Projet - Agent intelligent pour le jeu Quoridor

Code et Agent à remettre le 3 Avril 2022 sur Moodle (avant minuit) pour tous les groupes.

Rapport à remettre le 21 Avril 2022 sur Moodle (avant minuit) pour tous les groupes.

Consignes (en bref)

- Le projet doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de votre soumission, donnez votre rapport au format **PDF** (matricule1_matricule2_Projet.pdf)
- Lors de votre soumission, votre code `my_player.py` et ses dépendances (**requirements.txt**) doivent être au format **ZIP** (matricule1_matricule2_Projet.zip). Plus de détails sont fournis dans le sujet, notamment pour les dépendances et le fichier **requirements.txt**.
- Indiquez vos nom et matricules dans le fichier PDF et en commentaires en haut du fichier de code soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le TP.

1 Introduction

Munis de vos nouvelles expertises dans le domaine de l'intelligence artificielle, vous devez maintenant implémenter un agent qui puisse jouer efficacement au jeu Quoridor. Quoridor est un jeu de stratégie au tour par tour, en un contre un dont le principe est simple : arriver de l'autre coté du plateau avant son adversaire.

Le jeu se déroule sur un damier 9×9. Chaque joueur commence avec un pion placé sur un des côtés du damier (ligne 1 et 9, colonne 5) et doit se diriger vers le côté opposé. Chaque joueur possède 10 *murs* de longueur 2 qu'il peut décider de placer sur le damier afin d'empêcher la progression du pion adverse. À chaque tour, chaque joueur peut ainsi choisir entre 2 actions différentes :

1. Déplacer son pion de 1 case suivant l'une des 4 directions cardinales.
2. Placer l'un de ses murs où il le souhaite, horizontalement ou verticalement.

⚠Un mur ne peut être placé dès lors qu'il empêche le pion adverse de rejoindre le côté opposé du plateau par quelconque chemin. Il doit ainsi toujours être possible pour un joueur d'atteindre sa destination.

À cela s'ajoute une dernière exception : dans le cas où les deux pions se font face, un joueur peut sauter au dessus de son adversaire pour aller à la case suivante sauf si un mur a été placé entre le pion sauté et la case cible. Dans ce cas précis, le joueur a le droit exceptionnellement à un mouvement en diagonal.

Nous vous conseillons de regarder la courte vidéo explicative à l'adresse suivante afin de vous familiariser avec les règles du jeu : https://www.youtube.com/watch?v=6ISruhN0Hc0. Vous pouvez aussi vous entrainer à jouer à cette addresse : http://quoridor.di.uoa.gr. Pour les étudiants en présence, une version plateau du jeu vous sera disponible lors du laboratoire.

2 Énoncé du projet

Pour ce projet vous participerez par **équipe de deux** à un tournoi Challonge dont le but est de développer un agent automatique qui puisse jouer le mieux possible à Quoridor. Vous êtes libre d'utiliser les méthodes et les bibliothèques de votre choix. Les seules contraintes sont le langage de programmation (**Vous devez implémenter votre agent en python 3**), la possibilité de faire jouer votre agent (votre code compile) et la limite de temps : chaque agent a un budget temps de **20 minutes** à répartir sur l'entièreté de ses actions. Ainsi, outre une stratégie standard d'allouer un même temps d'exécution pour chaque action, il est tout a fait permis à un agent d'utiliser 15 minutes pour effectuer son premier coup, et d'utiliser les 5 minutes restantes pour la réalisation de tous les autres coups.

Votre code devra être soumis sur Moddle sous format **ZIP**. Après avoir reçu tout les agents, nous les feront jouer les un contre les autres. Le tournoi sera divisé en deux phases :

1. Phase de qualification : Les différents agents seront séparés dans plusieurs poules. Tout les joueurs dans la poule se rencontrent et chaque rencontre est composé de 5 parties. A l'issue, les joueurs sont classés sur leur nombre de victoires (et le cumul de leur score si égalité). Seul une partie des meilleurs joueurs par poule seront qualifiés pour la phase suivante (un ratio d'environ 50% de la poule).
2. Phase éliminatoire : Les joueurs s'affrontent les uns contre les autres en élimination directe jusqu'à la finale et la petite finale.

