Apprendre à naviguer en eaux turbulentes

Fabien ROGER

Résumé *Ce rapport étudie la possibilité qu'un agent virtuel puisse apprendre à se déplacer dans des courants turbulents. Plus précisément il est question ici d'une méthode qui permet de trouver une stratégie pour avancer dans l'eau en tirant parti des courants environnants. Pour cela, on présente le cadre général de l'apprentissage par renforcement. Le fonctionnement du Q-Learning, un algorithme qui permet la résolution du problème, est détaillé, ainsi qu'un élément de la preuve de sa convergence. La mise en pratique de l'algorithme pour le problème étudié pose des défis. On étudie la pertinence des réponses qu'on y apporte en analysant les résultats de l'algorithme quand on fait varier certains paramètres.*

Table des matières

1. Introduction à l'apprentissage par renforcement

1.1. Objectif de l'apprentissage par renforcement

L'approche classique pour diriger les déplacements d'un robot est de spécifier directement une fonction, appelée politique qui à un état donné attribue une action à effectuer. Mais cette approche ne peut être mise en oeuvre quand il est trop difficile de déterminer quelle est la meilleure action à effectuer, et on peut alors parfois utiliser à l'apprentissage par renforcement.

Dans l'apprentissage par renforcement, au lieu de créer une politique, on laisse l'agent expérimenter avec l'environnement, on évalue au fur et à mesure sa performance en lui donnant des récompenses quand le résultat de ses actions est proche du résultat souhaité, et on le laisse déterminer quelle est la politique qu'il doit employer pour maximiser ses récompenses, et donc accomplir l'objectif souhaité.

Du point de vue de l'agent, la situation se résume alors ainsi :

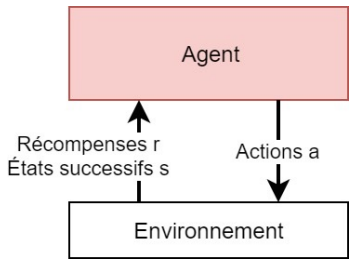

Figure 1. Schéma des interactions agent - environnement.

Il s'agit alors de déterminer une méthode qui détermine une "bonne politique" (dans un sens qu'on détaille plus tard) à partir des informations dont l'agent dispose : ses états successifs, les actions qu'il choisit et les récompenses qu'il reçoit.

1.2. Processus de décision markovien

Pour formaliser le problème, on introduit le concept de processus de décision markovien que l'on définit ainsi :
Un processus de décision markovien est un quadruplet $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R)$ où

- $\mathcal{S}$ est un ensemble d'états dans lequel l'agent peut se trouver
- $\mathcal{A}$ est un ensemble d'actions que l'agent peut réaliser
- $\mathbb{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0;1]\ ; (s, s', a) \mapsto \mathbb{P}_{s,s'}(a)$ où $\mathbb{P}_{s,s'}(a)$ est la probabilité d'attendre un état s' depuis un état s en ayant effectué l'action a. Elle est supposée indépendante des états précédents (i.e. l'état s caractérise intégralement l'environnement).
- $R : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}\ ; (s, s', a) \mapsto R_{s,a}^{s'}$ où $R_{s,a}^{s'}$ est la récompense obtenue lors du passage de s à s' en ayant effectué l'action a.

On se place ici dans le cadre d'un processus de décision markovien fini : $\mathcal{A}$ et $\mathcal{S}$ sont des ensembles finis.

1.3. Exemple de processus de décision markovien

On représente ci-dessous un exemple de processus de décision markovien simple à deux états et deux actions, où l'agent est un enfant qui peut soit jouer de la batterie soit lire, et qui aime réveiller son frère. On a choisi des récompenses R traduisant une légère préférence pour jouer de la batterie (récompense de 1), et une forte préférence pour réveiller son frère (récompense de 10). L'environnement est entièrement caractérisé par le fait que le frère est endormi ou éveillé, et on choisit des probabilités de transition $\mathbb{P}$ qui reflètent ce qui se passerait au bout de chaque heure. Une manière de représenter graphiquement le problème est alors d'afficher les fonctions $\mathbb{P}$ et R sur les arêtes d'un graphe pour chacune des actions possibles, comme on le ferait pour une marche aléatoire.

