

寻路系统技术专题

A星寻路算法

NavAstar

作者：Plane老师

微信：PlaneZhong

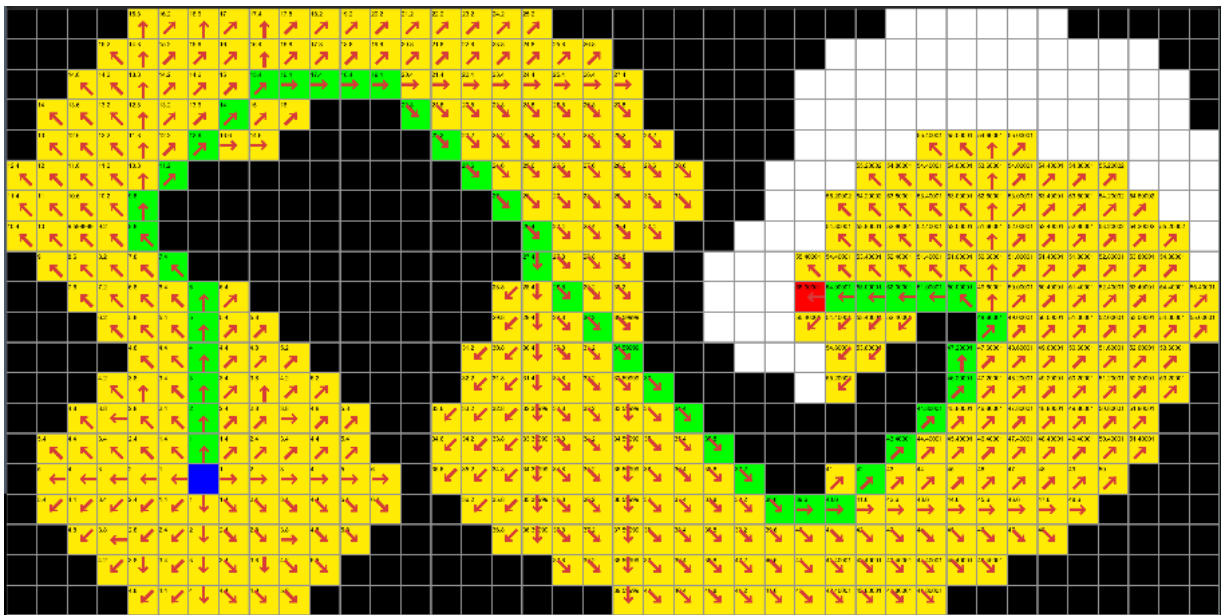
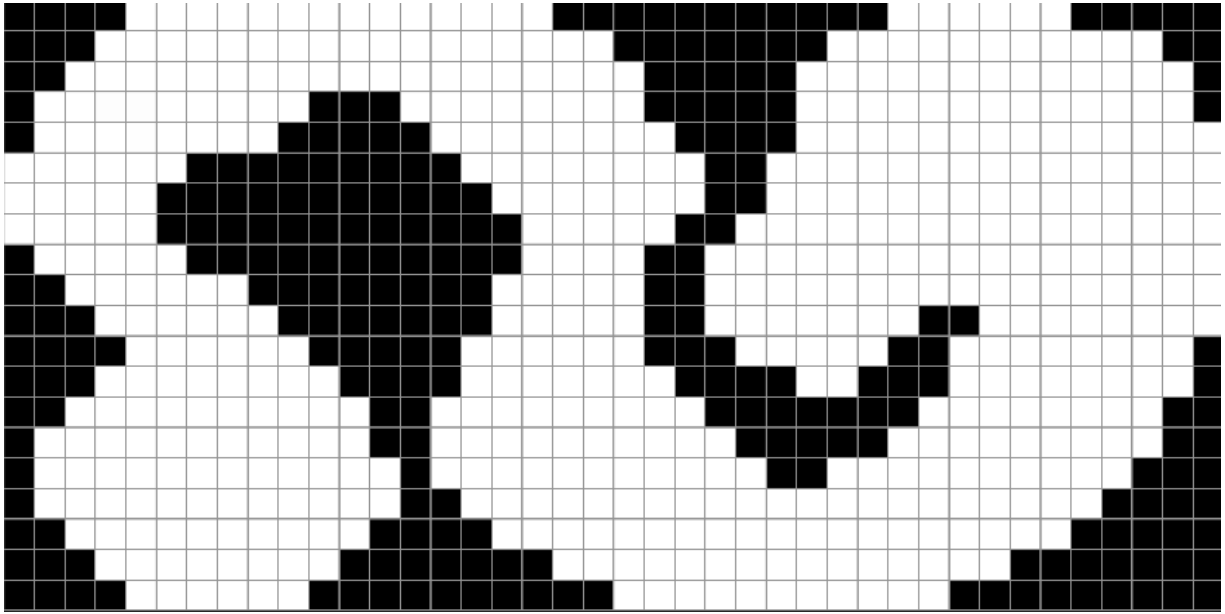
官网：www.qiqiker.com

邮箱：1785275942@qq.com

课程介绍

- 什么是寻路算法？
 - 计算出来游戏中人物角色在游戏场景里运动
 - 现实中地图软件实现路线计算
- 关于Unity自带的Navigation导航系统
 - 通过Navmesh来实现寻路
 - 只能在Unity环境中使用
- 学习A*算法的必要性：
 - 可以在Unity导航系统无法使用时实现寻路功能
 - 很多2D游戏项目移动是基于方格位置设计。
 - 服务器寻路，公共地图NPC行为，服务器角色AI等。
 - 了解Unity寻路原理，解决一些奇特的bug。
- 课程核心内容：
 - 搭建基础的可视化开发环境，可以方便地观察计算流程。

- 通过最直观的方格区块地图来讲解算法实现，便于学习理解。



- 为了加深对A*算法的理解课程中会涉及：
 - 宽度优先搜索(BFS);
 - 迪杰斯特拉算法(Dijkstras);
 - 贪心算法(Greedy);
- 讲解思路流程：
 - 创建一个用于演示算法结果及过程的可视化环境。
 - 实现最基础最暴力的寻路算法。
 - 展示各种算法的问题，对比计算性能差异。
 - 总结对比不同寻路方式优缺点。

