深蓝学院
shenlanxueyuan.com

基于搜索的路径规划

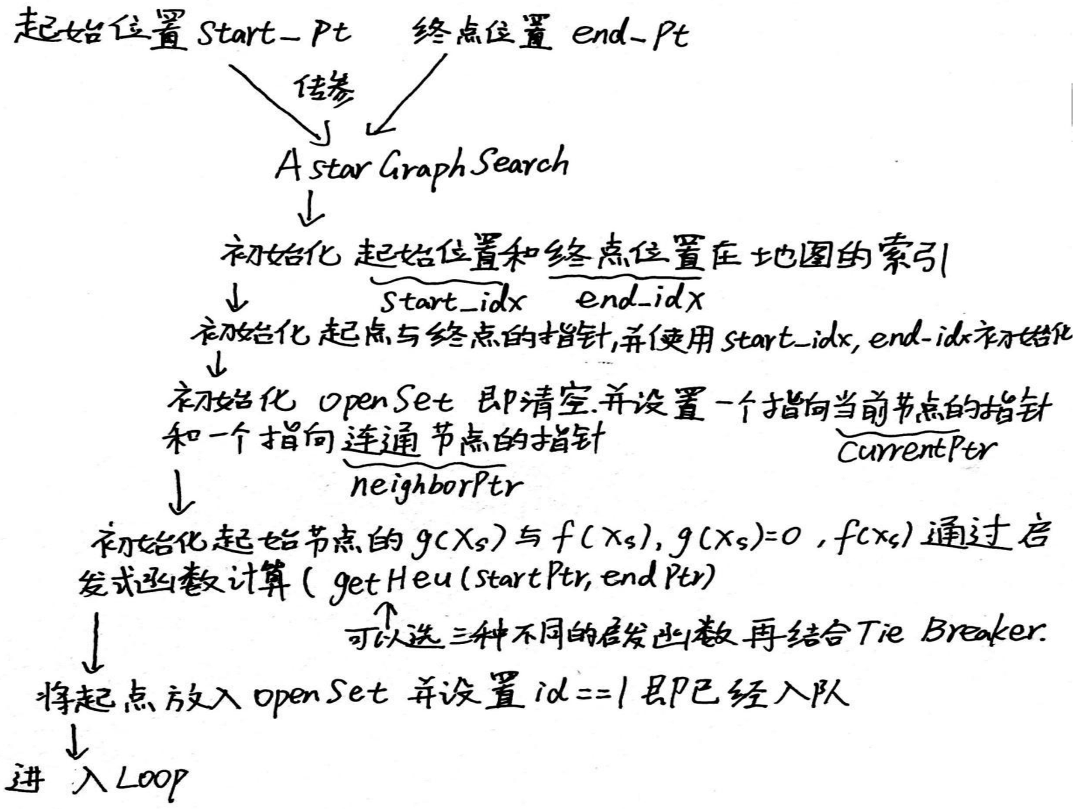
主讲人 JTJ

纲要

- 第一部分：A*
- 第二部分：JPS
- 第三部分：比较A*与JPS

A*

- 算法流程:



起始位置 Start_Pt　　终点位置 end_Pt

传参

A star Graph Search

初始化 起始位置和终点位置在地图的索引

start_idx　　end_idx

初始化起点与终点的指针,并使用 start_idx, end-idx 初始化

初始化 openSet 即清空.并设置一个指向当前节点的指针

和一个指向连通节点的指针

currentPtr

neighborPtr

初始化起始节点的 $g(X_s)$ 与 $f(X_s)$, $g(X_s)=0$, $f(X_s)$ 通过启发式函数计算(getHeu(startPtr, endPtr)

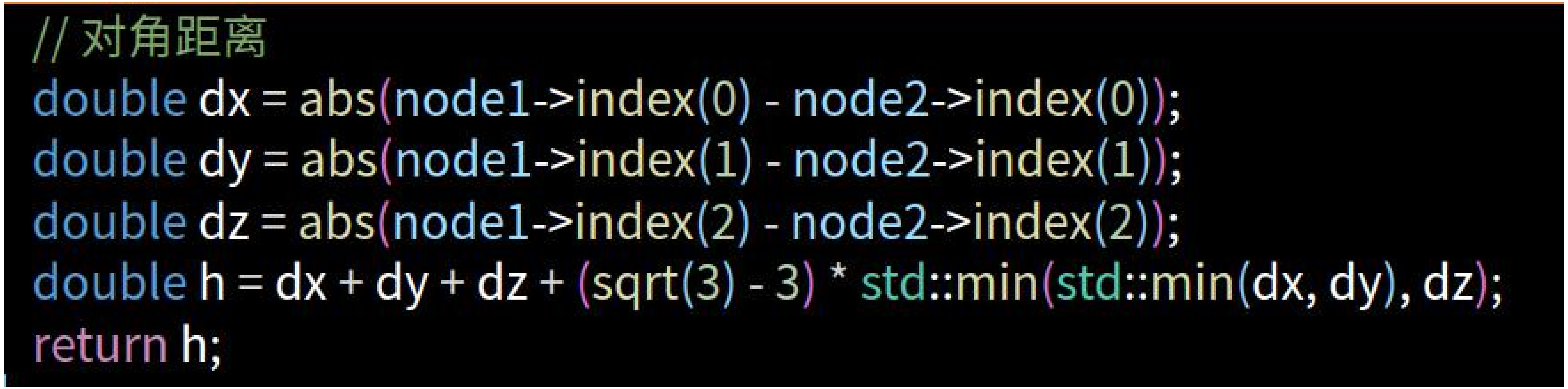
可以选三种不同的启发函数再结合 Tie Breaker.

将起点放入 openSet 并设置 id==1 即已经入队

进入 Loop

A*代码

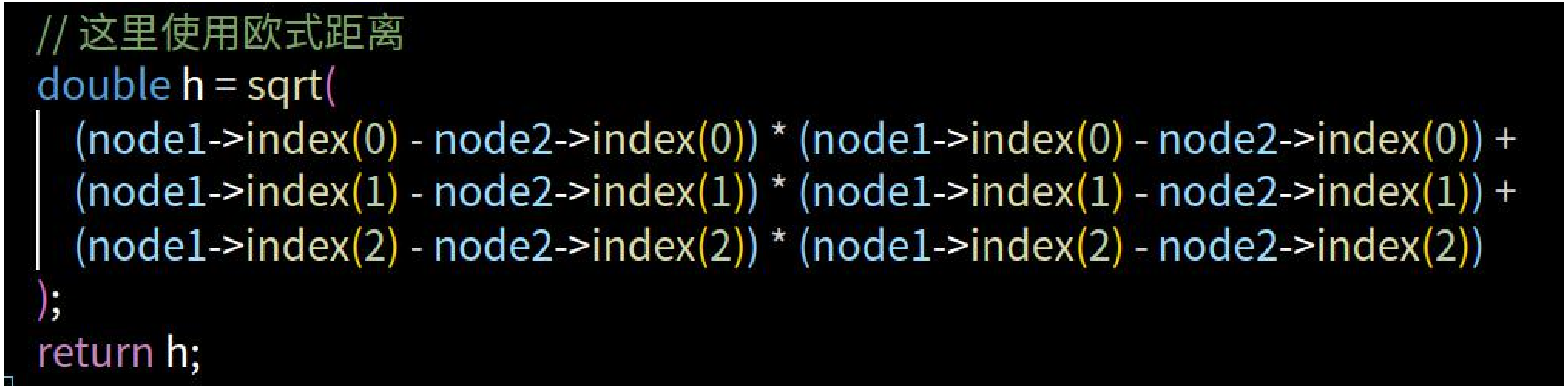
- STEP1：完善启发式函数
- 对角距离启发式函数



```
// 对角距离
double dx = abs(node1->index(0) - node2->index(0));
double dy = abs(node1->index(1) - node2->index(1));
double dz = abs(node1->index(2) - node2->index(2));
double h = dx + dy + dz + (sqrt(3) - 3) * std::min(std::min(dx, dy), dz);
return h;
```

