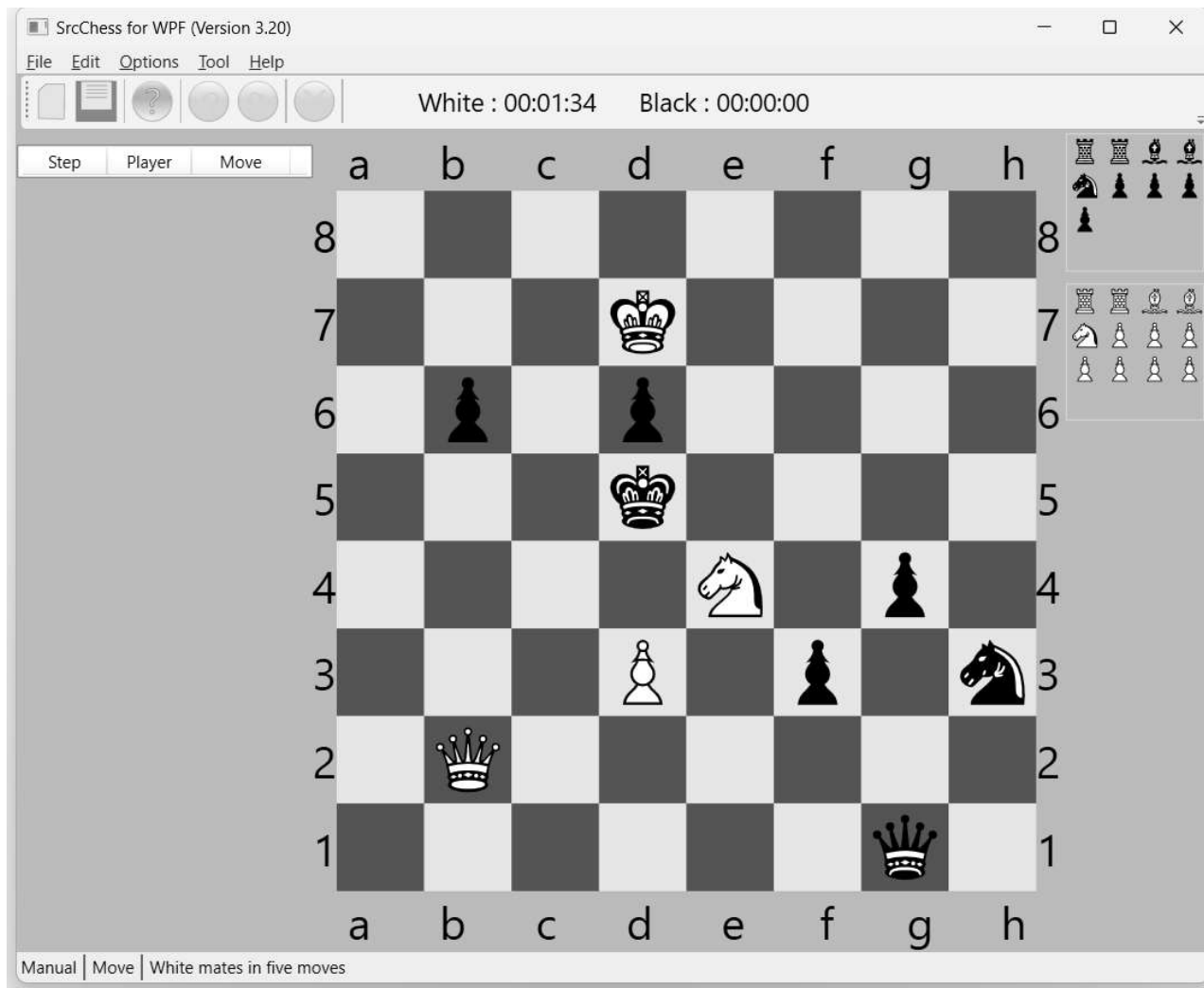# Chess Program in C#

Jacques Fournier
15 Sep 2023   GPL3   12 min read

SrcChess is a chess program built in C#

SrcChess is a chess program that supports a reasonable number of functions and is built using C#.

