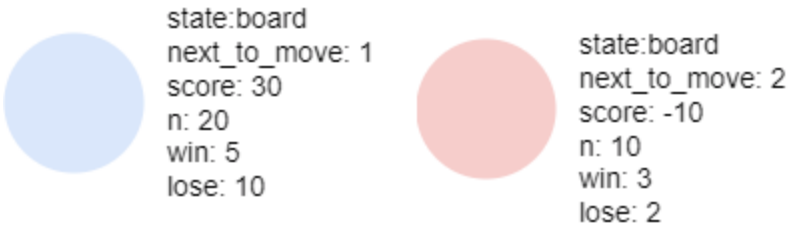
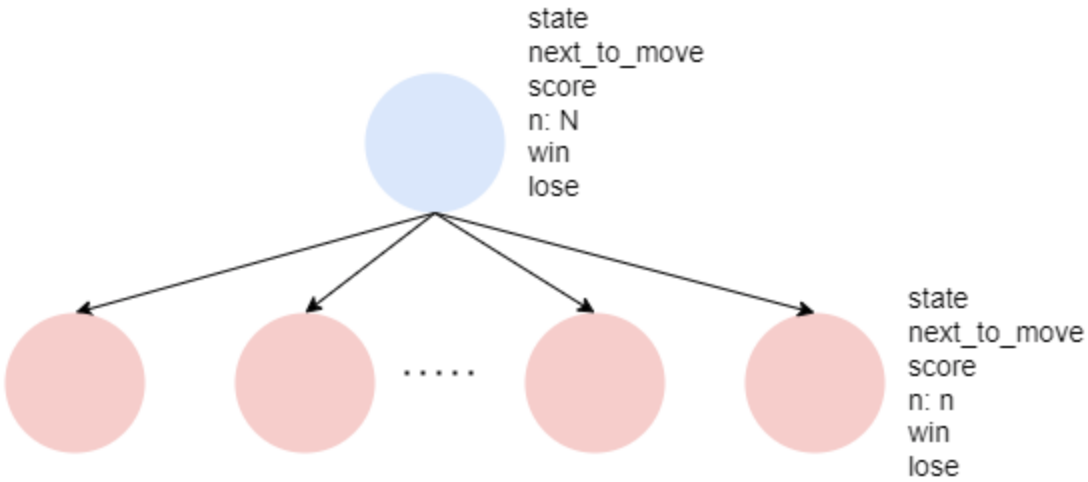
Cantris

Search Strategy: MCTS(Monte Carlo tree search)

- Node



  - state: 紀錄board的狀態
  - next_to_move: 紀錄下一步換player1 or player2
  - score: 紀錄player1贏player2多少分(負代表輸多少分)
  - n: 在Backpropagation中,通過該node幾次
  - win: 勝利的次數
  - lose: 失敗的次數
- MCTS Step1: Selection