搭建寻路基础工程

- 初始工程说明
 - Game视图与Canvas分辨率：2000X1000
 - 设置背景与区块根节点位置
 - 区块Prefab显示信息
- 创建初始地图区块
- 逻辑与显示分离（便于拆分到服务器）
- 区块状态显示控制
- 地图初始化
- 触控事件注册与触发处理
- 通过回调传参控制显示

基础寻路算法实现

- 核心思路：
 - 定义两个集合，一个「等待检测」的队列，另一个是「搜索完成」的队列（开放列表与关闭列表）。
 - 将起点所有相邻节点放入「等待检测」队列，检测队列中是否包含目标节点，如果包含则计算路径。如果不包含说明需要进行下一步的检测，于是从队列中取出一个节点，将该节点标记为「搜索完成」，并将当前进行检测节点的所有相邻节点放入「等待检测」队列。依次循环上述过程，直到找到目标节点为止。



「等待检测」队列



「搜索完成」队列



「等待检测」队列



「搜索完成」队列

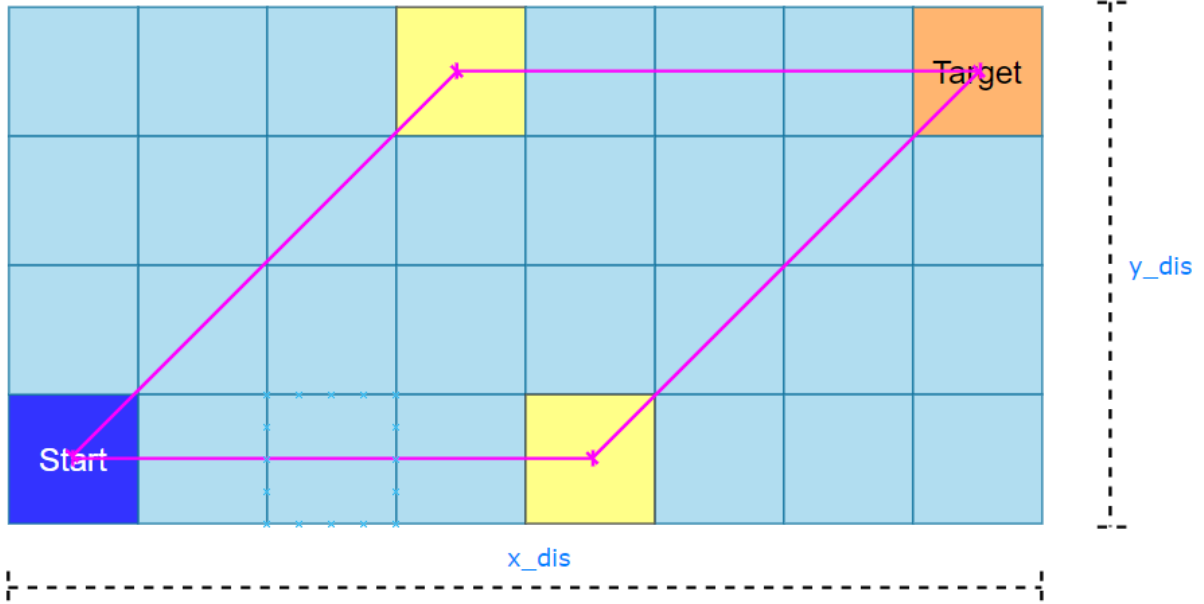


- 通用节点距离运算（基于索引号运算）：

$\text{slope_dis} = \text{MathF.Min}(x_dis, y_dis) \times 1.4f$

$\text{straight_dis} = \text{MathF.Max}(x_dis, y_dis) - \text{MathF.Min}(x_dis, y_dis)$

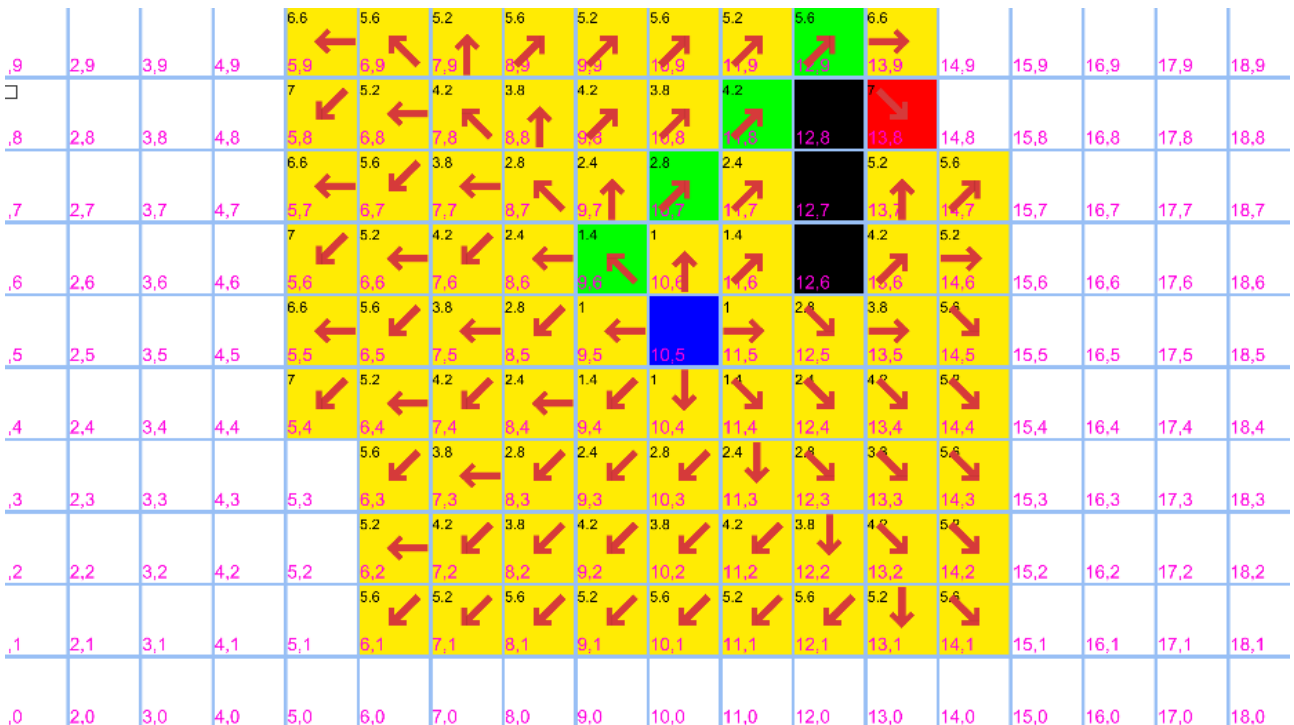
$$\text{dis_sum} = \text{slope_dis} + \text{straight_dis}$$

