

L I P 6

A

C

A

S

A

IA et Jeux



Jean-Gabriel Ganascia

IA & Jeux



1. IA & jeux d' échec une « longue histoire »

- ◆ **Turing, 1947:** première version du test
- ◆ **Turing, 1950:** premier programme de jeu (simulation manuelle)
- ◆ **Shannon, 1950:** plan d' action pour les échecs
- ◆ 1958: un ordinateur bat un homme au jeu d' échec
- ◆ **Simon, 1958:** « dans dix ans, un ordinateur sera champion du monde... »





1. IA & jeux d' échec une « longue histoire » (suite)

C

- ◆ 1966: un enfant de dix ans bat un ordinateur
- ◆ 1978: pari entre David Lévy (maître international) et John McCarthy
 - ◆ 1978: « CHESS 4.7 » est battu par D. Lévy
 - ◆ 1989: « Deep Thought » conçu par Hsu l'emporte sur D. Lévy
- ◆ 1989: match Kasparov « Deep Thought »
- ◆ Hsu est embauché par IBM, son programme est rebaptisé « Deep Blue »

A

S

A





1: IA & jeux d' échec une « longue histoire » (suite(suite))

C

- ◆ Février 1996, Philadelphie

- ◆ Match en 6 parties entre Kasparov et « Deep Blue » (3 parties pour Kasparov, 1 pour « Deep Blue », 2 nulles)

- ◆ 3-11 mai 1997, nouveau match

- ◆ Kasparov gagne la première partie
 - ◆ « Deep Blue » emporte la deuxième
 - ◆ Les trois parties suivantes furent nulles
 - ◆ Garry Kasparov déclare forfait après le 19ième mouvement...



LIP 6 1. Dernier épisode: 2003X3D - Fritz

A

C

A

S

A

- ◆ Kasparov contre X3D - Fritz
- ◆ Fritz: jeux d' échec
- ◆ X3D: réalité virtuelle



Katie Horn
Miss New-York





C

A

S

A



Jean-Gabriel Ganascia

Le jeu de Go

- ◆ Problème beaucoup plus difficile que les échecs
- ◆ Il a fallu attendre 2008 pour qu'un programme rivalise avec des professionnels
- ◆ 2008: Mogo gagne une partie face à Kim MyungWan 8^e dan
- ◆ 20 mars 2013: Crazy Stone bat Ishida Yoshio, 9^e dan

IA & Jeux



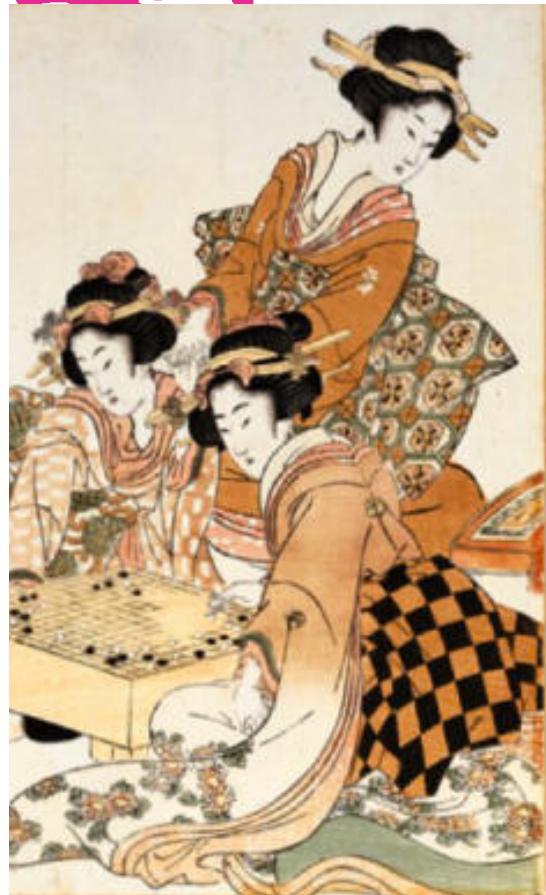


C

A

S

A



Jean-Gabriel Ganascia

Le jeu de Go (*suite*)

- ◆ Octobre 2015: AlphaGo de la société DeepMind bat le meilleur joueur européen, Fan Hui, 2^{ème} dan (5-0 parties lentes, 3-2 parties rapides)
- ◆ Mars 2016: AlphaGo l'emporte sur Lee Sedol, joueur 9^{ème} dan, considéré comme le meilleur au monde…

IA & Jeux





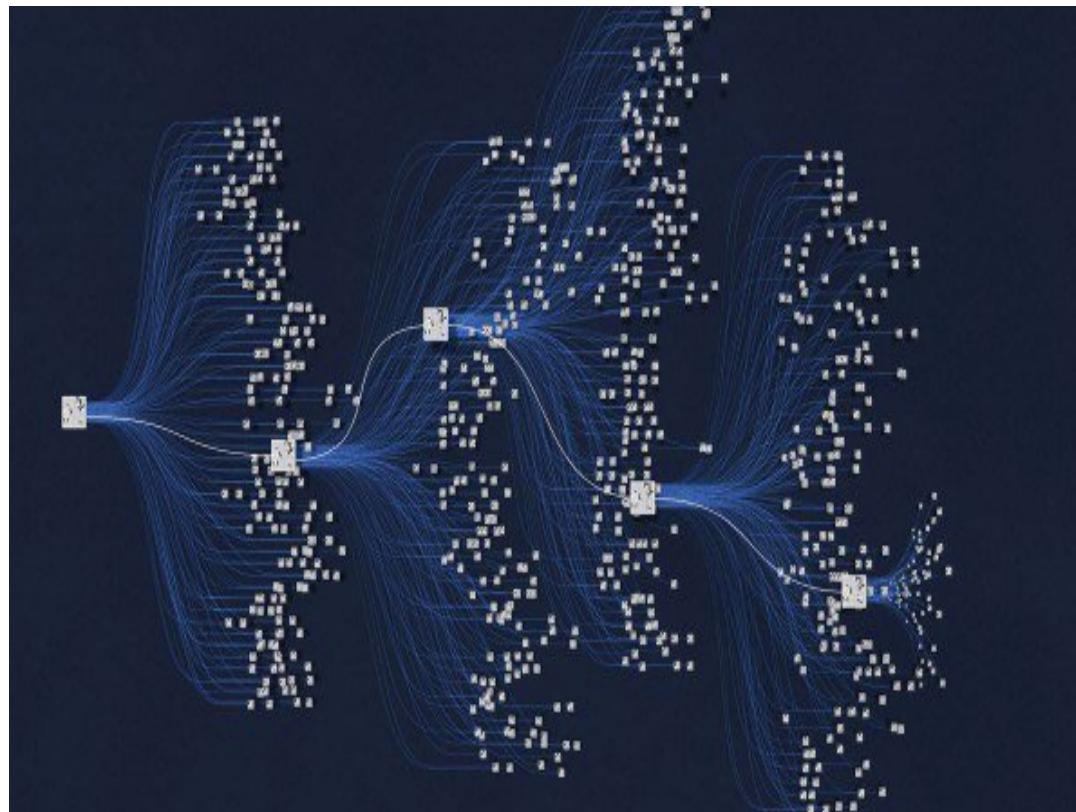
A

S

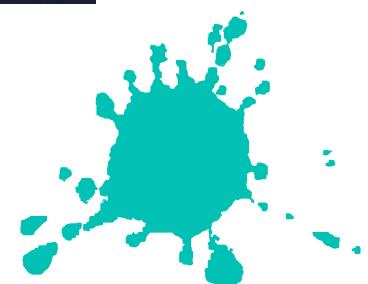
A

Jean-Gabriel Ganascia

2017 — AlphaGo Zero: la machine joue avec elle-même



IA & Jeux





A

C ♦ Écart

- ♦ Badinage, plaisanterie, ...

A ♦ Les sciences peuvent-elles conseiller les joueurs ?

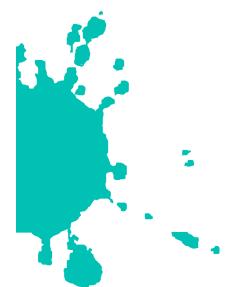
S

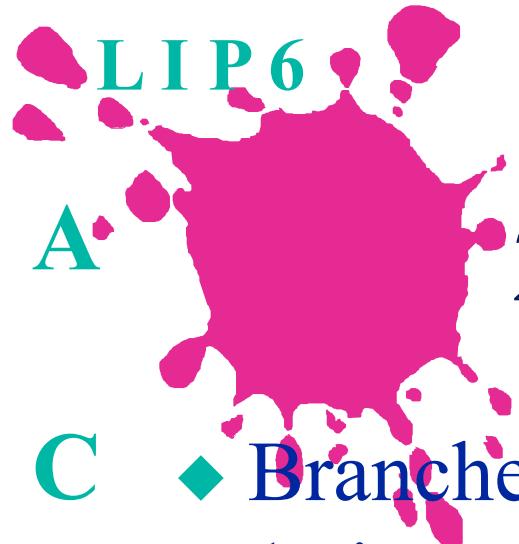
- ◆ XVII^e siècle: Pascal, Buffon
- ◆ XVIII^e siècle: Pierre-Remond de Montmord, Nicolas Bernoulli

A

- ◆ XIX^e siècle: Augustin Cournot
- ◆ XX^e siècle: Ernst Zermelo, Emil Borel, Johann von Neumann, Oskar Morgenstern, ...

Je





A

C

A

S

A

2.1. Théorie des jeux

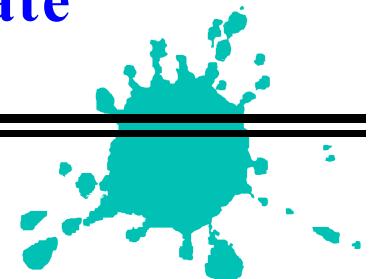
- ◆ Branche de la recherche opérationnelle (science des choix rationnels):
 - ◆ Acteurs multiples cherchant chacun à maximiser ses gains
 - ◆ Applications à l'économie, à la stratégie militaire ou politique, ...
- ◆ Principe de rationalité: « maximin »
 - ◆ Un joueur tente de maximiser ses gains sachant que son adversaire tente de minimiser ses pertes





2.2. Nicodème & Mithridate

	Mithridate	
A Nicodème	Mithridate parle	Mithridate se tait
Nicodème parle S	3 ans de prison pour Nicodème et Mithridate	Acquittement pour Nicodème, 5 ans pour Mithridate
Nicodème se tait A	5 ans pour Nicodème et acquittement pour Mithridate	1 an de prison pour Nicodème et Mithridate



2.2. Nicodème & Mithridate (suite)

- ◆ Quels seront les choix de Nicodème et de Mithridate s'ils ne communiquent pas ?
- ◆ Est-il possible que, tout en suivant leur intérêts individuels les hommes s'accordent sur une notion d'intérêt collectif ?
(applications éthiques et politiques)
- ◆ « Dilemme du prisonnier » itéré:
 - ◆ Que se passe t-il si Nicodème et Mithridate se retrouvent ?

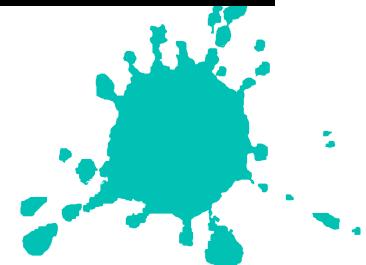




2.3. Choix multiples

Le “Baccara du bagne”

Mithridate	Pierre	Ciseaux	Papier
Nicodème			
Pierre	Nicodème 0 / Mithridate 0	Nicodème 1 / Mithridate 0	Nicodème 0 / Mithridate 1
Ciseaux	Nicodème 0 / Mithridate 1	Nicodème 1 / Mithridate 0	Nicodème 1 / Mithridate 0
Papier	Nicodème 1 / Mithridate 0	Nicodème 0 / Mithridate 1	Nicodème 0 / Mithridate 0





A

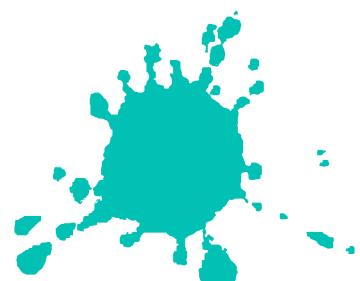
S

A

2.4. Joueurs multiples

C

- ◆ Chaque joueur possède n possibilités
 - ◆ Supposons que trois joueurs doivent choisir entre deux couleurs (bleu et vert)
 - ◆ Si les trois choisissent la même couleur, ils perdent tous 1
 - ◆ Si deux choisissent la même, et le troisième une couleur différente, ils lui donnent 2 chacun, en sorte que celui-ci gagne 4.





		Joueurs multiples			
		Le joueur 1 joue contre la coalition des autres: on se ramène à deux joueurs...			
		J2 Bleu	J2 Bleu	J2 Vert	J2 Vert
Tous les autres		J3 Bleu	J3 Vert	J3 Bleu	J3 Vert
Joueur 1		-1, -1, -1	-2, -2, 4	-2, 4, -2	4, -2, -2
S		Bleu			
A		Vert	4, -2, -2	-2, 4, -2	-2, -2, 4
					-1, -1, -1





A

: 2.5. Différents jeux

C

- ◆ Nombre de joueurs: 1, 2, n...
- ◆ finis (l'ensemble des stratégies des joueurs est fini), infinis

A

- ◆ Coopératifs, non coopératifs

S

- ◆ Simultanés (baccarat du bagne), séquentiels

A

- ◆ Somme nulle, somme non nulle (dilemme prisonniers)

- ◆ Répétés





A

S

A

2.5. Différents jeux: information

- ◆ **Information complète:** chaque joueur connaît ses possibilités d'action, celles des autres joueurs, les gains associés aux actions, les motivations des autres joueurs
- ◆ **Information parfaite:** chaque joueur à une connaissance de toute l'histoire du jeu

- ◆ Information complète et parfaite

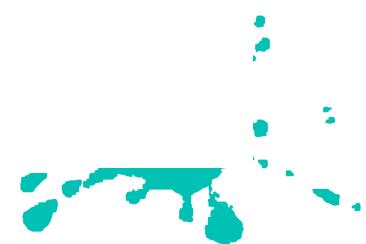
- ◆ Taquin (1 joueur)
 - ◆ Morpion, tic-tac-to, échec, dames, ...

- ◆ Information complète mais imparfaite

- ◆ « Baccarat du bagne », ...

- ◆ Information incomplète (tj. imparfaite)

- ◆ Bridge, poker, ...





A

3. Informatique et jeux

C ◆ Comment construire un programme qui gagne?

◆ Différences avec la théorie des jeux:

A ◆ Théorie des jeux: stratégie optimale, quelque soit l'adversaire, théorème, démonstration

S ◆ Programmation: fabrication de machine, apprentissage,...

A La théorie des jeux vise à supprimer le jeu...

A L'IA vise au contraire à faire jouer...

3. Différence théorie des jeux et programmation des jeux

- ◆ Jeu d' échec: plus de possibilités que de particules dans notre galaxie
- ◆ La théorie des jeux démontre qu' il existe une stratégie minimisant les pertes (partie nulle)
- ◆ L' IA fabrique des machines qui s' affrontent aux hommes, sans certitude de les gagner.





3.1. Solitaires: jeux à un joueur

A

C ♦ Recherche dans un graphe

♦ Exemple: taquin

A

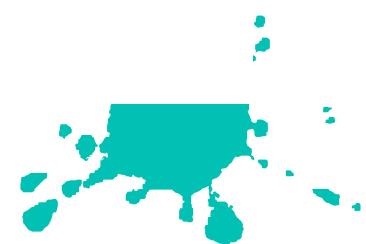
4	6	2
1		8
7	5	3



S

1	2	3
4		5
6	7	8

A



3.1. TAQUIN

A

◆ Représentation des états

Terme: `etat(1, 2, 3, 4, v, 5, 6, 7, 8)`

C

◆ Représentation de la condition terminale

`etat_final(etat(1, 2, 3, 4, v, 5, 6, 7, 8)).`

A

`transition(etat(v, E2, E3, E4, E5, E6, E7, E8, E9),
 etat(E2, v, E3, E4, E5, E6, E7, E8, E9)).`

`transition(etat(v, E2, E3, E4, E5, E6, E7, E8, E9),
 etat(E4, E2, E3, v, E5, E6, E7, E8, E9)).`

S

`transition(etat(E1, E2, v, E4, E5, E6, E7, E8, E9),
 etat(E1, v, E2, E4, E5, E6, E7, E8, E9)).`

`transition(etat(E1, E2, v, E4, E5, E6, E7, E8, E9),
 etat(E1, E2, E6, E4, E5, v, E7, E8, E9)).`

A

`transition(etat(E1, E2, E3, E4, E5, E6, v, E8, E9),
 etat(E1, E2, E3, v, E5, E6, E4, E8, E9)).`

...