A*代码

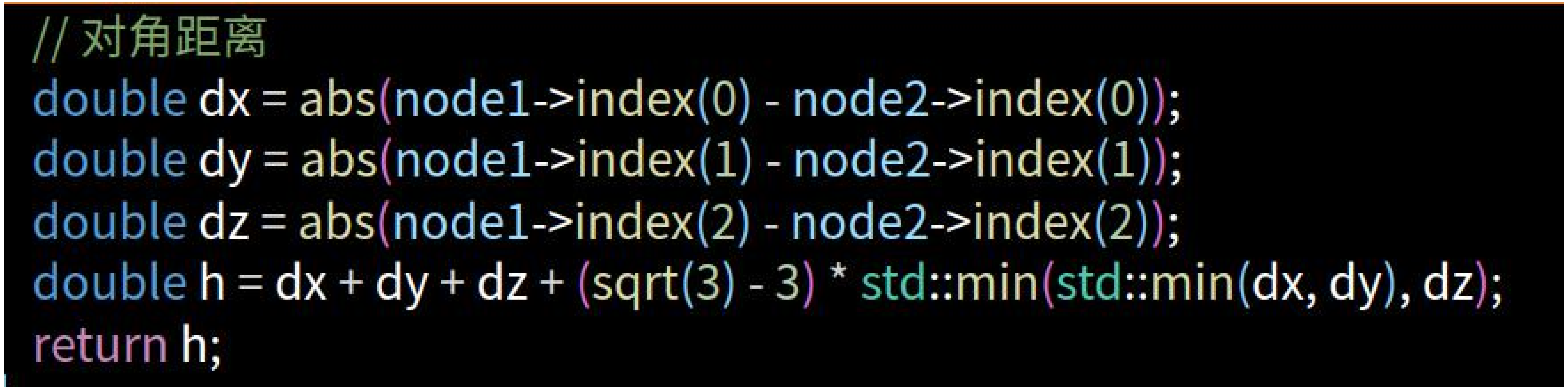
- STEP1：完善启发式函数
- 欧式距离启发式函数



```
// 这里使用欧式距离
double h = sqrt(
    (node1->index(0) - node2->index(0)) * (node1->index(0) - node2->index(0)) +
    (node1->index(1) - node2->index(1)) * (node1->index(1) - node2->index(1)) +
    (node1->index(2) - node2->index(2)) * (node1->index(2) - node2->index(2))
);
return h;
```

A*代码

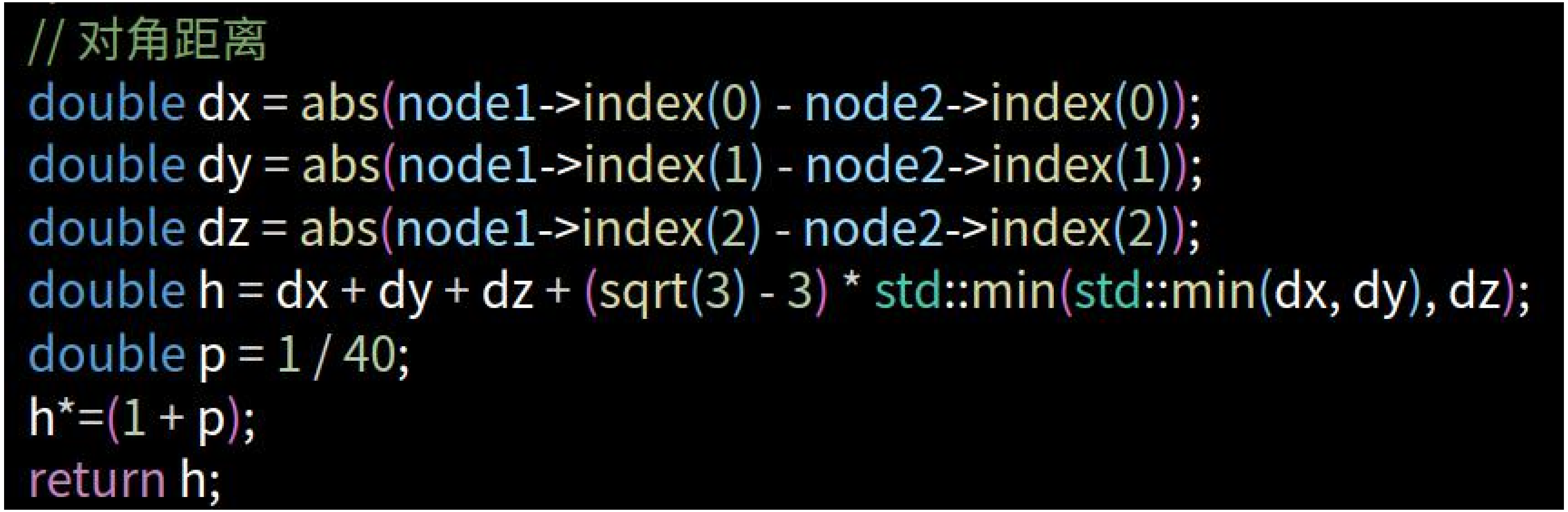
- STEP1：完善启发式函数
- 曼哈顿距离启发式函数



```
// 对角距离
double dx = abs(node1->index(0) - node2->index(0));
double dy = abs(node1->index(1) - node2->index(1));
double dz = abs(node1->index(2) - node2->index(2));
double h = dx + dy + dz + (sqrt(3) - 3) * std::min(std::min(dx, dy), dz);
return h;
```

A*代码

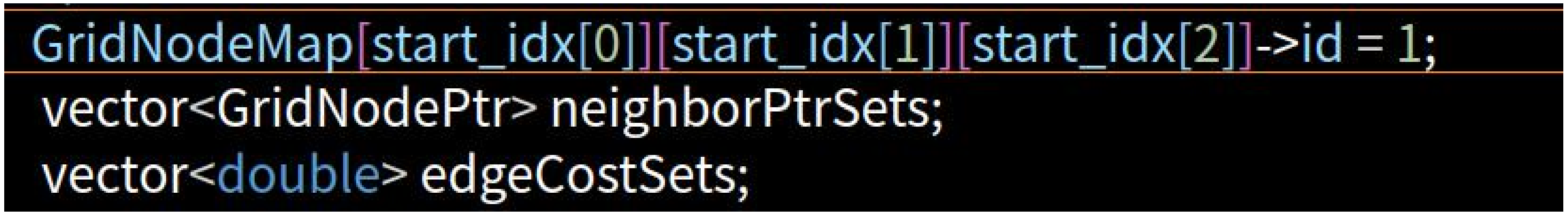
- STEP1：完善启发式函数(加入TieBreaker)
- 以对角距离启发式函数举例：



```
// 对角距离
double dx = abs(node1->index(0) - node2->index(0));
double dy = abs(node1->index(1) - node2->index(1));
double dz = abs(node1->index(2) - node2->index(2));
double h = dx + dy + dz + (sqrt(3) - 3) * std::min(std::min(dx, dy), dz);
double p = 1 / 40;
h*=(1 + p);
return h;
```

A*代码

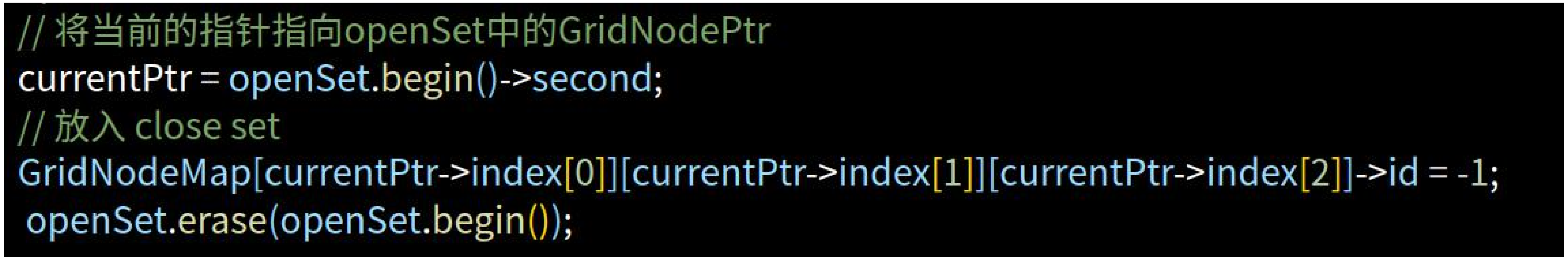
●STEP2：进入循环前需要将起始点放入openSet，也就是id设置为1



```
GridNodeMap[start_idx[0]][start_idx[1]][start_idx[2]]->id = 1;
 vector<GridNodePtr> neighborPtrSets;
 vector<double> edgeCostSets;
```

