Cheops v1.3
Manual for Users and Programmers

Tristan Miller
psychonaut@nothingisreal.com
https://logological.org/cheops

27 December 2016

1 Introduction

This document describes Cheops (CHess opponent simulator), an object-oriented chess-playing program.

2 Running Cheops

Once invoked, Cheops displays a title screen and prompts the user for who controls each player. If a computer opponent is chosen, a list of default static evaluation function weights is displayed. To accept the default values, type `done`; otherwise, enter the number of the weight to change. The program will then prompt for the new value. The program will allow the user to adjust the weights until `done` is entered.

Following player setup, the board is displayed and the game begins. The board is represented by a simple text diagram using the standard one-character piece abbreviations; white pieces are represented by small letters and black pieces by capitals. Each player makes his move in turn; human players are prompted for their move, and computer players move automatically. To make a move, enter the source and destination coordinates in standard algebraic notation—for example, `g1f3` moves the knight at g1 to f3. For pawn promotions, the promotion piece must be appended—for instance, `e7e8n` promotes to a knight. For castling, the user may either enter the source and destination coordinates of the king, or he may simply type `o-o` or `o-o-o` for kingside or queenside castling, respectively. The players may continue entering moves in this manner until a checkmate or draw state is reached. As in regular chess, a draw occurs upon stalemate, triple observance of the same board configuration with the same side to move, or observance of fifty moves without a capture or pawn advance.

In addition to legal moves, the following commands are also available as responses for human players. At the end of a game, no further moves are possible, and one of these commands must be entered.

`resign` forfeit the match

`bd` redisplay the board

`coords`	toggle display of board coordinates
`reverse`	reverse board display (black on top to white on top, or vice versa)
`think`	toggle display of computer "thinking"
`log`	display log of moves for the current game
`save`	save the move log to a file
`new`	start a new game (board in initial configuration with white to move)
`who`	set players and computer stats, if applicable
`quit`	exit the program

3 Implementation

3.1 Program structure

CHEOPS is comprised of four classes and a main program which provides an interface and ties everything together. The `ChessBoard` class is at the heart of the program; it is a fully encapsulated data structure allowing for the manipulation of chessboards and the pieces thereon. The board itself is represented by a privately-declared 64-element array, each cell containing an `enum` value representing the type and colour of the piece. There are also numerous flags and counters, the purposes of which should be obvious from the source code and its internal documentation. The class has just two major public functions—`can_move()`, which returns a boolean value indicating whether or not a given move is valid; and `do_move()`, which actually performs a move, rearranging the pieces on the board and updating the appropriate status flags. In addition to these, there are a score of small inline functions for performing menial status checks and type conversions. These functions add an extra layer of abstraction within the class, permitting the implementation of the board and pieces to be changed without having to modify the other functions in the class or its friends. Even the board arithmetic is self-contained to a degree—individual squares are represented not by integers 0 to 63 but by `enum` constants `a1` to `h8`, and incrementing ranks and files is done using constants `Rank` and `File`. This data hiding would make it easy to change the board array to, say, the popular 10×12 or 0x88 formats [1].

The remaining three classes define the players. At the top of the hierarchy is `Player`, an abstract base class serving as a template for its two derived classes, `HumanPlayer` and `ComputerPlayer`. The `Player` class itself is very small and contains no executable functions. It has just three main members: one boolean flag indicating whether the player is human, and two pure virtual functions, the first of which is used to get commands from the player, and the second to modify his stats. In `HumanPlayer`, the first function prints out the command prompt and relays the input back to the main program, while in `ComputerPlayer` it runs the minimax search algorithm to come up with the optimal move. The second function, while vestigial in `HumanPlayer`, is used in the other class to set the weights for the static evaluation function.

The `ComputerPlayer` class is one of the largest in CHEOPS, and thus deserves particular notice here. Because it needs to examine in detail the state of the chessboard, `ComputerPlayer` is a friend class of `ChessBoard`. It includes functions for determining potential moves, finding the distance between two board squares, assessing board configurations, and, of course, choosing the optimal move using a minimax search of the game tree. The actual criteria used to evaluate a particular board position are many and varied, and are treated specifically in the following section.

3.2 Static evaluation function

The heuristics for the positional evaluator of CHEOPS are based on those of GNU CHESS [4] and CRAY BLITZ [3]. Many of the criteria involve a calculation of distance between squares on the board; this distance function varies with the type of piece under consideration. For example, queens can move along any line, while rooks can move only on ranks and files; thus, the span between two fixed squares is not always equidistant for queens and rooks. Using the standard (Euclidean) means of computing distance is not feasible, even as an approximation, as it uses relatively slow floating-point calculations. Thus, functions were written for the actual piece-wise distance [5]. "Rook-wise" distance, known in mathematics as Manhattan distance, is represented by the following general formula:

$$D(x, y) = \sum_{i=1}^{m} |x_i - y_i| .$$

"Queen-wise" distance, known as Chebychev distance, is given by the following function:

$$D(x, y) = \max_{i=1}^{m} |x_i - y_i| .$$

Kings, which move the same as queens, also use Chebychev distance. Except that they can only reach half the squares on a chessboard, bishops also measure distance Chebychev-style. For knights and their unusual L-shaped moves, no suitable distance function could be found, so either of the above-noted functions is used as an approximation in the program.

Following is a detailed description of the static evaluator; any numerical constants cited refer to default values which may be changed by the user.

