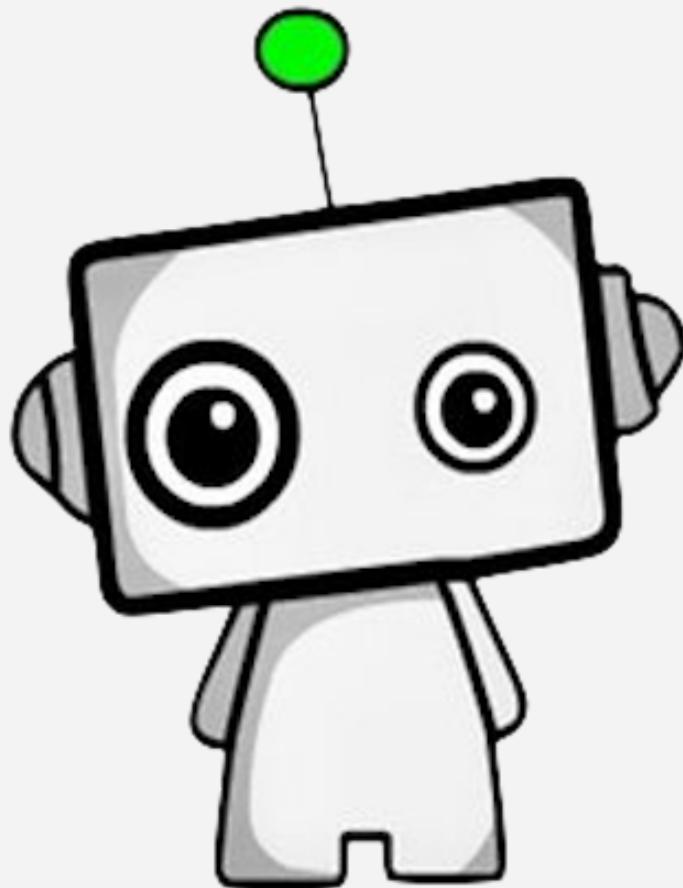


Rapport final

# Curi'agent

---

LAVENSEAU Louis



# SOMMAIRE

<b>CONTEXTE</b>	<b>2</b>
Description du sujet	2
Définitions	3
<b>CHOIX TECHNIQUES</b>	<b>3</b>
Technologies utilisées	3
Choix concernant l'environnement	4
Actions	4
Récompenses extrinsèques	5
Conditions venteuses	6
Motivation intrinsèque utilisée	6
Motivation intrinsèque sélectionnée	6
Implémentation	7
<b>AVANCEMENT DU CODE</b>	<b>8</b>
Étapes d'avancement	8
Codage d'un algorithme d'apprentissage par renforcement simplifié	8
Agrandissement de l'environnement et variation des récompenses/puniti	9
Implémentation de la Q-Function	10
Agrandissement de l'environnement et variation des récompenses/puniti	11
Implémentation d'une méthode de motivation intrinsèque et comparaison	12
Implémentation du vent	13
Implémentation du vent changeant	22
Implémentation d'un réseau de neurones pour remplacer la Q-table	23
Planning de travail	26
<b>PISTES D'AMÉLIORATION</b>	<b>27</b>

# I. CONTEXTE

## 1. Description du sujet

Le sujet tel que décrit dans le cahier des charges consiste en l'implémentation d'un algorithme d'apprentissage par renforcement et d'une récompense intrinsèque qui permette à un agent de trouver le plus court chemin entre deux points dans l'environnement du [projet IA du semestre 7](#). Cet environnement comprend un plateau de taille 300\*300, avec différentes conditions venteuses et coordonnées de point de départ et d'arrivée proposées selon trois cas :

“

- **cas a** :  $(x_0, y_0) = (100, 200)$  ;  $(x_f, y_f) = (200, 100)$  et le vent est constant à 50km/h et souffle dans la direction 30° (vers le nord-est).
  - **cas b** :  $(x_0, y_0) = (100, 200)$  ;  $(x_f, y_f) = (200, 100)$ .  
Si  $y > 150$ , le vent est constant à 50km/h et souffle dans la direction 180° (vers l'ouest).  
Si  $y \leq 150$ , le vent est constant à 20 km/h et souffle dans la direction 90° (vers le nord)
  - **cas c** :  $(x_0, y_0) = (200, 100)$  ;  $(x_f, y_f) = (100, 200)$ .  
Si  $y > 150$ , le vent est constant à 50km/h et souffle dans la direction 170° (vers l'ouest).  
Si  $y \leq 150$ , le vent est constant à 20 km/h et souffle dans la direction 65° (vers le nord-est).
- ”

L'idée est d'approximer cet environnement et de comparer les stratégies de l'algorithme A\* du projet du semestre dernier et de l'algorithme d'apprentissage par renforcement profond avec motivation intrinsèque de ce projet. Je suis aussi passé par des environnements intermédiaires simplifiés pour mieux appréhender chaque composante que ce projet contient. Ils sont détaillés dans la partie *III.1 Étapes d'avancement du code*.

J'ai implémenté une motivation intrinsèque (méthode du buffer) qui est décrite dans les parties *II.3 Motivation intrinsèque retenue* et *III.1 Étapes d'avancement* .

J'ai essayé de coder des fonctionnalités supplémentaires (vent changeant, apprentissage par renforcement profond, et obstacles) après avoir comparé les performances des deux projets, avec plus ou moins de succès. Elles sont décrites dans la partie *III.1.g),h),f)*.

## 2. Définitions

**Apprentissage par renforcement** : technique d'apprentissage automatique qui consiste à faire apprendre à un agent des actions dans un environnement en lui faisant maximiser une récompense quantitative.

**Etat** : combinaison de caractéristiques pertinentes vis-à-vis de l'environnement considéré qui définissent la situation de l'agent.

**Action** : manière d'agir de l'agent qui peut avoir une conséquence sur l'environnement et le faire changer d'état.

**Espérance de récompense d'un couple état-action** : quantification de la proximité d'un état à des récompenses ou punitions. Elle est calculée grâce à la Q-Function.

**Q-Table** : tableau multidimensionnel dans lequel sont stockées l'ensemble des espérances de récompense des couples état-action de l'environnement.

**Motivation intrinsèque** : ensemble des facteurs poussant un individu à atteindre un objectif indépendamment de toute considération extérieure. Dans l'apprentissage par renforcement, la motivation intrinsèque est utile quand les récompenses sont rares : elle consiste en l'ajout de récompenses intrinsèques qui poussent l'agent à se diriger vers les couples état-action qu'il connaît le moins.

**Exploitation** : choisir l'action qui maximise l'espérance de récompense (extrinsèque ou extrinsèque+intrinsèque)

**Exploration** : choisir l'action de manière aléatoire, ou en maximisant l'espérance de récompense intrinsèque.

# II. CHOIX TECHNIQUES

## 1. Technologies utilisées

J'ai utilisé le langage Python car étant interprété, il est particulièrement adapté à l'apprentissage par renforcement. Il m'a aussi permis d'avoir accès à des bibliothèques qui

m'ont facilité l'implémentation, notamment au niveau de l'interface graphique. Sur les cinq premiers fichiers, j'ai travaillé avec l'IDE Idle. Je l'ai choisi car il m'était familier. J'ai aussi utilisé Numpy pour manipuler plus facilement les tableaux. Lors de l'implémentation du vent, j'ai eu besoin d'avoir un retour graphique sur différents éléments :

- la distribution des récompenses extrinsèques dans l'environnement
- les actions optimales déterminées pour chaque état
- le déplacement de l'agent en direct
- le plus court chemin trouvé par l'agent après l'apprentissage

J'ai alors décidé d'utiliser les bibliothèques Matplotlib et Seaborn. Je me suis aussi mis à utiliser le notebook de Google Colab pour pouvoir manipuler plus aisément les différents scripts d'affichage post-entraînement après les apprentissages de l'algorithme. J'ai voulu utiliser Colab aussi pour la partie apprentissage par renforcement profond qui peut demander de l'hardware de pointe que je n'ai pas. Pour cette partie, j'ai utilisé TensorFlow .

## 2. Choix concernant l'environnement

### a) Actions

Pour coller au projet IA du semestre 7, j'ai repris le même plateau de taille 300\*300, décrit par une matrice de même taille. Le but de ce projet était de trouver le plus court chemin entre deux points en termes de temps. A chaque déplacement effectué, le temps mis était calculé. Le chemin total trouvé était évalué selon la somme des temps pris pour chaque déplacement. Avec l'apprentissage par renforcement, l'unité considérée est le nombre d'actions. C'est selon lui que les espérances de récompense, et donc l'évaluation de la pertinence de passage par un état, vont être distribuées. Ce nombre d'états peut être traduit comme étant une durée (exemple : une action représente une minute). Cependant, l'environnement considéré dans le projet IA du semestre 7 est un plateau. Un déplacement d'une case à l'autre y est traduit comme une distance. En gardant ce même plateau pour le projet Curi'agent, il est donc nécessaire de garder des actions de même distance. Sinon, chaque action ne correspondra pas à la même durée. Réduire le plus possible le nombre d'actions, ce que permet de faire l'apprentissage par renforcement, ne sera alors pas forcément synonyme de réduire le plus possible le temps de parcours mis pour atteindre la cible. Lors d'un déplacement en diagonal, la distance parcourue, égale à  $\sqrt{2}$ , est plus grande que lors d'un déplacement selon les deux axes, où la distance est égale à 1. Pour travailler seulement avec des coordonnées entières, j'ai donc fait le choix de permettre à

l'agent de ne se déplacer qu'exclusivement avec les quatre directions : est, ouest, nord et sud.

