Maastricht University Leading in Learning!

09-10-2020

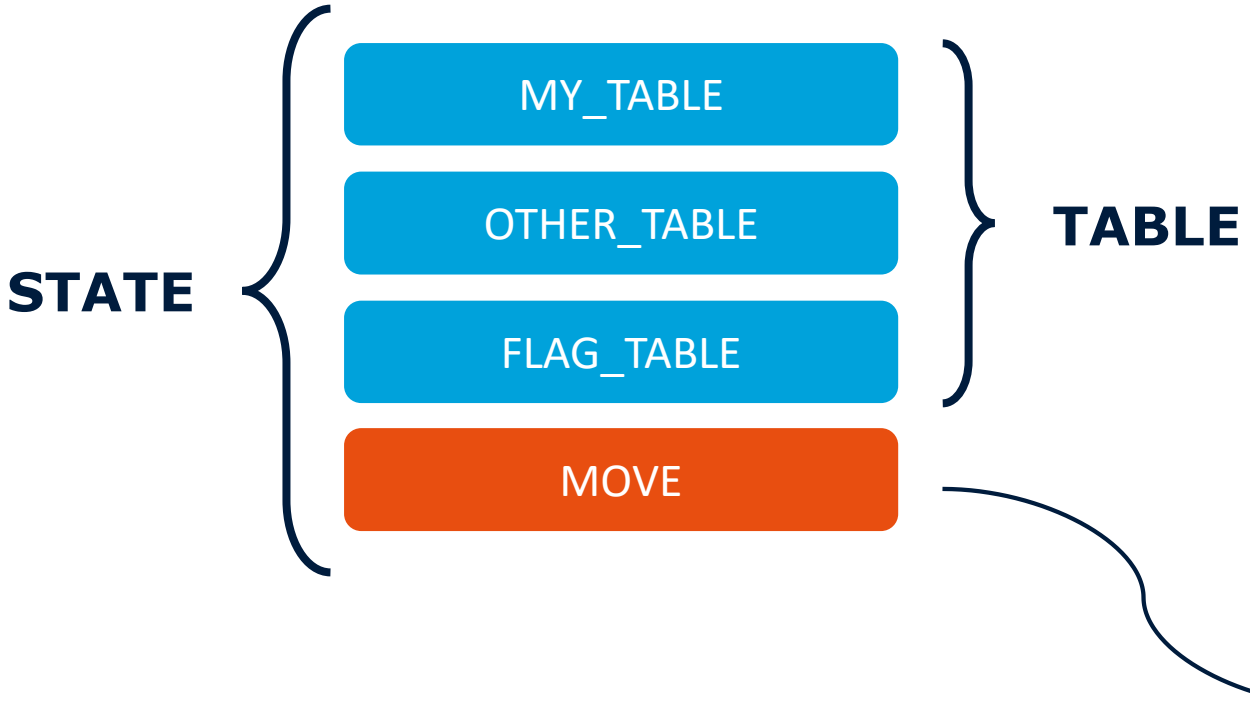
Breakthru

Game IA engine

Simone Grassi i6263794

BOARD RAPPRESENTATION

The structure that represent the current state of the game board



We need to extend the standard bitwise operation to operate between tables composed by two integers of 64 bit:
AND(), OR(), XOR(), COPY(), allZeros(), countSetBits()

```
MOVE {
    int sqr_from1; //index of the starting cell
    int sqr_to1; //index of the arriving cell
    int sqr_from2; //second possible move
    int sqr_to2;     //-1 if not present
    }
```

This is inserted in the state because this structure is used also to represent the children in the search tree so I need to know also the transition moves.