`transition(etat(E1, E2, E3, E4, E5, E6, E7, E8, v),
 etat(E1, E2, E3, E4, E5, v, E7, E8, E6)).`

3.1. TAQUIN: solution profondeur N parcours de graphe

C

```
taquin([E], 0) :- etat_initial(E), etat_final(E).  
taquin(Solution, N) :- etat_initial(EI),  
                      etat_final(EF),  
                      taquin(EI, [], Solution, EF, N),  
                      imprimer_solution(Solution).
```

A

S

```
taquin(EF, A, [], EF, N) :- longueur(A, LA), LA < N.  
taquin(EI, A, [EI|L], EF, N) :-  
    longueur(A, LA), LA < N,  
    transitions(EI, TR),  
    difference(TR, [EI|A], C),  
    member(EJ, C),  
    taquin(EJ, [EI|A], L, EF, N).
```

A





A **transitions(X, L) :-**
A **bagof(Y,**
S **(transition(X,Y);**
L) . **transition(Y, X)),**

A

Calcul des transitions





3.1.1. TAQUIN: solution optimale

A

:

C

A

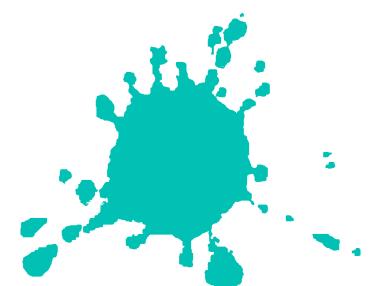
```
taquin(Solution) :- nombre(N),  
    taquin(Solution, N).
```

S

```
nombre(0).
```

```
nombre(N) :- nombre(P), N is P+1.
```

A

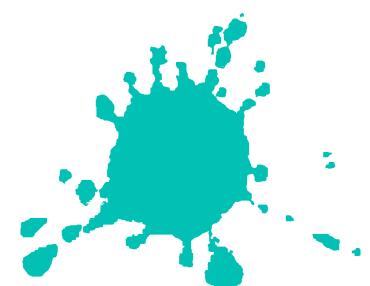




3.1.2. TAQUIN: introduction d'heuristiques

```
taquin_h :- etat_initial(E),  
          mise_a_zero,  
          gensym(etat, NE),  
          ecart_sol(E, V),  
          asserta(etat(E, NE, orphelin, V, 0)),!,  
          taquin_h.
```

A





3.1.2. TAQUIN: introduction d'heuristiques

A C A S A

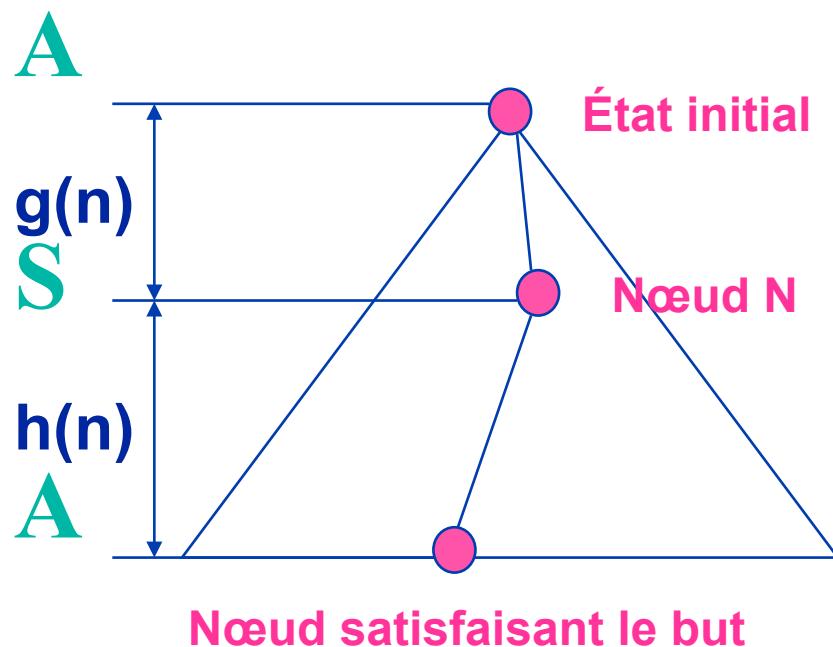
```
taquin_h :- etat(E, _, _, _, _),
           etat_final(E),
           reconstituer_solution(L),
           imprimer_solution(L).

taquin_h_ :- etat_courant(E), \
             \\ $E$  minimise
                       \
                       la fonction de coût f(E)
                       \
                       mais n'a pas été visité
             (transition(E, ES); transition(ES, E)),
             etat(E, NomPere, _, _, _),
             \+ etat(ES, _, _, _, _),
             ajout_etat(ES, NomPere), fail.

taquin_h_ :- taquin_h_.
```

Jean-Gabriel Ganascia



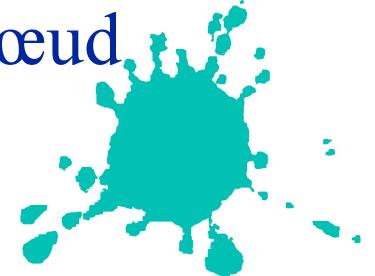


Jean-Gabriel Ganascia

Algorithme A* – rappels

- ◆ *Algorithme profondeur d'abord*
- ◆ *Fonction heuristique:*
$$f(n) = g(n) + h(n)$$
- ◆ $g(n)$ coût du chemin parcouru depuis l'état initial jusqu'au nœud n
- ◆ $h(n)$ estimation heuristique du coût du chemin optimal qui lie le nœud n avec au nœud solution.

IA & Jeux



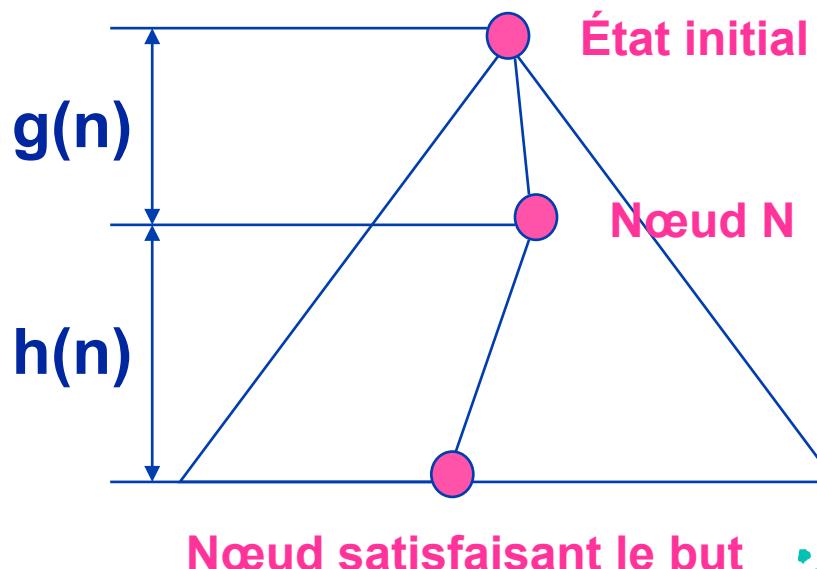


Algorithme A* – suite

C *Une fonction heuristique est dite admissiblessi*
h(s) $\leq h^*(s)$ pour tous les états s de E où h*(s) est le coût
A minimum depuis s jusqu' à la solution

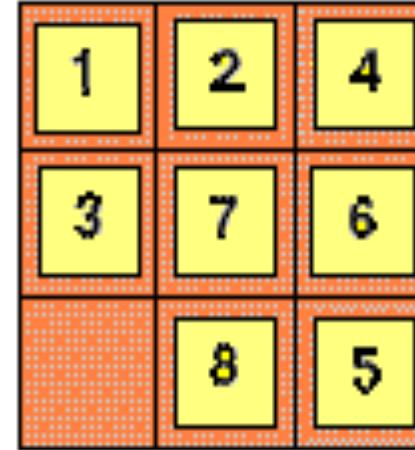
S *Si la fonction
heuristique
est admissible*
A *alors
l'algorithme A*
trouve l'optimal*

Jean-Gabriel Ganascia

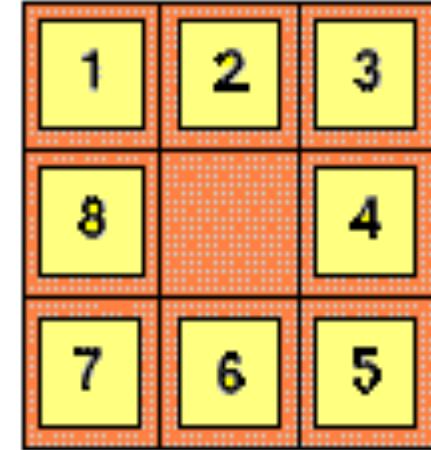


IA & Jeux

Jeu de taquin



Etat initial



Etat but

A

Exemple d' heuristiques admissibles:

$h_1(s)$: Nbre de pièces mal placées

S

$h_2(s)$: somme des distances Manhattan entre la position dans l'état i et la position dans l'état final de chaque pièce.

A

Pour les états représentés dans la figure on a

$h_1(s) = 5$

$h_2(s) =$



Distance de Manhattan de i à j

Définition: nombre minimal de déplacements pour aller de i à j

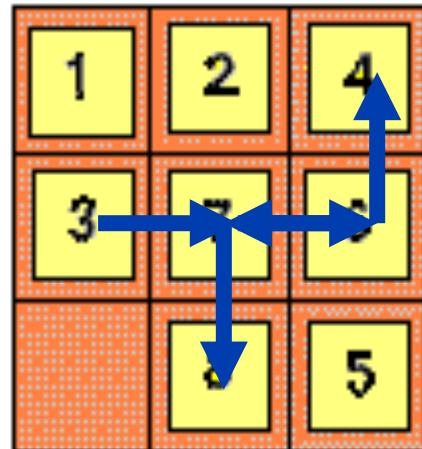
Distance de Manhattan de la position dans l' EI à celle dans l' EB

Pour 6: 2

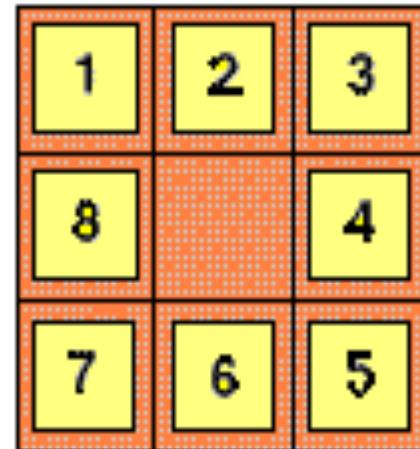
Pour 5: 0

Pour 3: 3

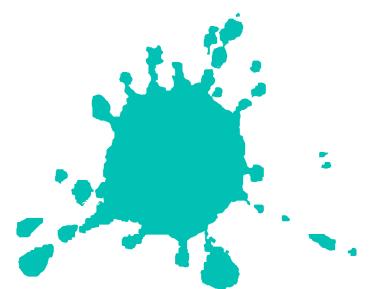
S



Etat initial



Etat but



3.1.2. Jeu de taquin

1	2	4
3	7	6
8	5	

Etat initial

1	2	3
8		4
7	6	5

Etat but

Exemple d' heuristiques admissibles:

$h_1(s)$: Nbre de pièces mal placées

$h_2(s)$: somme des distances Manhattan entre la position dans l'état i et la position dans l'état final de chaque pièce.

Pour les états représentés dans la figure on a

$$h_1(s) = 5$$

$$h_2(s) = 0 + 0 + 3 + 1 + 0 + 2 + 2 + 2 = 10$$

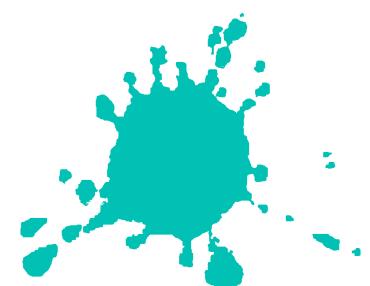
Remarque: Soit deux heuristiques admissibles, h_i et h_j

Si $\forall s \in E \ h_i(s) > h_j(s)$ on dit que h_i domine h_j



3.1.2. TAQUIN: introduction d'heuristiques Algorithme A*

- $f(s) = g(s) + h(s)$ $g(s)$: chemin parcouru
- A
- S Deux essais:
- $h(s) = 0$
- A $h(s)$ = distance de Manhattan





C

A

S

A

O

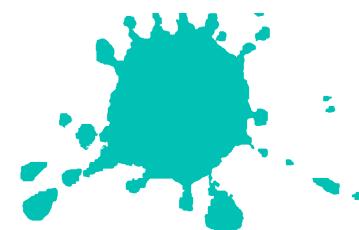
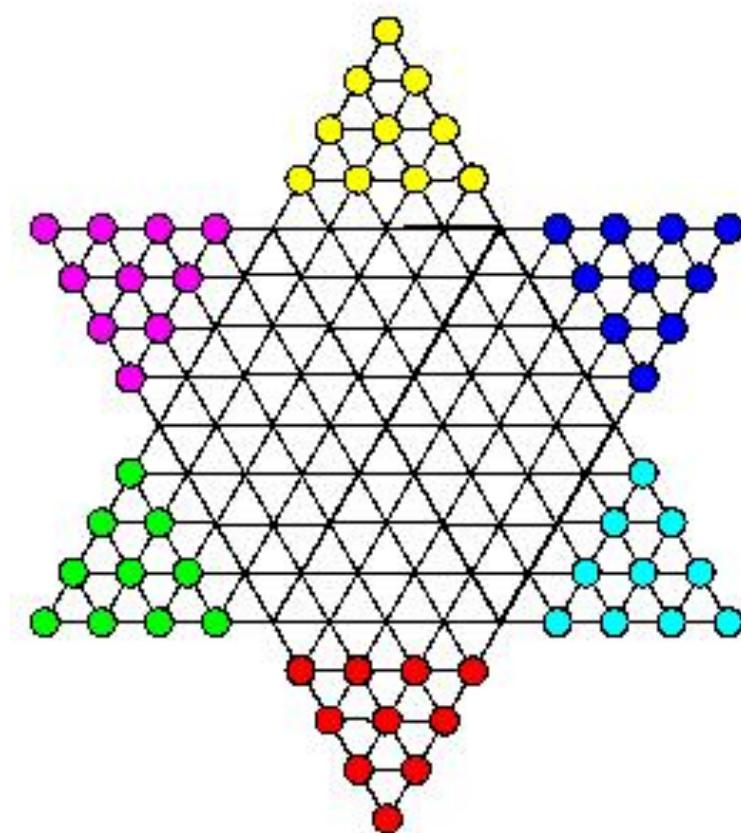
X

O

X

O

Quelques jeux à 2 ou plus





A

S

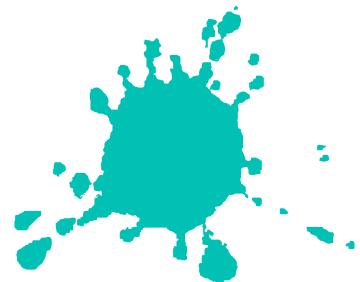
A

Jean-Gabriel Ganascia

Quatre notions essentielles

1. Arbre de jeu
2. Min-Max
3. Min-Max à profondeur finie
avec une fonction d'évaluation
4. Cela marche t-il? Et pourquoi?
5. Coupures α/β

IA & Jeux





Hexapion

A

C

A

S

A

Jean-Gabriel Ganascia

E0: Etat initial

○	○	○
●	●	●

E1: Les blancs avancent

○	○	○
	○	
●	●	●

E3: Les noirs prennent

	○	○
●		
●	●	●

E4: Les noirs avancent

	○	○
○		●
●	●	●

E5: Les blancs
prennent et gagnent

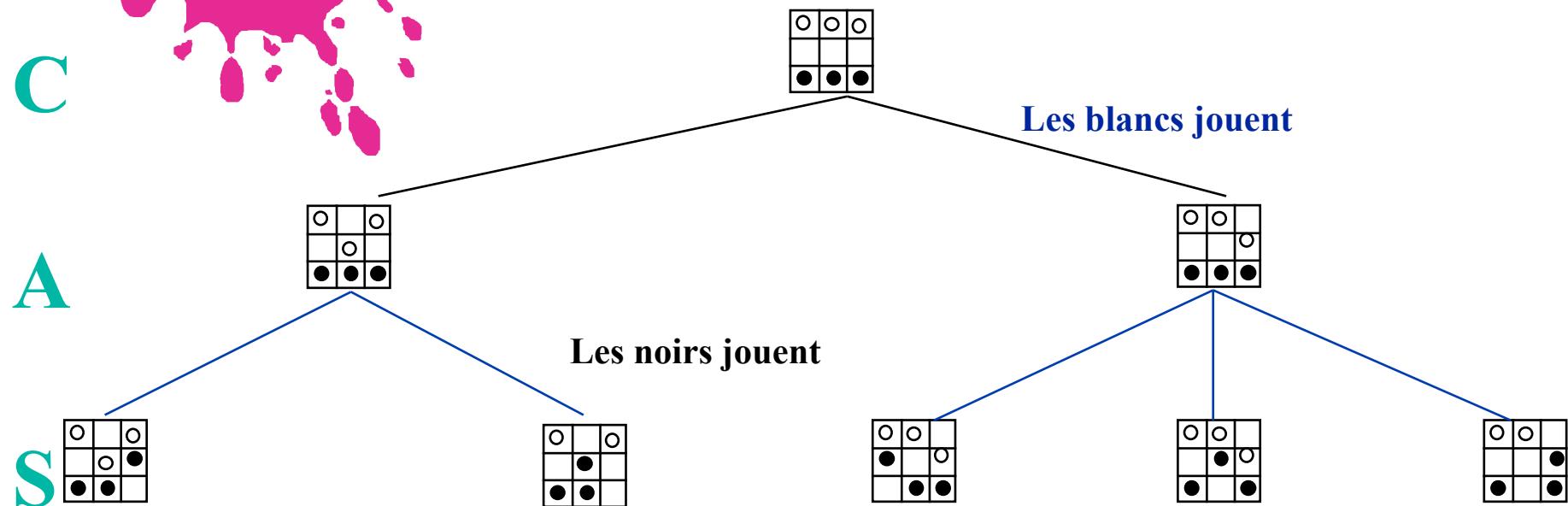
○	○	○
○		●
●	○	

UX



Hexapion

Arbre de jeux



A

Jean-Gabriel Ganascia

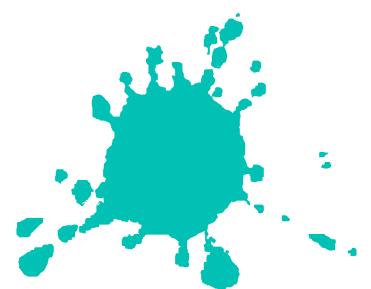
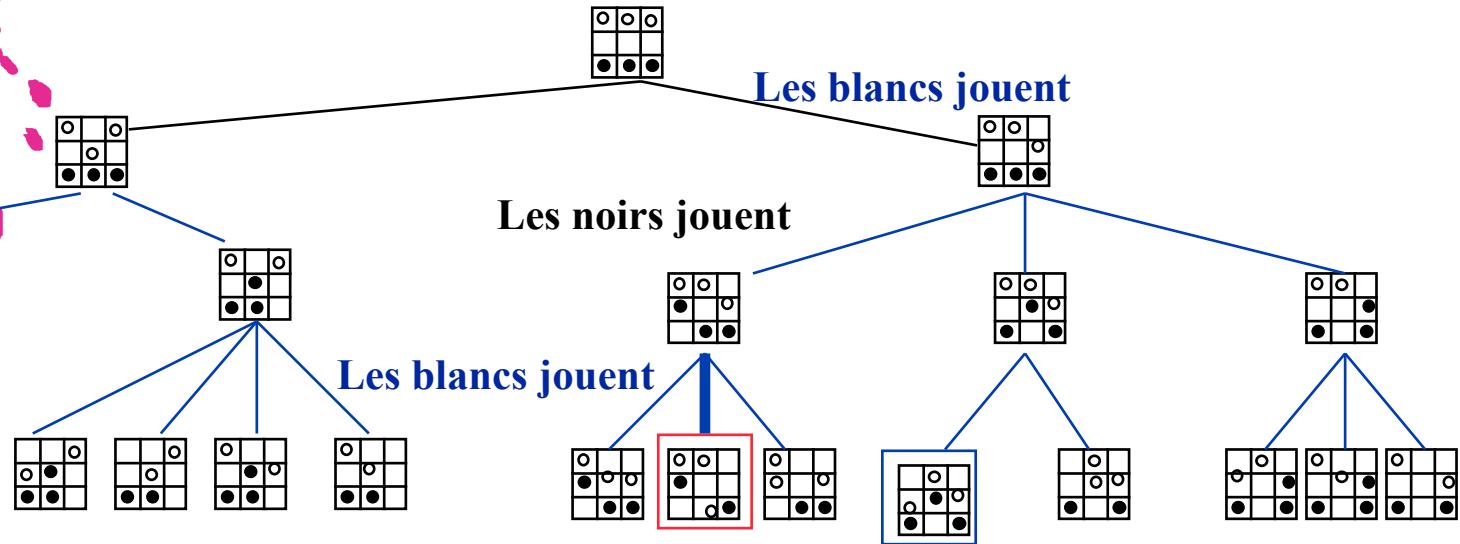
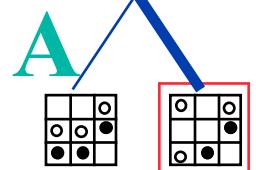
IA & Jeux

