hogeschool Windesheim

Winchess

Technical Design Document

AUTHORS:
Niels te Bogt
Gertjan Brouwer
Nick Chen
Henrico Pops

DATE: 17/04/2020

Version	Date	Description	Author
0.1	**23/04**	**Board representation**	**Henrico Pops & Nick Chen**

Contents

1. Introduction

//@TODO Write introduction

Board representation

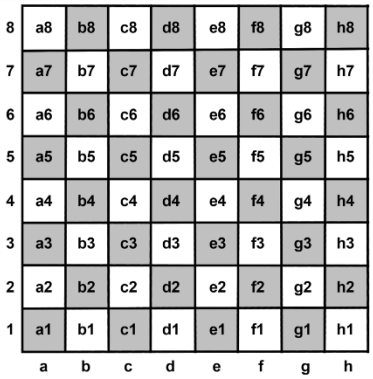
For a chess engine to work, it must know all the positions of the pieces on the board. A technical method to store this information is called a board representation. To find a method that will fit the chess engine the best, different methods have been compared to each other. Based on their pros and cons in speed, memory usage and complexity, a board representation that been selected to be used by the chess engine. These are the following analyses board representations:

Piece-Lists

Piece-lists are lists or arrays containing all the pieces on the board. This could be implemented in different ways. A 2d array could be used make the piece type (Rows) array containing the different locations (Columns) they are on. The array size will be 12 by 10, because there are 6 different pieces for both sides, and theoretically you can have a maximum of 10 pieces of a type[1]. In total it uses 960 bits for memory storage. Another method is to make one array filled with all the pieces. You could also use row-major ordering[2] to store the 2d array into a single array.

-	0	1	2	.. 10
0 - White pawns	A2	A3	A4	…
1 - White queens	D1	..	..	..
2 – Black knights	B8	G8	..	..
3 – Black rook	A8	H8	..	..
.. 12	..	..	..	..

Figure 1 A example of piece-lists with a 2d array



https://www.chessprogramming.org/Piece-Lists

[1] For example, if all the 8 pawns gets promoted to a rook, the maximum amount of rooks is reached: 10.
[2] https://en.wikipedia.org/wiki/Row-_and_column-major_order

Bitboard

Bitboard is a popular method compared to the other board representations. A bitboard has up to 64 bits, the same number of chess squares. To implement this method 12 bitboards will be needed for each piece and colour which results in 768 bits used. As CPU's are very efficient with bitwise operations, this method is one of the fastest methods generate moves. Because this method is used by many people, there is a lot of information online with can help with the implementation.

The bitboard of the white pawns in figure 2 would be:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
1	0	1	1	0	0	1	1
0	0	0	0	0	0	0	0

Table 1 Bitboard the chessboard in figure 2

https://www.chessprogramming.org/Bitboards

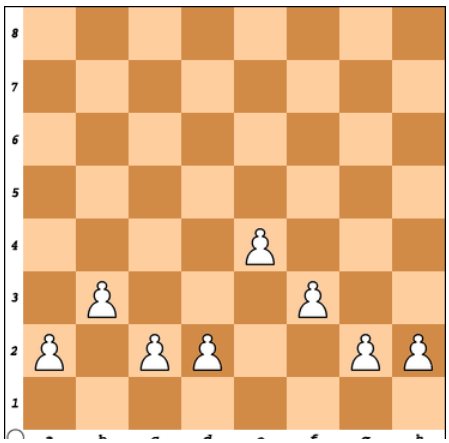


Figure 2 Chess board with white pawns

8x8 Board

The 8x8 Board is either a two-dimensional array ranged from 0 to 8 in both directions of a one-dimensional array ranged from 0 to 63. It uses 4 bits to represent the piece types which results in a total of 512 bits used. These kind of board representations are often used redundantly in bitboard programs to see in which piece is in a specific square.

