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

Liste des tableaux

1 C'est quoi IAvoid

Le IAvoid est :

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

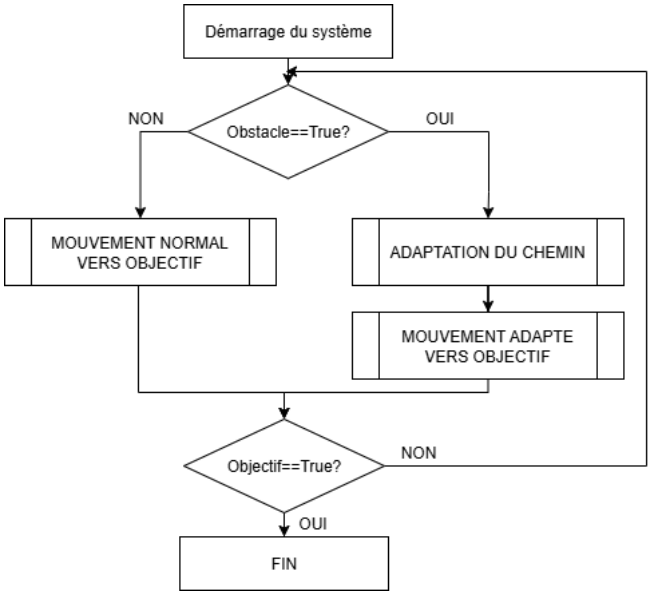


FIGURE 2.1 – Organigramme du fonctionnement général

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 fréquentés...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 choisit le **Q-learning algorithm** basé sur des recherches transversales sur ce principe d'IA. (possibilité de recherche plus comparative)

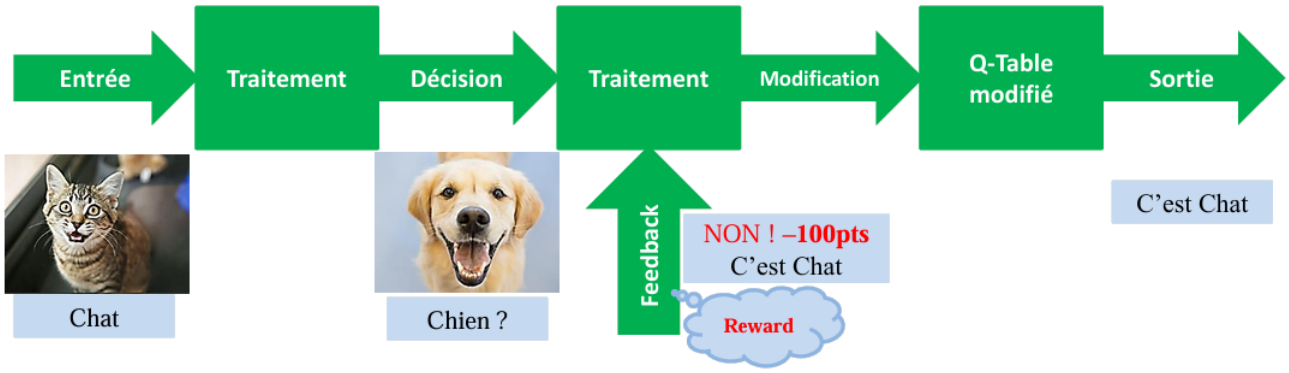


FIGURE 3.1 – Principe de Q-Learning algorithm

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)

$$Q(S,A) \longleftarrow Q(S,A) + \alpha.(R + \gamma.Q(S', Un') - Q(S,A)) \quad (1)$$

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

$$Q(S,A) = R(S,A) + \gamma.max(UnQ(S',A)) \tag{2}$$

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(Un 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) :

