Implémentation d'algorithme d'apprentissage développemental

Auteurs :

Théo Jaunet
Mickael Bettinelli

M2 Intelligence Artificielle

1 Introduction

L'IA développementale est un champs de recherche essayant de créer des agents qui puissent agir dans leur environnement sans a priori sur celui-ci. Dans ce travail, nous nous basons sur les notions d'interactions, de valence, de poids et de proclivité pour essayer de créer ce type d'agents. Une interaction est une séquence d'actions qu'un agent peut exécuter. La valence est alors la somme des récompenses de chaque action associée à cette séquence. Quand à lui, le poids permet d'évaluer la confiance que l'on peut accoder à la valence de l'interaction. Il est directement associé au nombre de fois qu'une interaction a été exéctuée. Enfin, la proclivité est la valeur calculée à partir de la valence et du poids permettant d'évaluer l'éfficacité de la séquence d'action. Plus cette dernière est élevée et plus l'interaction est bonne.

2 Partie 1 : Echauffement

Au cours de cet échauffement, nous avons mis en place un agent très basique avec pour seul but de faire une action tant qu'elle donne une reward positive, sinon faire une action aléatoire. Grâce à une implémentation aussi simpliste, l'agent, dans l'environnement le plus simple peut en quelques itérations converger vers la solution optimale. Cette solution est facilement trouvable puisqu'il a que quatre interactions possibles avec son environnement.

FIGURE 1 – Apprentissage de e1

FIGURE 2 – Apprentissage de l'alternance e1 / e2

3 Partie 2 : Apprentissage de séquence

Cette partie décrit de manière exhaustive tout les agents que nous avons créés en y détaillant les approndissements.

3.1 SmartAgent

Concept Le but de cet agent est de reprendre les principes de l'agent créé dans l'échauffement et d'en étendre les capacitées. En effet, cet environement a été créé car le premier agent ne pouvais pas fonctionner sur le nouvel environnement obligeant l'agent a alterner ses actions. Pour ce faire, nous avons émis l'hypothèse que l'agent précédent, ne possèdait pas assez d'information pour en extraire un shcéma. Nous avons mis en place une mémoire locale et de manière empirique, l'avons fixés à 20. Une fois bien informé, l'agent se doit de traiter correctement ces données. Pour ce faire, il recherche dans sa mémoire la meilleure séquence et la refait. Le but étant de remplir sa mémoire de "bonne séquence" afin de vite converger vers un optimum local. Pour éviter un renforcement trop fort, et pour remplir la mémoire, l'agent possède une curiosité. En effet , à l'initialisationn l'agent doit faire suffisament d'action aléatoires pour remplir sa mémoire. De plus, une fois sa mémoire remplie, l'agent éxplore son environnement avec une fonction d'exploration a rendement décroissante lui permetant de réaliser des actions totalement aléatoires en dépis de son optimum. Cette chance d'exploiration tends vers 0 pour éviter l'afflut de bruit dans la mémoire.

Avantages Ces attributs permettent à l'agent d'évoluer dans son environnement et d'en extraire rapidement des séquences assez longues. En effet, à la fin de 500 steps, l'agent possèdait une séquence de plus de 10 actions avec un rendement correct. En remplissant sa mémoire de sa meilleure séquence, l'agent converge vite sur une solution viable puis l'optimise. De plus l'agent possède une petite curiosétée afin de rajouter du "bruit" dans la mémoire pour y découvrir, possiblement, des meileures actions. Cette exploration reste toute fois suffisament basse pour converger rapidement (une seule action faite avec une chance d'exploration réduite a chaques itérations).

limites Cet agent, possède des limites assez importantes. En effet, l'agent est très dépendant de son babillage, étant donné qu'il ne possède qu'une faible curiositée, il converge vite sur des enviorennements stables et possidants

que peu d'interactions possibles. Par exemples, dans notre version du small loop, l'agent reste rapidement coincé dans une petite portion de l'environnement et y reste peu importe le nombre d'itérations suivantes.

