**人工智慧作業一 報告**

林佳穎 資工105級 40147026S

電話：0916518986

**一、開發機器軟硬體規格**

筆記型電腦型號：Acer Aspire S3

處理器：Inter(R) Core(TM) i5-2467M CPU @ 1.60GHz 1.60GHz

RAM：4.00GB(3.85 GB可用)

作業系統：Windows7 家用進階版 64位元作業系統

開發軟體版本：Code::Blocks 12.11

使用原因：這台筆電很好攜帶、系統操作順暢、CodeBlocks是慣用寫程式軟體

**二、實作方法**

**1.方法一**

演算法：BFS

資料結構：queue<queue<int>>

使用語言：C++

技術與操練要項：

內層queue用來存放砲的位置、外層queue用來實作BFS

while(還有盤面)

{

Pop外層queue最前面的queue<int>，將所有砲的位置依序pop出來，將盤面translate到目前的位置

分析目前砲能不能往下、右、左、上跳(用這個順序是因為，往下往右應該會較接近將，所以先檢查，因為queue為FIFO所以先存入的盤面會先展開)

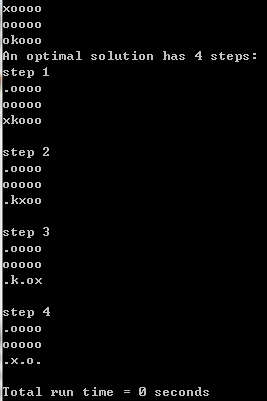
如果可以跳，檢查”砲”跳了之後位置是不是跟”將”一樣，如果一樣就是找到解答，跳出迴圈。如果不一樣就push進去queue<int>再push進去外層queue下一輪再展開。

}

如果有找到就印解答；沒找到就印No solution.

範例測資結果：3 5 3 2





BFS各種盤面測試結果分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 盤面大小 | 將的位置 | 是否跑出解 | 最少步數 | 耗用時間(秒) | 耗用空間(秒) |
| 4 4 | 3 2 | 是 | No solution | 0 | 616K |
| 3 3 | 是 | 2 | 0 | 620K |
| 10 10 | 3 2 | 是 | 4 | 0 | 784K |
| 5 7 | 是 | 5 | 0 | 800K |
| 7 8 | 是 | 8 | 0.008 | 1,008K |
| 10 10 | 是 | 12 | 0.809 | 3,432K |
| 30 30 | 3 2 | 是 | 4 | 0 | 780K |
| 7 8 | 是 | 8 | 0.012 | 1,224K |
| 10 10 | 是 | 12 | 1.173 | 3,656K |
| 15 15 | 是 | 14 | 19.665 | 13,072K(最大到1,452,484K) |
| 20 20 | 否(爆掉) |  | 38.706 | 2,056,784K |
| 50 50 | 3 2 | 是 | 4 | 0.001 | 792K |
| 7 8 | 是 | 8 | 0.018 | 1,232K |
| 10 10 | 是 | 12 | 2.158 | 3,472K(最大到120,168K) |
| 15 15 | 是 | 14 | 27.356 | 13,084K(最大到1,506,416K) |
| 20 20 | 否(爆掉) |  | 45.966 | 2,065,988K |

**2.方法二**

演算法：IDS

資料結構：stack<queue<int>>

使用語言：C++

技術與操練要項：

內層queue用來存放砲的位置、外層stack用來實作IDS

For(深度從2到n\*m-1)

{

每輪都從初始盤面開始展開

while(還有盤面)

{

Pop外層stack最上面的queue<int>

If(是否到目前限制深度)已經到了限制深度就不再展開。

Else{

還沒到了限制深度，將所有砲的位置依序pop出來，將盤面translate到目前的位置

分析目前砲能不能往上、左、右、下跳(用這個順序是因為，往下往右應該會較接近將，所以後檢查，因為stack為FILO所以後存入的盤面會先展開)

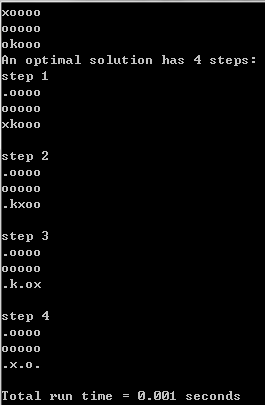
如果可以跳，檢查”砲”跳了之後位置是不是跟”將”一樣，如果一樣就是找到解答，跳出for迴圈。如果不一樣就push進去queue<int>再push進外層stack展開。

}

}

}如果有找到就印解答；沒找到就印No solution.