A*代码

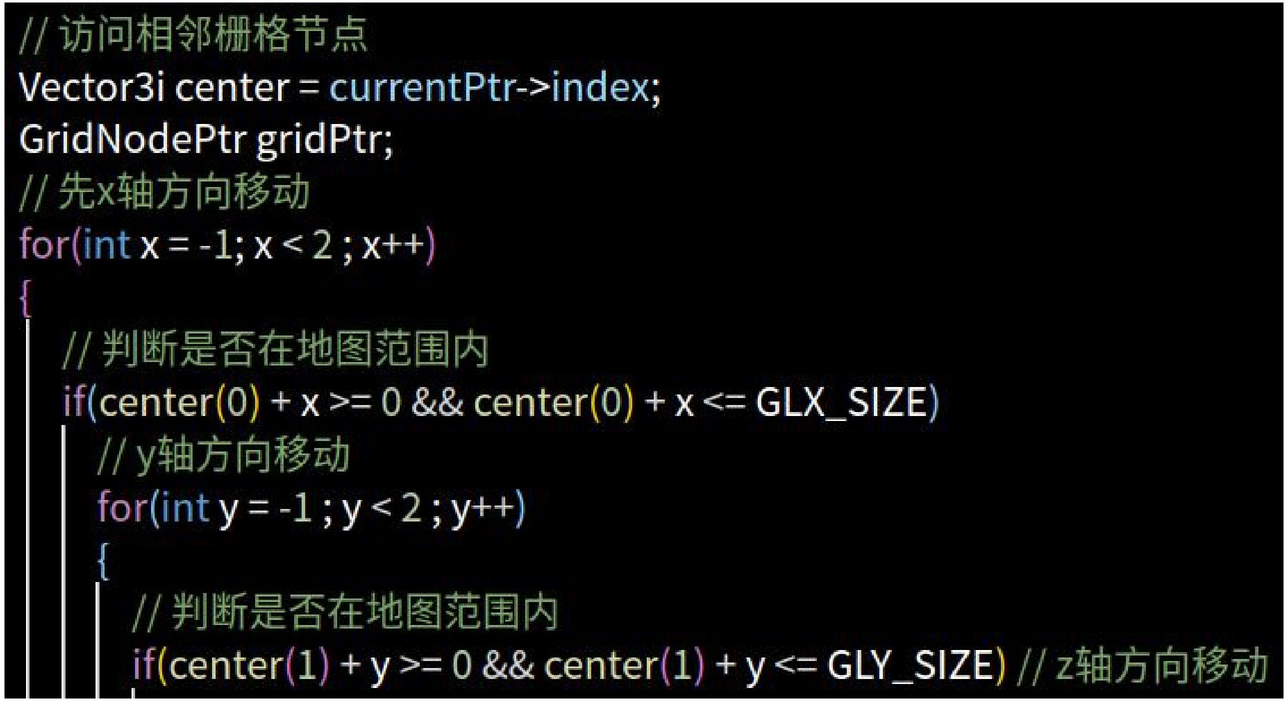
- STEP3：将openSet中最新的节点弹出并放入closeSet



```
// 将当前的指针指向openSet中的GridNodePtr
currentPtr = openSet.begin()->second;
// 放入 close set
GridNodeMap[currentPtr->index[0]][currentPtr->index[1]][currentPtr->index[2]]->id = -1;
 openSet.erase(openSet.begin());
```

A*代码

●STEP4：完善寻找邻居函数AstarGetSucc

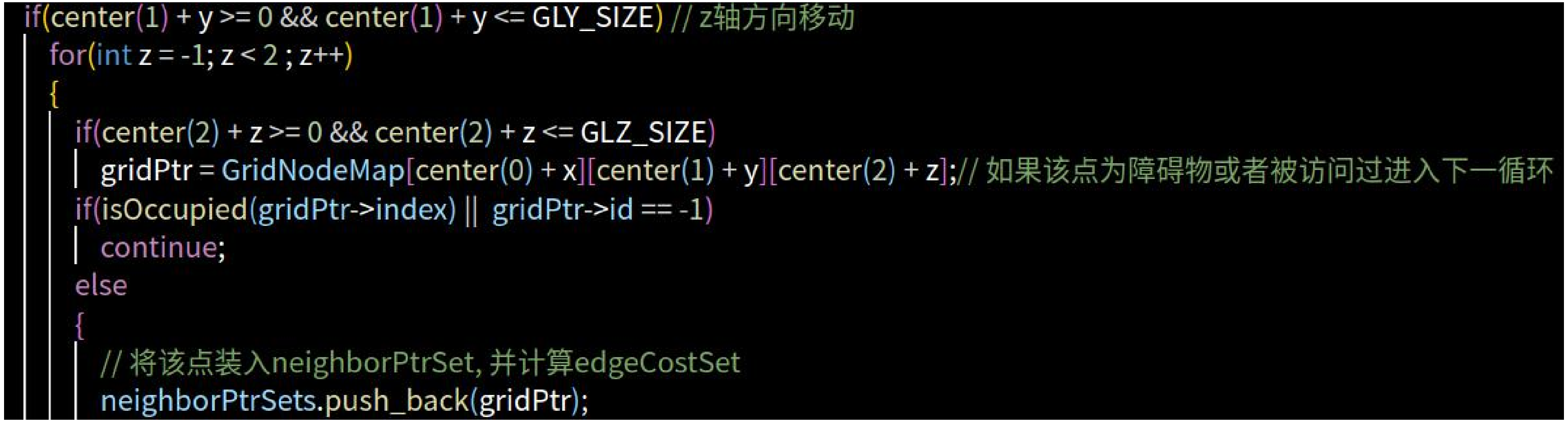


```
// 访问相邻栅格节点
Vector3i center = currentPtr->index;
GridNodePtr gridPtr;
// 先x轴方向移动
for(int x = -1; x < 2 ; x++)
{
    // 判断是否在地图范围内
    if(center(0) + x >= 0 && center(0) + x <= GLX_SIZE)
        // y轴方向移动
        for(int y = -1 ; y < 2 ; y++)
        {
            // 判断是否在地图范围内
            if(center(1) + y >= 0 && center(1) + y <= GLY_SIZE) // z轴方向移动
```

注：这里需要进行边界判断

A*代码

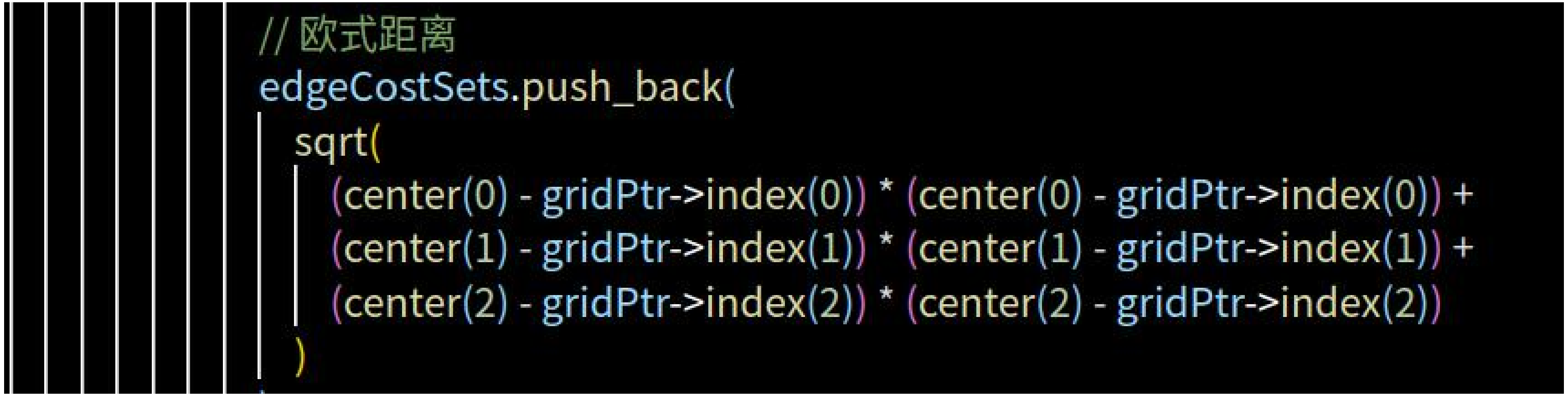
- STEP4：完善寻找邻居函数AstarGetSucc



```
if(center(1) + y >= 0 && center(1) + y <= GLY_SIZE) // z轴方向移动
    for(int z = -1; z < 2 ; z++)
    {
        if(center(2) + z >= 0 && center(2) + z <= GLZ_SIZE)
            gridPtr = GridNodeMap[center(0) + x][center(1) + y][center(2) + z];// 如果该点为障碍物或者被访问过进入下一循环
        if(isOccupied(gridPtr->index) ||  gridPtr->id == -1)
            continue;
        else
        {
            // 将该点装入neighborPtrSet, 并计算edgeCostSet
            neighborPtrSets.push_back(gridPtr);
```

A*代码

●STEP4：完善寻找邻居函数AstarGetSucc

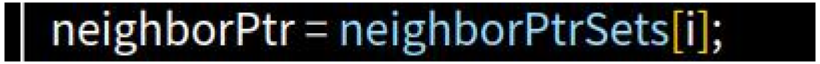


```
// 欧式距离
edgeCostSets.push_back(
    sqrt(
        (center(0) - gridPtr->index(0)) * (center(0) - gridPtr->index(0)) +
        (center(1) - gridPtr->index(1)) * (center(1) - gridPtr->index(1)) +
        (center(2) - gridPtr->index(2)) * (center(2) - gridPtr->index(2))
    )
```

