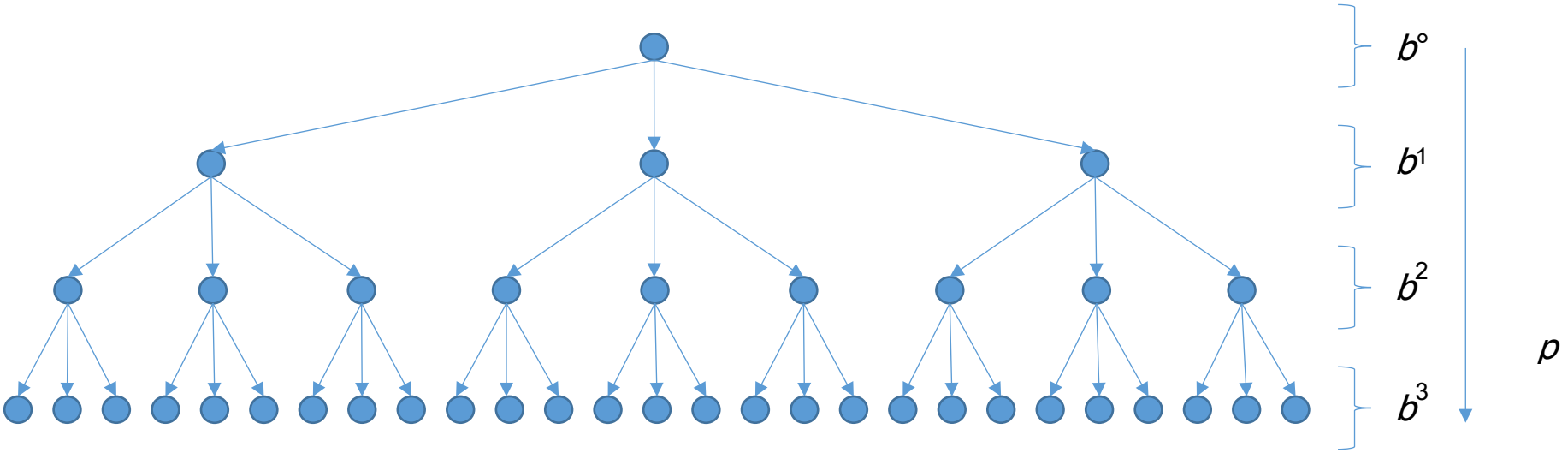
V Recherche heuristique Monte-Carlo

Problèmes inhérents à Minmax/Alpha-Beta

- Explosion combinatoire du nb de feuilles à évaluer avec *h* $O(b^p)$

b = facteur de branchement (nb moyen de coups) *p* = profondeur d'analyse

Exemple : *b*=3, *p*=3



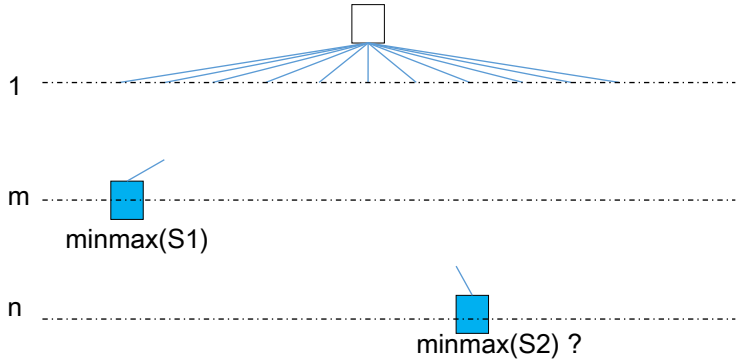
nb de feuilles = b^p

nb total de nœuds = $b^0 + b^1 + b^2 + b^3 + \ldots + b^p$ = $\sum_{k=0..p} b^k = \frac{b^{p+1}-1}{b-1}$

- **Difficultés pour trouver une bonne fonction heuristique *h***
  - Il faut modéliser des connaissances *expertes* (déclaratif/procédural)
  - Pas ou très peu réutilisables (*spécifiques* à chaque jeu)
  - Comment intégrer une méthode d'apprentissage dans *h* ?
  - De faible coût de calcul (car appelée en chaque feuille…)

On peut essayer de réduire le coût de calcul :

