Projet : création d'un agent aspirateur

Manhes Benoît, Pattier Lilian, Scalzo Pierre

Manb22079601, Patl21039604, Scap15089506

4 Octobre 2018

Université du Québec à Chicoutimi - Maîtrise informatique

Cours 8INF846 : Intelligence Artificielle

Table des matières

1 Le problème **3**

1.1 Type de problème . . . 3

1.2 Formulation du problème à simple état . . . 4

2 L'agent intelligent **4**

2.1 Agent basé sur les buts . . . 4

2.2 Les propriétés de l'agent . . . 4

2.3 Les actions . . . 4

2.4 Capteurs et effecteurs . . . 5

3 L'environnement **5**

3.1 Propriétés de l'environnement . . . 5

3.1.1 Partiellement observable . . . 5

3.1.2 Stochastique . . . 5

3.1.3 Séquentiel . . . 5

3.1.4 Dynamique . . . 5

3.1.5 Discret . . . 5

3.1.6 Unique agent . . . 5

3.2 Éléments . . . 5

3.3 Attribution des points . . . 6

4 Stratégie de recherche **6**

4.1 L'espace d'état . . . 6

4.2 Exploration non-informée : Recursive Depth-Limited Search . . . 7

4.3 Exploration informée : Greedy Search . . . 7

4.4 Vers un agent apprenant : optimisation de la fréquence d'observation . . . 8

5 Résultats obtenus **9**

5.1 Choix des algorithmes d'exploration . . . 9

5.2 Efficacité et efficience . . . 10

5.3 Complexité . . . 10

Introduction

L'objectif de ce projet est la conception d'un agent intelligent capable d'observer son environnement et de prendre une décision en fonction de celui-ci. Il prend ici la forme d'un robot aspirateur fourni de capteurs et d'effecteurs lui permettant d'évoluer dans son environnement. Ce dernier est un manoir constitué de nombreuses pièces que nous appellerons par la suite *cases* du fait de leur disposition en forme de grille. L'environnement génère sporadiquement de la poussière que le l'agent souhaite aspirer, ainsi que de bijoux qu'il veut au contraire ramasser.

FIGURE 1 – Exemple d'aspirateur sur lequel implémenter une intelligence artificielle

L'intérêt de ce projet est d'implémenter une intelligence artificielle basée sur un but. Afin de présenter et de répondre correctement au sujet nous décrirons dans un premier temps précisément le problème, l'agent et l'environnement. Puis nous discuterons des stratégies de recherche. Enfin nous analyserons les résultats obtenus.

D'autre part, afin d'éviter toute ambiguïté, il serait commode de préciser le vocabulaire utilisé et les analogies :

- Le robot aspirateur sera décrit par *l'agent* ;
- Le manoir sera *l'environnement* ou encore la *grille* ;
- Les pièces seront des *cases* ;
- La poussière et les bijoux n'auront pas de dénomination particulière mais seront réunis dans l'appellation *éléments* ;
- L'itinéraire est une liste d'éléments que l'agent doit ramasser dans un ordre précis. L'itinéraire est calculé pour rapporter le maximum de points ;
- Un cycle définit une durée durant laquelle l'agent n'a pas observé l'environnement. Logiquement, la fréquence et l'itinéraire ne sont pas recalculés au cours d'un cycle, mais entre deux cycles ;
- Les objectifs sont modélisés par une liste d'éléments à atteindre dans un ordre précis. Cette liste diffère cependant de l'itinéraire. En effet le premier élément de la liste des objectifs désigne le premier élément que l'agent doit atteindre. Lorsque celui-ci est traité, il est supprimé de la liste. L'itinéraire définit donc l'ordre des objectifs à atteindre durant le cycle, tandis que la liste des objectifs indique les éléments qui n'ont pas encore été traités pendant ce cycle.

1 Le problème

1.1 Type de problème

Notre problème est non-déterministe et partiellement observable : l'agent sait dans quel état il sera après l'exécution de ses actions, mais ne connaîtra pas son environnement avant de l'avoir observé de nouveau. Nous détaillerons les propriétés de l'environnement en partie 2.

