CSI4506 - Introduction à l'intelligence artificielle
Automne 2019
Université d'Ottawa - Faculté de génie

Snake AI
Groupe: P-14
Type 2

Professeur: Dr Caroline Barrière
Étudiant: Frédérik Laflèche (8616081)
Étudiant: Jérémie St-Pierre (8628942)

Date de la soumission: Le 6 décembre 2019

Feuille de temps

Membres du groupe	Tâche	Temps investi	Défi
Frédérik Laflèche	Document	6h	Analyser/Recherche
	Présentation	6h	Recherche
	Approche Neat	24h	Librairie neat-python et configuration
Jérémie St-Pierre	Document	3h	Analyser/Recherche
	Approche Q Learning	24h	Librairie pyqlearning (changé à la librairie gym)
	Implémentation du jeu Snake	12h	Mise à jour de l'état du jeu

Page d'introduction

Le but de ce projet est d'étudier et d'implémenter deux approches différentes de l'intelligence artificielle pour résoudre un problème. Suite à ceci, les deux approches seront analysé et comparé afin de déterminer quelle approche a eu plus de succès pour le problème spécifiquement. Le problème à résoudre est l'implémentation de l'intelligence artificielle sur le jeu simple Snake. Snake est un jeu vidéo ou le joueur manoeuvre une ligne qui grandit en longueur après avoir mangé une pomme où la ligne elle-même et les murs sont les obstacles primaires. Le but du jeu est de grandir la ligne le plus de fois possible, alors manger le plus de pommes sans foncer dans un obstacle.

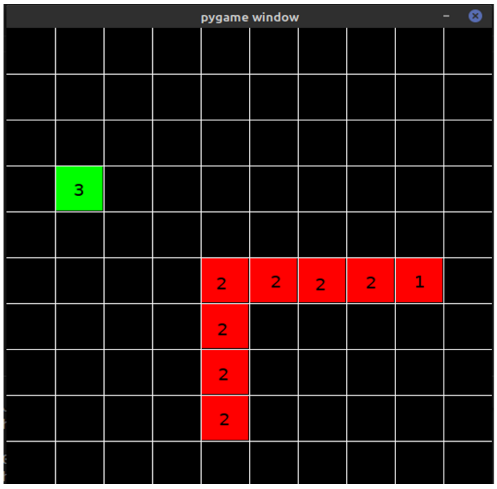
Nous allons complètement enlever le joueur et nous allons créer deux agents avec deux implémentations différentes aux réseaux de neurones et analyser la vitesse dont ces agents peuvent faire grandir le serpent dix fois, c'est à dire, manger dix pommes.

Les approches à l'intelligence artificielle que nous allon utilisé sont l'apprentissage par évolution et l'apprentissage par renforcement.

Ce projet est important car si nous réussissons à créer un agent qui peux jouer à Snake avec succès, ceci démontre que l'intelligence artificielle peut performé certaines tâches aussi bien que les humains. En applicants cette théorie à des enjeux majeurs dans la sociétés, nous pouvons utiliser l'intelligence artificielle pour remplacer des processus inefficace dans la vie de tous les jours et améliorer la qualité de vie des humains.

Table des matières

Description du jeu de données



Origine des données:

Les donnée provient directement du jeu “snake” dynamiquement en cours d’exécution. Ceci fut un grand défis de pouvoir synchroniser le flot de données de son origine jusqu’au agent en étapes d’apprentissage.

Contenue des données

Pour les données de l’état du jeu, on peut les classifiers comme cela:

1. La position des cases vides (**0**)
2. La position de la tête du “snake” (**1**)
3. La position du corp du “snake” (**2**)
4. La position de la pomme (**3**)

De plus, certaines données furent utilisé pour aider aux agents de s’approcher à une solution optimale. Ceux-ci se classifie comme étant des récompenses au actions.

