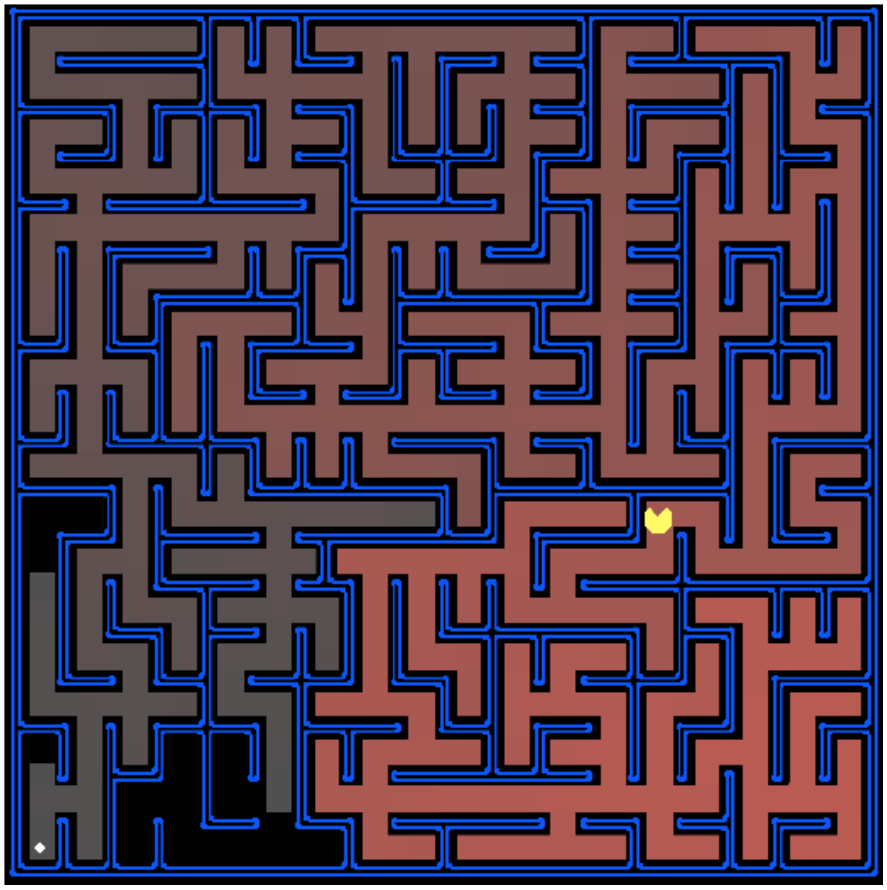
人工智慧專題 1 – 用搜尋法求解問題

目錄



專題說明

在本專題中，你的小精靈將在迷宮中找出既可以到達指定位置，也可以有效吃到食物的路徑。您要撰寫一個通用的搜尋演算法並將其應用在 Pacman 的各種情境。

本專題包含一個自動評分程式，供您對答案並自行評分，可用以下命令執行：

```
python autograder.py
```

本專題由多個 Python 檔案所組成，其中一些檔案需要閱讀和理解才能完成作業，其中一些可以忽略，所有程式碼細節請參閱附件 project1_search.zip：

需要編輯的檔案：

search.py	所有搜尋演算法。
searchAgents.py	所有基於搜尋的代理(已完成)。

可能需要檢視的檔案：

pacman.py	小精靈遊戲主檔，描述在此專題中所使用的 GameState 類別。
game.py	小精靈世界如何運作背後的邏輯，描述幾種支援類別，例如 AgentState、Agent、Direction 和 Grid。
util.py	用來實作搜尋演算法的資料結構。

可忽略的支援檔案：

graphicsDisplay.py	小精靈的圖形
graphicsUtils.py	支援小精靈圖形
textDisplay.py	小精靈的 ASCII 圖形
ghostAgents.py	控制幽靈的代理
keyboardAgents.py	控制小精靈的鍵盤介面
layout.py	讀取佈局檔和儲存其內容的程式碼
autograder.py	專題自動評分程式
testParser.py	解析自動評分測試和解答檔
testClasses.py	通用自動評分測試類別
test_cases/	每個問題的測試案例的目錄
searchTestClasses.py	專題 1 自動評分測試類別

要編輯與繳交的檔案：請繳交 search.py，不要變更也不要繳交其他檔案。

評分：您的程式將以 autograder 自動評分以確保技術上正確性。*請**不要**更改* autograder 程式中的任何函式或類別的名稱，否則將無法正確評分。但是，若有必要，我會單獨審查作業並對其進行評分，以確保您獲得應有的分數。

