人工智慧 Project

B0629021蔡雅如 B0629025郭宇芹 B0629046蔡宛凌 B0629049林仲岳

**硬體規格:**

* CPU:17-7700K
* GPU:RTX-2070Super
* RAM:HyperX FURY DDR4 3000 8G\*4
* OS:Window 10專業版

**實驗方法:**

先決條件要先認識A\* 演算法，簡單來說它是一種求解最短路徑最有效的直接搜尋法也是一種啟發式算法。在搜尋過程中使用啟發函數。啟發式評價(Heuristic Estimate)公式與實際值越接近，最終搜尋到目標格局的時間越快。

啟發式評價表示:

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

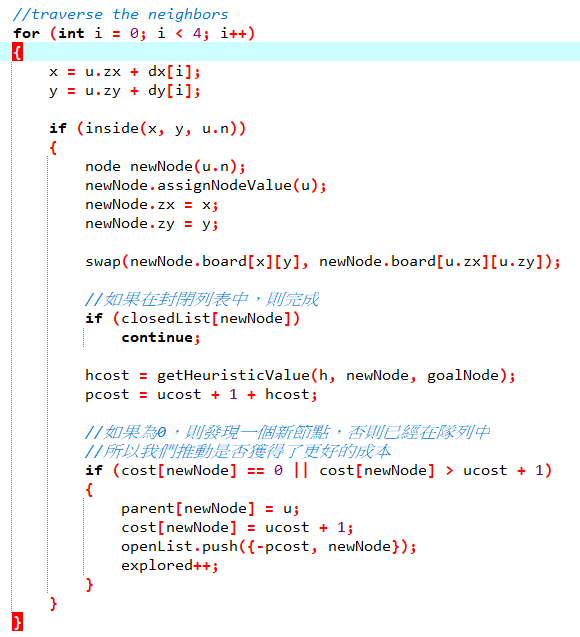
g(n): 從啟始點到目前節點的距離

h(n): 預測目前節點到結束點的距離(此為 A\* 演算法的主要評價公式)

f(n): 目前節點的評價分數

h(n)的好壞直接影響啟發式評價的好壞。

(下圖程式在說明如何尋訪)



**Manhattan:**

我們對每個圖塊執行一次搜索，從其目標位置開始，並記錄將圖塊移動到其他位置。對所有圖塊執行此操作會生成一組表，每個表代表每個磁貼的可能位置，就是距離目標位置的曼哈頓距離。因為我們只計算感興趣的圖塊的移動，而每次移動僅移動一個圖塊，而我們可以從中得出曼哈頓距離，進一步獲得一種可接受的啟發式方法。

一般而言，當我們將圖塊分區時，我們希望將在目標狀態下彼此接近的圖塊組合在一起，因為這些圖塊彼此之間相互作用最大。

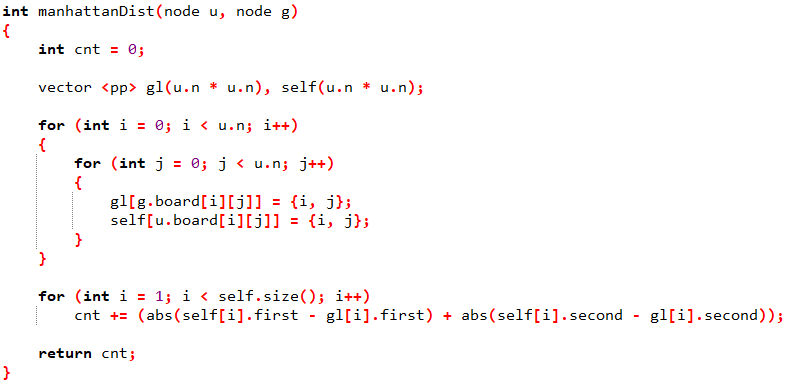
舉例:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 5 |
| 3 |  | 6 |
| 7 | 4 | 8 |

