**任务2-推箱子游戏**

目录

[1.问题描述 2](#_Toc108619752)

[2.问题分析 2](#_Toc108619753)

[2.1总体分析 2](#_Toc108619754)

[2.2具体分析 3](#_Toc108619755)

[2.2.1随机地图的生成 3](#_Toc108619756)

[2.2.2推箱子求解 4](#_Toc108619757)

[2.2.3图形界面显示 6](#_Toc108619758)

[3.实验过程 7](#_Toc108619759)

[3.1相关类文件概览 7](#_Toc108619760)

[3.2重要代码解析 8](#_Toc108619761)

[3.2.1求解代码 8](#_Toc108619762)

[3.2.2 寻路 9](#_Toc108619763)

[4.效果展示 10](#_Toc108619764)

[5.总结 13](#_Toc108619765)

# 1.问题描述

1. 使用windows form用c++制作一个推箱子地图生成编辑器，编辑器可以设置地图大小
2. 可以在地图上设置3钟标记：墙，可推动的box，小人起始点，目标点
3. 点击求解按钮，使用a星启发式算法求解推箱子问题，并将成功的步骤一步一步演示

推箱子游戏是一个益智类游戏，玩家需要移动小人将箱子推到指定位置中，当所有箱子都在目标位置上时成功通关。小人可以上下左右移动，一次走一个格并且不能通过墙。只有当推动后箱子所在的地方为地板，即没有障碍物时才可以推动此箱子。

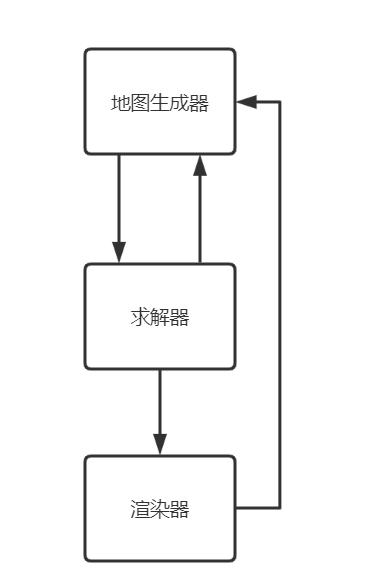
# 2.问题分析

## 2.1总体分析

要完成这一任务我们需要以下几个步骤：

1. 根据指定的长、宽自动生成一个可解的推箱子地图并进行显示
2. 求解解决问题的方案
3. 将求解的方案转化成小人的一步步移动进行显示

输入长、宽，点击生成按钮由地图生成器进行随机地图的生成，利用求解器进行这张地图的求解，判断是否可解，如果有解则进行地图的展示。点击求解按钮进行小人的一步步移动。



简单流程图

所以实现这项任务的关键在于随机地图的生成、怎样求解一个推箱子游戏、进行图形界面的显示。

## 2.2具体分析

### 2.2.1随机地图的生成

一个推箱子地图可以看成一个二维数组模型，不同的值代表不同的道具：墙、小人、箱子、目标点，所以生成一个地图就是构造一个二维数组

1. 根据输入的长、宽生成一个初始地图，周围一圈都是墙壁。
2. 在地图上随机放置小人位置
3. 随机生成一个墙或者箱子和目标点
4. 判断是否有解，有解则生成该地图
5. 返回步骤3进行迭代