Pour ce qui est de l'exploration et de l'exploitation, le pourcentage d'exploration (epsilon) évolue au fur et à mesure de l'apprentissage : les conditions venteuses permettent des parcours optimaux autres que juste aller tout droit vers l'état final. En début d'apprentissage, l'agent découvre une majorité de ces possibilités avec beaucoup d'explorations. Plus l'apprentissage avance, moins il a besoin d'explorer. Tout de même, j'ai remarqué que garder un pourcentage inférieur à au moins 66% d'exploration permettait d'avoir un apprentissage plus rapide. Arrivé à ce pourcentage environ, epsilon reste donc constant

### b) Récompenses extrinsèques

Pour orienter l'agent vers le point final, j'ai associé à cet état une récompense extrinsèque. Lors des différents scripts intermédiaires, j'ai eu l'occasion d'implémenter des punitions et récompenses entre l'état initial et final pour bien appréhender l'apprentissage par renforcement. Pour le script qui traduit l'environnement du projet IA du semestre 7, je n'ai laissé que cette récompense finale pour répondre au mieux à la problématique du projet.

Pour propager les espérances de récompenses extrinsèques, j'ai utilisé la Q-Function. Voici ci-après son expression :

$$Q(s, a)_{new} = Q(s, a)_{old} + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s, a)_{old}],$$

avec :

- $(s, a)$  : le couple état-action considéré.
- $Q(s, a)_{new}$  : la nouvelle espérance de récompense extrinsèque calculée pour le couple état-action  $(s, a)$  considéré.
- $Q(s, a)_{old}$  : l'ancienne espérance de récompense extrinsèque pour le couple état-action  $(s, a)$  considéré.
- $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$  : l'espérance de récompense extrinsèque maximale des couples état-actions de l'état suivant.
- $\alpha$  : ce coefficient module le poids de chaque ajout de récompense extrinsèque à la récompense extrinsèque déjà associée au couple état-action considéré.

- $\gamma$  : facteur d'amortissement. Plus il est faible, moins les espérances de récompenses des états éloignés du couple état-action considéré vont avoir de poids comparé à celles des états proches.

J'ai choisi une valeur d'alpha égale à 0,1 pour que chaque nouveau calcul de récompense n'évolue pas beaucoup. De cette manière, les différences d'espérances récompense entre les couples état-actions des états peu et beaucoup parcourus sont minimisés.

La valeur de gamma, quant-à-elle, est de 0,99. Comme il n'existe qu'un seul îlot de récompenses dans l'environnement, il n'y a pas de raisons de donner plus d'importance aux états proches plutôt qu'à ceux éloignés : toutes les actions doivent être choisies pour se rapprocher, en termes de nombre d'actions restantes, des seuls états finaux.

### c) Conditions venteuses

Comme expliqué dans la partie *III.1.f) Implémentation du vent*, j'ai implémenté le vent sous forme de coefficient qui "pousse" l'agent plus ou moins long lors d'un déplacement, selon la force du vent. Ainsi, les deux vitesses présentes dans le sujet IA du semestre 7 de 20 et 50 km/h sont traduites respectivement en un déplacement de deux et trois cases. Les différentes directions données ont été simplifiées : comme les quatre déplacements possibles sont vers l'est, l'ouest, le nord et le sud, une différence de dix ou vingt degrés n'aura pas forcément d'impact. Ainsi, seules les indications verbales (exemple : ouest, sud-est) ont été prises en compte. Les directions de vent selon un des quatre points cardinaux poussent l'agent seulement dans la direction considérée, tandis que les directions entre deux points cardinaux poussent l'agent dans les deux directions considérées. Si l'agent se déplace face au vent, il peut quand même avancer d'une case. Sinon, par exemple, il ne pourrait pas se déplacer du tout vers l'est si le vent soufflait vers l'ouest (ce qui pourrait poser des problèmes de convergence), car seulement quatre déplacements sont possibles.

## 3. Motivation intrinsèque utilisée

### a) Motivation intrinsèque sélectionnée

A la suite de mes recherches, j'ai trouvé une manière d'implémenter des récompenses intrinsèques, adaptée à mon environnement, présentée ci-après. Elle est facilement programmable avec un environnement discret comme le mien et confère une

bonne amélioration des performances de l'algorithme. Les autres méthodes que j'ai trouvées sont pour la plupart soit anciennes (2002) et sont donc trop simples pour être scalables à des environnements plus complexes (moins intéressant), soit sont difficilement apprénendables à mon niveau de maîtrise de l'apprentissage par renforcement. J'ai donc décidé de n'utiliser qu'elle.

Présentation de la méthode :

L'idée est de stocker des états éparses dans un buffer :

"Le module de curiosité épisodique ECO [Savinov et al., 2018] approfondit cette idée en s'inspirant de la mémoire épisodique. Le modèle proposé contient un module de comparaison capable de renvoyer un bonus si l'agent est proche ou loin des états contenus dans un buffer. Ainsi, il calcule la probabilité que le nombre d'actions nécessaire pour aller de l'état sélectionné dans le buffer à l'état courant soit inférieure à un seuil. En stockant des états éparses dans le buffer, l'agent pose des points de repères dans l'environnement et essaye de s'éloigner de ceux-ci, cela revient à partitionner l'environnement. La probabilité que l'agent soit écarté de chaque état du buffer est utilisée pour calculer la récompense intrinsèque. "

Source : <https://hal.archives-ouvertes.fr/hal-02272091/document>

### b) Implémentation

J'ai implémenté le buffer sous forme d'une liste de vecteurs (50,2). J'ai décidé de ne garder que cinquante états en mémoire pour ne pas surcharger l'apprentissage de calculs. L'ordre de grandeur du nombre a été trouvé empiriquement. Après que le buffer soit devenu complet, il continue de stocker de nouveaux états. Mais il enlève alors l'état stocké le plus ancien, de manière à avoir un buffer "glissant". Cela permet à l'agent d'avoir toujours accès à des informations à jour. Ensuite, j'avais au départ trouvé empiriquement les meilleures performances pour un ajout de chaque état dans le buffer avec une probabilité de 0.5 (stratégie présente dans le fichier *Etape5\_Motiv\_Intrinsic*). Je n'avais pas essayé d'ajouter chaque nouvel état parcouru au buffer car le module original ne le faisait pas. Finalement, il se trouve que j'arrive à augmenter les performances de cette manière. Chacun des états parcourus est donc ajouté au buffer dans les fichiers qui suivent. Concrètement, plus les états sont ajoutés souvent, plus la mémoire des états parcourus est courte, et plus ils sont ajoutés rarement, plus la mémoire est longue. A partir du script d'implémentation du vent, avoir une mémoire courte pousse l'agent à ne rester que peu de temps au même endroit. Comme il peut facilement se retrouver bloqué (il dérive), la mémoire courte l'aide plus rapidement à se débloquer. La mémoire la plus courte possible

est donc plus appropriée pour le projet. Enfin, l'environnement étant simple, l'écart entre l'état sélectionné et ceux du buffer se calcule directement (sans passer par des probabilités). Plutôt que de calculer la norme, l'algorithme calcule la somme du nombre d'actions à réaliser sur les abscisses et sur les ordonnées. C'est un choix que j'ai fait pour rester proche de l'idée du module de curiosité épisodique, qui estime l'écart en termes de nombre d'actions (même si le résultat reste le même).

Après avoir implémenté le vent, j'ai décidé de garder la même manière de calculer la distance en actions aux états du buffer : j'additionne la distance sur les abscisses et les ordonnées entre l'état de l'agent et ceux du buffer. Effectivement, si l'agent décide d'aller dans le sens du vent et fait donc un déplacement de trois cases en une action et que je considère que la distance en actions entre le nouvel état et l'état précédent est de 1 (car il n'a fait qu'une action), l'agent considérera qu'il est à un écart d'une action de cet état précédent. Or, s'il veut revenir vers cet état précédent, il se retrouvera face au vent, et devra réaliser trois actions dans ce sens pour y revenir. De plus, je calcule les distances selon les deux axes à partir de la case adjacente à l'agent du côté de la direction considérée plutôt que sur la réelle case où l'action vers l'emmener. Par exemple, si le vent souffle vers l'ouest et pousse de trois cases, je vais calculer l'espérance de récompense intrinsèque de l'action "aller vers l'ouest" en prenant l'écart entre la case juste à gauche de l'état courant et celles du buffer au lieu de prendre la case à trois cases de l'état courant (où l'agent va réellement se retrouver s'il choisit cette action). Je fais cela car sinon comme j'ajoute tous les états dans le buffer, l'agent cherche dès la première action à s'éloigner de l'état initial et privilégie donc les actions qui vont dans le sens du vent par rapport aux autres, ce qui doit éviter de faire la motivation intrinsèque.

