電腦對局理論（秋 2018）

期末專題 - 暗棋大賽

資工所碩二 劉又萱 R06922041

壹. 執行與編譯

- **編譯**
  - **Windows 系統:**
    - ✓ **安裝 mingw, cmake 即可使用 make 指令編譯**
    - ✓ **Compiler:** gcc 5.2.0 (Rev3, Built by MSYS2 project)
    - ✓ **make 指令:**先將./source/Makefile_windows 檔案重新命名為 makefile 後，下“make”即可編譯
    - ✓ **執行環境:** Windows10, Intel Core i7-8700 CPU, RAM:32GB,GPU:GeForce GTX 1080 Ti X2
  - **Linux 系統:**
    - ✓ **Compiler:** g++ 7.3.0 (Ubuntu 7.3.0-27ubuntu1~18.04)
    - ✓ **make 指令:**在 source 資料夾內下“make”即可編譯
    - ✓ **執行環境:** Ubuntu
- **執行**
  - **Windows 系統:**
    - ✓ **放置執行檔:**將編譯好的“search.exe”放入 CDC_package_windows_3.2/CDC_interface/CreateRoom/Search 內，或是 CDC_package_windows_3.2/CDC_interface/EnterRoom/Search。
    - ✓ **跑暗棋程式:**執行 CDC_package_windows_3.2/CDC_interface/CreateRoom/DarkChess.exe 即可開房(使用其 Search/search.exe 作為 AI)，並可以執行 CDC_package_windows_3.2/CDC_interface/EnterRoom/DarkChess.exe(使用其 Search/search.exe 作為 AI)，來進行雙 AI 連線對戰。
  - **Linux 系統:**
    - ✓ **放置執行檔:**將編譯好的“search”放入 CDC_package_linux_3.2/CDC_interface/CreateRoom/Search 內，或是 CDC_package_linux_3.2/CDC_interface/EnterRoom/Search。
    - ✓ **設定暗棋相關參數(如時間、循環盤面次數、對戰次數等):**

執行
/CDC_package_linux_3.2/CDC_interface/CreateRoom/DarkChess.exe 0

- ✓ **跑暗棋程式:**執行
./CDC_package_linux_3.2/CDC_interface/CreateRoom/DarkChess.exe 1 即可開房(使用其 Search/search 作為 AI)，並可以執行 CDC_package_linux_3.2/CDC_interface/EnterRoom/DarkChess.exe 1(使用其 Search/search 作為 AI)，來進行雙 AI 連線對戰。
- ✓ **看下棋紀錄:**執行
/CDC_package_linux_3.2/CDC_interface/CreateRoom/DarkChess.exe -r ./Search/board.txt

貳. 使用演算法

- **搜尋演算法**
  - ✓ **動態搜尋時間 time using rules (main.cc 行數:835)**

    搜尋時間=(剩餘時間)/(預計回合數-現在回合數+1)，如果剩餘時間勝太少，會將搜尋時間縮更短。如果只有一步可以走，則無需進行搜尋。

  - ✓ **主演算法 - NegaScout (main.cc 行數:559)**

    會依照每個盤面可走的步+ 空步/翻棋(null move pruning)來進行分數的搜尋。加入空步是因為有可能你不動棋子，取而代之的是翻棋或許分數會比較好，因此一併考慮進去。

    一開始，會先判斷該盤面是否已經存在於同形表中，如果存在的話就拿取其值最對應的 alpha、beta、m 更新，並先搜尋該同形表節點存的最佳著手。接者，利用 null/Zero window search 來搜尋，如果搜到結果比 m 大才需要 research、如果搜到的結果大於 beta 則即可 cut off。最後，當被 cut off 或是搜尋完後，把結果更新至 hashtable 中,以便未來搜到同一個盤面時,節省搜尋時間。值得注意的是，如果在深度為一的時候搜到 WIN 即可回傳結果，因為自己(maxNode)，已經找到一條必勝之路。