### 2.2.2推箱子求解

推箱子求解是完成这个项目最重要的一部分。我们采用广度优先搜索的算法来求解推箱子谜题。理由是：

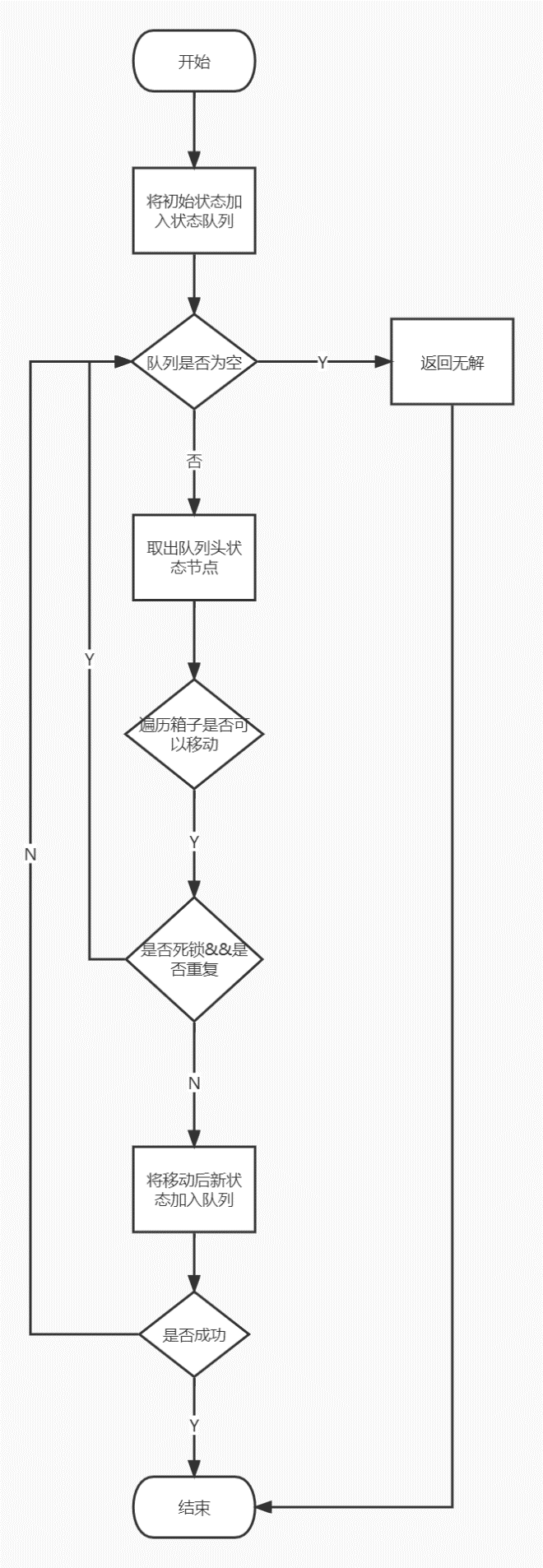
1. 广度优先可以计算出游戏从初始状态到解谜成功状态的最少步骤。

2. 根据状态链可以获得每一步的地图状态，这些状态序列就是问题的最短解题步骤。

对于下一步状态我们只需要考虑有箱子移动后的状态进行添加，对于单纯的小人移动位置后的状态是没有用的，如果仅仅是移动了小人而没有移动箱子，那么移动小人前和移动小人后的状态将是等价的。