最后使用欧氏距离计算f(n)

A*代码

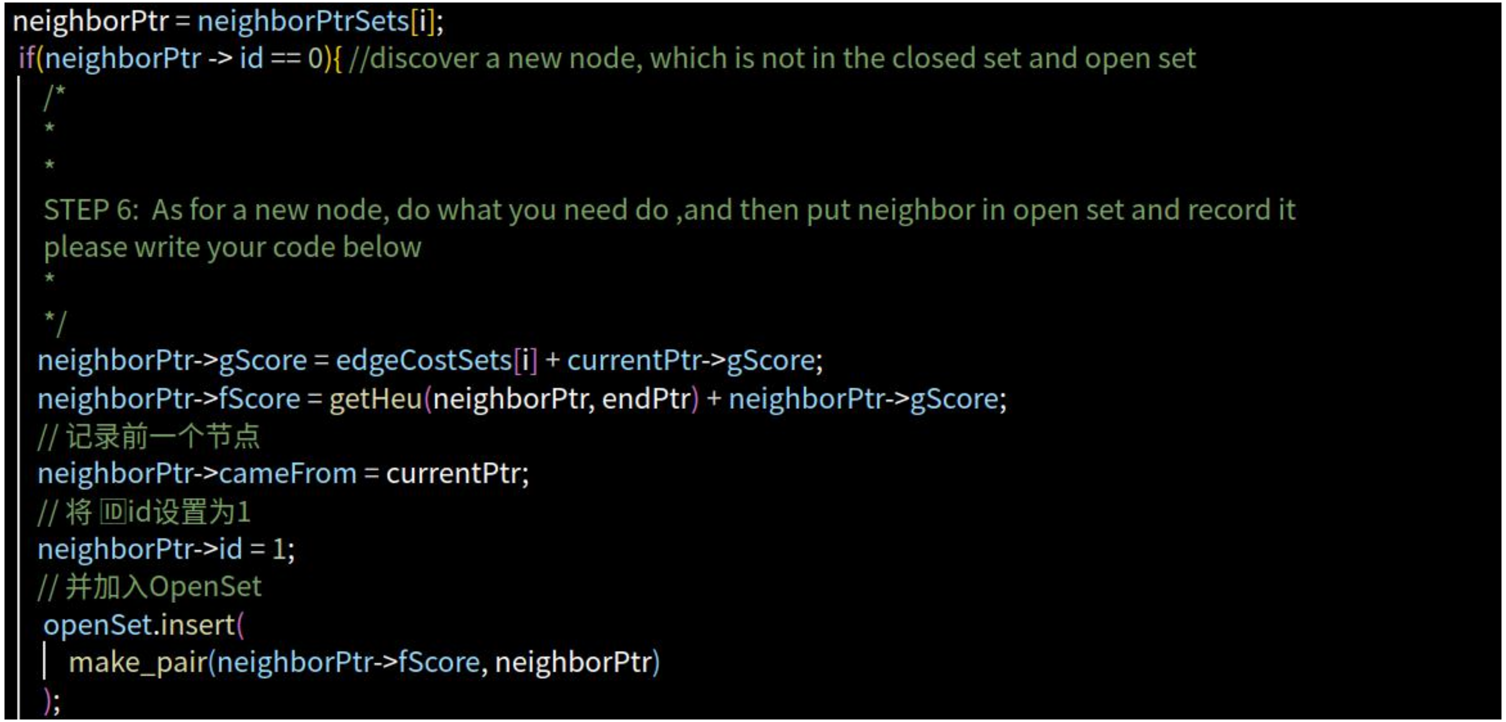
- STEP5：完善循环
- 这里将邻居节点赋值给neighborPtr，用于STEP6的条件判断



```
neighborPtr = neighborPtrSets[i];
```

A*代码

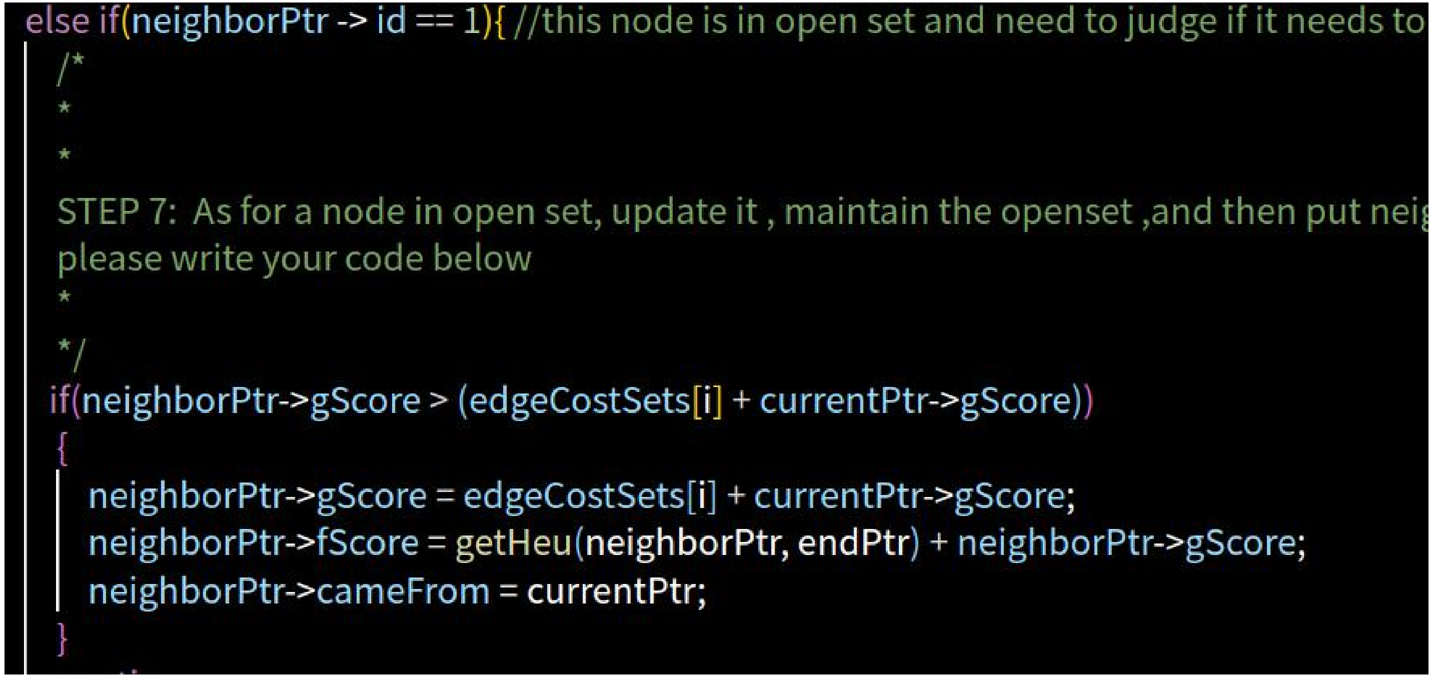
●STEP6：当该节点未被访问过，初始化g(n)



```
neighborPtr = neighborPtrSets[i];
 if(neighborPtr -> id == 0){ //discover a new node, which is not in the closed set and open set
    /*
    *
    *
    STEP 6:  As for a new node, do what you need do ,and then put neighbor in open set and record it
    please write your code below
    *
    */
   neighborPtr->gScore = edgeCostSets[i] + currentPtr->gScore;
   neighborPtr->fScore = getHeu(neighborPtr, endPtr) + neighborPtr->gScore;
   // 记录前一个节点
   neighborPtr->cameFrom = currentPtr;
   // 将 ID id设置为1
   neighborPtr->id = 1;
   // 并加入OpenSet
    openSet.insert(
       make_pair(neighborPtr->fScore, neighborPtr)
    );
```

