**人工智慧期末報告**

**Sokoban推箱子AI設計**

組長: 黃泰源 組員: 林育民、洪嘉桓

1. 摘要

「推箱子」是款經典的遊戲，玩家必須推動箱子至目標地點來達成目標。此次專題我們主要是設計一個AI來解決推箱子的關卡。

1. 動機

在人工智慧課堂上我們學到了許多以搜索求解的演算法，於是我們想找一個適合的題目來應用這些演算法。

「推箱子」這款遊戲主要目的是希望搬貨工人能將貨物順利推到目的地，將貨物推到目的地的這種行為應該是能用搜尋尋找路徑。

1. 環境介紹與分析

推箱子(如圖1)主要是由箱子、目的地、搬運工人、牆壁所構成的遊戲，牆壁會將地圖包圍起來，會有幾個與箱子數量相符的目的地以及一位搬運工人。搬運工人必須把所有的箱子推到目的地才算過關(如圖2)，要注意的是工人一次只能推一個箱子，工人並無法一次推動兩個以上的箱子。

在推箱子遊戲途中，玩家常常會不小心將箱子推到死結(如圖3紅叉部分)處導致不可挽回的錯誤；箱子在行進中有可能會受到其他箱子的阻擋而無法前進導致無法順利到達目的地。

1. 方法設計

推箱子如果不考慮路徑是不是最佳的話，可以將它視為CSP問題來解決，我們並不考慮路徑是不是最佳，我們只在乎所有箱子能不能到達目的地。

根據上面的敘述，我們將搜索方式設計為深度優先搜索，但在其中會加入一些heuristic function和一些箱子路徑上的判斷來決定哪個分支先做，所以整個搜索的方式會比較像hill climbing。

推箱子遊戲在目的地的選擇與箱子進入的先後次序至關重要，往往影響了整個地圖是否有解。在目的地的判斷(圖4)上我們以目的地周圍上下左右計算牆壁或箱子的個數來做為放入的先後次序，演算法的部分見虛擬碼1；在箱子的判斷(圖5)上，我們先選擇預放入的目的地，並使用A\*路徑來判斷路徑上的其他箱子有多少個，來決定哪一個箱子先放入該目的地，演算法的部分見虛擬碼2。

在箱子的移動中主要是搬運工人要配合箱子的移動，要到箱子預移動方向的反方向，這樣才符合工人推箱子的遊戲規則。箱子在選擇下一個位置時必須判斷下一個位置是否可以走，像是死結或是有其他箱子都不可以走；而箱子想移動的相反位置則要讓搬運工人能到達，像是牆壁或是有其他的箱子都不可以到達，判斷的示意圖如圖6。

由於我們希望能以搜索的方式求解，所以我們將搬運工人移動與箱子移動分成兩個函式並且遞迴呼叫，詳細的演算法在虛擬碼3與虛擬碼4。

1. 結論

在本次實作中不管是箱子移動或是搬運工人移動都會將狀態儲存，這無疑是會占用大量記憶體空間，雖然現今電腦記憶體都很大，但在問題的難度增加時，多少也會造成電腦的負擔。

由於為了實作方便boxMove函式與agentMove函式皆使用深度優先搜尋來實作，導致搬運工人在移動時並不是最佳解。在之後程式修改時可以考慮將agentMove的方法改成A\* search。