範例測資結果：3 5 3 2



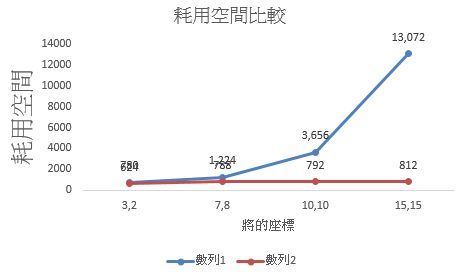
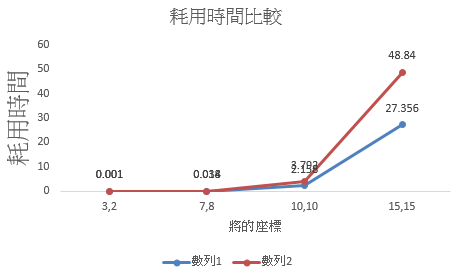
IDS各種盤面測試結果分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 盤面大小 | 將的位置 | 是否跑出解 | 最少步數 | 耗用時間(秒) | 耗用空間(秒) |
| 4 4 | 3 2 | 是 | No solution | 0.003 | 616K |
| 3 3 | 是 | 2 | 0 | 628K |
| 10 10 | 3 2 | 是 | 4 | 0 | 628K |
| 5 7 | 是 | 5 | 0.001 | 624K |
| 7 8 | 是 | 8 | 0.017 | 784K |
| 10 10 | 是 | 12 | 1.638 | 792K |
| 30 30 | 3 2 | 是 | 4 | 0.001 | 620K |
| 7 8 | 是 | 8 | 0.023 | 784K |
| 10 10 | 是 | 12 | 3.083 | 796K |
| 15 15 | 是 | 14 | 38.267 | 812K |
| 20 20 | 否(逾時) |  | 3600以上 |  |
| 50 50 | 3 2 | 是 | 4 | 0.001 | 624K |
| 7 8 | 是 | 8 | 0.034 | 788K |
| 10 10 | 是 | 12 | 3.792 | 792K |
| 15 15 | 是 | 14 | 48.84 | 812K |
| 20 20 | 否((逾時) |  | 3600以上 |  |

圖表分析：

根據資料數據，兩種方法皆取50x50的盤面資料來做比較

數列一(藍色)為BFS，數列二(紅色)為IDS



由上列圖表可發現，在耗用時間表現兩種方法皆為指數增長，雖然IDS時間增長較BFS快速，但在耗用空間上IDS的表現就大大的優於BFS。

**三、開發狀況與困難：**

此次作業實作的兩種方法皆為Tree search版本，因此有許多浪費在檢查重覆盤面。原本想增加避免展開重覆盤面的檢查，做成Graph search的版本，但是貯存所有盤面情形就會耗用很多空間，檢查盤面也會耗用很多時間，因此後來還是決定使用Tree search的版本。

此次作業實作遇到的困難就是，BFS在紀錄耗用空間時，發現在大盤面搜索時耗用空間會暴漲非常快速，一旦發現解又會迅速消退。猜測在小盤面也有這樣的情形，只是程式執行時間太短記錄上有困難。

**四、參考資料：**

C++ STL--stack/queue的使用方法：

<http://blog.csdn.net/j_dark/article/details/8887294>

C++ 計時方式

<http://annheilong.pixnet.net/blog/post/23852466-%E3%80%90%E9%9B%BB%E8%85%A6%E3%80%91c%2B%2B-%E8%A8%88%E6%99%82%E6%96%B9%E5%BC%8F>

在散布圖或折線圖顯示資料

<https://support.office.com/zh-tw/article/%E5%9C%A8%E6%95%A3%E5%B8%83%E5%9C%96%E6%88%96%E6%8A%98%E7%B7%9A%E5%9C%96%E9%A1%AF%E7%A4%BA%E8%B3%87%E6%96%99-4570a80f-599a-4d6b-a155-104a9018b86e?ui=zh-TW&rs=zh-TW&ad=TW>