



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

25 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 . . . . .	2
1 C'est quoi IAvoid . . . . .	3
2 Principe de fonctionnement . . . . .	3
3 L'apprentissage renforcé . . . . .	4
3.1 Q-Table . . . . .	4
3.2 Politique $\epsilon$ -cupide . . . . .	5
3.3 Equation de Bellman . . . . .	5
4 Simulation Python . . . . .	5
4.1 Hyperparamètres de la simulation . . . . .	5
4.2 Environnement de la simulation . . . . .	6
4.3 Résultats fin de la simulation . . . . .	7
4.4 Analyse des données . . . . .	8
5 Comparaison avec les mêmes paramètres . . . . .	9
5.1 Heatmap des zones les plus fréquentées . . . . .	9
5.2 Récompenses par itération . . . . .	10
5.3 Taux de réussite . . . . .	11
5.4 Conclusion . . . . .	11
6 Comparaison entre les paramètres . . . . .	11
6.1 Learning_rate . . . . .	12
6.2 Epsilon . . . . .	13
6.3 Discount_factor . . . . .	14
6.4 Conclusion . . . . .	15
7 Simulation réelle avec ARDUINO . . . . .	15
Annexes . . . . .	19
Bibliographie . . . . .	39

# Table des figures

2.1	Organigramme du fonctionnement général . . . . .	3
3.1	Principe de Q-Learning algorithm . . . . .	4
4.1	Organigramme général . . . . .	5
4.2	Environnement de simulation sur Python . . . . .	6
4.3	Simulation de la génération des obstacles . . . . .	7
4.4	Solution choisie pendant la dernière itération . . . . .	7
4.5	Meilleur solution choisie pendant une simulation/ meilleur récompense . . . . .	8
4.6	Stockage des données pour analyse . . . . .	8
4.7	Taux de succès d'une simulation . . . . .	9
4.8	Evolution de la récompense par itération . . . . .	9
4.9	Comparaison entre le cycle 4 et le dernier cycle . . . . .	9
5.1	Comparaison Heatmap - mêmes paramètres . . . . .	10
5.2	Comparaison Récompenses par itération lissées - mêmes paramètres . . . . .	10
5.3	Comparaison Taux de réussite - mêmes paramètres . . . . .	11
5.4	Comparaison récompense/steps - une simulation . . . . .	11
6.1	Comparaison Learning_rate - Heatmap . . . . .	12
6.2	Comparaison Learning_rate - Taux de réussite . . . . .	12
6.3	Comparaison Epsilon - Vision globale . . . . .	13
6.4	Comparaison Epsilon - Heatmap . . . . .	13
6.5	Comparaison Epsilon - Taux de réussite . . . . .	14
6.6	Comparaison Discount_factor - Heatmap . . . . .	14
6.7	Comparaison Discount_factor - Taux de réussite . . . . .	14
7.1	Chemin préféré - Simulation ARDUINO . . . . .	17
7.2	Test 1 - Simulation ARDUINO . . . . .	17
7.3	Test 2 - Simulation ARDUINO . . . . .	18
7.4	Test 3 - Simulation ARDUINO . . . . .	18

# Liste des tableaux

6.1	Liste des valeurs choisis pour la simulation réelle . . . . .	15
-----	---	----

## 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 :

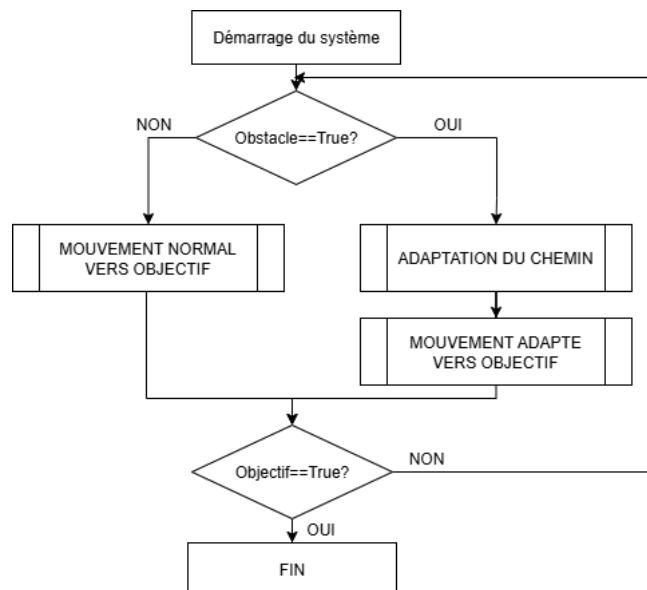


FIGURE 2.1 – Organigramme du fonctionnement général

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** vu 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)

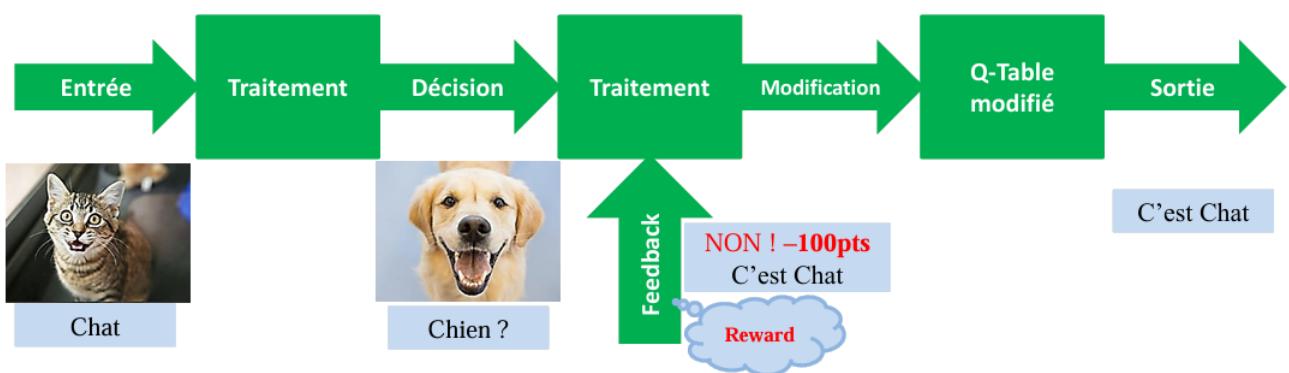


FIGURE 3.1 – Principe de Q-Learning algorithm

### 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.

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

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

### 3.2 Politique $\epsilon$ -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  $\epsilon$ , 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

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

$Q(S, A)$  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.

$\gamma$  est le facteur d'actualisation, représentant l'importance des récompenses futures.

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

## 4 Simulation Python

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

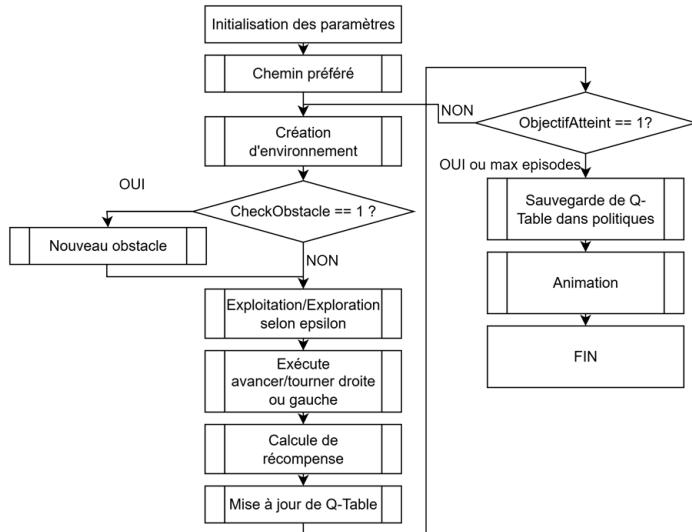


FIGURE 4.1 – Organigramme général

### 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

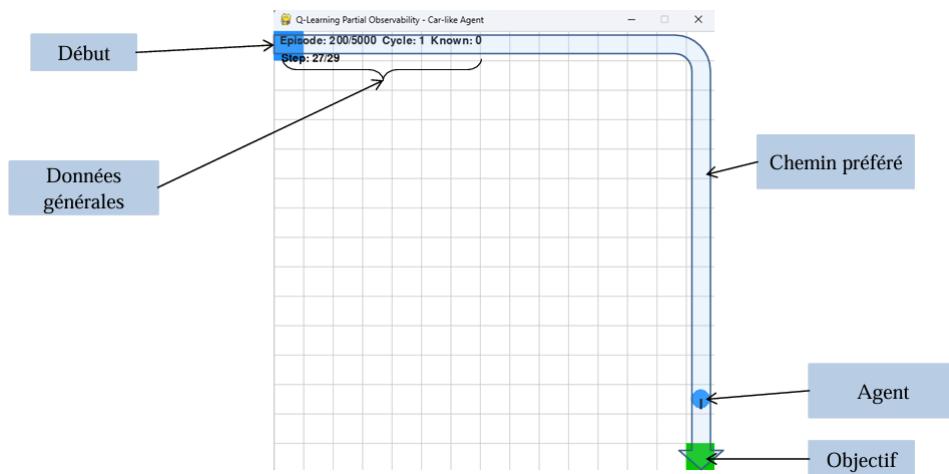


FIGURE 4.2 – Environnement de simulation sur Python

Comme déjà annoncé, le système se trouve avec des obstacles générés chaque fois d'une façon aléatoire et notre objectif est d'adapté 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.**

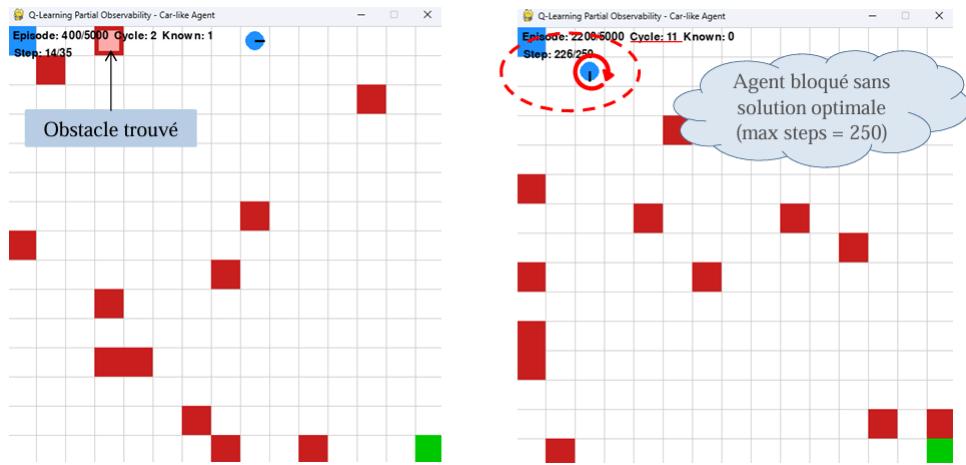


FIGURE 4.3 – Simulation de la génération des obstacles

### 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).

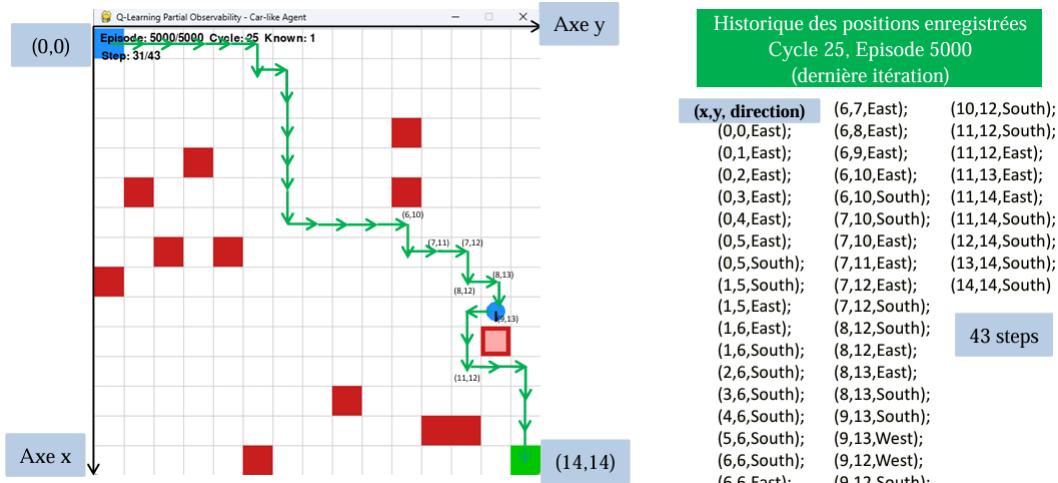


FIGURE 4.4 – Solution choisie pendant la dernière itération

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 meilleure solution à fur et à mesure)