TABLE RAPPRESENTATION

How to represent a layer of the board as a BITBOARD

	A	B	C	D	E	F	G	H	I	J	K	
11	0	1	2	3	4	5	6	7	8	9	10	
10	11	12	13	14	15	16	17	18	19	20	21	
9	22	23	24	25	26	27	28	29	30	31	32	[0]
8	33	34	35	36	37	38	39	40	41	42	43	
7	44	45	46	47	48	49	50	51	52	53	54	
6	55	56	57	58	59	60	61	62	63	64	65	
5	66	68	70	71	72	73	74	75	76	77	78	
4	79	80	81	82	83	84	85	86	87	88	89	[1]
3	90	91	92	93	94	95	96	97	98	99	100	
2	101	102	103	104	105	106	107	108	109	110	111	
1	112	113	114	115	116	117	118	119	120	121	122	123 124 125 126 127 128 129

One layer of the board is represented by a TABLE[uint64_t , uint64_t]

The representation with two 64 bit integers is not perfect, there is an overflow of 7 bit,

✗ 5.5% of the memory occupation is wasted

✗ No direct access

Memory occupation (64 bit CPU):

2*64+64 = 192bit with bitboard

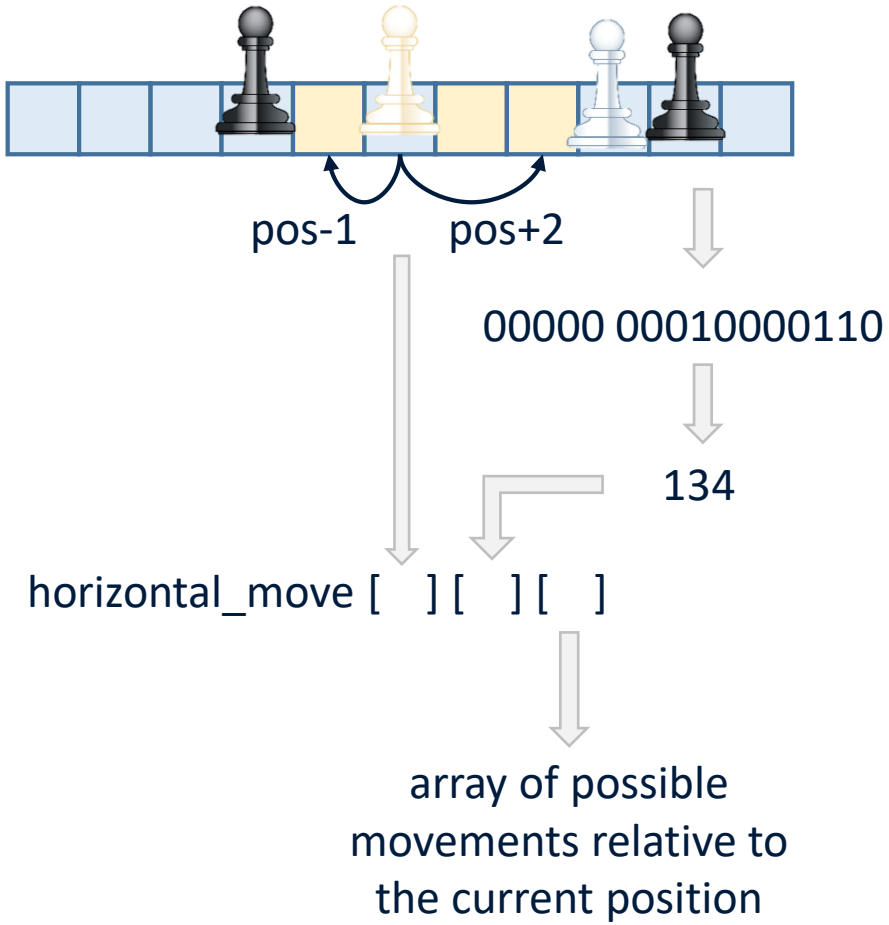
11*(64+11) = 825bit with an array of bool

