Typhoon Chess Engine

Scott Gasch

scott@wannabe.guru.org

Typhoon Chess Engine
by Scott Gasch

Published $Date$

This is a user's guide for the typhoon chess engine (http://wannabe.guru.org/scott/hobbies/chess). Its intended audience is the set of people who have downloaded a copy of my chess engine source code available at http://wannabe.guru.org/svn/typhoon/trunk or have downloaded a precompiled binary that someone else compiled. The goal of this manual is to document how to build and use the chess engine. If you don't care about how to use typhoon are instead looking for a more technical discussion of how to write your own chess engine you might try http://wannabe.guru.org/scott/hobbies/chess or simply dig into the source code (http://wannabe.guru.org/svn/typhoon/trunk) itself.

This user's guide is available in several formats:

- One large HTML file: http://wannabe.guru.org/scott/hobbies/chess/typhoon.html (http://wannabe.guru.org/scott/hobbies/chess/typhoon.html)
- Plain (7 bit ASCII) text: http://wannabe.guru.org/scott/hobbies/chess/typhoon.txt (http://wannabe.guru.org/scott/hobbies/chess/typhoon.txt)
- Adobe PostScript: http://wannabe.guru.org/scott/hobbies/chess/typhoon.ps (http://wannabe.guru.org/scott/hobbies/chess/typhoon.ps)
- Adobe Portable Document Format (PDF): http://wannabe.guru.org/scott/hobbies/chess/typhoon.pdf (http://wannabe.guru.org/scott/hobbies/chess/typhoon.pdf)
- Microsoft Rich Text Format (RTF): http://wannabe.guru.org/scott/hobbies/chess/typhoon.rtf (http://wannabe.guru.org/scott/hobbies/chess/typhoon.rtf)

Table of Contents

Chapter 1. Preliminaries

This is a the user's guide for the typhoon chess engine. It covers how to build, install, configure and use the engine.

If you are impatient, have a look at the Quick Start Guide section; it will take you through the basics of getting the engine installed and working. Come back to the rest of the user's guide if you run into difficulties.

Readers with more patience may skip the Quick Start and read the rest of the guide in order for a run though of how to get the engine installed and configured. This will also familiarize you with more advanced topics such as how to build your own opening book from PGN files, how to instruct typhoon to use Eugene Nalimov format endgame tablebases, how to run script files and how to execute commends automatically at engine startup time.

Feel free to email me (mailto:scott@wannabe.guru.org) with questions or problems. Before you do, though, please read this guide and see if your query has been addressed already.

1.1. Introduction

Typhoon is a chess playing program that I've been working on for a few years now as a hobby. It's ugly, unpolished and full of bugs. While it has its moments of brilliance, it is not yet as strong as some other freely available engines like Yace or Crafty. If you find a bug, especially one related to playing strength, I'd like to hear about it. (mailto:scott@wannabe.guru.org)

When I reach a point in this project where I am happy with the playing strength, usability, stability and portability of the engine I'll release it under the GPL or a similar license. Until then please consider the source code an alpha-quality prerelease. Do not redistribute, sell, or modify my chess engine.

Typhoon has a modest (and probably outdated) homepage on the Internet at http://wannabe.guru.org/scott/hobbies/chess/. Drop by and have a look. Typhoon has played in three (3) tournaments to date, CCT3 (http://www.vrichey.de/cct3/), CCT4 (http://www.vrichey.de/cct4/) and CCT5 (http://www.vrichey.de/cct5/). In CCT3 it placed 30th of 32. In CCT4 it placed 15th of 46. In CCT5 is placed 6th of 45. The engine also plays from time to time on the Internet Chess Club (http://www.chessclub.com) where it maintains a standard rating around 2500 and a blitz rating around 2700 (http://www.chessclub.com/cgi-bin/finger/finger.pl?handle=monsoon).

1.2. Acknowledgments

Thanks to all the members of the Computer Chess Club (CCC) (http://www.talkchess.com) discussion board especially Bob Hyatt (**Cray Blitz**, **Crafty**) and Bruce Moreland (**Ferret**, **Gerbil**) for their patience and willingness to explain chess-programming concepts.

Thanks to Eugene Nalimov and Ernst Heniz (**DarkThought**) for their continuing work on high quality endgame tablebases.

Thanks to Tim Mann for his continuing work on **xboard / WinBoard**. Thanks to Tom Kerrigan (**TSCP**, **Stobor**) for publishing **TSCP** source code which was the first chess engine I read and the reason I became interested in chess

programming. Thanks to Thorsten Greiner for writing and publishing the source to his **Amy** program and (again) to Bob Hyatt for writing and publishing the source to **Crafty**.

Thanks (again) to Ernst Heinz for publishing his research on computer chess.

Thanks to FM Vincent Diepeveen (**Diep**) for his discussions and expert advice. Many thanks to Dann Corbit for the initial port of typhoon to the **Microsoft Visual C/C++** and **Intel C++** compilers.

Thanks to IM Mark Chapman for his help with opening book lines and his patient expert analysis of chess positions.

Thanks to Peter McKenzie (**LambChop**, **Warp**) for discussing chess programming ideas and sharing his thoughts and advice.

Finally many thanks to Steve Timson (**Chester**) for sharing his good ideas and listening to my lousy ones... without his advice typhoon would surely not be as strong as it is today.

The binaries included in the typhoon distribution are based, in part, on code that I did not write. Such code remains under the copyright notice of it's author. I'm grateful to the original authors of the code listed below for sharing it and giving me permission to use it.

- `mtf.c` (Mersenne Twister random number generator) is Copyright (C) 1997 by Makoto Matsumoto and Takuji Nishimura. It has been included in typhoon with the authors' permissions.
- (part of) `system.c`, specifically the case insensitive string comparison and manipulation functions, were taken from the source code for the BSD C runtime library when they were found to be not present on Win32. This code was released under the BSD license and remains Copyright (C) 1987 by Regents of the University of California.
- *Win32 timer code* was donated by Dann Corbit during the initial port of typhoon to Win32. It's used with the author's permission. Thanks, Dann!
- `egtb.cpp` was written by Eugene Nalimov, released as part of crafty, and reused with the author's permission.

1.3. Quick Start Guide

This is the raw step-by-step guide for getting your copy working on your computer. If you have questions about any step, stop reading the Quick Start Guide and move on to the detailed explanation later in this document. In fact, if you are not impatient, go ahead and skip the whole Quick Start Guide; everything covered here is also covered more completely later on.

1. Verify that you are either running some flavor of **Windows NT (NT/2000/XP/Server 2003/Vista)**, **FreeBSD**, **Linux** or **OSX**.
2. Verify that you intend to run the chess engine on an x86 microprocessor (Intel Pentium, AMD Athlon, Intel-based Mac, etc...)
3. If you plan to compile the engine from source code, follow the steps below. If you have a precompiled binary for your system you can skip these.
   a. You will need either **gcc, g++**, and **gmake** or the **Microsoft Visual C++** compiler.
   b. You will also need a copy of the **nasm** assembler.
   c. Copy the source code to a directory on your machine (or use Subversion to get a snapshot).

d. If you're using **gcc, g++** and **gmake** have a look at `GNUmakefile` and make sure all variables look reasonable to you. Then, from the commandline type **`make PERF_COUNTERS=1`**. This will produce a binary image called `typhoon` if all goes well.

   If you're using **MSVC**, select the "Release" configuration and build it. If all goes well it will produce a binary called `typhoon.exe` in the `Release/` subdirectory. Note that you may have to edit the project settings to point to where `nasm.exe` can be found on your system.

e. If you want to build a multithreaded version of the engine with **gcc**, add **`MP=1`** to your **gmake** commandline. If you're interested in building a multithreaded engine with **MSVC** just build the "MP Release" configuration.

