中 国 地 质 大 学

本科生课程论文

课 程 名 称 面向对象程序设计课程设计

题　　目： 可视化路径搜索系统

教 师 姓 名

本科生姓名

本科生学号

本科生专业 计算机类

所 在 院 系 计算机学院

日　　期： 2022 年 7 月 1 日

课程论文评语

对课程论文的评语：

课程论文成绩：	评阅人签名：

目录

基于 Qt 的可视化路径搜索系统

一、课设题目

基于 Qt 图形库，设计 GUI 程序，实现地图的显示、编辑，并实现数据的读入与保存；运用寻路算法（A Star 等），绕过障碍物寻找到两点间的最短路径，并可视化显示搜索过程；编写识别算法，实现灰度图识别功能。

二、需求分析

1. 数据结构

封装二维字符数组表示二维地图矩阵，实现动态内存管理；使用 QPoint 对象保存起点与终点；设计 Coordinate 结构体表示坐标，提高运算性能，在寻路算法中以 list <Coordinate> 和 vector <Coordinate>分别表示网格遍历顺序与最终路径。

2. 设计地图画板

设计地图画板类 World，实现以下功能：

1) 网格与“点”的绘制；
2) 动态内存管理；
3) 智能适应窗口尺寸；
4) 多线程绘图；
5) 提供丰富接口。

3. 编写寻路算法

选取适当的算法，如 A Star、BFS 等，在保证保证寻路结果最优前提下，还能同时兼顾时间与空间效率，减少开销，提高性能。

4. 多线程运行

以多线程形式运行寻路算法与绘图函数，与 GUI 线程独立，保证程序运行的高效率与稳定性。

5. 实现文件读写功能

设计数据文件编码格式，在地图画板中集成文件读写类方法，并提供外部接口。

6. 日志系统

统筹管理程序运行各阶段的关键输出信息，以文本框形式统一输出，并实现自动编号。

7. 组件间的通讯

在类的内部主要使用 Qt 的信号槽机制，不同类之间使用对象指针调用公共接口，实现数据通讯。

三、总体设计

1. 系统模块划分

地图画板窗口

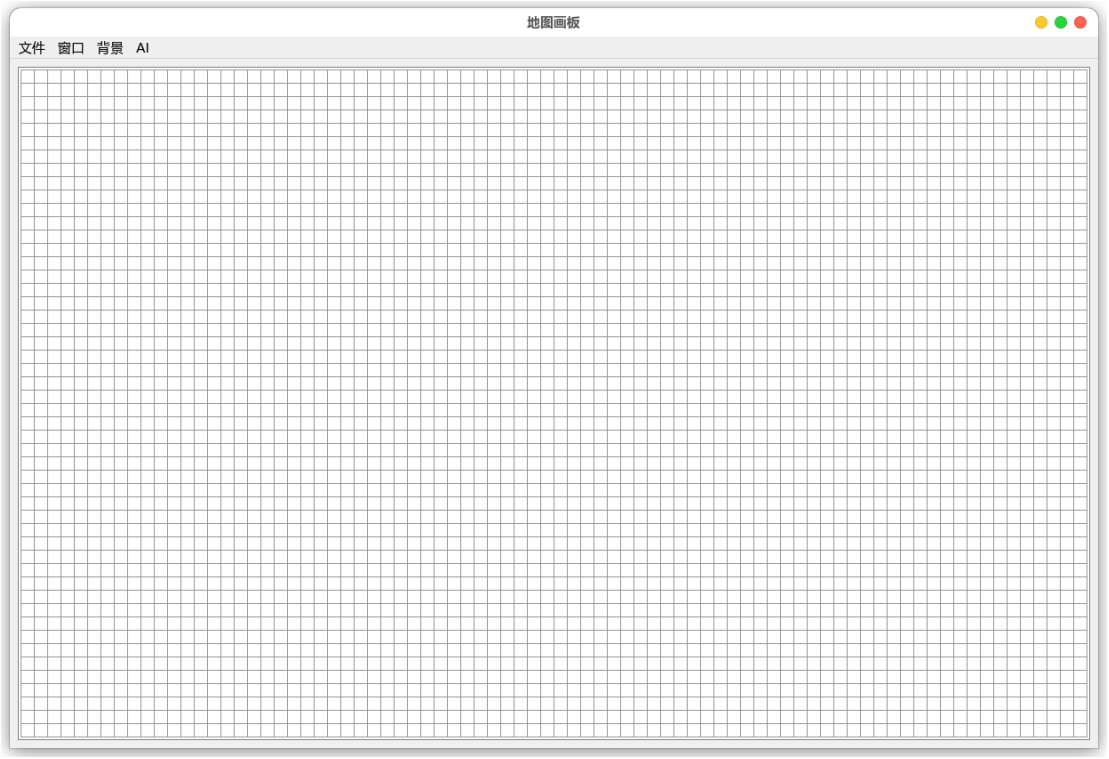


图 1 - 地图画板窗口

该窗口为程序的主窗口，主体为由 World 类实例化的 World 画板对象，左上角依次为*文件*、*窗口*、*背景*、*AI* 选项卡，分别提供文件读写与输出图片、控制面板窗口可见性、自定义背景的设置与清除、智能识别灰度图的功能开关，实现功能的拓展，并保持一定的美观性。

鼠标左键点击画板即可绘制障碍物，单击、长按均支持；若操作失误，鼠标右键点击障碍物方格即可清除该障碍物；放下一个障碍物后，按下 *Shift 键*，同时再次点击，可实现障碍物线段的绘制。

文件选项卡包含 *读取数据*、*保存为数据* 和 *保存为图片* 选项，提供二进制数据的读写和图片文件的输出功能；保存为数据时，用户可选择要保存哪些信息，如图所示。

图 2 - 保存数据对话框

窗口选项卡仅包含 *控制面板* 选项，选项左侧的“✔”符号的有无表示控制面板窗口是否生成，程序启动时会自动生成控制面板窗口。

背景选项卡包含 *打开背景* 和 *清除背景* 选项，选择打开背景后，程序会调用系统的文件对话框，供用户选取欲打开的图片；图片选取后，将生成图片截取对话框，用户可以拖动选择框，截取图片中想要的部分，点击确定按钮或按下回车键后，画板将加载选取的部分图片，将其作为背景。