Download program - 2.7 MB

Download source - 2.5 MB

Introduction

`SrcChess` is a chess program built in C#. Although it is not on par with commercial chess programs, `SrcChess` is beating me without any problem and therefore can be a serious opponent for casual players. The program supports a reasonable number of functions. Its biggest weaknesses are probably the lack of a good board evaluation function and of an end game database. 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.

I decided to make my program available so programmers can understand how a chess program works. I also hope some people will improve it.

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

Version 3.23	• Correct the Queen and King being in incorrect position after a board rotation	
Version 3.22	• Fixes a crash when trying to find the best move for a player who is in check and there are not enough pieces to do a mate	

Version 3.21	- Moves to .NET 7.0 - Improves search performance using dynamic PGO - Adds a reverse board option - Adds a *readme.txt* describing the source files - Removes the pawn promotion to a pawn - Finishes the generalization of the search engine
Version 3.20	- Runs only on .NET 6.0 - The main menu has been restructured. - The status bar now displays the number of threads running when the best move is being searched. - Restructures the code to separate: - the core chess routines - the search engine - the PGN parsing - the FICS interface (chess server) - the user interface - Fixes the crash which occurs when a game has more than 256 moves (512 ply) - The chess 50 move rule now correctly checks for 50 moves (was 50 ply). - Diables iterative deepening. The way the multi-threading is implemented makes the iterative deepening longer than when you're not using it. I will try to find a solution to that. - The translation table now works properly and improves the performance of the search. - Code clean-up has been done.
Version 3.03	Correct multi-thread problem when using translation table

	- Updates the project to use the .NET Framework 4.8 (was .NET Framework 4.0) - Compiles in x64 to support bigger translation table (was x86) - Adds a .NET 6.0 version of the program - Revises the code to - Supports the new C# syntax (nullable references, new switch expression, etc. - Improves the esthetic of the code (no more hungarian notation)
Version 3.02	Prompts user before switching to design mode
Version 3.01	- Corrects bug which permits a castling to occur with a rook which has been eaten if neither the rook nor the king has been moved. - Corrects an exception which was occurring when loading a PGN files with an invalid format. - Adds the game result to terminate the move list when saving a game in PGN format.

Version 3.0	Added difficulty levels
	
	Beginner: Uses a weak evaluation board (all pieces have the same value)
	
	                  2-Ply search
	                  No opening book
	Easy: Uses basic evaluation board
	                  2-Ply search
	                  No opening book
	Intermediate: Uses basic evaluation board
	                  4-Ply search
	                  Unrated opening book
	Advance: Uses basic evaluation board

4-Ply search
                    Master opening book
        More Advance: Uses basic evaluation board
                    6-Ply search
                    Master opening book
        Manual: Defines your own settings

- Simplifies the user interface
- Adds interface to the FICS (Free Internet Chess Server). You can now observe the following games in real time: Lightning / Blitz / Untimed and Standard time
- Adds tooltips in many dialog boxes and in the main interface
- Adds more than 100 mates in N move games
- Adds a warning for saving a board before leaving a game
- Moves the chessboard closer to the center
- Rewrites the PGN parser to handle bigger PGN files and to be more compliant with PGN specifications.
- The new parser comes with an improved advanced book and a new intermediate one. The new books have been created from a 2.77 millions TWIC games. Thank you to chess.com for the SCID file. The advanced book includes games from players with ELO rating of 2500 or more.
- Simplifies status bar
- Adds a progress bar when finding a best move or waiting for a move from FICS server
- Did a major code clean-up
- The game is now saving its last position and size.
- To come: Let users play game via FICS server.

