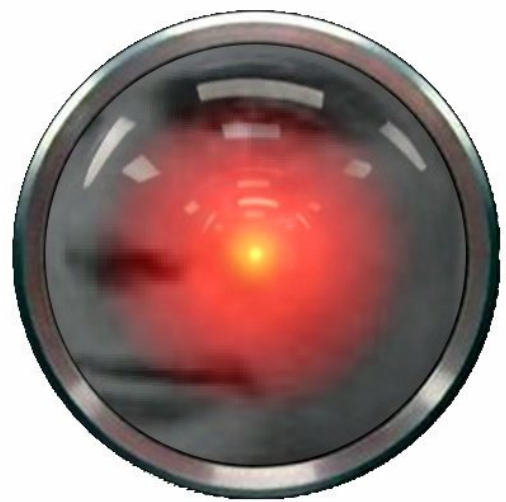
Intelligence Artificielle

Jeux Avancés



Laurent Simon

mél : simon@lri.fr

IM : lsimon@jabber.fr
Tél : 01 69 15 64 95

LRI - CNRS
Université Orsay Paris 11,
91405 Orsay Cedex

Contenu

1. **Algorithmique Avancées**
   - Écritures succinctes
   - Recherche Alpha-Béta sur des fenêtres réduites
   - Exploration en largeur
2. **Architectures performantes**
   - Considération de temps réel
   - Début, Milieu et Fin de partie
3. **Heuristiques Avancées**
   - Affiner la valeur de l'heuristique
4. **Les systèmes qui fonctionnent**
   - Échecs et Dames
   - Le futur des jeux

Contenu

1. **Algorithmique Avancées**
   - Écritures succinctes
   - Recherche Alpha-Béta sur des fenêtres réduites
   - Exploration en largeur
2. Architectures performantes
   - Considération de temps réel
   - Début, Milieu et Fin de partie
3. Heuristiques Avancées
   - Affiner la valeur de l'heuristique
4. Les systèmes qui fonctionnent
   - Échecs et Dames
   - Le futur des jeux

MeaCulpa Algorithme $\alpha\beta$

Une amélioration :un $\alpha\beta$ correct

```
fonction Max-value(état, alpha, beta)

 si etat-feuille(etat) alors  return(evalue(etat))
 pour chaque successeur s de état
    alpha := max(alpha,Min-value(s,alpha,beta))
    si alpha >= beta alors return(alpha)    % coupe beta
 return(alpha)
```

```
fonction Min-value(état, alpha, beta)

 si etat-feuille(etat) alors return(evalue(etat))
 pour chaque successeur s de état
    beta := min(beta,Max-value(s,alpha,beta))
    si beta <= alpha alors return(beta)   % coupe alpha
 return(beta)
```

Négamax

Une réécriture simple de MiniMax

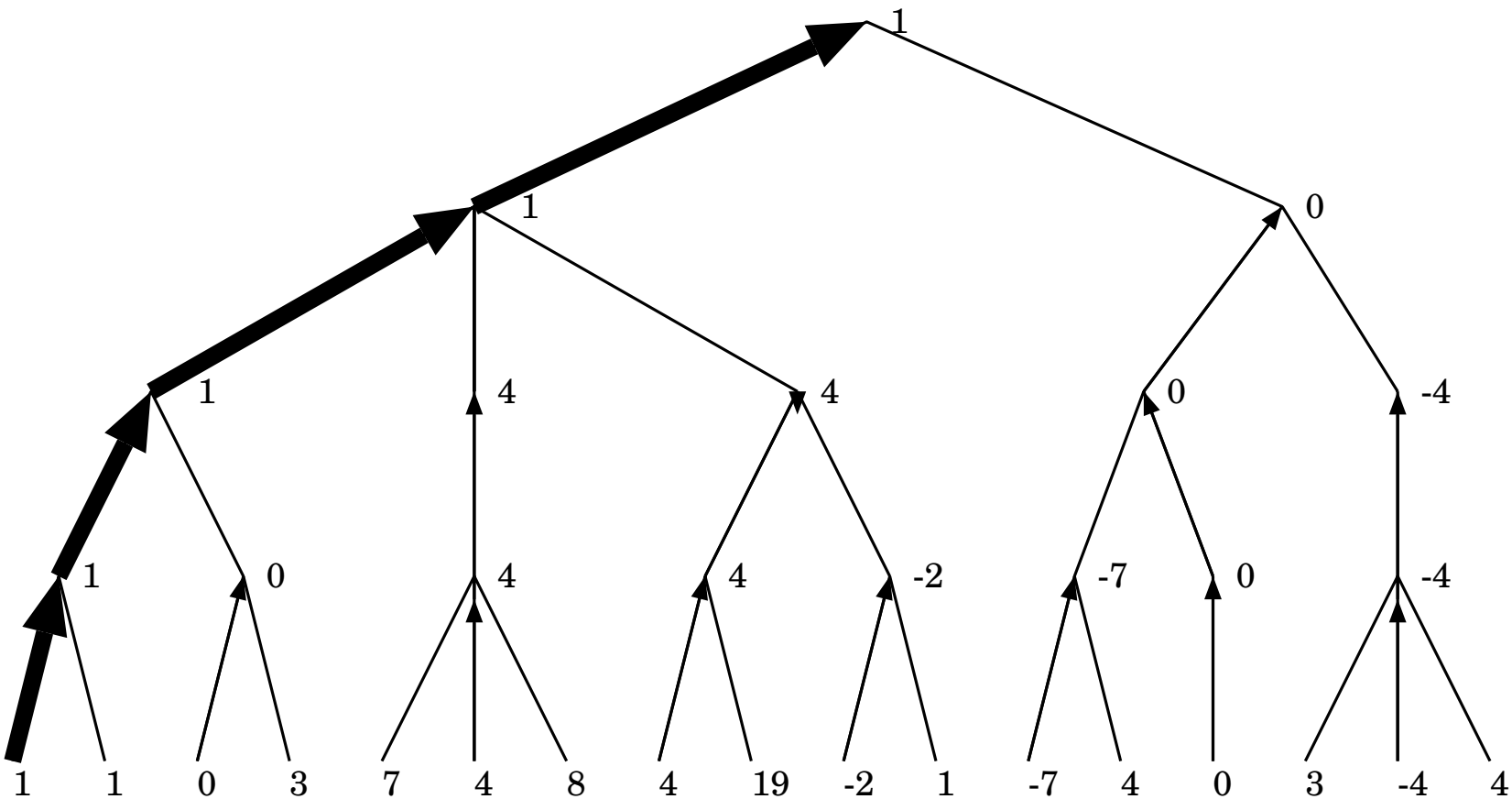
Idée

Au lieu d'alterner Max/Min, on se restreint à la fonction Max, en utilisant l'opposé du résultat à chaque niveau.

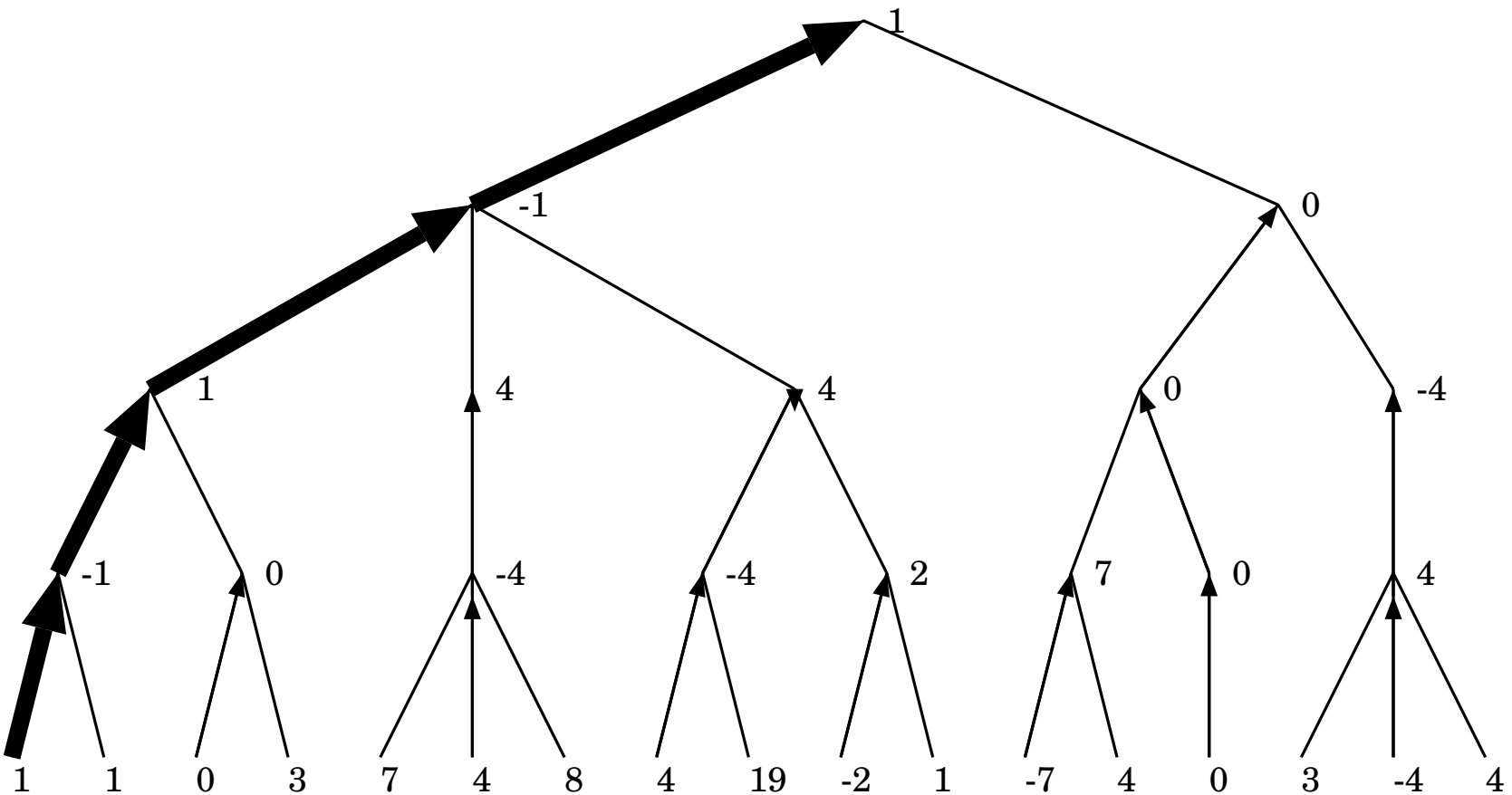
Algorithme NégaMax

```
Fonction NégaMax(Plateau P)
Debut
 Si P est une feuille
    Retourner EVALUE(P)
 Sinon
    Retourner Max(-MiniMax(Succ_1(P)),
                  -MiniMax(Succ_2(P)), ...
                  -MiniMax(Succ_N(P)))
Fin
```

Arbre de développement Minimax



Arbre de développement Négamax



$\alpha\beta$ version Négamax

Une réécriture simple de $\alpha\beta$

Algorithme Nég$\alpha\beta$

```
Fonction NegAB(Plateau P, alpha, beta)

 si terminal(P) alors  return(evalue(P))
 pour chaque successeur s de P
    alpha := max(alpha, -NegAB(s, -beta, -alpha))
    si alpha >= beta alors retourner(alpha)
 retourner(alpha)
```

$\alpha\beta$ avec information de coupe

Une réécriture plus informative de $\alpha\beta$

Idée

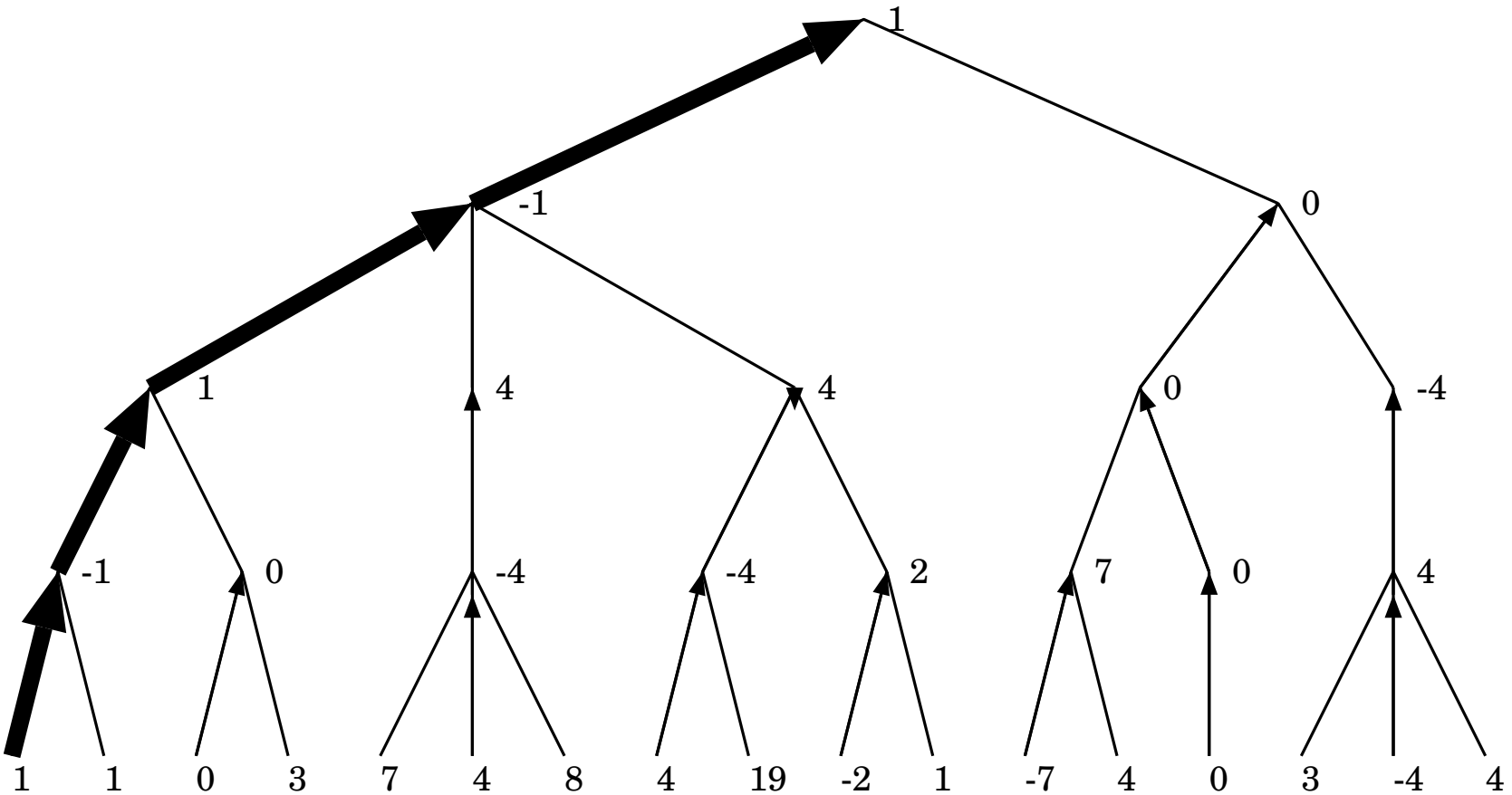
Remonter et exploiter les raisons de l'échec de $\alpha\beta$.

Algorithme NégaMax-Echec-$\alpha\beta$

```
Fonction NegABEchec(Plateau P, alpha, beta)
Debut
 Si P est une feuille Retourner EVALUE(P)
 Sinon
    Meilleur = -inf
    pour chaque successeur s de P
      Meilleur = max(Meilleur, -NegaABEchec(s, -beta, -a
      alpha = max(Meilleur, alpha)
      Si alpha >= beta alors retourner(Meilleur)
    Retourner(Meilleur)
Fin
```

$\alpha\beta$ avec information de coupe

Sur un exemple



Résultats sur $\alpha\beta$

Définition

On appelle fenêtre $\alpha\beta$ le couple $[\alpha, \beta]$ où α et β sont les deux paramètres de la procédure $\alpha\beta$.

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

Propriété de la taille des fenêtres

Soient deux fenêtres $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ telles que $\alpha_1 \leq \alpha_2 < \beta_1 \leq \beta_2$, l'algorithme $\alpha\beta$ appelé avec $[\alpha_2, \beta_2]$ explore moins de noeuds que celui appelé avec la fenêtre $[\alpha_1, \beta_1]$.

Résultats sur $\alpha\beta$

Définition

On appelle fenêtre $\alpha\beta$ le couple $[\alpha, \beta]$ où α et β sont les deux paramètres de la procédure $\alpha\beta$.

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

Propriété de la taille des fenêtres

Soient deux fenêtres $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ telles que $\alpha_1 \leq \alpha_2 < \beta_1 \leq \beta_2$, l'algorithme $\alpha\beta$ appelé avec $[\alpha_2, \beta_2]$ explore moins de noeuds que celui appelé avec la fenêtre $[\alpha_1, \beta_1]$.

Résultats sur $\alpha\beta$

Définition

On appelle fenêtre $\alpha\beta$ le couple $[\alpha, \beta]$ où α et β sont les deux paramètres de la procédure $\alpha\beta$.

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

Propriété de la taille des fenêtres

Soient deux fenêtres $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ telles que $\alpha_1 \leq \alpha_2 < \beta_1 \leq \beta_2$, l'algorithme $\alpha\beta$ appelé avec $[\alpha_2, \beta_2]$ explore moins de noeuds que celui appelé avec la fenêtre $[\alpha_1, \beta_1]$.

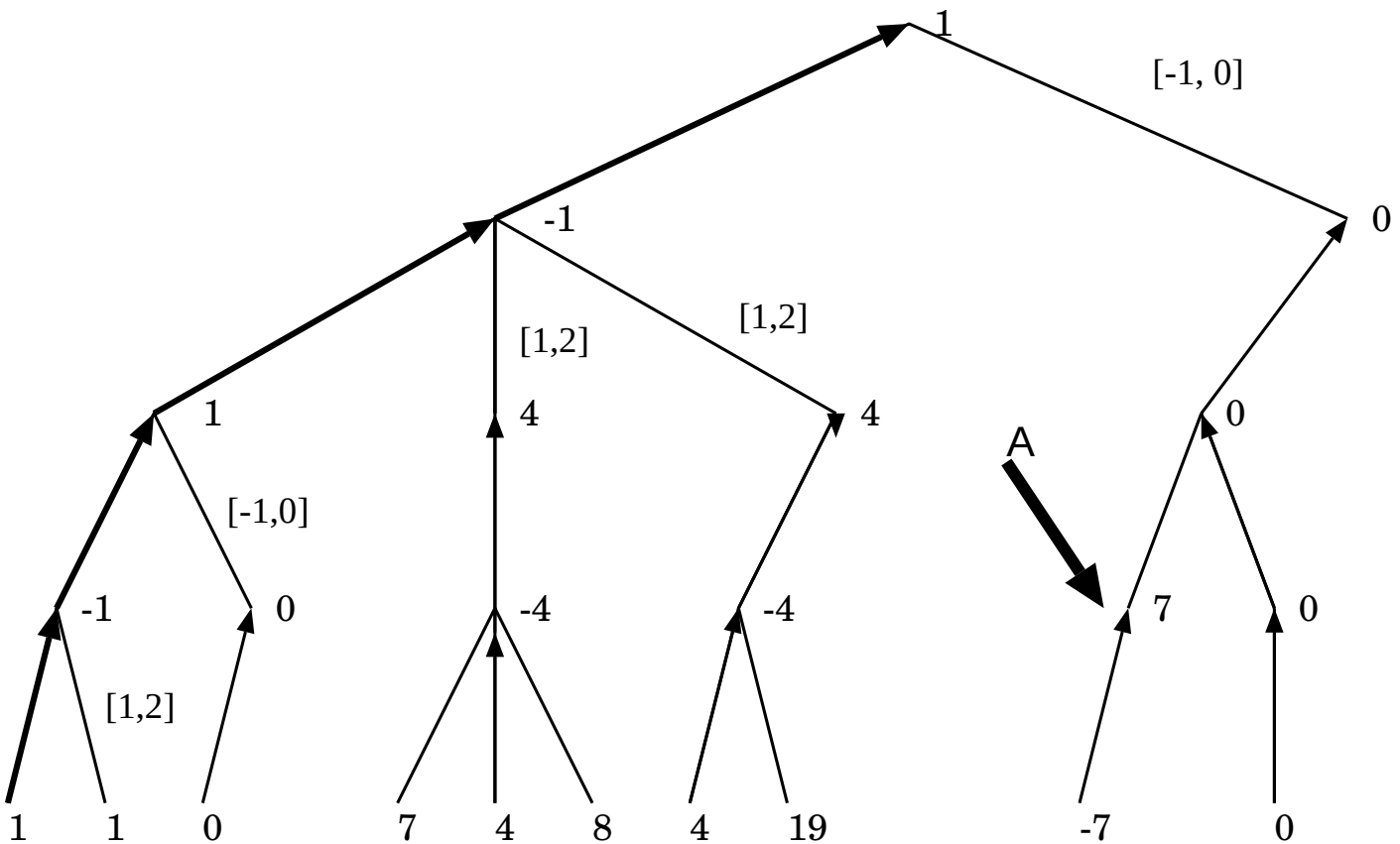
Utiliser les fenêtres et NégaABEchec

Idée : Seulement vérifier, à chaque niveau, que les meilleurs coups sont le plus à gauche. Si ce n'est pas le cas, recalculer la nouvelle meilleure valeur.