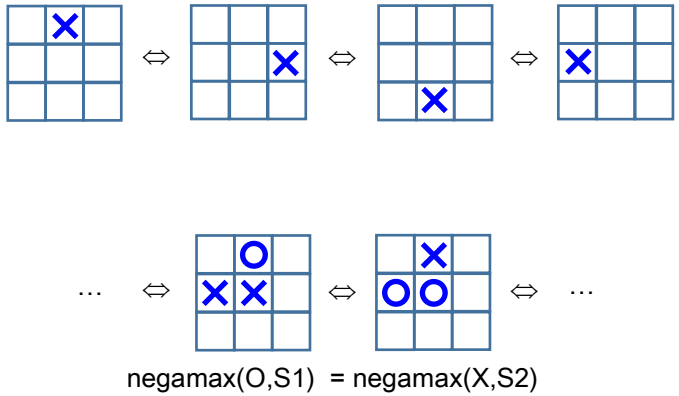
Mémorisation des situations déjà évaluées



si S2=S1et si m<n

=> réutiliser minmax(S1) pour évaluer S2

Exploitation des symétries



Un peu d'histoire …

1940-1946 Stanislaw Ulam - John Von Neuman

Caractérisation de la diffusion des neutrons dans une arme nucléaire – Proposition : remplacer le modèle basé équations différentielles par un modèle expérimental simulant une succession d'actions choisies de façon aléatoire. Si grand nombre d'expériences, les moyennes expérimentales converge vers les probabilités exactes.

1950 Rand Corporation – US Air Force

Diffusion des méthodes de Monte-Carlo *note* : « *Rand* » ne vient pas de Random mais de **R**esearch **And D**evelopment

1960 Henry P. McKean Jr

Application dans la résolution de certaines équations de méca. flu.

Début des travaux sur les algo évolutionnaires (algo génétiques)

1993-1996 Gordon –Salmond-Smith, Pierre Del Moral

Application en traitement du signal (filtrage particulaire et « Méthodes de MonteCarlo séquentielles »)

Fondations mathématiques exactes

2006 Le programme *Fuego* (basé MCTS) devient *Dan Master* au jeu de Go 9x9

2016 Le programme AlphaGo (MCTS + Machine Learning) devient 9-Dan Master au Go 19x19

2017 Généralisation de la méthode à de nombreux jeux de plateau et jeux vidéo

Objectifs de l'algorithme MCTS (Monte-Carlo Tree Search)

- Méthode approchée (minmax/alphabeta sont également approchées puisque la profondeur d'exploration est limitée du fait de l'explosion combinatoire)
- Algorithme « budget-dépendant » : dont la qualité ↑ en fonction du budget (temps ou mémoire)
- Méthode générale, indépendante du jeu particulier étudié (pas de connaissances expertes).
- Evite le coût de calcul de la fonction heuristique en la remplaçant par la simulation (avec tirage aléatoire des coups des joueurs jusqu'à une fin de partie)

MCTS se focalise sur l'analyse des nœuds les plus « intéressants » et utilise la *simulation* pour développer un coup à partir du nœud le plus intéressant. L'intérêt d'un nœud fait un compromis entre

l'exploration des nœuds encore peu analysés et

l'exploitation des nœuds totalisant un grand nombre de simulations gagnantes pour le joueur 1

Principe : étiqueter chaque nœud par un ratio **w/n** où :

w = nb de simulations se finissant par une victoire (win)

n = nb total de simulations effectuées sous ce noeud

On maintient d'autre part :

$N_{père}$ = nb total de simulations déjà faites sous le nœud père

c = paramètre d'exploration (en théorie = $\sqrt{2}$ mais éventuellement réglable)

L'algorithme MCTS propose de choisir le nœud i qui maximise la formule UCB1 (*Upper Confidence Bound)* pour l'exploration