3.2 TotalRecall

Concept En prennant en compte les problématiques des agents précédents, nous avons mis en place un nouvel agent, qui aurais pour mission d'éviter les optimums locaux. Avec ce but en tête, nous créé TotalRecall, un agent avec deux types de mémoire. La première , est la même que celle de l'agent précédent mais plus limitée à un nombre d'action. La deuxième, elle met en place une implémentation des interactions avec une sauvegarde de valence et podération de ces interactions. Pour cet agent, nous vons testé plusieurs façons de stocker et de mettre en place une évolution de la valence et du poids. Nous avons par exepmle tester de mettre en place un système de prédiction du resultat par rapport aux actions présente dans la mémoire et de réduire ce poids de façon a mettre en valeur les interractions avec des résultats fiable. L'implémentation finalement retenue fut, la mise en place d'un sysème de valence par moyenne totale de l'interation, et un poids qui équivaut au nombre de fois que l'agent a effectué cette interraction. Nous avons également couplé cet usage à une évaporation des poids de 1% par itération. Cet évaporation, a pour but de renforcer, lentement des interactions et d'éviter une spécialisation trop forte de l'agent. En effet, lors du choix de la meilleure proclivité, l'agent va finir par faire une action qu'il n'a pas fait depuis longtemps car jugée trop mauvaise. Ceci permet d'éviter une spécialisation de l'agent et, ainsi fournit un léger outil face à un environnement dynamique. Pour explorer son environnement, l'agent possède un babillage de quelques itérations, puis possèdes plusieures façons de sortir de l'optimal local. En effet, lors de son choix d'interations, l'agent se vois proposé des nouvelles interractions jamais faites. Ces interractions sont faites de plusieurs façon. Premièrement le "merge" d'interraction. Ici l'agent ce contente de metre bout a bout deux interactions aléatoire en en observe le résultat. Il peut également, générer de manière purement aléatoire une séquence d'action et la testée. Cet agent se sert de sa mémoire de plusieurs autre façons. Premièrement, dans sa méthode "whats_next", il choisis et génère des propositions d'interactions via sa dernière action. C'est a dire qu'il cherche dans sa mémoire des interations viables commençant par la où il en est. De plus l'agent possède une méthode purge qui change certaines actions jugées "trop mauvaise" par une autre action semie-aléatoire(inutile de remettre l'ancienne action).

Avantages L'un des plus grands avantage de cet agent, est son niveau d'abstraction. En effet, nous l'avons construit autours de sa capacitée a éxplorer et évaluer des situations. Cela lui permet d'avoir un résultat convennable et logique dans son environnement et ce, même si ce dernier évolue. De plus, il possède une robustesse au nombre d'action et resultats possibles. De plus, il possède une capacitée à "purger" ces interractions et de les tester afin d'en étudier les élements clés. Contrairement à l'autre agent, celui-ci cherche une séquence par rapport à sa dernière action et non une parmis les n dernières actions pouvant ne pas marcher juste dans la situation actuelle de l'agent qui aurait pus la downgrade et donc lui porter inutilement préjudice. Enfin, l'agent est moins guidé par ses actions que par son "envie" de découvrir.

limites Cependant, cet agent possède des limites assez radicales. En effet, à cause de son haut facteur d'exploration, l'agent parfois du mal a determiner un comportement optimal et même correctement converger. Son exécution requiert théoriquement trop d'itérations, même si en pratique cela est moins observable. Comme dit précédament, l'agent manque de spécialisation, ce qui l'empèche dans le small loop par exemple d'en faire le tour car il ne fait pas le lien entre sa position et ses interactions. Il augmente la taille de ses interactions en cas d'échec certes, mais se base uniquement sur la dernière action faite pour chercher la meilleure proposition d'interaction de qui, nous pensons, nuis à son évolution dans son environnement notament le small loop. Enfin, malgrés toutes ces possibilitées d'exploration, l'agent est trop dépendant de son babillage (pourtant court) pour explorer son environnement. Par exemple si une action primitive n'a pas été faite lors du babillage, il a de fortes chances qu'il mette du temps à la découvrir par la suite.