Algorithme P-$\alpha\beta$

```
Fonction PAB(Plateau P)
Debut
 Si P est une feuille Retourner EVALUE(P)
 Meilleur = -PAB(premierSucc de P)
 pour tous les autres successeurs s de P
      val = -NégaABEchec(s, -Meilleur, -(Meilleur-1))
      Si val > Meilleur alors
         Meilleur = -NégaABEchec(s, -inf, -val)
  Retourner(Meilleur)
Fin
```

P-$\alpha\beta$
Sur un exemple



Scout

Qui a dit toujours ?

Idée : Ne pas limiter à la branche la plus à gauche.

Algorithme Scout

```
Fonction Scout(Plateau P)
Debut
 Si P est une feuille Retourner EVALUE(P)
 Meilleur = -Scout(premierSucc de P)
 pour tous les autres successeurs s de P
      Si non TEST(s, -Meilleur-1)
         Meilleur = -Scout(s)
  Retourner(Meilleur)
Fin
```

TEST(v) Vérifie si la valeur minimax d'un noeud est supérieur à v.

Variantes de Scout

D'autres procédure utilisent NégaABEchec à la place de TEST.
Une version Négamax de Scout a aussi été proposée.
Mais l'idée reste la même : On utilise la propriété des fenêtres minimales.
Le comportement asymptotique de Scout est identique à $\alpha\beta$.
Dans certains jeux, il peut lui être supérieur (jusqu'à 40% pour l'Awélé). Semble adapté aux arbres profonds, avec facteur de branchement faible.

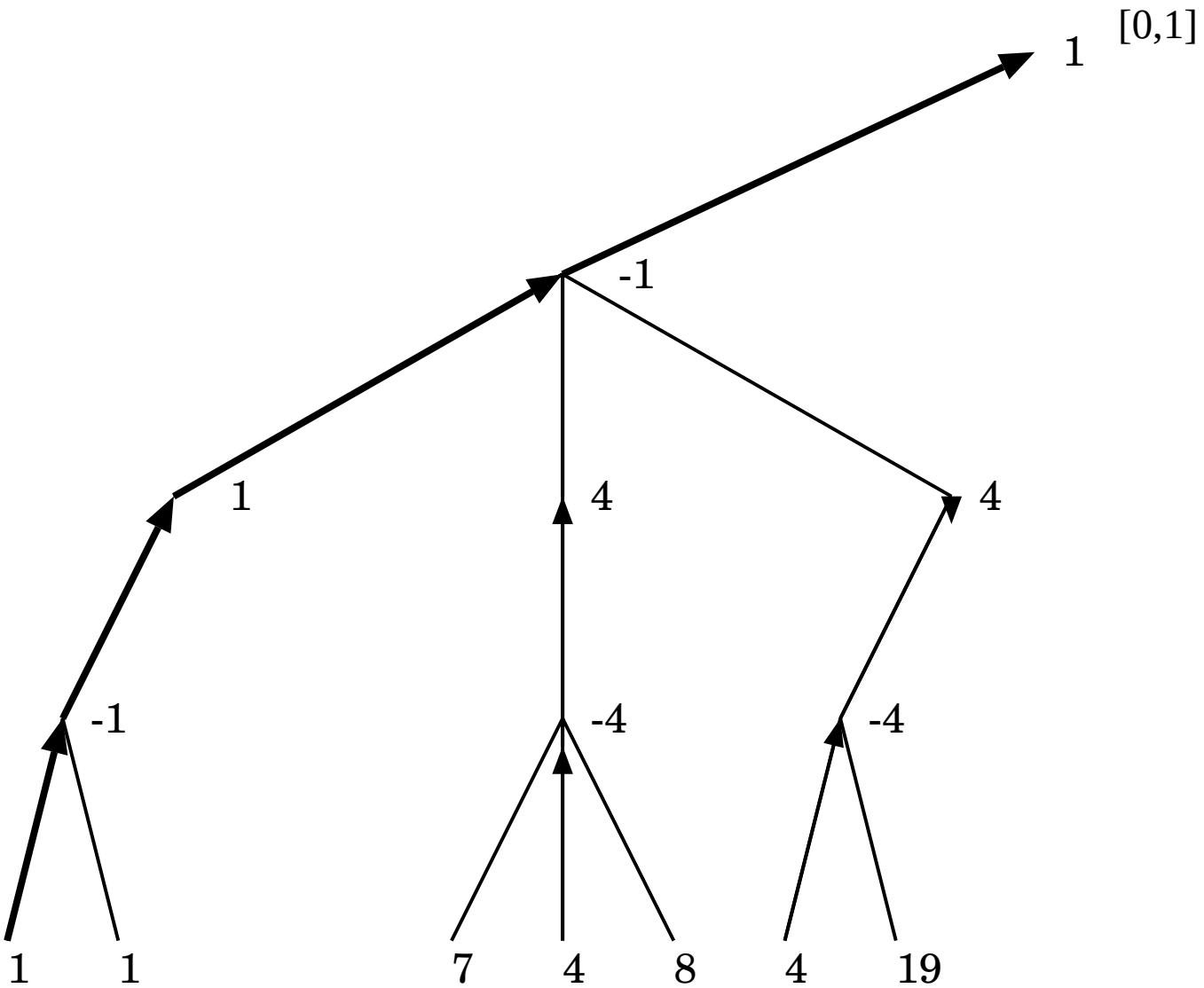
NegaC*

C'est notre dernier algorithme, qui pousse à l'extrème l'utilisation des fenêtres minimales.
Idée : Faire une recherche dichotomique de la valeur minimax de l'arbre en utilisant le test sur les fenêtres minimales.

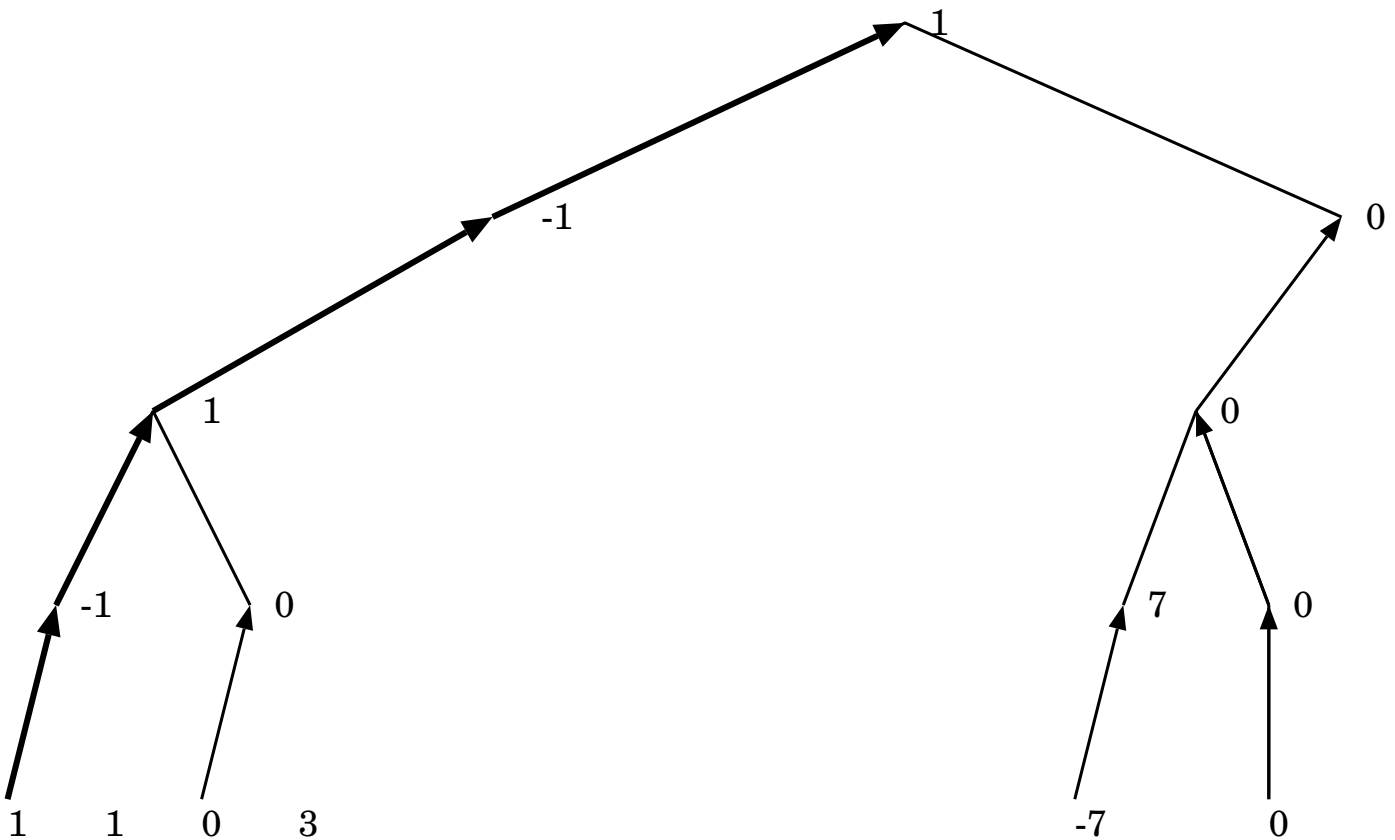
Algorithme NegaC*

```
Fonction NegaC*(Plateau P)
Debut
 Varinf = -inf
 varsup = +inf
 Tant que Varinf <> Varsup
   v = [(Varinf + Varsup) / 2]
   t = NégaABEchec(P, v, v+1)
   Si t > v alors
      Varinf = t
   Sinon
      Varsup = t
Fin
```

NegaC*
Sur un exemple



NegaC*
Sur un exemple



SSS*

Tous les algorithmes vus jusqu'à présent sont basés sur des recherches en profondeur. Peut-on faire un algo de recherche en largeur ? **Oui**, SSS* est un algorithme général de recherche dans les graphes ET/OU que l'on peut appliquer aux jeux.
L'idée est d'agrandir peu à peu des arbres ET/OU en choisissant de développer à chaque fois le plus prometteur. On manipule des Stratégies partielles (Arbre contenant la racine et dont chaque noeud Ami a au plus un fils).

SSS*

Tous les algorithmes vus jusqu'à présent sont basés sur des recherches en profondeur. Peut-on faire un algo de recherche en largeur ? **Oui**, SSS* est un algorithme général de recherche dans les graphes ET/OU que l'on peut appliquer aux jeux.

L'idée est d'agrandir peu à peu des arbres ET/OU en choisissant de développer à chaque fois le plus prometteur. On manipule des Stratégies partielles (Arbre contenant la racine et dont chaque noeud Ami a au plus un fils).

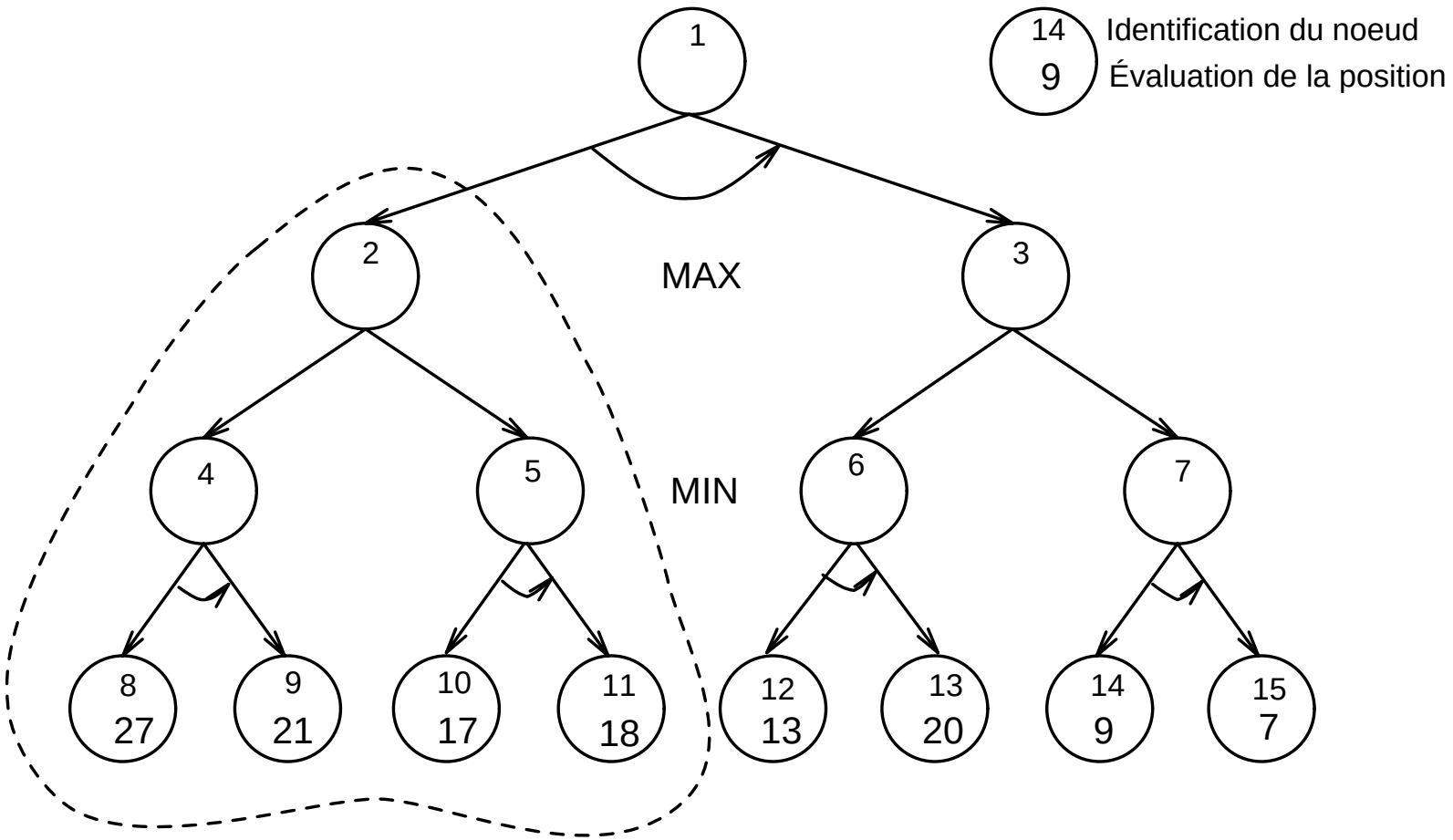
SSS*

Tous les algorithmes vus jusqu'à présent sont basés sur des recherches en profondeur. Peut-on faire un algo de recherche en largeur ? **Oui**, SSS* est un algorithme général de recherche dans les graphes ET/OU que l'on peut appliquer aux jeux.
L'idée est d'agrandir peu à peu des arbres ET/OU en choisissant de développer à chaque fois le plus prometteur. On manipule des Stratégies partielles (Arbre contenant la racine et dont chaque noeud Ami a au plus un fils).

SSS*

Tous les algorithmes vus jusqu'à présent sont basés sur des recherches en profondeur. Peut-on faire un algo de recherche en largeur ? **Oui**, SSS* est un algorithme général de recherche dans les graphes ET/OU que l'on peut appliquer aux jeux.
L'idée est d'agrandir peu à peu des arbres ET/OU en choisissant de développer à chaque fois le plus prometteur. On manipule des Stratégies partielles (Arbre contenant la racine et dont chaque noeud Ami a au plus un fils).

SSS*
Sur un exemple



Synthèse expérimentale
Sur l'Othello

p	minimax	Scout	α-β	SSS*
1	3 – 0,37	4 – 0,45	3 – 0,36	3 – 0,36
2	14 – 1,38	21 – 2.09	13 – 1,4	11 – 1,19
3	61 – 6,0	45 – 5,0	37 – 3,9	35 – 3,7
4	349 – 32,5	246 – 23,7	150 – 14,77	95 – 10,0
5	2050 – 185,7	615 – 61,6	418 – 41,4	292 – 19,2
6	13773 – 1213,5	2680 – 254,5	1830 – 172,9	1617 – 117,3

Contenu

1. Algorithmique Avancées
   - Écritures succinctes
   - Recherche Alpha-Béta sur des fenêtres réduites
   - Exploration en largeur
2. **Architectures performantes**
   - Considération de temps réel
   - Début, Milieu et Fin de partie
3. Heuristiques Avancées
   - Affiner la valeur de l'heuristique
4. Les systèmes qui fonctionnent
   - Échecs et Dames
   - Le futur des jeux

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin.

Découper le temps T que l'on a pour un coup en 3 :

1. Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
2. Si on ne trouve pas de coups gagnants, trouver un bon coup avec un $\alpha\beta$ classique en $8T/10$ secondes.
3. Vérifier en $T/10$ secondes que ce coup n'est pas mauvais percant à un horizon tactique.