Version 2.05	- Bugs corrections. More information in *Readme.txt* - Adds the `Refresh` option

Version 2.04	• Bugs corrections. More information in *Readme.txt* • New menu to create a snapshot of a game to help fixing bugs.
Version 2.03	• Adds a button to load PGN games without the moves
Version 2.02	• Bugs correction: Endless loop when reading/parsing PGN files
Version 2.01	• Bugs correction. More information in *Readme.txt*
Version 2.00	• Moves to WPF • New user interface • Adds list of piece sets to choose from • Thank you to Ilya Margolin for the XAML piece sets
Version 1.10	• Moves to .NET Framework V4 • Uses `System.Threading.Tasks` to simplify the multi-threading implementation • Improves the search engine and the board evaluation interface • Correct exception when resizing the `ChessControl` • For a more exhaustive list, look at the *readme.txt* file.
Version 1.00	• Improves the user interface • Improves the search engine and the board evaluation • Adds a new iterative depth-first fix ply search method • Corrects depth-first so it performs correctly • Adds timer • Games can now be saved in PGN format • Saved format is NOT compatible with previous version • For a more exhaustive list, look at the *readme.txt* file.

|	
Version 0.943.000	- Adds support for the threefold repetition rule - Adds support for the fifty-move rule - Adds a new interface to help adding new board evaluation methods to the game. For more information, reads the *BoardEvaluator.txt* file. - Adds a test mode to compare the performance and the efficiency of board evaluation method (**Tool** -> **Test Evaluation Method**...). Clean-up the code a little...
Version 0.942.000	- Ply count has been corrected so it represents a move by one player. - Adds an option to configure move shuffling (to add some random to game). It's now possible to disable it to make debug easier. - Adds timing information about search
Version 0.941.000	- Iterative deepening depth-first search is now working. You can now choose a fixed amount of time for finding a best move instead of a number of ply. - The opening book will choose more often usual openings.
Version 0.940.000	- Adds an option to enable/disable book opening - Adds an option to select the multi-threading mode - Adds an option to set the size of the transposition table - Search setting is now persisted - Corrects the transposition table algorithm - Decreases the points given for castling in the board evaluation - Iterative deepening depth-first search is close to be functional... but not yet.
Version 0.930.002	- Corrects exception occurring at the end of a game - New option to enable/disable the transposition table. (The option is off by default to correct a bug. Next version will enable it by

	default.)
Version 0.930.001	• Original posted version

Behind the Board

The program is developed in C# using Visual Studio 2010. 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 (finally a use for the multiple processors on my computer...). The search threads are low priority so as not to disturb the computer response too much.