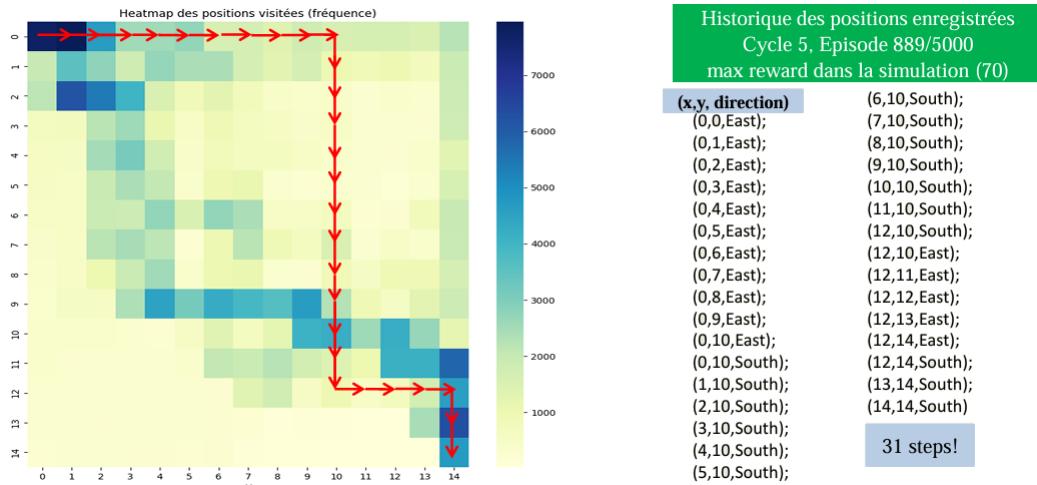


FIGURE 4.5 – Meilleur solution choisie pendant une simulation/ meilleur récompense

Cette solution donne la meilleure 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)

