**人工智慧在土木工程之應用**

**Final Project**

**使用A\*演算法尋找逃生路線**

**老師：洪士林 教授**

**學生: 0551283 林逸群**

1. **摘要**

人到了一個陌生的空間裡，尋找逃生的路線只要看地圖，但萬一是一個人無法進入的空間，則需要依賴機器人探索，機器人又是如何探索找到出口？現在主要還是由人操控，在未來，可能只需依賴機器人，因此讓機器人自行探索，會是一個有趣的問題，在找到出口之前，可能遇上各種障礙物。

本專題目的為應用在救災上，雖然有地圖，但在災難現場的未知是還是非常多，例如火源、崩塌、坑洞等無法通過的地區需要避開，考慮到無法真的使用機器人，因次用地圖模擬機器人路線，使用地圖編輯器假設以及設置一些障礙物，例如火災、崩塌、坑洞等無法通過之區域，並且設定開始位置以及探索目標順序，將地圖以網格的方式呈現，模擬真實情況。

1. **實作方法與設計**

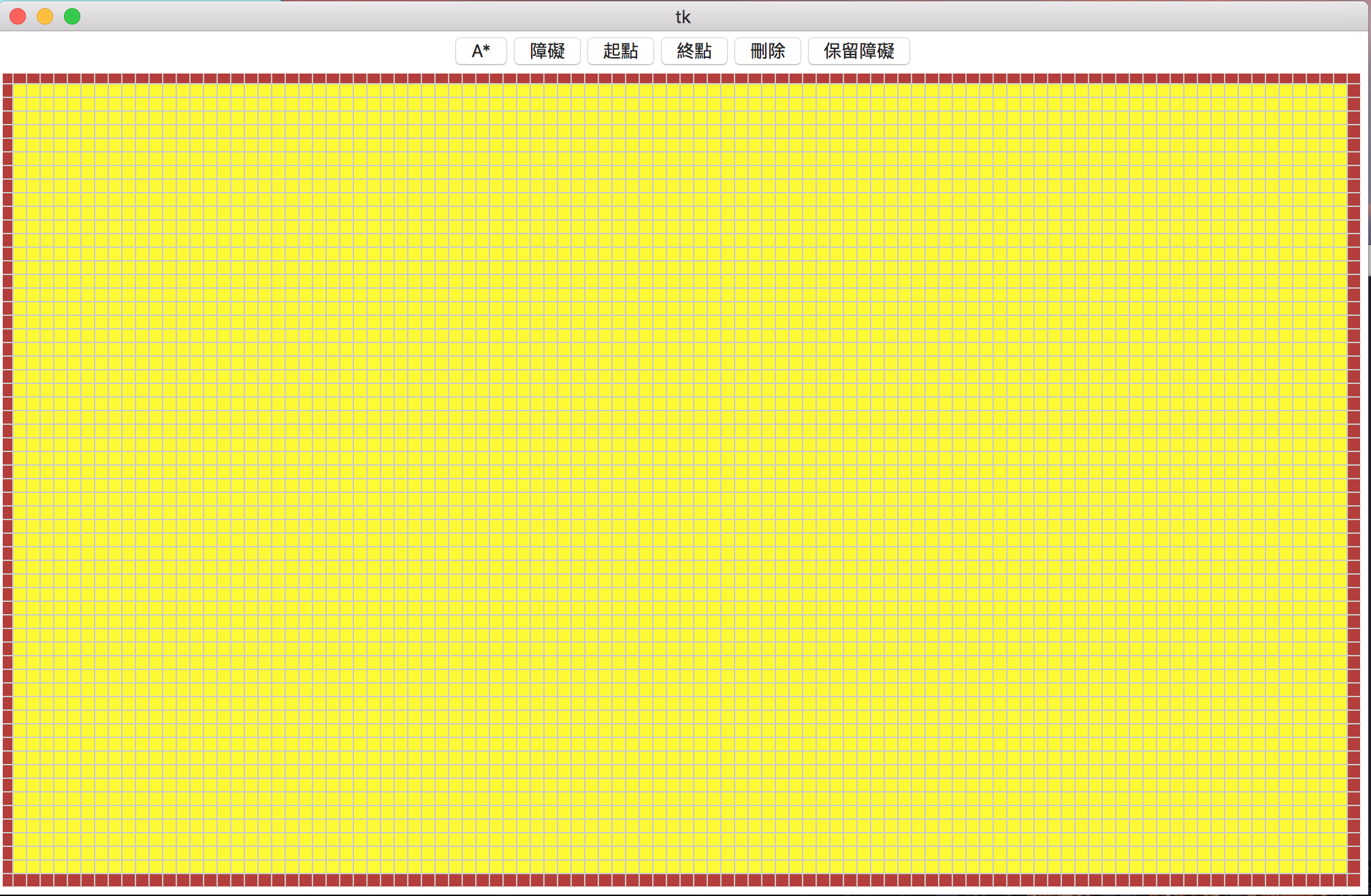
透過地圖編輯器，編輯房間、空間、障礙物，模擬真實情形的空間以及障礙物(火源、崩塌等)，本專題已python 語言撰寫，在地圖編輯器上設定開始、目標、障礙物，並使用A\*演算法進行探索。

1. **程式開發：**

本程式使用python 3.6版本撰寫，使用python3最簡潔的GUI套件Tkinter，因此在執行前要先確認電腦裡是否有Tkinter，執行指令為**python3 refactor.py 或是 python3 find\_goals.py**，程式碼在 git hub 上(<https://github.com/YICHUNLIN/Astar.git> ，Branch 切換到 multiGoal)。