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.
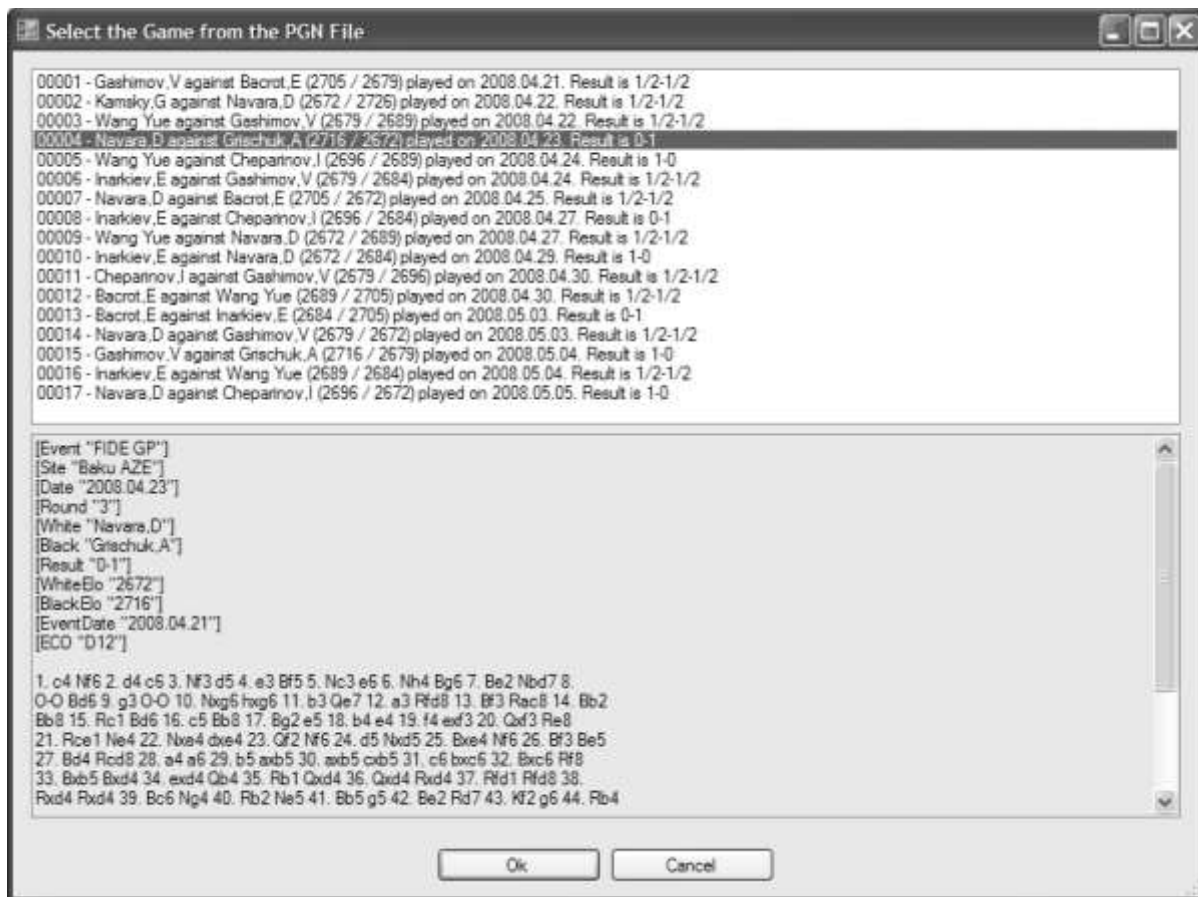
Building an Openings Book

A database of book openings is provided with the program. You can build your own openings book from any PGN file (easily found on the Web).

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*.
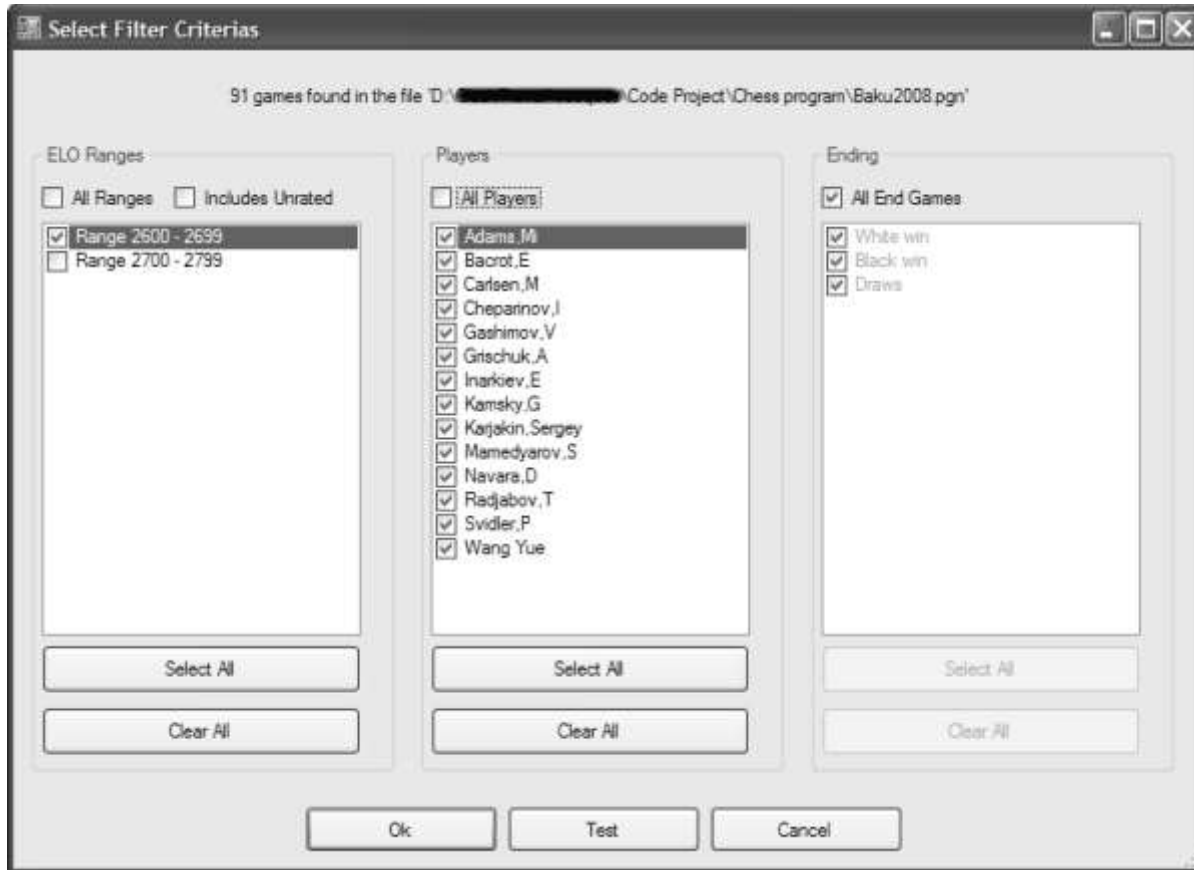
What Needs to be Improved?