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin.
Découper le temps T que l'on a pour un coup en 3 :

1. Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
2. Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
3. Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin.
Découper le temps T que l'on a pour un coup en 3 :

1. Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
2. Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
3. Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin.
Découper le temps T que l'on a pour un coup en 3 :

1. Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
2. Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
3. Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

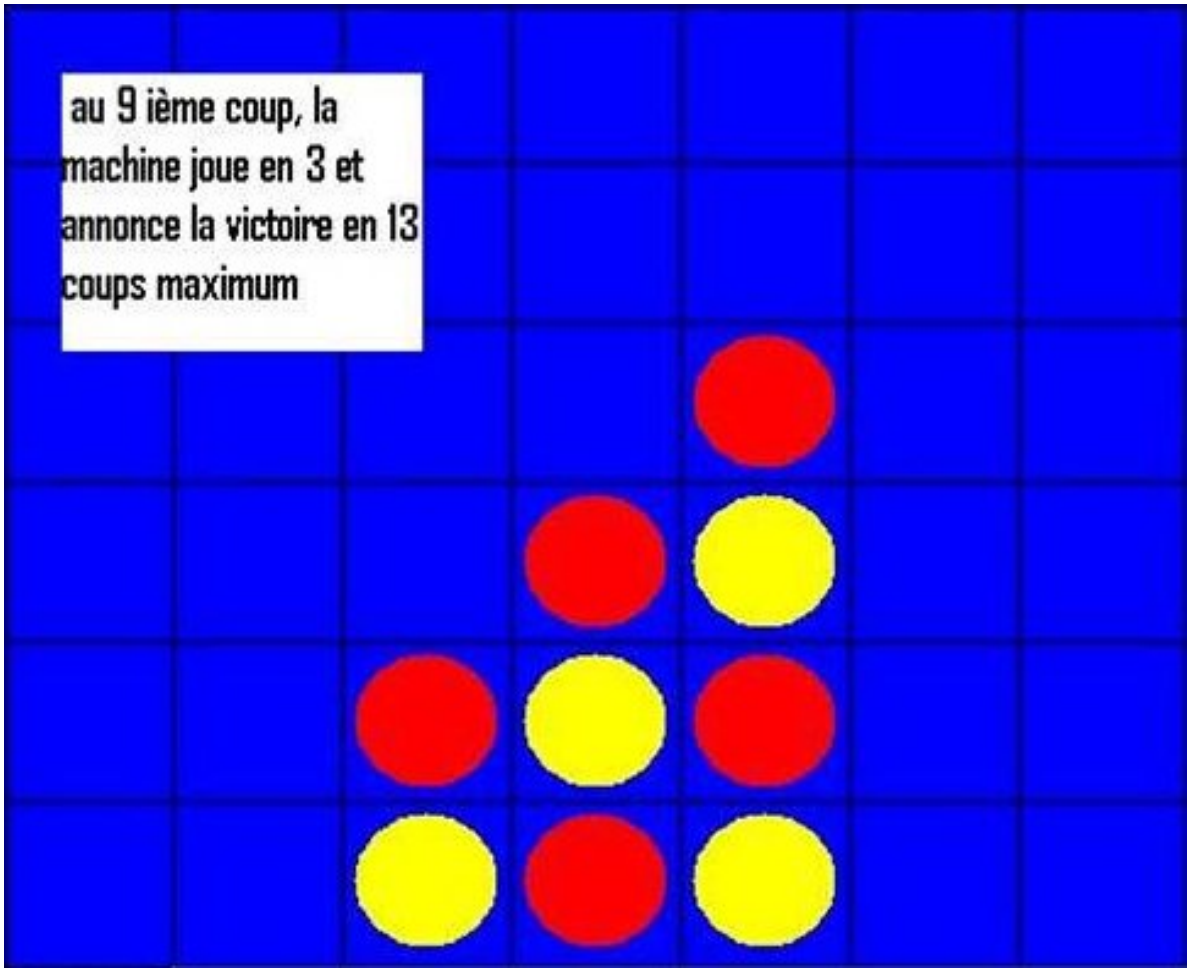
Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin.
Découper le temps T que l'on a pour un coup en 3 :

1. Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
2. Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
3. Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer les coups fatals

Exemple sur le Puissance 4

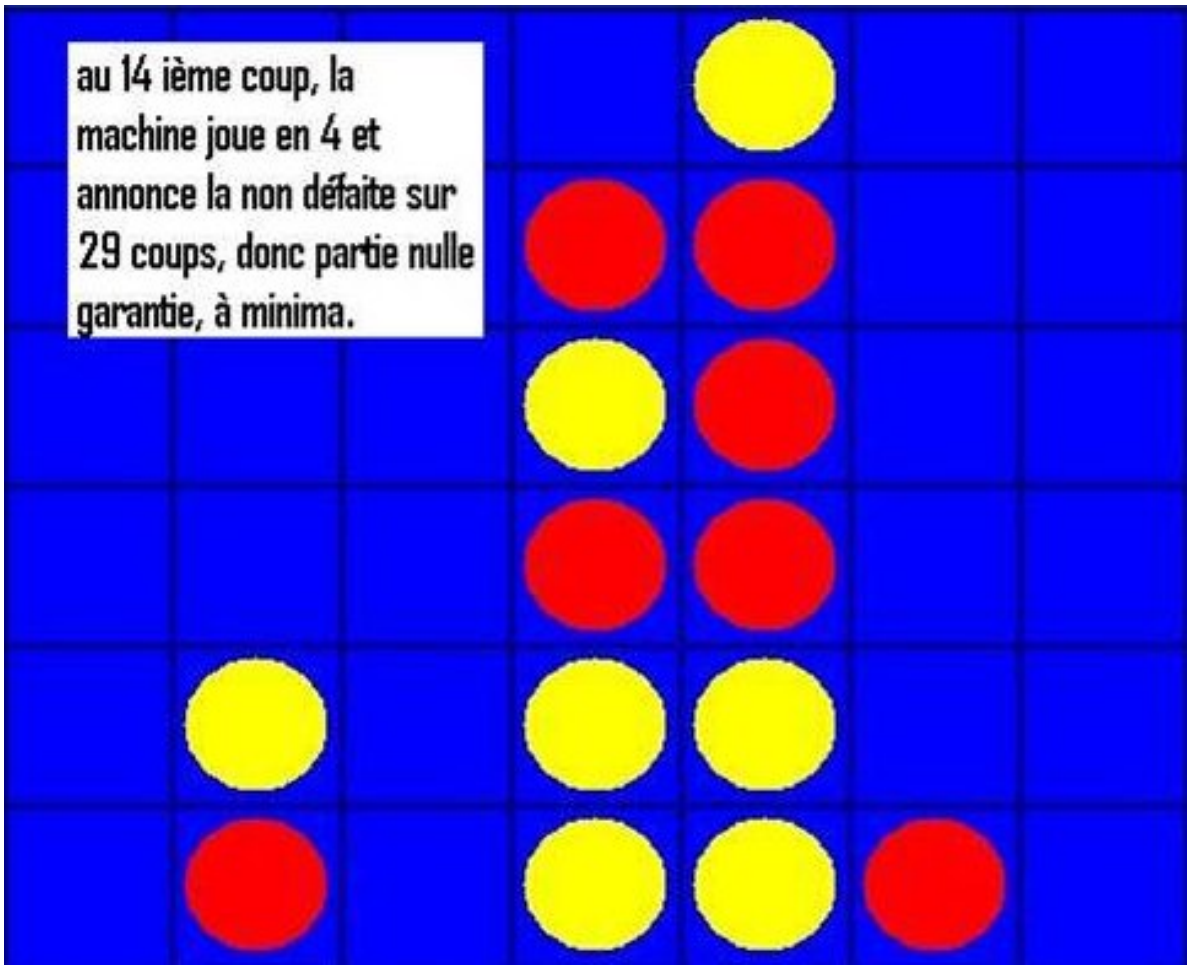
Images issues d'une applet en Java disponible sur le web.



Repérer les coups fatals

Exemple sur le Puissance 4

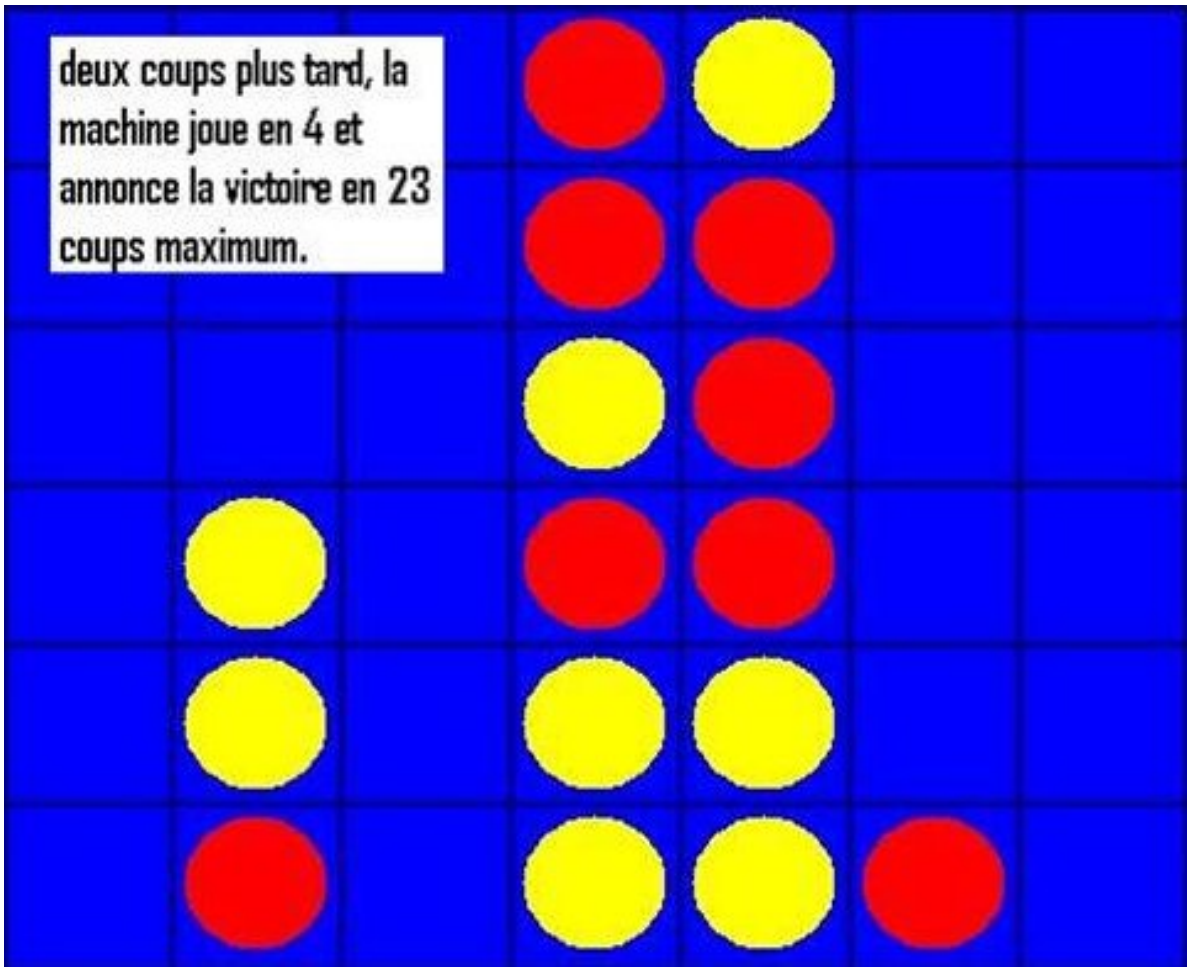
Images issues d'une applet en Java disponible sur le web.



Repérer les coups fatals

Exemple sur le Puissance 4

Images issues d'une applet en Java disponible sur le web.



Atténuation d'horizons

Zoomer où l'on va aller

Idée

Une fois que l'on a fait le choix du coup à jouer, passer un peu de temps pour voir si la fonction heuristique ne nous a pas trompé.

Deux solutions... Parmi d'autres

- Redévelopper un $\alpha\beta$ sur la branche sélectionnée.
- Redévelopper un $\alpha\beta$ sur la meilleure feuille visée.

Atténuation d'horizons

Reconsidérer la notion d'horizons

Définir l'horizon en fonction de l'intérêt des coups joués et non en fonction de leur nombre.

Principes

On part par exemple avec une profondeur $SX = 10.p$, où p est la profondeur au sens habituel.

- Un coup « moyen » diminue SX de 10.
- Un coup intéressant (prise de pièce, échec) ne diminue SX que de 2 ou 3.
- Un coup peu intéressant diminue beaucoup SX (de l'ordre de 35).

Résultat : les coups intéressants seront explorés plus profondément.

Problématique du temps réel

Questions

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?
- Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?
- Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?
- Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?
- Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?
- Comment jouer au mieux dans un temps donné ?

Iterative Deepening
Maîtriser le temps

Idée

Étendre peu à peu l'horizon de $\alpha\beta$.
Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening

Soit *n* initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- s'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter *n*.

Et les performances ?

ID semble refaire beaucoup la même chose. Est-ce que cela se ressent dans les performances ?

Iterative Deepening

Maîtriser le temps

Idée

Étendre peu à peu l'horizon de $\alpha\beta$.
Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening

Soit *n* initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- s'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter *n*.

Et les performances ?

ID semble refaire beaucoup la même chose. Est-ce que cela se ressent dans les performances ?

Iterative Deepening

Maîtriser le temps

Idée

Étendre peu à peu l'horizon de $\alpha\beta$.
Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening

Soit *n* initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- s'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter *n*.

Et les performances ?

ID semble refaire beaucoup la même chose. Est-ce que cela se ressent dans les performances ?

Iterative Deepening

Un algorithme pas si *inefficace*

La croissance exponentielle des arbres montre intuitivement que la grande majorité des noeuds (donc de la difficulté) se situent sur les feuilles...
Développer le dernier niveau coûte le plus cher, d'autant plus si on a un grand facteur de branchement.

Qui veut des chiffres ?

Une recherche à profondeur d et facteur de branchement b coûte

$$1 + b + b^2 + \ldots + b^{d-1} + b^d$$

Pour $(d = 5, b = 10)$ on a 11 111 noeuds.
Si on fait un ID, on aurait développé 12 345 noeuds au total (1 pour d=1, 11 pour d=2, 111 pour d=3, 1 111 pour d=4 . . .), soit 11% de noeuds supplémentaires (seulement).

Iterative Deepening

Des avantages de poids avec $\alpha\beta$

Avantages

- Grande souplesse dans la gestion du temps. Le programme peut donner la meilleure valeur trouvée à n'importe quel moment.
- Couplé à $\alpha\beta$, ID peut même devenir plus efficace qu'un seul $\alpha\beta$

Comment ?

$\alpha\beta$ élague d'autant plus que les meilleurs coups sont développés en premier. On profite donc de l'ancien $\alpha\beta$ à profondeur n pour ordonner les fils à profondeur $n + 1$.

Analyse des parties I

Passer du temps là où il faut

Concentrer l'effort de recherche de la machine dans les zones critiques

- Si tous les débuts de partie se ressemblent, ne pas y consacrer trop de temps
- Passer plus de temps au milieu de partie (ex Awalé) car les fins de partie ne demandent pas beaucoup de temps (élagage efficace dans l'Awalé).

Mais seule une connaissance profonde du jeu permet d'affiner les moments où la recherche de bon coups est la plus critique.

Analyse des parties II

Passer du temps là où il faut

3 phases communes à tous les jeux

- Début de partie
- Milieu de partie
- Fin de partie

Un bon programme de jeu doit avoir un comportement fondamentalement différent dans chacune de ces phases.

Agir sur les heuristiques

On peut faire évoluer la valeur de la fonction heuristique suivant l'avancement dans le jeu.

Exemples

- Awalé : faire des greniers en début de partie, puis simplement compter les graines en fin de partie
- Othello : Prendre des cases stratégiques, puis compter les pions.
- ...

La transition entre les différentes fonctions heuristiques doit être douce.

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même.

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

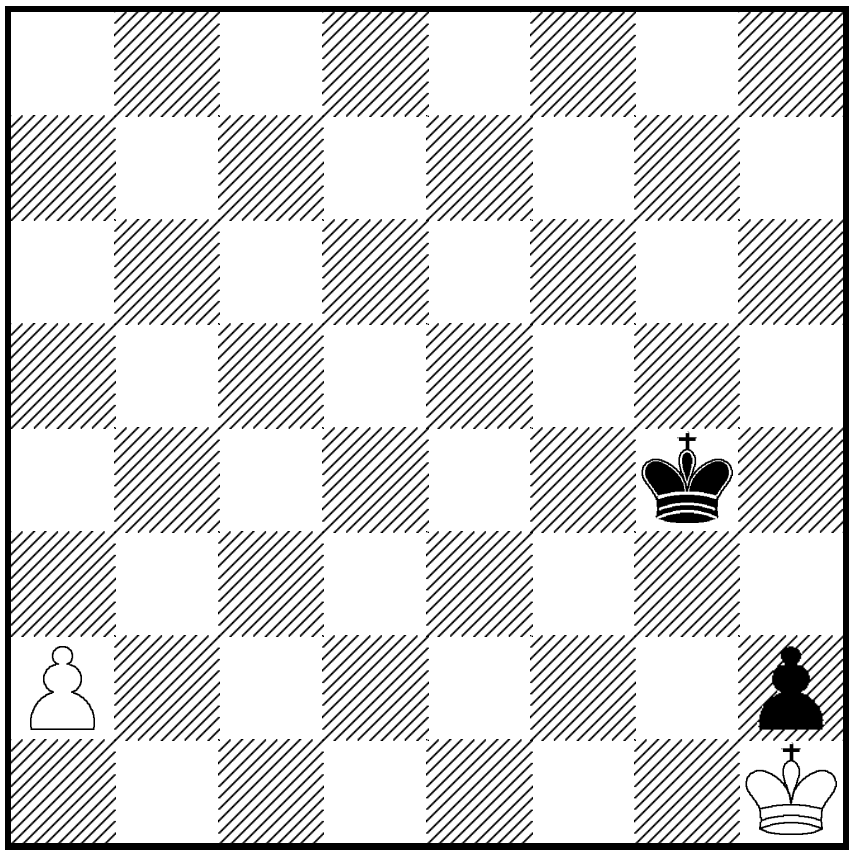
Ne pas oublier d'introduire du hasard...

Milieu de partie

Domaine de $\alpha\beta$ avec recherche éventuelle de coups profonds.

Fin de partie

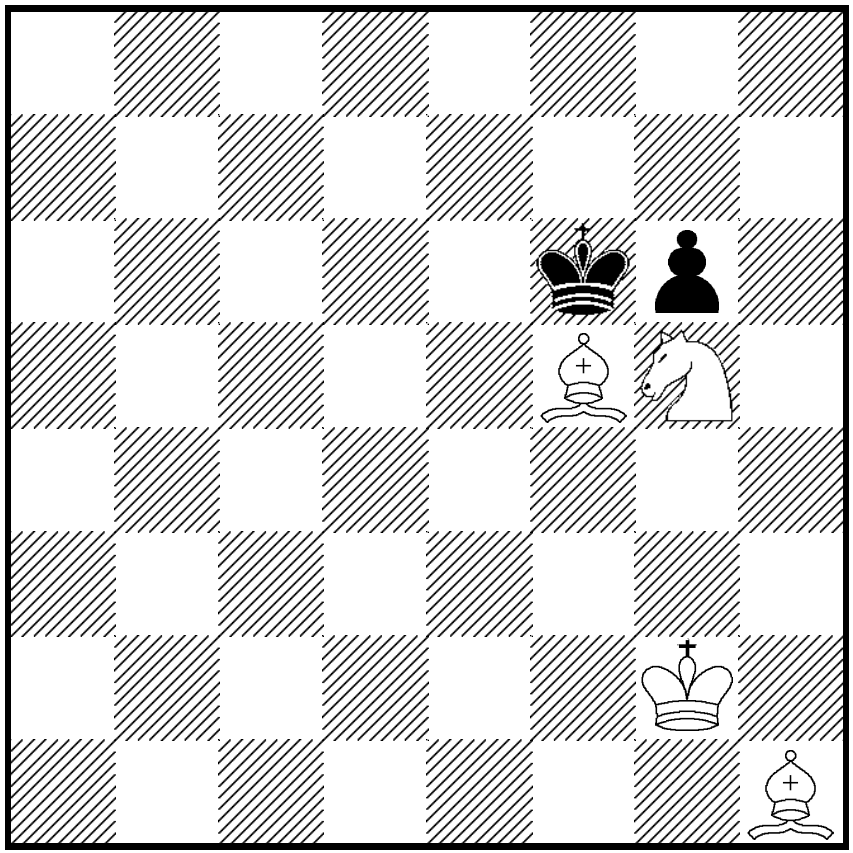
On n'en est pas arrivé là pour perdre (1/3) !



Blanc joue. Un problème d'horizon trompe Noir : il faudrait voir à 10 demi-coups pour bien jouer.

Fin de partie

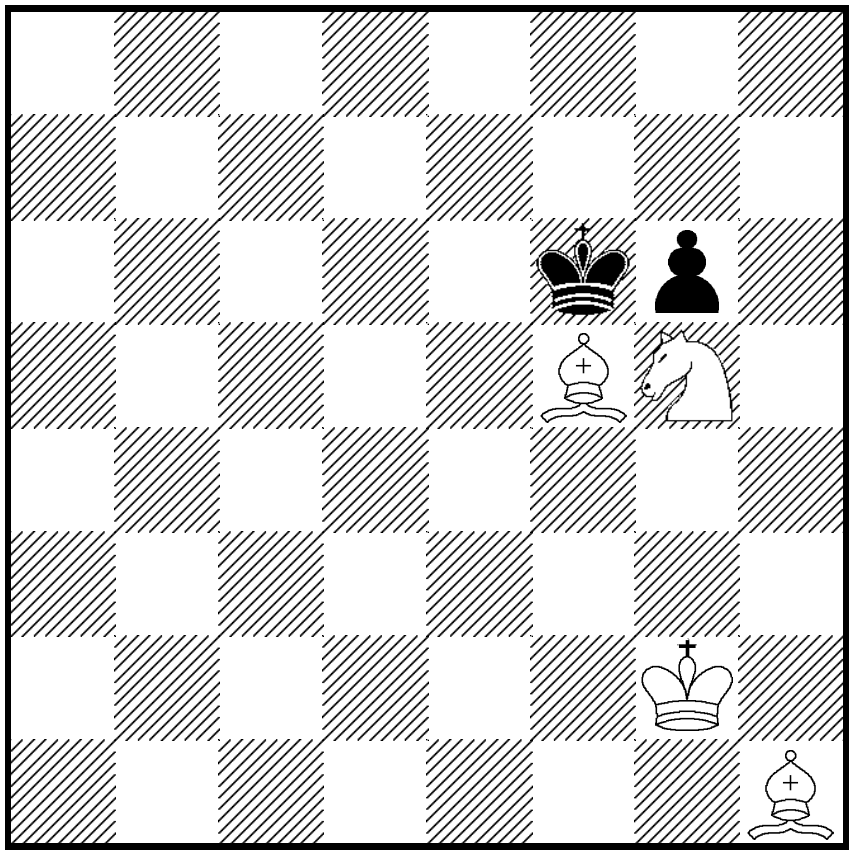
On n'en est pas arrivé là pour perdre (2/3) !



