C 语言设计程序：吃豆人

摘要

吃豆人是电子游戏历史上的经典街机游戏，游戏的主角小精灵的形象甚至被作为一种大众文化符号，一直沿用至今。《吃豆人》是易学难精的典型：控制吃豆人吃掉迷宫里面的所有豆子，同时尽可能躲避小鬼怪。一旦吃豆人吃掉能量药丸，它就可以在一定时间内反过来欺负小鬼怪了。特别值得一提的是，迷宫的左右出口是相通的，灵活应用往往带来柳暗花明的奇效。

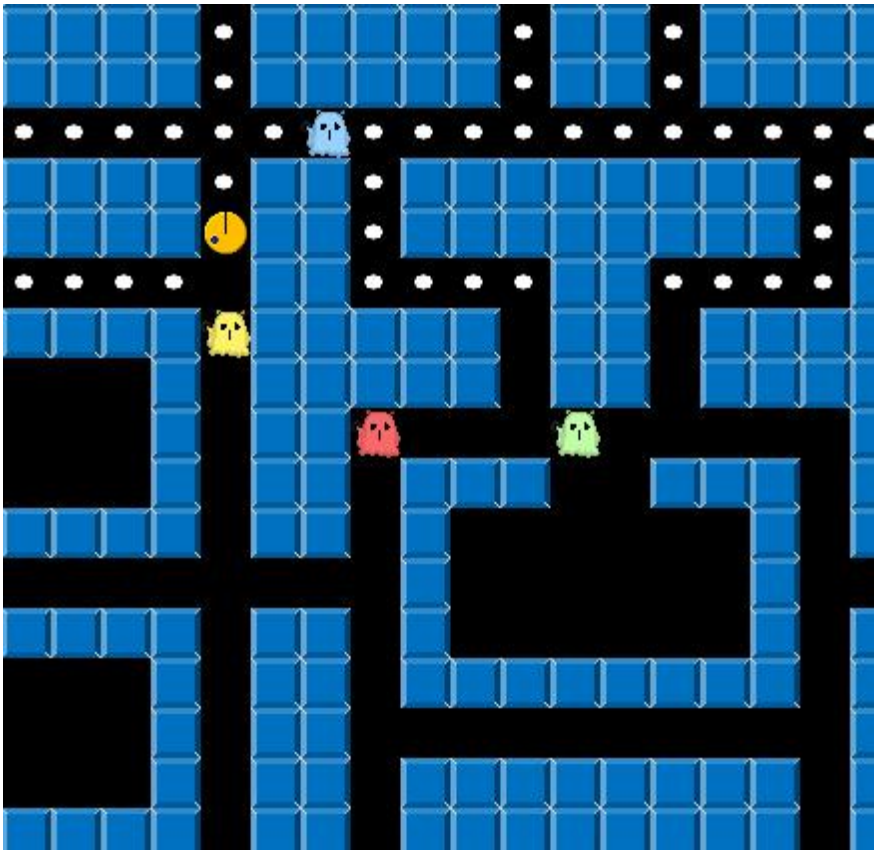
我们选择用 C++语言并结合当前流行的 AI 算法，编写吃豆人的程序作为我们的 C 语言面向程序设计课程的 project，期望编写出一个正常运行，鬼怪之间有默契，有合作的程序。

简介与问题描述

首先介绍吃豆人的游戏规则：控制吃豆人吃掉迷宫里面的所有豆子，同时尽可能躲避小鬼怪。吃豆人吃掉大力丸，它就可以在一定时间内反过来欺负小鬼怪了。吃豆人由玩家控制，共有十次机会，每次不更新地图，最终地图上无豆子即为吃豆人胜利。吃豆人嘴巴一张一合，形象描绘了“我吃，故我在”的生活态度。

其次将会介绍 project 主要用到的一些技术。Paceman 是一款益智休闲小游戏，能让你在闲暇之余享受游戏的乐趣。在编写的程序中，主要用到以下一些技术：

1. 对游戏界面的初始化和设定。
2. 析构函数。
3. 对 player 操作“合法性”的判断。
4. 对鬼移动的判断。
5. 利用 dfs 枚举鬼的所有行动方式，找出其中估价值最小的方案。
6. 鬼要捉到吃豆人最短路径的选择
7. 四个鬼之间的相互合作，就如下面图所显示的，四个鬼的行为并不是完全相同，蓝鬼和黄鬼拦住了吃豆人的去路，吃豆人失败是必定的。



再者介绍与其他项目相比具有的优势。基于上述技术，我们的程序不单单是实现了程序的正常运行，而且在此基础上保证了鬼与鬼之间的默契性，并根据游戏运行时出现的情况，寻找出解决方案，不断优化程序。总结来说，优势有以下几点：

① 鬼与鬼之间有很强的默契合作能力；

② 在吃豆人吃掉大力丸的一段时间内，鬼具有躲避吃豆人的“灵智”，避免被吃豆人吃掉；

③ 鬼运行路线计算方法不断优化，在保证游戏流畅运行的前提下，提高了游戏的难度，增加了游戏的趣味等。

最后介绍编写和运行程序过程中遇到和解决的问题：

1. 项目需要实现对 player 操作合法性进行判断
2. 项目需要提高鬼与鬼之间的契合度
3. 项目需要使鬼能够选择最短路线去追逐吃豆人

针对以上问题我们找出一下解决方案：

① 编写函数 Movementcheck，检查鬼和吃豆人位置和运动方向的合法性。

```
bool GameWidget::Movementcheck(int x,int y)
{
    if(x>=0&&x<Row && y>0&&y<Column)
    {
        if(Map[x][y]!=3 &&Map[x][y]!=2)
        {
            return true;
        }
    }
    return false;
}
```

②使不同的鬼分别“负责”吃豆人四个不同方向的堵截，体现出鬼与鬼之间的默契。

③使用 dfs 算法，枚举鬼堵截吃豆人能够走的几条路线，别选取最短的一条进行堵截。

小组分工

姓名	学号	专业	工作占比
郭鑫祥	10160537	信息与计算科学	40%
于海洋	21160927	信息与计算科学	40%

1） 郭鑫祥

在小组中负责界面的设计、对吃豆人及鬼行为进行判断、判断是否越界函数的编写、胜利及失败后的页面显示等

具体开发的功能有：

- 界面的设定和初始化
- 时序的控制及重新开局的初始化

- 一些移动中的动画的设置，吃豆人的动作的画面选择
- 判断是否越界及页面显示
- 对吃豆人和鬼的行为合法性的判断
- 鬼是如何配合移动的，编写函数得到每个鬼的移动方向（保证合法性）并移动鬼，再更新地图

2）于海洋

在小组中负责判断吃豆人是否碰到鬼函数的编写、地图的更新、鬼和吃豆人初始位置的设定和求地图上堵截吃豆人最短路的选取函数等。

具体实现的功能有：

- 判断吃豆人是否碰到了一个鬼；
- 地图更新的具体细节
- 判断玩家是否已吃完所有豆子
- 对鬼和吃豆人初始位置的设定
- 利用 bfs 求地图上每个点到达吃豆人当前点的最短路和应该走的方向