所以我们进行广度优先搜索时，我们在队列头中取出当前状态，考虑每一个箱子，并且将箱子进行移动，将变化后的状态加入队列中，循环这一步骤，直到找到解或者队列为空。

如果找到解，那么我们从最后一个状态进行回溯就可以得到求解路径。



求解流程图

**如何判断箱子是否可以移动？**

要判断箱子是否可以向某个方向移动，有两个条件要满足：

1. 人可以到达推动的位置
2. 箱子前进的位置为地板或者目的地

**如何进行小人每一步状态的获取？**

我们知道，按照上述广度优先搜索，我们获取的都是每一次箱子被移动后的状态，并没有前后两个状态间小人如何移动，这时候我们使用算法进行寻路并且记录路径就可以了。

**如何判断状态是否重复？**

通过对状态节点进行hash散列来快速判断是否重复

### 2.2.3图形界面显示

我们需要显示的有以下几种元素：墙、小人、目标点、箱子、地板。并采用EasyX进行图形界面的绘制。根据生成的二维数组进行相应元素的布局。

墙： 

小人： 

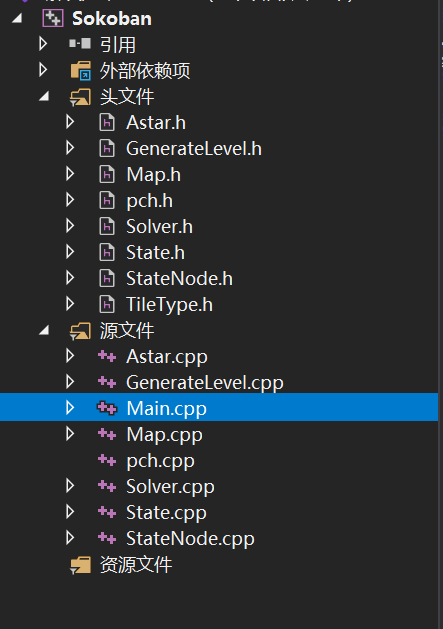
目标点：

箱子： 

地板： 