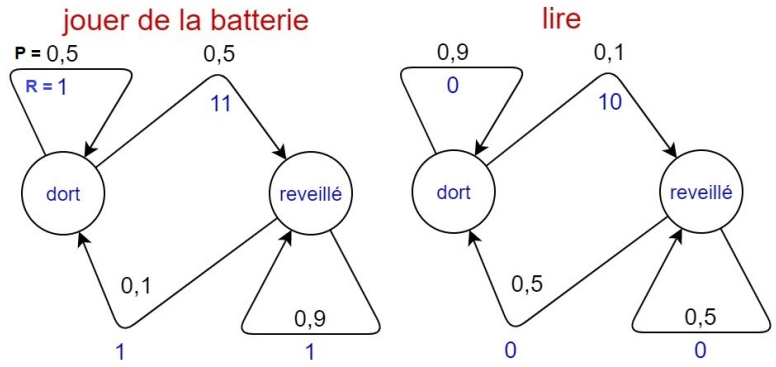

Figure 2. Schéma du processus de décision markovien : l'enfant et son frère

1.4. Fonction de qualité Q et politique optimale

Pour évaluer la qualité d'une politique π, on définit la matrice de qualité $Q^\pi \in M_{\mathcal{S},\mathcal{A}}(\mathbb{R})$:

$$Q^\pi(s,a) = \mathbb{E}\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ... | s_t = s, a_t = a\right)$$

Il s'agit de l'espérance des récompenses futures étant donnés un état de départ s et une action prise à cet instant a.

On introduit un coefficient de décroissance de l'importance des récompenses futures $\gamma \in [0;1[$ traduisant une tendance qu'on a à privilégier les actions immédiatement bénéfiques.

Ce qu'on souhaite alors déterminer est une politique optimale qui maximise l'espérance des récompenses futures, et donc chacune des cases de la matrice. Formellement, en notant π^* une politique optimale, on aura

$$\forall \pi \ \forall (s,a) \in \mathcal{S} \times \mathcal{A},\ Q^{\pi^*}(s,a) \geq Q^\pi(s,a)$$

Une telle politique existe toujours [1] et l'objectif est alors de concevoir un algorithme permettant de trouver une approximation de la matrice Q^* en interagissant avec l'environnement. Connaître cette matrice c'est en effet connaître la meilleure politique π^* puisque dans un état s il suffit de prendre l'action correspondant à qualité $Q^*(s,a)$ la plus élevée.

2. Le Q-Learning

2.1. Contexte

On se place dans une situation où l'agent connaît les états $\{s_t\}$ qu'il a parcouru, les actions qu'il a prises à chaque état et les récompenses qu'il a reçues lors de chaque transition, mais où les fonctions de transition $\mathbb{P}$ et R sont inconnues. On doit alors trouver un moyen d'inférer quelles sont les meilleures actions à prendre et quelle est la meilleure politique à partir des récompenses obtenues jusqu'au moment présent. Pour cela, on part d'une matrice Q, approximation de la matrice de qualité de la politique optimale Q^* définie plus haut, approximation qu'on affine au fur et à mesure.

Le Q-Learning et une méthode qui permet de trouver de cette manière une approximation de Q^*.

2.2. Trouver une relation permettant d'approximer Q^* : l'équation de Bellman

On trouve une relation permettant d'approcher Q^* en l'exprimant en fonction des états suivants qu'on peut atteindre et des récompenses qu'on obtiendra lors de ces transitions.
On trouve l'équation de Bellman :

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathbb{P}_{s,s'}(a) \left(R_{s,a}^{s_{t+1}} + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \right)$$

Informellement, quand on n'a pas Q^* mais son approximation Q le terme de droite, $R_{s,a}^{s_{t+1}} + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$, est une meilleure approximation que le terme de gauche car il prend en compte ce qu'on a appris lors de la transition de s à s' : la récompense $R_{s,a}^{s_{t+1}}$.
En faisant des petits pas dans la direction de cette meilleure approximation (le terme de droite), on peut ainsi espérer s'approcher au fur et à mesure de Q^*. D'où l'algorithme qui suit.

2.3. Présentation de l'algorithme

L'algorithme du Q-Learning est le suivant :

```
Q ← 0
s ← s_0
pour t = 1 à t_max faire
    a ← choisir action(Q,s)
    s', r ← pas dans l'environnement(a,s)
    Q ← mise à jour(Q,s,s',r)
fin
```

La mise à jour de la matrice se fait en se déplaçant avec une certaine vitesse d'apprentissage $\alpha \in [0;1]$ dans la direction d'une meilleure approximation, conformément à ce qu'on a expliqué un peu plus haut, ce qui donne la formule de

récurrence suivante :

Si $s = s_t$ et $a = a_t$

$$Q_{t+1}(s,a) = (1-\alpha)\, Q_t(s,a) + \alpha \underbrace{\left(r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a')\right)}_{Q \text{ corrigé}}$$

Sinon

$$Q_{t+1}(s,a) = Q_t(s,a)$$

On aimerait s'arrêter quand la matrice Q est suffisamment proche de Q^*, et donc qu'elle a arrêté de varier significativement d'une étape à l'autre. En pratique, à cause de variations importantes de Q dans les situations non idéales sortant du cadre strict du processus de décision markovien, on s'arrête plutôt au bout d'un temps $t_{\max}$ choisit en avance au bout duquel on constate graphiquement qu'en général l'agent ne fait plus de progrès.