Blanc joue et doit perdre son fou ou son cheval. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Fou, Cavalier contre Roi, Gagnante

Fin de partie

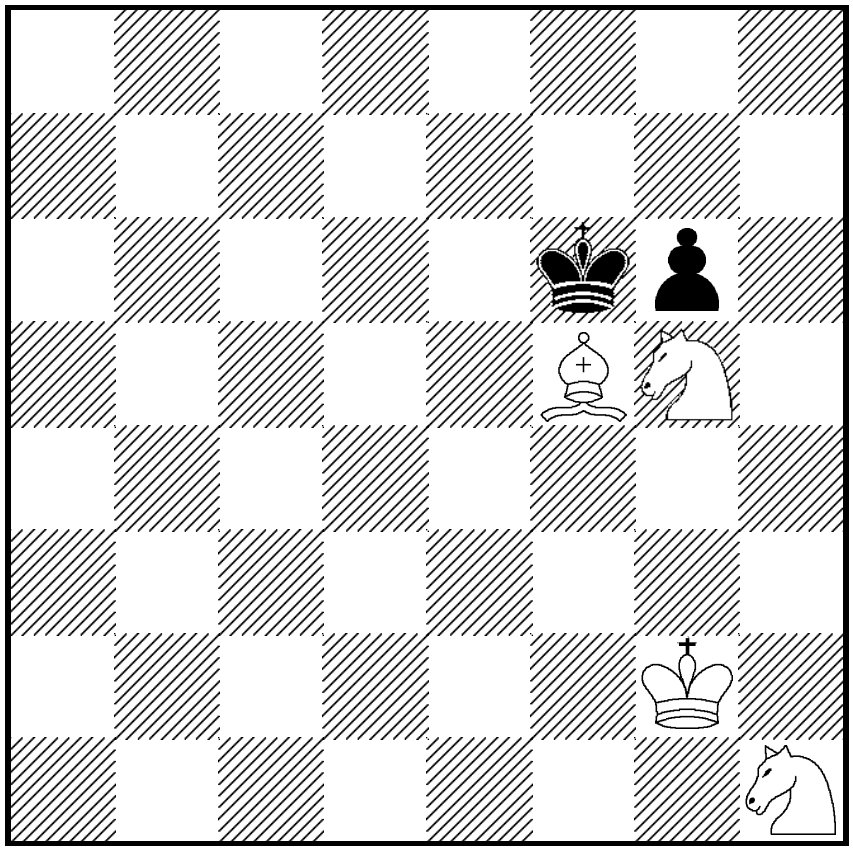
On n'en est pas arrivé là pour perdre (2/3) !



Blanc joue et doit perdre son fou ou son cheval. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Fou, Cavalier contre Roi, Gagnante

Fin de partie

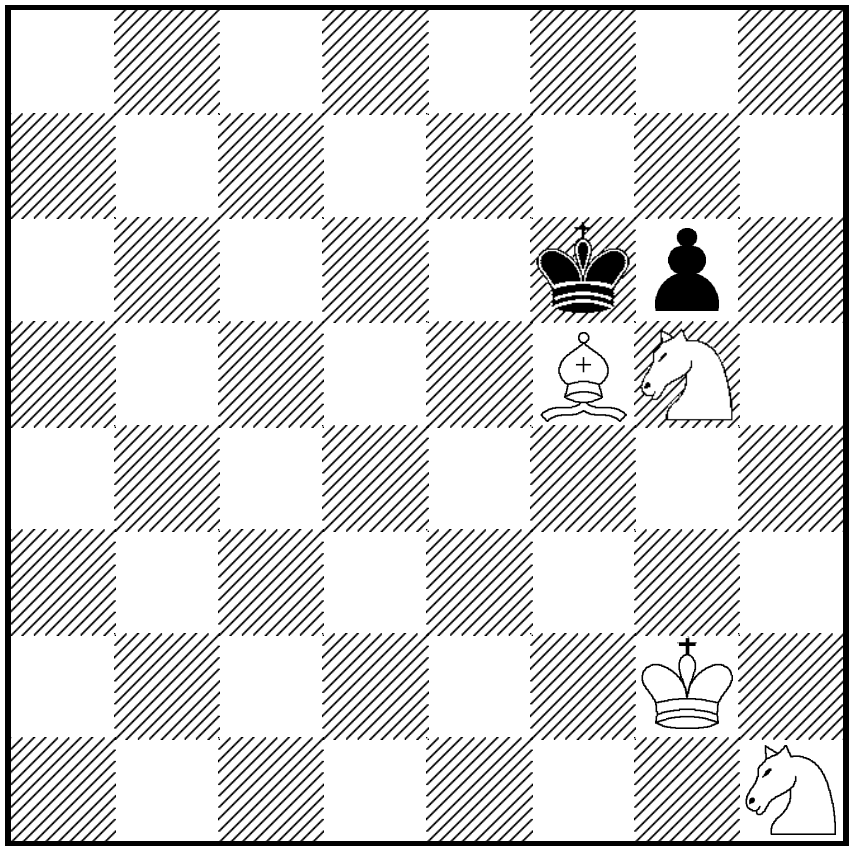
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

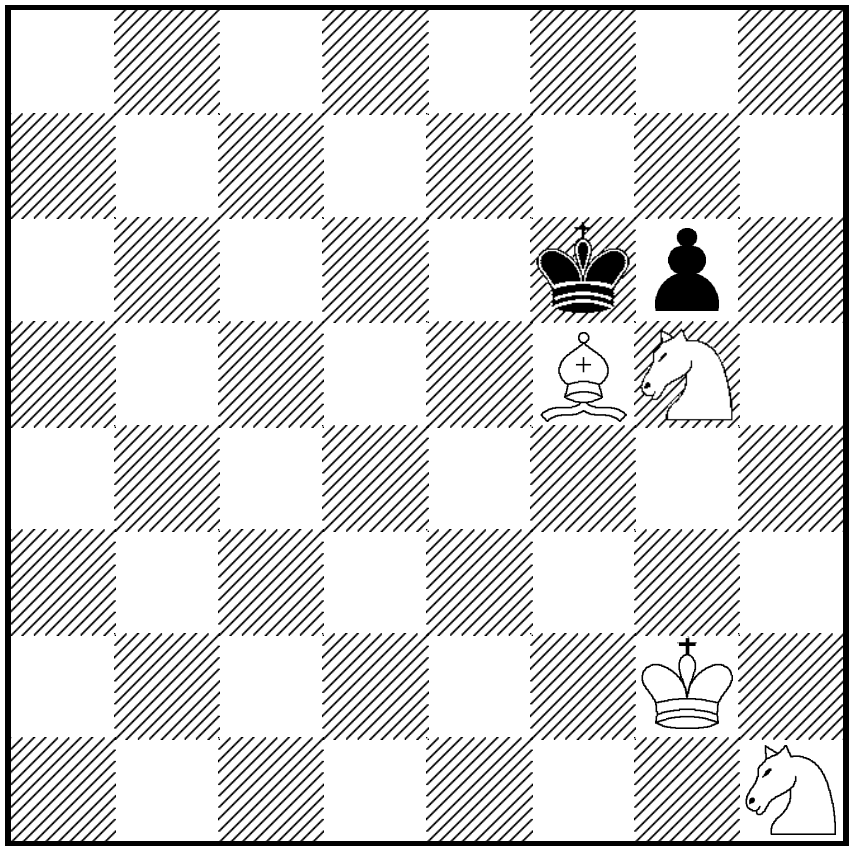
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

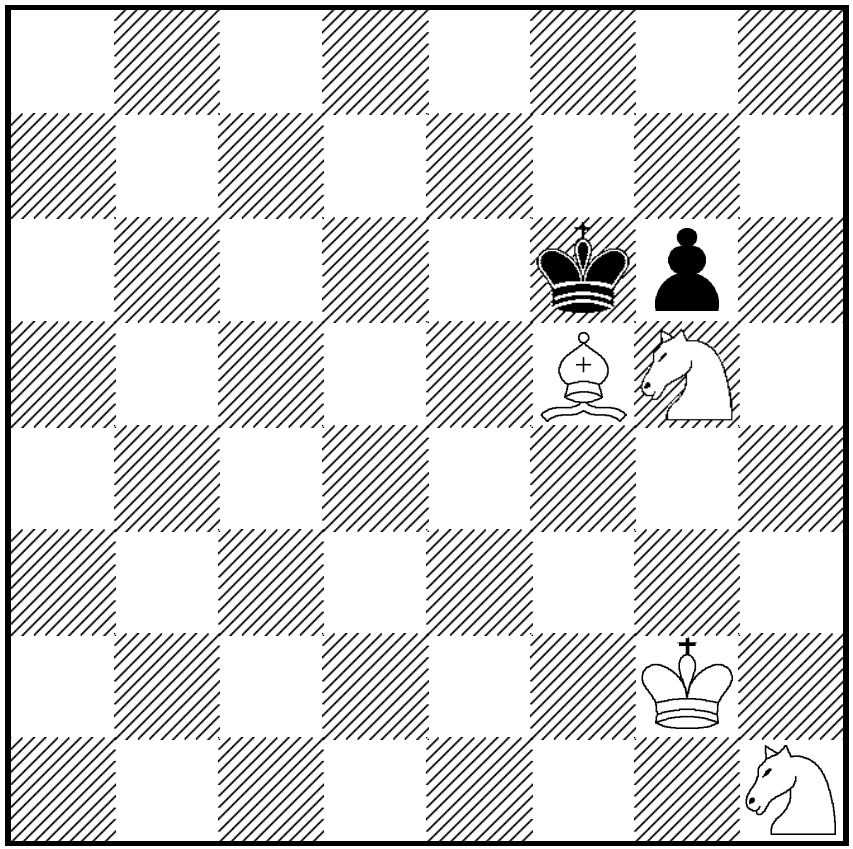
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie sont fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

D'autres cas pathologiques aux échecs : certaines finales imposent un plan très précis pour faire Mat en moins de 50 coups.
Il faut savoir reconnaître des fins de parties gagnantes ou perdantes. Il faut construire une bibliothèque de fermeture soit :

- par un expert
- par analyse rétrograde de toutes les positions finales du jeu

Dernière question : comment vérifier efficacement si un plateau de jeu a déjà été vu ?

Contenu

1 Algorithmique Avancées
- Écritures succinctes
- Recherche Alpha-Béta sur des fenêtres réduites
- Exploration en largeur

2 Architectures performantes
- Considération de temps réel
- Début, Milieu et Fin de partie

3 **Heuristiques Avancées**
- Affiner la valeur de l'heuristique

4 Les systèmes qui fonctionnent
- Échecs et Dames
- Le futur des jeux

Coups spéciaux I

Coups meurtriers

Principe : Si la valeur heuristique d'un coup chute brutalement, c'est que le coup joué est *meurtrier*.
En pratique : Une fois les coups meurtriers identifiés, les développer en priorité dans $\alpha\beta$.

Évaluation dans des zones calmes

Principe : si on évalue une position alors que des pièces peuvent encore être prises, on n'a pas de bonne estimation.
En pratique : tant que des pièces peuvent être prises, la fonction heuristique déroule les prises les plus importante jusqu'au bout avant évaluation.

Coups spéciaux II

Heuristique de coup nul

Principe : si un joueur peut joueur deux coups de suite et que la valeur heuristique du damier ne change pas beaucoup, le premier coup peut être écarté.
Problèmes :Dans certains cas, des parties (trop) importantes de l'arbre de recherche sont élaguées.

Apprentissage I

Apprendre avant de jouer

Commencer avec une heuristique idiote et laisser le programme apprendre par lui-même une bonne fonction heuristique, en jouant contre lui-même.

Apprendre pendant le jeu

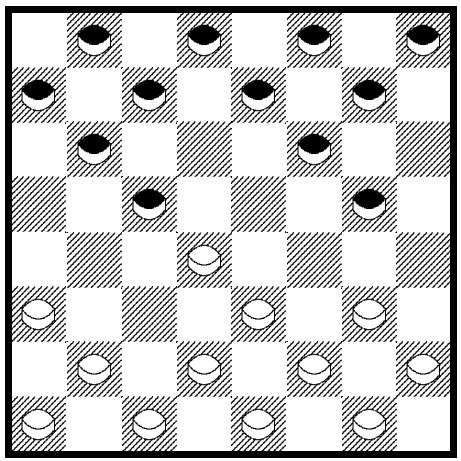
La différence entre l'estimation d'un noeud et sa vraie valeur (que l'on découvre plus tard) sert de base à l'apprentissage.

Contenu

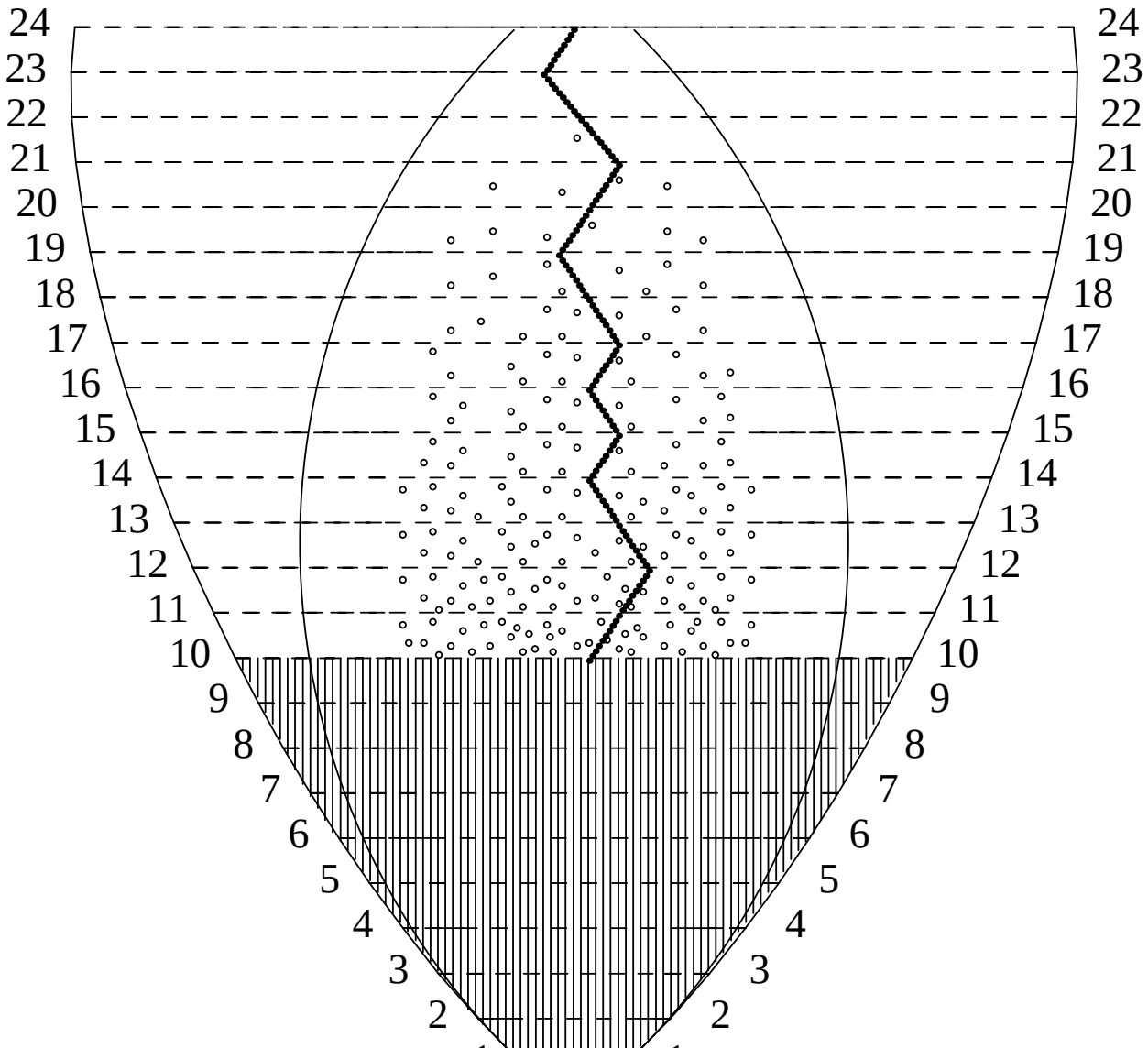
1. Algorithmique Avancées
   - Écritures succinctes
   - Recherche Alpha-Béta sur des fenêtres réduites
   - Exploration en largeur
2. Architectures performantes
   - Considération de temps réel
   - Début, Milieu et Fin de partie
3. Heuristiques Avancées
   - Affiner la valeur de l'heuristique
4. **Les systèmes qui fonctionnent**
   - Échecs et Dames
   - Le futur des jeux

Fermeture des Dames

L'un des prix IJCAI'05 a été donné à Schaeffer pour avoir notamment prouvé que l'ouverture aux dames ci-dessous n'était pas gagnante ni perdante :



