基于 A*寻路算法的模拟 AI 行为逻辑 BubbleMan 游戏

游戏设计结课实验

任课老师: 张军

实验时间: 20-21 学年 第 一 学期

软件支持: vs2017 unity 2019.4.9f1

硬件支持: win10 Mac 笔记本电脑,iPad2019,Apple Pencil 1.0

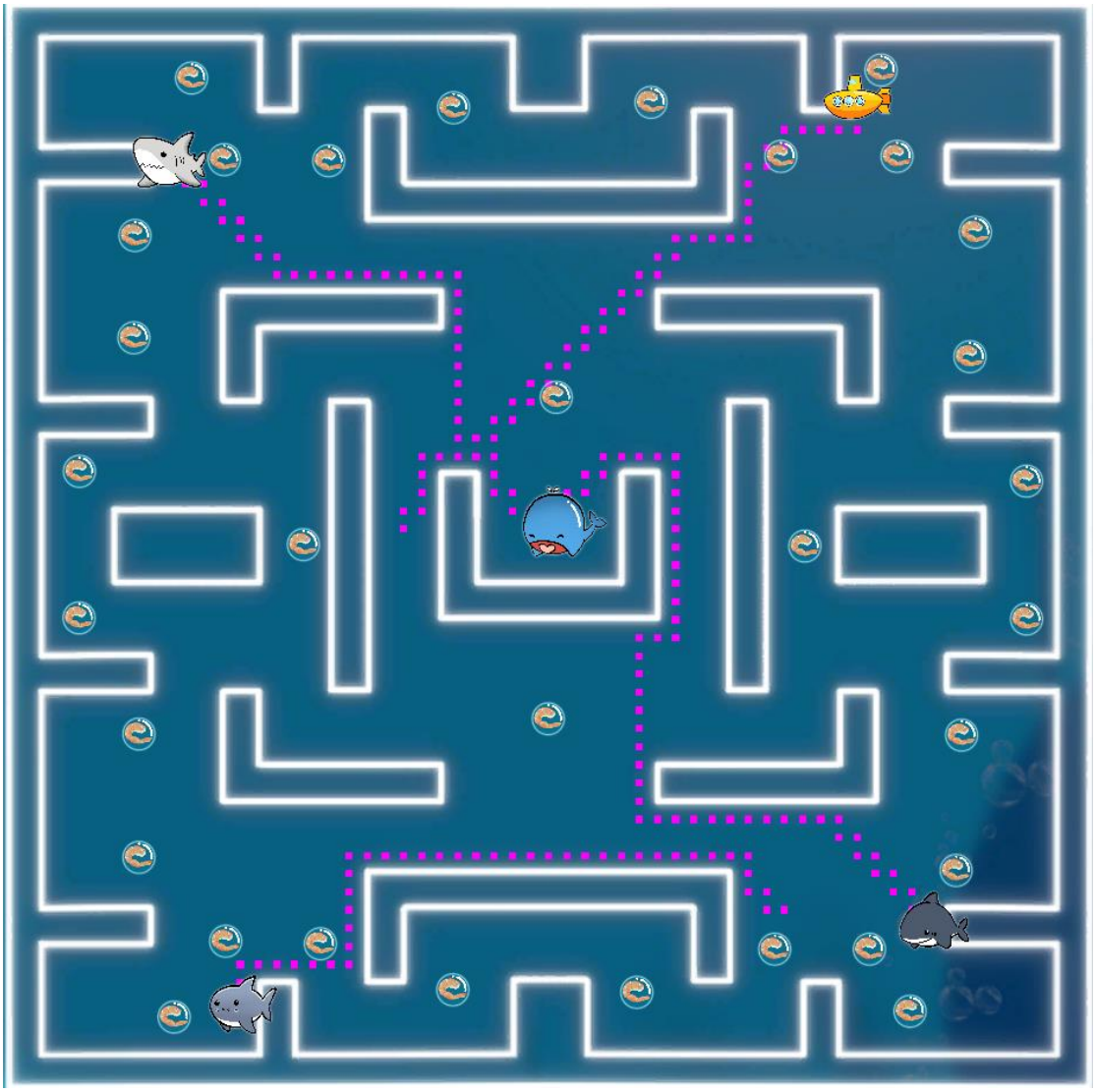
技术参考: 《游戏编程精粹 1》第三章人工智能与 A*算法

游戏版本: Bubble Man 4.0

2021-1-2

江南大学人工智能学院数媒 1801
李泽坤 王哲昊 杨洪齐

目录



摘要：吃豆人作为一款经典游戏，经久不衰，但是现代计算机上对于吃豆人的复刻却不是很好。本文研究吃豆人的 角色，导航，META AI 行为逻辑，并添加了创新机制，提供了一款复刻吃豆人的游戏 BubbleMan。

