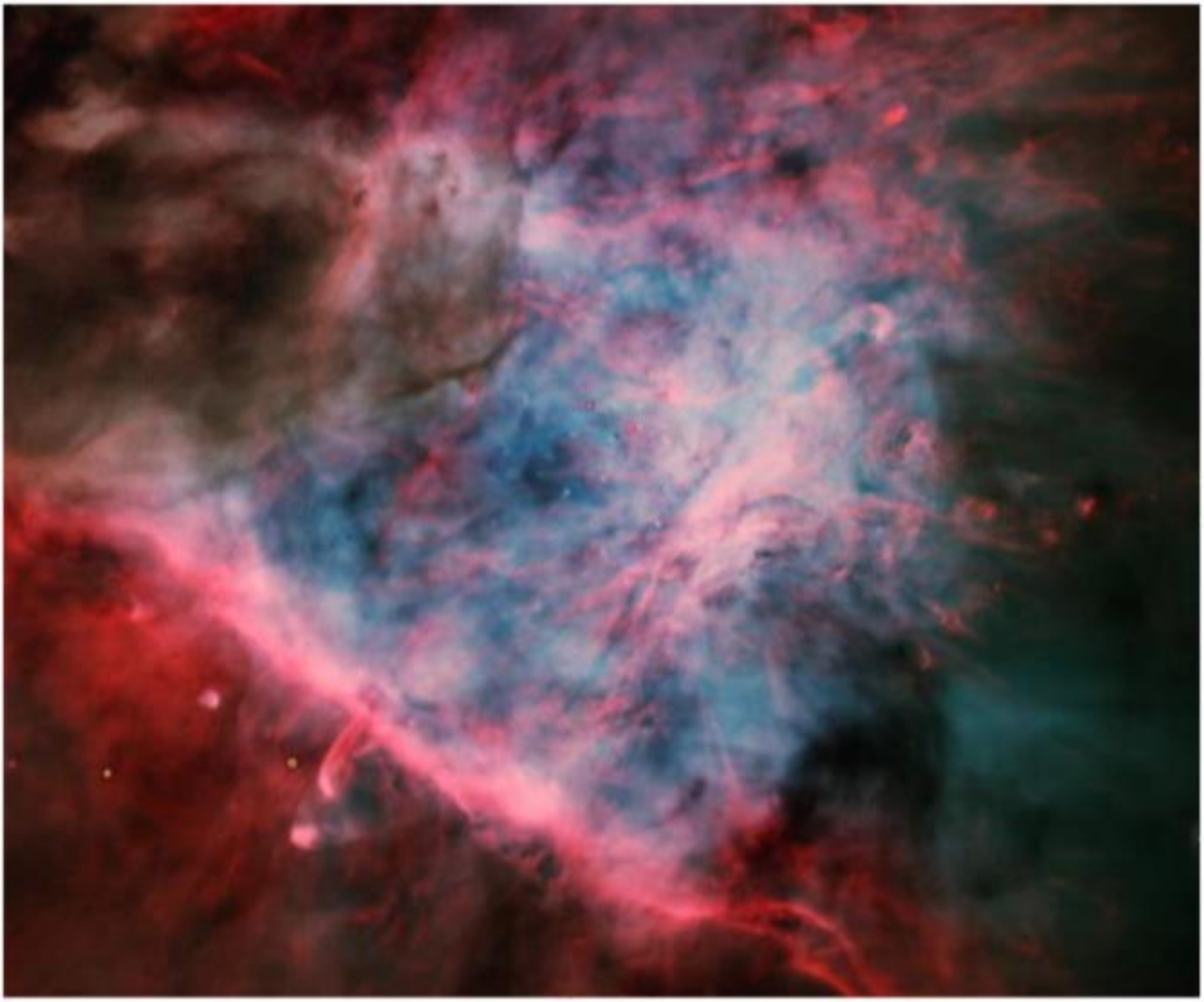

M42

A C++ LIBRARY FOR BITBOARD ATTACK MASK GENERATION

Syed Fahad | sydfhd@gmail.com | April 13, 2016

Introduction

M42 is a small library for Bitboard Attack Mask Generation. It is intended to be used by beginner/novice chess programmers. The purpose of this library is to provide the fastest attack mask generation methods available to chess programming amateurs, and to a save time and energy on coding and understanding those methods.