4. All you need to play chess is the typhoon binary itself and some chessboard GUI program (like Tim Mann's **xboard** (or **winboard**) (http://www.tim-mann.org/xboard.html)). To run typhoon under **xboard** just type **`xboard -fcp /path/to/typhoon`**. Note that typhoon requires version 4.2.3 or higher of **xboard** (or **WinBoard**). There is also a section about using typhoon under **xboard** or **WinBoard** later in this guide.

   If you want an opening book you can either download one from my site (http://wannabe.guru.org/scott/hobbies/chess/books) or build your own from PGN. If you're interested in the latter, see the guide section about building a book from scratch.

   If you're interested in doing anything non-trivial with the engine you'll probably want to read the sections on commandline arguments, commands, and tablebases later in this document.

If you have any problems, please read the rest of this guide before emailing me (mailto:scott@wannabe.guru.org).

1.4. System Requirements

Typhoon only runs on x86 microprocessor based systems. This means that your Intel Pentium or AMD Athlon system will work. This also means that more recent Apple Macs will run the program. I have not yet ported the code to any other processor architecture. It's possible that there will be a native AMD64 port of typhoon in the future (namely, when I buy a machine or when Apple releases a full 64-bit operating system).

Typhoon only runs under **Windows NT** (**Windows NT 4.0**, **Windows 2000**, **Windows XP**, **Windows Server 2003**, **Windows Vista**), **FreeBSD**, **Linux** and **OSX**.

Specifically, the engine, when built with Microsoft Visual C/C++, is known not to work with **Windows 95**, **Windows 98**, and **Windows ME**. Some third parties have reported that the engine does work under **Windows 98** when built with **Cygwin** but I have never tried this configuration. If you're not sure what version of Windows you are using, click "start" then "run" then type **`winver`**. If the version reported is greater than or equal to 5.0 then your operating system can definitely run typhoon.

The codebase may build on other operating systems for Intel-based processors. If **gcc** and **nasm** are available for your platform of choice, give it a try and let me know (mailto:scott@wannabe.guru.org) how it goes.

If you got a precompiled version of typhoon from some third party then, of course, you need to make sure the version you received was built for your system.

> **Note:** I do *not* currently provide a precompiled version of the code and that running any program someone else built for you is inherently risky. Make sure you trust the source of such an image and be careful running it. If you're unsure of the integrity of the source of the precompiled package it's better to build your own.

To run the engine your computer should have something reasonable like at least 128Mb or memory and around 10Mb of free hard drive space for an opening book.

The engine does not need to be run with administrative privileges but if you do run it with elevated privileges it will do some nice things like try to lock its memory and run with slightly raised priority.

While a chess GUI (like **xboard**, **WinBoard**, or **Arena**) is not technically "required" in order to typhoon, they make the experience more enjoyable. The engine itself has no UI to speak of -- it is its only output is text based. Therefore it's strongly recommended that you download and install some GUI frontend to use with typhoon. There is a section later in this guide that describes where to get **WinBoard** and how to get the engine working with it.

1.5. Compilation Guide

If you want to compile the chess engine from source code (which may be your only option since at present I do not distribute binary images of the engine) you will probably need to use **gcc/g++** with GNU **make**, **gcc/g++** with BSD-style **make** or **Microsoft Visual C/C++**.

I've built typhoon successfully on pretty much every version of **gcc** from 2.8 onward. I've also used all of the Microsoft compilers from **VC6.0** onward. As far as I know any of these should work fine with the caveat that the `typhoon.sln` file currently checked in and available on the source site is for **Microsoft Visual C/C++ 2003**.

> **Note:** I have never tried using **mingw** (**gcc** for **Windows**) or Intel's **icc** (which claims to compile **MSVC** projects) so if you try one let me know how it goes. The code is reasonably compiler agnostic so getting it to build on some other random C/C++ compiler should be relatively painless... I hope. If you have any success porting the codebase to another toolset, please drop me a line (mailto:scott@wannabe.guru.org).

In order to build `x86.asm` you will also need a copy of the Netwide Assembler (**nasm**) (http://nasm.sourceforge.net/wakka.php?wakka=HomePage). Some (much hairier) alternatives are to translate the assembly language to work with some other assembler or to just use the C-language versions of the routines in `x86.asm`. Neither option is recommended. Make sure that if you are building typhoon on a Mac you have a version of **nasm** that can produce mach0 format object files (i.e. it supports the **`-f macho`** commandline flag).

Start by copying everything from svn/typhoon/trunk (http://wannabe.guru.org/svn/typhoon/trunk) into a directory on your machine. While you can do this manually (or with wget/fetch), a cool alternative is to use **Subversion** (http://subversion.tigris.org/) (a version control system) to check out a read-only snapshot of the code from my server. The reason I do not have a pre-packaged code archive is because the code you can download from the URL above is a current snapshot of the chess engine; when you use the URL above you are getting the most recent update of the engine available.

Have a look at the `README` file to humor me. Then, what happens next depends largely on what compiler you have chosen to use.

If you decided on the **gcc/g++/gmake** toolset, take a look at the `GNUmakefile`. You might want to change some variables such as CC, CXX and NASM, and CPU based on the system you plan to build with. The defaults are probably reasonable in most cases.

If you have a BSD-style **make**, take a look at `Makefile` instead.

In order to build a single threaded release version of the chess engine, type **`make PERF_COUNTERS=1`**. A multithreaded release version is almost the same, just add an **`MP=1`** to the line. Whereas both **Linux** and **FreeBSD** based systems require explicit no build flags, to build for Apple **OSX** systems you should add an additional **`OSX=1`** to the build commandline.

While we're on the subject, here's a list of the preprocessor symbols that you can use when building typhoon and their effects:

Preprocessor symbol	Effect on image
DEBUG	Produces a much slower image that has extra checks enabled. Symbols are not stripped. See the section about DEBUG builds for more information.
TEST	Bakes some unit testcases into the image, enables the **`test`** command. Refer to the section about TEST builds for more information.
ASM	Causes **gcc** to produce intermediate assembly language (`.s`) files during the build.
EVAL_DUMP	Causes the engine to pay attention to all terms affecting a position's evaluation and be dump them after every eval. Read the section about EVAL_DUMP builds to learn more.
EVAL_TIME	Causes the engine to pay attention to how many processor cycles it spends evaluating each position.
PERF_COUNTERS	Enables several performance related counters in the engine.
BOUNDS_CHECKING	If you have patched your version of **gcc** to include **`-fbounds-checking`** , this builds an image with baked in bounds checking support.
MP	Enables the threadpool and search-splitting code needed to support more than one thread searching at a time. This has been tested on 2-cpu and 4-cpu machines. See also the *--cpus* commandline argument.
DUMP_TREE	Produces dumps of search trees in XML format. Tree dump files can be viewed with a web browser or using the **typhoonui.exe** viewer in the **Subversion** repository.
OSX	Produces a binary for Intel-based Apple Macs

Building with **MSVC** involves selecting a configuration and compiling. You probably want to build "Release" or "MP Release". The only wrinkle is that in the custom build step you will probably have to set the path to **nasm** on your system. The resulting image will be either `Release\typhoon.exe` or `MP Release\typhoon.exe` depending on which configuration you built.

The image you build can run stand-alone as a text-based chess engine with no opening book. But if you want to run in a more comfortable manner, read on to the next sections which cover the details of setup.

1.6. Using Subversion to get Typhoon

I use the **Subversion** version control system to develop typhoon. If you have **svn** installed on your system you can use it to check out a read-only snapshot of the source code on your machine, keep up to date with changes I make, access file histories, project branches and change logs.