Sur les cinq manches d'une partie, la couleur blanche/noire sera attribuée deux fois par joueur. La répartition de la dernière manche est aléatoire. Vous pourrez suivre les résultats de vos agents en live. Le tournoi aura lieu quelques jours après la remise concours du **3 Avril 2022**. Le lien du tournoi ainsi que la configuration finale du tournoi vous sera fourni ultérieurement.

3 Rapport

En plus d'implémenter votre agent, vous devez rédiger un rapport qui détaille votre stratégie de recherche et vos choix de conception Étant donné que vous êtes libre d'implémenter la solution de votre choix qu'elle soit inspirée de concepts vus en cours, ou de vos connaissances personnelles, nous attendons une explication claire et détaillée de votre solution. Précisément, votre rapport doit contenir au minimum les informations suivantes :

1. Titre du projet
2. Nom d'équipe sur Challonge, ainsi que la liste des membres de l'équipe (nom complet et matricule)
3. Méthodologie : Explication du fonctionnement de votre agent, des choix de conceptions faits (principes de l'heuristique utilisée, la stratégie choisie, prise en compte du time-out, spécificités propres de votre agent, utilisation de parallélisme, gestion des 20 minutes allouées, etc.)
4. Résultats et Évolution de l'agent : Reporter l'amélioration de votre agent en le testant contre vos versions précédentes et les implémentations qui vous sont fournies.
5. Discussion : Discutez des avantages et limites de votre agent final.
6. Référence (si applicable).

Le rapport ne doit pas dépasser **5 pages** et doit être rédigé sur une ou deux colonnes à simple interligne, avec une police de caractère 10 ou plus (des pages supplémentaires pour les références et le contenu bibliographique sont autorisées, ainsi que pour la page de garde). Vous êtes libre de structurer le rapport comme vous le souhaitez tant que vous incluez les éléments mentionnés précédemment.

4 Ressources fournies

Une implémentation Python du jeu Quoridor est fournie sur Moodle. Elle implémente l'ensemble des caractéristiques du jeu, et permet à un utilisateur de jouer contre un ordinateur via une interface graphique ou de faire s'affronter deux ordinateurs.

Avant toute chose, si vous n'êtes pas familié avec la programation orientée objet en python, vous pouvez consulter cette fiche explicative.

Les fichiers fournis sont :

- `gui.py` : contient le code pour afficher l'interface graphique. **Vous n'avez pas comprendre ce fichier, ni à le modifier.**
- `game.py` : contient l'ensemble des fonctions permettant de lancer une partie et de l'enregistrer. **Vous n'avez pas à modifier ce fichier.** Les details concernant les commandes pour lancer une partie vous sont fournis plus bas.
- `quoridor.py` : implémente toute la logique du jeu Quoridor à l'aide de différentes classes et fonctions. **Vous n'avez pas à modifier ce fichier.** Voici cependant une description du contenu du fichier pour vous aider à le comprendre :

1. `class Board` : contient tout l'information relative à une partie en cours. Cette classe contient :

(a) Un constructeur :

```
    def __init__(self, percepts=None):
        self.size = 9
        self.rows = self.size
        self.cols = self.size
        self.starting_walls = 10
        self.pawns = [(0, 4), (8, 4)]
        self.goals = [8, 0]
        self.nb_walls = [self.starting_walls, self.starting_walls]
        ...

```

(b) Des fonctions de vérifications des règles :

```
    def can_move_here(self, i, j, player)
    def is_pawn_move_ok(self, former_pos, new_pos, opponent_pos)
    def paths_exist(self)
    def is_wall_possible_here(self, pos, is_horiz)
    def get_actions(self, player)
    def is_action_valid(self, action, player)
    ...

