南台科技大學

電子工程研究所

碩士學位論文

電腦鼠迷宮演算法

Algorithms of Micromouse Maze Solving

研 究 生：吳一德

指導教授：黎靖

中華民國一〇〇年六月

南台科技大學

電子工程研究所

碩士學位論文

電腦鼠迷宮演算法

Algorithms of Micromouse Maze Solving

吳　一　德

指導教授：黎靖

中華民國一〇〇年六月

南台科技大學 碩士論文

電腦鼠迷宮演算法

研究生：吳一德

本論文業經審查及口試合格特此證明

論文考試委員

周榮華　　廖德祿

黎靖

指導教授　黎靖

所　　長　黎靖

中　華　民　國　一　〇　〇　年　六　月　二　十　八　日

摘　　要

本文探討電腦鼠走迷宮之迷宮演算法，探討的問題包括: 1. 未知迷宮狀態下的起點至終點搜尋演算法、2. 已知部分迷宮下如何有效率的搜尋未知迷宮的狀態、3. 已知迷宮下找出起點至終點間的最佳路徑等三大問題。對於問題1，本文提出四象限法，實驗結果顯示四象限法優於文獻上的其他方法；對於問題2，本文提出四象限洪水回程搜尋法，這是文獻上第一個解決問題2的方法。

關鍵字：迷宮搜尋演算法、輪型機器人、電腦鼠、迷宮最佳路徑

ABSTRACT

This article explains the maze solving algorithms used by the micromouse. The effectiveness of the algorithm will be determined by testing under following situations:1. When the maze structure is unknown, solve the maze.2. When only part of maze structure is known, find the path from the end point back to the starting point.3. When the maze structure is known, find the shortest path from the starting point to the end point. About according to question one, In this paper, The results showed that four-quadrant method is superior to other methods of reference, About according to question two, In this paper, four quadrant flooding backhaul search method, This reference of the first methods to solve the second problem.

Keyword(): Maze search algorithms, Wheel robot, Micromouse, Maze best path

致　　謝

本論文能夠完成，首先要感謝的是指導教授黎靖老師。老師嚴謹的治學態度，讓我不但在學術研究上學習到更謹慎的思考，也在日常生活上獲益良多。另外感謝成功大學廖德祿教授、周榮華教授的指導與建議，讓本論文能更盡完整。

感謝實驗室的成員們，季琦學長、偉翔學長、信強學長、宗書學長，雖然你們都已經畢業了，但是在此還是要謝謝你們，其中季琦學長、偉翔學長，記得你們快畢業的最後半年，認真在實驗室寫論文時，我時常的煩你們、鬧你們，你們都會回賣亂啦這句話；現在換我在寫論文時，才發現，論文真的要靜下心來才有辦法撰寫，不然寫完得到的結果都是亂七八糟、語句不通；在實驗的經過中，也要特別謝謝信強學長及宗書學長，時常的幫我解決任何問題；另外感謝跟我一起要畢業的戰友信宏，及剛踏入研究所一半旅程的晉嘉，跟你們一起研究電腦鼠及解決各種電子產品的問題，一路走來設計了數多種產品，得到了無數的佳績，也一起出國比賽，讓我的人生記錄下更多的風采；還有，實驗室神人勝中學長，樣樣都行，十八班武器，任何的電子問題你都有辦法解決，謝謝你在旁輔助解決實驗過程中的問題，讓產品能夠更順利，另外感謝老爹的陪伴，跟你從五專認識到現在，近乎 10 年了，你是我認識最久的友情，和你們認識是我很幸運的一件事，我會珍惜這份友情。

另外也謝謝阿寶，時常找我買雞排吃，雖然都都說說的，一直沒實現。也特別謝謝我的小玉，謝謝你的關心與體諒，讓我能沒有壓力地完成學位；還有飼養的貝貝及皮皮，雖然你們只會吃吃喝喝，心情不好就吐一下，心情好就叫一下，幫你們把屎把尿的從我研究所人生中度過。

我要把我最深的感謝留給我的家人。謝謝我的爸媽，你們讓我沒有經濟壓力地讀完碩士學位，時時常常關心我的論文完成度，今天我終於拿到這個學位，終於可以讓你們放下心上的一塊大石頭了。

目　　錄

表 目 錄

圖 目 錄

第一章 序論

電腦鼠競賽至今已超過 30 年的歷史，目前世界各地都舉辦各種大小型的電腦鼠競賽。其中亞洲地區的電腦鼠競賽，最為競爭的國家為中國大陸、日本、新加坡和台灣。日本已經舉辦了 31 屆的賽程，經由這些賽程的經歷，電腦鼠製作領域，目前被視為世界第一[1]；中國大陸也邁入三屆的電腦鼠競賽[2] ；台灣也舉辦了六屆的電腦鼠競賽[3] 。但是目前台灣製作的電腦鼠，無論加工技術及行走速度，還無法與日本較量。因此本文希望後續對於電腦鼠製作有興趣者，可以得到一些指引，讓台灣在電腦鼠領域能夠發揚光大。

電腦鼠的設計與製作可大致分成硬體與軟體兩部份，每一部分又各自包含了諸多不同的工程領域。硬體的任務主要是偵測電腦鼠週遭的環境並移動電腦鼠，軟體的功能則是理解迷宮環境，下達控制指令至不同的硬體子系統，導引電腦鼠在迷宮中行進。本文所提出的演算法，應用於台灣第六屆電腦鼠的競賽之中，且獲得佳績。

迷宮搜尋的演算法可應用於未來機器人的路徑搜尋，假如機器人處於一個未知的城市地下水道，機器人要經由交錯複雜的地下水道，找尋城市中某一處的出水口，並且要在此出水口進行管道的疏通，且留下清理過的紀錄，下次此機器人還要再次清理此出水口時，只需運用已知的地下水道路徑，找尋最佳路徑，直接移動到目的地；迷宮搜尋演算法另一方面也可以輔助電路設計的自動佈線，試著嘗試每一條電路的最佳路徑，找尋點與點之間的最佳連接橋樑，簡化人工佈線。

迷宮搜尋演算法可分為三大類：1.未知迷宮狀態下的起點至終點搜尋演算法、2.已知部分迷宮下如何有效率的搜尋未知迷宮的狀態、3.已知迷宮下找出起點至終點間的最佳路徑等。文獻上的探討多半集中在問題 1 的探討，常用的迷宮搜尋演算法有左手法則、右手法則及向心法則、A*演算法，與其諸多變形的 A*演算法等。問題 2 則未見到有探討此問題之文獻。問題 3 則可以使用洪水填充法(Flood-Fill algorithm)[4-8]的到最佳解，因此可被視為已經被克服。

現有的迷宮搜尋演算法，大多數的設計都必須將迷宮資料及演算法資料，個別存入微控制器的記憶體中，經由演算法的處理和修正更改資料內容，並挑選出最好的道路。從理論上講，這些演算法似乎沒有問題，可以很好地工作在任何迷宮上。但是，由於反覆的記錄、修正和管理整個資料，其過程非常複雜，容易犯錯誤又費時。基於這些原因，這些演算法是不容易在實際中進行，因此文獻[9]提出了一種高效率的迷宮搜尋演算法，找出迷宮終點且沒有記錄，利用公式計算出目前座標離中心的權重，與其向心法雷同作法，向心法是預先計算後存放權重於迷宮記憶體中，此文獻則反之；但是就算是使用較不會出錯的高效率演算法，將此演算法直接實際應用在實體迷宮中，直接利用電腦鼠身上的微控制器去進行除錯及撰寫程式，這樣的設計方式會大大的降低迷宮搜尋演算法的效率，因此就算在簡單的迷宮搜尋演算法，運用於實際電腦鼠之前，還是得事先使用各種軟體介面，設計迷宮搜尋演算法的軟體模擬介面 [10]。

文獻[11]提出了節省運算時間及記憶體的一種迷宮搜尋演算法，起初將每個迷宮方格位置分配初始值，接著當電腦鼠搜尋至新的迷宮方格時，確認目前的迷宮狀態以及初始分配值，了解目前迷宮資料後，會去判斷及建立相鄰邊的虛擬障礙，並且更改迷宮分配的初始值，這就有如向心法則的應用，直到電腦鼠搜尋至迷宮終點。

已知的迷宮中，找出兩點間的最佳路徑，文獻[12-13]說明了一個很好的方法，利用洪水流動的概念，將已知的迷宮注入洪水，此方法稱為洪水填充法；計算由 A 點至 B 點的最佳路徑，起初將 A 點注入洪水，並且依序的紀錄洪水流動的順序，直到遇到交叉路口，進而洪水產生了支線，以同等的流動順序繼續紀錄，當支線遇到無法繼續流動、或者與其他支線的洪水相碰時，才停止支線的流動，反覆的應用此方法，直到洪水遇到終點為止，當洪水填充法遇到終點結束後，只要沿著洪水的流動順序，由大至小或者由小至大，則可以知道已知迷宮中的最佳路徑。

文獻[14]將迷宮的每個位置個別放置移動方向，這方法就有如向心搜尋演算法的方式，起初建立一個相同大小的迷宮，接著在每個迷宮的位置各別放置固定的方向，當迷宮搜尋至該位置時，則取出該位置的方向，假設該方向有路徑可行走，因而成立條件，移動至下一處的迷宮位置，直到行走至終點；使用該方法需使用

較多的記憶體，來存放迷宮位置的資料，但此方法靈活性較不大，無法較快速的搜尋到終點，很容易行走更多的路徑，但唯一優勢的是可以節省微控制器的計算時間，運行至每個位置的時候，只需取出該位置的方向，大大的省掉複雜的運算過程。

目前大家應用迷宮搜尋時的演算法，大多偏向洪水填充法及向心法搜尋法，文獻[15]則將此兩種法則以及簡單的左手法則、右手法則，個別舉例說明，且應用於相同的迷宮，並分別將這四種法則的優缺點個別加以說明，統計出每個法則的效率及運算時間，這是個非常值得參考的文獻資料；另外除了使用各種搜尋的演算法外，文獻[16]將搜尋的演算法加入了封閉迷宮的小技巧，使用迴路的方式猜測死巷子，猜測平行邊的形式封閉迷宮，減少搜尋無意義的迷宮路徑，爭取競賽時的搜尋時間。

第二章 問題描述

電腦鼠的迷宮搜尋演算法包括下列三大問題：1.未知迷宮狀態下的起點至終點搜尋演算法、2.已知部分迷宮下如何有效率的搜尋未知迷宮的狀態、3.已知迷宮下找出起點至終點間的最佳路徑等三大問題，如下說明：

1. 未知迷宮狀態下的起點至終點搜尋演算法

   未知迷宮搜尋演算法的效率，會決定搜尋時探索路徑的長短，由某處設立為起始點，經此起始點進行迷宮的搜尋探索，了解迷宮狀態，並且找尋迷宮中所訂定的終點。

2. 已知部分迷宮下如何有效率的搜尋未知迷宮的狀態

   已知的迷宮搜尋又分為兩種迷宮問題演算法 1.運用已知的迷宮狀態，找尋從起點至終點的最佳路徑，2.試著嘗試再次搜尋迷宮狀態，找尋迷宮中極有可能的最佳路徑。

3. 已知迷宮下找出起點至終點間的最佳路徑

   迷宮起點至終點路徑的長度根據電腦鼠行走能力的不同可以有兩種不同的定義。對於一般只能走直線及轉 90 度角的電腦鼠而言，最佳路徑長度就相當於起點至終點中間必須通過的格子數(稱為曼哈頓長度)；但是對於能轉 45 度角走對角線路徑的電腦鼠而言，路徑長度則是歐幾里得路徑。兩者的差別如下圖 2.1 所示。

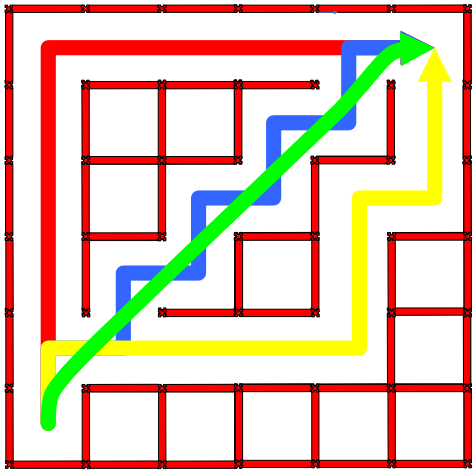


圖 2.1 曼哈頓與歐幾里得路徑示意圖

曼哈頓與歐幾里得距離：紅、藍與黃線分別表示所有曼哈頓距離都擁有一樣長度（10），綠線表示歐幾里得距離有 $5\times\sqrt{2}\approx 7.08$ 的長度，本文則使用曼哈頓路徑作為最佳路徑之求解。

為了能更清楚定義電腦鼠的迷宮搜尋問題，以下分別將電腦鼠競賽的規定、迷宮的實體、資料結構、圖形表示法加以說明。

2.1 迷宮實體

電腦鼠迷宮由16x16個迷宮格子組成，每個格子的邊長為18cmx18cm，若方塊有n面隔板則稱為n隔板格子(i.e. $n < 5$)。迷宮隔板的高度為5cm，厚度為1.2cm，隔板壁面的顏色為白色，頂部為紅色，壁面與頂部的塗料必須能反射紅外線的投射。迷宮的地面是以表面塗有黑色去光澤塗料的木材製成，塗料必須能吸收紅外線的投射。迷宮的「起點」位於迷宮角落，迷宮的「起點」應有三面隔板。迷宮的「終點」由迷宮中心的四個格子組成，這四個迷宮格子間不會有隔板，如圖2.2 所示。

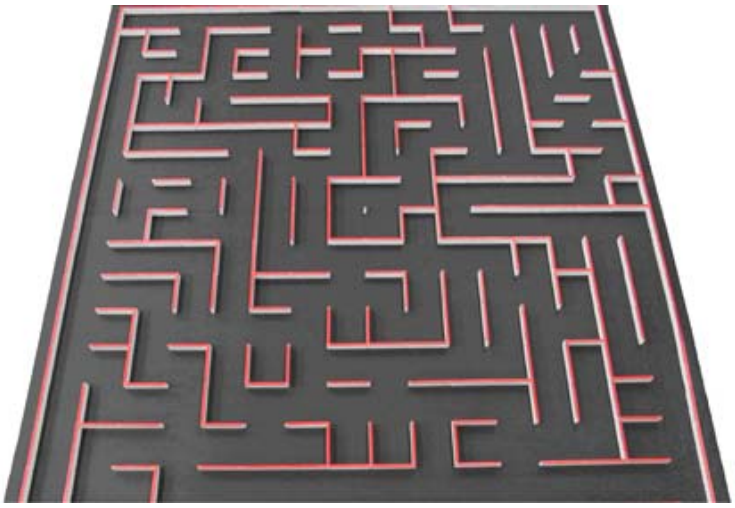


圖 2.2 16×16 迷宮實體圖

2.2 迷宮的資料結構

每一個迷宮格子有四個邊，每個邊是否存在隔板可以以一個位元表示，因此一個迷宮格子的隔板狀況可以以 4 個位元以 NWSE 的排列方式定義，其中 N 代表格子的北邊有隔板、W 代表格子的西邊有隔板、S 代表格子的南邊有隔板、E 代表格子的東邊有隔板。因此迷宮格子的隔板狀況可以用下圖 2.3 定義。

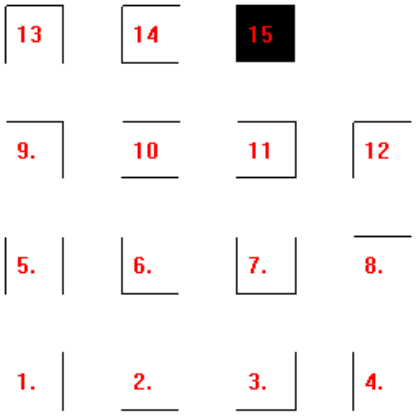


圖 2.3 迷宮格子的編碼

當電腦鼠搜尋迷宮時會使用紅外線感測器偵測格子的隔板狀態，剛開始時假設格子不存在隔板，當紅外線感測器偵測到某隔板時，必須將隔板的資訊加入包含該隔板的兩個相鄰的格子中，此時程式必須執行加入隔板的運算。注意，此運算應該用位元運算中的‘|’運算，不要用‘+’，否則會導致錯

誤。以下舉一例子說明：令 Cell[2][3]儲存第 2 列第 3 行的格子的狀態，如果在該格子的北方偵測到隔板，我們必須將此資訊加入 Cell[2][3]，則運算應寫成：

```
Cell[2][3] = Cell[2][3] | NORTH
```

要特別注意，當我們對某格子加入隔板時，必須同時對相鄰的格子加入相應的隔板，例如，針對 Cell[2][3]加入北方的隔板，我們必須在 Cell[3][3]加入南方的隔板，即

```
Cell[3][3] = Cell[3][3] | SOUTH
```

2.3 迷宮座標的定義

迷宮的狀態資料，本文使用了 8 位元的二維陣列來進行資料的存放，例如存放迷宮隔板的資訊則會宣告

```
Char Cell[16][16];                                    //存放迷宮隔板資訊
```

接著為了方便存取迷宮資料的二維陣列，本文宣告了一個 Location 資料結構，利用 Location 的資料結構成員含 char r, c，r 代表迷宮的橫軸位置、c 代表迷宮的縱軸位置如圖 2.4 所示

```
Typedef struct Position { char r, c;} Location;
```

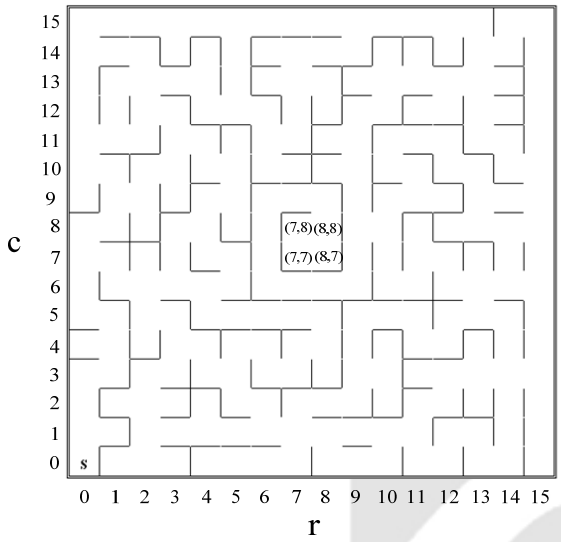


圖 2.4 迷宮座標表示圖

宣告 Location 資料結構後，往後的程式如果需要用到存放二維陣列的迷宮資訊時，只需宣告一個 Location 資料結構座標變數 p，接著若要將起點的迷宮隔板資訊 7 存放至 Cell 陣列時，如下演算法

```
Location p;
p.r = 0, p.c = 0;
Cell[p.r][p.c] = 7;
```

2.4 迷宮的圖形表示法

對於一個電腦鼠迷宮，我們可以使用一個圖形結構定義它。迷宮中起點、終點、及除了 2 隔板之外的格子都定義成節點；節點及節點之間以邊相連，邊的長度定義為該邊兩端點間的距離。為方便定義一個格子的邊長為 1 單位，則邊長就代表兩端點間，間隔的格子數。

圖 2.5 是一個 10×10 的迷宮，起點及死巷子的終點以黑色節點表示，分岔路口以藍色節點表示，迷宮終點則以紅色節點表示，邊以綠色線條表示，邊上的數字代表邊的長度。我們可以發現得到的是一個有許多迴路的平面圖。

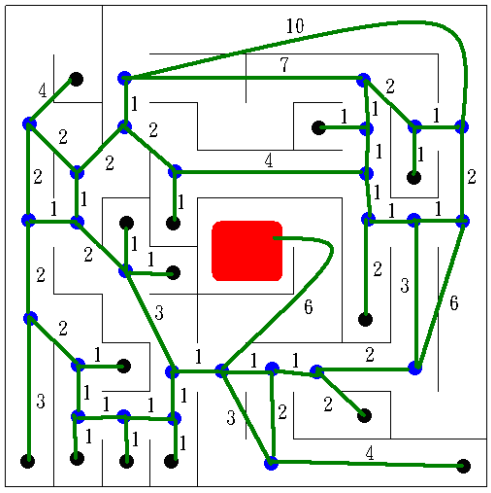


圖 2.5 一個迷宮範例及其對應的圖形

2.5 迷宮難度分析

在早期的迷宮難度較低，如圖 2.6 為 2001 年日本專業級預賽的地圖，只要

靠著簡單的法則即可快速的搜尋到終點，圖中範例使用右手法則來進行迷宮的搜尋；經過逐年的舉辦競賽，競賽的迷宮會特別經過設計，克制這些簡單的法則，逼著競賽隊伍創造新的方法，來改進迷宮搜尋的效率。