关键词：BubbleMan，A*寻路算法,AI 行为逻辑，游戏心理分析

引言

吃豆人，你可能没有玩过，但是大概率听过它，这是有史以来最受欢迎的街机，在被发售的十年内，生产了 40 万台，有 120 亿个硬币被投入。今年距离第一台吃豆人街机的诞生，正好是四十周年。值此时机，我们也想带着现代人的思考，结合 A*算法与人

工智能理论，引入了角色 AI，META AI ，导航 AI 复刻一下经典。本文将针对吃豆人 AI 行为逻辑和寻路算法做出一些研究，并复刻为游戏 BubbleMan。

2. 导航 AI A*寻路算法

A*算法在人工智能学术界中是一种历久弥新的算法。1968 年以来已用于解决各类问题，并且它对于路径规划问题也是非常有效的。

A*是一种在状态空间中进行搜索的算法，它主要是通过检查特定状态的相邻或邻接状态来搜索从起始状态到目标状态的最小代价路径。在路径规划问题中，状态由主体在游戏世界中占有的特定位置组成，一个邻接状态是通过移动主体到一个邻接位置而达到的。

2.1. 简化搜索区域

寻路的第一步是简化成容易控制的搜索区域。

怎么处理要根据游戏来决定了。例如，我们这款基于方块的游戏来说使用方块（四分之一个正方形）作为寻路算法的单元。其他的形状类型也是可能的（比如三角形或者六边形），但是正方形是最简单并且最适合我们需求的。

像那样去划分，我们的搜索区域可以简单的用一个地图大小的二维数组去表示。所以如果是 30*30 方块大小的地图，我们的搜索区域将会是一个有 3600 个正方形的数组。

2.2. Open 和 Closed 列表

BubbleMan 没有人的记忆力，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）

首先在 closed 列表中添加当前位置（我们把这个开始点称为点 “A”）。然后，把所有与它当前位置相邻的可通行小方块添加到 open 列表中。

现在 BubbleMan 需要判断在这些选项中，哪项才是最短路径，但是它要如何去选择呢？

在 A 星寻路算法中，通过给每一个方块一个和值，该值被称为路径增量。

2.3. 路径增量

确定路径时使用哪个正方形的关键在于以下方程式：

- F = G + H

  其中

- G =从起点 A 到网格上给定正方形的移动量，即沿着生成的路径移动到那里。
- H =从网格上的给定正方形移至最终目标点 B 的估计移动量。这通常称为启发式，可能会造成混淆。 之所以这样说，是因为这是一个猜测。 在找到路径之前，我们不知道实际距离，因为各种各样的东西都可能挡住道路（墙壁，水等）。

“移动量”只是个名词。在游戏中，这个概念很简单 –仅仅是四分之一个方块的数量。然而，在游戏中你可以对这个值做调整。例如：

本游戏中从八邻域计算最近点改为了四邻域计算，以防止 Ghost 穿过墙壁。

关于 G 值

G 是从开始点 A 到达当前方块的移动量（在本游戏中是指四分之一方块的数目）。

为了计算出 G 的值，我们需要从它的前继获取，然后加 1。所以，每个方块的 G 值代表了从点 A 到该方块所形成路径的总移动量。

关于 H 值

H 值是从当前方块到终点的移动量估算值。

移动量估算值离真实值越接近，最终的路径会更加精确。如果估算值停止作用，很可能生成出来的路径不会是最短的（但是它可能是接近的）。

为了提高运算效率，我们使用“曼哈顿距离方法”，它计算出距离点 B 剩下的水平和垂直的方块数量并加权。

2.4. 算法流程

知道如何计算每个方块的和值 F（F= G+H）

BubbleMan 会重复以下步骤来找到最短路径：

- 将方块 S 添加到 open 列表中，该列表有最小的和值。
- 将 S 从 open 列表移除，然后添加 S 到 closed 列表中。
- 对于与 S 相邻的每一块可通行的方块 T：

