

Kalah-Game

黃泰揚 00857114

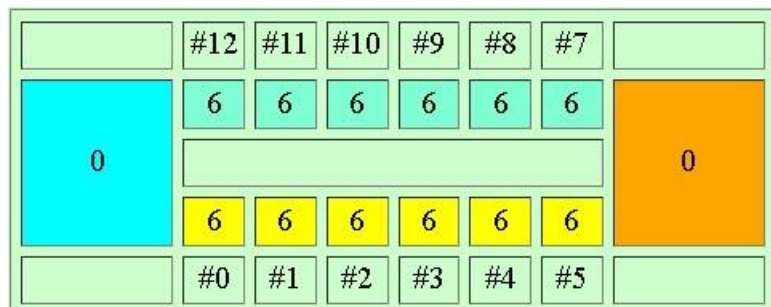
Kalah 問題描述/規則說明: <https://en.wikipedia.org/wiki/Kalah>

定義狀態空間:

將原始遊戲狀態轉換為陣列 `house[14]`，其中 #0 ~ #5 的洞(圖中翻譯為:“房子”)由 Player0 控制，#7 ~ #12 的洞由 Player1 控制，並將 `house[6]` 視為 Player0 的倉庫(圖中翻譯為:“商店”)，而將 `house[13]` 視為 Player1 的倉庫。

該圖為 $n=6$ 的初始狀態，每個洞裡會有 n 個種子:

EX:



狀態轉移:

規則 1. 玩家會輪流播種。在一個回合中，玩家從他們選的一個洞中移除所有種子。逆時針移動，輪流在每個洞裡丟一顆種子，包括玩家自己的倉庫，但不包括對手的倉庫。

EX:



規則 2. 如果最後播下的種子落在玩家擁有的空洞裡 (播種前)，而對面的洞裡有

種子，最後一顆種子和對面的種子都會被捕獲並放入玩家的倉庫，這個 action 稱作“空捕獲”。

EX:



注意:如果最後一粒種子落在對手的空洞裡，“空捕獲”不會發生

規則 3. 如果最後播下的種子落在玩家自己的倉庫中，玩家將獲得額外的移動機會。且玩家可以輪流進行的移動次數沒有限制。

EX:



目標狀態:

當一名玩家的任何洞中不再有任何種子時，遊戲結束。另一個玩家將所有剩餘的種子移到他們的倉庫，倉庫中種子最多的玩家獲勝。

EX:



根據上述規定，定義出種子遷移轉移函式:

```

bool relc(int h[],int pk){
    //判斷一個state經過一個action後的新state
    //輸入:h[]=原始陣列(狀態空間),pk=action(選取的種子)
    //輸出:boolean 表示回合是否連續,並且將輸入的state更改為action後的state
    int tmp=h[pk],idx=pk,ck=0,j=0,r=h[pk];
    for(;--r>0;++h[idx]){
        (++idx>13||(pk<6&&idx==13))?idx=0:(6<pk&&idx==6)?++idx:0;
        for(++idx%=14;1;++j)
            if(pk<6||j){
                (pk<6)?idx%=13:(idx==6)?++idx:0;
                (idx==j b)?ck=1:(j*7<=idx&&idx<j b&&!h[idx]&&h[12-idx])?h[j b]+=h[12-idx]+1,--h[idx],h[12-idx]=0:0;
                ++h[idx],h[pk]-=tmp;
                break;
            }
        for(int j=0,tf,i;j<2;++j){
            for(i=j*7,tf=1;i<j b;(h[i++])?tf=0:0);
            for(i=(j^1)*7;tf&&i<(j^1) b;h[i++]=0)
                h[(j^1) b]+=h[i];
        }
        return ck;
    }
}

```

實作細節:

1. 可以藉由輸入的 pk 判斷目前是哪邊的玩家在做動作。
2. 為判斷是否發生“空捕獲”，先用 idx 確認播種前的落點座標
3. 將判斷遊戲是否結束另外列為一種 state 較好實作。

Goal test 函式:

```

bool gend(int h[]) {
    bool ed1=0,ed2=0;
    for(int i=0;i<6;++i)
        ed1+=h[i],ed2+=h[i+7];
    for(int i=0;(!ed1||!ed2)&&i<6;++i)
        h[6]+=h[i],h[13]+=h[i+7],h[i]=h[i+7]=0;
    return (!ed1||!ed2);
}

```

實作細節:

用 ed1、ed2 來判斷是否結束並順便將沒放完的種子放入雙方倉庫。

用 minimax 來計算人機的策略:

```

int mnx(int h[],int d,int turn,int plr){
//判斷在某狀態下某玩家的選擇
//輸入:h[]=原始陣列,d=當前深度,turn=決定回傳max或min decision,plr=目前做決定的玩家
//額外輸入:dMAX=最大深度
//輸出:v=做出該抉擇後在到深度上限/遊戲結束時得到的最大分數
//額外輸出:ans=最終抉擇
if (gend(h)||d>=dMAX)return ((!stp)?h[6]-h[13]:h[13]-h[6]);
int v=((turn)?-99:99),sh[14];
for(int i=0;i<14;++i)
sh[i]=h[i];
for(int i=0;i<6;++i){
if(!sh[i+plr*7])continue;
v=((relc(h,i+plr*7))?(turn)?max(v,mnx(h,d+2,turn,plr)):min(v,mnx(h,d+2,turn,plr))):((turn)?max(v,mnx(h,d+1,turn,plr)):min(v,mnx(h,d+1,turn,plr)));
if(!d&&v>amv)
ans=i+plr*7,amv=v;
for(int j=0;j<14;++j)
h[j]=sh[j];
}
return v;
}

```

實作細節:

1. 用 turn 決定目前的 decision 要取 max 或取 min 可以避免兩個函式分開寫。
2. 由於 dMAX(搜尋深度上限)不會產生變化，所以將其定為全域變數。
3. ans(最終選擇)的結果相當於第 0 層(d=0)的最大值，由於限定的層數不方便參數傳遞所以也設為全域變數處理。
4. 遊戲規則中的“連續回合”，相當於[d+2,turn 不變,plr 不變]。
5. d(層數)和 turn 不可合併(由於 4.)。
6. 當某一個洞是空的，它不能選(continue)。
7. 終止條件有兩種:1.遊戲結束(Goal test=1) 或 2.層數到達上限(d=dMAX)

aB 剪枝:

```

int mnxab(int h[],int d,int turn,int plr,int A,int B){
//minimax aB剪枝版
++node;//計算總個節點數
if (gend(h)||d>=dMAX)return ((!stp)?h[6]-h[13]:h[13]-h[6]);
int v=((turn)?B:A),sh[14];
for(int i=0;i<14;++i)
sh[i]=h[i];
for(int i=0;i<6;++i){
if(!sh[i+plr*7])continue;
if((!sh[i+plr*7])||((turn&&v>=A)||(!turn&&v<=B))continue;
v=((relc(h,i+plr*7))?(turn)?max(v,mnxab(h,d+2,turn,plr,A,v)):min(v,mnxab(h,d+2,turn,plr,v,B))):((turn)?max(v,mnxab(h,d+1,turn,plr,A,v)):min(v,mnxab(h,d+1,turn,plr,v,B)));
if(!d&&v>amv)
ans=i+plr*7,amv=v;
for(int j=0;j<14;++j)
h[j]=sh[j];
}
return v;
}

```

實作細節:

1. AB 篩掉當前不可能的情況。

實驗內容描述:

1. 使用 minimax search algorithm 來製作一位檢金豆 AI 人機。

EX 輸入 1 然後輸入 0 0 後的遊戲畫面:

```

C:\Users\User\OneDrive\桌面\AI期末專案.exe
請輸入模式(1=與AI人機對戰/2=看人機互打/3=分析同狀態下的node拓展數/other=離開):1
選擇AI人機難度(0=隨機選/1~12=人機預判回合數/other=離開)和開始遊戲方向(0下/1上):0 0
回合:1

-----棋盤-----
倉庫1  #12 #11 #10 #9  #8  #7  倉庫0
      4   4   4   4   4   4   4
0 ----- 0
      4   4   4   4   4   4
倉庫1  #0  #1  #2  #3  #4  #5  倉庫0

輪到玩家回合
選擇我方的洞(0-5):4
你選了#4後棋盤狀態:

-----棋盤-----
倉庫1  #12 #11 #10 #9  #8  #7  倉庫0
      4   4   4   4   5   5
0 ----- 1
      4   4   4   4   0   5
倉庫1  #0  #1  #2  #3  #4  #5  倉庫0

人機選了#9後棋盤狀態:

-----棋盤-----
倉庫1  #12 #11 #10 #9  #8  #7  倉庫0
      5   5   5   0   5   5
1 ----- 1
      4   4   4   4   0   5
倉庫1  #0  #1  #2  #3  #4  #5  倉庫0

回合:2
輪到玩家回合
選擇我方的洞(0-5):

```

系統設計:

先輸入模式

模式 1 和人機對戰

輸入 人機等級和玩家選邊

人機等級為 0-12 的數字，0 代表隨便選/0 以上代表 minimax 的搜尋深度。

玩家選邊請參考定義狀態空間，並且玩家 0 總是先攻。

模式 2 人機互打

輸入 兩個人機的等級，配置同上題

接下來兩位人機會使用剪枝後的 minimax 互相對戰 1000 場，最後輸出每場結果/分數，此外，當兩位人機的等級都>0 時，只會對戰一場，因為結果都一樣。

模式 3 分析拓展數

輸入 狀態/玩家/深度

例如: 6 6 6 6 6 6 0 0 7 7 7 7 7 1 1 4

代表 0 7 7 7 7 7

0 1

6 6 6 6 6 6

的情況下輪到玩家 1 行動，求深度搜尋到 4 層的搜尋節點數。

該模式不會自動跳出，可以一直輸入。當出現” 程式結束” 時代表程式碼執行結束，需要再次重開。

此外，程式有做基本的防呆處理，基本上出現提示訊息後再次輸入即可。

2. 並且證明比起隨意猜測，該人機具有更高的勝率。

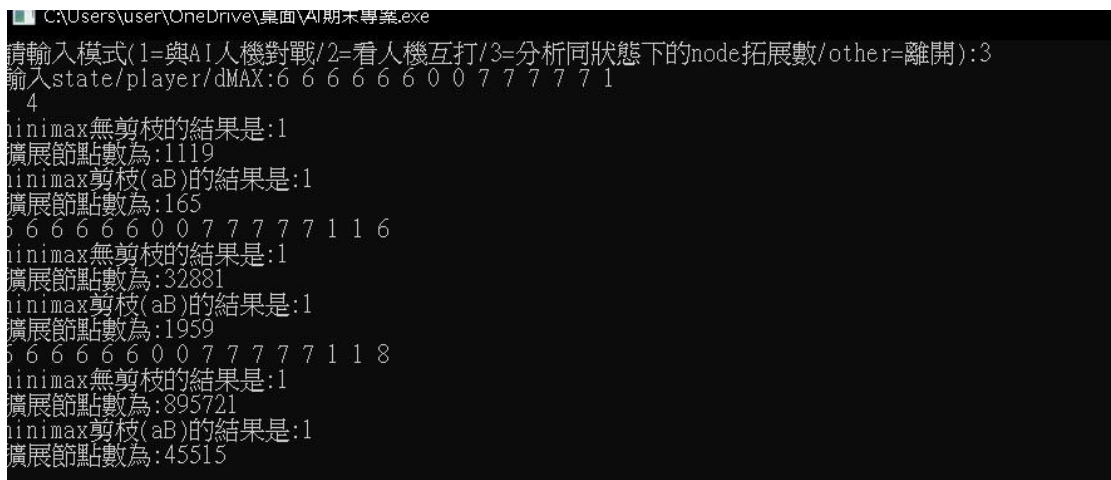


結果:玩家 0 分數:83 / 玩家 1 分數:929 (隨機猜打預測 1 層)

分數計算方式:誰贏加 1 分，平手的話各加一分，打 1000 場。

此外，隨機猜打預測 5 層的成績為玩家 0 分數:59 / 玩家 1 分數:948 可以看出預測層數越多，勝率越高。

3. 分析剪枝的拓展節點數差異&分析人機在不同搜尋深度下之性能差異:



可以看出 aB 剪枝的結果非常顯著，且隨著深度遞增，拓展數呈指數成長。

另外，由於較深的層數搜尋時間過長，為了更好看出指數成長，這邊提供幾筆測試結果表格:

State	Cutoff	Without Pruning	Pruning
[0 2 3 2 1 13 17 11 0 2 9 0 0 12] Player 0	8	139078	9342
	10	2646950	78466
	12	49667211	405370
	14	--	3121954
State	Cutoff	Without Pruning	Pruning
[7 2 15 0 1 0 5 0 5 7 6 1 14 9] Player 0	8	179129	7024
	10	4147291	84913
	12	93977634	891849
	14	--	5886505
State	Cutoff	Without Pruning	Pruning
[3 0 11 0 0 1 15 1 0 10 3 0 12 16] Player 1	8	104508	10223
	10	1899484	92197
	12	33749160	589564
	14	--	3305880
State	Cutoff	Without Pruning	Pruning
[11 0 2 2 2 3 9 1 0 2 12 2 12 14] Player 1	8	232376	10785
	10	4567908	81570
	12	88554078	679559
	14	--	5320541

結果討論與實驗心得:

該實驗的靈感來自於隔壁班人工智慧的作業，原本題目為給一個狀態/玩家/搜尋深度，回傳一個最大選擇；這個部分我為了練習 minimax algorithm，所以跟著寫完了。

但為了增加專案完整性，我補充了許多的功能，例如 aB 剪枝、人機對戰、計算拓展節點數等，做完研究後更加了解了該演算法以及剪枝的用法/效果。