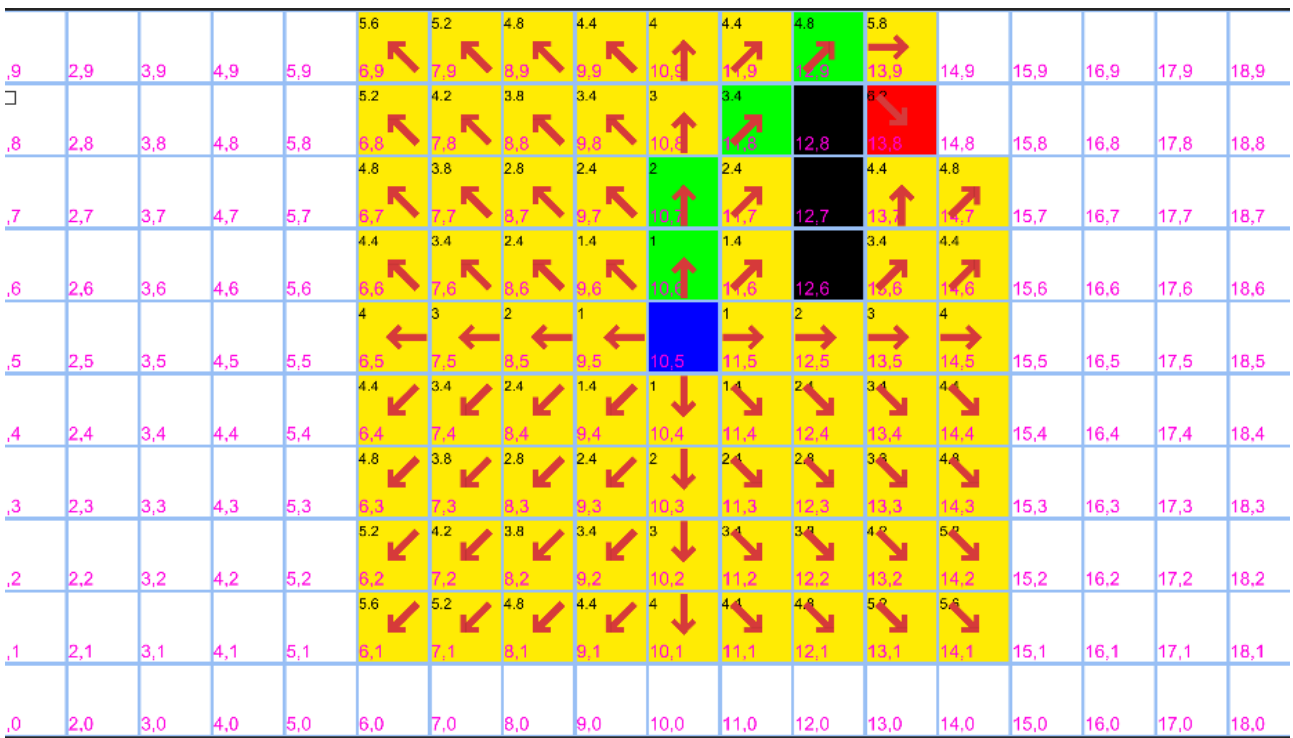


- 曼哈顿距离:轴向距离之和（避免使用浮点运算）： $\text{dis_sum} = \text{x_dis} + \text{y_dis}$

宽度搜索优先算法（BFS）

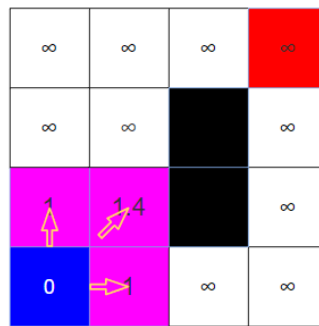
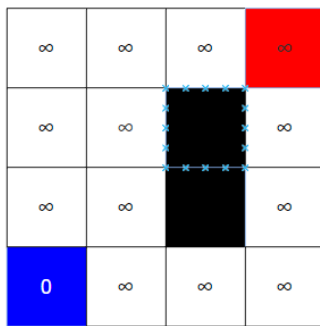
- 不考虑目标的可能位置，彻底地搜索整张图，直到找到目标。
- 采用异步方法展现查找过程,显示寻路结果
- 存在搜索范围过大，性能差，不是最短路径的问题
- 「顺时针」搜索与「正交-斜向」会呈现路径不同的寻路结果





迪克斯特拉算法 (Dijkstras)

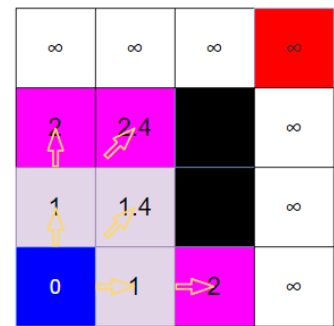
- 查找出来的永远是最短的路径
- 与宽度搜索算法相比，Dijkstras会基于路径总和最小的原则更新前向节点指向。



