Introduction

A chess program built in C#. Although it is not on par with commercial chess programs,. The program supports a reasonable number of functions.. One of its strengths is that it takes advantage of multiple processors when available. The program also includes a PGN filter that lets you import games in PGN format and build your own openings book.

## Features

This chess program features:

* Visual interface
* Multiple difficulty levels
* Database of book openings
* Loading / saving of game
* Undo / redo functions
* Reversing the board
* Player against computer
* Computer against computer
* Player against player
* Creating your own chess board (manually or from PGN)
* Hints for the player
* Etc.

## Versioning

|  |  |
| --- | --- |
|  | Added difficulty levels       Beginner:         Use a weak evaluation board (all pieces have the same value)                               2-Ply search                               No opening book       Easy:                Use basic evaluation board                               2-Ply search                               No opening book       Intermediate:   Use basic evaluation board                                4-Ply search                                Unrated opening book       Advance:          Use basic evaluation board                                4-Ply search                                Master opening book       More Advance: Use basic evaluation board                                 6-Ply search                                 Master opening book       Manual:             Define your own settings |

## Behind the Board

The program is developed in C# using Visual Studio . It uses the alpha-beta pruning search algorithm (and minimax for debugging) to search for the next best move. To decrease the number of moves to evaluate, the search algorithm uses a transposition table implemented with Zobrist hashing.

To further improve the performance of the search, the program uses one thread per processor found on the computer and splits the search among them .The search threads are low priority so as not to disturb too much the computer response.

The program uses a database of book openings. The one provided with the game was built from PGN files. The program also provides a PGN parser so you can build your own openings database using an option on the Tool menu. The parser also allows you to replay chess games downloaded from the Web in PGN format.

## Building an Openings Book

A database of book openings is provided with the program. You can build your own openings book from any PGN file .

The program includes a parser that allows you to import and filter the content of a PGN file according to parameters such as players or rankings. This filtered version of the PGN file can also be saved and used to create an openings book.

The openings book must be located in the directory containing the executable and named book.bin

Source Description

A chess program is not very complex in itself. But like a lot of software, the devil is in the details. This chess program contains around thousands of lines of codes (including remarks). The user interface is separated from the other classes so it can easily be changed.

The ChessBoard class is the most important since it contains the board abstraction. It also contains the logic to build the list of legal moves and to search for the best move. A little extra complexity was added to support multi-threading. However, the class is relatively small (less than 2000 lines). To improve the speed of the search, a list of legal moves for each {piece, piece position} is created once in the static constructor of the class.

* ChessBoard: Class constructor
* CopyFrom: Copy a board into another one
* Clone: Create a clone of the board
* ReadBook: Read an openings book from a file
* SaveBoard: Save the board to a stream
* LoadBoard: Load the board to a stream
* ResetBoard: Reset the board to initial position
* this[int iPos]: Default indexer, get or set a piece on the board
* GetEatedPieceCount: Return the number of pieces which have been captured for a given color
* DoMove: Do the specified move
* UndoMove: Undo the specified move
* WhitePieceCount: Number of white pieces on the board
* BlackPieceCount: Number of black pieces on the board
* IsCheck: Determine if a given color king is being directly attacked
* EnumMoveList: Enumerate all the possible moves for a given color
* FindBestMove: Find the best move for a given color using alpha-beta or minimax
* FindBookMove: Find a move in the openings book
* GetHumanPos: Return a human readable move from a move structure
* CancelPlay: Cancel the background search

The core logic of the search lies in the alpha-beta pruning function. This function can be used in two modes:

* Specific number of ply
* Iterative deepening depth-first search

The first method searches for the best move in a specified number of ply.

The second one tries to find the best move in a specific amount of time using an iterative depth-first search, increasing the number of ply for each search up to the moment when time is exhausted. At first glance, this method may seem less efficient since it performs the same search repeatedly. But in practice, the method reorders the moves between each search to optimize the alpha-beta cut-off. Another big advantage of this method is that the number of ply can be adjusted depending on the stage of the game. In particular, the end game holds fewer pieces on the board, so increasing the number of ply doesn't have the same impact as doing so in the middle of the game.

