# Theory of Computer Games
Final Project - Dark Chess

R04922058 鄭以琳

January 22, 2016

# 1 Introduction

Based on the given template codes, I implemented NegaScout searching procedure with a more advanced evaluating function. Transposition table is used to improve the efficiency. I also put effort in writing knowledge-based rules such as decision between moving and turning over a piece.

# 2 Implementation

## 2.1 Searching Algorithm - NegaScout

I implemented NegaScout algorithm in this project. Before searching with $[\alpha, \beta]$ window, perform a null window search to get a better chance of pruning further branches.

**Quiescent search** When encountered a terminal node, instead of directly returning the evaluated value of the current board, a quiescent search is invoked to avoid the horizontal effect. In quiescent search, only eating moves are considered. The evaluating function is called when there exists no capturing moves, or when capturing pieces is not the best choice.

**Iterative Deepening** Since the time for thinking is limited, I used iterative deepening so that when it's going to run out of time, the program stops searching deeper and returns the current best move. Iterative deepening can also be used to obtain a better move ordering. According to results from the previous iteration, NegaScount can search children with high scores first.

**Null Moves** In some situations, a player may decide to turn a piece instead of moving a piece. Therefore, the evaluation may be inaccurate if searching algorithm only considers moving face-up pieces. However, enumerating all possibilities of turning pieces is inefficient. Thus, I allow players to do null moves when there exists face-down pieces on the board to capture the idea of turning pieces.

## 2.2 Decision between Moving and Turning a Piece

For every face-down piece $p$, and every possible kind of piece $k$ that $p$ may be, the program calculates the score of board after turning $p$ to become $k$ and simulated 4 plys. Take Figure **??** for example, there are 3 face-down pieces *a2*, *b5* and *d2*. There are also three kinds of pieces these $p$ may be, which are red Guard, red Cannon and black Mister. Thus each $p$ will have three scores, indicating the score of board after 4 plys when $p$ is red Guard, red Cannon and black Mister individually. Compare these scores with

the best score of moving a face-up piece. If turning a piece has a good chance to achieve a better score (probability of "turning score greater than moving score" is greater than 0.5), the program will turn the piece with the highest sum of scores; otherwise, the program will move a face-up piece.

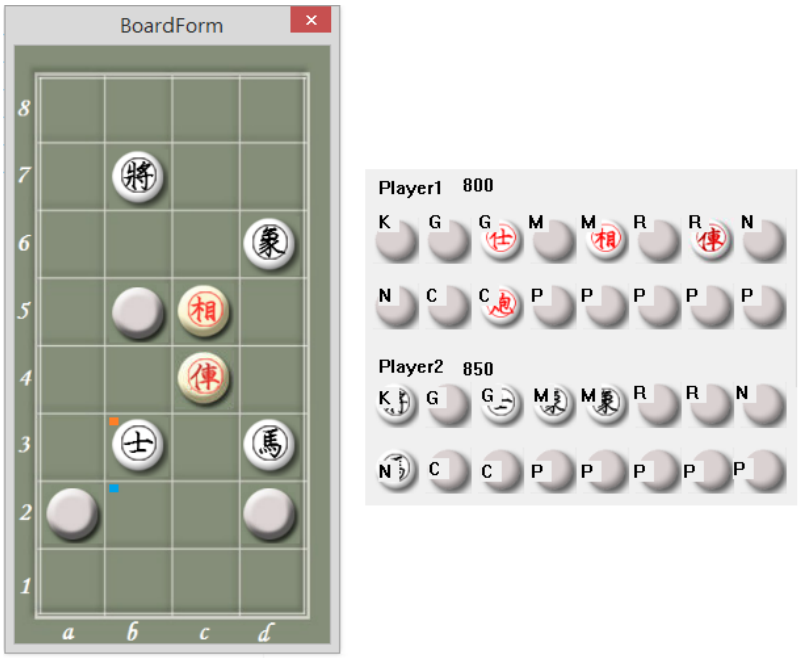


Figure 1

## 2.3 Evaluating Function

Evaluating function in my program is consists of two parts.

**Material Value** The values of pieces are set to:

| King | Guard | Minister | Rook | Horse | Cannon | Pawn |
|---|---|---|---|---|---|---|
| 12863 | 6431 | 3215 | 1607 | 803 | 805 | 200 |

When a kind of my piece can't capture any piece on the board, I set the material value of that kind of piece to 0. In this way, I can sacrifice those useless pieces in the end game if needed.

**Position Value** Generally, the side with the greater mobility, other things equal, has a better chance to win. My program calculates the absolute positional information, i.e. how many choices of moves a piece have.

## 2.4 Transposition Table

Transposition table saves the evaluation values for boards calculated before. An entry of the table includes the searching depth, the player makes the last move, the evaluated score of the board and the type of value(lower bound or exact). I implemented Zobrist Hashing with 32-bit keys and $2^{16}$ entries. Since there are limited entries in the hash table, scores calculated in quiescent search are not recorded.

One advantage of Zobrist Hashing is that the hash value of a board can be computed incrementally. Thus, my program won't re-compute the value after each move. It uses the property of XOR to modify the hash value incrementally after each move.

Before NegaScout searches deeper, it first checks the hash table. If hash hits and the hashed depth is greater than or equal to the current depth, value recorded is used. If the value in hash table is a lower bound, it is used as the initial value for searching; otherwise, it is returned directly as the result.

# Results

In the final project competition, I got 7.5 points(6 wins, 3 draws, 3 lose) in twelve games, which is the sixth place in class.

# References


[1] *Artificial Intelligence Improvement of Chinese Dark Chess*,
`http://www.csie.ntnu.edu.tw/~linss/Students_Thesis/2011_06_29_Lao_Yung_Hsiang.pdf`.

[2] *The Design and Implementation of Computer Dark Chess*
`http://handle.ncl.edu.tw/11296/ndltd/77847782641729393001`

[3] *Research and Implementation of Computer Dark Chess Program with Heuristics*
`http://handle.ncl.edu.tw/11296/ndltd/16576760897058684490`

[4] *Theory of Computer Games, Fall 2015, handouts*
`http://www.iis.sinica.edu.tw/~tshsu/tcg/2015/slides/slide8.pdf`
`http://www.iis.sinica.edu.tw/~tshsu/tcg/2015/slides/slide10.pdf`