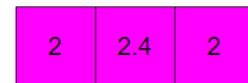
「等待检测」队列



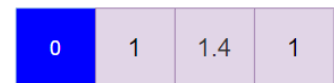
「搜索完成」队列



「等待检测」队列



「搜索完成」队列



- 算法在跑满的情况下能计算出来最优化的路径，但是计算量较大，无法准确定位到优化退出的时机。比如这是一个找到就中止的路径，不是最短路径：

3-9	1-9	2-9	3-9	4-9	5-9	6-9	7-9	8-9	9-9	10-9	11-9	12-9	13-9	14-9	15-9	16-9	17-9	18-9	19-9
3-8	1-8	2-8	3-8	4-8	5-8	6-8	7-8	8-8	9-8	10-8	11-8	12-8	13-8	14-8	15-8	16-8	17-8	18-8	19-8
3-7	1-7	2-7	3-7	4-7	5-7	6-7	7-7	8-7	9-7	10-7	11-7	12-7	13-7	14-7	15-7	16-7	17-7	18-7	19-7
1.2	3.8	3.4	3	3.4	3.8	4.2													
3.6	1.6	2.6	3.6	4.6	5.6	6.6	7-6	8-6	9-6	10-6	11-6	12-6	13-6	14-6	15-6	16-6	17-6	18-6	19-6
3.8	2.8	2.4	2	2.4	2.8	3.8													
3.6	1.6	2.6	3.6	4.6	5.6	6.6	7-5	8-5	9-5	10-5	11-5	12-5	13-5	14-5	15-5	16-5	17-5	18-5	19-5
3.4	2.4	1.4	1	1.4	2.4	3.4													
3.4	1.4	2.4	3.4	4.4	5.4	6.4	7-4	8-4	9-4	10-4	11-4	12-4	13-4	14-4	15-4	16-4	17-4	18-4	19-4
3	2	1		1	2														
3.3	1.3	2.3	3.3	4.3	5.3	6.3	7-3	8-3	9-3	10-3	11-3	12-3	13-3	14-3	15-3	16-3	17-3	18-3	19-3
3.4	2.4	1.4	1	1.4	2.4														
3.2	1.2	2.2	3.2	4.2	5.2	6.2	7-2	8-2	9-2	10-2	11-2	12-2	13-2	14-2	15-2	16-2	17-2	18-2	19-2
	2.8	2.4	2	2.4	2.8														
3-1	1-1	2-1	3-1	4-1	5-1	6-1	7-1	8-1	9-1	10-1	11-1	12-1	13-1	14-1	15-1	16-1	17-1	18-1	19-1
3-0	1-0	2-0	3-0	4-0	5-0	6-0	7-0	8-0	9-0	10-0	11-0	12-0	13-0	14-0	15-0	16-0	17-0	18-0	19-0

- 中断退出，计算完整张地图时可以得出最优路径：

7.2	6.8	6.4	6	6.4	6.8	7.2	7.6	8	8.4	9.4	10.4	11.4	12.4	13.4	14.4	15.4	16.4	17.4	18.4
0-9	1-9	2-9	3-9	4-9	5-9	6-9	7-9	8-9	9-9	10-9	11-9	12-9	13-9	14-9	15-9	16-9	17-9	18-9	19-9
6.2	5.8	5.4	5	5.4	5.8	6.2	6.6	7	8	9	9.999999	11	12	13	14	15	16	17	18
0-8	1-8	2-8	3-8	4-8	5-8	6-8	7-8	8-8	9-8	10-8	11-8	12-8	13-8	14-8	15-8	16-8	17-8	18-8	19-8
5.2	4.8	4.4	4	4.4	4.8	5.2	5.6	6	7.6	8.599999	9.599999	10.6	11.6	12.6	13.6	14.6	15.6	16.6	17.6
0-7	1-7	2-7	3-7	4-7	5-7	6-7	7-7	8-7	9-7	10-7	11-7	12-7	13-7	14-7	15-7	16-7	17-7	18-7	19-7
4.2	3.8	3.4	3	3.4	3.8	4.2	4.6	5	6.2	7.2	8.2	9.2	10.2	11.2	12.2	13.2	14.2	15.2	16.2
0-6	1-6	2-6	3-6	4-6	5-6	6-6	7-6	8-6	9-6	10-6	11-6	12-6	13-6	14-6	15-6	16-6	17-6	18-6	19-6
3.8	2.8	2.4	2	2.4	2.8	3.8	4.8	5.8	6.8	7.8	8.8	9.799999	10.8	11.8	12.8	13.8	14.8	15.8	16.8
0-5	1-5	2-5	3-5	4-5	5-5	6-5	7-5	8-5	9-5	10-5	11-5	12-5	13-5	14-5	15-5	16-5	17-5	18-5	19-5
3.4	2.4	1.4	1	1.4	2.4	3.4	4.4	5.4	6.4	7.4	8.4	9.4	10.4	11.4	12.4	13.4	14.4	15.4	16.4
0-4	1-4	2-4	3-4	4-4	5-4	6-4	7-4	8-4	9-4	10-4	11-4	12-4	13-4	14-4	15-4	16-4	17-4	18-4	19-4
3	2	1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0-3	1-3	2-3	3-3	4-3	5-3	6-3	7-3	8-3	9-3	10-3	11-3	12-3	13-3	14-3	15-3	16-3	17-3	18-3	19-3
3.4	2.4	1.4	1	1.4	2.4	3.4	4.4	5.4	6.4	7.4	8.4	9.4	10.4	11.4	12.4	13.4	14.4	15.4	16.4
0-2	1-2	2-2	3-2	4-2	5-2	6-2	7-2	8-2	9-2	10-2	11-2	12-2	13-2	14-2	15-2	16-2	17-2	18-2	19-2
3.8	2.8	2.4	2	2.4	2.8	3.8	4.8	5.8	6.8	7.8	8.8	9.799999	10.8	11.8	12.8	13.8	14.8	15.8	16.8
0-1	1-1	2-1	3-1	4-1	5-1	6-1	7-1	8-1	9-1	10-1	11-1	12-1	13-1	14-1	15-1	16-1	17-1	18-1	19-1
4.2	3.8	3.4	3	3.4	3.8	4.2	4.6	5	6.2	7.2	8.2	9.2	10.2	11.2	12.2	13.2	14.2	15.2	16.2
0-0	1-0	2-0	3-0	4-0	5-0	6-0	7-0	8-0	9-0	10-0	11-0	12-0	13-0	14-0	15-0	16-0	17-0	18-0	19-0

引入优先级队列PEQue

- 无优先级队列Dijkstras:



- 带优先级队列DijkstrasPro:



- BFS算法增加优先级数据录入，调整遍历顺序

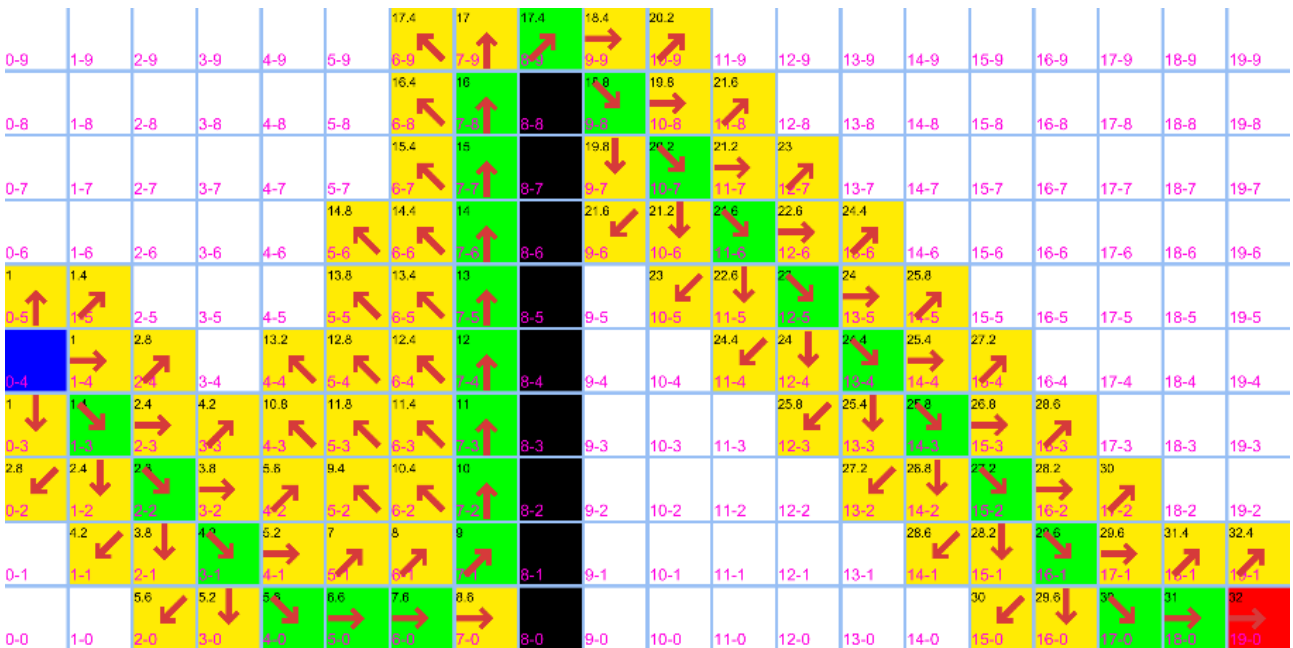
贪心算法 (Greedy)