歡迎來到小精靈

下載程式碼(project1_search.zip)，解壓縮並切換到目錄後，應能在命令列中輸入以下指令來玩 Pacman 遊戲：

```
python pacman.py
```

小精靈生活在一個閃亮的藍色世界中，有著扭曲的走道和美味的圓點。小精靈要掌握其領域的第一步就是要能夠有效地暢遊這個世界。

searchAgents.py 中最簡單的代理是一個只會向西移動的 GoWestAgent（一個簡單的反射代理）。此代理偶爾會贏：

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

但是，當需要轉彎時，這個代理可能會變得很糟：

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

如果小精靈卡住了，可以在終端機視窗中輸入 CTRL-c 來退出遊戲。

pacman.py 有許多選項，可透過以下指令檢視所有選項及其預設值：

```
python pacman.py -h
```

此外，本專題中的所有命令也存放在 commands.txt，以便複製貼上。

問題 1（3 分）：使用深度優先搜尋找到固定的食物

在 searchAgents.py 中，有一個已實作的 SearchAgent 規劃了一條穿越 Pacman 世界的路徑，然後逐步執行該路徑。用來規劃路徑的搜尋演算法尚未實作 - 留待您來完成。

首先，執行以下指令來測試 SearchAgent 是否正常：

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

以上指令告訴 SearchAgent 使用 tinyMazeSearch 作為其搜尋演算法，該演算法在 search.py 中實作。小精靈應該成功地穿越迷宮。

您要撰寫的搜尋演算法的虛擬碼可以在講義和課本中找到。請記住，搜尋節點不僅必須包含狀態，還必須包含重建到達該狀態的路徑所需的資訊。

重要說明：*所有搜尋功能都需要傳回一個動作清單(路徑)，這些動作(actions)將帶領 agent 從初始位置達到目標位置，這些動作必須是合法的移動（有效的方向，不能穿牆）。*

重要提示：*請確認有**使用** util.py 中提供的 Stack, Queue 和 PriorityQueue 佇列資料結構！這些資料結構實作了 autograder 所需的特定屬性。*

提示：*每個演算法都非常相似。DFS、BFS、UCS 和 A* 演算法僅在如何管理前緣的細節上有所不同。因此，先專注於讓 DFS 正確執行，其餘的應該相對簡單。實際上，一種可能的實作方法是只需要寫一個通用搜尋函式，再將特定演算法的策略當作參數傳給該函式來展開前緣。（不需要以此形式實作亦可獲得滿分）。*

實作 search.py 中的 depthFirstSearch （DFS） 演算法，您的程式應快速找到以下解答：

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

Pacman 將顯示其探索狀態，以及它們被探索(explored)的順序（較亮的紅色表示較早的探索）。探索順序是否符合您的預期？

*提示：*如果用 Stack 作為資料結構，則 DFS 演算法為 mediumMaze 找到的解決方案的長度應為 130（前提是您按照 getSuccessors 提供的順序將後續任務 push 到前緣;如果按相反的順序 push 它們，則可能會得到 246）。這是成本最低的解決方案嗎？如果不是，請想想深度優先搜尋做錯了什麼。

問題 2（3 分）：廣度優先搜尋

實作 search.py 中的 breadthFirstSearch（BFS） 演算法，並以與 DFS 相同的方式測試程式碼。

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

BFS 是否找到了成本最低的解決方案？如果不是，請檢查您的實作。

*提示：*如果小精靈對你來說移動得太慢，可使用選項--frameTime 0.

*注意：*如果您已經撰寫了通用的搜尋程式碼，那麼您的程式碼應該無需進行任何更改，即可適用於求解 8-puzzle 搜尋問題：

```
python eightpuzzle.py
```

問題 3（3 分）：改變成本函數

雖然 BFS 會找到一條以最少動作抵達目標的路徑，但我們可能希望找到在其他意義上「最佳」的路徑。考慮 mediumDottedMaze 和 mediumScaryMaze.