Just use the URL `http://wannabe.guru.org/svn/typhoon` as your repository path. To get the initial snapshot of the code issue the **`svn checkout`** command from a directory you created to house the typhoon source code:

```
svn checkout http://wannabe.guru.org/svn/typhoon/trunk
```

The command **`svn co`** is a shorthand version of **`svn checkout`**. Also, if you want more of the typhoon project than just the current code, you can omit the **`/trunk`** from the end of your command. Warning, this will checkout a bunch of papers, PGN files, opening books, etc; the disk space requirements for the full repository are non trivial.

In order to synchronize your enlistment with the current state of the code on my machine, use the **`svn up`** command. Another useful commands is **`svn log http://wannabe.guru.org/svn/typhoon`** which will give you a high level overview of what changed from revision to revision. An alternative is http://wannabe.guru.org/cgi-bin/svn.pl?operation=log. Using **`svn diff`** allows you to see source code changes at a file level.

```
svn up
```

When I am actively developing the engine I make at least one checkin a week (usually more like one a day). So if you want to track the bleeding edge, this is the way to do it.

1.7. Installation Guide

Once you have a typhoon binary image (either from building it yourself or from downloading a precompiled binary from some third party) you may want to do some simple setup work.

1. The first thing you may want to do is to install Tim Mann's graphical chessboard. The UNIX version is called **xboard** and the Windows version is called **WinBoard**. Both can be obtained from the site http://www.tim-mann.org. This program will provide a nice graphical user interface to typhoon which is a text-based engine. Without it you'll be stuck looking at chessboards on a commandline interface. There is a chapter later in this guide about how to configure the engine to work under WinBoard.

   An alternative to **xboard/WinBoard** is **Arena**, another graphical chessboard program which can be downloaded at http://www.playwitharena.com (http://www.playwitharena.com/). While I have only done preliminary testing with Arena, typhoon seems to work just fine as a WB2 engine under **Arena 1.99beta2**. Since I haven't done much with Arena, if you want to add typhoon as an Arena engine you're on your own.

2. After you've installed either **xboard**, **WinBoard** or **Arena** you'll need to copy the typhoon image into some directory on your hard drive. I usually use `C:\typhoon` but you can put it wherever you like. I'll be referring to the directory you put the image in as the "typhoon installation directory" from now on in this guide.
3. You now have all you need to play chess. However, without an opening move library the engine will play the same opening moves every game. If you want an opening move library you have two choices: either download one or build one yourself.

   The former choice is easier and I have several opening books available on my server. You can get them from http://wannabe.guru.org/scott/hobbies/chess/books. The larger the file, the more opening moves in the library. If you choose to download an opening library file, just pick one and save it to the typhoon installation directory as `book.bin`.

   The advantage of building your own opening library is that it's an easy way to tailor the playing style of the engine. It will only play the opening lines you train it with. To go this route you will need a PGN file full of games you want the engine to learn opening moves from. Read the section about making a custom opening book for detailed instructions.

4. Just as an opening library is a database of common opening moves for the engine to use, an endgame tablebase is a database of endgame positions that the engine can access during the endgame. Typhoon knows how to read Eugene Nalimov format EGTB files which are the same ones that the popular engine Crafty uses. These files are quite large; all 3-4-5 man files are over 7Gb in size compressed. A good resource for learning more aboue EGTB files (including where to get them) is http://www.aarontay.per.sg/Winboard/egtb.html.

   If you have EGTB files on your disk you can tell typhoon where they are using the "--egtbpath" commandline argument. This flag preceeds a quoted, semi-colon delimited path:

   ```
   typhoon --egtbpath "C:\egtb\three;C:\egtb\four;C:\egtb\five"
   ```

   You will know that typhoon found and could use the EGTB files if you see it produce a message like "Found 5-men endgame table bases." during startup. More information about the use of EGTB files appears later in the guide.

5. The final thing you'll need to do to run the engine on your computer is set the hash table sizes based on the amount of memory you have in your computer. If you run the engine stand-alone (i.e. not under a GUI like WinBoard) you can issue the command "dump sizes" to see how much memory the pawn hash table and main hash table are using. As you can see, the default is for the engine to use just over 300Mb of physical memory:

   ```
   white(1): dump sizes
   sizeof(PAWN_HASH_ENTRY). . . . . . . . . 88 bytes
   sizeof(HASH_ENTRY) . . . . . . . . . . . 16 bytes
   sizeof(MOVE) . . . . . . . . . . . . . . 4 bytes
   sizeof(ATTACK_BITV). . . . . . . . . . . 4 bytes
   sizeof(SQUARE) . . . . . . . . . . . . . 8 bytes
   sizeof(POSITION) . . . . . . . . . . . . 1428 bytes
   sizeof(MOVE_STACK) . . . . . . . . . . . 233992 bytes
   sizeof(PLY_INFO) . . . . . . . . . . . . 424 bytes
   sizeof(COUNTERS) . . . . . . . . . . . . 236 bytes
   sizeof(SEARCHER_THREAD_CONTEXT). . . . . 264056 bytes
   sizeof(GAME_OPTIONS) . . . . . . . . . . 1136 bytes
   sizeof(MOVE_TIMER) . . . . . . . . . . . 40 bytes
   sizeof(PIECE_DATA) . . . . . . . . . . . 16 bytes
   ```

```
sizeof(VECTOR_DELTA) . . . . . . . . . . . 4 bytes
sizeof(GAME_PLAYER). . . . . . . . . . . . 16 bytes
sizeof(GAME_HEADER). . . . . . . . . . . . 64 bytes
sizeof(GAME_MOVE). . . . . . . . . . . . . 60 bytes
sizeof(GAME_DATA). . . . . . . . . . . . . 72 bytes
sizeof(SEE_LIST) . . . . . . . . . . . . . 196 bytes
sizeof(BOOK_ENTRY) . . . . . . . . . . . . 36 bytes
-------------------------------------------------
Current pawn hash table size . . . . . . . 46137344 bytes (~44 Mb)
Current main hash table size . . . . . . . 268435456 bytes (~256 Mb)
```

If this amount exceeds the amount of physical memory on your machine the engine will be extremely slow. It is possible to reduce the memory footprint of the engine by setting the size of the main hash table. To do so, use the "--hash" commandline option.

This commandline flag takes one argument. This argument can be the number of entries in the table (at 16 bytes per entry). Or it can simply be a more friendly "desired size" of the table. Here are some examples of both forms:

```
typhoon --hash 16384
typhoon --hash none
typhoon --hash 256M
typhoon --hash 128K
```

The first example above allocates a hash table of 16384 (2^14) entries. Since each main hash entry is 16 bytes in size, the total memory used by 16384 main hash entries is approximately 256kb. The second form instructs the engine to use no hash table at all. This is not a recommended configuration but it should work. The third and fourth forms instruct the engine to allocate approximately 256mb and 128kb respectively.

Remember that the engine allocates other memory besides the hash table; for instance, each searcher thread requires approximately 75mb of storage. Therefore, if you tell the engine to only use 256mb of hash, do not be surprised to find the memory footprint of the engine is more like 375mb.

Chapter 2. Running Typhoon

This chapter covers typhoon's commandline arguments and command parser. It gives a list of useful commands for interacting with the engine. Note that if you are running the engine under **WinBoard** or some other GUI front end then your interface will send the commands to typhoon on your behalf. This is a much easier way to interact with engine and is recommended for most users. The following sections assumes you are an advanced user and running typhoon in text mode in order to interact with the program directly.

2.1. Commandline Arguments

The following section covers the commandline arguments available when starting the engine and the affect of each on its behavior.