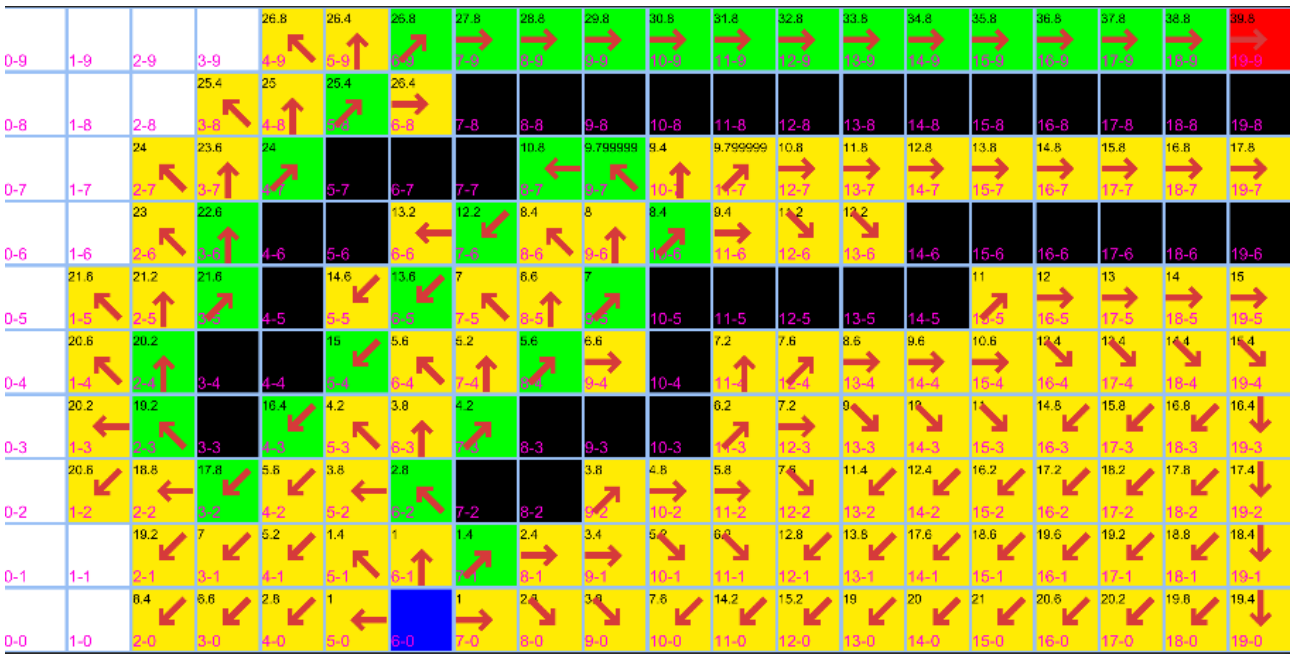
- 不追求全局最优解，只在每一步求得局部最优解的算法。

- 地形开阔，没有太多障碍物时速度极快。



- 地形复杂时，目标点阻挡时，容易走弯路：

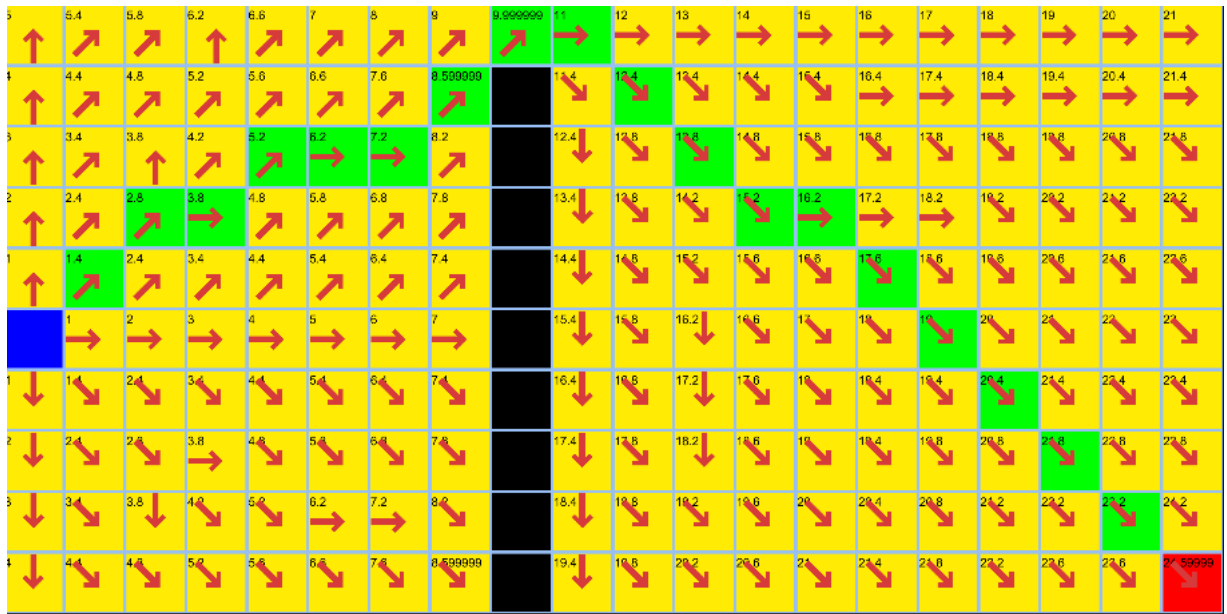




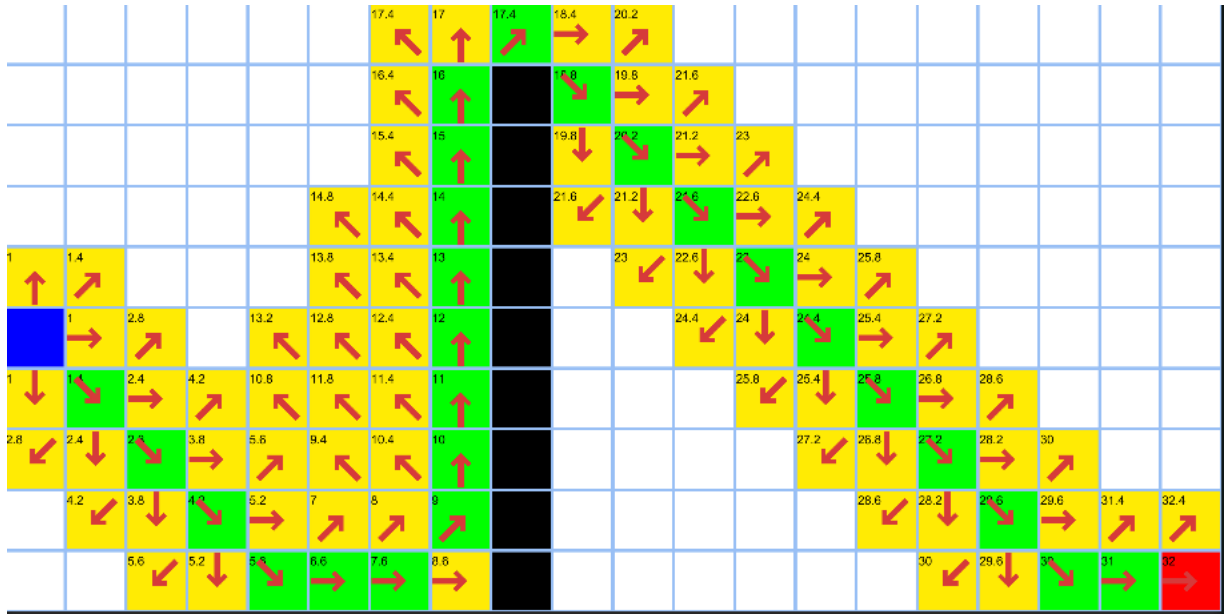
A*算法(Astar)