Fermeture des Dames



Dans les profondeurs de *Deep Blue*

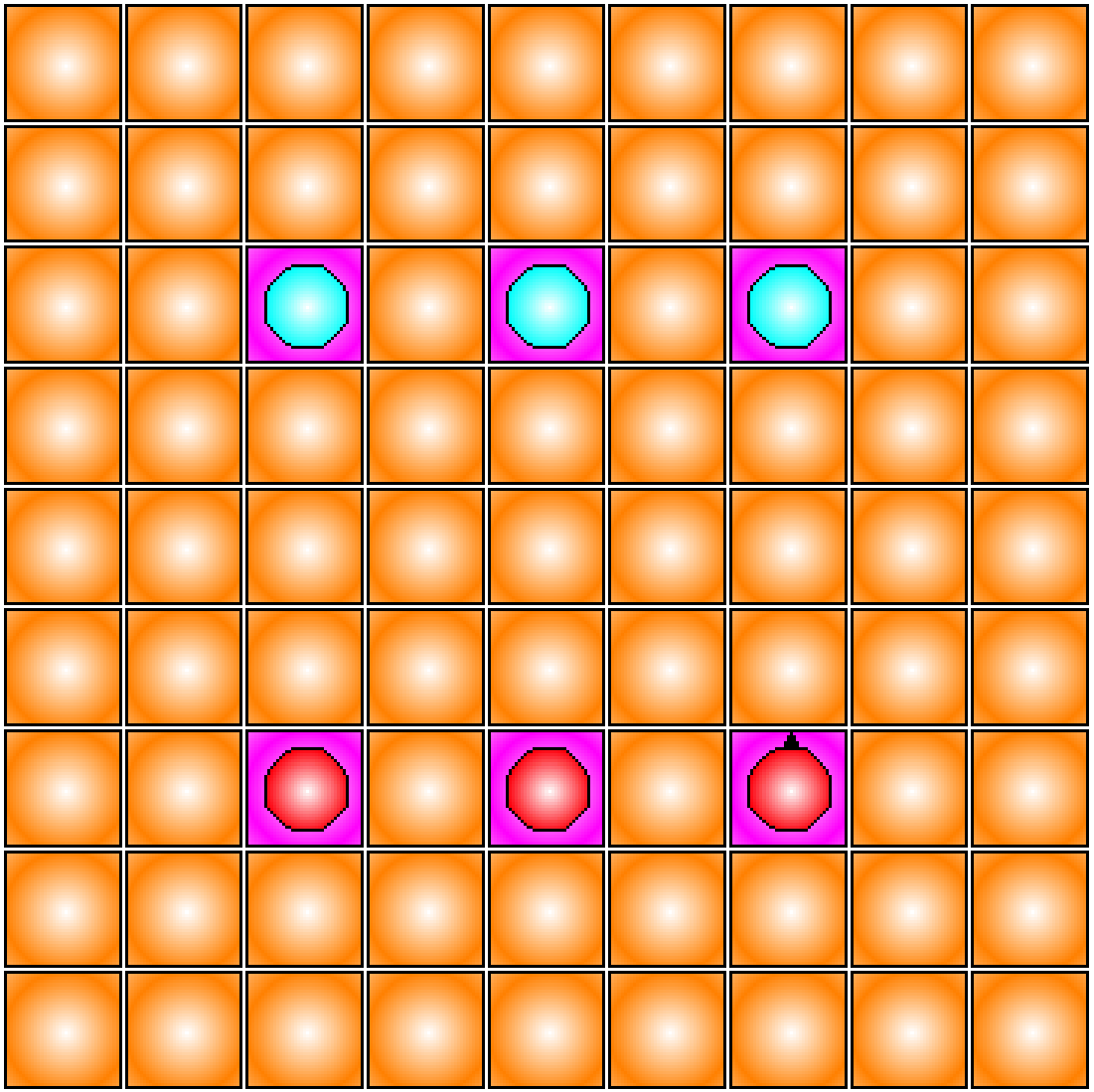
OCTI

Un jeu de 2002 créé pour résister aux machines

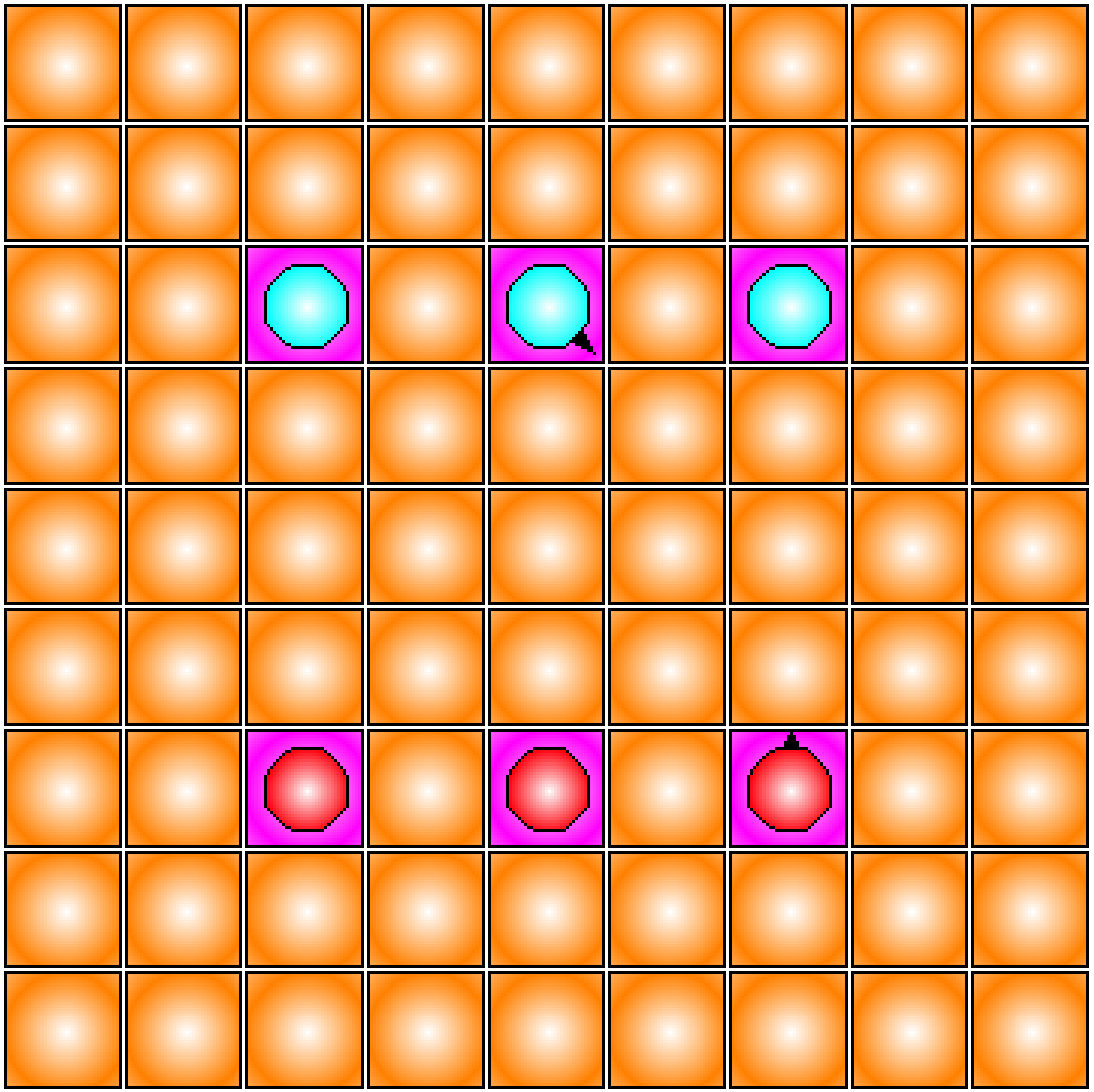
À vos idées !