# 3.实验过程

## 3.1相关类文件概览



类文件概览

Astar.cpp：算法相关类，用于求解最短路径

GenerateLevel.cpp：随机生成地图类

Main.cpp：主函数

Map.cpp：用于构建渲染图形界面

Solver.cpp：求解类

State.cpp：状态类，其中使用二维数组存储地图

StateNode.cpp：状态节点类，用于广度优先搜索

## 3.2重要代码解析

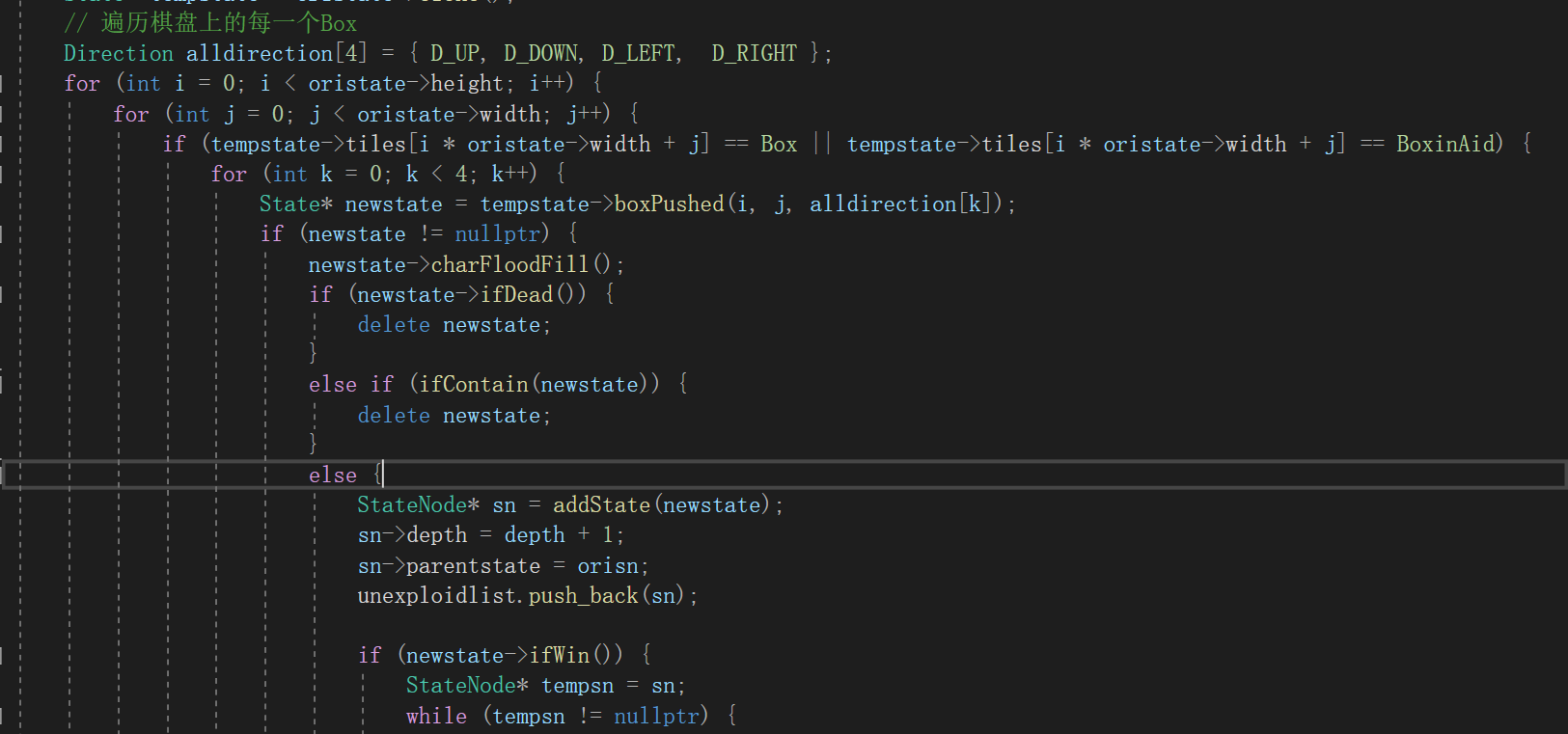
### 3.2.1求解代码

通过广度优先算法进行问题的求解。

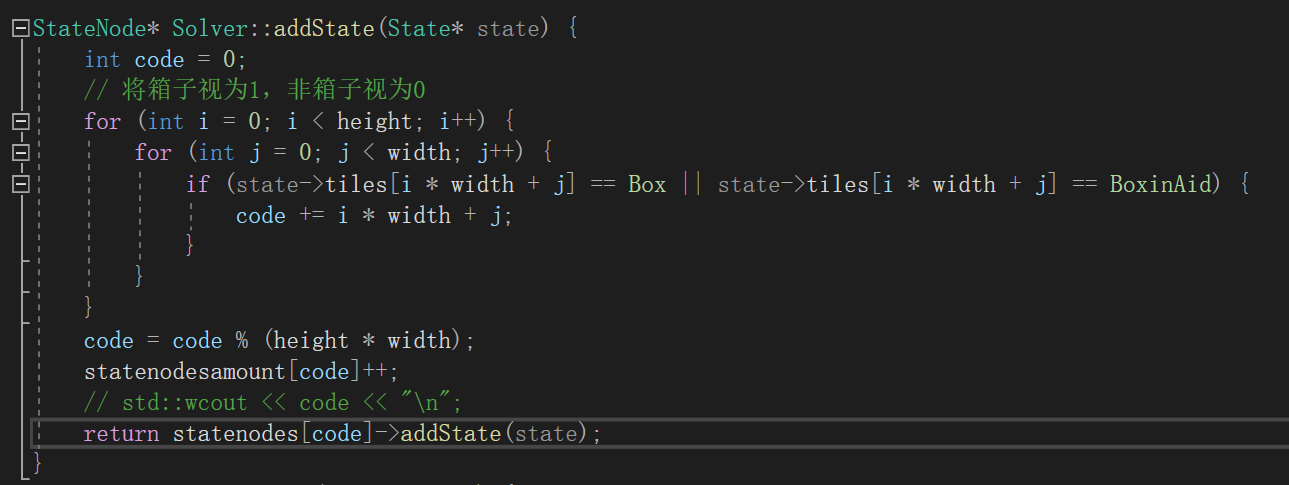
使用队列存储状态节点：



遍历box进行移动，添加新节点：



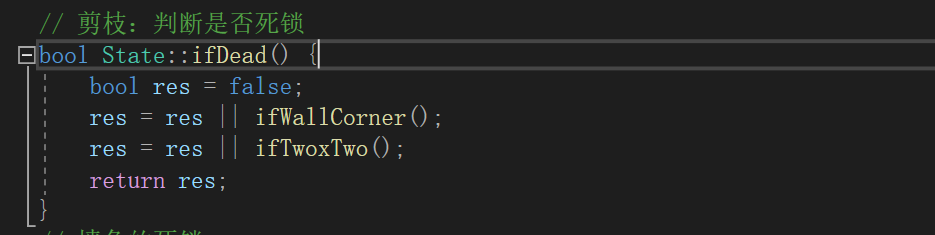
其中判断是否是重复状态采用hash散列结合链表实现



如上图代码所示，对于每一个新加入的状态。我们计算全部box所在的位置进行求和，除以格子总数确定散列的位置，加入其中，如果该位置已经存在，则加入该位置的链表之中，即桶散列方法。

