SAE 1.02 COMPARAISON D'ALGORITHMES

UNE IA POUR DIAMANT

Préparez-vous pour le grand tournoi final

OBJECTIF GÉNÉRAL

L'objectif de ce projet est de concevoir une intelligence artificielle pouvant jouer au jeu de société DIAMANT déjà programmé dans la SAE 1.01. On réalisera à cette fin une intelligence artificielle (IA) spécialisée dans le fait de jouer à plusieurs au jeu, entre 3 et 8 joueurs, en 5 manches, afin de gagner le plus de points possibles. Un tournoi entre IA aura lieu en public afin de déterminer les meilleures IA ! On me dit en coulisse qu'il y a des lots à gagner !

A vos CLAVIERS !

LE MOTEUR DE JEU

On fournit un moteur de jeu nommé de façon assez originale `moteur_diamant`. Ce moteur est capable de simuler des parties à un nombre quelconque de joueurs et un nombre de manches entre 1 et 5. Il est conseillé de décompresser le dossier fournit tel quel et de travailler directement dedans sans toucher à son architecture.

Pour tester le moteur de jeu, vous pouvez exécuter le fichier `main.py` qui contient un exemple de partie entre des IA basiques. Tout un tas d'informations décrivant la partie défileront sur la sortie standard ; vous pouvez les rediriger dans un fichier pour les lire.

Ces informations ont un but de vérification ; ce sera à vous dans votre IA de récupérer autrement les informations que vous voulez pour la développer.

Malgré des tests, il se peut que ce moteur comporte encore certains bugs. Si vous en voyez, vous pouvez le signaler et nous ferons des mises-à-jour. Toutefois, le fonctionnement global du moteur et son interaction avec les IA seront inchangées.

LA CLASSE IA

Dans le dossier IA se trouvent les fichiers contenant les IA. L'IA que vous allez développer sera programmée en écrivant les méthodes d'une classe dont le schéma général est donnée, **IA_Diamant**. Trois exemples d'IAs vous sont données :

- `IA_aleatoire.py` contient le code d'une IA qui continue ou rentre au camp avec une probabilité de 50 %

- `IA_temeraire.py` contient le code d'une IA qui continue toujours (elle finit donc toujours dans un piège).

- `IA_trouillarde.py` contient le code d'une IA qui rentre toujours (elle ne gagne donc jamais de point non plus).

Ces IA sont très mauvaises (surtout les deux dernières) mais elles permettent de faire des tests. Ces IA ne s'occupent absolument pas de l'état du jeu pour prendre des décisions, on peut donc espérer que vous saurez faire un peu mieux !

Pour écrire votre IA, vous prenez le fichier `IA_a_completer.py` et le copiez en changeant son nom. Il y a quatre méthodes à compléter. Surtout, ne pas changer leur nom et leurs paramètres ! Souvenez vous que les méthodes commencent toujours par le paramètre 'self'

- la méthode `__init__(self, match)` est appelée automatiquement en début de partie pour créer l'objet IA de votre classe. Elle reçoit une chaîne de caractère `match` qui contient un descriptif de la partie (voir plus loin). Si vous voulez ajoutez des données dans votre IA, c'est dans cette méthode que vous pouvez les initialiser, en ajoutant par exemple `self.valeur = 3` ou `self.historique = []` ou tout autre possibilité.

- la méthode `action(self, tour)` est appelée à chaque tour de jeu afin de prendre une décision. Elle doit répondre dans un délai qui sera de quelques secondes par la lettre 'X' pour explorer ou 'R' pour rentrer au camp de base. Elle reçoit en argument une chaîne de caractères `tour`, qui décrit pour le tour précédent quelles décisions ont été prises et quelle carte a été révélée (voir plus loin). Quand un joueur a décidé de rentrer mais que les autres continuent sans lui la manche, la méthode action de l'IA correspondante est tout de même appelée. Elle peut répondre ce qu'elle veut car sa réponse n'est pas prise en compte. Ceci permet juste à l'IA de connaître l'avancement des autres joueurs.