图 3 - 图片裁剪对话框

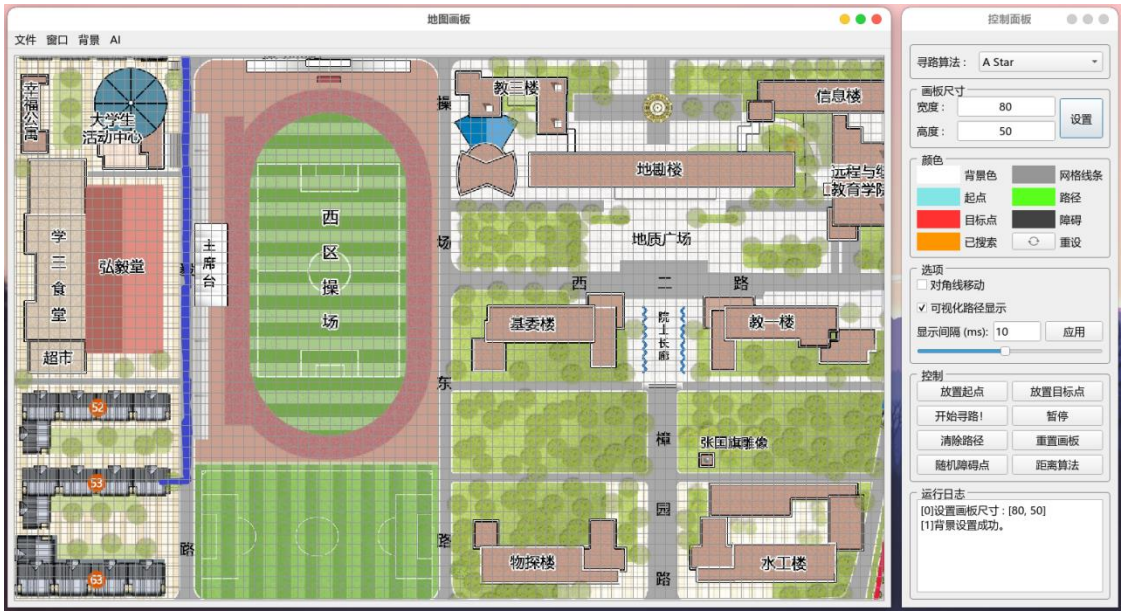


图 4 - 载入背景后的效果

AI 选项卡仅包括 *智能识别* 选项，选择后同样会调用文件对话框，待用户选取图片文件后，程序将调用算法，将图片转化为灰度矩阵，并载入地图画板中。

当智能识别源图片与画板长宽比相近，且画板分辨率够高时，识别效果较为出色；反之，则不能很好的提取源图片中的关键信息。

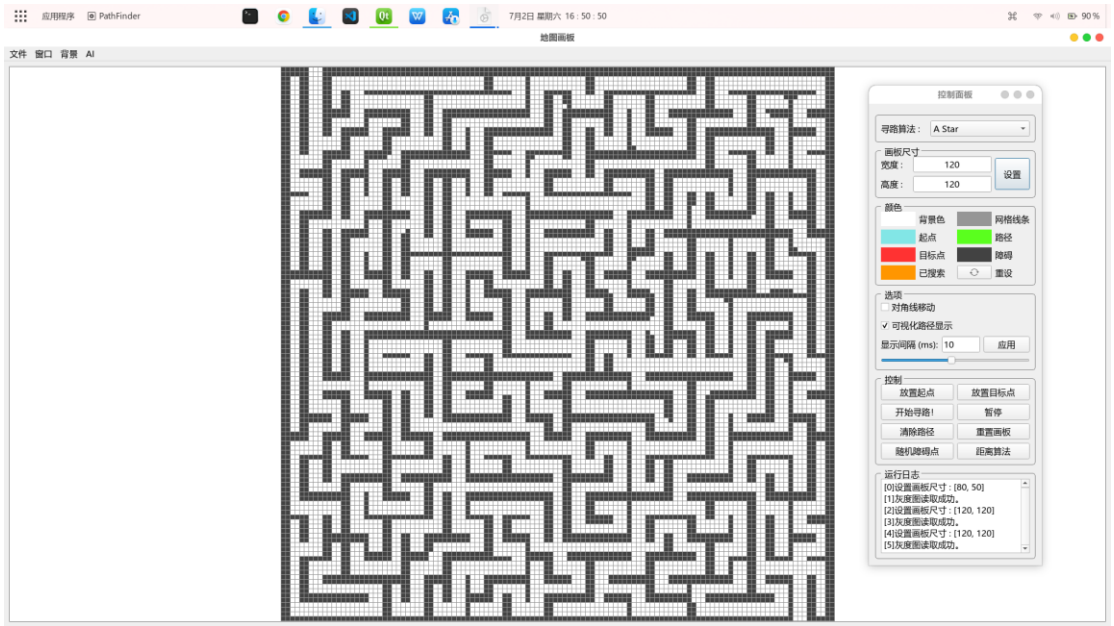


图 5 - 智能识别效果

控制面板窗口

提供各类控制选项，包括寻路算法选择、画板尺寸设置、地图各区域颜色自定义、程序运行状态控制开关等，并集成日志框，输出运行信息。

图 6 - 控制面板窗口

控制面板窗口伴随程序启动自动生成，亦可在地图画板窗口的窗口菜单中对其进行管理，该窗口由数个组合框构成，囊括了寻路算法、画板尺寸、颜色设置等自定义选项，还集成了程序运行控制开关。

该窗口采用了 Qt 的垂直布局与网格布局，调整窗口尺寸时，组件会自动缩放以适应新的尺寸。为了便于与地图画板交互，该窗口被设置为置顶显示，点击最小化或关闭可将其隐藏。

2. 数据类型、数据结构的设计

考虑本程序使用的地图为二维网格，故采用二维动态字符数组存储地图信息，封装实现内存的开辟与释放，提高性能同时减少空间开支。

对于寻路过程中遍历的节点，定义了 Coordinate 结构体，表示坐标信息，分别用 Vector 和 List 存储已遍历的点和路径，避免了大量使用 QPoint，减少了运算资源的消耗，提高了程序的运行性能。

使用 QPoint 对象存储起点与终点信息，QColor 存储各区域的颜色信息，QPixmap 作为画板类的重要组成部分。

输出数据文件时，将画板的长宽信息，起点、目标点的位置，已二进制形式写入 dat 文件；对于已遍历的点，以及可能包含的最终路径，将其各点的坐标以此二进制形式写入文件，以便逐一读出。

输入图片文件时，调用 QPixmap 的类成员函数 Pixmap::save(filename)，将数据以图片（*.png）的形式保存至指定位置。

3. 用户交互设计

为便于用户录入现实中的地图信息，提升工作效率，地图画板提供了背景设置与智能识别功能：

a. 使用背景设置功能，可将现实地图导入作为画板的背景，届时画板各块的颜色将自动变为透明，以方便用户对照地图。用户可根据地图的通行性，选择从空画板开始，绘制障碍物，或者从障碍画板开始，绘制路径，从而满足多样化的需求。

b. 使用智能识别功能，用户可先根据地图图片的比例与精细化程度调整地图画板的长与宽，以达到最佳识别效果。对于识别效果不佳的图片，用户可自行调整其对比度，以减少程序的识别难度。

在用户绘制障碍物的过程中，线段是复杂图形组成的最基本形式之一，为提高地图录入效率，提升用户使用体验，本程序为 *Shift 键* 设置了额外功能，即画下一个障碍物点后，按下 *Shift 键* ，同时再点击下一个点，程序将自动绘制两点间的水平或垂直线段，大大提升的用户使用体验。

四、模块设计

1. 地图画板类的设计

地图画板类继承自 QLabel，绘图功能的实现如下：

按照 QLabel 的尺寸，声明一个空的 QPixmap,使其填充预设的底色，然后以 QPixmap 为画板，用 QPainter 在 QPixmap 上做图，再将 QPixmap 设置为 QLabel 的背景，从而实现图片的绘制与现实功能。

相较于传统的 QPainter 直接绘图方式，本方法充分利用了 QPixmap 的离屏绘图特性，绘图时，可创造专用的绘图子线程，在保证能够高效率地绘图的同时，又不影响用户其他操作的响应，做到了时间与空间效率的双赢。

若用户设置了设定的背景图片，该图片将被加载到初始声明的 QPixmap 中，作为其内容，后续绘图行为仍在此 QPixmap 上完成。根据各元素进栈的顺序，后来的元素会覆盖在原先的元素上，故所绘制的网格，地图、路径等内容将覆盖在背景上，从而实现了类似“图层”的效果。

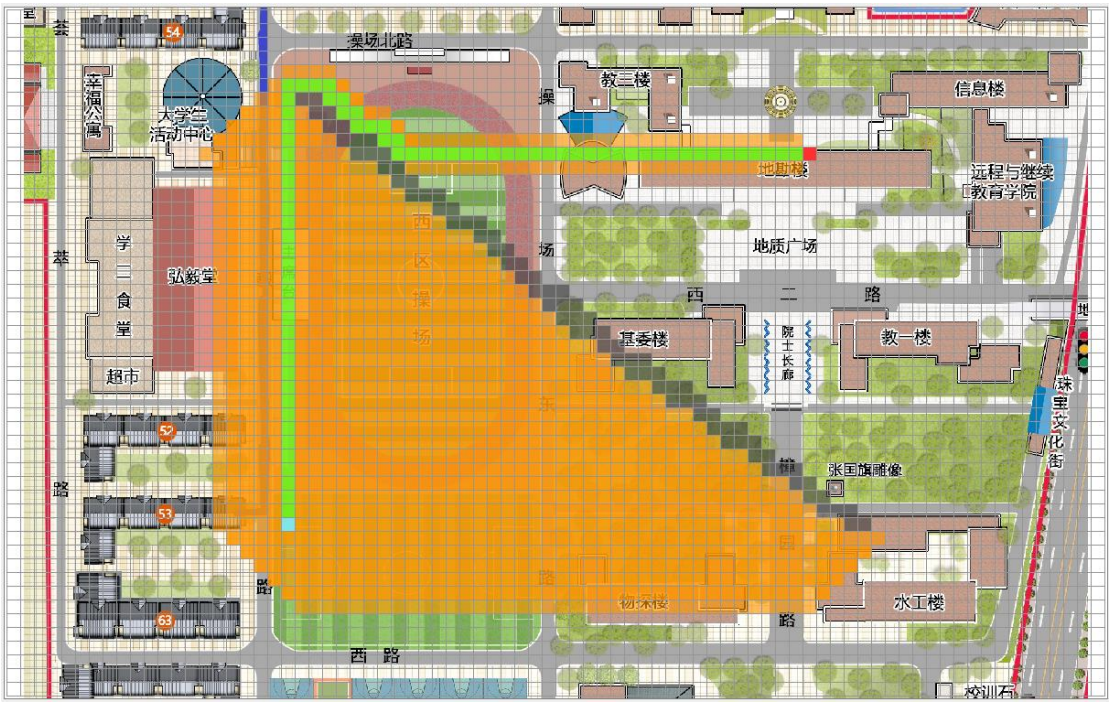


图 7 - 多图层的复杂图像生成效果

此外，我重载了该类的 resizeEvent，即尺寸变化事件。在外部主窗口尺寸发生变化时，其设置的窗口 Layout 将自动修改 World 画板的尺寸为合适的值，此时 resizeEvent 函数会被自动调用，根据此时画板的尺寸，以及画板网格的长宽，计算出最合适的网格大小，并将网格居中生成到恰当的位置。

