人工智慧 Project1

0856096 江陳建鏵

1. **用甚麼方法**

   主要：MiniMax

   優化：Alpha-Beta pruning + 使用 dictionary 紀錄最後一步

2. **如何寫的**

   MiniMax 終止遞迴的條件: 如果遊戲結束或是樹的深度等於 0 的時候，回傳 evaluation value

```
if self.game_over or depth == 0:
    score = self.evaluate(self.checkerboard, self.color)
    return [-1, -1, -1, -1, score]
```

找到目前棋盤上自己的棋子可以走的步伐，並依照 History Table 的值將所有可以走的步伐排序(遞減)，這個方法是我去網路查要怎麼優化 alpha-beta minimax 查到的

```
ValidMove = self.valid_move(player)

rating = []
for cell in ValidMove:
    move = str(cell[0]) + "," + str(cell[1]) + "," + str(cell[2]) + "," + str(cell[3])
    try:
        rating.append(self.HistoryTable[ move ])
    except KeyError:
        self.HistoryTable[ move ] = 0
        rating.append(self.HistoryTable[ move ])
ValidMove = [x for _, x in sorted(zip(rating, ValidMove), reverse = True )];
```

走出那些步伐後遞迴找下一步另一方會怎麼走，在遞迴之前會先記錄目前的狀態，遞迴後再恢復狀態

```
for cell in ValidMove:

    #記得現在的狀態
    broad_now = copy.deepcopy(self.checkerboard)
    gmae_over_now = self.game_over
    n_piece_now = copy.deepcopy(self.n_pieces)
    n_barrier = copy.deepcopy(self.n_barriers)
    budget_now = copy.deepcopy(self.budget)

    self.make_move(player, cell[0], cell[1], cell[2], cell[3]) #走valid move

    score = self.minimax(depth - 1, alpha, beta, 1 - player)

    #恢復之前的狀態
    self.checkerboard = copy.deepcopy(broad_now)
    self.game_over = gmae_over_now
    self.n_pieces = copy.deepcopy(n_piece_now)
    self.n_barriers = copy.deepcopy(n_barrier)
    self.budget = copy.deepcopy(budget_now)
```

若遞迴有傳值回來後就執行 alpha-beta pruning，看可不可以不用再展開其他的 child，以節省時間。

```
            if player == self.color: #Max
                if score[4] > best[4]:
                    best = score  # max value
                alpha = max(alpha, score[4])
                if beta <= alpha:
                    break
            else:
                if score[4] < best[4]:
                    best = score  # min value
                beta = min(beta, score[4])
                if beta <= alpha:
                    break
```

3. 我的發現

在做這個作業的時候，我參考網路上的教學影片很快地就可以寫出 minimax 還有 alpha-beta pruning 的演算法，花比較多的時間都在優化 AI 整體的速度跟思考 evaluation function 的設計。

優化 AI 速度: 因為每一顆棋子的每一條直線最多有 8 個位子可以走，這會導致 search tree 展開太多的 children，所以我簡化成每一條直線只能走棋子上下左右前一步跟上下左右最遠的地方，因此從最多 24 種可能縮減到 8 種可能，雖然這會減少 AI 的聰明程度但可以讓 search tree 更深。

Evaluation function 設計: 我一開始用棋盤上有多少個敵人的棋子還有多少自己的棋子，然後在跟 AI 玩的過程中發現我們可以給不同的權重，像是旗子那條 column 還有旗子左右兩邊的 column 的權重給得比敲掉磚塊的權重還高，代表棋子走到中間那三條 column(3, 4, 5)比敲掉磚塊還重要。我花很多時間在尋找棋子走到甚麼地方或做甚麼事情會比較強還有給不同的權重會造成甚麼結果。我在想這些參數可能就是機器學習在 train 的參數吧。

4. 如何執行

- 需要的 package

  numpy, math, copy, time
- 打開 command line 並輸入” python 0856096.py” 即可執行