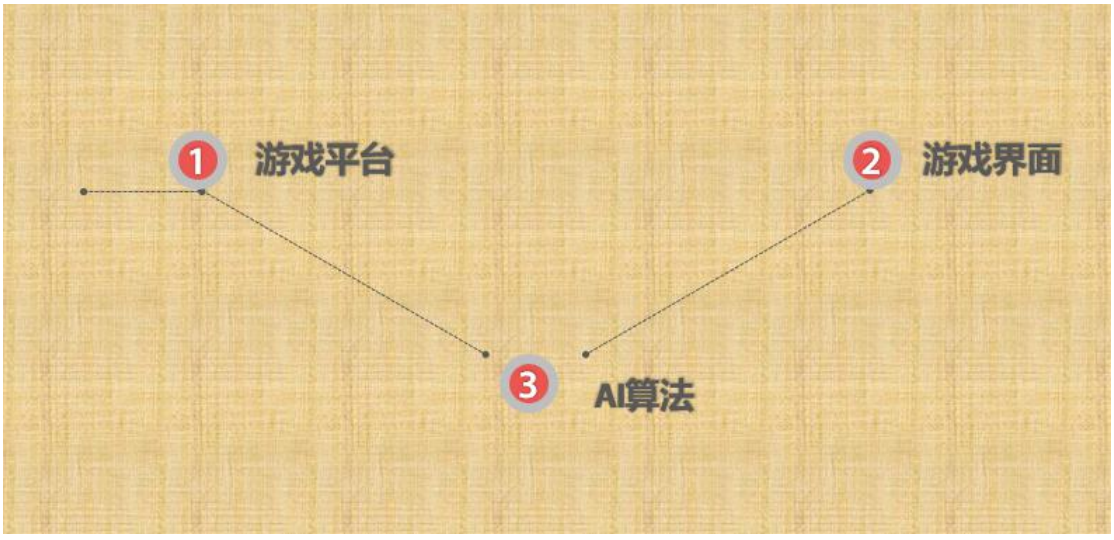
3）朋友

主要负责基于 QT 算法对游戏框架进行设计，负责 QT 算法的指导。

分析

经过分析和参考相关的游戏，我们认为吃豆人这款游戏的游戏规则应该是这样的：玩家可以根据键盘上的↑ ↓ ← → 控制吃豆人的前进方向，吃豆人每吃到一个豆子，分数就会加 1，吃光地图中的所有豆子后就会获得胜利。同时，地图中有 4 个幽灵，这些幽灵在每次游戏的开始会被在地图中间释放出来，并且释放之后就会追击吃豆人，假如吃豆人被追上，就会被幽灵吃掉导致死亡，然后会在出生点复活，每局有 10 次复活机会。在地图的四个角上有大力丸（红色的大豆子）当吃豆人吃掉大力丸后，短时间内就可以反吃 4 个幽灵，并且幽灵会朝着吃豆人的反方向逃跑。等到大力丸的时间过了，幽灵就会重新开始追击吃豆人。

为了实现这些功能，我们把这款游戏的实现分成 3 大部分，游戏平台，游戏界面，AI 算法。



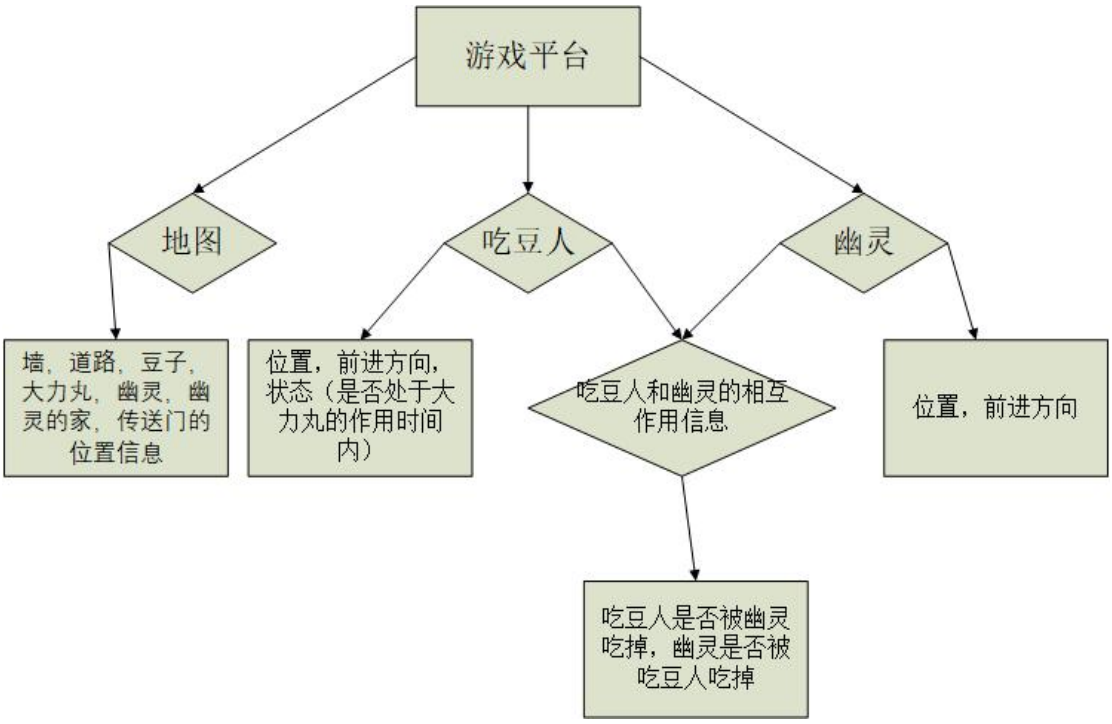
(1) 游戏平台

为了实现游戏平台的功能，我们分析得需要

1. 构建一个地图，地图上应该有墙，道路，豆子，大力丸，幽灵，幽灵

的家，传送门的位置信息

2. 构建关于吃豆人的数据结构储存吃豆人的所有信息，包括 吃豆人的位置，吃豆人的前进方向，吃豆人已经吃掉的豆子总数，吃豆人是否吃到大力丸并且在大力丸的作用时间内，吃豆人是否被幽灵吃掉，吃豆人是否把幽灵吃掉（处于大力丸状态）
3. 构建关于幽灵的数据结构存储幽灵的所有信息，包括 幽灵的位置，幽灵的前进方向，幽灵是否吃到吃豆人，幽灵是否被吃豆人吃掉
4. 构建关于幽灵和吃豆人的相互关系，包括 幽灵根据吃豆人的位置选择前进方向，幽灵是否被吃豆人吃掉，吃豆人是否被幽灵吃掉
5. 游戏开始和游戏结束的判断



