类似，只不过是把敌人的随机移动改成了用键盘事件[3]操控主角的移动，依然是将地图坐标放大为原来的6倍，而每次键盘事件只移动主角一小步以实现近似的连续移动。

主角的每次移动都要进行是否撞墙的判断，如果主角无路可走，即下一步坐标不为0，则不改变其坐标值。

同样的，主角在每一步的移动之中，也要进行与爆炸火花的碰撞检测，同样是使用矩形是否相交的判断方法，在这里就不多作赘述，具体的代码实现方法与Enemy类相类似。

4.4 Bomb 类

4.4.1 类的功能：

该类主要实现炸弹爆炸后的火花摧毁各类障碍物的效果。

4.4.2 实现思路：

为炸弹设置时间，在某一段时间内，炸弹一直维持一个矩形的状态。当达到爆炸时间时，则产生横竖交错的多个爆炸火花矩形。爆炸火花持续一段时间之后就消失，此时主角才可再次安放炸弹。

爆炸火花也分为上下左右四个方向，对每一个方向都会进行是否摧毁障碍物的判断。若爆炸火花的下一步坐标为2，即可摧毁的墙壁，则将其坐标值改为0，即夷为平地；若为3，则将其坐标值改为2，即变成摧毁的墙壁。

4.4.3 具体代码：

定时炸弹的具体实现方法如下：

```
if(bombexist==true){ // 主角已经安放过炸弹
    timecount++;
    if(timecount<40){ // 计时小于40时，则显示炸弹的图片，即尚未爆炸
    boom.drawImage(bomb, 30*Bomby, 30*Bombx, 30, 30, null);
    }
    if(timecount>=40&&timecount<50){ // 炸弹爆炸后的火花持续时间为10
        if(bombset==false){
            explode(); // 爆炸火花摧毁各类障碍物
        }
    }
    if(timecount==50){ // 炸弹冷却结束，重新装填
        createbomb(); // 为主角生成炸弹
    }
}
```

爆炸火花同样也是用一连串的小矩形实现的，可以利用循环结构实现多个爆炸火花的

生成，即当爆炸火花的下一步坐标为0时，可以继续循环向下一步生成一个火花特效矩形，若遇到障碍物，则判定为false，在循环体内即失效，爆炸火花中断，具体实现方法如下：

```
        Barrier.place[getBombwidth()][getBomblength()]=7;
        for(int i=0;i<power;i++){ // 根据炸弹威力确定爆炸范围,即火花特效矩形数
            if(up){
                int xarea=getBombwidth()-1-i;
                if(xarea<0){ // 如果炸到了外围边界则无效
                    xarea=0;
                }
                // 0为空地，2为可摧毁的墙壁，3为不可摧毁的墙壁
                if(Barrier.place[xarea][getBomblength()]==0||Barrier.place[x
area][getBomblength()]==2||Barrier.place[xarea][getBomblength()]==3){
                    // 如果为空地，则填充为爆炸火花特效
                    if(Barrier.place[xarea][getBomblength()]==0){
                        Barrier.place[xarea][getBomblength()]=7;
                    }
                    // 如果为可摧毁的墙壁，则填充为空地
                    if(Barrier.place[xarea][getBomblength()]==2){
                        Barrier.place[xarea][getBomblength()]=0;
                        score=score+10;
                        Framework.score.setText("score: "+Bomb.score);
                        up=false; // 已摧毁障碍物，不可再向前产生火花特效矩形
                    }
                    // 如果为不可摧毁的墙壁，则一次炸毁变为可摧毁的墙壁
                    if(Barrier.place[xarea][getBomblength()]==3){
                        Barrier.place[xarea][getBomblength()]=2;
                        up=false;
                    }
                }
            }
```

4.5 WinOrLose 类

4.5.1 类的功能：

该类主要实现敌人的生成与游戏输赢的判断。

4.5.2 实现思路：

每一个敌人用一条线程运行[4]，用数组存储运行敌人的多条线程。

每隔一段时间就判断一次敌人的生死，将已死亡的敌人从数组中删除。

游戏胜利的条件是存储敌人的数组为空，故每隔一段时间就判断一次游戏是否仍在进行，若游戏尚未结束，则不断地绘制数组里犹存的敌人画像。

4.5.3 具体代码：

同时创建单条线程和生成敌人的具体实现方法如下：

```
                if(Barrier.place[i][j]==4){ // 地图中为4的网格即为敌人的出生地
                Enemy ene=new Enemy(i*6, j*6); // 敌人为坐标点的6倍
                Thread ene1=new Thread(ene);
                Barrier.place[i][j]=0; // 将敌人出生地设置为空地
                ene1.start();
                EnemyList.add(ene); // 新增敌人到列表中去
```

利用循环结构对数组中犹存的敌人进行生死判断，判断的方法仍然是依赖于敌人矩形与爆炸火花特效矩形的相交判断的碰撞检测，同时对未死的敌人进行画像的绘制，具体实现方法如下：

```
public void paint(Graphics enemy){
        super.paint(enemy);
        Image Enemies=getToolkit().getImage("enemy.jpg");
        for(int i=0;i<EnemyList.size();i++){
            if(EnemyList.get(i).alive==true){ // 判断敌人死活，活着则绘制画像
                enemy.drawImage(Enemies, EnemyList.get(i).gety()*5,
EnemyList.get(i).getx()*5, 30, 30, null);
            }else{
                EnemyList.remove(i); // 死亡敌人从列表中移除
            }
        }
```

