



Chess Program in C#

Jacques Fournier

26 Oct 2020 GPL3

SrcChess is a chess program built in C#

[Download source - 2.5 MB](#)

[Download demo - 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 3.0	<p>Added difficulty levels</p> <ul style="list-style-type: none"> 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 <p>Simplified the user interface</p> <p>Added interface to the FICS (Free Internet Chess Server). You can now observe the following games in real time: Lightning / Blitz / Untimed and Standard time</p> <p>Added tooltips in many dialog boxes and in the main interface</p> <p>Added more than 100 mates in N move games.</p> <p>Added a warning for saving a board before leaving a game.</p> <p>Moved the chessboard closer to the center</p> <p>Rewrote the PGN parser to handle bigger PGN files and to be more compliant with PGN specifications.</p> <p>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.</p> <p>Simplified status bar</p> <p>Added a progress bar when finding a best move or waiting for a move from FICS server.</p> <p>Did a major code clean-up</p> <p>The game is now saving its last position and size.</p> <p>To come: Let users play game via FICS server.</p>
Version	Bugs corrections. More information in Readme.txt

2.05	Add 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	Add a button to load a 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	Move to WPF New user interface. Add list of piece sets to choose from Thank you to Ilya Margolin for the XAML piece sets	
Version 1.10	Move to .NET Framework V4 Use <code>System.Threading.Tasks </code> to simplify the multi-threading implementation Improves the search engine and the board evaluation interface Correct exception when resizing the <code>ChessControl</code> For a more exhaustive list, look at the <i>readme.txt</i> file.	
Version 1.00	Improves the user interface Improves the search engine and the board evaluation Add a new iterative depth-first fix ply search method Correct depth-first so it perform correctly Add 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 <i>readme.txt</i> file.	
Version 0.943 .000	Add support for the threefold repetition rule Add support for the fifty-move rule Add a new interface to help adding new board evaluation methods to the game. For more information, reads the <i>BoardEvaluator.txt</i> file. Add 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. Add 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	Add an option to enable/disable book opening Add an option to select the multi-threading mode Add an option to set the size of the transposition table Search setting is now persisted Correct the transposition table algorithm Decrease the points given for castling in the board evaluation Iterative deepening depth-first search is close to be functional... but not yet.	
Version	Corrects exception occurring at the end of a game.	

0.930 .002	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).	
Versi on 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 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 (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 source files and description. The number of lines appears in brackets after the name of the file. The code has a total of 9836 lines.

- *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.

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 .

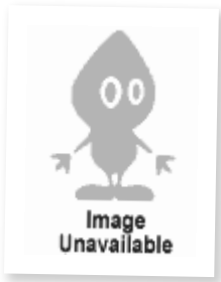
Zobrist Keys, Zobrist Hashing

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 \(GPLv3\)](#)

About the Author




Jacques Fournier

Web Developer Consyst SQL
Canada 🇨🇦

Consyst is a dynamic IT company specialized for more than 20 years in information technology architecture and in the development of innovative productivity tools for businesses. Rep++, the product at the core of its mission, can significantly accelerate the development 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

 **191 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.

[Permalink](#)
[Advertise](#)
[Privacy](#)
[Cookies](#)
[Terms of Use](#)

Article Copyright 2009 by Jacques Fournier
Everything else Copyright © [CodeProject](#),
1999-2020

Web06 2.8.20201022.1