1. **起始畫面：**

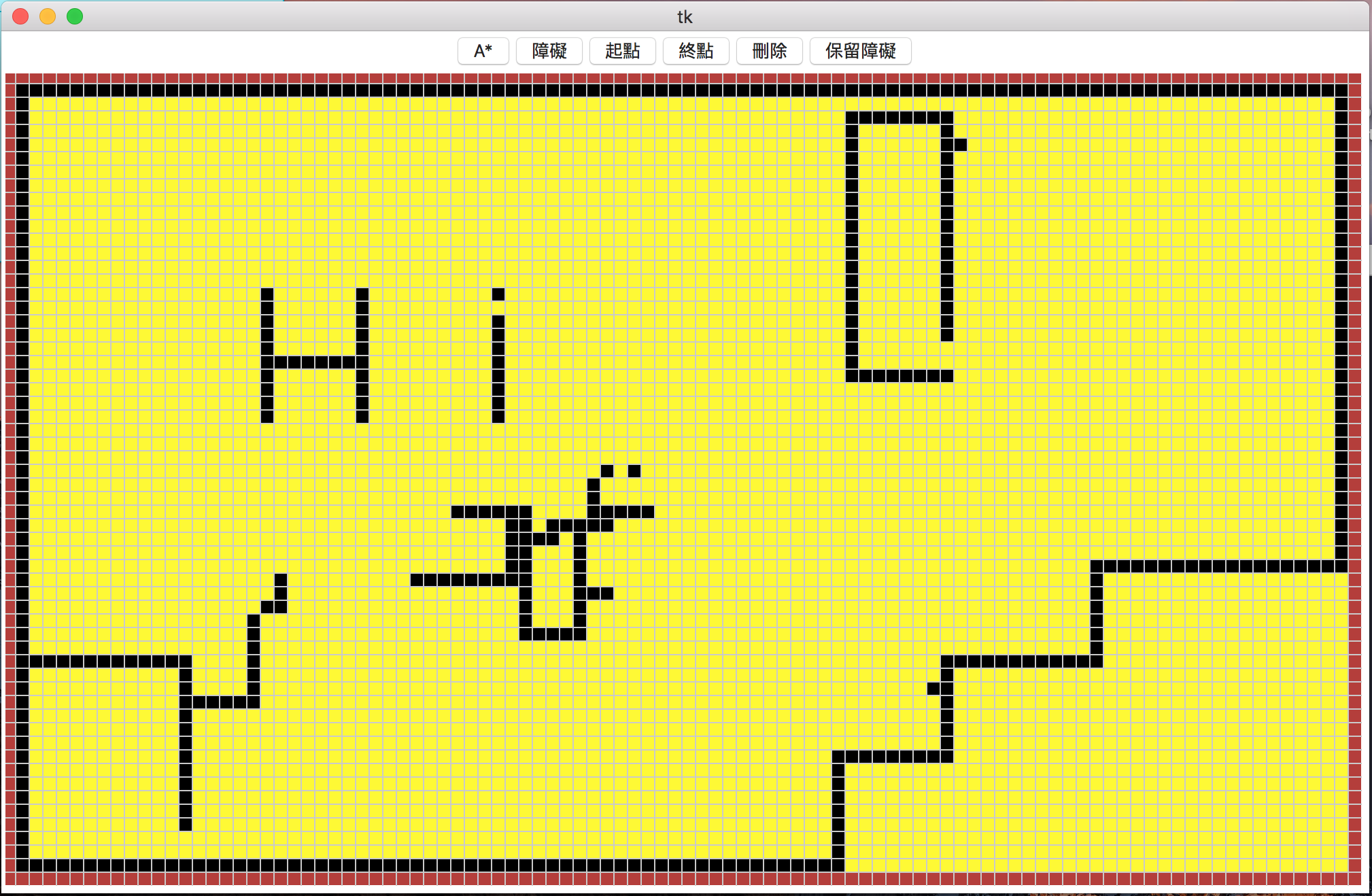
當執行程式後，會跳出這個畫面，上面有功能按鈕



圖一 起始畫面

1. **障礙物設定：**

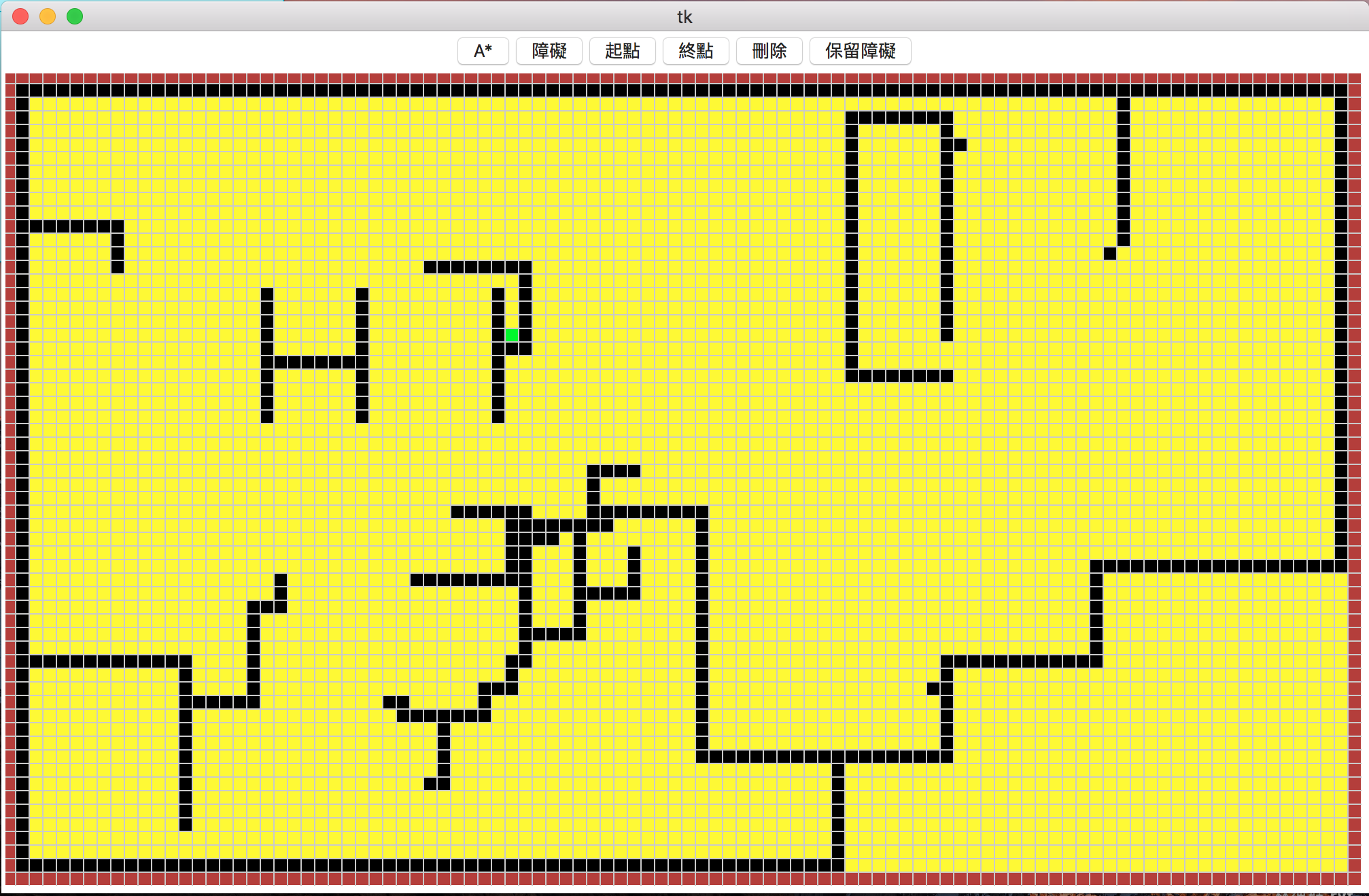
點擊障礙物按鈕，進入編輯障礙物模式，障礙物為黑色方塊



圖二 設定障礙物

1. **起點設定：**

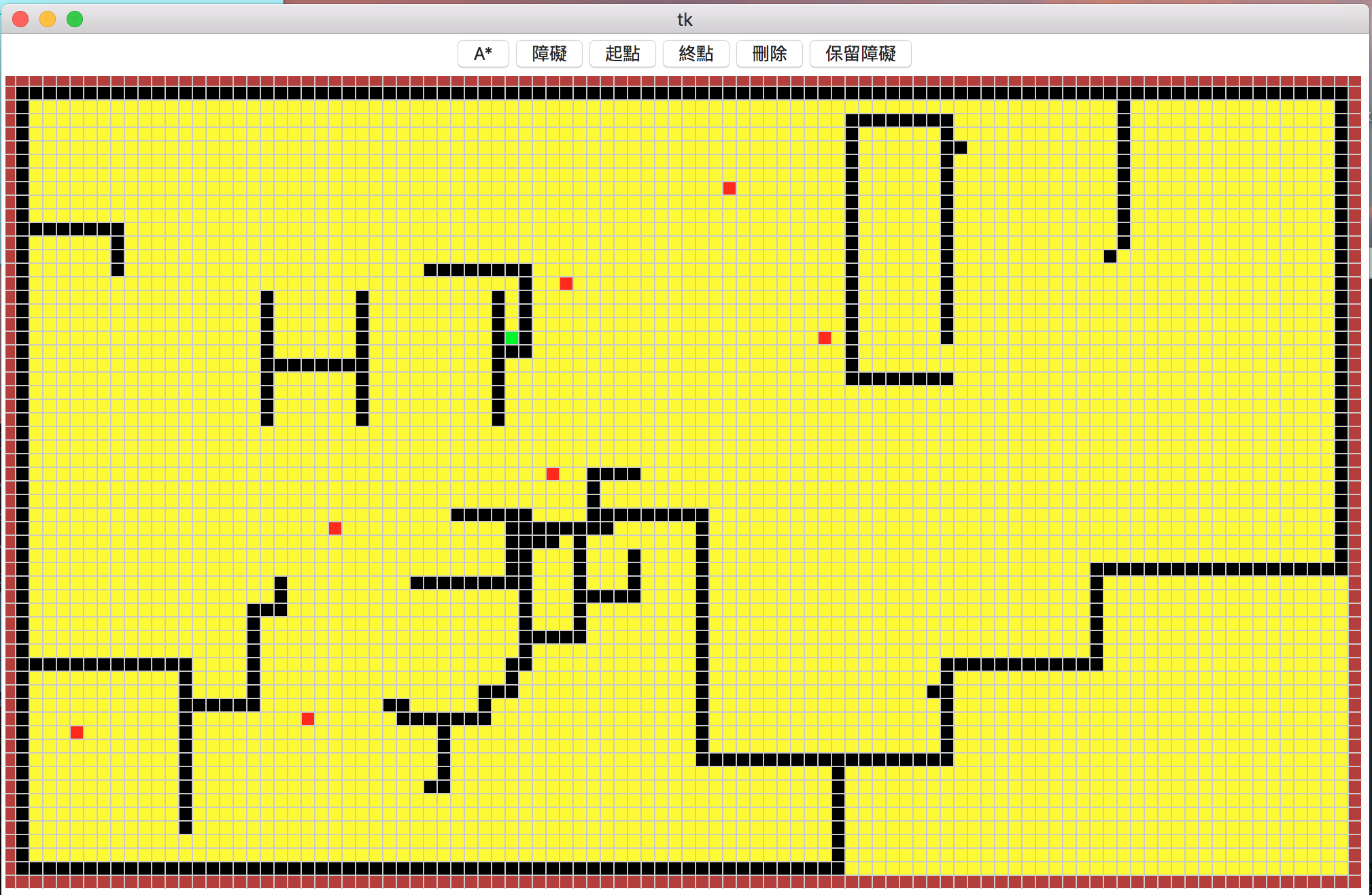
點擊起點按鈕，可以設定起點，起點為綠色方塊



圖三 設定起始點

1. **終點設定：**

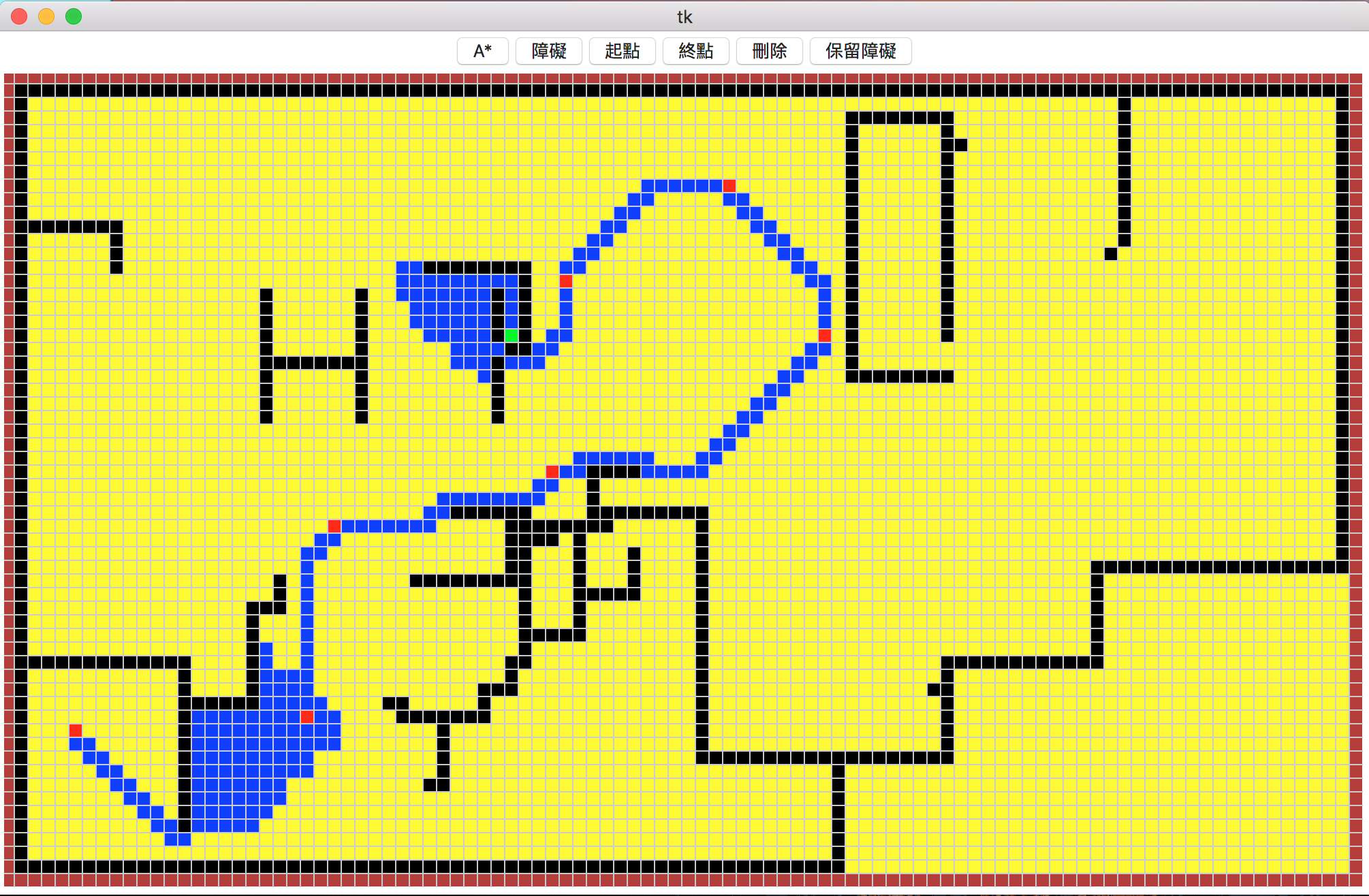
點擊終點按鈕，終點可以有很多個，會依照點擊順序進行