## III. AVANCEMENT DU CODE

### 1. Étapes d'avancement

J'ai décidé d'avancer étape par étape avant d'arriver à un algorithme d'apprentissage par renforcement avec motivation intrinsèque fonctionnel qui ait des performances satisfaisantes. A chaque étape, le(s) objectif(s), l'environnement, la manière d'orienter l'agent, les critères d'appréciation et les résultats sont décrits. Jusqu'au III.1.e) *Implémentation d'une méthode de motivation intrinsèque et comparaison*, le texte est identique à celui du rendu intermédiaire.

a) Codage d'un algorithme d'apprentissage par renforcement simplifié

Nom du script : *Etape1\_ARSimplifie.*

Objectif(s) :

- Appréhender l'apprentissage par renforcement.
- Comprendre l'intérêt de la Q-Function.

Environnement :

- Plateau 3\*3.
- L'agent commence en [0,0] et doit finir en [2,2].
- Une punition de 1 en [1,1] et une récompense de 1 en [2,2].

Orientation de l'agent :

L'algorithme donne l'information des récompenses et punitions des cases seulement à celles qui leur sont adjacentes.

Critères d'appréciation :

- A la fin de l'apprentissage, l'agent doit pouvoir aller en [2,2] en évitant la case [1,1]. S'il arrive soit dans la case [1,2], soit dans la case [2,1], il doit directement aller en [2,2].
- L'apprentissage doit être optimal : à chaque fois que l'agent peut apprendre quelque chose sur l'environnement, il l'apprend, et ne l'oublie pas jusqu'à écrasement des données => vérification à l'affichage de la Q-table à chaque étape de l'apprentissage.

Résultats :

Critères vérifiés.

b) Agrandissement de l'environnement et variation des récompenses/punitiōns

Nom du script : *Etape2\_ARSimplifie10x10.*

Objectif(s) :

- Tester la scalabilité de l'algorithme.
- Comprendre l'intérêt de backpropager les espérances de récompenses sur tout le plateau.

Environnement :

- Plateau  $10 \times 10$ .
- L'agent commence en [0,0] et doit finir en [9,9].
- Une récompense de 1 en [9,9] et des punitions de 1 sur chaque case de la diagonale principale pour le premier test et une récompense de 1 en [9,9] et des récompenses de 1 sur chaque case de la diagonale principale pour le deuxième test.

Orientation de l'agent :

Idem.

Critères d'appréciation :

- A la fin de l'apprentissage, pour le premier test, l'agent doit aller en [9,9] en évitant les cases de la diagonale principale. S'il arrive soit dans la case [8,9], soit dans la case [9,8], il doit directement aller en [9,9]. Pour le deuxième test, l'agent doit se comporter de la même manière, cette fois-ci en empruntant seulement les cases de la diagonale principale (et celles voisines).
- Apprentissage optimal : idem que dans le 1.

Résultats :

Critères vérifiés pour le premier test. Néanmoins quand il n'est ni à côté d'une case de la diagonale principale ou de la dernière case (donc dans la majorité des états), son déplacement est aléatoire => nécessité de commencer à backpropager les espérances de récompense sur chaque case du plateau pour l'orienter constamment vers la sortie.

Pour le deuxième test, l'agent se retrouve bloqué autour de la case [1,1] => même conclusion.

**c) Implémentation de la Q-Function**

Nom du script : *Etape3\_Q-Function.*

Objectif(s) :

Coder un réel algorithme d'apprentissage par renforcement.

Environnement :

Idem que dans le 1.

Orientation de l'agent :

Utilisation de la Q-Function pour "étaler les récompenses et punitions" sur l'ensemble des cases du plateau.

Critères d'appréciation :

- A la fin de l'apprentissage, l'agent doit pouvoir aller en [2,2] en évitant la case [1,1].  
Cette fois-ci, il doit emprunter le plus court chemin (ne pas faire de retour arrière).
- Apprentissage optimal : idem que dans le 1.

Résultats :

Critères vérifiés.

**d) Agrandissement de l'environnement et variation des récompenses/puniti**  
**ons**

Nom du script : *Etape4\_Q-FunctionAgrandi*.

Objectif(s) :

- Tester la scalabilité de l'algorithme d'apprentissage par renforcement avec Q-Function.
- Comprendre l'intérêt d'implémenter la motivation intrinsèque (ou curiosité).

Environnement :

- 10\*10 puis différentes tailles jusqu'à 300\*300.
- Tests de différentes punitions et récompenses sur le chemin. Récompenses de différentes valeurs sur la case du coin opposé au départ de l'agent.

Orientation de l'agent :

Utilisation de la Q-Function pour “étaler les récompenses et punitions” sur l'ensemble des cases du plateau.

Critères d'appréciation :

- Idem que dans le 3. Le chemin emprunté à la fin de l'apprentissage par l'agent est considéré comme appréciable s'il diminue la norme entre sa case et la case finale à chaque nouvelle action prise. Si des punitions sont mises sur le chemin, l'agent ne doit pas aller dessus. Si des récompenses sont mises sur le chemin, l'agent doit passer dessus.
- Les tailles des environnements deviennent trop grandes pour vérifier minutieusement que l'apprentissage est optimal. J'ai regardé l'évolution des valeurs pour quelques cas et considéré que l'apprentissage était suffisamment optimal s'il prenait un temps raisonnable (moins de 10 minutes environ), si chaque couple état/action avait une espérance de récompense à son terme, et si le premier critère était vérifié.

### Résultats :

Critères vérifiés. Dans le cas de récompenses intermédiaires cependant, j'ai remarqué que l'agent restait bloqué si leur valeur n'était pas inférieure aux récompenses présentes après (plus proches de la fin) dans l'environnement. J'ai dû donc mettre des valeurs de récompenses de plus en plus élevées au fur et à mesure que l'on se rapproche de la case finale. Au final, ne mettre une récompense que sur la case finale suffit pour que l'agent trouve cette case.

### e) Implémentation d'une méthode de motivation intrinsèque et comparaison

Nom du script : *Etape5\_MotivIntrinsic*. Au rendu intermédiaire en mars, je me suis rendu compte que l'apprentissage ne convergeait plus. J'ai réussi à résoudre le problème dans le fichier suivant.

### Objectif(s) :

- Avoir une récompense intrinsèque implémentée qui puisse accélérer l'apprentissage.

### Environnement :

- Différentes tailles pour appréhender la méthode jusqu'à l'environnement du projet IA du premier semestre : 300\*300, position initiale : [99,99], position finale : [199,199] (sans le vent implémenté).
- Une récompense sur la position finale.

### Orientation de l'agent :

Utilisation de la Q-Function pour "étaler les récompenses et punitions" sur l'ensemble des cases du plateau + utilisation de la somme des récompenses extrinsèques (de la Q-Function) et de la récompense intrinsèque calculée à l'aide de la méthode du buffer (première méthode décrite : "Nouveauté comme écart aux autres états") pour la phase d'apprentissage.

### Critères d'appréciation :

- J'ai décidé de procéder ainsi : l'agent découvre l'environnement jusqu'à arriver sur la case finale (= première phase d'apprentissage). Lors de cette phase d'apprentissage, l'agent parcourt aléatoirement l'environnement 90% du temps et 10% du temps (après quelques essais, il semble que ce pourcentage donne les meilleures performances) il se déplace en maximisant la somme de l'espérance de récompense extrinsèque et de l'espérance de récompense intrinsèque. Puis il revient au début (case initiale) et réalise une phase de test en tentant de trouver la case finale en

maximisant seulement les récompenses extrinsèques avec un nombre d'actions limité pour y parvenir. S'il n'y arrive pas, il retourne au début et refait une phase d'apprentissage. Puis une nouvelle phase de test. Il fait cela jusqu'à réussir à faire la phase de test (on considère que s'il réussit la phase de test, alors il connaît suffisamment l'environnement donc on arrête l'apprentissage). Un compteur est incrémenté à chaque nouvelle phase d'apprentissage réalisée. Plus le compteur est faible à la fin, plus l'apprentissage est considéré comme efficace. Pour comparer à quand aucune motivation intrinsèque n'est utilisée, j'ai fait le même test en utilisant seulement les récompenses extrinsèques lors des 10% du temps de la phase d'apprentissage. Les performances de motivation intrinsèque sont considérées comme appréciables si le compteur a une valeur bien inférieure quand les récompenses intrinsèques sont utilisées (minimum 10% de moins en comparant les moyennes sur 5,6 lancements).

- idem que dans le 4.

#### Résultats :

Critères vérifiés. Avec motivation intrinsèque, le nombre de phases d'apprentissage nécessaire est d'environ 270 contre 700 sans.

#### f) Implémentation du vent

Nom du script : Etape6\_Vent.

#### Objectif(s) :

- Correspondre à l'environnement décrit dans le cahier des charges.
- L'agent doit trouver le chemin le plus court en termes de temps et non de nombre d'actions ou de distance (il doit favoriser le déplacement le plus possible dans le sens du vent).
- La motivation intrinsèque doit permettre à l'agent de converger plus rapidement. Si les conditions venteuses ont tendance à "bloquer" (si le vent souffle fort vers une direction) l'agent sur une extrémité du plateau, elle doit aussi lui permettre de se débloquer.

#### Environnement :

- Idem que dans le 5. avec le vent en plus. Initialement, j'avais implémenté le vent sous forme de récompense/punition conférée au couple état/action choisi à chaque déplacement, selon si celui-ci va, ou non, plus ou moins dans le sens du vent. Cela n'a pas donné de résultats satisfaisants. Finalement, après discussion avec mon tuteur, j'ai décidé d'intégrer le vent sous forme de déplacements plus ou moins grands selon la direction du déplacement de l'agent en fonction du vent.