✓ **-77% of memory occupation**

✓ Bitwise operations

MOVE GENERATION 1

How to obtain moves from occupancy and current position

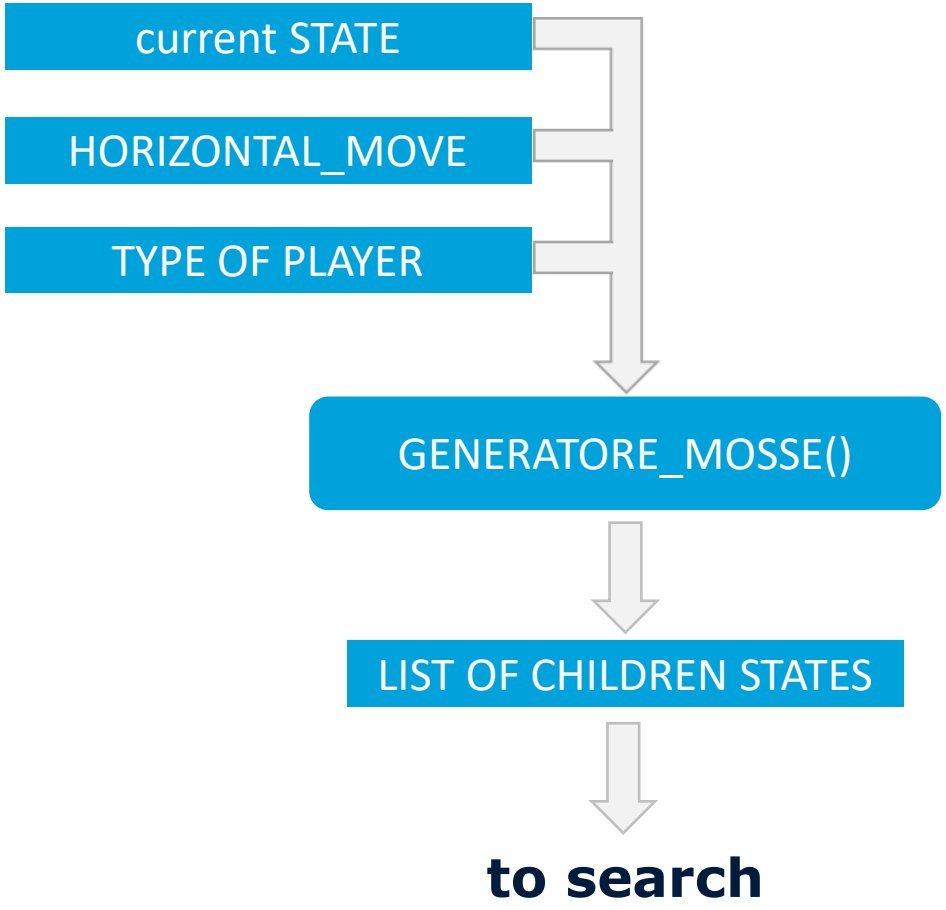


horizontal_move is a matrix, that given current position and the occupancy of all the pawns, returns an array with all the possible moves (rapresented as step from the current position).
It is generated in the first moment and used for the following searches, so is possible to obtain the moves without scan all the cells.

This mechanism is used by get_col_rank() to obtain the number representing the occupancy on the column, and get_row_rank() to obtain the occupation of the current row.

MOVE GENERATION 2

How children of a node in the search tree is generated



Switch my_board with other_board at every level

All sibilings generated at the same time

Possible moves: 2 pawns move, 1 capture, flag move or capture

RESULTS FROM TESTS:

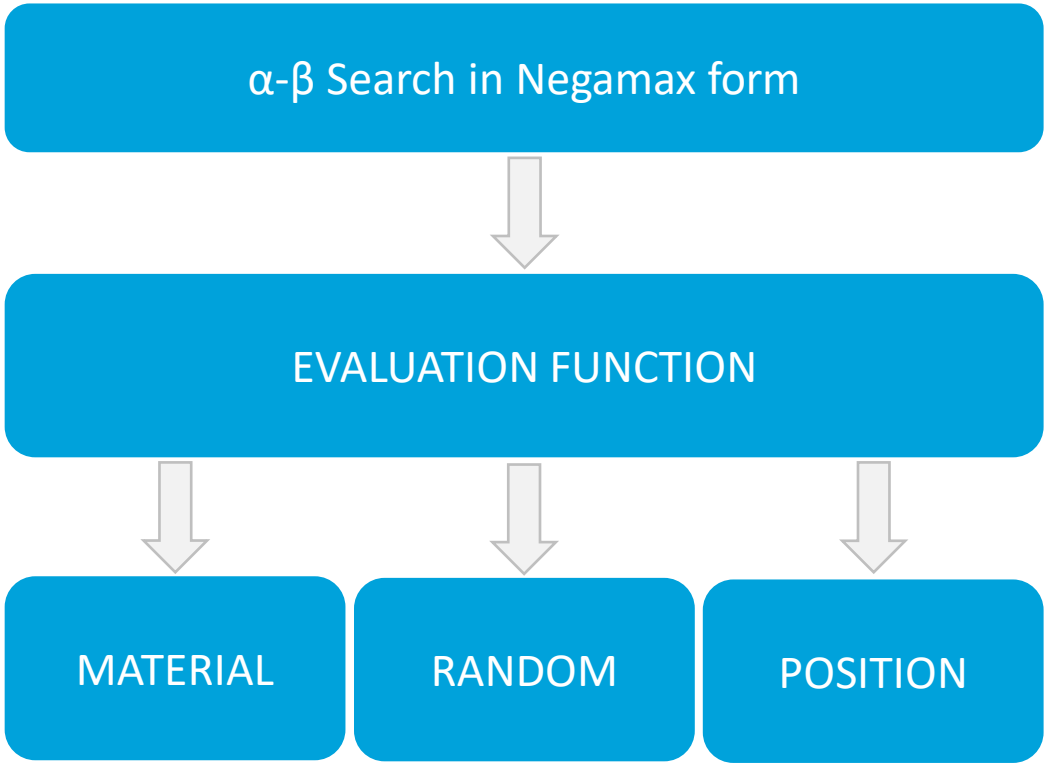
Average branching factor (no pruning)