The board evaluation function is minimalist. Improvements on this function will greatly enhance the level of playing of the program. Similarly, the end game stage of the program could benefit from the inclusion of an end game database.

There is no rating among the different openings; an opening is thus chosen randomly.

Improvements can also be made on the user interface. Adding a help file to the game would be welcome.

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 10,000 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 content of the different folders. For more information, please consult the *readme.txt* files in the root folder.

- Root folder: Game user interface
- *Core:* Contains the implementation of the chessboard, moves, moves history and board evaluation
- *FicsInterface*: Interface to FICS (Free Internet Chess Server)
- *GenericSearchEngine: Contains the generic search engine using MinMax or AlphaBeta algorithm*
- *Properties: Defines the different settings*
- *PgnParsing*: Parsing of chess game expressed in the Portable Game Notation
- *PieceSets*: Different chess piece sets
- *Resources*: Binary resource

Short Glossary

All terms can be easily found on the Web (Wikipedia is a good source).

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.
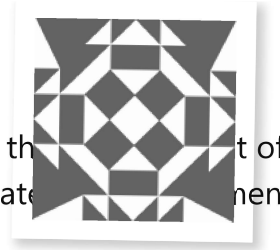
License

This article, along with any associated source code and files, is licensed under The GNU General Public License (GPLv3)

Written By

Jacques Fournier

Team Leader Consyst SQL

[+] Canada

Consyst is a dynamic IT company specialized for more than 20 years in information technology architecture and in th[...]t of innovative productivity tools for businesses. Rep++, the product at the core of its mission, can significantly accelerat[...]ent cycle of applications and services by reducing the duration of the design, coding, testing and maintenance stages.

Rep++ uses a model-driven approach supported by a powerful model execution mechanism. Essential complement to Visual Studio® (Microsoft®), Rep++ includes: an open and centralized model that is used to define, contain and manage all the metadata of an application set; toolkits and application frameworks that implement various flavors of the presentation layer; and specialized assistants that simplify the creation of applications and services for a variety of architectures and technologies. These elements provide a very high automation level, which enable businesses to focus their development efforts on where it counts: their business rules.

Comments and Discussions

▤ **260 messages** have been posted for this article Visit **https://www.codeproject.com/Articles/36112/Chess-Program-in-C** to post and view comments on this article, or click **here** to get a print view with messages.