- la méthode `fin_de_manche(self, raison, dernier_tour)` est appelée à chaque fin de manche. Elle sert à donner l'information aux IAs que la manche est terminée. Il n'y a pas de réponse attendue. Le paramètre `raison` peut valoir « `P2` » si la manche s'est terminée par révélation d'un piège (ici le piège 2), ou bien »`R` » si la manche est terminée car tout le monde est rentré aux camp. Le paramètre `dernier_tour` est un descriptif du dernier tour de la manche.

- enfin, la méthode `game_over(self, scores)` est appelée en fin de partie les scores de fin de partie. Cette méthode ne sert pas dans la simulation du jeu mais elle peut vous être utile pour effectuer des calculs, afficher des résultats etc en fin de partie.

LE DESCRIPTIF DE MATCH

En début de partie, la méthode `__init__` de votre IA est appelée et reçoit une chaîne de caractères qui décrit la partie. Cette chaîne de caractère a la forme suivante :

```
'5|3|bob,temeraire,jedi|1'
```

Ceci signifie qu'il s'agit d'une partie en 5 manches, à 3 joueurs, qui se nomment bob, temeraire et jedi (correspondant à des fichiers d'IA du même nom), et que le numéro de l'IA parmi eux est 1 (elle est donc ici dans l'exemple de l'IA temeraire).

LE DESCRIPTIF DE TOUR

A chaque tour, votre IA reçoit une chaîne de caractère qui décrit ce qui s'est passé le tour précédent. Cette chaîne est de la forme

```
'X,N,R|2'
```

En premier, vous trouvez les actions effectuées par les trois joueurs (plus d'actions si plus de joueurs)

- `X` signifie explorer

- `R` signifie rentrer

- `N` signifie « néant » car le joueur avait du déjà rentrer précédemment donc il ne fait pas de décision à ce tour.

Ensuite, vous trouvez la carte qui a été révélée. Il peut s'agir de :

- `2, 5, 12`, etc. : un entier indique un trésor en rubis

- `P1, P2, P3, P4` ou `P5` : indique un piège !

- `R` : indique une relique

- `N` : indique Néant car tous les joueurs sont rentrés donc aucune carte n'a été révélée.

Au premier tour, la chaîne de caractère est vide.

Comme vous le remarquez, les informations que vous recevez ne contiennent pas les détails du jeu, le nombre de rubis gagnés, les scores, les fins de manche, etc. C'est à vous de les déduire ! Utilisez des champs pour stocker ces informations. Ajoutez des méthodes ou des fonctions extérieures pour les calculer. Le moteur de jeu, lui, calcule tout ça et vous pouvez bien sûr regarder son code pour voir exactement ce qu'il calcule. Il est bien entendu interdit d'essayer de tricher et de récupérer des infos dans le moteur ! Votre IA doit se baser uniquement sur ce qu'elle reçoit.

D'ailleurs, un des premiers travaux à effectuer pour écrire votre IA est de « parser » les chaînes de caractères reçues à chaque tour afin de pouvoir utiliser les informations contenues à l'intérieur. Pour cela, vous pouvez utiliser les méthodes de `str` comme `split` ou `strip`, ou bien des techniques plus avancées comme l'extension `re`.

COMMENT LANCER UNE PARTIE ?

Dans votre fichier de travail (par exemple `main.py`) vous importez le fichier `moteur_diamant` et vous lancer la fonction partie avec les bons paramètres, comme dans l'exemple. L'import des fichiers d'IA se fera automatiquement si vous donnez un nom de fichier qui existe bien à la fonction. Les fichiers d'IA doivent se trouver dans le dossier IA.

Dans le fichier `moteur_diamant.py` vous trouverez un paramètre RAPPORT. Si ce paramètre est à `True` il fait un rapport de partie dans la sortie standard.

COMMENT PEUT FONCTIONNER UNE TELLE IA ?