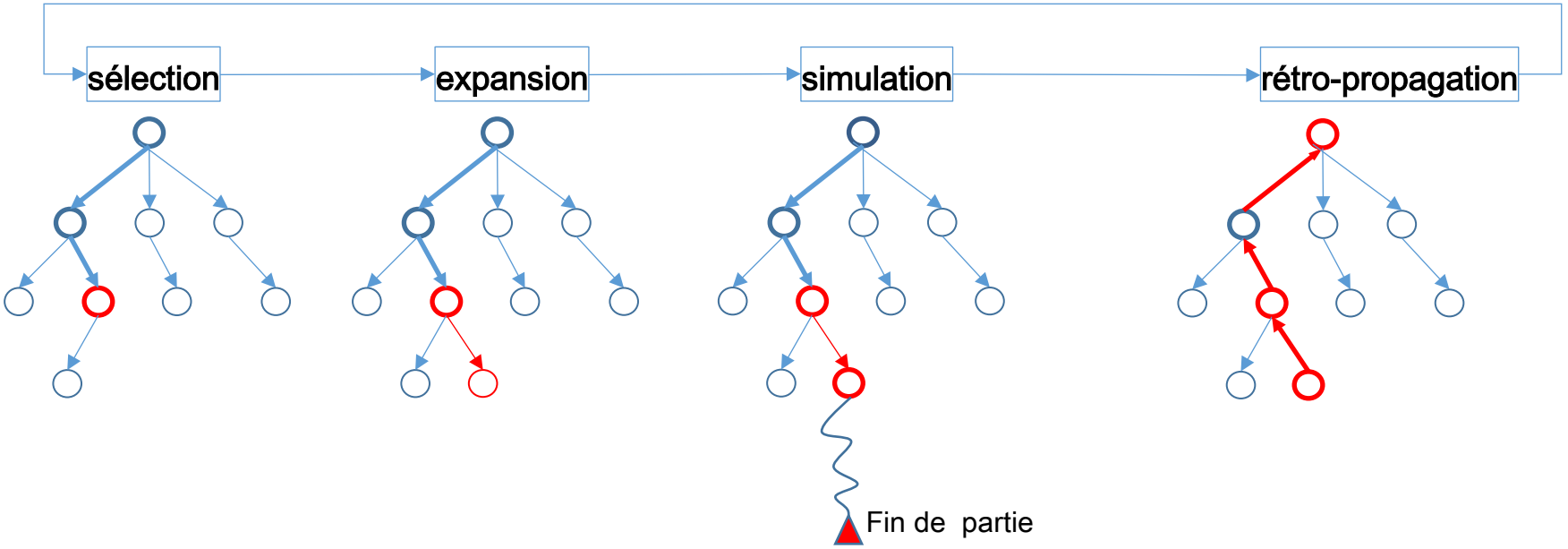
$$UCB1_i = \frac{w_i}{n_i} + \sqrt{\frac{2.log\ (N_{père})}{n_i}}$$

w_i/n_i représente l'estimation courante de l'intérêt du nœud, tenant compte des succès déjà rencontrés dans les noeuds successeurs. C'est la fonction d'**exploitation**.

$\sqrt{2}\sqrt{\frac{log\ (N)}{n_i}}$ représente la fonction d'**exploration** (élevée si le nœud i a peu de simulations).

L'algorithme boucle sur 4 phases tant que le « budget » (temps ou mémoire) le permet.

Une fois le budget épuisé, le coup reliant la raciine au meilleur nœud successeur est retourné.



Sélection : depuis le nœud racine on descend sur le nœud le +prioritaire (qui maximise UCB1).

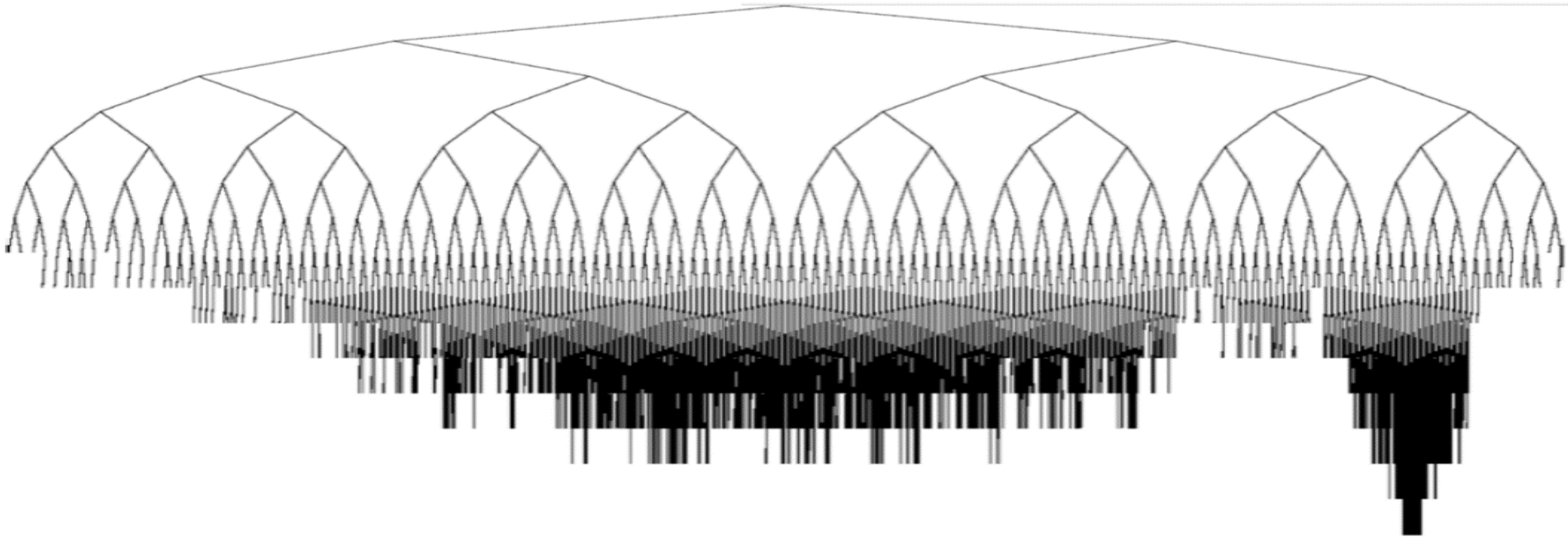
Expansion : ce nœud est développé à l'aide d'une action non encore envisagée.

