egdb.dll, egdb64.dll

Description

This library provides an API into several checkers endgame databases.
These include:

- Kingsrow 10-piece wld database version 1 (filenames ending in .cpr, .idx).
- Kingsrow 10-piece wld database version 2 (filenames ending in .cpr1, .idx1).
- Kingsrow 10-piece mtc database.
- Kingsrow 8-piece dtw database.
- Cake 8-piece wld database built by Martin Fierz.
- Chinook 8-piece wld database built by Jonathan Schaeffer.
- Kingsrow Italian 9-piece and partial 10-piece wld database version 1 (filenames ending in .cpr, .idx).
- Kingsrow Italian 10-piece wld database version 2 (filenames ending in .cpr1, .idx1).
- Kingsrow Italian 10-piece mtc database.
- Kingsrow Italian 8-piece dtw database.
- Chinook Italian 6-piece wld (note: as of 4/9/2005 this db has errors).

The API has functions to

- identify a database and return its attributes
- open a database for lookups
- lookup values in an open database
- get statistics on database lookups, cache hits, misses, etc.
- perform a crc check on each database file
- close a database

To use the library, include the header file egdb.h in the application source code and link with the lib file egdb64.lib (or egdb.lib for 32-bit builds). egdb64.dll (or egdb.dll for 32-bit executables) must be somewhere in the search path for dynamic libraries when the application runs. The .dll files are placed in the CheckerBoard directory when you install Kingsrow.

To lookup values in a database, it should first be opened using egdb_open(). This returns a handle to an open database. The handle is a pointer to a structure of db access function pointers. These access functions are then used for all subsequent communication with the driver. The handle also holds all state data that is used by each instance of an open driver, thus multiple handles can be opened from a single process without conflicting with one another. These drivers are safe for multi-threading.

Example program

This package contains an example program that demonstrates the use of the library. A project file for Visual Studio 2015 is included. The program opens a WLD database, reads some positions from a table, calls lookup() to get the database value of each position, and compares it to the known value that is also in the table. You can use this to verify that basic functions of the driver are working on your system.

The example is a console application. You should run it from a Command Prompt window. Since it uses egdb64.dll (or egdb.dll), this file should either be in the directory that you run from, or in a directory that Windows searches for dll files. The dll file is placed in the CheckerBoard directory when you install Kingsrow.

Excluded positions

The WLD databases do not have valid data for positions that are a capture, or for positions that would be a capture for the opposite side-to-move. There is no way to know this from the lookup() return value. It will be a win, loss, or draw value, but it will not be the correct value except by random luck. It will be whatever makes the compression work best. You should test that a position is not a capture or a non-side capture before calling lookup().

Version 2 of the Kingsrow WLD databases, both English and Italian types, only have data for one of the side-to-move colors in some 9 and 10 piece position subsets. If you call lookup for one of these positions with the excluded color, it will return the value EGDB_UNKNOWN.

The Kingsrow DTW databases only have data for one of the side-to-move colors in all subsets. If you call lookup for one of these positions with the excluded color, it will return the value EGDB_UNKNOWN. These databases also do not have data for positions that are captures, and there is no way to know this from the lookup() return value. You have to test that the position is not a capture before querying a DTW value.

*EGDB_API int __cdecl egdb_identify(const char *directory, EGDB_TYPE *egdb_type, int *max_pieces);*

egdb_identify() checks if an endgame database exists in directory. If so, the database type is written to egdb_type, and the maximum number of pieces for which the database has data is written to max_pieces. The function return value is 0 on success, or non-zero if no database is found in directory.

*EGDB_DRIVER *__cdecl egdb_open(EGDB_BITBOARD_TYPE bitboard_type, int pieces, int cache_mb, const char *directory, void (__cdecl *msg_fn)(char *));*

This function opens a database, allocates memory for caching db values, and reads indexing files. The driver normally emits status messages when it starts up, with information about files opened and memory used. If any errors occur, either during the open() call or in subsequent calls through the driver handle, the details will be reported through the message function.

The return value is a handle that can be used for subsequent access to the database. A NULL return value means that an error occurred. The likely reasons for errors are either the wrong directory path was given for the database files, or the driver was unable to allocate all the memory that it needed from the heap. If a NULL pointer is returned by the open function, an error message will be passed back to the application through the callback message function.

The first argument to egdb_open() is an enum which tells the driver which type of position bitboards will be used for lookups. The driver will tailor the lookup function pointer for the particular bitboard type that is requested. The driver can accept positions in either Kingsrow (EGDB_NORMAL_BITBOARD) or Cake (EGDB_ROW_REVERSED_BITBOARD) format. These formats are described in egdb.h.

The second argument to egdb_open() is the maximum number of pieces for which the database is being opened for lookups. Normally this will be 8 or 10, but a smaller number of pieces can be given. The driver will use less ram and initialize more quickly when a smaller than maximum number of pieces is given. The egdb_open function will error if a pieces argument is given that is greater than the capability of the database. The attributes of a database can be queried using the function egdb_identify before calling egdb_open.