- L'état final a été transformé en un groupe d'états finaux, pour ne pas que l'agent "saute" au-dessus sans l'atteindre.
- Une récompense sur cet "îlot final".

Orientation de l'agent :

Idem que dans le 5.

Critères d'appréciation :

- Après la phase d'apprentissage, l'agent doit réussir à aller jusqu'à la case finale en prenant un chemin autre qu'en ligne. Ses déplacements doivent tirer avantage du vent.
- idem que dans le 4.

Résultats :

Critères vérifiés.

Dans le cas a, un chemin optimal ne présente que des déplacements vers l'est et le nord. Au début, l'agent fait un léger virage vers l'ouest. Cela est dû au fait que le chemin le plus court requiert 134 actions : 100 sur l'axe vertical et 34 sur l'axe horizontal. Faisant des déplacements de trois cases sur l'axe horizontal, il a un excédent de deux cases ( $3*34=102$ ). L'apprentissage s'est arrêté alors qu'il n'avait pas encore suffisamment parcouru l'environnement pour utiliser ces deux cases dans la direction des états finaux. Pour le reste, le chemin est optimal.



Figure 1 : chemin emprunté par l'agent pour le cas a

L'axe des ordonnées est orienté vers le bas, les coordonnées selon y se lisent donc de haut en bas, contrairement aux abscisses. En comparaison, l'algorithme A\* du projet IA

du semestre 7 proposait ce parcours optimal (pour les mêmes quatre possibilités de déplacement) :



Figure 2 : chemin emprunté par l'agent pour le cas a dans le cadre de mon projet IA du semestre 7

L'image a été retournée pour que l'axe des ordonnées soit aussi orienté vers le bas. Le parcours n'est pas non plus centré mais il ne s'agit que d'un problème d'incrustation d'image avec Winforms. Comme le vent est uniforme, la répartition des déplacements vers le nord et vers l'est lors du parcours n'a pas d'impact sur sa pertinence. Les deux stratégies sont donc équivalentes en termes de performances. A noter que pour l'algorithme d'apprentissage par renforcement, l'impact du vent sur la navigation est traduit par des sauts de cases, tandis que pour l'algorithme A\*, il amène à un temps de parcours entre deux noeuds (l'équivalent des états avec l'A\*) plus ou moins long. Ayant fait le choix d'avoir un environnement avec des coordonnées entières, j'ai dû traduire les deux vitesses de vent (20 et 50 km/h) par des sauts de cases de 2 et 3 et non par un calcul savant qui aurait permis d'implémenter parfaitement les paramètres venteux du projets IA du semestre 7. Les performances en termes de nombre de noeuds parcourus pour l'un et d'états pour l'autre ou le "temps" estimé de parcours ne peuvent donc pas être comparés. Pour ce qui est du temps de calcul, l'A\* trouve une solution en 0,02 secondes environ, et l'algorithme d'apprentissage par renforcement met autour d'une dizaine de minutes.

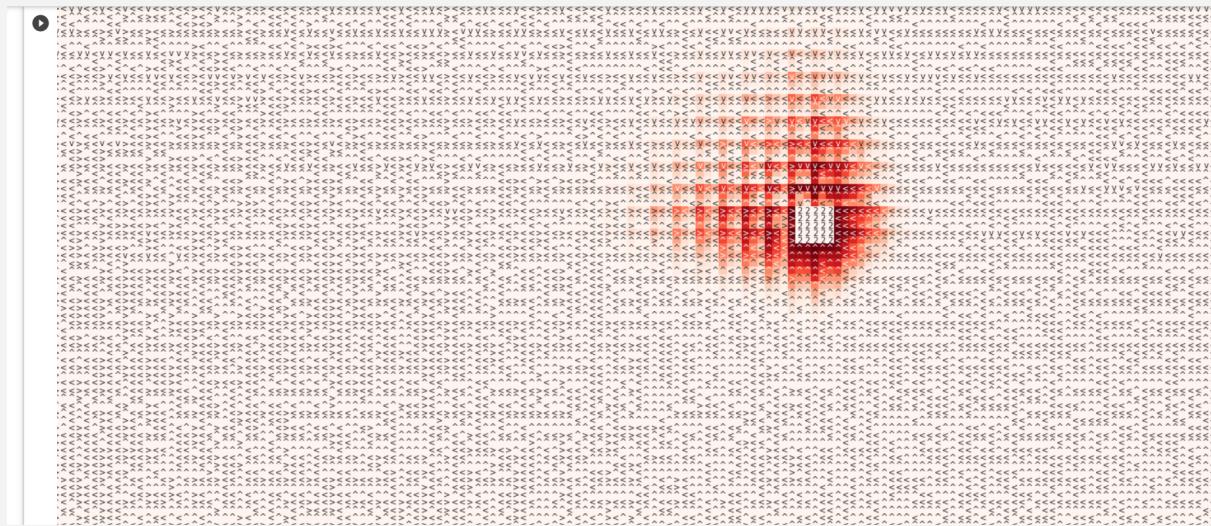


Figure 3 : distribution des récompenses extrinsèques et meilleures actions pour chaque état pour le cas a.

Plus les cases sont foncées, plus le couple état-action optimal (avec la plus grande espérance de récompense extrinsèque) associé à l'état correspondant a une espérance de récompense extrinsèque élevée. L'action correspondante est donc un déplacement vers l'ouest si l'annotation est “<”, vers l'est si elle est “>”, nord pour “^”, et sud pour “v”. Les récompenses extrinsèques ont peu été propagées loin des états finaux car l'apprentissage a été court par rapport à la taille de l'environnement (831 épisodes ici). Pour autant, elles l'ont été suffisamment pour que des actions optimales soient déterminées pour les états entre celui initial et ceux finaux.

Sans récompenses intrinsèques, l'agent se retrouve rapidement bloqué par le vent dans le coin supérieur droit du plateau, l'apprentissage ne converge donc pas :

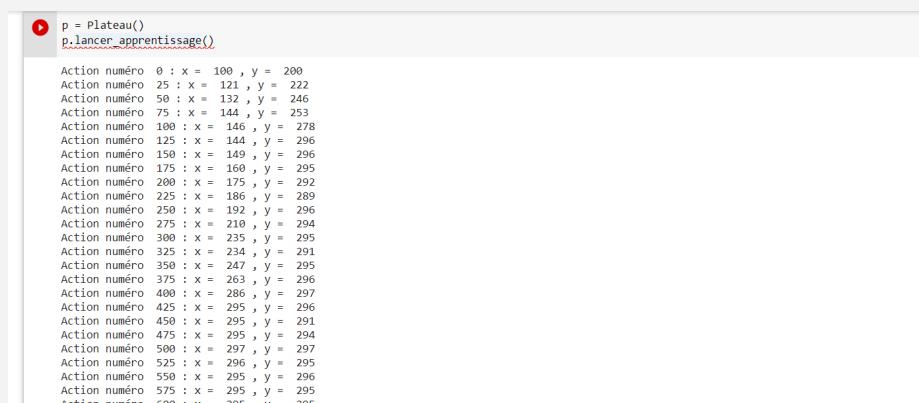
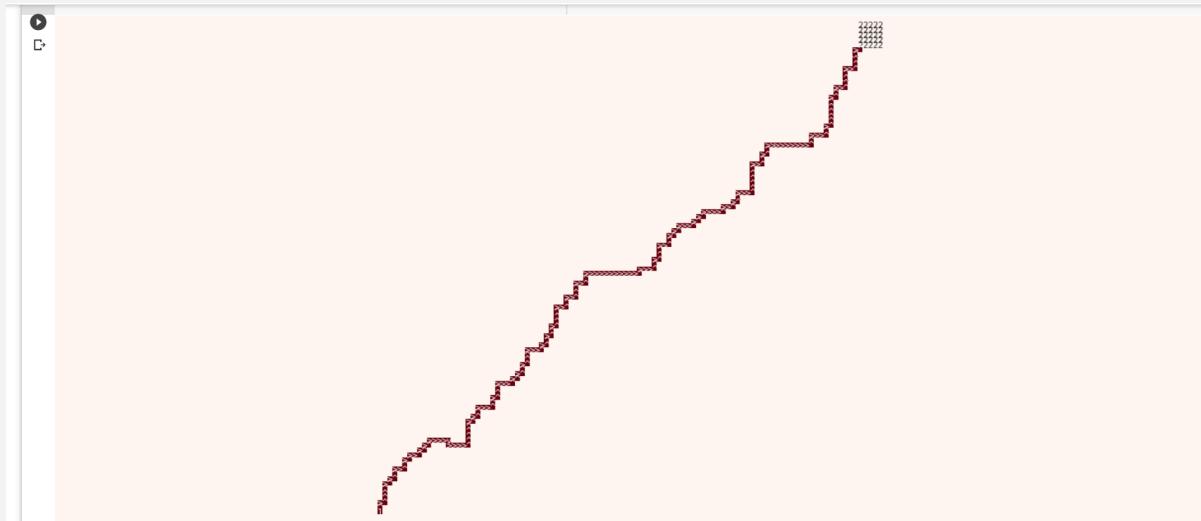


Figure 4 : coordonnées de l'agent pour un apprentissage sans récompenses intrinsèque dans le cas a.

Dans le cas b, un chemin optimal respecte les mêmes critères que pour le cas a. Ici, l'agent effectue un virage vers le sud qui n'est pas optimal. La raison est la même que pour

le cas précédent, mais cette fois-ci c'est dû au fait que l'agent a fini son parcours en (198, 198) au lieu de le finir en (200, 200), car il y a plusieurs états finaux. Pour le reste, le parcours est optimal.



