|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  | 版 本： | V1.0 | |  |
|  | |  | |  | 密 级： | 内部 | |  |
|  | |  | |  | 总页数： | 10 | |  |
|  | | | | | | | | |
| 物联网实验室技术文档  路径规划服务软件  MAPF算法概要设计说明 | | | | | | | | |
|  | | | | | | | | |
|  | **单 位** | |  | | | |  | |
|  | **编 写** | | 吕 昱 20230221 | | | |  | |
|  | **参与人** | | 曹松涛 | | | |  | |
|  | **校 对** | |  | | | |  | |
|  | **审 核** | |  | | | |  | |
|  | **批 准** | |  | | | |  | |
|  | | | | | | | | |
|  | | | | | | | | |
| **易通星云（北京）科技发展有限公司** | | | | | | | | |
|  | | | | | | | | |

**版本历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 日期 | 备注 |
| V1.0/初稿 | 吕昱 |  | 2023-02-21 | 第一个版本 |
|  |  |  |  |  |

**目录**

[1 范围 2](#_Toc9707)

[1.1 标识 2](#_Toc1991)

[1.2 概述 2](#_Toc30384)

[1.2.1 系统概述 2](#_Toc10345)

[1.2.2 软件概述 3](#_Toc9232)

[1.3 文档概述 3](#_Toc6682)

[2 引用文档 3](#_Toc27307)

[3 术语和定义 5](#_Toc22485)

[4 概要设计 6](#_Toc20209)

[4.1 算法概述 6](#_Toc23619)

[4.2 需突破关键算法技术及进度安排 7](#_Toc4364)

[4.3 算法核心技术点 8](#_Toc32355)

[4.3.1 静态路径规划 8](#_Toc28028)

[4.3.2 动态路径规划 8](#_Toc11334)

[4.4 部署视图 8](#_Toc30548)

[5 非功能设计 8](#_Toc15579)

[5.1 性能指标测试 8](#_Toc357)

[5.2 扩展性 9](#_Toc6362)

[5.3 设计约束 9](#_Toc30188)

[6 需求可追踪性 9](#_Toc15444)

[6.1 从软件需求规格到概要设计的追踪关系 9](#_Toc14454)

[6.2 从软件概要设计到需求规格的追踪关系 9](#_Toc23920)

**路径规划服务软件MAPF算法概要设计说明**

# 范围

## 标识

a. 标识号：IOT.MAPF/ALG/1.0。

b. 标题：路径规划服务软件MAPF算法概要设计说明。

c. 本概要设计说明书适用于路径规划服务器软件。

## 概述

### 系统概述

本项目系统包含现场智能硬件设备、物联网软件、路径规划服务器软件、监控管理（后端）软件如图 1-1所示。

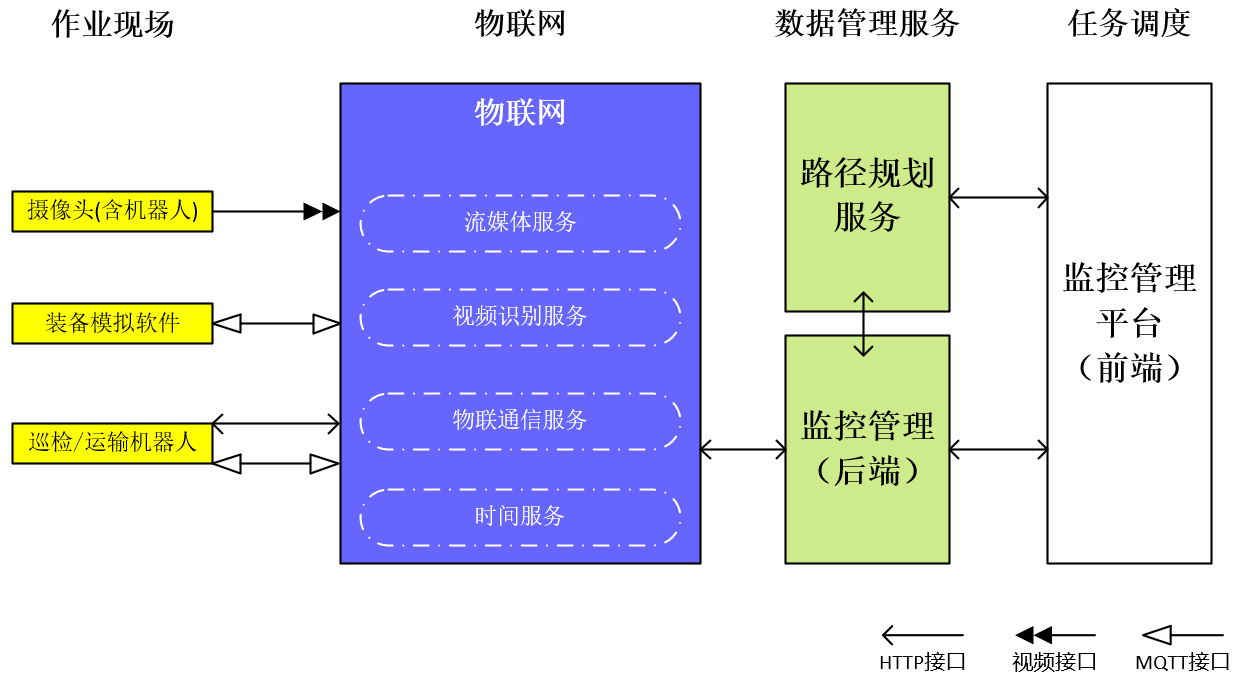


图 1-1系统组成图

### 软件概述

MAPF算法是路径规划服务软件的核心组成部份，它由C++实现，具有极高的运算性能。路径规划服务软件封装MAPF算法，对外部提供微服务接口，满足外部调用需要。路径规划服务软件架构如图 1-2。

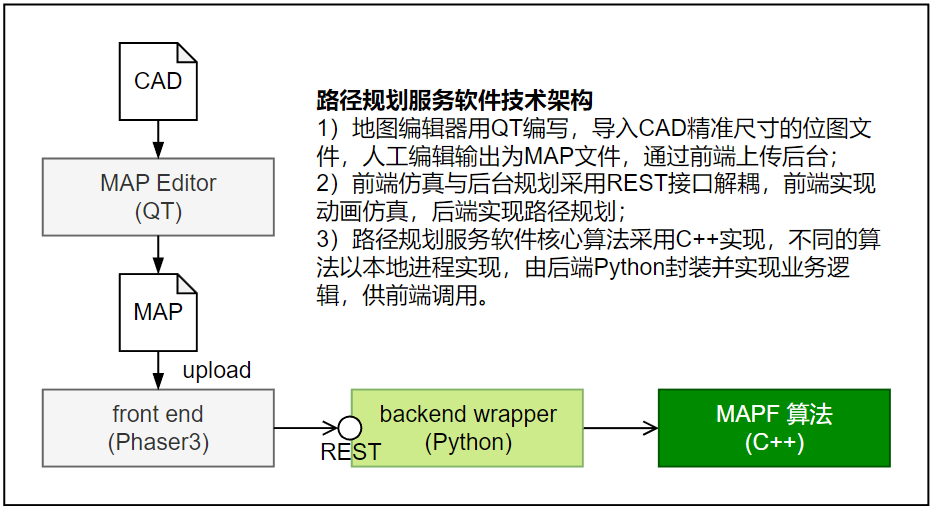


图 1-2路径规划服务软件架构图

## 文档概述

本文档重点描述路径规划服务软件的MAPF算法，如图 1-2中深绿色部份。

# 引用文档

[1] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-Based Search for Optimal Multi-Agent Pathfinding. Artificial Intelligence, 219:40–66, 2015.

[2] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 740–746, 2015.

[3] Eli Boyarski, Ariel Felner, Guni Sharon, and Roni Stern. Don't Split, Try to Work It Out: Bypassing Conflicts in Multi-Agent Pathfinding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 47-51, 2015.

[4] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 83-87, 2018.