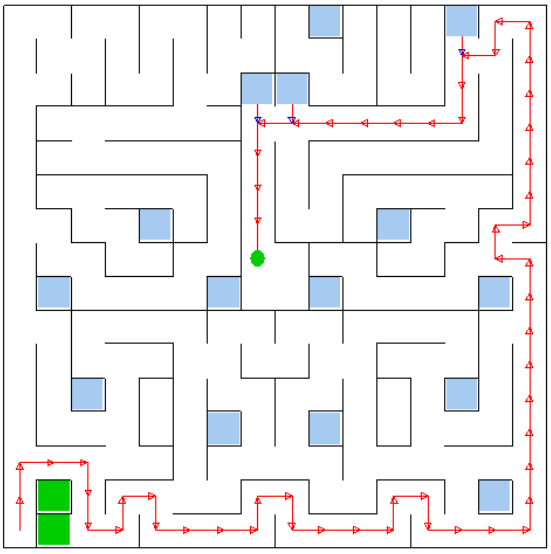


圖 2.6 2001 年日本專業級預賽迷宮地圖

圖 2.7 是日本 2010 年的專業級預賽的地圖，與圖 2.6 較早期的迷宮進行比較，可以發現難易度變的很困難，其中迷宮中出現了長直線的路徑、往復型迷宮、多死巷子的選擇、多重迴路的路徑、長銜接的 S 型迷宮、多彎道階梯迷宮等，因此將這些型態分類成各種圖形結構，這些圖形結構如圖 2.8 所示。

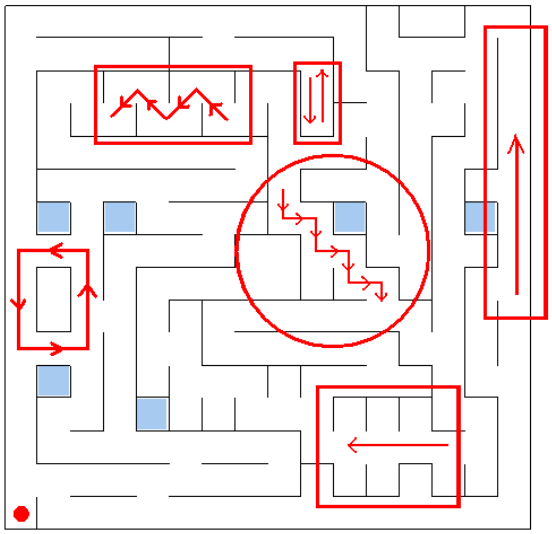


圖 2.7 2010 年日本專業級預賽迷宮地圖

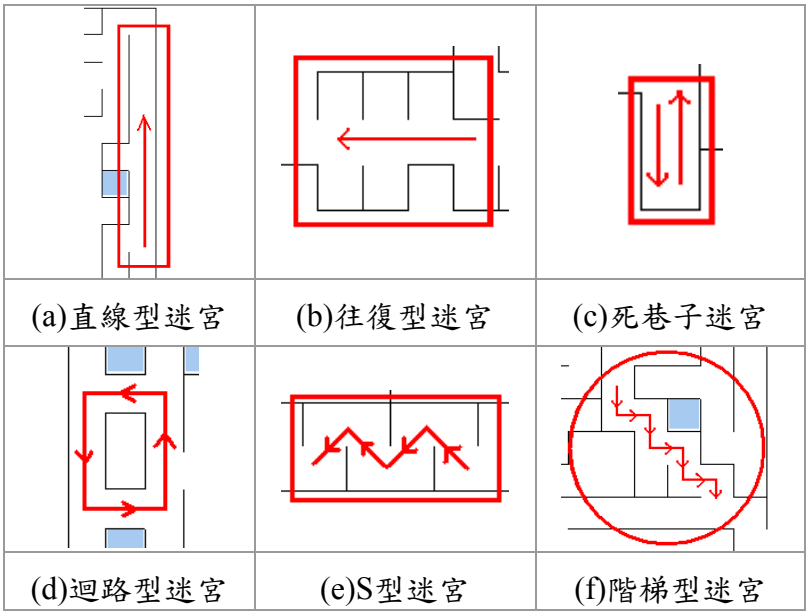


圖 2.8 典型的迷宮圖案

(a) 直線型迷宮

直線型迷宮如圖 2.8(a)所示，主要測試電腦鼠的直線速度，以及考驗電腦鼠行走長距離是否有失步的狀況發生，並且在這直線的路徑中，通常都會刻意設計由左右有隔板，變成只剩一面隔板，再轉變成兩面隔板的迷宮圖形，測試電腦鼠在高速行走時，是否會影響直行路徑。

(b) 往復型迷宮

往復型迷宮如圖 2.8(b)所示，俗稱超級停車場，主要測試電腦鼠的迴旋及位置校正的能力，也有另一因數是考驗迷宮演算法的思考方式，在此若往復型迷宮較長，常常會有很多參賽隊伍，會反覆的進入往復型迷宮。對於設計不良的電腦鼠容易在往復型迷宮中發生碰撞之情形。

(c) 死巷子迷宮

死巷子迷宮如圖 2.8(c)所示，為了增加迷宮探索的時間，以及考驗電腦鼠迴轉的運動能力，對於這種迷宮圖形，常常是造成比賽成績最大的差異性。

(d) 迴路迷宮

迴路迷宮如圖 2.8(d)所示，通常是用於考驗演算法的設計，不但演算法要有記憶行走過的路徑，且還要計算出下一步該往那個迷宮方塊繼續搜尋，在此通常都會有兩個以上的路徑要去抉擇。

(e) S型迷宮

S 型迷宮如圖 2.8(e)所示，在搜尋的時候，電腦鼠必須連續銜接 90 度轉彎的運動控制，這動作會造成電腦鼠無法得到完整的修正，因為轉彎的過程中，左輪馬達跟右輪馬達會有速度差，要一直進行加減速，當左輪及右輪進行加減速時，此時輪胎對於地面的摩擦係數就相對重要，在競賽的過程中，常常有隊伍因搜尋時間過久，輪胎沾滿了灰塵，輪胎對於地面的摩擦係數降低，在此迷宮圖形較易產生碰撞之情形。

但是在最佳路徑的移動時，目前一流的電腦鼠遇到 S 型迷宮，通常直接走實際路徑 V 字型，換言之，必須先移動為斜線角度，再連續交替的轉 V 字型。走 V 字型路線要修正車身的方位又更加困難，但是不行走 V 字型，又無法加快移動速度。所以 S 型迷宮等於測試程式的執行速度、紅外線的感測能力、及控制馬達的精準度。

(f) 階梯型迷宮

階梯形迷宮如圖 2.8(f)所示，搜尋的時候，就如同 S 型迷宮搜尋時候一樣，需要一直反覆的轉彎、反覆的控制馬達加減速，由於電腦鼠在進行轉彎的過程中，是無法使用任何裝置來進行位置的修正，因此轉彎的過程只單單依靠馬達速度差的控制，如果輪胎對於地面的摩擦係數降低時，階梯型迷宮圖形，常常也是造成電腦鼠碰撞的原因。

最佳路徑對於階梯型迷宮圖形，目前一流的電腦鼠通常在高速下，直接移動實際路徑行走斜線，所以階梯型迷宮也是測試程式的執行速度、紅外線的感測能力、及控制馬達的精準度。

第三章 迷宮演算法

本文將迷宮搜尋演算法分為下列三類說明：

3.1 未知迷宮狀態下的起點至終點搜尋演算法

未知迷宮的問題解法是電腦鼠搜尋迷宮最主要的核心。早期迷宮較為簡單，可以使用一些簡單的法則即可搜尋到迷宮終點，但目前的比賽中，迷宮通常都會被設計成無法用簡單的法則很快的找到終點，因此迷宮問題的解法目前陸續還都有人提出。本文探討沿壁法、深先搜尋法、洪水搜尋法、四象限法、四象限洪水搜尋法等。

3.1.1 沿壁法(Cell Following)

沿壁法是一個古老而簡單的演算法，電腦鼠在迷宮入口時任選其中一面隔板，然後沿著這一面隔板前進直到到達終點為止。如圖 3.1 所示，電腦鼠沿著隔板搜尋到死巷子時，則會往回走，且將回走的路徑關閉，反覆的運用此方法搜尋至終點；使用沿壁法搜尋較雜的迷宮時，通常都會將整個迷宮地圖搜尋近乎完整，才能找到終點。

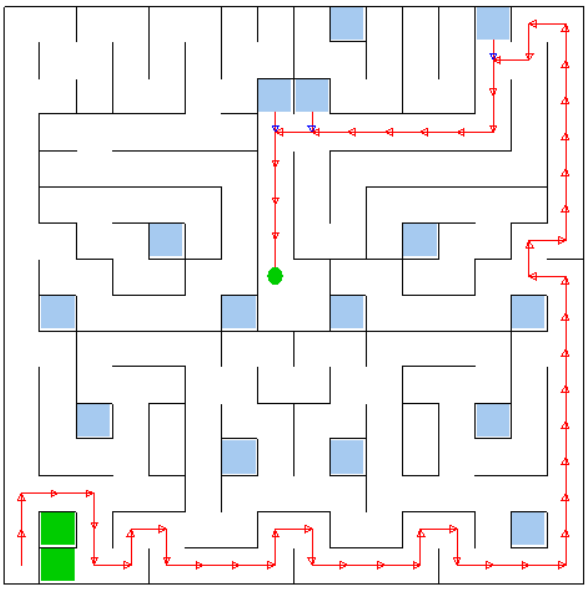


圖 3.1 2001 年日本專業級預賽地圖使用沿壁法搜尋迷宮

3.1.2 深先搜尋法

電腦鼠搜尋迷宮的問題中，由於電腦鼠事先並不知道迷宮的狀態，只能根據已經走過的路徑及目前格子週遭的環境，試著尋找到達終點的路徑。因此深度優先搜索法是比較直接而且容易實現的方法。

深先搜尋演算法中，當電腦鼠到達一個岔路節點後，就訪問其中一個未被搜尋的節點，直到死巷子節點或遇到已訪問過的岔路節點為止。接著，再回溯到前一個岔路節點，繼續依深度優先的原則追蹤其他相鄰節點。在搜尋的過程中，當遇到岔路節點時，依行走方向的選擇不同，可分成下列方法：

1. 右手法則：遇到岔路節點時優先右轉，其次是直走、左轉如圖 3.2
2. 左手法則：遇到岔路節點時優先左轉，其次是直走、右轉如圖 3.3
3. 中左法則：遇到岔路節點時優先走直線，其次是左轉、右轉
4. 中右法則：遇到岔路節點時優先走直線，其次是右轉、左轉
5. 亂數法則：沒有固定優先權，依亂數決定轉向。

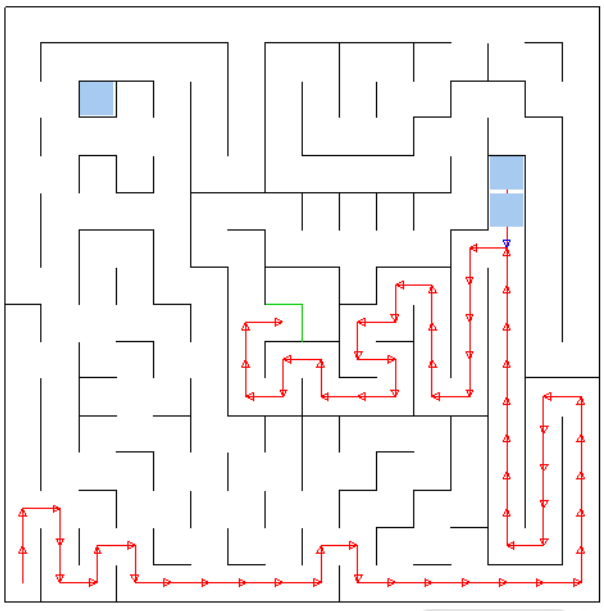


圖 3.2 右手法則

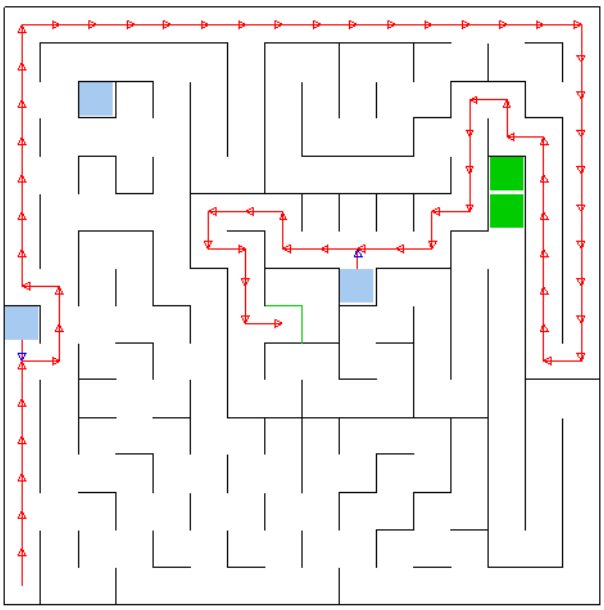


圖 3.3 左手法則

上述的方法都會搜尋圖形中的每一條路徑，每一個連通的節點，直到到達終點為止。因此，它們的優點是只要到終點的路徑存在，就一定能夠找到答案。

6. 向心法則：

向心深先搜尋法是一種改良的深先搜索法。此方法在遇到岔路時，會辨別目前的位置，相對於中心位置的方位，選擇向迷宮中心的邊優先進入。這種方式就如同觀光客在台北旅遊時，如果他要到 101 大樓，當他遇到岔路時，它就會抬頭看 101 大樓的方位，再選擇朝向 101 大樓的街道前進，這種方法通常比在岔路時隨機選擇一條路前進有效的多。由於向心搜尋演算法善用終點就在迷宮中心處的知識，因此搜尋路徑長度通常比較短。向心搜尋演算法通常先建立如圖 3.4 的權重圖所示，遇到岔路節點時優先往權重較小的格子前進。

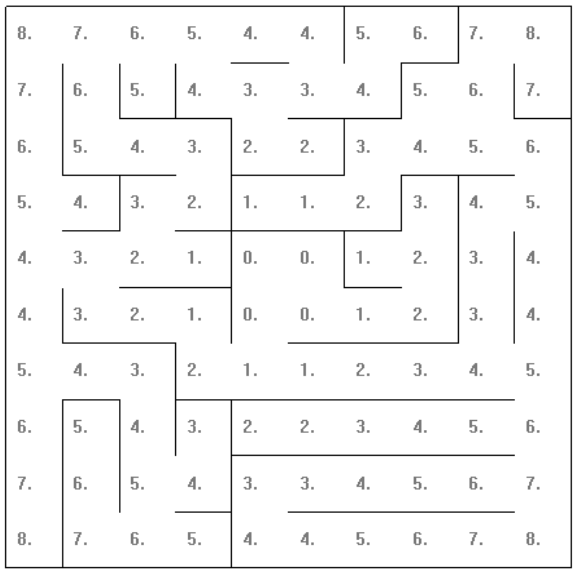


圖 3.4 10×10 向心權重表

圖 3.5 是一個向心搜尋法的範例，紅色的路徑是電腦鼠經過的路徑。採用向心搜尋演算法時，電腦鼠必須記憶走過的節點及路徑，避免重複走過已走過的路；圖 3.6 淺藍色的路徑代表死巷子，電腦鼠應該要有能力記憶這些死巷子，來加以簡化迷宮。簡化迷宮作法是當電腦鼠進入迷宮格子編碼為 7、11、13、14(即死巷的終點)時，它必須回頭離開這個格子，並且在離開時將一個虛擬的隔板加入，且離開格子，使之成為被封閉的格子，並持續封閉此邊上經過的格子，直到回到上一節點為止。利用這種方式，就可以正確的標識死巷子。依序這樣的封閉，電腦鼠就可以找到終點。

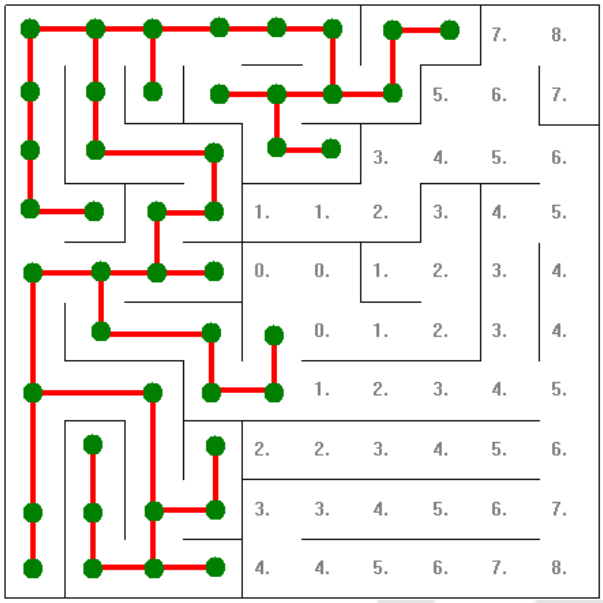


圖 3.5 向心深先搜尋法範例

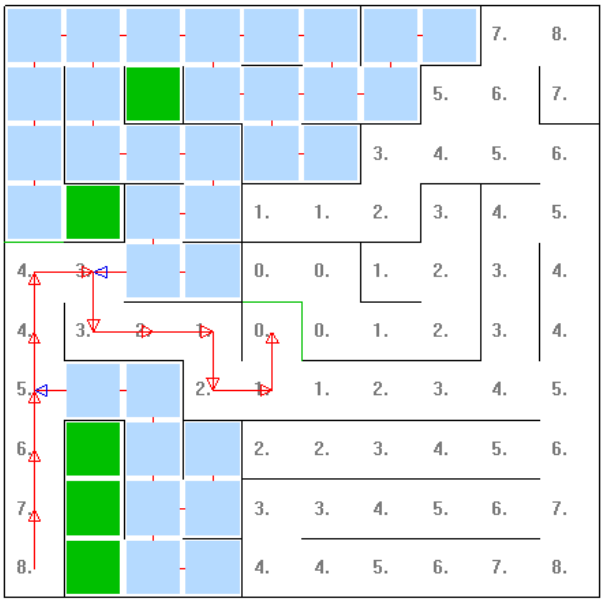


圖 3.6 向心深先搜尋法封閉死巷子

3.1.3 四象限法

四象限法為本文獨創的方法，本方法首先將迷宮分為四個象限，如圖 3.7 所示(使用 10x10 迷宮做為說明)。若電腦鼠位置 p 位於第一象限，則令終點位置為（4,4）；若電腦鼠位置 p 位於第二象限，則令終點位置為（4,5）；若電腦鼠位置 p 位於第三象限，則令終點位置為（5,5）；若電腦鼠位置 p 位於第四象限，則令終點位置為（5,4）。

