Search strategy

Board size = 4

Search method: minimax with α-β pruning

Deep:

前幾步30秒內無法search到leaf，所以一開始的深度只有3，之後再慢慢增加，最後三步可以完整search完。

Search tree:

有一個min跟max的function，要AI下棋時，會先進max的function，選擇下一棋後，再進min的function，模擬user下一棋，直到雙方都沒剩棋子。兩個function中分別用一個min跟max變數存目前的最大最小evaluation value。如果最後一動是AI下完，回傳最小值給max，max再下另一動，一直重複同樣動作直到限制深度內全部的動作都看過。最後一動是user下完也是同理。
樹的expand是將所有能用的棋，以隨機的開始位置在棋盤上有空位的地方一個一個擺放。

Evaluation function:

point = (sum(AI_chess)) - sum(User_chess))*2
+ Σ chess_wight in AI - Σ chess_wight in User
+ Σ chess_available_weight in AI - Σ chess_wight in User
+ penalty for 13

以現在版面上AI有2,8，user有5來舉例
1. AI_chess為現在版面上AI有的棋子，User_chess為現在版面上user有的棋子。乘以二是因為這是贏的首要條件，權重要大一點。以範例來說這裡就是(2+8-5)*2
2. Σ chess_wight in AI - Σ chess_wight in User 這行的目的是因為當最終分數同分時，留有較大棋子的一方獲勝，所以將較大的棋有較高的權重。2的權重是0，3的權重是1，5的權重是2，8的權重是4，13的權重是7，3的權重加5的權重會小於8的權重。以範例來說這裡就是(0+4-2)
3. Σ chess_available_weight in AI - Σ chess_wight in User 因為30秒內前幾步時沒辦法走完完整的樹，所以手上來有的棋也要加近考量因素。chess_available_weight的值皆為chess_wight的一半。以範例來說這裡就是(1/2 + 2/2 + 7/2 - 0/2 - 1/2 - 4/2 - 7/2)
4. penalty for 13是因為經驗上來說若出13時，對方手中還有8以下的棋，很容易就被很小的棋吃掉，所以在出13但user手中還有8以下的棋時，會給penalty，抑止AI太早或不適當的出13。

Board size = 6

Search method: MCTS + minimax with α-β pruning

在6x6中因為search tree更為龐大，若一開始就用minmax，30秒內search的深度太淺，在前幾步中便會下笨棋，所以在前6步我用了MCTS，因為6x6的棋盤太大，前幾步並不會直接影響遊戲的輸贏。為了不讓13考慮不周的情形下出現，MCTS中我先將13拿掉，在後面的minmax中才加入計算。

MCTS

Selection:是根據UCB公式，weight=1

Simulate : 隨機找棋跟空位子擺，直到走到底(雙方其用完)。

Back propagate:走到底後計算版面上的兩方總分數差作為分數，然後沿著path更新回去。每次Back propagate分數都會累加

Simulate的次數為，做到25秒的次數。最後選擇平均分數(分數總和/visit次數)最高的node作為下一步。

Minimax

Evaluation function 和4x4的一樣，search深度是從4持續加深。