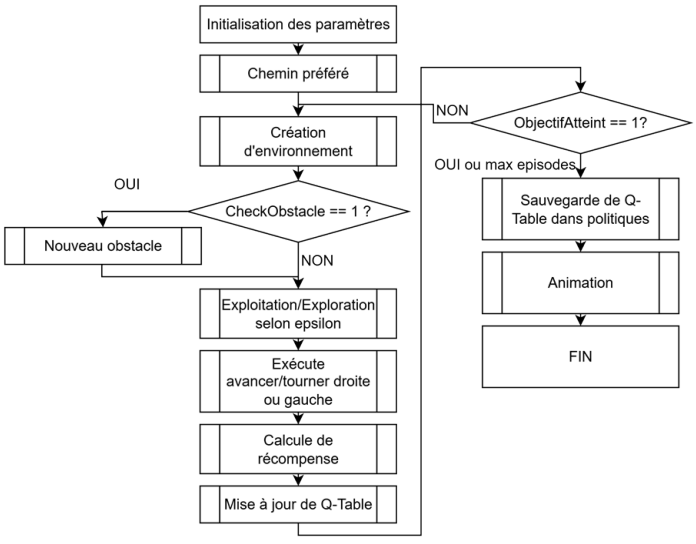


FIGURE 4.1 – Organigramme général

4.1 Hyperparamètres de la simulation

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

```
# -------------------------
# Hyperparameters
# -------------------------
#
learning_rate = 0.3
discount_factor = 0.9
epsilon = 0.7
epsilon_min = 0.01
epsilon_decay = 0.997
num_episodes = 5000
grid_size = 15
obstacle_change_interval = 200
max_steps_per_episode = 250
cell_size = 40
screen_size = grid_size * cell_size
tick_time = 240
episodes_output = "episode_data.xlsx"
```

4.2 Environnement de la simulation

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

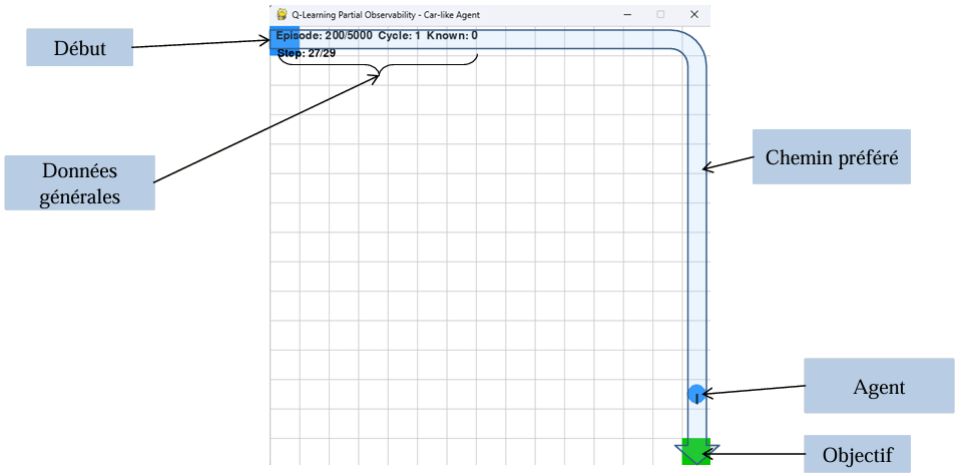


FIGURE 4.2 – Environnement de simulation sur Python

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**.