  - ✓ **搜尋翻牌演算法 - Chance Search (main.cc 行數:720)**

    由於翻牌翻出的棋子含有機率成分，而且是不能控制的，因此當搜尋某個位置是否要翻開的時候,必須要透過 NegaScout 得到該位置每種情況分數的期望值,並且最終選擇擁有最大期望值的位置來翻棋。值得注意的是，此最佳分數必須跟主演算法搜出來的分數做比較，如果有高於主演算法，表示翻該位置的棋子有可能獲得更

高分。

- ✓ **如何決定是否走棋或翻棋 (main.cc 行數:916)**

  當主演算法得到的結果為空步或者翻牌演算法得到期望值大於主演算法，則採用翻棋，不然就是走棋。這個設計的原因在於翻子其實充滿機率，即便得到的期望值可能比走子來得高，但是突然翻到一個比期望值低的子也是有可能的，反而造成分數墜落。

- **儲存拜訪過節點 - 同形表 transposition table**
  - ✓ **ZobristKey (anqi.cc 行數:448,482)**

    由於要建立同形表，首要條件就是要找到一個好的標號方式來標示盤面，且具由唯一性，同時必須兼具更新快速的特性，故選擇使用 ZobristKey 來進行編碼。

    首先，要生成每個棋子(包含未翻子以及空位)對應到盤面上每個位置的隨機數字(2^64bit)，對於暗棋而言，會開一個陣列為[Type of Fin][Board place]的 2^64bit 隨機函數集合，讓每個唯一的盤面透過 XOR 方式將盤面每個位置 enocde 對應的棋子。另外還會開一個[Player Cnt]的 2^64bit 隨機函數集合，由於雙方都有可能走入同個棋面，因此還需額外透過 XOR 方式記錄下個著手的數值。

    由於 ZobristKey 的編碼方式透過每個數值進行 XOR 來求得 hash key，好處是當你移動一個棋子的時候，只需要額外進行最多六次 XOR 就可以完成，不需要進行 32+1 次。

    盤面 hash 更新方法如下:

    (1) 換手: XOR 移動的玩家 XOR 下個玩家

    (2) 翻棋: XOR 翻棋位置原本的蓋棋狀態值 XOR 翻開得到的棋值

    (3) 吃棋:

    - ✧ 修改原先位置:XOR 該位置原先棋值 XOR 該位置空格值
    - ✧ 修改後來位置:XOR 該位置原先棋值 XOR 該位置後來棋值

  - ✓ **資料結構 (Hash.h)**

    由於我採用 2^64bit 的 ZobristKey，但是由於硬體上的限制，不可能開一個 2^64 大小的 hashtable，因此我選擇開(2^28+1)大小的 hashtable(如圖 1)，並透過將盤面得到的 key 值與 0xFFFFFFF(28bit)做 AND 位元運算，轉換成 28bit。這種轉換無可避免地會造成 hash clash，因此我設定每個 hashtable 最多包含

四個相同 hash index 但不同 hash key。

HashNode 0	HashNode [0,0]	HashNode [0,1]	HashNode [0,2]
HashNode 1	HashNode [1,0]	HashNode [1,1]	HashNode [1,2]
HashNode 2	HashNode [2,0]	HashNode [2,1]	HashNode [2,2]
…			
HashNode 0xFFFFFFF	HashNode [0xFFFFFFF,0]	HashNode [0xFFFFFFF,1]	HashNode [0xFFFFFFF,2]

圖 1. HashTable 整體結構圖

＊ Class HashNode (Hash.h 行數:12)