1.2 Formulation du problème à simple état

Nous allons formuler notre problème en cinq points :

- État initial : une case aléatoire de la grille ;
- Opérateurs : l'agent peut aspirer, ramasser, aller à gauche, à droite, en haut, en bas ou encore ne rien faire ;
- Fonction de succession : celle-ci sera déterminée par le but de l'agent. Toutefois, pour aller de la case actuelle à une case désirée, la longueur du parcours sera pour l'agent une distance de Manhattan.
- Test de but : obtenir le plus de points possibles ;
- Coût du chemin : du point de vue de l'agent, chaque action lui coûte 1 point d'énergie.

2 L'agent intelligent

2.1 Agent basé sur les buts

Notre agent ne fait pas qu'observer son environnement et chercher les pièces présentant de la poussière pour l'aspirer. Il doit également gérer le ramassage de bijoux, rendant le problème plus complexe. Sa fonction n'est alors plus seulement d'aspirer, mais d'adapter ses actions en fonction de son environnement. Pour cela nous avons mis en place un système de points : l'aspiration de poussière et le ramassage de bijoux lui en rapporte, alors qu'au contraire l'aspiration de ces derniers lui en font perdre.

L'agent a donc un but qui est de gagner un maximum de points. Il n'est pas un *agent réflexe*, mais bien un *agent basé sur un but.*

L'intelligence intervient dans l'exploration de son espace d'état qui peut être non-informée ou informée. Nous décrirons la différence et analyserons nos solutions par la suite.

2.2 Les propriétés de l'agent

Notre agent est caractérisé plus spécifiquement sous quatre propriétés inhérentes.

- Autonomie : aucune action n'est nécessaire à son fonctionnement une fois démarré. Il contrôle ses actions seul en fonction de son but ;
- Habileté sociale : dans le cadre de notre problème notre agent n'a pas besoin de cette propriété sachant qu'il n'y aura pas d'autre agent présent dans son environnement ;
- Réaction : grâce à ses capteurs l'agent est capable d'observer son environnement et de prendre en considération les changements potentiels, soit l'apparition de poussières et de bijoux ;
- Pro-action : les actions de notre agent sont basées sur un but.

2.3 Les actions

Notre agent a quatre types d'actions possibles :

- Aspirer : supprime tous les éléments d'une case ;
- Ramasser : ne fonctionne que sur les bijoux et le supprime si l'un est présent sur la case où l'action est réalisée ;
- Se déplacer : l'agent ne peut se déplacer à chaque itération que d'une case en haut, en bas, à droite ou à gauche ;
- Ne rien faire.

Ces actions ne sont que des outils de l'agent lui permettant d'atteindre son but. Néanmoins, chaque réalisation lui coûte 1 point d'énergie soustrait aux points comptabilisés en conséquence de ces actions.

2.4 Capteurs et effecteurs

Les capteurs sont les outils de l'agent pour observer son environnement. Ceux-ci rendent possible la construction d'un arbre de recherche en localisant la poussière et les bijoux.

Les effecteurs permettent quant à eux de réaliser les différentes actions.

3 L'environnement

3.1 Propriétés de l'environnement

3.1.1 Partiellement observable

Les capteurs de l'agent lui donne accès à l'ensemble de son environnement, mais les observations ne sont pas exécutées en continu.

3.1.2 Stochastique

Le prochain état de l'environnement n'est pas complètement déterminé par son état courant et l'action de l'agent. L'apparition de poussière et de bijou est aléatoire et leur position ne peut être anticipée, c'est un événement aléatoire sporadique.

3.1.3 Séquentiel

L'apparition de poussière dépend des apparitions passées et des actions de l'agent dans la mesure où elle ne peut avoir lieu sur une case déjà occupée.

3.1.4 Dynamique

L'agent et l'environnement s'exécutent sur deux fils d'exécution différents. Des éléments peuvent alors apparaître dans l'environnement avant même que l'agent ait fini de délibérer.

3.1.5 Discret

La propriété discrète s'applique à plusieurs niveaux. Le nombre de pièces, correspondant à des cases dans notre implémentation, est limité. Pour satisfaire au sujet nous en avons créé 100, limitant notre agent à un nombre fini d'actions possibles dans cet espace.