*Figure 5 : chemin emprunté par l'agent pour le cas b.*



*Figure 6 : chemin emprunté par l'agent pour le cas b dans le cadre de mon projet IA du semestre 7.*

Comme l'apprentissage ne converge pas avec seulement quatre directions, j'ai ajouté les déplacement en diagonales à l'algorithme A\*. Mis à part ce détail, il respecte les mêmes conditions que l'algorithme d'apprentissage par renforcement. Les déplacements vers le nord et l'est ne sont pas distribués pareil, mais cela ne change rien aux performances car dans tous les cas l'agent se déplace face au vent. Il y aurait pu y avoir des stratégies plus intéressantes avec plus de directions accessibles.

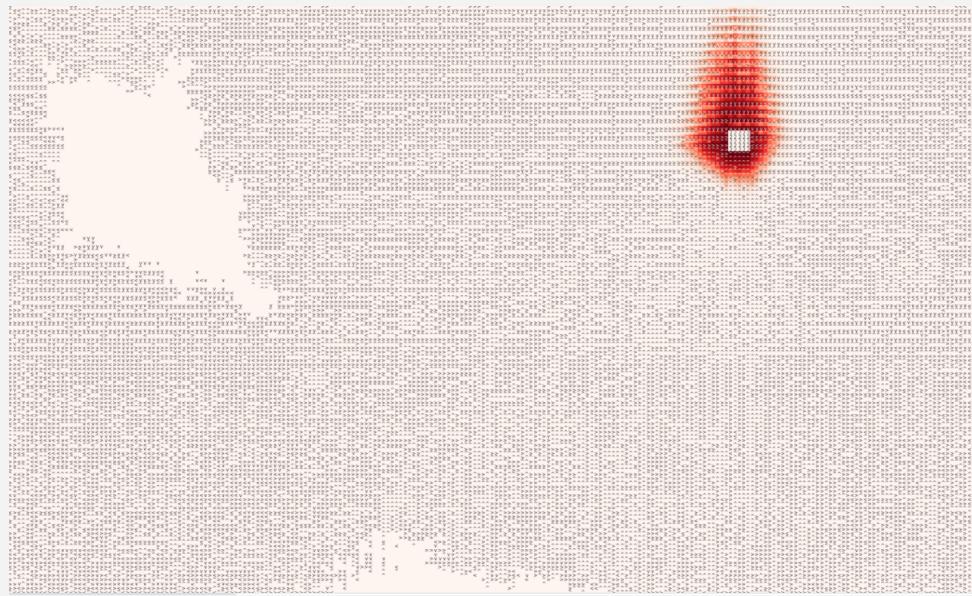


Figure 7 : distribution des récompenses extrinsèques et meilleures actions pour chaque état pour le cas b.

Sans récompenses intrinsèques, l'agent dérive rapidement vers le bord gauche du plateau. Il évolue alors sur l'axe des ordonnées : s'il descend en dessous de  $y = 150$ , alors il est repoussé vers le nord, et ainsi de suite. Il est donc bloqué, et l'apprentissage ne converge pas non plus :

ACTION NUMÉRO	x	y
Action numéro 421800 : x = 4 , y = 192		
Action numéro 421825 : x = 8 , y = 196		
Action numéro 421850 : x = 4 , y = 193		
Action numéro 421875 : x = 8 , y = 194		
Action numéro 421900 : x = 8 , y = 195		
Action numéro 421925 : x = 5 , y = 194		
Action numéro 421950 : x = 6 , y = 200		
Action numéro 421975 : x = 3 , y = 194		
Action numéro 422000 : x = 4 , y = 191		
Action numéro 422025 : x = 5 , y = 186		
Action numéro 422050 : x = 7 , y = 191		
Action numéro 422075 : x = 5 , y = 190		
Action numéro 422100 : x = 3 , y = 186		
Action numéro 422125 : x = 8 , y = 182		
Action numéro 422150 : x = 3 , y = 188		
Action numéro 422175 : x = 5 , y = 189		
Action numéro 422200 : x = 6 , y = 197		
Action numéro 422225 : x = 5 , y = 196		
Action numéro 422250 : x = 4 , y = 198		
Action numéro 422275 : x = 5 , y = 194		
Action numéro 422300 : x = 5 , y = 189		
Action numéro 422325 : x = 3 , y = 190		
Action numéro 422350 : x = 6 , y = 194		
Action numéro 422375 : x = 3 , y = 196		
Action numéro 422400 : x = 3 , y = 192		
Action numéro 422425 : x = 5 , y = 192		
Action numéro 422450 : x = 5 , y = 193		
Action numéro 422475 : x = 4 , y = 195		
Action numéro 422500 : x = 3 , y = 194		
Action numéro 422525 : x = 3 , y = 188		
Action numéro 422550 : x = 4 , y = 183		
Action numéro 422575 : x = 5 , y = 183		
Action numéro 422600 : x = 4 , y = 183		
Action numéro 422625 : x = 6 , y = 182		
Action numéro 422650 : x = 5 , y = 186		
Action numéro 422675 : x = 4 , y = 188		
Action numéro 422700 : x = 4 , y = 185		
Action numéro 422725 : x = 3 , y = 179		

Figure 8 : coordonnées de l'agent pour un apprentissage sans récompenses intrinsèque dans le cas b.

Pour le cas c, l'apprentissage met trop de temps à converger. Par conséquent, j'ai réduit la taille de l'environnement à 50, et adapté les coordonnées de l'état initial et des états finaux en conséquence. Le déplacement optimal présente des déplacements

exclusivement vers le nord sur la partie basse du plateau au début. Après avoir franchi l'équateur, les déplacements doivent être seulement orientés vers l'ouest et le nord.

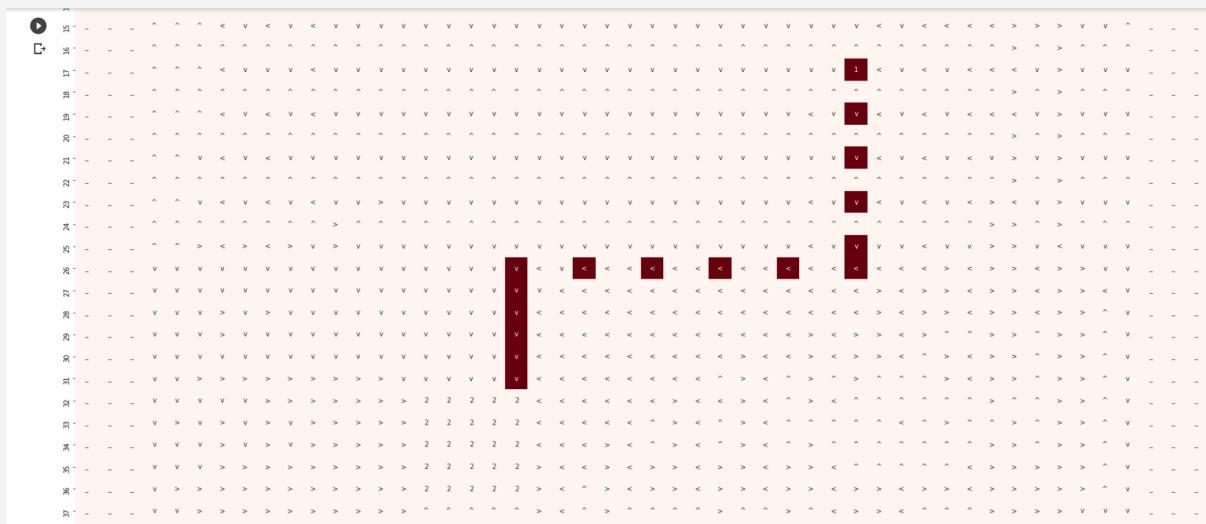


Figure 9 : chemin emprunté par l'agent pour le cas c avec un plateau de taille 50\*50.

L'agent a respecté les consignes données précédemment, le déplacement peut donc être considéré comme optimal.

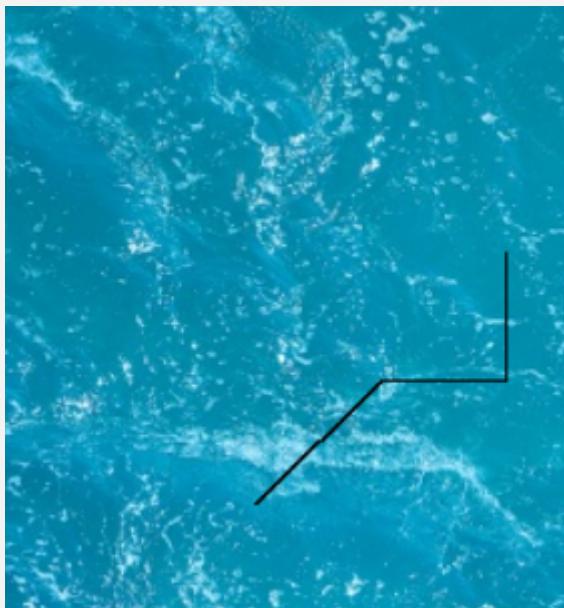


Figure 10 : chemin emprunté par l'agent pour le cas c dans le cadre de mon projet IA du semestre 7.

