R12922167 郭恩銘

1. 2. What algorithms and heuristics you've implemented & Details of the evaluation function:

我做的是如作業說明要求的 negascout + star 1 + transposition table
negascout 與 star 1 我是照投影片的公式寫的，詳細地說要接起來只是把投影片第七章第 29 頁 F4 呼叫 – F4() 的地方換成 – star_1() ，star_1() 裡面呼叫 F4() ，這樣改投影片上的 code 正負就對的起來。

比較複雜的是 transposition table ，我拿來做 hash key 的是版面每個棋子的位置與當前版面擲到的骰子，但這會有一個問題就是假如版面棋子只剩 1 跟 6，2.3.4.5 應該是用同樣的方式算出一樣的值，而且這個值和 1 .6 有關，但很明顯的 2.3.4.5 不是同一個 hash entry，這樣就會一直重複計算。所以我在 hash table 檢查完有沒有 hit 後，如果沒 hit 還會再看該骰子骰到的是不是死掉的棋子，是的話會檢查可以移動的棋子，例如上面情況算 2 時我會從 1 和 6 的結果挑比較大的出來，並把他更新到 2 的 hash table 裡。(這又衍伸了一個問題就是 1.6 要先做，為了解決這個問題，我程式會根據存活的棋子決定 star_1 的骰子 1-6 要以什麼順序做才會最順，實際上就是建表儲存 64 種存活的可能性的 best dice seq)

我 hash table 開 2 的 23 次方大，也就是說 hash index 是 23 個 bit，版面 encode 的 hash key 用兩個 unsigned long long int 有 64 * 2 = 128 個 bit，具體的觀察也是會在最下面解釋

接下來是審局函數，我的審局函數有點複雜，分成兩個部分(以下都以紅色方為準說明，藍色方同理)
第一部分是棋子的位置分數，按位置來看分數大約是這樣的比率 (但打 random 我會把離終點分數設高一點 XD)

1	1	0	0	-1
1	5	5	5	5
0	5	10	10	10
0	5	10	16	16
-1	5	10	16	MAX

但這不是直接拿來看棋子在哪個位置就加幾分，我會把該位置分再乘以棋子目前的 determinacy (就是如果只剩 3.4 兩個棋子，3 在骰子 1.2.3 都可以動，所以他的 determinacy 是 3，同理 4 的 determinacy 也是 3)，然後加總所有自己活著的棋子的該分數，最後再乘以一個第一部分 scale

第二部分是棋子的平均 determinacy 分與平均剩餘棋子分，我發現不加這個會變成所有棋子平均地緩慢向右下靠攏，但我想鼓勵我的棋子去吃自己的棋子，所以我會希望平均 determinacy 分愈高愈好，剩餘棋子愈少愈好 (但從 2 個棋子到 1 個棋子分數反而會變低，因為我覺得這在走鋼索)。第二階段同樣也會乘以一個第二部分 scale。

參數是平均 determinacy

1	1.001-2	2.001-3	3.001-4	4.001-5	5.001-6
1	3	5	8	9	9

剩餘棋子個數

1	2	3	4	5	6
9	9	7	5	3	1

除了這兩個部分我還加了一點巧思，因為同一個盤面不同玩家得到的審局分不會一樣，所以我把我的審局函數額外加上 ”下一個動的玩家如果可以吃對方的子，那他可以多得吃掉的最好的子的原始審局分數 * 吃掉的概率 ”。舉例來說對方的先鋒提供了對方 500 分的 PART1 score，但我方在骰子骰到 3.4 時可以吃掉他，那我方就可以額外獲得 500* 2/6 的分數。加了這點會讓我的先鋒棋子不太會冒險地走對方能大概率吃掉他的路線。

我的審局函數差不多就是上面說明的這些，但為了要快速計算剩下哪些棋子時每個棋子的 determinacy，這部分我一開始也會建表快速查找。

3. Discuss benefits of various enhancements:

我做的優化基本上都是像測試當天學長說過的，棋子在什麼位置時可以走哪幾個方向可以建表儲存起來之類的東西，統整起來有以下的優化;

1. 我會根據 64 種棋子存活的情形，建表儲存每種情形分別在 6 種骰子情況下是動哪個或哪兩個棋子。
2. 棋子在什麼位置可以動最多哪三種方向也建表儲存起來，並且把走斜對角的方向設成第一個走步。
3. 為了審局函數需求，我還建表記錄 64 種棋子存活的情形中，各情形下每個棋子的 determinacy
4. 骰子的順序我不是用 1-6 按順序找，而是假如 2 4 5 6 棋子存活，我會先去做存活的棋子 (ex : 2 與 4)，但我不是一口氣安排全部存活的棋子，而是在 2 做完後把 1 也安排到 2 後面再做 4，然後 4 做完後先做 3 再做 5.6，如此可以確保 transposition table 有很高的 hit rate，這樣 star 1 就能有效率的做很多骰

