

Université Claude Bernard



Lyon 1

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 accorder à la valence de l'interaction. Il est directement associé au nombre de fois qu'une interaction a été exécutée. Enfin, la proclivité est la valeur calculée à partir de la valence et du poids permettant d'évaluer l'efficacité 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 n'a que quatre interactions possibles avec son environnement.

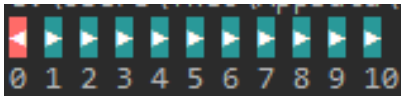


FIGURE 1 – Apprentissage de e1



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 apprentissages.

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 capacités. En effet, cet environnement a été créé car le premier agent ne pouvait pas fonctionner sur le nouvel environnement obligeant l'agent à alterner ses actions. Pour ce faire, nous avons émis l'hypothèse que l'agent précédent, ne possédait pas assez d'informations pour en extraire un schéma. Nous avons mis en place une mémoire locale et de manière empirique, avons limité sa taille à 20 emplacements. 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'initialisation l'agent doit faire suffisamment d'action aléatoires pour remplir sa mémoire. De plus, une fois sa mémoire remplie, l'agent explore son environnement avec une fonction d'exploration à rendement décroissante lui permettant de réaliser des actions totalement aléatoires en dépit de son optimum. Cette chance d'exploration tends vers 0 pour éviter l'afflux 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, après 500 steps, l'agent possédait une séquence de plus de 10 actions avec un rendement correct. En remplissant sa mémoire avec sa meilleure séquence, l'agent converge vite sur une solution viable puis l'optimise. De plus l'agent possède une petite curiosité afin de rajouter du "bruit" dans la mémoire pour y découvrir, possiblement, des meilleures actions. Cette exploration reste toute fois suffisamment basse pour converger rapidement (une seule action faite avec une chance d'exploration réduite à chaque 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 curiosité, il converge vite sur des environnements stables ne possédant 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 prenant en compte les problématiques des agents précédents, nous avons mis en place un nouvel agent, qui aurait pour mission d'éviter les optimums locaux. Avec ce but en tête, nous avons créé TotalRecall, un agent avec deux types de mémoires. La première, est la même que celle de l'agent précédent mais plus limitée à un nombre d'actions. La deuxième met en place une implémentation des interactions avec une sauvegarde de valence et pondération de ces interactions. Pour cet agent, nous avons testé plusieurs façons de stocker et de mettre en place une évolution de la valence et du poids. Nous avons par exemple essayé de mettre en place un système de prédiction du résultat par rapport aux actions présentes dans la mémoire et de réduire ce poids de façon à mettre en valeur les interactions avec des résultats fiables. L'implémentation finalement retenue fut la mise en place d'un système de valence par moyenne totale de l'interaction avec un poids qui équivaut au nombre de fois que l'agent a effectué cette interaction. Nous avons également couplé cet usage à une évaporation des poids de 1% par itération. Cet évaporation, à 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 possède plusieurs façons de sortir de l'optimal local. En effet, lors de son choix d'interactions, l'agent se voit proposer de nouvelles interactions jamais faites. Ces interactions sont faites de plusieurs façons. Premièrement le "merge" d'interaction. Ici l'agent se contente de mettre bout à bout deux interactions aléatoires en observant le résultat. Il peut également, générer de manière purement aléatoire une séquence d'action et la tester. Cet agent se sert de sa mémoire de plusieurs autres façons. Premièrement, dans sa méthode "whats_next", il choisit et génère des propositions d'interactions via sa dernière action. C'est à dire qu'il cherche dans sa mémoire des interactions viables commençant par la où il en est. De plus l'agent possède une méthode purge qui change certaines actions jugées "trop mauvaises" par une autre action semi-aléatoire (inutile de remettre l'ancienne action).

Avantages L'un des plus grands avantages de cet agent, est son niveau d'abstraction. En effet, nous l'avons construit autour de sa capacité à explorer et évaluer des situations. Cela lui permet d'avoir un résultat convenable et logique dans son environnement et ce, même si ce dernier évolue. De plus, il possède une robustesse au nombre d'actions et résultats possibles. Il possède aussi une capacité à "purger" ces interactions et de les tester afin d'en étudier les éléments clés. Contrairement à l'autre agent, celui-ci cherche une séquence par rapport à sa dernière action et non une parmi les n dernières actions pouvant ne pas marcher juste dans la situation actuelle de l'agent qui aurait pu 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 parvient parfois du mal à déterminer un comportement optimal et même correctement converger. Son exécution requiert théoriquement trop d'itérations, même si en pratique cela est moins observable. Comme dit précédemment, l'agent manque de spécialisation, ce qui l'empêche 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, nuit à son évolution dans son environnement notamment le small loop. Enfin, malgré toutes ces possibilités d'exploration, l'agent est trop dépendant de son babillage (pour autant 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 parmi 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 primitives. 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 recherche 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 souhaite 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 se 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 fournis.