Sur la première partie du parcours, la stratégie est la même (déplacement seulement vers le nord). Pour la partie  $y > 25$ , la répartition des déplacements vers le nord et l'ouest n'ont pas d'importance. La répartition choisie par l'algorithme A\* est différente, mais ne présente ni d'avantages ou d'inconvénients. Les deux algorithmes ont donc les mêmes performances ici, excepté pour les temps de calcul qui sont similaires au cas a.

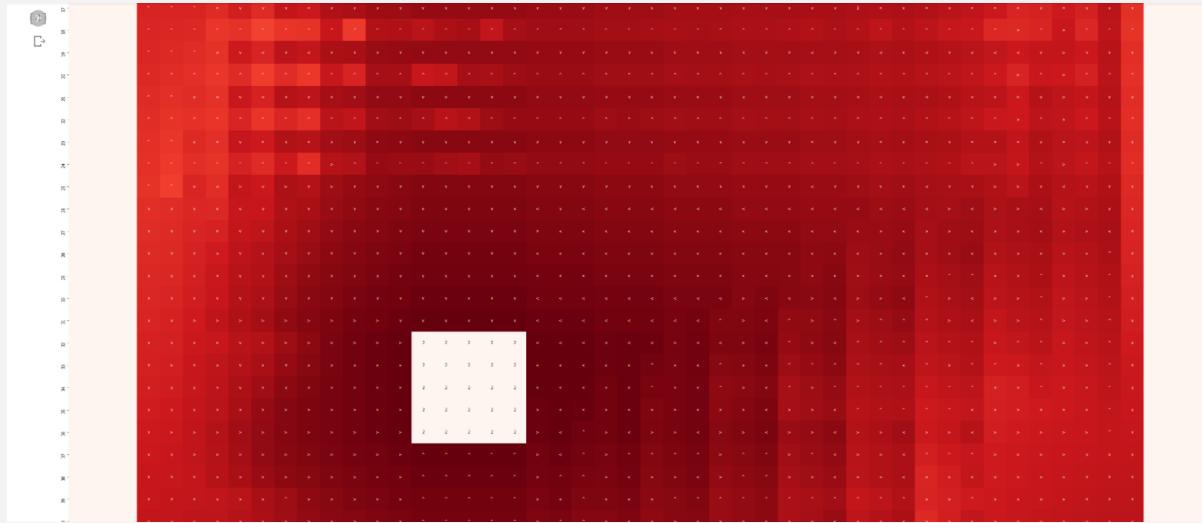


Figure 11 : distribution des récompenses extrinsèques et meilleures actions pour chaque état pour le cas c avec un plateau de taille 50\*50.

Comme l'environnement est petit, les espérances de récompenses extrinsèques se sont fortement étalées sur l'ensemble de l'environnement. On observe bien les distributions relativement homogènes trois par trois sur la droite des récompenses des états finaux et l'étalement de deux cases par deux cases des espérances de récompenses extrinsèques de la partie  $y > 25$  dans la partie  $y \leq 25$ .

Sans motivation intrinsèque, l'agent a aussi un parcours optimal (pour un nombre d'épisodes d'apprentissage plus faible) :

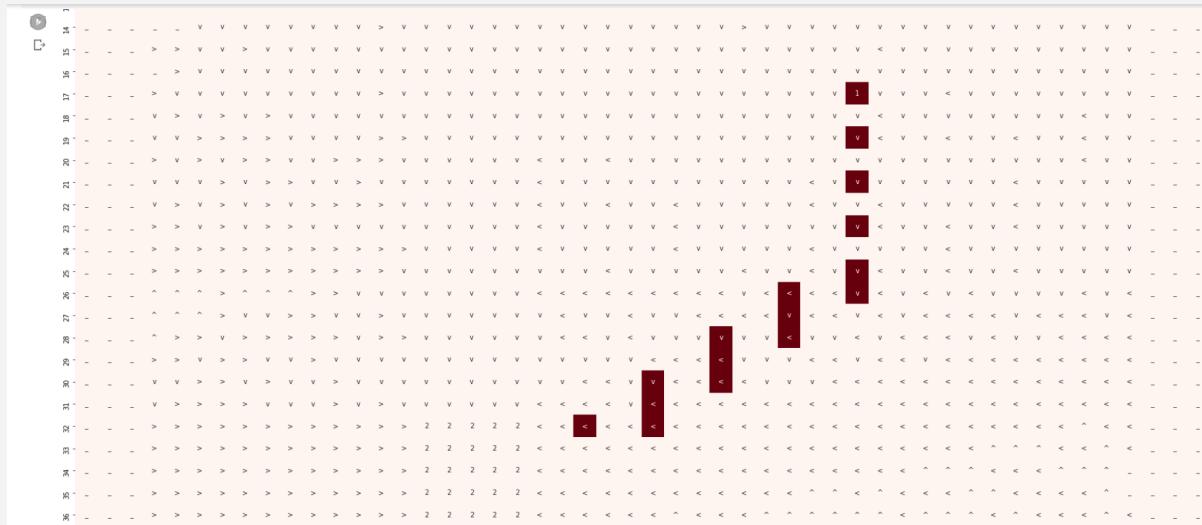
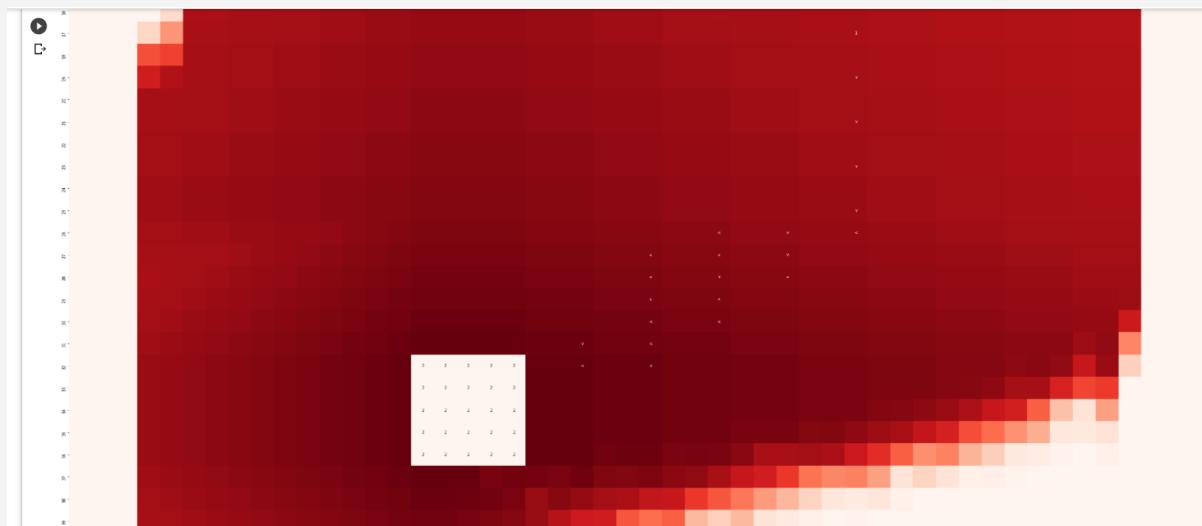


Figure 12 : chemin emprunté par l'agent pour le cas c sans motivation intrinsèque.

La distribution des récompenses extrinsèques montre que l'exploration de l'agent est plus dépendante des conditions venteuses que sans motivation intrinsèque :



*Figure 13 : distribution des récompenses extrinsèques et meilleures actions pour chaque état pour le cas c avec un plateau de taille 50\*50 et sans motivation intrinsèque.*

Effectivement, à partir de  $y > 25$ , l'agent est poussé fortement vers l'ouest. La partie basse droite du plateau a donc peu l'occasion d'être parcourue. Le raisonnement est le même pour le coin supérieur gauche. Avec la motivation intrinsèque, l'agent est poussé à découvrir en priorité les états qu'il connaît le moins. Le vent le pousse en priorité vers la partie ouest pour  $y > 25$  et nord pour  $y \leq 25$  du plateau (le nord est ici en bas du plateau), donc la motivation intrinsèque le pousse à aller vers la partie sud et la partie est. C'est pour cela que la distribution des récompenses extrinsèques pour le cas c avec récompenses intrinsèques est plus uniforme. C'est aussi pour cela que dans les cas a et b, où le vent éloigne l'agent de l'état final, l'apprentissage converge avec motivation intrinsèque et diverge sans, tandis que dans le cas c où le vent pousse vers l'état final, l'apprentissage est plus rapide sans motivation intrinsèque.