2.4. L'exploration

L'exploration de l'environnement pouvant être très longue, on choisit de passer plus de temps à explorer les états et actions dont on suspecte qu'ils fournissent des récompenses les plus élevées, donc ceux pour lesquels $Q(s,a)$ est plus grand que pour les autres états et actions. Il faut néanmoins continuer à explorer l'environnement aux alentours des états les plus intéressants pour ne pas rester coincé dans une stratégie sous optimale.

Ici, on utilise une des méthodes de choix de l'action à effectuer relativement simple mais très répandue dite "ϵ-gourmande" : on choisit l'action dont on estime qu'elle est la meilleure (d'après notre matrice Q) avec une probabilité $(1-\epsilon)$, et sinon on choisit une action au hasard parmi les actions possibles, donc avec une probabilité ϵ. Dans les faits, on prend $\epsilon \approx 0,15$

2.5. Un mot sur la complexité

La complexité spatiale est en $O(|\mathcal{S}||\mathcal{A}|)$: on doit juste retenir la matrice Q, et on n'a pas besoin de retenir tout le passé de l'agent.

La complexité temporelle est en $O(t_{\max}|\mathcal{A}|)$ car les étapes de choix d'action et de mise à jour de la matrice de qualité sont en $O(|\mathcal{A}|)$. Mais pour savoir si c'est une complexité raisonnable, encore faudrait-il savoir quelle valeur de $t_{\max}$ est nécessaire. La théorie qui permet de l'estimer est trop compliquée pour pouvoir être abordée ici. On se contente simplement de relever expérimentalement la durée au bout de laquelle la convergence semble avoir lieu.

2.6. Exemple simple

Pour l'exemple de l'enfant et son frère, on peut calculer à la main les valeurs de Q^* pour les différentes politiques possibles. En appliquant l'algorithme du $Q-$ *Learning* à ce processus de décision markovien, chacun des 4 coefficient de la matrice Q converge bien vers le coefficient correspondant de la matrice Q^*.

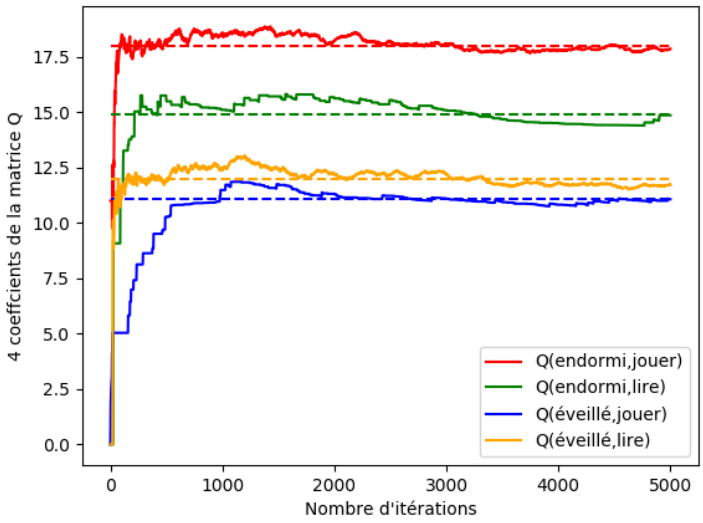

Figure 3. Évolution des valeurs de Q en fonction du nombre d'itérations.

3. Théorème de convergence

3.1. Énoncé

En appliquant l'algorithme décrit précédemment, on a le résultat suivant, démontré par Watkins en 1989 [2] :

— Si $\sum_{n=0}^{\infty} \alpha_n(s,a) = +\infty$ et $\sum_{n=0}^{\infty} \alpha_n(s,a)^2 < +\infty$
où n et le nombre de passage déjà effectué en (s,a).
Chaque (s,a) doit donc être visité une infinité de fois.
— Et $\gamma < 1$
— Et R est bornée

Alors $Q_t \underset{t\to+\infty}{\longrightarrow} Q^*$ presque sûrement.

3.2. Un élément de preuve

La démonstration est trop longue pour pouvoir la détailler, mais on peut comprendre pourquoi l'algorithme fonctionne en étudiant une partie de la preuve. On montre qu'en espérance, les corrections qu'on effectue au fur et à mesure ont tendance à rapprocher Q de Q^* [3].