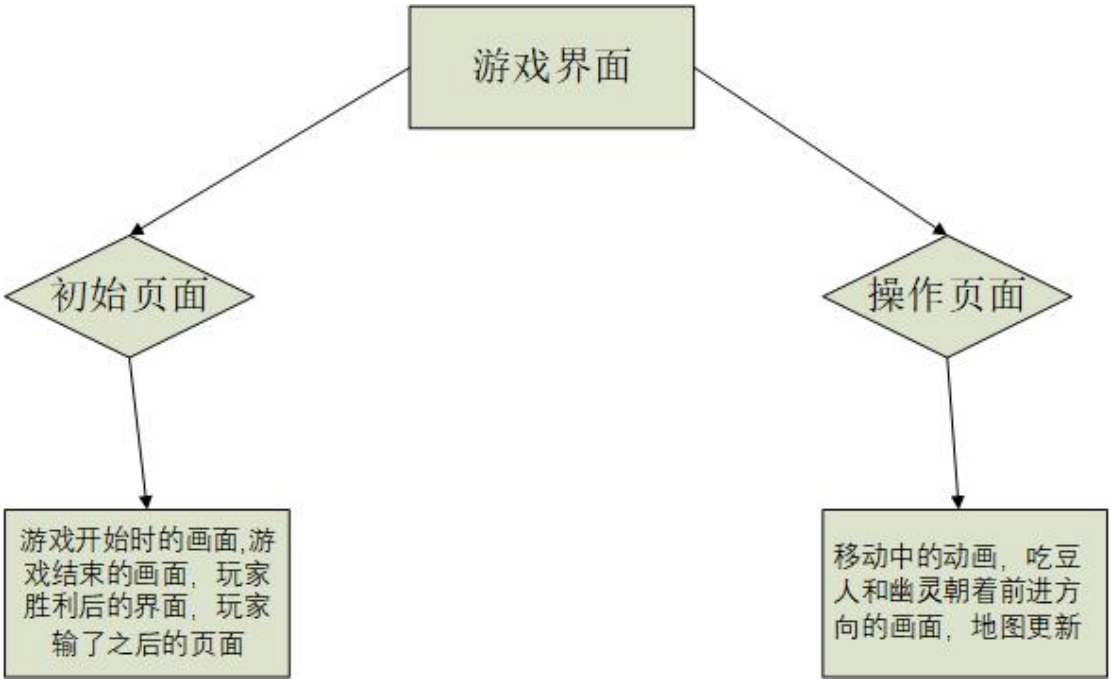
(2) 游戏界面

由于我们不懂关于 GUI 的一些知识，这一部分借鉴了网络上的一些模

板和同学的帮助。

经过我们分析，为了实现游戏界面的功能，我们需要

1. 初始页面，包括游戏开始时的画面设计，游戏结束的画面设计，玩家胜利后的界面显示，玩家输了之后的页面显示。
2. 操作页面，包括一些移动中的动画的设置，吃豆人的动作的画面选择，地图更新



(3) AI 算法

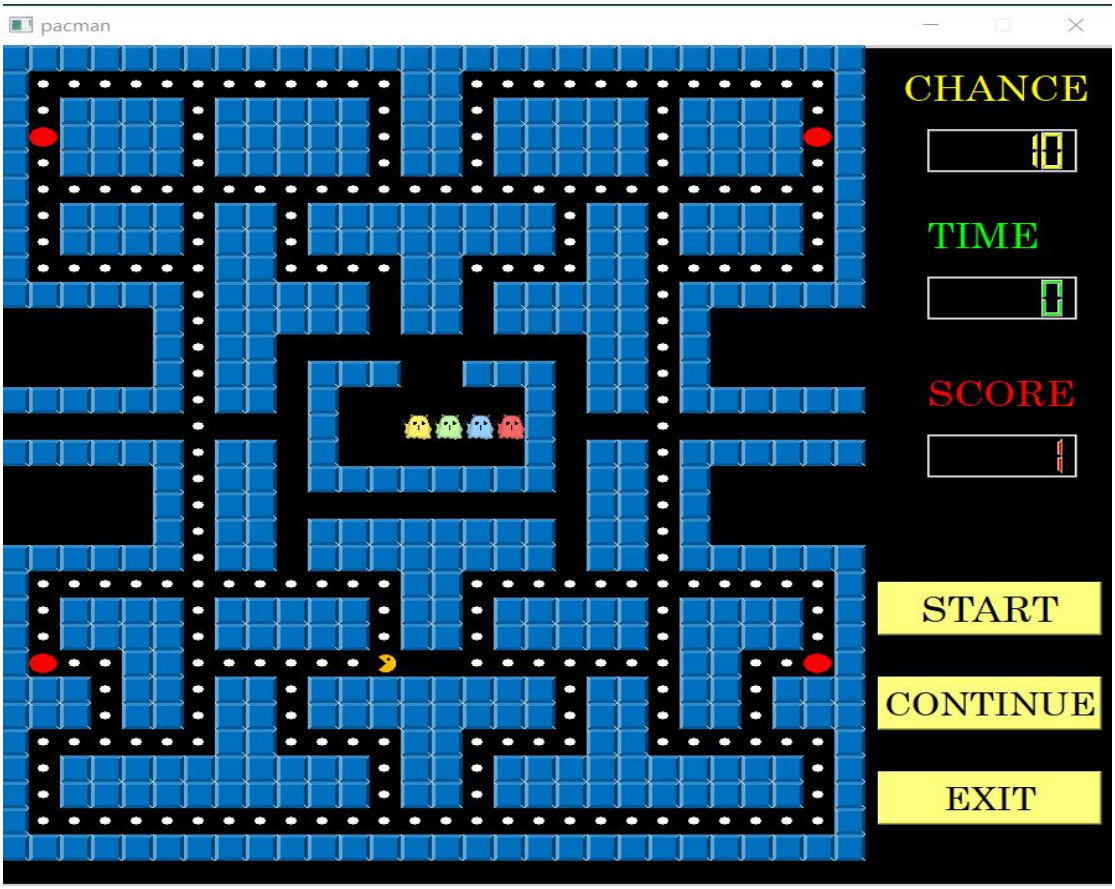
对于地图中的幽灵的移动路线，经过分析，需要编写一个算法使得幽灵更加智能，使得开局之后这些幽灵就开始追击吃豆人而不是处于随机巡逻状态。

设计

根据上面的分析，我们对此进行了下列设计

(1)对于游戏平台中的地图，我们构建一个二维数组作为地图的表达方式。

数组中不同的值对应数组不同的元素



地图中的墙可以用某个数字表示，地图中的道路用另外一个数字表示，这样就可以判断出吃豆人和幽灵的前进方向是否合法，如果合法，就朝着哪个方向移动，如果前进的方向前面有墙，就视为前进方向不合法，如果幽灵的前进方向不合法，那么就会自动换一个方向前进。如果吃豆人的前进方向不合法，就会停留在原地不动，直到玩家重新选择一个合法的前进方向。

(2) 一旦地图用二维数组表示出来，那么吃豆人，幽灵，豆子，大力丸的位置就可以用坐标(x,y)表示出来了。根据坐标，我们可以编写函数来判断各种条件，例如当吃豆人和豆子的坐标重合时，就认为吃豆人吃掉了豆子，当吃豆人和幽灵的坐标重合时，就认为吃豆人吃掉了幽灵或幽灵吃掉了吃豆人。当吃豆人和大力丸的坐标相同时，就认为吃豆人吃掉了大力丸，当吃豆

