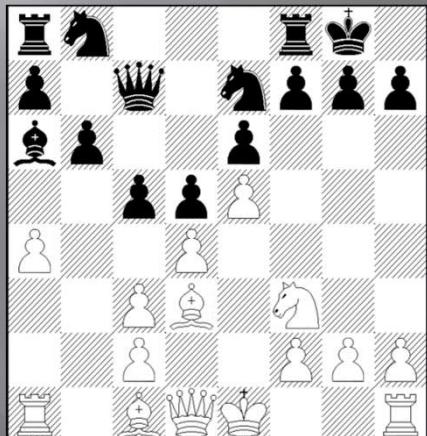


MOTEUR D'ÉCHECS ARCHITECTURE LOGICIELLE

Intelligence Artificielle



Pascal Tang - Janvier 2019

Moteurs d'échecs

- Moteurs d'échecs ?
- Minimax/alpha-beta pruning ?
- Evaluation functions ?
- Null-move pruning ?
- Bitboards ?
- LMR



1997

Deep Blue

- Defeated Kasparov
3.5-2.5
- Specialized hardware
- Typical depth of 6-8,
sometimes reached
depth 20

1997

Deep Blue

- Defeated Kasparov 3.5-2.5
- Specialized hardware
- Typical depth of 6-8, sometimes reached depth 20

2015

Stockfish

- Much stronger than any human grandmaster
- Software runs on any personal computer
- Reaches depths of 40+ in standard time controls

- Méthodes connexionnistes:
 - Alpha GO
 - Alpha Zero
- Méthodes connexionnistes en Echecs classiques:
 - LC0 en open source

Elo Ranking

- Humains:
 - Kasparov: 2851 Elo en 1999
 - Carlsen: 2872 Elo
- Machines:
 - StockFish: 3564 Elo CCRL
 - Houdini: 3529 Elo CCRL
 - Komodo: 3508 Elo CCRL
 - Leela Chess Zero: 3463 CCRL

Elo Ranking

P (A/C)	E(p)								
99 %	677	79 %	230	59 %	65	39 %	-80	19 %	-251
98 %	589	78 %	220	58 %	57	38 %	-87	18 %	-262
97 %	538	77 %	211	57 %	50	37 %	-95	17 %	-273
96 %	501	76 %	202	56 %	43	36 %	-102	16 %	-284
95 %	470	75 %	193	55 %	36	35 %	-110	15 %	-296
94 %	444	74 %	184	54 %	29	34 %	-117	14 %	-309
93 %	422	73 %	175	53 %	21	33 %	-125	13 %	-322
92 %	401	72 %	166	52 %	14	32 %	-133	12 %	-336
91 %	383	71 %	158	51 %	7	31 %	-141	11 %	-351
90 %	366	70 %	149	50 %	0	30 %	-149	10 %	-366
89 %	351	69 %	141	49 %	-7	29 %	-158	9 %	-383
88 %	336	68 %	133	48 %	-14	28 %	-166	8 %	-401
87 %	322	67 %	125	47 %	-21	27 %	-175	7 %	-422
86 %	309	66 %	117	46 %	-29	26 %	-184	6 %	-444
85 %	296	65 %	110	45 %	-36	25 %	-193	5 %	-470
84 %	284	64 %	102	44 %	-43	24 %	-202	4 %	-501
83 %	273	63 %	95	43 %	-50	23 %	-211	3 %	-538
82 %	262	62 %	87	42 %	-57	22 %	-220	2 %	-589
81 %	251	61 %	80	41 %	-65	21 %	-230	1 %	-677
80 %	240	60 %	72	40 %	-72	20 %	-240		

Déroulement des TDs

- Langage C/C++
- Travail en binôme ou trinôme
- Evaluation hebdomadaire
 - Production de code
 - Efficacité du travail en équipe
 - Autonomie
 - Mémoire de 10 pages

Planning

- S1-S2
 - Board Representation/Move generation
 - Search algorithms
 - TSCP
 - Winboard interfacing
- S3
 - Iterative deepening
 - Transposition hash
 - Time Control
 - Optimisation: gagner en vitesse
- S4-S5
 - Optimisation: gagner en profondeur
 - Move ordering
 - Search pruning/reductions
 - Opening book
 - Pondering
 - Learning
 - Tournaments