- ✧ **參數:**搜尋深度、旗標(精確值 1/上界值 2/下界值 3)、值、是否曾經走過、最好著手、hashKey
- ✧ **getIsAvailable (Hash.h 行數:43)**:確認 hashkey 是否存在於 hashtable 中，如果有就回傳旗標，不然回傳沒有存在(0)。同時，會記錄該 hashkey 在 hash index 中的第幾個 index。
- ✧ **getBestMove(Hash.h 行數:43)**:回傳該 hashkey 的最好著手。
- ✧ **getDepth(Hash.h 行數:60)**:回傳該 hashkey 的搜尋深度。
- ✧ **getValue(Hash.h 行數:66)**:回傳該 hashkey 的值。
- ✧ **updateVisit(Hash.h 行數:69)**:針對 DoMove 結束後，更新 hashkey 的實際走過次數。
- ✧ **checkVisit(Hash.h 行數:79)**:回傳該 hashkey 的拜訪次數，來避免進入循環盤面。
- ✧ **insertHash(Hash.h 行數:88)**:更新與放入 hashkey 的新數據。
  - ■ 當 hashtable 沒有資料，且還有空位則直接新增
  - ■ 當 hashtable 沒有資料，且沒有空位則覆蓋同 hash index 中最舊的資料
  - ■ 當 hashtable 有資料，但之前搜尋深度比較深，則不更新，否則將新的資料覆蓋舊資料。

- **搜尋順序 - Move Ordering (anqi.cc 行數:343)**

  將該盤面所有合法走步進行排序，當該走步為吃子步則優先搜尋。

- **動態調整搜尋深度**
  - ✓ Aspiration search (main.cc 行數:685)

    從深度為三開始搜尋，並根據給定的搜尋深度來依據搜尋，預定搜尋的層數會依據可走的步數來決定。當步數越少，要搜得越深，因為敵方可能正在佈殺手招式來引你入坑。

由於次遞增的向下搜尋，因此通常每次搜的結果都不會距離上一次搜尋出來的分數相差太遠，故 NegaScout 的 Search window 可以不用每次都是[-INF,INF]，取而代之的是先用[bestScore-THRESHOLD, bestScore+THRESHOLD]進行測試，如果超過，再重新搜尋。像是如果得出的結果(Score)小於 bestScore-THRESHOLD，則就用[-INF,Score]在搜一次；如果得出的結果(Score)大於 bestScore+THRESHOLD，則就用[Score, INF]在搜一次。

- ✓ **寧靜搜尋 Quiescent search (main.cc 行數:487) &**
  Static Exchange Evaluation (SEE) (main.cc 行數:414)

  為了避免 Horizon effect 效應，使得剛好搜到對自己有利的盤面就結束，結果忽略下一手被對方吃而扣更多分，因此必須要使用寧靜搜尋法來搜尋直到找到寧靜盤面為止。

  SEE 主要概念是，先蒐集雙方可以攻擊該位置的所有棋子，並且按照大小排序，以小子來進行雙方交互換子，如果最後換得的分數大於零(等於零也是不寧靜，如果你不現在吃該位置的子，敵方馬上返吃你)，表示我方會吃該位置的子(不寧靜)，因此 Quiescent search 必須繼續搜尋下去。

- ✓ Conditional depth extension (main.cc 行數:641)

  當該步為吃子步時或是欠行(可動步數小於等於 3)的時候，需要延伸多搜尋一層

- **經驗法則審局函數 - Knowledge heuristics (main.cc 行數:269)**
  - ✓ **一般情況(動態分數):**
    - ＊ **子力價值 Material Value:**

帥將	仕士	相象	硨車	傌馬	炮砲	兵卒
5095	4047	1523	761	450	820	200

計算盤面上無論未開或是開了的我所有棋子力價值-敵方所有的子力價值

- ＊ **動態兵卒價值(main.cc 行數:181-189):**當敵方的王還在，我的每個小兵會增加 100 分，希望小兵去吃王；當敵方的王不在，我的每個小兵會-100，降低這些小兵的價值。

- ＊ **王無天敵(main.cc 行數:192-196):**當對方的王死了，我的王就可以增加 1000 分，也就是希望把對方的王吃掉。

- ✓ **殘局情況:**
  - ＊ **時機:**當以下三種情況有一種發生時，即進入殘局(main.cc 行數:821)
    (1) 當盤面上沒有未翻開的子
    (2) 當盤面上所有棋子一共只剩 10 個以內
    (3) 當我方棋子比敵方棋子多七顆以上

  - ＊ **外加規則:**
    - ✧ **攻擊分數(距離威脅力計算)**AttackValue (main.cc 行數:131)