3.1.6 Unique agent

L'environnement ne présente qu'un seul agent pour satisfaire au sujet.

3.2 Éléments

Une case de l'environnement présente huit états possibles en fonction de la présence d'éléments :

- Une case vide ;
- Une case présentant de la poussière ;
- Une case avec un bijou ;
- Une case contenant l'agent ;

Puis toutes les combinaisons possibles avec ces trois derniers points.

3.3 Attribution des points

L'agent effectue des actions dans le but de maximiser ses points. Néanmoins c'est le rôle de l'environnement de les compter. La répartition sera telle que :

- Aspirer de la poussière : 8 points ;
- Ramasser un bijou : 5 points ;
- Aspirer un bijou : -20 points ;

Notons que l'agent ne peut *ramasser* de la poussière. De plus précisons que l'action *aspirer* s'effectue sur l'ensemble d'une case. Ainsi en cas de présence de poussière et de bijou sur une même case, l'environnement considérera les deux éléments dans le compte des points, octroyant à l'agent -12 points.

Nous avons fixé plus de points pour la poussière que pour le bijou car nous préférons que sa tâche principale soit d'aspirer et que le ramassage de bijoux soit secondaire. Néanmoins, ce que nous voulons le moins est que l'agent aspire un bijou, nous avons donc octroyé à cette action un plus gros malus.

4 Stratégie de recherche

Notre agent intelligent présente deux types de recherche au choix : tout d'abord une non-informée, puis une version améliorée avec une recherche informée.

Dans la suite nous caractériserons :

- Le nombre d'éléments = n ;
- La profondeur de l'arbre = k ;

4.1 L'espace d'état

Il s'agit de l'ensemble des états atteignables depuis l'état initial, par n'importe quelle séquence d'action. Celui-ci permet à l'agent de choisir la meilleure séquence possible maximisant ses points. La première remarque est que de ne pas limiter l'espace d'état empêche une exécution correct de l'algorithme le confrontant à trop de cas possible.

Nous imaginons bien qu'avec 100 cases avec ou sans poussière ou bijou, en prenant également en compte les positions possibles initiales de l'agent, l'espace d'état est bien trop important pour être analysé directement. Il y aurait en fait $2^{100} * 2^{100} * 100 = 1,606 * 10^{62}$ possibilités. Il faut alors restreindre cet espace d'états.

Nous considérons donc désormais qu'un état correspond à une case occupé par un élément. Le but est alors de parcourir cet espace de manière la plus optimisée possible de façon à ce que l'agent maximise sa performance.

Considérer que les cases occupées par un élément forme les noeuds de l'arbre d'exploration conduit encore a beaucoup de possibilités suivant la probabilité d'apparition.

Imaginons par exemple un espace à 3 états A,B et C possibles pour l'agent :

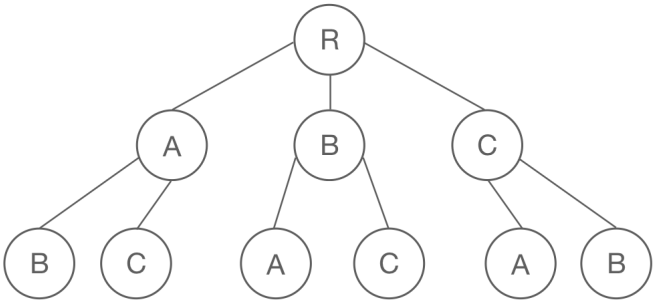

FIGURE 2 – Exemple d'explosion de l'espace d'état avec R l'agent et A,B,C des états possibles

Nous remarquons une explosion combinatoire de la forme $n!$ avec n le nombre d'états. (Nous n'avons pas représenté les noeuds de profondeur 3 sur ce schéma)

Une deuxième idée est de limiter la profondeur d'exploration de l'arbre. Comme vu sur la Figure 2 il serait possible de ne pas considérer l'ensemble des états et de se limiter à une profondeur k. Néanmoins la complexité exploserait encore avec $\binom{n}{k}$ possibilités.