Les algorithmes de recherche

- Minimax
- Negamax
- Alpha-beta pruning
- PVS (Principal Variation Search)

Minimax

Nous allons expliquer ici le principe minimax : commençons par choisir une fonction heuristique f définie sur l'ensemble des positions légales \mathcal{E} et à valeurs dans un ensemble ordonné \mathcal{O} . Cette fonction heuristique appelée ici *Fonction d'évaluation* est définie de manière à induire un ordre sur l'ensemble des position légales : $f(P)$ avec $P \in \mathcal{E}$, croît avec les possibilités de gain de *MAX* depuis la position P . De plus cette fonction est supposée *coïncidente*, c'est à dire, si P est une position terminale alors

$$f(P) = \begin{cases} -1_{\mathcal{O}} & \text{si } MAX \text{ a perdu} \\ 0_{\mathcal{O}} & \text{si la position } P \text{ est nulle} \\ +1_{\mathcal{O}} & \text{si } MAX \text{ a gagné} \end{cases}$$

La fonction *minimax* est alors définie par :

$$\text{minimax}(P) = \begin{cases} f(P) & \text{si } P \text{ est terminale} \\ \max_i(\text{minimax}(P_i)) & \text{si } MAX \text{ a le trait depuis } P \\ \min_i(\text{minimax}(P_i)) & \text{si } MIN \text{ a le trait depuis } P \end{cases}$$

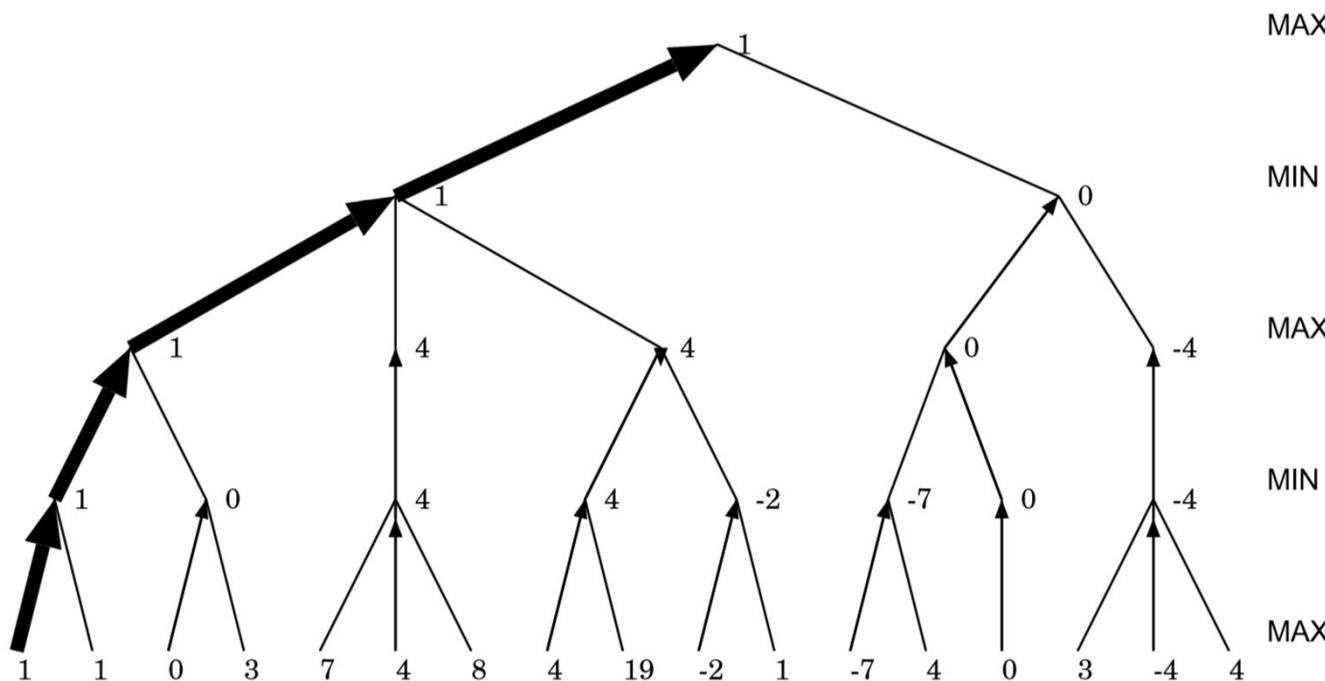
Minimax

En pratique, la valeur théorique de la position P ne pourra généralement pas être calculée : cela signifierait qu'on peut déterminer la valeur théorique de la position ; en conséquence, la fonction d'évaluation sera appliquée dans des positions non terminales. On considérera que plus la fonction d'évaluation est appliquée loin de la racine, meilleur est le résultat du calcul . C'est à dire qu'en examinant plus de coups successifs, nous supposons obtenir une meilleure approximation de la valeur théorique donc un meilleur choix de mouvement.

