

智慧拼盤

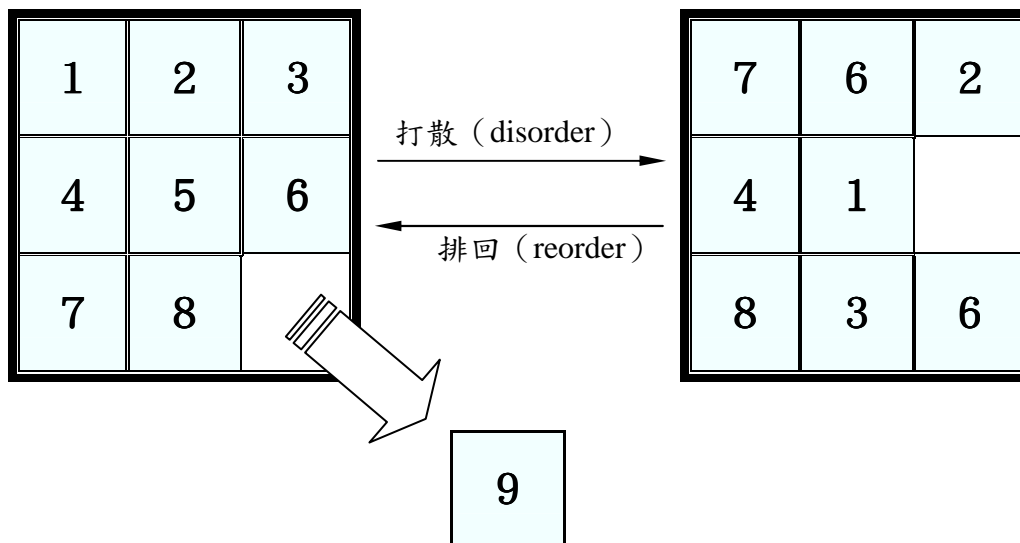
前言.....	1
問題界定.....	1
分析.....	2
需求 (Requirement)	2
智慧拼盤的表示法 (Presentation of Puzzle)	2
解題規則的表示法 (Presentation of Rules)	3
介面 (Interface)	3
設計.....	3
模組定義 (Definition of Modules)	3
樣式 (Patterns of blocks)	5
規則 (Rules to solve the Puzzle game)	6
檢討與策進.....	7

前言

- 這是在我於就讀銘傳資管系二年級時，選修的“專家系統”課堂中，任課老師曾憲宏所給定的題目。
- 當初對於專家系統的 tool 語言——CLIPS，感到滿新鮮的，所以就把這個問題的核心往自己身上攬，而將使用者介面讓其他組員去發揮。
- CLIPS 是 C Language Integrated Production System 的縮寫，由美國 NASA 太空中心的一個人工智慧部門所發展，就發展專家系統而言，是個很有用的工具。

問題界定

- 這次的題目：“智慧拼盤” (puzzle)，是一個 $n \times n$ 的格狀棋盤 (grid board) 遊戲。棋盤上的每一格都有一個正方形的積木 (block)，每個積木都有自己正確順序 (order) 的位置，為了區別，依序編上 1 到 $n \times n$ 的編號。
- 遊戲啟始時先將其中一格積木取走，造成的空缺 (blank)，使相鄰的積木可以移動至空缺上，並編上“0”。
- 如此，利用這個空缺，可以將棋盤上原先就定位的積木之順序打散 (disorder)。
- 玩遊戲的一方 (player) 所要完成的目標 (goal) 就是要將積木利用空格 (編號為 0)，排回 (reorder) 原先的位置。
- 這個題目的目的是要設計一個會玩智慧拼盤的專家系統，所以一開始由人將積木的順序打散，再要求電腦將其排回正確的位置。



分析

需求 (Requirement)