[5] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 442-449, 2019.

[6] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pages 6087-6095, 2019.

[7] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. CoRR, abs/2103.07116, 2021.

[8] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 193-201, 2020.

[9] Han Zhang, Jiaoyang Li, Pavel Surynek, Sven Koenig, and T. K. Satish Kumar. Multi-Agent Path Finding with Mutex Propagation. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 323-332, 2020.

[10] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 279-283, 2019.1.

[11] Stern R, Sturtevant N, Felner A, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks[C]//Proceedings of the International Symposium on Combinatorial Search, pages 151-158, . 2019, 10.

[12] Wen L, Liu Y, Li H. CL-MAPF: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints[J]. Robotics and Autonomous Systems, 2022, 150: 103997.

[13] Andreychuk A, Yakovlev K, Surynek P, et al. Multi-agent pathfinding with continuous time[J]. Artificial Intelligence, 2022, 305: 103662.

[14] Madar N, Solovey K, Salzman O. Leveraging Experience in Lifelong Multi-Agent Pathfinding[C]//Proceedings of the International Symposium on Combinatorial Search. 2022, 15(1): 118-126.

[15] Okumura K, Machida M, Défago X, et al. Priority inheritance with backtracking for iterative multi-agent path finding[J]. Artificial Intelligence, 2022, 310: 103752.

[16] Song S, Na K I, Yu W. Anytime Lifelong Multi-Agent Pathfinding in Topological Maps[J]. IEEE Access, 2023.

# 术语和定义

表3-1 术语和定义表

|  |  |
| --- | --- |
| 术语 | 定义 |
| MAPF | 多智能体路径查找（Multi-agent pathfinding），是对一组智能体从指定的起点位置到指定的目标位置的无碰撞路径规划，该问题一个优化问题。MAPF与图论的最短路径问题密切相关，当前已经提出了几种算法来解决MAPF问题。 |
| CBS | 基于冲突的搜索（Conflict-based search），是当前最先进的MAPF算法。 |
| STA\* | 时空A星（Space time A star）搜索算法是路径查找和图形遍历中使用的最好和流行的启发式路径搜索算法技术。 |

# 概要设计

## 算法输入与输出

【输入】

格式待定，内容为：

（0）算法类型，比如PIBT，CBS，RL（强化学习）。

（1）包含要参与路径规划的Agent数量。

（2）每个Agent的起点和终点。

（3）此前已经规划的若干Agent正在执行的路径（走过的路径点不需要）。

注意，（3）仅是针对在线实时规划的参数，不提供的话缺省为静态规划，如果给出这个参数，本算法输出（1）（2）中要求的路径会规避（3）中的路径冲突点。

【输出】

如果是PIBT类型算法，则输出格式如下，字符串。下面示例含义为从0时刻到74时刻（行）4个Agent（列）的网格位置。

var pibt\_solution=`

0:(21,20),(24,34),(6,3),(9,13),

1:(22,20),(24,33),(7,3),(8,13),

……

71:(35,19),(42,8),(36,44),(6,7),

72:(35,19),(42,8),(37,44),(6,7),

73:(35,19),(42,8),(38,44),(6,7),

74:(35,19),(42,8),(39,44),(6,7)

`

如果是CBS类型算法，则输出格式如下，字符串。下面示例含义为从0时刻，有3个Agent，分别是pts0、pts1、pts2的逐时刻的路径点。

var cbs\_solution = `

"pts0": [[8,8],[8,8],[8,8],[8,8],[8,16],[8,24],[8,24],[8,32],[8,40],[8,40],[8,40],[8,40],[8,32],[8,32],[16,32],[16,32],[16,24],[24,24],[32,24],[40,24],[48,24],[56,24],[64,24],[72,24],[80,24],[88,24],[96,24],[104,24],[112,24],[120,24],[120,32],[120,40],[120,48],[120,56],[120,64],[120,56],[120,48],[120,40],[120,32],[120,24],[120,16],[120,24],[120,32],[120,40],[120,48],[120,56],[120,64],[120,72],[120,80],[120,88],[120,96],[120,104],[120,112],[120,120],[120,128],[120,136],[120,144],[120,152],[120,160],[120,168],[120,176],[120,184],[120,192],[120,200],[120,208],[120,216],[120,224],[128,224],[136,224],[144,224],[152,224],[152,232],[160,232],[168,232],[176,232],[184,232],[192,232],[200,232],[208,232],[216,232],[224,232],[232,232]],

"pts1": [[232,232],[224,232],[216,232],[216,224],[208,224],[200,224],[192,224],[184,224],[176,224],[168,224],[160,224],[152,224],[144,224],[136,224],[128,224],[120,224],[120,216],[120,208],[120,200],[120,192],[120,184],[120,176],[120,168],[120,160],[120,152],[120,144],[120,136],[120,128],[120,120],[120,112],[120,104],[120,96],[120,88],[120,80],[120,72],[120,64],[120,56],[120,48],[120,40],[120,32],[120,24],[112,24],[112,16],[104,16],[96,16],[96,8],[88,8],[80,8],[72,8],[64,8],[56,8],[48,8],[40,8],[32,8],[24,8],[16,8],[8,8]],

"pts2": [[96,224],[104,224],[112,224],[120,224],[120,216],[120,208],[120,200],[120,192],[120,184],[120,176],[120,168],[120,160],[120,152],[120,144],[120,136],[120,128],[120,120],[120,112],[120,104],[120,96],[120,88],[120,80],[120,72],[120,64],[120,56],[120,48],[120,40],[120,32],[120,24],[120,16],[112,16],[104,16],[96,16],[88,16],[80,16],[72,16],[64,16],[56,16],[48,16],[40,16],[32,16],[24,16],[16,16]]

`

## 算法概述

在以前项目基础上，已经实现CBS基础算法跨平台编译与运行。

CBS基于冲突的搜索是最优最先进的混合MAPF算法，算法由两层搜索过程组成，底层搜索为每个agent搜索出一条有效路径。上层搜索负责检查路径冲突，并选择其中代价最小的分支重新进行底层路径搜索，直到上层搜索发现有效路径为止。

针对多AGV路径规划，采用基于冲突的搜索算法CBS，将MAPF分为两层去求解，下层采用优化的基于曼哈顿距离的STA\*算法解决多个单AGV路径规划问题，算法的上层采用约束树结点解决多个 AGV 路径间的冲突问题。

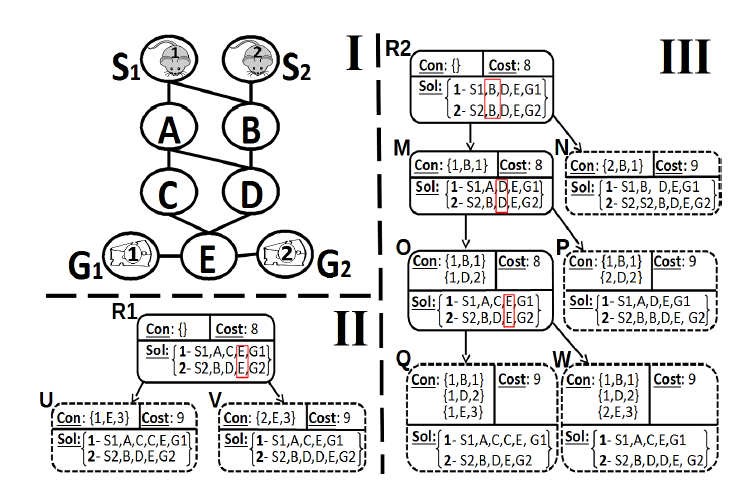
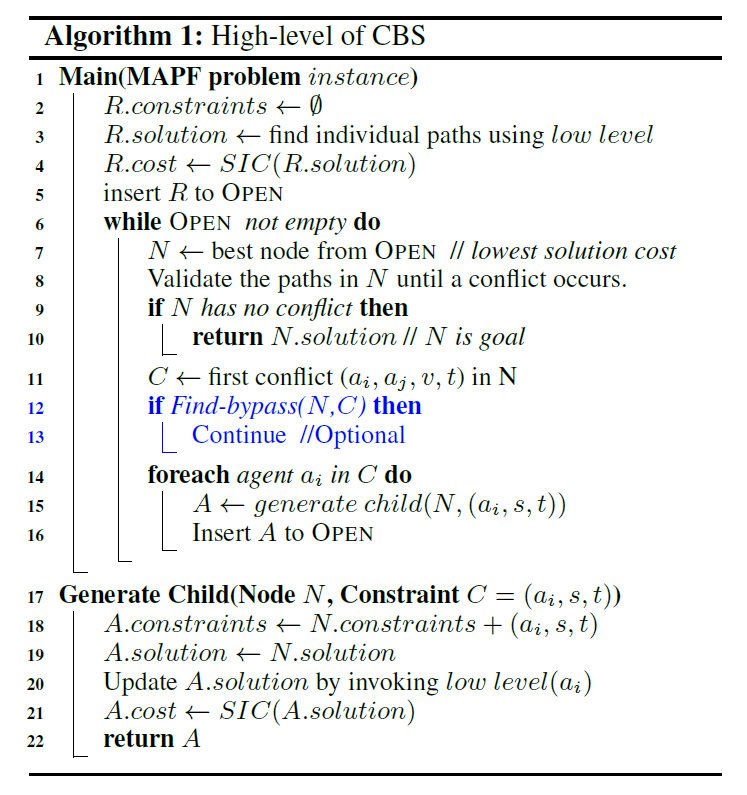