Minimax

```
function minimax(node, maximizingPlayer)
    if node is a terminal node
        return the value of node
    if maximizingPlayer
        bestValue := -∞
        for each child of node
            val := minimax(child, FALSE)
            bestValue := max(bestValue, val)
        return bestValue
    else
        bestValue := +∞
        for each child of node
            val := minimax(child, TRUE)
            bestValue := min(bestValue, val)
        return bestValue
(* Initial call for maximizing player *)
minimax(origin, TRUE)
```

Minimax

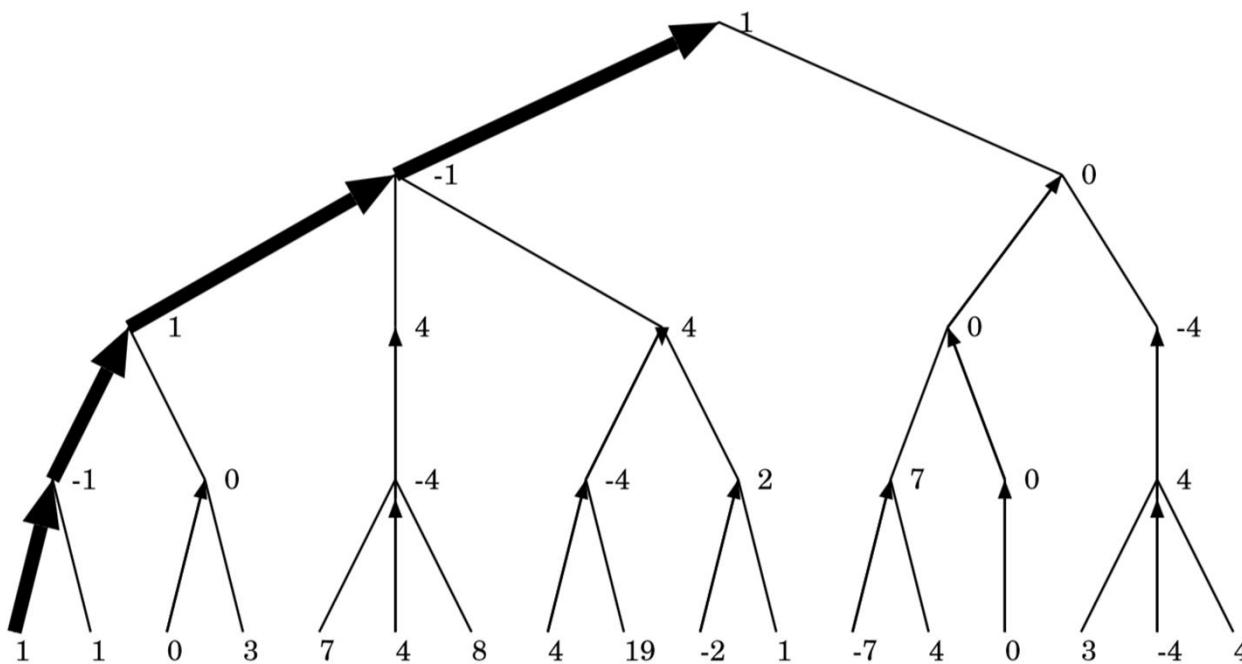


Application du Minimax sur un arbre de jeu d'échecs. Dans un nœud *MAX* (resp. *MIN*), le maximum (resp. le minimum) de la valeur minimax des nœuds fils est renvoyée.