  如果 T 在 closed 列表中：不管它。

  如果 T 不在 open 列表中：添加它然后计算出它的和值。

  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 和值是否更小。如果是，更新它的和值和它的前继。

2.5. 注意点

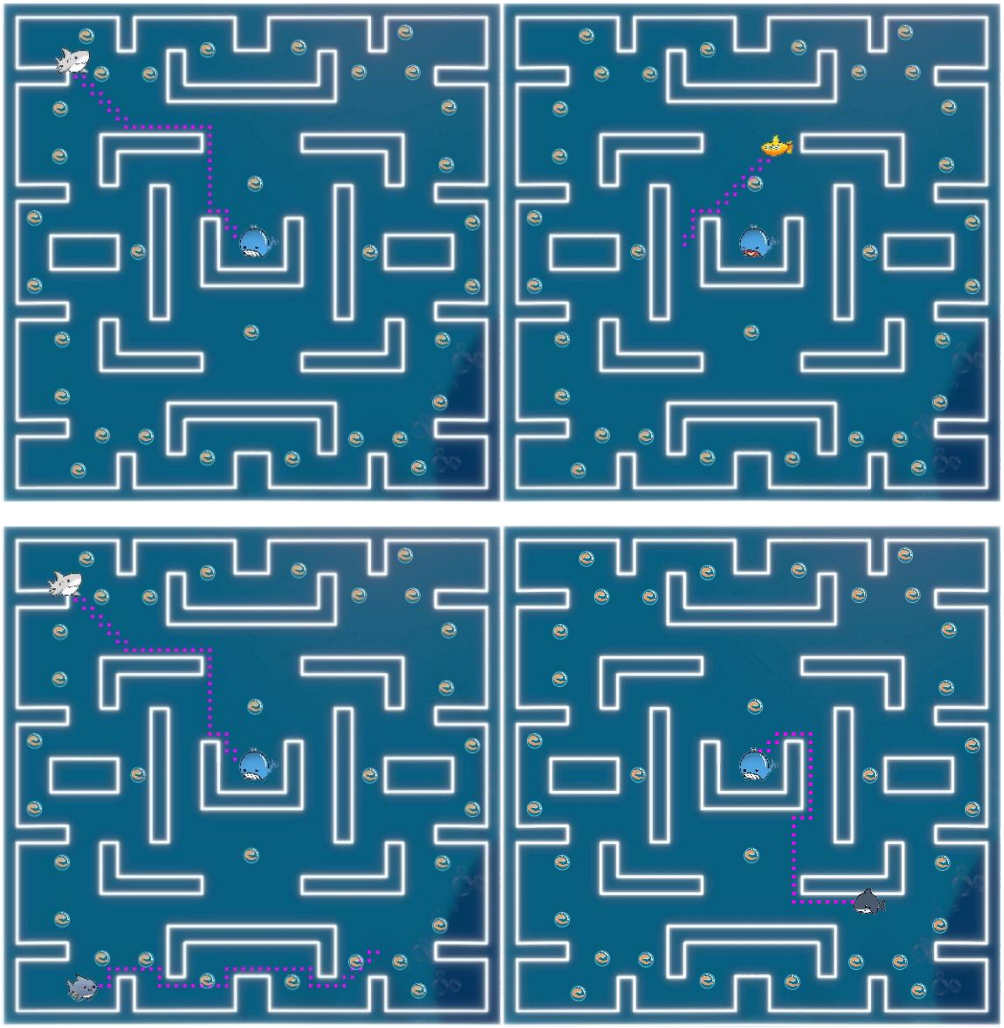
对与 BubbleMan 来说移动的位置并不每次都在四分之一方格交叉线上，所以需要对每个 BubbleMan 点进行近似处理。并且寻找 BubbleMan 近似点的四邻域可用坐标，作为 A*的 B 点。

3. 角色 AI Ghost 行为逻辑与游戏心理分析

3.1. Ghost 行为逻辑

游戏中共有四个 Ghost，Enemy1，Enemy2，Enemy3，Enemy4。

- Enemy1 的逻辑比较简单，直接追逐 BubbleMan。
- Enemy2 则相对复杂一些，寻路 BubbleMan 前方 4 个格子（8 个节点）进行堵截。
- Enemy3 以 Enemy1 位置和 BubbleMan 前方 1 个格子（2 个节点）连线的一倍延长线端点为目标移动。这样意味着有时候会和 Enemy1 一起追逐 BubbleMan，有时候会像 Enemy2 在前方堵截。
- Enemy4 最特殊，它直接追逐 BubbleMan，但是当接近 BubbleMan 一定范围内，就会原路返回一定移动量，然后继续追逐。