圖四 設定終點

1. **執行：**

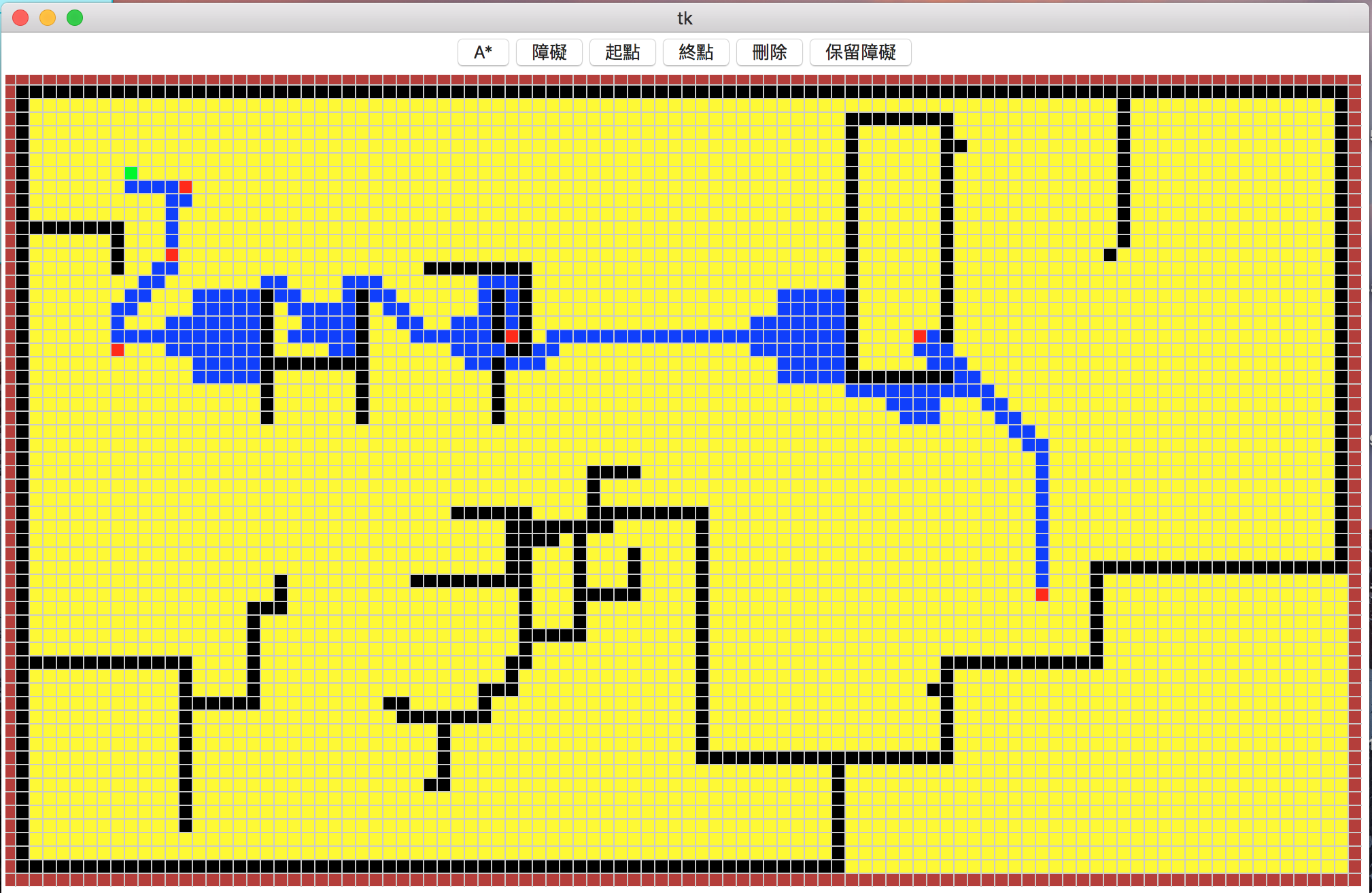
點擊A\*按鈕，從綠色方塊開始跑，根據終點設置順序移動，藍方塊為走過的路徑。



圖五 執行A\*

1. **刪除：**

點擊刪除按鈕，刪除全部，包括障礙物、起點、終點。點擊保留障礙按鈕，可以保留障礙物，重新設置起點及終點，可以重新執行搜尋。



圖六 刪除結果

1. **演算法**
2. **為何選擇A\*：**

在這個版本之前，有試過Dijkstra演算法，但是以悲劇收場，為何呢？因為Dijkstra演算法速度不夠快，範圍太廣，雖然最後仍可找到目標，但速度不夠快，太花時間，尤其是在尋找多目標的情況下，會發現整張地圖都被找過，相當沒效率，因此放棄。