1. The *--cpus* argument, followed by a number, can be used with multiprocessor builds of typhoon to indicate how many searcher threads should be created. The number of searcher threads should be set to the number of processors in the system:

   ```
   typhoon --cpus 2
   ```

2. The flag *--command*, followed by a quoted string, can be used to pass an initial command to the engine. Typhoon will execute the initial command before processing any user input. The initial command is often used to specify a script to execute.

   ```
   typhoon --command "bench"
   ```

3. The *--hash* argument takes a number indicating the size of the main hash table. This size must be an even power of two or it will be rounded downward. The *--hash* command allows you to tailor engine memory usage to the size of physical memory on your system. See the installation guide for a full discussion of how this argument works.

   ```
   typhoon --hash 16384
   typhoon --hash 512M
   typhoon --hash none
   typhoon --hash 128K
   ```

4. Use *--egtbpath* with a quoted string in order to set the path in which the engine should search for Nalimov format EGTB files. The string can contain more than one directory if the different directories are separated by semi-colons (;).

   ```
   typhoon --egtbpath "C:\TB\three;C:\TB\four;C:\TB\five"
   ```

5. Use *--batch* to indicate that the engine should never listen to user input from the console. This flag takes no additional parameters and must be used in conjunction with the *--command* flag. The presence of *--batch* causes the engine to skip starting an input thread and exit immediately after executing the initial command.

```
typhoon --command "script C:\ECM.EPD" --batch
```

2.2. Entering Moves

The most common input you'll probably send to the engine is a chess move. Typhoon understands moves in two (2) formats: Standard Algebraic Notation (SAN) and the **d2d4** format. When you enter a move in one of these formats that affects a piece of the side on move, the move will be made and the board redrawn.

2.3. WinBoard Commands

Typhoon supports many commands that are part of the **WinBoard** protocol. These commands are fully documented in Tim Mann's xboard engine interface document at http://www.tim-mann.org/xboard/engine-intf.html. I will briefly discuss a subset of them here.

- *xboard*, *random*, *hint*, and *variant*, and *edit* are not implemented in typhoon and are basically no-ops. Typhoon can only play regular chess, no variants are supported. To edit a position typhoon uses the newer opcode **setboard**.
- *quit* is used to exit the chess program.
- *new* is used to start a new game. This sets the computer to play black, resets the maximum search depth, resets the board, and clears all internal data structures.
- *force* puts the engine in "force mode" which means it plays neither side.
- *white* and *black* are used to tell the engine what color the opponent plays and what side has the move currently. For example **white** means the engine should play black and that it is current white's turn to move.
- *sd* can be used as an alternative to **set SearchDepthLimit** to set the maximum depth in ply that the engine should search a position. The maximum value is 63, the minimum is 1.
- *st* can be used to query or set the clock. If **st** is followed by a parameter the clock is switched into fixed time per move mode and the parameter specifies the number of seconds to search per move.
- *time* can be used in addition to **set ComputerTimeRemainingSec** to notify the engine about how much time remains on its clock. This value affects time per move allocation.
- *otim* can be used in addition to **set OpponentTimeRemainingSec** to notify the engine about how much time remains on the opponent's clock. This value affects draw-value calculation.
- *go* tells the engine that it plays the side on move and to begin searching immediately.
- *?* can be used to interrupt the search and instruct the engine to "move now".
- *result* is used to report the end of a game and its result. It requires an argument to indicate the result and allows an optional comment. For example: **result 1-0 {black resigns}**, **result 1/2-1/2 {stalemate}**. The optional comment, if supplied, becomes the result description in the PGN header of the game. This command triggers book learning and prints out a PGN record of the game.

- *undo* and *remove* can be used to take back one half-move or one full-move respectively.
- *easy* and *hard* instruct the engine to ponder (think on the opponent's time) or not to ponder respectively.
- *name* can be used in addition to **`set OpponentName`** to set the name of the opponent. This is used to construct PGN headers.
- *rating* can be used in addition to **`set ComputerRating`** or **`set OpponentRating`** to set the ELO rating of the computer and its opponent. This information is used in draw value calculation and PGN header construction. The **`rating`** command requires two (2) parameters; the first is the computer's rating and the second is the opponent's rating.
- *computer* can be used in addition to **`set OpponentIsComputer true`** to inform the engine that is it playing against another computer. This setting changes draw value calculation and affects some position evaluation terms.
- *rated* and *unrated* can be used to inform the engine that the game it is playing is rated or unrated. Typhoon uses this information in draw value calculation and when deciding whether to allow a move takeback. This is not part of the **WinBoard** protocol but rather is a hack I added to my copy of **WinBoard** in order to accommodate move takebacks on Internet Chess Servers.
- *level* is used setup the chess clock. This command takes three (3) parameters. The first is the number of moves per time period, the second is the computer's starting clock value and the third is the increment added to the computer's clock per move.
- *setboard* is used to setup a position on the chessboard. It requires an argument which is the position to setup in FEN format.

2.4. Miscellaneous Commands

There are still some other commands that have been hacked in and are not part of the **WinBoard** protocol. These commands may change at any time and should not be relied upon to remain constant in future versions.

Note also that these commands are hard to access when the engine is running under **WinBoard**. If you want to use them you might consider running the engine from the console.

- *board* can be used to dump a text-mode drawing of the current board position as well as the current FEN.
- *pgn* can be used to dump a PGN record of the current game.
- *fen* can be used to display the current board position in FEN format. With an argument this command behaves the same way that **`setboard`** does: it sets the current board position.
- *avoid*, *solution*, *id* and *script* are commands normally used to set test positions for the engine. They are covered in more detail in another section.
- *bench* is a command to run an engine speed benchmark. It is discussed in detail in another section.
- *book* is a command to manage opening books and is discussed in detail in another section.
- *dump* and *test* are commands to show internal engine state and run self diagnostic checks. See the source code for details.
- *eval* can be used to display a static evaluation score of the current board position. If the engine is built with EVAL_DUMP defined it will display the terms that combined to arrive at the eval score.
- *help* displays a brief command list.

- *set* is used to show or change the state of engine variables. It is discussed in detail in another section.
- *version* is used to display the version number and build configuration of the engine.

2.5. Typhoon Variables

Many aspects of the engine's behavior can be controlled by setting the values of variables. Variables in typhoon are names that hold some value. Variables can hold numbers, strings, times, boolean flags, and so on. To view the present state of a variable or to change it you use the **set** command.

With no arguments, **set** displays the present state of all system variables:

```
set
AnnounceOpening              => "TRUE"
BatchMode                    => "FALSE"
BlackPlayer                  => "typhoon"
BlackRating                  => 0
BlackDescription             => "Ver: 1.00 Build Time: 09:08:03 Jul  4 2006"
BlackIsComputer              => "TRUE"
BookFileName                 => "book.bin"
BookProbeFailures            => 0
ComputerTimeRemainingSec     => 600
EGTBPath                     => "C:\egtb\three;C:\egtb\four;C:\egtb\five"
GameDescription              => "(null)"
GameLocation                 => "(null)"
GameIsRated                  => "FALSE"
GameResultComment            => "(null)"
LastEval                     => 0
LogfileName                  => "typhoon.log"
MoveToPonder                 => "(null)"
MovesPerTimePeriod           => 4294967295
OpponentTimeRemainingSec     => 600
PendingInputEvents           => 0
PonderingNow                 => "FALSE"
PostLines                    => "TRUE"
SearchDepthLimit             => 63
SearchTimeLimit              => 0
SearchStartedTime            => 0.000000
SearchSoftTimeLimit          => 0.000000
SearchHardTimeLimit          => 0.000000
SearchNodesBetweenTimeCheck  => 0
ThinkOnOpponentsTime         => "TRUE"
ThinkingNow                  => "FALSE"
TournamentMode               => "FALSE"
VerbosePosting               => "FALSE"
WhitePlayer                  => "(null)"
WhiteRating                  => 0
WhiteDescription             => "(null)"
WhiteIsComputer              => "FALSE"
Xboard                       => "FALSE"
```

When used with one argument, **set** displays the value of a subset of the engine variables that begin with the user supplied argument. For example:

```
set s
SearchDepthLimit             => 63
SearchTimeLimit              => 0
SearchStartedTime            => 0.000000
SearchSoftTimeLimit          => 0.000000
SearchHardTimeLimit          => 0.000000
SearchNodesBetweenTimeCheck  => 0
```

In order to set the value of any variable, use the **set** command with two arguments: the first to indicate what variable name is being set and the second to supply a new value for that variable. For example:

```
set tourn t
```

Note that some variables are read only and cannot be set manually. To change such variables I'm afraid you're stuck editing the source code.