Negamax

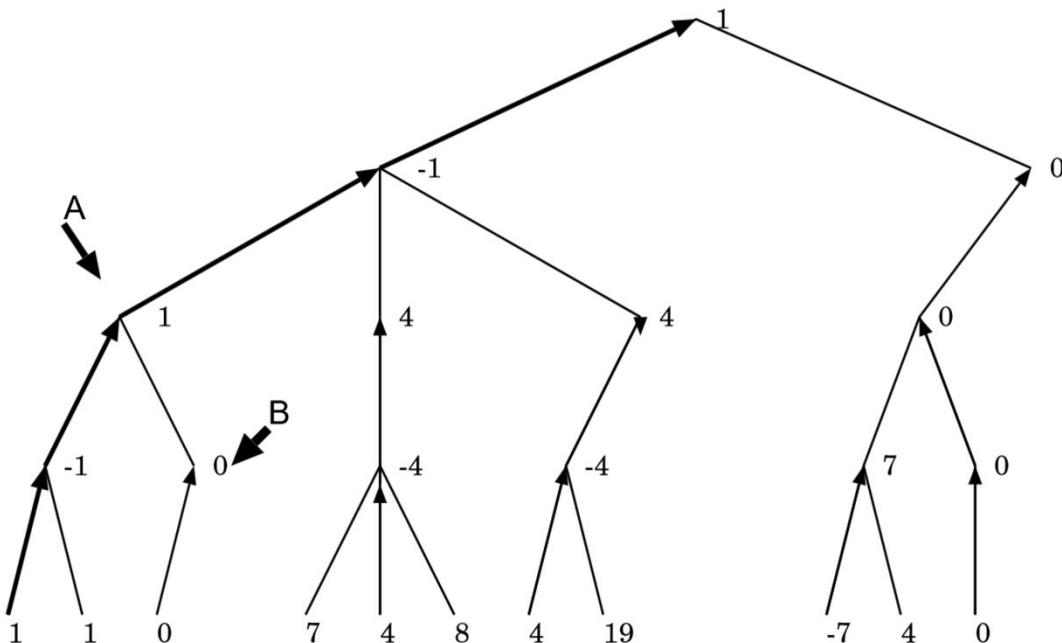
```
function negamax(node, depth, color)
    if node is a terminal node
        return color * the heuristic value of node
    bestValue := -∞
    for each child of node
        val := -negamax(child, depth - 1, -color)
        bestValue := max( bestValue, val )
    return bestValue
```

Negamax



Application du *Negamax* sur le même arbre. Nous retournons le maximum des négations des valeurs negamax des noeuds fils.

Alpha-beta



Après l'évaluation du premier fils du nœud **B**, on sait que la valeur negamax du nœud **B** sera supérieure à 0, l'algorithme $\alpha\beta$ ayant déjà évalué le premier fils du nœud **A**, sait que le nœud **A** choisira un fils dont la valeur negamax est inférieure à -1. Il est donc inutile d'analyser les autres fils du nœud **B** car le meilleur mouvement au nœud **A** ne peut être celui qui mène au nœud **B**.

Alpha-beta

```
function negamax(node, depth, α, β, color)
    if depth = 0 or node is a terminal node
        return color * the heuristic value of node
    bestValue := -∞
    for each child of node
        val := -negamax(child, depth - 1, -β, -α, -color)
        bestValue := max( bestValue, val )
        α := max( α, val )
        if α ≥ β
            break
    return bestValue
```

Initial call for Player A's root node

```
rootNegamaxValue := negamax( rootNode, depth, -∞, +∞, 1)
```

PVS

```
funct PVS(Position,  $\alpha$ ,  $\beta$ )  $\equiv$ 
   $\lceil M \leftarrow \text{ProchainCoup}(\text{Position});$ 
  if  $M = \text{null}$  then exit g(Position); fi
   $\text{Meilleur} \leftarrow -\text{PVS}(\text{position} \bullet M, -\beta, -\alpha);$ 
   $M \leftarrow \text{ProchainCoup}(\text{Position});$ 
  while  $M \neq \text{null}$  do
    if  $\text{Meilleur} > \alpha$  then  $\alpha \leftarrow \text{Meilleur};$  fi
    if  $\alpha \geq \beta$  then exit  $\text{Meilleur};$  fi
     $\text{Valeur} \leftarrow -\text{PVS}(\text{Position} \bullet M, -\alpha - 1, -\alpha);$ 
    if  $\text{Valeur} > \alpha \wedge \text{Valeur} < \beta$ 
      then  $\text{Valeur} \leftarrow -\text{PVS}(\text{Position} \bullet M, -\beta, -\text{Valeur});$  fi
    if  $\text{Valeur} > \text{Meilleur}$  then  $\text{Meilleur} \leftarrow \text{Valeur};$  fi
     $M \leftarrow \text{ProchainCoup}(\text{Position});$  od
   $\text{Meilleur} \rfloor.$ 
```