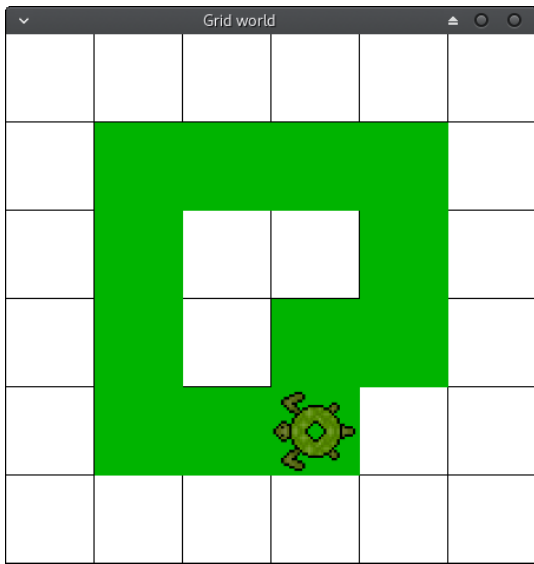


FIGURE 3 – Environnement Maze

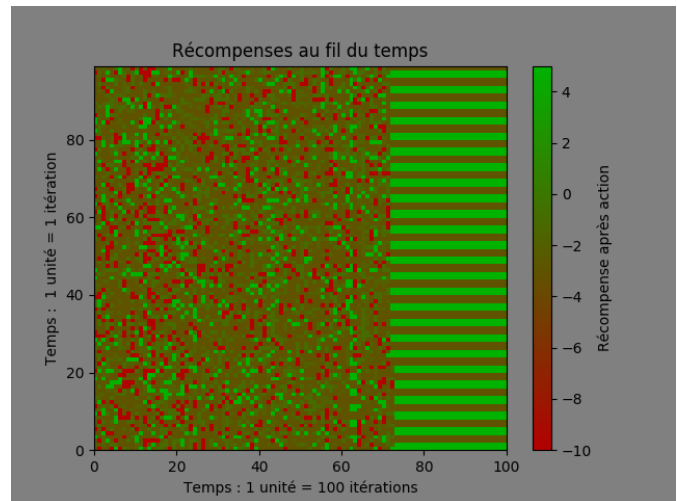


FIGURE 4 – Trace du CartesianAgent sur Maze

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 proclivité. Ici c'est donc l'action "Avancer" x2 puis "Tourner" x2. L'agent fait donc des allers et 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 voulu construire un agent plus égocentré, en effet, notre but était de nous mettre à la place de l'agent, imaginer ses connaissances de son environnement. On pourrait par exemple nous imaginer essayer de nous 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 à faire et que sa dernière action a été jugée "bonne" ou du moins suffisamment bonne, il l'a refait. Si toucher du vide nous a permis d'avancer, il y a de fortes chances que cela de cette situation se reproduise. En cas d'échec, l'agent possède plusieurs possibilités. 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 choisit celle qui possède la meilleure proclivité. Si jamais la recherche de séquence est infructueuse, 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 probabilité (pour ne pas le perturber) générer une séquence aléatoire (taille max limitée). De plus l'agent décrit sa situation actuelle non plus avec sa dernière action effectuée mais par les n dernières, pour ainsi trouver une solution plus adéquate. Pour évaluer et choisir une interaction, l'agent y stocke sa valence et sa proclivité qu'il met à jour régulièrement. L'agent possède également une forte évaporation afin de ne pas négliger les nouvelles interactions et de rapidement les évaluer.

Avantages Grâce à ces méthodes, l'agent va réaliser correctement des actions à l'aide des actions précédentes comme un mannequin (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 peu importe le nombre d'itérations il se comportera toujours de manière plus ou moins logique. De plus, dans cette implémentation, l'agent ne possède pas de présupposé sur son environnement. Son exécution peut être vue plus ou moins comme une "histoire". Puisque tout ce qu'il fait dépend des actions précédentes. Compte tenu de ces capacités de convergence, l'agent est capable de s'adapter à un environnement fortement dynamique. Il élimine également très bien le bruit test effectués sur le carte "line"

limites Comme toujours, cet agent dépend 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 préoccuper du reste de la carte. Sur la carte "large_maze" l'agent n'a réussi que quelques fois à faire le tour complet de la boucle. Mais ses résultats sont trop imprévisibles

4 Conclusion

Pour conclure, ce projet nous a permis d'étudier le domaine de l'ia développementale et en extraire des problématiques. Bien que notre objectif était de réaliser un agent capable d'effectuer à coup sûr le small loop, ceci

n'a pas été un franc succès, nous avons pu comprendre le fonctionnement des agents et comment les améliorer. La première supposition est que dans l'agent small loop d'Olivier Georgeon, son agent 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 solutions et qu'ainsi, il converge et explore plus vite. Nous avons pu voir une application concrète de l'épineuse problématique : La balance entre diversification et l'intensification des solutions. Au cours de l'implémentation des différents agents notamment des agents trop diversifiés 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 développé 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.