迷宮搜尋演算法

軟體的主要目的是在任何時候保持對硬體的控制，及決定如何移動以解決迷宮問題。軟體控制硬體的範圍包括理解感測器偵測得到的資訊，設定馬達的轉速，以及與外在周邊設備的通訊。由於每個馬達的速度都是獨立控制，因此可以經由增加或減少某一個馬達的速度對電腦鼠的行進路徑做修正。除了控制硬體，軟體也必須記憶電腦鼠在迷宮中走過的位置及當下的位置，並根據路徑選擇的演算法決定下一步的移動方式。

1. 迷宮的圖形表示法

圖 20 是一個 10×10 的迷宮，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，節點間的邊以綠色線條表示。我們可以發現圖中有許多迴路，因此電腦鼠必須有記憶功能，避免搜尋重複的節點，否則就可能陷入迴路中，一直兜圈圈。

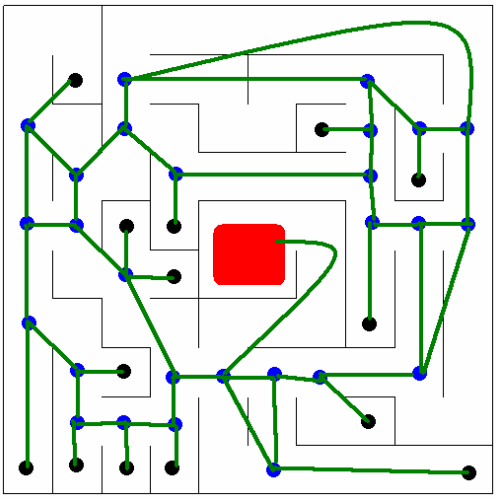


圖 20、迷宮範例與其對應之圖形

2. 迷宮的難度

3. 迷宮的搜尋演算法

電腦鼠的主要目標是解決迷宮問題，盡快讓電腦鼠到達迷宮的中心。為了完成這個任務，電腦鼠使用特定的迷宮搜索算法。

目前已經存在許多的搜索技術並被採用。幾個世紀以來，研究拓撲學和圖論的數學家已經探討建立迷宮的演算法及解決迷宮的演算法。然而，受到大多數電腦鼠微控制器的記憶體與速度的限制，他們開發的算法並不適合電腦鼠的使用。此外，由於數據搜索是電腦的基本功能，在過去幾十年，電腦科學家也開發了一些不同的搜索技術。同樣地，由於這些算法是建立在先進的微處理器上，因此即使他們非常的效率高且結果好，它們仍不能被執行在較落後的電腦鼠微控制器上。因此，電腦鼠通常使用的搜索演算法是沿壁法、深度優先法及洪水充填法等。

3.1 沿壁法(Wall 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牆，然後沿著這一面牆前進直到到達終點為止。沿壁法的程式非常簡單，但他不保證一定能解決迷宮搜尋問題。特別是目前的電腦鼠迷宮在設計時通常會設計成無法用沿壁法得到解答。下圖的迷宮就是一個例子，如果採用沿壁法，電腦鼠將會一直在最外圈繞圈圈。

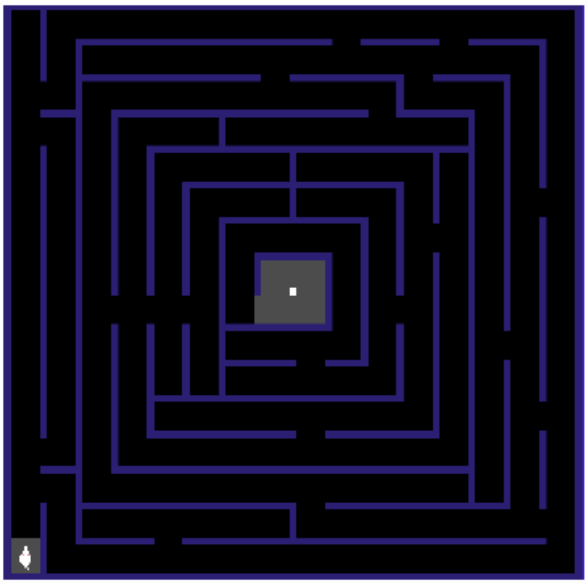


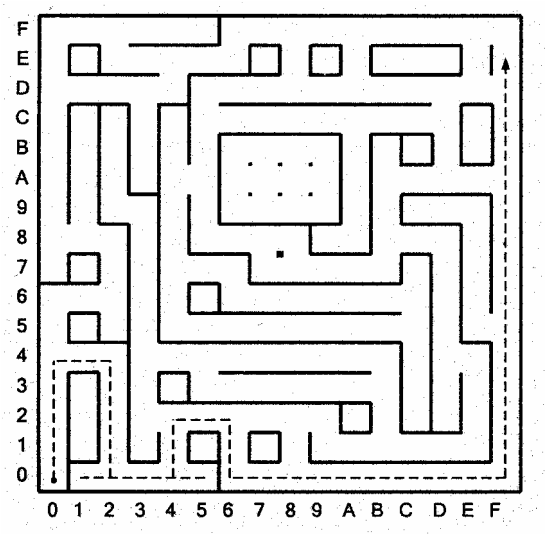
Figure 14

3.2 深先搜索法

深度優先搜索是一個直觀的迷宮搜演算法，電腦鼠在遇到岔路時隨機選擇一條沒有走過的路徑前進，如果進入死巷，電腦鼠折返回到路口，選擇另一條道路繼續往下走。此演算法會搜尋迷宮中的每一條路徑，每一個連通的格點，直到到達終點為止。它被稱為“深度優先”，因為，如果迷宮被表示成一顆生成樹，電腦鼠會優先搜尋下一層的節點，直到葉節點後再回頭。深度優先搜索法的優點是只要到終點的路徑存在，它一定能夠到達終點。

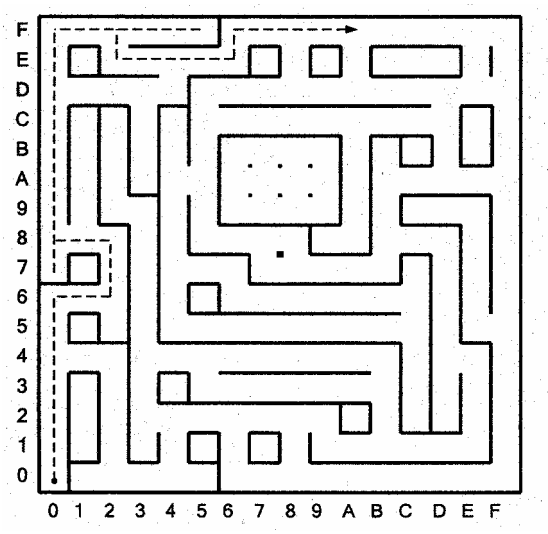
在深先搜尋演算法中，當電腦鼠到達一個岔路節點後，就訪問其中一個未被搜尋的節點，直到死巷子節點或遇到已訪問過的岔路節點為止。接著，再回溯到前一個岔路節點繼續依深度優先的原則追蹤其他相鄰節點。在搜尋的過程，當遇到岔路節點時，依行走方向的選擇不同，可分成下列方法:

1. 右手法則：遇到岔路節點時優先右轉，其次是直走、左轉。



右手法則

2. 左手法則：遇到岔路節點時優先左轉，接著是直走、右轉。



左手法則

3. 中左法則：遇到岔路節點時優先走直線，其次是左轉、右轉；

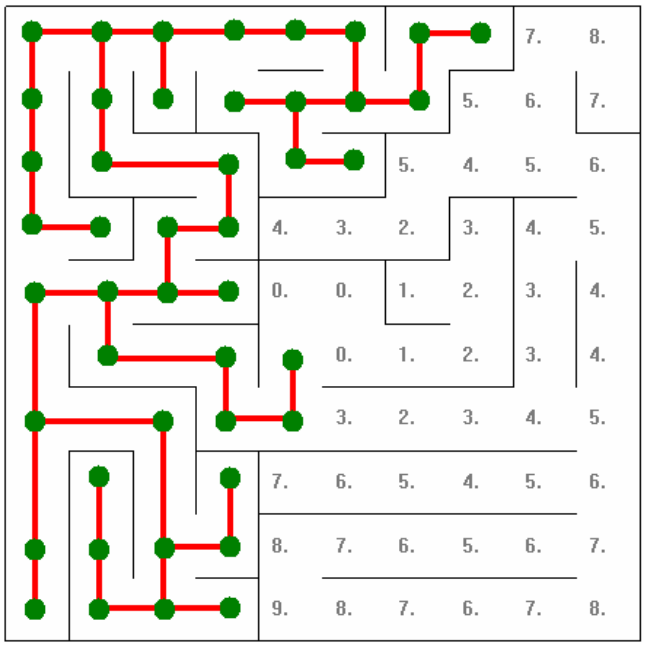
4. 中右法則：遇到岔路節點時優先走直線，其次是右轉、左轉；

5. 亂數法則：沒有固定優先權，依亂數決定轉向；

向心搜尋演算法是一種改良過的深先搜索法，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，選擇向迷宮中心的路口優先進入。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 大樓方位的街道前進，這種方法通常比在岔路時隨機選擇一條街道前進有效的多。因此，和傳統的深度優先搜索法不同的地方是，向心搜尋演算法善用終點就在迷宮中心處的知識，向心搜尋演算法的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 21 的權重表，遇到岔路節點時優先往權重較小的格點前進。

13.	12.	11.	10.	9.	8.	7.	6.	7.	8.
12.	11.	10.	9.	8.	7.	6.	5.	6.	7.
11.	10.	9.	8.	7.	6.	5.	4.	5.	6.
10.	9.	8.	5.	4.	3.	2.	3.	4.	5.
9.	8.	7.	4.	0.	0.	1.	2.	3.	4.
9.	8.	7.	4.	0.	0.	1.	2.	3.	4.
10.	9.	8.	5.	4.	3.	2.	3.	4.	5.
11.	10.	9.	8.	7.	6.	5.	4.	5.	6.
12.	11.	10.	9.	8.	7.	6.	5.	6.	7.
13.	12.	11.	10.	9.	8.	7.	6.	7.	8.

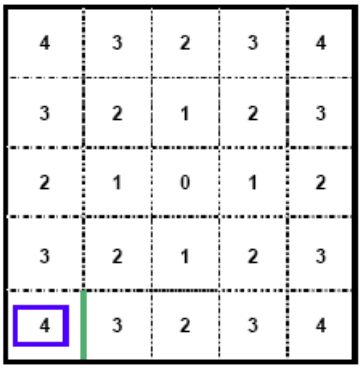
圖 21、向心法則權重表



向心搜尋演算法的範例

3.3 洪水填充法

洪水填充法，也稱為貝爾曼的算法(Bellman's algorithm)，與上述的種演算法相比是更適合的方法。該演算法使用距離和牆壁的知識去尋找到達迷宮中心的較短路徑。在這個該演算法中，每一個格子存在一個代表距離的物理量，此物理量代表該格子與迷宮中心可能的最短的曼哈頓距離(為方便本文就直接稱此物理量為距離)。剛開始假設迷宮中沒有牆壁存在，因此如下的迷宮，每個格子的距離如圖所示。

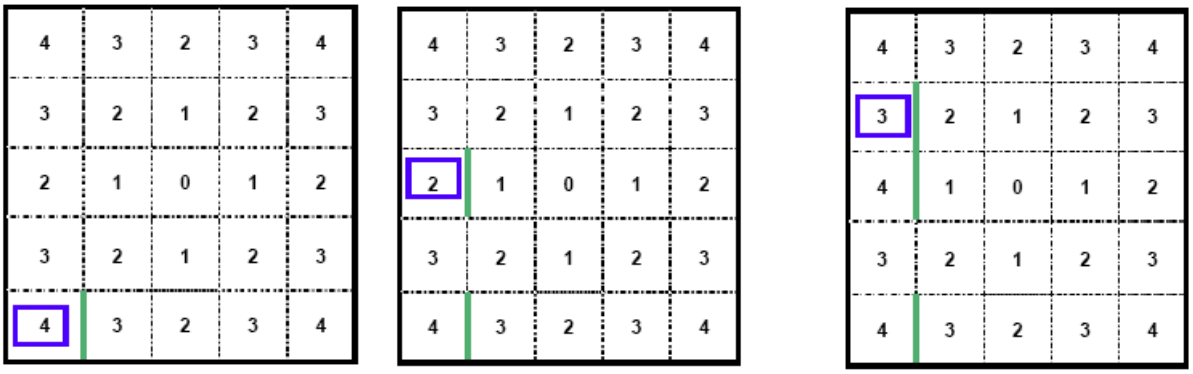


接著，當電腦鼠進入每一個格子時，它必須偵測該格子的牆壁的情況，並根據牆壁的情況更新該格子的距離。步驟如下:

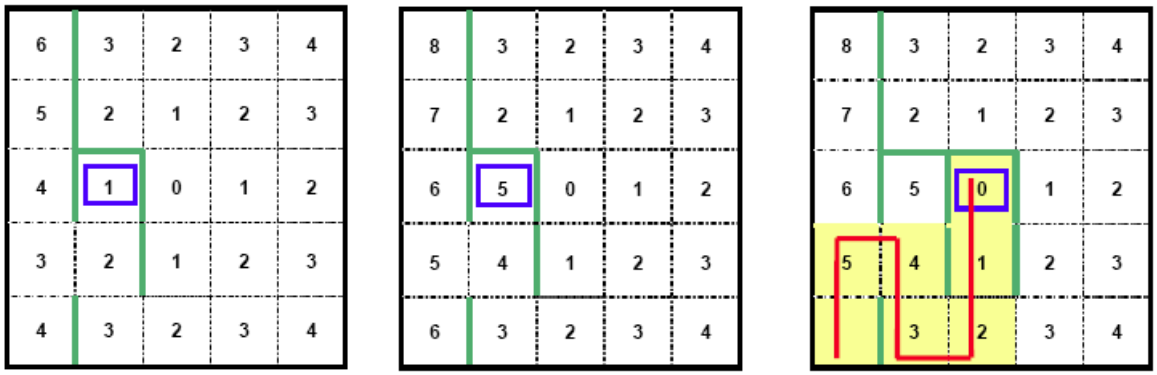
1. 偵測格子的牆壁情況，更新迷宮的資訊。
2. 根據牆壁的情況更新該格子的距離。
3. 找出相鄰且沒走過的格子中，距離值最小的格子，移動至該格子。如果所有相鄰的格子都被電腦鼠搜尋過，則回頭。

洪水填充法被稱為“廣度優先”演算法，因為如果迷宮被表示成一個生成樹，演算法會先搜查同一層的格子，接著才搜尋下一層的格子。洪水填充算法的主要優點是，它總能找到到達迷宮中心的最短路徑。但值得注意的是，此方法實際走的路徑被非最短路徑。

為了增加洪水填充法的效率，有一些修正的洪水填充法被提出。例如Modified Flood-Fill就是由正規的洪水填充法修正得到，此方法再更新距離值時，只針對需要的格子修正，而非全面性的更新。下列例子說明修正的洪水填充法。



1. 初使，電腦鼠事先沒有圍牆的資訊。電腦鼠的起點在左下角，看到右邊有牆，前進進入距離值較小的格子。
2. 電腦鼠繼續沿著距離值最小的道路前進，發現格子東邊有牆。相鄰的格子都有較大的距離值，因此它必須調整其目前的距離值成為比最低的鄰居值多 1。
3. 電腦鼠繼續前進到下一格子，發現新格子的東邊有牆，更新迷宮的資訊。再一次，相鄰的格子都有較大的距離值，因此它必須調整其目前的距離值成為比最低的鄰居值多1。



4. 電腦鼠繼續前進到左上角，更新牆壁和距離的資料，然後沿著最低距離值的道路朝向迷宮的中間移動。在當前位置，格子向南方的距離值不正確，因此必須更新。

5. 更新後的距離值如圖中所示。

6. 找到最短路徑，電腦鼠只要沿著距離值遞減的路徑移動就可以到達終點。

3.4 A*演算法

相對於洪水填充法採用廣先搜尋策略，A* 演算法採用了“佳者優先”(Best-First Search)的搜尋策略，路徑的好壞是經由下列公式評估:

$$f(n) = g(n) + h(n) \tag{12}$$

其中 f(n)是目前隔點的路徑評估長度，g(n)是起點到目前節點的距離，h(n)是預測目前節點到終點的距離，h(n) 主導著 A* 演算法的表現方式，有以下幾種情形:

1. h(n)=0:等同於洪水填充法，並且保證能找到最短路徑。
2. h(n)<目前節點到結束點的距離: A* 演算法保證找到最短路徑，h(n)越小，搜尋深度越深。
3. h(n)=目前節點到結束點的距離: A* 演算法僅會尋找最佳路徑，並且能快速找到結果。
4. h(n)>目前節點到結束點的距離: 不保證能找到最短路徑，但計算比較快。

A* 演算法的虛擬碼如下:

Add **START** to **OPEN** list

while **OPEN** not empty

get node **n** from **OPEN** that has the lowest **f(n)**

if **n** is **GOAL** then return path

move **n** to **CLOSED**

for each **n'** = CanMove(**n**, direction)

g(n') = **g(n)** + 1

calculate **h(n')**

if **n'** in **OPEN** list and new **n'** is not better, continue

if **n'** in **CLOSED** list and new **n'** is not better, continue

remove any **n'** from **OPEN** and **CLOSED**

add **n** as **n'**s parent

add **n'** to **OPEN**

end for

end while

if we get to here, then there is ***No Solution***

以上虛擬碼適用於一般遊戲中方格圖(Grid Map)的情形. 當要在真實地圖的路網(Road Map)上使用 A* 演算法時, g(n)與 h(n)的計算方式便會有所不同.