Group Game Project #1 Report

Team number: 5, Team name: tmp
ID & name: 0716206 陳昱丞（Leader）、0716221 余忠旻、0716202 張喬暢

Minimax with alpha-beta pruning:

我們利用 minimax 搜尋演算法設計這次的遊戲 AI。而我們使用的是 depth 3 的結構。depth 0 是目前的 state，depth 0 -> depth 1 選出 MAX，depth 1 -> depth 2 選出 MIN，depth 2 -> depth 3 選出 MAX，透過 DFS 搜尋和遞迴技巧將 depth3 用 evaluation function 算出來的值回傳到 depth 0。和一般的 minimax 不同的是，我們不只是採用 depth 3 的值，我們也會評估 depth 1 的值，並將兩者相加，且 depth 1 的比重較重。這是為了配合我們 evaluation function 的設計。我們的 evaluation 是判斷當手棋和周圍自己棋的連線關係，而不是評估整個 board 的分數。下一個部份會有更詳細的說明。我們主要是以 depth 1 的評估值為重，depth 3 的值作為當兩個 state 的 evaluation 值一樣時的 tie breaking。因此，depth 3 的值永遠不會超過 depth 1 的值，因為當下的影響永遠會比未來重要。接著就能根據 minimax 算出的最佳移動方式移動（return GetStep）。另外，我們還用 alpha-beta pruning 來增加程式執行效率。

Expansion:

在計算 node expansion 之前，我們要先想辦法儲存一個 node state。為了節省空間和增加計算效率，我們將 state 和其附帶資訊包裝成一個 bit string，使用 python 的 bitstring 套件進行實作。一個 state string 結構如下圖所示:

board part:

整個三維的 board 總共有 36 x 6 = 216 個格子，而每個格子有角落、空白、黑棋、白棋四種情況，因此需要 2 個 bits 儲存，board 部分總長就是 432 bits。令角落格為 11、空白格為 00、黑棋為 01、白棋為 10。

height part:

將 24 個 cells 的 height 包裝在 state string 裡面，這樣在對每個 state string 做 expanding 的時候，就不用再統計每個 cells 的高度。方便計算也能增加執行效率。height 有 0 到 6，需要 3 個 bits 進行儲存，因此 height 總長 3 x 24 = 72 bits。

position part:

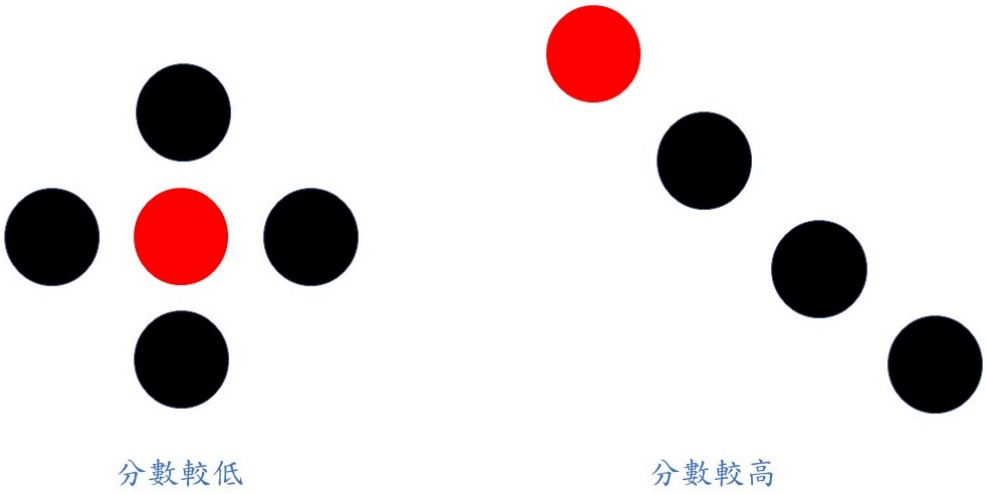
position 是儲存此 expanded state 下的棋是在 board 中的什麼位置。儲存此資訊是為了下方的 evaluation 會用到。432 個位置需要 9 個 bits 來儲存，因此 position 總長為 9 bits。