Parent node用以下公式選擇children node

$$\frac{q}{n} + c_{param} * \sqrt{\frac{2 * logN}{n}}$$

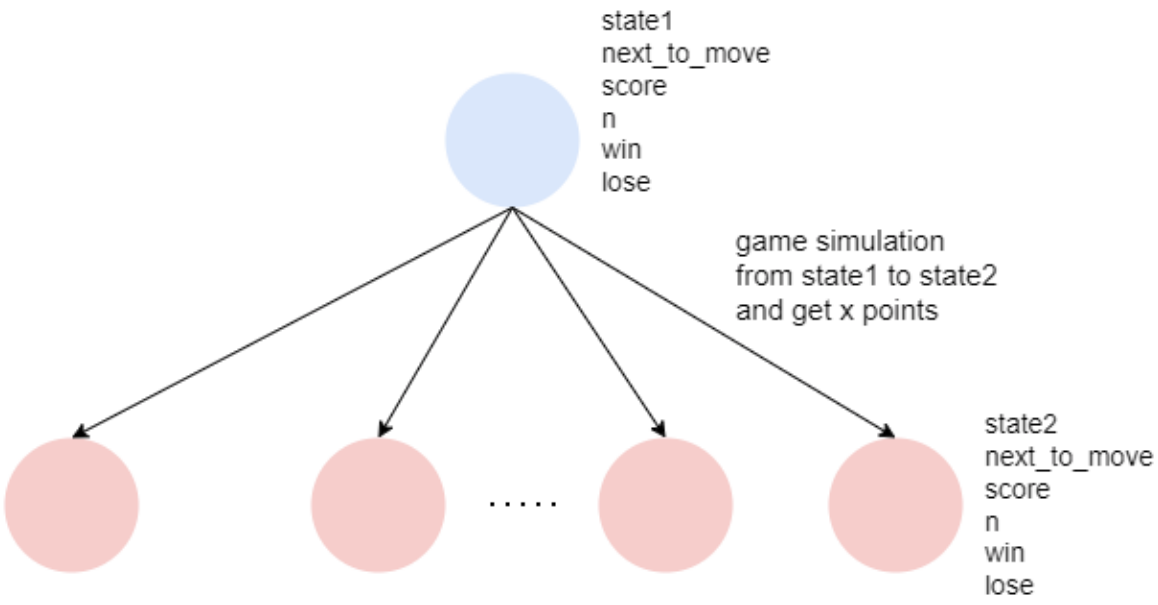
n為children node的n; N為 parent node的n

q有兩種算法(根據children node)

- q = win
- q = win - lose

c_param為hyperparamter,之後實驗中,會測試1.0及1.4

- MCTS Step2: Expansion



從state1進行expansion到state2時,會進行game simulation,並求得從state1到state2的得分為x分,

若parent node的next_to_move為1,則child node的score = parent node的score + x

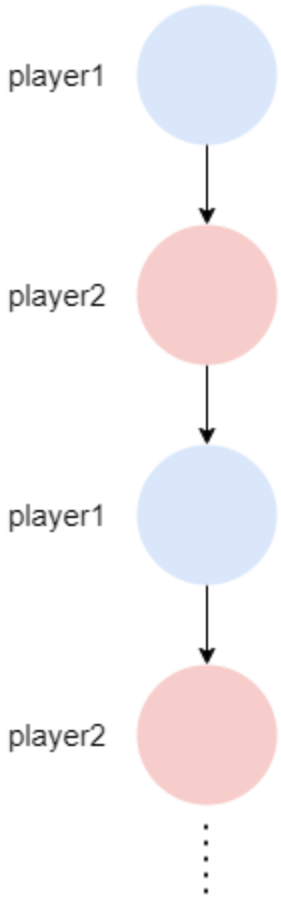
若parent node的next_to_move為2,則child node的score = parent node的score - x

- MCTS Step3: Rollout

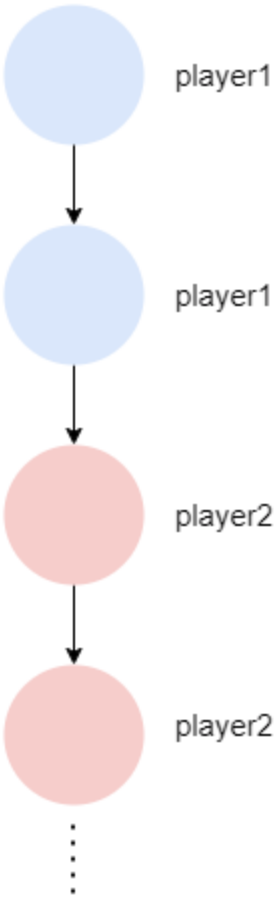
採用隨機策略進行遊戲

- MCTS Step4: Backpropagation
  1. 判斷勝負: 以leafnode的score進行判斷
     a. score > 0: player1 win
     b. score = 0: tie
     c. score < 0: player2 win
  2. 若parent node的next_to_move與勝方相同,則win+1,若與勝方相反,則lose+1
  3. n+1
- 8x4(6x3)與10x5的區別

8x4(6x3)