J'ai essayé d'ajouter un obstacle dans cet environnement pour voir le comportement de l'agent. Il est représenté par les "x" et est situé en x allant de 14 à 37 et y allant de 19 à 30. Voici le résultat :

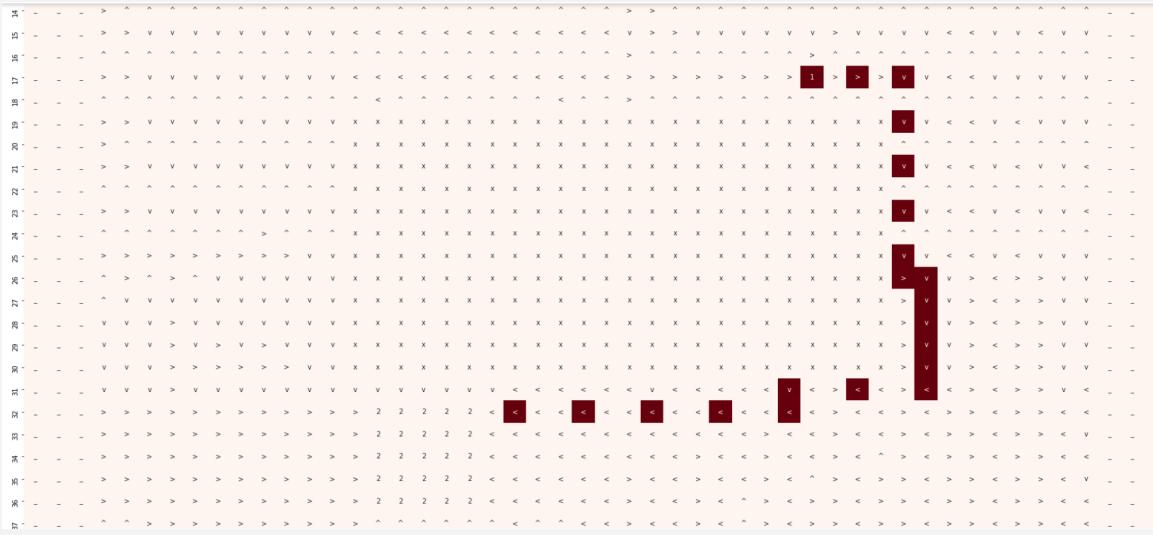


Figure 14 : chemin emprunté par l'agent pour le cas c avec un obstacle.

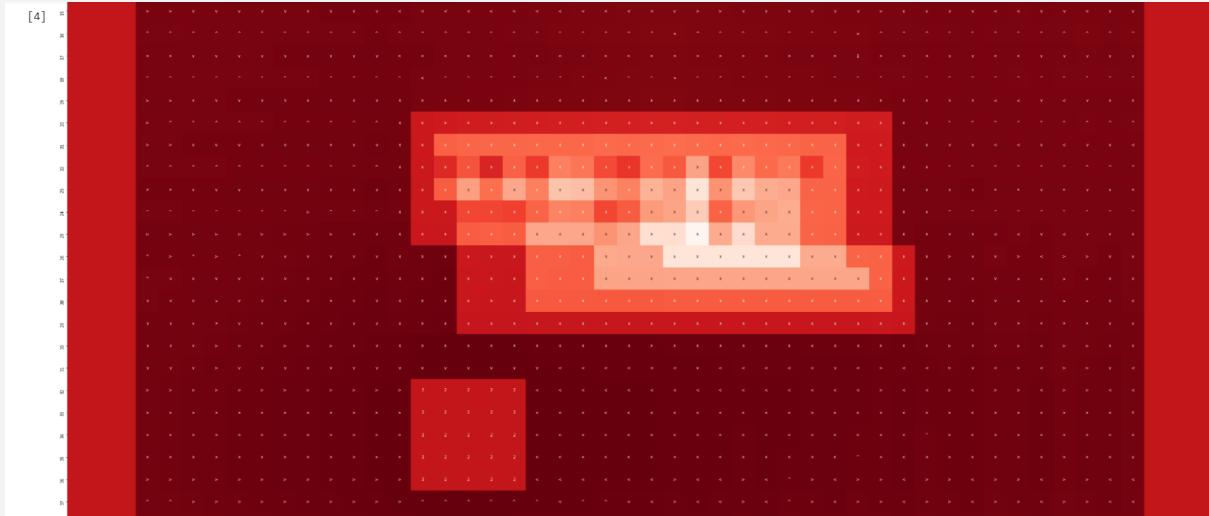


Figure 15 : distribution des récompenses extrinsèques et meilleures actions pour chaque état pour le cas précédent avec obstacles.

On voit que l'agent adopte la même stratégie mais en contournant l'obstacle. La distribution des récompenses extrinsèques, elle, montre que l'agent est beaucoup moins passé par les états de l'obstacle que sur le reste de l'environnement, comme prévu.

### g) Implémentation du vent changeant

L'idée était de permettre aux conditions venteuses d'évoluer de manière aléatoire dans l'environnement. Il aurait fallu décrire les états selon l'angle entre le déplacement de l'agent et le vent, entre le déplacement de l'agent et les états finaux, et la distance entre

l'agent et les états finaux. Cela aurait permis à l'agent d'apprendre à s'orienter en ne raisonnant pas géographiquement, mais en fonction des conditions dans lesquelles il est (vent, distance restante jusqu'aux états finaux, et leur direction), indépendamment de ses coordonnées. Cela ressemble plus à la manière dont un marin fonctionnerait. Après réflexion, je me suis rendu compte que les états pouvaient être décrits de deux manières, et que dans les deux cas, un problème se posait :

- L'angle avec la direction du vent, l'angle avec les états finaux et la distance jusqu'aux états finaux. Ici, les coordonnées  $(x,y)$  sont tout simplement remplacées par  $(r, \theta)$ , les coordonnées polaires. Ceci signifie que les états traduisent la même information, à savoir la position géographique de l'agent dans l'environnement. A chaque état correspond donc seulement une position. Finalement, avec cette description des états, l'agent va continuer à raisonner géographiquement. Comme le vent, lui, devient indépendant des coordonnées, l'agent va se mettre à raisonner en termes de moyennes de conditions venteuses pour chaque état de l'environnement. Les performances vont donc être moindres.
- L'angle entre la direction du vent et la direction vers les états finaux, et la distance jusqu'aux états finaux. Avec cette manière de procéder, le raisonnement de l'agent se rapproche de celui d'un marin, à savoir pas géographiquement : à un état peut correspondre plusieurs coordonnées (exemple : angle = 0, distance = 50. Ici, tous les points sur le cercle de rayon 50 et de centre de coordonnées égales à celles des états finaux peuvent correspondre, à condition que le vent souffle pile dans le dos de l'agent si celui-ci est tourné vers les états finaux). Le problème est que pour chacun de ces points, l'action optimale à prendre sera différente (exemple : au point le plus au nord, l'agent devra aller vers le sud, tandis qu'au point le plus à l'ouest, l'agent devra aller à l'est). Pour avoir un environnement qui puisse fonctionner, il faut alors modifier les actions : elles ne doivent plus être absolues mais relatives à la direction vers laquelle regarde l'agent (alors dans l'exemple précédent, l'action optimale sera : avancer).

J'ai donc préféré coder la partie suivante pour ne pas avoir à modifier l'implémentation du système de déplacements.

#### h) Implémentation d'un réseau de neurones pour remplacer la Q-table

Pour me passer de considérations géographiques, j'ai pensé à décrire les états non plus par les coordonnées de la case à laquelle ils sont associés ou selon les données décrites à la partie précédente, mais par des informations de leur environnement local (à la manière d'un lidar à 360 degrés). J'ai voulu essayer de coder cette nouvelle manière de

procéder avec de l'apprentissage par renforcement profond car cela s'y prêtait bien. Je n'ai malheureusement pas réussi, mais voici comment j'ai procédé :

Les états sont donc décrits par une matrice [5,5] qui traduit les cases autour de l'agent. Chacune est codée avec un 0 si elle est vide, avec un 1 si elle correspond à une récompense, et un -1 si elle correspond à une punition. Sans punitions, les cases éloignées de plus de deux cases de celles finales se retrouvent alors identiques. Par conséquent, dans un environnement de taille 300x300 avec un seul îlot de récompenses, l'agent gardera un comportement aléatoire (sauf s'il est très proche des récompenses) quelque soit la durée de l'apprentissage. J'ai donc décidé de changer de but : l'agent ne doit plus trouver le plus court chemin jusqu'à une localisation, mais plutôt se déplacer sans tomber sur des ouragans, des tempêtes, des tornades, des tourbillons, des pirates, des calamars géants, ou encore des krakens (comprendre : des punitions). A noter que l'agent continue de se déplacer selon les quatre directions et avec des sauts entiers. Ainsi, les coordonnées à chaque état continuent d'être calculées, et sont utilisées pour déterminer la matrice.