Plus formellement pour $a \in \mathcal{A}$ et $s \in \mathcal{S}$ on définit, en voyant l'état s' comme une variable aléatoire

$$Q_{\text{corr}}(s,a,s') = R_{s,a}^{s'} + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

Alors la relation de récurrence devient $Q(s,a) \longleftarrow \alpha Q(s,a) + (1-\alpha) Q_{\text{corr}}(s,a,s')$
Et alors

$$||\mathbb{E}\left(Q_{\text{corr}} - Q^* | Q\right)||_\infty \leq \gamma ||Q - Q^*||_\infty$$

où $||\mathbb{E}\left(Q_{\text{corr}} - Q^* | Q\right)||_\infty = \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) \left(R_{s,a}^{s'} + \gamma \max_{a'\in\mathcal{A}} Q(s',a') - Q^*(s,a) \right) \right|$

Cela découle à la fois du fait qu'en partant de deux politiques Q_1 et Q_2 on a, d'après un calcul détaillé en annexe,

$$||\mathbb{E}\left(Q_{\text{corr}1} - Q_{\text{corr}2} | Q_1, Q_2\right)||_\infty \leq \gamma ||Q_1 - Q_2||_\infty$$

Et du fait que Q^* est un de point fixe de l'opération. En effet ce que dit l'équation de Bellman sur la politique optimale :

$$Q^* = \mathbb{E}\left(Q^*_{\text{corr}}\right) = \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) \left(R_{s,a}^{s_{t+1}} + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a') \right)$$

3.3. Utilisation de l'élement de preuve

Ne connaissant pas la nature de nos déplacements dans l'environnement, à l'exception du fait qu'on passe avec une probabilité 1 une infinité de fois dans chacun des états, il est très difficile de passer de ce rapprochement de Q et de Q^* "en espérance" à une véritable preuve de convergence. On peut néanmoins se ramener, grâce aux éléments de preuves ci-dessus, à la situation que traite un théorème de Tommi Jaakkola et Michael I. Jord [3], un peu plus général, et dont la quantité $Q_t - Q^*$ vérifie les conditions. Il garantit alors que du moment que $\sum_{n=0}^{\infty} \alpha_n(s,a) = +\infty$ et $\sum_{n=0}^{\infty} \alpha_n(s,a)^2 < +\infty$, $\gamma < 1$ et R bornée, on a bien $Q_t - Q^* \underset{t\to+\infty}{\longrightarrow} 0$ presque sûrement, et donc que l'algorithme converge.

4. Application à la navigation dans des eaux turbulentes

4.1. Modélisation informatique du problème

On souhaite qu'un agent remonte des courants. On espère en effet qu'on peut faire mieux que d'aller simplement dans la direction souhaitée si on a à notre disposition des informations sur les courants à proximité. Il est difficile de savoir a priori quelles stratégies pourraient fonctionner, et on utilise ici le Q-Learning pour trouver une stratégie pour remonter le courant. Plus précisemment, il doit avancer le plus loin possible dans le sens $\vec{x}$ en un temps donné. L'objectif étant de montrer qu'une telle approche peut fonctionner, on ne se soucie pas du réalisme du modèle physique employé.

Le modèle physique employé est le suivant : l'agent est un point matériel dans un plan en 2 dimensions doté d'une direction. On a généré au préalable des cartes de courants avec à Lily Pad. Lily Pad est l'implémentation dans le le langage Processing (Java avec une librairie graphique) d'un algorithme de résolution des équations de Navier-Stokes en 2 dimensions [4]. L'agent doit se déplacer dans ces courants et il n'a pas d'influence sur les courants eux-mêmes. On considère une évolution par incréments de temps discrets δt : l'agent choisit l'angle duquel il va tourner $\delta\theta \in \{-\omega\delta t, 0, \omega\delta t\}$ et la norme de sa vitesse $v \in \{v_1, v_2\}$. On note ω la vitesse à laquelle l'agent peut tourner, $\vec{V}(\vec{M}(t), t)$ la vitesse des courants à sa position $\vec{M}$, θ l'angle entre $\vec{x}$ et sa direction $\vec{u}(\theta)$. L'évolution de sa position et de sa direction est alors régie par les équations :

$$\theta(t+\delta t) = \theta(t) + \delta\theta$$
$$\vec{M}(t+\delta t) = \vec{M}(t) + (\vec{V}(\vec{M},t) + v\vec{u}(\theta))\delta t$$

On fait varier progressivement le champ des vitesses des courants $\vec{V}$ pour imiter une véritable évolution des courants. Ne cherchant pas à être réaliste, toutes les distances évoquées par la suite seront exprimées en unités arbitraires.