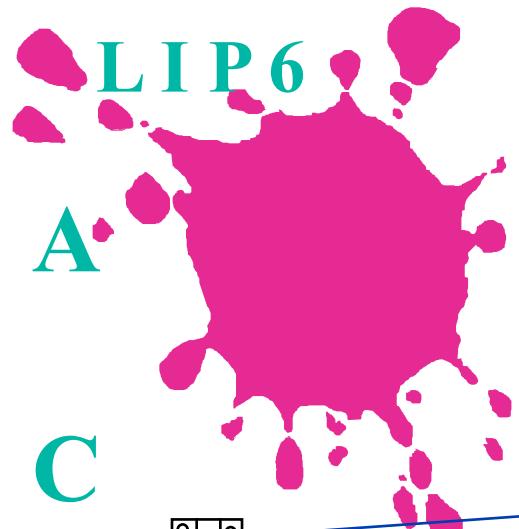




Hexapion

Arbre de jeux





A

C

A

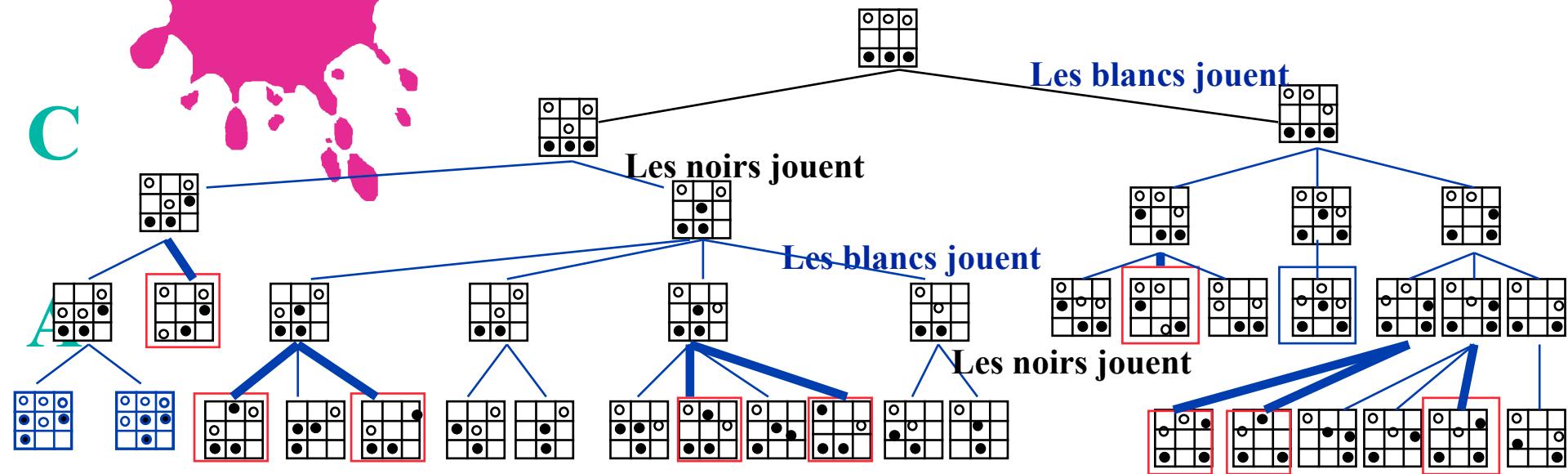
S

A

Jean-Gabriel Ganascia

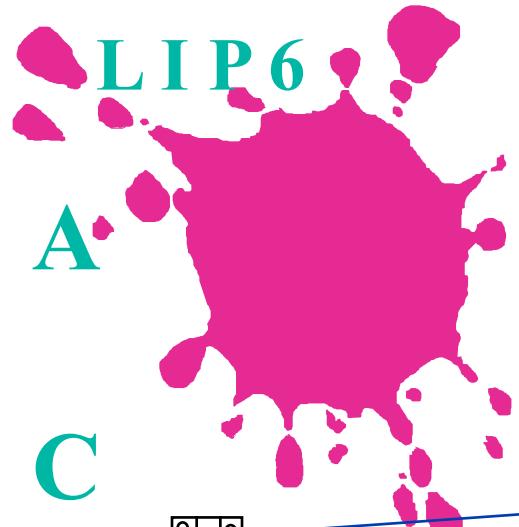
Hexapion

Arbre de jeux



IA & Jeux





Hexapion

Arbre de jeux

A

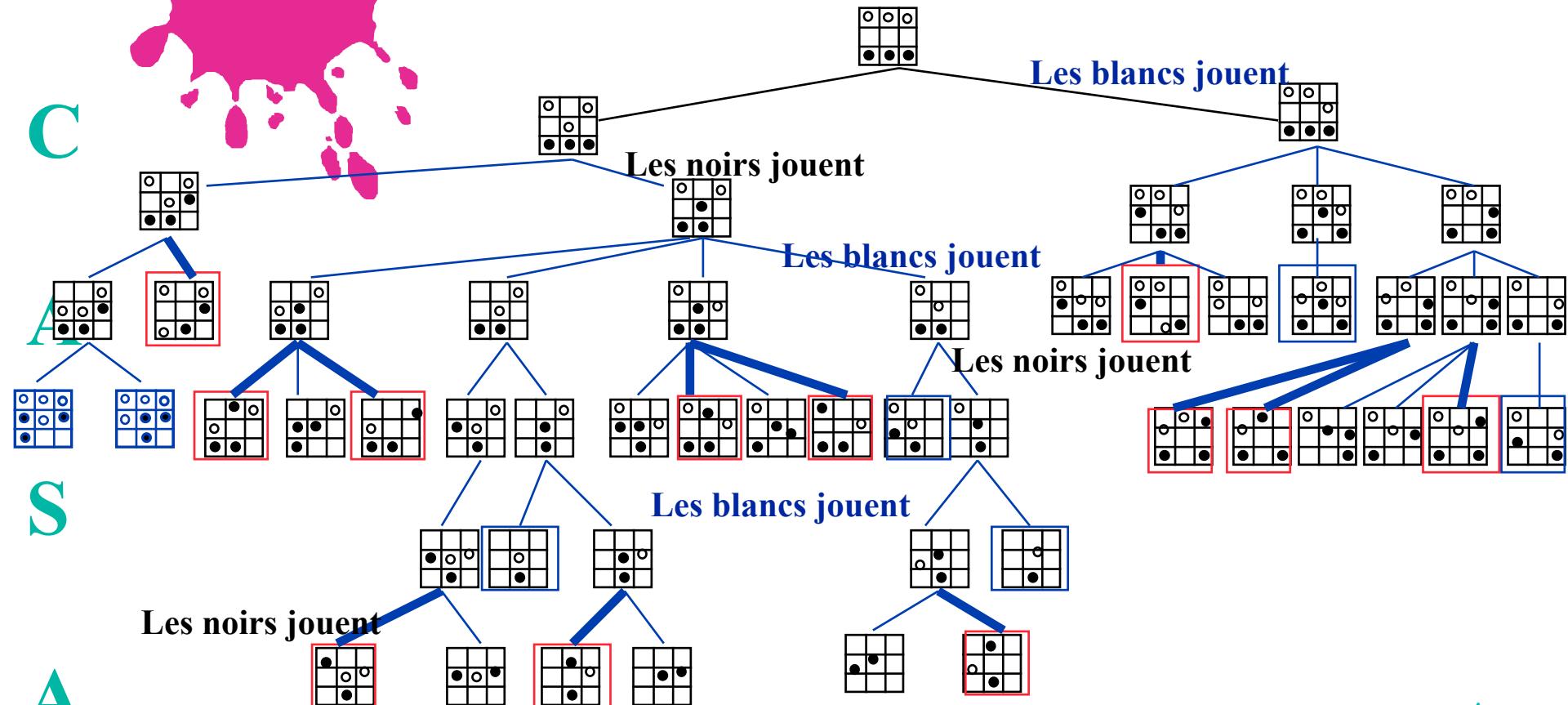
C

A

S

A

Jean-Gabriel Ganascia



IA & Jeux





A

S

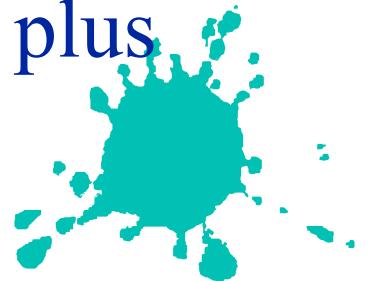
A

Jean-Gabriel Ganascia

Min-Max

- ◆ Le jeu est symétrique
- ◆ Ce que A gagne, B le perd
- ◆ A cherche à maximiser ses gains, B aussi...
- ◆ Le joueur A choisit, à chaque étape le coup qui lui est le plus favorable (MAX), sachant que le joueur B choisit celui qui est le plus défavorable à A (MIN)

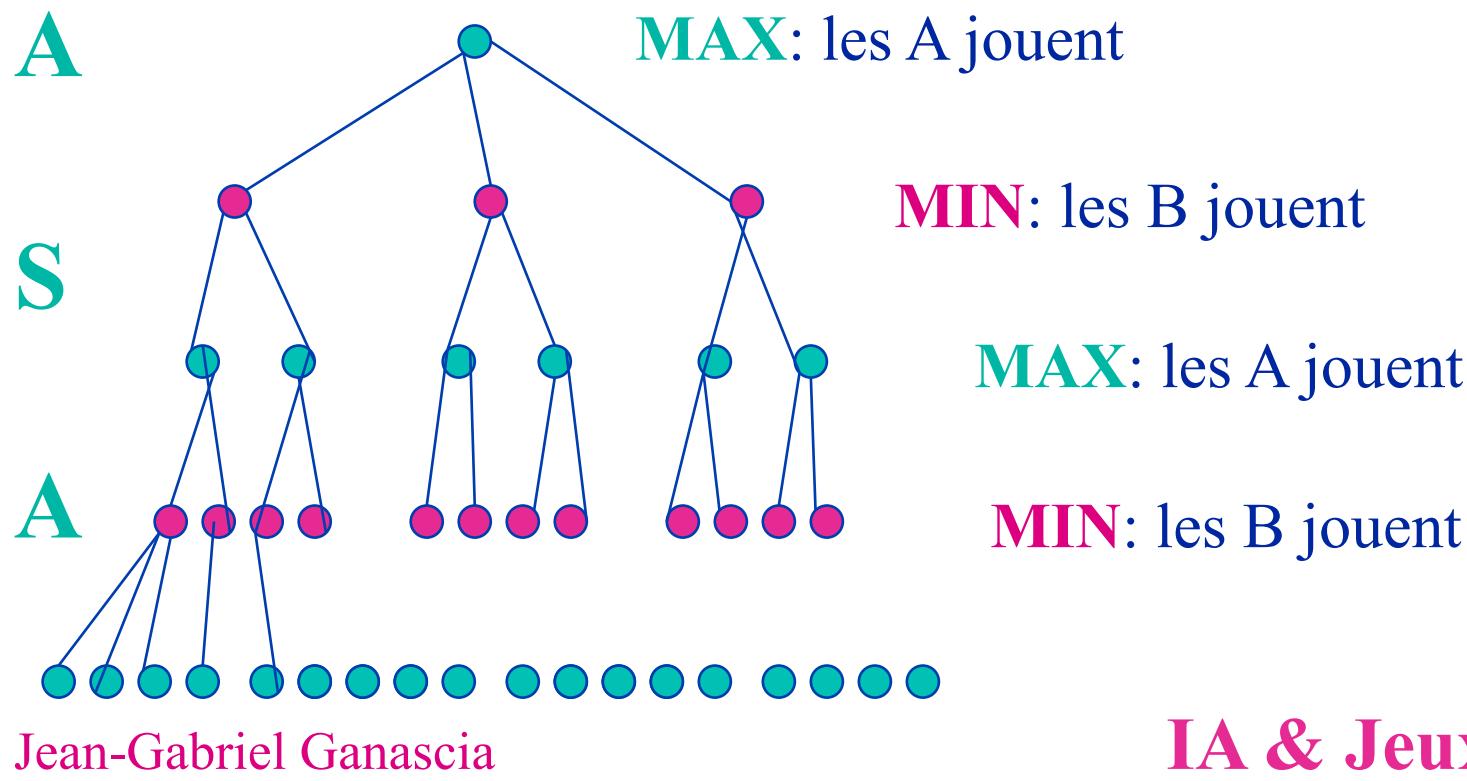
IA & Jeux





Min-Max (suite)

- ◆ Arbre de jeux
 - ◆ Niveaux Max: choix de A
 - ◆ Niveaux Min: choix de B

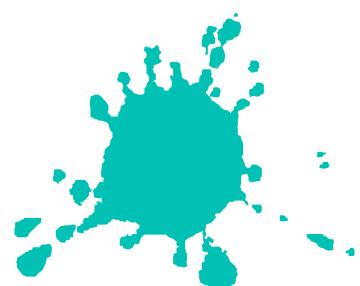
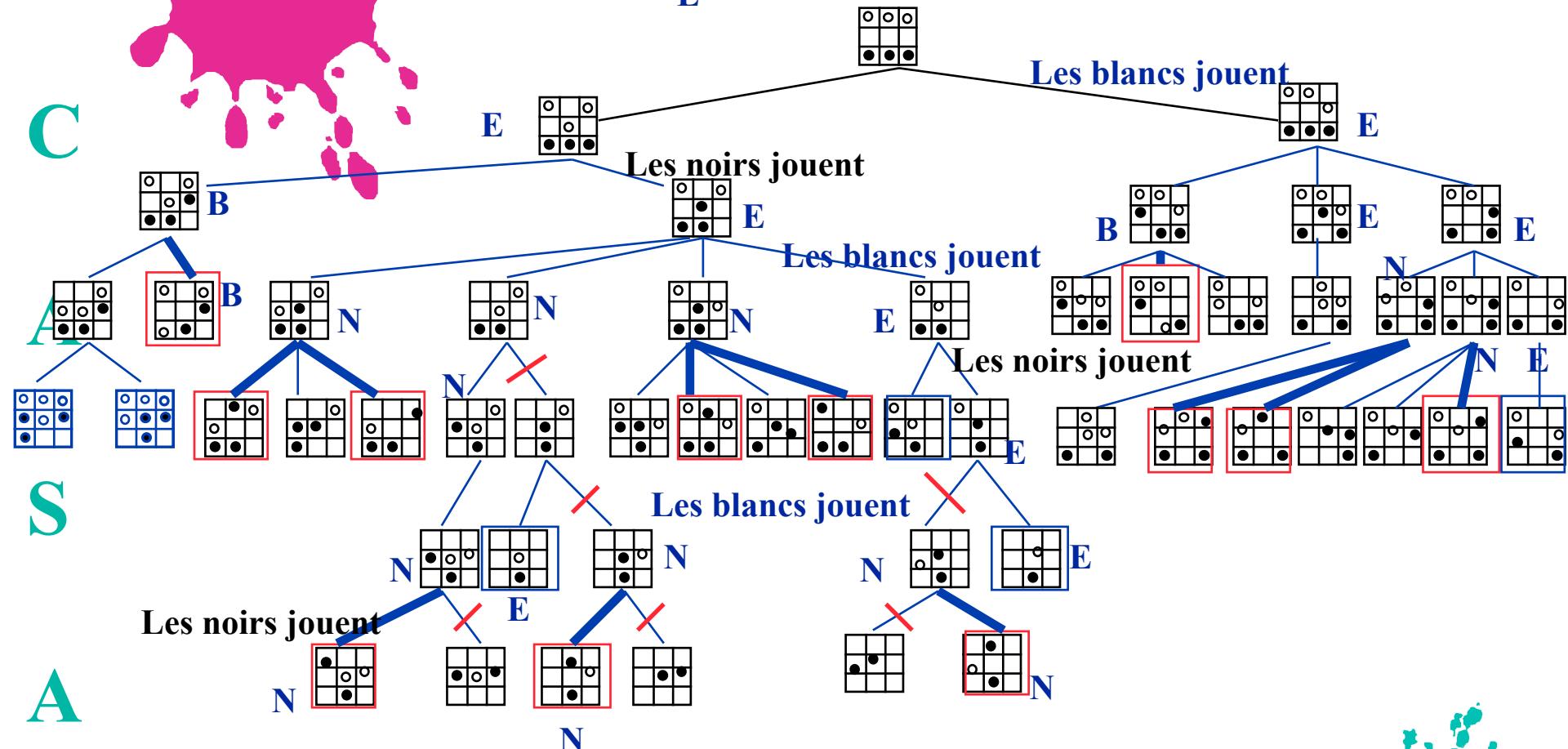


