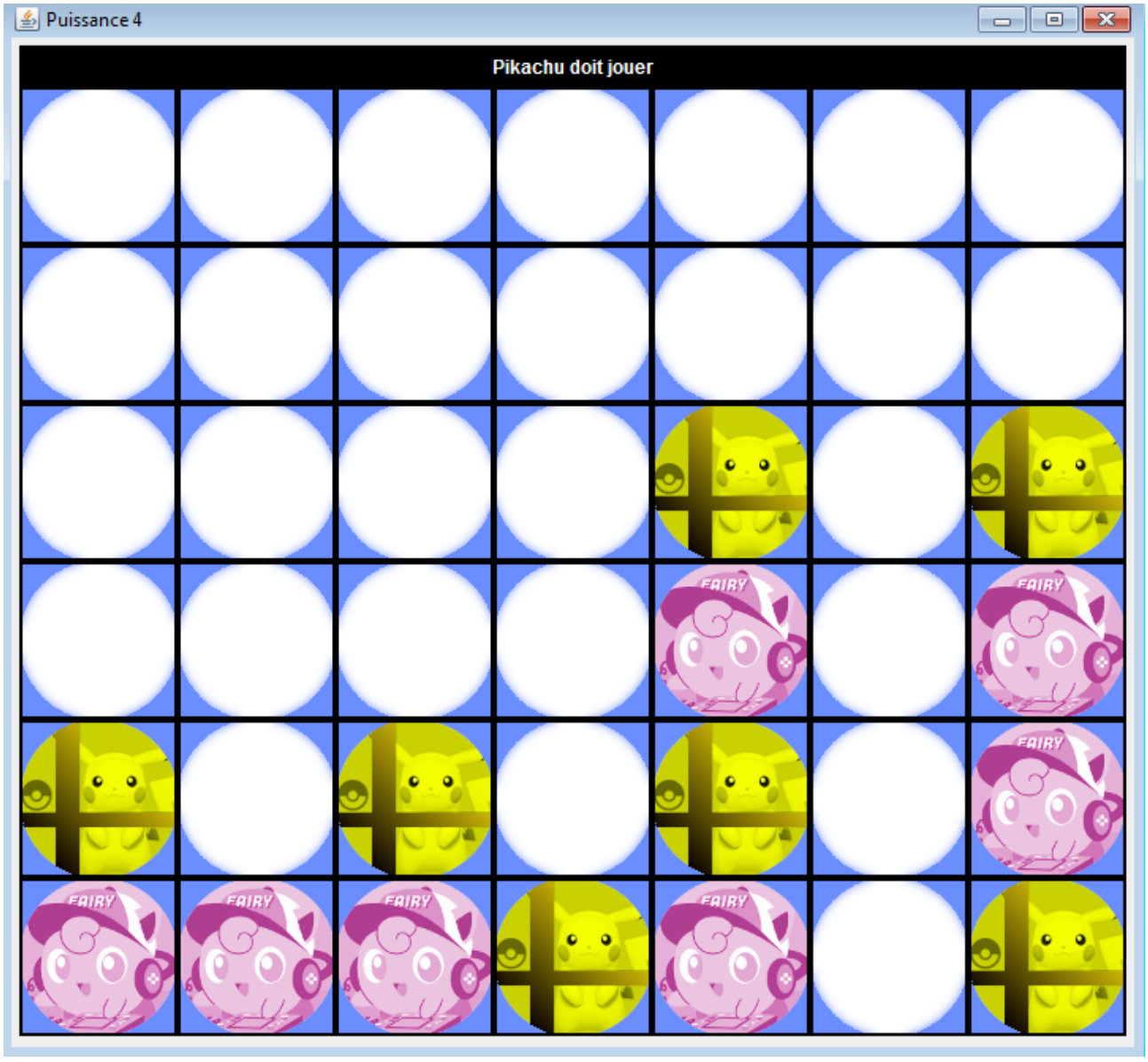
Intelligence Artificielle
Rapport de projet : Puissance 4 et MCTS

Lien Github : https://github.com/fornito2u/IA-2019



Introduction

L'algorithme « Upper Confidence bounds for Trees » (UCT) est une adaptation de l'algorithme « Upper Confidence Bound » (UCB) aux problèmes faisant intervenir des arbres, comme c'est le cas du jeu de Puissance 4.

Une position étant donnée, il est nécessaire d'effectuer un choix entre exploration et exploitation à l'aide de la formule intervenant dans UCB :

Pour un noeud donné :

$$x + C \sqrt{\left(\log(n_p)/n_f\right)}$$

Avec :
x : Pourcentage de victoire des simulations du fils (le nœud considéré).
C : Constante permettant de régler le compromis exploitation / exploration fixé à $\sqrt{2}$.
n_p : Nombre de parties du père.
n_f : Nombre de partie du fils (le nœud considéré).

En pratique, pour un nœud donné :
Si l'un des fils du nœud considéré n'est pas exploré, alors nous l'évaluons.
Sinon, nous considérons un nouveau nœud : Le fils de plus grande valeur UCT.
Nous aboutissons donc à une position non encore explorée en suivant un chemin qui rend la valeur de l'UCT maximale.

