# Programming Assignment #2

106B Introduction to AI

Group 7　楊承皓 0413220、陳亮融 0413234、王傳鈞 0416047

本次的小組程式作業，是需要使用某種自訂的下棋策略，在 Connect-5 的遊戲中打敗對手，也就是：在最短的步數內，將五顆代表自己顏色的棋子連成一線；而我方的下棋策略，是採用蒙地卡羅樹搜尋法(MCTS)，以下將逐一敘述 MCTS 的實作細節，例如：data structure、search implementation 等。

## 一、 在 search tree 上頭的 node 的 data structure

首先，每個 node 當然都必須儲存有當下的棋盤狀態；為了避免大量複製相同的資料在記憶體當中，因此我們實作一個名為 `state_view` 的介面。它的功能類似 C++ 17 的 `string_view`，提供一個「閱覽」某特定記憶體區塊資料的窗口，然而 `state_view` 並非唯讀，而是有修改介面 `state_view::insert`。以下的圖一顯示出使用 `state_view` 的結構，以及只是單純不斷重複儲存資料的差異。

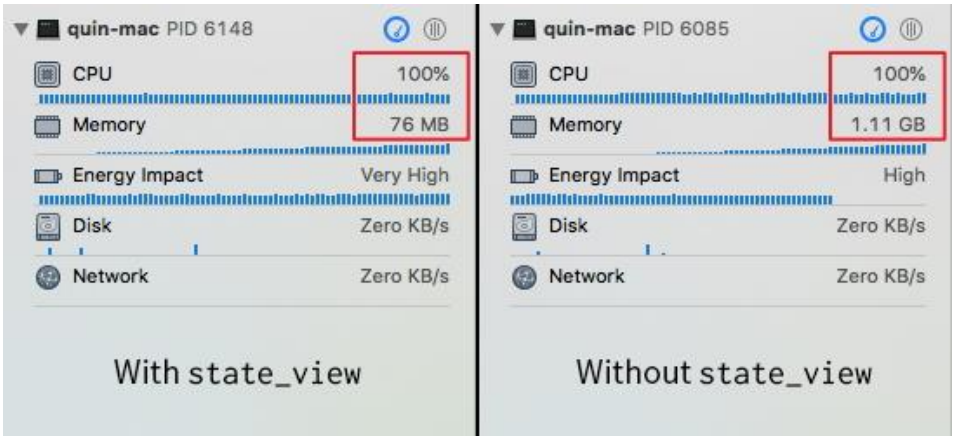


**Figure 1**　Comparison of using state_view and traditional method

接著，透過 `node::_pos` 以及 `node::_player`，可以得知這個 node 是由哪一方下棋和下在哪個位置。另外，`node::_win` 以及 `node:_total` 則是指出這個 node 已經贏了幾次和總共進行幾次對戰。

最後，每個 node 還會儲存：擁有幾個 child 的資料、自己所指向的 parent node 的 pointer，以利搜尋四階段的 back propagation 進行。

## 二、 有關於「成長」search tree 的四步驟

以下針對 MCTS 的 select、expand、simulate、back propagation 四步驟做說明。

### I. Selection

在 selection 的部分，我們選擇 UCB1 當作 utility function，將每一個 node 代入這個函數，並選擇數值最大的 node 進行下一個 expansion 步驟。

$$UCB1(node) = \frac{node :: \_win}{node :: \_total} + \sqrt{2} \times \sqrt{\frac{game's\ total\ ply}{node :: \_total}}$$

**Equation 1**　UCB1 function of a node

### II. Expansion

於前一步驟 Selection 選出的某個 node 之後，我們對此 node 去「擴展」它的 children，並且運用以下 heuristic function 來評分，由高到低排序選擇前十名的 children 做 simulate。

在這個步驟當中，我們開發了兩種模式：attack mode 及 defense mode，目的是求兼顧進攻與防守，讓我方的下棋策略不只有盡快完成五顆棋子的連線，也會注意敵方是否即將取得勝利，而需干擾敵方的下棋策略。