这样子判断是否重复就可以先定位到具体的散列位置，再在链表中判断是否存在，这样子效率较高。

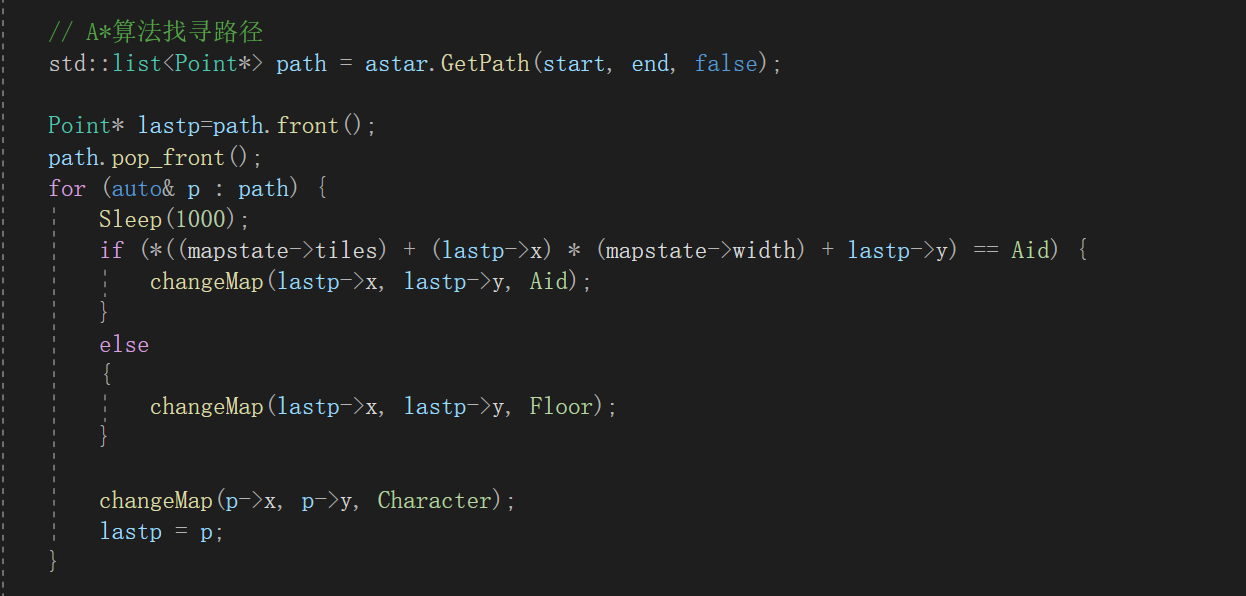
判断是否死锁：



这里有两种判断，一个是箱子推到墙角且不在目标点上，那么永远也无法推到了，二是四个箱子形成田子且不全在目标位置上的情况，箱子也无法再移动了，死锁判断可以有效进行剪枝，减少解空间数量。

### 3.2.2 寻路

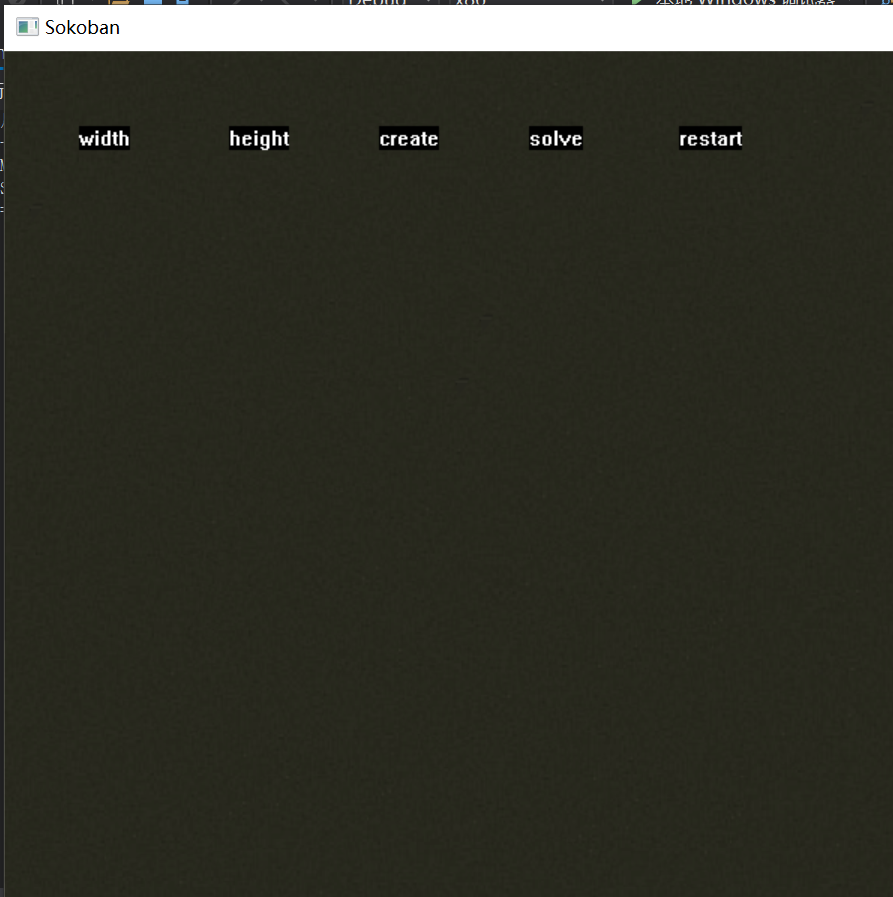
在我们获得解的状态节点后，我们需要完善前后两个节点之间小人的路径，这时候使用算法进行路径的找寻



