Projet apprentissage par renforcement - Snake

Olivier Goudet

February 13, 2023

Tout d'abord, télécharger le dossier *Snake_ Qlearning* sur Moodle et créer un projet contenant les fichiers java contenus dans ce dossier. Gérer le *build path* du projet en ajoutant la librairie *commons-lang3-3.12.0.jar* qui est contenue dans le dossier en tant qu'*external jar* pour votre projet.

Dans ce projet, on va considérer les règles du jeu Snake correspondant au projet Design pattern du semestre 1, en mode solo ou duel. Pour simplifier, il n'y a pas d'objets spéciaux à récupérer sur la carte. On fera des parties avec un nombre maximum de tours de 200. Le score total qu'un snake peut obtenir au cours d'une partie correspond au nombre de pommes mangées, plus 10 points s'il a mangé un snake ennemie. On pourra faire des évaluation de scores en moyenne sur 100 ou 1000 partie (cf. variable Ntest).

Implémentation de premières stratégies d'apprentissage pour le Snake en mode solo

Pour chaque méthode d'apprentissage implémentée, il est conseillé de la tester tout d'abord en pas à pas avec le lancement du programme de debug via l'interface graphique (*main_ debugMode_ solo*), puis de faire les évaluations avec le programme (*main_ standardMode_ solo*).

Exercice 1 : implémentation de la stratégie Tabular Q-learning

L'objectif de ce premier exercice est d'implémenter la stratégie Tabular Q-learning vue dans le cours 2. Tout est à faire dans la classe *TabuLarQLearning_ solo* qui implémente l'interface *Strategy*.

1. Dans le constructeur *TabuLarQLearning_ solo* créer une table Q sous la forme d'un dictionnaire *HashMap<String, double[]>*. L'idée sera de représenter un état du jeu par une chaîne de caractères qui sera la clé de ce dictionnaire. Pour chaque clé, on associera une vecteur de double dont la taille correspond au nombre d'actions possibles du Snake (ici 4 actions possibles).

2. Implémentez une fonction *public String encodeState(int idxSnake, SnakeGame snakeGame)* qui permet d'encoder l'état du jeu sous la forme d'une chaîne de caractère unique.
   On fera l'hypothèse simplificatrice qu'un état du plateau de jeu avec un seul serpent est parfaitement représenté par la position de la pomme, la position de la tête du serpent et de chacun de ses éléments de corps, mais sans tenir compte de la façon dont le serpent est enroulé sur lui même...

3. A l'aide de cette fonction *encodeState*, compléter la classe *TabuLarQLearning_ solo* qui correspond à l'algorithme présenté dans le cours 2.

4. Tester votre méthode pour des niveaux de différentes tailles avec un seul Snake qui se trouve dans le dossier *layouts/alone* du répertoire du projet. Quels résultats obtenez vous ? Comment évolue l'apprentissage au fil du temps ? Comment expliquez vous les différences de scores entre les modes train et test.

5. Est-ce que cet algorithme semble efficace plutôt pour des niveaux avec des plateaux petits ou grands ?

Exercice 2 : implémentation de la stratégie Approximate Q-learning

L'objectif de cet exercice est d'implémenter la stratégie ApproximateQlearning vue dans le cours 3. Tout est à faire dans la classe *ApproximateQlearning_solo* qui implémente l'interface *Strategy*. Pour faire les tests et les évaluations, il faudra donner cette stratégie au serpent dans les classe *main_debugMode_solo* et *main_standardMode_solo*.

1. Quels premières *features* proposez-vous d'implémenter pour encoder un couple (état, action) ?
2. Implémenter l'agorithme *ApproximateQlearning* vu dans le cours 3.
3. Tester votre méthode pour les très petits niveaux. Quels résultats obtenez vous ? Comment évolue l'apprentissage au fil du temps ?
4. Tester pour un plus grand niveau comme *smallNoWall_alone*. Quels résultats obtenez vous en comparaison par rapport à la stratégie de l'exercice 1.
5. Proposez de nouvelles *features* pour améliorer les scores.
6. Quels sont les limites de cet algorithme ?

Implémentation de premières stratégies d'apprentissage pour le Snake en mode duel

Dans cette partie, testera les IA sur des cartes avec deux agents qui se trouve dans le dossier *layouts/alone* du répertoire du projet. On pourra mesurer la performance de l'IA par rapport à des ennemies ayant différentes stratégies, que ça soit des stratégies scriptés disponibles dans le package strategy du projet ou bien d'autres stratégies d'apprentissage par renforcement que vous auriez pu coder.

Pour faire les tests et les évaluations en mode duel, il faudra donner deux objets "Strategy" différents (eventuellement de même type) aux deux serpent dans les classe *main_debugMode_duel* et *main_standardMode_duel*.

