TP2 - Le Tic-Tac-Toe

10 points

Octobre 2019

1 Introduction

Dans ce TP, vous allez implémenter plusieurs fonctions liées au jeu de Tic-Tac-Toe qui vous est fourni. Chaque exercice aura pour but de vous faire utiliser une des structures de données vues dans le module 2 et 3 pour améliorer notre jeu de Tic-Tac-Toe.

Veuillez bien utiliser les structures demandées. D'autres approches pourraient être plus simples, mais nous voulons vous faire travailler la matière vue en classe.

Le jeu est présentement fonctionnel. Si vous exécutez le code, vous pouvez jouer une partie contre l'ordinateur. .

2 Reconnaissance dans le code

Vous allez prendre le jeu de Tic-Tac-Toe qui a déjà été implémenté pour y ajouter des fonctions qui amélioreront l'expérience de l'usager.

Vous pouvez trouver le code du Tic-Tac-Toe dans le répertoire du TP, sous le dossier "code".

Le répertoire contient 3 fichiers :

- Main.py

  Fichier qui créer l'objet Tictactoe et qui débute une partie. Il n'y aura probablement rien à faire ici.

- TicTacToeGame.py

C'est ici que tout le jeu prend place.

Le constructeur (init) crée l'array qui nous servira à garder l'état du plateau de jeu en mémoire. Il est initialisé avec des " " au départ, puisque aucun coup n'est encore joué. On crée également un AI contre qui jouer (description dans la classe AI.py).

La fonction "startNewGame()" commence une nouvelle partie. Elle demande d'abord l'ordre de jeu (i.e si le joueur veut jouer en première ou en deuxième). Puis elle joue les tours (avec la fonction "playTurn()") jusqu'à ce que la partie se termine.

La fonction "playTurn()" joue un tour (donc 1 coup du joueur et un coup de l'AI). On respecte l'ordre de jeu donné précédemment et on vérifie entre les coups si quelqu'un a gagné, pour ne pas faire jouer des coups inutiles.

La fonction "playUserMove()" demande à l'utilisateur une case à jouer jusqu'à ce qu'il donne une case valide (chiffre de 1 à 9 qui n'est pas couramment rempli dans la grille). Quand on a une entrée correcte, on la met sur le plateau de jeu.

La fonction "printBoard()" imprime l'état courant du plateau à la console

La fonction "playAiTurn()" joue le tour de l'ordinateur. Pour l'instant elle appelle la fonction "playatrandom()" qui retourne un coup au hasard parmi les coups disponibles.

La fonction "checkgameover()" regarde chaque colonne, rangée et diagonal de l'état courant du plateau et retourne True s'il y a un tic-tac-toe, ou si le plateau est plein. Elle retourne False sinon.

- AI.py

  Contiens pour l'instant seulement la fonction "playatrandom(board)" qui prend un état du plateau de jeu et qui retourne un coup au hasard parmi les coups légaux.

3 AI plus intéressant (Arbres)

Pour l'instant les parties ne sont pas très intéressantes parce qu'il n'y a aucun défi. En effet, vu que l'AI joue au hasard, il n'a pratiquement aucune chance de gagner.

On veut un AI qui soit meilleur pour donner un défi au joueur. Pour ce faire vous allez coder la fonction "playGoodMove(board)" qui prend l'état actuel du plateau de jeu et qui retourne le meilleur prochain coup. Votre AI ne perdra plus!

Vous allez créer un arbre pour vous aider à choisir le coup optimal. Vous devriez créer une fonction qui bâtit un arbre en fonction de l'état courant du plateau. La racine de cet arbre sera l'état courant du plateau. Ensuite, chaque enfant de la racine sera un des états du plateau possible après un coup. Et ainsi de suite, jusqu'à atteindre une feuille, qui correspond à une fin de partie (soit une victoire ou une partie nulle). On vous conseille de vous définir une classe "Noeud" pour créer votre arbre.