La dernière idée serait d'octroyer une fréquence moyenne d'observation pour l'agent. A chaque observation il déterminerait le nombre optimum d'éléments à récupérer avant de recommencer une observation. Le nombre d'éléments récupérés ainsi que le gain réalisé à chaque parcours serait mémorisé. Il limiterait alors le parcours de son arbre à un certain nombre de noeuds optimal. En contrôlant le nombre de noeuds observé, on contrôle donc la complexité.

La prochaine étape est de parcourir un tel arbre afin de trouver une séquence d'états optimale pour satisfaire au mieux le but de l'agent, soit ici maximiser le nombre de points obtenus.

4.2 Exploration non-informée : Recursive Depth-Limited Search

Nous implémentons tout d'abord une exploration non-informée. La recherche d'un chemin optimal doit alors être réalisée avec uniquement les informations de l'arbre. Les informations internes des états ne peuvent être utilisés, ce qui veut dire que chaque branche de l'arbre sera analysée de la même manière.

Nous avons choisi la méthode de Depth-limited recursive search. Le principe s'initialise comme celui de la recherche en profondeur limitée. L'arbre est implémenté à l'aide de la pile de récursivité. Nous analysons une séquence complète d'états. Puis intervient la récursivité : nous remontons l'arbre en comparant chaque branche pour chaque noeud et choisissons celle qui nous permet au final la meilleure séquence dans le sens de la maximisation de points. L'exploration s'arrête lorsque toutes les branches de l'arbre limité en profondeur sont analysées.

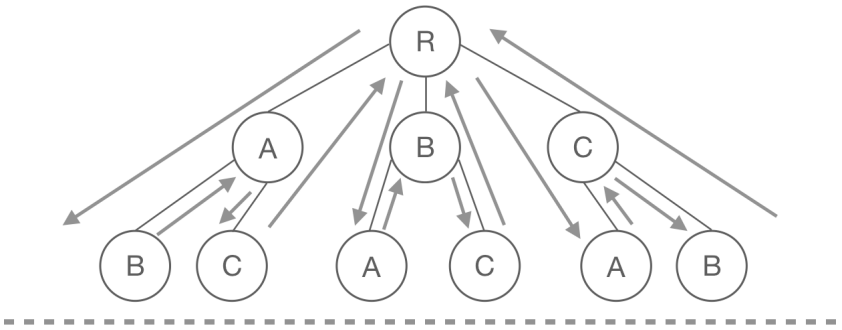

Figure 3 – Représentation de Recursive Depth-Limited Search

La figure 3 montre un exemple d'exploration pour 3 éléments A,B et C, avec une profondeur de 2. Nous remarquons qu'un élément parcouru est mémorisé afin de ne pas être estimé à nouveau. Nous limitons ainsi légèrement la complexité temporelle.

Les propriétés de cette exploration sont les suivantes :

- Incomplète : nous limitons l'exploration à une certaine profondeur ;
- Temps : $O(n^k)$ malgré le fait de ne pas reconsidérer au niveau de profondeur suivant un noeud déjà parcouru ;
- Espace : $O(n * k)$;
- Optimalité : non à cause de la limitation en profondeur ;

4.3 Exploration informée : Greedy Search

L'objectif est ici de guider le parcours de l'arbre grâce à une heuristique jusqu'à un point donné. Ce dernier est un élément de la grille choisi aléatoirement parmi ceux captés par l'agent. Nous le dénommerons par la suite comme la *cible*.

Dans le cadre d'une exploration Greedy Search, l'heuristique est calculée à chaque noeud. Elle consiste au calcul des bénéfices possibles pour chaque branche sous la forme :

```
heuristique = pointsObtenusAprèsAction - coûtDeLAction - coûtDuTrajet
```

le coût du trajet étant estimé par une distance de Manhattan. Chaque noeud parcouru est ensuite mémorisé afin de ne plus le prendre en compte lors du prochain calcul d'heuristique. En effet, pour satisfaire au sujet, chaque élément aspiré ou ramassé doit ensuite disparaître.