图 4‑1 上层CT树搜索算法逻辑

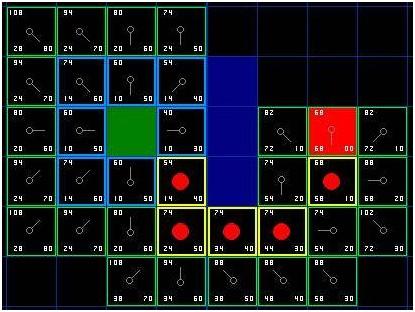
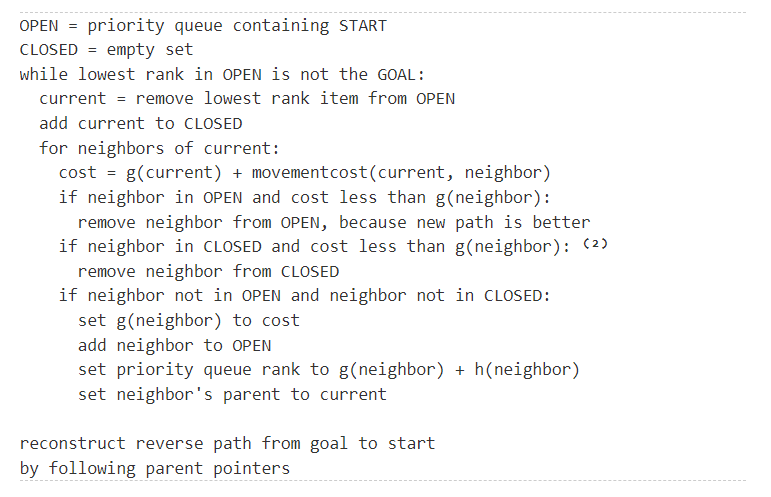


图 4‑2 底层STA\*算法逻辑

## 需突破关键算法技术及进度安排

CBS算法在若干场景效率很低，无法满足实时路径规划的需要。针对不同地图场景和优化目标的需要，拟定了如下的算法优化路径。

算法第一阶段解决静态路径规划的优化问题，第二阶段解决算法动态路径规划的问题。其中静态路径规划是输入地图环境和多车辆起点和终点，假定地图环境净空和多车同时开始启动，规划多车起点到终点的不冲突且总时间最优路径；动态路径规划在静态路径规划基础上，取消地图环境净空假定，提供正在运行的车辆的路径规划，规划新一批同时启动车辆的不冲突且总时间最优路径（不变动正在运行的车辆的路径规划）。

表 4-1 优化CBS技术点及进度安排

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **优化技术** | **人员** | **开始时间** | **结束时间** | **人日** |
| 走廊推理优化 | 吕昱 | 2023/2/11 | 2023/2/14 | 4 |
| 优先级冲突求解 | 曹松涛 | 2023/2/14 | 2023/2/17 | 4 |
| 旁路冲突求解 | 吕昱 | 2023/2/15 | 2023/2/17 | 3 |
| 高阶可接受启发函数WDG/CG/DG分析优化 | 曹松涛 | 2023/2/18 | 2023/2/20 | 3 |
| 矩形推理优化 | 吕昱 | 2023/2/18 | 2023/2/20 | 3 |
| 目标推理优化 | 曹松涛 | 2023/2/21 | 2023/2/24 | 4 |
| 互质体推理优化 | 吕昱 | 2023/2/21 | 2023/2/24 | 4 |
| 不相交分裂优化 | 曹松涛 | 2023/2/26 | 2023/2/28 | 3 |
| 开发静态规划算法 | 吕昱 | 2023/2/25 | 2023/2/28 | 4 |
| 确定静态规划算法方案 | 吕昱 | 2023/2/28 | 2023/3/10 | 11 |
| 封装算法库，供外部使用 | 曹松涛 | 2023/2/28 | 2023/3/10 | 11 |
| 确定动态算法技术方案 | 吕昱 | 2023/3/1 | 2023/3/6 | 6 |
| 开发动态规划算法 | 吕昱 | 2023/3/7 | 2023/3/17 | 11 |
| 更新算法封装库 | 曹松涛 | 2023/3/11 | 2023/3/23 | 13 |
| 开发算法库上层业务 | 曹松涛 | 2023/3/24 | 2023/3/31 | 8 |
| 联调路径规划算法并改进 | 吕昱 | 2023/3/18 | 2023/3/31 | 14 |

## 算法核心技术点

### 动态路径规划

【问题描述】

在线实时路径规划——需要解决静态规划存在的问题。CBS算法顶层不做修改，主要需要在底层每次针对每个Agent寻路的时候，找最短路径，同时规避掉约束表中的哪些冲突点位。底层算法需要规避CBS顶层父节点和本次冲突点构造的约束以外，还需要规避以前规划所要求的一些地图上的动态路径点。

约束表有两个约束表：

（1）一是CBS标准算法所要求的，就是顶层算法会评估本次规划产生的冲突点位，然后构造约束表后，传给底层算法，要求其规避。

（2）二是用户传入的此前已经规划并被占据的路径点。

【测试方案】

（1）

zxk-105x34.map

tasks: [{"s":[34,21],"e":[11,20]},{"s":[34,22],"e":[13,20]}]

（2）

zxk-105x34.map

tasks: [{"s":[34,21],"e":[11,20]},{"s":[34,22],"e":[13,20]}]

planned\_paths

"psd0":[[33,17],[33,18],[33,19],[33,20],[33,21],[33,22],[34,22]]

（3）

zxk-105x34.map

tasks:[{"s":[34,21],"e":[82,19]}]

（4）

zxk-105x34.map

tasks:[{"s":[82,19],"e":[20,12]}]

（5）

zxk-105x34.map

tasks:[{"s":[34,22],"e":[81,20]}]

（6）

zxk-105x34.map

tasks:[{"s":[81,20],"e":[33,7]}]

（0）

zxk-105x34.map

tasks: [{"s":[34,21],"e":[11,20]},{"s":[34,22],"e":[13,20]}]

无实质动态障碍，解如下。

"pts0": [[34,21],[33,21],[33,22],[32,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20],[12,20],[11,20]],

"pts1": [[34,22],[34,22],[34,22],[33,22],[32,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20]]

上述路径含义：pts0一马当先，pts1先等着，然后依次到达目标点。

**第一个动态约束场景——增加从两个起点前方经过，从上向下的动态约束。**

planned\_paths

"psd0":[[33,20],[33,21],[33,22],[33,23]]