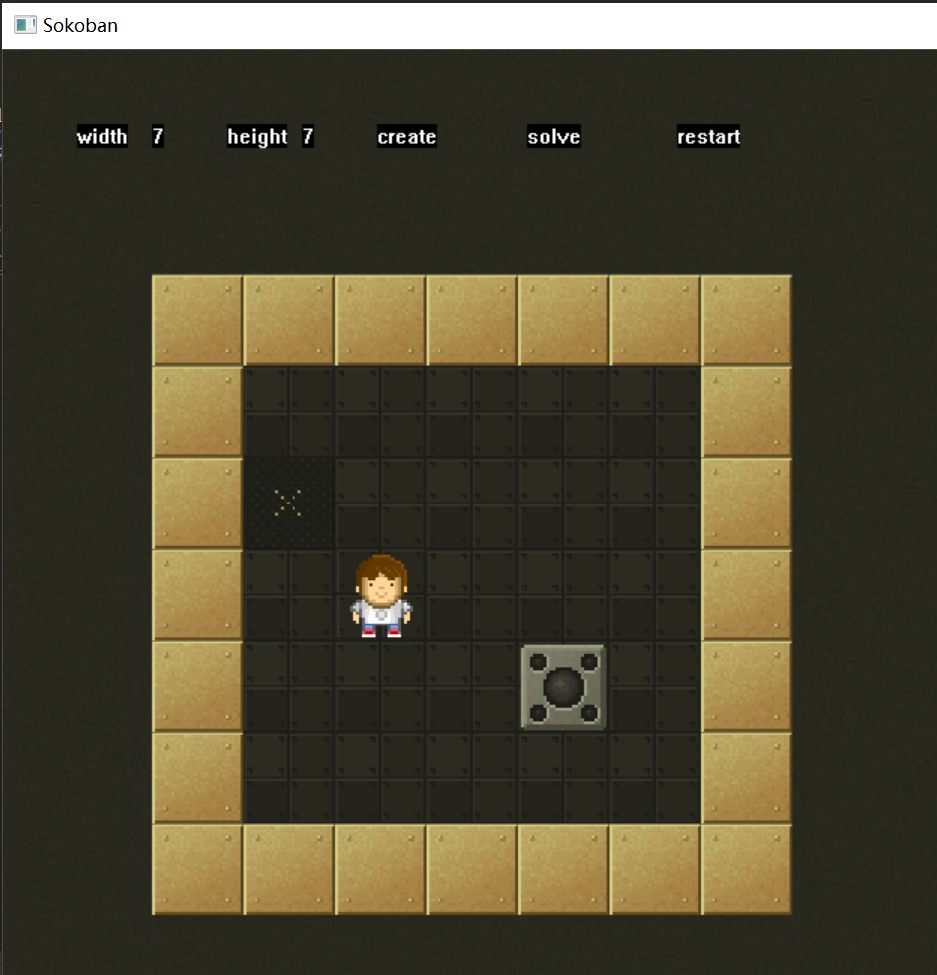
取得两个状态节点小人的位置，计算开始小人位置和结束小人位置，采用算法进行路径找寻。