episode_data.xlsx							Policy_database.xlsx						
<pre> Policy facing West (S=start, G=goal, X=obst, F/L=R=actions): S L F F L R L R L L F F L L L R F R L L L F F F F F F L L R R L F L F L L R R R F L F L L R R F L L L F F X R F R L F R L R X F R L L R R L F F L F L X F L L R R R F F X L R L F R L L L L R L L R R R L L L L F L F X L X F R R F F F F L L L F X F R R L L R L R F R F L F L F R R L L R R L L R L R L F F R R R R R R L R R R R R R X L L R L R F R F L R R F R F R F R L L L R L P L R L X R F R F R R L L L L L R R R L R R R L X X L R L F R L R X R L R R L R L F G Training finished. Saving all snapshots and episode data. Policies appended to 'policy_database.xlsx'. Episode data saved to 'episode data.xlsx'. </pre>							<pre> 1 x y facing action stacie config 2 0 0 North Start ((2, 2), (3, 3), (4, 10), 3 0 0 East Start ((2, 2), (3, 3), (4, 10), 4 0 0 South Start ((2, 2), (3, 3), (4, 10), 5 0 0 West Start ((2, 2), (3, 3), (4, 10), 6 0 1 North turn_left ((2, 2), (3, 3), (4, 10), 7 0 1 East forward ((2, 2), (3, 3), (4, 10), 8 0 1 South turn_right ((2, 2), (3, 3), (4, 10), 9 0 1 West forward ((2, 2), (3, 3), (4, 10), 10 0 2 North turn_right ((2, 2), (3, 3), (4, 10), 11 0 2 East turn_right ((2, 2), (3, 3), (4, 10), 12 0 2 South forward ((2, 2), (3, 3), (4, 10), 13 0 2 West turn_left ((2, 2), (3, 3), (4, 10), 14 0 3 North turn_right ((2, 2), (3, 3), (4, 10), 15 0 3 East turn_left ((2, 2), (3, 3), (4, 10), 16 0 3 South turn_right ((2, 2), (3, 3), (4, 10), 17 0 3 West forward ((2, 2), (3, 3), (4, 10), 18 0 4 North turn_left ((2, 2), (3, 3), (4, 10), 19 0 4 East turn_right ((2, 2), (3, 3), (4, 10), 20 0 4 South forward ((2, 2), (3, 3), (4, 10), 21 0 4 West turn_left ((2, 2), (3, 3), (4, 10), 22 0 5 North turn_right ((2, 2), (3, 3), (4, 10), 23 0 5 East turn_left ((2, 2), (3, 3), (4, 10), 24 0 5 South turn_right ((2, 2), (3, 3), (4, 10), 25 0 5 West turn_left ((2, 2), (3, 3), (4, 10), 26 0 6 North turn_right ((2, 2), (3, 3), (4, 10), 27 0 6 East forward ((2, 2), (3, 3), (4, 10), 28 0 6 West turn_left ((2, 2), (3, 3), (4, 10), 29 0 7 North turn_right ((2, 2), (3, 3), (4, 10), 30 0 7 East turn_left ((2, 2), (3, 3), (4, 10), 31 0 7 West turn_right ((2, 2), (3, 3), (4, 10), 32 0 8 North turn_left ((2, 2), (3, 3), (4, 10), 33 0 8 East turn_right ((2, 2), (3, 3), (4, 10), 34 0 8 West turn_left ((2, 2), (3, 3), (4, 10), 35 0 9 North turn_right ((2, 2), (3, 3), (4, 10), 36 0 9 East turn_left ((2, 2), (3, 3), (4, 10), 37 0 9 West turn_right ((2, 2), (3, 3), (4, 10), 38 0 10 North turn_left ((2, 2), (3, 3), (4, 10), 39 0 10 East turn_right ((2, 2), (3, 3), (4, 10), 40 0 10 West turn_left ((2, 2), (3, 3), (4, 10), 41 0 11 North turn_right ((2, 2), (3, 3), (4, 10), 42 0 11 East turn_left ((2, 2), (3, 3), (4, 10), 43 0 11 West turn_right ((2, 2), (3, 3), (4, 10), 44 0 12 North turn_left ((2, 2), (3, 3), (4, 10), 45 0 12 East turn_right ((2, 2), (3, 3), (4, 10), 46 0 12 West turn_left ((2, 2), (3, 3), (4, 10), 47 0 13 North turn_right ((2, 2), (3, 3), (4, 10), 48 0 13 East turn_left ((2, 2), (3, 3), (4, 10), 49 0 13 West turn_right ((2, 2), (3, 3), (4, 10), 50 0 14 North turn_left ((2, 2), (3, 3), (4, 10), 51 0 14 East turn_right ((2, 2), (3, 3), (4, 10), 52 0 14 West turn_left ((2, 2), (3, 3), (4, 10), 53 0 15 North turn_right ((2, 2), (3, 3), (4, 10), 54 0 15 East turn_left ((2, 2), (3, 3), (4, 10), 55 0 15 West turn_right ((2, 2), (3, 3), (4, 10), 56 0 16 North turn_left ((2, 2), (3, 3), (4, 10), 57 0 16 East turn_right ((2, 2), (3, 3), (4, 10), 58 0 16 West turn_left ((2, 2), (3, 3), (4, 10), 59 0 17 North turn_right ((2, 2), (3, 3), (4, 10), 60 0 17 East turn_left ((2, 2), (3, 3), (4, 10), 61 0 17 West turn_right ((2, 2), (3, 3), (4, 10), 62 0 18 North turn_left ((2, 2), (3, 3), (4, 10), 63 0 18 East turn_right ((2, 2), (3, 3), (4, 10), 64 0 18 West turn_left ((2, 2), (3, 3), (4, 10), 65 0 19 North turn_right ((2, 2), (3, 3), (4, 10), 66 0 19 East turn_left ((2, 2), (3, 3), (4, 10), 67 0 19 West turn_right ((2, 2), (3, 3), (4, 10), 68 0 20 North turn_left ((2, 2), (3, 3), (4, 10), 69 0 20 East turn_right ((2, 2), (3, 3), (4, 10), 70 0 20 West turn_left ((2, 2), (3, 3), (4, 10), 71 0 21 North turn_right ((2, 2), (3, 3), (4, 10), 72 0 21 East turn_left ((2, 2), (3, 3), (4, 10), 73 0 21 West turn_right ((2, 2), (3, 3), (4, 10), 74 0 22 North turn_left ((2, 2), (3, 3), (4, 10), 75 0 22 East turn_right ((2, 2), (3, 3), (4, 10), 76 0 22 West turn_left ((2, 2), (3, 3), (4, 10), 77 0 23 North turn_right ((2, 2), (3, 3), (4, 10), 78 0 23 East turn_left ((2, 2), (3, 3), (4, 10), 79 0 23 West turn_right ((2, 2), (3, 3), (4, 10), 80 0 24 North turn_left ((2, 2), (3, 3), (4, 10), 81 0 24 East turn_right ((2, 2), (3, 3), (4, 10), 82 0 24 West turn_left ((2, 2), (3, 3), (4, 10), 83 0 25 North turn_right ((2, 2), (3, 3), (4, 10), 84 0 25 East turn_left ((2, 2), (3, 3), (4, 10), 85 0 25 West turn_right ((2, 2), (3, 3), (4, 10), 86 0 26 North turn_left ((2, 2), (3, 3), (4, 10), 87 0 26 East turn_right ((2, 2), (3, 3), (4, 10), 88 0 26 West turn_left ((2, 2), (3, 3), (4, 10), 89 0 27 North turn_right ((2, 2), (3, 3), (4, 10), 90 0 27 East turn_left ((2, 2), (3, 3), (4, 10), 91 0 27 West turn_right ((2, 2), (3, 3), (4, 10), 92 0 28 North turn_left ((2, 2), (3, 3), (4, 10), 93 0 28 East turn_right ((2, 2), (3, 3), (4, 10), 94 0 28 West turn_left ((2, 2), (3, 3), (4, 10), 95 0 29 North turn_right ((2, 2), (3, 3), (4, 10), 96 0 29 East turn_left ((2, 2), (3, 3), (4, 10), 97 0 29 West turn_right ((2, 2), (3, 3), (4, 10), 98 0 30 North turn_left ((2, 2), (3, 3), (4, 10), 99 0 30 East turn_right ((2, 2), (3, 3), (4, 10), 100 0 30 West turn_left ((2, 2), (3, 3), (4, 10), 101 0 31 North turn_right ((2, 2), (3, 3), (4, 10), 102 0 31 East turn_left ((2, 2), (3, 3), (4, 10), 103 0 31 West turn_right ((2, 2), (3, 3), (4, 10), 104 0 32 North turn_left ((2, 2), (3, 3), (4, 10), 105 0 32 East turn_right ((2, 2), (3, 3), (4, 10), 106 0 32 West turn_left ((2, 2), (3, 3), (4, 10), 107 0 33 North turn_right ((2, 2), (3, 3), (4, 10), 108 0 33 East turn_left ((2, 2), (3, 3), (4, 10), 109 0 33 West turn_right ((2, 2), (3, 3), (4, 10), 110 0 34 North turn_left ((2, 2), (3, 3), (4, 10), 111 0 34 East turn_right ((2, 2), (3, 3), (4, 10), 112 0 34 West turn_left ((2, 2), (3, 3), (4, 10), 113 0 35 North turn_right ((2, 2), (3, 3), (4, 10), 114 0 35 East turn_left ((2, 2), (3, 3), (4, 10), 115 0 35 West turn_right ((2, 2), (3, 3), (4, 10), 116 0 36 North turn_left ((2, 2), (3, 3), (4, 10), 117 0 36 East turn_right ((2, 2), (3, 3), (4, 10), 118 0 36 West turn_left ((2, 2), (3, 3), (4, 10), 119 0 37 North turn_right ((2, 2), (3, 3), (4, 10), 120 0 37 East turn_left ((2, 2), (3, 3), (4, 10), 121 0 37 West turn_right ((2, 2), (3, 3), (4, 10), 122 0 38 North turn_left ((2, 2), (3, 3), (4, 10), 123 0 38 East turn_right ((2, 2), (3, 3), (4, 10), 124 0 38 West turn_left ((2, 2), (3, 3), (4, 10), 125 0 39 North turn_right ((2, 2), (3, 3), (4, 10), 126 0 39 East turn_left ((2, 2), (3, 3), (4, 10), 127 0 39 West turn_right ((2, 2), (3, 3), (4, 10), 128 0 40 North turn_left ((2, 2), (3, 3), (4, 10), 129 0 40 East turn_right ((2, 2), (3, 3), (4, 10), 130 0 40 West turn_left ((2, 2), (3, 3), (4, 10), 131 0 41 North turn_right ((2, 2), (3, 3), (4, 10), 132 0 41 East turn_left ((2, 2), (3, 3), (4, 10), 133 0 41 West turn_right ((2, 2), (3, 3), (4, 10), 134 0 42 North turn_left ((2, 2), (3, 3), (4, 10), 135 0 42 East turn_right ((2, 2), (3, 3), (4, 10), 136 0 42 West turn_left ((2, 2), (3, 3), (4, 10), 137 0 43 North turn_right ((2, 2), (3, 3), (4, 10), 138 0 43 East turn_left ((2, 2), (3, 3), (4, 10), 139 0 43 West turn_right ((2, 2), (3, 3), (4, 10), 140 0 44 North turn_left ((2, 2), (3, 3), (4, 10), 141 0 44 East turn_right ((2, 2), (3, 3), (4, 10), 142 0 44 West turn_left ((2, 2), (3, 3), (4, 10), 143 0 45 North turn_right ((2, 2), (3, 3), (4, 10), 144 0 45 East turn_left ((2, 2), (3, 3), (4, 10), 145 0 45 West turn_right ((2, 2), (3, 3), (4, 10), 146 0 46 North turn_left ((2, 2), (3, 3), (4, 10), 147 0 46 East turn_right ((2, 2), (3, 3), (4, 10), 148 0 46 West turn_left ((2, 2), (3, 3), (4, 10), 149 0 47 North turn_right ((2, 2), (3, 3), (4, 10), 150 0 47 East turn_left ((2, 2), (3, 3), (4, 10), 151 0 47 West turn_right ((2, 2), (3, 3), (4, 10), 152 0 48 North turn_left ((2, 2), (3, 3), (4, 10), 153 0 48 East turn_right ((2, 2), (3, 3), (4, 10), 154 0 48 West turn_left ((2, 2), (3, 3), (4, 10), 155 0 49 North turn_right ((2, 2), (3, 3), (4, 10), 156 0 49 East turn_left ((2, 2), (3, 3), (4, 10), 157 0 49 West turn_right ((2, 2), (3, 3), (4, 10), 158 0 50 North turn_left ((2, 2), (3, 3), (4, 10), 159 0 50 East turn_right ((2, 2), (3, 3), (4, 10), 160 0 50 West turn_left ((2, 2), (3, 3), (4, 10), 161 0 51 North turn_right ((2, 2), (3, 3), (4, 10), 162 0 51 East turn_left ((2, 2), (3, 3), (4, 10), 163 0 51 West turn_right ((2, 2), (3, 3), (4, 10), 164 0 52 North turn_left ((2, 2), (3, 3), (4, 10), 165 0 52 East turn_right ((2, 2), (3, 3), (4, 10), 166 0 52 West turn_left ((2, 2), (3, 3), (4, 10), 167 0 53 North turn_right ((2, 2), (3, 3), (4, 10), 168 0 53 East turn_left ((2, 2), (3, 3), (4, 10), 169 0 53 West turn_right ((2, 2), (3, 3), (4, 10), 170 0 54 North turn_left ((2, 2), (3, 3), (4, 10), 171 0 54 East turn_right ((2, 2), (3, 3), (4, 10), 172 0 54 West turn_left ((2, 2), (3, 3), (4, 10), 173 0 55 North turn_right ((2, 2), (3, 3), (4, 10), 174 0 55 East turn_left ((2, 2), (3, 3), (4, 10), 175 0 55 West turn_right ((2, 2), (3, 3), (4, 10), 176 0 56 North turn_left ((2, 2), (3, 3), (4, 10), 177 0 56 East turn_right ((2, 2), (3, 3), (4, 10), 178 0 56 West turn_left ((2, 2), (3, 3), (4, 10), 179 0 57 North turn_right ((2, 2), (3, 3), (4, 10), 180 0 57 East turn_left ((2, 2), (3, 3), (4, 10), 181 0 57 West turn_right ((2, 2), (3, 3), (4, 10), 182 0 58 North turn_left ((2, 2), (3, 3), (4, 10), 183 0 58 East turn_right ((2, 2), (3, 3), (4, 10), 184 0 58 West turn_left ((2, 2), (3, 3), (4, 10), 185 0 59 North turn_right ((2, 2), (3, 3), (4, 10), 186 0 59 East turn_left ((2, 2), (3, 3), (4, 10), 187 0 59 West turn_right ((2, 2), (3, 3), (4, 10), 188 0 60 North turn_left ((2, 2), (3, 3), (4, 10), 189 0 60 East turn_right ((2, 2), (3, 3), (4, 10), 190 0 60 West turn_left ((2, 2), (3, 3), (4, 10), 191 0 61 North turn_right ((2, 2), (3, 3), (4, 10), 192 0 61 East turn_left ((2, 2), (3, 3), (4, 10), 193 0 61 West turn_right ((2, 2), (3, 3), (4, 10), 194 0 62 North turn_left ((2, 2), (3, 3), (4, 10), 195 0 62 East turn_right ((2, 2), (3, 3), (4, 10), 196 0 62 West turn_left ((2, 2), (3, 3), (4, 10), 197 0 63 North turn_right ((2, 2), (3, 3), (4, 10), 198 0 63 East turn_left ((2, 2), (3, 3), (4, 10), 199 0 63 West turn_right ((2, 2), (3, 3), (4, 10), 200 0 64 North turn_left ((2, 2), (3, 3), (4, 10), 201 0 64 East turn_right ((2, 2), (3, 3), (4, 10), 202 0 64 West turn_left ((2, 2), (3, 3), (4, 10), 203 0 65 North turn_right ((2, 2), (3, 3), (4, 10), 204 0 65 East turn_left ((2, 2), (3, 3), (4, 10), 205 0 65 West turn_right ((2, 2), (3, 3), (4, 10), 206 0 66 North turn_left ((2, 2), (3, 3), (4, 10), 207 0 66 East turn_right ((2, 2), (3, 3), (4, 10), 208 0 66 West turn_left ((2, 2), (3, 3), (4, 10), 209 0 67 North turn_right ((2, 2), (3, 3), (4, 10), 210 0 67 East turn_left ((2, 2), (3, 3), (4, 10), 211 0 67 West turn_right ((2, 2), (3, 3), (4, 10), 212 0 68 North turn_left ((2, 2), (3, 3), (4, 10), 213 0 68 East turn_right ((2, 2), (3, 3), (4, 10), 214 0 68 West turn_left ((2, 2), (3, 3), (4, 10), 215 0 69 North turn_right ((2, 2), (3, 3), (4, 10), 216 0 69 East turn_left ((2, 2), (3, 3), (4, 10), 217 0 69 West turn_right ((2, 2), (3, 3), (4, 10), 218 0 70 North turn_left ((2, 2), (3, 3), (4, 10), 219 0 70 East turn_right ((2, 2), (3, 3), (4, 10), 220 0 70 West turn_left ((2, 2), (3, 3), (4, 10), 221 0 71 North turn_right ((2, 2), (3, 3), (4, 10), 222 0 71 East turn_left ((2, 2), (3, 3), (4, 10), 223 0 71 West turn_right ((2, 2), (3, 3), (4, 10), 224 0 72 North turn_left ((2, 2), (3, 3), (4, 10), 225 0 72 East turn_right ((2, 2), (3, 3), (4, 10), 226 0 72 West turn_left ((2, 2), (3, 3), (4, 10), 227 0 73 North turn_right ((2, 2), (3, 3), (4, 10), 228 0 73 East turn_left ((2, 2), (3, 3), (4, 10), 229 0 73 West turn_right ((2, 2), (3, 3), (4, 10), 230 0 74 North turn_left ((2, 2), (3, 3), (4, 10), 231 0 74 East turn_right ((2, 2), (3, 3), (4, 10), 232 0 74 West turn_left ((2, 2), (3, 3), (4, 10), 233 0 75 North turn_right ((2, 2), (3, 3), (4, 10), 234 0 75 East turn_left ((2, 2), (3, 3), (4, 10), 235 0 75 West turn_right ((2, 2), (3, 3), (4, 10), 236 0 76 North turn_left ((2, 2), (3, 3), (4, 10), 237 0 76 East turn_right ((2, 2), (3, 3), (4, 10), 238 0 76 West turn_left ((2, 2), (3, 3), (4, 10), 239 0 77 North turn_right ((2, 2), (3, 3), (4, 10), 240 0 77 East turn_left ((2, 2), (3, 3), (4, 10), 241 0 77 West turn_right ((2, 2), (3, 3), (4, 10), 242 0 78 North turn_left ((2, 2), (3, 3), (4, 10), 243 0 78 East turn_right ((2, 2), (3, 3), (4, 10), 244 0 78 West turn_left ((2, 2), (3, 3), (4, 10), 245 0 79 North turn_right ((2, 2), (3, 3), (4, 10), 246 0 79 East turn_left ((2, 2), (3, 3), (4, 10), 247 0 79 West turn_right ((2, 2), (3, 3), (4, 10), 248 0 80 North turn_left ((2, 2), (3, 3), (4, 10), 249 0 80 East turn_right ((2, 2), (3, 3), (4, 10), 250 0 80 West turn_left ((2, 2), (3, 3), (4, 10), 251 0 81 North turn_right ((2, 2), (3, 3), (4, 10), 252 0 81 East turn_left ((2, 2), (3, 3), (4, 10), 253 0 81 West turn_right ((2, 2), (3, 3), (4, 10), 254 0 82 North turn_left ((2, 2), (3, 3), (4, 10), 255 0 82 East turn_right ((2, 2), (3, 3), (4, 10), 256 0 82 West turn_left ((2, 2), (3, 3), (4, 10), 257 0 83 North turn_right ((2, 2), (3, 3), (4, 10), 258 0 83 East turn_left ((2, 2), (3, 3), (4, 10), 259 0 83 West turn_right ((2, 2), (3, 3), (4, 10), 260 0 84 North turn_left ((2, 2), (3, 3), (4, 10), 261 0 84 East turn_right ((2, 2), (3, 3), (4, 10), 262 0 84 West turn_left ((2, 2), (3, 3), (4, 10), 263 0 85 North turn_right ((2, 2), (3, 3), (4, 10), 264 0 85 East turn_left ((2, 2), (3, 3), (4, 10), 265 0 85 West turn_right ((2, 2), (3, 3), (4, 10), 266 0 86 North turn_left ((2, 2), (3, 3), (4, 10), 267 0 86 East turn_right ((2, 2), (3, 3), (4, 10), 268 0 86 West turn_left ((2, 2), (3, 3), (4, 10), 269 0 87 North turn_right ((2, 2), (3, 3), (4, 10), 270 0 87 East turn_left ((2, 2), (3, 3), (4, 10), 271 0 87 West turn_right ((2, 2), (3, 3), (4, 10), 272 0 88 North turn_left ((2, 2), (3, 3), (4, 10), 273 0 88 East turn_right ((2, 2), (3, 3), (4, 10), 274 0 88 West turn_left ((2, 2), (3, 3), (4, 10), 275 0 89 North turn_right ((2, 2), (3, 3), (4, 10), 276 0 89 East turn_left ((2, 2), (3, 3), (4, 10), 277 0 89 West turn_right ((2, 2), (3, 3), (4, 10), 278 0 90 North turn_left ((2, 2), (3, 3), (4, 10), 279 0 90 East turn_right ((2, 2), (3, 3), (4, 10), 280 0 90 West turn_left ((2, 2), (3, 3), (4, 10), 281 0 91 North turn_right ((2, 2), (3, 3), (4, 10), 282 0 91 East turn_left ((2, 2), (3, 3), (4, 10), 283 0 91 West turn_right ((2, 2), (3, 3), (4, 10), 284 0 92 North turn_left ((2, 2), (3, 3), (4, 10), 285 0 92 East turn_right ((2, 2), (3, 3), (4, 10), 286 0 92 West turn_left ((2, 2), (3, 3), (4, 10), 287 0 93 North turn_right ((2, 2), (3, 3), (4, 10), 288 0 93 East turn_left ((2, 2), (3, 3), (4, 10), 289 0 93 West turn_right ((2, 2), (3, 3), (4, 10), 290 0 94 North turn_left ((2, 2), (3, 3), (4, 10), 291 0 94 East turn_right ((2, 2), (3, 3), (4, 10), 292 0 94 West turn_left ((2, 2), (3, 3), (4, 10), 293 0 95 North turn_right ((2, 2), (3, 3), (4, 10), 294 0 95 East turn_left ((2, 2), (3, 3), (4, 10), 295 0 95 West turn_right ((2, 2), (3, 3), (4, 10), 296 0 96 North turn_left ((2, 2), (3, 3), (4, 10), 297 0 96 East turn_right ((2, 2), (3, 3), (4, 10), 298 0 96 West turn_left ((2, 2), (3, 3), (4, 10), 299 0 97 North turn_right ((2, 2), (3, 3), (4, 10), 300 0 97 East turn_left ((2, 2), (3, 3), (4, 10), 301 0 97 West turn_right ((2, 2), (3, 3), (4, 10), 302 0 98 North turn_left ((2, 2), (3, 3), (4, 10), 303 0 98 East turn_right ((2, 2), (3, 3), (4, 10), 304 0 98 West turn_left ((2, 2), (3, 3), (4, 10), 305 0 99 North turn_right ((2, 2), (3, 3), (4, 10), 306 0 99 East turn_left ((2, 2), (3, 3), (4, 10), 307 0 99 West turn_right ((2, 2), (3, 3), (4, 10), 308 0 100 North turn_left ((2, 2), (3, 3), (4, 10), 309 0 100 East turn_right ((2, 2), (3, 3), (4, 10), 310 0 100 West turn_left ((2, 2), (3, 3), (4, 10), 311 0 101 North turn_right ((2, 2), (3, 3), (4, 10), 312 0 101 East turn_left ((2, 2), (3, 3), (4, 10), 313 0 101 West turn_right ((2, 2), (3, 3), (4, 10), 314 0 102 North turn_left ((2, 2), (3, 3), (4, 10), 315 0 102 East turn_right ((2, 2), (3, 3), (4, 10), 316 0 102 West turn_left ((2, 2), (3, 3), (4, 10), 317 0 103 North turn_right ((2, 2), (3, 3), (4, 10), 318 0 103 East turn_left ((2, 2), (3, 3), (4, 10), 319 0 103 West turn_right ((2, 2), (3, 3), (4, 10), 320 0 104 North turn_left ((2, 2), (3, 3), (4, 10), 321 0 104 East turn_right ((2, 2), (3, 3), (4, 10), 322 0 104 West turn_left ((2, 2), (3, 3), (4, 10), 323 0 105 North turn_right ((2, 2), (3, 3), (4, 10), 324 0 105 East turn_left ((2, 2), (3, 3), (4, 10), 325 0 105 West turn_right ((2, 2), (3, 3), (4, 10), 326 0 106 North turn_left ((2, 2), (3, 3), (4, 10), 327 0 106 East turn_right ((2, 2), (3, 3), (4, 10), 328 0 106 West turn_left ((2, 2), (3, 3), (4, 10), 329 0 107 North turn_right ((2, 2), (3, 3), (4, 10), 330 0 107 East turn_left ((2, 2), (3, 3), (4, 10), 331 0 107 West</pre>						