- 為了達成更好的展示效果，於是選定了圖形化的使用者介面。
- 又為了簡化問題，於是先從3個階層著手，也就是3×3的智慧拼盤遊戲，作為測試規則的用途。
- 不過，為了將來能夠將系統套用於任何階層，所以規則制定時都很一般化，原則上這些規則要能在任何階層的智慧拼盤下運轉，至於能否在任何階層都求出解答，不是目前所要求的。
- 這個軟體的開發平台是在MS-DOS下的CLIPS v6.0上建立puzzle規則，並利用MS Windows 3.1下的MS Visual Basic設計使用者介面。

智慧拼盤的表示法 (Presentation of Puzzle)

- 代號說明如下：
 - “F”：表示該位置被凍結 (Frozen) 了。被凍結 (freeze) 的位置是不可變動的：一個積木不可以移到被凍結的位置，位於被凍結位置的積木也不能再移動；除非 unfreeze 的規則被啟動，來解除其 frozen 狀態。
 - “0”：表示該位置沒有積木 (block)，而形成一個空缺 (blank)。
 - “S”：Source 的縮寫，表示運作中 (on focus) 的 block 現在的位置。
 - “T”：代表 source 上的 block 欲前往的目標 (Target) 位置。也就是這個 source 積木在所有積木排好之後的正確位置。

- 以 3×3 的智慧拼盤為例，其啟始時的狀態可能如下：

	0	1	2	3	4
0	F	F	F	F	F
1	F	T			F
2	F		S		F
3	F			0	F
4	F	F	F	F	F

- 左圖中 F 的部分為邊界，故要標上 frozen，表示該位置不可能有 block 異動。
- 座標 (1,1) 標上 T、(2,2) 標上 S、(3,3) 標上 0，表示現在的 target block 之序號為 1，而其位於 (2,2) 上，我們要利用 (3,3) 上的 blank 將 block 1 由 (2,2) 移至 (1,1)

解題規則的表示法 (Presentation of Rules)

- 首先找出各種特殊的積木配置樣式 (patterns)，這些樣式都是我在玩智慧拼盤的過程中發現的。找到樣式後就用上述的表示法，標出樣式的型態。
- 在標示出樣式後，根據每一個樣式去撰寫移動到正確位置的規則。

介面 (Interface)

- CLIPS 部分的 I/O 介面設計，只用到下列兩個檔案：
 - 一個唯讀檔：puzzle.in，用來設定積木位置的“起始”值。
 - 一個寫入檔：puzzle.out，用來輸出整個執行的“過程”。
- 這樣設計的考量是為了，使用其他工具來設計它的使用者介面。這個軟體是使用 Visual Basic 來設計使用者介面：
 - 程式開始時在 VB 撰寫的使用者介面上，將智慧拼盤的 block 的順序打散，並將打散後的順序輸出至“puzzle.in”後，再以系統呼叫的方式執行 CLIPS 直譯器上的“puzzle.clp”程式
 - puzzle.clp 會將執行的過程輸出至“puzzle.out”。
 - 最後，VB 再讀取“puzzle.out”的內容，並將結果以動態的圖示呈現出來。

設計

- 下面幾節，就解 Puzzle 問題時，所設計的模組及使用了哪些規則作一個說明：

模組定義 (Definition of Modules)

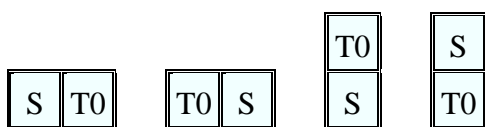
➤ 為了方便程式撰寫的管理，於是利用 CLIPS 的 “defmodule” 指令定義了九個模組（module），分別說明如下：

- MAIN：定義了一些啟始時的 fact，以及 block 的結構定義。
- FrozenCoord：定義一些將某個座標凍結（freeze）及解凍（unfreeze）的功能。
- I/O：從 puzzle.in 輸入啟始值，並將每一步積木移動的過程輸出至 puzzle.out。
- LowLevelShift、MidLevelShift、HiLevelShift，以及 Shift：定義一些 block 移動有關的機制。只是為了供最上層的 Puzzle Rules 容易使用，必須層層地封裝成更容易使用的模組。
- ChangeTarget：切換現在想要排列的 target 積木。
- Start：作一些啟始的動作。
- PuzzleRules：是最主要的部分，所有和解智慧拼盤的規則都在這裡定義。