IA & Jeux



Hexapion

Principe du
Minimax



LIP 6

A

C

A

S

A

◆ Morpion 3×3

Tic-tac-to (Morpion 3×3)

X		
		X
O	X	O

◆ Représentation:
`jeu(E1, E2, E3, E4, E5, E6, E7, E8, E9)`



A

C

A

S

A

Tic-tac-toe programmation du minimax

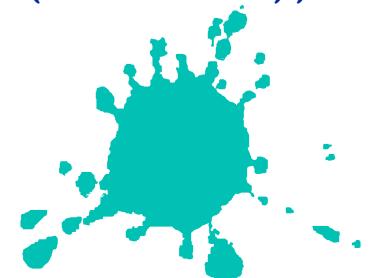
◆ Représentation d'un état

- ◆ Terme: `jeu(E1, E2, E3, E4, E5, E6, E7, E8, E9)`

◆ Condition terminale

```
etat_gagnant(jeu(C, C, C, _, _, _, _, _, _, C), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(_, _, _, C, C, _, _, _, _), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(_, _, _, _, _, C, C, C), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(C, _, _, C, _, _, C, _, _), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(_, C, _, _, C, _, _, C, _), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(_, _, C, _, _, C, _, _, C), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(C, _, _, _, C, _, _, _, C), C) :- ((C == noir) ; (C == blanc)).  
etat_gagnant(jeu(_, _, C, _, C, _, C, _, _), C) :- ((C == noir) ; (C == blanc)).
```

E1	E2	E3
E4	E5	E6
E7	E8	E9





Tic-tac-toe

programmation du minimax

◆ Transitions

```
C transition(J, _, _) :- (etat_gagnant(J, _); etat_nul(J)),!, fail.  
transition(jeu(vide, X2, X3, X4, X5, X6, X7, X8, X9), C,  
          jeu(C, X2, X3, X4, X5, X6, X7, X8, X9)).  
  
A transition(jeu(X1, vide, X3, X4, X5, X6, X7, X8, X9), C,  
          jeu(X1, C, X3, X4, X5, X6, X7, X8, X9)).  
  
A transition(jeu(X1, X2, vide, X4, X5, X6, X7, X8, X9), C,  
          jeu(X1, X2, C, X4, X5, X6, X7, X8, X9)).  
  
transition(jeu(X1, X2, X3, vide, X5, X6, X7, X8, X9), C,  
          jeu(X1, X2, X3, C, X5, X6, X7, X8, X9)).  
  
S transition(jeu(X1, X2, X3, X4, vide, X6, X7, X8, X9), C,  
          jeu(X1, X2, X3, X4, C, X6, X7, X8, X9)).  
  
transition(jeu(X1, X2, X3, X4, X5, vide, X7, X8, X9), C,  
          jeu(X1, X2, X3, X4, X5, C, X7, X8, X9)).  
  
A transition(jeu(X1, X2, X3, X4, X5, X6, vide, X8, X9), C,  
          jeu(X1, X2, X3, X4, X5, X6, C, X8, X9)).  
  
transition(jeu(X1, X2, X3, X4, X5, X6, X7, vide, X9), C,  
          jeu(X1, X2, X3, X4, X5, X6, X7, C, X9)).  
  
transition(jeu(X1, X2, X3, X4, X5, X6, X7, X8, vide), C,  
          jeu(X1, X2, X3, X4, X5, X6, X7, X8, C)).
```



Tic-tac-toe

programmation du minimax

- ◆ Etat initial:

```
jeu_init(  
    jeu(vide, vide, vide, vide, vide, vide, vide, vide, vide)).
```

- ◆ Joueur:

```
joueur_max(blanc).  
joueur_min(noir).
```

- ◆ Transitions:

```
transitions(Jeux_i, Joueur, Liste_suivants) :-  
    bagof( Js,  
          transition(Jeux_i, Joueur, Js),  
          Liste_suivants  
    ).
```



A

programmation du minimax

◆ Min-Max:

```
min_max(_, J, _, +1, _) :- joueur_max(Max),  
                           etat_gagnant(J, Max), !.  
min_max(_, J, _, -1, _) :- joueur_min(Min),  
                           etat_gagnant(J, Min), !.  
min_max(_, J, _, 0, _) :- etat_nul(J),!, increment_compteur.  
min_max(j_max, Jcourant, Jsuiv, N, Prof):-joueur_max(M),  
                                         transitions(Jcourant, M, LS),!,  
                                         → min_max_min(LS, RR, Prof),  
                                         choix_max(RR, Jsuiv, N).  
  
min_max(j_min, Jcourant, Jsuiv, N, Prof):- joueur_min(M),  
                                         transitions(Jcourant, M, LS),!,  
                                         → min_max_max(LS, RR, Prof),  
                                         choix_min(RR, Jsuiv, N).
```



Programmation du minimax

◆ min_max_min:

```
min_max_min([], [], _).  
min_max_min([J|L], [[J, Ns]|LL], P) :- PP is P + 1,  
    min_max(j_min, J, _, Ns, PP), min_max_min(L, LL, P).
```

◆ choix_max:

```
choix_max([[_ , R]|L], Js, Rs) :- choix_max(L, Js, Rs),  
    Rs > R, !.
```

```
choix_max([[J, R]|_], J, R).
```

◆ min_max_max:

```
min_max_max([], [], _).  
min_max_max([J|L], [[J, Ns]|LL], P) :- PP is P + 1,  
    min_max(j_max, J, _, Ns, PP), min_max_max(L, LL, P).
```

Un exemple

A

◆ Position de départ:

o	o	#
#	#	o
.	.	.

Les o jouent

C

A

S₀

A

0

o	o	#
#	#	o
.	.	.

o	o	#
#	#	o
.	o	.

o	o	#
#	#	o
.	o	#

Max

-1

o	o	#
#	#	o
.	.	o

Min

Max

o	o	#
#	#	o
o	#	.

o	o	#
#	#	o
o	.	#

A

Second exemple

Les o jouent

A

◆ Position de départ:

o	o	#
#	#	.
.	.	.

Max

C

Min

o	o	#
#	#	.
.	.	.

A

o	o	#
#	#	.
.	.	.

o	o	#
#	#	.
.	.	#

o	o	#
#	#	.
o	#	.

o	o	#
#	#	.
o	.	#

o	o	#
#	#	.
.	o	.

o	o	#
#	#	.
.	o	.

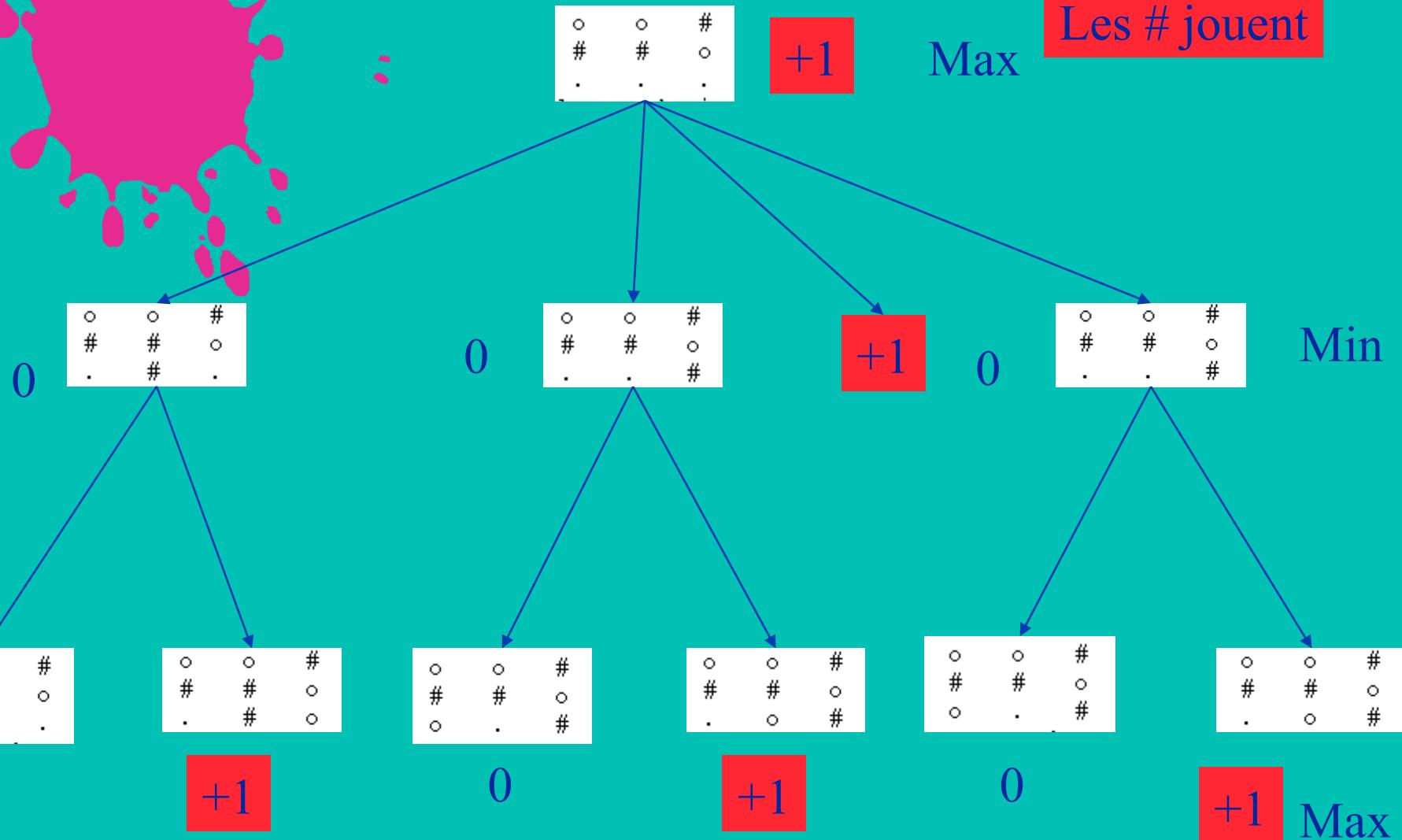
Max

A

-1

-1

Troisième exemple





A Notion de fonction d'évaluation

C

- ◆ Aux échecs, il n'est pas possible d'aller jusqu'au bout

A

- ◆ Critères d'évaluation des positions

S

- ◆ Pièces présentes

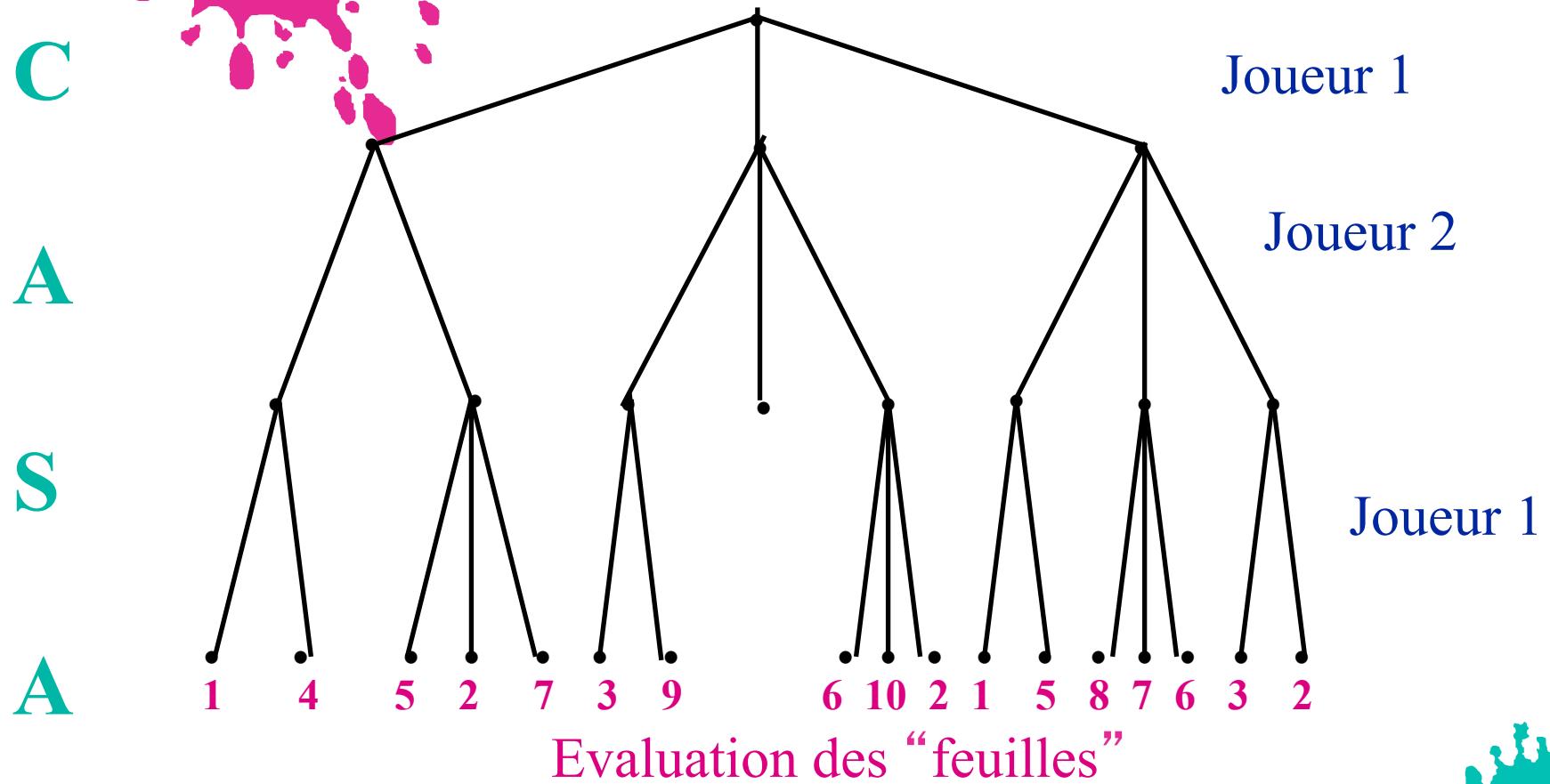
- ◆ Position (occupation du centre, des côtés, ...)

A

- ◆ « Minimax » appliqué jusqu'à une profondeur de n « demi-coups »



Profondeur 3 demi-coups





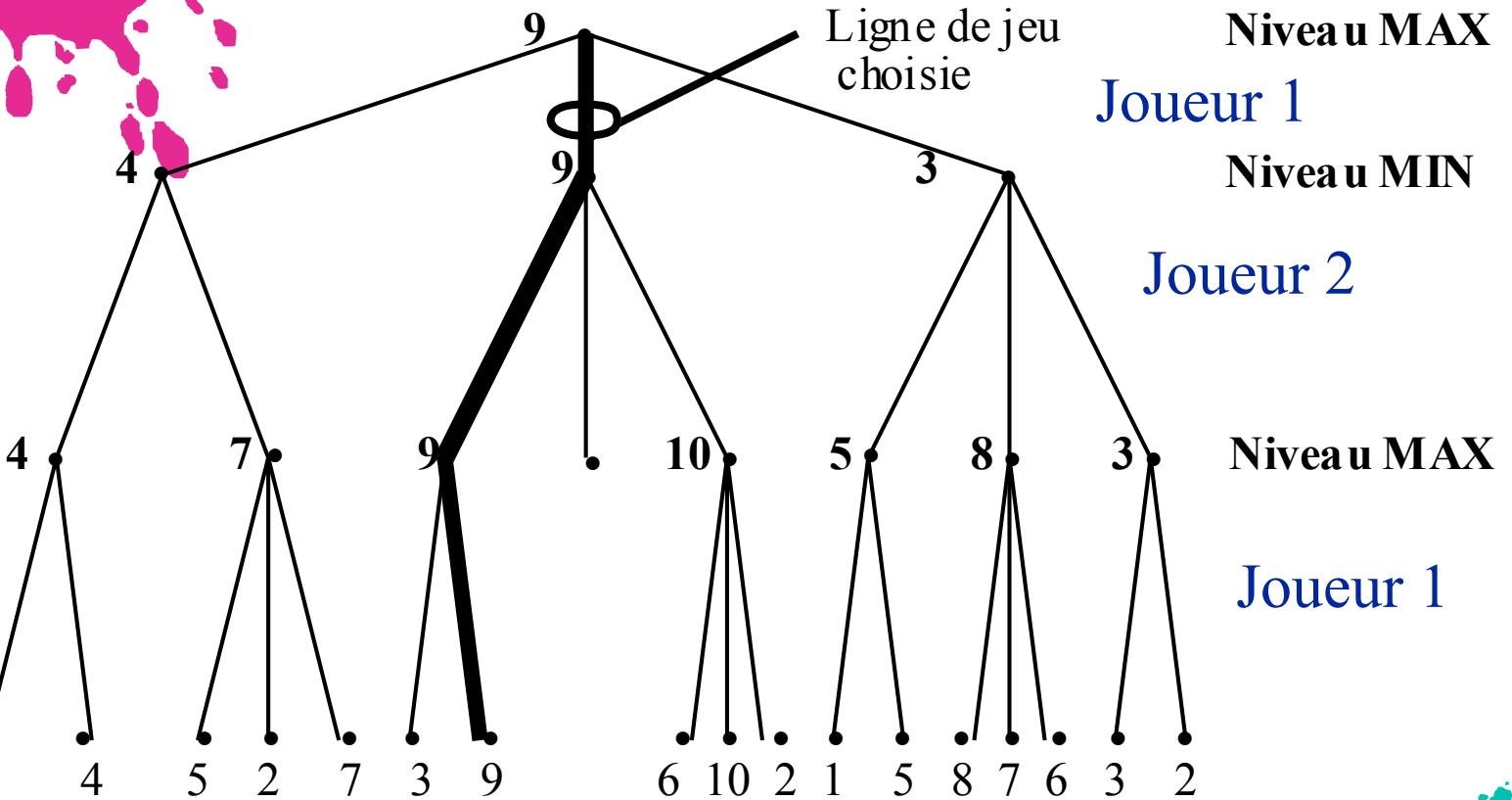
C

A

S

A

Minimax



Programmation MINIMAX avec évaluation à n demi-coups

◆ Min-Max:

```

C min_max(_, J, _, +G, _) :- joueur_max(Max),
                               etat_gagnant(J, Max), !.
min_max(_, J, _, -G, _) :- joueur_min(Min),
                               etat_gagnant(J, Min), !.
A min_max(_, J, _, 0, _) :- etat_nul(J),!, increment_compteur.
min_max(_, J, _, E, P) :- prof_max(P),!, evaluation(J, E). ←
min_max(j_max, Jcourant, Jsuv, N, Prof):-not(prof_max(P))
                                         joueur_max(M),
                                         transitions(Jcourant, M, Ls),!,
                                         min_max_min(Ls, RR, Prof),
                                         choix_max(RR, Jsuv, N).
S

