

# IA pour les Jeux

## Licence Informatique et Vidéoludisme

Nicolas Jouandeau

n@up8.edu

2024

## évaluation

- ▶ C/C++ Racket Ocaml Prolog Pike Java sh Python
- ▶ évaluation : TPs + PRJ
- ▶ TP individuel
- ▶ PRJ en groupe (2-4)
  - compétition Breakthrough
  - pendant les 2 ou 3 dernières semaines
  - un rendu (programme et rapport 1p) par groupe
  - <https://ludii.games/>
- ▶ ia orientée jeux de plateaux (échecs-like)
  - 2 joueurs en tour par tour : breakthrough
  - et des variantes
  - <https://www.chessprogramming.org>
- ▶ ia de jeux de stratégie temps-réel (Real Time Battle)
  - bataille de tank en temps réel, 1 tank par personne
  - <https://realtimebattle.sourceforge.net/>

## C/C++

- ▶ solution impérative avec `struct` et sans `class`
- ▶ avec conteneurs C++ (`list`, `vector`, `unordered_map`)
- ▶ <https://en.cppreference.com/w/>

## Racket et Ocaml

- ▶ solution fonctionnelle
- ▶ <https://racket-lang.org/>
- ▶ <https://ocaml.org/>

## Prolog

- ▶ solution descriptive
- ▶ <https://www.swi-prolog.org/>

## Pike

- ▶ syntaxe C interprétée garbage-collecté orienté type entier et string
- ▶ <https://pike.lysator.liu.se/>

## Java

- ▶ langage objet compilé pour JVM
- ▶ <https://www.java.com/fr/>

## sh

- ▶ Bourn shell, abrégé `bsh`, lui-même abrégé `sh`
- ▶ interpréteur de command shell des environnements UNIX
- ▶ <http://www.opengroup.org/unix/online.html>

## Python

- ▶ apprentissage automatique (machine learning)
- ▶ avec NumPy, Keras, PyTorch et TensorFlow
- ▶ <https://www.python.org/>
- ▶ <https://numpy.org/>
- ▶ <https://keras.io/>
- ▶ <https://pytorch.org/>
- ▶ <https://www.tensorflow.org/?hl=fr>

## Définition de l'IA

- ▶ compréhension et la construction d'entités intelligentes
- ▶ intelligence : processus de la pensée dont l'objectif est de percevoir, comprendre, prévoir, manipuler un monde plus étendu que soi-même
- ▶ objectif de l'IA : systématiser et automatiser les tâches intellectuelles

## Deux définitions

- ▶ Penser et agir rationnellement (i.e. conformément à ses connaissances et à une fonction d'évaluation)
- ▶ Penser et agir comme un humain (i.e. conformément au test de Turing)

## Domaines concernés

- ▶ les jeux
- ▶ et également la planification, la programmation d'agents et de systèmes autonomes, le diagnostic, la robotique, la compréhension des langages

## Sociétés savantes, conférences et revues

- ▶ Association for the Advancement of Artificial Intelligence (AAAI), Special Interest Group on Artificial Intelligence (ACM-SIGAI), Association for Computational Linguistics (ACL), International Computer Games Association (ICGA)
- ▶ IJCAI, ECAI, ICML, NeurIPS, IEEE-TAAI, IEEE-CoG, TCIAIG, ACG, CG

## fonction heuristique d'évaluation : les échecs

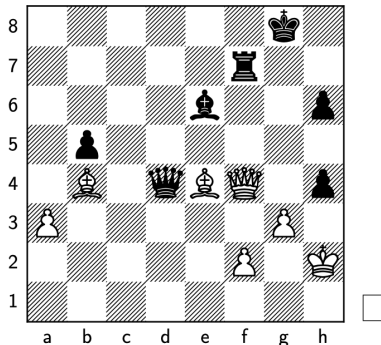
- ▶ King, Queen, Rook, Bishop, kNight et Pawn
- ▶ blanc en majuscule et noir en miniscule
- ▶  $D$  (respect.  $d$ ) les pions doublés (2 pions sur une colonne)
- ▶  $S$  (respect.  $s$ ) les pions bloqués (pièce devant)
- ▶  $I$  (respect.  $i$ ) les pions isolés (pas de pions dans les colonnes adj.)
- ▶  $M$  (respect.  $m$ ) la mobilité
- ▶  $f$  fonction d'évaluation pour le joueur blanc
- ▶ 
$$f = 200(N_K - N_k) + 9(N_Q - N_q) + 5(N_R - N_r) + 3(N_B + N_N - N_b - N_n) + N_P - N_p - (D - d + S - s + I - i)/2 + (M - m)/10$$