Nous évaluons cette position par un nombre fixé de simulations (marche aléatoire).
Nous mettons enfin à jour le score de chacun des noeuds par lesquels nous sommes passés en suivant le chemin qui rend la valeur de l'UCT maximale.

Le noeud de départ correspond à la position actuelle, nous mettons donc à jour, pour chaque descente, évaluation et remontée dans l'arbre,
la valeur UCT des fils de la racine.

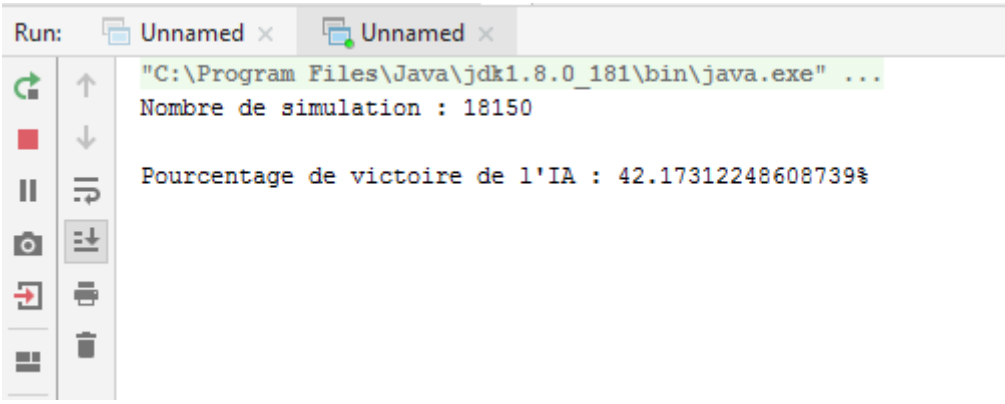
Au terme d'un nombre fixé de descentes dans l'arbre, nous effectuons l'action qui conduit au fils de la racine qui a la meilleure valeur UCT.

Question 1

- L’affichage du nombre de simulation réalisée à été affiché en créant un compteur dans la classe IA. On incrémente ce compteur à chaque fois qu’on effectue les 4 étapes de la méthode UCT (Sélection, Développement, Simulation, Backpropagation), c’est à dire à chaque tour de boucle.

- Pour calculer l’estimation de la probabilité de gagner depuis un état donné, nous utilisons simplement le calcul suivant : (nbVictoires/nbParties)*100

Avec :
. nbVictoire : Attribut d’un nœud de l’arbre définissant le nombre de simulation résultant en une victoire.
. nbParties : Attribut d’un nœud de l’arbre définissant le nombre de simulation effectué passant par le nœud.



Question 2

En toute logique, plus le temps est haut, plus le nombre simulation augmente.
Plus le nombre de simulation augmente, plus l’IA possède une estimation précise de l’efficacité de ses coups, ce qui augmente la qualité de son prochain coup.

Dans notre cas, notre IA à des difficultés à nous battre malgré l’implémentation de la méthode UCT quand il réfléchit moins de 6 secondes, soit près 18 000 simulations effectués pour chaque mouvement.

Question 3

L'amélioration des simulations via le choix d'un coup gagnant à permis d'accélérer l'algorithme. En effet, lorsque qu'un coup gagnant est possible, l'IA ne suit plus la méthode UTC et donc n'effectue plus les 4 étapes nécessaire à son fonctionnement.

Cela diminue grandement le nombre d'opérations à effectuer ce qui explique pourquoi le temps que l'IA prend à choisir un mouvement est réduit.

Question 4

Cette question ne nous concerne pas car nous avons codé en Java.
L'utilité de l'option -O3 est d'optimiser la vitesse d'exécution (et non de compilation) du programme.

Question 5

Le mode « Max » fait que l'algorithme joue le coup correspondant au nœud avec la plus grande valeur UTC parmi les enfants directes du nœud racine.
Le mode « Robuste » fait que l'algorithme joue le coup correspondant au nœud avec le plus grande de simulations parmi les enfants directes du nœud racine.

De base notre code effectue le choix du meilleur coups selon l'algorithme « max ».

Ces deux modes conduisent parfois à des coups différents en fin de partie.

L'algorithme « Max » semble plus efficace en fin de partie. En effet, puisque que le nombre de partie possible est plus faible en fin de partie (il reste moins de possibilité de jeu), la « valeur d'intérêt » hypothétique d'une seul simulation devient plus grande quand début de partie.

Selon cette logique, il devient plus intéressant de choisir un coup en fonction du nombre de victoire plutôt que du nombre de simulation, ce qui rend l'alogithme « Max » plus viable en fin de partie.

Question 6

Il y a un facteur de branchement moyen d’environ 7, car le plateau de puissance 4 classique comporte 7 colonnes, donc 7 successeurs.

Il est néanmoins possible qu’il descende en dessous de 7 en fin de partie.

Dans le pire cas, le plateau est rempli sans qu’aucun joueur n’ait gagné avant. Le plateau fait 6 par 7 cases, on a donc une profondeur maximale de 42.

Soit environ 7^42 états dans l’arbre minimax.

Un ordinateur à 3Ghz effectue environ 3 milliards d’opérations par seconde.

Il faut environ donc 7^42/3M ~ 10^26 secondes soit environ 10^18 années pour calculer le premier déplacement.