```

(c) Des fonctions d'action :

```
    def add_wall(self, pos, is_horiz, player)
    def move_pawn(self, new_pos, player)
    def play_action(self, action, player)
    ...

```

(d) Diverses fonctions utilitaires :

```
    def pretty_print(self)
    def clone(self)
    def get_shortest_path(self, player)
    def min_steps_before_victory(self, player)
    def is_finished(self)
    def get_score(self, player=PLAYER1)
    def dict_to_board(dictio)
    def load_percepts(csvfile)
    ...
```

2. `class Agent` : Chaque agent fonctionne sur un serveur XML-RPC à part. Il ne vous est pas demandé de comprendre le fonctionnement de ce type de serveur mais sachez qu'il sert à exécuter un script python de manière totalement isolé et sur lequel il est possible d'effectuer des requêtes. Cela permet d'utiliser différents blocs de code (en l'occurrence ici vos agents) sans avoir à regrouper tout ces différents blocs de code ensemble. Cette classe est une classe abstraite. Lorsque vous allez implémenter votre agent, il devra respecter les exigences de la classe `Agent`, et **devra au minimum contenir la fonction :** `play()` dont les arguments sont explicités dans `quoridor.py`. Cette fonction prend en entrée l'état actuel du jeu, et doit renvoyer la prochaine action à effectuer.

— `my_player.py` : votre agent. **Vous devez modifier ce fichier.**
— `random_player.py` : un agent aléatoire si vous voulez gagner une partie facilement. ;)
— `greedy_player.py` : un agent glouton en guise de baseline.

Pour lancer une partie il faut au préalable lancer un serveur XML-RPC pour chaque agent que vous souhaitez utiliser à l'aide de la commande suivante :

```
python path/to/agent.py --bind (ou -b) ADDRESS --port (ou -p) PORT
```

en remplaçant l'adresse `ADRESS` par votre adresse locale (‘`localhost`’) et le port local `PORT` sur lequel vous voulez que le serveur communique. Par exemple :

```
python path/to/random_player.py --bind localhost --port 8000
python path/to/greedy_player.py --bind localhost --port 8080
```

⚠**Chaque agent doit être sur un port différent (p.e., 8000, et 8080).**

Une fois le(s) serveur(s) démarré(s) sur leur thread respectif, pour lancer une partie, il vous suffit de lancer la commande suivante :

```
python game.py AGENT1 AGENT2 (--time SECONDS) (--verbose) (--no-gui)
```

en remplaçant `AGENT1` et `AGENT2` par une adresse locale (`http://localhost:PORT` en remplaçant `PORT` par le port du serveurs XML-RPC de l'agent que vous voulez confronter) ou par ‘`human`’ pour jouer vous même. Par exemple :

```
python game.py http://localhost:8000 http://localhost:8080 --time 300
ou
python game.py http://localhost:8080 human --time 300
```

Pour obtenir la documentation complète de `game.py`, exécutez la commande : '`game.py --help`'.

⚠**Utilisateurs de Windows** : si vous utilisez l'application Ubuntu ("Windows Subsystem for Linux (WSL)") pour lancer la partie, il se peut que vous n'arriviez pas à afficher l'interface graphique et que vous ayez un message d'erreur "`WARNING: Unable to load GUI, falling back to console.`". Si c'est le cas, vous pouvez suivre la procédure suivante :

1. Installer un serveur X11 pour windows, tel que Xming à l'adresse suivante : https://sourceforge.net/projects/xming/.
2. Ajouter "`export DISPLAY=:0;`" à votre fichier `~/.bashrc`.
3. Fermer et ré-ouvrir Ubuntu.
4. Démarrer *Xming* (simplement en cliquant sur l'icône dans le menu démarrer de windows).
5. Tester l'installation de *Xming* en lançant '`xeyes`' ou '`xcalc`'. Si ça marche, vous pouvez réessayer la commande `python game.py`.