## fonction heuristique d'évaluation : les échecs

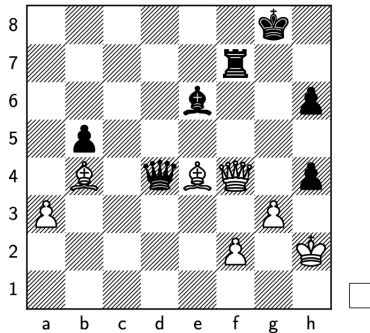
► 
$$f = 200(N_K - N_k) + 9(N_Q - N_q) + 5(N_R - N_r) + 3(N_B + N_N - N_b - N_n) + N_P - N_p - (D - d + S - s + I - i)/2 + (M - m)/10$$

## exemple de position



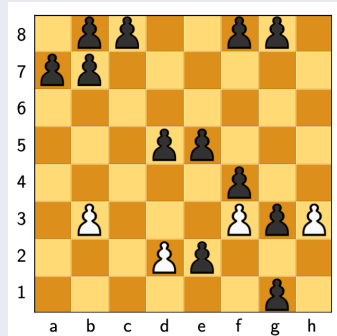
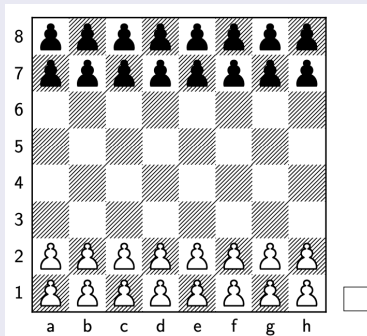
## évaluation d'une position

$$\begin{aligned}
 \blacktriangleright f &= 200(N_K - N_k) + 9(N_Q - N_q) + 5(N_R - N_r) + 3(N_B + N_N - N_b - N_n) + N_P - N_p - (D - d + S - s + I - i)/2 + (M - m)/10 \\
 &= 5(0 - 1) + 3(2 - 1) - (0 - 2 + 0 - 1)/2 + (43 - 39)/10 \\
 &= -0.1
 \end{aligned}$$



## Breakthrough (usuellement 8x8)

- ▶ jouer uniquement avec des pions (2 lignes par coté)<sup>a</sup>
- ▶ déplacement devant et en diag de 1 case
- ▶ capture uniquement en diag de 1 case
- ▶ victoire à la première promotion

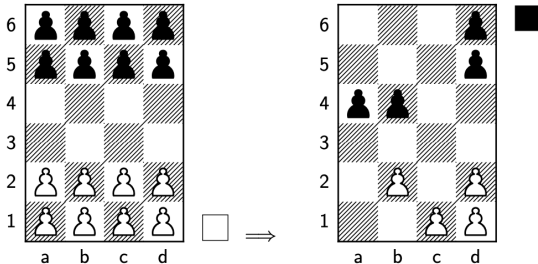


## Breakthrough : les variantes

- ▶ misère : le joueur qui fait une promotion perd
- ▶ à captures forcées : les captures sont obligatoires si elles sont possibles
- ▶ à coups simultanés : chaque joueur annonce son coup et en cas de collision, les coups s'annulent et personne ne bouge, un arbitre annonce les coups nuls
- ▶ à information imparfaite : chaque joueur ne voit que ses pions, parfois un pion peut disparaître (ce qui signale une capture adverse), il ne sait pas quand il capture en diagonal, un déplacement devant un pion est annulé si la case n'est pas vide, un arbitre annonce les coups nuls

## Breakthrough (2)

- ▶ board de 8x8 :  $3^{64}$  positions  $\approx 10^{30}$
- ▶ board de 6x4 :  $3^{24}$  positions  $\approx 10^{11}$
- ▶ exemple de séquence : 2a3a-5a4a 2b3b-4a3b  
2c3b-6a5a 3a4a-5b4a 3b4a-6b5b 4a5b-6c5b  
1a2a-5a4a 2a3a-5b4b 3a4b-5c4b 1b2b-????
- ▶ quel est le meilleur coup pour noir ?