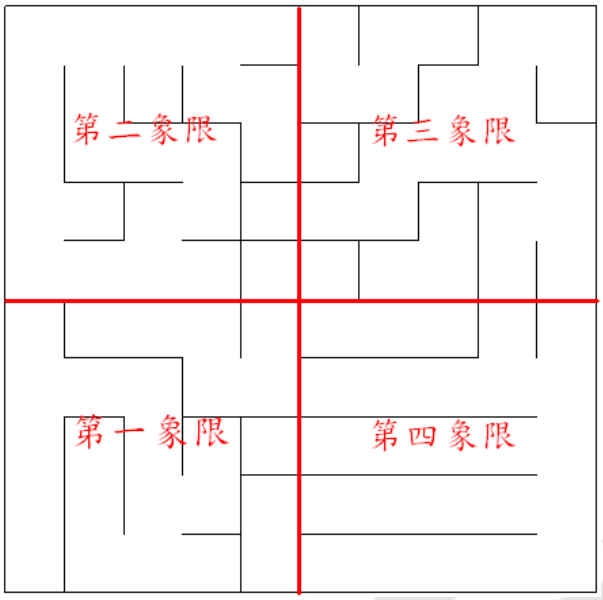


圖 3.7 四象限迷宮

接著將四個象限的格點分別填入權重，利用此權重來猜測目前位置離終點多遠，如圖 3.8 所示。

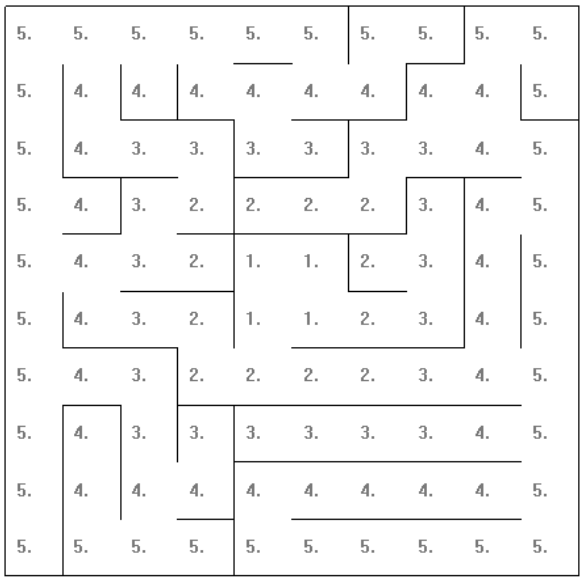
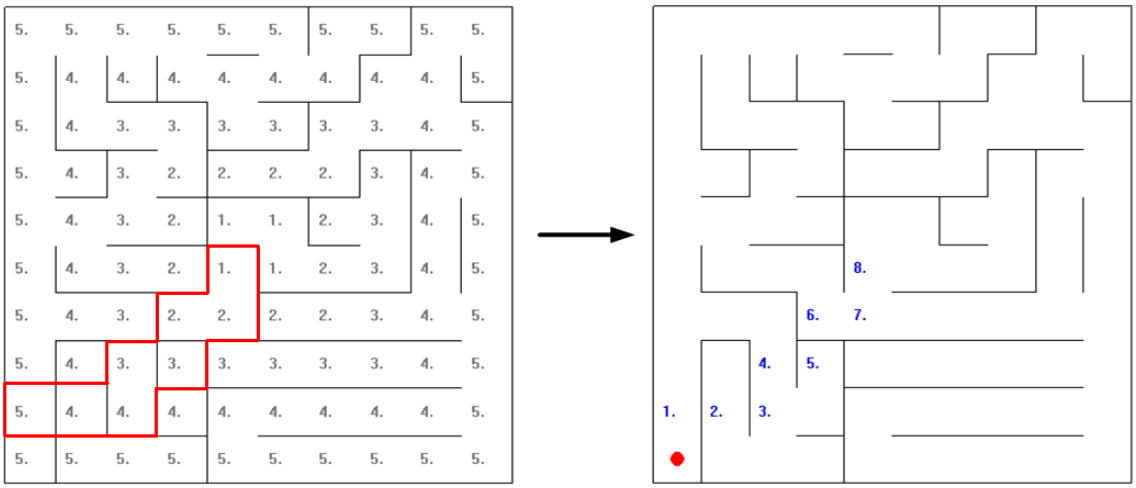


圖 3.8 四象限權重表

四象限的演算法利用已定的迷宮權重，計算電腦鼠要移入的迷宮位置是否離終點較近，以及計算該移入的迷宮位置是否可達終點。圖 3.9 是使用四象限的演算法猜測終點的示意圖，左邊的圖示編列迷宮的權重，右邊的圖則為四象限的演算法猜測後的結果，不需推算整個迷宮的權值，可減少程式運算時間。



(a) 四象限編列迷宮的權重　　(b) 四象限的演算法猜測後的結果

圖 3.9 使用階梯式權重路徑猜測終點

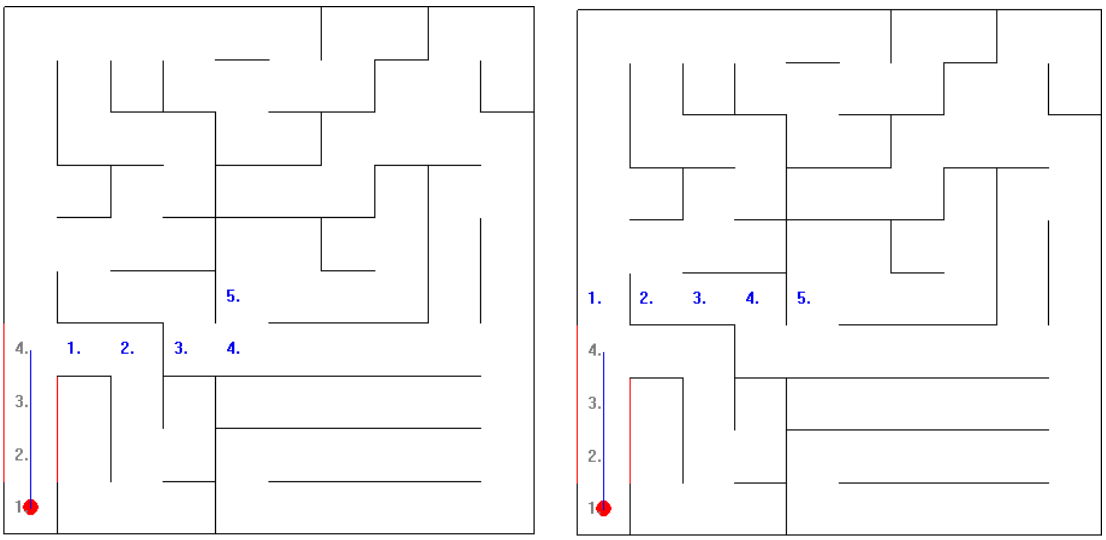
圖 3.9(b)的紅色圓點位置為電腦鼠目前位置，經由迷宮的牆面偵測後，可以知道前方有路，因此將紅色圓點的前方位置設為要移入的位置（0,1）數字 1，接著利用終點猜測演算法如下：

```
p =  預計行走位置;
int a[4], ans;
for (i = 0; i < 10; i++)
     for (j = 0; j < 10; j++)
              Maze_number[i][j] = (迷宮走過) ? 255 : 0;     // Maze_number 初始化

Maze_number[p.r][p.c] = Branch_Weight = 1;
while(p 不是終點)
{
      if (Maze_number[p.r][p.c+1] > 0)       a[0] = 255;
      else            a[0] = Weight[p.r][p.c+1];
      if (Maze_number[p.r-1][p.c] > 0)        a[1] = 255;
      else            a[1] = Weight[p.r-1][p.c];
      if (Maze_number[p.r][p.c-1] > 0)        a[2] = 255;
      else            a[2] = Weight[p.r][p.c-1];
      if (Maze_number[p.r+1][p.c] > 0)        a[3] = 255;
      else            a[3] = Weight[p.r+1][p.c];

   if (有未搜尋的相鄰格子)
  {
      ans = 0;
      for (i = 1; i < 4; i++)
          if(a[i] < a[ans])   ans = i;     //判斷較小權重位置
      i = 0x01 << ans;                     //決定行進方向
      p =  下一個位置;
      Branch_Weight++;
      Maze_number[p.r][p.c] = Branch_Weight;
  }
  else  電腦鼠回頭;
}
```

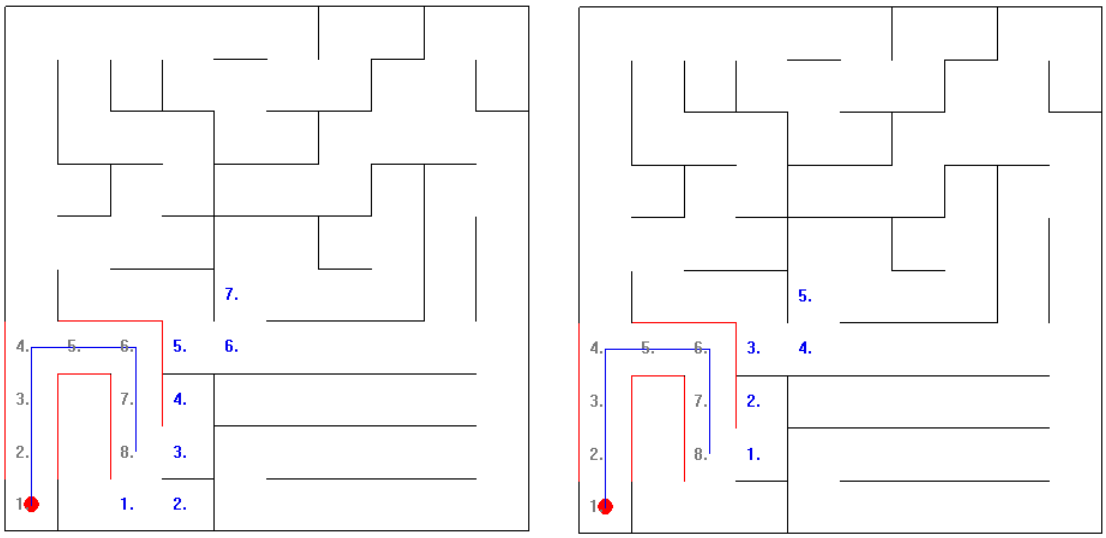
如圖 3.10 所示，當電腦鼠位於（0,3）位置時，發現（1,3）、（0,4）兩個相鄰位置可移入，此時需要比較哪一個位置比較接近終點，電腦鼠則選擇離終點較近的位置移入，但在本例中，相鄰的兩格點與終點的距離一樣，為了說明更多狀況，在此選擇轉彎優先來決定，選擇移入（1,3）的位置。



(a) （1,3）的位置可移入　　(b) （0,4）的位置可移入

圖 3.10 四象限終點遇到兩條權重相同之狀況

移入（1,3）的位置後繼續搜尋至（2,1）的位置，如圖 3.11 所示，(a)圖為移入（2,0）的位置權重為 7，(b)為移入（3,1）的位置權重為 5，在此權重不相同，因此選擇（3,1）的位置，較低的權重移入繼續搜尋。



(a)移入（2,0）的位置權重為 7　　(b)移入（3,1）的位置權重為 5

圖 3.11 搜尋至（2,1）的位置遇到兩條以上可移入之狀況

電腦鼠持續的搜尋至（3,2）的位置，如圖 3.12 的死巷子問題，此時會利用每搜尋一個迷宮位置，編列遞增數字，由數字大往數字小遞減回頭至上一個交叉節點的位置（2,1）繼續執行四象限的演算法。

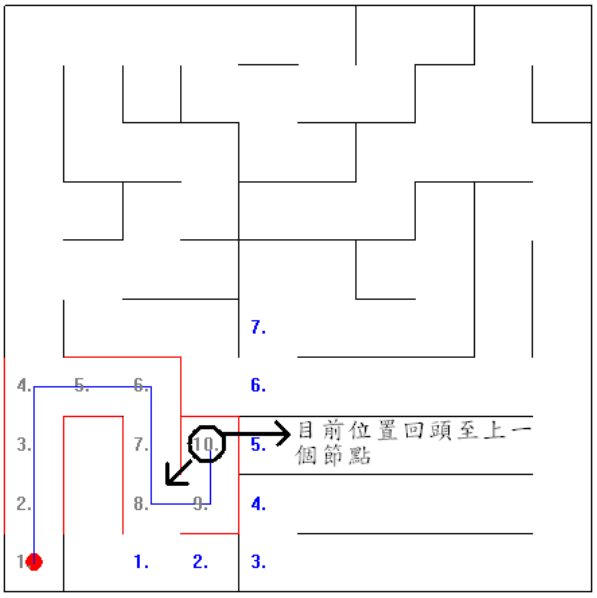


圖 3.12 四象限搜尋法遇到死巷子依照編列的數字回頭

接著移入（2,0）的位置繼續搜尋，此時要注意，每搜尋一個迷宮位置的數字編列要多遞增 1，用意是為了下次回頭時使用，將概念如圖 3.13 說明，下次要回頭時只需依照編列的數字大到小移動，即可回到上一個交叉節點的位置。

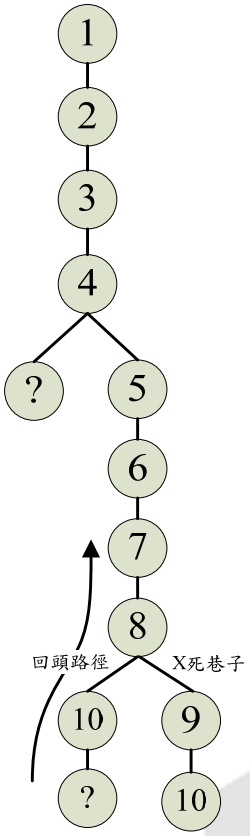


圖 3.13 搜尋時的樹狀結構

繼續搜尋至（2,0）的位置，此時將搜尋迷宮位置的數字編列 10，如圖 3.14 所示，接著會繼續執行四象限演算法，選擇權重較小的位置移入，但是如果權重路徑無法到達終點時，四象限演算法則會放棄此路徑，反覆前述步驟直到電腦鼠到達終點。

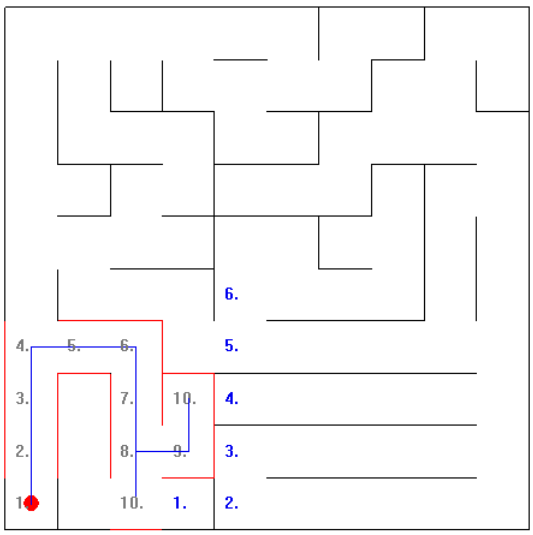


圖 3.14 目前在（2,0）位置的權重

當搜尋位置回到（0,4）後，下一步要移入（0,5）的位置，此時（0,5）屬於第二象限，因此終點位置更改為（4,5），如圖 3.15 的權重路徑所示。

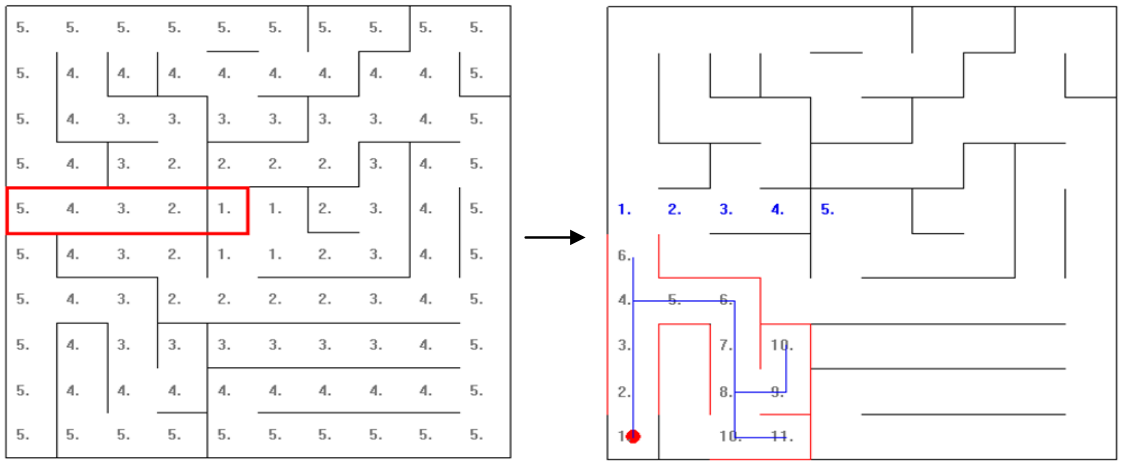


圖 3.15 移入迷宮位置為第二象限

圖 3.16 為四象限演算法搜尋的完成圖，搜尋的過程中總共移動了 70 步，圖中編列的數字為每搜尋一個迷宮位置的紀錄值，到達終點後，只需依照一路搜尋過

來的紀錄值，由數字大往數字小的路徑移動即可回到起點，不需使用多餘的演算法找出已知迷宮的最佳路徑，節省很多運算的時間。

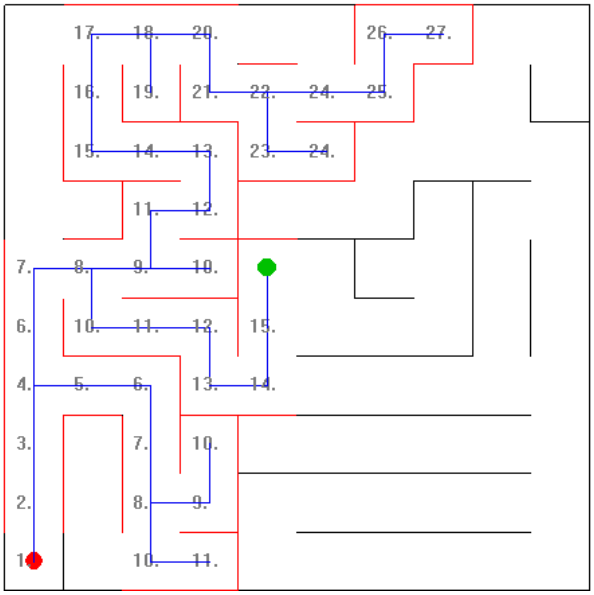


圖 3.16 四象限演算法完成圖

3.1.4 洪水搜尋法

洪水搜尋法如圖 3.17 所示，由於使用原始迷宮 16x16 大小來做示範，迷宮標誌起來較複雜，因此本章節使用 10x10 迷宮作為示範。設迷宮的終點為（4,4）的位置，編碼為 0，其餘另外三個終點位置則強制設立為封閉節點，以「*」作為標誌，與終點相鄰的節點權重為 1，與權重為 1 的節點相鄰之節點權重為 2，並以此類推。本演算法只從終點編列至電腦鼠目前位置，其餘的區域則標示為「*」。

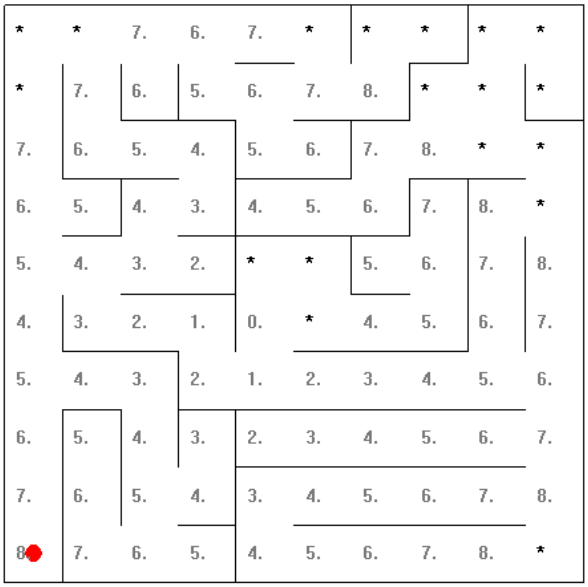


圖 3.17 由終點編列至起點的洪水搜尋法

接著電腦鼠會由權重大的位置走到權重小的位置，由移動的路徑中，若遇到兩條相同的權重值，必須選擇要轉彎優先或者是直走優先，由於選擇轉彎比較浪費時間，因此本文採用直走優先策略。

圖 3.17 電腦鼠依序的由 8→7→6→5 前進，在位置（0,3）遇到相同權重的下一格子，選擇直走至（0,4）迷宮位置，發現下一步要移入的格子之權重比目前大時，則會重新執行洪水搜尋法到目前的格子為止，得到如圖 3.18 所示的結果。圖中可知，往前及往後格子的權重都相同，則會優先選擇未探索過（0,5）的格子繼續搜尋，到達（0,5）位置後，再次依序由較大權重往較小權重移動，迷宮位置（0,5）→（1,5）→（2,5）…，反覆前述步驟直到電腦鼠到達終點。