# 4.效果展示

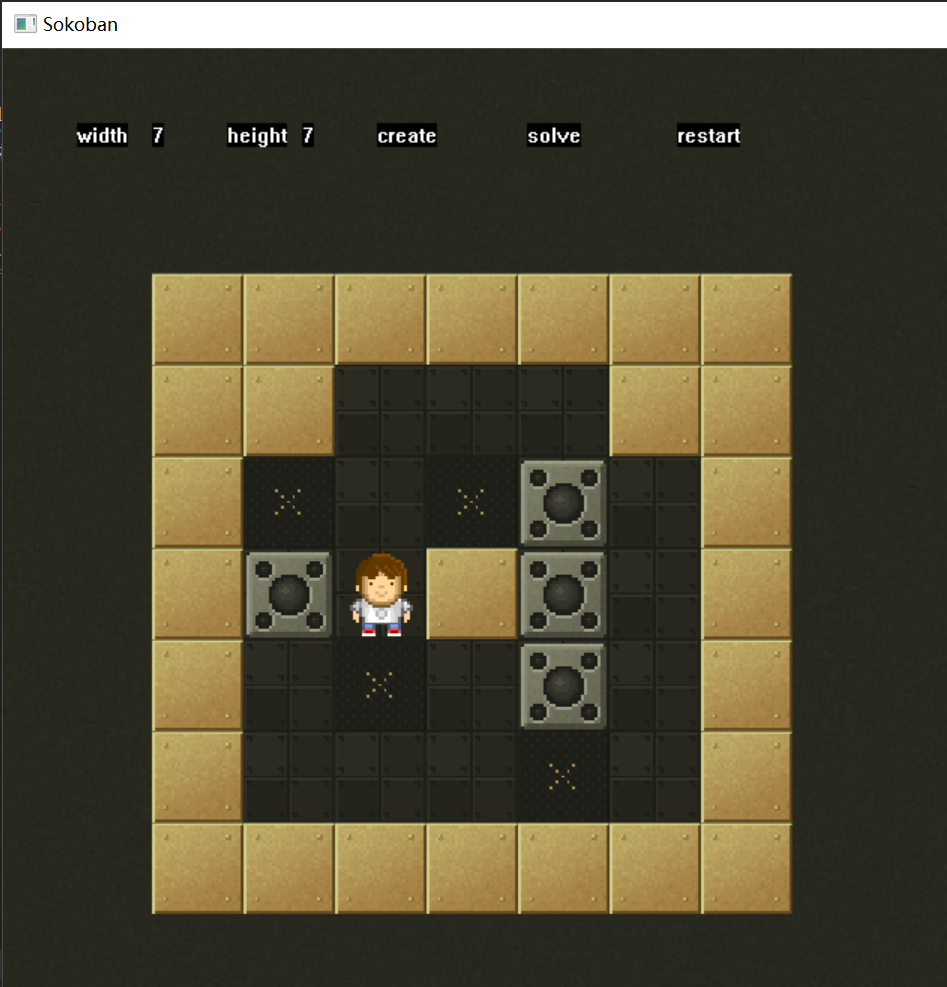
初始界面：



进行长、宽的输入，点击create进行地图的生成：

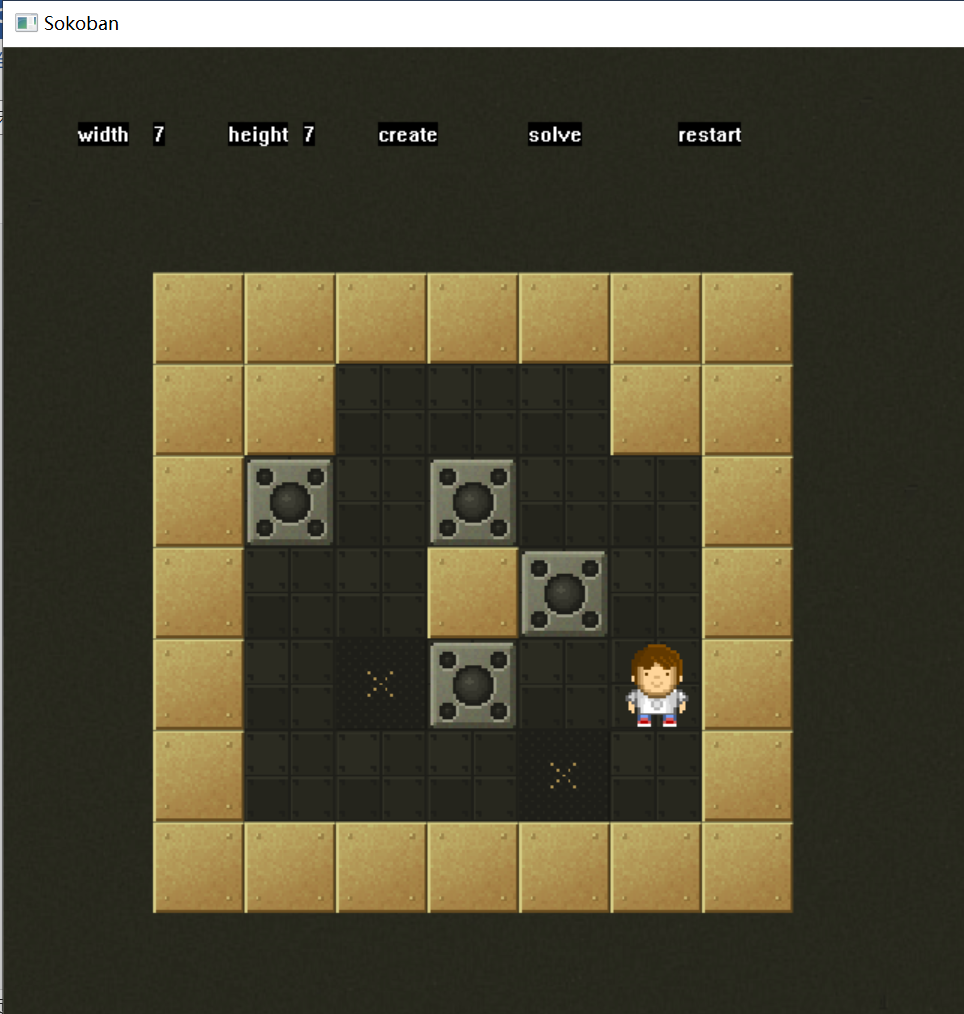


生成的简单地图

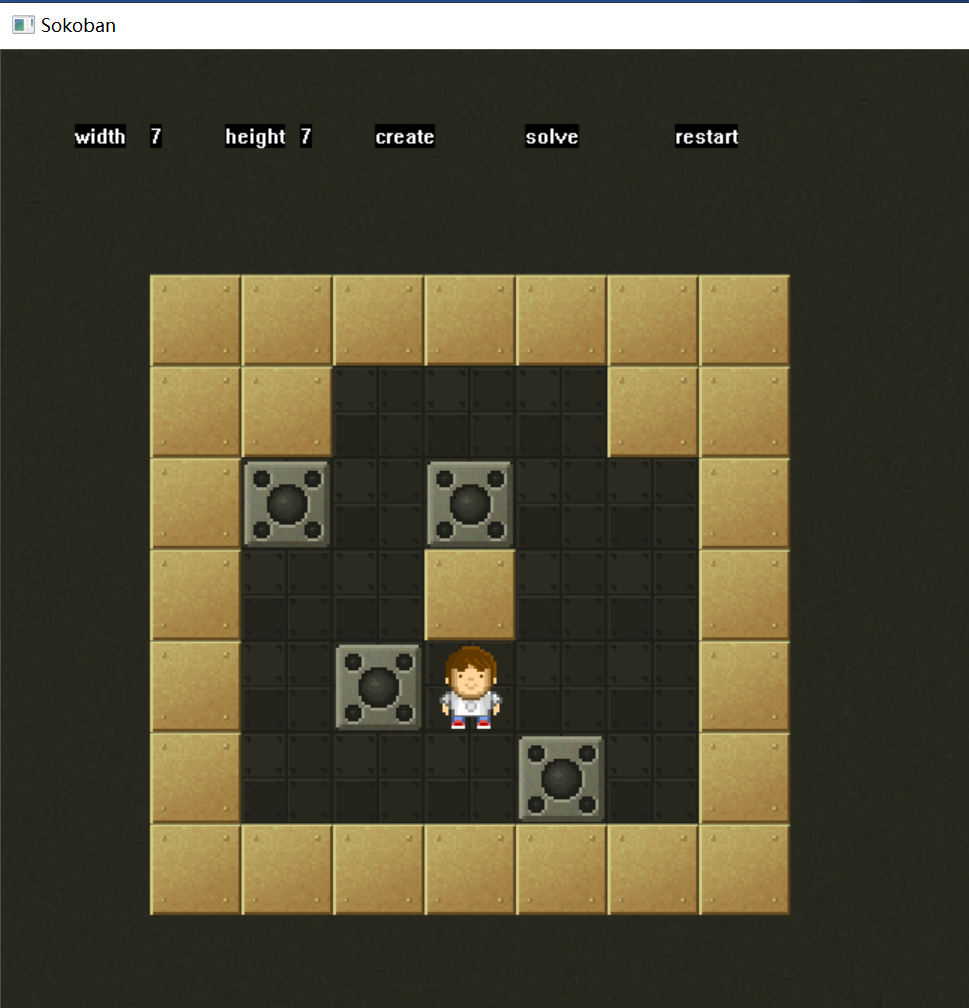


较为复杂的地图

点击solve进行求解：



求解过程中



完成游戏

再次点击create可以再次生成，逐步迭代至复杂地图

点击restart后可以重新设置长、宽

# 5.总结

本次项目中，我使用了广度优先搜索算法和算法进行问题的求解。并且结合图形界面进行显示。通过这次实验，我加深了对广度优先搜索算法和算法的理解，知道游戏开发与算法密切相关。