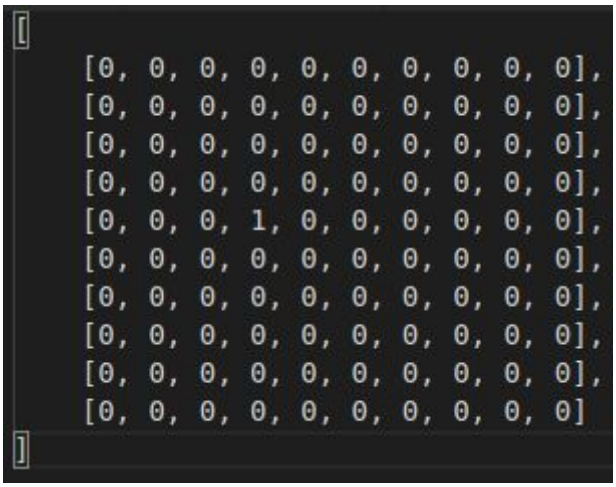
1. La grandeur du “snake”
2. La distance relative entre la tête du “snake” et de la pomme

Pour les données d’action, on peut les classifier comme cela:

1. Se déplacer vers la droite (➡)
2. Se déplacer vers la gauche (⬅)
3. Se déplacer vers le haut (⬆)
4. Se déplacer vers le bas (⬇)

Taille des données:

Les données sous leur forme d'origine sont généralement des ensembles de coordonnées. Cependant, lorsqu'il vient le temps de les appliquer pour les agents, leurs tailles réelle deviennent la taille de la matrice du jeu. Par exemple, pour une matrice de 10 X 10, chacun des attributs utilisé deviennent de taille 100. Cela est nécessaire pour la condition "one-hot" des entrées des réseaux de neurones.



Échantillonnage des données:

Comme mentionné ci-dessus, nous obtenons une explosion d'attributs à appliquer aux réseaux de neurones. Cela porte problème sur la performance de l'agent surtout lorsqu'il vient le temps d'appliquer des actions en temps réelle sur un jeu en exécution. Par cela, voici la stratégie d'échantillonnage utilisé pour attaquer ce problème:

1. Conserver une taille de matrice du jeu minimal (10 X 10)
2. Utiliser le moins d'attribut possible (exclure la position du corp du "snake")

Il serait aussi possible d'appliquer un réseaux de neurones convolutionnel pour ainsi filtrer les données en une composition plus légère d'entrée au réseau qui apprend.

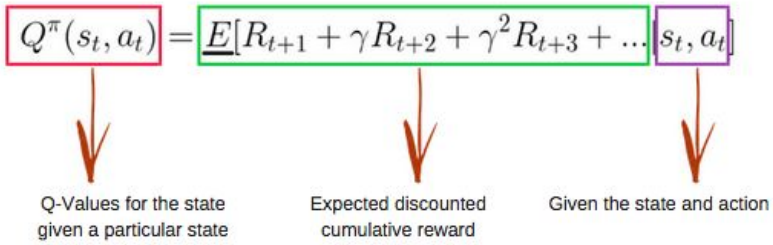
Détail des approches explorées

Nom et description des méthodes (que font-elles)

La première approche exploré est l'approche par évolution NEAT. NEAT est l'acronyme pour NeuroEvolution of Augmenting Topologies. Cette approche change non seulement les poids de chaque noeud mais change aussi la structure du réseau de neurones en utilisant l'historique des gènes, la spéciation et un développement incrémental. L'historique des gènes est utilisé

afin de pas réutilisé le même trajet entre deux noeuds qui a déjà été utilisé dans le passé. La spéciation est utilisé pour grouper des modèles avec des structures similaires et les comparés entre elle-même afin de ne pas affecté les modèles qui commence lentement. Le développement incrémental est utilisé en commençant le modèle avec une structure très simple et avec les itérations de l'algorithme, la structure vas augmenter de taille et de complexité. Commencé avec une structure complexe du début demanderait beaucoup trop puissance de calcul à chaque itération.