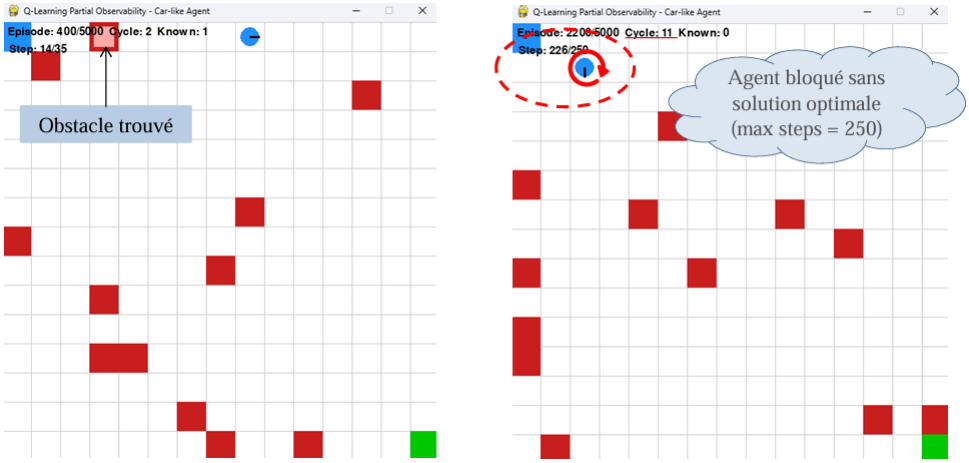


FIGURE 4.3 – Simulation de la génération des obstacles

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.

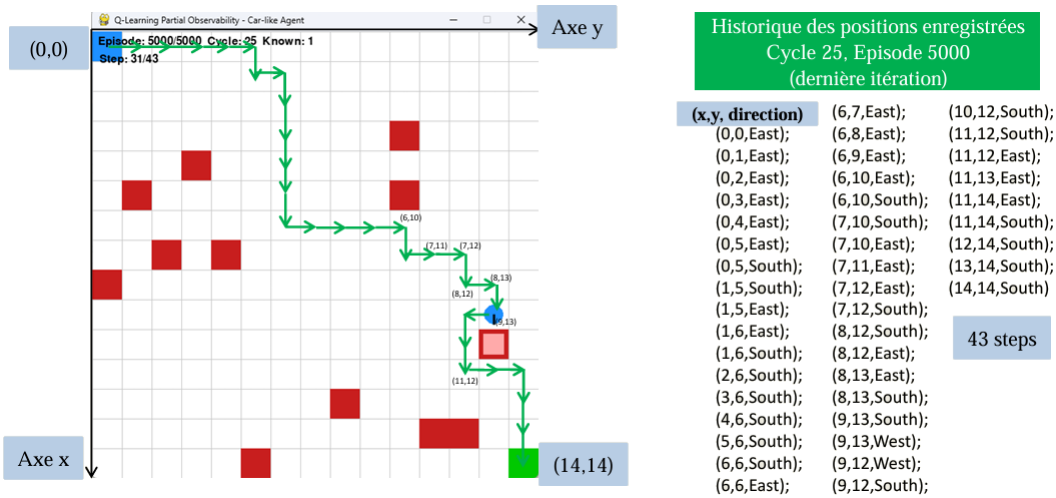


FIGURE 4.4 – Solution choisie pendant la dernière itération

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 meilleur solution à fur et à mesure)

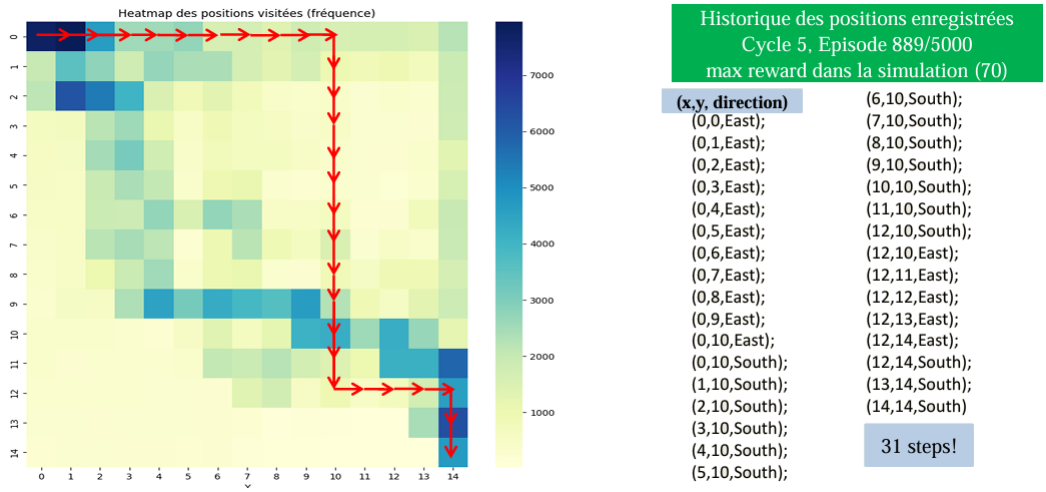


FIGURE 4.5 – Meilleur solution choisie pendant une simulation/ meilleur récompense