## A\* : recherche de plus court chemin

- ▶ un état  $s$  = une position = un ensemble de pions sur un plateau
- ▶  $p_i$  la position initiale
- ▶  $p_f$  la position finale
- ▶  $n$  itérations
- ▶  $\text{free}(s)$  retourne vrai si  $s$  n'est pas en collision
- ▶  $\text{nextMoves}(s)$  retourne la liste des mouvements en  $s$
- ▶  $\text{applyMove}(s, m)$  applique  $m$  sur  $s$
- ▶  $\text{dist}(a, b, c)$  estime la distance de  $a$  à  $c$  passant par  $b$
- ▶  $\mathcal{H}$  la table de hashage des positions déjà évaluées  
 $\mathcal{H}[s'] = s$  ssi  $s' \leftarrow \text{applyMove}(s, m)$
- ▶  $\mathcal{L}$  la liste des positions à évaluer

## A\* : recherche de plus court chemin

```
1 fonction astar (  $p_i$ ,  $p_f$ ,  $n$  ) :  
2    $\mathcal{L} \leftarrow \{ p_i \}$  ;  
3    $\mathcal{H} \leftarrow \emptyset$  ;  
4   for  $n$  times do  
5     if  $|\mathcal{L}| == 0$  then break ;  
6      $s \leftarrow \mathcal{L}.\text{pop\_front}()$  ;  
7     if  $s == p_f$  then  
8       return mkSolution (  $\mathcal{H}$ ,  $s$ ,  $p_i$  ) ;  
9      $\mathcal{M} \leftarrow \text{nextMoves}(s)$  ;  
10    for each  $m \in \mathcal{M}$  do  
11       $s' \leftarrow \text{applyMove}(s, m)$  ;  
12      if free ( $s'$ ) then  
13        if  $s' \notin \mathcal{H}$  then  
14           $\mathcal{H}[s'] \leftarrow s$  ;  
15           $\mathcal{L} \leftarrow \mathcal{L} + s'$  ;           // by distance order  
16        else  
17          if  $\text{dist}(p_i, \mathcal{H}[s'], p_f) + 1 > \text{dist}(p_i, s', p_f)$  then  
18             $\mathcal{H}[s'] \leftarrow s$  ;  
19             $\mathcal{L} \leftarrow \mathcal{L} + s'$  ;           // by distance order  
20  return  $\emptyset$  ;
```

## A\* : retourner le chemin trouvé

```
1 fonction mkSolution (  $\mathcal{H}$ ,  $s$ ,  $e$  ) :  
2    $S \leftarrow \emptyset$  ;  
3   while not-interrupted do  
4      $S.\text{push\_front}(s)$  ;  
5     if  $s == e$  then break ;  
6      $s \leftarrow \mathcal{H}[s]$  ;  
7   return  $S$  . ;
```

## exemple de problème

- ▶ grille de 4x4 avec obstacles notés #
- ▶ un personnage @ et un objectif \$

```
#$.  
@#.  
..#.  
....
```



## A\* : solution

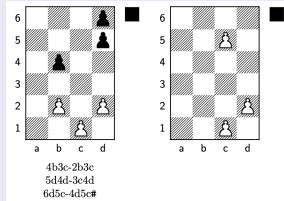
#\$. .	#\$. .	#\$. .	#\$. .	#\$. .	#\$. .	#\$. .	#\$. .
@#. .	.#. .	.#. .	.#. .	.#. .	.#. .	.#. .	.#. @
..#. .	@. #.	.@#. .	..#. .	..#. .	..#. .	..#@	..#. .
....	....	....	.@..	..@.	...@	....	....
#\$. @	#\$@.	#@..					
.#. .	.#. .	.#. .					
..#. .	..#. .	..#. .					
....	....	....					