在搜索樹展開時，有些枝節已經很清楚沒有解，但還是會將它尋訪過一次；在之後的修改上可以想辦法把這些狀態prune掉，以減少搜索空間。

在沒有路到指定的box旁邊時，搬運工人會毫無意義的將圖尋訪過一遍，這樣會浪費很多時間，應該用一些方法提前停止搬運工人的搜索。

在本此實作中搬運工人並不會主動去搬移其他的箱子，之後的改進可以考慮讓搬運工人能主動搬移形成阻礙的箱子。

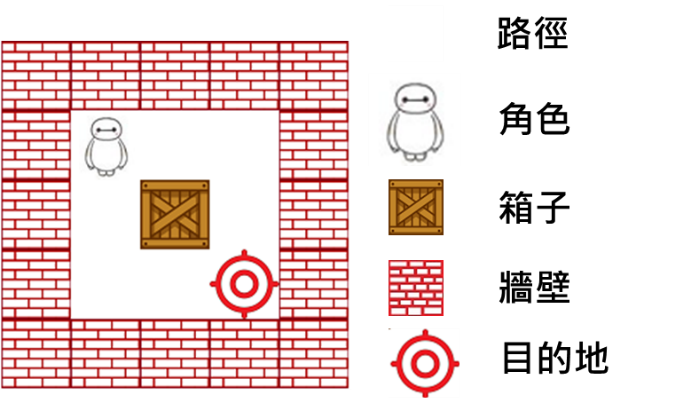


圖1、推箱子的環境

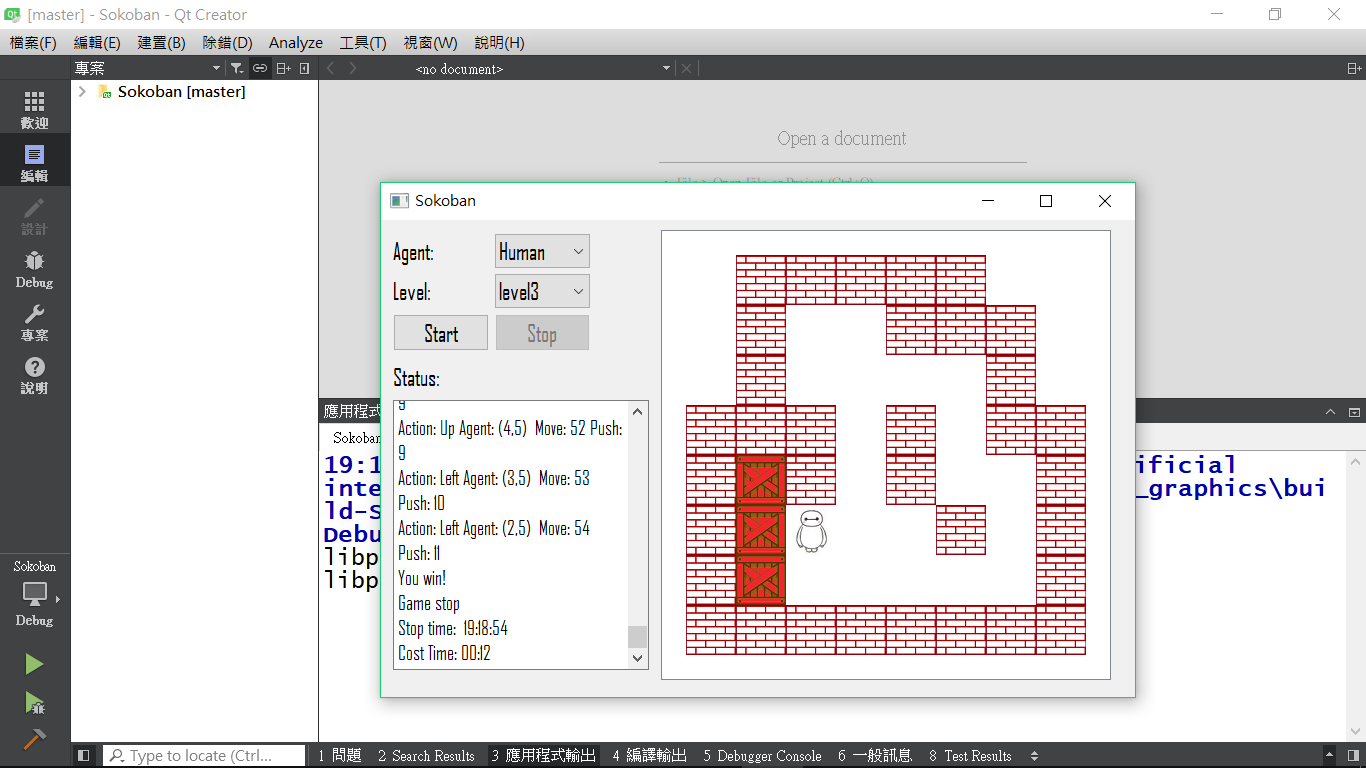
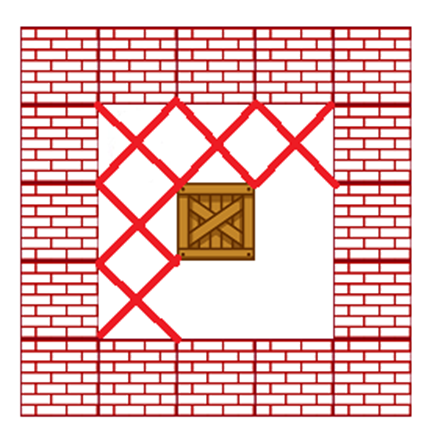
 