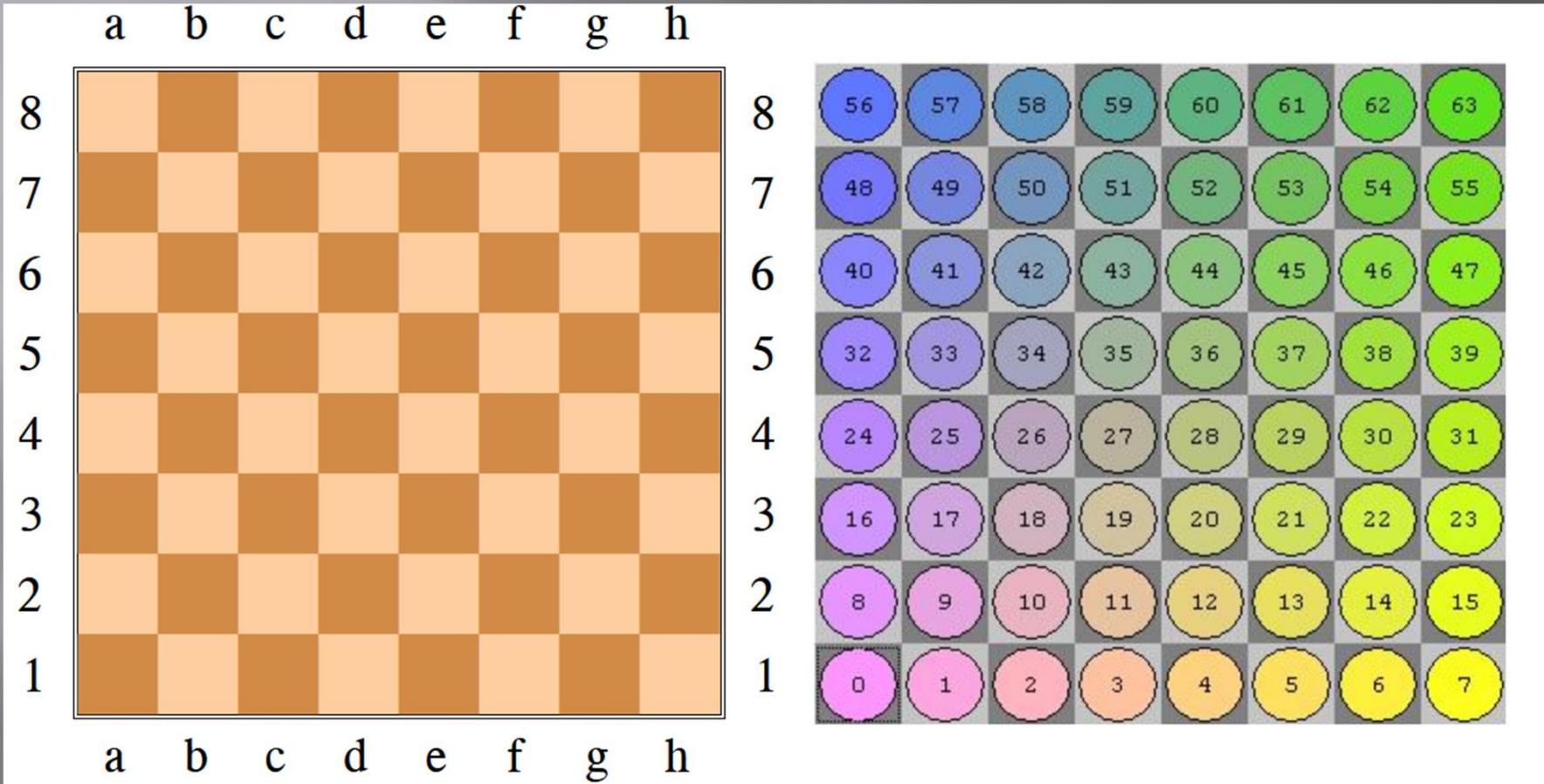
Alpha-beta properties

- Considered *backward pruning*; does not influence the root value
- Values for other nodes are an upper or lower bound
- Better move ordering = more pruning
- Time complexity with perfect ordering: $O(b^{m/2})$