The following lists the source files and description. The number of lines appears in brackets after the name of the file..

* *Assembly.cs*(34)  
  Assembly file for .NET application.
* *Book.cs*(359)  
  Implements the book openings.
* *ChessBoard.cs*(1990)  
  Implements the chess board regardless of the user interface. This is where the core logic of the program lies (search, legal moves, etc.). The search function is implemented using minimax and alpha-beta algorithms, using multi-threading when possible.
* *ChessControl.cs*(1510)  
  User interface for the chess board. Implemented as a UserControl.
* *ChessControl.Designer.cs*(86)  
  Visual Studio generated code for the control.
* *frmAbout.cs*(16)  
  About dialog box.
* *frmAbout.Designer.cs*(110)  
  Visual Studio generated code for the form.
* *frmChessBoard.cs*(1236)  
  Main form containing all the other controls (ChessControl, MoveViewer, etc.).
* *frmChessBoard.Designer.cs*(499)  
  Visual Studio generated code for form.
* *frmCreatePGNGame.cs*(114)  
  Interface to convert a PGN file into a book openings database.
* *frmCreatePGNGame.Designer.cs*(97)  
  Visual Studio generated code for the form.
* *frmGameParameter.cs*(165)  
  Parameters of the game.
* *frmGameParameter.Designer.cs*(218)  
  Visual Studio generated code for the form.
* *frmPGNFilter.cs*(340)  
  Parameters for filtering a PGN file.
* *frmPGNFilter.Designer.cs*(309)  
  Visual Studio generated code for the form.
* *frmPGNGamePicker.cs*(209)  
  Choosing from PGN game.
* *frmPGNGamePicker.Designer.cs*(103)  
  Visual Studio generated code for the form.
* *LostPiecesControl.cs*(299)  
  Control used to show the captured pieces.
* *LostPiecesControl.Designer.cs*(63)  
  Visual Studio generated code for the control.
* *MoveViewer.cs*(192)  
  Control used to show the moves.
* *MoveViewer.Designer.cs*(87)  
  Visual Studio generated code for the control.
* *PGNParser.cs*(765)  
  Parser for PGN notation.
* *PgnUtil.cs*(816)  
  Utility class for PGN files.
* *Program.cs*(21)  
  Main program.
* *TransTable.cs*(232)  
  Transposition table implementation.

#### . Ply

A ply consists of a half move (a move of one side only). A 4-ply search means to search 2 moves in advance.

#### PGN

Portable Game Notation, or PGN, is a notation used to record chess games. PGN is widely used as it is easy to read by users and to process by computers. Many chess games and events are published in the PGN format. The parser allows the chess program to read these files.

#### Minimax

Minimax is a recursive algorithm use for choosing the next move in a game. A tree of legal moves is built and played. Each move is evaluated using an evaluation function. The computer makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves.

#### Alpha-beta Pruning

The alpha-beta pruning function is an improvement of the minimax search method. It reduces the number of nodes to evaluate by eliminating a move when at least one possibility was proved worse than a previously evaluated one.

#### Transposition Table

A transposition table is a hashing table that records the previous moves' evaluations so they will not have to be re-evaluated. Transposition tables are used to speed up the search of the game tree. They are implemented using Zobrist hashing .

#### Zobrist Keys, Zobrist Hashing

To implement a transposition table, it is important to determine if two boards are equivalent in configuration and in potential moves. To do so, we can just compare the pieces of the two boards, but we must also take into account castling and en-passant moves, as they constrain possible moves. The only problem is that this method is a quite long when considering it has to be used millions of times to evaluate each move. Zobrist hashing simplifies this process by assigning each board position a 64-bit signature; instead of checking each piece one by one to see if the board has already been evaluated, we just compare the two 64-bit values