```

```

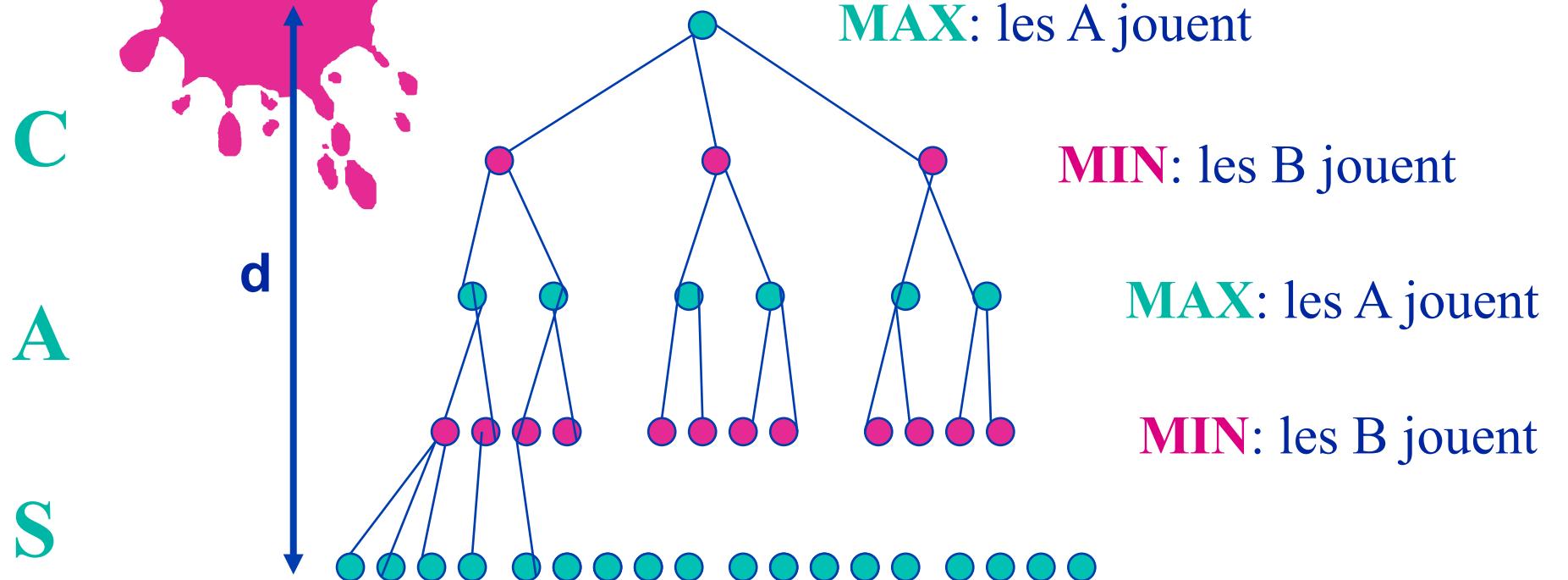
A min_max(j_min, Jcourant, Jsuv, N, Prof):- not(prof_max(P))
                                         joueur_min(M),
                                         transitions(Jcourant, M, Ls),!,
                                         min_max_max(Ls, RR, Prof),
                                         choix_min(RR, Jsuv, N).

```





Constatation empirique



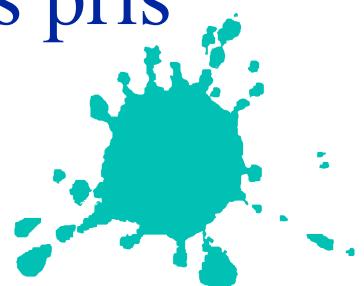
- ◆ Les programmes qui reposent sur le Minimax produisent généralement de meilleurs résultats lorsque l' on accroît la profondeur de recherche d...





Arguments heuristiques en faveur de l' anticipation

- ◆ Visibilité: plus on se rapproche de la fin, plus l' issue du jeu devient apparente et donc plus l' évaluation doit être pertinente.
- ◆ Filtrage: tandis qu'une évaluation statique est calculée sur la base des propriétés d' une seule position, la valeur renvoyée par le Minimax intègre tous les nœuds pris en considération dans la recherche





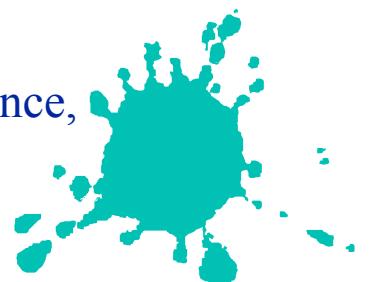
Pourquoi y a-t-il un problème?

- C ◆ Des travaux conduits au débuts des années quatre-vingts montrent que cet argument (en particulier celui du filtrage) est erroné. « Effet d' horizon »
- A ◆ Dans un classe de jeux dits *pathologiques*, plus l' exploration devient profonde, plus la qualité du résultat se dégrade...

Beal, D. An analysis of minimax. In Advances in Computer Chess, 2, page 103-109, 1980

A Nau, D. An investigation of the causes of pathology in games. Artificial Intelligence, 19 (3): 257-278, Novembre 1982

Pearl, J. On the nature of pathology in game searching. Artificial Intelligence, 20 (4): 427-453, July 1983





Exemple de jeux pathologiques: les jeux de Pearl

- C ◆ Échiquier 2^c sur 2^c , où c est un entier > 0
- A ◆ Affectation aléatoire de 1 avec la probabilité p et de 0 avec la probabilité $1-p$ sur les cases de l' échiquier
- S ◆ Mouvement: diviser l' échiquier en deux parties égales et choisir de conserver l' une ou l' autre moitié
 - ◆ Le premier joueur divise verticalement
 - ◆ Le second horizontalement
- A ◆ Les joueurs continuent alternativement jusqu' à ce qu' il ne reste qu' une seule case
 - ◆ Si la case contient 1, le dernier joueur gagne, sinon il perd



II

Jeu de Pearl

A: $c = 2 - 2^c = 4$

C

A

S

A

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

0	1
0	1
1	0
1	1

1	0
0	1

1	1
0	0

0	1
0	1

1	0
1	1

1	0
0	1

0	1
1	0

1	0
1	0

1	0
1	0

0	0
1	1

1	1
1	1



Tentatives d' explication

A

- ◆ Michon: la pathologie dépend de la probabilité de distribution du facteur de branchement

C

- ◆ Bratko & Gams, Pearl,: essais avec une évaluation multiple (et pas seulement 2 valeurs)

A

- ◆ Bratko & Gams, Nau, Beal, Lustrek etc. notent que la similitude des positions proches élimine la pathologie

S

- ◆ Pearl montre que pour éliminer la pathologie, l' erreur de la fonction d'évaluation doit décroître exponentiellement avec la profondeur de la recherche

A

- Conclusion:* la pathologie n'est généralement pas observée sur les jeux réels parce que les valeurs des positions ne sont pas indépendantes les unes des autres





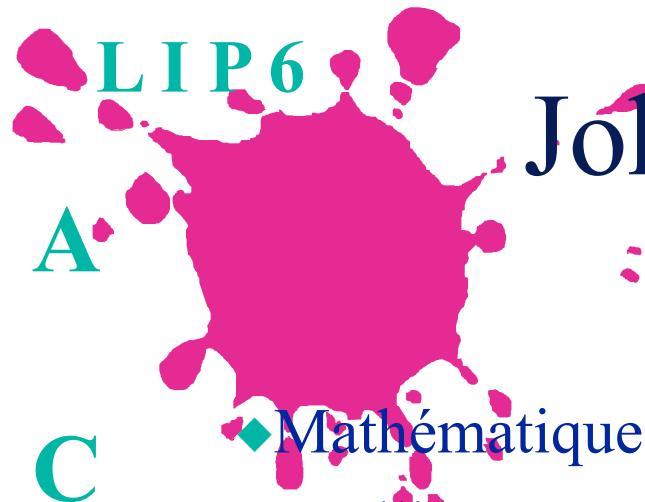
LIP6 Mitja Luštrek, Matjaž Gams, Ivan Bratko

IJCAI – 2005

C Modèle fondé sur un évaluation réelle

- ◆ Nécessaire pour jeux dont l' issue est multivaluée (Othello, etc.)
- ◆ Dans les jeux où l' issue est binaire (Échecs, Dames etc.) où éventuellement ternaire si l' on considère la partie nulle, les valeurs multiples expriment l' incertitude sur l' issue
- ◆ Scheucher & Kaindl montrent qu' une fonction d' évaluation binaire se comporte très médiocrement pour le jeu d' échec





C

A

S

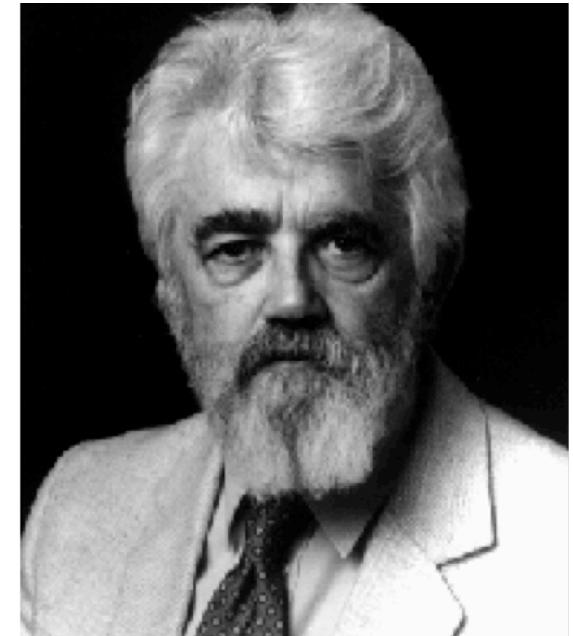
A

Jean-Gabriel Ganascia

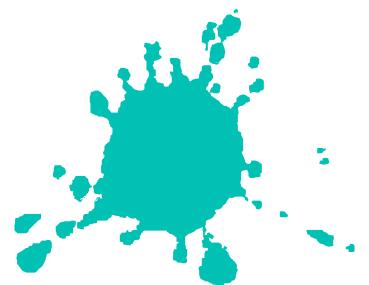
John McCarthy – 1927-2011

Œuvre

- ◆ Mathématique & théorie du calcul
- ◆ Intelligence artificielle
- ◆ Langages (ALGOL, LISP, Elephant)
- ◆ Temps partagé
- ◆ Jeux – jeu d'échec
- ◆ Calcul des situations
- ◆ Logique non-monotones – circonscription
- ◆ Créativité
- ◆ Philosophie
- ◆ Curiosités: fouille de données, logique modale, ...
- ◆ Fantaisies



IA & Jeux



LIP6

Jeu d'échec

A

◆ La drosophile de l'IA...

C

◆ Quelques dates

◆ 1912 Torres y Quevedo

A

◆ 1948 Wiener – description d'une machine qui joue aux échecs

S

◆ 1950 C. Shannon – article sur programme d'échec

A

◆ 1952 A. Turing écrit le premier programme d'échec

Jean-Gabriel Ganascia



IA & Jeux





A

C

A

S

A

Min

Max

+6

+6

+10

+6

+25

+13

+4

+3

+2

+14

+21

Économies...