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

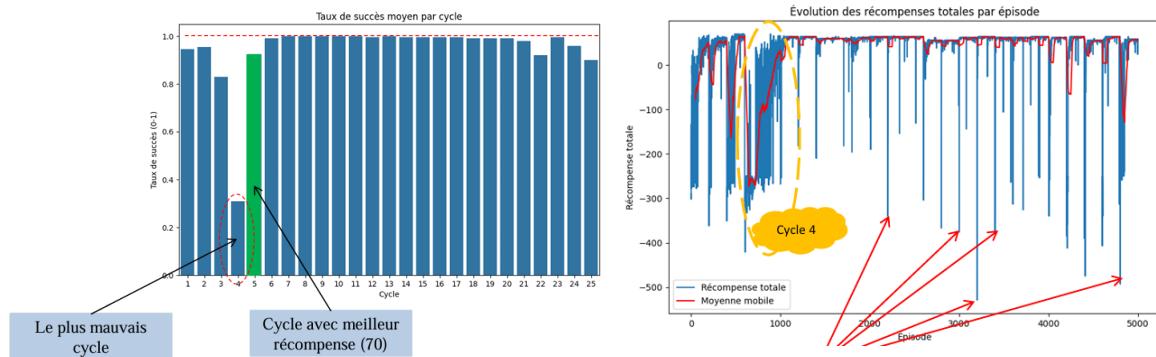


FIGURE 4.7 – Taux de succès d'une simulation

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

episode	cycle	path	total_reward	step	reached_goal	obstacle_config
790	4	(0,0,East)(0,1,East)	48	53	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)	-40	61	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)(0,0,South)	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	88	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,South)	8	93	VRAI	((1, 8), (1, 11), (4, 14))
772	4	(0,0,East)(0,0,South)	95	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))
693	4	(0,0,East)(0,0,North)	-10	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))
793	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))
795	4	(0,0,East)(0,1,East)	-22	123	VRAI	((1, 8), (1, 11), (4, 14))
799	4	(0,0,East)(0,0,North)	-22	123	VRAI	((1, 8), (1, 11), (4, 14))
759	4	(0,0,East)(0,0,North)	-24	125	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)(0,1,East)	-37	138	VRAI	((1, 8), (1, 11), (4, 14))
774	4	(0,0,East)(0,1,East)	-38	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))

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

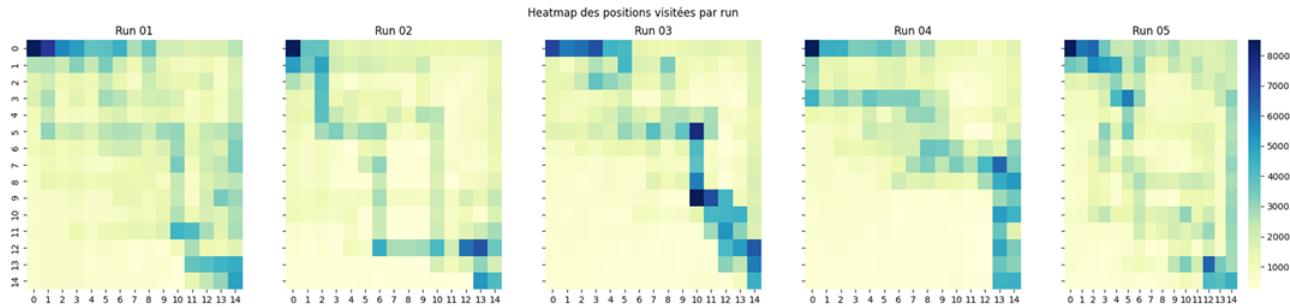


FIGURE 5.1 – Comparaison Heatmap - mêmes paramètres

Pour information, notre chemin préféré initié au système est de partir vers le coin haut-droite 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 NxN 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