When you use the **set** command, the letter case of a variable name is not significant; the names "WhitePlayer" and "whiteplayer" refer to the same variable in typhoon. Additionally as you noticed in the example above, you may abbreviate any variable or value name when using the set command as long as the abbreviation is unique. For example you could use **set tourn t** to achieve the same result as **set TournamentMode True** with less typing.

Chapter 3. Opening Book

A chess engine's opening book is a library of moves in different positions that it can access and play in leu of a searching for a move. These moves are usually drawn from well known opening lines played by human masters. The primary purpose of an opening book is to impart some level of understanding of opening theory to the engine. A secondary goal of an opening book is to vary the deterministic play of the engine.

3.1. Opening Book Commands

Commands affecting typhoon's opening book are prefixed with the **`book`** opcode. They are:

- *book name*, which sets the opening book filename. If you do not use this command the engine defaults to using `book.bin` as the opening book. If you want to use something else or to create a new opening book, use this command to override the name of the opening book file.

  ```
  book name newbook.bin
  ```

- *book import*, which can be used to import book learning from an old book into a new one or to merge opening lines from a PGN file into the current opening book. Any book editing or learning that takes place in a position is recorded in typhoon's `book.edt` file. If you wish to apply these book changes to another book in the future, use **`book input`**. This command also is how you create or merge new lines into the opening book. For more information about creating an opening book, see the appropriate section.

  ```
  book import /home/scott/typhoon/pgn/twic.pgn
  ```

- *book dump learning*, which shows book learning in the current position.
- *book dump moves*, which shows a list of all moves in the opening book at the current position.
- *book tourn*, which displays or toggles tournament mode. This is equivalent to the **`set TournamentMode`** command.
- *book openings*, which sets the name of the book opening lines file for use when announcing opening lines.

3.2. Downloading a Pre-built Opening Book

Creating your own custom opening book has the advantage of allowing you complete control over what line are included. But it requires some time, a machine with a lot of memory, and a good source of PGN data.

If you want to save yourself the trouble of making a custom opening book you can choose from several pre-built typhoon books which are available at http://wannabe.guru.org/scott/hobbies/chess/books.

3.3. Building a Custom Opening Book

You may want to build a custom opening book with typhoon. This section will describe the process for you and give a few tips about book building.

To begin you will need to collect the games you want to include in your custom book into a single PGN file. PGN is a standard format for storing chess games in text files. Programs like the freely available **scid** which can be obtained from http://scid.sourceforge.net/ can be very helpful when you are trying to organize and maintain large collections of chess games. Commercial programs like **ChessBase** do a good job too, of course.

Typhoon knows how to read PGN format files and import the moves from each game in the PGN file into it's opening book. However at this time typhoon's PGN reader is a little picky about what it can process. In general it's pretty good but it will not read games with move lists that do not have a space between the move number and the move. It also can become confused by variations or comments in the PGN notation. I suggest you use a chess database to normalize the PGN file you intend to use for your opening book before sending it through typhoon.

Once you have your PGN file ready you have to decide whether you want to merge the openings in the PGN file with typhoon's book or create a new book from scratch. The engine has an opening book filename that you can set via the **`book name`** command. If this file exists then the engine will append new openings to it. If this file does not exist then the engine will create it. By default, this file is called `book.bin`. Here's how to override the default:

```
white(1): book name D:\typhoon\regence.bin
Opening book name set to "D:\typhoon\regence.bin"
```

Now that you have set the name of the opening book you can import your PGN file to create the new opening lines. To do this type **`book import`** ***`file.pgn`***. Of course replace ***`file.pgn`*** with the name of your PGN file. Typhoon will read the PGN file one game at a time and store the moves in the book file in BookName. If typhoon encounters a game containing an error or a move it does not understand it will output a line like "** BAD Game NNNN (line NNNNNN) saw=XXX... skipped" and discard all moves in that game. Unfortunately the PGN parser is fairly picky which is why normalizing your PGN input using a chess database program before building a book is a good idea. Here's what to expect:

```
white(1): book import D:\typhoon\pgn\misc\regence.pgn
The opening book D:\typhoon\regence.bin does not exists, creating new book
Stage 1: reading and parsing PGN
** BAD Game 19498 (line 416067) saw="xc2"... skipped.
** BAD Game 53900 (line 1148893) saw="Rxf8+"... skipped.
** BAD Game 72836 (line 1560210) saw="O-O"... skipped.
Done reading PGN.
Straining out unpopular positions to compact buffer...
Compacting the opening book... one moment.
Done, compacted 2517462 positions into 135201.
Sorting the book... this may take a while.
Merging book and writing to disk.
Book successfully built...
```

> **Note:** Book building can take quite a while, especially on a machine with limited memory. I suggest you have at least 1Gb of memory to build a large opening book. If you find that building a book takes multiple hours you can decrease the number of entries in typhoon's "membook" by adjusting the MemBookSize variable. Note that the

> smaller you make the "membook", though, the more often typhoon will flush unpopular positions from the opening book. Everytime this happens you risk losing good book lines.
>
> It is safe to terminate typhoon while it is building an opening book if it takes too long. However there is no way to pick up where it left off and all work on the opening book will be lost.

When you have finished building an opening book I recommend you exit typhoon and restart the engine. This is not strictly required but is a good idea. Once you have **`quit`** typhoon you can check to make sure your new book exists. To use it, simply use the **`book name`** command to instruct typhoon to use an alternate opening book. An alternative is to rename the file you downloaded as `book.bin`, the engine's default opening book name.

```
white(1): quit
Freeing dynamic memory allocations...
Closing logfile...
Terminating input thread...

D:\typhoon\Release>dir ..\regence.bin
 Volume in drive D is New Volume
 Volume Serial Number is 1412-36C8

 Directory of D:\typhoon

10/10/2002  04:14 PM         4,867,236 regence.bin
               1 File(s)      4,867,236 bytes
               0 Dir(s)  58,596,573,184 bytes free
```

3.4. Book Learning

At the end of every game typhoon adjusts its opening book by updating the win/loss statistics on the line of opening moves it just saw played. This leads to a positive/negative reinforcement of different openings and, hopefully, to superior play over a long period of time. Typhoon will not adjust the opening book after bullet games, after it beats a low-rated opponent, or after it is beaten by a high rated opponent (see the **`rating`** command). It will also not adjust the opening book if the losing side lost by forfeit.

When typhoon adjusts the opening book it records the learning in a file on disk called `bklearn.dat` as well as in the book file. This allows you to view the learning information periodically and watch which opening lines are receiving what kind of reinforcement.