► drdruuull

## chemins le plus long possible ?

- dans une grille de 4x4 ?
- dans une grille de 5x5 ?
- dans une grille de 6x6 ?

## séquence la plus courte à Breakthrough à captures forcées



## exemple de problèmes possibles

- ▶ séquence la + courte (respect. + longue) pour une position
- ▶ positions atteignables avec une longueur de séquence fixée à partir d'une position
- ▶ positions à une séquence de longueur fixée de la victoire

## Depth Limited Search (DLS)

- ▶ recherche en profondeur limitée
  - jusqu'à un état *WIN*
  - interrompue sur un état *LOST*
  - interrompue à la profondeur *DLS\_MAX\_DEPTH*
- ▶ variables globales
  - *current\_s* la position courante
  - $\mathcal{H}$  la table de hachage des profondeurs des états évaluées
  - *solved* un booléen à vrai si une solution est trouvée
- ▶ `applyMove(m)` joue/ajoute *m* sur *current\_s*
- ▶ `undoMove(m)` déjoue/retire *m* sur *current\_s*
- ▶ `applyMove(m)` et `undoMove(m)` modifient *current\_s*

problème d'horizon de la fonction d'évaluation / profondeur

- ▶ évaluation à *DEPTH*  $\neq$  évaluation à (*DEPTH* + 1)

## DLS : recherche en profondeur limitée

```

1 fonction DLS ( d ) :
2   if solved then return ;
3    $\mathcal{H}[\text{current\_s}] \leftarrow d$  ;
4   if  $h(\text{current\_s}) < h(\text{best\_s})$  then
5      $\text{best\_s} \leftarrow \text{current\_s}$  ;
6   if current_s == WIN then
7     solved  $\leftarrow$  true ;
8     return ;
9   if current_s == LOST or d == DLS_MAX_DEPTH then
10    return ;
11   $\mathcal{M} \leftarrow \text{nextMoves}()$  ;
12  for each  $m \in \mathcal{M}$  do
13    applyMove ( m ) ;
14    if current_s  $\notin \mathcal{H}$  or  $\mathcal{H}[\text{current\_s}] > d$  then
15      DLS ( d + 1 ) ;
16    undoMove ( m ) ;
17    if solved then return ;

```

## DLS : recherche en profondeur limitée

- ▶ sans état global *current\_s*
- ▶ avec un état local *s*

```

1 fonction DLS ( s, d ) :
2   if solved then return ;
3    $\mathcal{H}[s] \leftarrow d$  ;
4   if  $h(best\_s) > h(s)$  then
5      $best\_s \leftarrow s$  ;
6   if  $s == WIN$  then
7     solved  $\leftarrow true$  ;
8     return ;
9   if  $s == LOST$  or  $d == DLS\_MAX\_DEPTH$  then
10    return ;
11   $\mathcal{M} \leftarrow nextMoves(s)$  ;
12  for each  $m \in \mathcal{M}$  do
13     $s' \leftarrow applyMove(s, m)$  ;
14    if  $s' \notin \mathcal{H}$  or  $\mathcal{H}[s'] > d$  then
15      DLS (  $s'$ ,  $d + 1$  ) ;
16    if solved then break ;

```

## DLS : recherche en profondeur limitée

- avec la solution : *solution\_size* et *best\_s*

```
1 fonction DLS ( s, d ) :  
2   if solution_size  $\neq$  0 then return ;  
3    $\mathcal{H}[s] \leftarrow d$  ;  
4   if  $h(\text{best\_s}) > h(s)$  then  
5      $\text{best\_s} \leftarrow s$  ;  
6   if  $s == \text{WIN}$  then  
7      $\text{solution\_size} \leftarrow d$  ;  
8     return ;  
9   if  $s == \text{LOST}$  or  $d == \text{DLS\_MAX\_DEPTH}$  then  
10    return ;  
11   $\mathcal{M} \leftarrow \text{nextMoves}(s)$  ;  
12  for each  $m \in \mathcal{M}$  do  
13     $s' \leftarrow \text{applyMove}(s, m)$  ;  
14    if  $s' \notin \mathcal{H}$  or  $\mathcal{H}[s'] > d$  then  
15       $\text{solution}[d] \leftarrow m$  ;  
16      DLS (  $s'$ ,  $d + 1$  ) ;  
17    if solution_size  $\neq$  0 then break ;
```