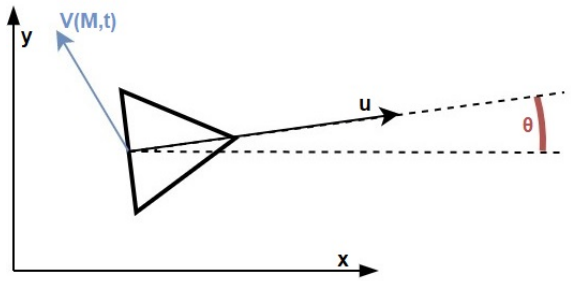

Figure 4. Modélisation physique.

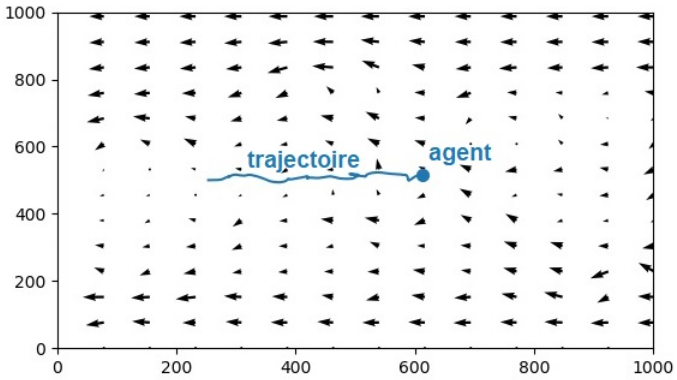

Figure 5. Capture d'écran d'un agent remontant le courant.

4.2. Modélisation du problème dans le cadre d'un processus de décision markovien

On veut se placer dans le cadre d'un processus de décision markovien fini. On a déjà défini l'ensemble des actions, qui est fini : il s'agit de $\mathcal{A} = \{-1, 0, 1\} \times \{v_1, v_2\}$. Il s'agit maintenant de définir les états dans lesquels il peut se trouver. On veut à la fois que l'agent ait à sa disposition des informations suffisantes sur son environnement : l'angle avec la direction dans laquelle il doit se déplacer, l'angle qu'il forme avec les courants sur lui et un peu autour de lui, et la force de ces courants. Mais le nombre de ces états doit être fini, et raisonnablement petit pour que l'algorithme converge en un temps raisonnable. Pour résoudre ce problème, on peut employer une méthode communément employée (par exemple par [5, 6]) : discrétiser les angles. On transforme cette grandeur réelle (ou plus rigoureusement ce nombre flottant qui peut être dans 2^{64} états différents s'il est stocké sur 8 octets) en un entier de taille plus raisonnable pour l'utilisation qu'on en fait. Pour cela, on ne transmet pas à l'agent un angle $\alpha \in [0; 2\pi[$, mais plutôt $\left\lfloor \frac{2\pi\alpha}{n} \right\rfloor \in [|0; n-1|]$ et on choisit $n = 6$. On peut procéder de la même manière pour discrétiser la force d'un courant.

L'ensemble des états choisis a donc la forme $\mathcal{S} = [\![1, 10]\!] \times [\![1, 6]\!] \times [\![1, 6]\!] \times [\![1, 2]\!]$ et celui des actions $\mathcal{A} = [\![1, 5]\!] \times [\![1, 2]\!]$ soit $|\mathcal{S} \times \mathcal{A}| = 7200$: c'est le nombre de coeffcients à approcher, qui impose déjà un temps de calcul relativement grand (de l'ordre de la dizaine de minutes). En effet, il faut plusieurs de millions d'itérations pour parvenir à une solution satisfaisante.

Il faut aussi choisir quelles sont les récompenses qu'on donne à l'agent, le coefficient γ de décroissance de l'importance des récompenses futures. On choisit $r_t = x_{t+1} - x_t$ et $\gamma = 0,9 \simeq 1$ pour avoir $Q^*(s, a) = \mathbb{E}\left(r_t + \gamma r_{t+1} + ...\right) \simeq x_{t+T} - x_t$ (avec $T = 1/(1-\gamma)$) qui est la distance parcourue dans un futur proche.

4.3. Mise en pratique

On entraîne 40 agents avec les paramètres indiqués pendant $4,5 \times 10^6$ pas sur une carte de courant. Ces entraînements nécessitent en pratique plusieurs heures de calcul sur un ordinateur personnel tournant avec Python. Ils font 50 tests d'une durée de 500 itérations sur 50 cartes de courants différentes et on relève quelle distance ils ont parcourue. Ces cartes sont différentes de celles employées pour l'entraînement. On compare la moyenne des distances atteintes lors de ces 50 tests, qu'on appelle par la suite "performance", à celle d'un algorithme "naïf" qui avance en ligne droite dans la direction cible. Les performances des agents se répartissent par rapport à l'algorithme naïf de la façon suivante :