A*代码

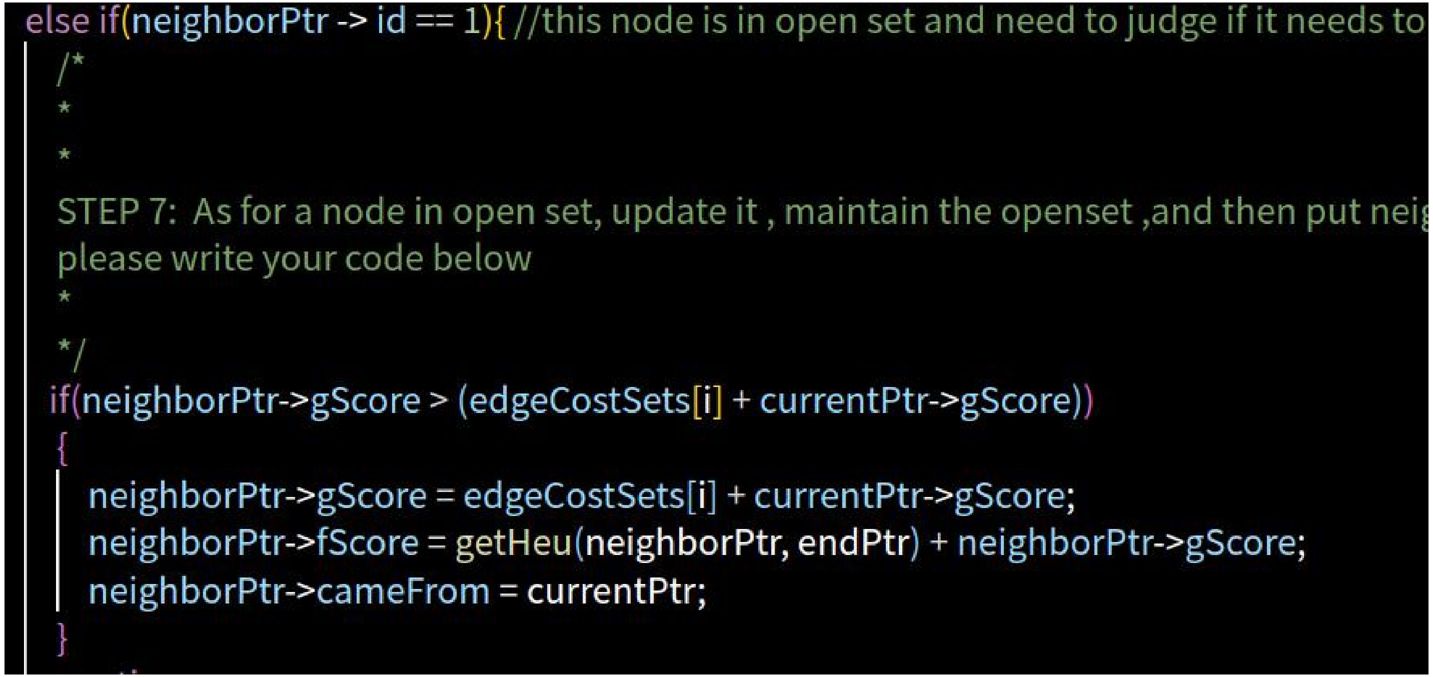
●STEP7：如果该节点被访问过，则取g的最小值



```
else if(neighborPtr -> id == 1){ //this node is in open set and need to judge if it needs to
  /*
  *
  *
  STEP 7:  As for a node in open set, update it , maintain the openset ,and then put neig
  please write your code below
  *
  */
 if(neighborPtr->gScore > (edgeCostSets[i] + currentPtr->gScore))
  {
    neighborPtr->gScore = edgeCostSets[i] + currentPtr->gScore;
    neighborPtr->fScore = getHeu(neighborPtr, endPtr) + neighborPtr->gScore;
    neighborPtr->cameFrom = currentPtr;
  }
```

A*代码

●STEP7：如果该节点被访问过，则取g的最小值



```
else if(neighborPtr -> id == 1){ //this node is in open set and need to judge if it needs to
    /*
    *
    *
    STEP 7:  As for a node in open set, update it , maintain the openset ,and then put neig
    please write your code below
    *
    */
   if(neighborPtr->gScore > (edgeCostSets[i] + currentPtr->gScore))
    {
       neighborPtr->gScore = edgeCostSets[i] + currentPtr->gScore;
       neighborPtr->fScore = getHeu(neighborPtr, endPtr) + neighborPtr->gScore;
       neighborPtr->cameFrom = currentPtr;
    }
```

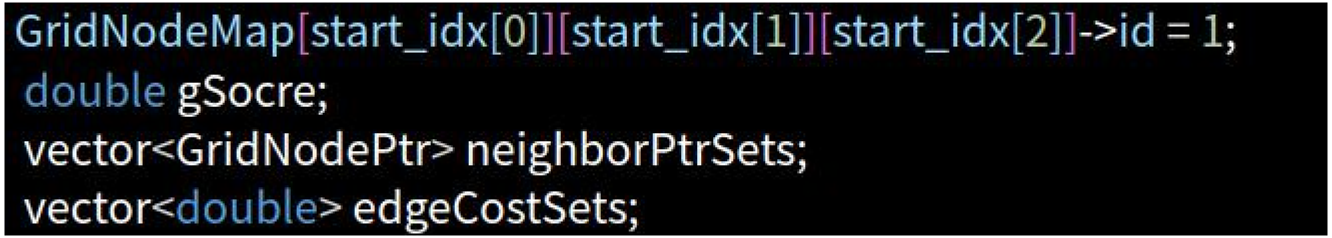
JSP

第二部分：JSP

第三部分：比较A*与JPS

JSP代码

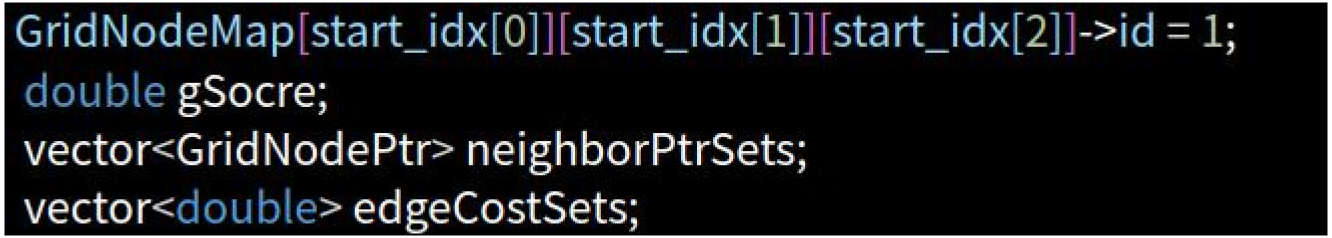
- JSP的实现方式和A*相同，只在寻找邻居节点方法上有所不同。
- STEP1：直接利用A*的启发式函数即可。
- STEP2：再进入循环前起点放入openSet，并设置id为1



```
GridNodeMap[start_idx[0]][start_idx[1]][start_idx[2]]->id = 1;
double gSocre;
vector<GridNodePtr> neighborPtrSets;
vector<double> edgeCostSets;
```

JSP代码

●STEP3：将openSet中最新的节点弹出并放入closeSet

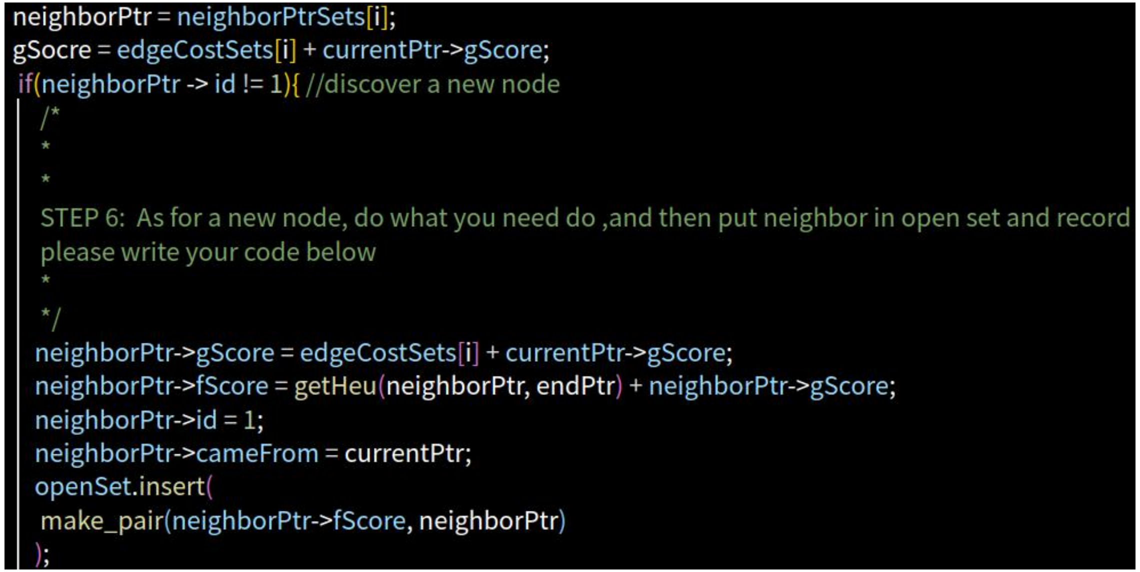


```
GridNodeMap[start_idx[0]][start_idx[1]][start_idx[2]]->id = 1;
 double gSocre;
 vector<GridNodePtr> neighborPtrSets;
 vector<double> edgeCostSets;
```