为实现该类的功能，我在其声明中主要定义了以下函数及变量：

```
bool saveImage(const QString& filename) const;
bool saveData(const QString&, bool, bool, bool,
                          bool, bool) const;
bool loadData(const QString& filename);
bool setBgFlag(const QPixmap& image);

void findPath();
void stopFindPath();

void resetAll();
void clearPath();

void drawGrids();
void drawObstacle(int worldX, int worldY);
void drawVisited(int worldX, int worldY);
```

上述函数用来实现输出图片、输出数据、读取数据、设置背景、路径搜索、网格刷新、图形绘制等功能。

实现这些功能的成员函数具有良好的封装性，程序的逻辑部分仅需调用这些函数，便可完成上述功能的任意组合，实现更强大的功能。

2. 主窗口的设计

主窗口继承自 QMainWindow，通过菜单栏集成了输入输出控制、背景设置和窗口管理等控制开关，该窗口的主体是由地图画板类实例化的对象，是本程序的核心组成部分。

主窗口类设有 World 指针成员变量，在构造函数中将其生成。程序运行过程中，主窗口通过该指针调用地图画板的成员函数，控制其行为。

为实现以上功能，我在主窗口类的定义中声明了以下成员变量，用以响应用户的菜单操作，以及控制生成其他窗口。

```
private:
    World* world;
    ControlDialog* controlDialog;

    QAction* actionShowControlDialog;
    QAction* actionLoadData;
    QAction* actionLoadGray;
    QAction* actionSaveAsData;
    QAction* actionSaveAsImage;
    QAction* actionLoadBg;
    QAction* actionClearBg;
```

其中，World 和 ControlDialog 指针分别指向其构造函数中创建的地图画板与控制面板窗口，且地图画板先于控制面板生成，以 World 指针作为构造参数，生成控制面板窗口，以便后续组件间实现通讯。

定义中的七个 QAction，依次对应菜单栏的各个选项卡。它们将在类的构造函数中与相应的事件函数或 Lambda 表达式相连接，通过这些事件函数，实现预想的功能。

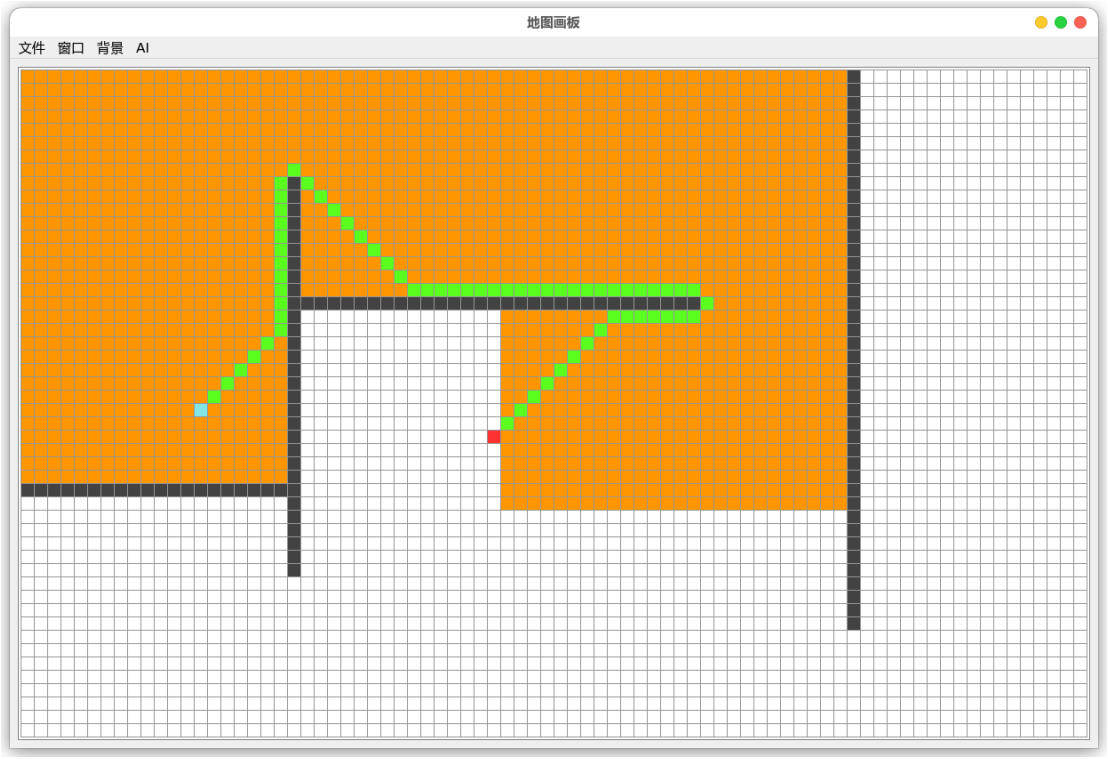


图 8 - 主窗口与地图画板的实现效果

3. 控制面板的设计

控制面板窗口集成了程序运行中可能用到的各种控制按钮与选项，其中即包含了地图画板的尺寸，颜色等外观选项，又包含了移动规则，寻路算法等逻辑控制选项。

我们可以在控制面板中选择寻路时可否对角线移动，是否可视化显示路径搜索过程，若显示，时间间隔为多少，还可以按照覆盖率，放置随机障碍点，以测试寻路算法性能，亦或是根据实际问题的需要，设置不同的距离算法。

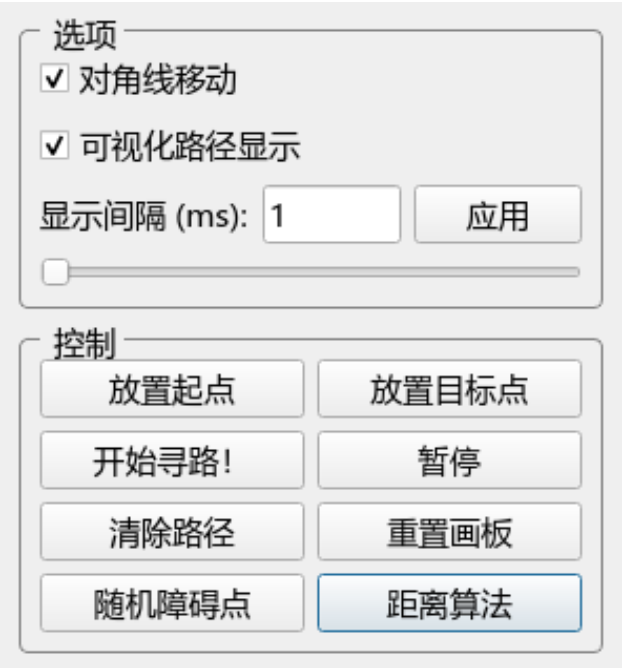


图 9 - 控制面板核心功能区

按下相应的按钮后，我们可以在弹出的对话框中设置 0.001 到 1 之间的障碍覆盖率，还可以根据实际需求，选择曼哈顿距离算法或者欧几里得距离算法。

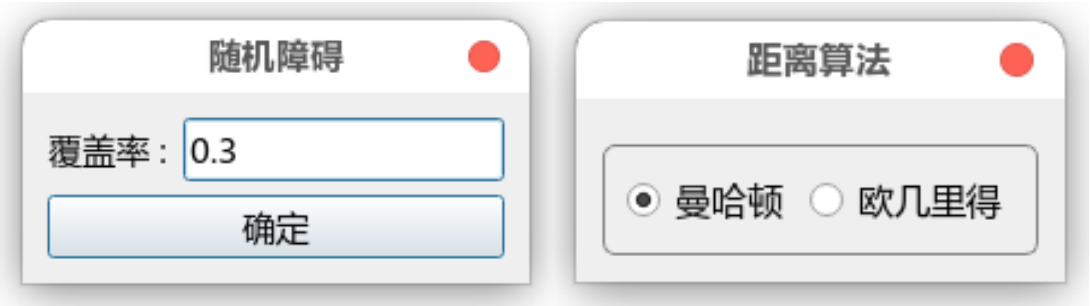


图 10 - 随机障碍和距离算法对话框

控制面板类有以下主要函数：

```
public slots:
    void onSetWorldSize();
    void onFindPathMethodChanged(QString);
    void logInfo(QString msg);
    void logError(QString msg);

public:
    void logMessage(QString msg, bool error = false);
```

