# Report

R05922156 黃子瑋

## 1.How to compile

環境: windows 10

Compiler: GCC 5.3.0

Compile 方法: 直接 make 就可以了

因為我是在 windows 下用 GCC 編譯，所以有修改 main.cc 的 160 行及 167 行的參數

## 2.What I have implement

## 3.The heuristic I use

## 4.motivation

**上面三項我放在一起來說明:**

(1) NegaScout:

動機是因為這是這次強制規定使用的演算法 XD

大致上與講義中老師教的 NegaScout(NegaMax formed)差不多

至於搜尋的深度我有做點小小的判斷

剛開局會搜很淺(大概四層左右)然後越接近終局搜尋的深度會越深

(2) Hash Table:

這個也是強制規定要使用的

所以也是跟講義上的 Transposition table 差不多

這邊用到的 Zobrist 隨機數我用了 64-bit

然後開了 [棋子種類]*[位置]還有 color[2]的陣列去存這些隨機數

計算的時候採用 XOR(比較快)，整個 table 的大小我用 2^25 個 entry

本來是打算開 2^32 個 entry 的但是我的電腦 memory 不夠所以無法

一樣有做點小小的判斷，就是剛開局的時候不做 hash

到中盤可能要開始會有比較深的計算的時候才 hash

不過其實好像也沒省到多少時間 XD

(3) 審局函數:

我的審局函數大致分成三個部分:

棋子權重:

這邊就用最簡單的靜態加權

帥最大然後兵最小(炮大概在象跟車之間)

動態權重:

本來有加入動態權重

例如在對方已經沒有帥跟兵的情況下

卒的分數會變成 0

但是實作完後不知到哪裡出了問題

會變成亂走

我覺得很有可能是因為把某個權重變成 0 後因為那個東西會等於直接被忽略

導致本來有的可以走的不會搜不到

所以後來還是決定用靜態的~

相對距離:

就是算出目前盤面上所有已翻開棋子之間的 manhattan distance

搜索深度:

雖然是深度的概念但我是用一個簡單的變數去記錄總回合數

每過一手就+1 ,然後記錄每一個棋子在第幾回合死掉

用這些數據去算出審局函數，越早把對方吃掉可以得越高分

因為這樣可以避免一種情況：若是走１步跟走１０步都可以得到同樣的分數，他會亂走

而加入這個深度可以解決此問題

(5)翻棋策略

我翻棋採用的策略是會去看目前檯面上所有已經翻開的棋子

然後以此去計算為翻開的棋子的安全機率

例如:


在上圖這個情形下，若我方是紅色

1 號位置只有翻出紅帥時才安全若翻出其他的都是危險的

所以這個位置的安全機率就是 1/27

2 號位置可以翻出所有的紅色加上黑色的車馬包卒，所以安全機率就是(13+2+2+2+5)/27 = 24/27

依此類推算出所有暗棋的安全機率去判斷該翻哪個

(4)寧靜搜索:

其實我不太知道我這樣有沒有算寧靜搜索

本來是打算在整個城市中都使用寧靜搜索，但是因為時間上的問題來不及完成

所以最後只在一個特定情況下會做寧靜搜索

但是在那個情況下我也只是往後多看一層而已

雖然有用到同樣的概念但是沒有真的搜到底

所以不知道算不算ＸＤ