Simulation : le jeu est ensuite simulé par des actions tirées aléatoirement jusqu'à une fin de partie

Le label w/n du nœud est actualisé en fonction du résultat.

Rétro-propagation : le label w/n des ascendants est actualisé sur la branche qui remonte à la racine.

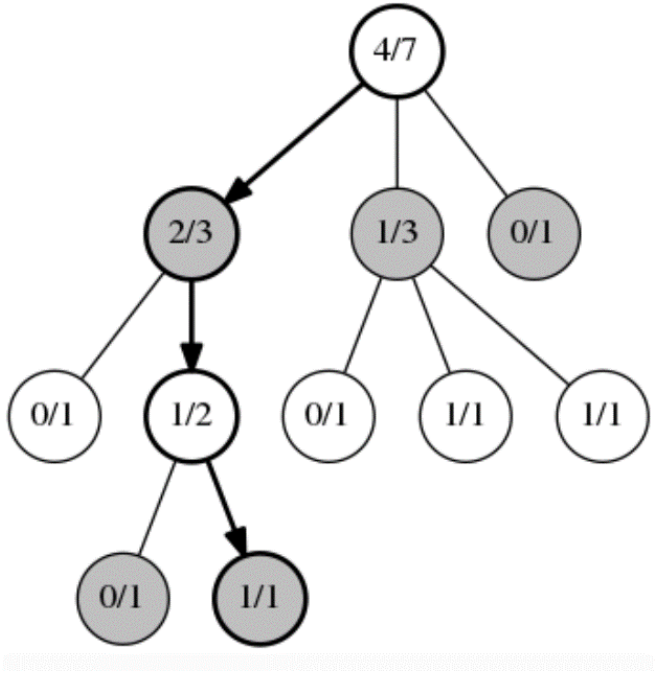
Ce développement conduit à une exploration asymétrique, exemple :



Un arbre de développement asymétrique

(tiré de *A Survey of Monte Carlo Tree Search Methods*, Cameron B. Browne*, Member et al, IEEE Transactions on CI and AI in games, vo4, n°1, march2012*)

Sélection



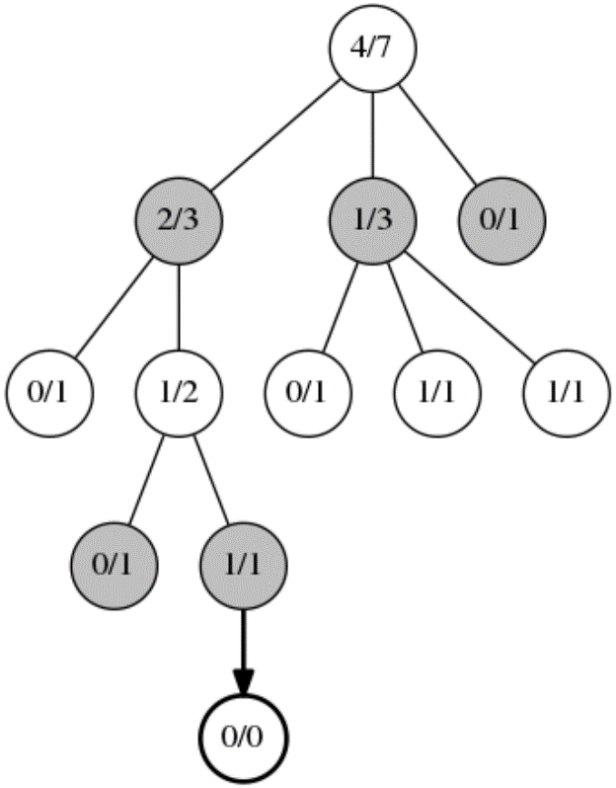
En partant du nœud racine, on sélectionne la feuille *i* la plus intéressante, qui maximise