Getting Started

Using M42 library is as simple as it can be, just include the header file (#include "m42.h") and compile "m42.cpp" along with the source files of your program. No DLLs, no fancy precompiled libraries, no special command lines. You also need to call `M42::init()` in the main function of your program.

SUPPORTED COMPILERS

M42 is intended to be used under a variety of C++ compilers on a variety of system. M42 has been tested on g++ and Visual C++ compilers. M42 should compile fine under pretty much every C++ compiler, and if it doesn't, you can of course email your issues (see the cover page).

EXAMPLE.CPP

A file by the name "example.cpp" is included to demonstrate the usage and some functions of M42 library. You don't need to set any special command line options to compile it. Just compile it like any other source file. For example, under g++, use the command line "g++ m42.cpp example.cpp –o m42" and then run the compiled program by "./m42".

Functions provided by M42 library

The M42 library provides a handful of functions for attack mask generation of all standard chess pieces.

SLIDER ATTACK MASK GENERATION FUNCTIONS

For rooks, bishops, and queens, use the functions `M42::rook_attacks(<square>, <occupancy>)`, `M42::bishop_attacks(<square>, <occupancy>)`, `M42::queen_attacks(<square>, <occupancy>)`, respectively. These functions take the square of the piece as their first argument (a1 = 0, a2 = 1, ..., h8 = 63), and the occupancy bitboard as their second argument. Note that if the piece itself is included in the occupancy bitboard, it would cause **NO** problem. See the function reference table below for complete list of functions.

NON-SLIDER ATTACK MASK GENERATION FUNCTIONS

In this category, M42 provides functions for handling single pieces as well as multiple pieces. Function that work for multiple pieces are particularly useful in evaluation.

Pawns

For pawn attacks, use the function `M42::pawn_attacks(<color of pawn>, <square of pawn>)`. The first argument, the color of pawn, must be `0` if the color is white, or `1` if the color is black. You also have the template function `M42::calc_pawn_attacks(<pawns bitboard>)` which takes the color as template argument, and the pawns bitboard which may contain multiple pawns.

King and Knight

M42 provides functions `M42::king_attacks(<square of king>)` and `M42::knight_attacks(<square of knight>)` for attack mask generation of kings and knights respectively, given the square of the piece. You also have `M42::calc_king_attacks(<bitboard of kings>)` and `M42::calc_knight_attacks(<bitboard of knights>)` for attack mask generation of *multiple* kings and knights, particularly useful in evaluation of attacks. These functions, although slower than their single-piece attack mask generator brothers, are still very fast.

Function Reference Table

FUNCTION NAME	DESCRIPTION	INPUT	OUTPUT	NOTE
`uint64_t M42::king_attacks(int sq)`	Returns the bitboard of squares that're attacked by a king on the given square	The square of the king	Bitboard of all the squares that the king can *potentially* move to	
`uint64_t M42::calc_king_attacks(uint64_t kings)`	Returns the bitboard of squares that're attacked by the kings	The bitboard of the kings	Bitboard of all the squares that the kings can *potentially* move to	This function can handle *multiple* kings so it's slower than **king_attacks()**
`uint64_t M42::knight_attacks(int sq)`	Returns the bitboard of squares that're attacked by a knight on the given square	The square of the knight	Bitboard of all the squares that the knight can *potentially* move to	
`uint64_t M42::calc_knight_attacks(uint64_t knights)`	Returns the bitboard of squares that're attacked by the knights	The bitboard of the knights	Bitboard of all the squares that the knights can *potentially* move to	This function can handle *multiple* knights so it's slower than **knight_attacks()**
`uint64_t M42::pawn_attacks(int cl, int sq)`	Returns the bitboard of squares that're attacked	**cl** as the color of the pawns (0 for white, 1 for	Bitboard of the squares *potentially attacked* by the pawn	