【正确求得解】

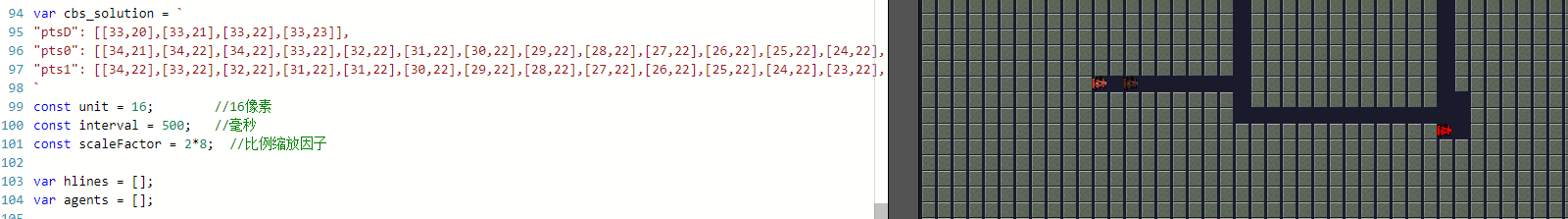
最初发现没有任何影响！这是一个bug，Instance载入地图数据的时候，没有动态载入地图的宽度，导致载入的动态障碍数据计算错误，修改bug后，可以寻到路径如下。

"pts0": [[34,21],[34,22],[34,22],[33,22],[32,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20],[12,20],[11,20]],

"pts1": [[34,22],[33,22],[32,22],[31,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,20],[20,19],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20]]

这个路径pts1先领先前行，pts0在后面第3步等待一次，后续在[20,20]，pts1向上走一步，等待pts0超越，再继续前行。

为便于观察动画运行效果，可在前端可视化界面，把动态障碍路径也作为solution填入，并将其设置为第一个位置，并且设置特别的颜色，便于观察。



**第二个动态约束场景——在[20,20]点占据一定时间后让开。**

planned\_paths

"psd0":

[[20,12],[20,13],[20,14],[20,15],[20,16],[20,17],[20,18],[20,19],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20],[20,20]]

【无法求得正确解】

"pts0": [[34,21],[33,21],[33,22],[32,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20],[12,20],[11,20]],

"pts1": [[34,22],[33,22],[33,23],[33,22],[32,22],[31,22],[30,22],[29,22],[28,22],[27,22],[26,22],[25,22],[24,22],[23,22],[22,22],[21,22],[20,22],[20,21],[20,21],[20,21],[20,20],[19,20],[18,20],[17,20],[16,20],[15,20],[14,20],[13,20]]

检查到冲突，在如下代码没有正确处理。



当出现动态障碍冲突的时候，如果简单退出，则发现在本案例中，pts0（前车）没有等待20,20位置障碍消失。

【算法解读】

引入动态约束后，CBS类generateRoot寻找初始路径，给出了不同的初始路径，规避了动态障碍的冲突。但初始解路径是不考虑相互之间冲突的，冲突由上层算法进行检测，发现冲突，重新更新约束表，再调用底层算法findPath（以及findShortestPath）寻找新路径。

源码中上层代码比较混乱，主要功能流程就是FOCAL保存了带权重的最小值节点，从而方便在搜索空间中选择合适的候选节点。

路径规划上层算法：

该函数的执行过程与后面分裂子节点，每个子节点操作基本相同，分为：

1）调用generateRoot函数。对每个Agent调用findPath，paths\_found\_initially，对paths\_found\_initially数组赋值，放到openlist和focallist，然后findConflicts。

2）开始循环。

出focallist，对出栈节点，updatePaths，将paths数组赋值。

是否无冲突，如果无冲突，则找到解决方案。

有冲突，是否启发寻径过，如果没有WDG递归，CG。（这部份展示不考虑精读）。

扩展节点1分2。

### 关联任务调度

【问题描述】

动态路径规划解决当前正在运行状态下的可以持续插入新路径规划。对于一次较复杂的任务，一般包含从停车位到取货点，停留在取货点若干时间，从取货点到送货点，停留若干时间，回到停车位。对于这样的复杂任务，特别是多个这样复杂任务并行的环境下，完成动态路径规划是相对复杂的问题，需要解决动态路径规划的时机问题。

【解决方案】

（1）任务关系表如下，类似一个链表，一个出库任务比如开始于0#任务（链表头），分为1#从停车点到取货点，2#停留，3#取货点到送货点，...。

|  |  |  |  |
| --- | --- | --- | --- |
| **任务** | **任务类型** | **任务状态** | **后续任务** |
| 0# | 队列头 | 规划/完成/执行中... | 1 |
| 1# | 路径任务 | 规划/完成/执行中... | 2 |
| 2# | 路径任务 | 规划/完成/执行中... | 3 |
| 3# | 路径任务 | 规划/完成/执行中... | 4 |
| 4# | 维护任务（不占通道） | 规划/完成/执行中... | ^ |

SELECT task\_id, next\_task\_id

FROM task\_chain

INNER JOIN (

SELECT 'start\_task' AS a # 111为头节点Id

UNION

SELECT @next\_id AS a

FROM task\_chain,

(SELECT @next\_id := 'start\_task') n

WHERE task\_id = @next\_id

AND @next\_id := next\_task\_id

# 其他条件

) ids

ON task\_chain.task\_id = ids.a;

（2）路径任务表如下，是占用地图的所有路径相关的任务，包括运输任务，停机等待任务，寻路任务等，它与路径规划密切相关。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **路径任务** | **车辆** | **实际开始时间** | **实际完成时间** | **规划开始时间** | **规划完成时间** |
| 1# | AGV1 |  |  | 0 | 3 |
| 2# | AGV1 |  |  | 4 | 10 |

（3）规划路径表如下，记录规划的路径，可以考虑用关系数据库或时序数据库存。但需要注意规划之后修改的问题，可以分为临时表和生效表。临时表用于过程中的数据记录。生效表是用户确认后的。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **时间（timestep）** | **AGV1** | **AGV2** | **AGV3** | **AGV4** | **……** | **AGV50** |
| 0 | （3，3） | （6，3） | （1，9） |  |  |  |
| 1 | （3，4） | （6，4） | （2，9） |  |  |  |
| 2 | （3，5） | （5，4） | （3，9） |  |  |  |
| 3 | （4，5） |  | （3，8） |  |  |  |
| 4 |  |  | （3，7） |  |  |  |
| 5 |  |  | （4，7） |  |  |  |
| 6 |  |  | （5，7） |  |  |  |
| …… |  |  |  |  |  |  |

SELECT \* from ((select max(ts) as agv1 from `public.xc\_equipment` where dis\_voltage is not null) n,(select max(ts) as agv2 from `public.xc\_equipment` where dis\_current is not null) m)

## 前端调试

前端所需要的地图文件来源：从Tile中导出csv文件，该csv文件是一个-1，0组成的csv标准格式文件，需要将-1替换为2，0保持不变（通道）。然后将该文件拷贝到前端可视化工程对应的地图文件夹中（ly-phaser3-examples\public\assets\tilemaps\csv\）。