L'IA que vous concevez doit savoir jouer à plusieurs, et elle sera testée contre différentes autres IA pour voir son score moyen obtenu sur un grand nombre de parties. Dans un premier temps, vous pourrez la tester contre les trois IA simples fournies, ensuite si vous « élevez » plusieurs IA vous pourrez les faire jouer entre elle avant de garder la meilleure.

Voici quelques idées pour concevoir une IA qui jouera mieux que les IA basiques fournies.

Une IA « faite à la main »

Avant de se lancer dans les choses compliquées, il y a quand même des choses simples à programmer… Quelques exemples :

1) au premier tour, on va toujours explorer

2) s'il n'y a pas de pièges et pas grand-chose à gagner sur les cartes, on va toujours explorer !

3) s'il y a peu à gagner et un très grand danger, on va presque toujours rentrer !

Et ainsi de suite, vous pouvez déjà implémenter ce genre de règles dans une IA avant de faire quelque chose de plus complexe. Il est possible de faire une IA assez complexe en utilisant ce genre de règles dites « heuristiques » en prévoyant de nombreux cas de figure.

Quelques idées pour des IA plus générales

- c'est un jeu de hasard et de prise de risque donc l'IA doit aussi prendre des décisions basées sur le hasard ! Elle doit être imprévisible.

- Les décisions étant des prises de risques, ce risque doit être mesuré. Si le risque est grand, il faut souvent rentrer, sinon il faut souvent explorer. Enfin… ça dépend aussi de ce que ça peut rapporter ! Pensez à vous quand vous jouez. Une IA va donc analyser la situation actuelle et se donner une probabilité de rentrer, comme 72 %. Il suffit alors de tirer un

nombre entre 1 et 100 au hasard, si on obtient 72 ou moins on rentre. Mais tout l'art de l'IA sera de calculer la bonne probabilité en fonction de la situation actuelle.

- pour calculer ces probabilités, il va falloir se baser sur les paramètres du jeu… combien vous avez de rubis, combien en ont les adversaires, combien il y a de rubis posés sur les cartes, combien de pièges sont sortis… toutes ces infos, il faut en fait les calculer d'après les informations reçues à chaque tour ! Les décisions ne seront pas les mêmes en début de manche quand il n'y a pas de piège, qu'en fin de manche quand il y a plein de rubis mais plein de pièges sortis !

- voici un exemple de raisonnement simple pour essayer de classer les situations: vous mettez à jour le nombre de rubis, le nombre de pièges sortis etc.. Et vous essayez en vous basant sur ces infos d'évaluer si la situation est risquée (une note de 0 à 5 par exemple) et si elle est intéressante (une note de 0 à 5, est-ce qu'il y a des rubis, des reliques à ramasser?). En vous basant sur ces deux notes, vous décidez de la probabilité de rentrer.

- Mais comment trouver pour chaque type de situation la « bonne » probabilité ? Ce sera forcément de l'essai et erreur. Une méthode élémentaire est de décider de certaines probabilités selon votre idée, de tester votre IA, de modifier ces probabilités pour voir si ça fait mieux, et ainsi de suite …

- Une solution plus avancée est de générer un grand nombre d'IA. Chacune de ces IA serait construite selon le même principe mais ses valeurs numériques (les probabilités) seraient différentes. En faisant jouer ces IA entre elles, vous pouvez arriver à déterminer la meilleure.

- On peut même aller plus loin en utilisant des techniques proches des algorithmes dits génétiques : on teste un grand nombre d'IA (disons 100), on garde les 10 meilleures, et on complète à 100 en introduisant des petites mutations (on change un peu les probabilités) sur les meilleures IA. On garde à nouveau les 10 meilleures et on recommence… Vous pouvez essayer de faire ça avec l'IA aléatoire en trouvant la probabilité optimale, ce n'est sûrement pas 50 % ! (Ceci dit cette IA sera forcément pas terrible puisqu'elle ne tient aucun compte des risques et des intérêts à agir).

