

Group Game Project #1

Group24

李亭萱 0816016、林佳萱 0816047

1. Introduction to the problem

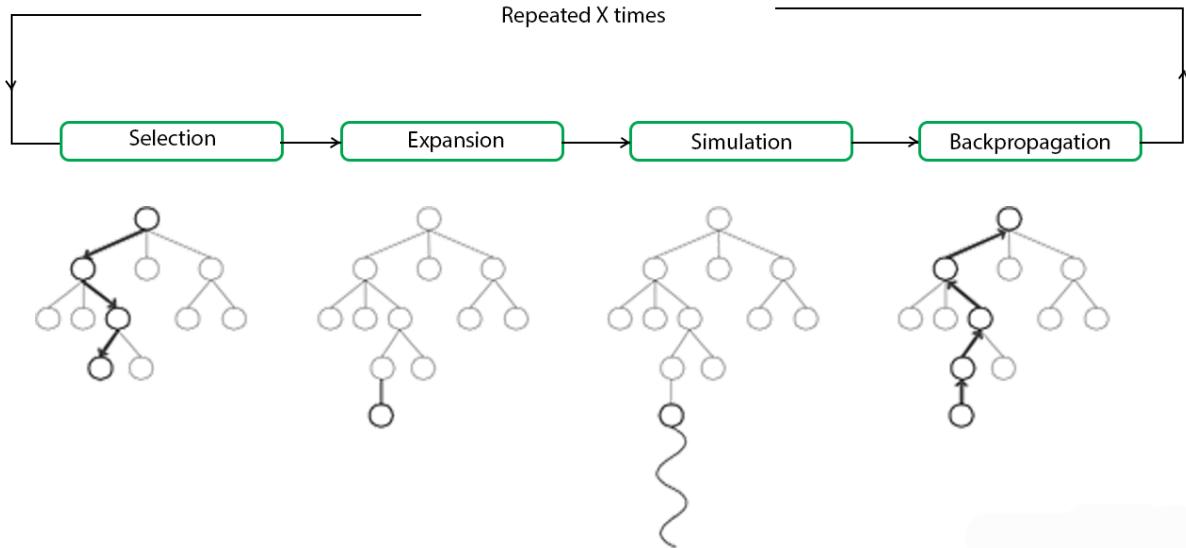
本次的小組作業是需要使用某種自訂的遊戲策略，在 Battle sheep 這個遊戲中取得勝利，也就是要與其他羊群競爭，想辦法取得更多地盤。而我們這組採取的遊戲策略是利用 Monte Carlo Tree Search(MCTS) 實作出 game agent，以下將逐一敘述 MCTS 的實作細節以及實驗結果和心得。

2. How game AI works

- 首先，每次對戰的第一步(選擇起始位置)，我方會選擇邊界上(至少一個方向為牆)具有較多相鄰空格的位置下，也就是每次都會先隨機選一個邊界上的位置，然後從此位置開始往外擴展，去檢查其兩層內的空格數(檢查 18 個位置)，假如其中有大於 10 個空格就會優先選它當作起始位置。選擇此位置的策略為：對於整個盤面而言，此位置的「自由度」較高，有機會從周圍的多個方向延伸出去，不會因為被敵方擋住去路而只能「故步自封」，淪落到提早失去遊戲資格。
- 另外，對戰途中產出的指令(羊群位置、羊群數量、方向)，則是由 MCTS agent 負責的部分。每次的對戰，MCTS agent 都必須知道當下的 player turns 和 unoccupied cells (available moves to choose)，才能從目前 state 現有的 available moves 去 expand node，再從這些 node 去模擬對戰 (roll-out)，然後每一次就從這些 node 中選出最好的 node 來當作下一個 move。在每次的對戰中，MCTS agent 都必須記錄當下盤面的 state 以及整個樹狀結構(node, number of rollouts)。
- 我們會利用 MCTS 來做 tree search，是因為在模擬對戰的過程中，計算的過程是以樹狀結構紀錄，從 root node 往下展開後，根據 child node 的對戰結果，往回 backup 紿中途的每個 node 來計算勝率。以 battle sheep 這個遊戲來說，若真的去計算每一個 node 會讓整個樹狀結構太大，而且也會超過時間限制，因此我們利用 MCTS 來減小 search space，同時又能夠搜尋最大勝率的 node。

3. Our Algorithm

- MCTS主要分成 selection、expansion、simulation、backpropagation 四大步驟，fig(1). MCTS search。
 1. Selection: 選擇要進行擴展的 child node。
 2. Expansion: 展開下一層。
 3. Simulation(rollout): 計算這個 node 的 value(估計勝算)。
 4. Backpropagation: 由下往上更新各個 node 自己的勝率。



fig(1). MCTS search

- 以下將逐一介紹MCTS agent中三個重要的class以及其function的作用。
MCTSMeta: store constant parameters, 方便進行experiment, 尋找best parameters的時候更改參數, 比較結果。

Node: 在 search tree 上頭每個 node 都代表各自的 game state。

- MCTS 的data structure:

1. `root_mapstate/ root_sheepstate`: 首先, 每個node都必須儲存當下的棋盤狀態。
2. `num_rollouts`: 這個node總共進行幾次對戰。
3. `player`: 這個node是由哪一方在下棋。
4. 最後, 其中 node 的 DS 包括儲存其擁有的 child 個數、自己所指向的 parent node 的 pointer, 來幫助 backpropagation 的進行。

- 其中每個 node 都有各自的 evaluation 值, 我們選用的 evaluation function 是 UCT, UCT 這個 formula 的目的就是為了要取得 exploration 和 exploitation 之間的平衡。

UctMctsAgent: Handles MCTS 的整個樹狀結構。

> Selection

針對 selection 的部分, 我們採用 UCT(Upper Confidence Bounds To Trees)當作 utility function, 這個 formula 可以讓 MCTS 在平均較高勝率的 node 間移動, 同時探索少數可能 node 之間的平衡。將 node 帶入 UCT formula 後, 選出數值最高的 node 進行下個步驟的 expansion。

$$\frac{w_i}{n_i} + c \sqrt{\frac{\ln t}{n_i}}$$

fig(2). UCT formula.