- 相同地图，相同起始点目标点寻路结果对比：

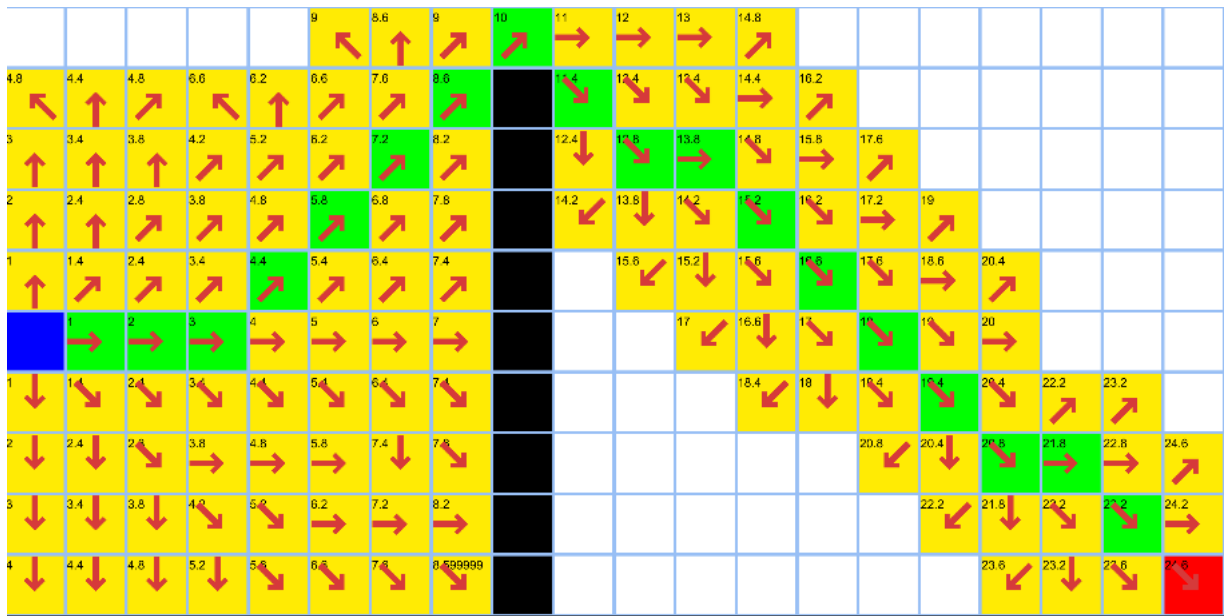
- Dijkstras算法



- Greedy算法



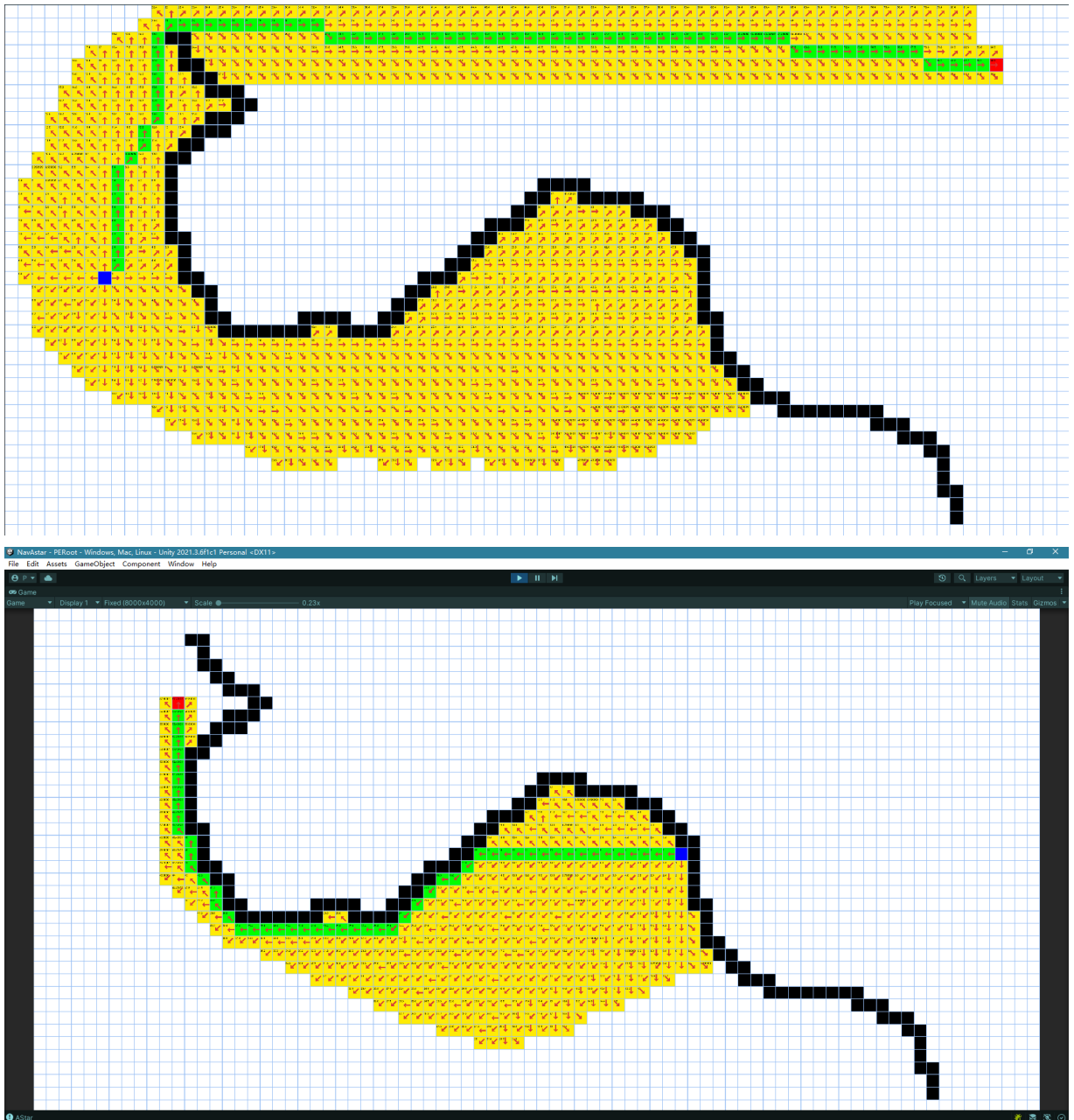
- Astar算法

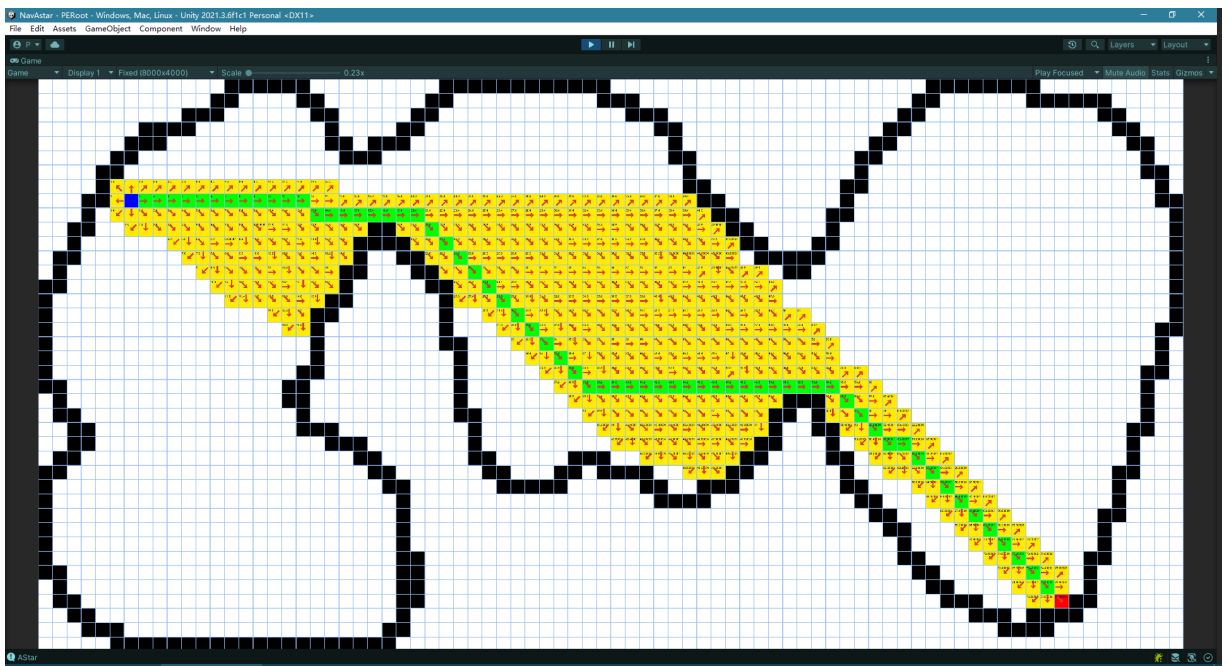
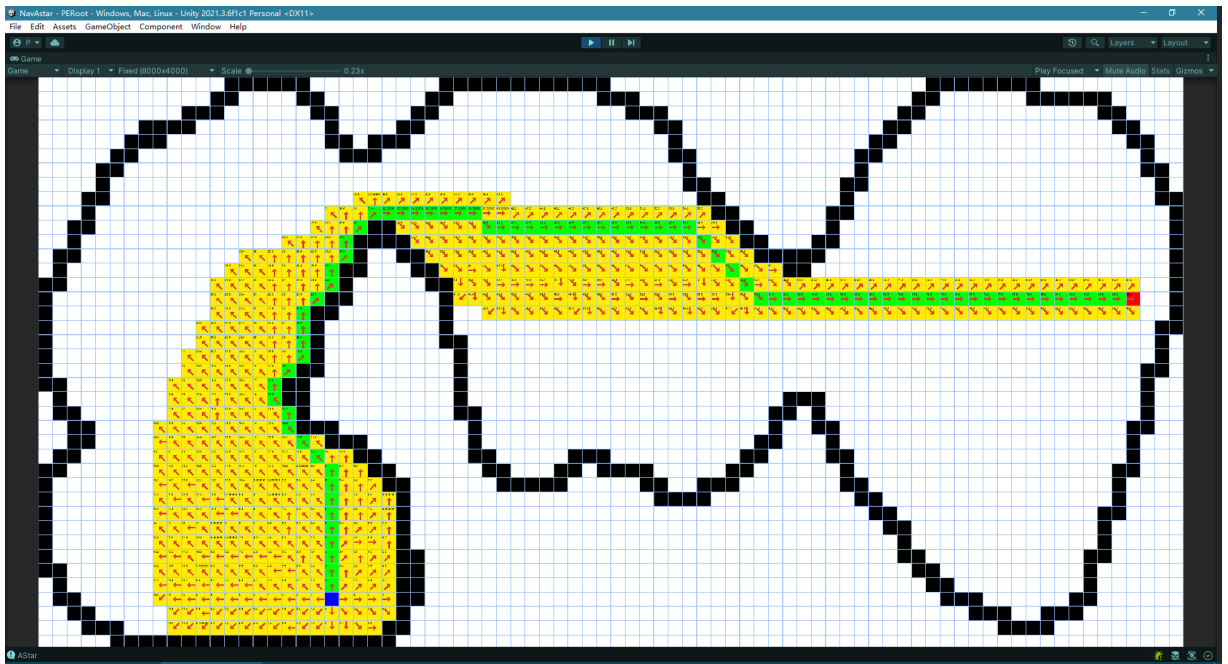


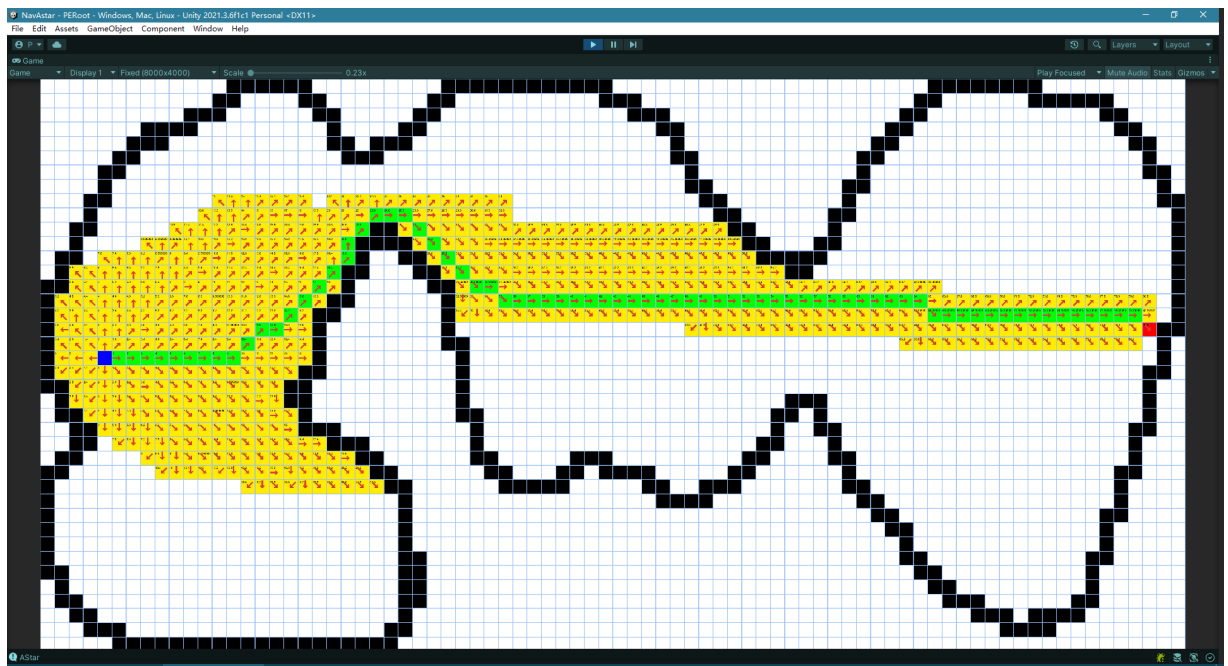
- 综合对比计算性能与寻路效果: (Astar是最优解)

!	BreadthFirst	Len:25.4 Time:0.02203369
!	Dijkstras	Len:24.59999 Time:0.07403564
!	Greedy	Len:32 Time:0.01019287
!	AStar	Len:24.6 Time:0.03964233

- 展示Astar算法动态查询过程







- 维基百科地址：
[广度优先搜索](#)
[戴克斯特拉算法](#)
[贪心算法](#)
[A*搜索算法](#)