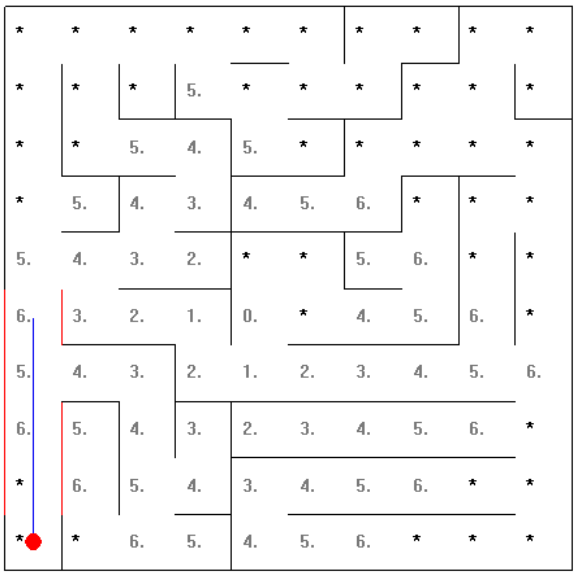


圖 3.18 遇到移入的迷宮權重較大時，重新執行洪水搜尋法

3.1.5 四象限洪水搜尋法

此章節要提出的方法是，結合四象限演算法及洪水搜尋法，將四象限的搜尋概念，套用於洪水搜尋法，就如圖 3.19 的規劃：

- 第一象限→洪水起點為（4,4）
- 第二象限→洪水起點為（4,5）
- 第三象限→洪水起點為（5,5）
- 第四象限→洪水起點為（5,4）

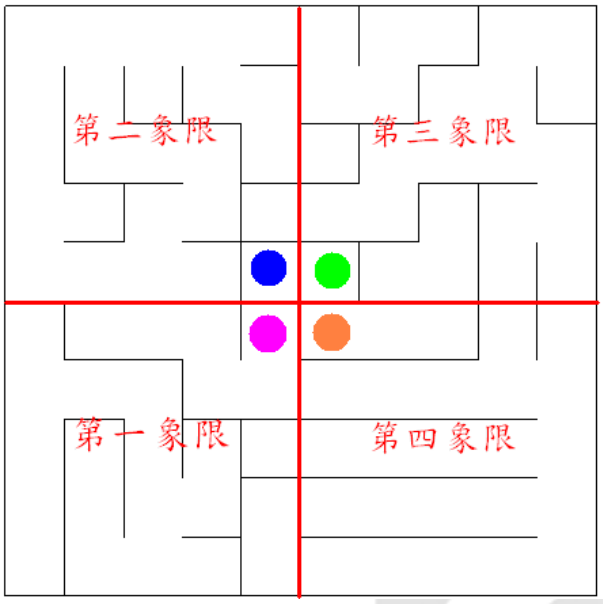


圖 3.19 四象限洪水搜尋法的終點定義圖

因此由此概念，電腦鼠由左下起點位置開始移動時，洪水則由第一象限的(4,4)開始編列，如圖 3.20 所示，當洪水由終點編列至電腦鼠的位置後，則會停止洪水的編列，其餘的部分還是如 3.1.3 章節一樣；猜測電腦鼠目前位置離終點有多遠，猜測需要行走多少路徑，因此這動作我們叫作建立權重，離終點權重的多寡，是決定下一步移動的路徑。

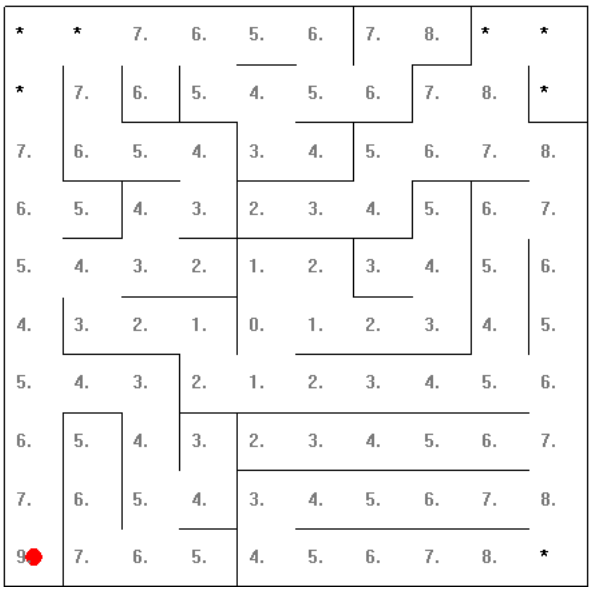


圖 3.20 四象限洪水編列

知道了權重的編列後，則電腦鼠由權重大往權重小的概念，持續的進行移動搜尋，直到遇到相鄰的位置都為相同的權重值，則會重新進行洪水的編列，這些洪水搜尋之規則都與 3.1.3 章節相同，只有遇到如圖 3.21 中，當電腦鼠移動至第二象限時，此時終點的位置就換置於（4,5），因此洪水則由終點位置（4,5），開始進行重新編碼的動作。

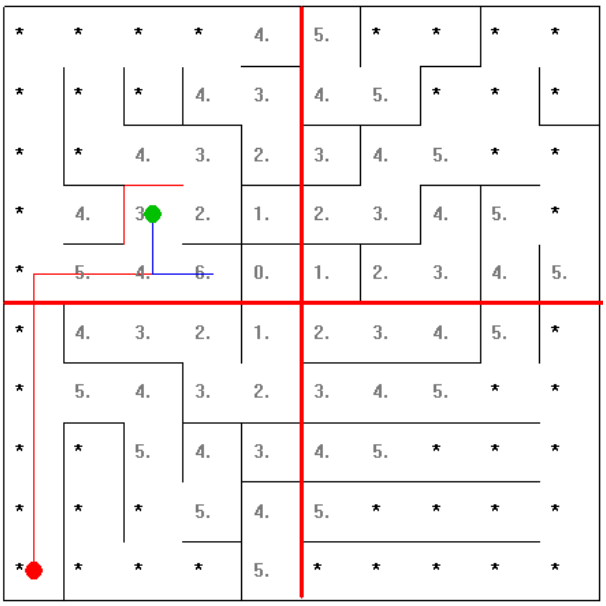


圖 3.21 第二象限之編碼

圖 3.22 則為電腦鼠的位置，至第三象限時，洪水的起點設定至（5,5）的位置，並由此座標進行洪水的編碼，因此將此動作說明如下：

```
目前座標 p
 if ( p.r < 5) p.r = 4; else p.r = 5;
 if ( p.c < 5) p.c = 4;else p.c = 5;
```

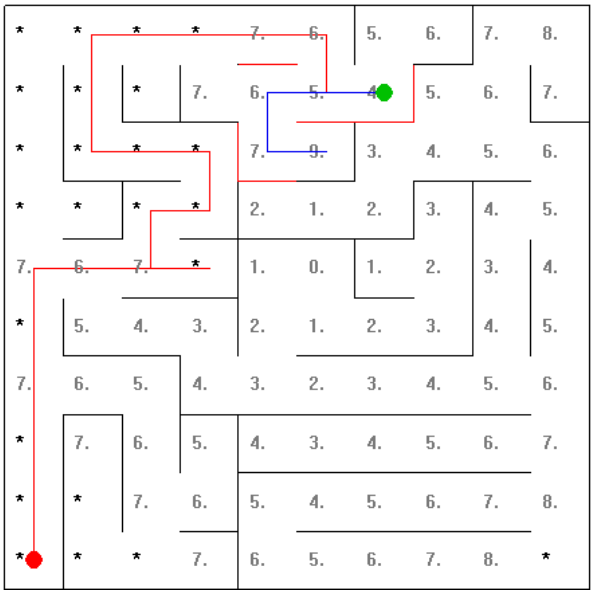


圖 3.22 第三象限之編碼

因此使用了四象限洪水搜尋圖 3.23 之迷宮，共搜尋行走了 63 步到達終點，且由起點至終點的距離為 12 步，改良了此方法，減少了很多搜尋的路徑。

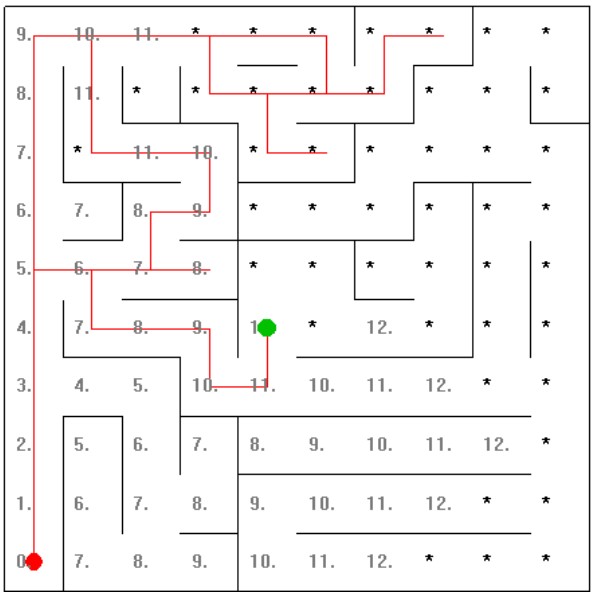


圖 3.23 四象限洪水搜尋至終點

3.2 已知部分迷宮下由終點回到起點的搜尋演算法

電腦鼠要找出迷宮起點到終點的最短路徑，有賴於徹底搜尋過迷宮的每一個格點，因此如何有效率的讓電腦鼠由迷宮終點回到起點，且有效的搜尋未知的格點是很重要的。本文發展的回程搜尋演算法如下說明。

3.2.1 洪水回程搜尋法

續 3.1.4.3 章節，電腦鼠搜尋到終點後，如圖 3.23，會再次執行洪水演算法，且電腦鼠由數字大的格點向數字小的格點移動，因此電腦鼠依序由（4,4）→（4,3）→（5,3）位置移動，發現相鄰的格點中有數字更小且尚未探索的格點（5,3），電腦鼠因此會前往（5,3）位置並持續搜尋至（8,3）的位置，如圖 3.24 所示，遇到兩條可移動的路徑時，重新執行洪水回程演算法，電腦鼠再次的由權重大移動至權重較小的位置，直到遇到無權重較小時，則重新執行洪水回程演算法，以此類推。

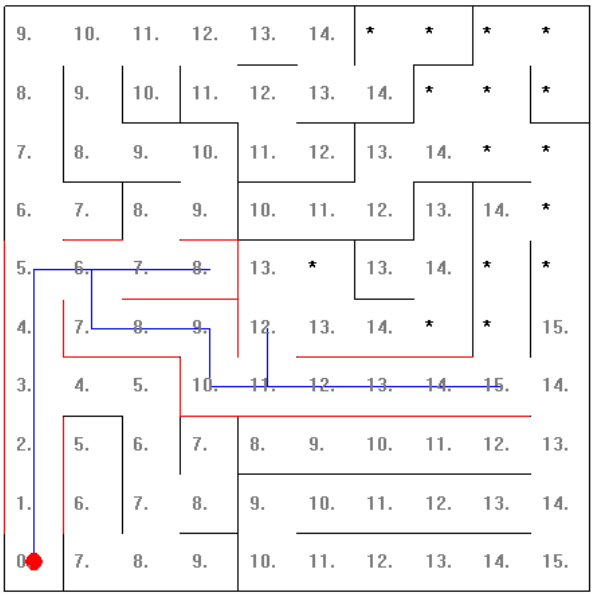


圖 3.24 回程洪水演算法至兩條路徑可行走之處

圖 3.25 顯示由迷宮位置(8,3)由權重大行走至權重小的格點，到達位置(4,2)，發現死巷子後重新執行洪水回程演算法之權重編列，由圖中可以發現回程洪水演算法一直試著尋找回到起點的路徑，揣測是否有路徑可以由(1,3)位置回到起點。

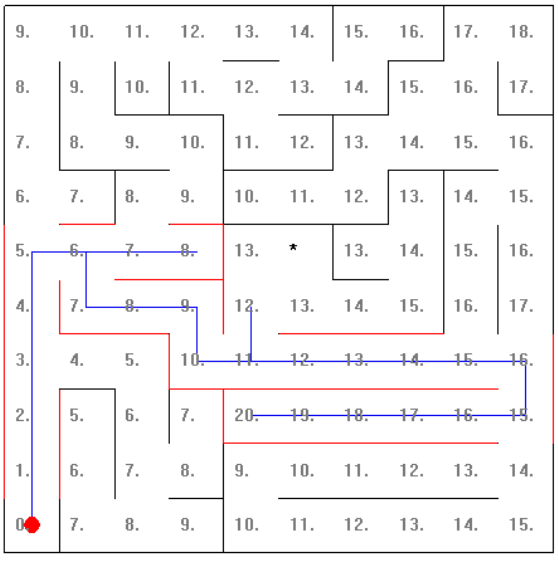


圖 3.25 回程搜尋路徑至死巷子

當移動到位置（4,1）時，發現有路的權重較大，重新執行洪水回程演算法的編列如圖 3.26，且移動至位置（4,0），此時無較小權重可移動，又重新執行洪水回程演算法如圖 3.27；反覆的執行洪水回程演算法，直到回到起點的位置。

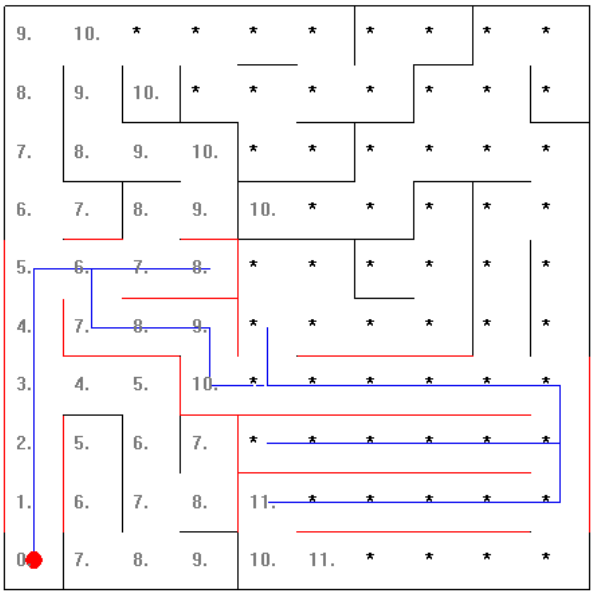


圖 3.26 回程搜尋路徑一

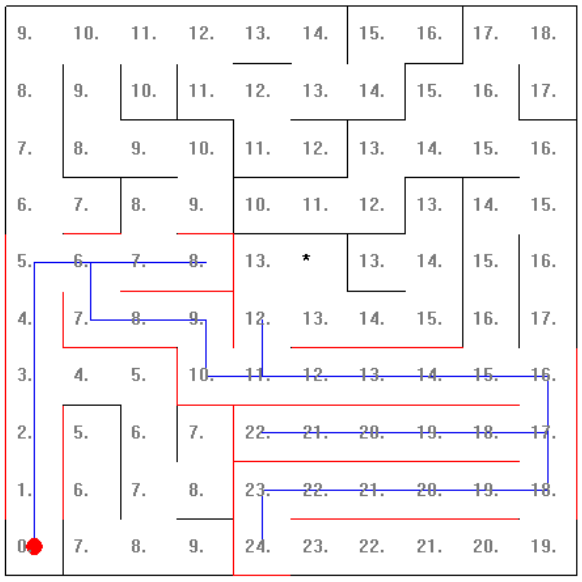


圖 3.27 回程搜尋路徑二

3.2.2 交錯式回程搜尋法

此方法也是套用洪水的概念去搜尋，藉著已知的迷宮狀態，當電腦鼠到達終點要返回起點時，會由起點的迷宮位置開始編列洪水至終點的位置；接著電腦鼠藉由編列完成的洪水編碼，依序由數字大走至數字小的位置，移動至迷宮位置（8,3），此時相鄰的位置有尚未搜尋過，交錯式洪水回程搜尋法會由迷宮終點當

洪水的起點，電腦鼠目前的位置作為洪水終點，重新執行洪水演算法，如圖 3.28 所示。

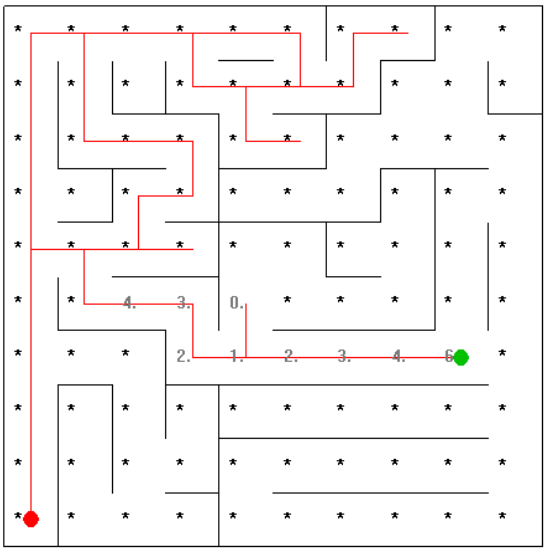


圖 3.28 迷宮終點作為洪水起點的回程搜尋

洪水演算法編列完成後，會由編列的起始點位置開始依序的遞增判斷，是否有編列的位置尚未搜尋過，圖中並無此問題。

接著再換由迷宮起點作為洪水演算法的起點，電腦鼠目前位置作為洪水演算法的終點，開始執行洪水演算法，如圖 3.29 所示，執行洪水演算法後，依序由數字大往數字小的編列值搜尋，反覆的執行這些動作，達到交錯式洪水回程搜尋法之特點。

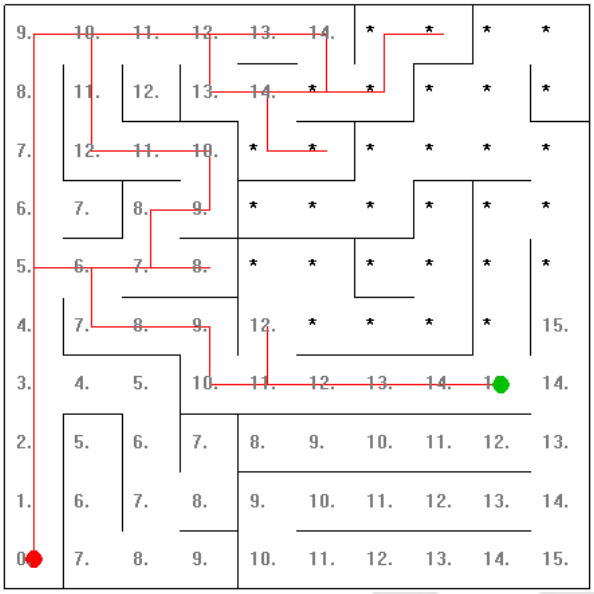


圖 3.29 迷宮起點作為洪水起點的回程搜尋

3.2.3 已知部分迷宮下由終點回到起點交錯式洪水回程搜尋法較佳原因

此回程搜尋法，多了終點至電腦鼠目前位置的洪水編碼，但是由於 10x10 的迷宮看不出效果，本文在此使用 16x16 迷宮如圖 3.30 的已知迷宮編碼之狀況，綠色的位置是電腦鼠回程搜尋時的目前位置，迷宮中的數字則是由迷宮終點當洪水起始點，開始編列的洪水數值，接著圖中可以發現，圈起來的位置，尚未探索過，且可能會使得最佳路徑變短，因此這圈起來的位置，很值得電腦鼠再次的移動過去探索迷宮狀態，因此使用交錯式迷宮搜尋演算法，在此則發揮了很大的功效，電腦鼠在此狀況下會移動至圈起來的迷宮位置，進行迷宮回程搜尋最佳路徑的探索，電腦鼠只需依靠洪水編碼的數值，依序的由數值大移動到數值小，到達未探索之迷宮位置；勝過於 3.2.1 章節的洪水回程搜尋法，如圖 3.31 所示，只搜尋離迷宮起點較近的路徑。