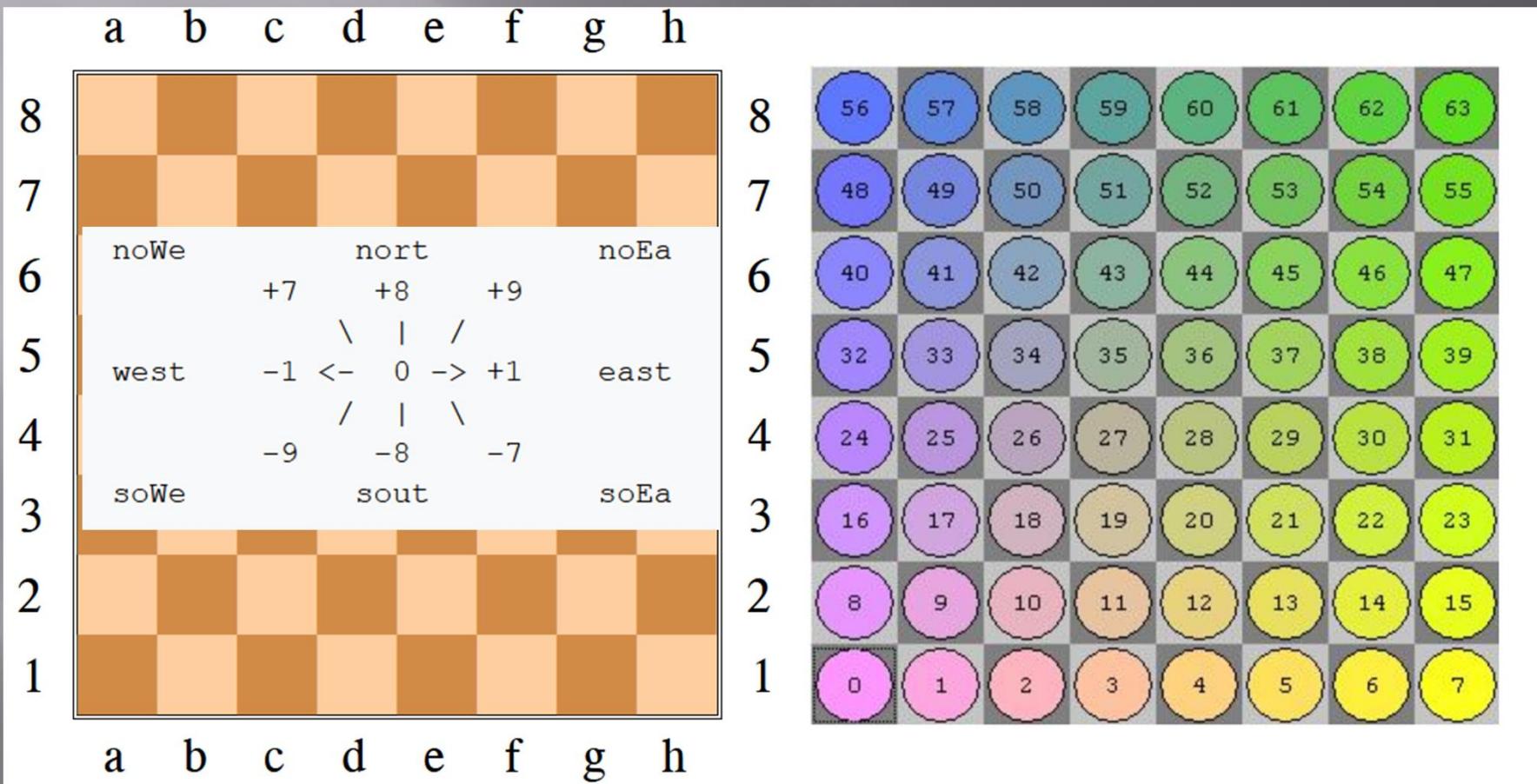
Board representation

```
[[['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
 ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R']]
```

Board Representation



Board Representation



Board representation

```
enum enumSquare
{ a1, b1, c1, d1, e1, f1, g1, h1,
  a2, b2, c2, d2, e2, f2, g2, h2,
  a3, b3, c3, d3, e3, f3, g3, h3,
  a4, b4, c4, d4, e4, f4, g4, h4,
  a5, b5, c5, d5, e5, f5, g5, h5,
  a6, b6, c6, d6, e6, f6, g6, h6,
  a7, b7, c7, d7, e7, f7, g7, h7,
  a8, b8, c8, d8, e8, f8, g8, h8 };
```