圖2、通過關卡示意圖 圖3、死結示意圖

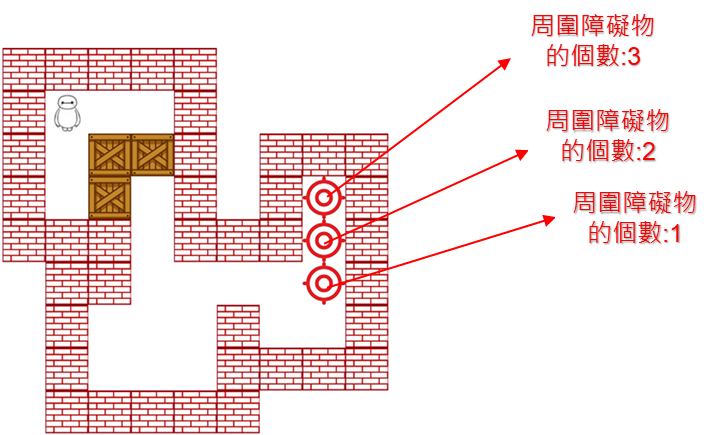


圖4、目的地判斷示意圖

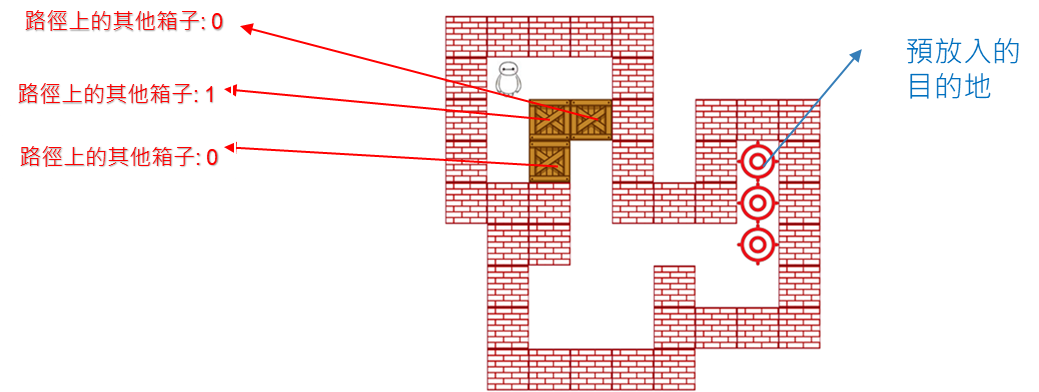


圖5、箱子判斷示意圖

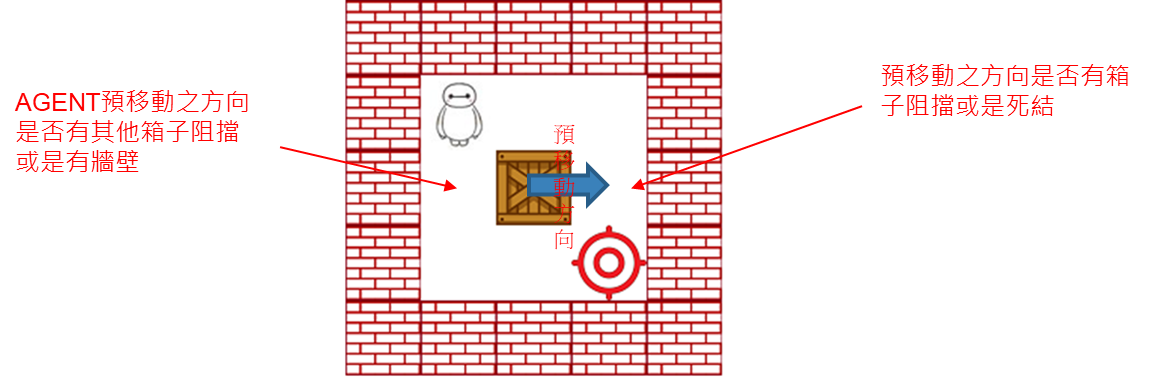


圖6、箱子移動判斷示意圖

Algorithm EvaluateTarget

Input: A set A of targets, boxes position, and a maze of sokoban game

Output: Barrier number for each targets

Step1: Pick up one target in set A, and erase the target from set A.