定義了 state node 的儲存方式後，接著就可以計算他的 expansion。先考慮一步棋會有 24 個 cells 可以下，用 for 迴圈迭代所有下法，迭代前先判斷該位置的高度有沒有到 6 顆棋子。若沒有，則該位置為 valid move。在 board 上找到你要下的 position，將 position 改成你的棋子(黑為 01、白為 10)，接著將對應的 height 值加 1，最後再將修改過後的 board、height、position 打包成一個新的 bit string，append 進 list 中。過程中不需要涉及資料的儲存，全部都僅需要純數學運算便可完成。藉由此方式可將時間消耗量縮到最小。

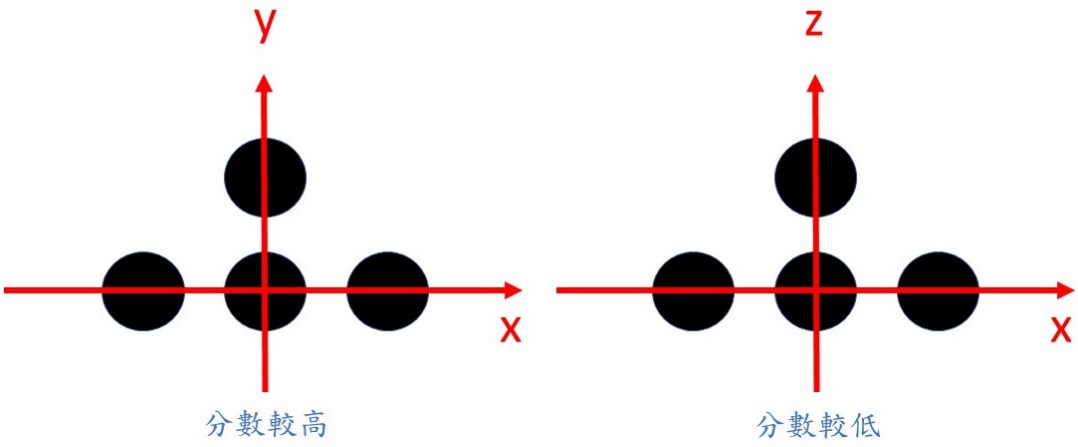
Evaluation:

我們評估一手棋的分數，是計算該手下下去的那顆棋子和周圍的自己棋子能造成多少連線，造成愈多條連線的分數愈高。連線中又分兩顆連線、三顆連線、四顆連線，其中四顆連線的分數最高，遠高於兩顆和三顆連線(下圖一)，因為遊戲規則為先搶到四顆連線的一方會有巨大優勢。

而連線的方向也會造成影響。我們會優先考慮 x-y 平面的連線，有 z 方向向量的連線分數會被打折(下圖二)。因為此遊戲有 stack 性質，要完成有高度的連線會有一定難度。



圖一



圖二

Experiments and experiences:

最後是實驗的部分。這裡我們主要分析防守策略，攻擊方(黑)皆是用第一部分的 minimax 做出分數最高的選擇。防守方(白)則假設自己同樣為黑棋、同樣做 minimax 找到黑方最好的下法，並下該手棋以阻擋黑方。但若整場都在阻擋，防守方將很可能陷入整場無法連線的窘境。因此，我們調整策略，防守端前期進行阻擋、後期進行進攻。下表是攻防比例和勝負之間的關係:

策略	勝負	比分 (攻 : 防)
0 步防 32 步攻	Loss	262 : 190
10 步防 22 步攻	Win	153 : 224

15 步防 17 步攻	Win	118 : 236
20 步防 12 步攻	Win	69 : 265
25 步防 7 步攻	Win	86 : 229
32 步防 0 步攻	Loss	110 : 108

由上表可以發現，若完全不防守，先攻方將佔有優勢，最後贏下比賽。而若前 10 步進行防守、後面進攻，防守方最後會獲勝。這證明了防守方前期先防守會對比賽勝負造成相當大的影響。但若全部整場進行防守則會以些微落差輸掉比賽。最後我們決定以贏最多分的 20 步攻 12 步防作為最終的防守策略。

Contribution:

陳昱丞: expand 函式開發、主體策略構想、Report 撰寫
余忠旻: evaluation 函式、minimax 函式開發
張喬暢: Report 撰寫