在這裡我們可以看到圖塊5、3、4、8放錯了位置，因此我們計算出每個圖塊的m距離分別為2、3、2、1，因此該狀態的曼哈頓啟發式值為8(h=8)。

下圖程式碼Manhattan Distance Calculation

兩點之間的曼哈頓距離是使用一個簡單的公式計算的--**Mdist = |x2-x1|+|y2-y1|**

****

**Linear+Manhattan:**

如果兩個圖塊'a'和'b'在同一行或同一列中，並且它們的目標位置在同一行或同一列中，並且其中一個圖塊的目標位置被另一個圖塊阻擋，則它們之間在那一行(列)存在線性衝突。

而線性衝突啟發式是曼哈頓距離的一個顯著改進，它適用於目標行或列中的切片，但相對於彼此反轉。例如，假設给定狀態的第一行按順序包含圖塊（2 1），但在目標狀態下它們按順序出現（1 2）。要反轉它們，其中一個切片必須移出第一行，以允許另一個切片通過，然後移回第一行。由於這兩個移動不計入任一區塊的曼哈頓距離，因此可以將兩個移動添加到這兩個區塊的曼哈頓距離的總和而不違反可接受性。

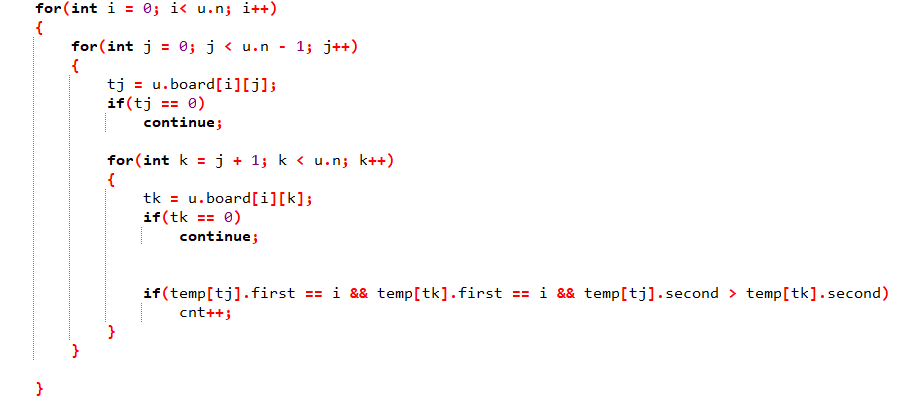
舉例:

|  |  |  |
| --- | --- | --- |
| 4 | 2 | 5 |
| 1 |  | 6 |
| 3 | 8 | 7 |

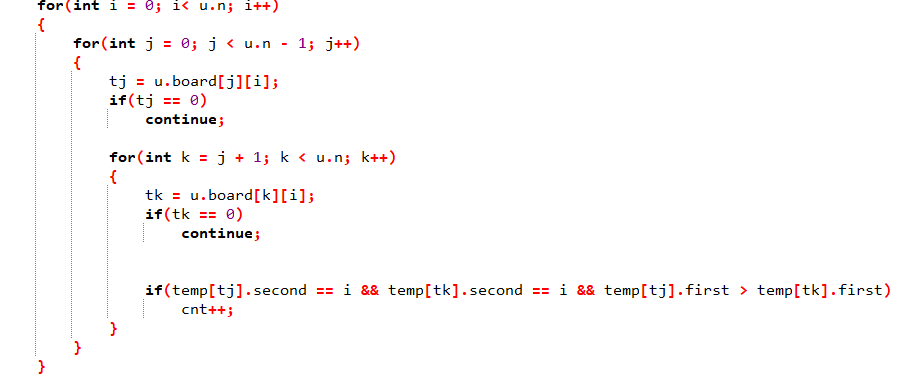
在這種情況下，我們看到圖塊4和圖塊1處於線性衝突中，因為我們看到圖塊4在同一列中位於圖塊1的目標位置的路徑中，反之亦然，圖塊8和圖塊7也在同一列中線性衝突，因為8站在同一行中圖塊7的目標位置的路徑中。因此，在這裡我們看到有2個線性衝突。