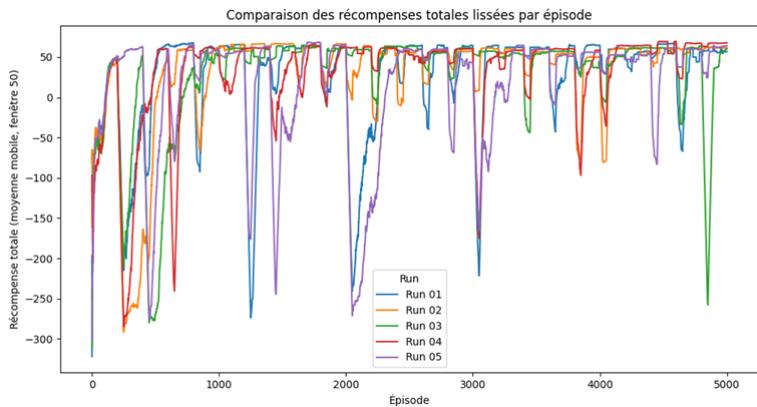


FIGURE 5.2 – Comparaison Récompenses par itération lissées - mêmes paramètres

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

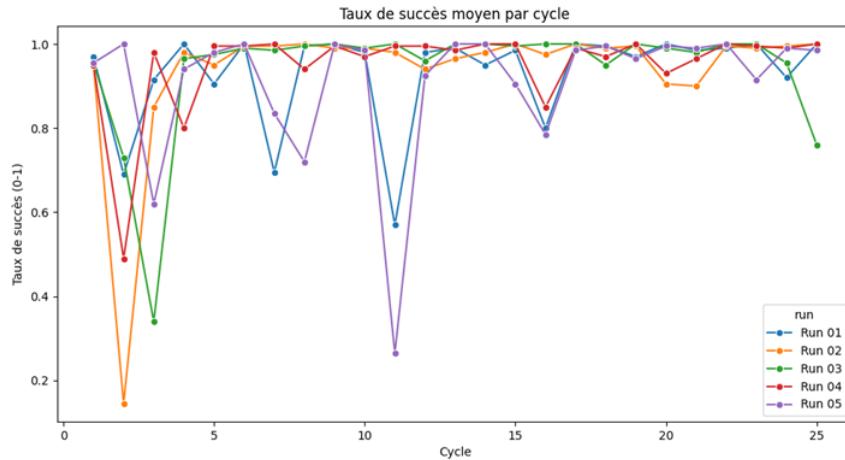


FIGURE 5.3 – Comparaison Taux de réussite - mêmes paramètres

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

### 5.4 Conclusion

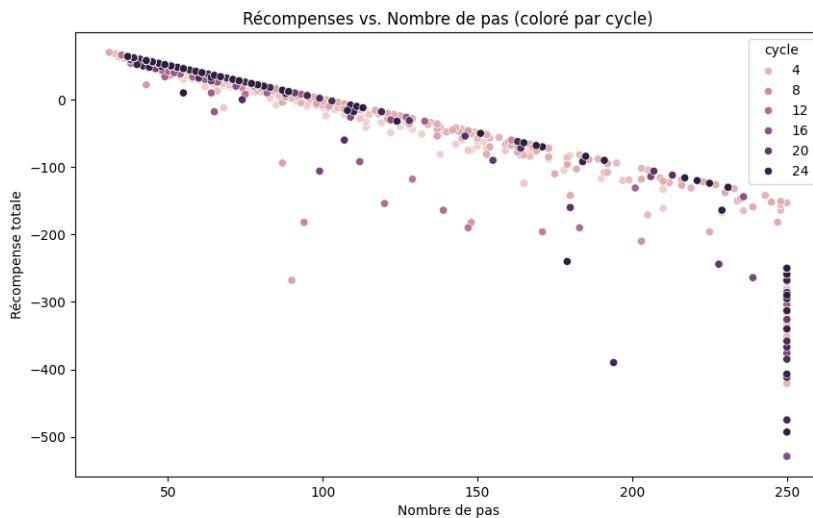


FIGURE 5.4 – Comparaison récompense/steps - une simulation

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/steps** 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 choisis pour la simulation : 0.3 - 0.4 - 0.5 - 0.7 - 0.9

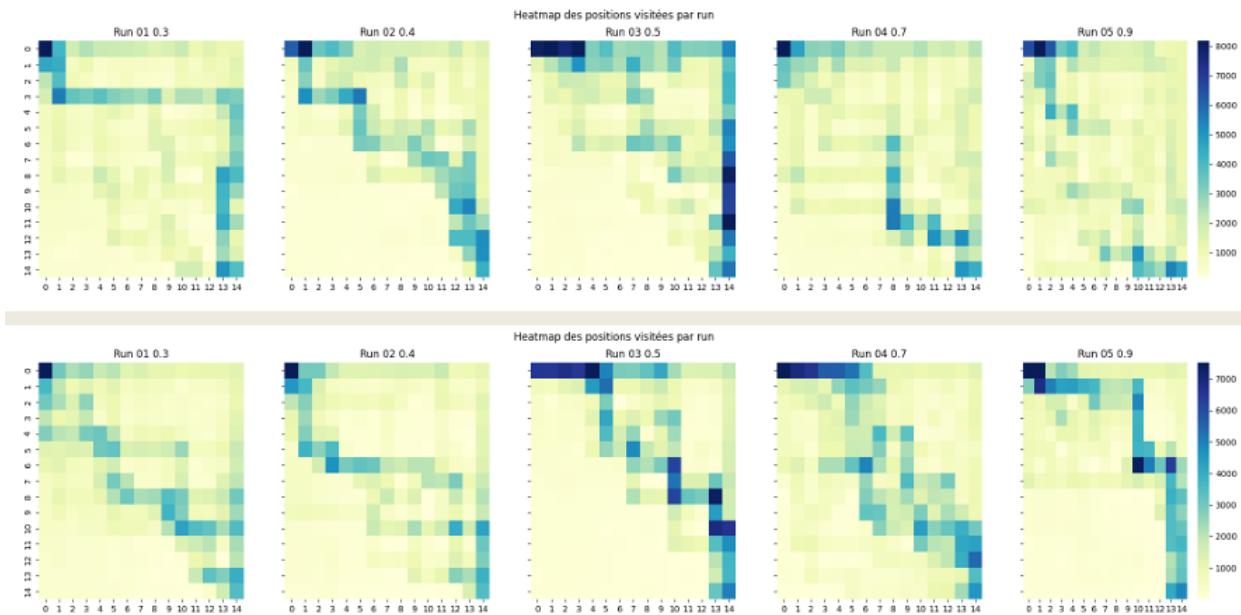


FIGURE 6.1 – Comparaison Learning\_rate - Heatmap

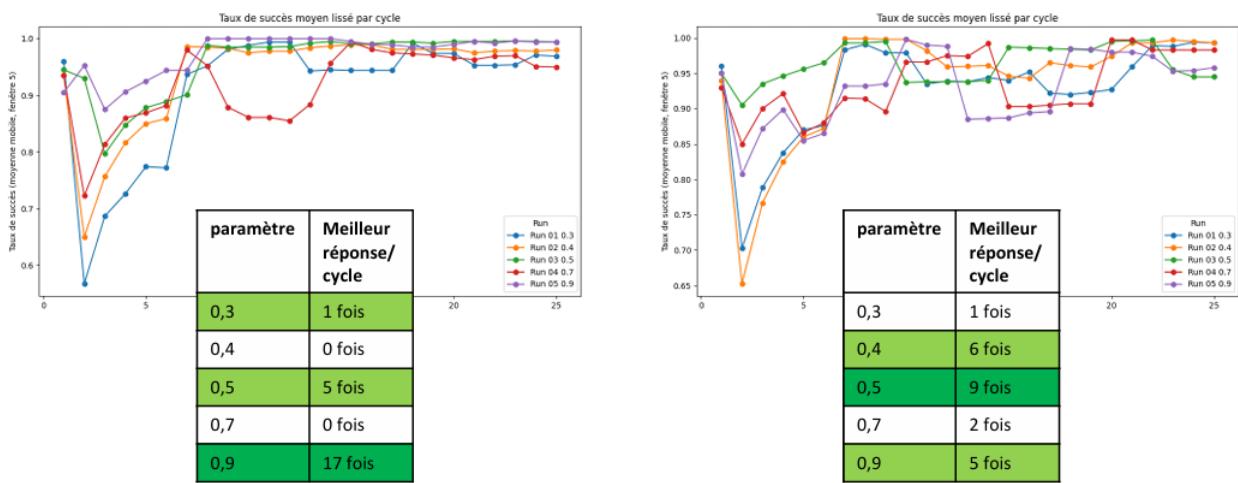


FIGURE 6.2 – Comparaison Learning\_rate - Taux de réussite

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

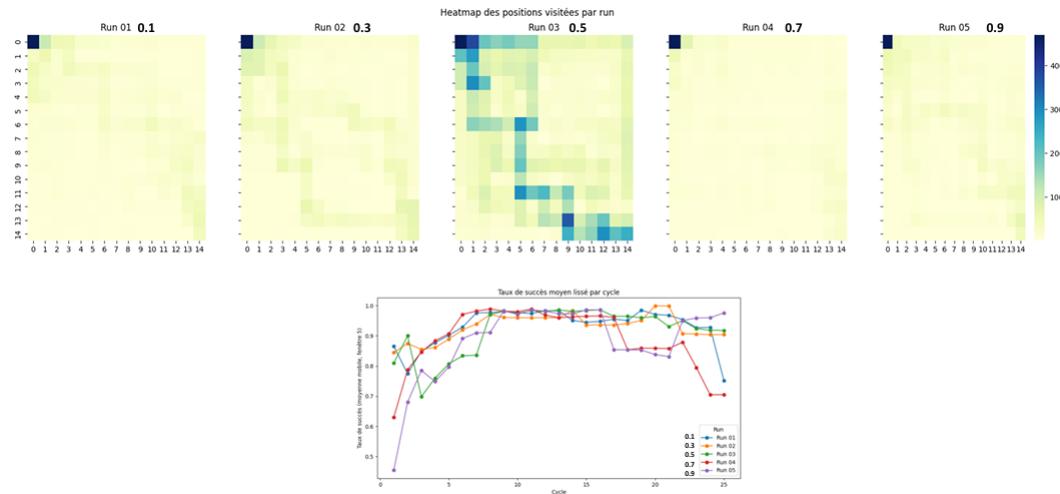


FIGURE 6.3 – Comparaison Epsilon - Vision globale

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

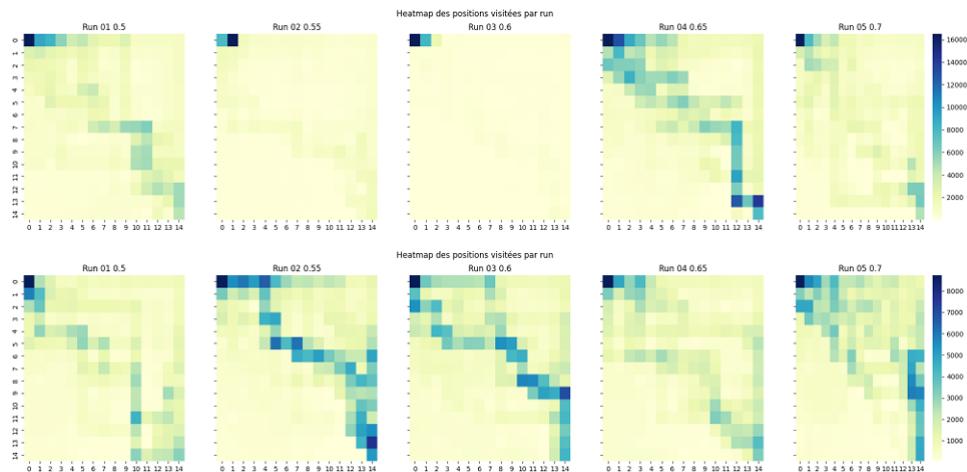


FIGURE 6.4 – Comparaison Epsilon - Heatmap

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

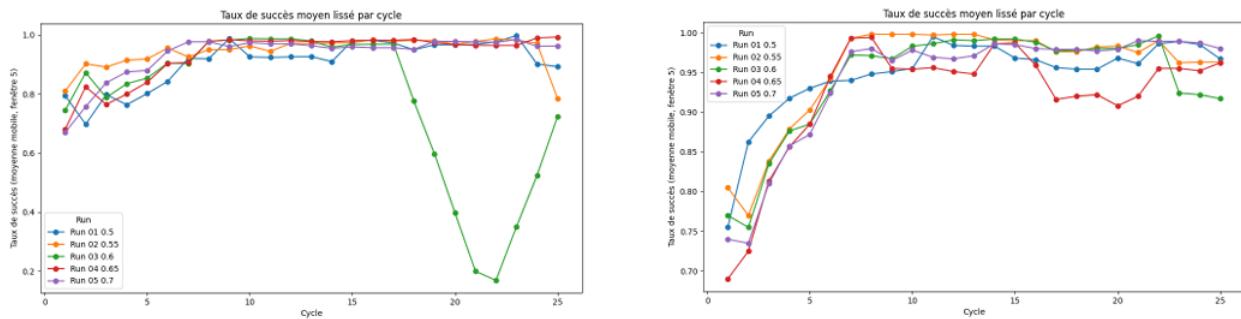


FIGURE 6.5 – Comparaison Epsilon - Taux de réussite

### 6.3 Discount\_factor

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

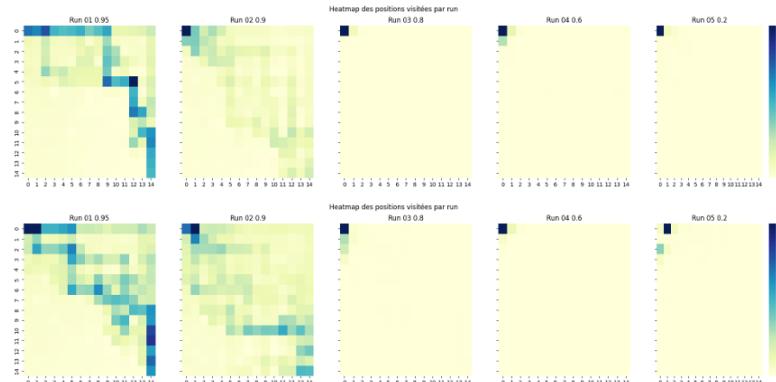


FIGURE 6.6 – Comparaison Discount\_factor - Heatmap

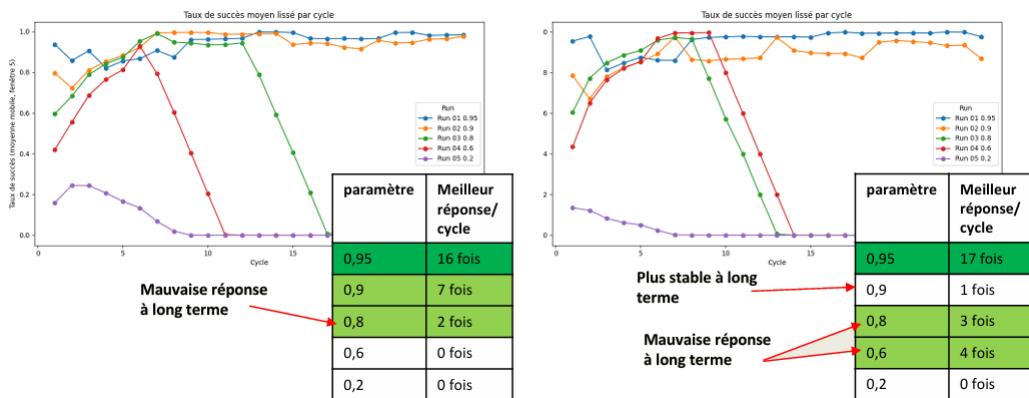


FIGURE 6.7 – Comparaison Discount\_factor - Taux de réussite

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

A 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.

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
```

<pre> 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 &gt;&gt; GOAL +100 &lt;&lt; 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 &gt;&gt; OBSTACLE en (2,4) </pre>	<pre> 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 == </pre>
---	---

On définit tout d'abord un chemin préféré au système comme ci-contre :

Ce chemin est déjà prédéfinie au système, mais avec la politique  $\epsilon$ -cupide, le robot prend des

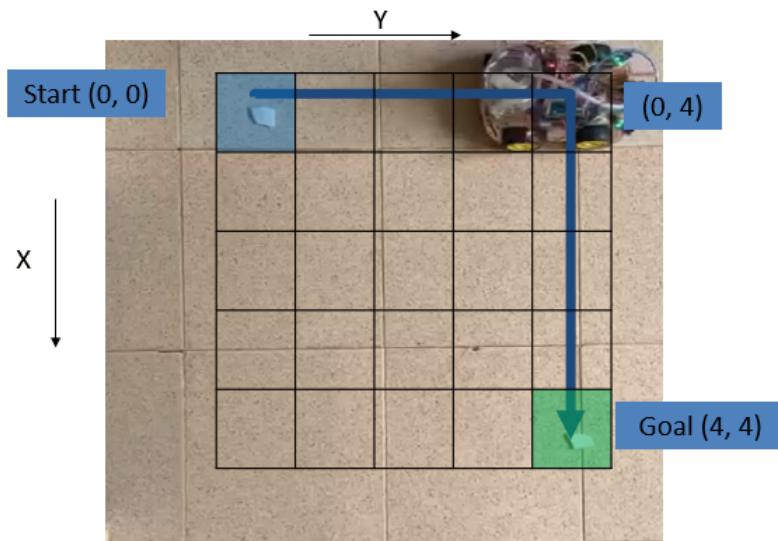


FIGURE 7.1 – Chemin préféré - Simulation ARDUINO

décisions aléatoires avec la probabilité  $\epsilon$  pour explorer de nouvelles solutions plus optimales.

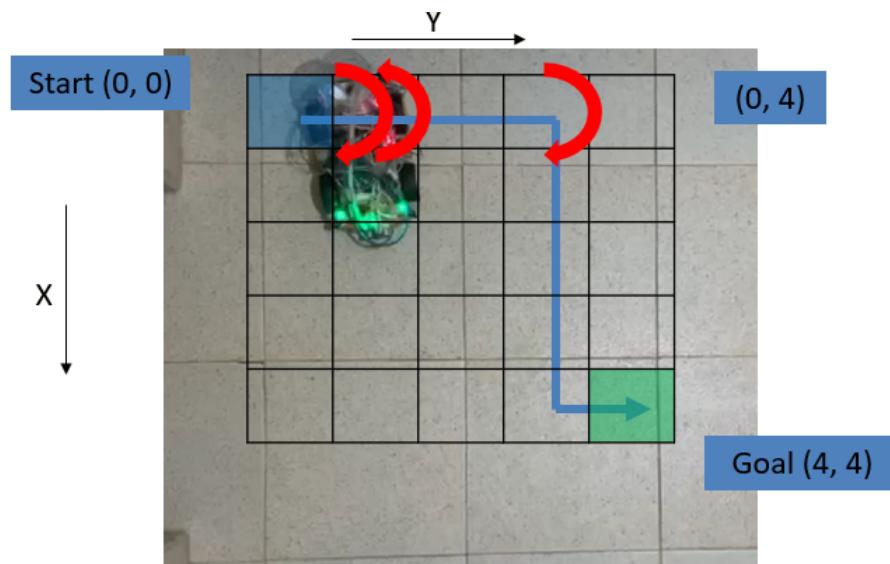


FIGURE 7.2 – Test 1 - Simulation ARDUINO

Parfois cette exploration donne des chemins plus optimales à fur et à mesure exemple Test1.

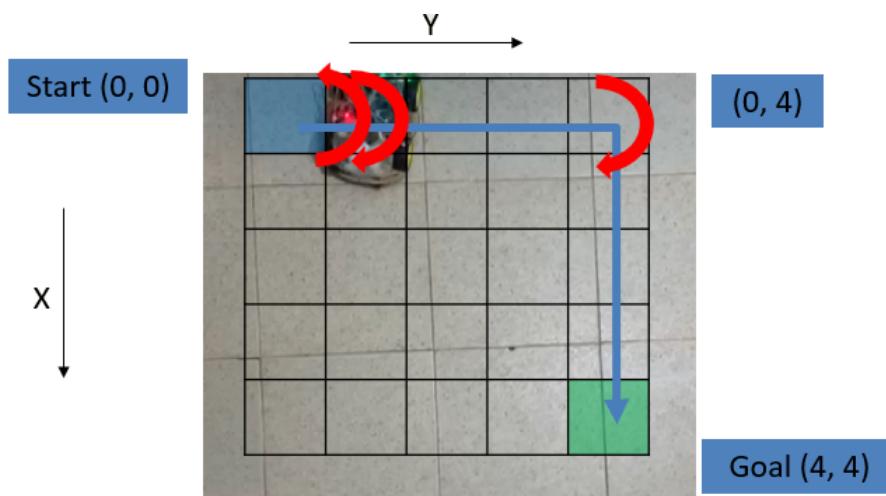


FIGURE 7.3 – Test 2 - Simulation ARDUINO

Sinon parfois le système reste coincé sans solution ou bien épuise le maximum des mouvements possible pendant un cycle :

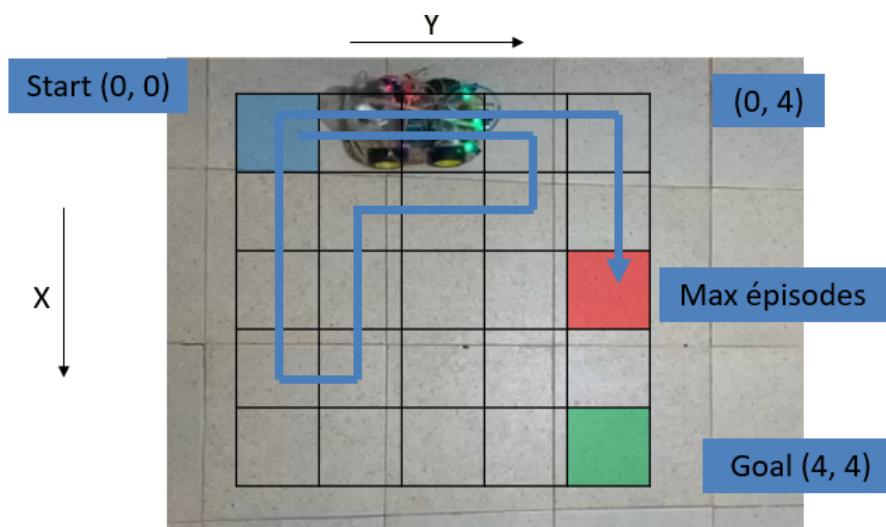


FIGURE 7.4 – Test 3 - Simulation ARDUINO

# 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
51 class Environment:
52     def __init__(self):
53         self.start_pos = (0, 0)
54         self.start_facing = 1 # East
55         self.goal = (grid_size - 1, grid_size - 1)
56         self.true_obstacles = set()
57         self.known_obstacles = set()
58         self.position = self.start_pos
59         self.facing = self.start_facing
60
61     def reset(self, keep_known=False):
62         self.position = self.start_pos
63         self.facing = self.start_facing
64         if not keep_known:
65             self.known_obstacles = set()
66         self.reveal_neighborhood()
67         return (self.position[0], self.position[1], self.facing)
68
69     def add_random_obstacles(self, num):
70         self.true_obstacles = set()
71         attempts = 0
72         while len(self.true_obstacles) < num and attempts < num * 30:
73             attempts += 1
74             x = random.randint(0, grid_size - 1)
75             y = random.randint(0, grid_size - 1)
76             coord = (x, y)
77             if coord == self.start_pos or coord == self.goal:
78                 continue
79             self.true_obstacles.add(coord)
80         while not self.is_path_possible():
81             if not self.true_obstacles:
82                 break
83             self.true_obstacles.pop()
84
85     def is_path_possible(self):
86         visited = set()
87         queue = deque([(self.start_pos, self.start_facing)])
88         visited.add((self.start_pos[0], self.start_pos[1], self.start_facing))
89         while queue:
90             (x, y), f = queue.popleft()
91             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
96     ↪ self.true_obstacles:
97         if (nx, ny, f) not in visited:
98             visited.add((nx, ny, f))
99             queue.append(((nx, ny), f))
100    # Try turn_left
101   nf = (f - 1) % 4
102   if (x, y, nf) not in visited:
103       visited.add((x, y, nf))
104       queue.append(((x, y), nf))
105    # Try turn_right
106   nf = (f + 1) % 4
107   if (x, y, nf) not in visited:
108       visited.add((x, y, nf))
109       queue.append(((x, y), nf))
110   return False
111
112 def reveal_neighborhood(self):
113     x, y = self.position
114     dx, dy = deltas[self.facing]
115     nx, ny = x + dx, y + dy
116     if 0 <= nx < grid_size and 0 <= ny < grid_size:
117         coord = (nx, ny)
118         if coord in self.true_obstacles:
119             self.known_obstacles.add(coord)
120
121 def step(self, action):
122     done = False
123     if action == 1: # turn_left
124         self.facing = (self.facing - 1) % 4
125         reward = -1
126         self.reveal_neighborhood()
127         return (self.position[0], self.position[1], self.facing), reward, done
128     elif action == 2: # turn_right
129         self.facing = (self.facing + 1) % 4
130         reward = -1
131         self.reveal_neighborhood()
132         return (self.position[0], self.position[1], self.facing), reward, done
133
134     # action == 0: forward
135     dx, dy = deltas[self.facing]
136     tx = max(0, min(self.position[0] + dx, grid_size - 1))
137     ty = max(0, min(self.position[1] + dy, grid_size - 1))
138     target = (tx, ty)
139
140     if target == self.position: # hit wall
141         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))
173     for ox, oy in env.known_obstacles:
174         pygame.draw.rect(screen, PINK, (oy * cell_size + 6, ox * cell_size + 6,
175                                     cell_size - 12, cell_size - 12))
176
177     pygame.draw.rect(screen, BLUE, (env.start_pos[1] * cell_size, env.start_pos[0] *
178                                     cell_size, cell_size, cell_size))
179     pygame.draw.rect(screen, GREEN, (env.goal[1] * cell_size, env.goal[0] *
180                                     cell_size, cell_size, cell_size))
181
182     font = pygame.font.SysFont(None, 22)
183     txt = font.render(f"Episode: {episode}/{num_episodes} Cycle: {cycle} Known:
184     ↪ {len(env.known_obstacles)}", True, BLACK)
185     screen.blit(txt, (8, 6))
186
187     def get_learned_path(env, Q, max_steps=max_steps_per_episode):
188         sim_env = Environment()
189         sim_env.start_pos = env.start_pos
190         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 *
218         ↪ radius), 3)
219         txt = font.render(f"Step: {index}/{len(path)-1}", True, BLACK)
220         screen.blit(txt, (10, 30))
221         pygame.display.flip()
222         clock.tick(tick_time)
223         index += 1
224         time.sleep(0.3)
225     return True
226
227 def print_policy_to_console(Q, env):
228     for f in range(4):
229         print(f"\nPolicy facing {dir_names[f]} (S=start, G=goal, X=obst,
230         ↪ F/L/R=actions):")
231         for x in range(grid_size):
232             row = []
233             for y in range(grid_size):
234                 pos = (x, y)
235                 if pos == env.start_pos:
236                     row.append('S ')
237                 elif pos == env.goal:
238
```

```

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):
338

```

```

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":
342                 str(config_key)
343             })
344             df = pd.DataFrame(rows)
345             sheet_name = f"Config_{idx}_{len(config_key)}jobs"
346             df.to_excel(writer, sheet_name=sheet_name[:31], index=False)
347             print(f"Policies appended to '{filename}'.")

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,
355     ↪ screen_size))
356     if not no_render:
357         pygame.display.set_caption('Q-Learning Partial Observability - Car-like
358         ↪ Agent')
359
360     env = Environment()
361     Q = np.zeros((grid_size, grid_size, 4, 3)) # Updated for 3 actions
362     policy_db_local = {}
363     episode_data = [] # To store paths and rewards
364     cycle = 0
365
366     env.true_obstacles = set()
367     env.reset(keep_known=False)
368     if preferred_path:
369         print(f"Applying preferred path for empty map: {preferred_path}")
370         initialize_q_with_preferred_path(Q, preferred_path, env)
371
372     for episode in range(num_episodes):
373         if not no_render:
374             for evt in pygame.event.get():
375                 if evt.type == pygame.QUIT:
376                     pygame.quit()
377                     sys.exit()
378
379         if episode % obstacle_change_interval == 0:
380             cycle += 1
381             if episode == 0:
382                 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
385             ↪ ({len(env.true_obstacles)} obstacles.)")
386
387     state = env.reset(keep_known=False)
388     done = False
389     steps = 0
390     total_reward = 0
391     path = [state]
392
393     while not done and steps < max_steps_per_episode:
394         steps += 1
395         x, y, f = state
396         if random.random() < epsilon:
397             action = random.randint(0, 2) # Updated for 3 actions
398         else:
399             action = int(np.argmax(Q[x, y, f]))
400
401         new_state, reward, done = env.step(action)
402         total_reward += reward
403         path.append(new_state)
404         nx, ny, nf = new_state
405         best_next = int(np.argmax(Q[nx, ny, nf]))
406         td_target = reward + discount_factor * Q[nx, ny, nf, best_next]
407         td_error = td_target - Q[x, y, f, action]
408         Q[x, y, f, action] += learning_rate * td_error
409         state = new_state
410
411     if not done:
412         last_x, last_y, last_f = state
413         Q[last_x, last_y, last_f, :] -= 0.02
414
415     # Store episode data
416     episode_data.append({
417         'episode': episode + 1,
418         'cycle': cycle,
419         'path': path,
420         'total_reward': total_reward,
421         'steps': steps,
422         'reached_goal': done,
423         'obstacle_config': str(get_obstacle_config_key(env.true_obstacles))
424     })
425
426     print(f"[Cycle {cycle}] Episode {episode+1}/{num_episodes}: steps={steps},
427             ↪ reached={done}, eps={epsilon:.3f}, total_reward={total_reward:.2f}")
428
429     if episode % obstacle_change_interval == obstacle_change_interval - 1 or
430     episode == num_episodes - 1:
431         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,
436         ↪ tick_time=int(tick_time / 2))
437         if not cont:
438             break
439
440     epsilon = max(epsilon_min, epsilon * epsilon_decay)
441
442     print('Training finished. Saving all snapshots and episode data.')
443     # if policy_db_local:
444     #     save_policies_append(policy_db_local, use_csv=use_csv)
445
446     # Save episode data to CSV
447     episode_df = pd.DataFrame([
448         {
449             'episode': data['episode'],
450             'cycle': data['cycle'],
451             'path': ';' .join([f"({x},{y},{dir_names[f]})" for x, y, f in
452             ↪ data['path']]),
453             'total_reward': data['total_reward'],
454             'steps': data['steps'],
455             'reached_goal': data['reached_goal'],
456             'obstacle_config': data['obstacle_config']
457         } for data in episode_data
458     ])
459     params_df = pd.DataFrame({
460         'learning_rate': [learning_rate],
461         'discount_factor': [discount_factor],
462         'epsilon': [epsilon],
463         'epsilon_min': [epsilon_min],
464         'epsilon_decay': [epsilon_decay],
465         'num_episodes': [num_episodes],
466         'grid_size': [grid_size],
467         'obstacle_change_interval': [obstacle_change_interval],
468         'max_steps_per_episode': [max_steps_per_episode]
469     })
470
471     with pd.ExcelWriter(episodes_output, engine='openpyxl', mode='w') as writer:
472         params_df.to_excel(writer, sheet_name='Hyperparameters', index=False)
473         episode_df.to_excel(writer, sheet_name='EpisodeData', index=False)
474         print(f"Episode data saved to {episodes_output}.")
475
476     if not no_render:
477         last_path = get_learned_path(env, Q)
478         index = 0
479         clock = pygame.time.Clock()
480         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 +
492 → 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
504 → Observability')
504     parser.add_argument('--no-render', action='store_true', help='Disable Pygame
505 → rendering')
505     parser.add_argument('--use-csv', action='store_true', help='Save policies as CSV
506 → instead of Excel')
506     args = parser.parse_args()
507
508     preferred_path = ['forward']*(grid_size-1) + ['turn_right'] +
509 → ['forward']*(grid_size-1)
509     main(preferred_path=preferred_path, no_render=args.no_render,
510 → use_csv=args.use_csv)
```

## 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 2
5 #define IN1 22
6 #define IN2 23
7 #define IN3 24
8 #define IN4 25
9 #define ENB 3
10 #define TRIG_PIN 9
11 #define ECHO_PIN 10
12 #define GRID_SIZE 5
13 #define NUM_ACTIONS 3
14 #define CELL_DIST 10
15 #define MOVE_TIME 300
16
17 #define TURN_TIME 700
18 #define ACTION_WAIT 1000
19 #define RESET_DELAY 10000
20 #define MAX_STEPS 50
21
22 // Q-LEARNING HYPERPARAMETERS - OPTIMISS
23 #define LEARNING_RATE 0.5           // Apprentissage rapide
24 #define DISCOUNT 0.9              // Focus objectif proche
25 #define EPSILON 0.3
26 #define EPSILON_MIN 0.05          // 5% exploration minimale
27 #define EPSILON_DECAY 0.95        // Dcroissance rapide
28 #define NUM_CYCLES 50
29
30 // Tables q-learning
31 int8_t Q[GRID_SIZE][GRID_SIZE][4][NUM_ACTIONS] = {0};
32
33 // Variables d'tat
34 int pos_x = 0, pos_y = 0, facing = 1;
35 int goal_x = 4, goal_y = 4;
36 bool known_obstacles[GRID_SIZE][GRID_SIZE] = {false};
37 float epsilon = EPSILON;
38
39 // TABLEAUX D'ENREGISTREMENT DES DONNES - OPTIMISS
40 int8_t cycle_steps[NUM_CYCLES];           // Nombre d'isodes (actions) par cycle
41 int cycle_rewards[NUM_CYCLES];            // Rcompense totale par cycle (int au lieu de
42     ↪ float)
42 byte cycle_success[NUM_CYCLES];           // 0=chec, 1=succs (byte au lieu de bool)
43
44 // Statistiques en temps rel (pas de stockage complet)
45 int current_cycle = 0;
46 int current_episodes = 0;
47 int current_reward = 0;
48 unsigned long cycle_start_time = 0;

```

```

49
50 // Constantes
51 const int deltas[4][2] = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
52 const char* dir_names[4] = {"North", "East", "South", "West"};
53 const char* action_names[3] = {"FORWARD", "TURN_LEFT", "TURN_RIGHT"};
54
55 void setup() {
56     pinMode(TRIG_PIN, OUTPUT);
57     pinMode(ECHO_PIN, INPUT);
58     pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
59     pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
60     pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
61     pinMode(LED_BUILTIN, OUTPUT);
62     digitalWrite(LED_BUILTIN, LOW);
63
64     Serial.begin(9600);
65     randomSeed(analogRead(0));
66
67     // Initialisation des tableaux
68     for (int i = 0; i < NUM_CYCLES; i++) {
69         cycle_steps[i] = 0;
70         cycle_rewards[i] = 0;
71         cycle_success[i] = 0;
72     }
73
74     Serial.println(F("== Q-Learning Robot - Mmoire Optimise =="));
75     Serial.println(F("Start: (0,0) facing East | Goal: (4,4)"));
76     Serial.println(F("Cycles: 50 | Terminologie: Cycle = parcours, pisode =
77 ↪ action"));
78     Serial.println(F("Dbut de l'apprentissage... \n"));
79     initializeQWithPreferred();
80     blinkDuringReset();
81 }
82
83 void blinkOnAction() {
84     digitalWrite(LED_BUILTIN, HIGH);
85     delay(100);
86     digitalWrite(LED_BUILTIN, LOW);
87     delay(100);
88 }
89
90 void blinkDuringReset() {
91     unsigned long startTime = millis();
92     bool ledState = false;
93
94     Serial.println(F("== FIN DE CYCLE == Repos 10s =="));
95
96     while (millis() - startTime < RESET_DELAY) {
97         if ((millis() - startTime) % 1000 < 500) {
98             if (!ledState) {
99                 digitalWrite(LED_BUILTIN, HIGH);

```

```

99         ledState = true;
100    }
101 } else {
102     if (ledState) {
103         digitalWrite(LED_BUILTIN, LOW);
104         ledState = false;
105     }
106 }
107 delay(10);
108 }
109 digitalWrite(LED_BUILTIN, LOW);
110 }

111 void initializeQWithPreferred() {
112     int sim_x = 0, sim_y = 0, sim_f = 1;
113     float bonus = 50.0;
114     for (int i = 0; i < 4; i++) {
115         Q[sim_x][sim_y][sim_f][0] += bonus;
116         sim_y += deltas[sim_f][1];
117     }
118     Q[sim_x][sim_y][sim_f][2] += bonus;
119     sim_f = (sim_f + 1) % 4;
120     for (int i = 0; i < 4; i++) {
121         Q[sim_x][sim_y][sim_f][0] += bonus;
122         sim_x += deltas[sim_f][0];
123     }
124 }
125 }

126 int getDistance() {
127     digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
128     digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
129     digitalWrite(TRIG_PIN, LOW);
130     long duration = pulseIn(ECHO_PIN, HIGH);
131     return duration * 0.034 / 2;
132 }

133 bool revealNeighborhood() {
134     int dx = deltas[facing][0];
135     int dy = deltas[facing][1];
136     int nx = pos_x + dx;
137     int ny = pos_y + dy;
138     if (nx >= 0 && nx < GRID_SIZE && ny >= 0 && ny < GRID_SIZE) {
139         int dist = getDistance();
140         if (dist < CELL_DIST && !known_obstacles[nx][ny]) {
141             known_obstacles[nx][ny] = true;
142             Serial.print(F(">>> OBSTACLE en ("));
143             Serial.print(nx); Serial.print(","); Serial.print(ny);
144             Serial.println(F(")")));
145             // VRIFFIER SI OBSTACLE AU GOAL
146             if (nx == goal_x && ny == goal_y) {
147
148
149

```

```

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

```

199     if (action == 1) { // turn left
200         turnLeft(TURN_TIME-20);
201         facing = (facing - 1 + 4) % 4;
202         reward = -1;
203         Serial.println(F(" -1"));
204     }
205     else if (action == 2) { // turn right
206         turnRight(TURN_TIME);
207         facing = (facing + 1) % 4;
208         reward = -1;
209         Serial.println(F(" -1"));
210     }
211     else { // forward
212         int dx = deltas[facing][0];
213         int dy = deltas[facing][1];
214         int nx = pos_x + dx;
215         int ny = pos_y + dy;
216         if (nx < 0 || nx >= GRID_SIZE || ny < 0 || ny >= GRID_SIZE) {
217             Serial.println(F(" MUR -5"));
218             reward = -5;
219         }
220         else if (known_obstacles[nx][ny]) {
221             Serial.print(F(" OBS("));
222             Serial.print(nx); Serial.print(","); Serial.print(ny);
223             Serial.println(F(") -10"));
224             reward = -10;
225
226             if (nx == goal_x && ny == goal_y) {
227                 reward = -999;
228             }
229         }
230         else {
231             int dist = getDistance();
232             if (dist < CELL_DIST) {
233                 known_obstacles[nx][ny] = true;
234                 Serial.print(F(" NEW_OBS("));
235                 Serial.print(nx); Serial.print(","); Serial.print(ny);
236                 Serial.println(F(") -10"));
237                 reward = -10;
238
239                 if (nx == goal_x && ny == goal_y) {
240                     Serial.println(F("!!! GOAL BLOQU !!!"));
241                     reward = -999;
242                 }
243             }
244             else {
245                 moveForward(MOVE_TIME);
246                 pos_x = nx;
247                 pos_y = ny;
248                 reward = -1;
249                 Serial.println(F(" -1"));

```

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

```
301
302     for (int i = 0; i < NUM_CYCLES; i++) {
303         Serial.print(i + 1); Serial.print(",");
304         Serial.print(cycle_steps[i]); Serial.print(",");
305         Serial.print(cycle_rewards[i]); Serial.print(",");
306         Serial.println(cycle_success[i]);
307
308         total_episodes += cycle_steps[i];
309         total_rewards += cycle_rewards[i];
310         if (cycle_success[i]) success_count++;
311     }
312
313     Serial.println(F("\n--- RSUM ---"));
314     Serial.print(F("Moy episodes: "));
315     Serial.println((float)total_episodes / NUM_CYCLES);
316     Serial.print(F("Moy reward: "));
317     Serial.println((float)total_rewards / NUM_CYCLES);
318     Serial.print(F("Succs: "));
319     Serial.print((float)success_count * 100 / NUM_CYCLES);
320     Serial.println(F("%"));
321 }
322
323 void loop() {
324     /*
325     // Chemin Prfr
326     executeAction(0, 1);
327     executeAction(0, 2);
328     executeAction(0, 3);
329     executeAction(0, 4);
330     executeAction(2, 5);
331     executeAction(0, 6);
332     executeAction(0, 7);
333     executeAction(0, 8);
334     executeAction(0, 9);
335     blinkDuringReset();
336     reset();*/
337     static int cycle = 0;
338
339     if (cycle >= NUM_CYCLES) {
340         // AFFICHER LES STATISTIQUES
341         printStatistics();
342
343         Serial.println(F("\n==== MODE EXPLOITATION ==="));
344         epsilon = 0;
345         reset();
346
347         while (true) {
348             int action = chooseAction();
349             executeAction(action, 0);
350             if (pos_x == goal_x && pos_y == goal_y) {
351                 blinkDuringReset();
```

```

352         reset();
353     }
354 }
355 }

// DBUT DU CYCLE
reset();

359
360 bool done = false;
361 bool cycle_invalid = false;
362 int episodes = 0;
363 int total_reward = 0;

364
365 Serial.print(F("\n--- Cycle "));
366 Serial.print(cycle + 1);
367 Serial.print(F(" (eps="));
368 Serial.print(epsilon, 2);
369 Serial.println(F(") ---"));

370
371 while (!done && episodes < MAX_STEPS) {
372     int action = chooseAction();
373     int old_x = pos_x, old_y = pos_y, old_f = facing;

374
375     float reward = executeAction(action, episodes);
376     total_reward += (int)reward;

377
378     // VRIFIER SI OBSTACLE AU GOAL
379     if (reward == -999) {
380         Serial.println(F(">>> REDMARRAGE CYCLE <<<"));
381         cycle_invalid = true;
382         blinkDuringReset();
383         break;
384     }

385
386     if (pos_x == goal_x && pos_y == goal_y) done = true;

387
388     // Mise jour Q
389     int8_t max_next = Q[pos_x][pos_y][facing][0];
390     for (int a = 1; a < NUM_ACTIONS; a++) {
391         if (Q[pos_x][pos_y][facing][a] > max_next) max_next =
392             Q[pos_x][pos_y][facing][a];
393     }

394     Q[old_x][old_y][old_f][action] = (int8_t)constrain(
395         Q[old_x][old_y][old_f][action] + (int8_t)round(LEARNING_RATE * 4 * (reward +
396             DISCOUNT * max_next - Q[old_x][old_y][old_f][action])),
397             -128, 127
398     );
399
400     episodes++;
}

```

```
401 // ENREGISTRER LES DONNES
402 if (!cycle_invalid) {
403     cycle_steps[cycle] = episodes;
404     cycle_rewards[cycle] = total_reward;
405     cycle_success[cycle] = done ? 1 : 0;
406
407     Serial.print(F("Fin: "));
408     Serial.print(episodes);
409     Serial.print(F(" ep, R="));
410     Serial.print(total_reward);
411     if (done) Serial.println(F(" GOAL!"));
412     else Serial.println();
413
414     epsilon = max(EPSILON_MIN, epsilon * EPSILON_DECAY);
415     blinkDuringReset();
416     cycle++;
417 }
418 }
419 }
```

# Bibliographie

- Q-Learning in Reinforcement Learning - GeeksforGeeks :  
<https://www.geeksforgeeks.org/machine-learning/q-learning-in-python/>