Précisons que cette exploration est réalisée pendant l'observation de l'agent et donc avant toute action. La planification de son itinéraire est mémorisée sous forme de liste. Ce qui implique toujours que l'environnement peut ajouter de nouveaux éléments qu'il ne détecterait pas pendant ses actions.

L'exploration est donc une forme de Greedy Search avec mémorisation des noeuds parcourus :

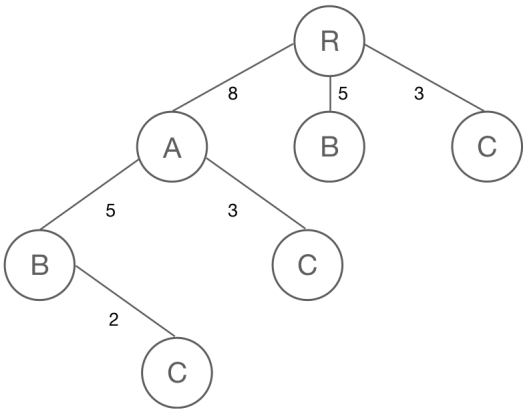

FIGURE 4 – Représentation de Greedy Search avec mémorisation des noeuds

Nous voyons sur la Figure 4, pour 3 éléments A,B et C la cible, l'exploration réalisée par l'agent jusqu'à sa cible. Dans son désir de maximiser ses points il préférera au premier noeud passer par A, puis par B, pour enfin arriver à C. Notons que si le chemin optimal était de passer par C directement, à défaut d'atteindre peu d'éléments, permettra une nouvelle observation.

Les propriétés inhérentes à notre algorithme d'exploration sont alors :

- Complet : grâce à la mémorisation des noeuds aucune boucle n'est possible et la cible sera toujours atteinte, c'est la vérification double-état ;
- Temps : $O(n^2)$ dans le pire des cas, c'est-à-dire en parcourant tous les éléments détectés, et en testant chaque prochain noeud en dernier. Le nombre d'opération est alors :

$$n + (n-1) + (n-2) + ... + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Espace : $O(n)$ car nous mémorisons les noeuds parcourus ;
- Optimalité globale : non car nous ne choisissons que localement la meilleure solution ;
- Optimalité locale : oui ;

4.4 Vers un agent apprenant : optimisation de la fréquence d'observation

Pour comprendre comment la fréquence d'observation a été modélisée, rappelons sa définition et justifions-la par un exemple :

La fréquence d'observation désigne le nombre d'éléments à atteindre avant de procéder à une nouvelle observation de l'environnement.

Exemple : L'agent observe l'environnement, il décide d'un itinéraire à effectuer. Dès lors qu'il a traité un nombre d'éléments égal à la fréquence d'observation, il démarre un nouveau cycle. Bien sûr, si le nombre d'éléments

dans cet itinéraire est inférieur à la fréquence, l'agent procède à une nouvelle observation une fois ses actions terminées.

D'autre part voici les attributs concernés pour la modélisation des fréquences et le calcul de leur score :

- `int frequenceObs` : fréquence d'observation utilisée par l'agent ;
- `int nbElementCycle` : nombre d'élément ramassés durant le cycle ;
- `boolean debutCycle` : désigne si l'agent commence un nouveau cycle ou non ;
- `double[][] tabFrequence` : il s'agit d'un tableau de 2 lignes et de `FREQUENCE_MAX` colonnes, comme nous pouvons le voir sur cette figure :

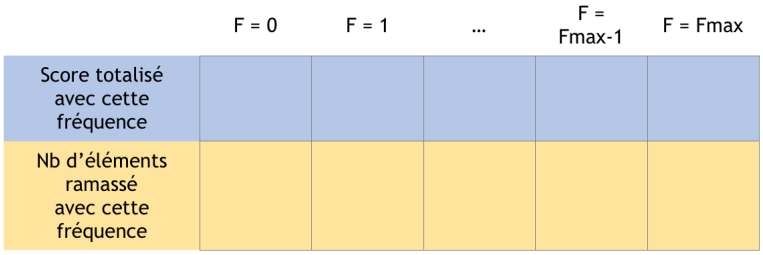

FIGURE 5 – Tableau utilisé par l'agent pour l'optimisation des fréquences

Sur la Figure 5, la $i^{\text{ème}}$ colonne correspond au caractéristiques de la fréquence $F = i + 1$. La première ligne correspond au score `nbrPointsGagnés - nbrPointsPerdus` totalisé par l'agent en utilisant cette fréquence. La deuxième ligne (en jaune) correspond au nombre total d'éléments ramassés par l'agent avec cette fréquence.

