

TP numéro 2– Jeu Intelligence artificielle

D'après B. Grau

Télécharger l'archive python <http://www.ensiie.fr/~guillaume.burel/download/TPIA2019-python.tgz> .

Ces archives ont été créées à partir de <https://github.com/aimacode>, les différents algorithmes sont décrits dans Artificial Intelligence: A Modern Approach : <http://aima.cs.berkeley.edu/>

Le but du TP est d'expérimenter les algorithmes minmax et alphabeta.

Ce projet permet de jouer au Tic-tac-toe (morpion). Il développe un arbre de jeu complet. Vous allez le modifier afin de le faire s'arrêter à une profondeur donnée et appliquer une fonction d'évaluation que vous implémenterez.

Le projet est une implémentation des algorithmes donnés en cours et rappelés en fin de document, à de petites variantes près :

- la profondeur est implémentée en partant de 0 à ProfMax, et il faut fixer une ProfMax.
- le calcul du nombre de nœuds développés.

Le rendu comportera un rapport au format **pdf** plus le fichier **games.py modifié**. Il est à déposer sur exam.ensiie.fr : dans **ia-fisa-tp2** avant le **13 mars 2024**. Il doit **comporter les réponses aux différentes questions et les modifications apportées au code**.

Le fichier à modifier pour le TP : essentiellement games.py et les fichiers à utiliser : *games*.

I-Découverte du jeu

Le jeu peut être testé via l'utilisation de jupyter notebook et games.ipynb. Cela vous permet d'explorer pas à pas comment le jeu est implémenté (la partie propre au tic-tac-toe et l'algorithme minmax avec et sans alphabeta) ainsi que la programmation du jeu interactif. L'exemple fig52 sur lesquels sont effectués des tests est donné dans games.ipynb.

II- Tests du tic-tac-toe par minmax avec et sans alphabeta

Sans limite de profondeur (profondeur mise à 20) : un état terminal du tictactoe renvoie 10, -10, 0 selon que la machine gagne, perd, fait match nul (états terminaux), et l'arbre est construit jusqu'aux états terminaux.

1. Combien de nœuds sont créés à chaque tour de la partie par minmax, à quelle profondeur explore-t-on l'arbre à chaque tour ? Donner les valeurs calculées par l'algorithme et justifier une borne supérieure théorique pour ces nombres.
2. Combien de nœuds sont créés à chaque étape de la partie par alphabeta pour la même partie que précédemment. Quelles observations en tirez vous ?

PS : le nombre de nœuds développés est calculé dans [expandedNodes](#).

III- Création d'une fonction d'évaluation

Créer une fonction d'évaluation fEval, appelée quand la profondeur profMax est atteinte. NB. La fonction renvoie toujours une valeur par rapport au joueur pour qui on calcule un mouvement.

Reprendre la fonction d'évaluation vue en cours :

Nombre de lignes ouvertes seulement pour la machine – nombre de lignes ouvertes seulement pour l'adversaire. Ou choisissez en une autre à votre convenance (à décrire dans le rapport le cas échéant).

1. Rendre les modifications faites dans le code et les tests permettant de vérifier qu'il est correct : imprimer les valeurs de la fonction d'évaluation **aux feuilles**, qui doit jouer (X ou O) avec l'état associé à la profondeur atteinte.
2. Tester avec différentes profondeurs en notant les coups joués. Commenter le jeu de la machine selon des profondeurs différentes (sur des mêmes débuts de partie) - Pour donner une profondeur limite, changer la valeur de profMax.

IV – Élagage maximum

Afin de maximiser le nombre de coupes avec alphaBeta, il faut examiner en premier les nœuds conduisant vers une valeur intéressante. On peut l'évaluer en ordonnant les nœuds développés en descente selon la fonction d'évaluation.

1. Modifier alphaBeta avec l'ordonnement des nœuds et tester : comparer les nombres de nœuds développés avec et sans ordonnancement. Rendre le code, et les tests effectués (ce qui a été testé et les résultats obtenus).

V – Étendre le jeu à un damier plus grand.

1. Rendre le code du nouveau jeu et des tests
2. La fonction d'évaluation est-elle efficace dans ce nouveau contexte ? Sinon, proposez en une autre.

Algorithme DecisionMinMax (e,J)

{ Décide du meilleur coup à jouer par le joueur *J* dans la situation *e* }

Début

Pour chaque *coup* de CoupJouables(e,J) faire
 valeur[coup] = ValeurMinMax(Applique(coup,e),J,false)

Fin pour

retourner (*coup* tel que *valeur[coup]* est maximale)

Fin

Algorithme ValeurMinMax (e,J,EstUnEtatMax,pmax)

{ Calcule la valeur de *e* pour le joueur *J* selon que *e* EstUnEtatMax ou pas
et **pmax la profondeur maximale** }

Début

Si PartieGagnante(e,J) Alors retourner(+ValMax)

Sinon Si PartiePerdante(e,J) Alors retourner(-ValMax)

 Sinon Si PartieNulle(e,J) Alors retourner(0)

 Sinon

Si pmax=0 Alors retourner h(s,J)

 Sinon

 vals = vide

 Pour *s* parmi les successeurs de *e* faire

 ajouter ValeurMinMax(s,J,not(EstUnEtatMax),pmax-1) à vals

 Fin pour

 Si EstUnEtatMax

 Alors retourner(maximum de vals)

 Sinon retourner(minimum de vals)

 Fin si

 Fin si

 Fin si

Fin si

Fin

Algorithme alpha-beta**Algorithme DecisionAlphaBeta (e,J)**

{ Décide du meilleur coup à jouer par le joueur *J* dans la situation *e* }

Début

alpha = -ValMax

Pour chaque *coup* de CoupJouables(e,J) faire

val = ValeurAlphaBeta(Applique(coup,e),J,alpha,+MaxVal,false)

 Si (*val*>alpha)

 Alors

action = *coup*

 alpha = *val*

 Fin si

Fin pour

retourner *action*

Fin

```

Algorithme ValeurAlphaBeta (e,J,alpha,beta,EstUnEtatMax,pmax)
{ Calcule la valeur de e pour le joueur J selon que e EstUnEtatMax ou pas
  et pmax la profondeur maximale }
Début
  Si PartieGagnante(e,J) Alors retourner(+ValMax)
  Sinon Si PartiePerdante(e,J) Alors retourner(-ValMax)
    Sinon Si PartieNulle(e,J) Alors retourner(0)
      Sinon
        Si pmax=0 Alors retourner h(s,J)
        Sinon
          Si EstUnEtatMax
            Pour s parmi les successeurs de e faire
              alpha =
MAX(alpha,ValeurAlphaBeta(s,J,alpha,beta,not(EstUnEtatMax),pmax-1)
              Si alpha >= beta
                Alors retourner(alpha) /* coupe beta */
              Fin si
            Fin pour
            retourner(alpha)
          Sinon
            Pour s parmi les successeurs de e faire
              beta =
MIN(beta,ValeurAlphaBeta(s,J,alpha,beta,not(EstUnEtatMax),pmax-1)
              Si beta <= alpha
                Alors retourner(beta) /* coupe alpha */
              Fin si
            Fin pour
            retourner(beta)
          Fin si
        Fin si
      Fin si
    Fin si
  Fin si
Fin

```