如我們所知，啟發式值是給出解決該問題所需的移動次數的理論最小值的值，我們可以看到一個線性衝突導致將兩個移動添加到最終的啟發式值（h），因為一個圖塊將必須移動到一邊，以便讓目標狀態位於已移動圖塊後面的圖塊騰出空間，然後向後移動兩次，從而保留啟發式的可接納性。

(下圖程式碼為檢查同一行tj和tk是否目標位置，然後tj是否在tk的右側)



(下圖程式碼為檢查同一列tj和tk是否目標位置，然後tj是否在tk的下側)

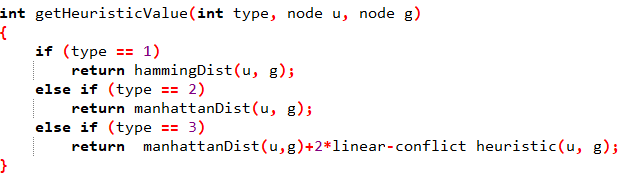


線性衝突總是與曼哈頓距離相結合以獲得該狀態的啟發式值，並且每個線性衝突都會向曼哈頓距離增加2次移動，如上所述，因此上述狀態的“ h”值為

曼哈頓距離+ 2 \*線性衝突數，而該狀態的曼哈頓距離是：10

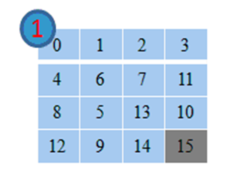
最後得h：10 + 2 \* 2 = 14

(下圖程式碼是在講解計算“ h”值)