Par la suite la performance d'une fréquence est déterminée par le nombre de points rapporté en moyenne par éléments ramassé. Pour calculer la performance d'une fréquence `f` il suffit de faire le rapport :

```
tabFrequence[f-1][0] / tabFrequence[f-1][1]
```

Etudions désormais ce que réalise l'agent. Premièrement il doit effectuer des mesures à analyser. Pour cela lors du choix de la fréquence, si un nombre d'éléments récupérés avec une certaine fréquence est considéré comme insuffisant (`tabFrequence[f-1][1]<ECHANTILLON_MIN`), l'agent va choisir cette fréquence afin d'avoir un score plus précis pour celle-ci.

Lorsque toutes les fréquences ont été suffisamment utilisées par l'agent, celui-ci va simplement choisir celle avec la performance la plus haute.

Un début de cycle est détecté lorsque le nombre d'élément ramassé est égal à la fréquence d'observation utilisée (`frequenceObs == nbrElementCycle`) ou lorsque qu'il n'y a plus d'objectifs à atteindre. Lors d'un début de cycle, le score et le nombre d'éléments récupérés par l'agent vont être additionnés dans le tableau de fréquences de l'agent. Ainsi les statistiques sont mises à jour après chaque cycle où elles ont été utilisées.

5 Résultats obtenus

5.1 Choix des algorithmes d'exploration

Le choix des algorithmes d'exploration a été réalisé en prenant en compte le but de l'agent et l'environnement dans lequel il évolue.

La solution de Recursive Depth-Limited Search nous a semblé la plus adaptée à notre problème. Elle permet de réaliser un chemin optimal dans un espace d'état limité à une certaine profondeur définie.

D'autre part Greedy Search est apparu comme étant adpaté à notre problème, dans le sens où l'agent a un but de maximisation de points. Une optimalité locale est donc suffisante pour ce genre de problème.

5.2 Efficacité et efficience

La mise en place d'algorithmes d'exploration permet d'assurer l'efficacité et l'efficience de l'agent. Ces deux propriétés sont vérifiées du fait qu'il soit basé sur le but de maximiser ses points.

Il reste toutefois une variable à considérer : le nombre d'actions réalisées avant une nouvelle observation de l'environnement. La solution d'analyse des fréquences permet de répondre à ce problème en déterminant le nombre optimal d'éléments à aspirer ou ramasser avant une nouvelle observation, selon les résultats obtenus précédemment. Nous avons finalement implémenté un agent apprenant.

5.3 Complexité

Nous notons une grande différence de complexité entre les explorations informée et non-informée. En effet, la solution Recursive Depth-Limited Search présente une limite pour une probabilité d'apparition d'éléments trop importante. Dans ce cas le temps de calcul surpasse les apparitions, empêchant l'agent de planifier un trajet.

Afin de résoudre ce problème, nous avons mis en place un *safe mode* consistant à commuter l'agent en exploration informée, diminuant grandement la complexité temporelle. Notons que de cette manière nous améliorons également la complexité spatiale, ce qui démontre bien la supériorité de ce type d'exploration à défaut d'avoir une solution optimale.

Conclusion

Au cours de ce projet nous avons réussi à mettre en place un environnement et un agent fonctionnels et distincts, dont les intéractions sont directement observables grâce à une représentation graphique, et paramètrables grâce aux différents menus sur des fenêtres spécifiques. Nous avons particulièrement élaboré deux méthodes d'exploration, informée et non-informée, ainsi qu'une forme d'optimisation par la recherche intelligente d'une fréquence d'observation optimale, permettant de redéfinir une exploration adaptée à l'environnement.