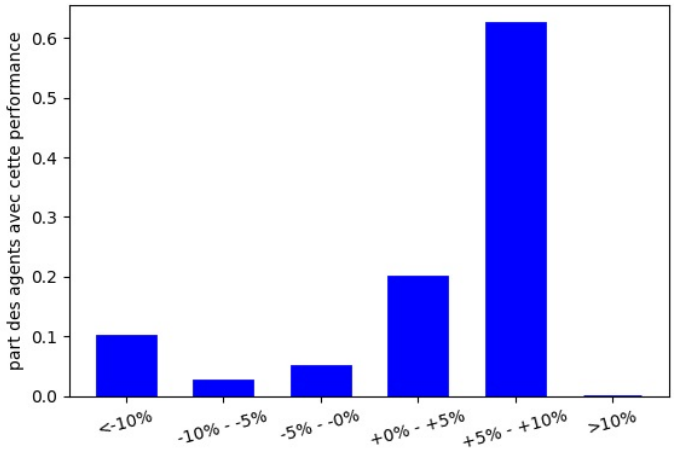

Figure 6. Réparatition des performances des agents.

On lit sur cet histogramme que 60% des agents ont parcourue une distance supérieure à celle parcourue par l'algorithme naïf d'entre 5% et 10% de cette distance, qu'ils battent l'alogrithme naïf d'un écart entre 5% et 10%. Le Q-Learning permet bien de trouver une solution meilleure que l'algorithme naïf. Étant donné qu'il est complexe de trouver une meilleure stratégie "à la main" que cet algorithme naïf, le Q-Learning permet bien d'apporter une solution satisfaisante à ce problème.

On remarque en revanche qu'on a une forte variabilité des résultats des différentes agents : certains agents sont restés "coincés" dans de mauvaises stratégies : environ 15% des agents ont une performance moins bonne que celle de l'agorithme naïf. On peut expliquer ces variations par le fait qu'on est sorti du cadre strict du

processus de décision markovien où l'état de l'agent et de l'environnement est entièrement caractérisé par $s \in \mathcal{S}$, ce qui n'est manifestement pas le cas ici. On n'a donc ici pas la garantie qu'on ait trouvé la stratégie optimale.

4.4. Étude de l'influence d'un paramètre de l'agent

On peut se demander quel impact la nature des informations qu'on donne à l'agent a sur ses performances. Pour évaluer cela, on se propose de faire varier la distance de prévisualisation "pdist" entre l'agent et le lieu où le sens des courants est relevé.

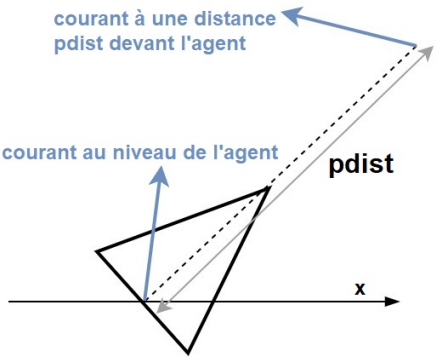

Figure 7. Ce que l'agent voit du monde.

On évalue alors la performance de l'algorithme pour une gamme de valeurs de "pdist" exactement de la manière indiquée ci-dessus. Ici on s'intéresse à la performance de l'agent dont la performance est la médiane des autres pour avoir une idée de la performance globale des agents, sans être influencé trop fortement par les résultats des agents coincés dans de mauvaises stratégies. L'incertitude indiquée est l'incertitude (à 95%) sur la valeur moyenne (sur ses 50 tests) de la performance de cet agent médian.

