Table of Contents

Home * Chess * Moves * **Encoding Moves**

Encoding Moves inside a chess program refers to both game records or game notation, and search related generation, make and unmake move to incremental update the board. During generation, moves are stored inside move lists, and best moves or refutation moves failing high inside the search are stored inside the

transposition table, Killer slots, PV- or refutation table, and moves, or certain aspects of a move, such as origin square and the target square are used to index butterfly boards for history- or countermove heuristic.

Move encoding in game records and game databases is usually designed to minimize size, while in search efficiency in generating is the main concern, considering board representation and other data structures like attack and defend maps. In general, move encoding either comprehend full information, that is contains involved pieces and square coordinates, or more common, omits the redundant piece information and relies on a adequate board representation to lookup the pieces by square. In Alpha-Beta like search, an important consideration is lazy move generation, to delay acquisition of certain information until it is really needed, which might be never in case of Cut-nodes.

Information Required

From-To Based

The information required to uniquely describe a move is the initial square, also called from-, origin- or departure square, and the target square, also called to- or destination square, and in case of promotions the promoted piece code. While this from-to information is also sufficient for castling in standard chess, due to the otherwise impossible double king step, it might not in Chess960. Therefore and also for efficiency reasons, castles are tagged as "special" moves. Such a move encoding conveniently fits inside a 16-bit word , 6 bits for from-to square each to index a butterfly board, still leaves a nibble for flags for move kind and promoted piece code, for instance this arbitrary flags:

code	promotion	capture	special 1	special 0	kind of move
0	0	0	0	0	quiet moves
1	0	0	0	1	double pawn push
2	0	0	1	0	king castle
3	0	0	1	1	queen castle
4	0	1	0	0	captures
5	0	1	0	1	ep-capture
8	1	0	0	0	knight-promotion
9	1	0	0	1	bishop-promotion

10	1	0	1	0	rook-promotion
11	1	0	1	1	queen-promotion
12	1	1	0	0	knight-promo capture
13	1	1	0	1	bishop-promo capture
14	1	1	1	0	rook-promo capture
15	1	1	1	1	queen-promo capture

Extended Move Structure

The information which piece performs the move and which piece is captured (if any) is implicit given by the from-to squares, with the requirement to lookup the current board before making the move, but in case of captures not after making or before unmaking the move. Some programs use a 32-bit double word as extended move structure at generation time as well for making/unmaking moves, with the upper bits used for a move score scaled by various move ordering techniques for instance dedicated piece-square tables and/or history heuristic, and perhaps two three bit codes for the moving and captured piece (if any) which also implies a kind of MVV-LVA coding. Also the extended move may apply composite indices for incremental update of Zobrist- or BCH keys.

Having the piece codes inside the move structure safes the board lookups during make, and makes storing captured pieces on a ply stack for unmake needless. Of course for space considerations to store moves inside transposition table, Killer slots, PV- or refutation table, the compact 16-bit move structure is still adequate for coordinate and move kind comparison with the lower 16 bits of the extended move structure.

C++ Sample

Rather than using bitfield move structures or classes in C and C++, most programmers rely on scalar integers, such as *short* and *int* for 16- or 32-bit words, but implement the composition and extraction while writing and reading various structure elements by explicit shifts and masks, likely encapsulated thought an interface with most likely inlined functions (or macros) to hide the internal representation. Further extended move structures might either embed or inherit this most compact base structure or class, which might already rely the native 32-bit integer type to avoid x86 16-bit optimization issues.

```
class CMove {
   CMove(unsigned int from, unsigned int to, unsigned int flags) {
      m_Move = ((flags & 0xf)<<12) | ((from & 0x3f)<<6) | (to & 0x3f);
   }
   void operator=(CMove a} {m_Move = a.m_Move;}

   unsigned int getTo() const {return m_Move & 0x3f;}
   unsigned int getFrom() const {return (m_Move >> 6) & 0x3f;}
   unsigned int getFlags() const {return (m_Move >> 12) & 0x0f;}

   void setTo(unsigned int to) {m_Move &= ~0x3f; m_Move |= to & 0x3f;}
   void setFrom(unsigned int from) {m_Move &= ~0xfc0; m_Move |= (
from & 0x3f) << 6;}
   ...

   bool isCapture() const {return (m_Move & CAPTURE_FLAG) != 0;}
   ...

   unsigned int getButterflyIndex() const {return m_Move & 0x0fff;}
   bool operator==(CMove a) const {return (m_Move & 0xffff) == (a.
m_Move & 0xffff);}
   bool operator!=(CMove a) const {return (m_Move & 0xffff) != (a.
m_Move & 0xffff);}

   unsigned short asShort() const {return (unsigned short) m_Move;}

protected:
   unsigned int m_Move; // or short or template type
};
```

Various Encodings and Decorations

Algebraic Notation

Based on Philipp Stamma's short Algebraic chess notation, a move can be described by the moving piece code and destination square. In case of disambiguating moves if two (or more) identical pieces can move to the same square, the file of departure, or if files are identical as well, the rank or both file and rank are given. A capture move, denoted by the symbol 'x' (takes) does not explicitly specify the captured piece and requires a look to the board as well.

Chess programs usually use algebraic notations concerning the user interface and Portable Game Notation - for appropriate conversion they have to deal with disambiguating source squares, that is need to be aware of all other moves of this piece-kind to the destination square to determine whether the from square needs additional file and/or rank information despite the moving piece.

Direction-Target

Another move encoding alternative motivated by direction wise target square serialization in bitboards is not to use the from-square but target square and direction. While the information is non-ambiguous it needs some effort with ray-lookup and bitscan to determine the from-square.