$$UCB1_i = \frac{w_i}{n_i} + \sqrt{\frac{2.log\ (N_{père})}{n_i}}$$

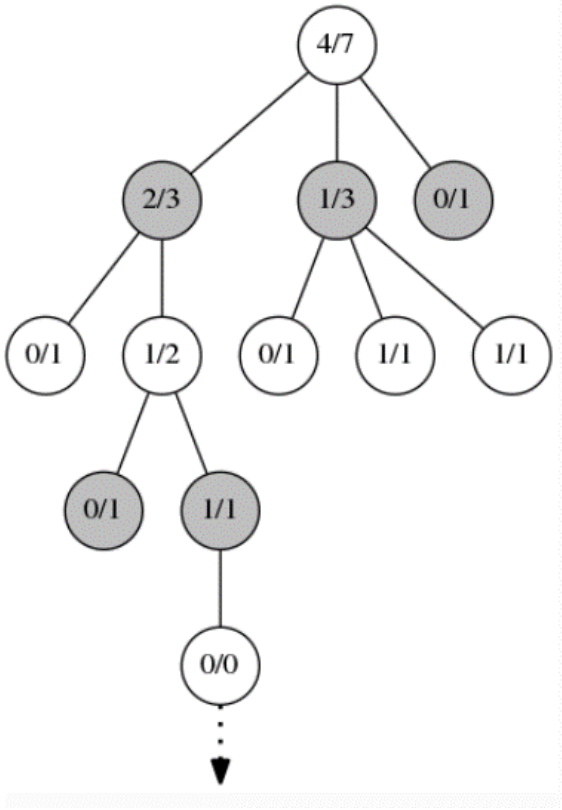
Expansion

(0/0 est prioritaire sur 1/, 1 lui-même prioritaire sur 0/1)

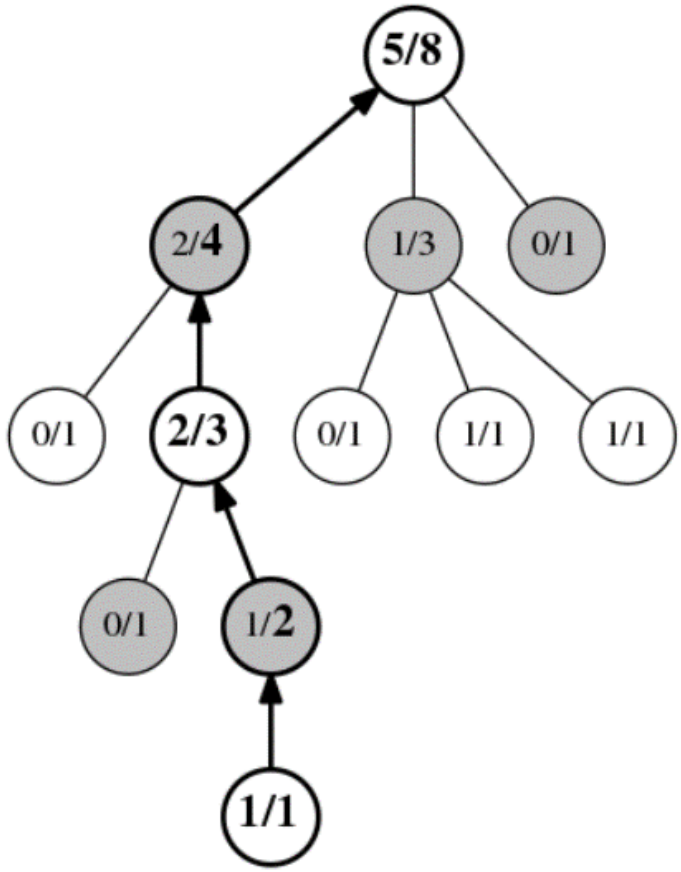
On choisit aléatoirement un coup



Simulation



Rétropropagation



Déroulement sur un exemple depuis le début

(voir fichier pdf MCTS_appliqué_à_2joueurs)

Déroulement sur un solitaire

(voir https://www.youtube.com/watch?v=UXW2yZndl7U)

On remplace la valeur (gagné => 1 perdu = 0) par une valeur quelconque.

Applications à d'autre types de jeux

- Pacman
- Fable legend
- Poker
- …

Bibliographie

- Chang, Hyeong Soo; Fu, Michael C.; Hu, Jiaqiao; Marcus, Steven I.. *"An Adaptive Sampling Algorithm for Solving Markov Decision Processes"*, Operations Research. **53**: 126–139, 2005.
- Cameron B. Browne et al, *A Survey of Monte Carlo Tree Search Methods*, , IEEE Transactions on Computational Intelligence and AI in Games, vol 4, n°1, 2012.