BFS、DFS呢？沒有使用的原因也是因為效率上的問題，因為沒必要每個點都去搜尋，貪婪演算法就別說了，還有機會找不到。

1. **A\* 演算法：**

有玩過線上遊戲或是RPG(Role-Playing Game)遊戲的朋友就會知道，遊戲中尋找NPC(Non-Player Character)走路的路線就是使用A\*演算法或是Dijkstra演算法，相當有趣。

使用A\* 演算法就是因為快，而且如果能通過一定能找到目標，Dijkstra 演算法可以說是A\*的前身，也可以說A\*是Dijkstra的進化版，為什麼呢？A\* 決定下一步的方式G function 與 H function 組成，其中H function 為啟發是函數(**Heuristic**)，G為移動的代價(Cost)，F = G + H。

1. H值：

H值評估方法則是使用曼哈頓方法(Manhattan Method)計算出Manhattan distance，曼哈頓方法為現在位置加上目標位置 X 軸差以及Y軸差 相加再乘10，10\*(abs(start.x - goal.x) + abs(start.y – goal.y))。

1. G 值：

假設現在位置上下左右是10，左上、右上、左下、右下是14，因為走對角比較快，所以設為14。

1. Open List：

Open List 為有可能成為路線的一個陣列，再評估要如何走下一步時，需要從Open List 裡選擇一個最大值成為路徑，F = G + H，將擴展的點加入Open List，成為可能的路徑。