改變成本函數可以鼓勵小精靈找到不同的路徑。例如，我們可以在幽靈較多的地區對危險每走一步計入較多成本，或者在食物豐富的地區對每走一步計入較少的成本，而理性的小精靈應該調整其行為作為回應。

在 search.py 的 uniformCostSearch 函式中實作 UCS 搜尋演算法。現在您應該在以下三個佈局中成功抵達終點，以下代理都是 UCS 代理，僅在它們使用的成本函數上有所不同（代理和成本函數已幫您寫好）：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

問題 4（3 分）： A* 搜尋

實作 aStarSearch 圖形搜尋，將啟發式函數作為參數。啟發式方法有兩個參數：搜尋問題中的狀態（主參數）和問題本身（用來參考資訊）。 search.py 中的 nullHeuristic 啟發式函數就是一個簡單的例子。

您可以用曼哈頓距離(manhattenHeuristic)啟發式方法(已在 searchAgents.py 的 manhattenHeuristic 中實作）在原始問題中測試您的 A* 實作，即透過迷宮找到一條路徑到達指定位置。

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
```

您應該看到，A* 找到的最佳解比 UCS 稍微快一點（在我們的實作中展開了大約 549 個搜尋節點，但優先順序的關聯可能會使您的數字略有不同）。在 openMaze 上，各種搜尋策略的表現如何？

問題 5（3 分）：找到所有角落的食物

A*的真正威力只有在更具挑戰性的搜尋問題中才能顯現出來。現在，您將面對一個新的問題並為其設計啟發式方法。

*在角落迷宮(corner maze)*問題中，有四個圓點，每個角落各一個。新的搜尋問題是找到穿越迷宮並觸及所有四個角落的最短路徑(無論迷宮是否真的有食物)。請注意，對於 tinyCorners 迷宮，最短路徑並不一定是最先到達最近的食物！*提示*：通過 tinyCorners 的最短路徑需要 28 步。

注意：在處理問題 5 之前，請確認已完成問題 2，因為問題 5 建立在您對問題 2 的答案之上。

在 searchAgents.py 中實作 CornersProblem 搜尋問題。您需要選擇一種狀態表示形式，該表示形式對檢測是否已到達所有四個角所需的所有資訊進行編碼。現在，您的搜尋代理應該解決：

```
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

要獲得滿分，您必需要定義一個抽象的狀態表示法，*該表示法不會*包含不相關的資訊（例如幽靈的位置，額外食物的位置等）。尤其是，不要使用

Pacman GameState 作為搜尋狀態。如果你這麼做的話，你的程式將會非常非常慢。（而且結果會是錯的。）

*提示：*在實作中，您唯一需要參考的遊戲狀態是小精靈初始位置和四個角落的位置。

你所實作的 breadthFirstSearch 在 mediumCorners 上應該展開不到 2000 個搜尋節點。如果用啟發式方法（與 A* 搜尋一起使用）可以減少所需的搜尋量。

為了讓 A* 演算法能夠執行 CornersProblem，需要在 CornersProblem 中實作以下幾個成員函式：

1. **getStartState**：回傳問題的初始狀態，應包含 Pacman 的起始位置和所有角落的初始狀態。這是 A* 演算法的起點。
2. **isGoalState**：判斷給定狀態是否為目標狀態。在 CornersProblem 中，這表示 Pacman 是否已經到達所有四個角落。
3. **getSuccessors**：回傳從目前狀態可以到達的下一個狀態（即相鄰的可能狀態）。每個後繼者應包含：
    - 後繼者狀態（新的 Pacman 位置和更新後的角落拜訪情況）
    - 對應的動作
    - 前往該狀態的步成本（通常為 1）
4. **getCostOfActions**：計算一組動作的總成本。這裡可以假設每一步的成本都為 1。

問題 6（3 分）：角落問題：啟發式

注意：在處理問題 6 之前，請確認已完成問題 4，因為問題 6 建立在問題 4 的答案之上。

在 cornersHeuristic 中，為 CornersProblem 實作一種複雜而且一致的啟發式方法。

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
```

注意 AStarCornersAgent 表示