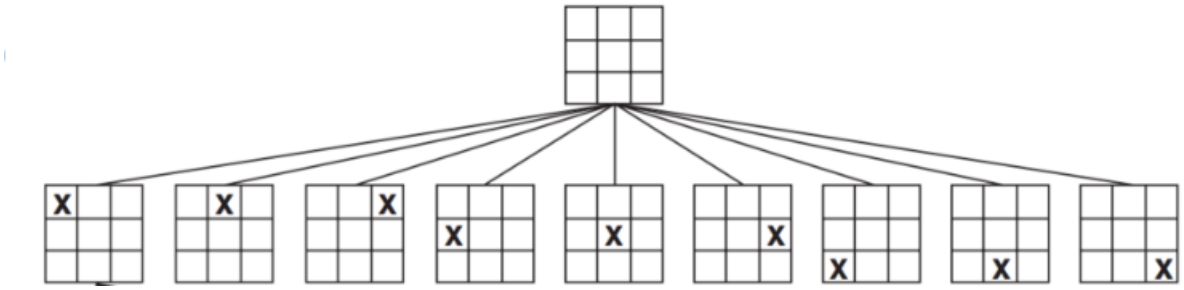

Figure 1: État de l'arbre bati avec la plateau vide après un coup

Après avoir créé cet arbre, nous allons trouver le meilleur coup possible. Pour ce faire, nous allons utiliser un algorithme appelé "miniMax". Batissez tout d'abord l'arbre, avec en racine l'état actuel du plateau.

Par la suite, parcourez l'arbre de façon récursive. Quand vous atteignez une feuille, vous allez retourner 1 si c'est l'AI qui a gagné la partie, -1 si c'est le joueur qui a gagné la partie ou 0 si la partie est nulle.

Le noeud parent recevra une valeur de chaque enfant. Ce noeud prendra parmi les valeurs reçues la meilleure valeur pour lui et la retournera. Si ce

noeud simule présentement le coup de l'AI, le meilleur valeur est la valeur maximum retourner par les enfants. Au contraire, si ce noeud simule présentement un coup du joueur, la meilleure valeur sera la valeur minimum retournée par les enfants.

Les valeurs vont ainsi remonter jusqu'à la racine. Celle-ci choisira le coup qui lui donne la valeur maximale et retournera le numéro de la case qui correspond à ce coup.

Les détails de l'implémentation sont à votre discrétion, tant que vous créez un arbre et que vous utilisiez l'algorithme "minimax" tel qu'indiqué.

Voici une illustration de l'algorithme sur un plateau auquel il reste 3 coups possibles.

