AI Capstone: Group Project #1

0816183 謝承恩、0816097 薛乃仁

`group 1: small brain agent >:[`

National Yang Ming Chiao Tung University

1 AI Framework: UCT

在 MCTS 中有以下分支

- Pure MCTS

  沒有做其他額外更動的 MCTS 。

- UCT(MCTS with UCB)

  在做選擇的時候計算 Upper Confidence Bound 後選出最佳的選項向下做模擬，因爲會考慮 Exploration 和 Exploitation ，所以會比原本的隨機選擇模擬還好。

- MCTS with $\alpha - \beta$ pruning

  因爲在這個遊戲中，並非雙人對戰，因此我們想不太到一個好的方法去做 pruning ，導致最後並沒有選擇這個架構。

- MCTS with RAVE

  RAVE 假設是相同的動作不管先後順序都應該一起做評估的計算，但是我們認爲這個遊戲中，並不是同個動作就能達成遊戲結果，而且也有過程中也嚴重的先後問題，所以最後也沒有採用這種做法。

在這個遊戲中，我們最後選擇了使用 MCTS 加上 UCB 來去做計算最佳動作。

1.1 State

我們針對這個遊戲做出了記錄 state 的結構，我們可以由當下的結構判斷出還能移動的玩家、當下玩家合法的動作與當下遊戲的分數，且可依照執行的動作生成出下一步的 state 。

因爲我們並沒有對 MCTS 做除了 UCB 以外的操作，因此我們主要放心力的地方就是優化處理速度，在 state 中我們主要是以 np.array() 做記錄，我們在存取 MapStat 時順便把不同方向移動的地圖存下來。

```
self.dir = np.ones((6, 12, 12))*-1
self.dir[0,2::2,1:] = self.board[1:11:2,:-1]
self.dir[0,1::2,:] = self.board[::2,:]
self.dir[1,2::2,:] = self.board[1:11:2,:]
self.dir[1,1::2,:-1] = self.board[::2,1:]
self.dir[2,:,1:] = self.board[:,:-1]
self.dir[3,:,:-1] = self.board[:,1:]
```

```
self.dir[4,::2,1:] = self.board[1::2,:-1]
self.dir[4,1:-2:2,:] = self.board[2::2,:]
self.dir[5,::2,:] = self.board[1::2,:]
self.dir[5,1:-2:2,:-1] = self.board[2::2,1:]
```

藉由直接存取不同方向的地圖，我們就能將獲得合法動作以下方程式去做查找

```
for d in range(6):
    legal_direction = player_cell & (self.dir[d] == 0)
    indices = np.where(legal_direction)
    for i, j in zip(indices[0], indices[1]):
        for m in range(1, self.sheep_state[i][j]):
            legal_action.append(BattleSheepMove(self.next_move_player, i, j, d, m))
```

基本上在選擇初始位置也是相同的方法，這樣的寫法省略了需要對每個可移動的點做方向移動且判斷是否合理而提升了我們搜尋速度 (約 7 倍)，使我們在時間內可以做大量的計算。

1.2 MCTS Node

Node 中我們存取了當下的 state 與衍生出來的子節點，而我們可以拿來計算當下最好的子節點去做模擬，R 爲每個玩家所有模擬結果的分數加總。

$$A_t = \begin{cases} \mathsf{argmax}_i(\hat{\mu}_i(t-1) + c\_param \times \sqrt{\frac{2\ln f(t)}{T_i(t-1)}}), & \text{if } t > K; \\ t, & \text{otherwise.} \end{cases}$$

$$\hat{\mu}_i = \mathsf{softmax}(\frac{R}{\Sigma R})$$

我們對分數取 softmax 的原因在於，基本上每個玩家下一步棋的獲得 reward 都是 3 ，因此如果下了一步很糟的棋使未來的平均得分比其他玩家低很多的時候就會獲得近乎爲 0 的分數。

1.3 MCTS

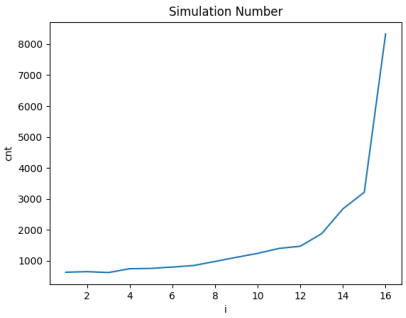
因爲我們可以獲得各個狀態下玩家的分數，所以我們可以在建構 MCTS 時選擇不同的搜尋深度，也可以設定不同的 c_param 去決定當下要偏向 Exploration 或 Exploitation ，而在模擬的環節我們使用隨機的動作來做更新 state 。

2 Experiments

我先測試了在運行遊戲中每次 Agent 能模擬的次數，基本上在第一步時能模擬 600 至 700 次，而第二到四步則會因爲每個玩家能夠下的步數急速上升影響，導致模擬次數下降到 500 至 600 。