JSP代码

●STEP4：完善一下循环

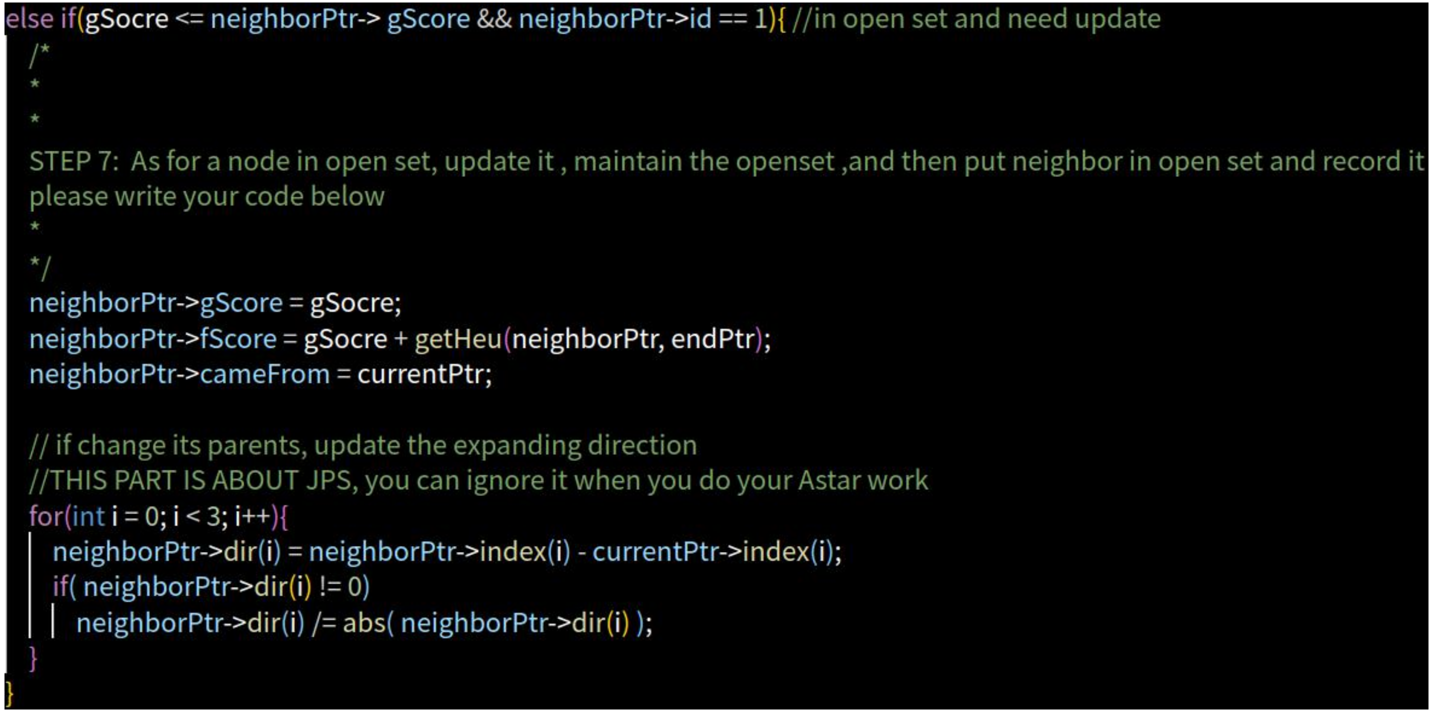
●STEP6：当该节点未被访问过，初始化g(n)



```
neighborPtr = neighborPtrSets[i];
gSocre = edgeCostSets[i] + currentPtr->gScore;
 if(neighborPtr -> id != 1){ //discover a new node
    /*
    *
    *
    STEP 6:  As for a new node, do what you need do ,and then put neighbor in open set and record
    please write your code below
    *
    */
   neighborPtr->gScore = edgeCostSets[i] + currentPtr->gScore;
   neighborPtr->fScore = getHeu(neighborPtr, endPtr) + neighborPtr->gScore;
   neighborPtr->id = 1;
   neighborPtr->cameFrom = currentPtr;
   openSet.insert(
    make_pair(neighborPtr->fScore, neighborPtr)
   );
```

JSP代码

●STEP7：如果该节点被访问过，则取g的最小值



```
else if(gSocre <= neighborPtr-> gScore && neighborPtr->id == 1){ //in open set and need update
    /*
    *
    *
    STEP 7:  As for a node in open set, update it , maintain the openset ,and then put neighbor in open set and record it
    please write your code below
    *
    */
    neighborPtr->gScore = gSocre;
    neighborPtr->fScore = gSocre + getHeu(neighborPtr, endPtr);
    neighborPtr->cameFrom = currentPtr;

    // if change its parents, update the expanding direction
    //THIS PART IS ABOUT JPS, you can ignore it when you do your Astar work
    for(int i = 0; i < 3; i++){
        neighborPtr->dir(i) = neighborPtr->index(i) - currentPtr->index(i);
        if( neighborPtr->dir(i) != 0)
            neighborPtr->dir(i) /= abs( neighborPtr->dir(i) );
    }
}
```

比较A*与JPS

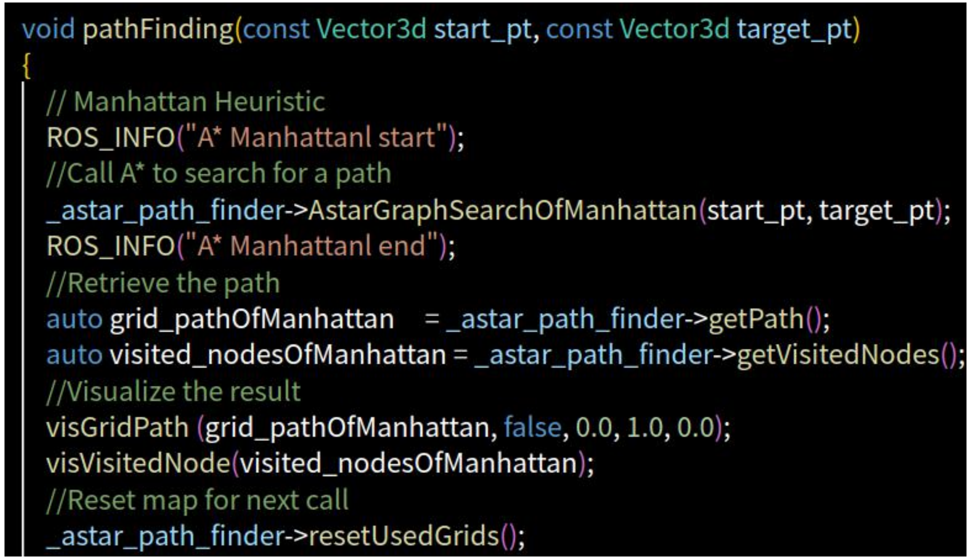
第三部分：比较A*与JPS

比较A*与JPS

我采用的是同一地图、起点、终点，分别使用A*、JPS、和带有Tie Breaker的A*

通过修改demo_node.cpp中的pathFinding函数，仿照A*函数的调用方式，改写A*与JPS。

右图是使用曼哈顿启发函数的A*算法，其他改进算法可以仿照此形式。



```
void pathFinding(const Vector3d start_pt, const Vector3d target_pt)
{
    // Manhattan Heuristic
    ROS_INFO("A* Manhattanl start");
    //Call A* to search for a path
    _astar_path_finder->AstarGraphSearchOfManhattan(start_pt, target_pt);
    ROS_INFO("A* Manhattanl end");
    //Retrieve the path
    auto grid_pathOfManhattan    = _astar_path_finder->getPath();
    auto visited_nodesOfManhattan = _astar_path_finder->getVisitedNodes();
    //Visualize the result
    visGridPath (grid_pathOfManhattan, false, 0.0, 1.0, 0.0);
    visVisitedNode(visited_nodesOfManhattan);
    //Reset map for next call
    _astar_path_finder->resetUsedGrids();
```

比较A*与JPS

- 最终运行数据（同一地图、起点）：
- 不带Tie Breaker，使用不同启发式函数的A*。

Manhattan 运行时间与访问的栅格数目	Euclidean 运行时间与访问的栅格数目	Diagonal Heuristic 运行时间与访问的栅格数目
0.036455ms 23	0.346818ms 587	0.052045ms 32
0.072460ms 49	0.096193ms 111	0.067125ms 68
0.048710ms 16	0.216330ms 197	0.040437ms 24
0.071777ms 26	0.176682ms 206	0.050921ms 27
0.065640ms 31	0.160579ms 120	0.045417ms 26
0.065832ms 23	0.515590ms 543	0.080121ms 49
0.093293ms 51	0.411410ms 578	0.057278ms 37