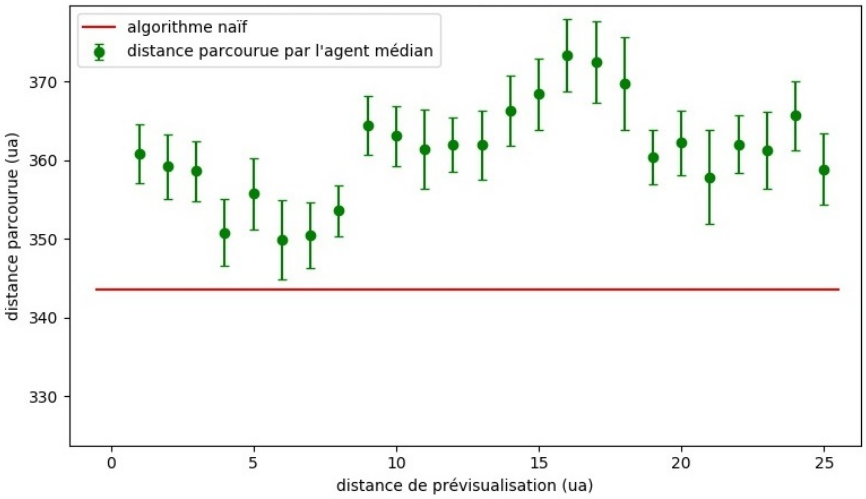

Figure 8. Performance médiane en fonction de la distance de prévisualisation

On observe qu'un maximum est atteint aux alentours de pdist = 15 u.a. et diminue plutôt fortement si pdist est trop grand ou trop faible. Comme on s'y attendait, la nature des informations fournies a une influence importante sur les performances des agents. Ces mesures ont aussi pour intérêt de nous indiquer quelles sont les informations utilisées par l'agent. Celles-ci sont en effet généralement difficiles à déterminer puisque l'agent ne peut pas nous expliquer quelle stratégie il emploie. Ici, puisque "pdist" a une influence importante sur les performances, on sait donc que le sens des courants devant l'agent est une information pertinente.

4.5. Étude de l'influence de la finesse de la discrétisation

On peut se demander quel impact les simplifications faites lors de la modélisation du problème dans le cadre d'un processus de décision markovien fini a sur les performances de l'agent. Pour évaluer cela, on se propose de faire varier la finesse de la discrétisation des angles entre les courants et l'agent : au lieu d'associer à chaque angle une entier dans $[|1;6|]$ on lui associe un entier dans $[|1;4|]$ (moins de finesse, moins d'informations), $[|1;6|]$ ou $[|1;8|]$ (plus grand finesse, plus d'informations), et on évalue les performances des agents pour chacun de ses niveaux de finesse, de la même manière qu'on l'a fait quand on fait au paragraphe précédent.

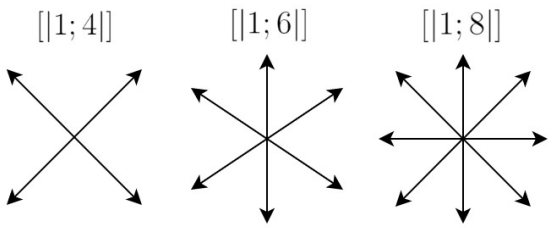

Figure 9. Différentes niveaux de finesse de la discrétisation

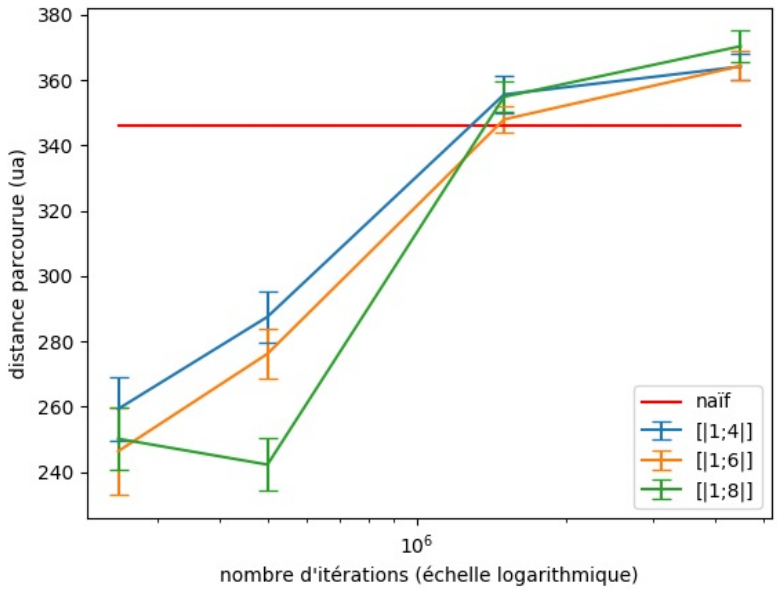

Figure 10. Évolution de la performance pour différents niveaux de discrétisation des angles

On remarque que même avec une discrétisation peu fine (pour $[|1;4|]$) l'agent a de meilleures performances que l'algorithme naïf (on passe par dessus la ligne rouge). Il est même surprenant qu'on arrive à des résultats aussi bons. Les performances finales sont quand même moins bonnes que pour des discrétisations plus fines pour $[|1;8|]$), comme on s'y attendait : l'agent arrive bien à prendre parti des informations plus précises qu'on lui donne. En revanche l'agent met plus de temps à atteindre de bonnes performances. Lors de la résolution d'un problème avec le Q-Learning, il faut donc arriver à trouver un équilibre entre la finesse de la modélisation et le nombre d'itérations qu'on est prêt à calculer.