The base material value of a pawn is 100 points. Isolated pawns (those which have no friendly pawns on adjacent files) are given a penalty which is a function of the file they occupy: pawns on the outermost files lose 12 points, those on files 2 and 7 lose 14, those on 3 and 6 lose 16, and those on the innermost lose 20. Pawns whose advance is restrained by an opposing pawn on an adjacent file, and which have no friendly pawn to support that advance, are considered to be backward, and lose 6 points. Doubled pawns—that is, pawns which occupy a file containing other friendly pawns—are penalized 12 points. Finally, pawns are given a bonus for proximity to the opposite edge of the board, effectively receiving 1 point for each square distant from their home edge.

Knights start off at 330 points, and are given a 1-point penalty for each square distant from either king. Knights are also given a bonus for inverse Chebychev distance to the centre; the bonus starts at 0 in the corners and increases by a factor of 10 per square towards the centre.

Bishops also have a base value of 330 points. The presence of both king's and queen's bishops is necessary to avoid a penalty of 20 points. Individual bishops receive bonuses for proximity to centre, calculated and weighted (by default) in the same manner as for knights. Bishops also receive a bonus for mobility and "X-ray" mobility through pieces except pawns. This bonus is 1 point for every square under attack, plus a one-time 8-point bonus for attacks on an enemy queen, rook, or king. The weights for the mobility and X-ray mobility attacks are separately adjustable.

A rook is worth 500 points initially. Rooks receive a bonus for inverse Manhattan distance to the enemy king; the bonus increases by 20 points for every square closer to the king. Mobility and X-ray mobility bonuses are handled the same as those for bishops, and while the default weights are the same, they are independently adjustable. Also, rooks lose 10 points for the presence of friendly pawns on their file.

Queen heuristics are very simple: they start off at 900 points and receive bonuses for their proximity to the centre (in increments of 10) and to the enemy king (in increments of 20).

A king has no material value; such a number would be irrelevant as the capture of the king signals the end of the game. Instead, kings are given a penalty in steps of 10 for their proximity to the centre of the board. This penalty is a function of the stage of the game, which is determined by the combined base material value of the remaining pieces. If this sum is below a user-defined threshold (default 1500), the board is considered to be in endgame and the centre penalty is lifted. Kings are given a 20-point bonus for castling, and a 16-point penalty if they occupy a square which has no immediately adjacent friendly pawns.

The score for a board is, in effect, the sum of the scores of the opponent's pieces subtracted from the sum of the scores of the current player's pieces. If the static evaluator inspects a board in checkmate, however, this evaluation algorithm is dispensed with and the board is given a score of positive or negative infinity, depending on which side has been mated. For obvious reasons, this score is not user-adjustable. For a draw, however, the board value is user-defined (default 0). The rationale for this decision is for chess matches: when a computer has won the first four games in a six-game match, for example, there is no further need to win subsequent games; a draw may therefore be valued as a win ($+\infty$). On the other hand, a computer losing 2–3 should count a draw as a loss ($-\infty$) in the sixth and final game.

3.3 Game tree search

CHEOPS uses as its game tree search a standard minimax search with alpha–beta cutoff. One feature which was planned but dropped from the implementation was the option to time-limit the computer opponent's moves. It was felt that this option was more trouble than was worth to implement as it would have necessitated the installation of an iterative deepening algorithm. Such a feature would have resulted in better gameplay at the cost of exponentially longer thinking time. The human playtesters found that using a regular three- or four-ply search, typically completed within ten seconds, was challenging enough for them.

Another feature left out of the final release was the option for players (human

or computer) to offer draws. It was decided that having the computer offer a draw would be a sure sign to its opponent that it is losing. Conversely, a computer opponent should never accept a draw offer, even if it is losing, since the possibility should always be reserved for the human making a mistake should the game continue. That said, heuristics for forced draws (stalemates, triple occurrence, and the fifty-move rule) were still included in the program; a winning program will generally avoid such a condition.

3.4 User interface

CHEOPS features only a simple text board display and command-line interface reminiscent of GNU Chess [2]. Rather than have the static evaluation function weights read from a file, as originally planned, a menu-based system was implemented allowing the user to conveniently view and change individual weights as the program is running. For reasons of portability, CHEOPS assumes nothing about the user's display terminal, and as such, long streams of output are not paused after every page. Experience has shown that the program is best run in an environment which supports at least fifty rows and forty columns, and/or in a windowing system that includes a scroll-back buffer. Because of the highly modularized nature of the CHEOPS source code, it would not be difficult to add a system-dependent GUI for input and output.

4 Copyright

References

[1] Eppstein, D. "Board representations." *Game Programming*. `http://www.ics.uci.edu/~eppstein/180a/s97.html` (14 April 1997)

[2] *GNU Chess*. Free Software Foundation. `https://www.gnu.org/software/chess/` (26 December 2016)

[3] Hyatt, R. M., A. E. Gower and H. L. Nelson. "Cray Blitz." *Computers, Chess, and Cognition*. New York: Springer–Verlag, 1989.

[4] Stanback, J. "Heuristic descriptions for *CHESS*." *GNU Chess*. Free Software Foundation, 1987. `http://www.imsa.edu/~stendahl/comp/txt/gnuchess.txt` (11 May 1999)

[5] Wilson, D. R. and T. R. Martinez. "Improved heterogeneous distance functions." *Journal of Artificial Intelligence Research*, 6 (1997) pp. 1–34