La deuxième approche exploré est l'approche par renforcement (Q learning). L'approche Q learning apprend avec la fonction Q* appliqué dans un réseau de neurones. Il faut définir les récompenses, suivre une stratégie (π) définie et développé par ajustement de Qπ(s, a) et Q(s,a)



Explication du choix de ces méthodes

Nous avons choisis ces méthodes pour comparer deux implémentations du réseau de neurones artificiels. C'est à dire un réseau évolutionnaire et un réseau stable. Dans le cas du réseau évolutionnaire, le nombre de couche cachée et de noeud varie par itérations. Tandis qu'un réseau stable a une structure prédéfinie qui varie seulement au niveau des poids

Détails de la mise en œuvre (package / ré-implémentation)

- **Apprentissage par évolution**: package neat-python [1].
- **Apprentissage par renforcement**: package pyqlearning [2].
    - Cependant, le package utilisé dans l'analyse des résultats sera le package gym [3]. pyqlearning a beaucoup manqué de la documentation

Feature engineering (exploration des attributs)

Approche évolutif

Dans le cas de l'approche NEAT, le package neat-python devais utilisé un fichier de configuration avec plus de soixantes différents attributs qui pouvait être modifié. C'est peu dire qu'il y a eu beaucoup de essaies et erreurs. L'approche NEAT utilise un nombre fixe d'entrée, alors il fallait trouver une façons de trouver des entrées pertinente à chaque stage du jeux snake. Au début, nous avons utilisé les coordonnées de la tête du serpent, les coordonnées de la nourriture et les coordonnées de la direction du serpent, c'est à dire 6 entrées. Cet approche n'a pas eu beaucoup du succès, alors nous avons décidé de donner la matrice du jeu comme entré au réseau de neurones, mais le package n'acceptait pas ce type d'entrée. Finalement, nous avons transformé la matrice en une liste à une dimension, contenant 200 entrées. L'approche NEAT utilise le terme fitness pour évaluer si une tel génération a eu du succès ou non. Donc il a fallu établir une méthode pour donner un bon fitness à une génération promettante et une mauvaise fitness à une génération non promettante, de façons que le réseaux applique la spéciation, ajuste les poids et garde l'historique des structures de réseau de neurones qui ont l'aire à fonctionner. Nous avons commencé par donner un bon fitness lorsque le serpent trouvait une nourriture, mais nous avons constaté rapidement que ceci arrivait peu fréquemment et par chance. Alors il fallait récompensé l'agent lorsqu'il faisait un mouvement vers la nourriture (ce qui ressemble beaucoup à l'approche par renforcement), alors nous donnons aussi des petits incréments au fitness lors d'un mouvement vers la nourriture et des incréments lors d'un mouvement contraire à la nourriture ou lorsque le serpent fonçais dans le mur. De plus, il fallait jouer avec plusieurs autres valeurs de configuration! Par exemple, la grandeur de la population dans une génération (donné essayé: 1, 10, 50, 100), le nombre de générations (donné essayé: 5, 10, 20, 50, 100), nombre de noeud caché ajouté après chaque générations (donné essayé: 0,1,2,5). Finalement, il fallait aussi essayé différent fonction d'activation, comme sigmoid, random, tanh, etc.

Approche par renforcement

Dans le cas de l'approche Q Learning, le package pyqlearning n'étais pas bien documenté et présentait des paramètres dont l'impacte de ceux-ci semblait aléatoire. Par cela, le package gym fut celui qui fut utilisé pour les résultats de ce projet. L'approche par Q Learning présente une nouvelle façon de fournir des données à l'agent. Ceux-ci sont les récompenses (valeur Q). Au début la récompense donnée en dépendance d'une action de l'agent était simplement la longueur obtenue du snake:

```
return self.game.get_score()
```

Cette approche n'a pas eu de succès puisqu'aucune rétroaction étais fournis à l'agent pour l'action décidé. Par cela, la récompense fut modifié à ceci:

```
food_dist = np.sqrt(((snake_pos[0] - food_pos[0]) ** 2) + ((snake_pos[1] -
food_pos[1]) ** 2))
rows = self.game.get_game_map_rows()
rel_dist = np.sqrt(((rows) ** 2) + ((rows) ** 2))
return self.game.get_score() - (food_dist / rel_dist)
```

L'apprentissage sans un réseaux de neurones de fait ainsi avec la modification d'une table Q. Cette table fut donc indexé par les données récolté du jeu sous forme de matrice en liste à une dimension. Cette matrice est sortie directement du package pygame [4] (package responsable du "rendering" du jeu "snake" utilisé pour implémenter ce projet).