1-1

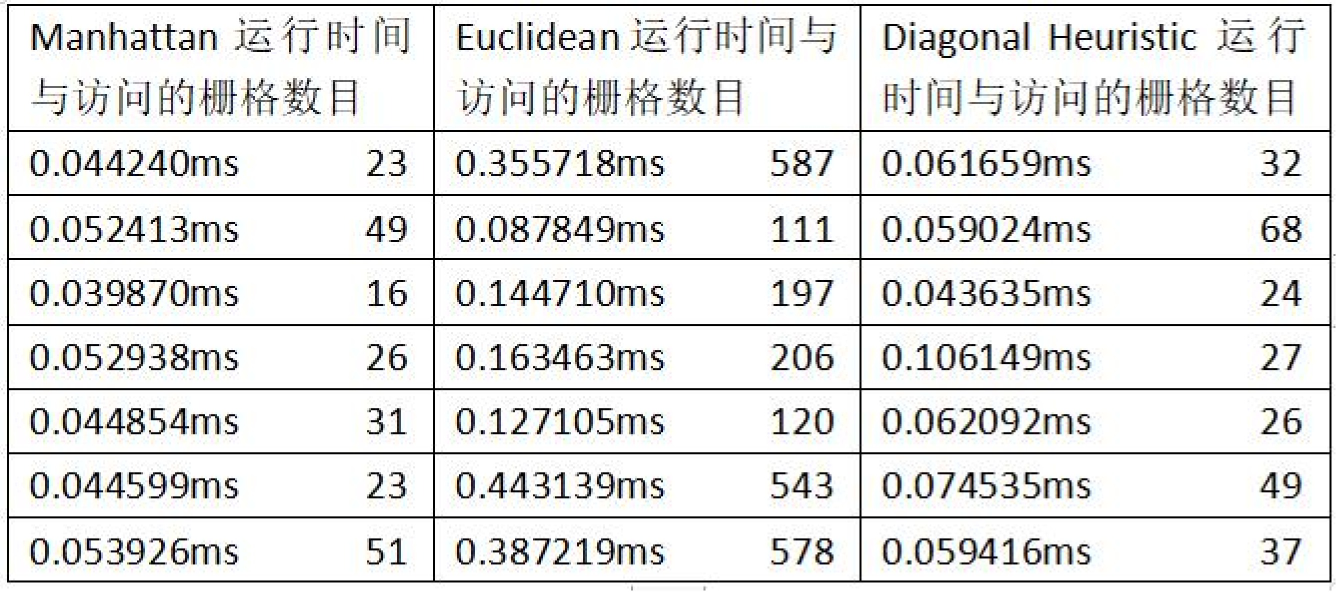
比较A*与JPS

通过观察可以粗略的看出运行时间：EuclideanDiagonalManhattan

堆栈使用空间对比:EuclideanDiagonal HeuristicEuclidean

比较A*与JPS

- 最终运行数据：
- 使用Tie Breaker，使用不同启发式函数的A*。



Manhattan 运行时间与访问的栅格数目	Euclidean 运行时间与访问的栅格数目	Diagonal Heuristic 运行时间与访问的栅格数目
0.044240ms 23	0.355718ms 587	0.061659ms 32
0.052413ms 49	0.087849ms 111	0.059024ms 68
0.039870ms 16	0.144710ms 197	0.043635ms 24
0.052938ms 26	0.163463ms 206	0.106149ms 27
0.044854ms 31	0.127105ms 120	0.062092ms 26
0.044599ms 23	0.443139ms 543	0.074535ms 49
0.053926ms 51	0.387219ms 578	0.059416ms 37

1-2

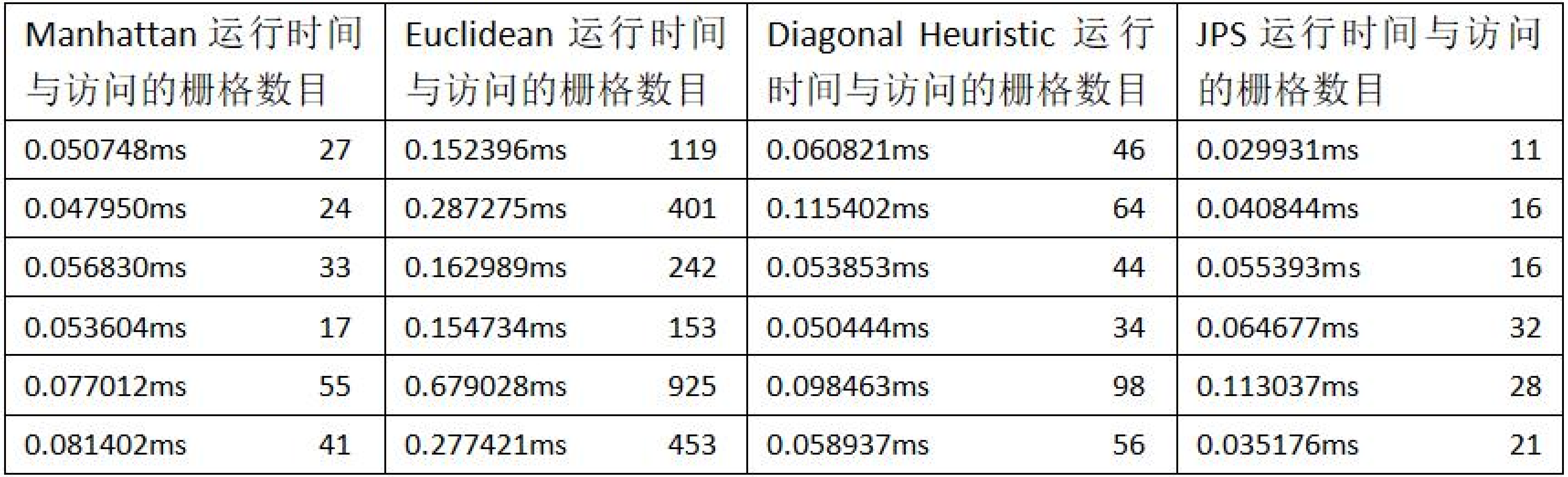
比较A*与JPS

通过图1-1和图1-2中的数据可知,在一些情况下Tie Breaker是可以起到加速作用的，其中对Manhattan和Euclidean加速效果较佳，但是对Diagonal Heuristic加速效果较差，可能是因为参数没有设置好的原因，在此参数环境下Tie Breaker对Diagonal Heuristi起到了副作用，运行时间加长了。

Manhattan和Euclidean的加速可以用Tie Breaker打破了路径的对称性来解释。

比较A*与JPS

- 最终运行数据：
- 不带Tie Breaker，使用不同启发式函数的A*与JPS。

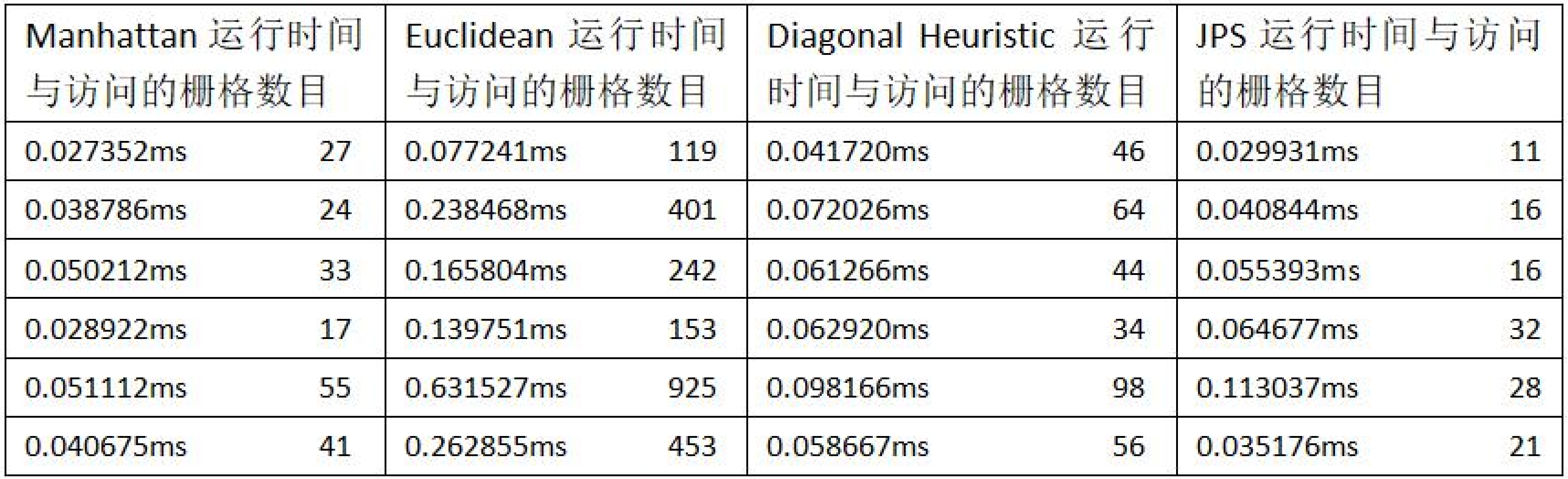


Manhattan 运行时间与访问的栅格数目	Euclidean 运行时间与访问的栅格数目	Diagonal Heuristic 运行时间与访问的栅格数目	JPS 运行时间与访问的栅格数目
0.050748ms 27	0.152396ms 119	0.060821ms 46	0.029931ms 11
0.047950ms 24	0.287275ms 401	0.115402ms 64	0.040844ms 16
0.056830ms 33	0.162989ms 242	0.053853ms 44	0.055393ms 16
0.053604ms 17	0.154734ms 153	0.050444ms 34	0.064677ms 32
0.077012ms 55	0.679028ms 925	0.098463ms 98	0.113037ms 28
0.081402ms 41	0.277421ms 453	0.058937ms 56	0.035176ms 21