3.2. 游戏心理分析

这样进行 AI 设计，可以使玩家觉得敌人是毫无规律的乱动，但又不像随机那样难以琢磨，导致玩家觉得失败是游戏的错

3.3. Ghost 行为逻辑算法

- **Enemy1 略**
- **Enemy2 略**
- **Enemy3 略**
- **Enemy4**

```
    bool reverse = false;
                if (!reverse)
                {
                    if (transform.position != wayPoints[0])
                    {
                        Vector2 temp = Vector2.MoveTowards(transform.position, wayPoints[0],
speed);
                        GetComponent<Rigidbody2D>().MovePosition(temp);
                    }
                    else
                    {
                        switch (State)
                        {
                            case 0:
                                FindingPath(grid.player.position, grid.destPos.position);
                                break;
                            case 1:
                                FindingPath(grid.player.position, home.transform.position);
                                break;
                            case 2:
                                dots =
GameObject.Find("Dots").gameObject.GetComponent<DotPosList>().dotlist[3].transform.position;
                                FindingPath(grid.player.position, dots);
                                if ((grid.player.position - grid.destPos.position).sqrMagnitude
```

```
<= 100)
                                {
                                    State = 0;
                                    speed = 0.05f;
                                }
                                break;
                            case 3:
                                FindingPath(grid.player.position, grid.player.position);
                                break;

                        }
                        wayPointsGos = quad.GetComponent<Grid>().pathObj.ToArray();
                        LoadAPath(wayPointsGos);
                        if (State != 3)
                        {
                            if (wayPoints[0] == transform.position)
                                wayPoints.RemoveAt(0);
                        }
                        lastwayPoints.Insert(0, wayPoints[0]);
                        if (lastwayPoints.Count >= 17)
                            lastwayPoints.RemoveAt(16);
                        if ((grid.player.position - grid.destPos.position).sqrMagnitude <= 60)
                            reverse = true;

                    }
                }
                else
                {
                    if (transform.position != lastwayPoints[index])
                    {
                        Vector2 temp = Vector2.MoveTowards(transform.position,
lastwayPoints[index], speed);
                        GetComponent<Rigidbody2D>().MovePosition(temp);
                    }
                    else
                    {
                        index++;
                        if (index == lastwayPoints.Count)
                        {
                            index = 0;
                            reverse = false;
                            lastwayPoints.Clear();
```

```
                    FindingPath(grid.player.position, grid.destPos.position);
                    wayPointsGos = quad.GetComponent<Grid>().pathObj.ToArray();
                    LoadAPath(wayPointsGos);

                }
            }
        }
        break;
```

4. META AI BubbleMan 机制与游戏心理分析

4.1. BigForcePlay 惊吓机制

BigForcePlay 是一种特殊的道具，BubbleMan 获得后会使所有 Ghost 暂停 4 秒，其间触碰 Ghost 会使四个 Ghost 处于惊吓状态返回各自的巢穴。