MAPF算法输出的结果拷贝到https://github.com/lvyv/phaser3-examples.git工程启动的前端代码中：

（1）修改地图文件。

（2）修改修改显示尺寸。

（3）cbs\_solution=`来自算法的输出文件`。

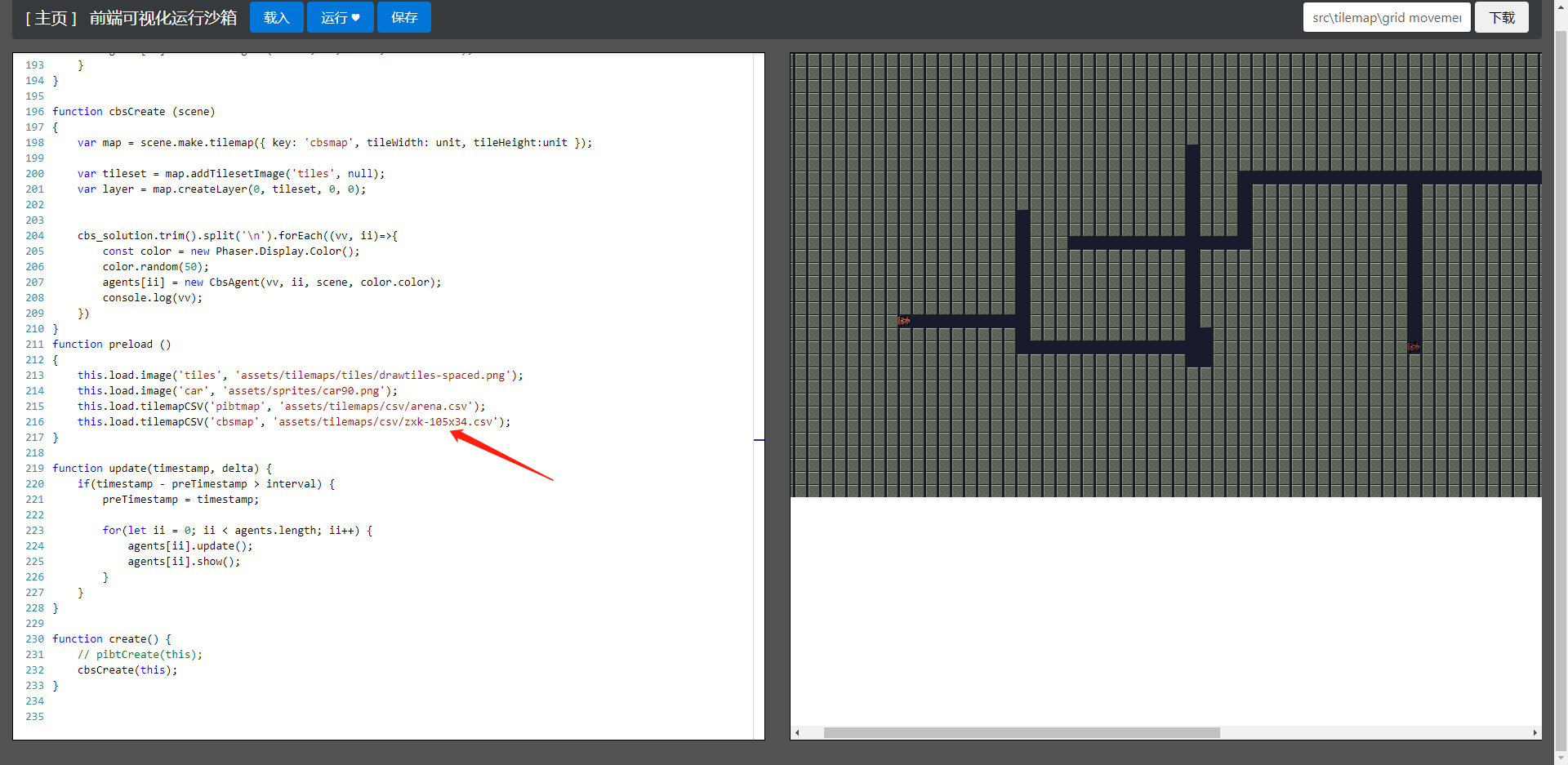


图 4-3 算法结果可视化——修改显示地图

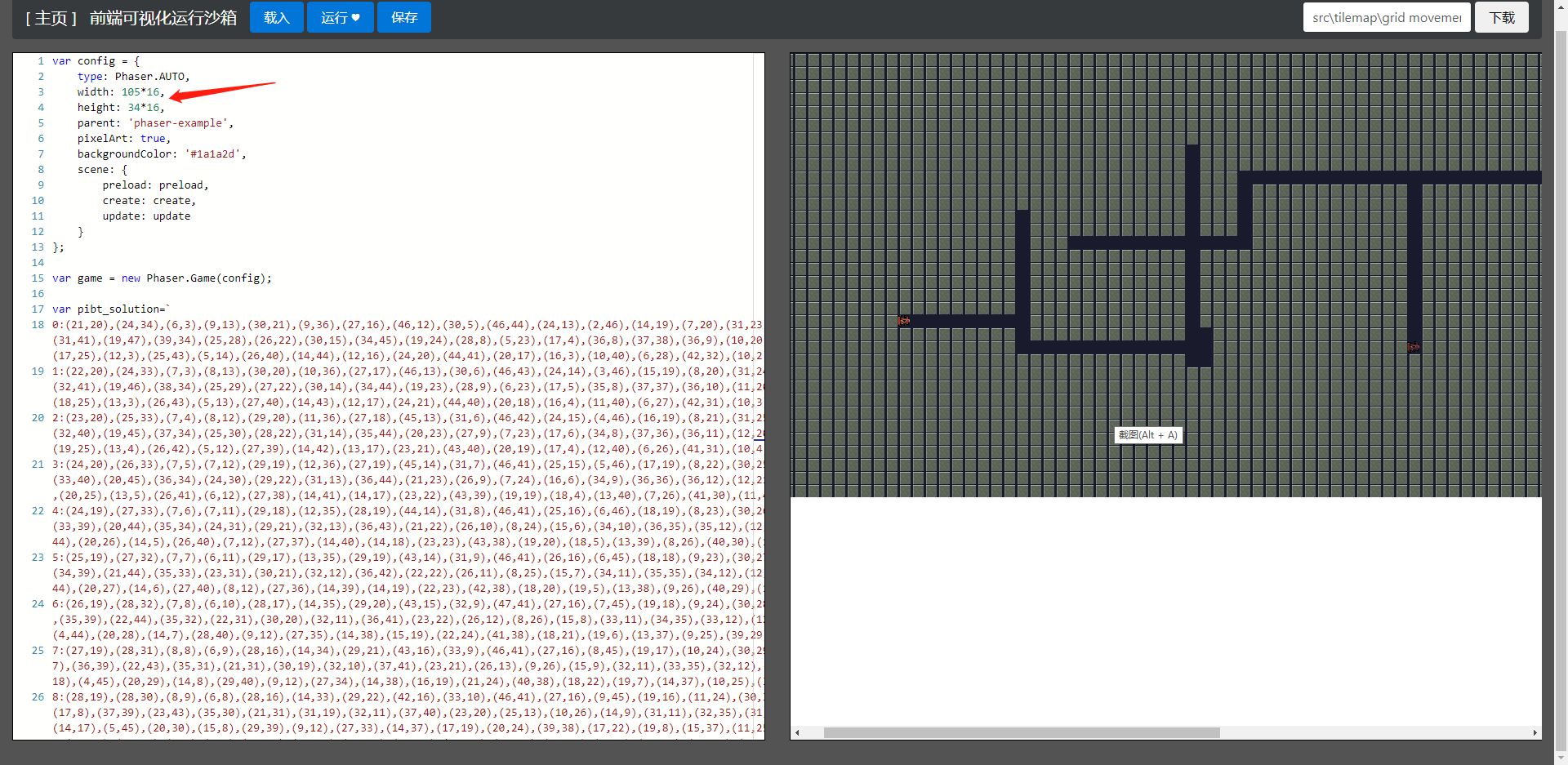


图 4-4 算法结果可视化——按地图修改显示尺寸

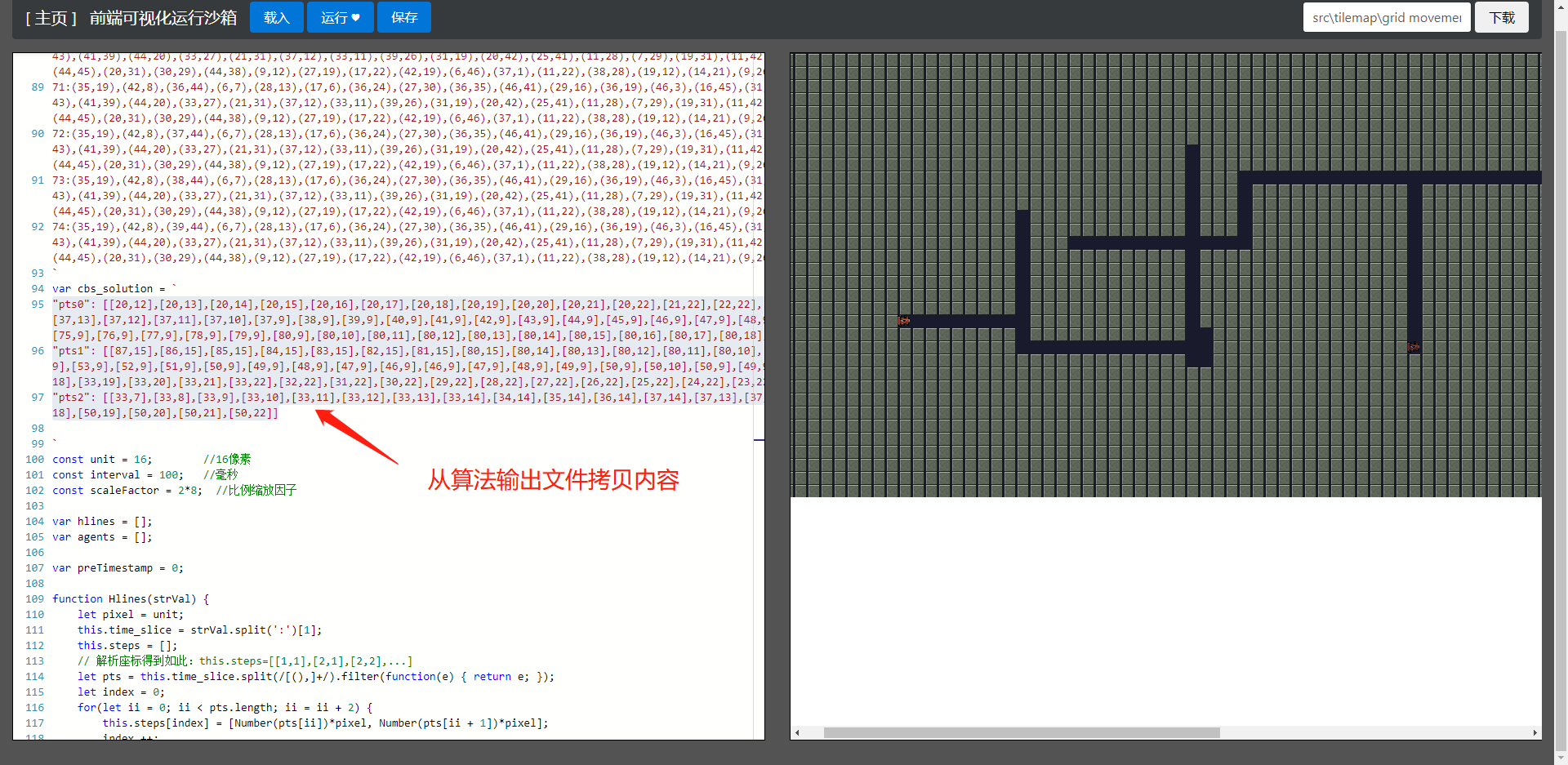


图 4-5 算法结果可视化——拷贝源于算法求解的路径

## 部署视图

MAPF算法软件可部署在Linux或Windows操作系统上，支持国产自主可控硬件服务器上运行。

部署的方式：采用docker部署。

tar zxvf mptools-mapf-scheduler.tar.gz

cd mptools-mapf-scheduler

sudo docker build -t cbs\_rt:v1 .

docker run -p 7080:7080 --name mapf\_scheduler cbs\_rt:v1

表 4-12 MAPF算法部署主机配置

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 部署内容 | 部署方式 | CPU | 内存 | 数据盘 | 所需端口 |
| 服务器 | Docker/K8S | 32核 | 64G | 2T | 18081 |