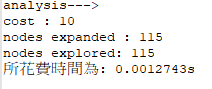


**實驗結果:**

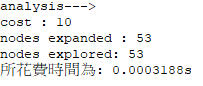
**第一題:**

****

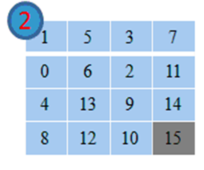
**Linear+ Manhattan:**

****

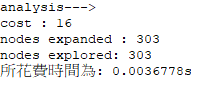
**Manhattan:**

****

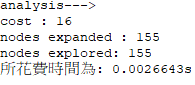
**第二題:**

****

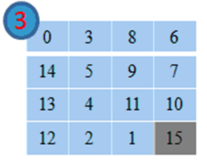
**Linear+Manhattan:**

****

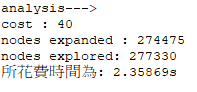
**Manhattan:**

****

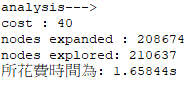
**第三題:**

****

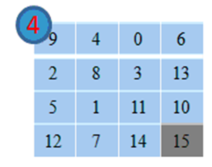
**Linear+ Manhattan:**

****

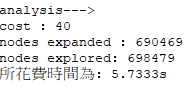
**Manhattan:**

****

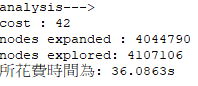
**第四題:**

****

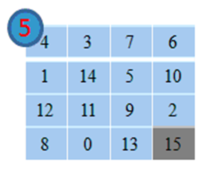
**Linear+ Manhattan:**

****

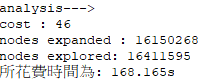
**Manhattan:**

****

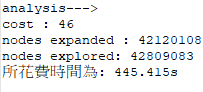
**第五題:**

****

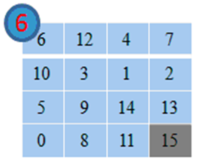
**Linear+ Manhattan:**

****

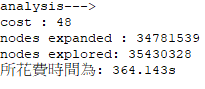
**Manhattan:**

****

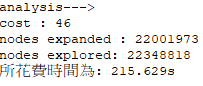
**第六題:**

****

**Linear+ Manhattan:**

****

**Manhattan:**

****

**結果討論:**

我們嘗試用了Manhattan和Linear + Manhattan啟發式算法來實現A\*演算法。第一題到第三題皆為Manhattan較快，第四 、五題則為Linear+ Manhattan較快，第六題卻是Manhattan較快，原本預測第六題應該也為Linear+ Manhattan比較快，但結果卻是Manhattan距離效果更好，不太明白為甚麼!我們程式算法中，Linear+ Manhattan擴展了更多的節點，但它的結果並不是最優的。之後我們便在思考假如答案不是最佳的，很可能問題是出在演算法，可能有時候會高估到圖中一個節點的成本。Linear + Manhattan 啟發式方法應該始終於和Manhattan之類的距離估計啟發式方法結合使用，它不像在每一行/每列中給出兩倍Linear 數那樣簡單。

**作業心得:**

蔡雅如:

這次的作業和平行程式設計第十章的內容很像，剛好最近教到，不過平行程式的方法是用shearsort等等的作法，和人工智慧所使用的A\*演算法不同，雖然上到相同內容的時候很興奮，但後來發現做法不同，還是得花點時間研究。這次最主要認識了Manhattan distance 和 Linear Conflict + Manhattan distance兩個算法，因為是以前沒遇過的東西，所以花了不少時間。最重要的是組員都很厲害!!

郭宇芹:

這次project做了讓程式跑智慧拼圖，題目規定的A\*演算法在第二次小考是以game tree的形式呈現，那時候就查到了些關於曼哈頓距離的資料，也看到了有關拼圖遊戲的程式內容，但是組員在撰寫程式的時候才知道了還有線性關係的方法，雖然我的電腦根本跑不動這個程式碼，因為他對於硬體有一定的規格要求。

這種讓電腦自己跑遊戲的程式其實是第一次接觸，跟現在畢業專題在做的AI又有一定的差異，不需要訓練、label之類的，也沒有資料庫需要建立，不過如果能預先設定好15puzzle的全部情況並且存進資料庫，那也是很酷的一件事！

蔡宛凌:

在做這次作業之前，我又去複習了一下之前上課所教過的智慧拼盤跟A\*演算法，然後以A\*演算法為基底，再去推演理解了Manhattan和Linear + Manhattan啟發式演算法兩種演算法的原理，開始實作時，再去網路上查了一些函式來改改看，跟組員一起debug，最後看到第六題是曼哈頓比較快的時候有點疑惑，後來覺得有可能是演算法高估成本的問題。這次作業也算是讓我更了解Manhattan和Linear + Manhattan這兩種演算法在做什麼，有點收穫。

林仲岳:

這次做智慧拼盤遇到的問題主要就是測試時間比較多，有時候會有bug記憶體會耗到爆掉，因為一直無限展開，如果遇到這個問題解決方法就是設定overflow的值，讓它不會無限展開。其他的bug都還好，大概修一修就解決了，但最後的結果第六題Linear + Manhattan演算法比較慢這點比較奇怪一點啦，因為照理來說應該是要Linear + Manhattan比較快才對，我覺得問題可能是linear演算法程式還需要修改。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 蔡雅如 | 郭宇芹 | 蔡宛凌 | 林仲岳 |
| 工作分配 | 程式修改，查詢資料 | 程式修改，查詢資料 | 程式修改，查詢資料 | 程式撰寫，查詢資料，debug |
| 貢獻比例 | 23% | 23% | 23% | 31% |

**參考資料:**

<https://stackoverflow.com/questions/35552661/can-linear-conflict-heuristic-cause-more-nodes-to-be-created-and-explored-than-m>

<https://wood-du.cn/2019/04/23/N-puzzle-Problem/>

<http://blog.minstrel.idv.tw/2004/12/star-algorithm.html>

<https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle/>

<https://www.codespeedy.com/compute-manhattan-distance-between-two-points-in-cpp/>

<https://codereview.stackexchange.com/questions/205935/npuzzle-solver-using-a-with-manhattan-linear-conflict-updated-code>