人工智慧 HW3 BreakThrough
資工 112 許哲安

1. MacBook Pro (15-inch, 2019) Processor: 2.3 GHz 8-Core Intel Core i9 Memory: 16 GB 2400 MHz DDR4 Graphics: Radeon Pro 560X 4 GB Intel UHD Graphics 630 1536 MB

OS: Mac os Big Sur

開發語言:python 3

會選擇這樣的規格沒有什麼特別原因，就剛好開學時師大旁的電腦店有優惠，所以就購買了

電話:0905235578

執行方式：有以下兩種

(1) 於 terminal 進入 breakthrough folder ，執行 python3 –m main

(2) 將 main.py, Engine.py, AI.py 與 images folder 匯入 replit，並執行

運作方式： –滑鼠點選欲移動的棋子，再點選欲到的位置

–按 z 退回上一步，按 r 重回初始盤面 （此功能確定能在 replit 使用，本機端不確定，詳細可以參考 demo 影片）

DEMO 影片：https://youtu.be/Kb2qgpP9LrE

2.

(1)方法：主要利用 python 套件 pygame 去做所有介面設計

(2)資料結構：利用 python class 存取棋盤與走步的相關資訊

(3)操練要項：

a.棋盤表示：利用 2D 8*8 array 表示棋盤

b.走步產生：儲存還在棋盤上的棋子之所有可能的走步方式於 array 中

c.節點資訊：<1> GameState:

board (棋盤)

whiteToMove(是否換白色棋子)

movelog(儲存所有歷史走步資訊)

gameEnd(是否遊戲結束)

<2>Move:

startRow(起始 row 位置)

startCol(起始 col 位置)

endRow(結束 row 位置)

endCol(結束 col 位置)

pieceMoved(起始位置的棋子 white or black)

pieceCaptured(結束位置的棋子 blank or white or black)

moveID(每個走步的 ID)

d.局面重複：沒有考慮局面重複的問題

e.跑多深：深度可到 AI.py 的第 5 行 DEPTH 更改深度，目前預設深度是 4，深度 5 的會稍微慢一些

f.逾時作負：目前深度 4 所需要的時間不需要太久，所以沒有考慮到需限制多少時間

g.記憶體會不會爆：深度太深有機會，不過目前深度 4 沒有這個問題

e, f, g 均是以 NegaMaxAlphaBeta 演算法為前提，如果是 MinMax 與 NegaMax，會慢非常多，深度可能不用太深記憶體就會爆，詳細表現於測試盤面表現中說明。

(4)測試盤面表現：

測試在深度為 4 的情況下，不同搜索演算法，總共需要拜訪的節點數，在這邊 NegaMaxAlphaBeta 測試了 6 步，而 MinMax 與 NegaMax 只測試 2 步，在第一次搜索就可以發現 NegaMaxAlphaBeta 只探索了 2325 個節點，約是 MinMax 與 NegaMax 的 1/10，有明顯的差距，也的確

於測試的過程中需等待較長的時間，才能決定下一步的走步。我也發現在遊戲剛開始的過程中，搜索節點數理應越來越多，因為隨著黑白漸漸離開初始位置，下一步的選擇也就逐漸增加，不會像初始盤面只有第一排的有走步的機會。

NegaMaxAlphaBeta: 2325, 5969, 5860, 9903, 20001, 14381

MinMax: 293982, 352266

NegaMax: 292469, 347693

(5)功能優點：

功能請參考第一題的執行方式與 demo 影片，優點我認為有

1. 標註顏色給使用者，讓他知道他要移的棋子有哪些合理的走步
2. 能人跟人、人跟機器、機器跟機器對下，操作請參考 demo 影片
3. 能選擇深度與不同的搜索演算法，操作請參考 demo 影片
4. 有歷史走步提供使用者

3.參考網站

(1) Chess Engine in Python - Part 1 - Drawing the board - YouTube
參考 pygame 棋盤繪製方法
(2) Chess Engine in Python - Part 2 - Moving the pieces - YouTube
參考走步的產生方式
(3) Chess Engine in Python - Part 3 - Undo moves, start generating valid chess moves - YouTube
參考收回走步與產生合法的走步
(4) Chess Engine in Python - Part 5 - Generating all possible moves - YouTube
參考產生所有走步的方式
(5) Chess Engine in Python - Part 10 - Move animation/highlighting, reset and end game text - YouTube
參考走步的動畫、重回設定與結束字體
(6) Chess Engine in Python - Part 11 - Random Move AI - YouTube
參考產生隨機走步方法
(7) Chess Engine in Python - Part 12 - Greedy Algorithm and MinMax without recursion - YouTube
參考 MinMax 演算法實作方式
(8) Chess Engine in Python - Part 14 - Nega Max and Alpha Beta Pruning - YouTube
參考 Nega Max 與 Alpha Beta Pruning 演算法實作方式
(9) Chess Engine in Python - Part 15 - More bug fixes and Move Log Display - YouTube
參考 pygame 顯示歷史走步方式

4.狀況與困難：

最大的困難是剛開始的時候，因為不知道如何開始，尤其是面對一個自己過去從來沒有碰過的項目，完全沒有概念，到底要用哪種語言寫？那種語言有支援介面的部分？我是要從零開始，還是找 source code 改？要選擇 breakthrough 還是愛因斯坦棋？起初光選擇語言的部分就煩惱了蠻久的，因為老師有提供用.NET 寫的版本，但參考之後我發現自己對於整體的架構與語言的熟悉度不佳，之後又考慮單純用 c，但思考後又覺得介面的部分以目前我對 c 的了解，最好用的方式只能單純用 terminal print，不太適合，況且目前比較流行的語言也不是 c，在資料查找方面比較匱乏，可能造成花非常多時間寫，但功能效果不彰等問題，所以也作罷。後來有找到用 javascript p5 做 chess 的教學影片，跟著一步一步做後，效果很不錯，但我其實也不熟悉 javascript ，考量到這個問題，決定查詢 python chess AI 等相關資訊，因為我發現 breakthrough 遊戲，其實就是 chess，只是棋子只有一種，走步的方式也大同小異，那也與 javascript 相比之下，我 python 比較熟悉，最後就一步一步跟著網路上的教學，從無到有，把 chess 的，改成 breakthrough，過程中也非常驚訝，其實沒有那麼複雜，架構也沒有那麼難懂，也摸了一些 pygame 模組，收穫非常多，