Check Flag

It might be interesting to determine whether a move gives check in advance during generation time, to possibly score them higher for move ordering, i. e. to don't reduce or even extend them, and also to safe in check detection after make move. However, as mentioned, lazy move generation is required, to delay acquisition of information until it is really needed, which might be never in case of Cut-nodes.
Additionally an early determined checking move, even if failing high, usually implies a huge sub-tree due to check extensions, while an early non checking moves likely makes a cheaper cutoff for most futile Cut-nodes. Therefor, determining checking moves in advance is not that recommended for most board-representations.

Move Index

If the generated moves inside a move list are deterministic, one may encode moves as relative list index. Since the maximum number of moves per positions seems 218 [1] [2] , one byte is enough to index the move. This encoding was used in early game databases.
Two Positions with 218 Moves each — A. Dickins (1968) [3]

R6R/3Q4/1Q4Q1/4Q3/2Q4Q/Q4Q2/pp1Q4/kBNN1KB1 w - - 0 1

3Q4/1Q4Q1/4Q3/2Q4R/Q4Q2/3Q4/1Q4Rp/1K1BBNNk w - - 0 1

Move Enumeration

Based on Influence Quantity of Pieces one may enumerate all moves, to specify a unique determined move number with respect to moving piece, from- and to squares, captured piece (if any) and promoted piece inside a 16-bit range.

Pawn Moves

Move Kind	Ranks	Files	Directions	Target Combinations	Cardinality
Pushs	5	8	1	1	40
Promotions	1	8	1	4	32
Double Pushs	1	8	1	1	8
Total Pushs	6	8	1	1+2/3	**80**
Captures	5	7	2	5	350
Captures & Promotions	1	7	2	4*4 [4]	224
En passant	1	7	2	1	14
Total Captures	6	7	2		**588**
Total Pawns	6		3		**668**

Piece Moves

Reversible and Captures, Influence Quantity of Pieces times six target combinations for empty and five possible capture targets.

Piece	Influence Quantity	Cardinality
Knight	336	2016
King	420	2520
Bishop	560	3360
Rook	896	5376
Queen	1456	8736
Total	3668	**22008**

All Moves

Sheet of all moves, considering Castling (but no null move):

Piece	Move Kind	From-To	None	Captures	per Side	Total
Pawn	Promotions	8	32	-	32	64
	Captures & Promotions	14	-	224	224	448
	Double Pushs	8	8	-	8	16
	Pushs	40	40	-	40	80
	Captures	70	-	350	350	700
	En passant	14	-	14	14	28
	Total		**80**	**588**	**668**	**1336**
King	Castles	2	2	-	2	4
		420	420	2100	2520	5040
	Total		**422**	**2100**	**2522**	**5044**
Knight		336	336	1680	2016	4032
Bishop		560	560	2800	3360	6720
Rook		896	896	4480	5376	10752
Queen		1456	1456	7280	8736	17472
	Total		**3248**	**16240**	**19488**	**38976**
Total			**3750**	**18928**	**22678**	**45356**

See also

- Influence Quantity of Pieces
- Move Generation
- Move Ordering

Forum Posts

- Subject: Maximum Number of Legal Moves by Andrew Shapira, CCC, May 08, 2005
- max amount of moves from a position? by Srdja Matovic, CCC, June 10, 2011 » Chess Maxima
- Contest: Find Position with the most moves by Charles Roberson, CCC, December 09, 2011
- Move encoding by Russell Reagan, CCC, April 21, 2014
- Killer and move encoding by Fabio Gobbato, CCC, August 14, 2014 » Killer Heuristic

External Links

- Move Representation - Computer Architecture and Languages Laboratory, University of Maribor

References

1. ^ Subject: Maximum Number of Legal Moves by Andrew Shapira, CCC, May 08, 2005
2. ^ max amount of moves from a position? by Srdja Matovic, CCC, June 10, 2011
3. ^ Re: max amount of moves from a position? by Árpád Rusz, CCC, June 11, 2011
4. ^ PxP not possible during promotion, therefore only four capture targets

What links here?

Page	Date Edited
Algebraic Chess Notation	Sep 25, 2017
Andrew Shapira	May 7, 2017
C	Feb 19, 2018
Chess	Jan 21, 2018
Chess Position	Sep 10, 2017
Copy-Make	May 23, 2017
Data	Nov 26, 2017
Encoding Moves	Mar 27, 2016
Fabio Gobbato	Mar 6, 2015
Game Notation	Jan 11, 2018
Incremental Updates	Sep 6, 2017
Influence Quantity of Pieces	Jul 21, 2017
Killer Heuristic	Sep 14, 2017
Killer Move	Feb 16, 2017
Make Move	Mar 2, 2016
MicroChess	Jul 14, 2015
Mini Chess	Jun 7, 2013
Move Generation	Jan 29, 2018
Move List	Jul 19, 2017
Moves	Feb 19, 2018
Napoleon	Jun 25, 2017
Nemorino	Dec 21, 2017

Page	Date Edited
Origin Square	Oct 10, 2016
Othello	Jan 4, 2018
Pawn Pushes (Bitboards)	Apr 5, 2013
Pieces versus Directions	Oct 6, 2016
Promotions	Jun 6, 2017
Scid	Apr 15, 2017
Sliding Pieces	Aug 3, 2017
Target Square	Oct 26, 2017
Unmake Move	Jun 10, 2017
Vice	Mar 8, 2016
Warren D. Smith	Sep 12, 2015

Up one Level