Exercice 3 : implémentation de la stratégie TabularQlearning_duel

1. Adapter la stratégie TabularQlearning_solo pour créer la stratégie TabularQlearning_duel de façon à ce qu'elle puisse prendre en compte le fait qu'il y a un autre serpent sur la carte. Il faudra faire attention à bien encoder de façon différente le serpent, qui est en train d'utiliser la stratégie, du serpent ennemie. L'identifiant du serpent (0 ou 1) qui utilise la stratégie est donné en entrée des fonctions *chooseAction* et *update*.
2. Faites des tests en donnant la stratégie *TabularQlearning_duel* à l'un des serpents et une autre stratégie comme *StrategyRandom* ou *StrategyAdvanced* à l'autre serpent Quels résultats obtenez vous ?
3. Faites des tests sur la carte verySmallNoWall_fairduel en mettant le champ boolean randomFirstApple = true (emplacement de la première pomme défini aléatoirement sur la carte) , avec deux serpents qui ont chacun la stratégie TabularQlearning_duel (instanciée par deux objets différents). Vers quels comportement et quels scores pour les deux serpents l'algorithme converge pour ces deux cartes ? Pourquoi ?
4. Faites des tests sur les cartes *verySmallNoWall_fairduel* et *verySmallNoWall_unfairDuel* en mettant le champ boolean randomFirstApple = false (emplacement de la première pomme déterministe défini selon la carte), avec deux serpents qui ont chacun la stratégie TabularQlearning_duel (instanciée par deux objets différents). Vers quels comportement et quels scores pour les deux serpents les algorithmes en mode test convergent pour ces deux cartes ? Pourquoi ?

Exercice 4 : implémentation de la stratégie ApproximateQlearning_duel

1. Adaptez la stratégie ApproximateQlearning_solo pour créer la stratégie ApproximateQlearning_duel de façon à ce qu'elle puisse prendre en compte le fait qu'il y a un autre serpent sur la carte.
2. Quels features serait-il intéressant d'ajouter ?
3. Evaluez les différences de score en affectant la stratégie TabularQlearning_duel au premier serpent et la stratégie ApproximateQlearning_duel au deuxième serpent. On activera boolean randomFirstApple = true. Vers quels résultats semble converger les scores des deux serpent en mode test sur la carte *verySmallNoWall_fairduel*, ainsi que sur des cartes plus grandes comme*smallArenaNoWall*. Pourquoi ou *arenaNoWall*?

Exercice 5 : implémentation de la stratégie ApproximateQlearning_test_challenge

A partir de tous les tests que vous avez pu effectuer l'idée est maintenant de calibrer la meilleur stratégie Approximate Qlearning possible qui sera confrontée aux IA des autres étudiants ou bien à d'autre stratégies inconnues. Cette stratégie, devra pouvoir fonctionner en mode "test" sans plus d'apprentissage, et toutes les fonctions ou classes internes dont vous auriez besoin doivent être définies dans cette même classe.

1. Sélectionnez les features qui vous semble les plus pertinentes (on veillera à rester raisonnable dans le temps de calcul global de l'ensemble des features).
2. Entraînez les poids de votre algorithme Approximate Qlearning. Une fois que les poids vous conviennent, il faudra les définir "en dur" dans le constructeur de la classe pour pouvoir les utiliser en mode test sans plus d'apprentissage.

Jouer contre son IA

Vous pouvez jouer vous même contre une de vos IA en mode duel en utilisant le script *main_batchMode_duel*. Il faudra donner au premier serpent une instanciation de la classe *StrategyHuman*.

Bonus (facultatif) : implémentation des stratégies de deep Q learning.

Implémentez les deux types de stratégies qui utilisent un réseaux de neurones et vues dans le cours 4.

Annexe : fonctions du jeu Snake utiles pour la réalisation des algorithmes

- Dans la classe **SnakeGame**:
  - la fonction *getSnakes()* donne accès à la liste des agents snakes qui contient ou ou deux agents (mode solo ou duel). Chacun de ces agents Snake est caractérisé par un identifiant entier "id" qui correspond aussi à sa position dans la liste. NB : pour simplifier la gestions, les snakes morts ne sont pas retirés de la liste, mais ils n'interviennent plus dans le jeu et ne sont plus affichés.
  - fonction getItems() : donne accès à la liste des items sur le plateau et contient donc un item de type APPLE.
  - fonctions *getSizeX()* et *getSizeY()*, donnent la taille de la carte.
  - fonction getWalls() : donne acces à un tableau de boolean correspondant à la position des murs.
- Dans la classe **Snake**:
  - la fonction getPositions() donne accès à la liste des positions du Snake. La première position de cette liste correspond à sa tête, les autres à son corps.
  - la fonction isDead() permet de savoir si le snake est mort ou pas.