图 2-1

比较A*与JPS

- 最终运行数据：
- 使用Tie Breaker，使用不同启发式函数的A*与JPS。



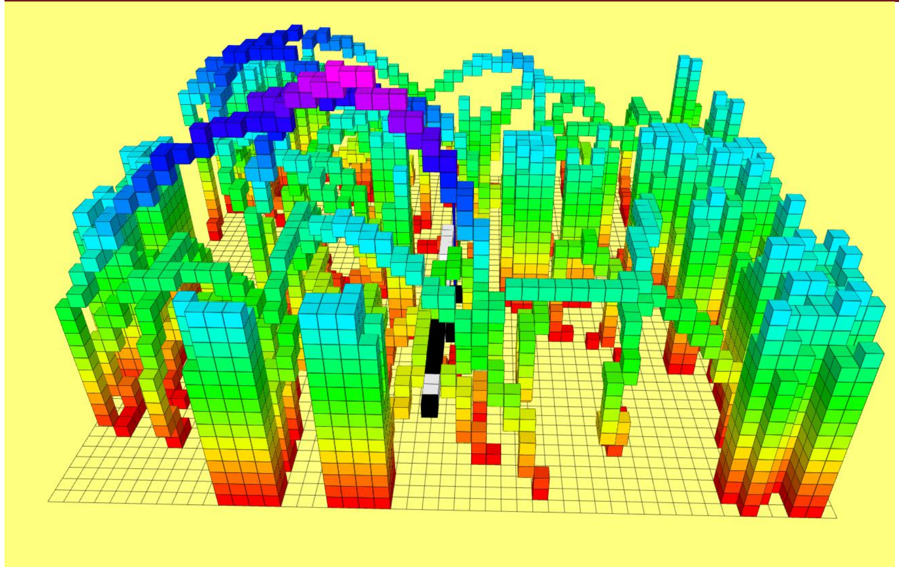
Manhattan 运行时间与访问的栅格数目		Euclidean 运行时间与访问的栅格数目		Diagonal Heuristic 运行时间与访问的栅格数目		JPS 运行时间与访问的栅格数目	
0.027352ms	27	0.077241ms	119	0.041720ms	46	0.029931ms	11
0.038786ms	24	0.238468ms	401	0.072026ms	64	0.040844ms	16
0.050212ms	33	0.165804ms	242	0.061266ms	44	0.055393ms	16
0.028922ms	17	0.139751ms	153	0.062920ms	34	0.064677ms	32
0.051112ms	55	0.631527ms	925	0.098166ms	98	0.113037ms	28
0.040675ms	41	0.262855ms	453	0.058667ms	56	0.035176ms	21

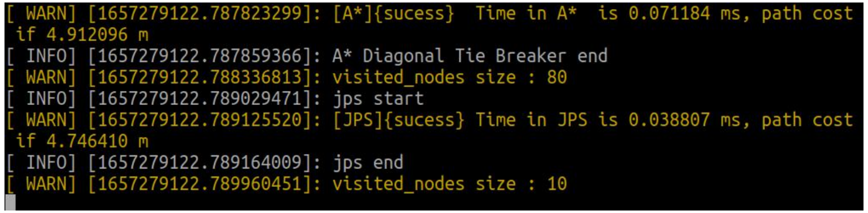
图 2-2

比较A*与JPS

- 观察结果：
- 通过图2-1和图2-2数据对比可知，JPS在某些情况下的速度是比没有使用Tie Breaker的哈密顿A*算法要快的，在障碍物较多的地方JPS算法的表现要比Euclidean和Diagonal Heuristi好得多，但是在较为空旷的地区则恰恰相反。

比较A*与JPS



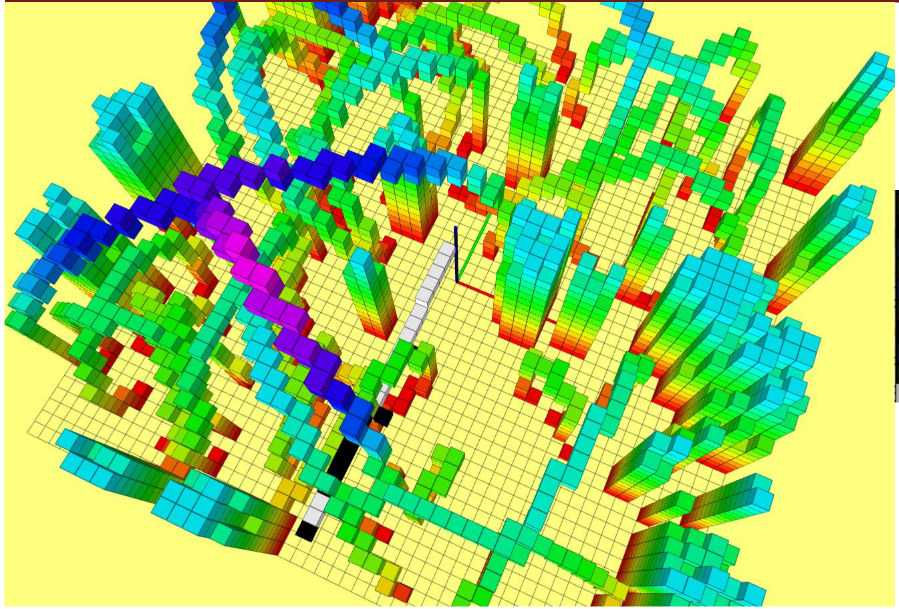


```
[ WARN] [1657279122.787823299]: [A*]{sucess}  Time in A*  is 0.071184 ms, path cost
 if 4.912096 m
[ INFO] [1657279122.787859366]: A* Diagonal Tie Breaker end
[ WARN] [1657279122.788336813]: visited_nodes size : 80
[ INFO] [1657279122.789029471]: jps start
[ WARN] [1657279122.789125520]: [JPS]{sucess} Time in JPS is 0.038807 ms, path cost
 if 4.746410 m
[ INFO] [1657279122.789164009]: jps end
[ WARN] [1657279122.789960451]: visited_nodes size : 10
```

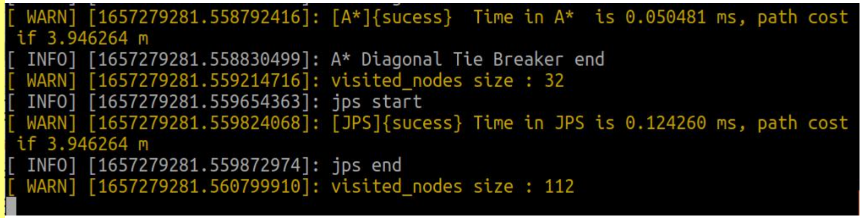
上图为运行结果，可以得知在障碍物较多的地方JPS算法的速度比A*要快。

其中黑色为JPS的路径白色为 使用TieBreaker和对角启发式函数的A*算法的路径

比较A*与JPS



其中黑色为JPS的路径白色为 使用TieBreaker和对角启发式函数的A*算法的路径



```
[ WARN] [1657279281.558792416]: [A*]{sucess}  Time in A*  is 0.050481 ms, path cost
 if 3.946264 m
[ INFO] [1657279281.558830499]: A* Diagonal Tie Breaker end
[ WARN] [1657279281.559214716]: visited_nodes size : 32
[ INFO] [1657279281.559654363]: jps start
[ WARN] [1657279281.559824068]: [JPS]{sucess} Time in JPS is 0.124260 ms, path cost
 if 3.946264 m
[ INFO] [1657279281.559872974]: jps end
[ WARN] [1657279281.560799910]: visited_nodes size : 112
```

通过上图可以观察得出在空旷区域，JPS的速度比A*要慢上很多

深蓝学院
shenlanxueyuan.com

感谢各位聆听！
Thanks for Listening