3.3 CartesianAgent

Concept Le CartesianAgent a comme principal objectif de trouver la meilleure séquence d'actions parmis une liste pré-existante. Pour cela, il génère un certain nombre de séquences à partir des actions primitives qu'il peut exécuter. Comme son nom l'indique, il génère toutes les séquences possible à partir de ses actions disponibles en se limitant à une taille de séquence maximum égale à la cardinalité de son ensemble d'actions primivites. L'agent trie

ensuite les bonnes séquences d'actions des mauvaises en les essayant et en mettant à jour lors de chaque test leur valence et leur poids.

Le grand nombre de séquences d'actions possible est un frein à l'apprentissage de l'agent sur les séquences. Plusieurs méthodes permettent d'éviter ce problème. La première permet à l'agent de bootstrap les valences des séquences qu'il voit pour la première fois. Pour cela, il utilise la valence de la plus grande sous séquence qu'il a exécuté au moins une fois. Selon la sous séquence utilisée, le bootstrap est relativement précis, mais sur de longues séquences d'actions, ce mécanisme peut faire gagner de nombreuses itérations. La seconde méthode mise en pratique est une heuristique. L'objectif est de ne pas apprendre la valence moyenne de chaque interactions, mais à la place d'apprendre celles des interactions qui sont susceptibles de servir à l'agent. Le CartesianAgent utilise une politique e-greedy que l'on nomme aussi "curiosité". Cette politique alterne entre une heuristique gloutonne et une heuristique d'exploration. La gloutonne rechercher la séquence d'actions fournissant la plus grande proclivité alors que l'heuristique d'exploration choisie une interaction au hasard dans celles connues. La curiosité de l'agent diminue selon une courbe exponentielle décroissante, ainsi plus l'expérience se déroule et moins l'agent souhait explorer ses possibilités. Ses tendances à renforcer les interactions fortes se voient augmentée au fil du temps. Ajouté à cela, le poids de chaque séquence profite d'une légère évaporation (diminution de 0.001%). Dans le cas où une actions n'est plus effectuée depuis trop longtemps (car valence trop faible par exemple), son poids se dissipe lentement ce qui rend la séquence invisible aux yeux de l'agent. Ce système permet donc à l'agent d'avoir une vue globale de ses possibilité grâce à la curiosité, mais aussi de renforcer les meilleures interactions.

Enfin, le CartesianAgent a la capacité de créer de nouvelles séquences en sa basant sur sa mémoire des actions, résultats et récompenses passées. A chaque prise de décision lors du choix d'une nouvelle interaction, l'agent recherche dans sa mémoire toutes les fois où il a déjà exécuté cette séquence. Si sa séquence ainsi que celle qui suit ont une proclivité positive alors il les fusionne pour créer une nouvelle interaction qui prend comme valence la somme des valences des deux sous séquences ainsi qu'un poids faisant la moyenne de celles-ci. L'agent peut ensuite apprendre sur cette nouvelle séquence composite comme il le fait avec les autres.

Avantages Ses diverses heuristiques lui permettent de trier relativement rapidement les séquences dont il a besoin et, comme dit ci-dessus, le bootstrap lui fait gagner un certain temps. Le CartesianAgent apprend donc facilement sur les premières interactions générées à partir d'un produit cartésien. Il trouve très souvent des solutions correctes sur les environnements que nous lui avons fournit.