Infos sur `www.octi.net`

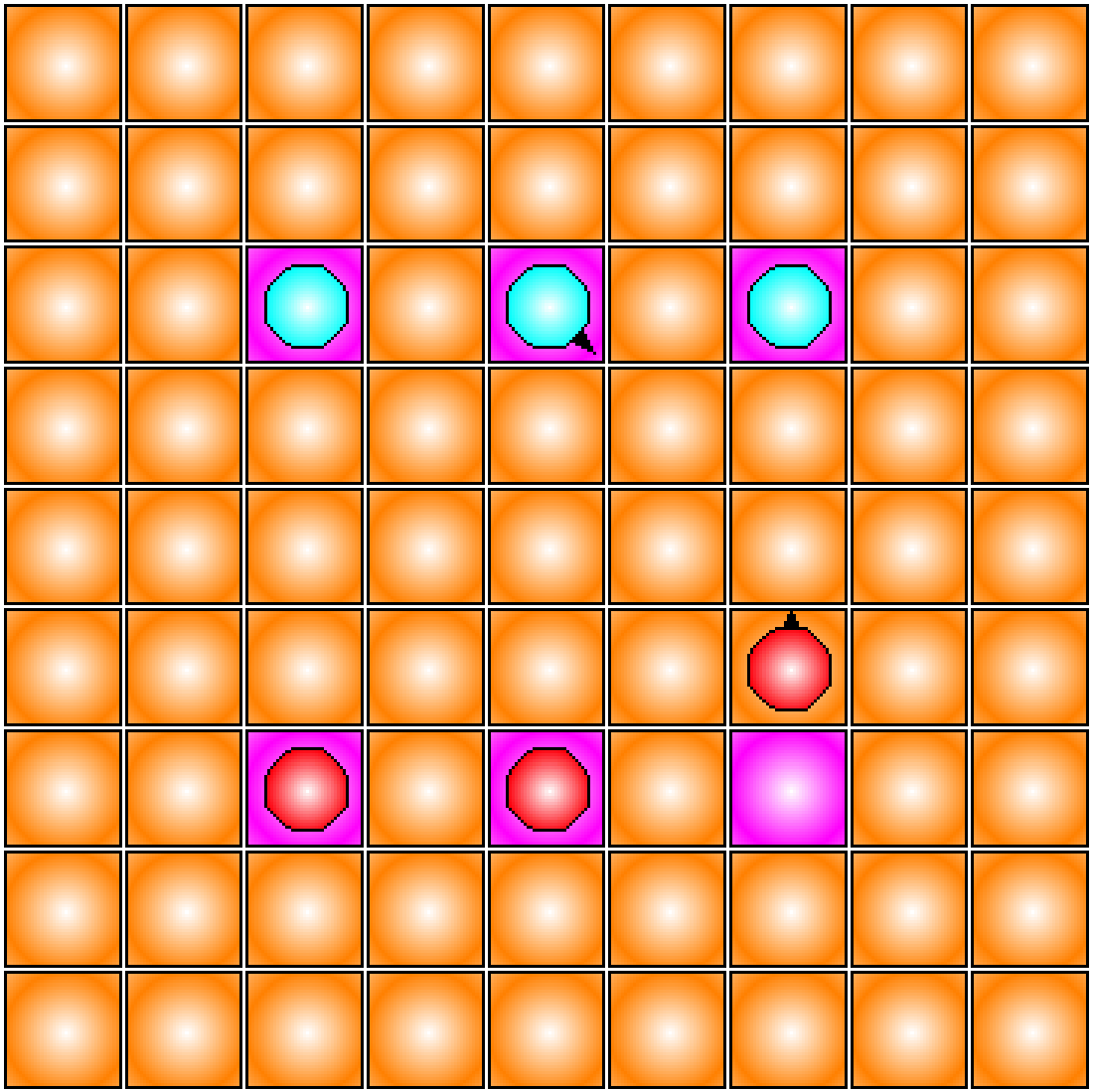
Exemple de partie OCTI



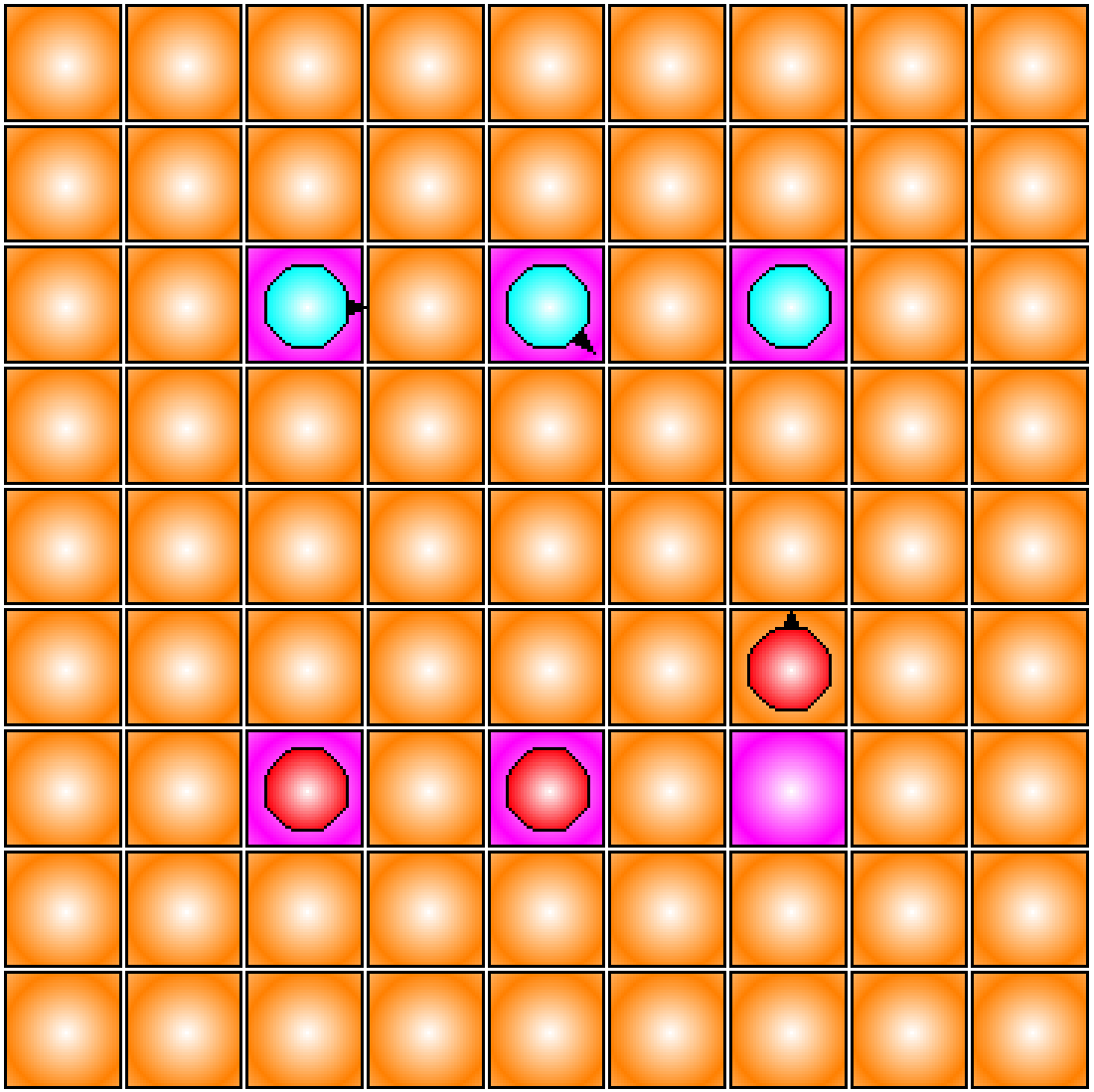
Exemple de partie OCTI



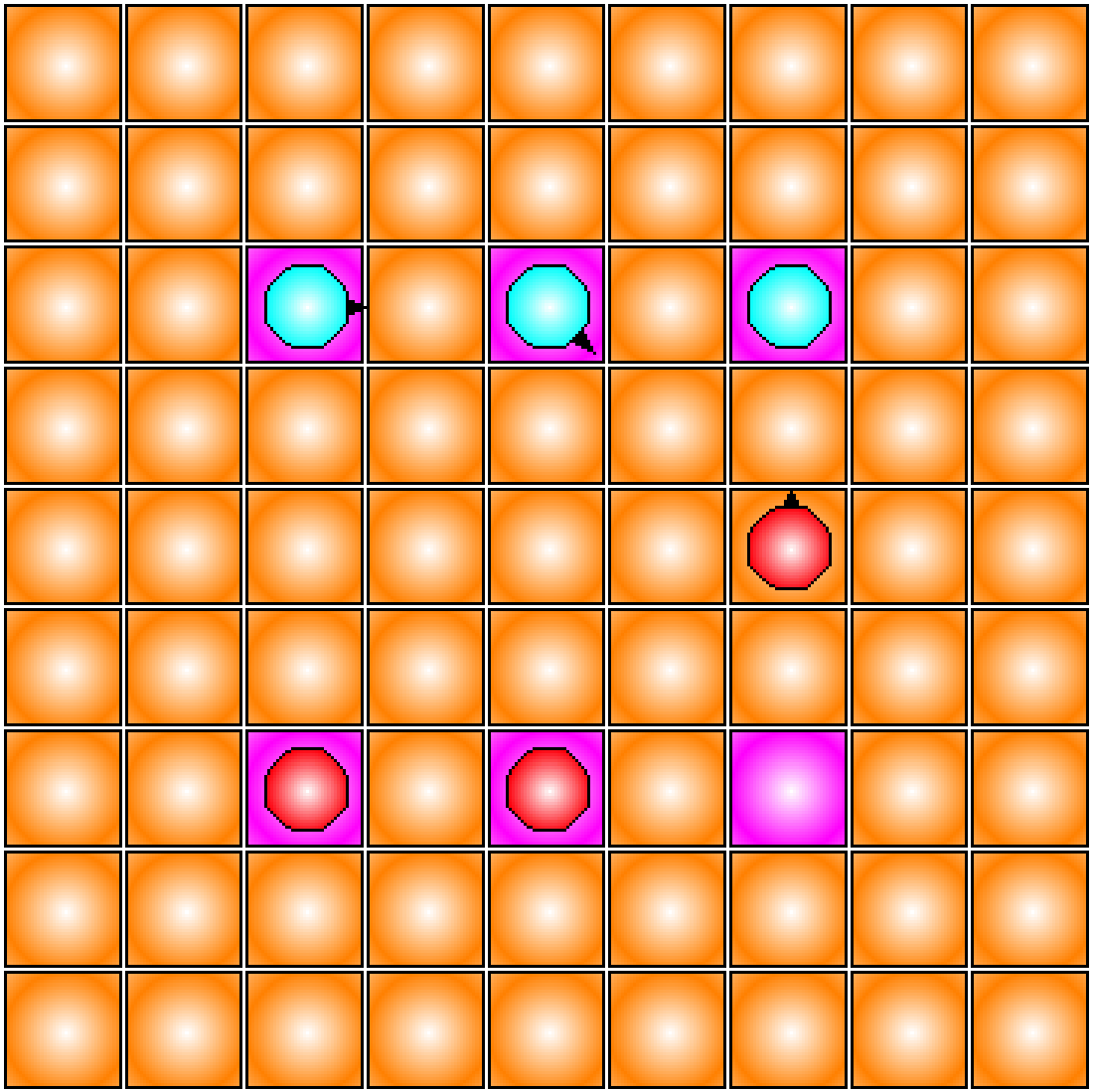
Exemple de partie OCTI



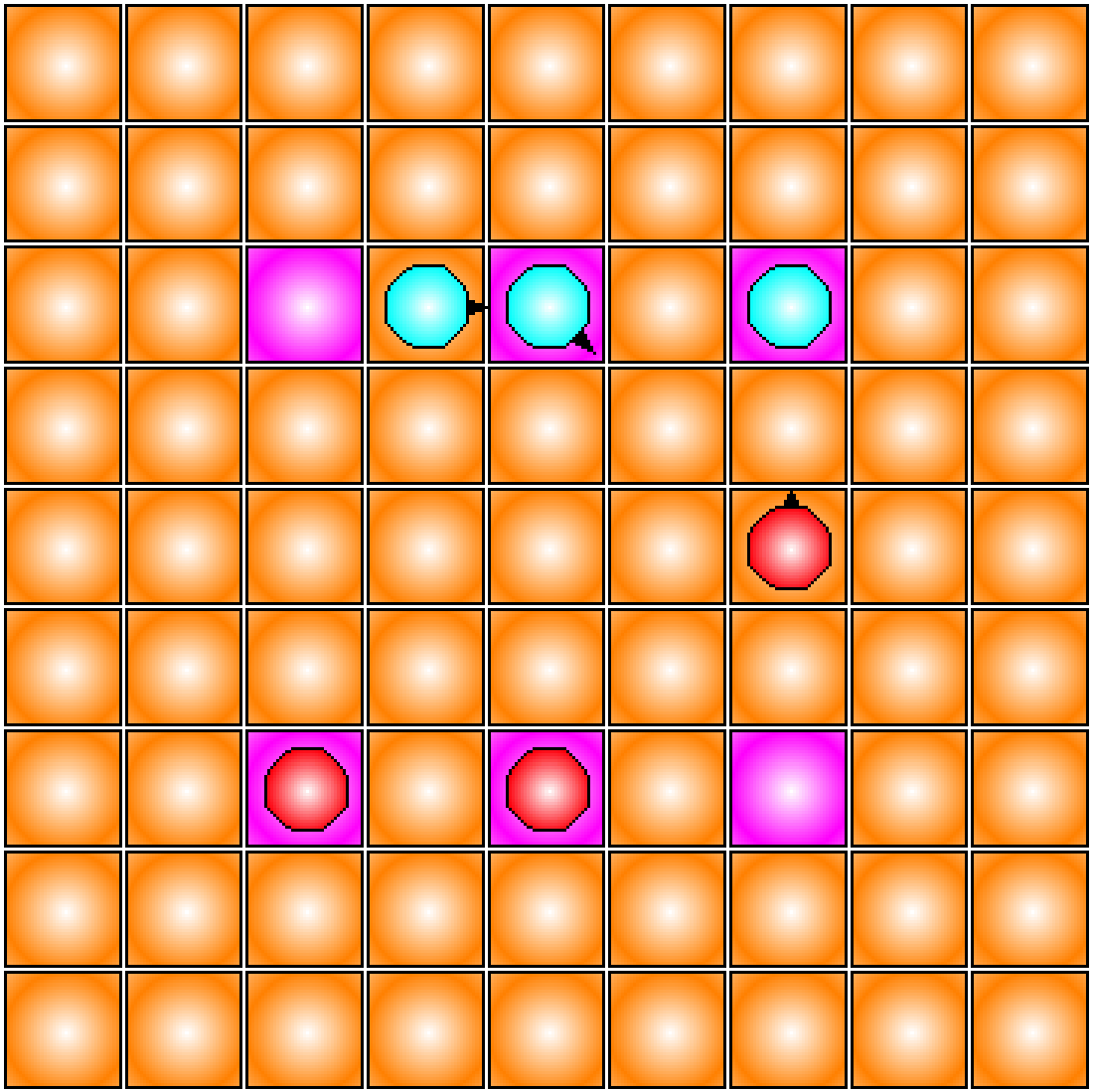
Exemple de partie OCTI



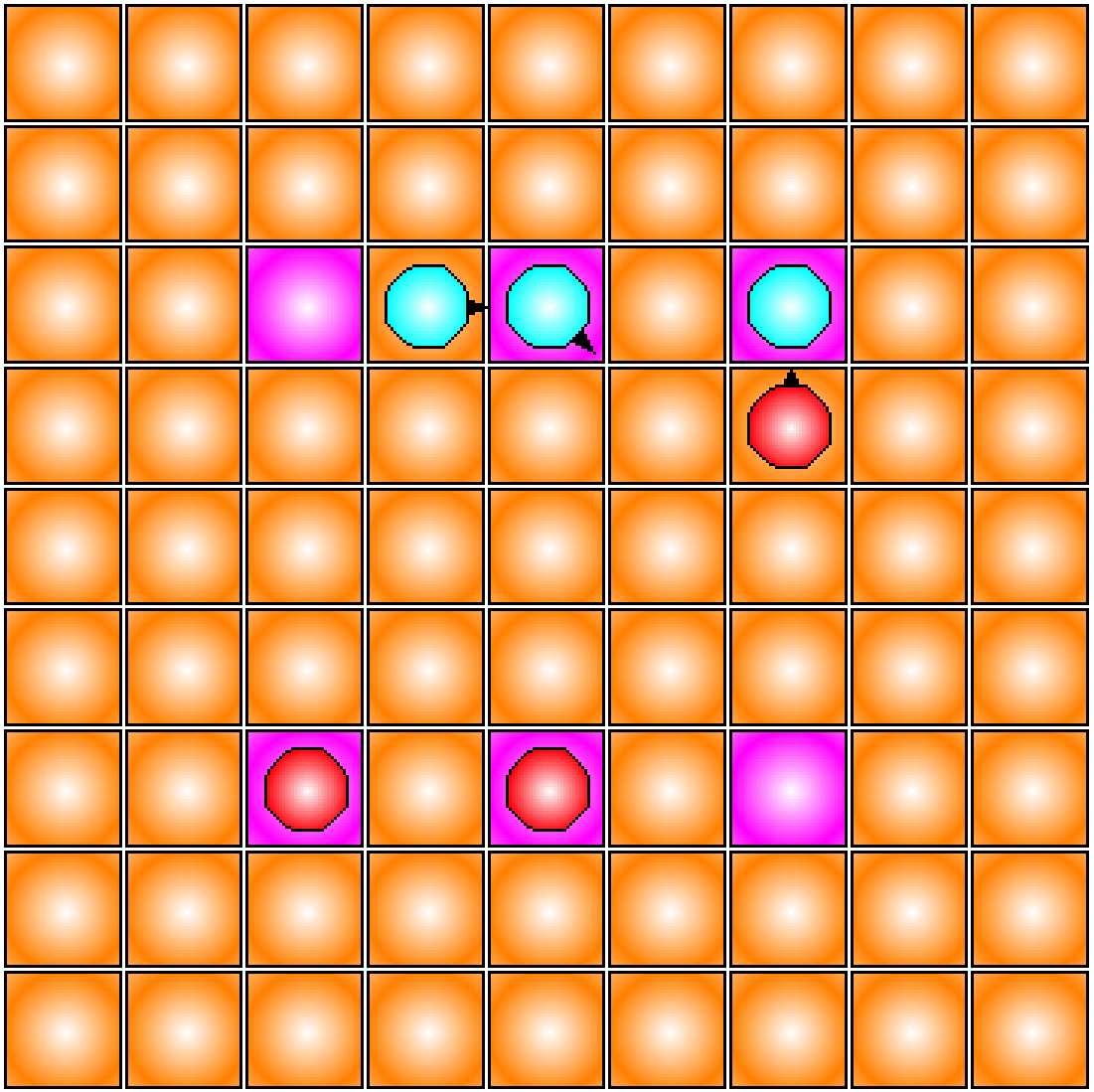
Exemple de partie OCTI



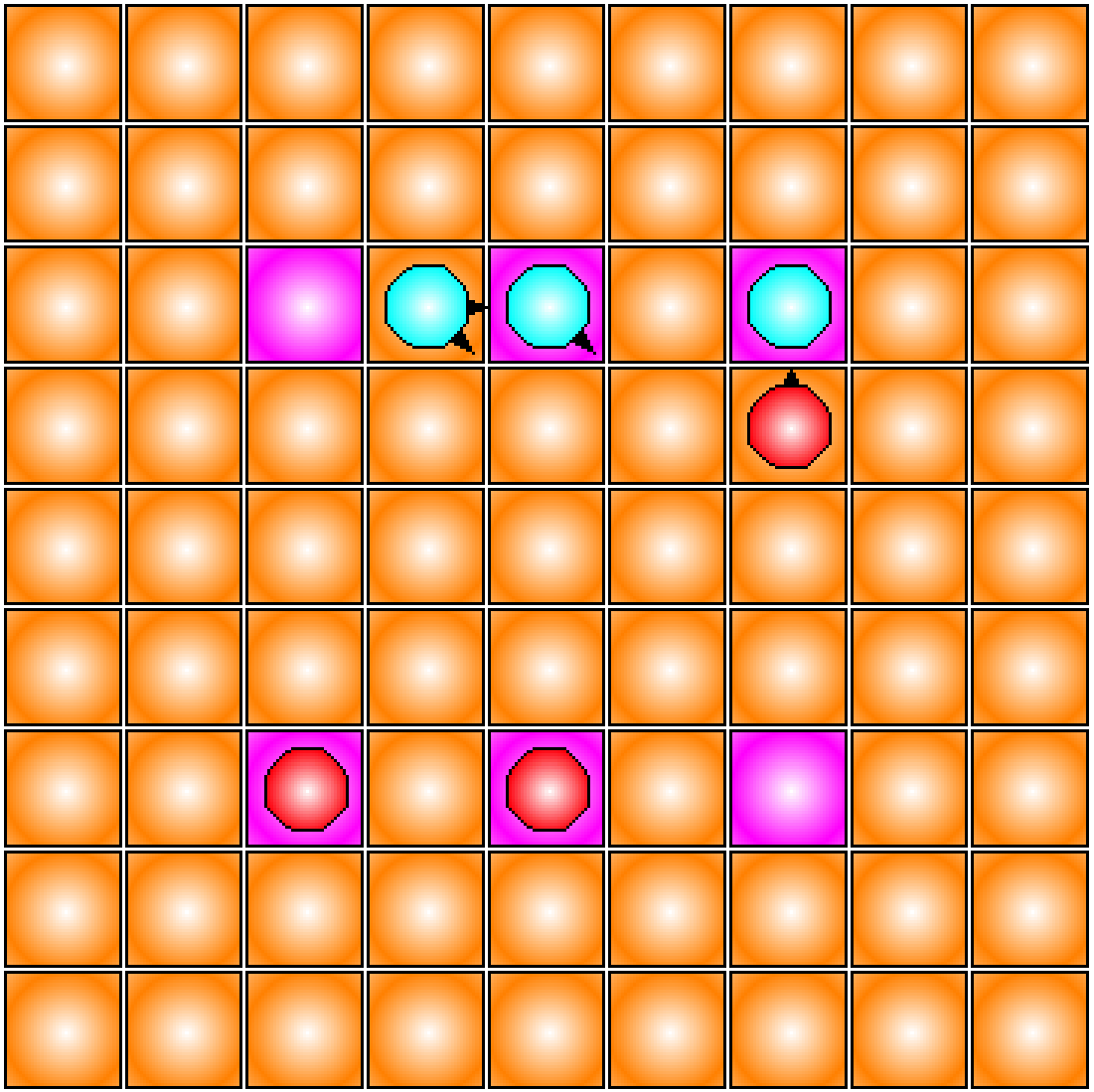
Exemple de partie OCTI



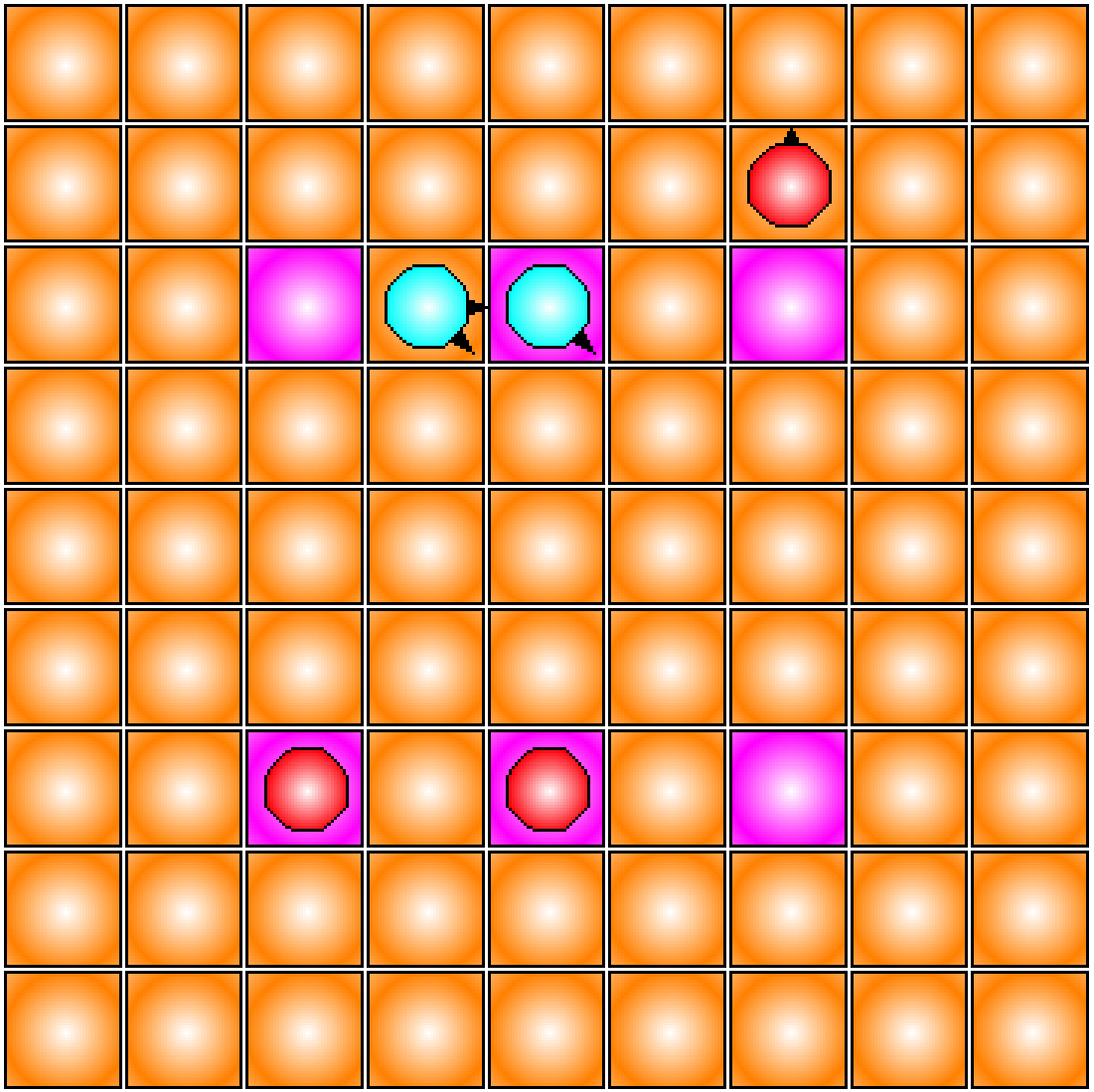
Exemple de partie OCTI



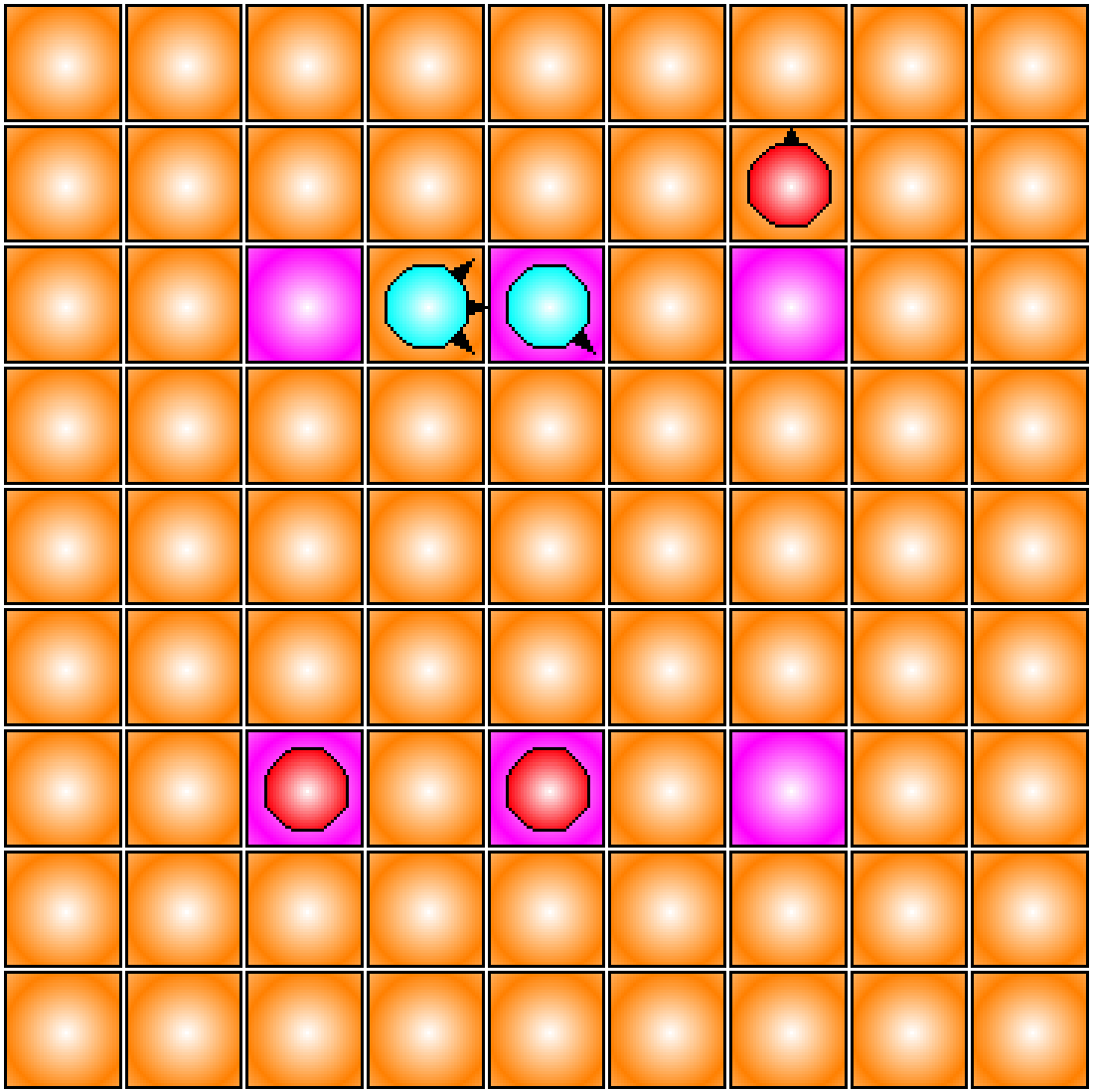
Exemple de partie OCTI



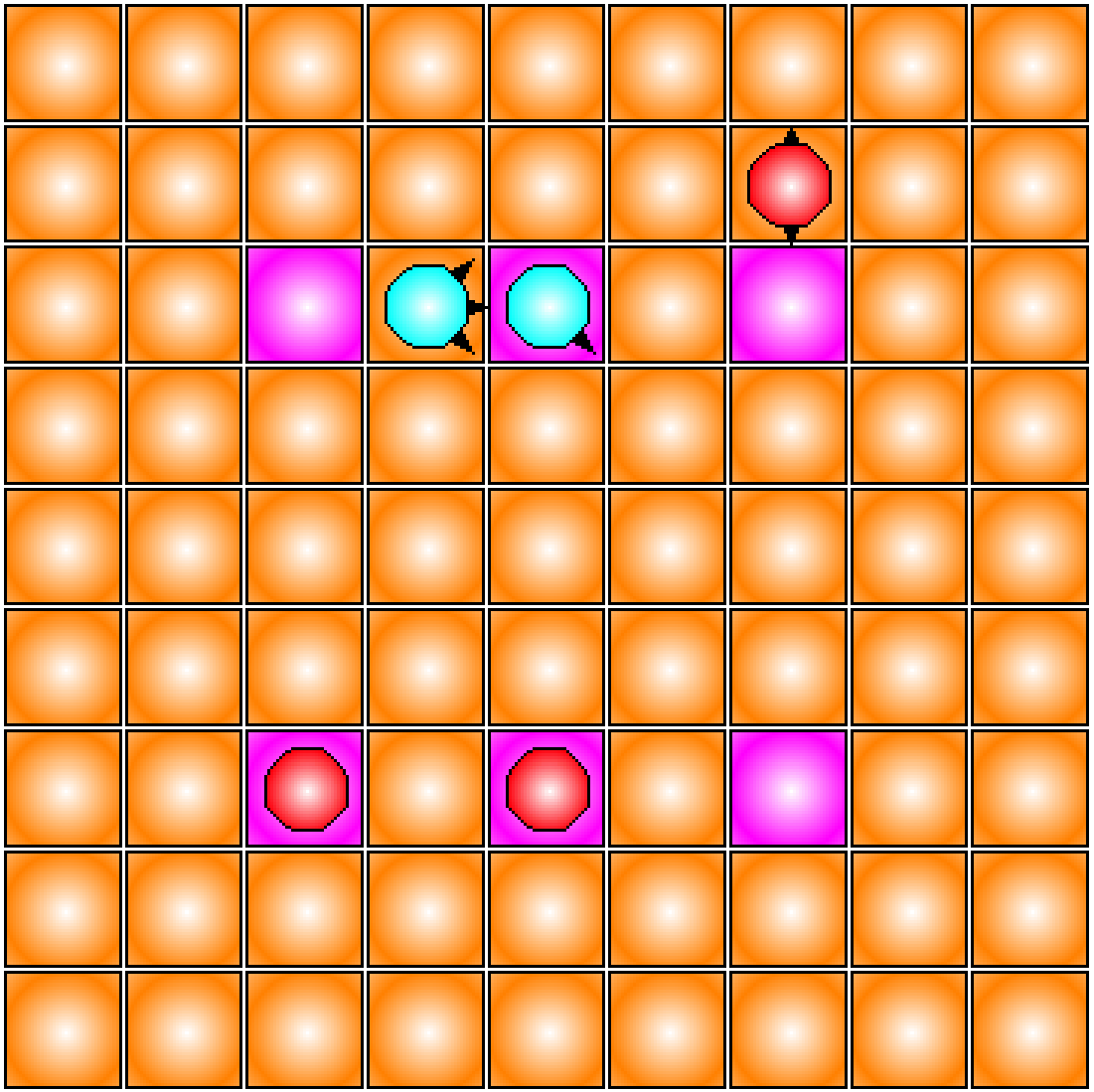
Exemple de partie OCTI



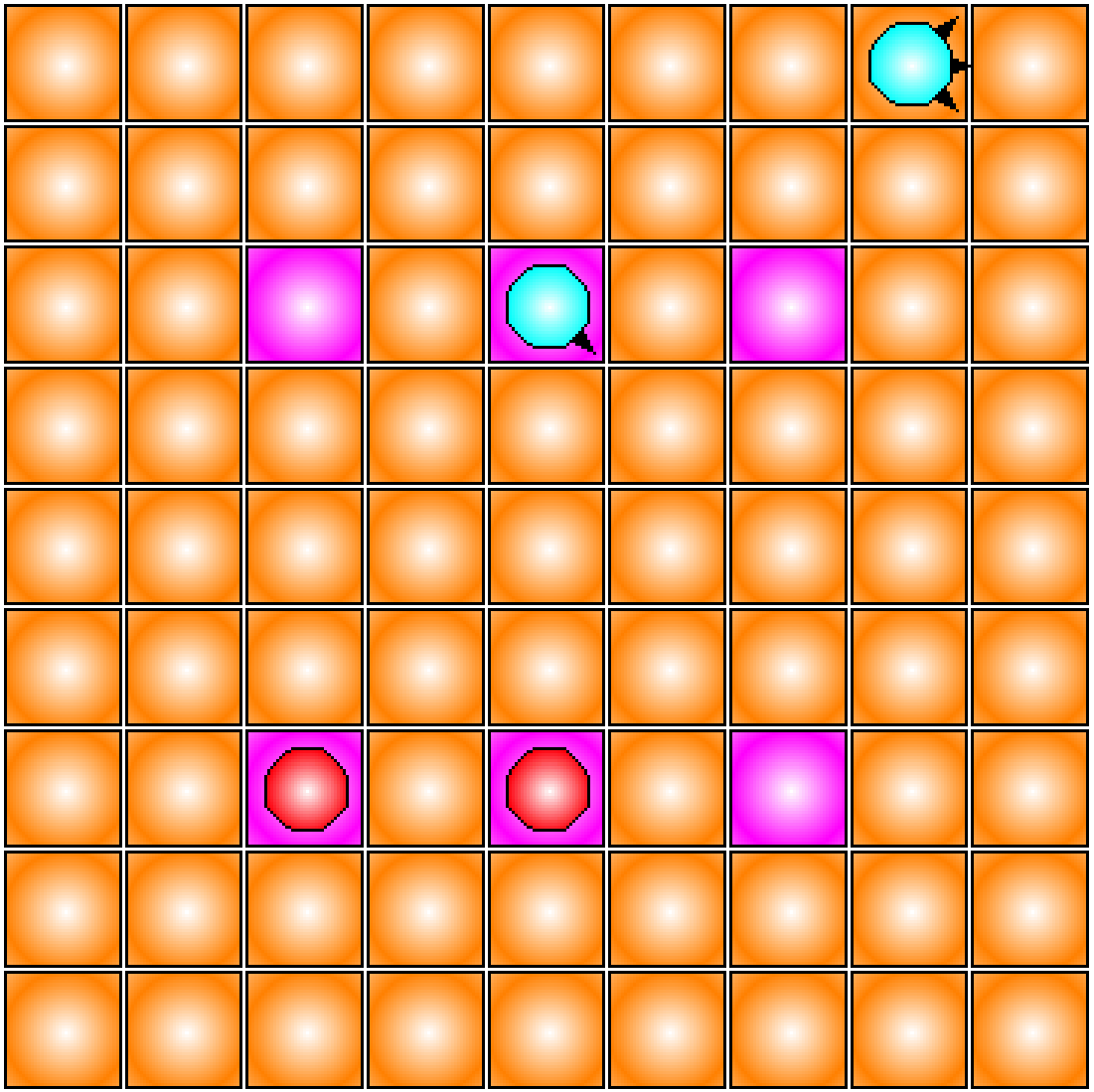
Exemple de partie OCTI



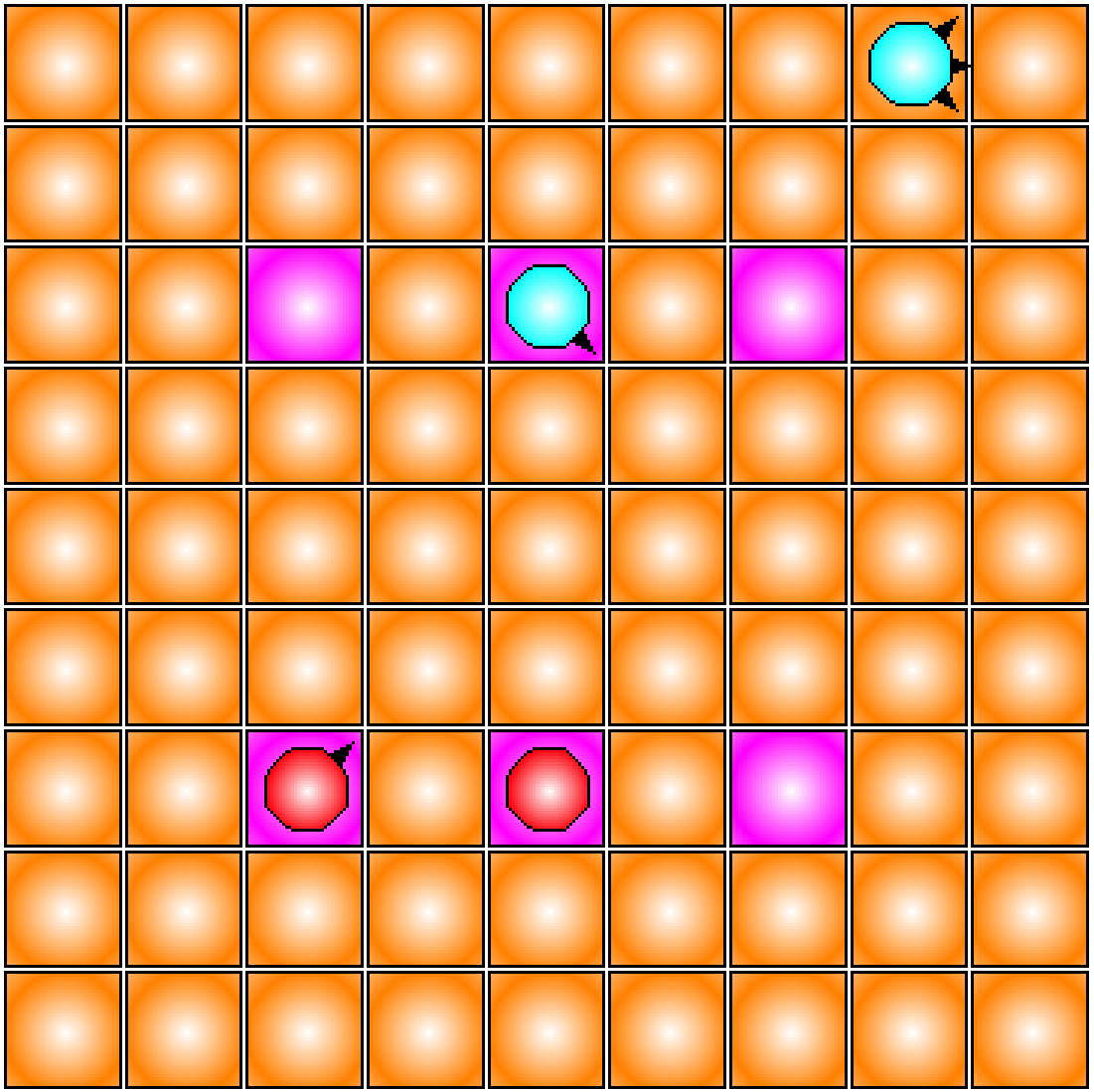
Exemple de partie OCTI



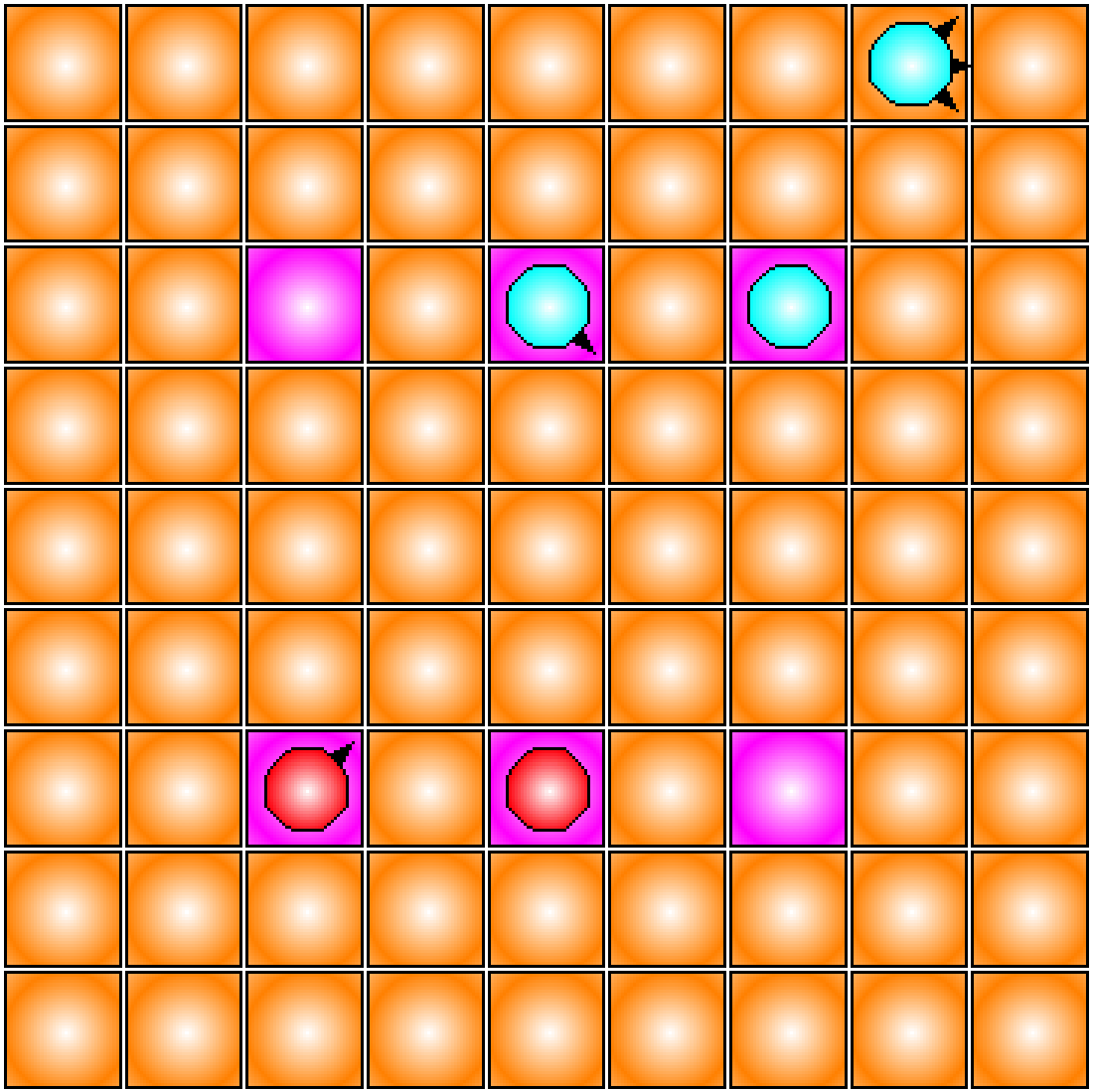
Exemple de partie OCTI



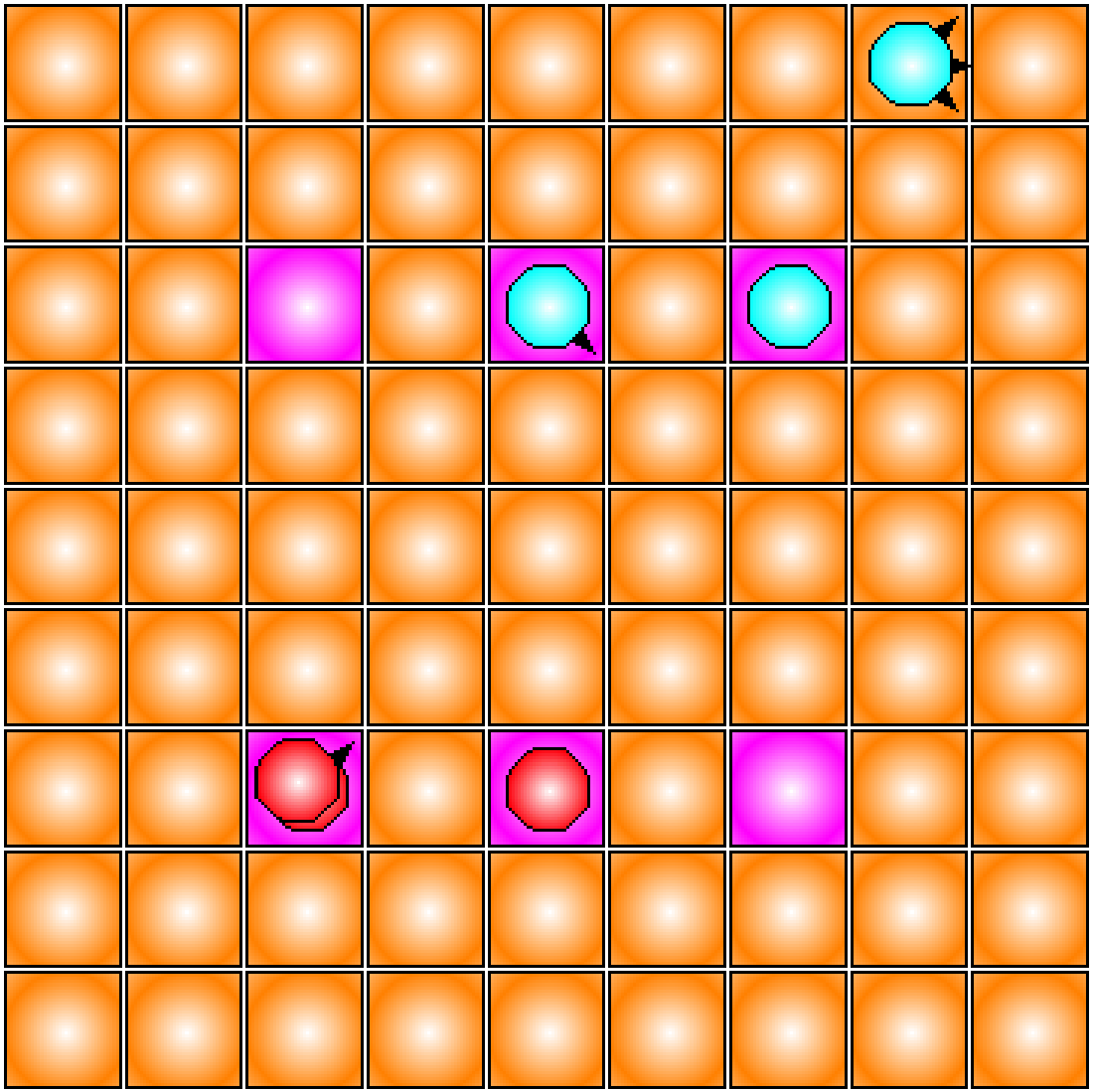
Exemple de partie OCTI



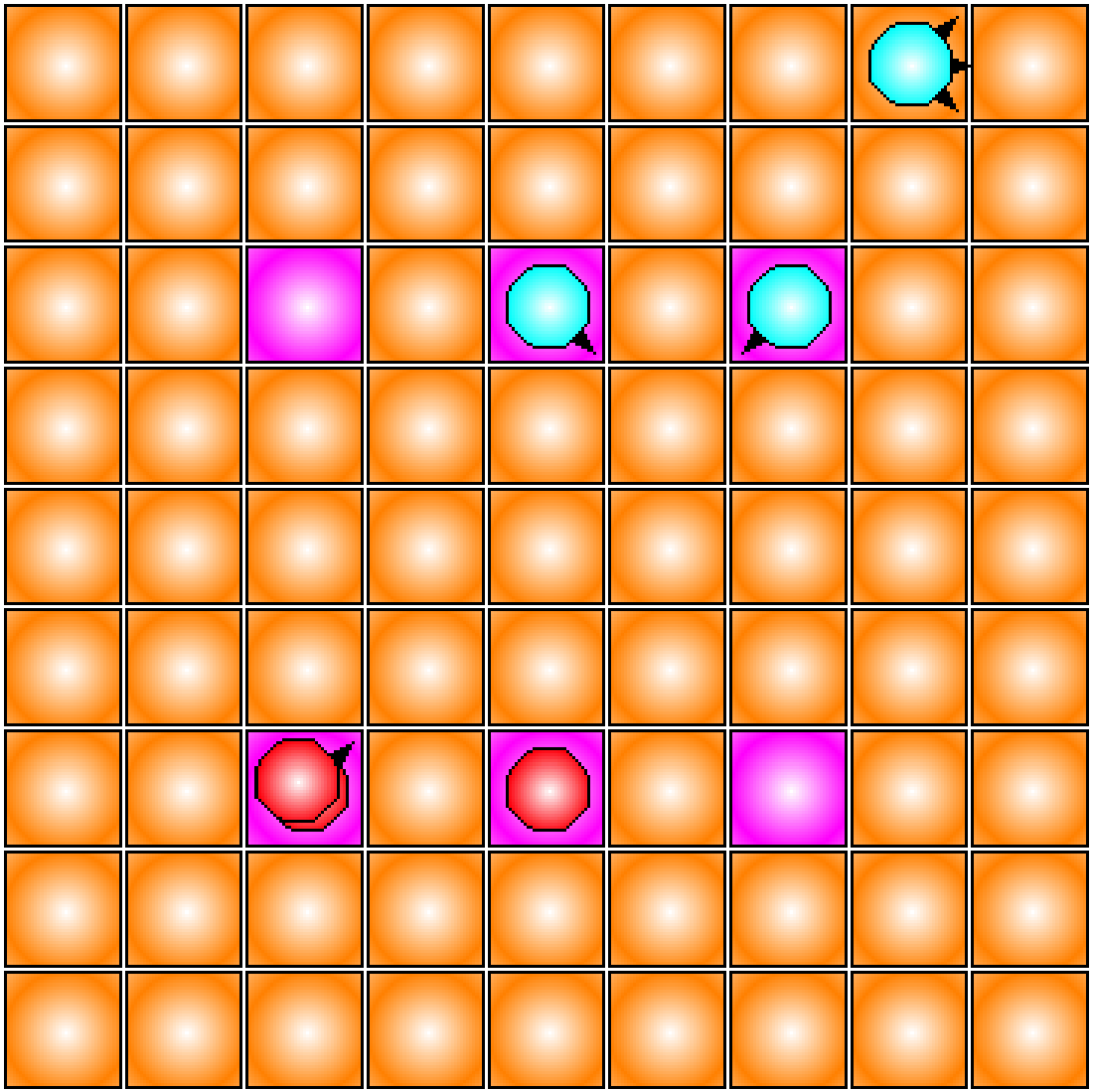
Exemple de partie OCTI



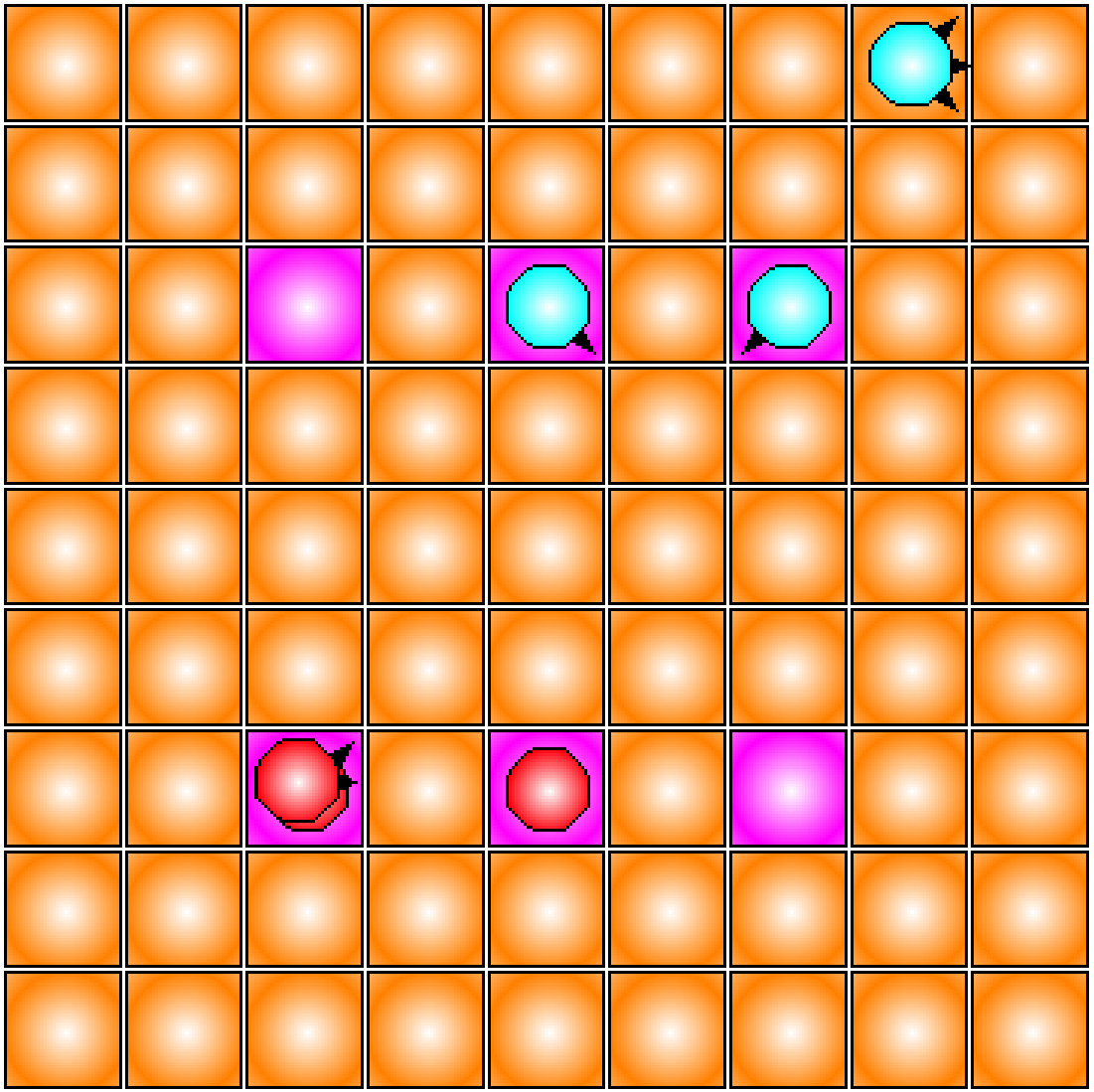
Exemple de partie OCTI