## 软件更新

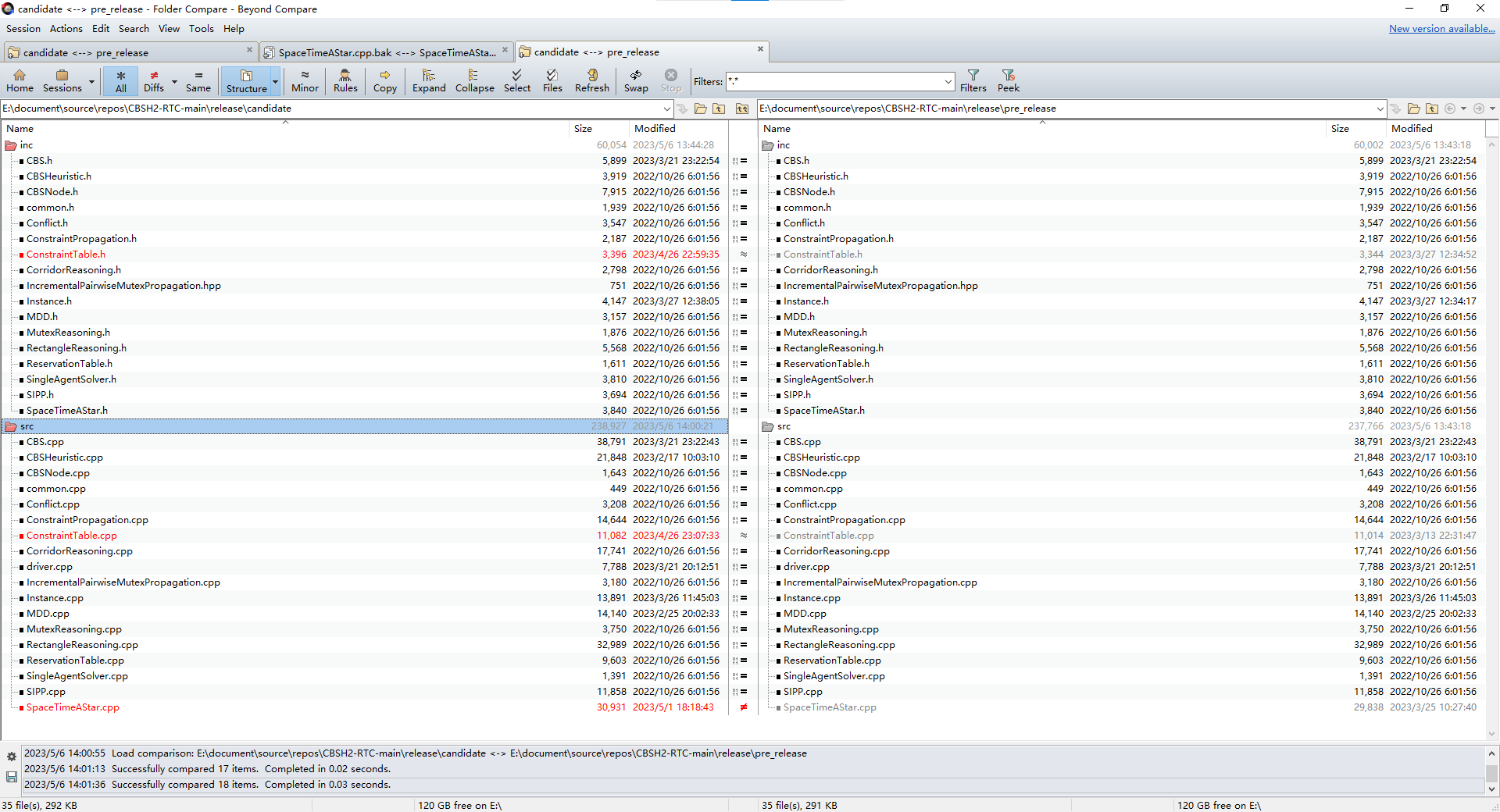
### 地图更新

地图更新请参考《地图制作说明》。

### 执行程序更新

（1）比较源码修改。

使用beyondCompare，比较windows下代码修改的部份。



（2）更新编译服务器上的源码。

（3）在ubuntu上编译。

在虚拟机ubuntu 18.04.5的windows的共享目录中的发布源码目录，直接编译。

cd build

make

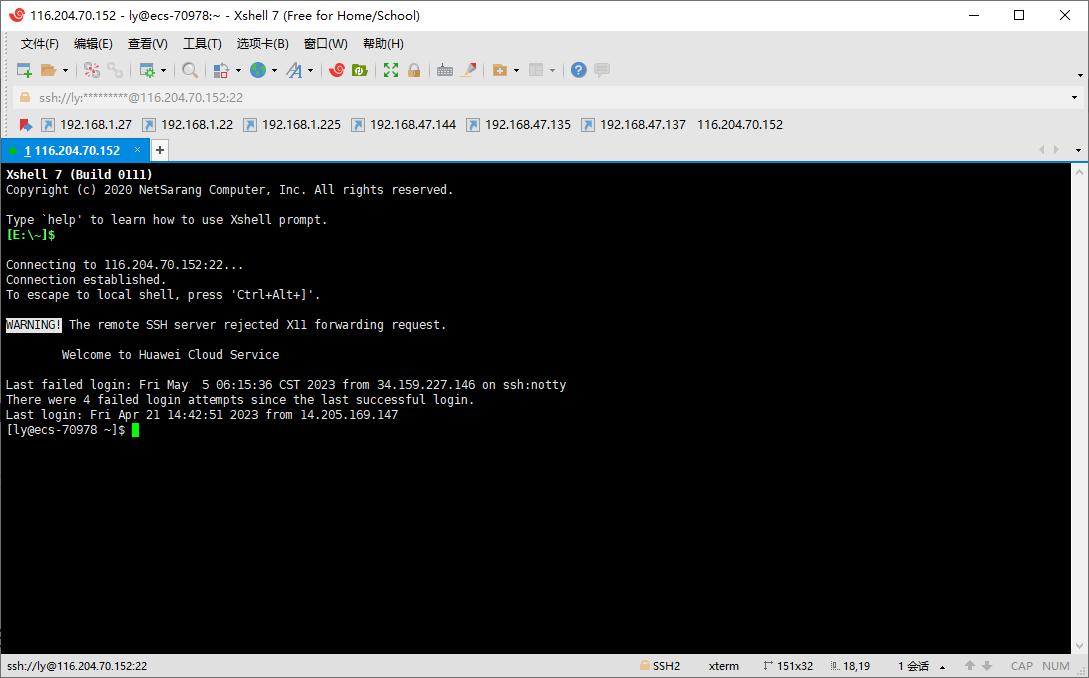
编译成功，目标文件为build目录下的cbs。

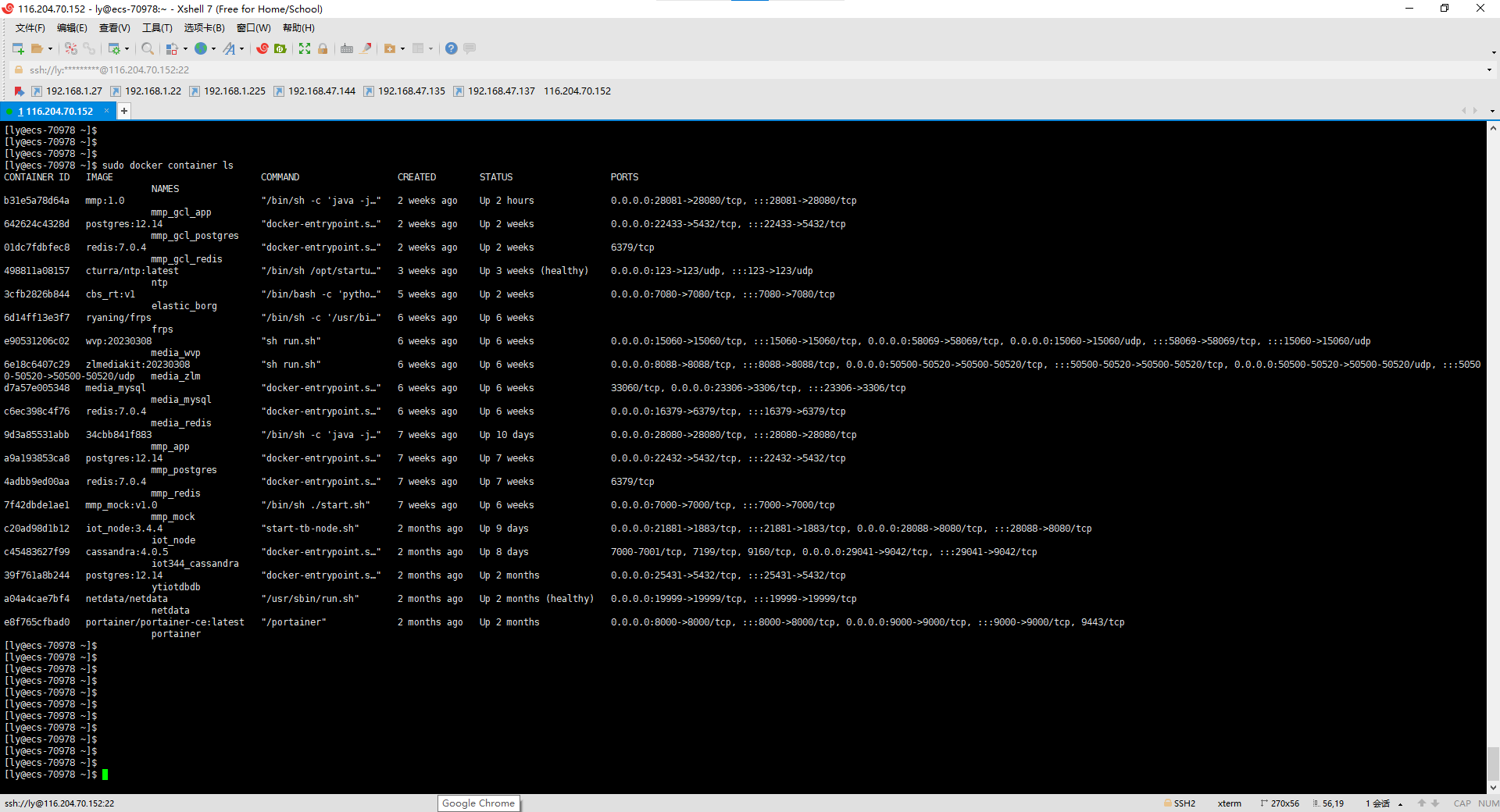
（4）测试。

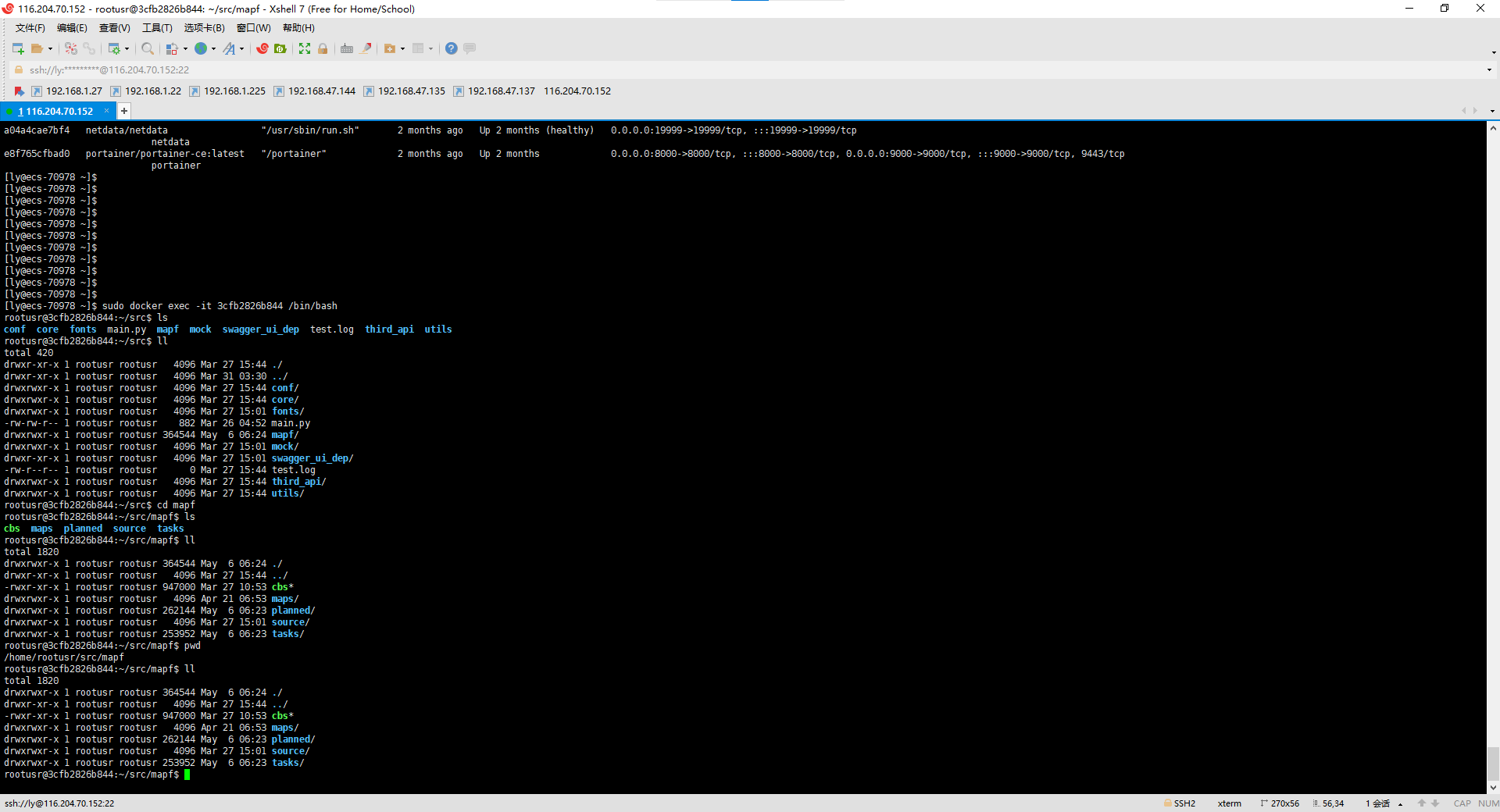
./cbs -m zxk-105x34.map -a bug\_zxk105x34\_2.scen -p bug\_dynamic2.paths -o test.csv --outputPaths=bug\_res\_2\_paths105x34.json -k 2

程序输出规划路径。

（5）发布和部署。







在容器内备份。

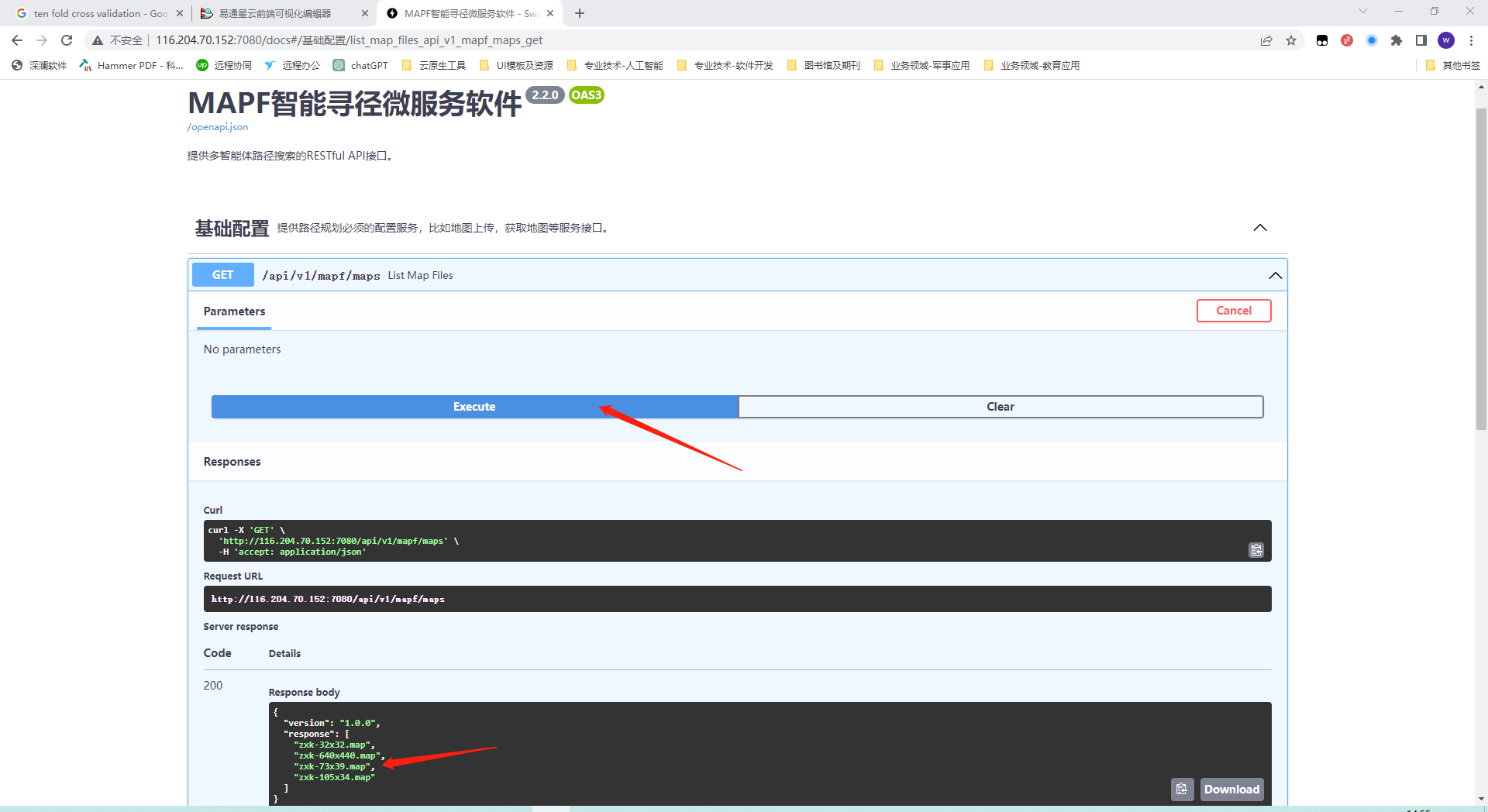
mv cbs cbs.bk

在容器外拷贝和测试。

sudo docker cp cbs 3cfb2826b844:/home/rootusr/src/mapf/cbs

访问如下地址。

http://116.204.70.152:7080/docs



# 非功能设计

## 性能指标测试

采用地图集待定，原则为6000x4000尺寸像素地图，750x500图块范围进行路径搜索，单地图最大Agent支持50个。

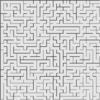
      

图 5-1 地图集设计

测试工具待定，测试环境暂定为CPU配置英特尔i5-7200U 2.50GHz，内存16GB。

## 扩展性

待定。

## 设计约束

待定。

# 需求可追踪性

## 从软件需求规格到概要设计的追踪关系

表 6-1 从软件需求规格到二次细化设计的追踪关系

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 软件需求规格说明 | | 概要设计 | |
| 章节号 | 名称 | 章节号 | 名称 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 从软件概要设计到需求规格的追踪关系

表 6-2 从概要设计到软件需求规格的追踪关系

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 概要设计 | | 软件需求规格说明 | |
| 章节号 | 名称 | 章节号 | 名称 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |