



Royaume du Maroc
Université Mohammed V de Rabat
École Mohammadia d'Ingénieurs



Département : Génie Électrique
Rapport de projet IA

IAvoid : Robot industrielle agissant contre des
obstacles aléatoires

21 décembre 2025

Réalisé par :
Nouamane SOUADI

Encadré par :
Pr. Asmae AZZAM JAI

Année Universitaire : 2025/2026

Table des matières

Table des figures	2
Liste des tableaux	3
1 C'est quoi IAvoid	4
2 Principe de fonctionnement	4
3 L'apprentissage renforcé	5
3.1 Q-Table	5
3.2 Politique ϵ -cupide	6
3.3 Equation de Bellman	6
4 Simulation Python	6
4.1 Hyperparamètres de la simulation	6
4.2 Environnement de la simulation	7
4.3 Résultats fin de la simulation	8
4.4 Analyse des données	9
5 Comparaison avec les mêmes paramètres	10
5.1 Heatmap des zones les plus fréquentées	10
5.2 Récompenses par itération	11
5.3 Taux de réussite	12
5.4 Conclusion	12
6 Comparaison entre les paramètres	12
6.1 Learning_rate	13
6.2 Epsilon	14
6.3 Discount_factor	15
6.4 Conclusion	16
7 Simulation réelle avec ARDUINO	16
Annexes	18

Table des figures

2.1	Organigramme du fonctionnement général	4
3.1	Principe de Q-Learning algorithm	5
4.1	Organigramme général	6
4.2	Environnement de simulation sur Python	7
4.3	Simulation de la génération des obstacles	8
4.4	Solution choisie pendant la dernière itération	8
4.5	Meilleur solution choisie pendant une simulation/ meilleur récompense	9
4.6	Stockage des données pour analyse	9
4.7	Taux de succès d'une simulation	10
4.8	Evolution de la récompense par itération	10
4.9	Comparaison entre le cycle 4 et le dernier cycle	10
5.1	Comparaison Heatmap - mêmes paramètres	11
5.2	Comparaison Récompenses par itération lissées - mêmes paramètres	11
5.3	Comparaison Taux de réussite - mêmes paramètres	12
5.4	Comparaison récompense/steps - une simulation	12
6.1	Comparaison Learning_rate - Heatmap	13
6.2	Comparaison Learning_rate - Taux de réussite	13
6.3	Comparaison Epsilon - Vision globale	14
6.4	Comparaison Epsilon - Heatmap	14
6.5	Comparaison Epsilon - Taux de réussite	15
6.6	Comparaison Discount_factor - Heatmap	15
6.7	Comparaison Discount_factor - Taux de réussite	15

Liste des tableaux

6.1	Liste des valeurs choisis pour la simulation réelle	16
-----	---	----

1 C'est quoi IAvoid

Le IAvoid est :

-**IAVOID** : C'est un robot industriel (ex. déplacer des charges lourdes) capable d'agir face à des obstacles.

-**IAVOID** : Le processus de traitement de l'action est 100% basé sur l'IA (Intelligence Artificiel) afin de modifier le chemin optimal à un chemin efficace contre les obstacles

IAvoid est un robot concis pour le domaine industriel, conçu généralement pour lever des charges et les déplacer dans un **environnement incertain**.

Dans un environnement incertain, le IAVOID se trouvent chaque fois des obstacles d'une façon aléatoire. Supposant que notre environnement est un monde de **NxN grids**, le robot a 3 actions possibles : **FORWARD**, **TURN_RIGHT**, **TURN_LEFT**.

(Une modélisation 3d du système est envisageable)

2 Principe de fonctionnement

Voici l'organigramme explicatif du fonctionnement général du système :

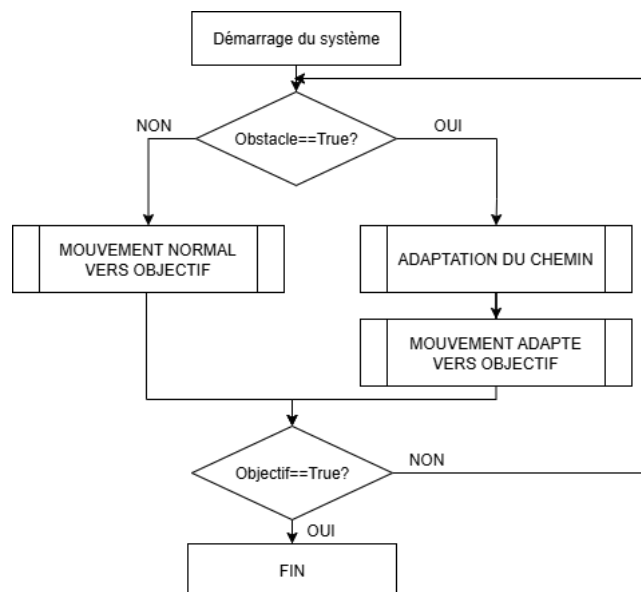


FIGURE 2.1 – Organigramme du fonctionnement général

Le système est lancé dans la première fois sans base de données, avec un chemin préféré au préalable seulement. L'objectif du système est alors de modifier ce chemin préféré à un autre chemin plus **optimal** qui marchera pour la majorité des obstacles qui va trouver dans chaque cycle d'apprentissage.

Chaque fois le système enregistre l'historique des actions faites, et analyse les taux de succès, les zones les plus fréquentées...etc.

Ressources et bibliothèques :

PYTHON

pygame : Simulation en Python

pandas : Gestion des BD et stockage sur Excel

mumpy : Calcul matriciel

random : Génération aléatoire des obstacles

ARDUINO

pour simulation réelle simple et plus simplifiée (et réalisable en terme des moyens)

3 L'apprentissage renforcé

Pour ce cadre d'apprentissage de notre système, notre IA doit être capable d'apprendre à fur et à mesure que le système est en marche. Il s'agit bien d'un apprentissage dit **renforcé** vue qu'il est basé sur les expériences de notre système et les résultats qu'il a trouvé.

Plus exactement, j'ai choisi le **Q-learning algorithm** basé sur des recherches transversales sur ce principe d'IA. (possibilité de recherche plus comparative)

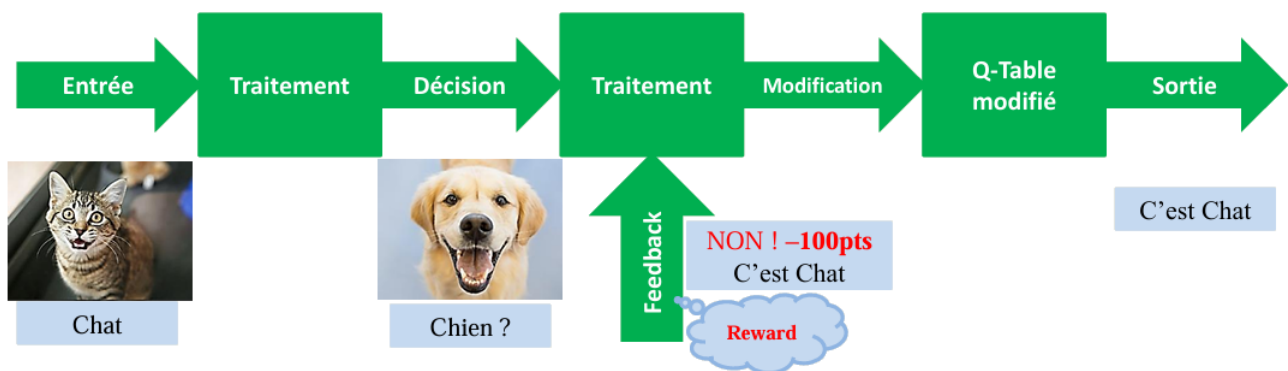


FIGURE 3.1 – Principe de Q-Learning algorithm

3.1 Q-Table

A fur et à mesure l'algorithme actualise le Q-Table et prend la décision avec le minimum des pertes (maximiser les récompenses)

$$Q(S, A) \leftarrow Q(S, A) + \alpha \cdot (R + \gamma \cdot Q(S', A') - Q(S, A)) \quad (1)$$

Avec : S : l'état actuel.

A : l'action entreprise par l'agent.

S' : l'état suivant vers lequel l'agent se déplace.

A' : la meilleure action suivante dans l'état S'.

R : la récompense reçue pour avoir pris l'action A dans l'État S.

γ (Gamma) : le facteur d'actualisation qui équilibre les récompenses immédiates avec les récompenses futures.

α (Alpha) : le taux d'apprentissage déterminant la quantité de nouvelles informations affectant les anciennes valeurs Q.

3.2 Politique ϵ -cupide

Pour l'exploitation : L'agent choisit l'action avec la valeur Q la plus élevée avec probabilité $1-\epsilon$. Cela signifie que l'agent utilise ses connaissances actuelles pour maximiser les récompenses.

Pour l'exploration : Avec probabilité ϵ , l'agent choisit une action au hasard, explorant de nouvelles possibilités pour savoir s'il existe de meilleures façons d'obtenir des récompenses. Cela permet à l'agent de découvrir de nouvelles stratégies et d'améliorer sa prise de décision au fil du temps.

3.3 Equation de Bellman

$$Q(S, A) = R(S, A) + \gamma \cdot \max(U_n Q(S', A)) \quad (2)$$

$Q(S, A)$ est la valeur Q d'une paire état-action donnée.

$R(S, A)$ est la récompense immédiate pour avoir pris des mesures a dans l'état S.

γ est le facteur d'actualisation, représentant l'importance des récompenses futures.

$\max(U_n Q(S', A))$ est la valeur Q maximale pour l'état suivante de toutes les actions possibles.

4 Simulation Python

Voici le principe globale du code python de la simulation (voir Annexe A pour le complément) :

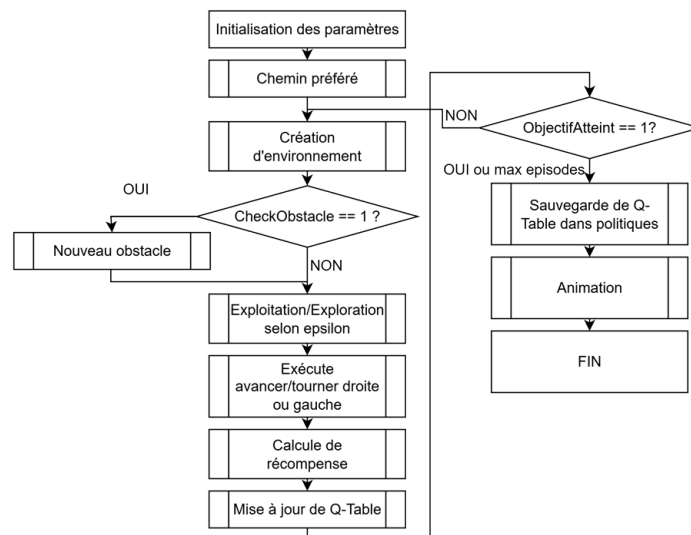


FIGURE 4.1 – Organigramme général

4.1 Hyperparamètres de la simulation

Chaque fois qu'on veut analyser l'effet des paramètres du système on modifie les paramètres ci-dessous :

```

1 # -----
2 # Hyperparameters
3 # -----
4 #
5 learning_rate = 0.3
6 discount_factor = 0.9
7 epsilon = 0.7
8 epsilon_min = 0.01
9 epsilon_decay = 0.997
10 num_episodes = 5000
11 grid_size = 15
12 obstacle_change_interval = 200
13 max_steps_per_episode = 250
14 cell_size = 40
15 screen_size = grid_size * cell_size
16 tick_time = 240
17 episodes_output = "episode_data.xlsx"

```

4.2 Environnement de la simulation

Voici la composition de l'environnement simulé sur Python à l'aide de

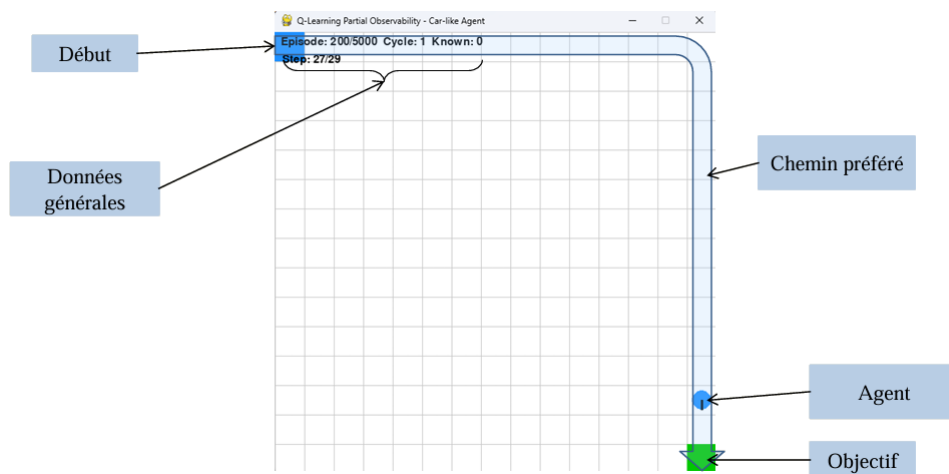


FIGURE 4.2 – Environnement de simulation sur Python

Comme déjà annoncé, le système se trouve avec des obstacles générés chaque fois d'une façon aléatoire et notre objectif est d'adapter notre chemin de façon de se retrouver à notre position objectif mais avec **le chemin le plus convenable et stable pour même des futurs obstacles.**

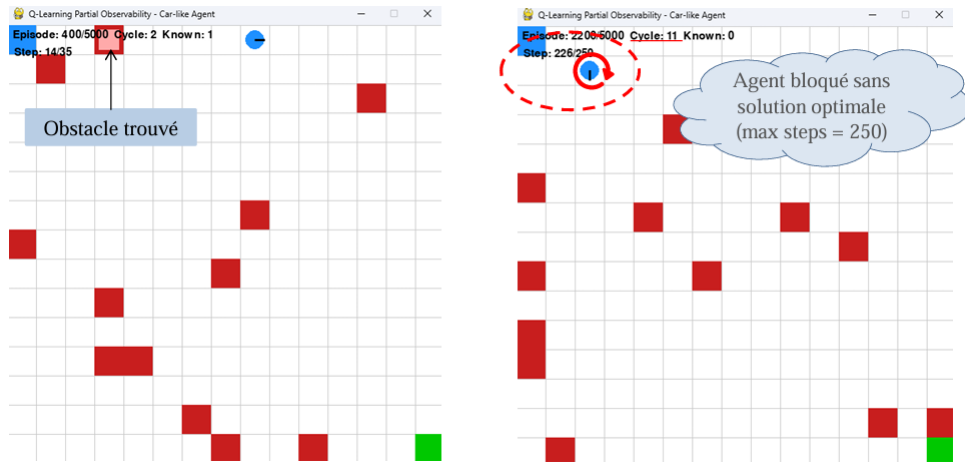


FIGURE 4.3 – Simulation de la génération des obstacles

4.3 Résultats fin de la simulation

Dans ce code Python la simulation est composée de **25 cycles de 200 épisodes pour chacun** avec une limitation de **250 itération/step** possible dans chaque épisode (pour éviter une simulation infinie lorsque le système ne trouve pas la bonne solution et reste coincé).

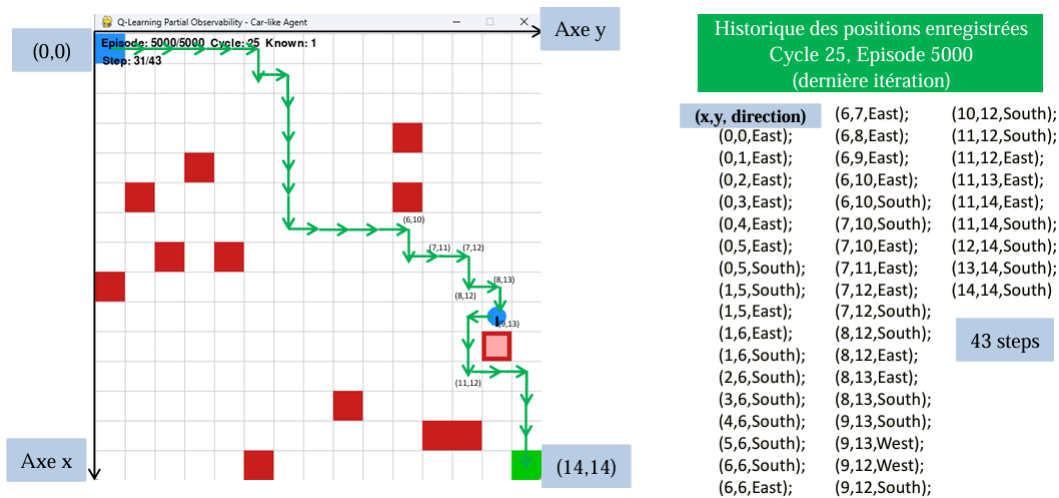


FIGURE 4.4 – Solution choisie pendant la dernière itération

En fait, la dernière itération n'est pas forcément la bonne solution que le système a trouvé, et c'est à cause de l'emplacement des obstacles qui change durant la simulation. (Si l'emplacement a resté fixe le système peut développer avec chaque itération une meilleur solution à fur et à mesure)

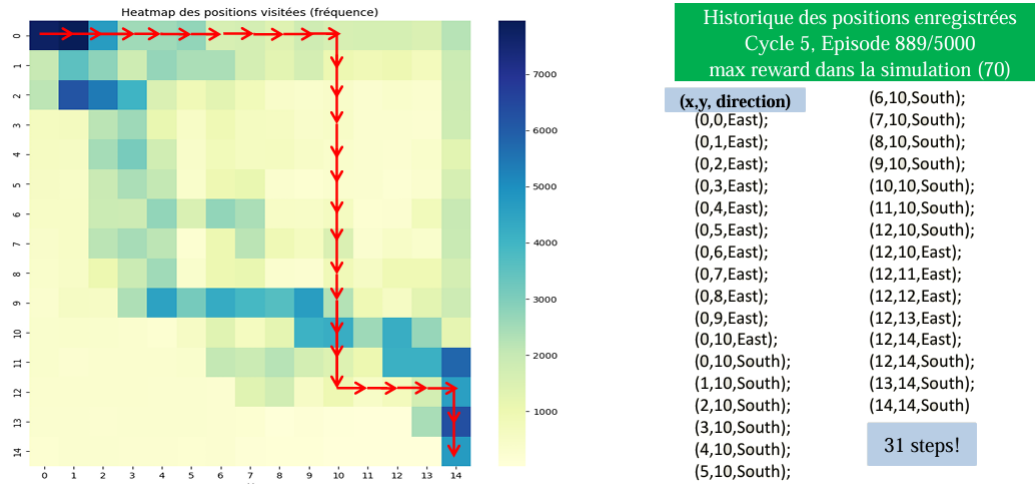


FIGURE 4.5 – Meilleur solution choisie pendant une simulation/ meilleur récompense

Cette solution donne la meilleur récompense pendant toute la simulation qui se manifeste par un chemin très court et évitant le maximum des obstacles.

4.4 Analyse des données

Après la fin de la simulation toutes les données sont enregistrées sur des fichiers Excel (facile à manipuler)

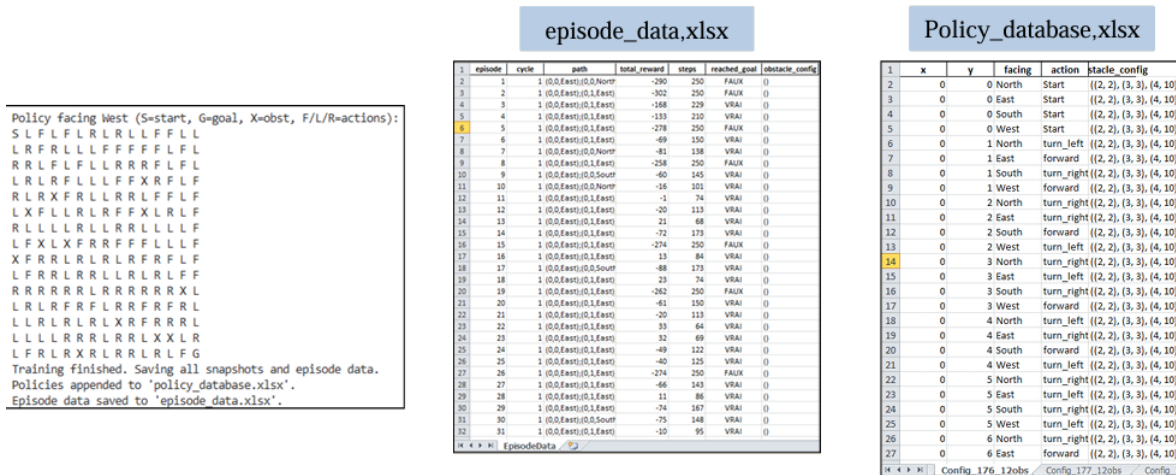


FIGURE 4.6 – Stockage des données pour analyse

N.B. : L'enregistrement des données sur "policy_database.xlsx" est annulé après pour optimier le temps de la simulation.

Dans notre simulation, le **cycle 4** était le plus mauvais cycle simulé avec le total des récompenses le plus faible. Généralement s'est produit avec une nouvelle structure d'emplacement des obstacles que le système vient d'essayer comprendre et agir. Et le **cycle suivant, le cycle 5** a marqué au contraire la meilleur récompense même si le taux de réussite du cycle n'était pas maximal par rapport aux autres cycles, ce qui montre l'évolution rapide de notre système

et la réaction prise face à un grand défi au cycle 4.

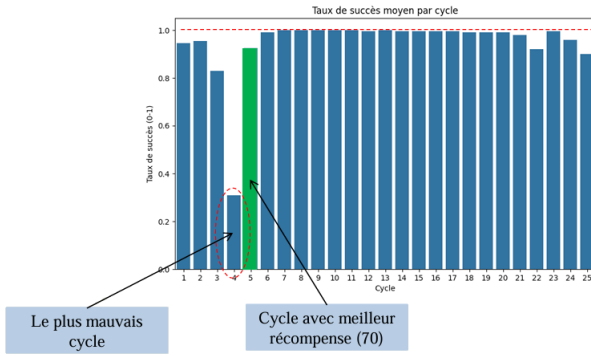


FIGURE 4.7 – Taux de succès d'une simulation

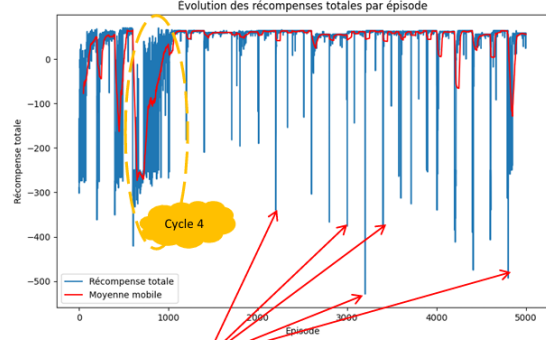


FIGURE 4.8 – Evolution de la récompense par itération

Dans la figure 4.8, on remarque que pendant des itérations le système est coincé sans solution qui est très visible avec les pics négatifs

epist	cycle	path	total_reward	steps	reached_goal	obstacle_conf
790	4	(0,0,East)(0,1,East)	40	50	VRAI	((1, 8), (1, 11), (4, 14)
727	4	(0,0,East)(0,1,East)	40	61	VRAI	((1, 8), (1, 11), (4, 14)
783	4	(0,0,East)(0,1,East)	36	65	VRAI	((1, 8), (1, 11), (4, 14)
735	4	(0,0,East)(0,1,East)	32	69	VRAI	((1, 8), (1, 11), (4, 14)
744	4	(0,0,East)(0,1,East)	24	77	VRAI	((1, 8), (1, 11), (4, 14)
771	4	(0,0,East)(0,0,North)	24	77	VRAI	((1, 8), (1, 11), (4, 14)
781	4	(0,0,East)(0,1,East)	24	77	VRAI	((1, 8), (1, 11), (4, 14)
766	4	(0,0,East)(0,1,East)	16	85	VRAI	((1, 8), (1, 11), (4, 14)
755	4	(0,0,East)(0,0,South)	14	87	VRAI	((1, 8), (1, 11), (4, 14)
770	4	(0,0,East)(0,0,North)	12	89	VRAI	((1, 8), (1, 11), (4, 14)
732	4	(0,0,East)(0,1,East)	8	93	VRAI	((1, 8), (1, 11), (4, 14)
778	4	(0,0,East)(0,1,East)	8	93	VRAI	((1, 8), (1, 11), (4, 14)
791	4	(0,0,East)(0,0,North)	8	93	VRAI	((1, 8), (1, 11), (4, 14)
772	4	(0,0,East)(0,0,South)	6	95	VRAI	((1, 8), (1, 11), (4, 14)
760	4	(0,0,East)(0,0,South)	2	99	VRAI	((1, 8), (1, 11), (4, 14)
779	4	(0,0,East)(0,0,South)	2	90	VRAI	((1, 8), (1, 11), (4, 14)
730	4	(0,0,East)(0,1,East)	-5	93	VRAI	((1, 8), (1, 11), (4, 14)
756	4	(0,0,East)(0,1,East)	-6	107	VRAI	((1, 8), (1, 11), (4, 14)
663	4	(0,0,East)(0,0,North)	-12	105	VRAI	((1, 8), (1, 11), (4, 14)
736	4	(0,0,East)(0,1,East)	-16	117	VRAI	((1, 8), (1, 11), (4, 14)
789	4	(0,0,East)(0,1,East)	-16	117	VRAI	((1, 8), (1, 11), (4, 14)
725	4	(0,0,East)(0,1,East)	-18	119	VRAI	((1, 8), (1, 11), (4, 14)
729	4	(0,0,East)(0,1,East)	-20	121	VRAI	((1, 8), (1, 11), (4, 14)
799	4	(0,0,East)(0,1,East)	-22	123	VRAI	((1, 8), (1, 11), (4, 14)
759	4	(0,0,East)(0,0,North)	-22	123	VRAI	((1, 8), (1, 11), (4, 14)
764	4	(0,0,East)(0,0,North)	-32	124	VRAI	((1, 8), (1, 11), (4, 14)
731	4	(0,0,East)(0,1,East)	-34	135	VRAI	((1, 8), (1, 11), (4, 14)
782	4	(0,0,East)(0,0,South)	-37	134	VRAI	((1, 8), (1, 11), (4, 14)
774	4	(0,0,East)(0,1,East)	-41	138	VRAI	((1, 8), (1, 11), (4, 14)
665	4	(0,0,East)(0,0,North)	-44	145	VRAI	((1, 8), (1, 11), (4, 14)

Cycle 4

episode	cycle	path	total_reward	steps	reached_goal	obstacle_conf
4891	25	(0,0,East)(0,1,East)	64	37	VRAI	((3, 10), (4, 3), (5, 15)
4852	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4861	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4862	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4863	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4864	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4865	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4866	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4867	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4868	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4869	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4870	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4871	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4872	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4873	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4874	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4875	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4876	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4877	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4878	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4879	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4880	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4881	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4882	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4883	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4884	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4885	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4886	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4887	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4888	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4889	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4890	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4891	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4892	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4893	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4894	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4895	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4896	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4897	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4898	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4899	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4900	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4901	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4902	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4903	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4904	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4905	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4906	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4907	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4908	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4909	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4910	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4911	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4912	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4913	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4914	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4915	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4916	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4917	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4918	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4919	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4920	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)
4921	25	(0,0,East)(0,1,East)	58	43	VRAI	((3, 10), (4, 3), (5, 15)

Dernier cycle

FIGURE 4.9 – Comparaison entre le cycle 4 et le dernier cycle

5 Comparaison avec les mêmes paramètres

cette section est crucial pour l'optimisation de notre système, nous allons partir d'une comparaison entre **5 simulations** avec les mêmes paramètres et puis, pour l'optimisation de notre système réel, vers une comparaison de l'effet de chaque paramètre sur les performances du robot dans la prochaine section.

5.1 Heatmap des zones les plus fréquentées

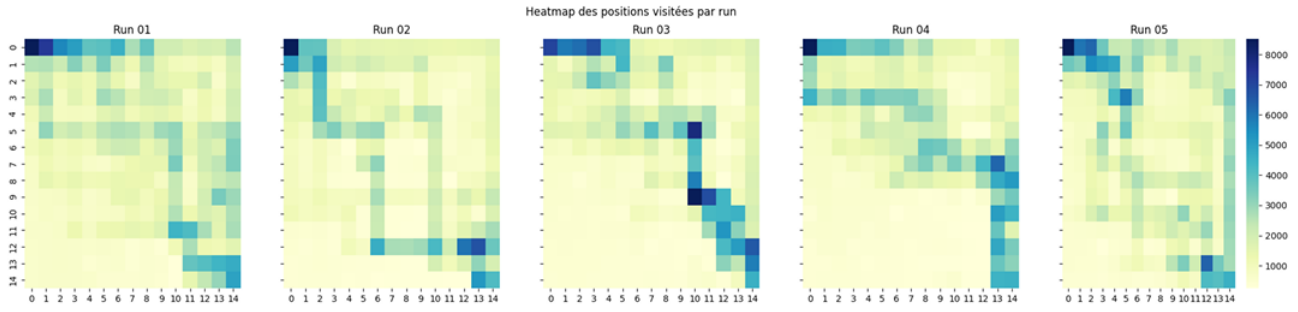


FIGURE 5.1 – Comparaison Heatmap - mêmes paramètres

Pour information, notre chemin préféré initié au système est de partir vers le coin haut-droit et après descendre vers l'objectif. On remarque alors que le système a la même tendance d'apprentissage vers un chemin optimal qui passe généralement de la diagonale de notre environnement de $N \times N$ grids.

Ce choix du chemin diagonal est logique déjà puisque c'est la plus courte distance envisageable mais la distribution autour de ce chemin n'est pas consistante vue que les obstacles peuvent se générer aléatoirement autour de ce chemin.

5.2 Récompenses par itération

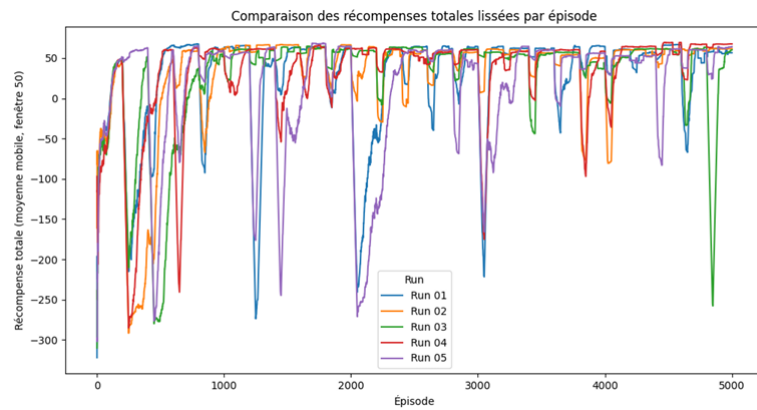


FIGURE 5.2 – Comparaison Récompenses par itération lissées - mêmes paramètres

Le système a la même tendance de se coincé dans les premières itérations, tout à fait normal puisque le système vient de commencer son apprentissage avec un minimum d'expérience, reste le moment de **"failure"** une variable aléatoire encore une fois de la nature du problème posé au système.

5.3 Taux de réussite

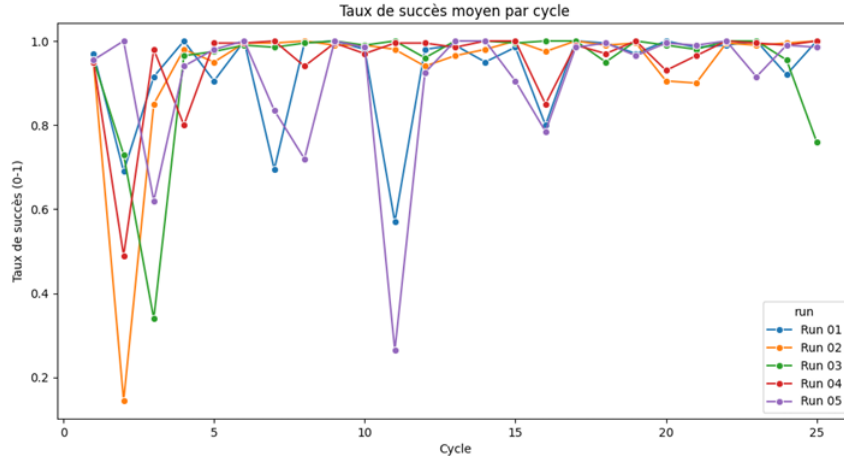


FIGURE 5.3 – Comparaison Taux de réussite - mêmes paramètres

Ce paramètre est très sensible à la structure d'emplacement des obstacles

5.4 Conclusion

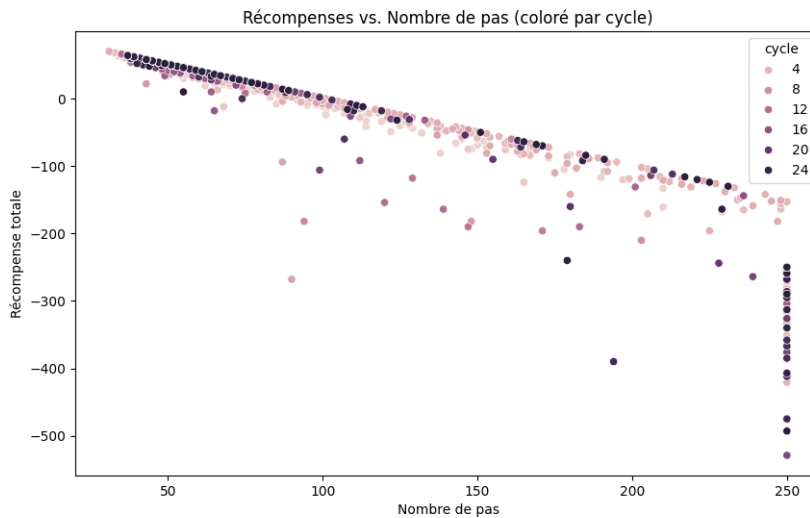


FIGURE 5.4 – Comparaison récompense/steps - une simulation

La réponse du système est sensible aux obstacles qui est une variable aléatoire incontrôlable (en nombre et emplacement) d'où les biais entre les simulations mais on constate qu'il y a la même tendance d'apprentissage à travers le Heatmap et la même corrélation **récompenses/s-teps** et pente d'apprentissage similaire le moment qu'on a les mêmes paramètres.

6 Comparaison entre les paramètres

On a Trois paramètres majeurs qui affecte les performances du système : "**Learning_rate**", "**Epsilon**", "**Discount_factor**". Nous allons analyser chaque paramètre indépendamment

des autres. (Une analyse plus efficace sera de générer plusieurs simulations et modifier automatiquement les paramètres jusqu'à trouver une solution optimale ce qui est envisageable après).

6.1 Learning_rate

Valeurs choisies pour la simulation : 0.3 - 0.4 - 0.5 - 0.7 - 0.9

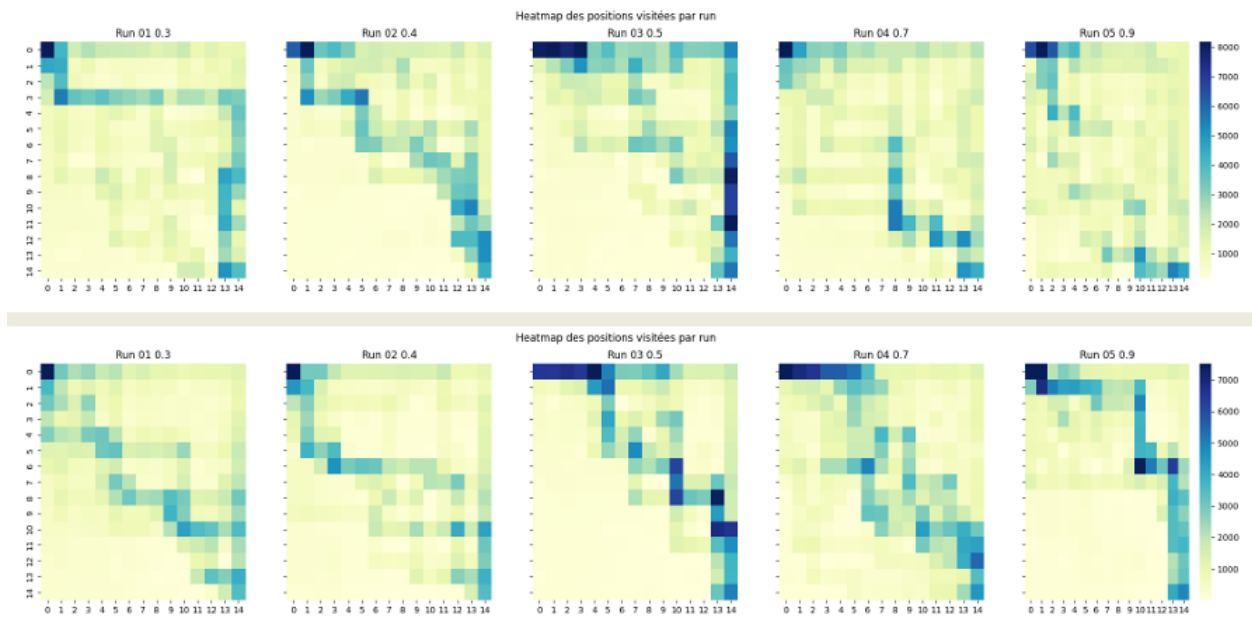


FIGURE 6.1 – Comparaison Learning_rate - Heatmap

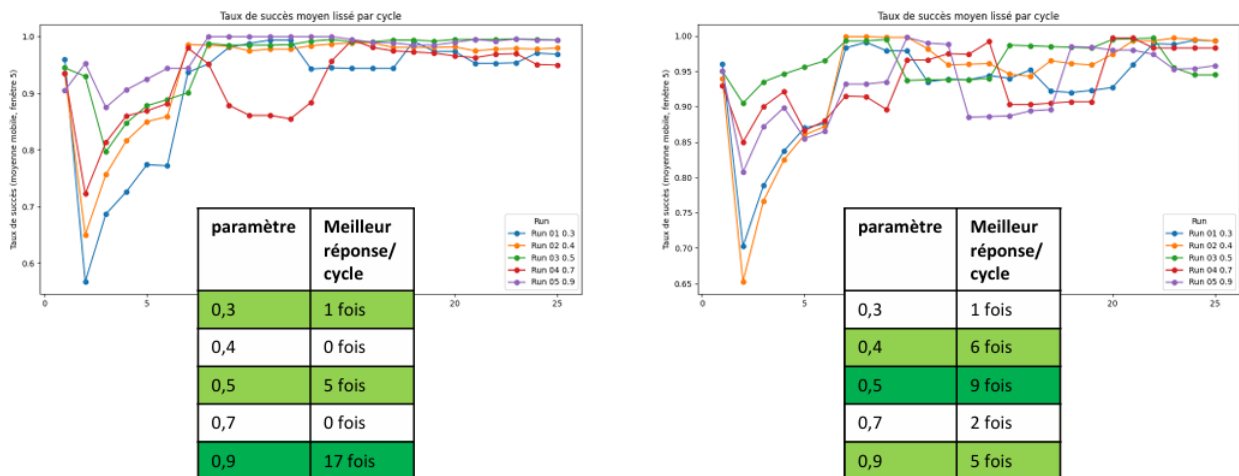


FIGURE 6.2 – Comparaison Learning_rate - Taux de réussite

Plus le Learning_rate est grand plus le système construit des chemins structurés et fiables (consistants). Face à des obstacles aléatoires et parfois complexes, le choix **d'augmenter le Learning_rate** est judicieux pour des taux de réussite plus optimal.

6.2 Epsilon

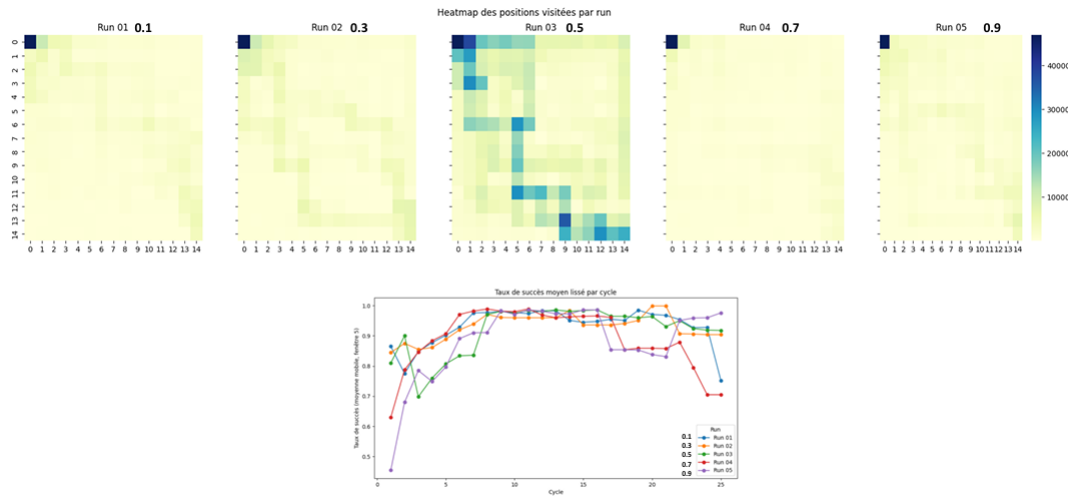


FIGURE 6.3 – Comparaison Epsilon - Vision globale

Pour une première analyse on donne à Epsilon des valeurs dispersées : 0.1 - 0.3 - 0.5 - 0.7 - 0.9, on trouve qu'il faut avoir **une valeur proche de 0.5**. En fait, le "Epsilon" en question est **une valeur initié au début de la simulation** qui est perdue par la suite, et donc tout dépend de la complexité des obstacles générés.

Pour une deuxième analyse, on donne à Epsilon : 0.5 - 0.55 - 0.6 - 0.65 - 0.7

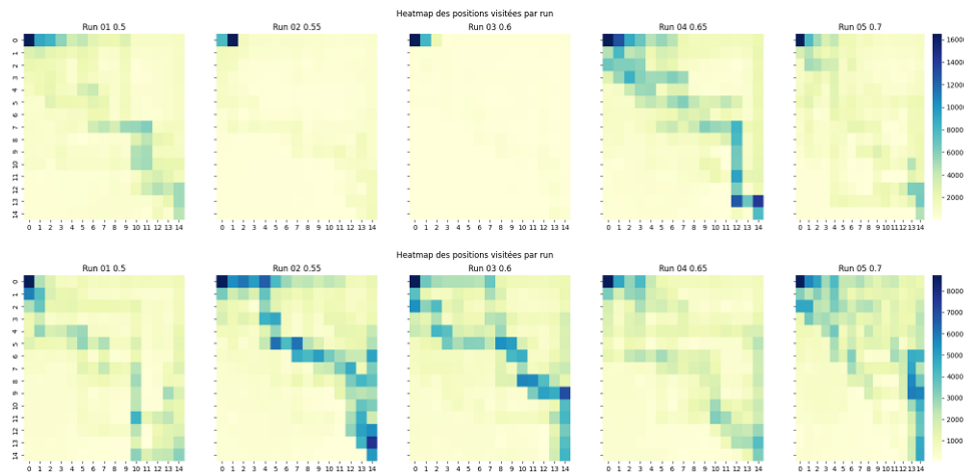


FIGURE 6.4 – Comparaison Epsilon - Heatmap

Par la suite le système a une réponse aléatoire indépendamment de la valeur d'Epsilon

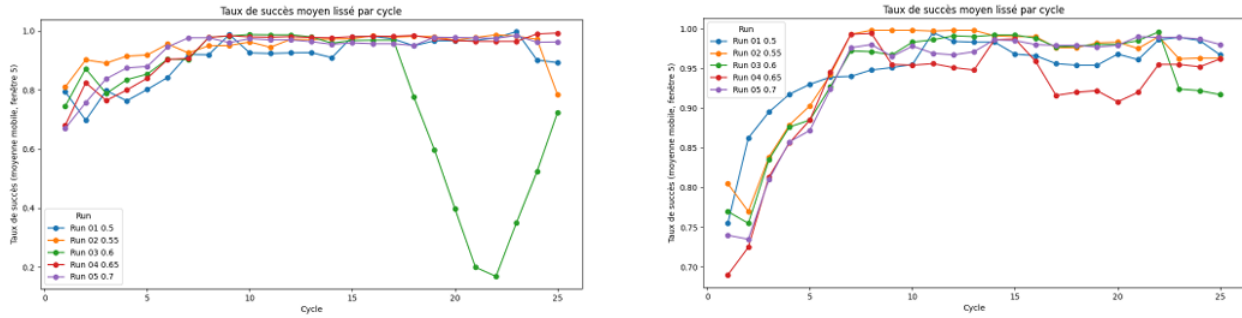


FIGURE 6.5 – Comparaison Epsilon - Taux de réussite

6.3 Discount_factor

Passant maintenant au Discount_factor, ce paramètre définit comment on donne les récompense au système chaque fois il fait des mauvaises choix ou bien ne donne pas des solutions optimisées.

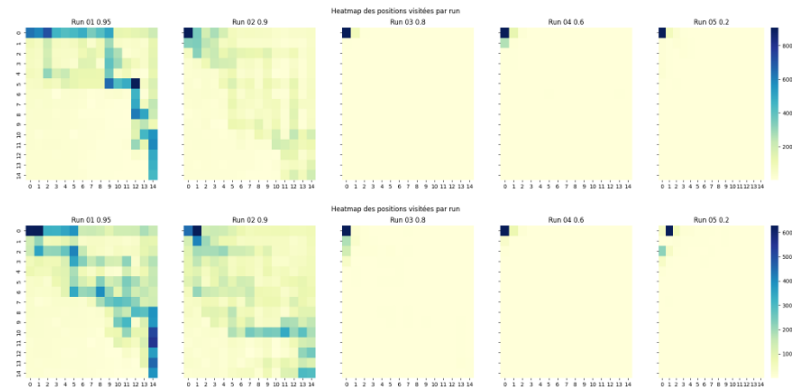


FIGURE 6.6 – Comparaison Discount_factor - Heatmap

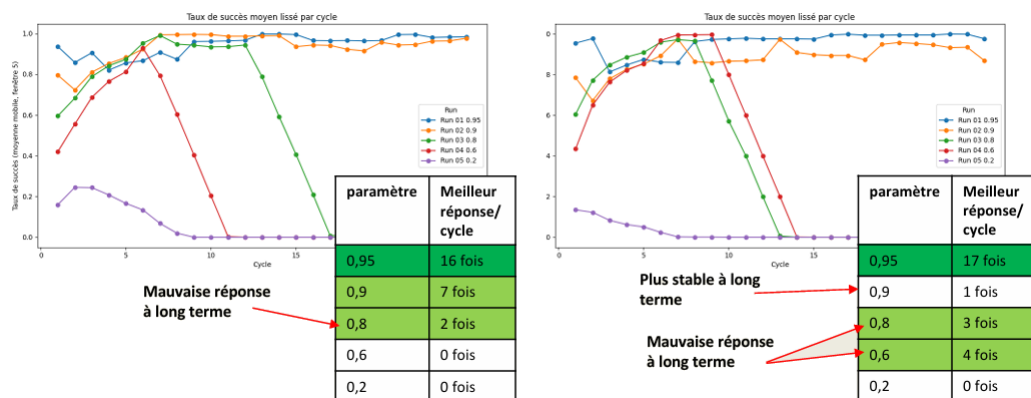


FIGURE 6.7 – Comparaison Discount_factor - Taux de réussite

Face à des valeurs très grandes de ce paramètre, le système est forcé alors de trouver des solutions optimale et naviguer les opportunités possibles.

6.4 Conclusion

Pour conclure, à l'aide de cette analyse on a pu comprendre l'effet de chaque paramètre indépendamment des autres, reste encore de trouver des valeurs optimales en harmonie qui sera bien évidemment un compromis entre les avantages et inconvénients de l'ensemble.

Pour cela, et pour l'instant on va utiliser seulement une estimation des valeurs choisis dans le tableau 6.1, en attendant de développer une simulation plus efficace qui nous donnera des valeurs plus efficaces.

TABLE 6.1 – Liste des valeurs choisis pour la simulation réelle

Learning_rate	Epsilon	Discount_factor
0.9	0.65	0.95

7 Simulation réelle avec ARDUINO

À l'aide de code ARDUINO dans l'annexe B, nous allons essayer d'implémenter la solution à notre système réel. Les ressources qu'on a sont : Carte ARDUINO UNO, capteur Ultrasonc pour la détection des obstacles, deux moteurs avec un driver et une batterie pour l'alimentation continue du système. Reste seulement comment enregistrer les données communiquées via le Serial d'ARDUINO pour le fonctionnement normal du système sinon pour l'instant on peut annuler la fonctionnalité de déplacement.

Exemple sortie du système pour analyse des performances :

```
=== Q-Learning Robot - Mmoire Optimise
Start: (0,0) facing East | Goal: (4,4)
Cycles: 50 | Terminologie: Cycle =
    ↳ parcours, pisode = action
Dbut de l'apprentissage...'
```

```
--- Cycle 1 (eps=0.30) ---
```

```
E0: FORWARD (0,0) -1
E1: TURN_LEFT (0,1) -1
E2: TURN_LEFT (0,1) -1
E3: FORWARD (0,1) -1
E4: FORWARD (0,0) MUR -5
E5: TURN_LEFT (0,0) -1
E6: FORWARD (0,0) -1
E7: FORWARD (1,0) -1
E8: FORWARD (2,0) -1
E9: FORWARD (3,0) -1
E10: FORWARD (4,0) MUR -5
E11: TURN_LEFT (4,0) -1
```

```
E12: FORWARD (4,0) -1
E13: FORWARD (4,1) NEW_OBS(4,2) -10
E14: TURN_LEFT (4,1) -1
E15: TURN_RIGHT (4,1) -1
E16: TURN_RIGHT (4,1) -1
E17: FORWARD (4,1) MUR -5
E18: FORWARD (4,1) MUR -5
E19: TURN_LEFT (4,1) -1
E20: TURN_LEFT (4,1) -1
E21: FORWARD (4,1) -1
E22: FORWARD (3,1) -1
E23: TURN_LEFT (2,1) -1
E24: FORWARD (2,1) -1
E25: FORWARD (2,0) MUR -5
E26: TURN_LEFT (2,0) -1
E27: TURN_LEFT (2,0) -1
E28: FORWARD (2,0) -1
E29: FORWARD (2,1) -1
E30: FORWARD (2,2) -1
E31: FORWARD (2,3) -1
E32: FORWARD (2,4) MUR -5
```

```

E33: TURN_LEFT (2,4) -1
>>> OBSTACLE en (1,4)
E34: FORWARD (2,4) OBS(1,4) -10
E35: FORWARD (2,4) OBS(1,4) -10
E36: TURN_LEFT (2,4) -1
E37: FORWARD (2,4) -1
E38: TURN_RIGHT (2,3) -1
E39: FORWARD (2,3) -1
E40: FORWARD (1,3) -1
E41: FORWARD (0,3) MUR -5
E42: TURN_LEFT (0,3) -1
E43: FORWARD (0,3) -1
E44: FORWARD (0,2) -1
E45: TURN_LEFT (0,1) -1
E46: FORWARD (0,1) NEW_OBS(1,1) -10
E47: TURN_LEFT (0,1) -1
E48: FORWARD (0,1) -1
E49: FORWARD (0,2) -1
Fin: 50 ep, R=-114
=== FIN DE CYCLE === Repos 10s ===

--- Cycle 2 (eps=0.30) ---
E0: FORWARD (0,0) -1
E1: FORWARD (0,1) -1
E2: TURN_LEFT (0,2) -1
E3: FORWARD (0,2) MUR -5
E4: TURN_LEFT (0,2) -1
E5: TURN_LEFT (0,2) -1
E6: FORWARD (0,2) -1
E7: TURN_LEFT (1,2) -1
E8: FORWARD (1,2) -1
E9: FORWARD (1,3) -1
E10: TURN_RIGHT (1,4) -1
E11: FORWARD (1,4) -1
E12: FORWARD (2,4) -1
E13: FORWARD (3,4) -1
>>> GOAL +100 <<<
Fin: 14 ep, R=83 GOAL!
=== FIN DE CYCLE === Repos 10s ===

--- Cycle 3 (eps=0.30) ---
E0: FORWARD (0,0) -1
E1: TURN_LEFT (0,1) -1
E2: FORWARD (0,1) MUR -5
E3: TURN_RIGHT (0,1) -1
E4: FORWARD (0,1) -1
E5: FORWARD (0,2) -1
E6: FORWARD (0,3) -1
E7: TURN_RIGHT (0,4) -1

```

```

E8: FORWARD (0,4) -1
>>> OBSTACLE en (2,4)
E9: TURN_RIGHT (1,4) -1
E10: FORWARD (1,4) -1
E11: FORWARD (1,3) -1
E12: FORWARD (1,2) -1
E13: FORWARD (1,1) -1
E14: TURN_RIGHT (1,0) -1
E15: FORWARD (1,0) -1
E16: TURN_RIGHT (0,0) -1
E17: FORWARD (0,0) -1
E18: FORWARD (0,1) -1
E19: TURN_RIGHT (0,2) -1
E20: TURN_LEFT (0,2) -1
E21: FORWARD (0,2) -1
E22: FORWARD (0,3) -1
E23: TURN_RIGHT (0,4) -1
E24: FORWARD (0,4) -1
E25: TURN_RIGHT (1,4) -1
E26: TURN_LEFT (1,4) -1
E27: FORWARD (1,4) OBS(2,4) -10
E28: FORWARD (1,4) OBS(2,4) -10
E29: FORWARD (1,4) OBS(2,4) -10
E30: FORWARD (1,4) OBS(2,4) -10
E31: FORWARD (1,4) OBS(2,4) -10
E32: TURN_LEFT (1,4) -1
E33: TURN_RIGHT (1,4) -1
E34: TURN_LEFT (1,4) -1
E35: TURN_RIGHT (1,4) -1
E36: TURN_LEFT (1,4) -1
E37: TURN_LEFT (1,4) -1
E38: FORWARD (1,4) -1
E39: TURN_LEFT (0,4) -1
E40: FORWARD (0,4) -1
E41: TURN_LEFT (0,3) -1
E42: FORWARD (0,3) -1
E43: TURN_RIGHT (1,3) -1
E44: TURN_RIGHT (1,3) -1
E45: TURN_LEFT (1,3) -1
E46: TURN_RIGHT (1,3) -1
E47: TURN_RIGHT (1,3) -1
E48: TURN_RIGHT (1,3) -1
E49: TURN_RIGHT (1,3) -1
Fin: 50 ep, R=-99
'=== FIN DE CYCLE === Repos 10s ===
  ↪

```

Annexes

Annexe A : Code complet simulation Python

```
1 import pygame
2 import sys
3 import numpy as np
4 import random
5 import pandas as pd
6 import time
7 from collections import deque
8 import argparse
9 import uuid
10
11 # -----
12 # Hyperparameters
13 # -----
14 #
15 learning_rate = 0.3
16 discount_factor = 0.9
17 epsilon = 0.7
18 epsilon_min = 0.01
19 epsilon_decay = 0.997
20 num_episodes = 5000
21 grid_size = 15
22 obstacle_change_interval = 200
23 max_steps_per_episode = 250
24 cell_size = 40
25 screen_size = grid_size * cell_size
26 tick_time = 240
27 episodes_output = "episode_data.xlsx"
28
29 # Colors
30 WHITE = (255, 255, 255)
31 BLACK = (0, 0, 0)
32 RED = (200, 30, 30)
33 PINK = (255, 170, 170)
34 GREEN = (0, 200, 0)
35 BLUE = (30, 144, 255)
36 GREY = (200, 200, 200)
37
38 # Directions: 0-North (up), 1-East (right), 2-South (down), 3-West (left)
39 deltas = [(-1, 0), (0, 1), (1, 0), (0, -1)]
```

```

40 dir_names = ['North', 'East', 'South', 'West']
41
42 # Actions: 0-forward, 1-turn_left, 2-turn_right
43 action_names = ['forward', 'turn_left', 'turn_right']
44 act_chars = ['F', 'L', 'R']
45 action_to_idx = {name: idx for idx, name in enumerate(action_names)}
46
47 # -----
48 # Environment class
49 # -----
50 class Environment:
51     def __init__(self):
52         self.start_pos = (0, 0)
53         self.start_facing = 1 # East
54         self.goal = (grid_size - 1, grid_size - 1)
55         self.true_obstacles = set()
56         self.known_obstacles = set()
57         self.position = self.start_pos
58         self.facing = self.start_facing
59
60     def reset(self, keep_known=False):
61         self.position = self.start_pos
62         self.facing = self.start_facing
63         if not keep_known:
64             self.known_obstacles = set()
65         self.reveal_neighborhood()
66         return (self.position[0], self.position[1], self.facing)
67
68     def add_random_obstacles(self, num):
69         self.true_obstacles = set()
70         attempts = 0
71         while len(self.true_obstacles) < num and attempts < num * 30:
72             attempts += 1
73             x = random.randint(0, grid_size - 1)
74             y = random.randint(0, grid_size - 1)
75             coord = (x, y)
76             if coord == self.start_pos or coord == self.goal:
77                 continue
78             self.true_obstacles.add(coord)
79         while not self.is_path_possible():
80             if not self.true_obstacles:
81                 break
82             self.true_obstacles.pop()
83
84     def is_path_possible(self):
85         visited = set()
86         queue = deque([(self.start_pos, self.start_facing)])
87         visited.add((self.start_pos[0], self.start_pos[1], self.start_facing))
88         while queue:
89             (x, y), f = queue.popleft()
90             if (x, y) == self.goal:

```

```

91         return True
92         # Try forward
93         dx, dy = deltas[f]
94         nx, ny = x + dx, y + dy
95         if 0 <= nx < grid_size and 0 <= ny < grid_size and (nx, ny) not in
↪ self.true_obstacles:
96             if (nx, ny, f) not in visited:
97                 visited.add((nx, ny, f))
98                 queue.append((nx, ny, f))
99         # Try turn_left
100         nf = (f - 1) % 4
101         if (x, y, nf) not in visited:
102             visited.add((x, y, nf))
103             queue.append((x, y, nf))
104         # Try turn_right
105         nf = (f + 1) % 4
106         if (x, y, nf) not in visited:
107             visited.add((x, y, nf))
108             queue.append((x, y, nf))
109         return False
110
111     def reveal_neighborhood(self):
112         x, y = self.position
113         dx, dy = deltas[self.facing]
114         nx, ny = x + dx, y + dy
115         if 0 <= nx < grid_size and 0 <= ny < grid_size:
116             coord = (nx, ny)
117             if coord in self.true_obstacles:
118                 self.known_obstacles.add(coord)
119
120     def step(self, action):
121         done = False
122         if action == 1: # turn_left
123             self.facing = (self.facing - 1) % 4
124             reward = -1
125             self.reveal_neighborhood()
126             return (self.position[0], self.position[1], self.facing), reward, done
127         elif action == 2: # turn_right
128             self.facing = (self.facing + 1) % 4
129             reward = -1
130             self.reveal_neighborhood()
131             return (self.position[0], self.position[1], self.facing), reward, done
132
133         # action == 0: forward
134         dx, dy = deltas[self.facing]
135         tx = max(0, min(self.position[0] + dx, grid_size - 1))
136         ty = max(0, min(self.position[1] + dy, grid_size - 1))
137         target = (tx, ty)
138
139         if target == self.position: # hit wall
140             reward = -5

```

```

141         self.reveal_neighborhood()
142         return (self.position[0], self.position[1], self.facing), reward, done
143
144     if target in self.true_obstacles:
145         reward = -10
146         self.reveal_neighborhood()
147         return (self.position[0], self.position[1], self.facing), reward, done
148
149     self.position = target
150     self.reveal_neighborhood()
151     if self.position == self.goal:
152         reward = 100
153         done = True
154     else:
155         reward = -1
156     return (self.position[0], self.position[1], self.facing), reward, done
157
158 # -----
159 # Utilities
160 # -----
161 def get_obstacle_config_key(obstacles):
162     return tuple(sorted(obstacles))
163
164 def draw_grid(screen, env, episode, cycle):
165     screen.fill(WHITE)
166     for x in range(0, screen_size, cell_size):
167         pygame.draw.line(screen, GREY, (x, 0), (x, screen_size))
168     for y in range(0, screen_size, cell_size):
169         pygame.draw.line(screen, GREY, (0, y), (screen_size, y))
170
171     for ox, oy in env.true_obstacles:
172         pygame.draw.rect(screen, RED, (oy * cell_size, ox * cell_size, cell_size,
173 ↪ cell_size))
174     for ox, oy in env.known_obstacles:
175         pygame.draw.rect(screen, PINK, (oy * cell_size + 6, ox * cell_size + 6,
176 ↪ cell_size - 12, cell_size - 12))
177
178     pygame.draw.rect(screen, BLUE, (env.start_pos[1] * cell_size, env.start_pos[0] *
179 ↪ cell_size, cell_size, cell_size))
180     pygame.draw.rect(screen, GREEN, (env.goal[1] * cell_size, env.goal[0] *
181 ↪ cell_size, cell_size, cell_size))
182
183     font = pygame.font.SysFont(None, 22)
184     txt = font.render(f"Episode: {episode}/{num_episodes} Cycle: {cycle} Known:
185 ↪ {len(env.known_obstacles)}", True, BLACK)
186     screen.blit(txt, (8, 6))
187
188 def get_learned_path(env, Q, max_steps=max_steps_per_episode):
189     sim_env = Environment()
190     sim_env.start_pos = env.start_pos
191     sim_env.start_facing = env.start_facing

```

```

187     sim_env.goal = env.goal
188     sim_env.true_obstacles = set(env.true_obstacles)
189     state = sim_env.reset(keep_known=False)
190     path = [state]
191     done = False
192     steps = 0
193     while not done and steps < max_steps:
194         x, y, f = state
195         action = int(np.argmax(Q[x, y, f]))
196         new_state, _, done = sim_env.step(action)
197         path.append(new_state)
198         state = new_state
199         steps += 1
200     return path
201
202 def animate_path(screen, path, env, episode, cycle, tick_time=60):
203     index = 0
204     clock = pygame.time.Clock()
205     font = pygame.font.SysFont(None, 22)
206     while index < len(path):
207         for event in pygame.event.get():
208             if event.type == pygame.QUIT:
209                 return False
210             draw_grid(screen, env, episode, cycle)
211             x, y, f = path[index]
212             cx = y * cell_size + cell_size // 2
213             cy = x * cell_size + cell_size // 2
214             radius = cell_size // 3
215             pygame.draw.circle(screen, BLUE, (cx, cy), radius)
216             drow, dcol = deltas[f]
217             pygame.draw.line(screen, BLACK, (cx, cy), (cx + dcol * radius, cy + drow *
↪ radius), 3)
218             txt = font.render(f"Step: {index}/{len(path)-1}", True, BLACK)
219             screen.blit(txt, (10, 30))
220             pygame.display.flip()
221             clock.tick(tick_time)
222             index += 1
223     time.sleep(0.3)
224     return True
225
226 def print_policy_to_console(Q, env):
227     for f in range(4):
228         print(f"\nPolicy facing {dir_names[f]} (S=start, G=goal, X=obst,
↪ F/L/R=actions):")
229         for x in range(grid_size):
230             row = []
231             for y in range(grid_size):
232                 pos = (x, y)
233                 if pos == env.start_pos:
234                     row.append('S ')
235                 elif pos == env.goal:

```

```

236         row.append('G ')
237     elif pos in env.true_obstacles:
238         row.append('X ')
239     else:
240         a = np.argmax(Q[x, y, f])
241         row.append(act_chars[a] + ' ')
242     print(''.join(row))
243
244 def initialize_q_with_preferred_path(Q, preferred_path, env):
245     sim_env = Environment()
246     sim_env.start_pos = env.start_pos
247     sim_env.start_facing = env.start_facing
248     sim_env.goal = env.goal
249     sim_env.true_obstacles = set()
250     state = sim_env.reset(keep_known=False)
251     q_bonus = 50.0
252     for action_name in preferred_path:
253         if action_name not in action_to_idx:
254             print(f"Invalid action '{action_name}' in preferred path. Skipping.")
255             continue
256         action_idx = action_to_idx[action_name]
257         x, y, f = state
258         Q[x, y, f, action_idx] += q_bonus
259         new_state, _, done = sim_env.step(action_idx)
260         state = new_state
261         if done:
262             break
263
264 # -----
265 # Excel/CSV append helper
266 # -----
267 from openpyxl import load_workbook
268
269 def save_policies_append(policy_db, filename="policy_database.xlsx", use_csv=False):
270     if use_csv:
271         for idx, (config_key, q_table) in enumerate(policy_db.items()):
272             rows = []
273             for x in range(q_table.shape[0]):
274                 for y in range(q_table.shape[1]):
275                     for f in range(4):
276                         pos = (x, y)
277                         if pos == (0, 0):
278                             action = "Start"
279                         elif pos == (q_table.shape[0] - 1, q_table.shape[1] - 1):
280                             action = "Goal"
281                         elif pos in config_key:
282                             action = "Obstacle"
283                         else:
284                             action_idx = int(np.argmax(q_table[x, y, f]))
285                             action = action_names[action_idx]
286             rows.append({

```



```

287         "x": x, "y": y, "facing": dir_names[f],
288         "action": action, "obstacle_config": str(config_key)
289     })
290     df = pd.DataFrame(rows)
291     csv_file = f"policy_config_{idx}_{len(config_key)}obs.csv"
292     df.to_csv(csv_file, index=False)
293     print(f"Policy saved to '{csv_file}'.")
294 else:
295     try:
296         book = load_workbook(filename)
297         existing_sheets = [ws.title for ws in book.worksheets]
298         with pd.ExcelWriter(filename, engine="openpyxl", mode="a",
299 ↪ if_sheet_exists="new") as writer:
300             start_idx = len(existing_sheets)
301             for idx, (config_key, q_table) in enumerate(policy_db.items(),
302 ↪ start=start_idx):
303                 rows = []
304                 for x in range(q_table.shape[0]):
305                     for y in range(q_table.shape[1]):
306                         for f in range(4):
307                             pos = (x, y)
308                             if pos == (0, 0):
309                                 action = "Start"
310                             elif pos == (q_table.shape[0] - 1, q_table.shape[1]
311 ↪ - 1):
312                                 action = "Goal"
313                             elif pos in config_key:
314                                 action = "Obstacle"
315                             else:
316                                 action_idx = int(np.argmax(q_table[x, y, f]))
317                                 action = action_names[action_idx]
318                                 rows.append({
319                                     "x": x, "y": y, "facing": dir_names[f],
320                                     "action": action, "obstacle_config":
321 ↪ str(config_key)
322                                 })
323                 df = pd.DataFrame(rows)
324                 sheet_name = f"Config_{idx}_{len(config_key)}obs"
325                 df.to_excel(writer, sheet_name=sheet_name[:31], index=False)
326 except FileNotFoundError:
327     with pd.ExcelWriter(filename, engine="openpyxl", mode="w") as writer:
328         for idx, (config_key, q_table) in enumerate(policy_db.items()):
329             rows = []
330             for x in range(q_table.shape[0]):
331                 for y in range(q_table.shape[1]):
332                     for f in range(4):
333                         pos = (x, y)
334                         if pos == (0, 0):
335                             action = "Start"
336                         elif pos == (q_table.shape[0] - 1, q_table.shape[1]
337 ↪ - 1):

```

```

333         action = "Goal"
334     elif pos in config_key:
335         action = "Obstacle"
336     else:
337         action_idx = int(np.argmax(q_table[x, y, f]))
338         action = action_names[action_idx]
339     rows.append({
340         "x": x, "y": y, "facing": dir_names[f],
341         "action": action, "obstacle_config":
↪ str(config_key)
342     })
343     df = pd.DataFrame(rows)
344     sheet_name = f"Config_{idx}_{len(config_key)}obs"
345     df.to_excel(writer, sheet_name=sheet_name[:31], index=False)
346     print(f"Policies appended to '{filename}'.")
347
348 # -----
349 # Main training loop
350 # -----
351 def main(preferred_path=None, no_render=False, use_csv=False):
352     global epsilon
353     pygame.init()
354     screen = None if no_render else pygame.display.set_mode((screen_size,
↪ screen_size))
355     if not no_render:
356         pygame.display.set_caption('Q-Learning Partial Observability - Car-like
↪ Agent')
357
358     env = Environment()
359     Q = np.zeros((grid_size, grid_size, 4, 3)) # Updated for 3 actions
360     policy_db_local = {}
361     episode_data = [] # To store paths and rewards
362     cycle = 0
363
364     env.true_obstacles = set()
365     env.reset(keep_known=False)
366     if preferred_path:
367         print(f"Applying preferred path for empty map: {preferred_path}")
368         initialize_q_with_preferred_path(Q, preferred_path, env)
369
370     for episode in range(num_episodes):
371         if not no_render:
372             for evt in pygame.event.get():
373                 if evt.type == pygame.QUIT:
374                     pygame.quit()
375                     sys.exit()
376
377         if episode % obstacle_change_interval == 0:
378             cycle += 1
379             if episode == 0:
380                 env.true_obstacles = set()

```

```

381         print(f"[Cycle {cycle}] Starting with empty obstacle map.")
382     else:
383         env.add_random_obstacles(max(4, int(grid_size * 0.8)))
384         print(f"[Cycle {cycle}] New true obstacles set
↪ ({len(env.true_obstacles)} obstacles).")
385
386     state = env.reset(keep_known=False)
387     done = False
388     steps = 0
389     total_reward = 0
390     path = [state]
391
392     while not done and steps < max_steps_per_episode:
393         steps += 1
394         x, y, f = state
395         if random.random() < epsilon:
396             action = random.randint(0, 2) # Updated for 3 actions
397         else:
398             action = int(np.argmax(Q[x, y, f]))
399
400         new_state, reward, done = env.step(action)
401         total_reward += reward
402         path.append(new_state)
403         nx, ny, nf = new_state
404         best_next = int(np.argmax(Q[nx, ny, nf]))
405         td_target = reward + discount_factor * Q[nx, ny, nf, best_next]
406         td_error = td_target - Q[x, y, f, action]
407         Q[x, y, f, action] += learning_rate * td_error
408         state = new_state
409
410     if not done:
411         last_x, last_y, last_f = state
412         Q[last_x, last_y, last_f, :] -= 0.02
413
414     # Store episode data
415     episode_data.append({
416         'episode': episode + 1,
417         'cycle': cycle,
418         'path': path,
419         'total_reward': total_reward,
420         'steps': steps,
421         'reached_goal': done,
422         'obstacle_config': str(get_obstacle_config_key(env.true_obstacles))
423     })
424
425     print(f"[Cycle {cycle}] Episode {episode+1}/{num_episodes}: steps={steps},
↪ reached={done}, eps={epsilon:.3f}, total_reward={total_reward:.2f}")
426
427     if episode % obstacle_change_interval == obstacle_change_interval - 1 or
↪ episode == num_episodes - 1:
428         cfg_key = get_obstacle_config_key(env.true_obstacles)

```

```

429     policy_db_local[cfg_key] = Q.copy()
430     print(f"[Cycle {cycle}] Snapshot saved for config (obs={len(cfg_key)}).")
431     learned_path = get_learned_path(env, Q)
432     print(f"[Cycle {cycle}] Learned path length: {len(learned_path)}")
433     print_policy_to_console(Q, env)
434     if not no_render:
435         cont = animate_path(screen, learned_path, env, episode + 1, cycle,
↪ tick_time=int(tick_time / 2))
436         if not cont:
437             break
438
439     epsilon = max(epsilon_min, epsilon * epsilon_decay)
440
441     print('Training finished. Saving all snapshots and episode data.')
442     # if policy_db_local:
443     #     save_policies_append(policy_db_local, use_csv=use_csv)
444
445     # Save episode data to CSV
446     episode_df = pd.DataFrame([
447         {
448             'episode': data['episode'],
449             'cycle': data['cycle'],
450             'path': ';'.join([f"({x},{y},{dir_names[f]})" for x, y, f in
↪ data['path']]),
451             'total_reward': data['total_reward'],
452             'steps': data['steps'],
453             'reached_goal': data['reached_goal'],
454             'obstacle_config': data['obstacle_config']
455         } for data in episode_data
456     ])
457     params_df = pd.DataFrame({
458         'learning_rate': [learning_rate],
459         'discount_factor': [discount_factor],
460         'epsilon': [epsilon],
461         'epsilon_min': [epsilon_min],
462         'epsilon_decay': [epsilon_decay],
463         'num_episodes': [num_episodes],
464         'grid_size': [grid_size],
465         'obstacle_change_interval': [obstacle_change_interval],
466         'max_steps_per_episode': [max_steps_per_episode]
467     })
468     with pd.ExcelWriter(episodes_output, engine='openpyxl', mode='w') as writer:
469         params_df.to_excel(writer, sheet_name='Hyperparameters', index=False)
470         episode_df.to_excel(writer, sheet_name='EpisodeData', index=False)
471     print(f"Episode data saved to {episodes_output}.")
472
473     if not no_render:
474         last_path = get_learned_path(env, Q)
475         index = 0
476         clock = pygame.time.Clock()
477         font = pygame.font.SysFont(None, 22)

```

```

478     running = True
479     while running:
480         for evt in pygame.event.get():
481             if evt.type == pygame.QUIT:
482                 running = False
483         draw_grid(screen, env, num_episodes, cycle)
484         if index < len(last_path):
485             x, y, f = last_path[index]
486             cx = y * cell_size + cell_size // 2
487             cy = x * cell_size + cell_size // 2
488             radius = cell_size // 3
489             pygame.draw.circle(screen, BLUE, (cx, cy), radius)
490             drow, dcol = deltas[f]
491             pygame.draw.line(screen, BLACK, (cx, cy), (cx + dcol * radius, cy +
↪ drow * radius), 3)
492             txt = font.render(f"Step: {index}/{len(last_path)-1}", True, BLACK)
493             screen.blit(txt, (10, 30))
494             index += 1
495         else:
496             index = 0
497         pygame.display.flip()
498         clock.tick(tick_time)
499
500     pygame.quit()
501
502 if __name__ == '__main__':
503     parser = argparse.ArgumentParser(description='Q-Learning with Partial
↪ Observability')
504     parser.add_argument('--no-render', action='store_true', help='Disable Pygame
↪ rendering')
505     parser.add_argument('--use-csv', action='store_true', help='Save policies as CSV
↪ instead of Excel')
506     args = parser.parse_args()
507
508     preferred_path = ['forward']*(grid_size-1) + ['turn_right'] +
↪ ['forward']*(grid_size-1)
509     main(preferred_path=preferred_path, no_render=args.no_render,
↪ use_csv=args.use_csv)
510

```

Annexe B : Code complet ARDUINO

```

1  // Q-Learning Obstacle Avoidance Robot in 5x5 Grid World - Arduino Uno
2  // Avec enregistrement de toutes les donnees d'apprentissage
3  // Terminologie : Cycle = ensemble complet du dbut au goal, pisode = chaque action
4  #define ENA 5
5  #define IN1 6
6  #define IN2 7
7  #define IN3 8
8  #define IN4 9
9  #define ENB 10
10 #define TRIG_PIN 11
11 #define ECHO_PIN 12
12 #define GRID_SIZE 5
13 #define NUM_ACTIONS 3
14 #define CELL_DIST 20
15 #define MOVE_TIME 1000
16 #define TURN_TIME 500
17 #define ACTION_WAIT 1000
18 #define RESET_DELAY 10000
19 #define MAX_STEPS 50
20 #define LEARNING_RATE 0.2
21 #define DISCOUNT 0.9
22 #define EPSILON 0.3
23 #define EPSILON_MIN 0.01
24 #define EPSILON_DECAY 0.997
25 #define NUM_CYCLES 50 // Rduit pour conomiser la mmoire
26
27 // Tables Q-learning
28 int8_t Q[GRID_SIZE][GRID_SIZE][4][NUM_ACTIONS] = {0};
29
30 // Variables d'tat
31 int pos_x = 0, pos_y = 0, facing = 1;
32 int goal_x = 4, goal_y = 4;
33 bool known_obstacles[GRID_SIZE][GRID_SIZE] = {false};
34 float epsilon = EPSILON;
35
36 // TABLEAUX D'ENREGISTREMENT DES DONNES - OPTIMISS
37 int8_t cycle_steps[NUM_CYCLES]; // Nombre d'pisodes (actions) par cycle
38 int cycle_rewards[NUM_CYCLES]; // Rcompense totale par cycle (int au lieu de
    ↪ float)
39 byte cycle_success[NUM_CYCLES]; // 0=chec, 1=succs (byte au lieu de bool)
40
41 // Statistiques en temps rel (pas de stockage complet)
42 int current_cycle = 0;
43 int current_episodes = 0;
44 int current_reward = 0;
45 unsigned long cycle_start_time = 0;
46
47 // Constantes
48 const int deltas[4][2] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

```

```
49 const char* dir_names[4] = {"North", "East", "South", "West"};
50 const char* action_names[3] = {"FORWARD", "TURN_LEFT", "TURN_RIGHT"};
51
52 void setup() {
53     pinMode(TRIG_PIN, OUTPUT);
54     pinMode(ECHO_PIN, INPUT);
55     pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
56     pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
57     pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
58     pinMode(LED_BUILTIN, OUTPUT);
59     digitalWrite(LED_BUILTIN, LOW);
60
61     Serial.begin(9600);
62     randomSeed(analogRead(0));
63
64     // Initialisation des tableaux
65     for (int i = 0; i < NUM_CYCLES; i++) {
66         cycle_steps[i] = 0;
67         cycle_rewards[i] = 0;
68         cycle_success[i] = 0;
69     }
70
71     Serial.println(F("=== Q-Learning Robot - Mmoire Optimise ==="));
72     Serial.println(F("Start: (0,0) facing East | Goal: (4,4)"));
73     Serial.println(F("Cycles: 50 | Terminologie: Cycle = parcours, pisode =
    ↩ action"));
74     Serial.println(F("Dbut de l'apprentissage...\n"));
75     initializeQWithPreferred();
76 }
77
78 void blinkOnAction() {
79     digitalWrite(LED_BUILTIN, HIGH);
80     delay(100);
81     digitalWrite(LED_BUILTIN, LOW);
82     delay(100);
83 }
84
85 void blinkDuringReset() {
86     unsigned long startTime = millis();
87     bool ledState = false;
88
89     Serial.println(F("=== FIN DE CYCLE === Repos 10s ==="));
90
91     while (millis() - startTime < RESET_DELAY) {
92         if ((millis() - startTime) % 1000 < 500) {
93             if (!ledState) {
94                 digitalWrite(LED_BUILTIN, HIGH);
95                 ledState = true;
96             }
97         } else {
98             if (ledState) {
```

```

99         digitalWrite(LED_BUILTIN, LOW);
100         ledState = false;
101     }
102 }
103     delay(10);
104 }
105     digitalWrite(LED_BUILTIN, LOW);
106 }
107
108 void initializeQWithPreferred() {
109     int sim_x = 0, sim_y = 0, sim_f = 1;
110     float bonus = 50.0;
111     for (int i = 0; i < 4; i++) {
112         Q[sim_x][sim_y][sim_f][0] += bonus;
113         sim_y += deltas[sim_f][1];
114     }
115     Q[sim_x][sim_y][sim_f][2] += bonus;
116     sim_f = (sim_f + 1) % 4;
117     for (int i = 0; i < 4; i++) {
118         Q[sim_x][sim_y][sim_f][0] += bonus;
119         sim_x += deltas[sim_f][0];
120     }
121 }
122
123 int getDistance() {
124     digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
125     digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
126     digitalWrite(TRIG_PIN, LOW);
127     long duration = pulseIn(ECHO_PIN, HIGH);
128     return duration * 0.034 / 2;
129 }
130
131 bool revealNeighborhood() {
132     int dx = deltas[facing][0];
133     int dy = deltas[facing][1];
134     int nx = pos_x + dx;
135     int ny = pos_y + dy;
136     if (nx >= 0 && nx < GRID_SIZE && ny >= 0 && ny < GRID_SIZE) {
137         int dist = getDistance();
138         if (dist < CELL_DIST && !known_obstacles[nx][ny]) {
139             known_obstacles[nx][ny] = true;
140             Serial.print(F(">>> OBSTACLE en ("));
141             Serial.print(nx); Serial.print(","); Serial.print(ny);
142             Serial.println(F(")"));
143
144             // VRIFIER SI OBSTACLE AU GOAL
145             if (nx == goal_x && ny == goal_y) {
146                 Serial.println(F("!!!!!! OBSTACLE AU GOAL (4,4) - CYCLE INVALIDE
147                 ↪ !!!!!"));
148                 return true; // Signaler qu'il faut redmarrer le cycle
149             }
150         }
151     }

```



```

149     }
150 }
151 return false; // Pas d'obstacle au goal
152 }
153
154 void reset() {
155     pos_x = 0;
156     pos_y = 0;
157     facing = 1;
158     memset(known_obstacles, 0, sizeof(known_obstacles));
159
160     // Vrifier si obstacle au goal ds le dbut
161     if (revealNeighborhood()) {
162         // Obstacle au goal dteet - attendre repositionnement
163         Serial.println(F(">>> Attente repositionnement obstacle du goal - nouveau
↪ cycle..."));
164         blinkDuringReset();
165         reset(); // Ressayer le cycle
166     }
167 }
168
169 int chooseAction() {
170     if (random(1000) / 1000.0 < epsilon) {
171         return random(NUM_ACTIONS);
172     } else {
173         int best = 0;
174         for (int a = 1; a < NUM_ACTIONS; a++) {
175             if (Q[pos_x][pos_y][facing][a] > Q[pos_x][pos_y][facing][best]) best = a;
176         }
177         return best;
178     }
179 }
180
181 float executeAction(int action, int step) {
182     float reward = 0;
183
184     // Affichage simplifi
185     Serial.print(F("E"));
186     Serial.print(step);
187     Serial.print(F(": "));
188     Serial.print(action_names[action]);
189     Serial.print(F(" ("));
190     Serial.print(pos_x); Serial.print(","); Serial.print(pos_y);
191     Serial.print(F(")"));
192
193     blinkOnAction();
194
195     if (action == 1) { // turn left
196         turnLeft(TURN_TIME);
197         facing = (facing - 1 + 4) % 4;
198         reward = -1;

```

```
199     Serial.println(F(" -1"));
200 }
201 else if (action == 2) { // turn right
202     turnRight(TURN_TIME);
203     facing = (facing + 1) % 4;
204     reward = -1;
205     Serial.println(F(" -1"));
206 }
207 else { // forward
208     int dx = deltas[facing][0];
209     int dy = deltas[facing][1];
210     int nx = pos_x + dx;
211     int ny = pos_y + dy;
212     if (nx < 0 || nx >= GRID_SIZE || ny < 0 || ny >= GRID_SIZE) {
213         Serial.println(F(" MUR -5"));
214         reward = -5;
215     }
216     else if (known_obstacles[nx][ny]) {
217         Serial.print(F(" OBS("));
218         Serial.print(nx); Serial.print(","); Serial.print(ny);
219         Serial.println(F(") -10"));
220         reward = -10;
221
222         if (nx == goal_x && ny == goal_y) {
223             reward = -999;
224         }
225     }
226     else {
227         int dist = getDistance();
228         if (dist < CELL_DIST) {
229             known_obstacles[nx][ny] = true;
230             Serial.print(F(" NEW_OBS("));
231             Serial.print(nx); Serial.print(","); Serial.print(ny);
232             Serial.println(F(") -10"));
233             reward = -10;
234
235             if (nx == goal_x && ny == goal_y) {
236                 Serial.println(F("!!! GOAL BLOQU !!!"));
237                 reward = -999;
238             }
239         }
240         else {
241             moveForward(MOVE_TIME);
242             pos_x = nx;
243             pos_y = ny;
244             reward = -1;
245             Serial.println(F(" -1"));
246         }
247     }
248 }
249 }
```

```
250     if (pos_x == goal_x && pos_y == goal_y) {
251         Serial.println(F(">>> GOAL +100 <<<"));
252         reward = 100;
253     }
254
255     revealNeighborhood();
256     //delay(ACTION_WAIT);
257     return reward;
258 }
259
260 void moveForward(int ms) {
261     analogWrite(ENA, 200); analogWrite(ENB, 200);
262     digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
263     digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
264     delay(ms);
265     stopMotors();
266 }
267
268 void turnLeft(int ms) {
269     analogWrite(ENA, 180); analogWrite(ENB, 180);
270     digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
271     digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
272     delay(ms);
273     stopMotors();
274 }
275
276 void turnRight(int ms) {
277     analogWrite(ENA, 180); analogWrite(ENB, 180);
278     digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
279     digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
280     delay(ms);
281     stopMotors();
282 }
283
284 void stopMotors() {
285     digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
286     digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
287 }
288
289 // FONCTION POUR AFFICHER LES STATISTIQUES - SIMPLIFI
290 void printStatistics() {
291     Serial.println(F("\n===== STATISTIQUES ====="));
292     Serial.println(F("Cycle,Episodes,Reward,Success"));
293
294     int total_episodes = 0;
295     long total_rewards = 0;
296     int success_count = 0;
297
298     for (int i = 0; i < NUM_CYCLES; i++) {
299         Serial.print(i + 1); Serial.print(",");
300         Serial.print(cycle_steps[i]); Serial.print(",");
```

```
301     Serial.print(cycle_rewards[i]); Serial.print(",");
302     Serial.println(cycle_success[i]);
303
304     total_episodes += cycle_steps[i];
305     total_rewards += cycle_rewards[i];
306     if (cycle_success[i]) success_count++;
307 }
308
309 Serial.println(F("\n--- RSUM ---"));
310 Serial.print(F("Moy pisodes: "));
311 Serial.println((float)total_episodes / NUM_CYCLES);
312 Serial.print(F("Moy reward: "));
313 Serial.println((float)total_rewards / NUM_CYCLES);
314 Serial.print(F("Succs: "));
315 Serial.print((float)success_count * 100 / NUM_CYCLES);
316 Serial.println(F("%"));
317 }
318
319 void loop() {
320     static int cycle = 0;
321
322     if (cycle >= NUM_CYCLES) {
323         // AFFICHER LES STATISTIQUES
324         printStatistics();
325
326         Serial.println(F("\n=== MODE EXPLOITATION ==="));
327         epsilon = 0;
328         reset();
329
330         while (true) {
331             int action = chooseAction();
332             executeAction(action, 0);
333             if (pos_x == goal_x && pos_y == goal_y) {
334                 blinkDuringReset();
335                 reset();
336             }
337         }
338     }
339
340     // DBUT DU CYCLE
341     reset();
342
343     bool done = false;
344     bool cycle_invalid = false;
345     int episodes = 0;
346     int total_reward = 0;
347
348     Serial.print(F("\n--- Cycle "));
349     Serial.print(cycle + 1);
350     Serial.print(F(" (eps="));
351     Serial.print(epsilon, 2);
```

```

352 Serial.println(F(" ---"));
353
354 while (!done && episodes < MAX_STEPS) {
355     int action = chooseAction();
356     int old_x = pos_x, old_y = pos_y, old_f = facing;
357
358     float reward = executeAction(action, episodes);
359     total_reward += (int)reward;
360
361     // VRIFIER SI OBSTACLE AU GOAL
362     if (reward == -999) {
363         Serial.println(F(">>> REDMARRAGE CYCLE <<<"));
364         cycle_invalid = true;
365         blinkDuringReset();
366         break;
367     }
368
369     if (pos_x == goal_x && pos_y == goal_y) done = true;
370
371     // Mise jour Q
372     int8_t max_next = Q[pos_x][pos_y][facing][0];
373     for (int a = 1; a < NUM_ACTIONS; a++) {
374         if (Q[pos_x][pos_y][facing][a] > max_next) max_next =
375         ↪ Q[pos_x][pos_y][facing][a];
376     }
377
378     Q[old_x][old_y][old_f][action] = (int8_t)constrain(
379     ↪ Q[old_x][old_y][old_f][action] + (int8_t)round(LEARNING_RATE * 4 * (reward +
380     ↪ DISCOUNT * max_next - Q[old_x][old_y][old_f][action])),
381     -128, 127
382     );
383
384     episodes++;
385 }
386
387 // ENREGISTRER LES DONNES
388 if (!cycle_invalid) {
389     cycle_steps[cycle] = episodes;
390     cycle_rewards[cycle] = total_reward;
391     cycle_success[cycle] = done ? 1 : 0;
392
393     Serial.print(F("Fin: "));
394     Serial.print(episodes);
395     Serial.print(F(" ep, R="));
396     Serial.print(total_reward);
397     if (done) Serial.println(F(" GOAL!"));
398     else Serial.println();
399
400     epsilon = max(EPSILON_MIN, epsilon * EPSILON_DECAY);
401     blinkDuringReset();
402     cycle++;

```

```
401     }  
402 }  
403
```