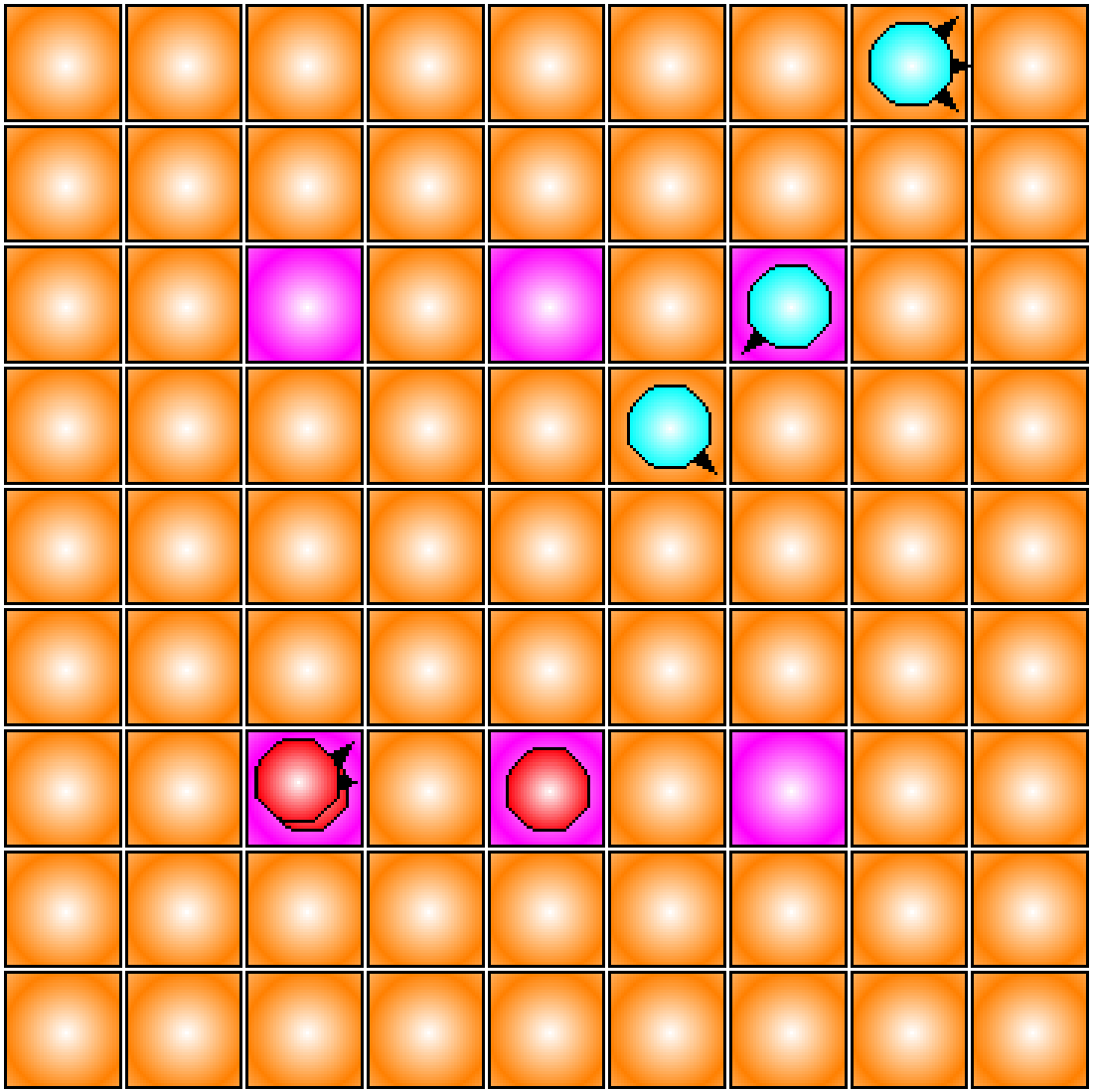
Exemple de partie OCTI



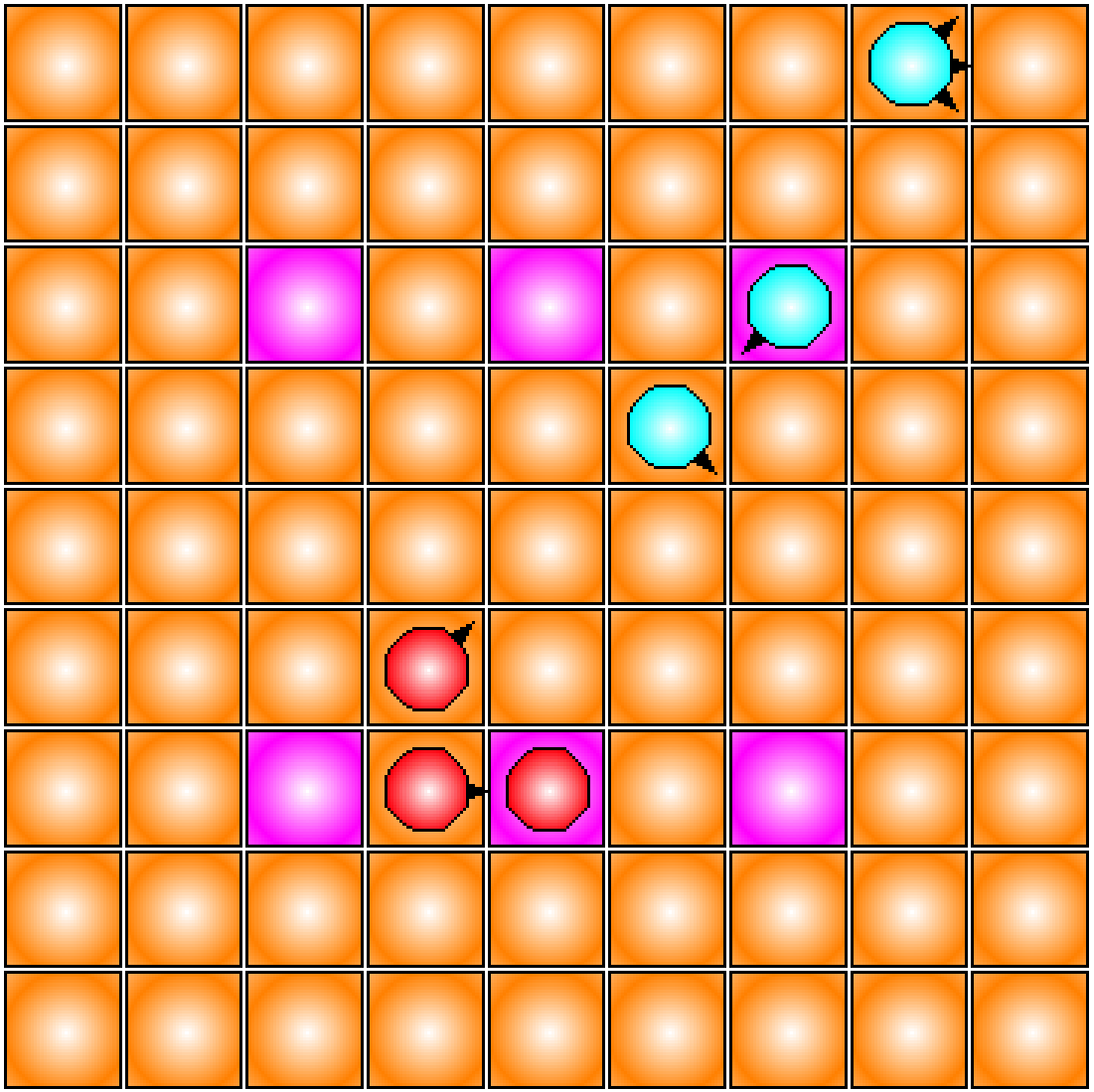
Exemple de partie OCTI



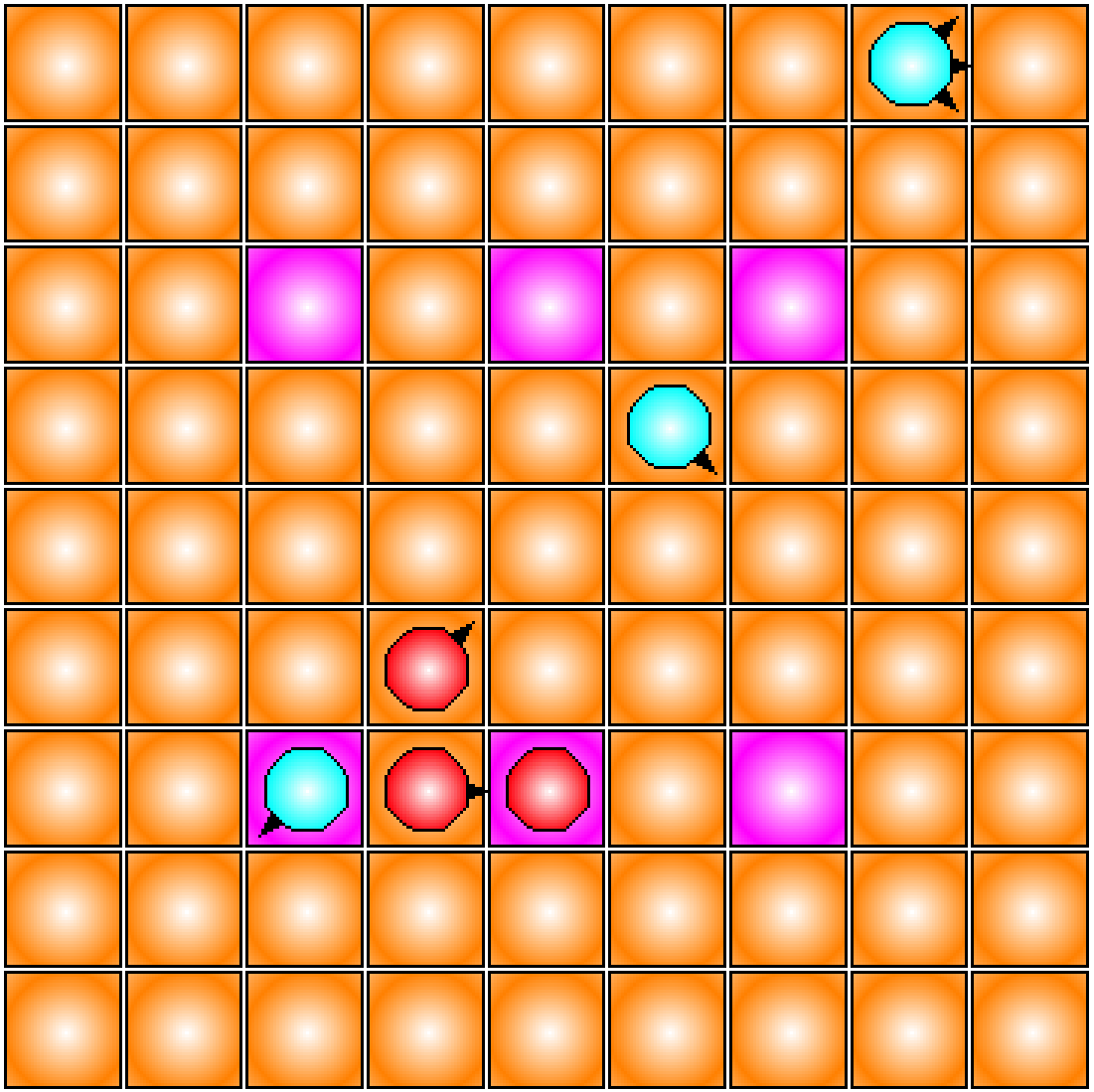
Exemple de partie OCTI



Exemple de partie OCTI



Exemple de partie OCTI



Ce qui était vrai en 1960...

En guise de conclusion

[Samuel, 1960] « Just as it was impossible to begin the discussion of game-playing machines without referring to the hoaxes of the past, it is equally unthinkable to close the discussion without a diagnosis. Programming computers to play games is but one stage in the development of an understanding of the methods which must be employed for the machine simulation of intellectual behavior. As we progress in this understanding it seems reasonable to assume that these newer techniques will be applied to real-life situations with increasing frequency, and the effort devoted to games... will decrease. Perhaps we have not yet reached this turning point, and we may still have much to learn the study of games. »