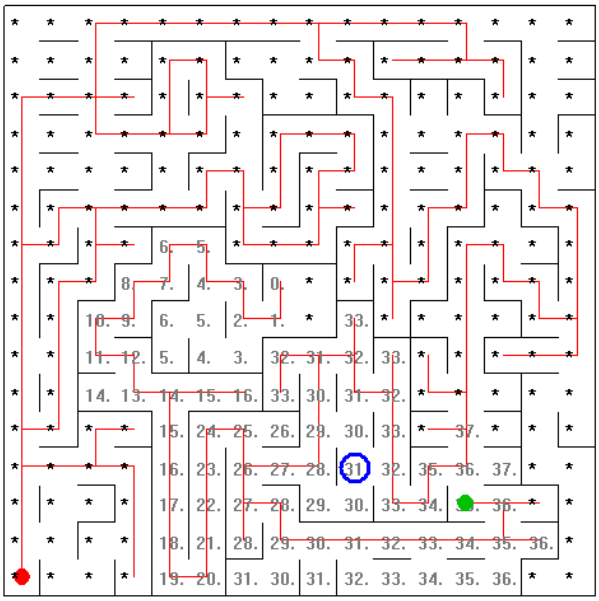


圖 3.30 交錯式回程搜尋之優點

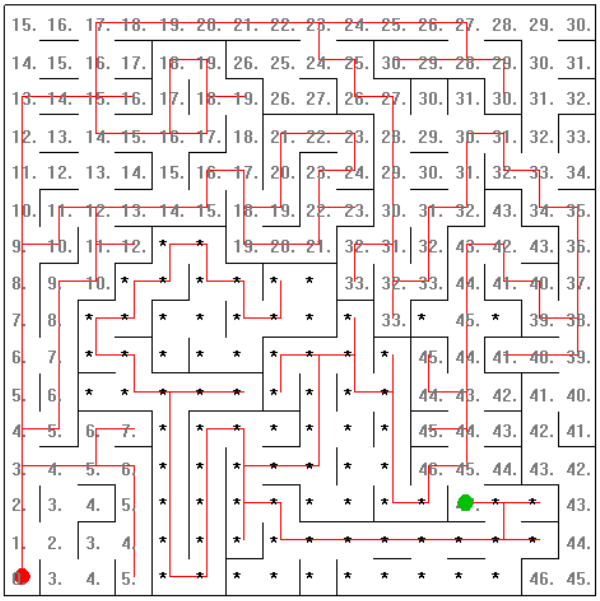


圖 3.31 由迷宮起點作為洪水起點的回程搜尋法

3.3 已知迷宮下起點至終點之最佳路徑

洪水迷宮搜尋法，意味著就有如水流般的移動，如圖 3.32 已知迷宮地圖所示，迷宮中淺色的方塊代表未探索過或者死巷子的路徑，這些死巷子的路徑為不可行進的路徑；起初洪水由終點綠色的位置注入，令終點綠色的迷宮編碼初始值為 0，此時洪水會往有路的地方去移動，且每當移動迷宮方格一格時，則會將新的迷宮方格編列累加 1 的迷宮編碼。

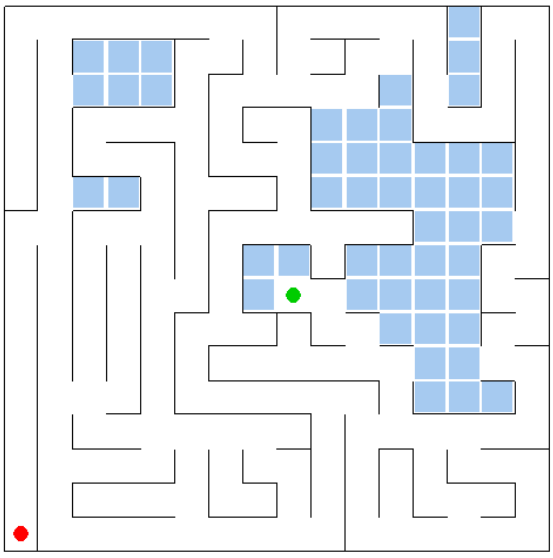


圖 3.32 迷宮地圖

已知洪水演算法依序的編列號碼如圖 3.33 所示，洪水演算法由綠色終點的位置，依序的將迷宮位置（8,7）= 0 、（9,7）= 1、（9,6）= 2、（10,6）= 3、（10,5）= 4。

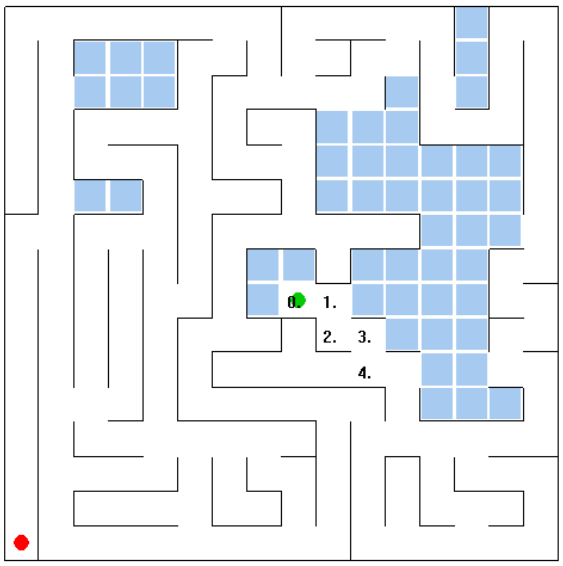


圖 3.33 已知洪水演算法編列至號碼 4

直到遇到兩條以上可移動的路徑時，此時兩條路徑要同等編號編列，如圖 3.34 所示，左右各平均的遞增編列號碼，迷宮（10,5）的左右位置各編列 5，依序延續此動作；當遇到死巷子時，或者新的迷宮位置已編列，則會停止此路徑的洪水編列。

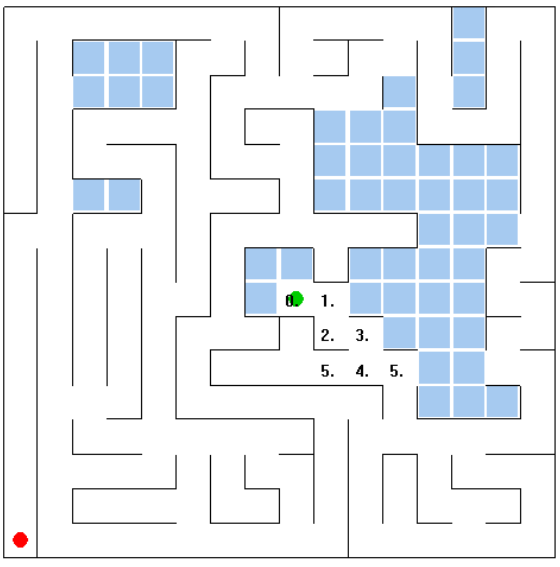


圖 3.34 遇到兩條路徑的洪水編列方式

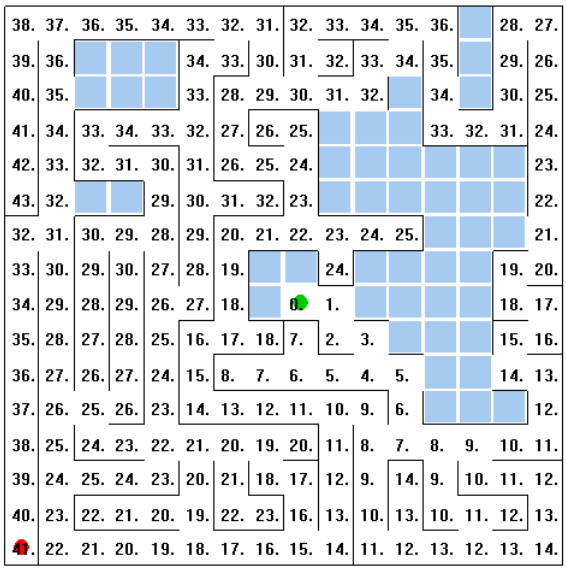


圖 3.35 已知迷宮洪水編碼完成圖

圖 3.35 為洪水編碼完成的示意圖，可以發現其實由迷宮起點位置，到迷宮中心的終點位置，只要利用編碼的數字，由數字大往數字小的位置去移動，就可以直接到達終點；但是依序由數字大到小的方式移動到終點，移動的行徑中常常會遇到多條相同權重的路徑可選擇，這時就意味著，有兩條以上的路徑可以到達終點，因此在相同權重的地方，進行路徑的分析，分析那條路徑的直線較多、那條路徑的轉彎較多，則取其一；接著我們將洪水編碼概念說明如下：

```
int code ;
Location p , q;                                          // p=電腦鼠目前座標
int flood_munber[p.r][ p.c] = 0;
AddQ(p);                                                 //堆疊 p 點座標於佇列
while (佇列不是空的)
{
    p = DeleteQ( ); code =目前座標編碼;
    for (unsigned dir = EAST; dir <= NORTH; dir <<= 1) //分別檢查四周的方向
    {
        q =   相鄰 p 的 dir 座標;
        if(目前座標的 dir 方向有路  && q 沒有編碼過)
        {
            flood_munber [q.r][q.c] = code + 1;
            AddQ(q);
        }
    }
}

void addQ(Location p)
```

```
{
   堆疊 p 點座標於佇列;
}
Location DeleteQ( )
{
   return  後進先出從佇列中取出座標;
}
```

第四章 結果與討論

本文採用 2001 年至 2010 年全日本電腦鼠比賽的迷宮比較本文提出的各種演算法的效果，迷宮的詳細狀態請參考附錄。迷宮的名稱由年份加上英文字，英文字的意義分別代表：

E　：初級競賽

EP ：專業級預賽

F　：專業級決賽

全日本電腦鼠比賽始於 1980，每年大約於 11 月舉辦之競賽，比賽分 2 天進行，包含初賽及複賽，迄今本項比賽已連續舉辦了 31 屆。本項比賽除歡迎各國隊伍報名參賽外，並主動邀請各國的冠軍隊伍參賽，因此本項競賽等同於世界賽，是目前全世界水準及難度均最高的電腦鼠競賽。初級競賽是日本國內的中學學生競賽的迷宮場地，迷宮的困難度較低；專業級競賽是提供專業人士參加的競賽，分預賽及決賽，預賽基本上是資格賽，迷宮難度較高，只要能在規定時間內完成比賽都會頒發一份證書，並進入決賽；決賽的迷宮難度又比預賽的高。

4.1 初級競賽

迷宮可以使用圖形定義，而圖形的深先搜尋法的時間複雜度與邊的個數成正比，因此邊數的多寡不失為判斷迷宮複雜度的指標。由於迷宮實際的行走難度與迷宮的圖案有關，因此迷宮的最短路徑長度及 90^{o} 轉彎數可視為困難度指標。

表 1 為 2003 年至 2010 年日本初級競賽迷宮的複雜度及困難度統計。我們可以發現至 2008 年以後，迷宮的複雜度似乎降低了，推測是為了讓更多中學生能夠完成比賽，以提高初學者的參賽興趣，因此降低了難度。

表 1 日本初級競賽迷宮的複雜度及困難度

迷宮 / 項目	2003E	2004E	2005E	2006E	2007E	2008E	2009E	2010E
節點數	75	54	96	69	69	39	35	46
邊數	91	69	106	75	74	53	43	55
曼哈頓最短路徑長度	75	86	71	71	37	52	52	56
曼哈頓最短路徑的轉彎數	49	49	44	50	10	21	17	21

表 2 比較不同的演算法在表 1 的迷宮上執行，由起點至終點經過的路徑長度。為了能在不同的迷宮中比較，定義演算法的效能為

$$\eta = \frac{\text{演算法得到的起點至終點的路徑長}}{\text{迷宮的曼哈頓最短路徑長}} \qquad \text{方程式 1}$$

經由上式，統計出表 2 的各法則運用於 2003 至 2010 年日本初級競賽的迷宮效能，其中η越大代表效能越差，η越低則反之，表中可以發現，初級競賽的演算法效能由高至低排名，依序為 1.四象限法 2.洪水搜尋法 3.向心法等，其中深先搜尋法的向心法則的效能，還比四象限洪水搜尋法的效能還優勢，在此可知困難度較低的迷宮，使用簡單法則的向心法則，還比使用複雜度較高的四象限洪水搜尋法還要有優勢。

表 2 各法則運用於 2003 至 2010 年日本初級競賽迷宮效能統計表

法則 / 迷宮	深先搜尋法							四象限法	洪水搜尋法	四象限洪水搜尋法
	中左	中右	左中	左右	右中	右左	向心			
2003E	1.8	1.9	4.6	1.4	4.6	1.1	2.2	1.7	1.8	1.6
2004E	3.5	3.8	3.2	4.0	3.2	3.9	1.1	1.0	1.1	2.9
2005E	4.4	2.9	4.2	2.1	4.4	1.3	2.5	1.2	1.6	2.5
2006E	5.2	3.2	5.2	1.8	4.2	2.6	1.0	1.3	1.4	1.8
2007E	6.4	6.4	4.7	7.2	6.5	7.4	2.7	1.8	3.4	4.6
2008E	3.8	4.3	3.3	1.1	5.8	1.2	1.2	1.2	1.4	1.9
2009E	5.2	4.9	4.9	5.5	4.7	5.0	3.3	1.8	1.7	2.3
2010E	1.5	1.2	3.0	2.5	3.9	4.8	2.0	1.4	1.5	1.8
平均效能	4.0	3.6	4.1	3.2	4.7	3.4	2.0	1.4	1.7	2.4

4.2 專業級預賽

表 3 為 2001 至 2010 年日本專業級預賽迷宮的複雜度及困難度統計。由表 3 可知，曼哈頓的最短路徑轉彎數大多平均維持在 20 至 30 之間，無越來越多的趨勢，因此，2001 至 2010 年專業級預賽的迷宮複雜度幾乎相同，推測是為了評斷電腦鼠是否達到一定的水準；圖 4.1 顯示了專業級預賽迷宮的複雜度趨勢，似乎無逐年增高之趨勢。

表 3 日本專業級預賽迷宮的複雜度及困難度

迷宮 / 項目	2001 EP	2002 EP	2003 EP	2004 EP	2005 EP	2006 EP	2007 EP	2008 EP	2009 EP	2010 EP
節點數	54	90	62	56	76	90	36	77	61	76
邊數	64	99	70	60	91	102	45	101	80	87
曼哈頓最短路徑長度	53	61	51	53	56	57	46	64	58	66
曼哈頓最短路徑的轉彎數	22	24	21	23	26	32	19	31	35	19

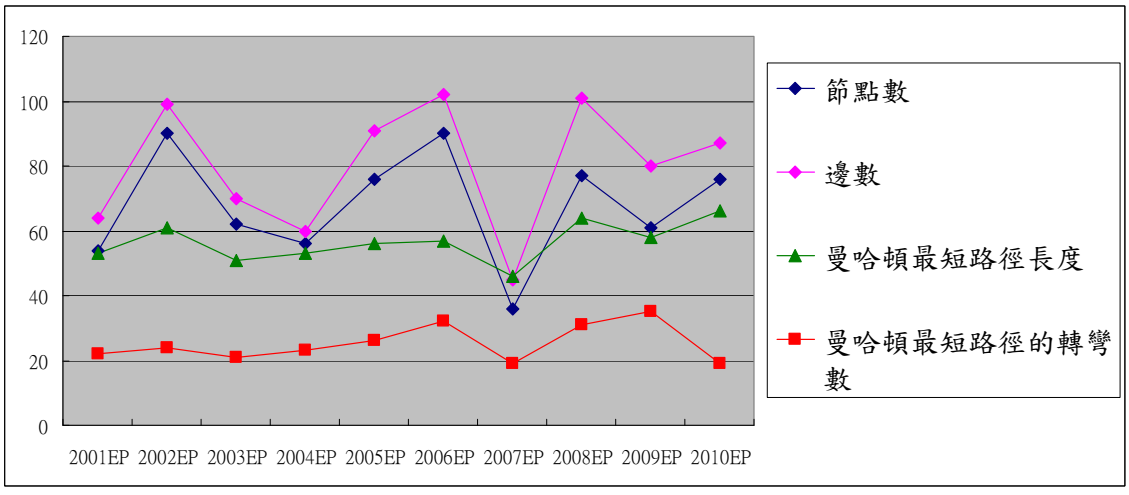


圖 4.1 將表 3 以圖示顯示，迷宮困難度都較為平均

表 4 為各法則運用於 2001 至 2010 年日本專業級預賽迷宮效能統計表，表中的η越大代表效能約低，η越小則反之；其中深先搜尋法的各種法則，還是一樣向心法則較為優勢，其次為右左法則；其它三種法則依序的效能排名為 1. 四象限法、

2. 洪水搜尋法、3.四象限洪水搜尋法，接著將表中的資訊統計成圖 4.2，由圖中可知，唯有深先搜尋法的排名有更變，其餘的法則排名無更動，使用間單的深先搜尋法的法則來搜尋迷宮，可以發現法則效能遠比其他法則還要差 2 倍左右，這意味著使用簡單的深先搜尋法是很容易在預賽中被淘汰的，無法達到電腦鼠的一定水準。

表 4 各法則運用於 2001 至 2010 年日本專業級預賽迷宮效能統計表

法則 / 迷宮	深先搜尋法							四象限法	洪水搜尋法	四象限洪水搜尋法
	中左	中右	左中	左右	右中	右左	向心			
2001EP	4.2	4.8	3.6	1.6	1.2	5.5	1.7	1.6	2.9	3.7
2002EP	1.8	3.1	5.5	4.8	5.0	2.2	1.8	1.8	1.3	1.7
2003EP	5.8	5.9	5.4	4.8	3.4	6.6	1.7	3.3	1.0	1.5
2004EP	5.5	4.8	5.9	3.2	6.4	1.5	2.3	2.1	3.2	2.7
2005EP	6.1	2.6	4.8	5.8	6.1	3.4	1.8	1.3	1.9	2.5
2006EP	5.4	6.1	5.3	2.5	5.1	3.5	5.3	2.3	2.9	3.9
2007EP	4.8	4.8	5.7	6.7	5.1	4.7	1.6	1.6	2.3	3.2
2008EP	5.4	6.0	1.1	2.8	1.1	1.5	2.0	1.4	1.2	1.6
2009EP	5.2	5.3	5.3	4.0	5.7	2.6	3.8	1.8	1.9	2.6
2010EP	3.5	5.5	3.3	4.7	2.9	3.1	1.5	1.5	1.4	1.8
平均效能	4.8	4.9	4.6	4.1	4.2	3.5	2.4	1.9	2.0	2.5

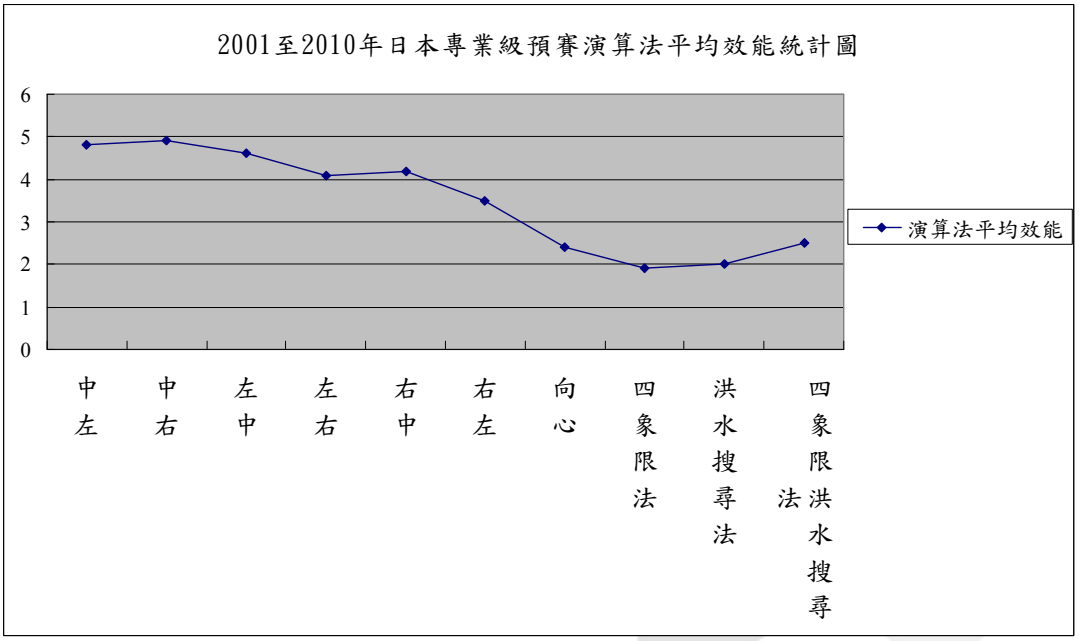


圖 4.2 2001 至 2010 年日本專業級預賽演算法平均效能

4.3 專業級決賽

表格 2 為 2001 至 2010 年日本專業級決賽迷宮的複雜度及困難度統計，相對的此 2001 至 2010 年的競賽迷宮，屬於全世界最高等級的迷宮，各個國家參加競賽必須先於專業級預賽中脫穎而出，才能夠晉級專業級決賽；由表中可知，迷宮的困難度有越來越困難的趨勢，節點數目由 36 增加到 98，相對的也就是迷宮的叉路變多，迴路也跟著變多；為了能夠更加理解，將表格的數值統計成圖示如圖 4.3 所示，曼哈頓最短路徑的轉彎數有越來越多的趨勢，預賽的轉彎數約落在 20 至 30 之間，決賽有逐年的增加，落在 20 至 50 之間，這意味著 2001 至 2010 年的迷宮困難度有逐漸增加之情況。