En cas de problèmes, n'hésitez pas à communiquer avec votre chargé de laboratoire.

5 Environnement virtuel Conda

De la même façon qu'au Devoir1, il est important pour la correction de pouvoir aisément reproduire l'environnement sous lequel vous travaillez. Il en va de même pour le travail collaboratif et surtout pour l'organisation du tournoi.

⚠**L'ensemble des commandes suivantes sont à réaliser avec la console Conda. Se référer à la fiche pratique Conda pour l'installation.**

Création d'un environnement Conda

(base) conda create --name projet python=3.8

*Une validation est demandée par la saisie du caractère **y**.*

Activation de l'environnement virtuel

(base) conda activate projet

*La console Conda devrait afficher **(projet)** comme préfix.*

Lancement d'un agent

(projet) python path/to/random_player.py --bind localhost --port 8000

L'agent random devrait être en écoute sur le port 8000 en attente du début de partie face à un autre agent.

Installation d'une dépendance

Une fois l'environnement actif dans la console Anaconda, on peut simplement utiliser **pip** pour installer un package à l'environnement.

(projet) pip install numpy

*Néanmoins, il faudra donc bien penser à générer le fichier **requirements.txt** afin de nous permettre d'exécuter votre agent dans un environnement fonctionnel, sous peine de ne pas pouvoir concourir au tournoi.*

Requirements

Le projet étant assez libre, il est possible (non essentiel) d'ajouter des librairies telle que numpy par exemple. Il est donc nécessaire de fournir un fichier **requirements.txt** pour que l'on puisse installer votre agent. L'environnement Conda et pip permettent d'exporter simplement ces dépendances via la commande suivante.

```
(projet) pip freeze > requirements.txt
```

*Attention, il faut veiller à exécuter cette commande avec l'environnement activé (présence du préfix entre parenthèse). Il suffit maintenant de fournir le fichier **requirements.txt** dans le rendu ZIP.*

Vérification final

Afin de s'assurer que l'export nous permettra de rouler votre agent lors du tournoi, il est intéressant de créer un nouvel environnement, d'y installer les dépendances exportées puis d'y faire jouer son agent.

```
(base) conda create --name tmp python=3.8
(base) conda activate tmp
(projet) pip install -r requirements.txt
```

*Si vous êtes capable de rouler une partie entre votre agent et un autre agent via cet environnement **tmp**, vous êtes assurés que nous pourrons installer et utiliser votre agent.*

6 Code à produire

Vous devez remettre un fichier `my_player.py` avec votre code. Pour ce faire, vous être entièrement libre d'utiliser la méthode que vous voulez. Si vous utiliser des librairies externes (`numpy`, `keras`, `pytorch`, etc.), veuillez les lister dans un fichier lors de votre soumission (`requirements.txt`).

Au minimum, votre solution devra hériter de la classe `Agent` et implémenter la fonction centrale `play()`. Cette fonction doit retourner une action selon l'état actuel du jeu :

Input : `self:Agent, percepts:dict, player:int, step:int, time_left:float`.

Toute l'information de la partie (placement des murs, positions des pions, etc.) est contenue dans l'objet `percepts`. Vous pouvez utiliser la fonction `dict_to_board()` dans `quoridor.py` pour convertir le dictionnaire `percepts` en une instance de la classe `Board`.

Output : `action`

Une action est un tuple contenant 3 éléments (`kind, i, j`), définis comme suit :

- `kind (char)` : est le type d'action réalisée. Respectivement, `P`, `WH`, et `WV` correspondent à un déplacement de pion, à un placement de mur horizontal, et à un placement de mur vertical.
- `i (int)` coordonnée horizontale (de gauche à droite) associée à l'action.
- `j (int)` coordonnée verticale (de haut en bas) associée à l'action.