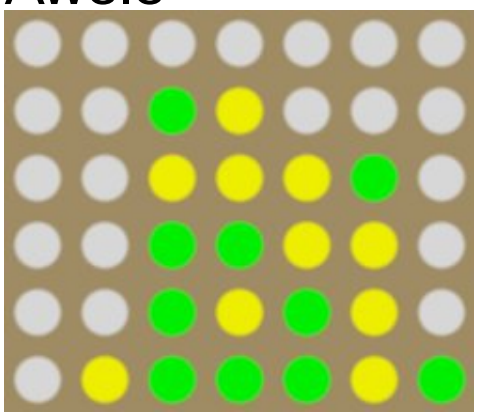
Cette citation est toujours d'actualité... 45 ans après...

Les projets IFIPS

Du pain sur la planche... Dès demain

2003 Puissance 4

2004 Awélé



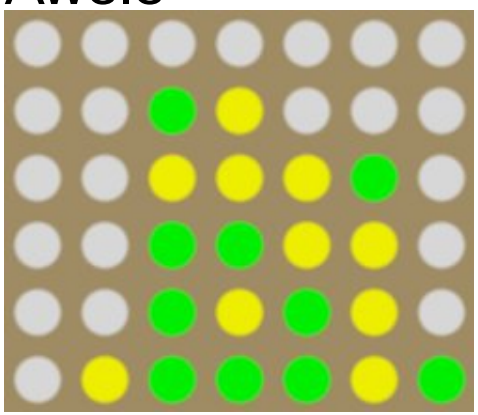
Cette année... ? Des amazones !

Les projets IFIPS

Du pain sur la planche... Dès demain

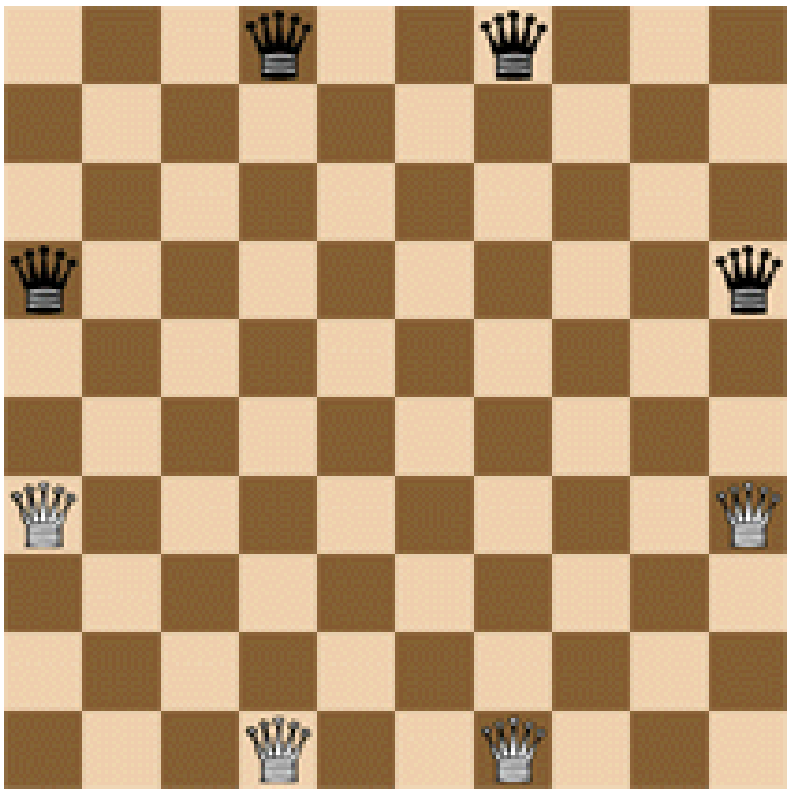
2003 Puissance 4

2004 Awélé

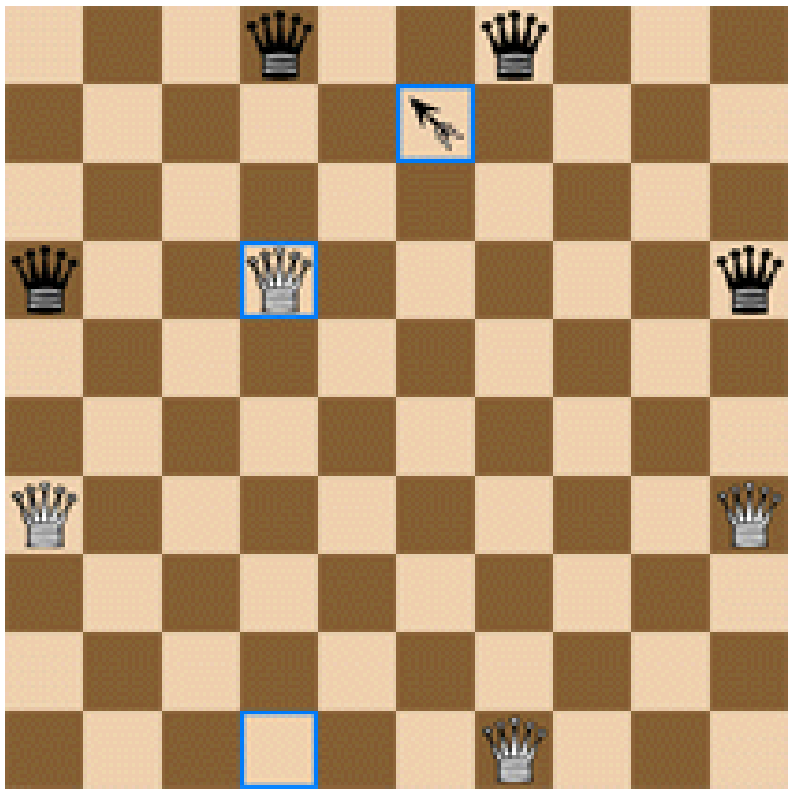


Cette année... ? **Des amazones !**

Votre projet
Quelques détails



Votre projet
Quelques détails



Votre projet
Quelques détails sur les Amazones

Projet Amazones : modalités pratiques

- Vous fonctionnerez par groupes de 4
  - Algo de recherche,
  - Heuristique,
  - Bibliothèque d'ouverture,
  - Bibliothèque de fermeture (??)
  - Interfacage avec l'arbitre
- La note :
  - 5 points sur le classement dans le tournois,
  - 10 points sur le dossier,
  - 5 points sur le code et la présentation orale.
- Ce projet n'est pas prioritaire sur les autres projets.
- Rendu