人和传送门的坐标相同时，就认为吃豆人达到了传送门，然后吃豆人会被传送到另外一个传送门

(3) 对于构建吃豆人的信息，我们是这样设计的。

用一个类(Class)来表示吃豆人的信息，这个类里面包括吃豆人的二维位置坐标，吃豆人目前的前进方向，判断吃豆人是否死亡的布尔值，

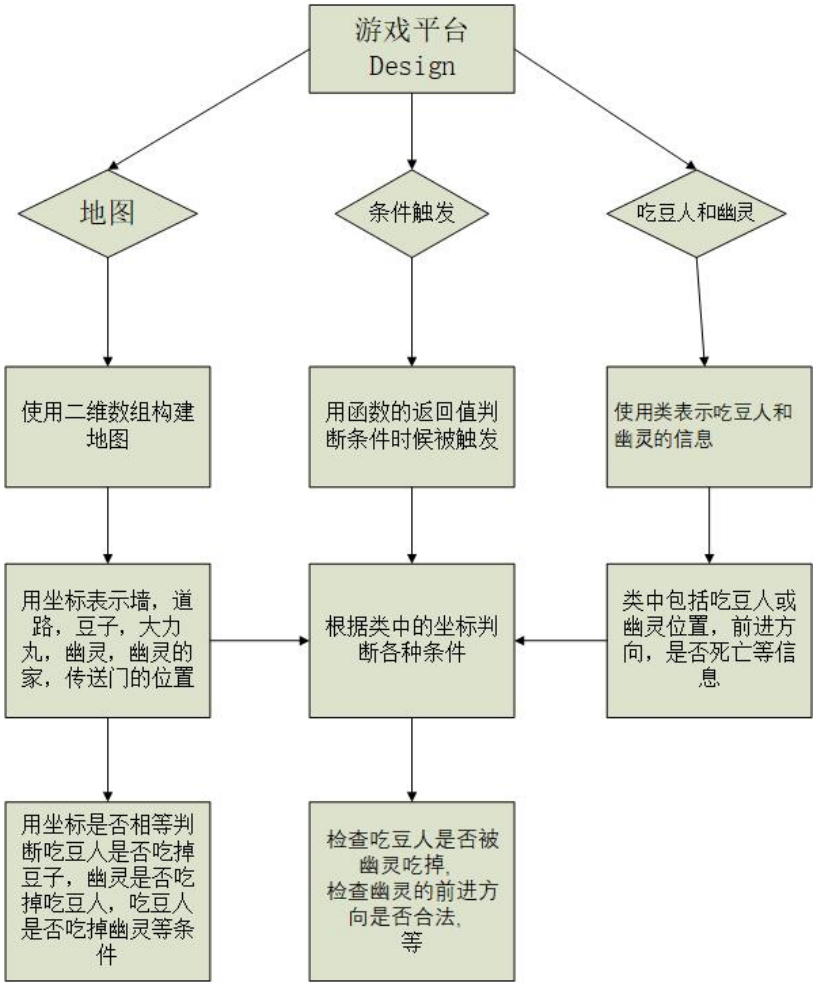
(4) 对于构建幽灵的信息，我们设计了一个向量来储存四个幽灵的信息，每个向量的分量是一个类，这个类里面包括幽灵的位置

(5) 对于判断各种触发条件，我们需要编写不同的函数

1. 检查某位置是否是墙：判断数组中的某位置是否合法（吃豆人能站在这个点上）
2. 检查吃豆人前进方向是否合法：向某一方向移动吃豆人 假如下一个位置合法则移动吃豆人
3. 检查吃豆人是否被幽灵吃掉：假如吃豆人与幽灵重合则吃豆人和幽灵回到初始位置，并且减少 1 次机会
4. 检查幽灵是否被吃豆人吃掉：假如在大力丸时效内的吃豆人吃掉了幽灵，则幽灵回到对应的初始位置
5. 检查幽灵的前进方向是否合法：通过函数得到每个幽灵的移动方向使得幽灵的前进方向是合法的，并移动幽灵
6. 判断玩家是否已吃完所有豆子：如果所有豆子被吃完，则触发胜利条件，如果没有全部被吃完，不触发任何条件
7. 判断是否越界：为了避免出现幽灵和吃豆人超过地图边界，需要增加

一个检查是否越界的函数

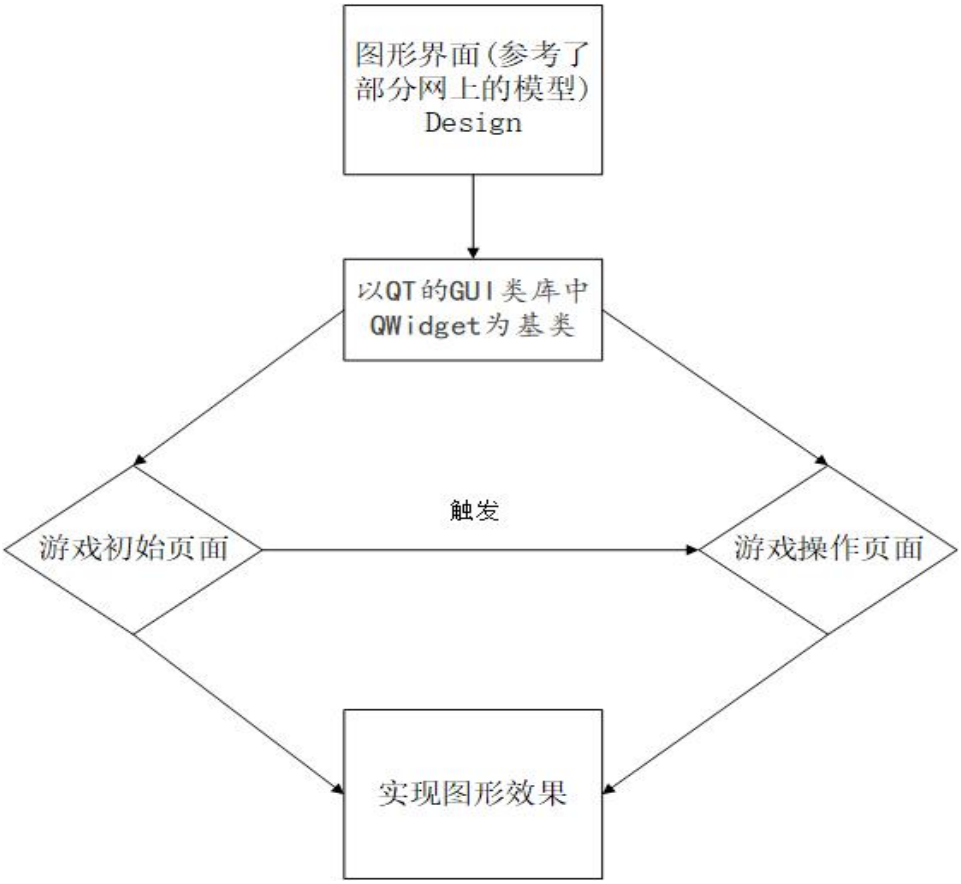
8. 检查更新地图：根据时序控制，每当吃豆人和幽灵完成一格的移动后，检查上述函数是否被触发，若触发，根据上述函数更新地图，吃豆人和幽灵的位置，若没有触发，根据前进方向更新地图，吃豆人和幽灵的位置