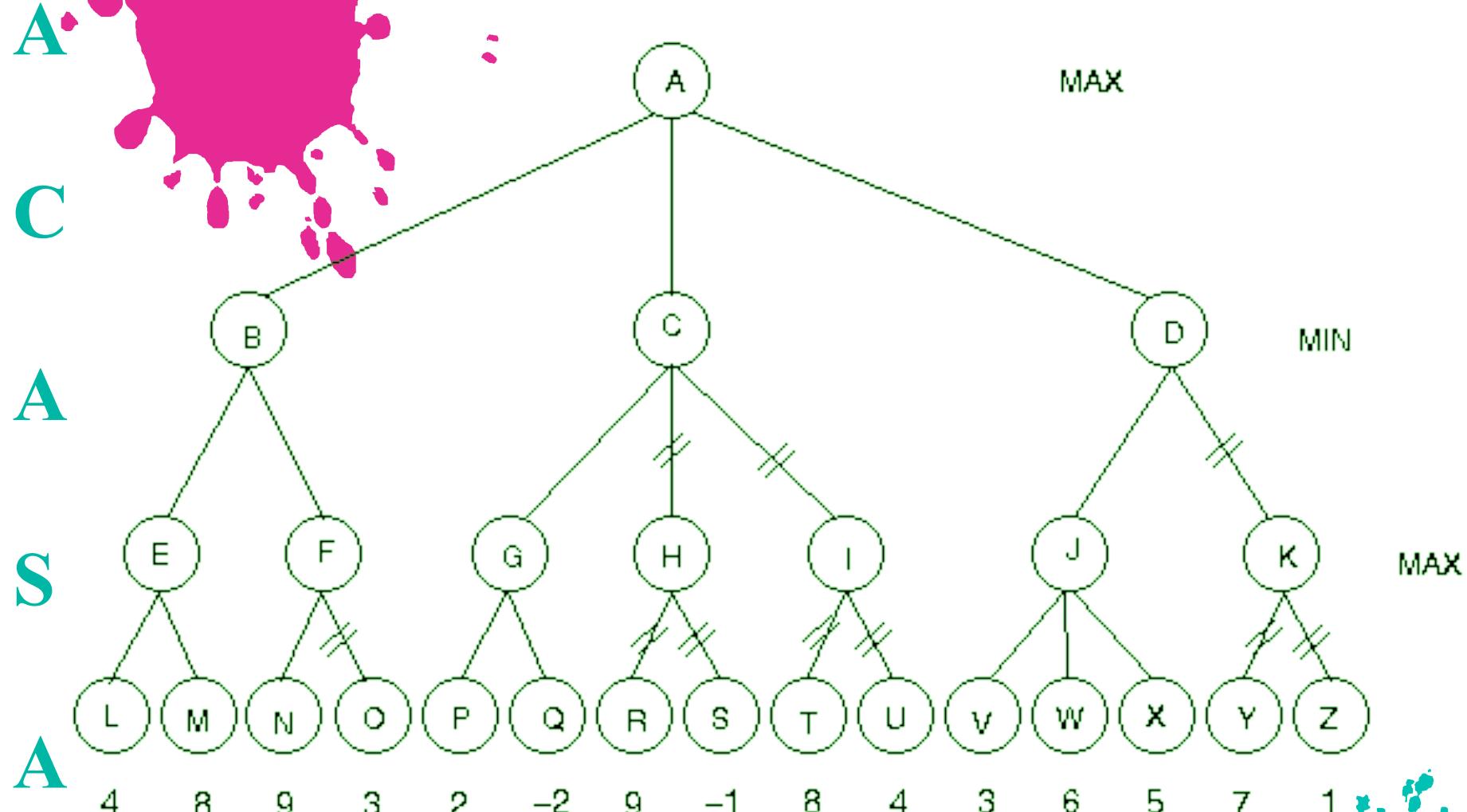
IA & Jeux



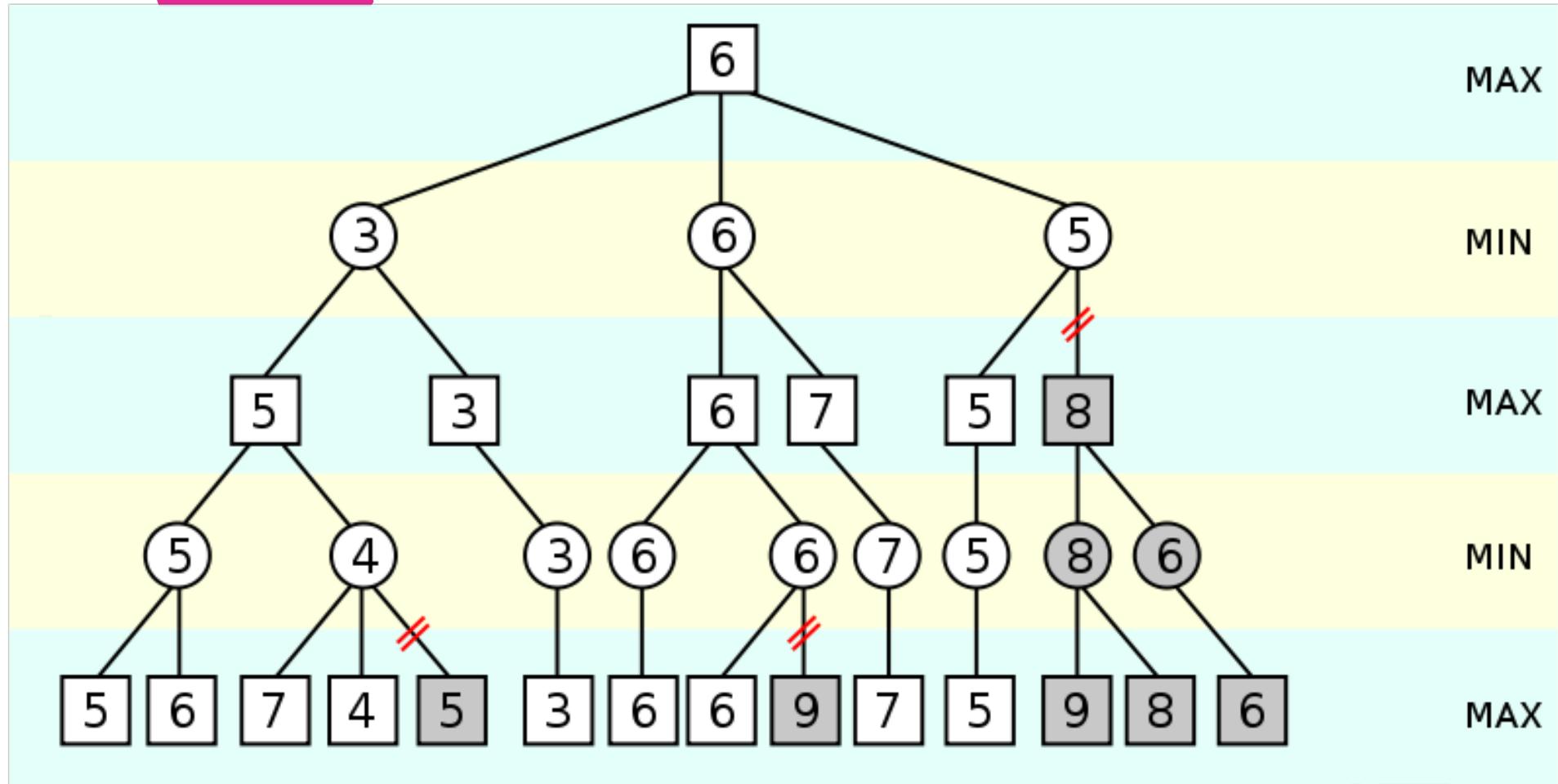
Jean-Gabriel Ganascia



Autre exemple



Alpha-beta coupe sur l'arbre min-max



LIP6

Jeu d'échec

A

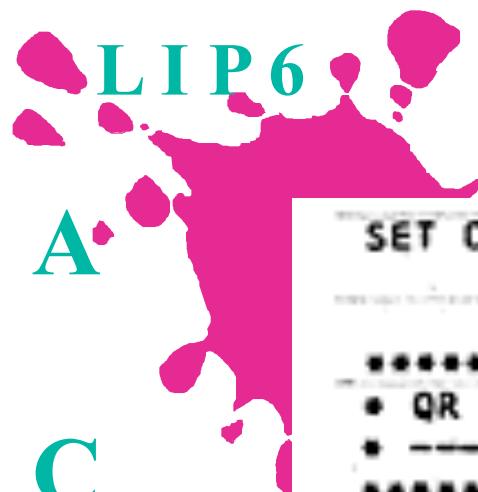
- ◆ La drosophile de l'IA...

C

- ◆ Quelques dates

- ◆ 1912 Torres y Quevedo
- ◆ 1948 Wiener – description d'une machine qui joue aux échecs
- ◆ 1950 C. Shannon – article sur programme d'échec
- ◆ 1952 A. Turing écrit le premier programme d'échec
- ◆ 1956 Los Alamos Chess – premier programme à jouer aux échecs
- ◆ **1956 J. McCarthy invente la recherche alpha-beta**
- ◆ 1957 A. Bernstein – premier programme qui joue une partie
- ◆ **1962 Kotok – McCarthy – le premier programme qui joue de façon crédible (publié au MIT)**





A

C

A

S

A

Jean-Gabrie

Programme Kotok – McCarthy

SET OF TABLES NUMBER 4 MOVE IS *QP - Q4
BLACK

* QR * QN * QB * Q * K * KB * KN * KR *
* --- * --- * --- * --- * --- * --- * --- *

* QRP* QNP* QBP* * KP * KBP* KNP* KRP*
* --- * --- * --- * --- * --- * --- * --- *

* * * * * * * *
* * * * * * * *

* * * * QP * * *
* * * * --- * * *

* * * * QP * * *
* * * * --- * * *

* * * * * * * *
* * * * * * * *

* QRP* QNP* QBP* * KP * KBP* KNP* KRP*
* * * * * * * *

* QR * QN * QB * Q * K * KB * KN * KR *
* * * * * * * *

WHITE

X



LIP6

Jeu d'échec

A

- ◆ La drosophile de l'IA...

C

- ◆ Quelques dates

A

- ◆ 1912 Torres y Quevedo
- ◆ 1948 Wiener – description d'une machine qui joue aux échecs
- ◆ 1950 C. Shannon – article sur programme d'échec
- ◆ 1952 A. Turing écrit le premier programme d'échec
- ◆ 1956 Los Alamos Chess – premier programme à jouer aux échecs
- ◆ **1956 J. McCarthy invente la recherche alpha-beta**
- ◆ 1957 A. Bernstein – premier programme qui joue une partie
- ◆ **1962 Kotok – McCarthy – le premier programme qui joue de façon crédible (publié au MIT)**
- ◆ **1968 D. Levy – grand maître aux échecs – parie avec McCarthy qu'aucun ordinateur ne le battra avant 10 ans (*montant 6 mois de salaire*)**
- ◆ **1978 D. Levy gagne son pari en gagnant Chess 4.7**

S

A

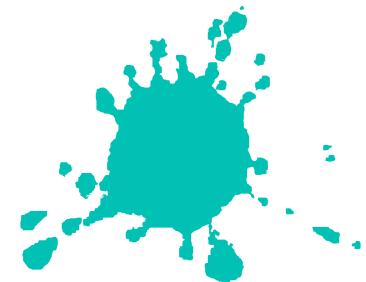
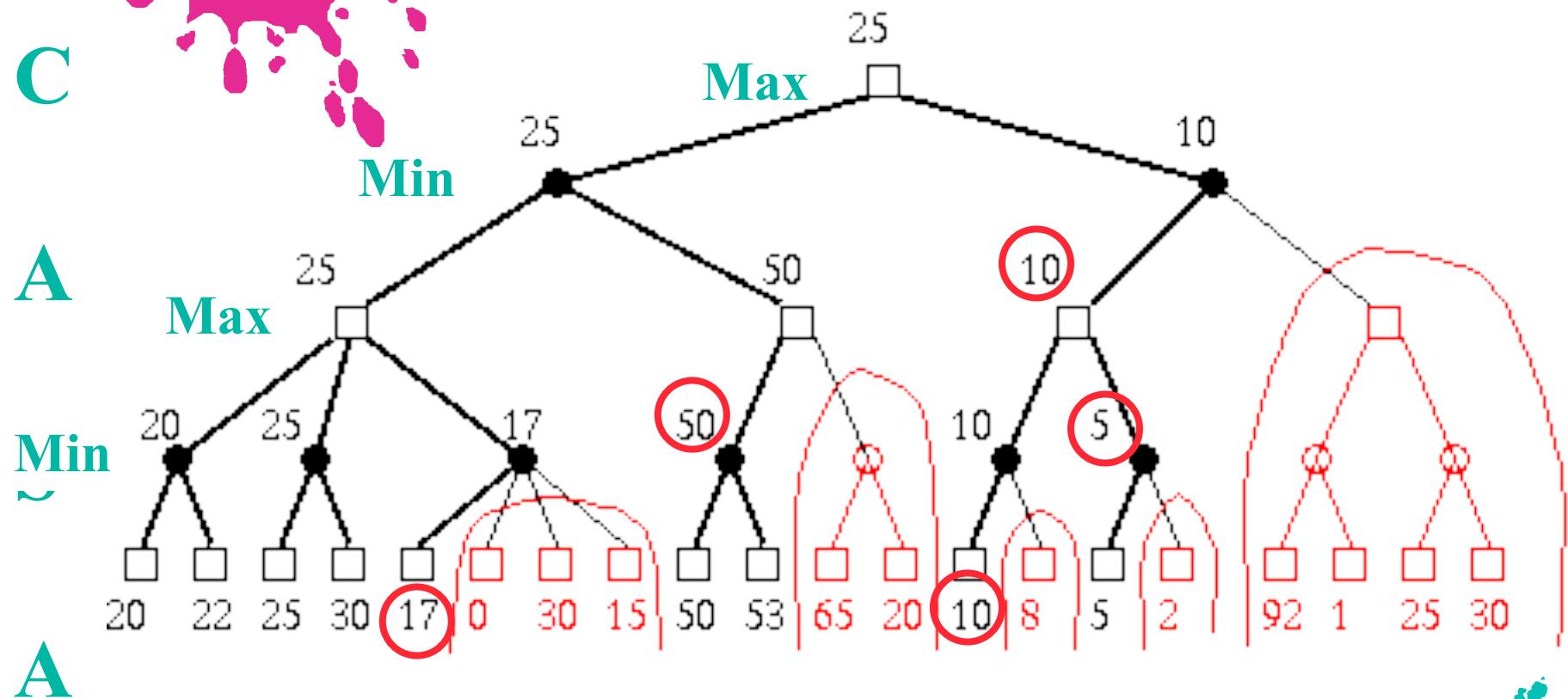
fin cours 1





A
C

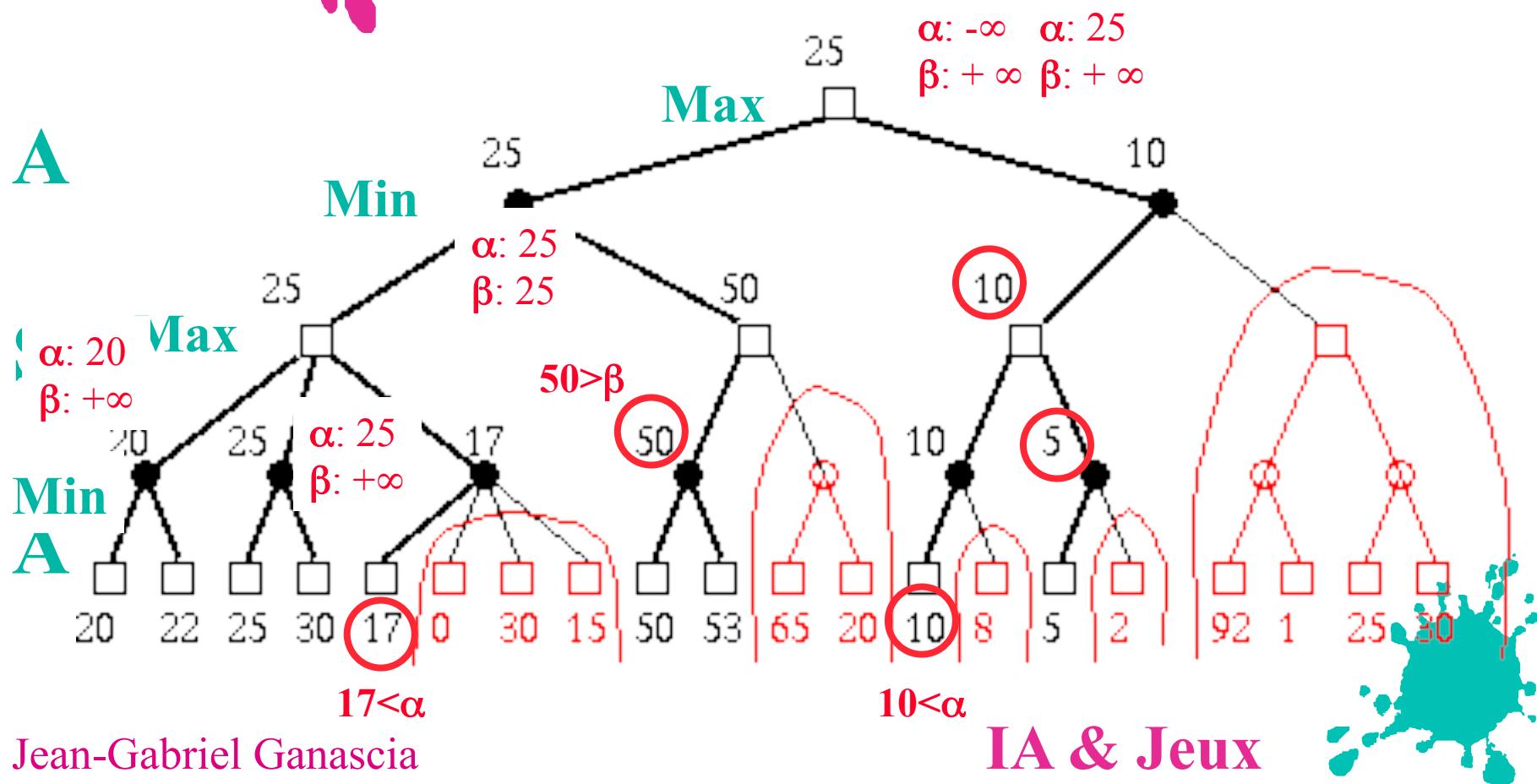
Algorithme Alpha - Beta





Algorithme Alpha - Beta

- α : borne inférieure des maximum
- β : borne supérieure des minimum





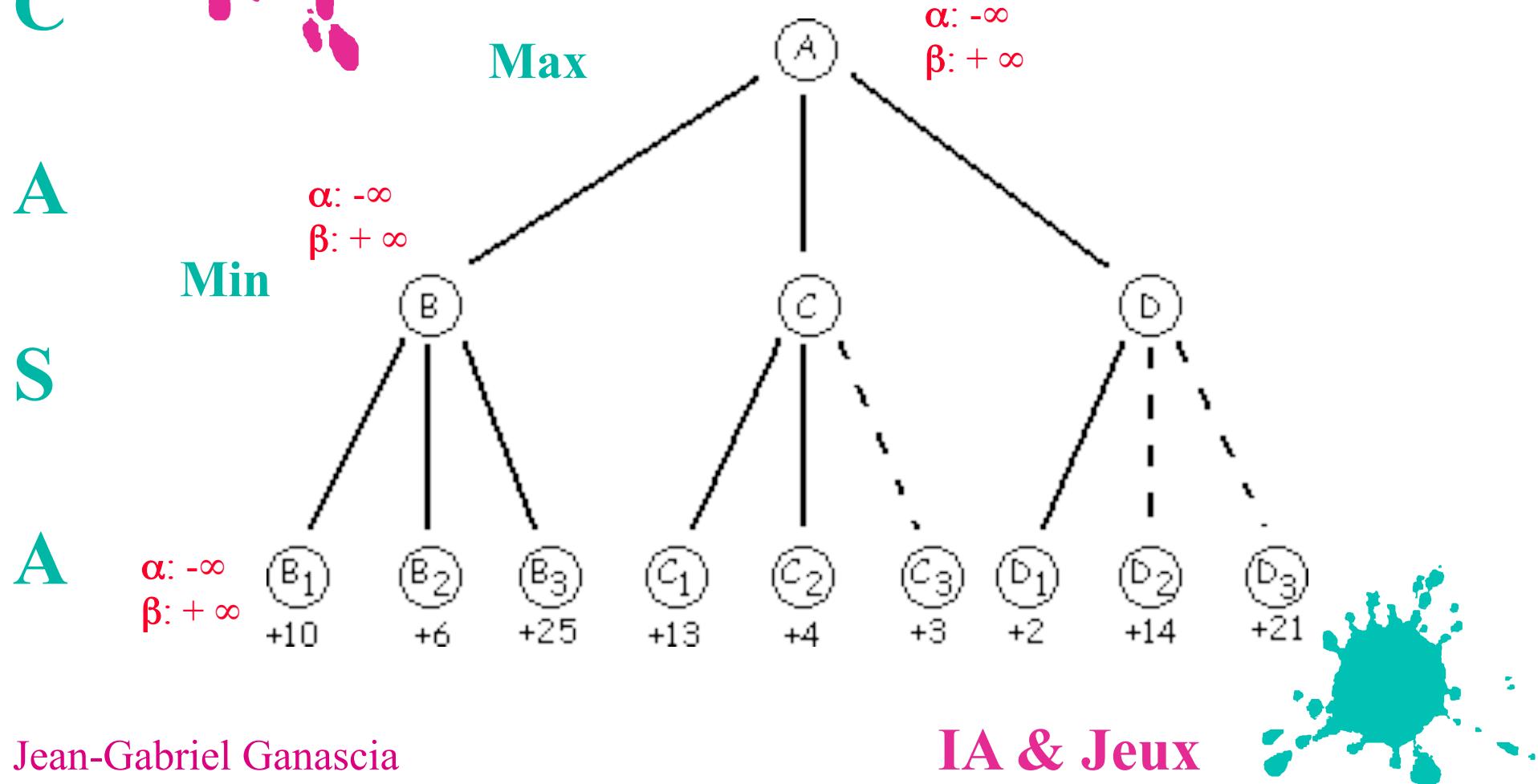
Algorithme alpha-beta

```
Valeur-noeud(n, alpha, beta)
    si n est terminal alors retourner eval(n)
        sinon soient f1, ..., fj les fils de n
            si n est max alors
                pour i de 1 à j
                    et tant que alpha < beta faire
                        alpha <- max(alpha,Valeur-noeud(fi, alpha, beta))
            retourner alpha
            sinon pour i de 1 à j
                et tant que alpha < beta faire
                    beta <- min(beta,Valeur-noeud(fi, alpha, beta))
            retourner beta
            fsi
        fsi
```



Simulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum

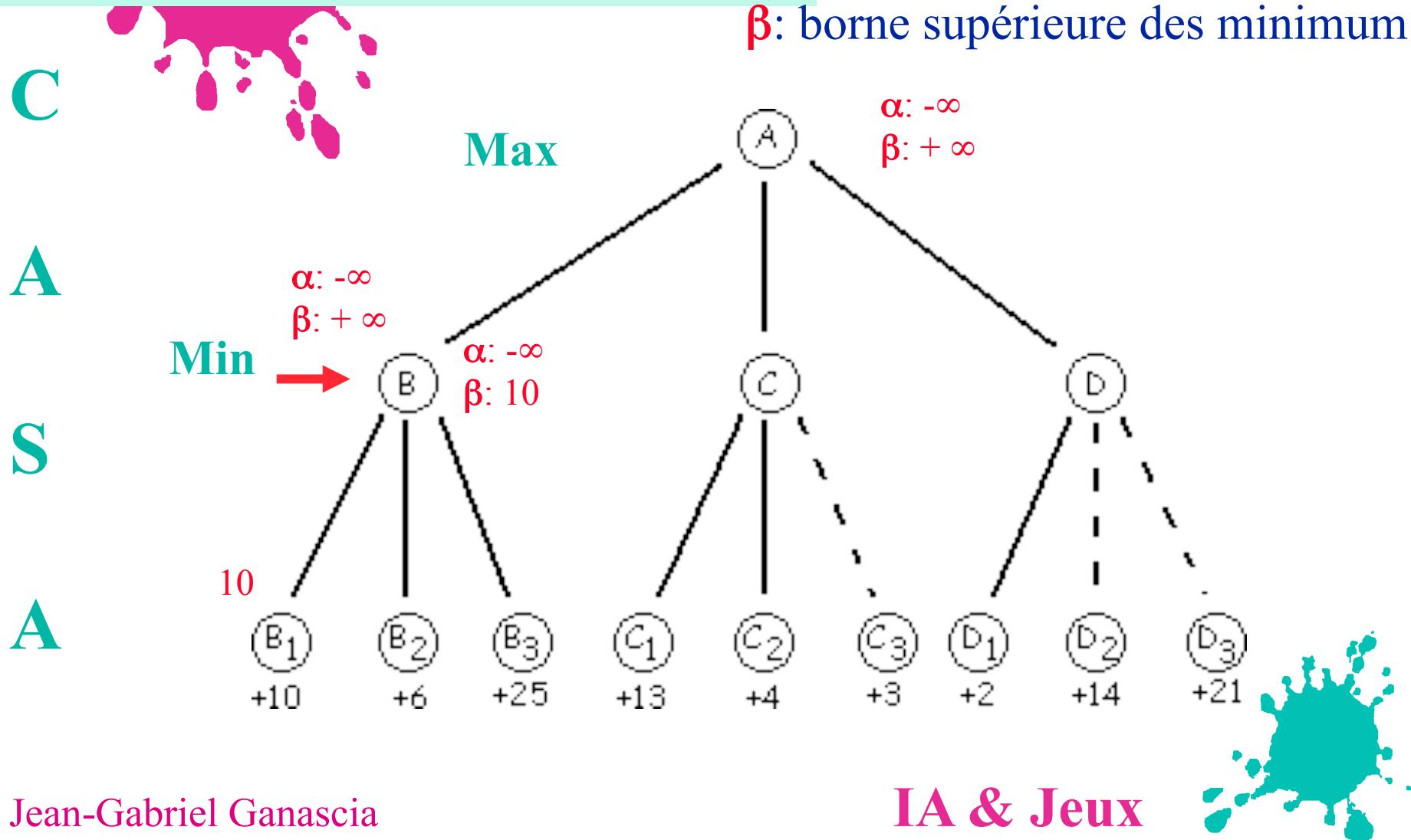


noeud min

```
tant que alpha < beta faire  
beta <- min(beta,Valeur-  
noeud(fi, alpha, beta))  
Retourner beta
```

ulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum



noeud min

```
tant que alpha < beta faire  
beta <- min(beta,Valeur-  
noeud(fi, alpha, beta))  
Retourner beta
```

ulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum

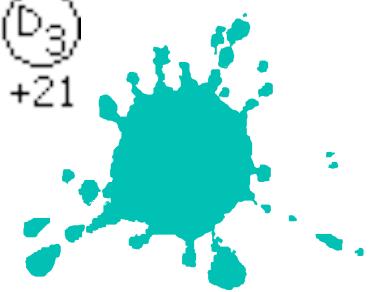
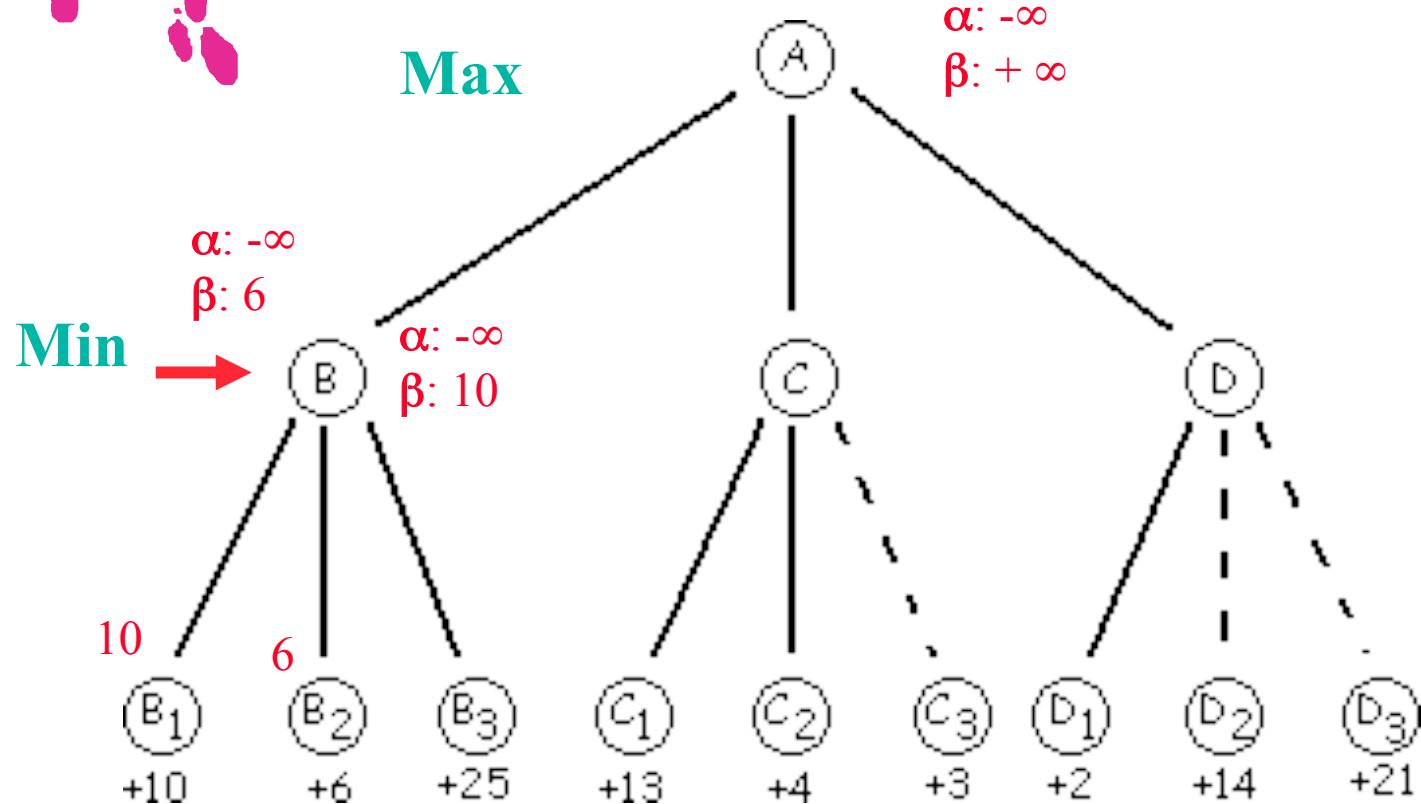


C

A

S

A

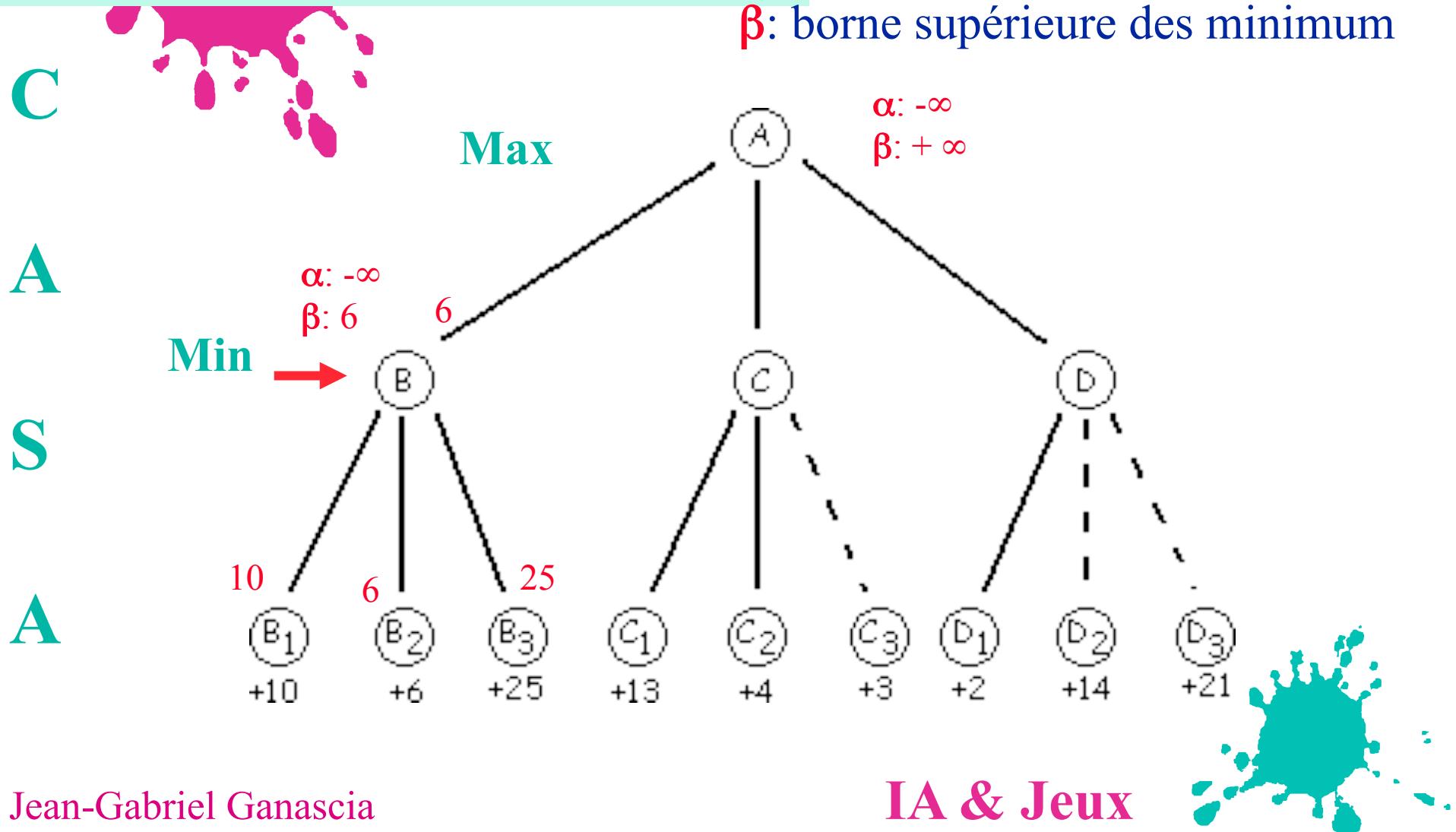


noeud min

```
tant que alpha < beta faire  
beta <- min(beta,Valeur-  
noeud(fi, alpha, beta))  
Retourner beta
```

ulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum



noeud max

tant que alpha < beta faire

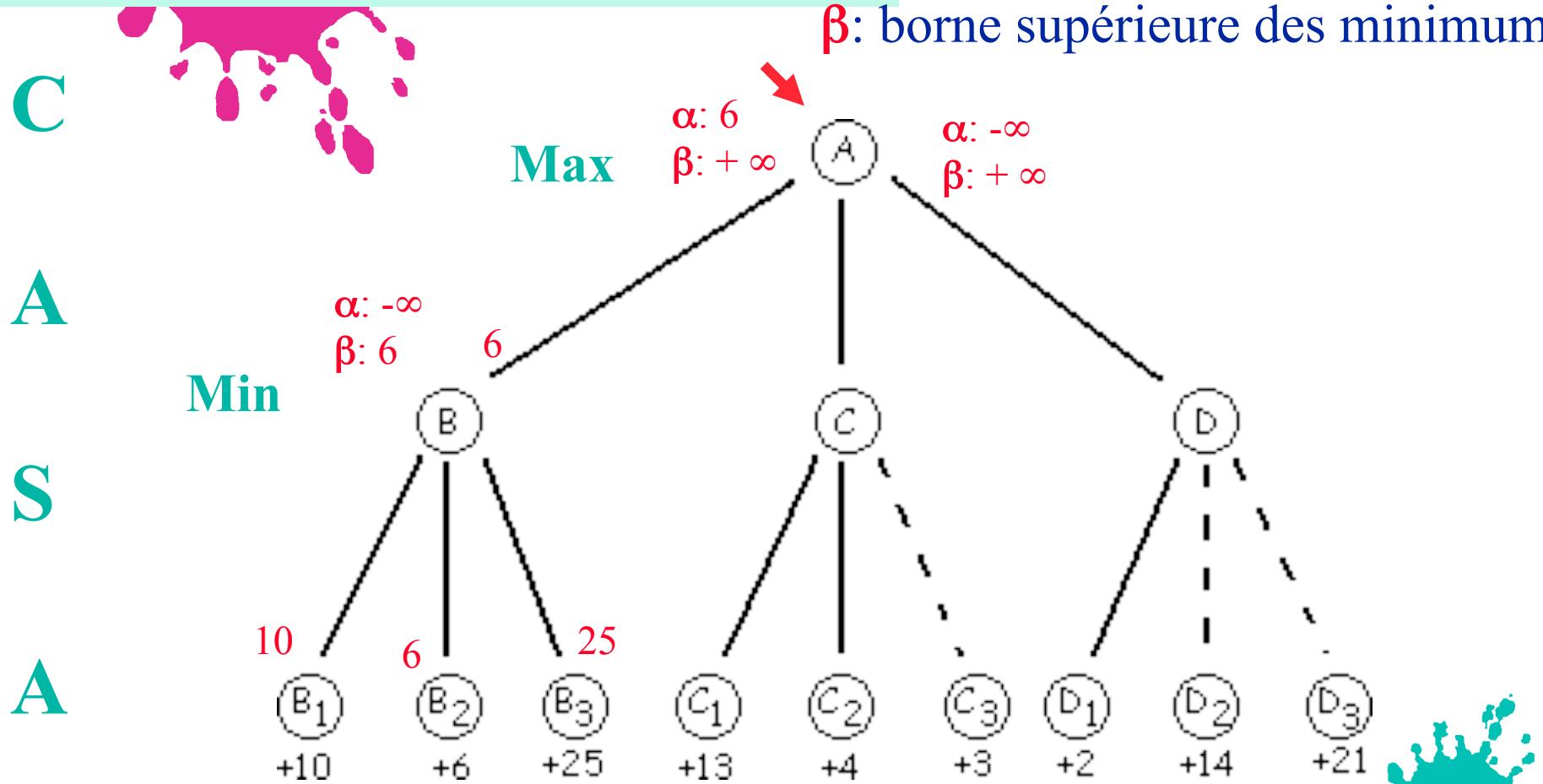
alpha <- max(alpha,

valeur-noeud(fi, alpha, beta))

Évaluation alpha-beta

α : borne inférieure des maximum

β : borne supérieure des minimum

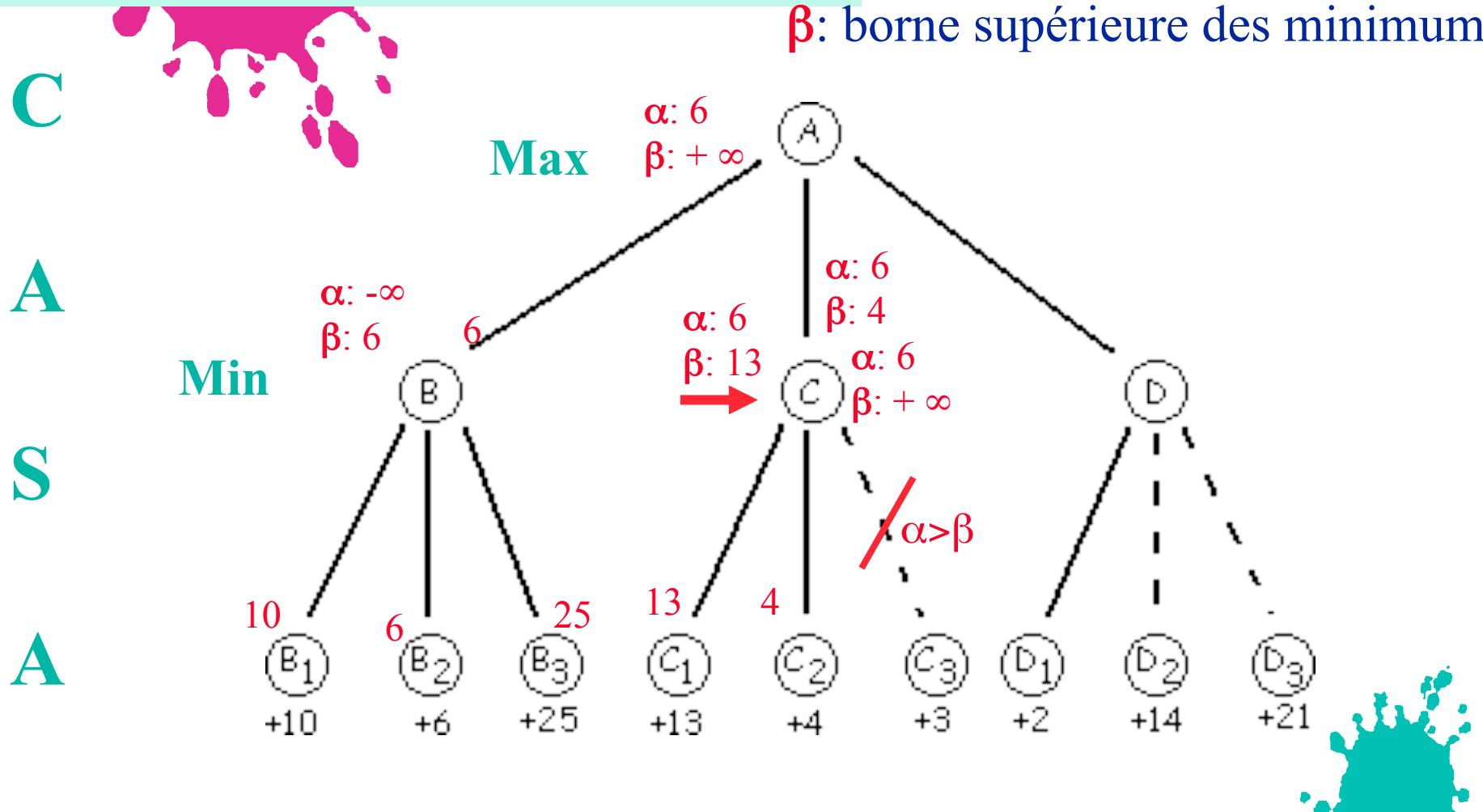


noeud min

```
tant que alpha < beta faire  
beta <- min(beta,Valeur-  
noeud(fi, alpha, beta))  
Retourner beta
```

ulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum

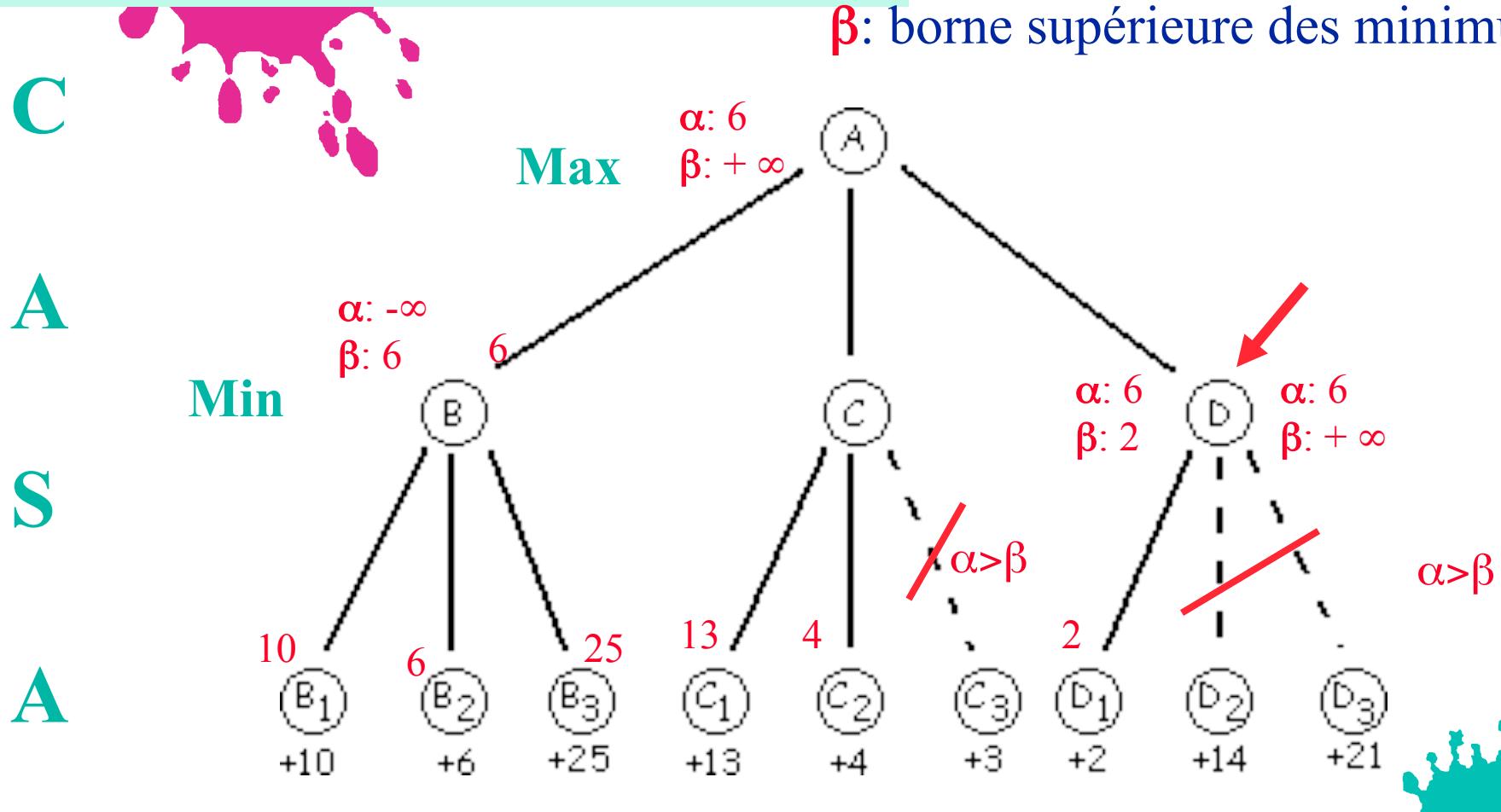


noeud min

```
tant que alpha < beta faire  
beta <- min(beta,Valeur-  
noeud(fi, alpha, beta))  
Retourner beta
```

ulation alpha-beta

α : borne inférieure des maximum
 β : borne supérieure des minimum





Programmation alpha-beta

```
min_max(_, J, _, +1, _, A, B, A, B) :- joueur_max(Max),  
    etat_gagnant(J, Max), !.  
  
min_max(_, J, _, -1, _, A, B, A, B) :- joueur_min(Min),  
    etat_gagnant(J, Min), !.  
  
{ min_max(_, J, _, 0, _, A, B, A, B) :- etat_nul(J), !.  
min_max(j_max, Jcourant, Jsuiv, N, Prof, Alpha, Beta,  
    Alpha, BetaN) :- joueur_max(M),  
    transitions(Jcourant, M, Ls),  
    min_max_min(Ls, RR, Prof, Alpha, Beta),  
    choix_max(RR, Jsuiv, N),  
    min(Beta, N, BetaN).  
  
min_max(j_min, Jcourant, Jsuiv, N, Prof, Alpha, Beta,  
    AlphaN, Beta) :- joueur_min(M),  
    transitions(Jcourant, M, Ls),  
    min_max_max(Ls, RR, Prof, Alpha, Beta),  
    choix_min(RR, Jsuiv, N),  
    max(Alpha, N, AlphaN).
```



Alpha-beta (suite)

```
min_max_min([], [], _, _, _).
min_max_min([J|L], [[J, Ns]|LL], P, Alpha, Beta) :-
    PP is P + 1,
    min_max(j_min, J, _, Ns, PP,
             Alpha, Beta, AlphaN, BetaN),
    si(   Ns >= AlphaN,
          min_max_min(L, LL, P, AlphaN, BetaN),
          min_max_min([], LL, P, AlphaN, BetaN)).
```





Alpha-beta (suite)

```
min_max_max([], [], _, _, _).
min_max_max([J|L], [[J, Ns]|LL], P, Alpha, Beta) :-
    PP is P + 1,
    min_max(j_max, J, _, Ns, PP,
            Alpha, Beta, AlphaN, BetaN),
    si(   Ns <= BetaN,
          min_max_max(L, LL, P, AlphaN, BetaN),
          min_max_max([], LL, P, AlphaN, BetaN)).
```





A

Alpha – beta (fin)

C

```
choix_max([[_ , R] | L], Js, Rs) :  
    choix_max(L, Js, Rs),  
    Rs > R, !.
```

A

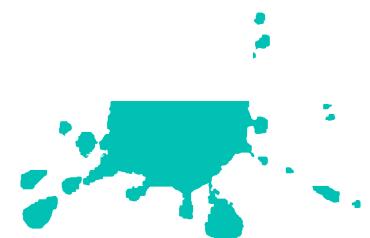
```
choix_max([[J, R] | _], J, R).
```

S

```
choix_min([[_ , R] | L], Js, Rs) :  
    choix_min(L, Js, Rs),  
    Rs < R, !.
```

A

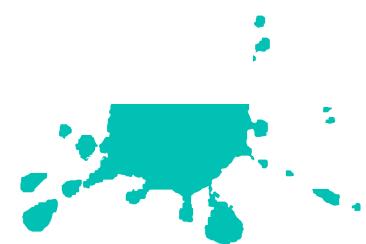
```
choix_min([[J, R] | _], J, R).
```





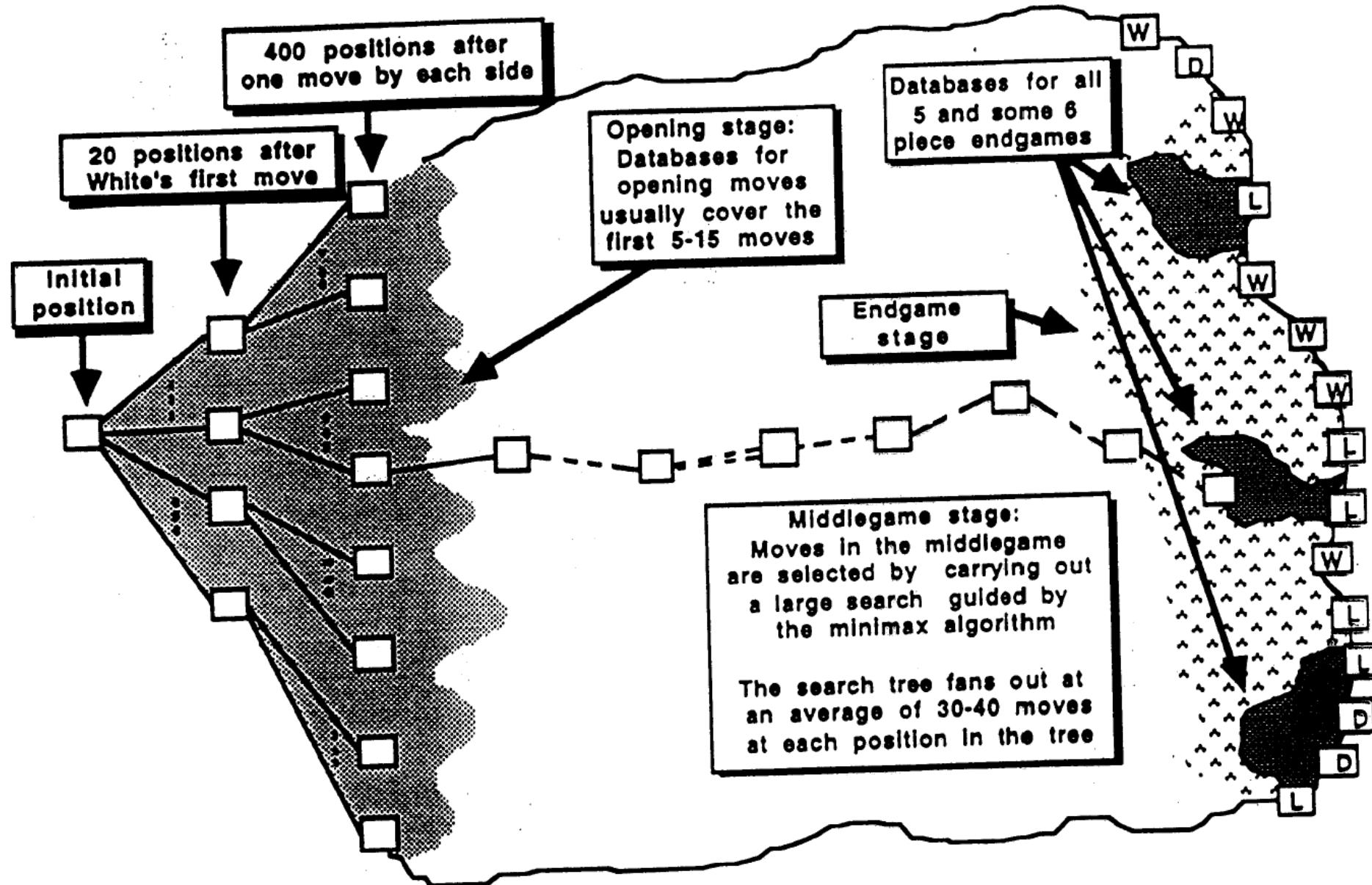
Jeux d' échec

- C ◆ Débuts de parties: catalogues d' ouvertures
- A ◆ Milieu de partie: stratégies, occupation de
l' échiquier...
- S ◆ Fins de parties: problèmes d' échec
- A ◆ Processeurs parallèles très puissants
- ◆ Fonction d' évaluation variable selon la profondeur: largeur sur deux demi-coup,
profondeur, ensuite largeur





Phases de jeux



LIP 6

A C Questions A ouvertes



- ◆ Pourquoi la force brute l' emporte ?
- ◆ Notion d' effet d' horizon
- ◆ L' échec actuel de l' intelligence artificielle:
 - ◆ Étude sur les joueurs d' échec
 - ◆ Des systèmes qui apprennent

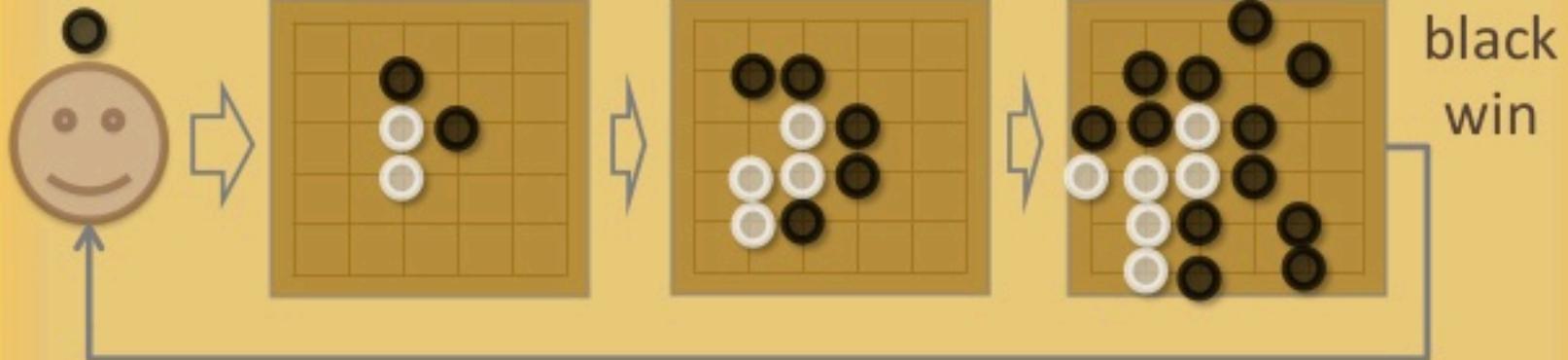
Jean-C

AlphaGo: Deep Q-learning: $S, A = \text{vecteur} + \text{NN}$

A

Reinforcement Learning

C



A

S

A

Jean

Reward (Feedback)

black
win

white
win

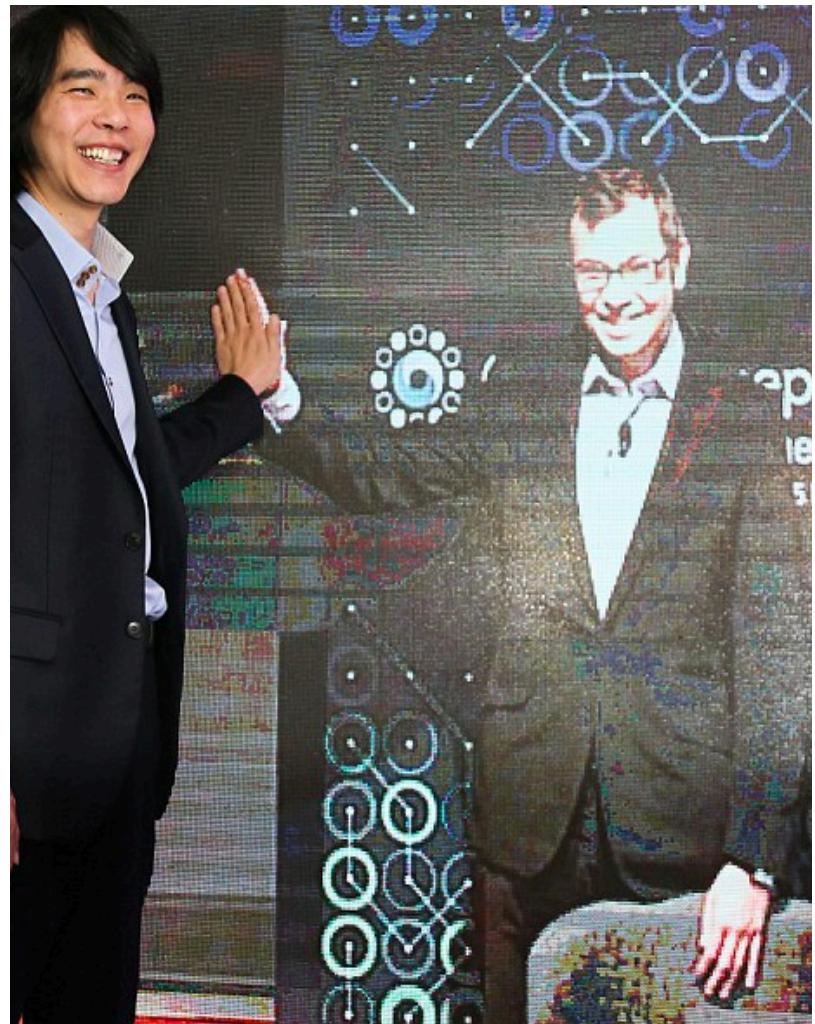
Reward (Feedback)



- C ◆ Couplage Deep-Learning
- ◆ Q learning
- A ◆ $Q(S, A)$:
- ◆ Utilisation de vecteurs
 - ◆ Utilisation de 4 réseaux neuronaux
 - ◆ Stratégie: plan pour le jeu
 - ◆ Prédiction adverse
 - ◆ Politique d'apprentissage
 - ◆ Prédiction de la valeur espérée (valeur)

Jean-Gabriel Ganascia

AlphaGo



IA & Jeux