1. Close List：

Close List 為走過的路徑，從Open List 選擇最佳路徑放入Close List。

1. **Dijkstra 演算法：**

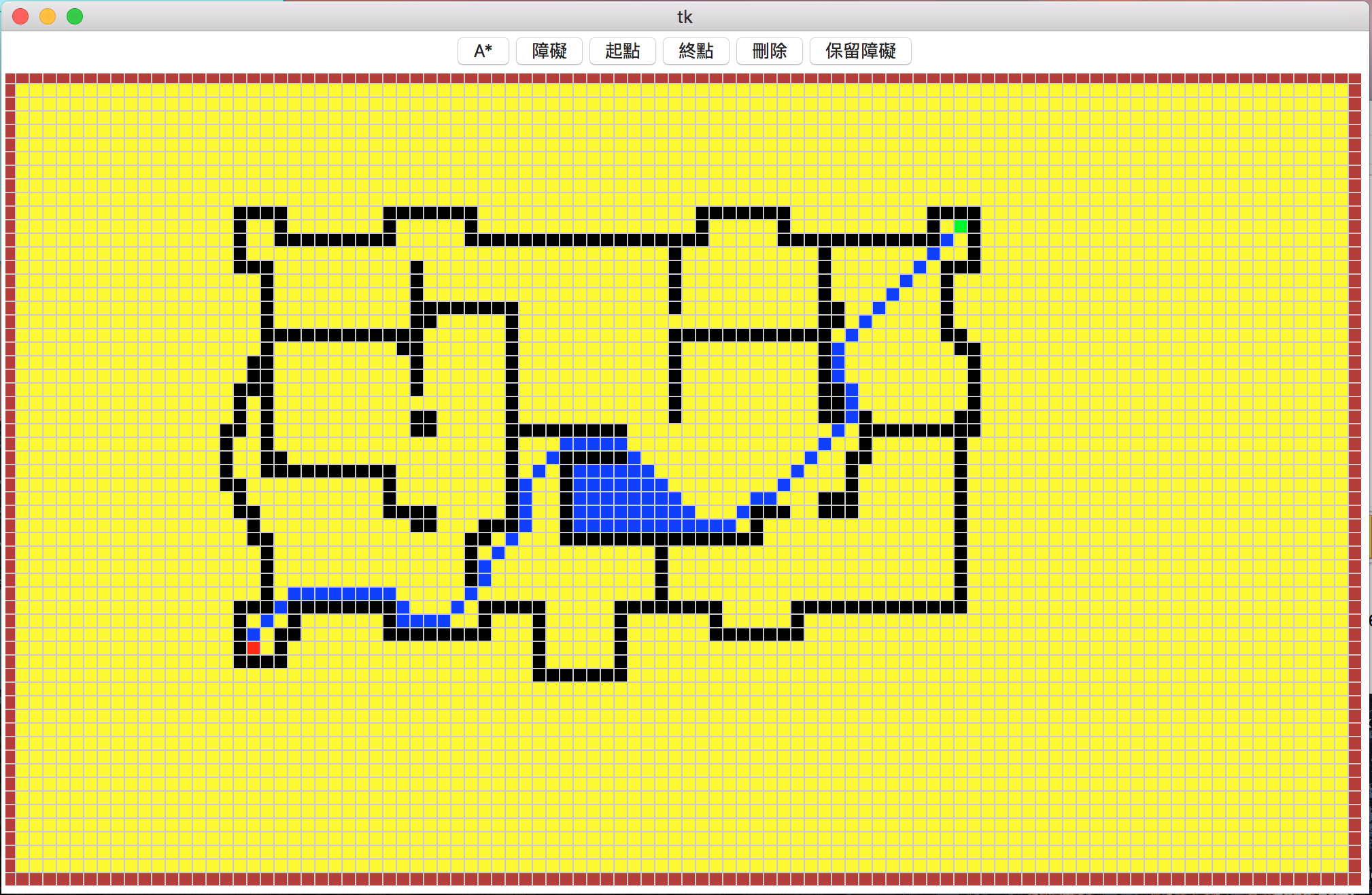
雖然Dijkstra演算法速度不及A\*，但也是一個好的演算法，若我們不知道目標，也就是說是一個沒有目標的搜索，Dijkstra演算法就會更適合，原因在於如果我們不知道哪個點最近，沒辦法使用A\*算距離，可以透過Dijkstra 計算每一區的距離，好比說Router的Routing Table 使用的方法就是Dijkstra，用它來算出封包該走的路徑，如何Route 比較省。

1. **H value**

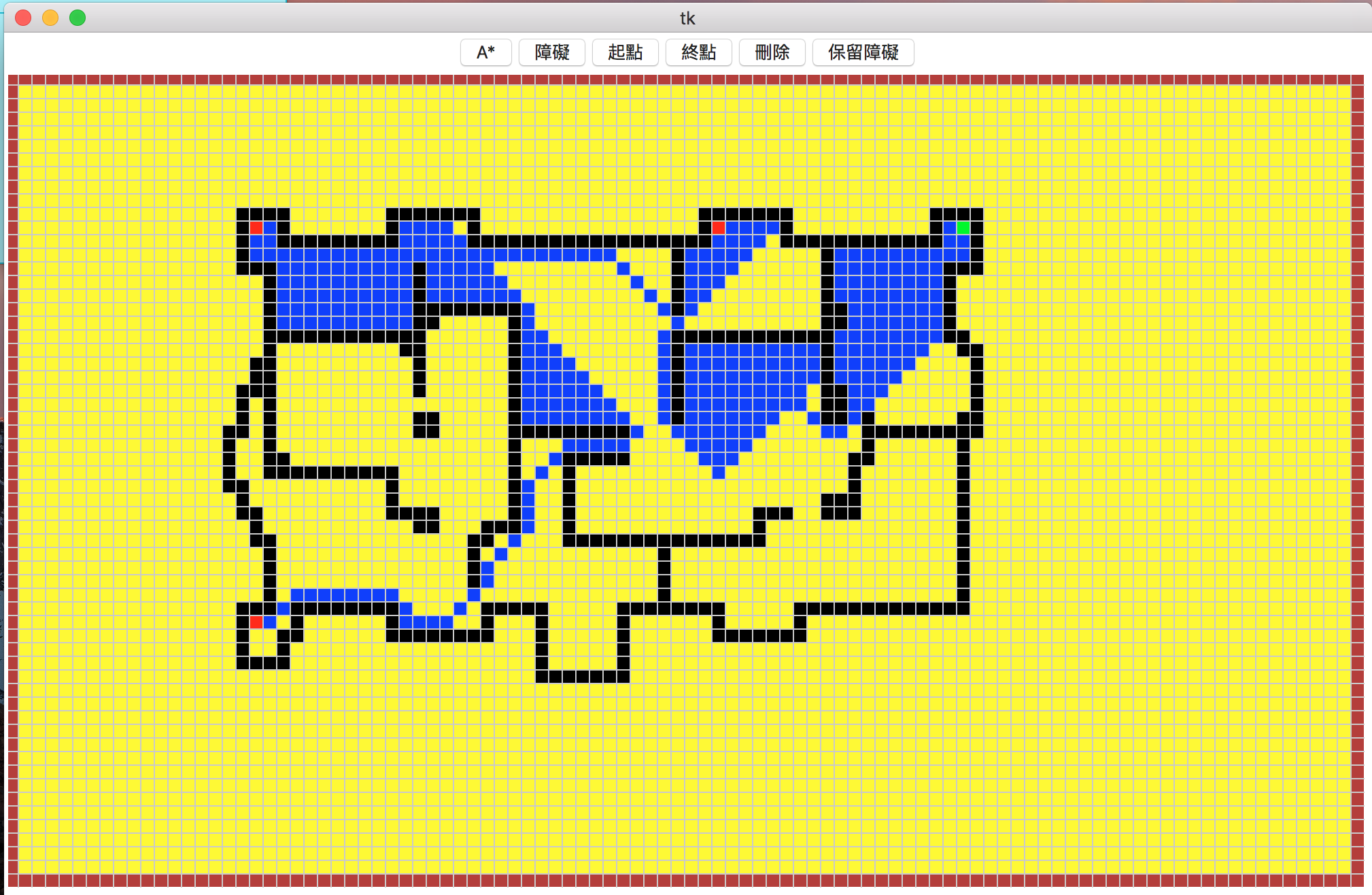
H value決定搜尋的速度，決定是否找不找得到目標，計算H value 的方法有很多種，例如 歐式距離、Cosine similarity、曼哈頓距離等方法，歐式距離與曼哈頓距離為普遍的使用，因為邏輯很簡單，未選擇 Cosine similarty 原因在於邏輯比較複雜，相較之下需要較大的運算量，Cosine similarty 經常被使用於相似度的運算，例如在做資迅檢索時，先將文章的關鍵字(特徵值)取出，將輸入另一篇文章的特徵值與目標文章特徵值做 Cosine similarty，得到相似分數。