	by the pawn of the given color on the given square	black), and **sq** as the square of pawn		
`template <unsigned Cl> uint64_t M42::calc_pawn_attacks(uint64_t pawns)`	Returns the bitboard of squares that're attacked by the pawns of the given color	**Cl** as the color of the pawns (0 for white, 1 for black), and **pawns** as the bitboard of the pawns	Bitboard of the squares *potentially attacked* by the pawns	This function can handle *multiple* pawns so it's slower than **pawn_attacks ()**
`uint64_t M42::rank_attacks(int sq, uint64_t occ)`	Returns the bitboard of squares of the rank (of the given square) that're attacked by the rook on the given square and occupancy	**sq** as the square of the rook, and **occ** as the bitboard of occupancy	Bitboard of all the squares in the rank of the given square that the rook can *potentially* move to	
`uint64_t M42::file_attacks(int sq, uint64_t occ)`	Returns the bitboard of squares of the file (of the given square) that're attacked by the rook on the given square and occupancy	**sq** as the square of the rook, and **occ** as the bitboard of occupancy	Bitboard of all the squares in the file of the given square that the rook can *potentially* move to	
`uint64_t M42::calc_rook_attacks(int sq, uint64_t occ)`	Returns the bitboard of all the squares that're attacked by the rook on the given square and occupancy	**sq** as the square of the rook, and **occ** as the bitboard of occupancy	Bitboard of all the squares that the rook can *potentially* move to	Use the faster **rook_attacks ()** instead
`uint64_t M42::diag_attacks(int sq, uint64_t occ)`	Returns the bitboard of squares of the anti-diagonal (of the given square) that're attacked by the bishop on the given square and occupancy	**sq** as the square of the bishop, and **occ** as the bitboard of occupancy	Bitboard of all the squares in the file of the given square that the bishop can potentially move to	
`uint64_t M42::adiag_attacks(int sq, uint64_t occ)`	Returns the bitboard of squares of the diagonal (of the given square) that're attacked by the bishop on the given square and occupancy	**sq** as the square of the bishop, and **occ** as the bitboard of occupancy	Bitboard of all the squares in the file of the given square that the bishop can potentially move to	
`uint64_t M42::calc_bishop_attacks(int sq, uint64_t occ)`	Returns the bitboard of all the squares that're attacked by the bishop on the given square and occupancy	**sq** as the square of the bishop, and **occ** as the bitboard of occupancy	Bitboard of all the squares that the bishop can *potentially* move to	Use the faster **bishop_attacks()** instead

`uint64_t M42::rook_attacks(int sq, uint64_t occ)`	Similar as **calc_rook_attacks**, but faster	See **calc_rook_attacks()**	See **calc_rook_attacks()**
`uint64_t M42::bishop_attacks(int sq, uint64_t occ)`	Similar as **calc_bishop_attacks**, but faster	See **calc_bishop_attacks()**	See **calc_bishop_attacks()**
`uint64_t M42::queen_attacks(int sq, uint64_t occ)`	Returns the bitboard of all the squares that're attacked by the queen on the given square and occupancy	**sq** as the square of the queen, and **occ** as the bitboard of occupancy	Bitboard of all the squares that the queen can *potentially* move to

Copyright

More about M42

WHY I WROTE M42

I wrote M42 because of unavailability of complete Bitboard attack mask generation libraries. In my experience, most beginner chess programmers face two stumbling blocks in Chess Programming – one, alpha-beta search and two, magic hashing based move generation. Often after spending much effort on studying magic hashing based move generation, they simply give up and settle on slower alternative methods. Some others go for Pradu Kannan's library for slider pieces but I feel that it's too ugly code for my taste (I do not, in any way, mean to insult Mr. Pradu Kannan by this). I have spent most of my Chess Programming time trying to experiment with different ways of move generation. So it was natural for me to go ahead to write M42.

HOW YOU CAN IMPROVE M42

- Add functions to handle multiple slider pieces by flood fill algorithms (very easy to do, see this and this for how to do it)
- Add support for hardware instructions (esp. `_pext_u64()` for magic move generation)
- Optimize for 32-bit environments
- Maybe extend this library to help in evaluation

If you manage to do something interesting with this library, do drop me a line at my email (see front page).

Postscript

Hope you enjoy using M42 as much as I enjoyed developing it!