这些槽函数实现了控制地图画板变更尺寸，修改寻路算法等功能，也向地图画板和主窗口提供了日志输出接口，以实现运行日志统一输出。

图 11 - 日志输出效果

4. 寻路算法的设计

A Star 算法

A Star 算法，或者称为 A*算法，是一种基于格子（Grid）的寻路算法，也就是说会把我们的地图看作是由 w * h 个格子组成的矩阵，因此寻得的路径也就是由一连串相邻的格子所组成的路径。

启发式搜索

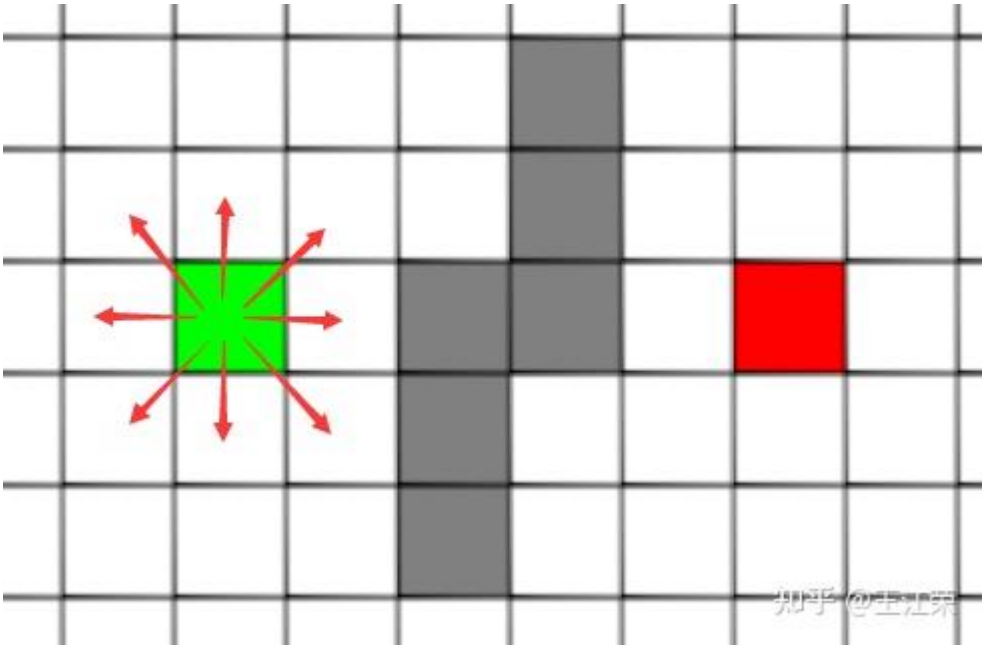


图 12 - 示例地图

我们知道一个格子可以往八个方向（开启对角线移动）移动，那么往哪个方向移动才能让我们更接近目标呢？例如如果我们沿着起点一直往左走，将永远到达不了目标点。

如果我们能在每次移动前做一个评估，则能有效减少移动距离。例如原本我们起点和目标点的直线距离为 5 个格子（不考虑障碍物），若向左走后距离变为 6，而向右走后变为 4，那么理论上我们应该向右走才能更快到达目标点。

通过评估来找到合适路径的算法我们称之为启发式算法，即优先搜索最有可能产生最佳路径的格子。A Star 正是这样的算法，因此可以避免掉很多歪路（不必要的计算），提高效率。