Au départ, je voulais avoir un réseau qui prend en entrée la matrice, et en déduit une espérance de récompense extrinsèque pour chaque action (déplacement vers l'est, l'ouest, le sud ou le nord). Le réseau a donc quatre sorties, et l'indice de la valeur maximale donne l'information de l'action à réaliser. Les informations associées aux états parcourus sont stockées dans des listes (états, états suivants, récompenses, et actions). Pour garder le plus possible une indépendance dans les données, elles sont insérées aléatoirement. Pour ne pas stocker trop de données, les listes ont une taille maximale de 15000 données. Arrivé à ce niveau-là, elles se vident d'une de leur valeur à chaque nouvelle donnée entrée. A la fin de chaque épisode, le réseau est entraîné avec les listes. L'idée est alors d'utiliser les récompenses et les nouveaux états pour calculer les targets avec l'équation de Bellman :

$$Q(s, a) = r + \gamma * \max(Q(s_{t+1}, a_{t+1}))$$

A chaque entraînement donc, le modèle minimise la perte entre la liste des états ( $Q(s, a)$ ), et la somme de la liste des récompenses et de gamma multiplié aux maximums de récompenses extrinsèques parmi les couples état-actions des états suivants ( $r + \gamma * \max(Q(s_{t+1}, a_{t+1}))$ ). Mais chaque état suivant de la liste des états suivants est ajouté selon l'action choisie pour l'état parcouru. De ce fait, le maximum de récompense extrinsèque pour tous les couples état-actions ( $\max(Q(s_{t+1}, a_{t+1}))$ ) ainsi que la récompense ( $r$ ), sont liés à l'action choisie (le  $a$  dans  $Q(s, a)$ ). La perte doit donc seulement être calculée selon la sortie du réseau correspondant à cette action. N'ayant pas trouvé de paramètres de la méthode `fit()` de TensorFlow qui permette de choisir quelle perte de sortie minimiser à chaque descente de gradient, j'ai essayé de calculer la perte à la

main et d'utiliser la méthode `minimize()` sur mon optimizer pour cette perte. Elle est calculée comme ceci (méthode des moindres carrés) :

$$\text{perte} = (\text{prediction} - \text{target})^2 * \text{action}$$

La variable `prediction` correspond à la prédiction du réseau sur la première partie de l'équation de Bellman, tandis que la `target` correspond à la deuxième. La variable `action` est un tableau de taille quatre rempli de zéros et avec un "1" à l'index correspondant à l'action choisie. De cette manière, seule la perte selon cette action est minimisée. J'ai eu malheureusement une erreur que je n'ai pas réussi à résoudre. Le script correspondant s'appelle "*Etape7\_ARProfond*". Voici l'erreur :

```
LookupError: No gradient defined for operation 'IteratorGetNext' (op type: IteratorGetNext)
```

J'ai donc décidé de procéder différemment : utiliser quatre réseaux différents, c'est-à-dire un par action. Chacun est entraîné à prédire l'espérance de récompense extrinsèque pour chaque déplacement (est, ouest, nord, et sud). De cette manière, j'ai pu utiliser la méthode `fit()` et ne pas avoir à calculer la perte. Pour le reste, le fonctionnement est globalement le même. Le nombre de listes a été multiplié par quatre pour composer quatre bases de données (une par réseau). Lors d'un déplacement en exploitation, l'action choisie est celle correspondant à la sortie du réseau maximale parmi les 4. L'état, l'action optimale choisie, l'état suivant sélectionné en conséquence, ainsi que la récompense, sont donc stockées dans les listes de la base de données associée à l'action. A la fin de chaque épisode, les quatre réseaux sont entraînés, chacun avec leur base de données. Ils sont constitués d'une couche `Flatten` en entrée pour aplatisir la matrice, d'une couche `Dense` de taille 32 (par défaut) en `ReLU` (aussi par défaut, car je n'ai pas pensé nécessaire d'utiliser une autre fonction d'activation en particulier), puis d'une couche `Dense` de taille 1 pour la sortie avec `Sigmoid`, car il n'y a qu'une sortie. La perte utilisée est "binary\_crossentropy" pour la même raison. Le fichier correspondant s'appelle "*Etape8\_ARProfond2*".

Pour ce script, la perte commence à environ 0,7 en début d'apprentissage et finit négative à partir de quelques dizaines d'épisodes. L'agent adopte alors un comportement étranger : il ne va que vers l'ouest, jusqu'à la limite du plateau. J'ai décidé d'enlever le vent car il faisait diverger le modèle correspondant au déplacement vers l'est.

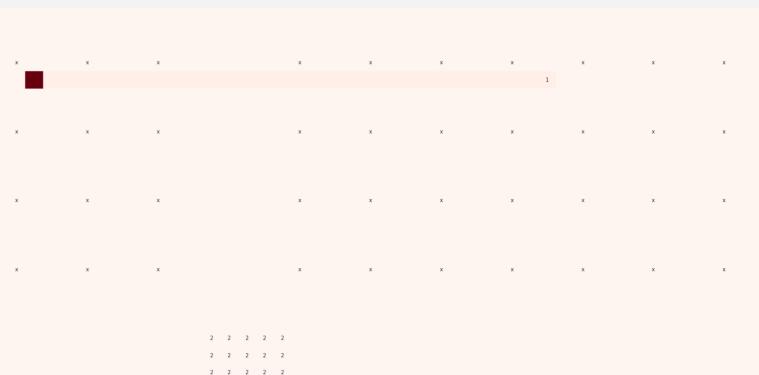


Figure 16 : chemin emprunté par l'agent. Les "x" correspondent aux obstacles.

Un résultat pas très convaincant...

## 2. Planning de travail

Période	Description des tâches réalisées
21/01/2021 - 14/02/2021	<ul style="list-style-type: none"> <li>- Formation à l'apprentissage par renforcement et étude de la nécessité d'utiliser un réseau de neurones</li> <li>- Recherches sur l'usage des réseaux de neurones dans le cadre de l'apprentissage par renforcement profond pour évaluer sa faisabilité dans le cadre du projet</li> <li>- Recherches sur les différentes motivations intrinsèques utilisables</li> <li>- Evaluation de la difficulté et de la pertinence de l'environnement selon les résultats des recherches/formations, et changement/adaptation si nécessaire</li> <li>- Rédaction du script <i>Etape1_ARSimplifie</i>.</li> </ul>
14/02/2021 - 28/02/2021	<ul style="list-style-type: none"> <li>- Rédaction des scripts <i>Etape3_Q-Function</i>, <i>Etape2_ARSimplifie10x10</i> et <i>Etape4_Q-FunctionAgrandi</i>.</li> </ul>

28/02/2021 - 18/03/2021	<ul style="list-style-type: none"> <li>- Rédaction du script <i>Etape5_MotivIntrinseque</i></li> <li>- Rédaction du rapport</li> <li>- Réflexion et tests sur la manière d'implémenter le vent</li> </ul>
18/03/2021 - 14/04/2021	<ul style="list-style-type: none"> <li>- Rédaction du script <i>Etape6_Vent</i></li> <li>- Comparaison avec l'algorithme A*</li> <li>- Rédaction du rapport</li> </ul>
14/04/2021 - 27/04/2021	<ul style="list-style-type: none"> <li>- Formation sur l'usage de réseaux de neurones dans le cadre de l'apprentissage par renforcement profond</li> <li>- Rédaction des scripts <i>Etape7_ARProfond</i> et <i>Etape8_ARProfond2</i></li> <li>- Fin de rédaction du rapport</li> <li>- Réalisation de la vidéo</li> </ul>

## IV. PISTES D'AMÉLIORATION

J'aurais bien aimé réussir à faire fonctionner le dernier script d'apprentissage par renforcement profond. Il aurait fallu que je le débogue plus en profondeur pour vérifier que les bases de données contiennent les bonnes informations. N'étant pas un expert des réseaux de neurones, il se peut fortement que les hyper-paramètres des modèles ne soient pas bien choisis. J'aurais aimé en tester plus.

Concernant l'implémentation de l'environnement, une amélioration intéressante aurait été de permettre à l'agent de se déplacer selon plus de directions. Cela aurait fait naître des stratégies de déplacements vis-à-vis des conditions venteuses bien plus intéressantes que de seulement prendre les actions qui permettent à l'agent de se rapprocher des états finaux et de les répartir sur le parcours selon les conditions venteuses locales. Une autre manière d'obtenir des stratégies efficaces aurait été d'implémenter la stratégies du vent changeant avec des déplacements relatifs à l'angle vers lequel est tourné l'agent. Cette méthode, qui implique un environnement continu, aurait pu être

combinée avec l'apprentissage par renforcement profond pour prédire les états optimaux. Le système de récompenses intrinsèques de mémoire épisodique que j'ai utilisé aurait pu être adapté. J'aurais alors pu estimer de vrais probabilités d'écart en actions avec ceux du buffer plutôt que de calculer de simples distances. Cela aurait aussi été l'occasion d'en implémenter un nouveau, plus abstrait, et plus proche de ce qui peut être fait actuellement dans l'état de l'art.

Comme j'ai décidé d'avancer étape par étape dans le code (car je n'avais jamais fait d'apprentissage par renforcement avant), j'ai fait évoluer le code initial au fur et à mesure des scripts. Comme celui-ci était très spécifique à un environnement discret en damier avec déplacements selon les deux axes et les deux sens, je me retrouve à la fin avec un code rigide, qui est difficilement adaptable à de nouvelles modifications de l'environnement. Par exemple, les méthodes `deplacer_horizontalement()`, `deplacer_verticalement()`, et `mettre_a_jour_q_table()`, fonctionnent en soustrayant les nouvelles coordonnées aux anciennes pour remplir la Q-table. Si j'avais voulu modifier la description des états, j'aurais pu ne pas pouvoir me baser sur les coordonnées absolues (comme je l'ai fait pour l'apprentissage par renforcement profond) pour faire la transition entre deux états et remplir la Q-table. Penser dès le début à ces possibles futures modifications m'aurait permis d'avoir un code plus représentatif de ce qu'est l'apprentissage par renforcement.

Quoi qu'il en soit, j'ai compris l'importance de réfléchir au fur et à mesure d'un projet exploratoire où l'on va, avec quels objectifs, et quel critères d'appréciation choisir, au risque de trop s'éparpiller et de baisser en performance sinon. J'ai aussi beaucoup appris techniquement grâce à ce projet. Que ce soit au niveau de la réflexion et de l'implémentation d'un réseau neuronal dans un système plus large, ou de l'apprentissage par renforcement plus généralement. Ce paradigme qui m'était inconnu m'est bien plus familier et j'ai aujourd'hui d'autant plus envie de continuer d'explorer ses possibilités, cette fois-ci pour répondre à des problématiques plus utiles.