> Expansion

在上一個步驟 selection 選出某個 node 之後，我們針對這個 node 去擴展它的 children。在這個遊戲中，適當的防守和進攻是相輔相成的，如果策略集中在進攻模式的話，我方很快就會被敵方擋住去路，反而言之若只是拼命的防守，就有可能會忽略敵方的勝利(特別厲害的對手/ first place)，因此為了兼顧進攻與防守，我們會探索所有的 legal move。

> Simulation(roll_out)

完成前兩步驟後，game AI 會針對所有已經 expand 出來的 node 進行 simulation。MCTS agent 會模擬我方和其他玩家輪流隨機對戰，一直持續直到分出勝負，結束後得到的結果代表這個 node 好壞的估值，類似 evaluation value。雖然我們的 time limit 是設定4秒，但是根據 Monte Carlo 方法的精神：隨機取樣，只要以這個 node 為 root node 的 child node 總模擬次數夠大，就可以趨近真正的值。

> Backpropagation

持續前面 simulation 的步驟一直到遊戲結束，並將勝負結果作為 reward 回傳，往 search tree 的 root node 逐一更新。

不斷重複進行 selection、expansion、simulation、backpropagation 四個步驟，直到執行時間超4秒時，就將最佳的 node 輸出給 agent。

4. Experiments and results

> Experiment 1: Modify parameter

goal: find best exploration parameter for MCTS agent

- fig(2).UCT 的算式，左邊 w/n 是 exploitation(利用過往經驗)，算式右邊的部分是 exploration(總模擬次數 N 越大，或 node 本身被模擬次數 n 越小，exploration 佔的比重越大)，做 exploitation 和 exploration 的目的就是要獲得最大的 long-term reward。

- UCT(UCB for Tree Search)是一種不平衡的 MiniMax Search tree，UCT 在選 node 時，使用 UCB(Upper Confidence Bound) 值作為選點的依據，也就是在擴展 node 時，會選出 UCB 值最高的點進行深度搜尋。在平均獲勝率合理且設定適當的 parameter c 值的情況下，UCT 對於表現較好的 node (獲勝率較高)會探索的比較深入，這樣一來就可以減少搜尋一些不必要的 node，降低 search space，提高效率。

- 一開始 exploration parameter 為 0(但不能賦0, 會出錯), 接著 exploration 比重逐漸增加。因為我們兼顧 exploration 與 exploitation, 最終選擇 exploration = 0.5, 也得到不錯的 reward。

> Experiment 2: init_pos 相鄰空格數量所考量的層數

goal: find best init_pos for MCTS agent

- 本來計畫要讓 init_pos 也下去跑 MCTS, 但是有時間限制的問題, 所以先去找了具有五個鄰近空格的位置, 再從中挑選init_pos。

- 後來嘗試將檢查的層數增加為 2(搜尋的相鄰格數增加為 18 格), 因為自由度提高, 所以結果也是滿意的(不容易被擋住)。

> Experiment 3: reward的賦值

goal: assign optimal reward to get better performance

- 我們的 MCTS agent 中 node 的 value 是以 reward 去計算勝率的, 因此好的reward重要到可以帶領game agent拿第一。

- 因為我們一開始設定的reward是採取遊戲結束後的最終名次依照遊戲規則給分(5, 3, 2, 1), 但效果沒有很滿意。

- 由於我們希望可以拿到較前面的名次, 所以後來決定使用自己定義的分數下去賦值, 由第一到第四名依序是 5, 3, 1, 0 的分數, 以達到追求前兩名的效果。

5. Conclusion and experiences

因為我們本身對 Battle sheep 這個桌遊不熟悉, 所以一開始要花一點時間理解這個遊戲。在理解完遊戲規則後, 我們一開始鎖定兩個方向:DQN 和 MCTS, 剛開始研究了蠻多 DQN 的演算法, 也嘗試了各種寫法, 但結果不太理想且考量到時間限制, 再加上也不知道預先 train 好的 model 要如何傳入 agent。後來決定以 MCTS 當作遊戲策略之後, 首先遇到的問題是要如何設計適合的 DS 去儲存 node, 才不會在 expand 和 simulation 卡住, 我們後來分成 tree node 去儲存node本身的資料(action、parent node、reward等), 然後 MCTS agent 去存整個樹狀結構的資料(root 的 state、rollout、expansion)。在本次作業中, 我們還有遇到方向判定的問題, 因為回傳的值是沒有方向, 所以要自己利用地圖(mapStat 和sheepStat)去設定條件, 計算出我們需要的資訊。還有, 我們也思考了很久到底要如何設計良好的 reward 才能夠優化 MCTS。雖然遇到蠻多問題的, 但完成之後很有成就感, 之前寫python都是一直call library, 這次的路途雖然曲折, 但學到很多東西, 謝謝教授和助教！

6. Contributions of individual team members

- code: 林佳萱 0816047
- report: 李亭萱 0816016