评估函数

前面我们说了要对每个可能到达的格子进行评估，来判断应该先往哪个格子走，因此我们需要一个评估函数来计算。

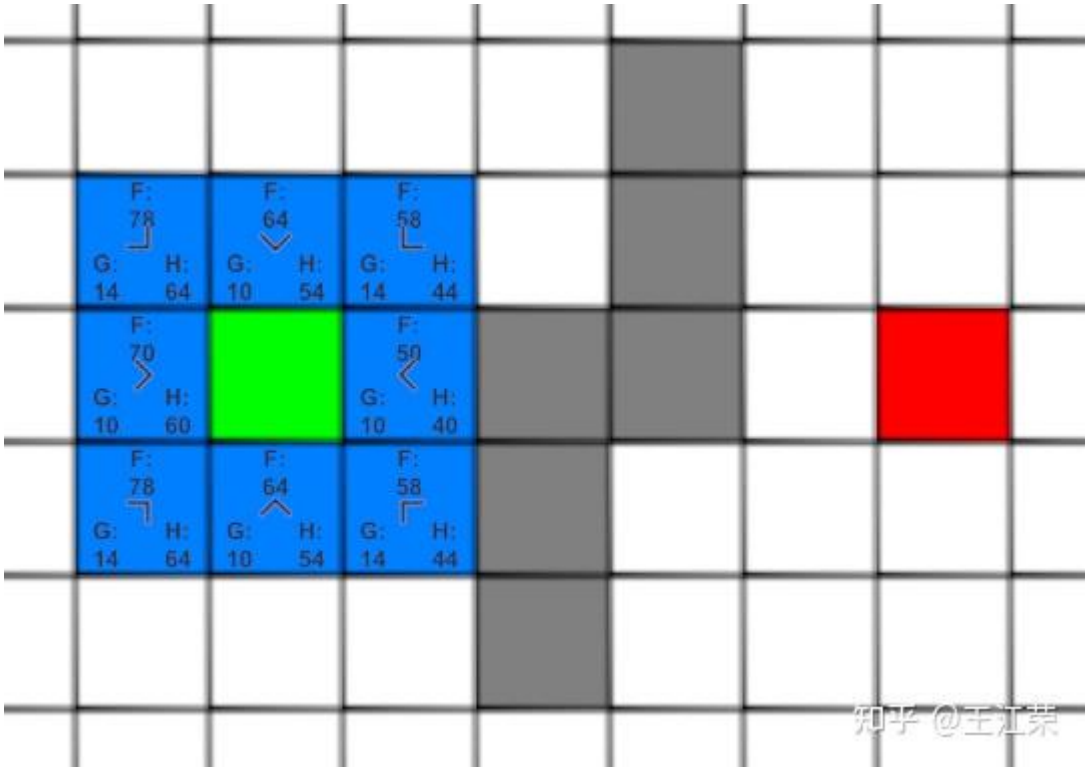
对于任意一个格子 n，其**评估函数**如下：

$$f(n) = g(n) + h(n)$$

其中 g(n) 指的是从起始格子到格子 n 的实际代价，而 h(n) 指的是从格子 n 到终点格子的估计代价。

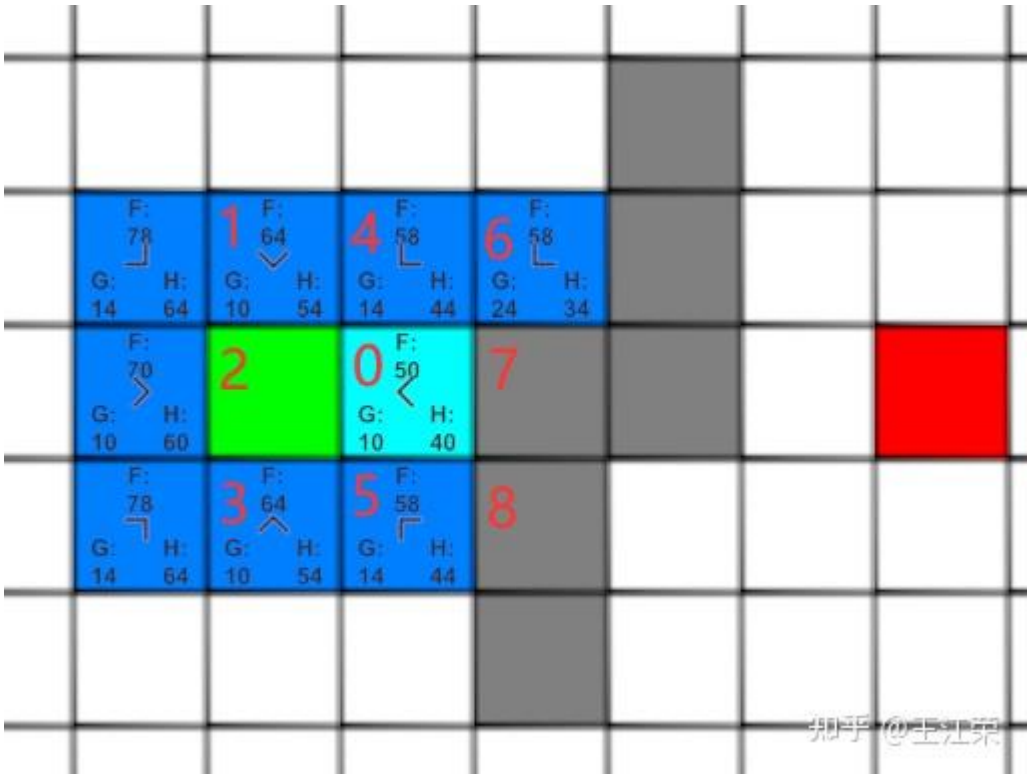
具体寻路过程

第一步：因为我们的起点可以往周边 8 个格子移动，那么我们就要用评估函数计算它周边格子的值，来看看往哪走比较好，得到结果如下（使用对角线距离评估）：



因为我们是通过绿色格子计算得到这 8 个格子的，因此它们都指向绿色格子（格子中的箭头），或者称绿色格子是它们的 parent。

第二步：我们找到第一步 8 个格子中 f(n)值最小的格子，然后再计算它周边格子的 f(n)，如下图：



显然，格子 0 的 f(n)值是最小的，所以我们接着计算格子 0 周围 8 个点的 f(n)值。

第三步：

我们从剩下的 8 个深蓝色的格子中再找出 f(n)最小的格子，并接着嵌套寻找周围 f(n)值最小的点，以此为基点向后迭代。

不断重复此过程，直至发现某个格子周边有个格子是终点格子。

最后一步：

我们把这个格子标记为最终路径，并沿着它的 parent 一直向上走，把经过的每个点都标记为路径，直到起始点，**这样，我们就得到了利用 A Star 算法计算的最短路径。**

若遍历所有可行点后，仍未到达终点，则说明在当前地图下，起点和目标点之间的合法路径不存在。

五、运行测试

1. 测试环境说明

系统类型： Manjaro Linux x86_64

内核版本： Linux Kernel 5.15

桌面环境： Gnome 42.2

2. 测试内容

a. 地图绘制

在控制面板的画板尺寸组合框中，输入画板的宽与高（默认是 80 和 50），点击旁边的设置按钮，主窗口中便会生成相应的地图网格。

鼠标左键按下并拖动，即可绘制障碍物；点击控制面板中的放置起点，再将光标移到适当的位置，按下左键，即可放置起点，同理可以在选定的位置放置目标点。

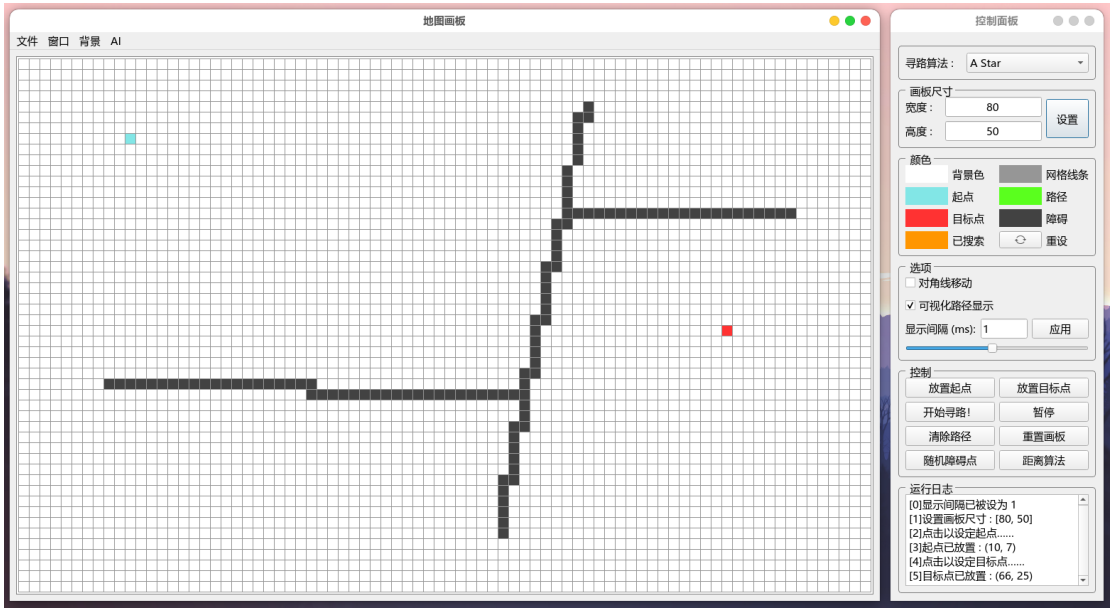


图 13 - 绘制地图

现在，一个简单的地图就绘制好了。

b. 简单寻路

地图绘制完毕后，我们直接点击控制面板的开始寻路按钮，寻路算法便会开始允许，短暂等待后即可在地图中描出找到的最佳路径，以及运行过程中遍历的点。

根据我们要解决的问题的实际情况，我们还可以选择合适的寻路算法（A Star 算法能适应大多数情况，故被设置为默认算法），并设置是否允许对角线移动（默认不允许），以及是否可视化现实路径搜索过程，和显示的时间间隔，更好的贴和实际应用场景。

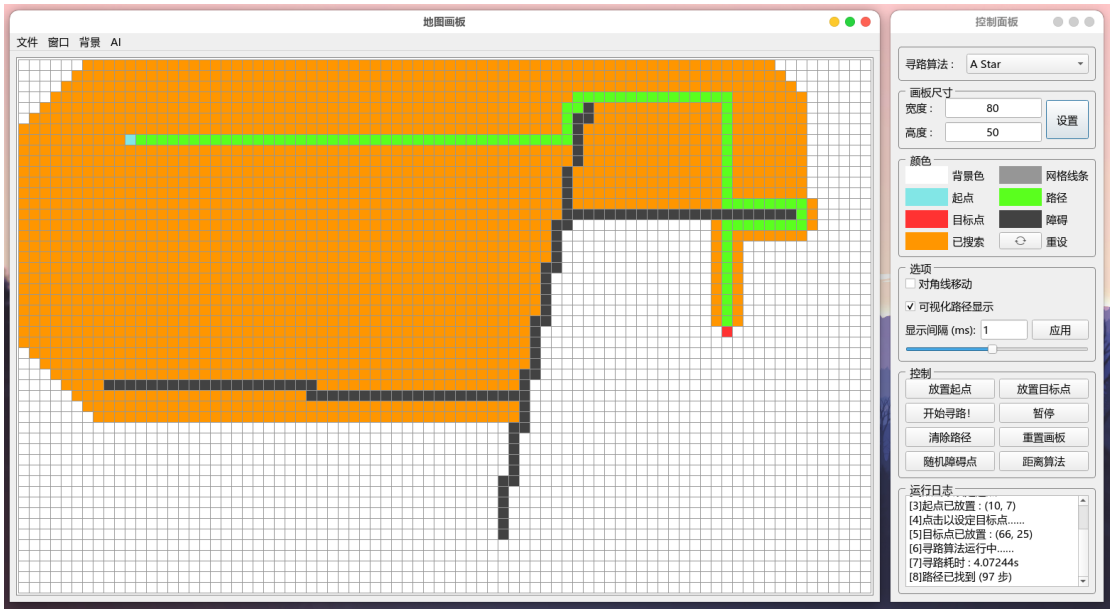


图 14 - A Star 算法寻路结果

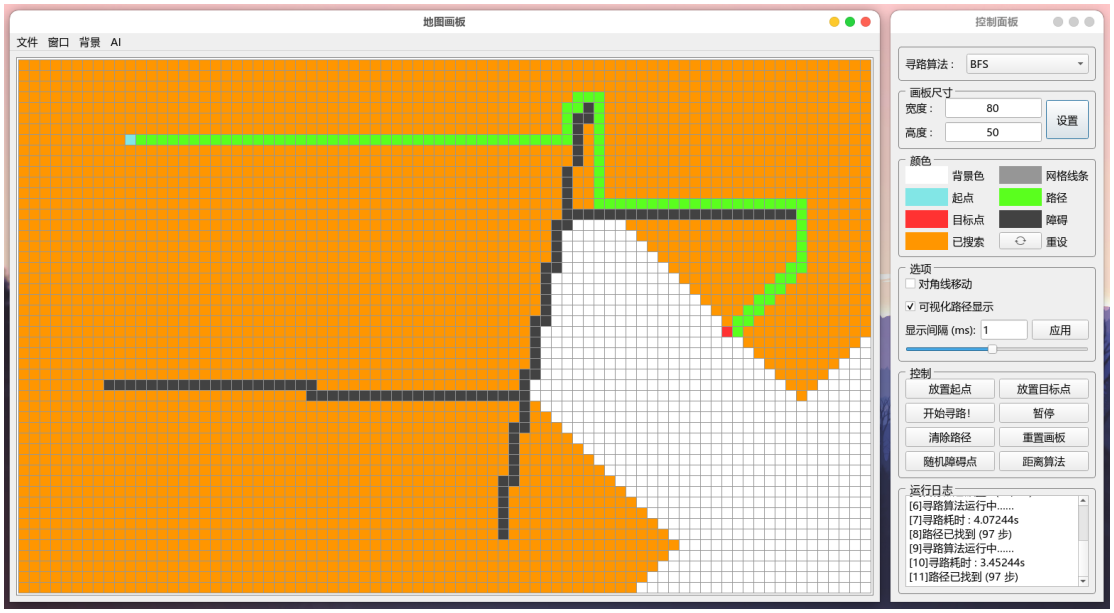


图 15 - BFS 算法寻路结果

我们注意到，分别使用 A Star 和 BFS 算法对同一地图进行路径搜索，虽然得到的路径是不同的，但两条路径的长度却是一致的，并且我们可以从数学上证明，此两种方法得到的最终路径一定是最优路径。