Unfortunately typhoon cannot learn new opening moves from game play -- it can simply change the weight of moves it already knows in a given position. In order to teach typhoon new moves, use the **`book import`** command to merge a new PGN file with the current opening book.

```
STDIN> result 0-1 {MoonShot checkmated}

[Event "Rated blitz computer chess game"]
[White "MoonShot (computer)"]
[Black "typhoon 0.906 (00:42:58, Oct  8 2002) (computer)"]
[WhiteElo "2795"]
```

```
[BlackElo "2731"]
[Result "0-1"]
[Opening: "[B92] Sicilian : Najdorf, Opovcensky Variation"]
[Description: "{MoonShot checkmated}"]

1. e4 c5 2. Nf3 d6 3. d4 cxd4 4. Nxd4 Nf6 5. Nc3 a6
6. Be2 e5 7. Nb3 Be7 8. O-O O-O 9. Be3 Be6 10. f4 exf4
11. Rxf4 Nc6 12. Nd5 Bxd5 13. exd5 Ne5 14. a4 Nfd7 15. Rb4 b6
16. Qf1 a5 17. Rb5 Bg5 18. Bd4 Rc8 19. c3 Re8 20. Re1 Nc4
21. Bxc4 Rxe1 22. Qxe1 Rxc4 23. Nd2 Rxa4 24. Qd1 Rxd4 25. cxd4 Be3
26. Kh1 Bxd4 27. b4 a4 28. Nf3 Be3 29. Qxa4 Qc8 30. Qa3 Bf2
31. h3 Qc4 32. Qa8 Nf8 33. Qe8 h5 34. g3 Qd3 35. Kg2 Be3
36. Ng1 Qc2 37. Kh1 Qf2 38. Qxe3 Qxe3 39. Kg2 Nd7 40. g4 Qd3
41. Rxb6 Nxb6 42. Nf3 Nxd5 43. Kg3 h4 44. Kf2 Qe3 45. Kf1 Nf4
46. Nxh4 g5 47. b5 gxh4 48. b6 Qe2 49. Kg1 Qg2
{MoonShot checkmated}
```

```
                |     c5 (+0.00)
                |     d6 (+0.00)
                |   cxd4 (+0.00)
                |    Nf6 (+0.00)
                |     a6 (+0.00)
                |     e5 (+0.00)
                |    Be7 (+0.00)
                |    O-O (+0.00)
                |    Be6 (+0.00)
                |   exf4 (+0.00)
                |    Nc6 (+0.00)
                |   Bxd5 (+0.00)
                |    Ne5 (+0.00)
                |   Nfd7 (-0.90)
                |     b6 (-0.79)
                |     a5 (-0.08)
                |    Bg5 (-0.12)
                |    Rc8 (+0.32)
                |    Re8 (-0.02)
                |    Nc4 (+0.14)
                |   Rxe1 (+0.67)
                |   Rxc4 (+0.22)
                |   Rxa4 (+0.72)
                |   Rxd4 (+0.63)
                |    Be3 (+0.78)
                |   Bxd4 (+0.75)
                |     a4 (+0.75)
                |    Be3 (+0.81)
                |    Qc8 (+1.21)
                |    Bf2 (+1.33)
                |    Qc4 (+2.87)
                |    Nf8 (+4.59)
                |     h5 (+6.03)
                |*    Qd3 (+9.50)
                |*   Be3 (+12.06)
                |*   Qc2 (+13.36)
```

```
|**   Qf2 (+13.78)
|**  Qxe3 (+14.91)
|**   Nd7 (+15.16)
|**   Qd3 (+15.44)
|**  Nxb6 (+16.04)
|**  Nxd5 (+17.53)
|********   h4  (+57.50)
|********   Qe3 (+57.52)
|********   Nf4 (+57.56)
|********    g5 (+57.60)
|********  gxh4 (+57.62)
|********   Qe2 (+57.64)
|********   Qg2 (+57.66)

BookLearn: Revising opening book line...
BookLearn: Changed 26 book positions.
```

Chapter 4. Endgame Tablebases

Endgame tablebases are special databases that contain game theoretic information about different board configurations with very few pieces left on the board. When a chess engine finds a position in a tablebase it will play perfectly by using the tablebase data. For example, by accessing a tablebase an engine might instantly determine that a certain configuration of king vs. king, knight and bishop is a mate in 38 for the stronger side.

Eugene Nalimov has created a collection of endgame tablebases that typhoon knows how to access during its search. At present all 3, 4 and 5 man endgames have a corresponding tablebase and some 6 man endgames also have a tablebase. The drawback of tablebases is they require a large amount of hard disk space to store and are slow to access. The benefit of tablebases is that they can drastically improve engine endgame play in certain positions.

This chapter is about tablebases, where to find them, how to generate them, how to validate them, how to compress them, and how to use them with typhoon.

4.1. An Overview of Tablebases

As the previous section explains, tablebases are endgame databases that contain information the engine can use to play perfectly in some endgames with a low number of pieces on the board. There are ten (10) 3-man tablebases, sixty (60) 4-man tablebases, and two-hundred twenty (220) 5-main tablebases. At present some 6-man tablebases have been generated but are not in wide use because of their large disk space requirements.

Typhoon uses Nalimov format tablebases. These tablebases come two (2) ways: compressed and uncompressed. Typhoon can use either type. The compressed variety use much less disk space than their uncompressed equivalents and are only slightly slower to access. Nalimov tablebases are compressed with a program called `datacomp.exe`. You can distinguish compressed files from uncompressed ones by looking at the extension: compressed tablebases end with `.emd`.

To store all 3 and 4-man tablebases (compressed) on your hard drive you will need approximately 31Mb. If you want to store all 3, 4 and 5-man tablebases (compressed) you'll need more like 7.5Gb of drive space. It is estimated that the full set of 6-man tablebases (when they are available) will consume approximately 1Tb (1024Gb) of drive space. And for you psychopaths out there, if you wrote one tablebase entry on every atom in the universe you could still not store a 32-man endgame tablebase file.

Now that you know how much of your hard drive these things will use you may (or may not) want to know where to get them. One option is to download them from Bob Hyatt's FTP site at http://ftp.cis.uab.edu/pub/hyatt/TB. Remember we're talking about transferring 7.5Gb of data; you'll need a nice fast connection to even consider this. With a 56K modem this will take 40 hours (at least) -- probably more.

An alternative to transferring the tablebases is to generate them on your own computer. This way you only have to download the program that makes the tablebases -- when it runs it will use your computer's CPU to compute the tablebase data and save it on your hard drive. The `tbexe.zip` and `tbgen.zip` files on the above FTP site contain the program, source code, and a README file that explains the generation process. I have never been though this personally but I have heard that it takes about one week of processing time on a computer with a reasonably fast processor and large amount of memory to generate a full set of 3, 4 and 5-man EGTBs. It should also be noted that the generation program creates *uncompressed* tablebase files so you will need approximately 40Gb of drive space to attempt this. Once you have generated the tablebases, though, you can use `datacomp.exe` to compress them and save some space.

Another choice is to buy these tablebases on CD. Chessbase (http://www.chessbase.com), Gambitsoft (www.gambitsoft.net), and Convekta (http://store.convekta.com/) all sell sets of Nalimov tablebases on multiple CDs/DVDs.

Finally, this section would be incomplete without a link to Aaron Tay's great frequently asked question (FAQ) list about endgame tablebases which is online at http://www.chesskit.com/aarontay/Winboard/egtb.html. This helpful page covers different tablebase formats, how to validate your tablebases, how to generate your tablebases and more.