(6) 对于图形部分的设计，我们参考了网上的模板和借助同学的帮助，得到的设计如下：

使用 QT 编写，以 QT 的 GUI 类库中 QWidget 为基类，定义了两个界面类，以实现游戏的两大界面。

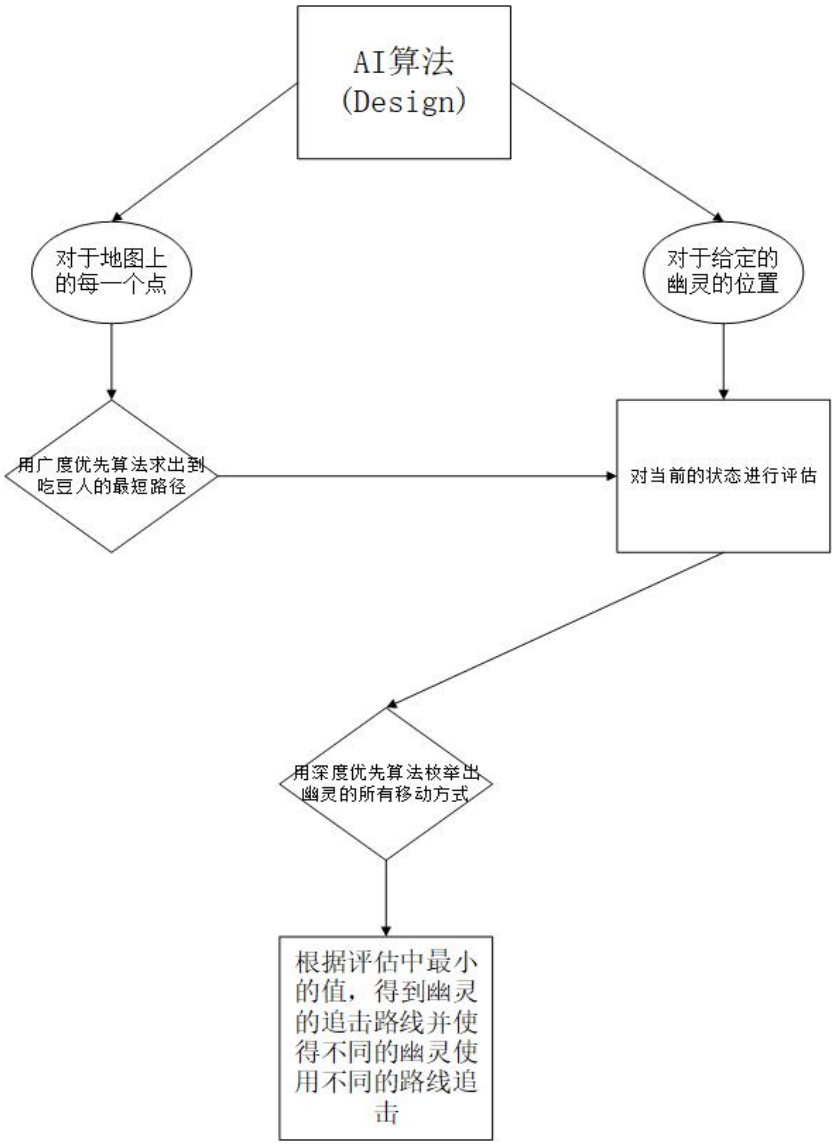
分为游戏初始界面与游戏操作界面，初始界面触发操作界面，基于游戏规则以及 AI 的算法实现，实现人机对战的图形化效果。



(7) 对于 AI 算法的设计，

由于幽灵随机移动会使这个游戏变得毫无挑战性，因此，必须设计算法使得在游戏开始之后就立即追击吃豆人

首先我们利用广度优先算法求地图上每个点到达吃豆人当前点的最短路和应该走的方向，然后对广度优先算法所要用到的数组进行初始化，并对当前局面做一个评估。最后，对于每一个幽灵确定的位置，用深度优先算法枚举出所有的可能路线，找出其中最小的评估值最小的方案，并使得不同幽灵走不同路线



实施

1. 首先用二维数组创建地图基本元素，其中，0 代表空道路，1 代表豆子，2 代表墙，5 代表空心的墙，3 代表幽灵的家，4 代表大力丸

```
int Map[31][28]=
{
    {2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2},
    {2,1,1,1,1,1,1,1,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,2},
    {2,1,2,2,2,2,2,1,2,2,2,2,2,1,2,2,1,2,2,2,2,2,1,2,2,2,2,1,2},
    {2,4,2,2,2,2,2,1,2,2,2,2,2,1,2,2,1,2,2,2,2,2,1,2,2,2,2,4,2},
    {2,1,2,2,2,2,2,1,2,2,2,2,2,1,2,2,1,2,2,2,2,2,1,2,2,2,2,1,2},
    {2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2},
    {2,1,2,2,2,2,2,1,2,2,2,1,2,2,2,2,2,2,2,2,2,1,2,2,2,1,2,2,2},
    {2,1,2,2,2,2,2,1,2,2,2,1,2,2,2,2,2,2,2,2,2,1,2,2,2,1,2,2,2},
    {2,1,1,1,1,1,1,1,2,2,1,1,1,1,2,2,1,1,1,1,2,2,1,1,1,1,1,1,2},
    {2,2,2,2,2,2,2,1,2,2,2,2,2,0,2,2,0,2,2,2,2,2,1,2,2,2,2,2,2},
    {5,5,5,5,5,5,2,1,2,2,2,2,2,0,2,2,0,2,2,2,2,2,1,2,5,5,5,5,5},
    {5,5,5,5,5,5,2,1,2,2,2,0,0,0,0,0,0,0,0,0,0,2,2,1,2,5,5,5,5,5},
    {5,5,5,5,5,5,2,1,2,2,2,0,2,2,2,2,0,0,2,2,2,0,2,2,1,2,5,5,5,5,5},
    {2,2,2,2,2,2,2,1,2,2,2,0,2,3,3,3,3,3,3,3,2,0,2,2,1,2,2,2,2,2,2},
    {0,0,0,0,0,0,0,1,0,0,0,2,3,3,3,3,3,3,3,2,0,0,0,1,0,0,0,0,0,0,0},
    {2,2,2,2,2,2,2,1,2,2,2,0,2,3,3,3,3,3,3,3,2,0,2,2,1,2,2,2,2,2,2},
    {5,5,5,5,5,5,2,1,2,2,2,0,2,2,2,2,2,2,2,2,2,0,2,2,1,2,5,5,5,5,5},
    {5,5,5,5,5,5,2,1,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,2,1,2,5,5,5,5,5},
    {5,5,5,5,5,5,2,1,2,2,2,0,2,2,2,2,2,2,2,2,2,0,2,2,1,2,5,5,5,5,5},
    {2,2,2,2,2,2,2,1,2,2,2,0,2,2,2,2,2,2,2,2,2,0,2,2,1,2,2,2,2,2,2},
    {2,1,1,1,1,1,1,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2},
    {2,1,2,2,2,2,2,1,2,2,2,2,2,1,2,2,1,2,2,2,2,2,1,2,2,2,2,1,2},
    {2,1,2,2,2,2,2,1,2,2,2,2,2,1,2,2,1,2,2,2,2,2,1,2,2,2,2,1,2},
    {2,4,1,1,2,2,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,2,2,1,1,4,2},
    {2,2,2,1,2,2,1,2,2,1,2,2,2,2,2,2,2,2,2,2,2,1,2,2,1,2,2,2},
    {2,2,2,1,2,2,1,2,2,1,2,2,2,2,2,2,2,2,2,2,2,1,2,2,1,2,2,2},
    {2,1,1,1,1,1,1,1,2,2,1,1,1,1,2,2,1,1,1,1,2,2,1,1,1,1,1,1,2},
    {2,1,2,2,2,2,2,2,2,2,2,2,1,2,2,1,2,2,2,2,2,2,2,2,2,2,1,2,2},
    {2,1,2,2,2,2,2,2,2,2,2,2,1,2,2,1,2,2,2,2,2,2,2,2,2,2,1,2,2},
    {2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2},
    {2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2}
};
```

2. 其次编写吃豆人的移动函数，先判断将要移动的方向是否合法，假如吃豆人与幽灵相遇则吃豆人回到初始位置，如果吃豆人在无敌时间，则幽灵回到对应的初始位置，移动完更新地图

```
void GameWidget::movePacman(int dir)
{
    int xx=Thepacman.x+dx[dir];
    int yy=Thepacman.y+dy[dir];

    if(Movementcheck(xx,yy))
    {

        Thepacman.x=xx;
        Thepacman.y=yy;
        Thepacman.face=dir;
    }

    for(int i=0; i<=3; i++)
    {
        if(meetGhost(i))
        {
            if(inv>0)//during invTime, Ghost is killed
            {
                initialGhost(i);
                PlayerPoint+=50;
            }
            else//Pacman is killed
            {
                if(!alreadycut)
                {
                    Chance_of_Pacman--;
                    ui->rounddisplay->display(Chance_of_Pacman);
                    alreadycut=true;
                    timer->stop();
                    viewupdate->stop();
```