      自己那方透過擴散的方式[3]，來計算與可以吃的子(另一方)的距離，並利用距離跟敵方該子的子力加值來加分，希望兩者距離越近越高分。計算完兩方的攻擊分數後，利用分數差表示確實的額外加分分數。

    - ✧ **包夾敵方最大子** [3] (main.cc 行數:198-264)

      由於殘局最重要的就是包夾敵方，將敵方子一一消滅，因此我會找出兩方前兩大的子，如果是以下四種情況，則會進行包夾，因為通常這四種情況為必勝。

      (1) 我方最大子大於對方最大子，且次大子也大於對方次大子
      (2) 我方最大子大於對方最大子，且次大子等於對方次大子
      (3) 我方最大子等於對方最大子，但次大子大於對方次大子

      計算我方最大子到敵方次大子+我方次大子到敵方次大子的距離，並讓距離越小的時候，分數越高分，像是必贏分數=剩餘距離 * 敵方次大子子力價值/100。

- **避免和局**
  - ✓ **計算未吃子與未翻子的次數**(notFlipCnt) (main.cc 行數:584)

    當搜尋到盤面時，notFlipCnt>=29，就把同形表分數改為零。意義上表示，如果你現在處於優勢，為了避免合局，你會找其他正分的步來走；相反的，如果你現在處於劣勢，希望和局，你會走該盤面讓他合局。

  - ✓ **計算重複盤面次數**(RepeatCnt) (main.cc 行數:587)

    當搜尋到盤面時，RepeatCnt>=2，就把同形表分數改為零。意

義上表示，如果你現在處於優勢，為了避免合局，你會找其他正分的步來走；相反的，如果你現在處於劣勢，希望和局，你會走該盤面讓他合局。

參. 實驗

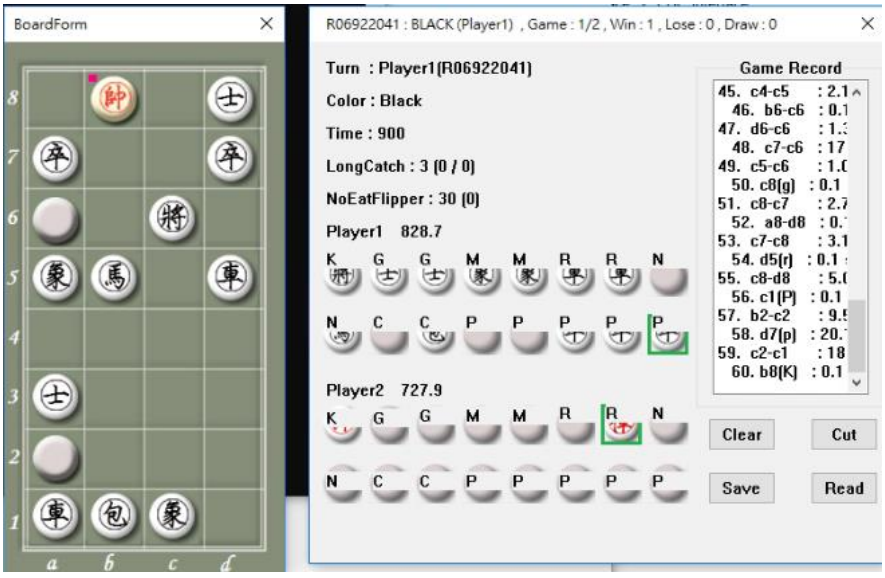
在助教公布暗棋程式碼之前，我一直反覆閱讀電腦對局理論[5]以及上課投影片，讓整體流程的設計比較全面，以便利用公布程式到比賽的兩周內順利完成智慧的暗棋程式。

比賽前我跟助教的程式可以達到 80%勝 20%和 0%輸，其實從一開始把助教的搜一層的 search Max/search Min 改成搜十層的 NegaScout 版本後，仍然維持在中局就 100%和局；接者，開始撰寫 transposition table，來加速加深 NegaScout 搜尋結果，這時就可以變到進入殘局才 100%和局；於是我開始撰寫殘局的審局函數，把棋子之間的距離考慮進去，並慢慢改進該函數，使得殘局的收尾有了改善，偶爾可以贏幾勝，大約 33%勝 66%和。有了大致上的架構後，我加上了動態深度搜尋、寧靜搜尋法避免水平效應、翻牌搜尋法避免翻到不該翻的位置、避免和局的條件、持續修改審局函數，最終可以勝率大於和局了!