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)

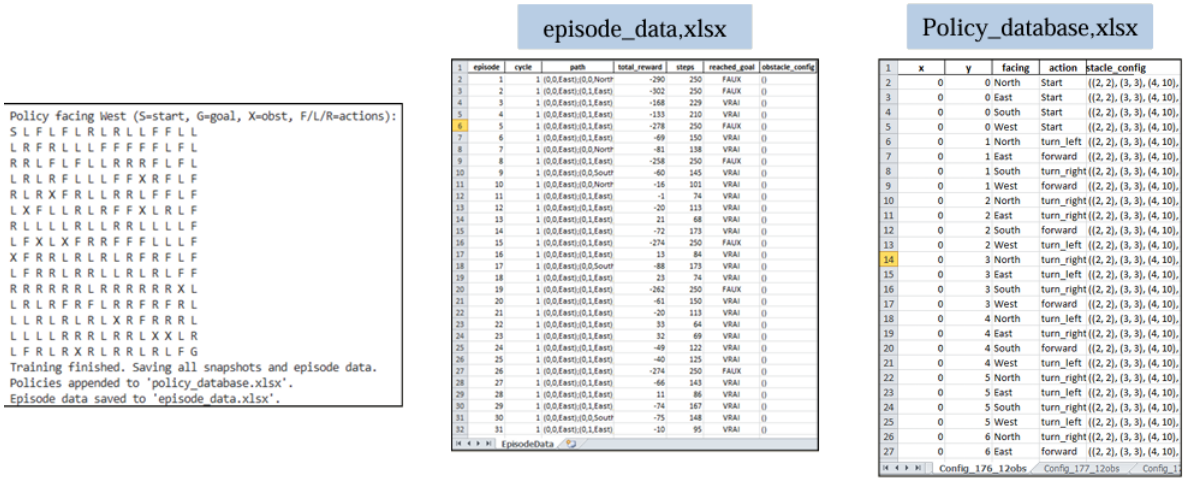


FIGURE 4.6 – Stockage des données pour analyse

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 défit au cycle 4.