4.2. 游戏心理分析

“给玩家喘息的机会，而不只是一味地增加压力。”这样的设计让整个游戏充满刺激，但又不至于没有还手之力。

4.3. BigForcePlay 惊吓机制算法

```
public int State = 0;//0 为正常寻路 1为四散模式 2为保护dot模式 3为震惊模式
private void OnTriggerEnter2D(Collider2D collision)
{
    GameObject enemy = GameObject.Find("EnemyAll").gameObject;
    if (collision.name == "Hero")
    {
        foreach (Transform child in enemy.transform)
        {
            child.gameObject.GetComponent<GhostMoveAll>().State = 3;
        }
        GameObject.Find("Audio_Alarm").gameObject.GetComponent<AudioSource>().Play();
        GameObject.Find("AimTxt").gameObject.GetComponent<AimTxt>().isBig = false;
        GameObject.Find("Maze").gameObject.GetComponent<Animator>().SetBool("isAlarm",
```

```
true);
            Destroy(this.gameObject);
        }
    }
        switch (State)
        {
            case 3:
                tempTime += Time.deltaTime;
                if (isDie)
                {

GameObject.Find("Maze").gameObject.GetComponent<Animator>().SetBool("isAlarm", false);
                    foreach (Transform child in parentObj.transform)
                    {
                        child.gameObject.GetComponent<GhostMoveAll>().speed = 0.1f;
                        child.gameObject.GetComponent<GhostMoveAll>().State = 1;
                        child.gameObject.GetComponent<GhostMoveAll>().tempTime = 0;

                    }

                    isDie = false;
                }
                else if (tempTime >= 4)
                {
                    State = 0;
                    tempTime = 0;

GameObject.Find("Maze").gameObject.GetComponent<Animator>().SetBool("isAlarm", false);
                }
                break;
            case 1:
                foreach (Transform child in parentObj.transform)
                {
                 if ((child.transform.position - home.transform.position).sqrMagnitude <= 3)
                    {
                        child.gameObject.GetComponent<GhostMoveAll>().speed = 0.05f;
                        child.gameObject.GetComponent<GhostMoveAll>().State = 0;
                    }
                }
                break;
            default:
                break;
```

```
}
if (collision.gameObject.GetComponent<GhostMoveAll>().State == 3
    ||
    collision.gameObject.GetComponent<GhostMoveAll>().State == 1)
{
    collision.gameObject.GetComponent<GhostMoveAll>().isDie = true;
}
```

4.4. ProtectDots 保护机制

当游戏剩余若干 dots 时，则会触发保护机制，四个 Ghost 会加快速度保护剩余 dots，如果 BubbleMan 接近这些 dots 一定距离，保护的 Ghost 就会追向 BubbleMan。

4.5. 游戏心理分析

玩家总是在游戏快要结束时放松，所以给游戏增加一些难度。如果因此输掉游戏，也会觉得可惜，忍不住再玩一次。

4.6. ProtectDots 保护机制算法

用一个 List 维护剩余 dots 列表。

```
public List<GameObject> dotlist = new List<GameObject>();
                        dots =
GameObject.Find("Dots").gameObject.GetComponent<DotPosList>().dotlist[0].transform.position;
                        FindingPath(grid.player.position, dots);
```

5. 美术设计思路

基于吃豆人的游戏设计理念，希望能够创造属于自己的独特角色，最初的构思是主角为小鲨鱼，后来将其改成敌人，主角替换成小鲸鱼，（最后灵感枯竭把潜水艇也加了进去）

6. 项目分工

游戏理念提出 王哲昊

制作大纲制定 王哲昊 李泽坤

寻路算法脚本实现 李泽坤

AI 行为逻辑脚本实现 李泽坤

游戏 UI 功能脚本 王哲昊

场景与关卡搭建 王哲昊

脚本调试与参数调校 王哲昊 李泽坤

动画机配置 王哲昊

音乐配置 王哲昊

关卡设计与美术 杨洪齐

实验报告制作 李泽坤

Post 文稿设计与制作 李泽坤

演示视频录制 李泽坤

7. 参考文献与资料

1. https://www.gamedev.net/articles/programming/artificial-intelligence/a-pathfinding-for-beginners-r2003/
2. Mark DeLoure 编 王淑礼 张磊 译 姚审 校 游戏编程精粹 1 人民邮电出版社

8. 写在最后

我们的确不是很擅长制作游戏。巡遍全网都没有找到好点子。

但是偶然间看到了一个关于吃豆人的视频。

“做一个吃豆人吧！“——哲昊说。“吃豆人！Pacman！”洪齐附和道。

于是，做了一学期 3D 游戏的我们，又重拾了对于经典的热爱。

这次，我真正体会到了做游戏的快乐，明白了老师说的那把火。——李泽坤。

的确。我们不是什么大佬，不能做出什么令人惊叹的作品。甚至除了哲昊，我们两个也不是什么游戏发烧友。但是对于这个吃豆人，我们确实拿出了较为认真的态度，基本还原了吃豆人的行为逻辑和关卡效果，除此之外还加入了例如幽灵保护豆子，被吃会惊恐四散的机制。还原经典的同时，也多了一份后来人的思考。

希望您在玩的时候也能重新体会曾经街机时代的快乐并感受到我们制作时的喜悦。

另:我真的挺喜欢玩这个游戏的!起码满足了“如果自己都不想玩,那做的肯定不好”的要求。