Mailbox

```
int mailbox[120] = {  
    -1, -1, -1, -1, -1, -1, -1, -1, -1,  
    -1, -1, -1, -1, -1, -1, -1, -1, -1,  
    -1,  0,  1,  2,  3,  4,  5,  6,  7, -1,  
    -1,  8,  9, 10, 11, 12, 13, 14, 15, -1,  
    -1, 16, 17, 18, 19, 20, 21, 22, 23, -1,  
    -1, 24, 25, 26, 27, 28, 29, 30, 31, -1,  
    -1, 32, 33, 34, 35, 36, 37, 38, 39, -1,  
    -1, 40, 41, 42, 43, 44, 45, 46, 47, -1,  
    -1, 48, 49, 50, 51, 52, 53, 54, 55, -1,  
    -1, 56, 57, 58, 59, 60, 61, 62, 63, -1,  
    -1, -1, -1, -1, -1, -1, -1, -1, -1,  
    -1, -1, -1, -1, -1, -1, -1, -1, -1  
};  
  
int mailbox64[64] = {  
    21, 22, 23, 24, 25, 26, 27, 28,  
    31, 32, 33, 34, 35, 36, 37, 38,  
    41, 42, 43, 44, 45, 46, 47, 48,  
    51, 52, 53, 54, 55, 56, 57, 58,  
    61, 62, 63, 64, 65, 66, 67, 68,  
    71, 72, 73, 74, 75, 76, 77, 78,  
    81, 82, 83, 84, 85, 86, 87, 88,  
    91, 92, 93, 94, 95, 96, 97, 98  
};
```

- Tour en a4 (case 32), peut-elle aller à gauche ? On soustrait 1 => 31 (h5).
- En utilisant la mailbox, on a 61. On soustrait de 1, on obtient 60. Comme mailbox[60]==-1, alors le coup est illégal.

Offset move generation

```
int side; /* the side to move */
int xside; /* the side not to move */

BOOL slide[6] = {FALSE, FALSE, TRUE, TRUE, TRUE, FALSE};
int offsets[6] = {0, 8, 4, 4, 8, 8}; /* knight or ray directions */
int offset[6][8] = {
    { 0, 0, 0, 0, 0, 0, 0, 0 },
    { -21, -19, -12, -8, 8, 12, 19, 21 }, /* KNIGHT */
    { -11, -9, 9, 11, 0, 0, 0, 0 }, /* BISHOP */
    { -10, -1, 1, 10, 0, 0, 0, 0 }, /* ROOK */
    { -11, -10, -9, -1, 1, 9, 10, 11 }, /* QUEEN */
    { -11, -10, -9, -1, 1, 9, 10, 11 } /* KING */
};

for (i = 0; i < 64; ++i) { /* loop over all squares (no piece list) */
    if (color[i] == side) { /* looking for own pieces and pawns to move */
        p = piece[i]; /* found one */
        if (p != PAWN) { /* piece or pawn */
            for (j = 0; j < offsets[p]; ++j) { /* for all knight or ray directions */
                for (n = i;;) { /* starting with from square */
                    n = mailbox[mailbox64[n] + offset[p][j]]; /* next square along the ray j */
                    if (n == -1) break; /* outside board */
                    if (color[n] != EMPTY) {
                        if (color[n] == xside)
                            genMove(i, n, 1); /* capture from i to n */
                        break;
                    }
                    genMove(i, n, 0); /* quiet move from i to n */
                    if (!slide[p]) break; /* next direction */
                }
            }
        } else { /* pawn moves */
            }
    }
}
```

Evaluation

- La plus simple: Evaluation matérielle
= somme des valeurs des pièces présentes sur l'échiquier
 - Roi: infini
 - Pion: 100
 - Reine: 8 pions
 - Tour: 5 pions
 - Fous: 3 pions
 - Cavalier: 3 pions

Interfaçage Winboard

- <http://hgm.nubati.net/>

Etude TSCP

- Etude code de TSCP
 - <http://www.tckerrigan.com/Chess/TSCP/>

Liens

- <https://www.echecs.club/regles/>
- https://www.chessprogramming.org/Main_Page