```
-p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
```

可接受性與一致性：請記住，啟發式只是接收搜尋狀態並傳回數值的函數，這些數值估計到達最接近目標的成本，越有效的啟發式方法會傳回越接近實際目標成本的值。要啟發式的值要*被接受(admissible)*，必須是到達最接近目標的實際最短路徑成本的下限（並且不能為負數）。為了*保持一致(consistency)*，它必須還要另外具有以下特性：如果一個動作的成本為 *c*，那麼採取這個動作頂多只能讓啟發式下降 *c*。

請記住，可接受性不足以保證圖形搜尋的正確性 - 您需要更強的一致性條件。然而，可接受的啟發式方法通常也是一致的，特別是如果它們來自放鬆問題。因此，通常最容易從腦力激盪開始，討論可接受的啟發式方法。一旦你有了一個可接受的啟發式方法，並且執行良好，你也可以檢查它是否確實是一致的。保證一致性的唯一方法是透過理論證明。但是，通常可以透過驗證對於您所展開的每個節點，其後續節點的 f-value 是否大於或等於該節點來檢測不一致。此外，如果 UCS 和 A* 傳回不同長度的路徑，則啟發式為不一致。這個東西很麻煩！

複雜啟發式：簡單啟發式是總是傳回零 （UCS） 的啟發式方法並且會計算出完成時的真實成本。前者不會為您節省任何時間，而後者會造成 autograder 的 timeout。您需要減少總計算時間的啟發式方法，但對於這個作業而言，autograder 只會檢查節點數目（並強制實施合理的時間限制）。

評分：您的啟發式方法必須是複雜的非負一致啟發式方法，才能獲得任何分數。請確認您的啟發式方法在每個目標狀態下都傳回 0，並且永遠不會傳回負值。根據啟發式展開的節點數，您的得分如下：

展開的節點數	**分數**
2000 以上	0/3
2000 以內	1/3
1600 以內	2/3
1200 以內	3/3

*請記住：*如果您的啟發式方法不一致，將不會得到*任何*分數，因此請小心！

問題 7（4 分）：吃掉所有的食物

現在，我們將解決一個困難的搜尋問題：小精靈要在盡可能少的步驟中吃掉所有的食物。為此，我們需要定義一個新的搜尋問題，把食物清除問題正式化（已在

searchAgents.py 的 FoodSearchProblem 中實作），這個問題的解答是小精靈吃掉所有食物的路徑。對於本專題，解答不考慮任何幽靈或無敵食物顆粒;解答僅取決於牆壁，一般食物和小精靈的位置。（當然，幽靈會破壞解決方案的執行！這個問題將在下一個專題再考慮）。如果您正確撰寫了通用的搜尋方法，則具有空啟發式（相當於成本一致搜尋）的 A *在 testSearch 時應該可以快速找到最佳解，而且無需更改您的程式（總成本為 7）。

注意: AStarFoodSearchAgent 是
`-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic` 的簡寫。

您應該發現，即使對於看似簡單的 tinySearch，UCS 也開始變慢。我們的實作在展開 5057 個搜尋節點後需要 2.5 秒才能找到長度為 27 的路徑，以上結果供您參考。

注意：在處理問題 7 之前，請務必完成問題 4，因為問題 7 建立在您對問題 4 的回答之上。

在 searchAgents.py 中填寫 **foodHeuristic** 並為 FoodSearchProblem 提供一*致的*啟發式方法。在 trickySearch 迷宮中執行您的代理：

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

我們的 UCS 代理在大約 13 秒內找到最佳解決方案，探索超過 16000 個節點。

任何複雜的非負一致啟發式方法都將獲得 1 分。請確認您的啟發式方法在每個目標狀態下都會傳回 0，並且永遠不會傳回負值。根據啟發式展開的節點數，您將獲得額外的分數：

展開的節點數	**分數**
15000 以上	1/4
15000 以內	2/4
12000 以內	3/4
9000 以內	4/4（滿分;中等難度）
7000 以內	5/4（額外分數;很難）

*請記住：*如果您的啟發式方法不一致，您將不會獲得*任何*分數，因此請小心！

問題 8（3 分）：次佳解搜尋

有時，即使用了 A *和一個好的啟發式方法，找到通過所有點的最佳路徑也很困難。在這些情況下，我們仍然希望快速找到一條相當好的路徑。在這個專題中，您將撰寫一個總是貪婪地吃掉最接近食物的代理。在 searchAgents.py 中已為您實作 **ClosestDotSearchAgent**，但是它缺少一個找到指向最接近食物的路徑的關鍵函式。

在 **searchAgents.py** 中實作 **findPathToClosestDot** 我們的代理在一秒鐘內解決了這個迷宮（次佳解！），路徑成本為 350：

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

*提示：*完成 **findPathToClosestDot** 的最快方法是填寫 **AnyFoodSearchProblem**，它缺少了目標測試。然後，使用適當的搜尋函式解決此問題，最後解出答案的程式應該會很短！

您的 ClosestDotSearchAgent 並不總是能找到穿越迷宮的最短路徑。請確認你知道為什麼，並試著想出一個小例子，反覆去最近的食物不會找到吃掉所有食物的最短路徑。

繳交

請將 search.py 和 searchAgents.py 上傳到課輔系統，**不要**上傳整個 project1_search.zip 檔或目錄中的其他檔案，否則 autograder 將無法正確評分。