Step2: Calculate number of barrier ( such as box or wall ) surround the target.

Step3: If set A is empty terminate and return the result, otherwise go to Step1.

虛擬碼1、目的地的判斷

Algorithm EvaluateBox

Input: A target, a set B of boxes, and a maze of sokoban game

Output: Barrier number for each boxes

Step1: Pick up one box in set B, and erase the box from set B.

Step2: Get the path from the box to the target by A\* algorithm.

Step3: Calculate number of the other boxes on the path from the box to the target.

Step4: If set B is empty terminate and return the result, otherwise go to Step1.

虛擬碼2、箱子的判斷

**function** boxMove( a set A of targets, a set B of boxes, agentPos, bool plan, state agentWantState , set agentExplored ) **return** true or false ( true means the branch find solution, false doesn’t )

**if** plan is true **then**

EvaluateTarget(A, B) and **sort** targets according to its barriers( descending )

**clear** B.explored

**clear** agentExplored set

**for** each target t in set A **do**

**if** plan is true **then**

EvaluateBox( t, B ) and **sort** boxes according to its barriers( ascending )

**for** each box in B **do**

**insert** box.position **into** box.explored

**declare** a list of state: possibleState

**for** each action in ACTION **do**

nextState.action = action

nextState.boxWantPosition = box.position + action

nextState.agentWantPosition = box.position - action

**if (** nextState.boxWantPosition has no other boxes or deadlock ) and

**(** nextState.agentWantPosition has no other boxes or wall ) and

nextState.boxWantPosition is not in box.explored **then**

**append** nextStateto possibleState

**sort** possibleState according to distance from possibleState to target( ascending )

**for** each state in possibleState **do**

nextState.position = box.position + action

**insert** nextState into box.explored

**if** agentTurn( A, B, agentPos, state, agentExplored ) **then**

**return** true

**else**

**return** false

**return** false

虛擬碼3、箱子的移動

**function** agentMove( a set A of targets, a set B of boxes, agentPos, bool plan, state agentWantState , set agentExplored ) **return** true or false ( true means the branch find solution, false doesn’t )

agentDst = agentWantState.boxWantPosition – action

**if** agentDst **equal to** agentPos **then**

**push** agentWantState.action **to** actionStack

**update** agentPos, B.position

**clear** agentExplored

bool match\_target = false

**if** agentWantState.boxWantPosition is in B.position **then**

**if** agentWantState.boxWantPosition is in A.position **then**

**erase** agentWantState.boxWantPosition **from** B

**erase** agentWantState.boxWantPosition **from** A

match\_target = true

bool has\_solution = false

**if** match\_target is true **then**

has\_solution = boxTurn( A, B, agentPos, true, agentWantState, agentExplored)

**else**

has\_solution = boxTurn( A, B, agentPos, false, agentWantState, agentExplored)

**if** has\_solution is true:

**return** true

**else**

**pop from** actionStack

**return** false

**else**

**declare** a list of state: possibleState

**insert** agentPos **into** agentExplored

**for** each action in ACTION **do**

nextState.agentCurrentPosition = agentPos + action

**if (** nextState has no other boxes or wall ) and

nextState.agentCurrentPosition is not in agentExplored **then**

**append** nextStateto possibleState

**if** possibleState is empty **then**

**return** false

**else**

**sort** possibleState according to distance from possibleState to agentWantState.agentWantPosition( ascending )

**for** each nextState in possibleState **do**

**push** nextState.action **into** actionStack

**if** agentTurn( A, B, nextState.agentCurrentPosition, agentWantState, agentExplored ) **then**

**return** true

**else**

**pop** action **from** actionStack

**return** false

虛擬碼4、搬運工人的移動