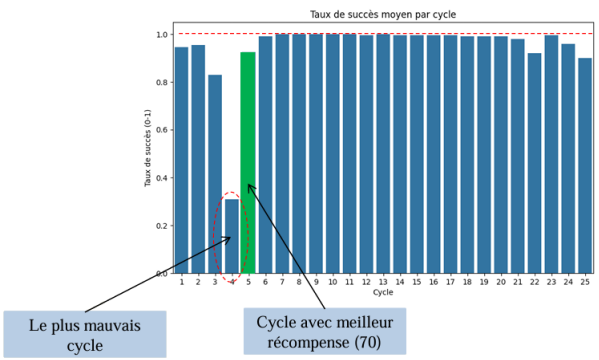


FIGURE 4.7 – Taux de succès d'une simulation

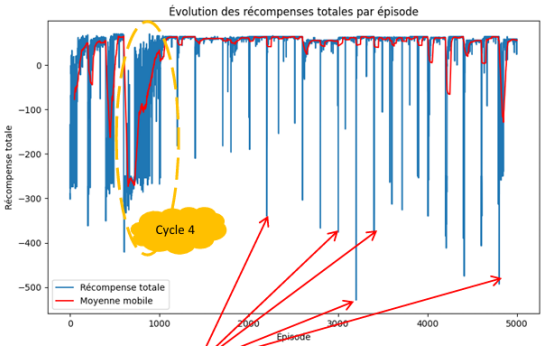


FIGURE 4.8 – Evolution de la récompense par itération

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

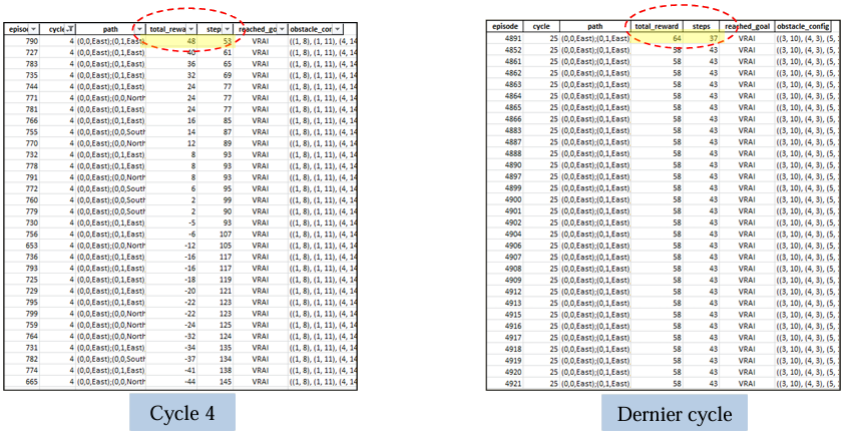


FIGURE 4.9 – Comparaison entre le cycle 4 et le dernier cycle

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

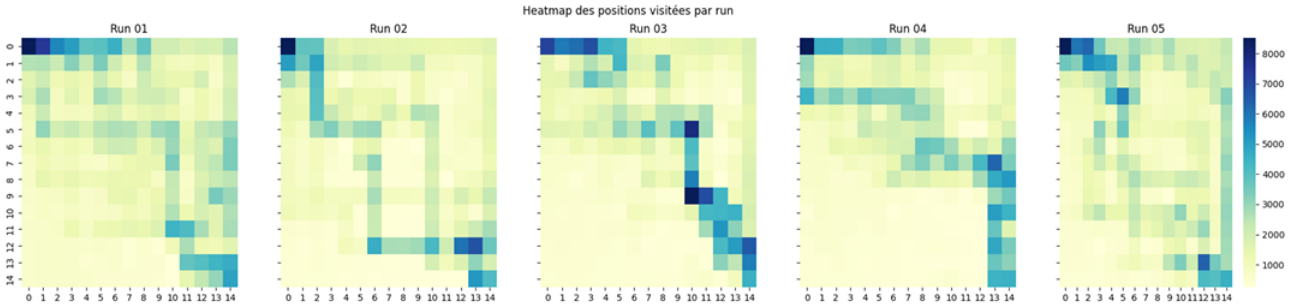


FIGURE 5.1 – Comparaison Heatmap - mêmes paramètres

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 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