而我們依照不同的搜尋深度以及不同的 c_param 做出了總計 4 個不同的 Agent ，分別爲

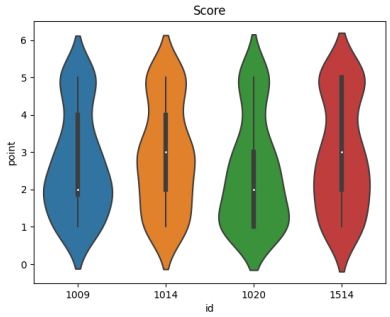
- 1009: 搜尋深度 10, c_param 0.9
- 1014: 搜尋深度 10, c_param 1.4
- 1020: 搜尋深度 10, c_param 2.0



1: simulation number

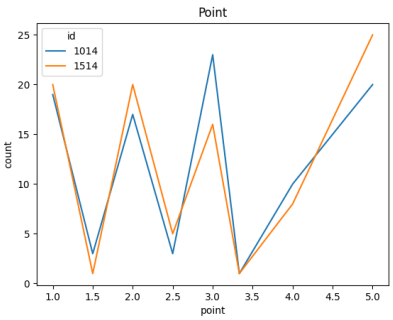
- 1514: 搜尋深度 15, c_param 1.4

而我們依照不同 Agent 的排列順序各自進行四場遊戲，總計統計了 96 場遊玩資訊後進行分析。首先我們可以發現如果 c_param 太高會因爲太常去嘗試不同路徑而導致無法找到最佳的路徑，而太低會只嘗試某幾種路徑反而找不到所有路徑的最佳解，因此我們認爲在 1.4 時的成果最好，而實驗結果也可以看到基本上在 1.4 時平均獲得分數大於其他數值 1 點。



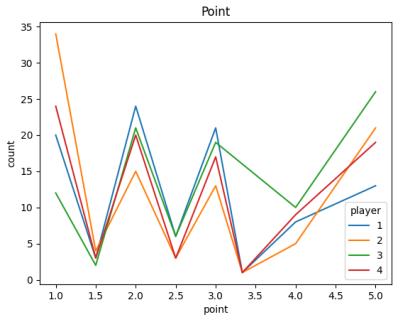
2: match result

在深度的問題中，我們可以發現搜尋深度較深的 Agent 在獲得第一的次數比搜尋深度淺的數量還要多，但是在分數較低的狀況下也是較多，我們認爲是因爲搜尋深度較深有時會因爲搜尋次數比較少，導致前期如果出現一次致命性的錯誤就會有極大的後果，但是如果沒遇到這個狀況就可以因爲得知眞正的結果使分數上升。



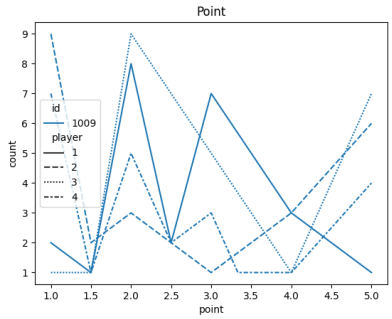
3: match result

而在順序的影響中，我們發現這個遊戲對於玩家順序有很大的影響

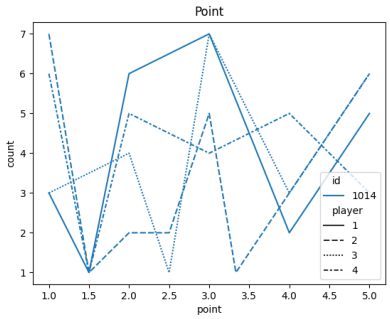


4: match result

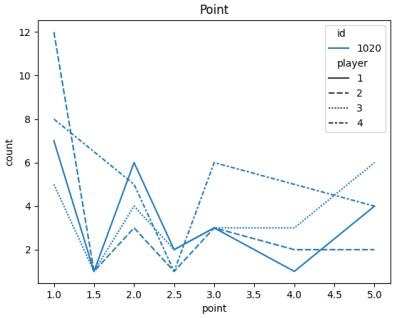
第一個玩家因爲完全沒有其他人下棋的資訊，所以獲得第一的機會直接少於其他玩家將近一半，而這也會影響到不同的 Agent 行爲。



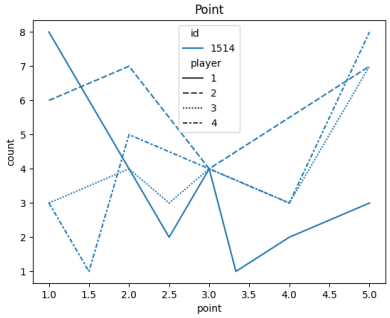
(a) Agent 1009



(b) Agent 1014



(c) Agent 1020



(d) Agent 1514

5: Player Order

我們可以發現在玩家爲第一、二個進行移動的人時，我們可以降低搜尋深度，使他做更多次的搜尋，作出比較好的動作，而在較晚進行移動的玩家可以做更深度的搜索。

3 Experiences

這次的遊戲隨機成分太高了，基本上因爲地圖，玩家順序，玩家決策頃向等等有太多的面向可以去考量與調整，而因爲每次進行測驗也要花費許多時間，因此我們最終還是將主力方在如何加速搜索次數上，因爲在 MCTS 中，搜尋次數需要到一定的程度決策才會好，而又有多位玩家同時遊玩，與其考量每個面

向，不如花時間在搜索去選出較好的動作。

4 Contributions

姓名	分工比
薛乃仁	80%
謝承恩	20%

表 1: Contributions