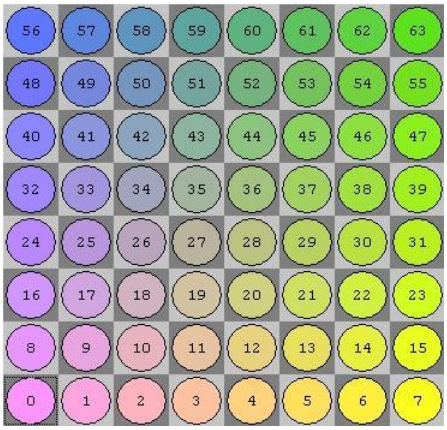


Figure 3 Example of the index of a one-dimensional array

With the 8x8 Board it is also simple to change the position of a piece. For example, if there was a pawn on index 11 and it wanted to move 1 square forward. The only thing that needs to be done is to add the existing index with 8.

https://www.chessprogramming.org/8x8_Board

10x12 Board

The 10x12 Board includes the 8x8 board array, surrounded by files and ranks (columns and rows) to prevent having to check if an index is in bound for move generation. The two ranks at the bottom and top are necessary to ensure that even knight jumps from the corners, result in valid array indices, greater or equal zero and less than 120. For example, to make a move for the knight at index 91, you could add 8 to the current index. Because this result ends up being 99 the move is marked as invalid. This method uses 960 bits for storage.

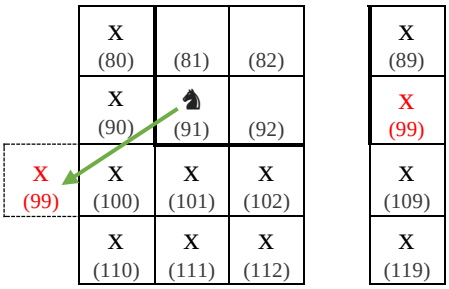


Figure 4 Knights move

https://www.chessprogramming.org/10x12_Board

0x88

0x88 is a square centric board representation. The coordinates of the board uses 8 bits in total, 4 bits for the rank and 4 bits for the file, to index the piece- or empty square codes.

$65_{16} = \underbrace{0101}_{\text{File}}\ \underbrace{0011}_{\text{Rank}}$

	A	B	C	D	E	F	G	H								
8	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
7	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
6	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
5	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
4	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
3	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
2	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	A	B	C	D	E	F	G	H								

Figure 5 Board representation of 0x88

On figure 5 the board representation of 0x88 can be observed. Instead of an 8x8 board 0x88 uses the size of 2 boards with the right board being unused, which results in 1024 bits being used. Because it uses hexadecimals, it has a neat little trick to figure out if a move is out of bounds. If a coordinate, that is part of board, gets converted from hexadecimal to binary the numbers will look like 0xxx 0xxx, and if it gets compared with the number 0x88 the result is always 0. With all the other numbers the result will always be bigger than 0. 0 as result means the number is on the board. This can be seen in table 2.

Hexadecimal coordinate	**Binary**	**Board position**	**& 0x88 (**1000 1000**)**
65	0101 0011	f6	0 (0)
4C	0100 1100	-	8 (0000 1000)
8C	10001100	-	88 (1000 1000)

Table 2 Example of comparing hexadecimal locations with 0x88

https://www.chessprogramming.org/0x88

Conclusion

After analysing all the different board representations, it has been concluded that bitboard will be used. Compared to the others bitboards implementation is straightforward as there is a lot of information available online. It also uses less memory than some others and is fast as it uses a lot of bitwise operations. The chess engine also might need a square centric board representation. If needed an 8x8 board will be implemented.

Representation	**Memory space**	**Complexity** (Straightforward – Normal -Complex)
Piece-List	960 bits	Complex
Bitboard	768 bits	Normal
8x8 Board	512 bits	Straightforward
10x12 Board	960 bits	Normal
0x88	1024 bit	Complex