除了跟助教程式比賽外，我還有跟同學們比賽來測試程式，也順便修正了不少沒發現到的問題。

肆. 比賽(The 12th NTU CSIE CUP of Computer Chinese Dark Chess Competition)

- **結果:** 6 勝 4 和 2 敗，得 8 分，第 3 名/全班 25 人
- **精彩的結束盤面:**



- **收穫:**

(1) 由於我一開始沒有把 30 步沒吃子會合局寫進演算法中，因此有一局就在我還有 6 子敵方勝 2 子(快要勝利)的時候，被對方和局了。於是我立刻再與下一個人 PK 之前馬上把此紀錄進去，避免再次被對方合局。

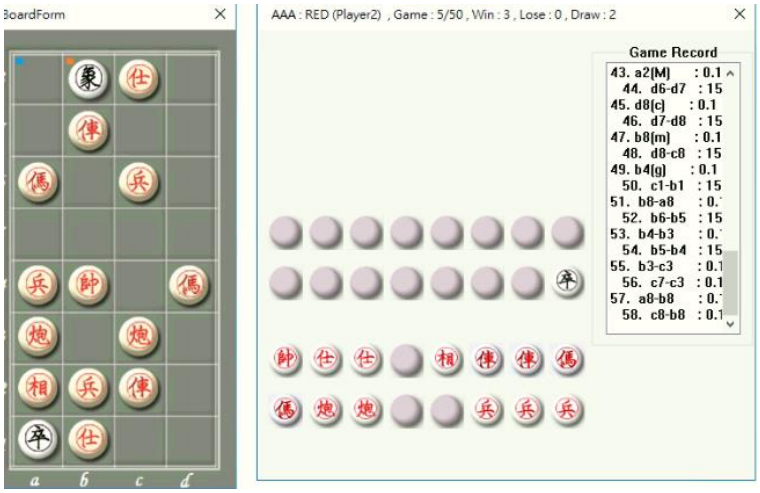
(2) 我覺得我的殘局還是需要待改進，除了(1)以外的合局，還有兩次合局也是我原本很佔上風的(我 6~8, 對方 3~4)，因為走入循環盤面而面臨合局。

(3) 我覺得暗棋開局很容易不是大好就是大壞，雖然我有寫進 chance search，但是還是偶爾會很倒楣的翻到被別人吃掉的子，像是有次當還有十個蓋住的棋子時，我(紅)翻開帥旁邊的棋子，想說多半能夠被我的帥吃掉，沒想到一翻開是卒，馬上分數大跌。

(4) 在 6 場 12 局的比賽中，我有跟第一名的程式打到，結果為 1 和 1

敗，但是我比較不解的是，比賽前一個小時我才跟他比過兩次，而且結果是 3 贏 2 和，這讓我覺得我們的對打結果很神祕。

以下是我們事前對局的部分殘局盤面:



伍. 參考文獻

[1] 謝曜安-電腦暗棋之設計與實作 2008

[2] 謝政孝-暗棋中棋種間食物鏈關係之探討與實作 2010

[3] 勞永祥-電腦暗棋之人工智慧改良 2011

[4] 徐讚昇-電腦對局理論 2017

[5] 徐讚昇-電腦對局理論網站 2018
http://www.iis.sinica.edu.tw/~tshsu/tcg/2018/index.html

陸. 討論

本次暗棋與 R06922025 林家緯、R06922034 王皓正、R06922049 徐誌鴻，共同討論與競爭比賽的結果。我覺得這門課有朋友一起修課、一起討論、一起進步是個很好的學習方式，最終我們四個也囊括了第一、第三、第五以及第十名的佳績。