3. 接下来是判断函数，判断吃豆人是否被鬼吃掉(其他的几个判断函数就不一一展示了)

```
bool GameWidget::meetGhost(int i)
{

    if(Thepacman.x==Ghost[i].x &&Thepacman.y==Ghost[i].y)
    {
        return true;
    }
    return false;

}
```

4. 编写地图更新函数，根据之前判断函数的结果来决定如何更新地图

```
void GameWidget:: updateMap()
{
    int x=Thepacman.x;
    int y=Thepacman.y;
    if(Map[x][y]==0)
    {
        return;
    }
    if(Map[x][y]==1)
    {
        PlayerPoint++;
        NumberofCoins++;
        ui->scoredisplay->display(PlayerPoint);
        Map[x][y]=0;
        return;
    }
    if(Map[x][y]==4)
    {
        PlayerPoint= PlayerPoint+2;
        NumberofCoins++;//???2
        ui->scoredisplay->display(PlayerPoint);
        Map[x][y]=0;
        inv=INCVTIME;
        ui->timedisplay->display(inv);
        timercount->start();
        //  ghosttimer->setInterval(800);
        return;
    }
}
```

5. 利用广度优先算法求地图上每个点到达吃豆人当前点的最短路和应该走的方向

```
int GameWidget::Bfs_dist(Paceman A)
{
    if(Map[A.x][A.y]==2)return 0;
    queue<Paceman>q;
    q.push(A);
    int dg=3;
    while(!q.empty())
    {
        Paceman u=q.front();
        if(G[u.x][u.y].size()<4&&degree[u.x][u.y]>2||A.x==u.x&&A.y==u.y)
        {
            G[u.x][u.y].push_back(Paceman(u.x,u.y));
        }
        q.pop();
        for(int i=1; i<=4; i++)
        {
            Paceman v;
            v.x= u.x+dx[i];
            v.y= u.y+dy[i];
            if(Over_map(v.x,v.y)==1) continue;

            if(dist[A.x][A.y][u.x][u.y]+1<dist[A.x][A.y][v.x][v.y])
            {
                dist[A.x][A.y][v.x][v.y]=dist[A.x][A.y][u.x][u.y]+1;
                if(dist[A.x][A.y][v.x][v.y]==1)
                {
                    orien[A.x][A.y][v.x][v.y]=i;

                }
                else orien[A.x][A.y][v.x][v.y]=orien[A.x][A.y][u.x][u.y];
                q.push(v);
            }
```

利用广度优先算法枚举幽灵的所有移动方式 找出期中估价值最小的方案

```
int GameWidget::Dfs_role(int ghost_num)
{
    if(ghost_num==5)
    {
        for(int i=1; i<=4; i++)
        {
            // Find_aim(Ghost[i],Thepacman,i,tt[i]);

            Q[i]=G[Thepacman.x][Thepacman.y][tt[i]-1];
        }
        if(min_steps>Evaluate_steps())
        {
            min_steps=Evaluate_steps();
            for(int i=1; i<=4; i++)
            {
                ins[i]=orien[ Ghost[i].x ][ Ghost[i].y ][ Q[i].x ][ Q[i].y ];
            }
        }
        return 0;
    }
    for(int i=1; i<=4; i++)
    {
        if(vist[i]) continue;
        tt[ghost_num]=i;
        vist[i]=1;
        Dfs_role(ghost_num+1);
        vist[i]=0;
    }
    return 0;
}
```

6. 每个幽灵对不同路径的选择方案和判断方案

```
int  GameWidget::Get_value(int aim_x,int aim_y,int ghost_num)
{
    int P_A=dist[Thepacman.x][Thepacman.y][aim_x][aim_y];
    int G_A=dist[Ghost[ghost_num].x][Ghost[ghost_num].y][aim_x][aim_y];
    int debuf=0;
    if(P_A<G_A) debuf=-(int)parameter[1];
    int debuf2=int(parameter[2]);
    int debuf3=0;
    if(P_A<=2&&P_A<=2) debuf+=parameter[3];
    if(P_A<=1&&P_A<=2) debuf=parameter[3];
    if(P_A<=2&&P_A<=1) debuf=parameter[3];
    if(P_A<=1&&P_A<=1) debuf+=parameter[3];
    if(orien[Thepacman.x][Thepacman.y][aim_x][aim_y]==Thepacman.face) debuf2+=parameter[2];
    return 50-G_A/2-P_A/2+(degree[aim_x][aim_y])*4+debuf-abs(G_A-P_A)+debuf2+debuf3+Find_exit(aim_x,aim_y);
}
int  GameWidget:: swp(int A,int B)
{
    P[0]=P[A];
    P[A]=P[B];
    P[B]=P[0];
    return 0;
}

int  GameWidget:: qsrt(int cnt)
{
    for(int i=1; i<=cnt; i++)
    {
        int k=1;
```

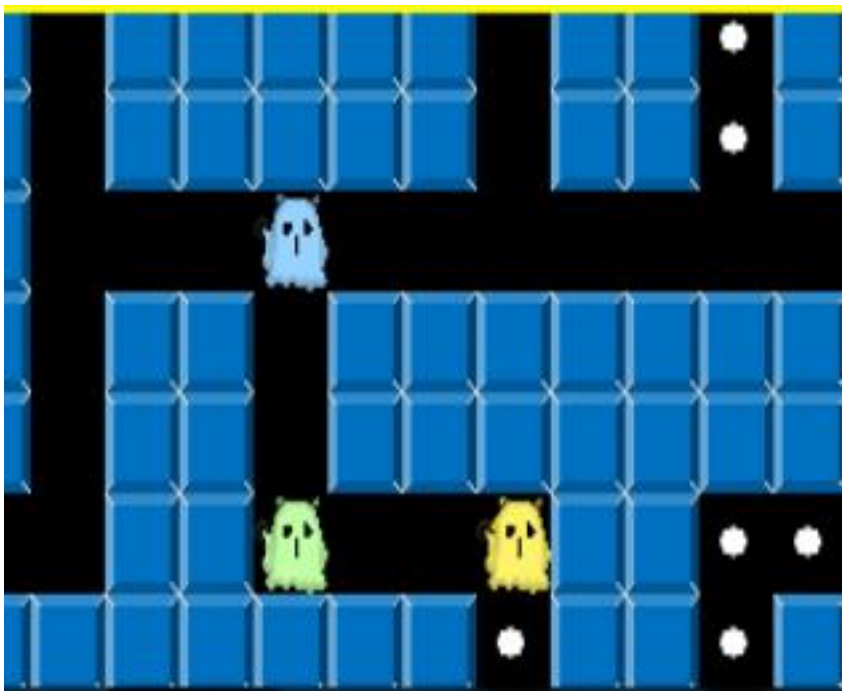
测试与调试

在测试的过程中根据出现的问题、bug 找出根源所在，并设计新的算法或是找出解决方案，实现版本的不断更新：

Version 1.0

Bug：

1. 使用相同策略下四个鬼重叠后无法再分开的问题。
2. 鬼始终在 paceman 的后面，无法实现前方堵截的目的。
3. 四个鬼行为相似。如图所示，三个鬼选择的路线相同，导致三个鬼都处于跟随吃豆人的状态，不能实现堵截。



