

This is version 3.6 of KnightCap. There have been substantial changes since version 2.4, hence the new major number. The main difference is KnightCap now learns its eval. See below for details.

General info:

KnightCap is a chess program written for the Fujitsu AP1000+ parallel computer (running AP/Linux). It will also run on most unices, although you may need to tweak the includes.h file and Makefile.

The principal differences between KnightCap and other chess programs are:

KnightCap has an optional fully rendered 3D interface, giving a feel much more like an "over the board" game.

KnightCap was developed to run on a parallel, distributed-memory machine, although it also runs on normal Unix boxes.

KnightCap does not have an opening book---instead it keeps a file (brain.dat) of losing moves and inserts them in the hash table at the start of each search. At present it has about 1500 entries, and this makes it a pretty competitive opening player.

KnightCap learns the parameters of its evaluation function as it plays. The most dramatic example of how this helps is an experiment we conducted on FICS in which KnightCap learnt from a 1650 player to a 2100 player in just 300 games. See <http://keating.anu.edu.au/~jon/papers/knightcap.ps.gz> for more info on its learning algorithm.

KnightCap now beats gnuchess consistently and is within "coo-ee" of crafty, although I think it needs deeper search or some more dramatic selective search to be truly competitive with the best micro programs.

If you have comments/suggestions etc then send them to [Andrew.Tridgell@anu.edu.au](mailto:Andrew.Tridgell@anu.edu.au) or [Jon.Baxter@anu.edu.au](mailto:Jon.Baxter@anu.edu.au). Andrew gets tons of email about some other bits of software he's written so please be patient if he is very slow in replying.

KnightCap is available under the GNU public license.

The original chess engine and 3-D interface was written by Andrew Tridgell. Jonathan Baxter added the learning and patches to the eval, move generation, and various other bits and pieces.

KnightCap currently plays on FICS ([ics.onenet.net/5000](http://ics.onenet.net/5000)) under the pseudonyms KnightCap and WimpKnight (experimental version), and on ICC ([chessclub.com](http://chessclub.com)) under the pseudonym KnightC. It currently has a rating of around 2300 on FICS and 2500 on ICC (they are quite different versions---this README refers to the current ICC version).

### Building KnightCap:

If you want the fancy graphics display then you will need an OpenGL graphics library and the SGI Glut toolkit. The free Mesa library is fine.

Edit the Makefile, following the comments.

Then type "make". When it doesn't work fix the problems or call on a local C programming guru.

The code currently assumes you have a 32 bit machine and that "long long" is a 64 bit quantity. We'll probably fix this sometime.

### Running KnightCap -----

After you start it up use the right button to access the main menu. Its obvious from there.

You can rotate the board by dragging with the left mouse button. Move pieces by dragging on the piece, also with the left mouse button.

If you don't compile with the fancy graphics then your get a simple color ascii display. If you have a mono terminal then start with the -B option.

You should also try the "help" command once you start it up, and the -h switch on the command line when you run it. There are lots of options you can set.

For "normal" play I recommend at least the -A switch (so that it thinks on opponents time) and the -H switch, which is used to specify how much memory to use for the hash tables.

KnightCap needs heaps of memory for hash tables. It uses a memory intensive variant of alpha-beta so it will be much more affected by lack of memory than other programs. Try to give it at least 16MB, preferably more, but don't specify so much that it begins to swap!

### KnightCap's Learning:

KnightCap currently has two different kinds of learning: opening book and evaluation function learning. The opening learning works as follows. After each game KnightCap loses, it inserts the losing move (see analyse\_game in brain.c for how this is calculated) in a permanent brain file (brain.dat). These entries are inserted into KnightCap's hash table at the start of each search. This means KnightCap will avoid playing the same losing line again.

The evaluation function learning is also applied at the end of each game, although updates only occur every 10 games at the moment (You can alter this with the

MAX\_ROUNDS constant in td.c - **now updates every game - J.A**). It works on the principle of temporal differences, that is, it updates the coefficients in such a way as to try to keep the evaluation function constant throughout the game. See td.c for details. You can turn this on or off in the Makefile. The hash table is much smaller with it on, and this can affect performance. Three files are written out at the end of each game: large\_coefs.h, small\_coefs.h and coefs.dat. These files all contain the same information: the value of KnightCap's evaluation function coefficients. The two .h files are included by eval.c ("large\_coefs.h" contains the coefficients in units where 1 pawn = 10000, small\_coefs.h has 1 pawn = 100). coefs.dat is a binary file that gets read when KnightCap starts up, and overwrites the values specified in the .h files. If you have a favourite coefficients file, make sure you save it in a different directory so KnightCap doesn't overwrite it.