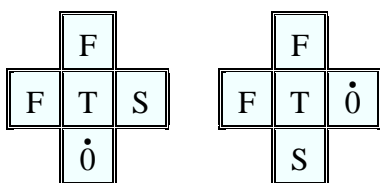
樣式 (Patterns of blocks)

- 總共使用了七類樣式，其中的每格的狀態符號所代表的意義如前面所敘述的一樣，另外在一些符號上加上批點代表“isn't”：如 $\dot{0}$ ，表示 isn't blank。
- 各種樣式表示如下：

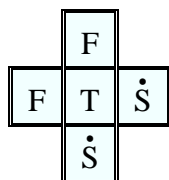
➤ Pattern 1 :



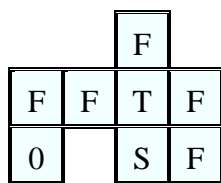
➤ Pattern 2 :



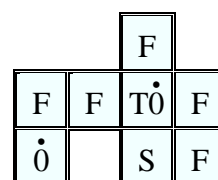
➤ Pattern 3 :



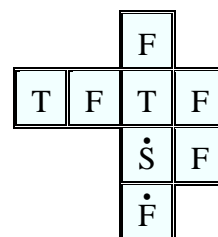
➤ Pattern 4 :



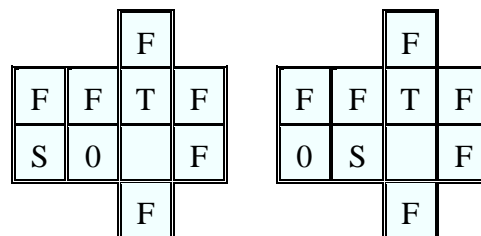
➤ Pattern 5 :



➤ Pattern 6 :



➤ Pattern 7 :



規則 (Rules to solve the Puzzle game)

➤ 根據上面的 patterns，可以分別撰寫下面七個 rules：

➤ rule 1：

```
if (pattern 1)
then (direct shift)
```

➤ rule 2：

```
if (pattern 2)
then (convert into pattern 1)
```

➤ rule 3：

```
if (pattern 3) and (not pattern 7)
then (convert into pattern 2)
```

➤ rule 4：

```
if (pattern 4)
then (do some conversion)
```

➤ rule 5：

```
if (pattern 5)
then (convert into pattern 4)
```

➤ rule 6：

```
if (pattern 6)
then (convert into pattern 5)
```

➤ rule 7：

```
if (pattern 7)
then (do some conversion)
```

檢討與策進

- 雖然這個程式是以 3×3 的智慧拼盤為例，但由於分析階段即考慮了規則表示法的一般性，在設計階段對封裝也非常重視，故要擴充成更高階的 puzzle 或有新的 rules 要加入時，都很方便。
- 在程式撰寫的過程中，深覺得若能使用 function 來將較“低階”的程式包裝起來，將可大大地降低 coding 的複雜度。
- 此外，CLIPS 語言若能提供“named constant”，將可使程式更富彈性。
- 據說 CLIPS 有支援 object-oriented 的語法，真的很想嚐試看看，可惜沒機會接觸相關的資料。
- 整個程式的架構是使用 top-down 的方式建構起來的，在可讀性，及日後的維護方面都較容易。
- 據任課老師的評語，我這個程式的 rules 是以 data-driven 的方式撰寫，較偏 forward (bottom-up) 的方式；照老師的看法，這個題目較適合利用 goal-driven，也就是 backward 的方式來解。
- 當初我是想由底層制定一些普遍性的 rules，使整個系統能在沒有很明確指示下，就能解決問題。結果程式雖然能跑，不過卻可能付出了較大的代價，所幸從中也學到了許多東西。