Sol：

命令四个鬼分别瞄准 paceman 的上下左右不同的四个方向，从而避免四个鬼重叠、始终跟在 paceman 后面及四个鬼行为相似问题的出现。

相关更新的代码展示：

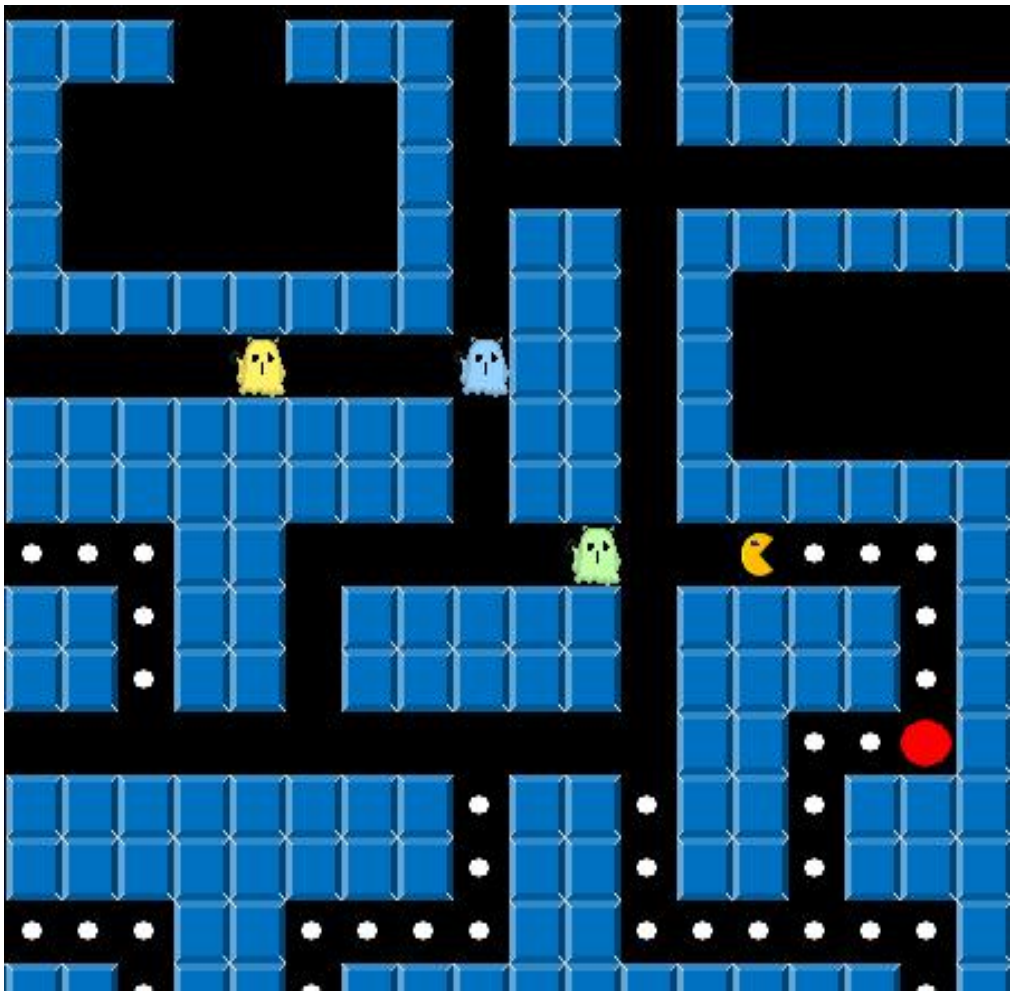
代码展示的是命令四个鬼走不同的路线。

```
int GameWidget::Divide_role()
{
    runner=0;
    int min_dist=(int)parameter[0];
    int max_dist=0;
    for(int i=1; i<=4; i++)
    {
        if(dist[Ghost[i].x][Ghost[i].y][Thepacman.x][Thepacman.y]<min_dist)
        {
            min_dist=dist[Ghost[i].x][Ghost[i].y][Thepacman.x][Thepacman.y];
            runner=i;
        }
        if(dist[Ghost[i].x][Ghost[i].y][Thepacman.x][Thepacman.y]>max_dist)
        {
            max_dist=dist[Ghost[i].x][Ghost[i].y][Thepacman.x][Thepacman.y];
            interceptor=i;
        }
    }
    if(inv>(int)parameter[5]) runner=0;
    for(int i=1; i<=4; i++)
    {
        if(i!=runner&&i!=interceptor)Find_aim(i);
        if(i==runner) Q[i]=Thepacman;
        if(i==interceptor) Find_aim(i);
    }
    for(int i=1; i<=4; i++)
    {
        ins[i]=orien[ Ghost[i].x ][ Ghost[i].y ][ Q[i].x ][ Q[i].y ];
    }
    return 0;
}
```

Version 2.0

Bug：

- 多个鬼一起尾追概率非常大，不能前方堵截，version 1.0 遇到的问题未完全解决。如图所示，黄、绿、蓝三个鬼均是呈尾随 paceman 的状态。



- 一旦鬼离吃豆人很近，四个鬼的策略几乎一样，体现不出合作的效果。

sol：

选取离 paceman 最近的三个连通度大于 2 的点，这里的连通度是指到达一个点能够走的方向数目。

相关代码展示：

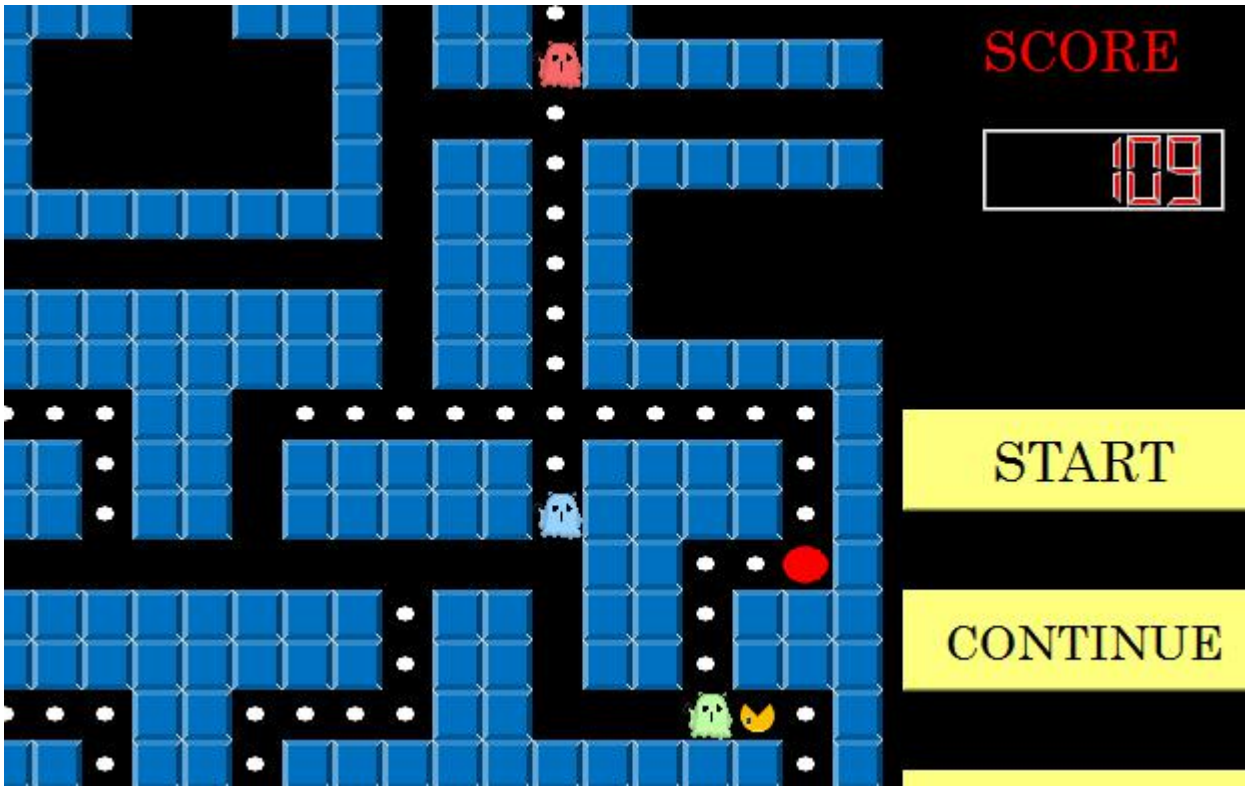
```
int GameWidget::Dfs_role(int ghost_num)//dfs?????????
{
    if(ghost_num==5)
    {
        for(int i=1; i<=4; i++)
        {
            // Find_aim(Ghost[i],Thepacman,i,tt[i]);

            Q[i]=G[Thepacman.x][Thepacman.y][tt[i]-1];
        }
        if(min_steps>Evaluate_steps())
        {
            min_steps=Evaluate_steps();
            for(int i=1; i<=4; i++)
            {
                ins[i]=orien[ Ghost[i].x ][ Ghost[i].y ][ Q[i].x ][ Q[i].y
            }
        }
        return 0;
    }
    for(int i=1; i<=4; i++)
    {
        if(vist[i]) continue;
        tt[ghost_num]=i;
        vist[i]=1;
        Dfs_role(ghost_num+1);
        vist[i]=0;
    }
    return 0;
}
```