1. **範例套用**
2. **建築物探索：**

可以透過放置障礙的方式繪製出房子的形狀。

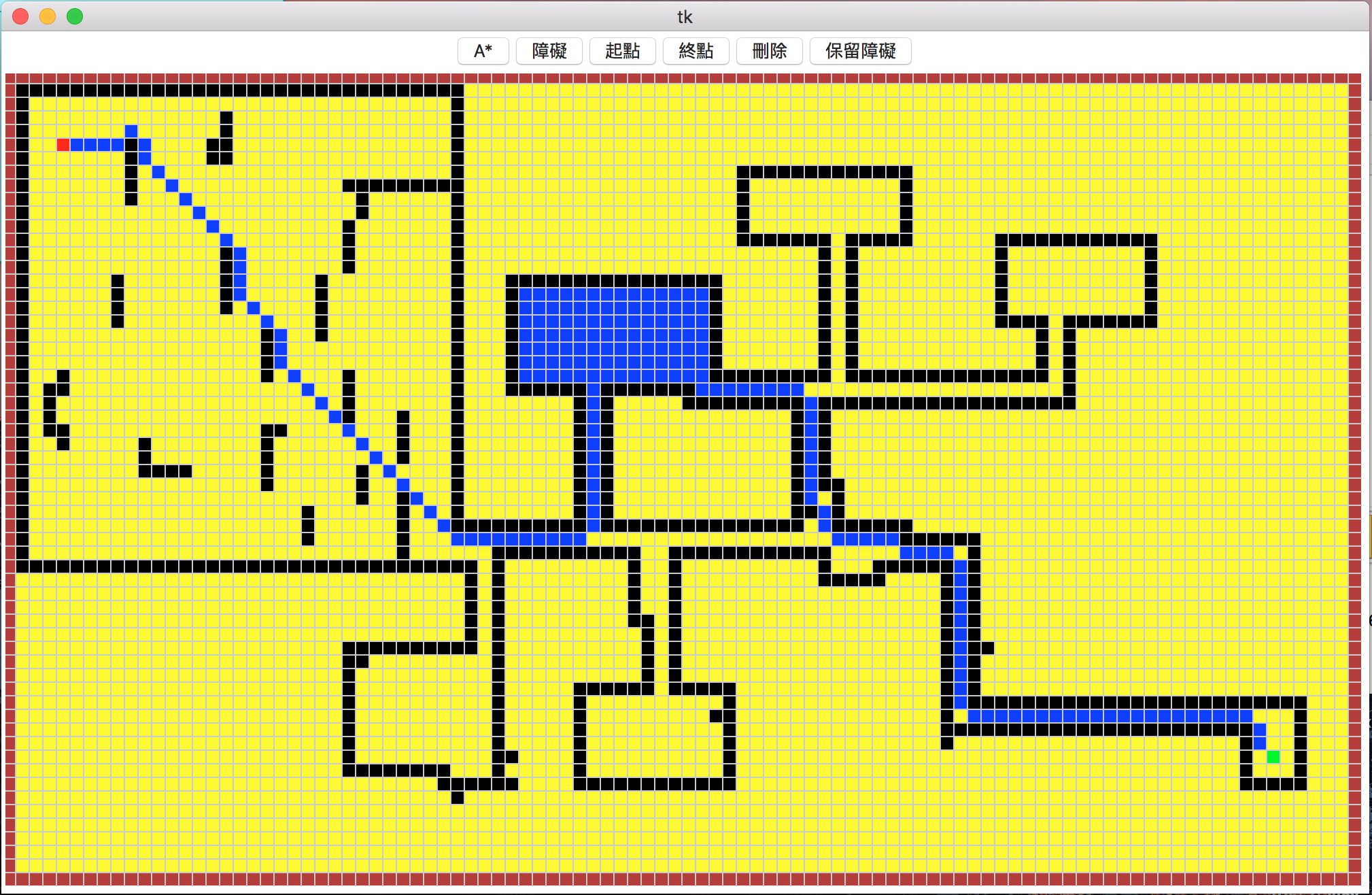


圖七 建築物探索ㄧ



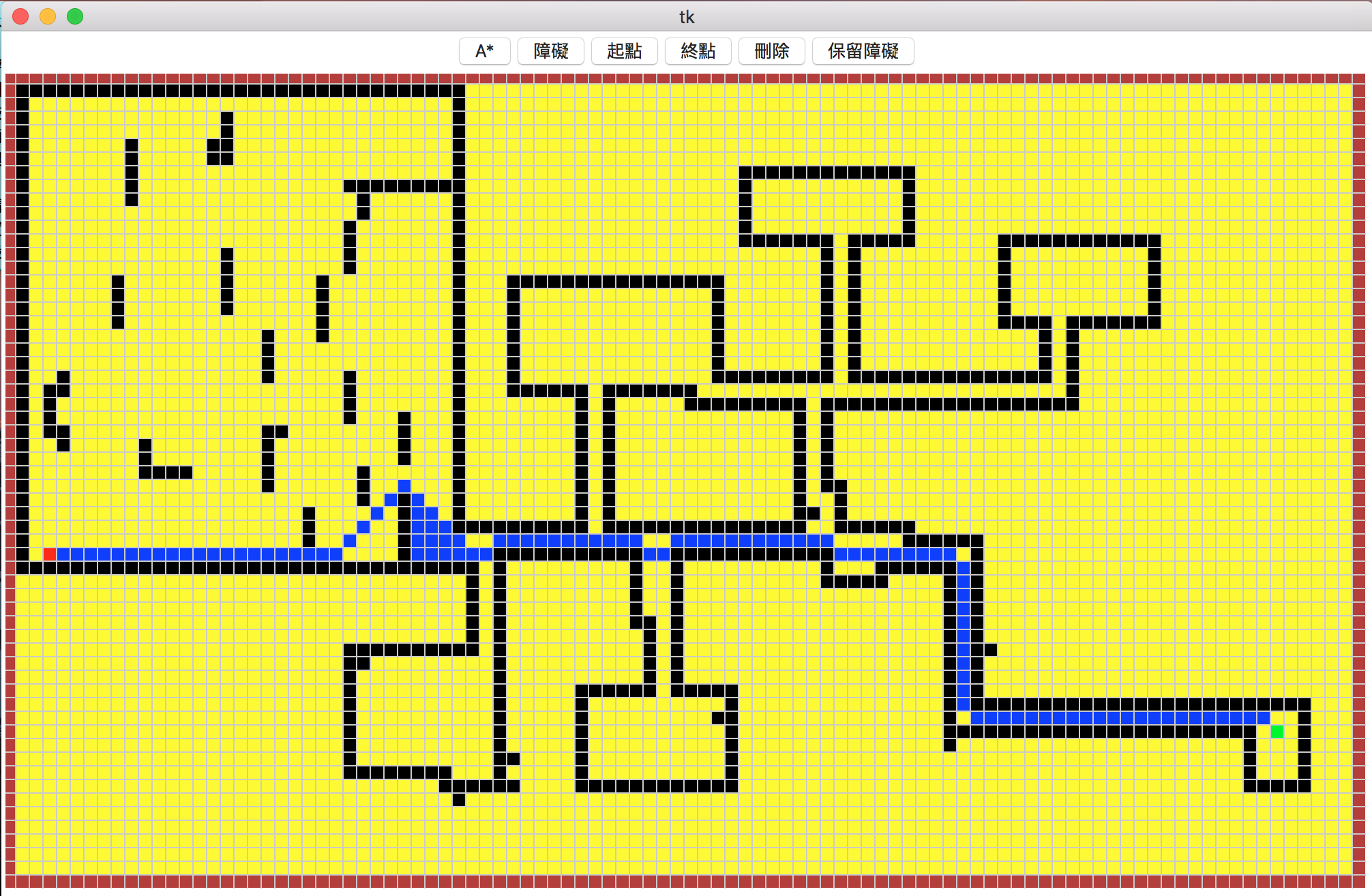
圖八 建築物探索二

1. **迷宮探索：**



圖九 迷宮探索一

當看到這張圖，會有個疑問，為何不走底下那條？不是比較近嗎? 因為H Function不同所以有所不同，再進入分岔前的當下，因為使用曼哈頓方法的緣故，機器人只知道上面那條路計算結果比較近，所以不走下面。



圖十 迷宮探索二

這次走下面那條路了。

1. 結論

演算法不虧是人類偉大的產物，能夠解決很多問題，雖然說Dijkstra 有些地方比不上A\*，但仍然有它存在的價值。Heuristic Function 不只使用曼哈頓方法，或許也能嘗試使用歐幾里德距離、切比雪夫距離、夾角餘旋等方法，使用相似度的方式去決定Heuristic Function，效果或許也會不同。

1. 心得

在課堂上聽到A\*演算法時，就覺得很興奮，為什麼？因為很親民，小的時候玩電玩，角色總是能夠自己閃避障礙物，並且找到NPC，覺得似曾相識，因為使用的技術就是A\*，所以決定實作A\*演算法。剛好這學期修python課，學了一些python程式設計，可以測試這學期的學期成果，因次就用python 的 tkinter 套件，做出一個地圖編輯器，並且在上面實作A\*演算法。

經過這個專題，對於A\*演算法與Dijkstra演算法更加了解，對於python 語言更加熟悉。