c. 添加背景

我们点击主窗口菜单栏的背景-打开图片选项，在弹出的文件对话框中选择我们的图片路径并打开。

这时，程序将调用图片裁剪对话框，允许我们裁剪原图片，选中我们需要的部分，作为地图画板的背景，增加使用时的灵活性。

图 16 - 图片裁剪对话框

待我们裁剪好图片，该对话框会自动销毁，此时地图画板已将背景设置为我们所选中的部分图片。

为了便于观测与绘制地图，载入图片后，画板各区域的颜色都会被设置为半透明，并置于背景的上层，以保证良好的可观测性。

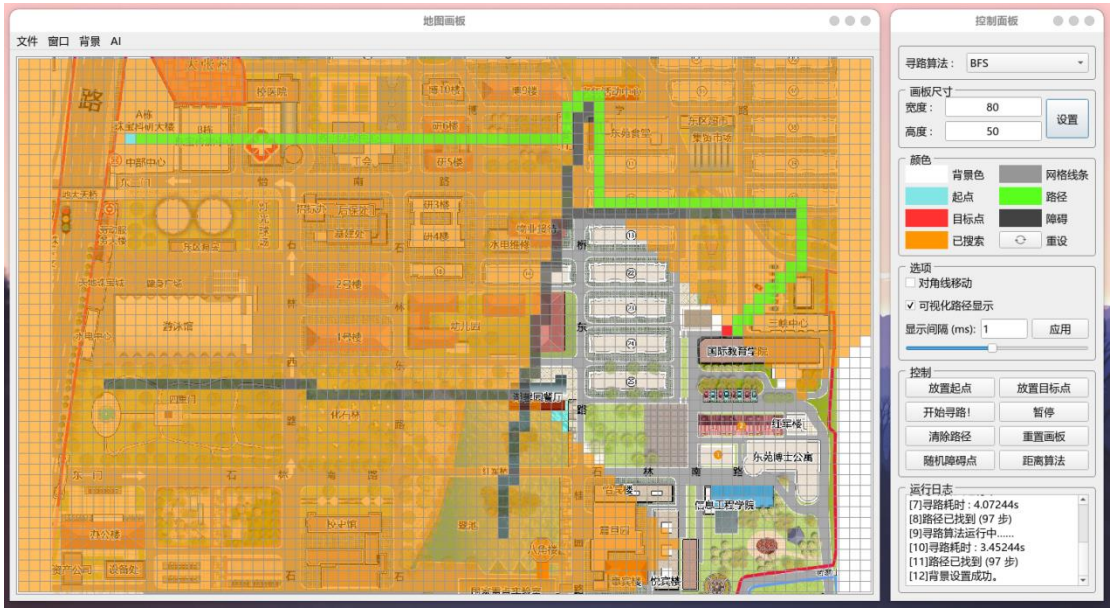


图 17 - 设置背景后的界面

背景设置成功后，我们还可以点击主窗口的背景-清除背景选项，这时地图画板的背景将恢复成控制面板中设定的背景色（默认是白色），且各区域的颜色将恢复原先的不透明度。

d. 随机障碍点

为了测试算法性能，和程序运行稳定性，我们可以生成随机障碍点地图进行测试。

点击控制面板中的随机障碍点按钮，在弹出的对话框中输入障碍点的覆盖率（范围从 0.001 到 1），点击确认，此时地图画板中便会随机放置一定量的障碍点。

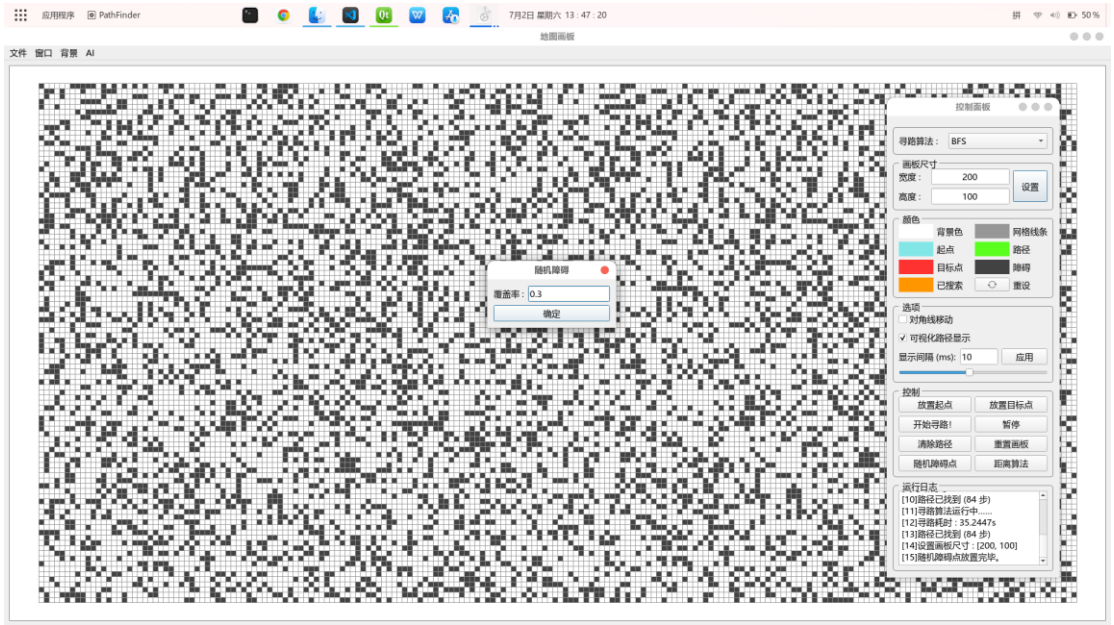


图 18 - 障碍点生成效果

我们点击开始寻路，程序便会在此复杂场景中寻找路径。在 A Star 算法的加持下，不用多久即可找到最佳路径。

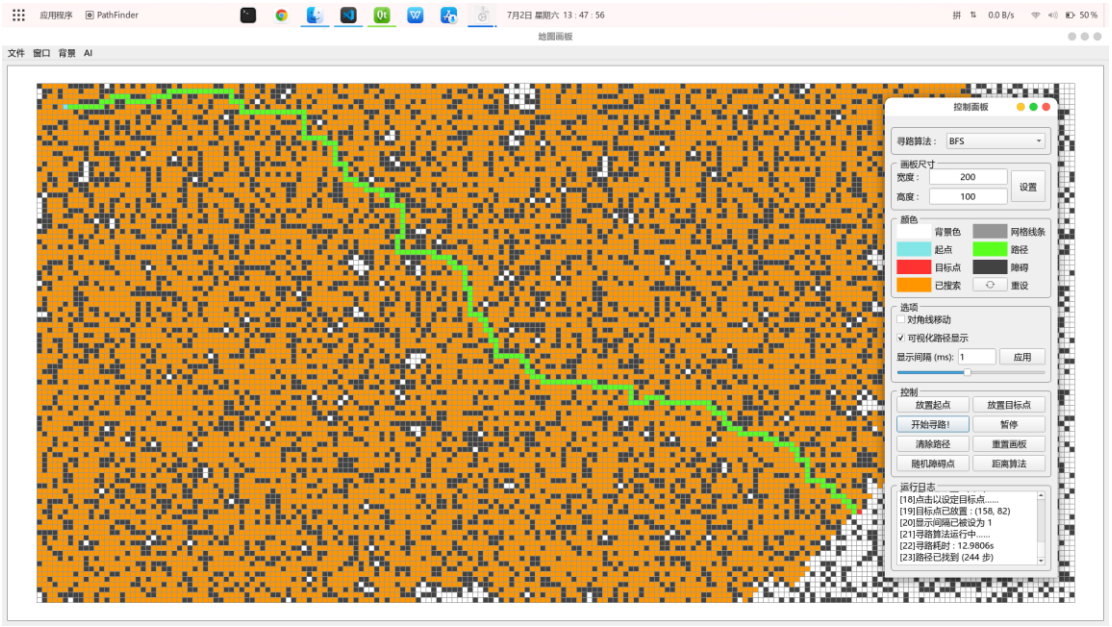


图 19 - 随机地图寻路结果

e. 智能识别

首先，我们需要调整画板的尺寸以更好地适应识别源图片，并适当的拉大窗口，使显示效果更好。接着，我们点击主窗口菜单栏的 AI 选项卡，选中智能识别，此时会弹出一个文件对话框，我们在对话框中选择智能识别源图片，再点击确认。