5. Conclusion

5.1. Discussion de l'efficacité du Q-Learning

Il a été démontré que le Q-Learning converge vers la meilleure solution dans les conditions d'un processus de décision markovien, et on a observé cette convergence dans un cas simple. Dans le cas plus complexe et intéressant d'un point de vue pratique de la navigation en eaux turbulentes, l'algorithme fournit des solutions satisfaisantes même si l'étude menée sur la discrétisation nous indique qu'il est peu probable que la solution optimale au problème puisse être trouvée dans un temps de l'ordre de l'heure sur un ordinateur personnel. Le Q-Learning souffre en effet d'une explosion du nombre d'états et donc de la durée d'apprentissage dès qu'on augmente la finesse des entrées ou des sorties.

5.2. Le Q-Learning et les autres méthodes d'apprentissage par renforcement

Pour remédier à ses limitations, on peut utiliser d'autres méthodes d'apprentissage par renforcement, capable par exemple de transférer des apprentissages entre états "proches". Le Q-Learning est alors une méthode de base sur laquelle de nombreuses autres méthodes ont été construites. Le nombre et la performance de ces méthodes ont explosé lors de ces dernières années, notamment avec le développement de l'apprentissage profond qui tire parti des puissances de calcul grandissantes et du développement des connaissances sur les réseaux de neurones pour arriver à des performances exceptionnelles. Par exemple, une méthode baptisée le Deep Q-Learning développée par Deep Mind permet à un agent d'apprendre à jouer à la plupart des jeux sortis sur la console Atari [7], et d'autres méthodes encore plus poussées ont permis de créer une intelligence artificielle capable de battre les meilleurs joueurs de Go [8].

6. Annexe

Calcul de l'équation de Bellman

$$
\begin{aligned}
&Q^*(s,a) = \mathbb{E}\left(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a\right) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbb{P}_{s,s'}(a) \mathbb{E}\left(r_t + \gamma r_{t+1} + ... | s_t = s, a_t = a, s_{t+1} = s'\right) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbb{P}_{s,s'}(a) \left(\mathbb{E}\left(r_t\right) + \gamma \mathbb{E}\left(r_{t+1} + ... | s_t = s, a_t = a, s_{t+1} = s'\right)\right) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbb{P}_{s,s'}(a) \left(R_{s,a}^{s_{t+1}} + \gamma \mathbb{E}\left(r_{t+1} + ... | s_{t+1} = s', a_{t+1} = a^*\right)\right) \\
&= \textstyle\sum_{s' \in \mathcal{S}} \mathbb{P}_{s,s'}(a) \left(R_{s,a}^{s_{t+1}} + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')\right)
\end{aligned}
$$

Calcul de la contractence en espérance

$$
\begin{aligned}
&||\mathbb{E}\left(Q_{\text{corr1}} - Q_{\text{corr2}} | Q_1, Q_2\right)||_\infty \\
&= \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) \left(\gamma \max_{a'\in\mathcal{A}} Q_1(s',a') - \gamma \max_{a'\in\mathcal{A}} Q_2(s',a') \right) \right| \\
&\leq \gamma \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) \left| \max_{a'\in\mathcal{A}} Q_1(s',a') - \max_{a'\in\mathcal{A}} Q_2(s',a') \right| \\
&\leq \gamma \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) \max_{a'\in\mathcal{A}} |Q_1(s',a') - Q_2(s',a')| \\
&\leq \gamma \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathbb{P}_{s,s'}(a) ||Q_1 - Q_2||_\infty \\
&= \gamma ||Q_1 - Q_2||_\infty
\end{aligned}
$$

Paramètres de la simulatin simple : $\epsilon = 0,15$, $\gamma = 0,8$, $\alpha_n = 1/n^{0,7}$

Paramètres de la simulatin complète : $\epsilon = 0,15$, $\gamma = 0,9$, $\alpha_n = 1/n^{0,8}$

Références

[1] Andrew Barto et Richard Sutton : *Reinforcement Learning : An Introduction*. MIT Press, 2018.

[2] Chris Watkins : *Learning from Delayed Rewards*. Thèse à l'Université de Cambridge, 1989.

[3] Tommi Jaakkola, Michael I. Jordan et Satinder P. Singh : On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:10–13, 1994.

[4] Gabriel D. Weymouth : Lily pad : Towards real-time interactive computational fluid dynamics, 2015. `https://arxiv.org/pdf/1510.06886.pdf`.

[5] Gautam Reddy, Antonio Celani, Terrence J. Sejnowski et Massimo Vergassola : Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences*, 113:33, 2016.

[6] Simona Colabrese, Kristian Gustavsson, Antonio Celani et Luca Biferale : Flow navigation by smart microswimmers via reinforcement learning. *Physical Review Letters*, 118:2–3, 2017.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra et Martin Riedmiller : Playing atari with deep reinforcement learning, 2013. `https://arxiv.org/pdf/1312.5602.pdf`.

[8] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel et Demis Hassabis : Mastering the game of go without human knowledge. *Nature*, 550:354—359, 2017.