Gold (flag): 3780

Silver: 5200

SEARCH ENGINE

How to search the next move and evaluate it



Breakthru is a game with a lot of one-death moves, so it's turned out that there are few usefull features to evaluate a position. According to the tests the most effective ones are:

- Materials: to try to reduce the number of opponent's pawns without loosing ours
- Position: to prefer moves that reduce the distance with the flag, this allows to make the game more aggressive and go closer to difensive/offensive positions
- Random: added during the tournament, only to prevent long and boring loops ''do-undo'', with the aim to wait for a good state to start an offensive, its weight is lower than the others.

ENHANCEMENTS

How to improve the search engine on my hardware

Iterative deepening

+

Time dependent depth

+

Transposition Table with Zobrist hashing 64 bit

This solution allows to reach the depth of 3 in a reasonable time using my laptop, but in different occasions this resulted to exceed the 10 minutes available for a match. To solve this hardware problem I've implemented a time-dependent strategy: until the 50% of the remaining time the depth is fixed to 3, from 50% to 20% it's reduced to 2 (with a reduction of time per move of 50%) and after the 20% the depth is reduced to 1. This was the most effective strategy because in the end the pawns are very closer and it's also possible to win, finishing the opponent's time.

The replacing strategy choosen is to mantain the deeper element, the TT table is implemented as a map to have an access O(1).

Other improvements, such as killer moves or searching windows show to be poorly effective or counterproductive.

RESULTS FROM TESTS: Average searching time (depth 2) = 23sec (depth 3) = 50sec

UI & GAME MENU’

The simple user interface on the shell

- Select gold\silver mode
- PRINT TABLE on shell
- Search my next move
- Insert opponent’s move
- Undo
- Remaining time display
- Win\Lose detector

```
Starting searching with depth 3
Move: h6xg6
Move: d6xe6
        a       b       c       d       e       f       g       h       i       j       k
11      .       .       .       .       .       .       .       .       .       .       .
10      .       .       .       X       X       X       X       X       .       .       .
9       .       .       .       .       .       .       .       .       .       .       .
8       .       X       .       .       O       O       O       .       .       X       .
7       .       X       .       O       .       .       .       O       .       X       .
6       .       X       .       .       O       @       O       .       .       X       .
5       .       X       .       O       .       .       .       O       .       X       .
4       .       X       .       .       O       O       O       .       .       X       .
3       .       .       .       .       .       .       .       .       .       .       .
2       .       .       .       X       X       X       X       X       .       .       .
1       .       .       .       .       .       .       .       .       .       .       .
Remaining time:95% 26.7576s
Game's menu:
        1. My turn
        2. Other turn
        3. Undo
        4. Exit
->
```

POSSIBLE IMPROVEMENTS

Possible future enhancements that need more tests

MonteCarlo Evaluation or NN based Evaluation: the evaluation function is weak because one-death moves are common

Bitwise operation with Magic Numbers instead O(n) operations on board

Python or C++ based Graphic UI

ENDGAME database or solver