使用 dfs 算法枚举鬼的所有移动方式，并选取联通度大于 2 的点作为优先选择的方向，从而避免上述情况的出现。

Version 3.0

Bug：

- 吃豆人必死局面中，追求连通度高的方向，在即将拦截成功并且吃掉吃豆人的时候向相反方向移动，从而使得吃豆人逃脱。如图所示的绿鬼，明明只要向右走就能吃掉吃豆人，可是为了追求连通度更高的上，使吃豆人逃掉。



Sol：

最近的一个鬼紧追吃豆人，其他的鬼都对地图做一个评估，计算一个权值：

对于地图上的一个点 A，对鬼 G 而言，假设吃豆人在点 P

1）如果 dist[P][A]<dist[G][A] ，吃豆人先到达这个点，施加一个惩罚

2）如果 orientation[P][A]=face, 认为这些点很好的符合在吃豆人行进前方“堵截”的行为，施加一个加成

3）如果 dist[G][A]<<dist[P][A] ，吃豆人可能 “换路”绕过目标点，施加一个惩罚

4）如果 degree[A]>2(连通度)，十字路口或者丁字路口，施加一个加成

代码展示如下：

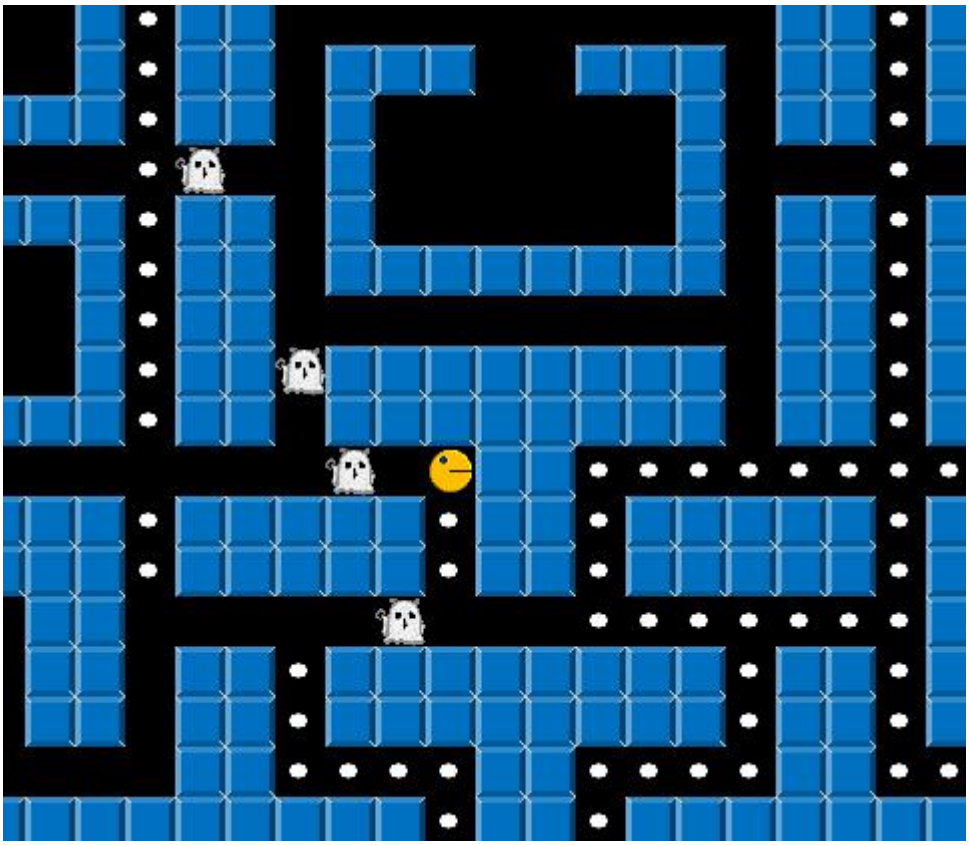
```
int GameWidget::Bfs_dist(Paceman A)//bfs ??? (Get_dist)
{
    if(Map[A.x][A.y]==2)return 0;
    queue<Paceman>q;
    q.push(A);
    int dg=3;
    while(!q.empty())
    {
        Paceman u=q.front();
        if(G[u.x][u.y].size()<4&&degree[u.x][u.y]>2||A.x==u.x&&A.y==u.y)
        {
            G[u.x][u.y].push_back(Paceman(u.x,u.y));
        }
        q.pop();
        for(int i=1; i<=4; i++)
        {
            Paceman v;
            v.x= u.x+dx[i];
            v.y= u.y+dy[i];
            if(Over_map(v.x,v.y)==1) continue;

            if(dist[A.x][A.y][u.x][u.y]+1<dist[A.x][A.y][v.x][v.y])
            {
                dist[A.x][A.y][v.x][v.y]=dist[A.x][A.y][u.x][u.y]+1;
                if(dist[A.x][A.y][v.x][v.y]==1)
                {
                    orien[A.x][A.y][v.x][v.y]=i;

                }
                else orien[A.x][A.y][v.x][v.y]=orien[A.x][A.y][u.x][u.y];
                q.push(v);
            }
        }
    }
    return 0;
}
```

Version 4.0

Bug：

大力丸状态下，鬼没有躲避吃豆人

如图所示，在吃豆人吃掉大力丸无敌时间内，鬼并没有选择远离，而是朝着吃豆人的方向移动，导致被吃豆人吃掉。



Sol：

补丁，当进入长直路时，使鬼前去吃豆人朝向前方的长直路第一个路口(连通度大于 2 的点)堵截。

相关代码展示：

```
int GameWidget:: Get_degree()
{
    for(int i=1; i<=Row; i++)
    {
        for(int j=1; j<=Column; j++)
        {
            for(int k=1; k<=4; k++)
            {
                int tx=i+dx[k];
                int ty=j+dy[k];
                if(Over_map(tx,ty)||Map[tx][ty]==2) continue;
                if(Map[tx][ty]!=2) degree[tx][ty]++;
            }
        }
    }

}
int flag1;
```

结果与结论

得到最终的程序，运行正常，且在追逐吃豆人的过程中，鬼与鬼之间有良好的默契性，往往都是能够将吃豆人堵截在长直通道内。

通过编写程序，提高自己编程能力的同时，也学会了很多课外的东西，提高了自己发现问题解决问题的能力。