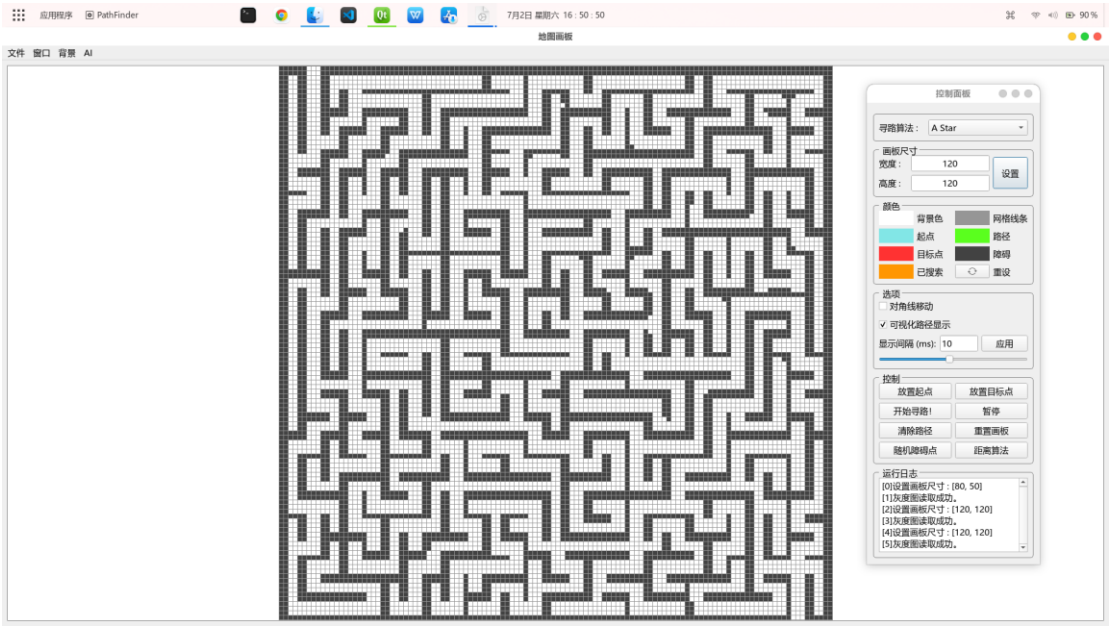


图 20 - 智能识别效果

此时，智能识别结果以显示在地图画板之中，我们可以对其修改、加工、美化，也可以设置起点、目标点后直接寻路。

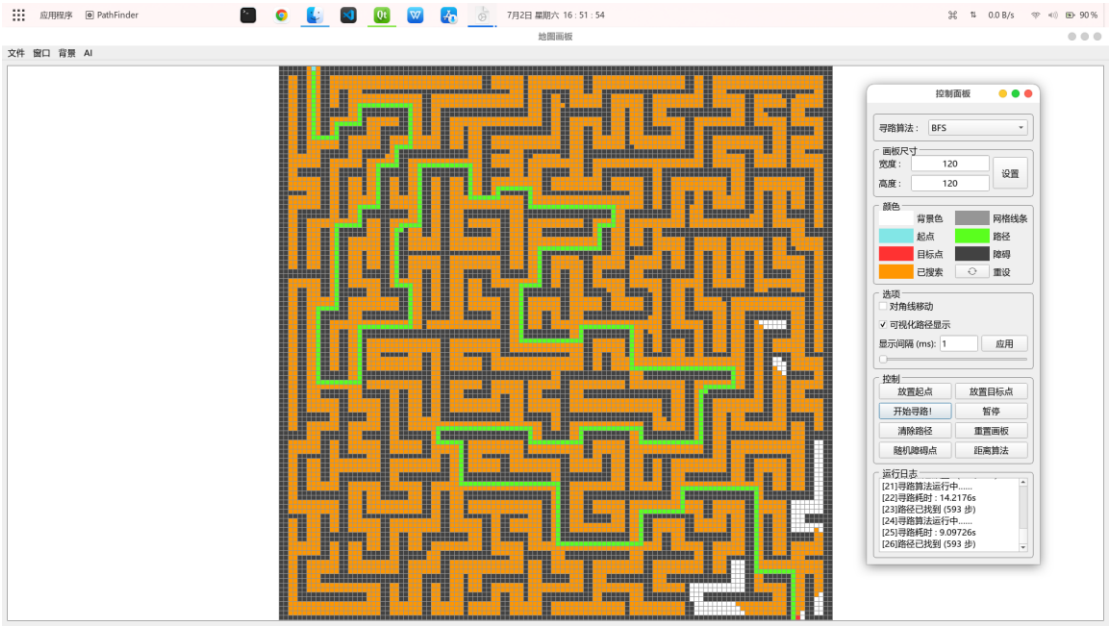


图 21 - 对智能识别结果进行寻路

只需片刻，智能识别读入的迷宫便被破解。

f. 自定义颜色

找到控制面板中的颜色组合框，双击需要更改的颜色标签，在弹出的颜色对话框中选择目标颜色后确认返回，便可修改指定区域的颜色。

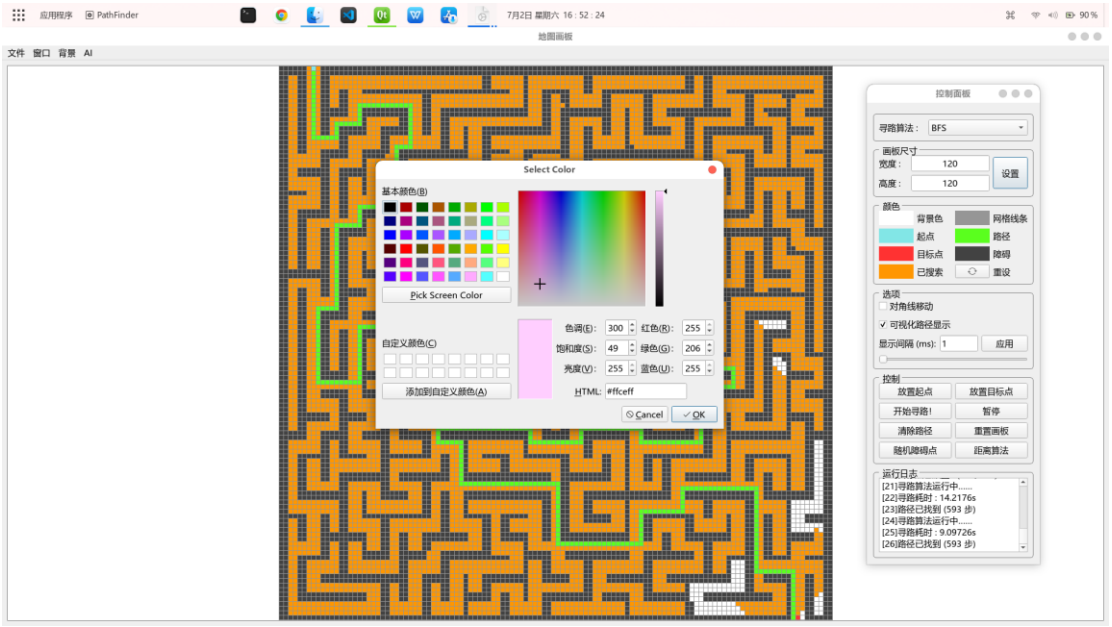


图 22 - 颜色选择对话框

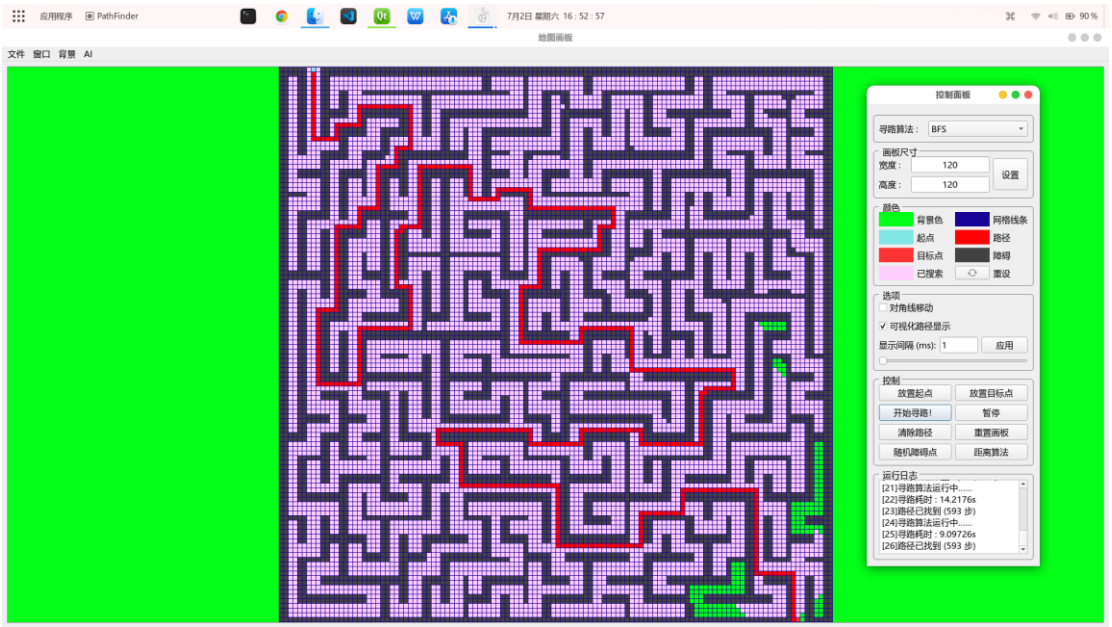


图 23 - 颜色修改效果

g. 保存数据

工作结束后，我们点击主窗口菜单栏的文件菜单，可以选择保存为图片或保存为数据选项，这里我们选择保存为数据，以便下次调用。

点击保存为数据后，会弹出保存数据对话框。

图 24 - 保存数据对话框

确认后，在弹出的文件对话框中设置我们要保存的位置和文件名，地图数据便会以二进制形式输出至指定位置。

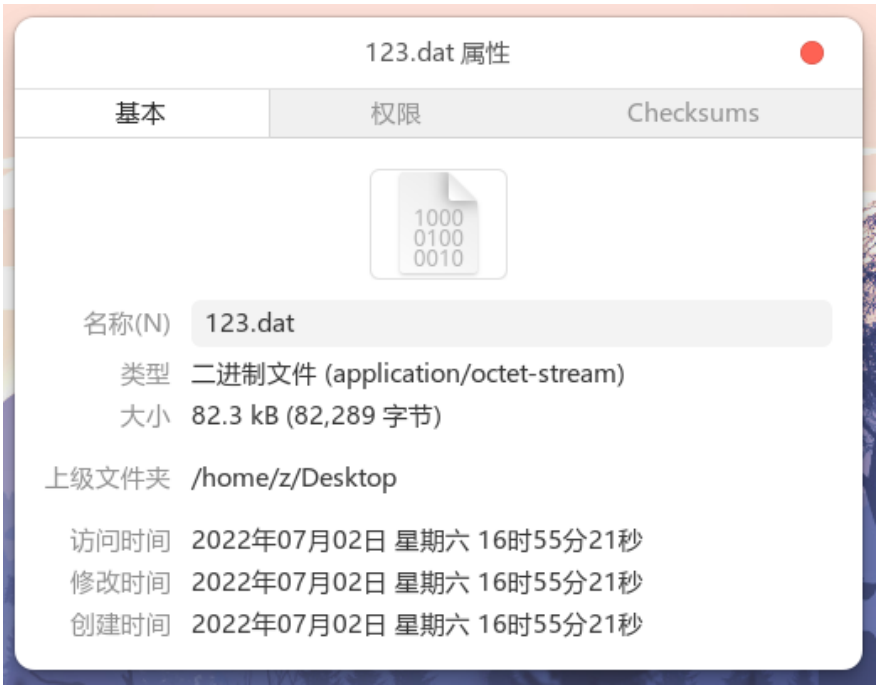


图 25 - 输出的二进制文件信息

h. 读取数据

重新打开程序，在主窗口的菜单栏中选择文件-读取数据，在弹出的文件对话框中选择我们此前保存的数据文件，点击确认即可。

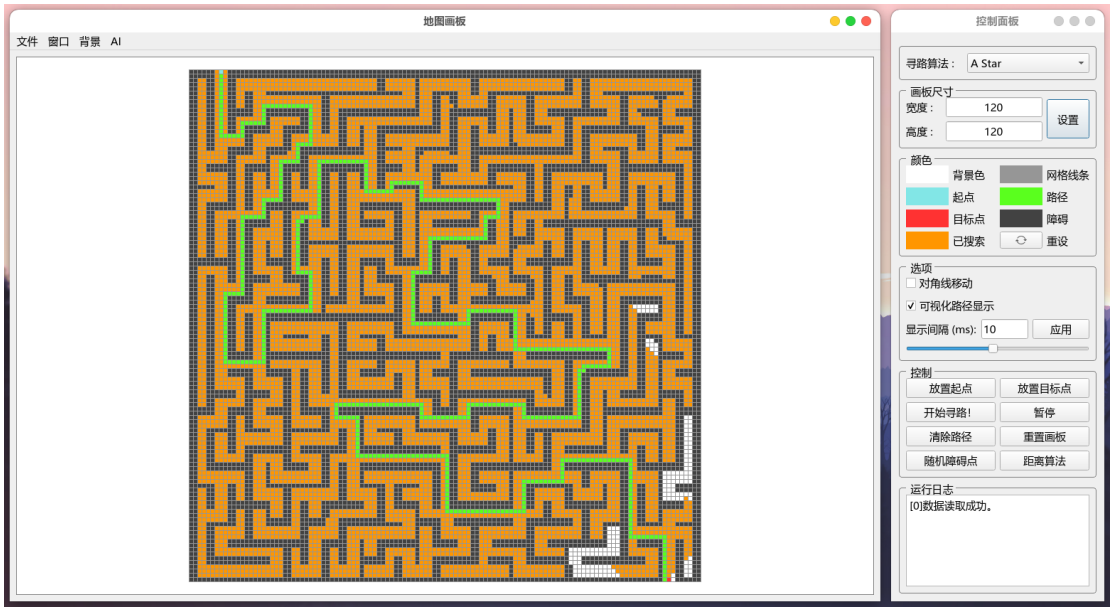


图 26 - 读取数据结果

至此，我们便回复了此前的工作环境。

六、心得体会