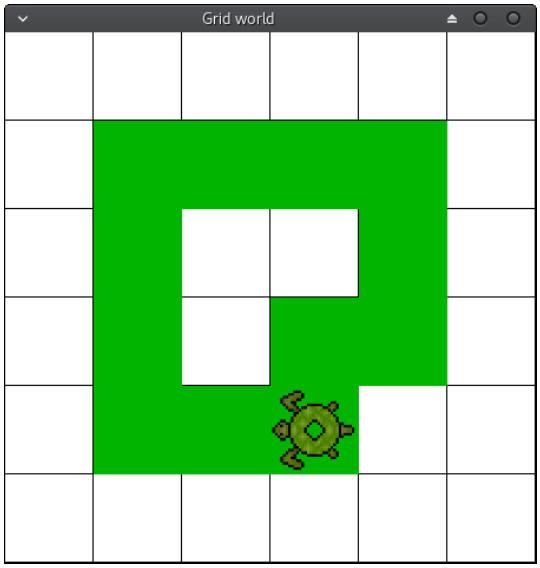

FIGURE 3 – Environnement Maze

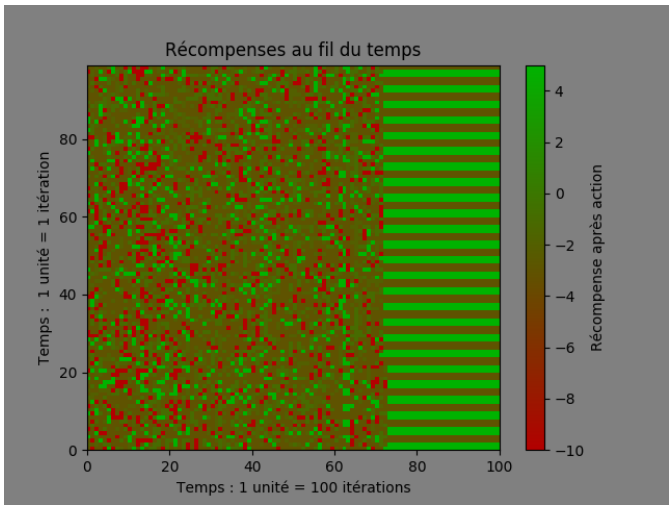

FIGURE 4 – Trace du CartesianAgent sur Maze

Voici la trace de l'agent sur le Maze. Sur la figure de droite nous pouvons voir les récompenses recues par l'agent pour chaque action qu'il a exécuté. Les récompenses sont représentées par une couleur allant du rouge au vert selon leur valeur. Cette représentation des données est simplement une agrégation des récompenses sur un graphique. Ainsi la récompense recue de la première action faite par l'agent se situe en bas à gauche du graphique, puis la

seconde est juste au dessus de la première, et ainsi de suite jusqu'à la 100ième qui se placera juste à droite de la première. L'unité de temps est donc différente selon les axes (1 unité = 1 itération sur l'ordonnée, et 1 unité = 100 itération sur les abscisses).

Nous remarquons que les traces rouges disparaissent brusquement aux alentours de la 7000ième itération (abscisse = 70). C'est à ce moment là que l'agent arrête complètement d'explorer ses interactions et qu'il se lance dans une politique entièrement gloutonne. Comme nous le voyons, un pattern émerge du graphe, l'agent exécute en boucle les actions de l'interaction qui à -selon sa vision des choses- la meilleure procilivité. Ici c'est donc l'action "Avancer" x2 puis "Tourner" x2. L'agent fait dont des allers er retours sur une longueur.

limites La possibilité de générer de nouvelles séquences à partir de sa mémoire est théoriquement un bon point qui pourrait permettre à l'agent de s'adapter plus facilement à des environnements complexe. Mais en pratique la génération d'interactions est assez inefficace et l'agent peine à en créer de réellement utile.