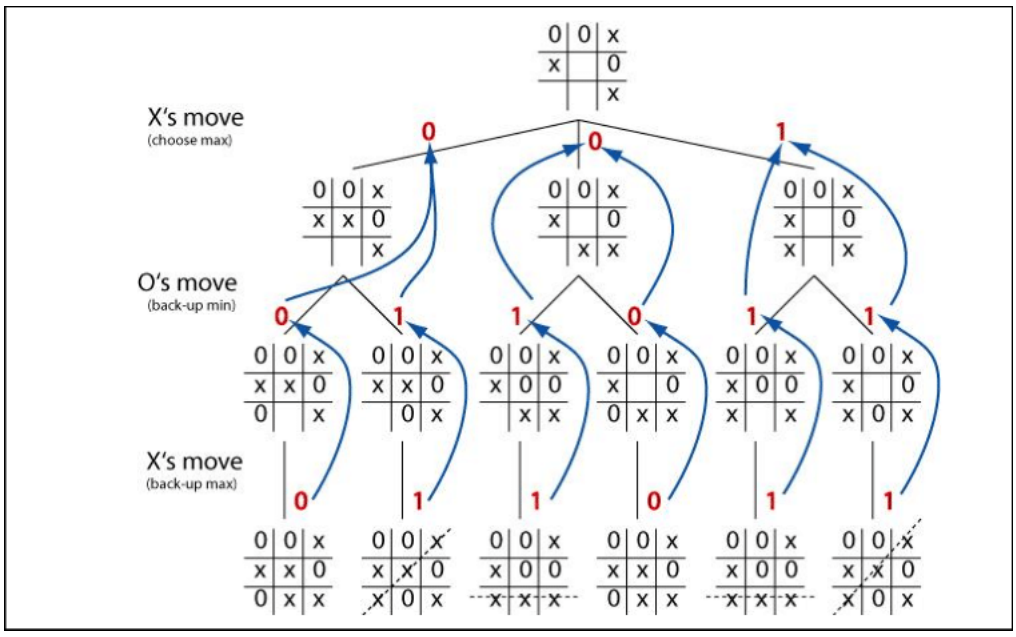

Figure 2: Minimax pour 3 coups

Une fois votre fonction terminée, remplacez l'appel courant à la fonction "playatrandom()" par un appel à la fonction "playgoodmove()" dans la fonction "playaiturn()" de TicTacToe.py

4 Fonction Undo (Stack)

On veut ajouter une fonction de Undo au jeu. À chaque nouveau tour, l'utilisateur pourra taper "u" à la place d'un numéro pour effacer son dernier coup (et celui de l'AI) et recommencer à l'état précédent du plateau.

Pour ce faire , vous allez implémenter un Stack. Créer tout d'abord une classe Stack, qui devra contenir les fonctions "push()" et "pop()".

Instanciez ensuite un Stack dans l'objet "TicTacToeGame". À chaque fois qu'un coup est joué, ajouté le au Stack.

Créer ensuite la fonction "Undo()", qui retrouvera l'état du plateau juste avant le dernier coup du joueur et qui le mettra comme état courant du plateau. Vous devriez également afficher le plateau de nouveau. Le joueur devrait pouvoir faire plusieurs "undo" de suite sans problème pour retourner plusieurs coups dans le passé.

Finalement, arranger le tout pour détecter quand le joueur tape "u" et faire le traitement approprié.

5 Fonction Replay (Liste chainée)

On veut ajouter une fonction de "replay" à notre jeu de Tictactoe.

Créez tout d'abord une structure "replay" qui devra contenir les fonctions "back()", qui affiche le coup qui précède celui qui vient d'être affiché, et "next()", qui affiche le coup qui succède celui qui vient d'être affiché.

On vous laisse décider des détails de l'implémentation de cette structure, mais elle doit respecter deux contraintes : la création de la "replay" doit être O(n) et les fonction "back()" et "next()" doivent être O(1).

Quand une partie se termine, demandez à l'utilisateur s' il désire revoir sa partie. S' il répond par l'affirmative, afficher le premier coup de la partie. Donnez-lui ensuite la possibilité de soit afficher le prochain coup (next()) ou

afficher le coup précédent (back()). Si le coup suivant ou précédent n'existe pas ou que l'utilisateur tape "q", quitter la partie.

Ajouter une "replay" à la classe "TicTacToeGame". À chaque coup joué, ajoutez-le au "replay". Assurez-vous de prendre en compte les "undo" du joueur. Les coups qui sont "undo" devraient être remplacés dans votre "replay" par les nouveaux coups du joueur.

6 Modalité de remise

Vous aurez deux semaines après l'intra pour compléter le travail. La date de remise sera donc le 31 Octobre 2019 avant 11h55 sur StudiUM.

Le travail doit être remis en équipe d'au plus deux personnes.

Dans la mesure du possible, envoyez vos questions sur le forum du cours pour en faire bénéficier au plus grand nombre.

Votre remise consistera d'une archive "Tp2.zip" (ou autre type d'archive). Elle contiendra 5 fichiers: "Main.py", "TicTacToeGame.py", "AI.py", "Stack.py" et "Queue.py".

Votre travail sera évalué sur sa fonctionnalité (90%) et la clarté du code (10%).