- Attack Mode

  針對多下的一顆棋子，對六個方向延伸的直線進行以下掃描法：

  `Step 0`：`rate = 0`

  `Step 1.a`：若遇到我方的棋子，`rate = rate + 1`

  `Step 1.b`：若遇到敵方的棋子，`rate = rate - 1`

  `Step 1.c`：若遇到空白的位置，`rate = rate + 0`

  `Step 2`：持續掃描，直到觸及棋盤邊界

  獲得六個方向各自的 rate，經由 score = $10^{rate}$ 的換算 (例如：rate = 3，則 score = 1000)，轉而得到六個方向各自的 score，最後將六個方向 score 加總，得到 attack mode 的總得分。

- Defense Mode

  分析多下的一顆棋子，能夠阻斷敵方連續出現的棋子數：

  阻擋 4 顆 → 5000 分

  阻擋 3 顆 → 500 分

  阻擋 2 顆 → 50 分

  阻擋 1 顆 → 5 分

III. Simulation and Back Propagation

完成前述兩個步驟之後，game AI 會針對所有已經 expand 出來的 node 進行 simulation。我方和敵方輪流使用同一個 simulation heuristic 彼此持續對戰，直到有一方獲勝或棋盤完全被下滿棋子 (和局) 為止。

- Simulation Heuristic

  針對棋盤上的每一點，向兩邊延伸出的三條直線做雙向掃描，並根據下表做評分，最後計算評分總和。

| Previous | Now | Score |
|---|---|---|
| Empty | Empty | set to 0 |
| Empty | Player | set to 2 |
| Empty | Opponent | set to 1 |
| Player | Empty | +2 |
| Player | Player | +2 |
| Player | Opponent | set to 1 |
| Opponent | Empty | +2 |
| Opponent | Player | set to 2 |
| Opponent | Opponent | +2 |

**Table 1**　Simulation Heuristic Table

經以上步驟，我們紀錄每個點： (1)三個方向中最大的分數 max (2)三個方向分數加總 total；優先選擇 max 最大的一點做為下一步棋子的落點，若有 max 相同的點，則以 total 大者為優先。

IV. Back Propagation

持續前面所提之 simulation 的步驟，直到分出勝負，並將勝負結果並回傳，往 search tree 的 root node 逐一做更新。

不停地進行 selection、expansion、simulation、back propagation 四步驟，當運作時間超過 5 秒時，就將最佳的一個棋子落點位置輸出給 judge。

## 三、 綜觀整個 program 的執行過程

每次對戰的第一步棋，如果是我方先下，則下在棋盤正中央 108 號位置；選擇此位置的原因是：對於整個棋盤而言，此位置的「自由度」最高，周圍六個方向都能自由地「延伸」出去。

另外，整個 program 將分為兩個執行緒同步運作：thread 1 是負責處理 text I/O，當成與 judge.exe 的溝通介面；thread 2 則是整個 MCTS。如此的設計，可以讓 game AI 可以進行模擬的時間更加的充裕，不會為了等待 I/O 而暫停模擬的工作。

## 四、 關於 team project 的心得

完成本次作業可以說是路途曲折，從決定要以 MCTS 來當作下棋策略開始，首先遇到的問題是怎麼設計適當的 data structure，以避免接下來 expand 還有 simulation 時，會儲存太多重複的資訊而消耗大量記憶體。接著，設計出良好的 heuristic function 也是下了不少功夫，雖然說之前作業曾考慮過 tic-tac-toe 的 function，但是本次的棋盤位置數比九宮格大上太多，仔細探究之後就能察覺到有巨大的差異存在。最後，建議應提供一個能在 Linux 環境執行的 judge.o，至少能在系計中的 Arch 4.14.13-1 、FreeBSD 11.1-release-p9 運行，以減少為了驗證程式而需要切換執行環境的困擾。

## 五、 執行環境

- Environment_1：

OS: macOS

Compiler and version: Apple LLVM 9.1.0

- Environment_2：

OS: Linux Arch

Compiler and version: Clang C++ 6.0.0

- Source Code：

https://github.com/hare1039/Quintuple-Go