表 5 日本專業級決賽迷宮的複雜度及困難度

迷宮 / 項目	2001F	2002F	2003F	2005F	2007F	2008F	2009F	2010F
節點數	36	22	51	54	55	68	64	98
邊數	41	31	58	66	61	80	79	105
曼哈頓最短路徑長度	52	53	40	43	71	80	59	57
曼哈頓最短路徑的轉彎數	19	21	15	16	33	35	49	34

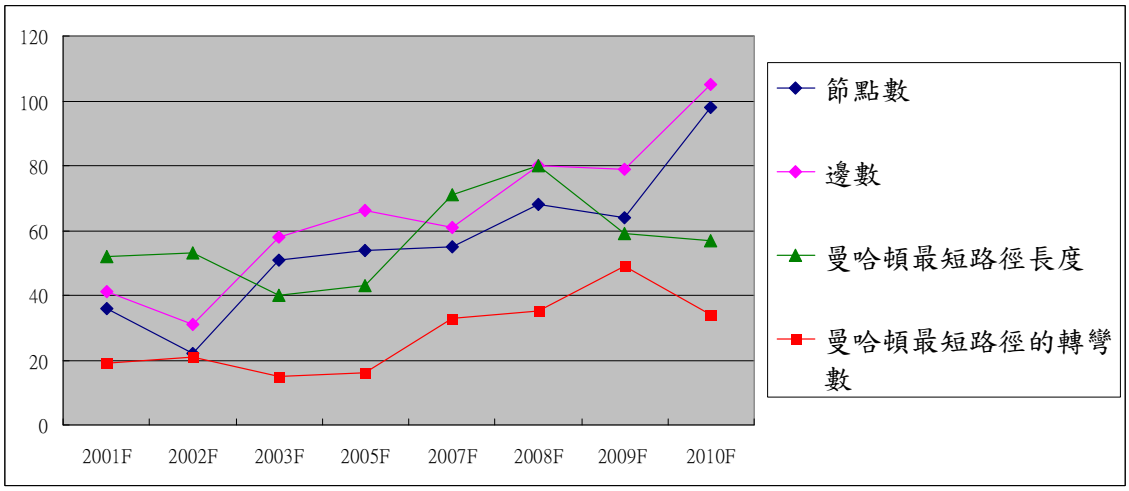


圖 4.3 將表 5 以圖示顯示，節點及邊數有越來越高的趨勢

表 6 為各法則運用於 2001 至 2010 年日本專業級決賽迷宮效能的統計表，表

中的η越大代表效能約低，η越小則反之；對於日本專業級決賽迷宮的難度，可知最佳效能為洪水搜尋法，其次為四象限法，由表中的 2010F 地圖，四象限法的效能遠遠比洪水搜尋法還要差，也較差四象限洪水搜尋法，由此狀況可知，往後的迷宮複雜度會越來越高，很有可能使用四象限法來搜尋迷宮效能會很差；另一方面，深先搜尋法中的中右法則，平均效能勝過於向心法則，推測歷屆以來大多數的參賽隊伍都使用向心法則，因此專業級的決賽迷宮，競賽單位都有特別經過設計，抵制使用向心法則，讓使用向心法則的參賽隊伍無法得到優異的成績，逼參賽隊伍想出創新的搜尋演算法，相關效能統計圖如圖 4.4 所示。

表 6 各法則運用於 2001 至 2010 年日本專業級決賽迷宮效能統計表

法則 / 迷宮	深先搜尋法							四象限法	洪水搜尋法	四象限洪水搜尋法
	中左	中右	左中	左右	右中	右左	向心			
2001F	1.2	2.2	3.4	3.9	4.0	4.3	1.2	1.3	1.2	1.7
2002F	3.7	5.5	3.8	4.1	4.0	3.1	1.2	1.1	2.3	3.4
2003F	5.6	1.7	5.6	4.4	5.1	2.4	2.7	2.8	2.4	3.0
2005F	1.9	1.7	4.2	2.3	5.8	6.0	4.3	1.8	1.5	1.8
2007F	3.0	2.4	1.2	2.1	1.3	1.3	4.0	2.1	1.6	2.7
2008F	3.4	2.0	1.2	2.0	2.3	3.4	2.3	2.0	1.7	2.3
2009F	5.2	5.6	5.2	2.7	4.6	3.9	1.8	1.8	1.9	3.2
2010F	5.2	1.9	4.2	6.0	5.5	2.4	6.6	2.5	1.8	2.6
平均效能	3.7	2.9	3.6	3.4	4.1	3.4	3.0	1.9	1.8	2.6

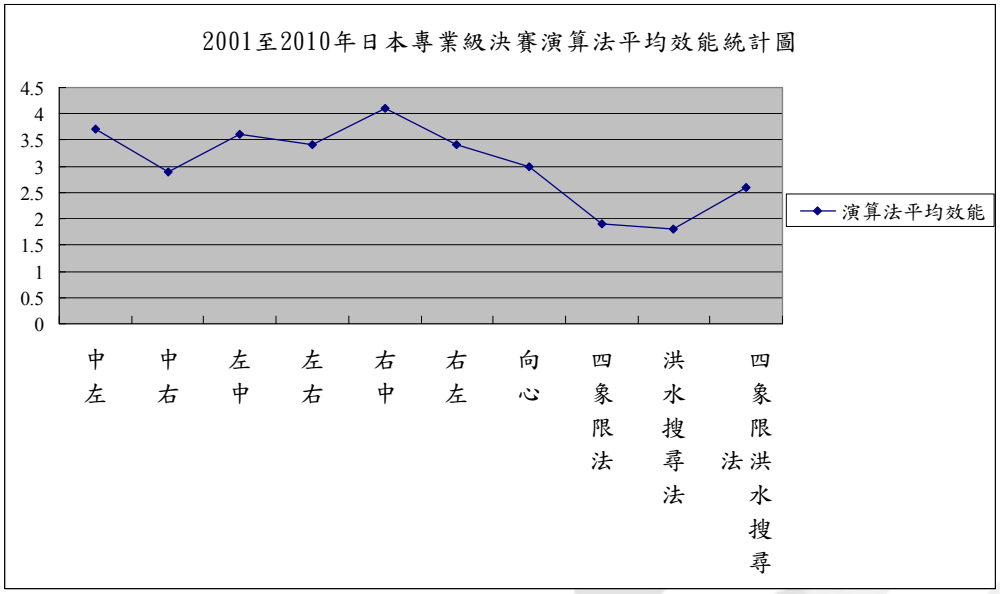


圖 4.4 2001 至 2010 年日本專業級決賽演算法平均效能

第五章 結論與未來展望

本文針對電腦鼠走迷宮之迷宮演算法探討，提升搜尋迷宮演算法之效能，一方面會影響電腦鼠搜尋迷宮的時間，另一方面能夠增加搜尋到迷宮中最佳路徑的機率，本文在此提出了多種迷宮搜尋演算法，並且探討迷宮的複雜度，從未知的迷宮情況下，找尋到迷宮終點，更近一步的在次找尋迷宮中的最佳路徑，直到回到起點；接著運用洪水演算法從已知的迷宮規劃出最佳路徑。

藉由電腦鼠走迷宮競賽的主題，使得全世界的各個單位，都在探討人工智慧搜尋迷宮的思考方式，運用各種方法在最短的時間內找尋到終點，這研究主題非常值得深入去探討，有利於未來機器人找尋地點之方法。

回程搜尋的探討是個很重要的課題，假若能夠很快速的第一次搜尋到終點，但是搜尋的路徑並不是迷宮的最短路徑，則必須經由回程的搜尋，再次找尋迷宮中的最短路徑，猜測迷宮中極有可能的最短路徑，找出迷宮中僅有一條的最短路徑，才是迷宮搜尋的主要關鍵。

最佳路徑的選擇，本文使用曼哈頓路徑長度作為最佳路徑的選擇，往後可以計算歐幾里得路徑的長度，作為迷宮最佳路徑的選擇。

路徑的規劃可應用於未來機器人的路徑移動，行走對於自己較有利的路徑，減少移動的時間，可以提高機器人的整體效率，就如人類的行走方式，並不會選擇死板板的行走方式，而會運用各種閃避的技巧，揣摩各種可行的移動方式，達到更迅速的移動到目的地，這些應用的技巧，都是經由碰撞、學習的經歷，慢慢的知道自己可行走的範圍，進而改善移動方式。

參考文獻

[1] Japan Micromouse Competition, http://www.ntf.or.jp/mouse/

[2] The 2010 China Micromouse Competition, http://www.micromouse.com.cn/

[3] The 2010 Taiwan Micromouse Competition, http://www.eecs.stut.edu.tw/~robot2010/

[4] H. Choset, M. K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, Principles of robot motion: Theory, Algorithms, and Implementations, MIT Press, Boston, Jun. 2005.

[5] A* algorithm tutorial, http://www.geocities.com/jheyesjones/astar.html

[6] P. Lester, "A* path_finding for beginners," http://www.policyalmanac.org/games/aStarTutorial.htm

[7] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge, May 2004

[8] A* search algorithm, http://en.wikipedia.org/wiki/A*_search_algorithm

[9] C. Yifeng, Z. Hengkai, W. Wanggen, Y. Xiaoqing, "A high efficiency center-first-routing-theorem algorithm of micro-mouse," 2008 International Conference on Audio, Language and Image Processing, pp. 778-793 , Jul. 2008

[10] C. Jianping, W. Jianzhong, H. Meimei, H. Jian, "A micromouse maze sovling simulator," 2010 2nd International Conference on Future Computer and Communication, pp. V3-686 - V3-689 , May 2010

[11] L. Wyard-Scott, Q.-H.M. Meng, "A potential maze solving algorithm for a micromouse robot," IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, pp. 614 - 618 , May 1995

[12] A.F. Golda, S. Aridha, D. Elakkiya, "Algorithmic agent for effective mobile robot

navigation in an unknown environment," 2009 International Conference on Intelligent Agent & Multi-Agent Systems, pp. 1 - 4 , Jul. 2009

[13] B.H. Kazerouni, M.B. Moradi, P.H. Kazerouni, "Variable priorities in maze-solving algorithms for robot's movement," IEEE International Conference on Industrial Informatics , pp.181 - 186 , Aug. 2003

[14] V.S. Gordon, Z. Matley, "Evolving sparse direction maps for maze pathfinding," 2004. Congress on Evolutionary Computation , pp. 835 - 838, Jun. 2004

[15] M.J. Sadik-Adil, A. Dhali-Maruf, M.A.B. Farid-Hasib, "A comprehensive and comparative study of maze-solving techniques by implementing graph theory," 2010 International Conference on Artificial Intelligence and Computational Intelligence, pp. 52 - 56, Oct. 2010

[16] L. Xihua, J. Xang, J. Xudan-xu, L. Xiao-Haiyue, "An improved algorithm of the exploring process," 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems, pp. 324 - 328, Oct. 2010

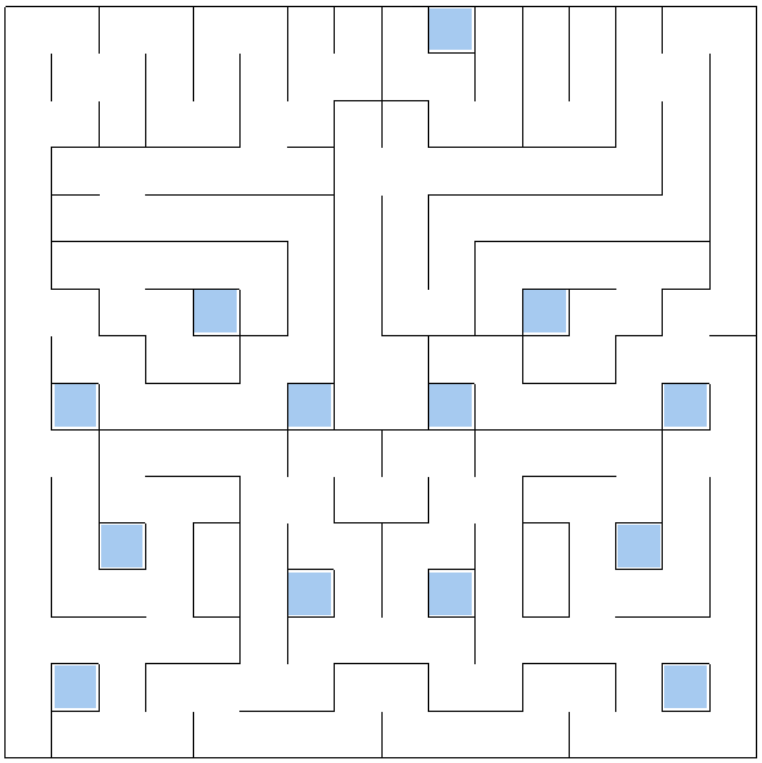
[17] M. Sharma, K. Robeonics, "Algorithms for micro-mouse," 2009 International Conference on Future Computer and Communication , Kuala Lumpar , pp. 581 - 585, Apr. 2009

[18] R. Tarjan, "Depth-first search and linear graph algorithms," SIAM Journal of Computing, Vol. 1, No. 2, pp. 146 - 160, Jun. 1972

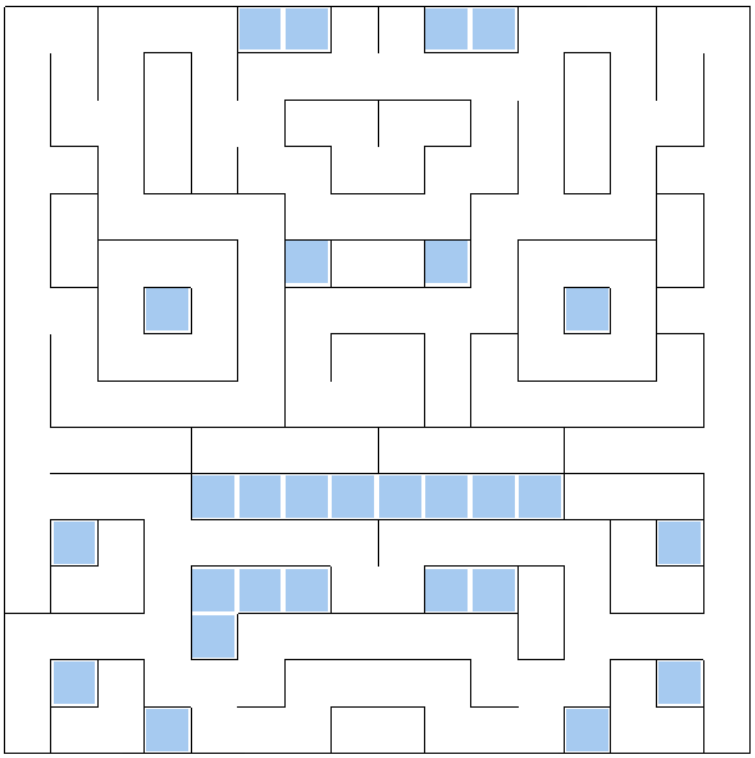
[19] S. Mishra, P. Bande, "Maze solving algorithms for micro mouse," 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems , Bali , pp. 86 - 93, Dec. 2008

[20] R. Dechter, J. Pearl, "Generalized best-first search strategies and the optimality of a*," Journal of the ACM, Vol. 32, No. 3, pp. 505 - 536, Jul. 1985