Note that KnightCap has 4 different stages in its eval: OPENING, MIDDLE, ENDING and MATING. MATING stage coefficients are never altered---most of them are ignored as the mating stage code is designed mostly to drive the opponent's king into the corners. All the other coefficients can change if you have evaluation learning turned on.

At present, KnightCap uses a couple of scratch files to store coefficient update information during and between games: update.dat, grad.dat and rounds.dat. You can look in wnorm.dat to see the L1 norm between KnightCap's current coefficient vector and the one it was compiled with, and angle.dat to see the change in angle (in radians) between KnightCap's current coefficient vector and the one at the last update.

#### Modifying the Eval:

Perhaps the easiest and most fun place to start modifying KnightCap is in its evaluation function. With the learning, you can add new features and corresponding coefficients, set the new coefficients to plausible values (remembering that 1 pawn = 10000) or just 0, and then set KnightCap playing on ICS or against some other program and watch it learn the coefficient values. You can control the learning rate with the TD\_ALPHA constant in td.c. You can also vary the TD\_LAMBDA parameter, which controls whether the evaluation function is adjusted to predict only the eval at the next move (TD\_LAMBDA = 0) or the outcome of the game (TD\_LAMBDA = 1). Values in between interpolate these two behaviors.

Adding new coefficients is particularly easy. Just edit large\_coefs.h and insert the new coefficients wherever you want, remembering to add a comment with the name in it, and also to add the same coefficients in each of the four stages (stages are delimited by /\* %OPENING% \*/ , /\* %MIDDLE% \*/ , etc). Then compile "extract.c" (in UTILS) and run it in the directory where the new large\_coefs.h is. This will automatically generate eval.h and names.h which contain the new names and ordering for the coefficients (you might want to save backups of these). Once you have generated these, you can compile KnightCap and away you go.

### Using with Xboard:

Yes, KnightCap can be used with XBoard. Start KnightCap with the -n and -X options on the command line. For example

```
xboard -fcp 'KnightCap -n -X -A'
```

then play as usual. KnightCap doesn't yet support editing positions (except when connected to FICS). It does understand time controls etc and most other stuff that XBoard throws at it.

### Playing against other programs (e.g Crafty):

You can use xboard to run matches between Knightcap and other programs like crafty (or even KnightCap itself). This is particularly useful if you have introduced new coefficients and want to look at how they get learnt, or if you mess with other areas of the program and want to test it. In the Test subdirectory you'll see a script "matchN" which tells you how to do it.

### Playing on ICS:

Yes, KnightCap can be used as an interface to ICS. Its a bit rudimentary at the moment, and is not really intended for general consumption but it does work.

To start it up use the -c option like this:

```
KnightCap -c 'telnet ics.onenet.net 5000'
```

Make sure you use board style 12 (use "set style 12" once logged in).

The ICS interface was designed to let KnightCap play unattended, not as a general ICS interface, so you may find bugs.

We have also provided two Bourne shell scripts "startics" and "startfics" to give you an idea of how to run KnightCap automatically on ICS.

### Mesa and Glut:

To build KnightCap with the 3D display you need a OpenGL compliant graphics library and the Glut toolkit from SGI. Both of these are available freely on the net. To help make things a bit easier for some people I have included pre-compiled versions of these

libraries on the KnightCap ftp site for Linux-intel and Solaris2-sparc systems. Use these pre-compiled packages only if you are lazy and short on time.

You may also need to set your LD\_LIBRARY\_PATH environment variable to point at the Mesa-2.1/lib directory so that the dynamic loader can find the libraries when KnightCap starts up.

Source code:

The source code for KnightCap is available from <http://syseng.anu.edu.au/lsg/KnightCap> or <ftp://samba.anu.edu.au/pub/KnightCap>

Feedback and patches are welcome.

Andrew.Tridgell@anu.edu.au Jon.Baxter@anu.edu.au