总体来说，这次课程设计对我来说是一个很大的挑战，同时也带来了很大的收获。我在上面前前后后投入了五周的时间，写了 3400+行的代码，经过这次锻炼，我对程序开发有了全新的理解。

针对此次课程设计，我选择使用 Qt 图形库基于 C++语言进行开发。经过一段时间的自学，我很快便能写出老师布置的三个小作业（计算器、记事本、画板），更为关键的是，我在这个过程中激发出了对 Qt 框架的浓厚兴趣，也深深的喜欢上了 Qt 的信号与槽机制，其自带的大量小巧而完善的类，为程序员提供了无限的可能。

得益于 Qt 的跨平台特性，我选择了 Linux 作为工作系统进行此次程序开发，学习了许多 Linux 系统的命令与调试技巧，并以此为契机，打造了一套 Linux 开发环境。

经历了前期的探索，我锁定了我的课设主题——可视化路径搜索系统。作为地理高考生，地图在我的学习经历中可谓再熟悉不过，将兴趣爱好与自己的专业结合起来，开发效率自然毋庸置疑。短短的五周时间内，我获取并学习了许多优秀的开源项目，并从中获得了很多新的思想与方法，最终一步步的实现了这个庞大的项目。

经历了这次课程设计，我对面向对象程序设计的理解到了一个全新的层面，独立开发能力也得到了极大的提升。此外，通过这次长战线的程序开发，我还学会了如何从互联网中获取信息，如何搜索关键词，如何高效地与人交流。为了支持开源社区建设，此次的大作业，我也将上传至我的 GitHub 仓库。

七、参考

1. 《Qt 5.9 C++开发指南》——王维波 等
2. 《A-Star（A*）寻路算法原理与实现》——王江荣/知乎
   https://zhuanlan.zhihu.com/p/385733813
3. ImageCropper——Leopard-C/GitHub
   https://github.com/Leopard-C/ImageCropper