Dans le cas d'un placement de mur, les coordonnées (i, j) sont celles des interlignes entre les cases. Si le mur est vertical (resp. horizontal), la coordonnées j (resp. i) indique la position **centrale** du mur. Dans le cas d'un déplacement de pion, les coordonnées i, j sont celles de la position absolue visée.

⚠**Dès lors que l'agent soumet une action, il doit s'assurer que celle-ci est bien valide (i.e., elle est autorisée par les règles du jeu). Si une action non autorisée est soumise, le joueur perd automatiquement la partie au profit de son adversaire. Un vérificateur de validité d'actions est disponible pour vous aider.**

Pour obtenir un example minimal d'un joueur fonctionnel (mais mauvais), vous pouvez vous regarder les agents `random_player.py` et `greedy_player.py`.

7 Quelques conseils pour vous aider

— Prenez-y vous tôt. Essayez d'enrichir votre rapport au fur et à mesure de l'avancée de votre projet. Cela vous fera gagner du temps lors du rendu du projet et vous permettra de garder le fil de vos recherches.
— Assurez-vous de bien comprendre le fonctionnement théorique des méthodes que vous souhaitez utiliser avant de les implémenter. Également, il est plus que conseillé de réaliser d'abord les agents du Morpion et de Puissance 4 du deuxième laboratoire.
— Prenez le temps de comprendre la structure du code de `quoridor.py` en y mettant des "*print statements*" et des commentaires. Si cela peut vous sembler laborieux, vous gagnerez beaucoup de temps par la suite lors de vos éventuels *debugging* (qui arriveront quasi certainement :-).
— Commencez par réaliser des agents simples, puis rajoutez des fonctionnalités petit à petit.
— Trouvez le bon compromis entre complexité et facilité d'implémentation lors de la conception de votre agent. Le temps de calcul pour chaque action mis à disposition de votre agent étant limité, il est également à considerer.
— Travaillez efficacement en groupe, et répartissez vous bien les tâches.
— Tirez le meilleur parti des séances de TP encadrées afin de demander des conseils aux chargés de laboratoire.
— Profitez de ce travail de groupe pour vous initier à l'outil de développement collaboratif Git ainsi qu'aux bonnes pratiques de l'outil.

8 Critères d'évaluation

L'évaluation portera sur la qualité du rapport et du code fournis, ainsi que des résultats de votre agent face à plusieurs agents d'une difficulté croissante. **Le classement de votre agent durant le tournoi n'aura aucune influence sur l'évaluation mais sera l'objet de point bonus sur la note finale.** Dès lors, il est tout a fait possible d'être éliminé rapidement lors du concours et d'obtenir une bonne évaluation, ou même d'être premier au concours et obtenir une mauvaise évaluation (p.e., à cause d'un rapport négligé). Les principaux critères d'évaluation sont les suivants :

1. Qualité générale et soin apporté au rapport.
2. Qualité de l'explication de la solution mise en œuvre.
3. Qualité du code (présence de commentaire, structure générale, élégance du code).
4. Performance face à différents agents d'évaluation (agents aléatoires, agents gloutons, d'autres agents intelligents, etc.)

⚠**La note minimale sera attribuée à quiconque ayant copier/coller une solution ou proposant un code contenant une erreur à la compilation. Notez que vu la liberté laissée à l'implémentation, il est très aisé de détecter les cas de plagiats**

⚠**L'évaluation sera effectué sur une machine Linux raw, veuillez donc à bien spécifier (si nécessaire) les bibliothèques externes dans un fichier** `requirements.txt` **afin de pouvoir les installer avec la commande** `pip install -r requirements.txt`**.**

⚠**Le jury préférera une solution simple accompagnée d'une riche explication logique à une solution trop complexe mettant en œuvre des concepts mal-compris de l'élève.**

Bon travail, bonne chance, et surtout, amusez vous!