子，快速增加 lower bound 與 upper bound。

具體來說，優化大概能讓我的程式在 Mac M2 pro 上，以 18 秒左右的時間搜開局的 7 層，到中期 7 層大概是 5 秒，後期 1 秒不到就能跑完 。不過因為開局 4 步如果都搜 18 秒會超時，所以我的時間控制是每一步除了基本時間還有大約 10 秒的額外思考時間，如果前一步超時太多下一步的搜尋深度就是 6 層，如果前一步用很少時間下一步就還原成 7 層。而 6 層都可以穩定在 3 秒內結束。

不做優化我的程式連 6 層都有困難，所以預先計算真的很重要 XD
此外，我發現我 hash table 開 2 的 23 次方還是會一直 hash collision，
但這是 mac data 區段的極限大小了，而因為如果我把 hash table 開在 heap 我不知道會不會和 stack 撞在一起導致 crash 所以我就沒開更大，不然 hash collision 好像還滿可惜的 (不過我倒是有想過做 chaining 或 open addressing，但後來都在弄審局函數就只能作罷 QQ)。

4. Experiment results and findings of your implementation:

這部分我想就直接說明我測最多的審局函數部分(說明與 random baseline 比較的結果)，前面已經提過一些參數設定了這邊就額外提兩個發現:

我一開始設定的 position score 是按離終點的距離算的 ，從距離終點遠 (4) 到近 (1) 大約是 0 : 5 : 10 : 15，但我發現這麼做的話棋子會走到我不希望的地方，例如:
開局在 A3 的棋子不知道為什麼永遠只會走 A4 因為他不想被吃掉。
或是在 A4 的棋子寧可走 A5 也不願走 B5，但最左下角和最右上角的格子我覺得完全沒有任何意義，因為棋子走到那邊基本上沒有人會吃他離終點又有 4 步那麼遠，結果又骰到他一遍他就又從 A5 走到 B5，與其這樣不如一開始就走 B5 還比較容易直接衝到終點。

(重搬一次上面的表)

1	1	0	0	-1
1	5	5	5	5
0	5	10	10	10
0	5	10	16	16
-1	5	10	16	MAX

所以我把 A5 和 E1 設了一點 penalty，這樣至少可以有效遏止自己的棋子無意義地走到這兩個點上，改了這個打 random 50 場大概可以多贏 2-3 場，畢竟 random

吃掉自己前鋒的機率不會太高 (就算骰到可以吃自己前鋒的棋也只有大約 1/3 的機率會真的吃掉)，就可以賭一下讓自己棋子往前衝

此外，我還有試過不同的審局 PART1 與 PART2 權重，但我發現好像 1 : 2 會是最好的，PART1 的比率也最好貼近 0 : 5 : 10 : 20，把離終點的分數再設更高會導致 PART2 score 被稀釋掉，然後棋子就不會吃自己而是集體往前送頭，但每個棋子的 determinacy 根本沒有高到可以做這種挑戰。結果打 random 就會一口氣多輸 5-10 場，所以我覺得每部分的 score 都還滿重要的。

最後再放上今天比賽的心得，我發現我把防守點位的棋子分放太高，所以我的棋面就會變成被動防守，骰運不好沒有守到對面的先鋒就會直接被打穿。但我試著改變棋子位置的分數結果動了一下(調低防守點位分數,調高終點分數) 審局分直接炸開，棋子不知道為什麼完全不前進連輸對手 7 場，算是我審局寫了太多參數又沒好好調整的缺陷。

然後今天好像都是我再燒線前三十秒每次都搜七層拖大家的時間 (每場最後都是只剩我和對手在對打就被放在台上檢視，可以看到我的程式開局想超久)，真的非常抱歉 Orz。也謝謝教授和助教這個學期能開這麼有趣的課，自己寫的 AI 能成功跑起來還跑得像個人類真得很有成就感。可以的話希望這堂課也能開成上下學期的課，不然我覺得比一次賽就結束滿可惜的，學完整學期的課後應該能做不同隨機遊戲的 AI 寫不同的審局函數來比一個學期 XD。

以上就是我的心得，再次感謝教授與助教 Orz