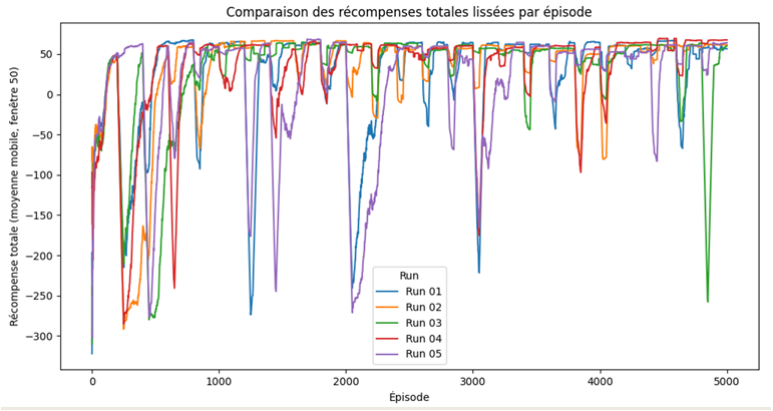


FIGURE 5.2 – Comparaison Récompenses par itération lissées - mêmes paramètres

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

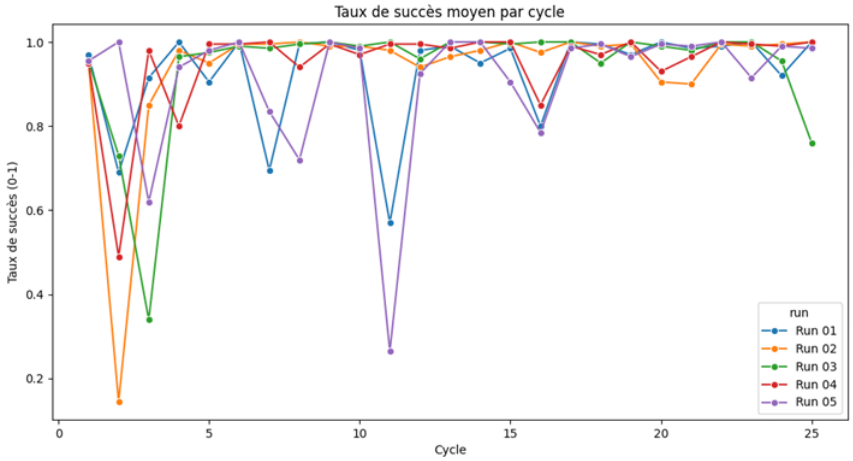


FIGURE 5.3 – Comparaison Taux de réussite - mêmes paramètres

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

5.4 Conclusion

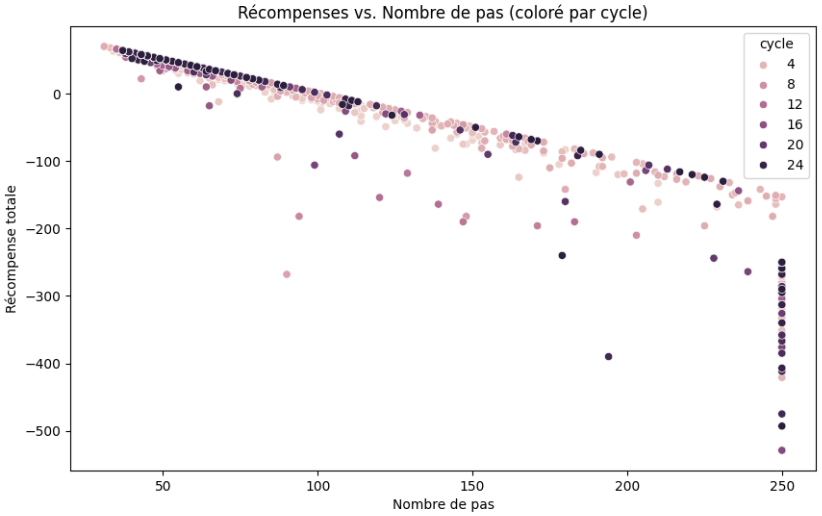


FIGURE 5.4 – Comparaison récompense/steps - une simulation

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