4.2. Using Nalimov Format Tablebases with Typhoon

Once you have some Nalimov format tablebase files (compressed or uncompressed) getting typhoon to use them is pretty easy. Use the **`set egtbpath`** command to tell typhoon where on your hard drive to look for the tablebase files. If your tablebase files are in more than one directory, just separate the directories by semi-colons (;).

```
set egtbpath D:\typhoon\egtb\three;D:\typhoon\egtb\four
Rescanning EGTB path...
...Found 4-men endgame table bases.
```

When you change the value of the **`EGTBPath`** variable typhoon automatically rescans the path looking for tablebase files. Scanning can take a few seconds. If everything goes well you will see a message like "...Found N-men endgame table bases".

> **Note:** It is recommended you use complete sets of endgame tablebase files. If you try to use 5-man tablebases without some 4-man tablebases or 4-man tablebases without some 3-man tablebases the engine can become confused and misplay endgames. It is acceptable to use only some 5-man files if you have all 4-man and 3-man files, though. Likewise it is acceptable to use some 6-man files if you have all 3, 4 and 5-man files to support them.

Chapter 5. Benchmarks

Benchmarking is the process of measuring the speed of the chess engine. This chapter deals with how to run and interpret the results of the benchmark.

5.1. The `bench` Command

The command to begin a benchmark is **`bench`**. Be aware that the benchmarking process can take a while and be prepared to wait.

The **`bench`** command runs the same benchmark that Bob Hyatt built into the **Crafty** chess engine (at least the copy of **Crafty** that I have, which is a few versions out of date). It consists of searching in a series of positions and computing the overall nodes (positions) per second during the searches. This is a good overall speed test for typhoon.

```
bench
Beginning Bob Hyatt's crafty benchmark sequence.
This takes a while -- please be patient.

DEPTH LIMIT --> stop searching now
move c3c2
DEPTH LIMIT --> stop searching now
move e5e6
DEPTH LIMIT --> stop searching now
move f4f5
DEPTH LIMIT --> stop searching now
move d7f5
DEPTH LIMIT --> stop searching now
move b7e4
DEPTH LIMIT --> stop searching now
move c8f5

Benchmark results:
  122409037 nodes searched
  495 sec
  247057 overall nps
```

5.2. Benchmark Results

Please see http://wannabe.guru.org/scott/hobbies/chess/benchmarks.html for up-to-date benchmark results. If you benchmark typhoon and your system is not on the list below please email me (mailto:scott@wannabe.guru.org) your results and I will include them here.

Chapter 6. Test Suites

One way to measure the strength of a chess engine is to run it against a test suite -- a collection of positions and solution moves (or moves to avoid). Typhoon supports running test suites and this chapter deals with how to run a test suite against the engine. It also contains a table of test suite results reported by other users.

6.1. Test Suite Commands

The typhoon commands that support test suites are **`solution`**, **`avoid`**, **`id`**, and **`script`**. The **`solution`** command sets up a solution move; typhoon can handle up to three (3) solutions per position. The **`avoid`** command sets up a move to avoid; typhoon can handle up to three (3) avoid moves per position. The **`id`** command simply adds a free form text name for a test position. And the **`script`** command reads the contents of a file on disk and treats everything in it as user input.

> **Note:** You can set up either **`solution`** moves or **`avoid`** moves but not both in the same position.

Here's an example of how these commands work together.

```
st 5
script \typhoon\tests\wac.epd
SCRIPT> setboard 2rr3k/pp3pp1/1nnqbN1p/3pN3/2pP4/2P3Q1/PPB4P/R4RK1 w - -
SCRIPT> solution Qg6
SCRIPT> id "WAC.001"
SCRIPT> go

 1u   -1.91  00:00:00.08  486          PV= 1. Qe3 Nxe5 [Q] 2. dxe5 [Q] <-2.00>
 1u   -1.56  00:00:00.11  556          PV= 1. Ne8 <-2.00>
 1.   -1.56  00:00:00.12  576          PV= 1. Ne8 <-2.00>
 2.   -1.18  00:00:00.21  1099         PV= 1. Ne8 Qf8 <-2.00>
 3+   -0.35  00:00:00.30  3727         Qg6! ++
FORCED MATE --> stop searching now
 3.  +MATE1  00:00:00.40  8406         PV= 1. Qg6 fxg6 2. Nxg6+ [+] [MATE]
tellothers depth=3, score=+MATE1, n=8406, nps=20863, PV= 1. Qg6 fxg6 2. Nxg6+ [+] [MATE]
move g3g6
...
```

In the above example the user first types **`st 5`**. This sets the clock to fixed-time-per-move mode and tells the engine to search exactly five (5) seconds per position. Next the user types **`script /typhoon/tests/wac.epd`**. This begins the execution of a script file. From now on typhoon will take commands from this script file one at a time until the file has been completely executed. Commands that typhoon reads from a script have "SCRIPT>" printed before them. The script uses **`setboard`**, **`solution`** and **`id`** to set up a test position, a solution move, and the position name. Then the script sends typhoon the **`go`** command and the engine starts thinking. Because the engine has been told to search exactly five (5) seconds per move, it will stop searching this position when five (5) seconds are up. At this time it will count the position as solved if the move it plays matches any of the solution move(s) (or does not match any of the avoid move(s)). It also keeps track of how long it took to see the correct move. Once a script has finished running completely typhoon will print out some final statistics about the test suite run and begin to watch the keyboard for commands again.

```
TEST SCRIPT execution complete.  Final statistics:
    correct solutions   : 272
    incorrect solutions : 28
    total problems      : 300
    total nodecount     : 40376223
    avg. search speed   : 125601.9 nps
    avg. solution time  : 0.4 sec
    avg. 1st move beta  : 0.936

Time-to-solution histogram:
00.00s .. 00.20s: ******************************* (73)
00.20s .. 00.40s: ************************************************** (119)
00.40s .. 00.60s: ************** (31)
00.60s .. 00.80s: ******* (15)
00.80s .. 01.00s: *********** (34)
   not solved   : **************** (28)
```

6.2. Test Suite Results

Here are some test suite results obtained by typhoon running on my development machine which is an AMD Athlon XP 1.533Ghz / 512Mb. In the table the suite abbreviation "ECM" refers to "Encyclopedia of Chess Middlegames" (http://wannabe.guru.org/scott/hobbies/ECM.EPD) and "WAC" refers to "Win At Chess" (http://wannabe.guru.org/scott/hobbies/chess/WAC.EPD)

suite	sec/move	score	date	hardware
ECM	20	674 / 879	Dec 22, 2004	Dual 1.533ghz Athlon MP
WAC	20	296 / 300	Jan 11, 2002	Dual 1.533ghz Athlon MP

Chapter 7. Advanced WinBoard Configuration

This chapter covers how to get typhoon running under **WinBoard**, a free, open source graphical front-end for chess engines. For more information about **WinBoard** or to download it, see http://www.tim-mann.org/xboard.html. This chapter is also relevant to **xboard**, WinBoard's X11-based version. More information about these chess GUI programs is also available in Aaron Tay's WinBoard FAQ (http://www.aarontay.per.sg/Winboard/Winboard.html).

7.1. Playing Locally

Probably the most common way you will want to use typhoon with **WinBoard** is to play local chess matches against the program using **WinBoard** as the display. To do this you will need to use **WinBoard's** `-fcp` and `-fd` commandline options like this:

```
winboard -fcp "typhoon" -fd typhoon_directory
```

The `-fcp` option sets **WinBoard's** *first chess program* and the `-fd` sets the *first chess program directory*. You should use the directory you installed typhoon in after the `-fd` option. Once **WinBoard** is running, set the "Machine Plays Black" or "Machine Plays White" menu option.

Probably the best way to remember this is to create a little batch file to start up **WinBoard** for playing locally against typhoon.