1. 參考資料

[1] A\*路徑探索 (<https://swf.com.tw/?p=67> )

[2] Wiki Dijkstra (<https://zh.wikipedia.org/wiki/戴克斯特拉算法> )

[3] 曼哈頓距離(<https://zh.wikipedia.org/wiki/曼哈頓距離> )

**附錄一 程式使用說明**

1. 使用前請先確定是否有安裝GUI 套件 Tkinter
2. 將程式碼下載 <https://github.com/YICHUNLIN/Astar.git>
   1. 將 Branch 切換至 multiGoal
   2. 如果有使用 git ，可以下指令

git clone <https://github.com/YICHUNLIN/Astar.git>

1. 執行
   1. Windows 使用 CMD 用 cd 進入Astar資料夾
   2. MacOSX 與其他Linux 系統 使用 console 或是 Terminal 進入Astar 資料夾
   3. 下指令 python3 refactor.py 或是 python3 find\_goals.py
2. 使用
   1. 功能列
   2. 必須設定起點與終點才能執行，起點只能有一個，終點可以有很多個，會依照設定順序跑
   3. 點擊刪除可以將所有東西(障礙、起點、終點刪除)
   4. 點擊保留障礙只會刪除起點與終點的設定，會保留障礙物
   5. 點擊A\* 可以開始執行
3. 結果