同样利用循环体判断游戏的结束与否，若仍在进行，则不断地调用上述paint()函数对敌人进行画像的绘制，知道游戏结束才调用win()或lose()函数弹出胜负窗口，具体实现方法如下：

```
        while(true){
            try {
                Thread.sleep(50);
            } catch (InterruptedException e) {
                // TODO Auto-generated method stub
                e.printStackTrace();
            }
            if(gg==true){ // 如果游戏还在进行
                repaint();
                if(EnemyList.size()==0){ // 杀光敌人则胜利
                    win(); // 跳出胜利窗口，可进入下一关
                    gg=false; // 中断游戏
                }
                if(Character.death==true){ // 主角死亡则失败
```

```
                Bomb.bombexist=false; // 炸弹清空
                Bomb.timecount=0; // 计时归零
                lose(); // 跳出失败窗口
                gg=false; // 中断游戏
            }
        }
    }
```

4.6 Framework 类

4.6.1 类的功能：

该类主要实现障碍物、敌人、主角以及炸弹的生成，同时对游戏的菜单及主界面进行设计。

4.6.2 实现思路：

调用Java的抽象窗口工具包（awt）中与菜单（menu）相关的设计包[5]即可轻而易举地生成简易的菜单。

游戏的初始化需要调用Barrier类在地图上随机生成障碍物，WinOrLose类负责生成敌人，而Character类和Bomb类则初始化主角与炸弹。同时我们还需要设置每次通关后敌人数目与炸弹威力的增加。加入计分板方便玩家在游戏过程中查看自己当前的得分情况。

4.6.3 具体代码：

利用Java的抽象工具包创建菜单的具体实现方法如下：

```
public Framework(){
    setFramework();
    mb=new MenuBar();
    game=new Menu("menu");
    start=new MenuItem("start");
    again=new MenuItem("again");
    quit=new MenuItem("quit");
    start.addActionListener(this);
    again.addActionListener(this);
    quit.addActionListener(this);
    game.add(start);
    game.add(again);
    game.add(quit);
    mb.add(game);
    setMenuBar(mb);
    score=new JLabel("score: "+Bomb.score);
    score.setBounds(0, 600, 60, 20);
    power=new JLabel("power: "+Bomb.power);
```

```
        power.setBounds(540, 600, 60, 20);
        add(score);
        add(power);
        addWindowListener(new close());
    }
```

游戏初始化的具体实现方法如下：

```
    public static void creategame(){
        Character.nowy=6; // 主角出生地
        Character.nowx=6;
        Barrier.createplace(); // 初始化场地，详见Barrier类
        WinOrLose.EnemyList.clear(); // 清空敌人列表
        WinOrLose.createenemy(); // 初始化敌人，详见Enemy类
        WinOrLose.gg=true; // 还未结束游戏
        Character.death=false; // 主角还未死亡
        Bomb.score=0; // 得分清零
        Framework.score.setText("score: "+Bomb.score);
        Bomb.power=WinOrLose.power; // 炸弹等级还为2
        Framework.power.setText("power: "+Bomb.power);
    }
```

五、 后期优化

由于主角的移动是采用放大坐标后缓慢移动每一小步的方法来实现近似平稳的，故玩家操控的主角很有可能恰好停在了两个空地之间。而这一点微小的差别往往很难被玩家们的肉眼给发现，即玩家以为主角已经抵达下一个空地，可以躲避爆炸火花的威胁，但实际上主角的画像仍有一小部分残留在上一个空地上，此时炸弹爆炸即会伤及到主角的一小部分突出的画像，则判定游戏失败，造成游戏体验良好度的下降。

除此之外，我们在前文也频繁提到敌人有可能在初始地图生成时被随机生成的障碍物给困住，因为程序也仅仅是设置敌人可以上下左右移动一格而已，并不能保证其周围的障碍物不构成封闭空间。但实际上这一点对玩家的游戏体验并不会造成太大的影响，而且该问题也是难以避免的。

后期我们将对游戏进行以下五个方面的优化：

1. 通过创建与存储障碍物一样的矩形数组来存储炸弹，使得玩家能够操控主角一次性安放多个炸弹，将敌人围住，以增加游戏的趣味性；

2. 在游戏的初始界面中提供复联 4 的多角色选择功能，让玩家能够操控自己喜欢的超级英雄进行游戏；

3. 在地图上设置一些功能各异的道具，让主角拾到后可以获得不同的能力来对抗敌人；

4. 游戏地图上的障碍物也可以实现多样化，即包括一些受到炸弹攻击后呈现出不同属性的墙壁，如爆炸墙壁等等；

5. 敌人的移动将不再是简单的上下左右随机移动，而是可以通过搜索最短路径的方法

来找到主角的所在位置，并不断地像主角靠近。同时，随着关卡的提升，我们还可以设置敌人的移动速度加快等等来增加游戏的难度。

六、 参考资料

【1】https://www.jianshu.com/p/7cacc73c96c0. 简书. 轻荷. Particle system_useArrayList & Rectangle. 2015.11.13 22:33.
【2】https://stackoverflow.com/questions/8472148/java-rectangle-intersect-method. Stack overflow. rmp2150. Java Rectangle Intersect Method. 2011.12.12 9:21.
【3】https://blog.csdn.net/qq_36761831/article/details/81545050. CSDN. Hern（宋兆恒）. Java KeyEvent（键盘事件）. 2018.08.09 22:22.
【4】https://www.cnblogs.com/riskyer/p/3263032.html. 博客园. you Richer. Java 线程详解.
【5】类 java.awt.Menu 的使用.
https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/awt/class-use/Menu.html.