实验报告

姓名：杨镇源　　学号：211240045

完成进度：

项目第二阶段必做内容：已完成。
添加音乐：已完成。
机器人策略：已完成。

进一步优化：

一、对部分游戏逻辑进行优化

参考原游戏，优化了炸弹的爆炸逻辑，使之能穿过其他玩家放置的炸弹。

二、增添了游戏音效。

1. 游戏的背景音乐。因为我写的 Q 版泡泡堂本身的艺术设计即是以万圣节为灵感，所以增添的背景音乐取材自“奥拉星”万圣节活动，与主题契合。
2. 按钮点击时的音效。
3. 炸弹爆炸时的音效。
4. 游戏人物吃到道具时的音效。
5. 人物死亡时的音效。

另外考虑到我的炸弹人形象本身就是像素人，没有手臂，所以有关动画并没有做。

机器人策略：

```
while(!q.empty())
{
    node cur = q.front();
    q.pop();
    int x = cur.x;
    int y = cur.y;
    vis[x][y] = true;
    if(map[x][y] == ' ')
    {
        if_find_goal = true;
        goal = node(x,y);
        break;
    }
    if (judge_escape(x+1, y))
    {
        pre[x+1][y] = node(x, y);
        vis[x+1][y] = true;
        q.push(node(x+1, y));
    }
    if (judge_escape(x-1, y))
    {
        pre[x-1][y] = node(x, y);
        vis[x-1][y] = true;
        q.push(node(x-1, y));
    }
    if (judge_escape(x, y-1))
    {
        pre[x][y-1] = node(x, y);
        vis[x][y-1] = true;
        q.push(node(x, y-1));
    }
    if (judge_escape(x, y+1))
    {
        pre[x][y+1] = node(x, y);
        vis[x][y+1] = true;
        q.push(node(x, y+1));
    }
}
```

在本次项目实验的 AIController 设计过程中，我对原算法进行的较大的调整。我采用的是多套不同的宽度优先搜索策略混合，搭配“栈”来进行智能寻路。

首先，在机器人速度为零的条件下，利用 itemAt()扫描整个游戏界面，并以此构造字符型二维数组。为了规避全局变量的使用，我在 AIController 的私有成员中申请了 15*20 的二维数组，一个队列和一个栈。这样可以做到两个机器人分别寻路，互不干扰。

第一套宽度优先搜索策略是为了做到“抢占道具”和“攻击玩家”。我将第一套宽度优先搜索的搜索目标设置成道具或者玩家（非自己），且可到达的位置包括空地、软墙、玩家，而炸弹火焰可能波及到的地方，炸弹，硬墙不可到达。利用 pre[x][y]数组存储路径，找到目标后立即退出搜索循环，将 if_find_goal 置为 true；否则搜索队列为空，将 if_find_goal 置为 false。

第二套宽度优先搜索策略是为了躲避炸弹的爆炸伤害。与第一套宽度优先搜索策略类似，修改的部分只有搜索目标（炸弹火焰波及不到的地方）和是否可到达该位置的判断（炸弹火焰可能波及处可到达，不可停留）。

```
if(if_find_goal)
{
    while(goal.x != start.x || goal.y != start.y)
    {
        node temp = pre[goal.x][goal.y];
        if(goal.x - temp.x== -1)
            s.push(0);
        else if(goal.x - temp.x == 1)
            s.push(1);
        else if(goal.y-temp.y == -1)
            s.push(2);
        else
            s.push(3);
        goal = temp;
    }
}
```

由于搜索完成之后，若 if_find_goal 为 true，则可以认为此时我们已经知道目标的位置。利用 pre[x][y]数组在搜索过程中已经储存的路径，结合迭代的思想，很容易将最优路线复盘。不过注意到这种复盘是倒着（意思是从终点到起点）的，所以我想到了用“栈”这种数据结构来存储路径，依据后进先出的原则，复盘后的栈顶就是该机器人第一步该如何走。

由于每次机器人速度为零，都会利用 itemAt()扫描整个游戏界面，实际上路线策略是实时更新的，以此为基础，机器人可以表现出智能性。

遇到的困难：

机器人策略方面：扫描游戏界面时，可能出现人物与火焰、人物与炸弹、人物与道具重合的情况。若此时不加以特判，可能出现丢失 Player 本体的情况。这样会导致确定寻路起点时（利用 image->x()，image->y()，结合字符数组），访问空指针，最终程序 crashed 的情况。我的解决策略是，当碰到火焰、道具、炸弹等对象时，再以其为起点进行一次碰撞检测，以玩家>火焰>道具的优先级确定这个位置在字符数组中对应的字符。

游戏音乐方面：QUrl::fromLocalFile（"文件路径"）的理解一开始出现问题，且没有意识到 QSoundEffect* buttonSound 的定义可以写在构造函数里。导致自己第一次尝试的时候总把 buttonSound 作为临时变量，最终出现音乐无法正常播放的问题。在这里感谢李晗助教在添加游戏音乐方面提供的无私帮助。

实验心得：

在机器人策略选取的过程中，我查阅了不少路径搜索算法，包括 bfs、A*、Alpha-Beta 剪枝、在线算法等。这些算法各有各的特点，但是我觉得并不是复杂的算法在项目里跑出来的效果就一定好。有时候简单的算法，配合精巧的设计，也能起到很好的效果。

致谢：

和我一起学问题求解的同学：朱家辰；
指导我优化游戏音乐的助教：李晗学长。