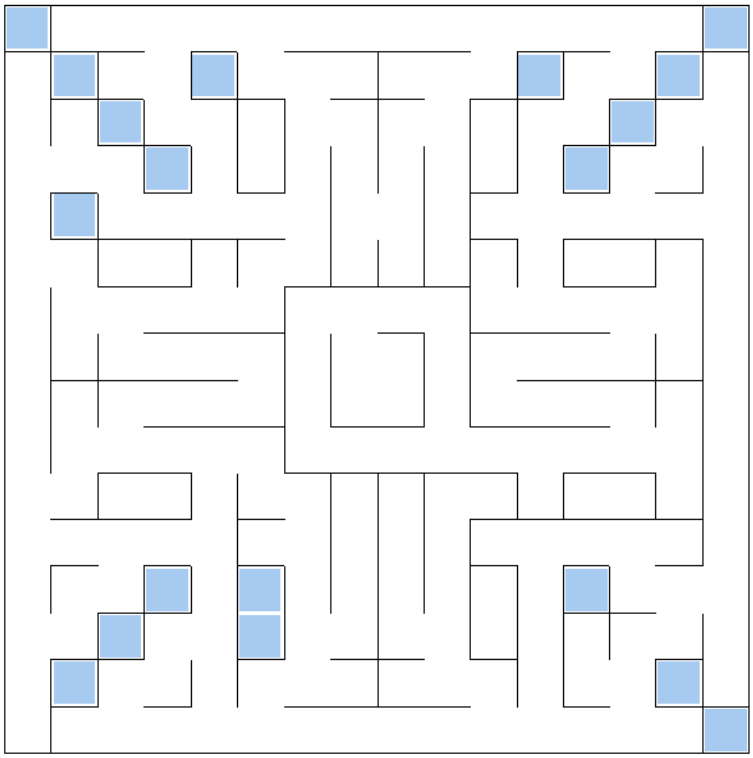
附錄



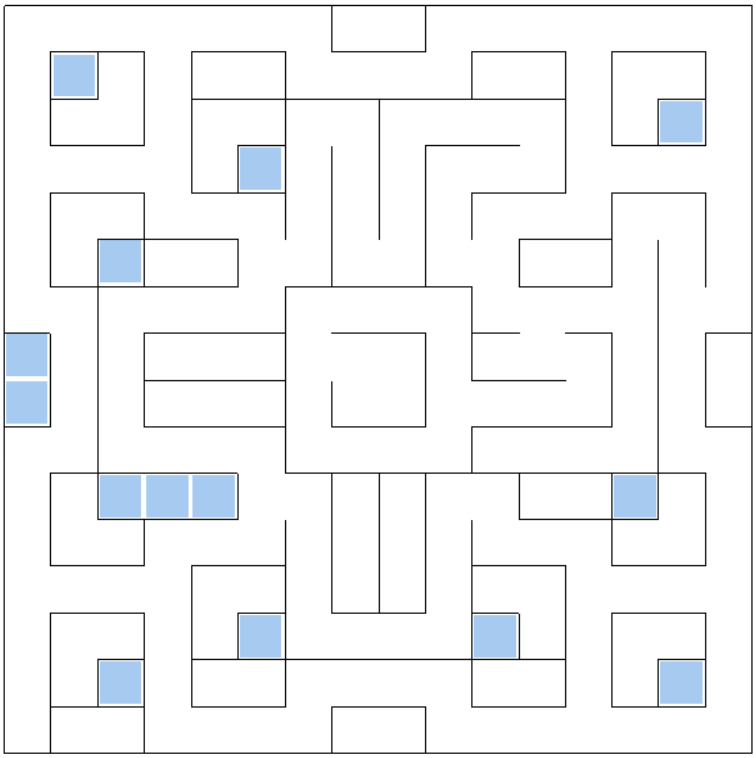
2001EP



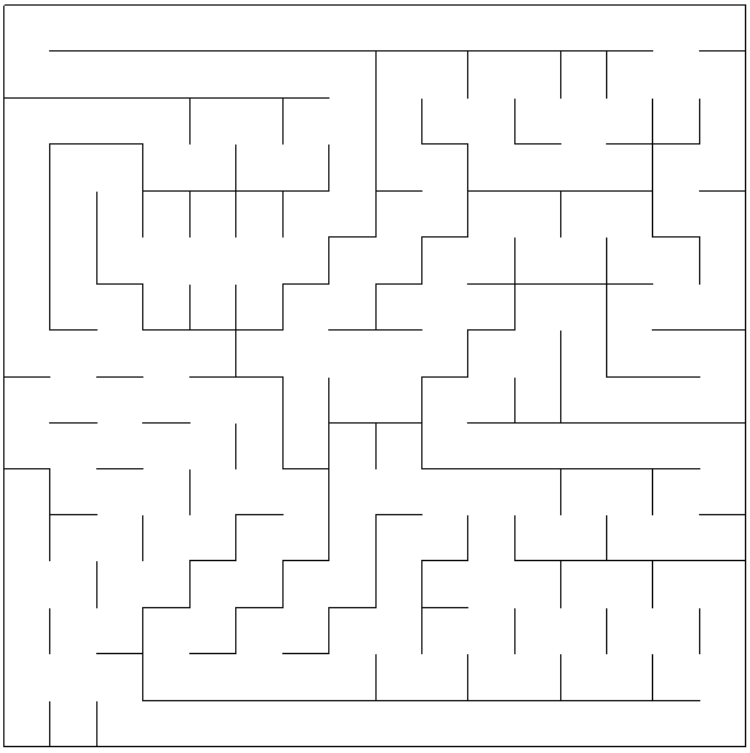
2001F



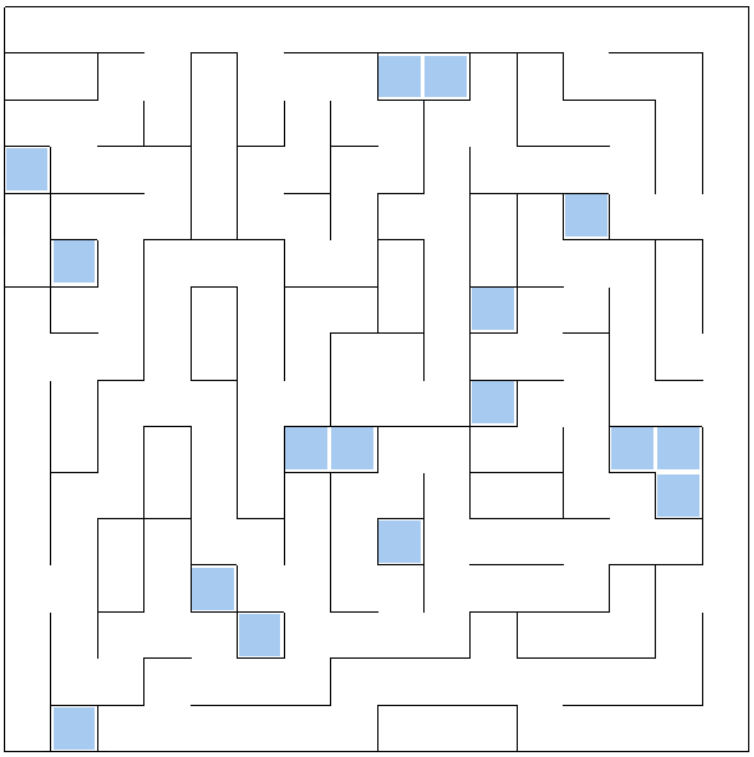
2002EP



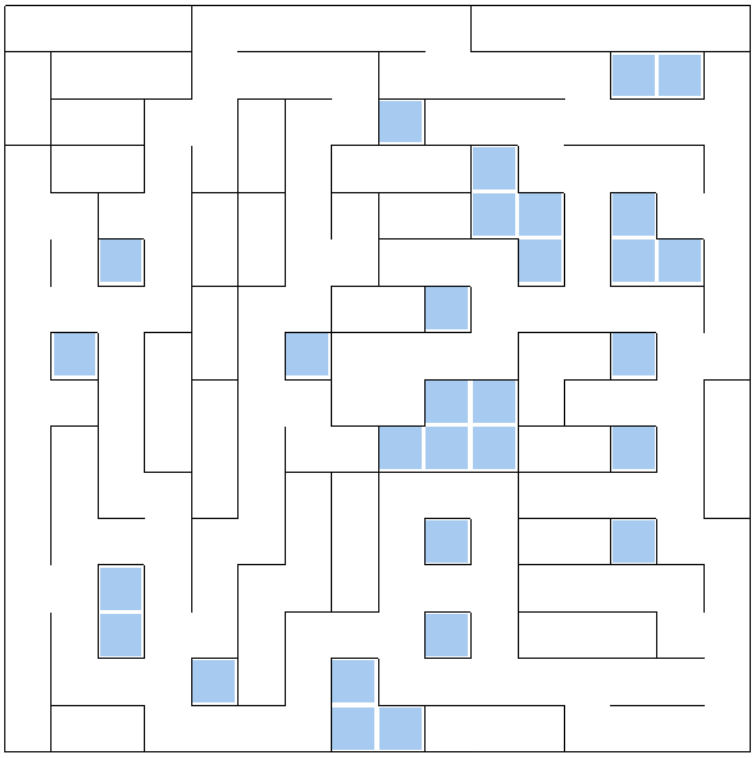
2002F



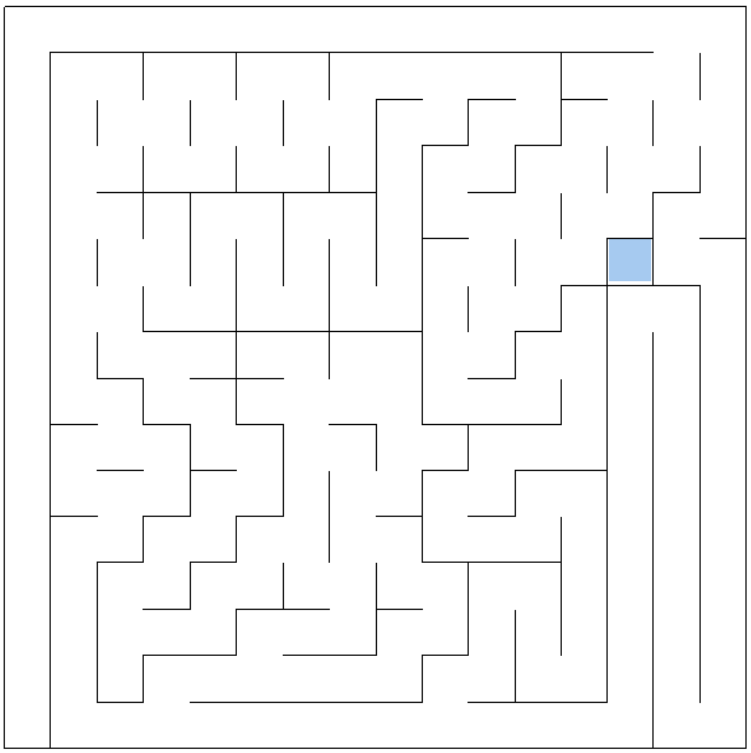
2003E



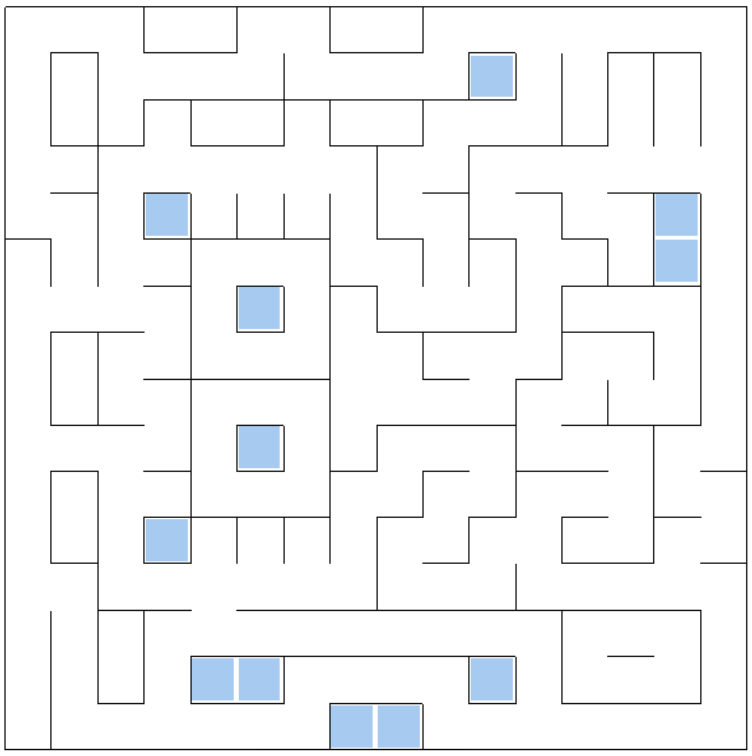
2003EP



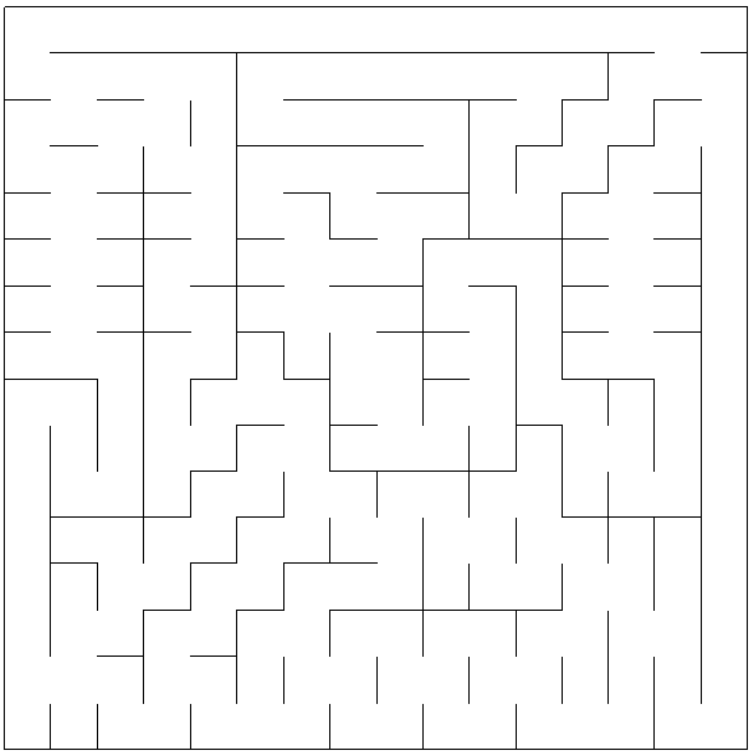
2003F



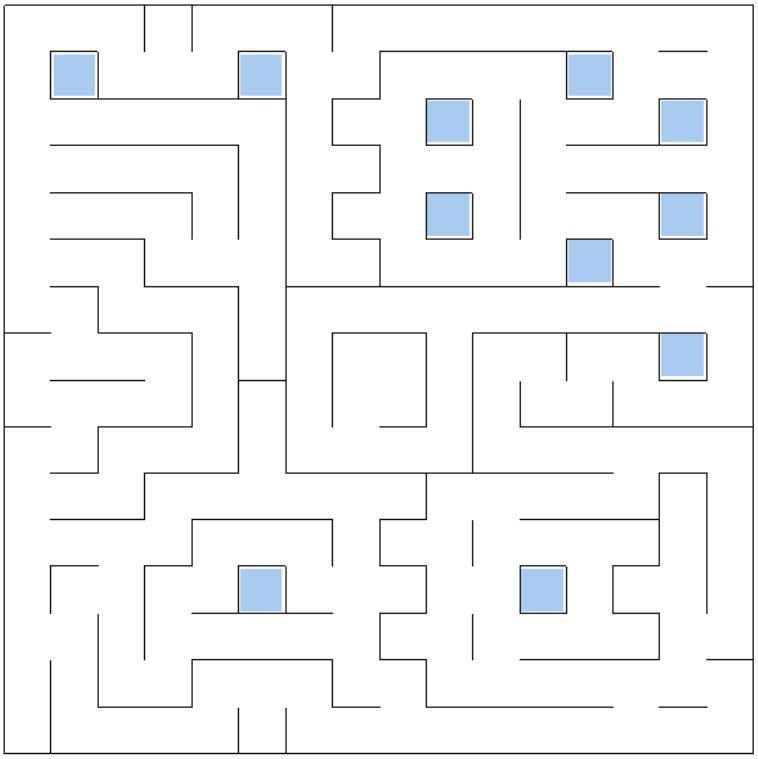
2004E



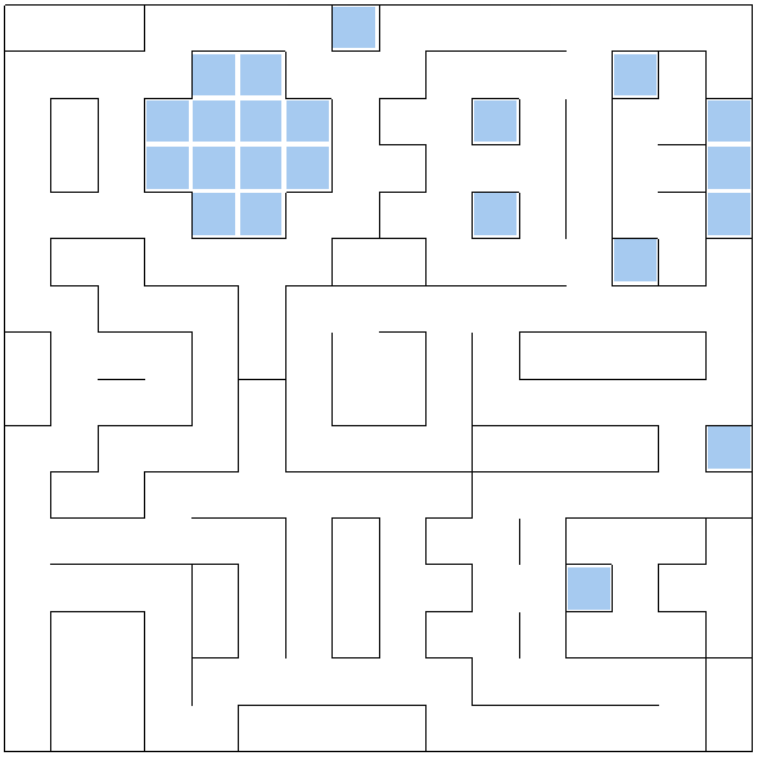
2004EP



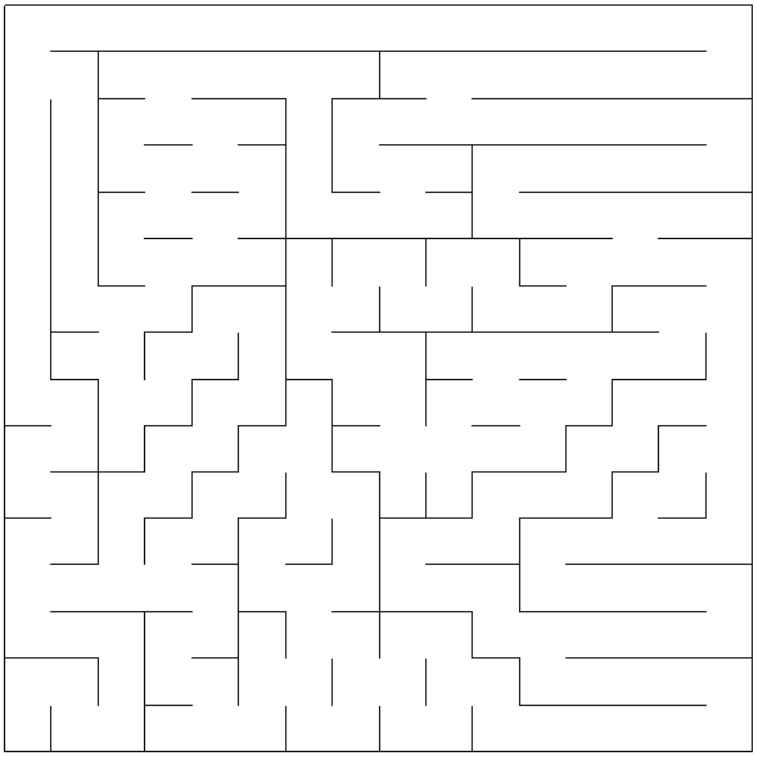
2005E



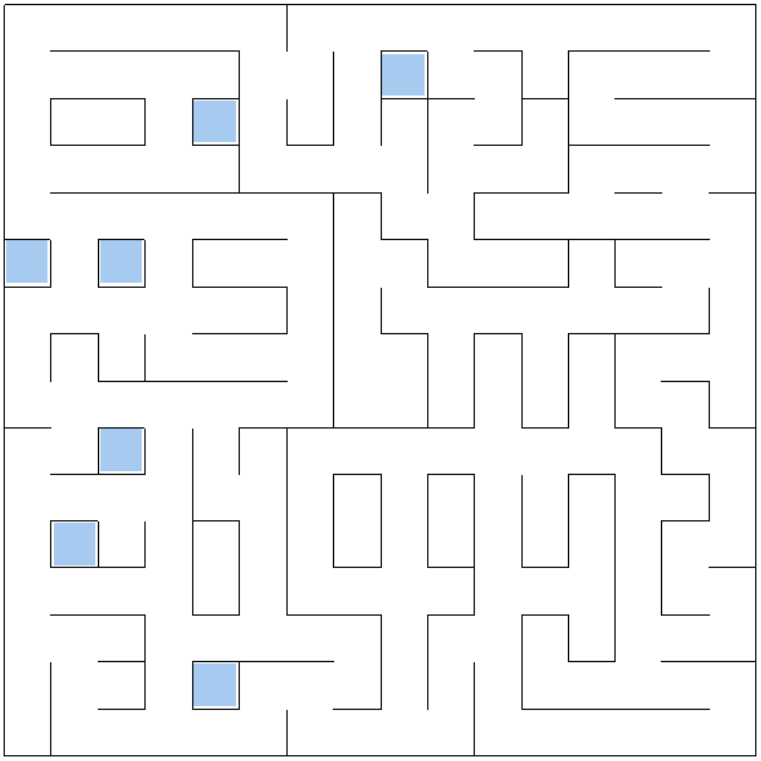
2005EP



