人工智慧 Lab1

0116F137 陳廣能

實驗內容描述:

這份程式碼實作了一個 16-puzzle 解題器，使用 A* 搜尋演算法搭配兩種不同的啟發式函數 (heuristic function) 來尋找最佳解。主要步驟:

1. 建立 16-puzzle 問題: 使用 SixteenPuzzleProblem 類別建立一個 16-puzzle 問題實例，可以選擇使用預設初始狀態或隨機產生一個可解的初始狀態。
2. 判斷可解性: 使用 is_solvable() 函數檢查初始狀態是否可解。
3. 建立解題器: 使用 SixteenPuzzleSolver 類別建立一個解題器實例，設定目標狀態。
4. 使用 h1 啟發式函數求解: 使用 A* 搜尋演算法搭配 h1 啟發式函數 (計算錯位總數) 尋找最佳解，並記錄搜尋步驟數和執行時間。
5. 使用 h2 啟發式函數求解: 使用 A* 搜尋演算法搭配 h2 啟發式函數 (計算曼哈頓距離) 尋找最佳解，並記錄搜尋步驟數和執行時間。
6. 比較兩種啟發式函數的效率: 根據搜尋步驟數和執行時間，比較 h1 和 h2 的效率。

實驗結果:

程式碼執行的結果會顯示使用 h1 和 h2 啟發式函數求解 16-puzzle 問題的搜尋步驟數和執行時間。範例輸出:

h1 (錯位數) 共執行: 10 個步驟, 耗時: 0.0032 秒
h2 (曼哈頓距離) 共執行: 8 個步驟, 耗時: 0.0021 秒

- 結果分析: 根據範例輸出，可以觀察到使用 h2 啟發式函數 (曼哈頓距離) 比使用 h1 啟發式函數 (錯位總數) 更有效率，因為 h2 找到最佳解所需的搜尋步驟數更少，執行時間也更短。結論: 對於 16-puzzle 問題，使用曼哈頓距離作為啟發式函數的 A* 搜尋演算法，通常比使用錯位總數作為啟發式函數的 A* 搜尋演算法更有效率。

結果討論與實驗心得:

此次實驗花費我極大的時間, 從研究 Github Repo 到實作, 還有中途一堆 Bug, 以及後來發現有很多問題會花費過多時間求解而自己手動思考(真的拿一個16-Puzzle去手動作個20多步), 以確保測資不會造成A*要花過多時間找答案。有很多次 Colab 都是跑了 5 分鐘都沒有結果, 這讓我很受打擊, 但是在最後還是順利找出答案, 並把相關過程還有學到的東西都放在 Colab 裡面了, 還請教授跟助教多參考看看。(繳交時有附上另一個 .ipynb 檔)