3.4 DullAgent

Concept Suite à tout ces essais, nous avons voulus construire un agent plus égocentré, en effet, notre but était de nous mettre à la place de l'agent, imaginer ses connaissance de son environnement. On pourrait par exemple nous imaginer essayer de se déplacer dans le small loop dans le noir. Pour ce faire, nous avons créé un agent fortement dépendant de ses actions précédentes pour en déterminer les suivantes. De plus, cet agent possède une très légère exploration afin d'élargir ses horizons. Si l'agent n'a rien a faire et que sa dernière action a été jugée "bonne" ou du moins suffisament bonne, il l'a refait, comme nous marcherons dans un labirynte. Si toucher du vide nous a permis d'avancer, il y a de fortes chances que cela de cette situtation se reproduise. En cas d'échec, l'agent possède plusieurs possibilitées. Soit chercher dans sa mémoire une séquence d'actions qui lui a permis d'obtenir a nouveau une action jugée "bonne". Si cette recherche lui a renvoyé plusieurs séquences, l'agent choisis celle qui possède la meilleure proclivité . Si jamais la recherche de séquence est infrutueuse, l'agent effectue une action aléatoire (on peut traduire cela par du manque d'information). En cas d'absence de séquence, l'agent peut également avec une très faible probabilitée (pour ne pas le perturber) générer une séquence aléatoire et se taille procédurale(taille max limitée). De plus l'agent descrit sa situation actuelle non plus avec sa dernière action effectuée mais par les n dernières, pour ainsi trouver une solution plus adéquate. Pour évaluter et choisir une interaction, l'agent y stocke sa valence et sa proclivitée qu'il me a jour régulièrement. L'agent posssède également une forte évaporation afin de ne pas négliger les nouvelles interactions et de rapidement les évaluer.

Avantages Grâce à ces méthodes, l'agent réaliser est correctement dépendant de ces actions précédentes comme un manequin (d'où son nom). Cela permet d'avoir dès les premières itérations des résultats logiques. Cet agent converge donc assez vite, mais continue de toujours s'optimiser ce qui fait que peut importe le nombre d'itération il se comportera toujours de manière plus ou moins logique. De plus, dans cette implémentation, l'agent ne possède de présuposé sur son environnement uniquement son système motivationnel. Son éxécution peut être vue plus ou moins comme une "hisoire". Puisque tous ce qu'il fait dépend des actions précédentes. Compte tenus de ces capacitées de convergence, l'agent est capable de s'adapter à un environement fortement dynamique. Il élimine également très bien le bruit test effectués sur le carte "line"

limites Comme toujours, cet agent dépends de ses actions pour trouver une solution correcte. De plus l'agent se retrouve très facilement dans des optimums locaux. En effet, il manque d'abstraction pour voir le problème au complet. Par exemple, dans le small loop, l'agent va vite faire une petite boucle, sans se préocuper du reste de la carte. Sur la carte "large_maze" l'agent à résussis que quelques fois a pouvoir faire le tours complet en boucle. Mais ces résultats son trop imprévisibles

4 Conclusion

Pour conclure, ce projet nous a permis d'étudier le domaine de l'ia développementalle et en extraire des problèmatiques. Bien que notre objectif de réaliser un agent capable d'effectuer a coup sur le small loop n'a pas été

un franc succès, nous avons pour comprendre le fonctionnement des agents et comment les amméliorer. La première supposition est que dans l'agent small loop d'Olivier Georgeon possède l'action toucher à gauche et toucher à droite ce qui lui permet (selon nos suppositions) de réaliser plus rapidement une topologie de la carte. Nous pensons également, que l'implémentaion d'Olivier possède moins de candidats possibles de solution et qu'ainsi, il converge et explore plus vite. Nous avons pu voir une application concrète de l'épineuse problèmatique : La balance en diversification et l'intensification des solutions. Au cours de l'implémentation des différents agents notament des agents trop diversifés comme TotalRecall ou trop intensif comme SmartAgent. Il reste toute fois un grand nombre d'améliorations possibles. Pour n'en citer que quelques unes, DullAgent pourrait par exemple avoir un système motivationnel évolutif afin d'encore mieux s'adapter à son environnement et interpréter lui même le résultat de ses actions. On pourrait également imaginer un système d'ennuis plus poussé que celui développer pour éviter les optimum locaux et ainsi, dans le small loop pouvoir tout explorer. Attention toute fois à ne pas nuire à l'exécution globale. Enfin, on pourrait envisager une manière sans pré-supposé sur l'environnement de choisir une limite à la taille des séquence afin d'en extraire des actions avec un sens important par exemple toucher puis avancer pour éviter les collisions.