Analyse des résultat

Méthode d'évaluation

La méthode d'évaluation que nous avons utilisé est très simple: quelle approche à l'intelligence artificiel a pu créer le meilleur snake. C'est à dire, qu'elle snake a mangé le plus de nourriture en le moins d'itérations.

Résultats individuels

Approche évolutif

Les résultats ont vraiment été mauvais. Puisqu'il y avait tellement de variables à considérer, il était difficile de savoir quoi changer de façons à voir une grande amélioration. Au tout début de l'implémentation de cette approche, il a été difficile de combiner notre implémentation du jeu snake avec le package existant neat-python. Alors, il fut un grand succès la première fois que nous sommes arrivés à faire le snake bougé de manière autonome sans input du clavier. Donc avec la méthode d'essaies erreurs et en implémentant tous les changements mentionné dans la section "Feature Engineering" , nous avons pu créer un agent qui a pu créer un snake qui a mangé trois nourritures.

Approche par renforcement

Les résultats ont été vraiment mauvais pour l'implémentation avec le package pyqlearning. L'application d'un réseau convolutionnel avec le manque de documentation fut une tâche colossal. Donc, une technique d'apprentissage sans l'utilisation de réseau de neurone fut utilisé. Cela fut réalisé à l'aide de la solution du package gym, offrant ainsi un façon efficace de définir un environnement. Certaine difficulté furent encourue pour faire la traduction du jeu snake à l'environnement compatible avec gym. Donc, par essai et erreur et en implémentant les changements mentionnés dans la section: "Feature Engineering", nous avons pu créer un agent qui a pu créer un snake qui a mangé trois nourritures.

Résultats comparatifs

	Approche évolutif	**Approche renforcement**	**Approche gloutonne**
Itération pour grandir le snake 5 fois	N'a pas été en mesure de grandir le snake 5 fois	~1000 parties de snake	~10 parties de snake
Gagnant après 10 itération			x
Gagnant après 100 itération			x
Gagnant après 1000 itération			x

Conclusion

Pour conclure, nous avons comparé deux approches différentes à l'application des réseaux de neurones en intelligence artificielle afin de comparer leurs efficacité sur le jeux simple snake. La première approche étudié et implémenté fut celle de NEAT, c'est à dire NeuroEvolution of augmenting topologies avec le package neat-python. Cet approche n'a pas eu beaucoup de succès dû à la complexité de configuration du package python. La deuxième approche étudié et implémentée fut celle de Q learning avec le package gym. Cet approche eu un peu plus de succès, mais quand même des résultats un peu décevant. Nous avons chacun passé une journée au complet, c'est à dire 24 heures séparés en plusieurs jours à étudié et essayé d'implémenter ses approches à l'intelligence artificielle et les résultats n'étais pas impressionnant. Mais, à l'intérieur de 30 minutes nous avons implémenté nous même une approche vorace où immédiatement, les résultats étais fort supérieur. Alors, il faut se poser la question: Est ce que c'était notre implémentation des réseaux de neurones qui était mauvaise ou bien est que c'est l'approche comme telle qui n'est pas optimal pour le jeu snake en spécifique.

Annexe A: GitHub du projet

https://github.com/jstpi047/CSI4506-project

Références

[1] NEAT-Python “NEAT Overview”. Consulté le 2019-10-30, https://neat-python.readthedocs.io/en/latest/neat_overview.html

[2] pyqlearning “Reinforcement Learning Library: pyqlearning”. Consulté le 2019-10-30, https://pypi.org/project/pyqlearning/

[3] Gym “Gym”. Consulté le 2019-12-06 https://gym.openai.com/

[4] Pygame “Pygame”. Consulté le 2019-12-06 https://www.pygame.org/news