10x5



Experiments

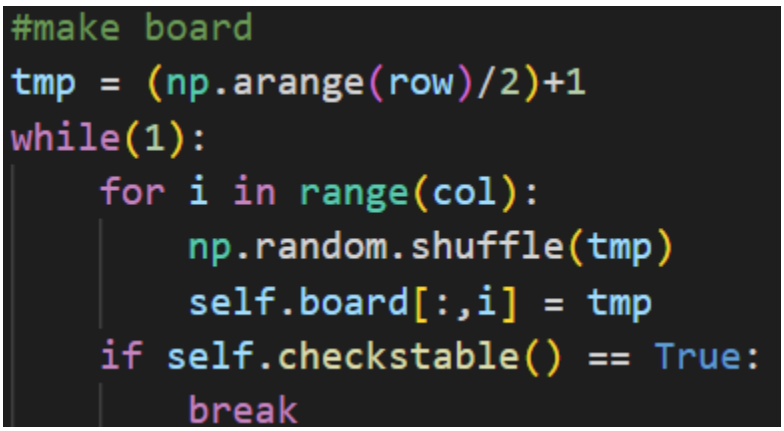
- 接下來的實驗中,都會使用到以下的基礎設定
  - 模擬10000場對決
  - 兩個玩家: al(implemented by myself) and oponent
  - oponent一律採取random moves
  - 6x3跟8x4相似,所以只會針對8x4及10x5進行實驗
  - 先後手隨機決定(因為先後手會影響勝率,先手勝率如下表所示)
    - 8x4

win	tie	lose
0.5453	0.0287	0.4260

    - 10x5

win	tie	lose
0.5929	0.0187	0.3884

  - 每次模擬都是隨機產生一個合法的盤面(使用助教提供的程式碼)



```
#make board
tmp = (np.arange(row)/2)+1
while(1):
    for i in range(col):
        np.random.shuffle(tmp)
        self.board[:,i] = tmp
    if self.checkstable() == True:
        break
```

- exp 0:
  - setting:
    - al採取random moves

- result:
  - 8x4

win	tie	lose
0.4881	0.0281	0.4838

  - 10x5

win	tie	lose
0.4940	0.0175	0.4885

- exp 1:
  - setting:
    - ai採取MCTS
    - 每步進行的MCTS模擬次數 = **500** for 8x4 and **250** for 10x5
    - c_param=**1.0**(when doing MCTS); c_param=**1.0**(when selecting best move after MCTS)
  - 變因:
    1. q = win count
    2. q = win count - lose count
  - result:
    - 8x4

	win	tie	lose
1	0.8790	0.0048	0.1162
2	0.9486	0.0046	0.0468

    - 10x5

	win	tie	lose
1	0.7865	0.0062	0.2073
2	0.9043	0.0041	0.0916

- conclusion: q=win-lose結果較好。傳統的MCTS只會計算win count,但我認為lose count也一樣重要,而結果也確實如此。

- exp 2:
  - setting:
    - al採取MCTS
    - 每步進行的MCTS模擬次數 = **500** for 8x4 and **250** for 10x5
    - q=win-lose
  - 變因:
    1. c_param=**1.0**(when doing MCTS); c_param=**1.0**(when selecting best move after MCTS)
    2. c_param=**1.0**(when doing MCTS); c_param=**0.0**(when selecting best move after MCTS)
  - result:
    - 8x4

	win	tie	lose
1	0.9486	0.0046	0.0468
2	0.9884	0.0014	0.0102

    - 10x5

	win	tie	lose
1	0.9043	0.0041	0.0916
2	0.9856	0.0014	0.0130

  - conclusion: c_param=**1.0**(when doing MCTS); c_param=**0.0**(when selecting best move after MCTS)結果較好。在下方式子中,右項是為了確保在進行Selection時,能選擇較少被visit的node,所以c_param設為1。但在MCTS結束並要選擇最好的node(move)時,右項就不需要被考慮,所以c_param設為0。

$$\frac{q}{n} + c_{param} * \sqrt{\frac{2 * logN}{n}}$$

Final Settings

- maximum search time = 25s
- c_param=**1.0**(when doing MCTS); c_param=**0.0**(when selecting best move after MCTS)
- q = win count - lose count

Reference

https://github.com/int8/monte-carlo-tree-search