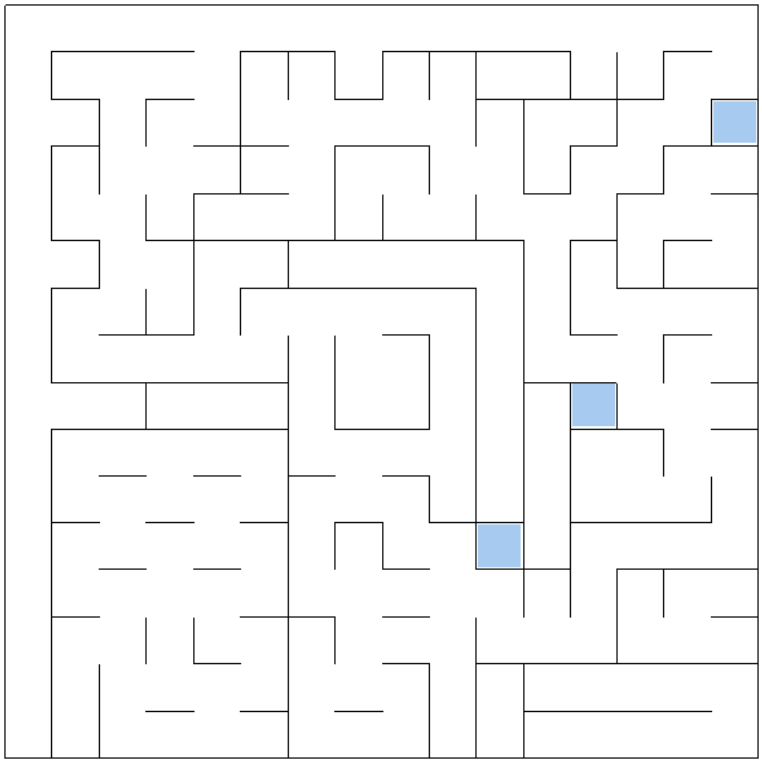
2005F



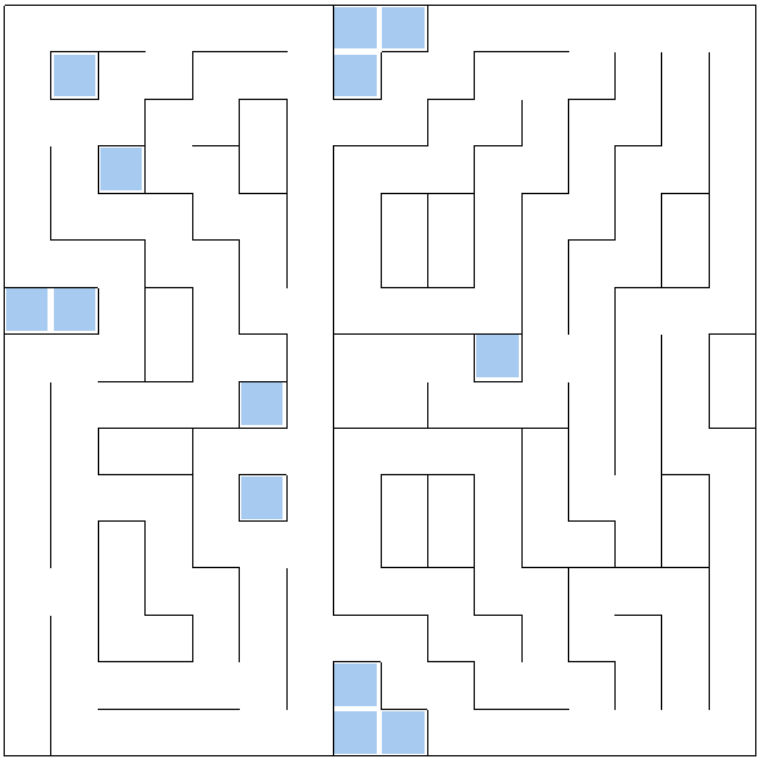
2006E



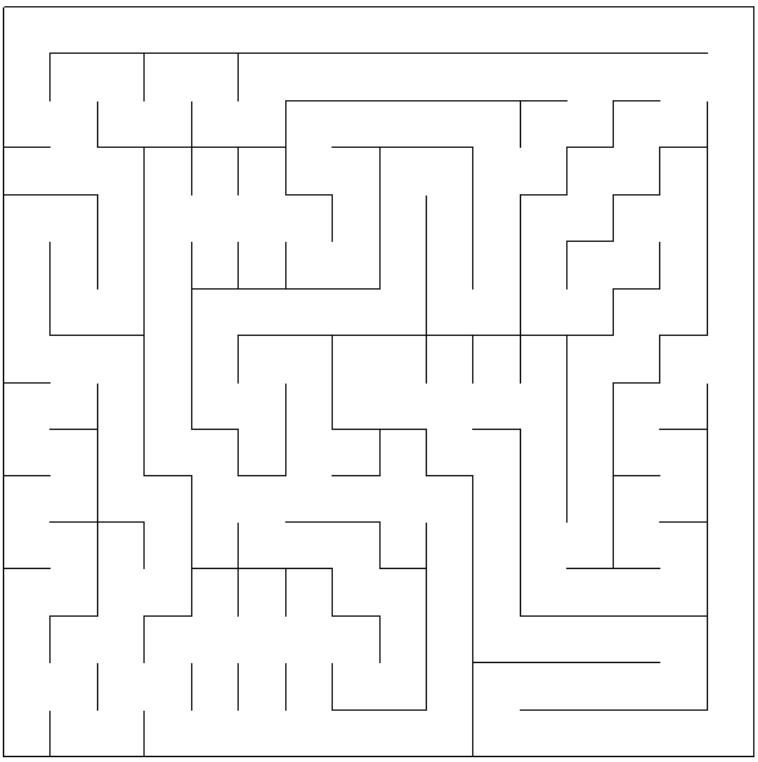
2006EP



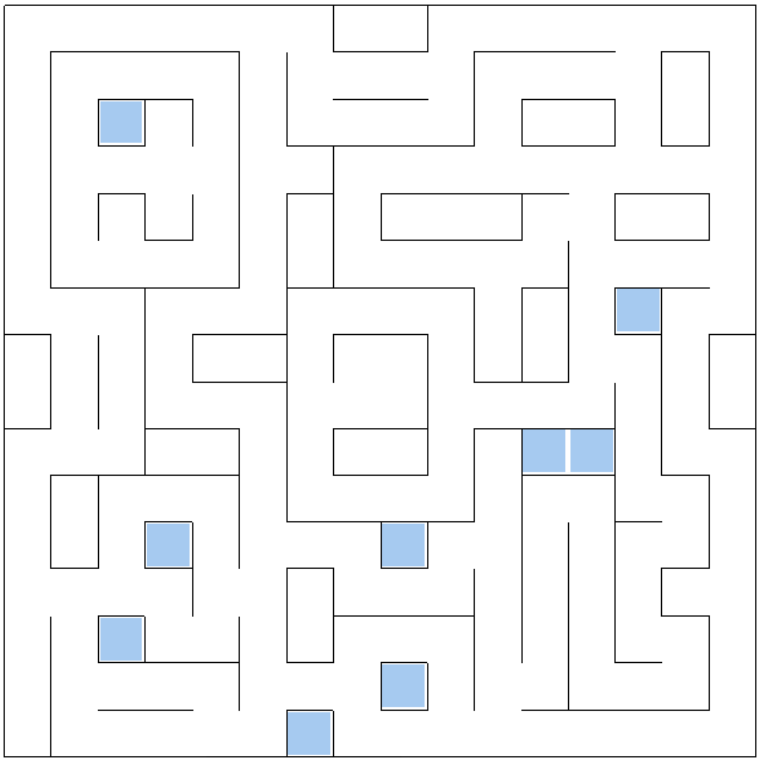
2007E



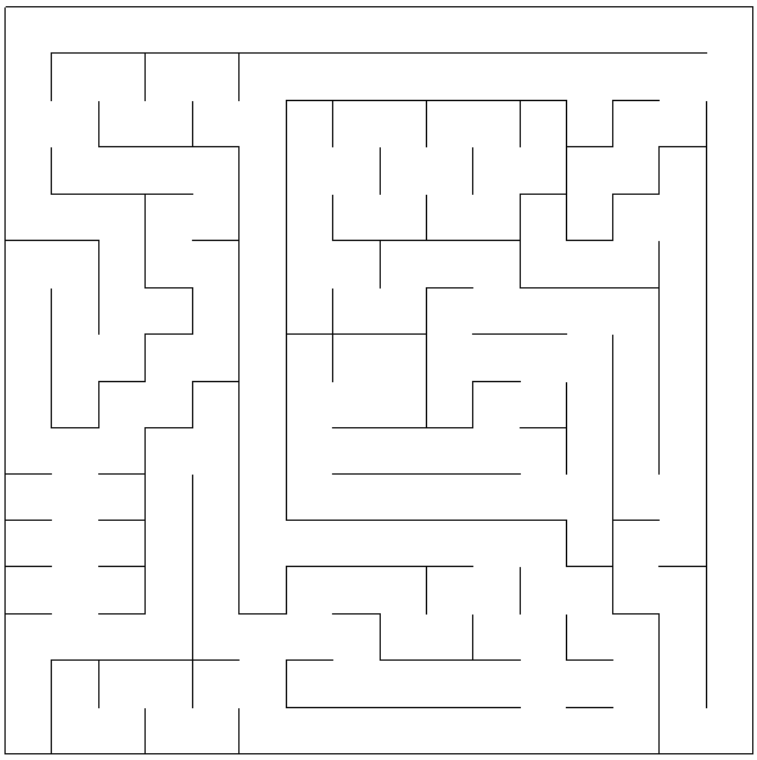
2007EP



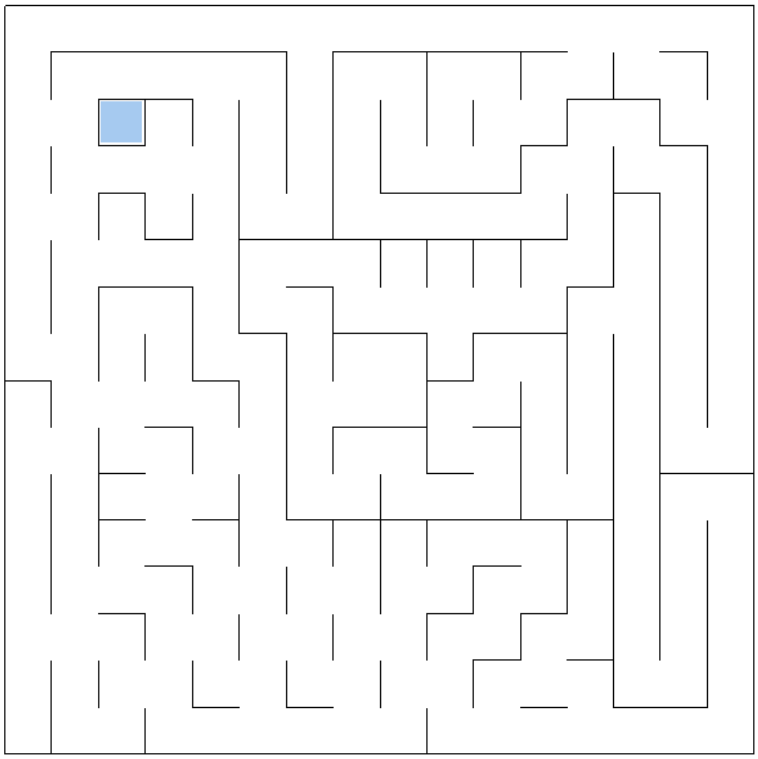
2007F



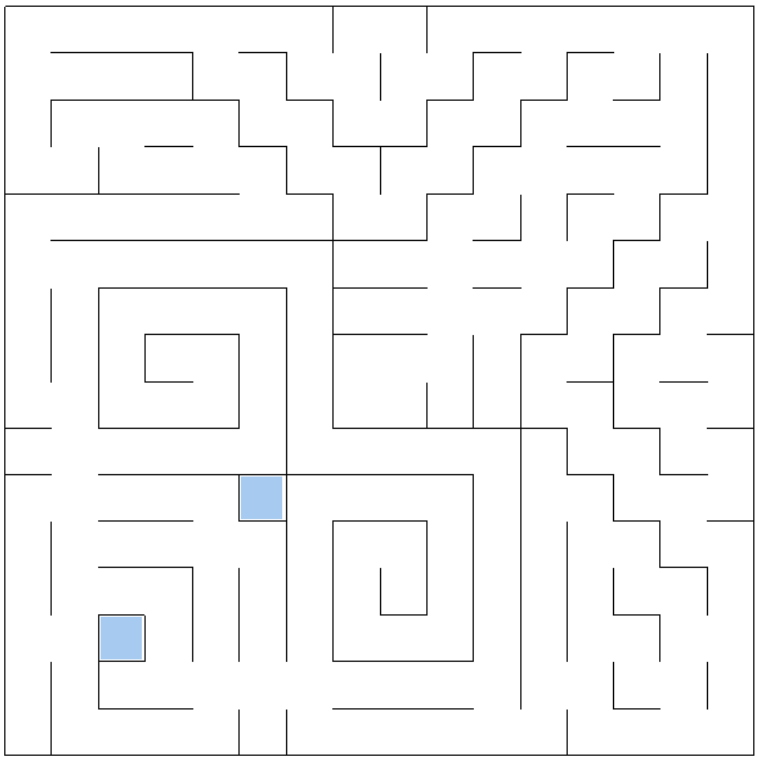
2008E



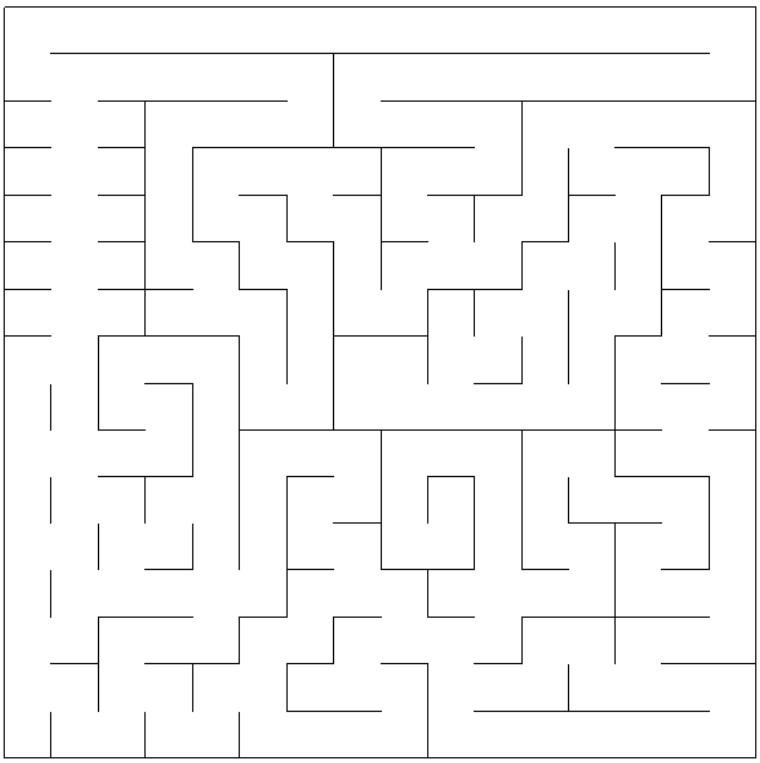
2008F



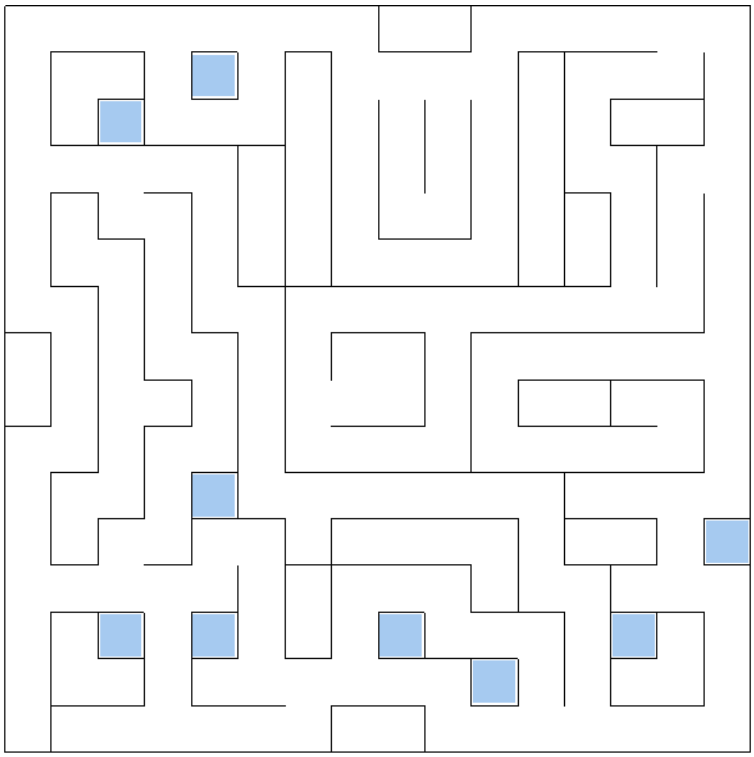
2008EP



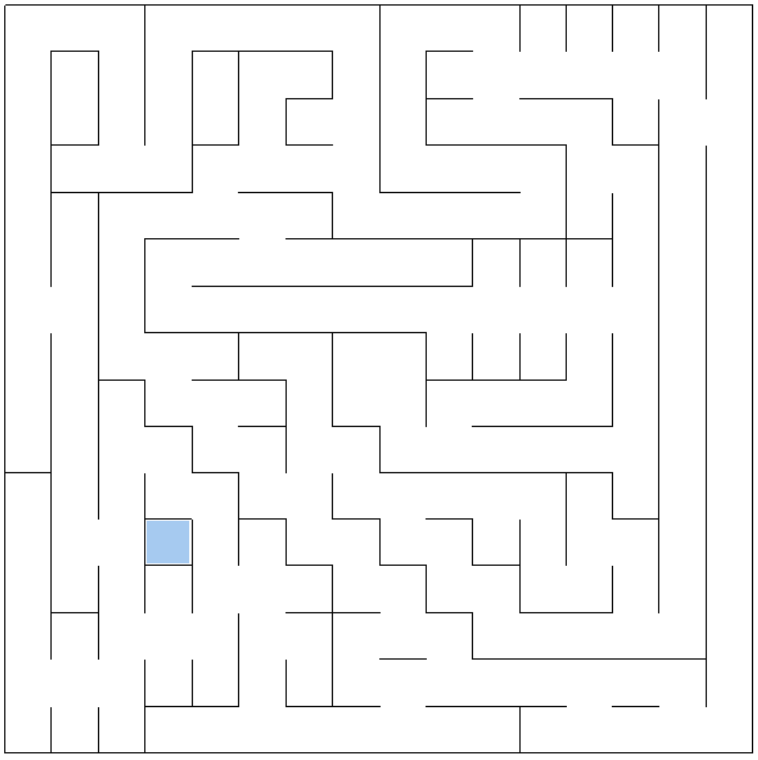
2009EP



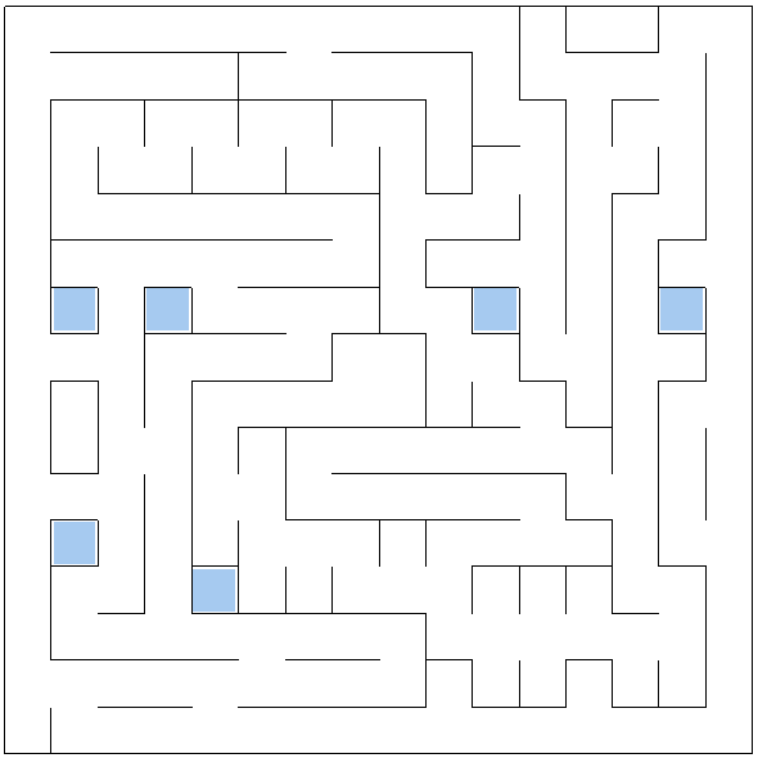
2009F



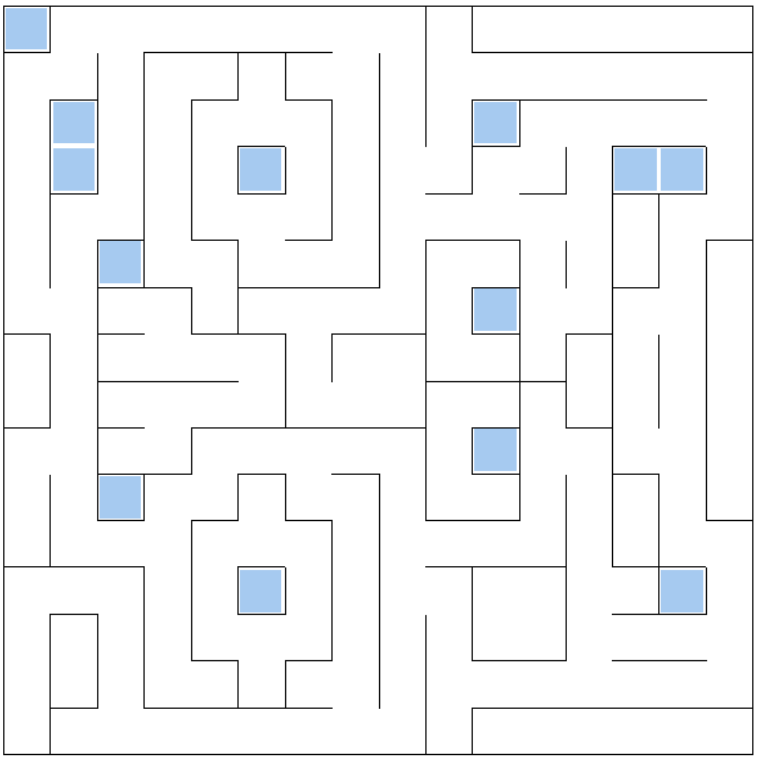
2009E



2010F



2010EP



2010E