The third argument to egdb_open() is the number of megabytes (2^20 bytes) of heap memory that the driver will use. The driver will use some minimum amount of memory even if you give a value which is lower than this minimum. For a database that is being opened to use 8 pieces this minimum is between 35 and 40mb, which includes 10mb of cache buffers. Any memory that you give above this minimum is used exclusively to create more cache buffers. Cache buffers are allocated in 4k blocks. If the memory argument is large enough the driver will allocate static buffers for some of the smaller subsets of the database. In the statistics counters these are called 'autoload' buffers.

The last argument to egdb_open() is a pointer to a function that will receive status and error messages from the driver. The driver normally emits status messages when it starts up, with information about files opened and memory used. If any errors occur during the open call, the details will be reported through the message function. A message function that writes messages to stdout can be provided like this:

```
void print_msg(char *msg)
{
    printf(“%s”, msg);
}
```

which can be passed to a driver like this:

```
handle = egdb_open(EGDB_NORMAL, 10, 5000, "c:/db_english/wld_v2", print_msg);
```

This callback function is also used to report error messages during lookups. In Kingsrow I write these messages to a log file.

*int (__cdecl *lookup)(struct egdb_driver *handle, EGDB_BITBOARD *position, int color, int cl);*

The lookup() function returns the value of a position in an open database. It takes 4 arguments -- a driver handle, position, color, and a conditional lookup boolean.

- handle is the value returned by egdb_open().
- position is the position to lookup.
- color is the side-to-move, either EGDB_BLACK or EGDB_WHITE.
- cl is the conditional lookup argument. If it is non-zero, then the driver will only get the value of the position if the position is already cached in ram, otherwise EGDB_NOT_IN_CACHE will be returned. If the conditional lookup argument if zero, the driver will always attempt to get a value for the position even if it has to read a disk file to get it. In kingsrow I calculate a priority for each position that gets passed to handle->lookup(). If the priority is low, I set the conditional lookup argument non-zero, else I set it zero to unconditionally lookup the position.

For the WLD databases, the values returned by lookup are defined by the enums EGDB_UNKNOWN, EGDB_WIN, EGDB_LOSS, EGDB_DRAW, and EGDB_NOT_IN_CACHE.

For the MTC databases, the values returned by lookup are either the number of plies to a conversion move, or the value MTC_LESS_THAN_THRESHOLD for positions which are close to a conversion move. The Kingsrow MTC databases do not store positions which are closer than MTC_THRESHOLD plies to a conversion move. The lookup function will only return even values, because the database represents the value internally as (distance / 2). The true value is either the value returned, or (value - 1). An application can infer the true even or odd value of a position by looking up the values of the position's successors. If the best successor value is the same as the position's, then the position's true value is 1 less than the returned value. The MTC database only has values for positions that are a win or a loss. You cannot tell this from the return value from lookup(). Before calling lookup in the MTC database, check that the position is a win or a loss using the WLD database.

For DTW databases, the values returned by lookup are (number_of_plies_to_win_or_loss / 2). To convert the return value to the actual depth to win or loss, multiply by 2, and add 1 if the position is a win. Depths to win are always and odd number of plies, losses an even number of plies. Queries on positions that are draws return a meaningless value, and you cannot know this from simply looking at the return value. You must query a WLD db first. Similarly, positions that are captures do not have valid db data and return a meaningless value. Also keep in mind that the DTW db has data for only one side to move in each db subset. Lookup will return EGDB_SUBDB_UNAVAILABLE if you query a position for which no data is available for that color. See the section “Excluded positions” above. In general, to obtain the DTW value of a position that is excluded from the db because of side-to-move, or capture status, do a recursive query of successors until a valid DTW value can be propagated back to the root position.

All the WLD databases supported by this driver (Kingsrow, Cake, and Chinook) will not return valid values for positions where either the side to move has a capture or where the opposite side would have a capture if it was that side's turn to move. There is no way to tell this from the return value, which will normally be one of the win, loss, or draw values. The application must test for these captures and avoid calling lookup for any positions where they are present. The MTC databases do not have this restriction.

The Cake database does not have data for positions where one side has more than 4 pieces. If you give the lookup function a position where one side has more pieces than it knows about, it returns the value EGDB_UNKNOWN. The Chinook database has all positions for up to 6 pieces, but the 7 piece and 8 piece slices do not have data for more than 4 pieces on a side. The Kingsrow databases have data for positions with up to 5 pieces on a side.

*int (__cdecl *verify)(struct egdb_driver *handle);*

The driver has tables of good crc values for all db files. The verify() function performs a crc check on each index and data file in the open database. It returns 0 if all crcs compared ok. If any file fails the check, the verify call returns a non-zero value, and error messages are sent through the callback msg_fn that was provided when the driver was opened.

*int (__cdecl *close)(struct egdb_driver *handle);*

The close() function frees all resources used by the driver. If the application needs to change anything about an egdb driver after it has been opened, such as number of pieces or megabytes of ram to use, it must be closed and then opened again with the new parameters. Close returns 0 on success, non-zero if there are any errors.

*void (__cdecl *reset_stats)(struct egdb_driver *handle);*

*EGDB_STATS *(__cdecl *get_stats)(struct egdb_driver *handle);*

The API provides a couple of functions for gathering statistics on db lookups. get_stats() returns a pointer to a structure of various counts that have accumulated since the counts were last reset. reset_stats() resets all the counts to 0.