Rendus Attendus

Le Rapport

Cette SAE étant assez libre pour vous dans la façon de procéder, il est très important que vous documentiez votre analyse, vos idées, votre façon de développer votre IA. Nous vous demandons donc d'écrire un rapport (pdf entre 4 et 10 pages) contenant :
- vos réflexions, vos analyses, vos idées pour développer une IA qui puisse jouer convenablement au jeu
- le travail que vous avez accompli, comment vous avez procédé pour créer cette IA, les tests qui ont été effectués, les modifications
- la façon dont l'IA finale que vous rendez fonctionne, de manière générale et éventuellement dans les détails

Ce rapport sera écrit dans un français convenable et sans fautes d'orthographe.

Votre travail

Pendant la SAÉ

- Travail en **binôme obligatoire**. Exceptionnellement par 3, si pas d'autre possibilité et uniquement sur autorisation de vos enseignant.e.s. Il est important de trouver un binôme qui soit à peu près du même niveau que vous, avec qui vous pouvez travailler à deux et vous répartir le travail (vous devez cependant maîtriser l'ensemble du code à la fin). Le travail en projet est une occasion de beaucoup progresser, aussi ne laisse
- Toutes les fonctions doivent être documentées avec des spécifications (docstring), et raisonnablement commentées. Les noms des fonctions et des variables doivent être judicieusement choisis. Comme précisé plus haut, les spécifications des fonctions et docstrings servent à développer intelligemment votre programme, et il est dommage voire inutile de les faire à la fin, on doit s'appuyer dessus pour bâtir le programme !
- S'aider d'un groupe à l'autre est possible, recopier un code tel quel est interdit. Nous avons des moyens informatiques de comparaison des codes (JPlag, entre autres). Le code de votre binôme doit être le vôtre, et chacun.e des membres du binôme doit être capable de l'expliquer intégralement, que ce soit « votre partie » ou non.
- Le projet doit être fonctionnel **SUR LES MACHINES DE L'IUT**. À vous de tester votre projet sur ces machines avant de le rendre.
- Vous déposerez votre projet dans la **rubrique Travaux de l'espace Initiation au Développement sur eCampus**, sous la forme d'une **archive** au format

`NOM1_NOM2.zip` où NOM1 et NOM2 sont les noms des deux membres du binôme (on a décidé de plus vous embêter avec les tar.gz).

- Cette archive contiendra :
    - un fichier python nommé IA_NOM1_NOM2.py
    - le rapport nommé RAPPORT_NOM1_NOM2.pdf

Attention, le nom respect des formats ci-dessous entraîne une diminution de la note finale.

DEADLINE : vendredi 14 janvier à 23h59

ÉVALUATION

Des soutenances en binôme auront lieu à la suite.

Les **compétences** qui seront évaluées sur ce projet seront à priori :

- *Respect du cahier des charges "dépôt" (deadline, archive nettoyée et au bon format, programme fonctionne directement sans erreurs et sans rien installer de nouveau sur les machines de l'IUT), contenant les deux fichiers avec les noms corrects*
- *La qualité du rapport : attention, la part de la note correspondant au rapport sera importante, pensez-y tout au long de votre SAE pour noter ce que vous faites.*
- *Spécifications : chaque fonction ou méthode doit documenter sa **spécification***
- *La qualité du code : commentaires, découpage en fonctions, constantes, nommage correct…*
- *Considérer tous les cas particuliers d'un programme (ex: tel paramètre fait-il planter le programme?) Attention, un programme qui plante est disqualifié de la partie et se retrouve avec 0 points.*
- *Enfin, la qualité de votre IA ! Comment elle se comporte face à des IA basiques, des IA plus avancées et dans le GRAND TOURNOI FINAL !*

ET MAINTENANT?

Go ! N'hésitez surtout pas à questionner vos enseignant.e.s si vous avez des questions. De plus, un salon dédié à la SAé permettra de dialoguer sur Discord. Bon travail! Amusez-vous bien et n'ayez pas peur des pièges !