## DLS : initialiser la recherche

- ▶ fixer la profondeur max
- ▶ vider la table des états déjà évalués
- ▶ initialiser la taille de la solution
- ▶ définir *best\_s* avec l'état courant
- ▶ lancer la recherche en profondeur limitée

```
1 DLS_MAX_DEPTH  $\leftarrow$  10 ;  
2  $\mathcal{H} \leftarrow \emptyset$  ;  
3 solution_size  $\leftarrow$  0 ;  
4 best_s  $\leftarrow$  p ;  
5 DLS ( p, 0 ) ;
```

## Breadth First Search (BFS)

- ▶ recherche en largeur d'abord (limitée en profondeur)
  - jusqu'à un état *WIN*
  - interrompue sur un état *LOST*
  - interrompue à la profondeur *BFS\_MAX\_DEPTH*
- ▶ variables globales
  - *solution\_state* l'état correspondant à la victoire trouvée
  - *best\_s* la meilleure solution trouvée
- ▶ variables locales
  - $\mathcal{L}$  la liste des états à évaluer
  - $d$  la profondeur courante
  - $\mathcal{L}'$  la liste des fils de  $\mathcal{L}$



## Breadth First Search (BFS)

```

1 fonction BFS (  $\mathcal{L}$ ,  $d$  ) :
2   if solution_state  $\neq 0$  then return ;
3    $\mathcal{S} \leftarrow \emptyset$  ;
4   for each  $s \in \mathcal{L}$  do
5      $\mathcal{M} \leftarrow \text{nextMoves} ( s )$  ;
6     for each  $m \in \mathcal{M}$  do
7        $s' \leftarrow \text{applyMove} ( s, m )$  ;
8        $\mathcal{S} \leftarrow \mathcal{S} + (s', m)$  ;
9    $\mathcal{L}' \leftarrow \emptyset$  ;
10  for each  $(s, m) \in \mathcal{S}$  do
11    if  $s \notin \mathcal{H}$  then
12       $\mathcal{H}[s] \leftarrow m$  ;
13      if  $h(\text{best\_s}) > h(s)$  then  $\text{best\_s} \leftarrow s$  ;
14      if  $s == \text{WIN}$  then
15         $\text{solution\_state} \leftarrow s$  ;
16        return ;
17      if  $s == \text{LOST}$  or  $d == \text{BFS\_MAX\_DEPTH}$  then
18        return ;
19       $\mathcal{L}' \leftarrow \mathcal{L}' + s$  ;
20  BFS (  $\mathcal{L}'$ ,  $d + 1$  ) ;

```

### défaut de DLS

- ▶ première solution possiblement sous-optimale
- ▶ rechercher toutes les solutions pour obtenir la solution optimale

### défaut de BFS

- ▶ première solution = solution optimale
- ▶ coût du calcul et du stockage de  $\mathcal{L}$

## Iterative Deepening Search (IDS)

- ▶ recherche DLS à profondeur itérative
- ▶ première solution = solution optimale
- ▶ pas de calcul et de stockage de  $\mathcal{L}$

```
1 fonction IDS ( p ) :  
2   solution_size  $\leftarrow$  0 ;  
3   best_s  $\leftarrow$  p ;  
4   for depth  $\in$  [1; IDS_MAX_DEPTH] do  
5      $\mathcal{H} \leftarrow \emptyset$  ;  
6     DLS_MAX_DEPTH  $\leftarrow$  depth ;  
7     DLS ( p , 0 ) ;  
8     if solution_size  $\neq$  0 then break ;
```