> **Note:** Typhoon's thinking lines may not be compatible with **WinBoard** and can cause it to crash when you enable the "Show Thinking" menu option. I advise you not to do this.
>
> Also, typhoon's support for "Analysis Mode" in **WinBoard** is incomplete at the time of writing and may lead to problems. Please don't enable either of these WinBoard features for now. Both shortcomings will be addressed in future releases of the engine.

If everything is working right typhoon should play instantly in the early moves of a new game. If typhoon has to think for more and a second or two about early moves it may not see the opening book.

If you run into trouble getting typhoon to work under **WinBoard** I suggest looking at `typhoon.log`. This file, which is created by typhoon every time it is invoked, will contain a record of the interaction between the engine and **WinBoard** and may help to isolate the problem. You can also enable **WinBoard** logging by using the `-debug` commandline option to **WinBoard**. This will cause the creation of `winboard.log` which also may help you track down the problem.

7.2. Engine vs. Engine Matches

With **WinBoard** it's possible to play one chess engine against another. This section explains how to set this up and how to run engine vs. engine matches as fairly as possible.

To run two engines under **WinBoard** just use both the `-fcp` and the `-scp` commandline options. `-fcp` sets the *first chess program* while `-scp` sets the *second chess program*. You may want to use the related `-fd` and `-sd` options too. Then, once **WinBoard** is running, set the "Machine Plays Both" option and watch them fight.

It's really not advisable to test engines by playing them under WinBoard on a single processor machine. However running on a two single-threaded engines on a multi-processor (or multi-core) machine is a great way to test engines.

If you do choose to run an engine vs. engine match on a single processor machines you should turn off pondering on both engines so that they do not constantly compete for system processor resources. The next step is to make sure that each engine has equal access to the machine's memory. When playing engine-engine matches, you should set the hash table sizes of both engines by hand to roughly half the total memory on the machine. Be absolutely sure that combined memory requirements of the two engines does not exceed the amount of physical memory on your machine or you will run into swapping which will slow the engine(s) down severely.

7.3. Playing on an Internet Chess Server

Internet Chess Servers are machines on the Internet that allow multiple people/engines to connect and play chess against one another. **WinBoard** knows how to connect to and communicate with these servers and therefore you can use it to play typhoon against other people and machines connected to the same Internet Chess Server. This is a good way to test two chess engines because when they play over the Internet, unlike when they both play on the same machine, they do not have to fight each other for access to the processor and memory resources of a single computer.

> **Note:** It's unethical and against the rules of most Internet Chess Servers to use a computer engine to make moves on a human account. Your human rating should not be aided by machine and your opponents have the right to know they are playing against a computer. Most Internet Chess Server administrators are pretty good at catching people who cheat and banning them from playing. Please don't use typhoon to cheat on Internet Chess Servers! Likewise, if you are running typhoon on a chess server I would be grateful if you would list the program name and hardware configuration in your account's finger notes.

To connect to an Internet Chess Server with typhoon+WinBoard use the following commandline options when invoking **WinBoard** in addition to the normal `-fcp` and `-fd` options described in the last sections: `-zp` (enable chess engine to ICS code), `-ics` (connect to ICS server), `-icshost` (ICS server hostname), `-xpopup` (no popup windows), `-xautoraise` (no popup windows), `-xexit` (no popup windows), and `-reuse` (no need to restart the engine process, reuse the current one). You may also want to use `-icsHelper` (program name to use to connect to the ICS, something like timestamp.exe or timeseal.exe), `-zippyPassword` and `-zippyPassword2` (passwords are for controlling the chess engine remotely). For more information about what these commands do, see the documentation in `winboard.hlp` and `README.zippy`, both of which are included with **WinBoard**.

Here's a sample script based on the one I use to connect typhoon to the Internet Chess Club (http://www.chessclub.com). This script loops forever restarting the engine and **WinBoard** if they lose connection to the ICS server or terminate for some reason. You can break out of the script by pressing ^C.

```
@echo off
:again
cd /d C:\typhoon
"C:\program files\winboard\winboard.exe" /zp /ics /icshost
 chessclub.com /icshelper C:\progra~1\winboard\timestamp.exe /fcp
 C:\typhoon\release\typhoon.exe /xzt /xexit /xpopup /xautoraise
```

```
 /reuse /debug /zippyPassword xxx /zippyPassword2 yyy
sleep 3
kill -f typhoon.exe
kill -f winboard.exe
sleep 3
goto :again
```

There are many different Internet Chess Servers on the 'Net. Some are free, others cost money to use. Some are very busy and others are pretty sparsely used. There's a list of chess servers on Tim Mann's website at http://www.tim-mann.org/ics.html.

Chapter 8. Testing Typhoon

Typhoon was released in the present, somewhat buggy state mainly so that I could benefit from a large pool of testers. Therefore, I am very interested in reports about how the engine does. This chapter is about how you can help test typhoon. Thanks for helping to make the engine stronger!

8.1. A DEBUG build

With a simple modification to the build instructions presented in an earlier section it is possible to produce a DEBUG build of the chess engine. A DEBUG build is an engine that runs much more slowly than normal because it is very carefully double checking every calculation for errors. If an error is found a DEBUG build makes it easier to understand and fix the problem than a normal build.

In order to make your own DEBUG build, just add **`DEBUG=1`** to the commandline when you build the engine. The engine will be called `_typhoon` instead of the usual `typhoon`.

I try to run DEBUG builds of the engine for long periods of time in order to expose bugs. If you have a spare machine and some time it would be great if you would be willing to do the same. If your DEBUG build crashes, send me an email (mailto:scott@wannabe.guru.org) with the message and output of the **`version`** command.

8.2. A TEST build

A TEST build is one that has some extra testcode compiled into it. TEST builds will also run the testcode once automatically at startup. I run TEST builds periodically in order to make sure that I have not broken anything with my changes. TEST builds are also helpful when you are trying to port the engine to a new platform; if you get a TEST build to run cleanly (especially a TEST/DEBUG build) then you can be pretty sure the port is good.

If you want to build your own TEST engine, just add **`TEST=1`** to the **make** commandline.

8.3. An EVAL_DUMP build

An image built with the **`EVAL_DUMP`** preprocessor symbol defined will output several debugging messages every time it runs the evaluation routine (see `eval.c`). It will also break down the terms contributing to the eval score when the user types the **`dump eval`** command.

These features are useful if you are curious as to what the engine thinks of a position. Of course, it doesn't make sense to run the evaluator on a position that is not quiet.

Since the engine produces output everytime it runs an evaluation and because the engine typically runs the evaluation routine hundreds of thousands of times per second, you should not attempt to search with an engine build using the EVAL_DUMP flag. It's too verbose.

If you find a position where you think the engine's evaluation is totally wrong, please send it to me. While I can't promise anything (some positions are too complicated for static analysis), I will definitely take a look.

8.4. Typhoon Crashes

The engine should hardly ever crash. If you find a reproducible way to crash the engine I definitely want to hear about it. Please send me an email (mailto:scott@wannabe.guru.org) describing how you managed to crash the engine and the output of the **`version`** command.

If you have a `typhoon.core` file, send that along with a copy of your `typhoon` image. Better yet, build a DEBUG build, crash it, and send me the `_typhoon.core`, and `_typhoon` files. That makes it easier to figure out what went wrong.

8.5. Typhoon Blunders

While not as severe as outright engine crashes, I am also interested in positions where the program makes a terrible move. I'm not a strong chessplayer so your impressions about the engine's strengths, weaknesses, trends that indicate errors in judgment, and so on are also very welcome. Please email them to me (mailto:scott@wannabe.guru.org) along with the output of the **`version`** command.