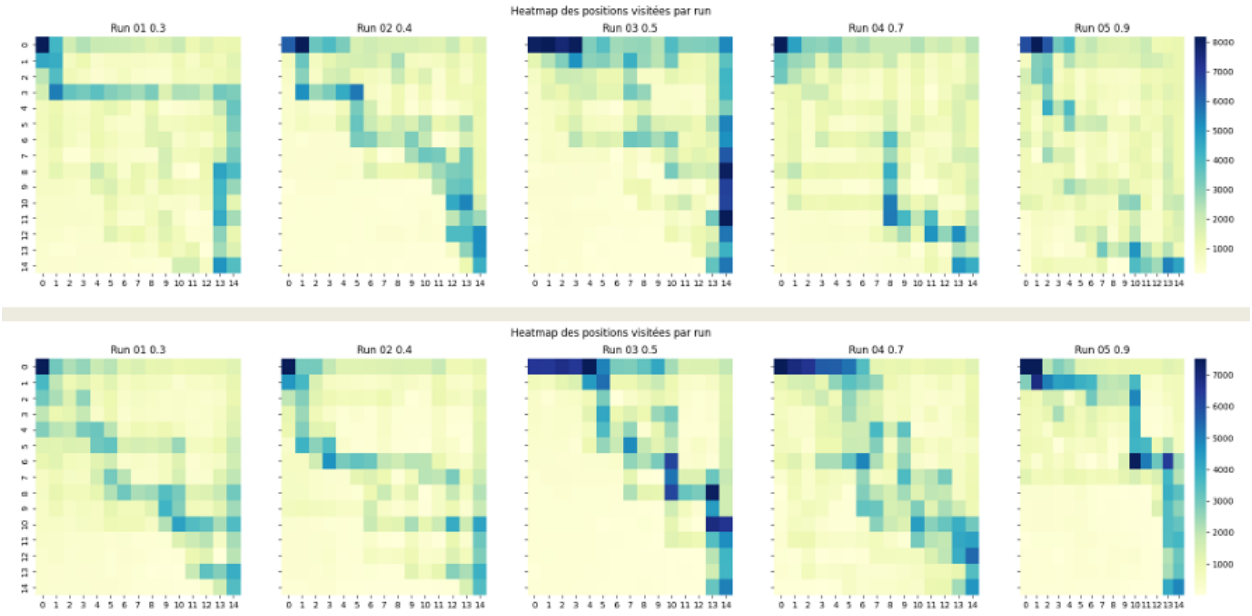
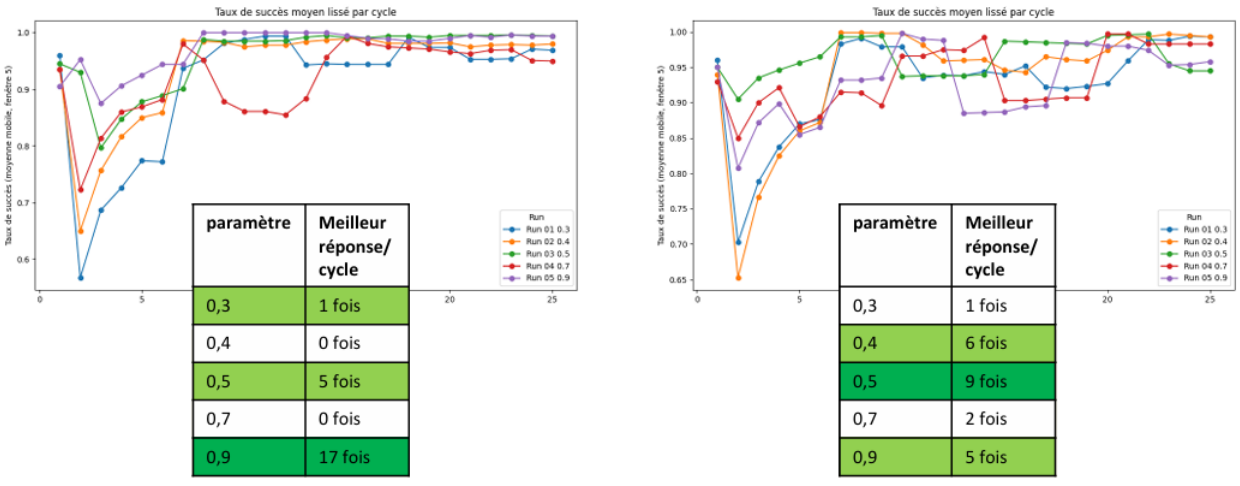


FIGURE 6.1 – Comparaison Learning_rate - Heatmap



paramètre	Meilleur réponse/ cycle
0,3	1 fois
0,4	0 fois
0,5	5 fois
0,7	0 fois
0,9	17 fois

paramètre	Meilleur réponse/ cycle
0,3	1 fois
0,4	6 fois
0,5	9 fois
0,7	2 fois
0,9	5 fois

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 prafois complexes, le choix **d'augmenter le Learning_rate** est judicieux pour des taux de réussite plus optimal.

6.2 Epsilon

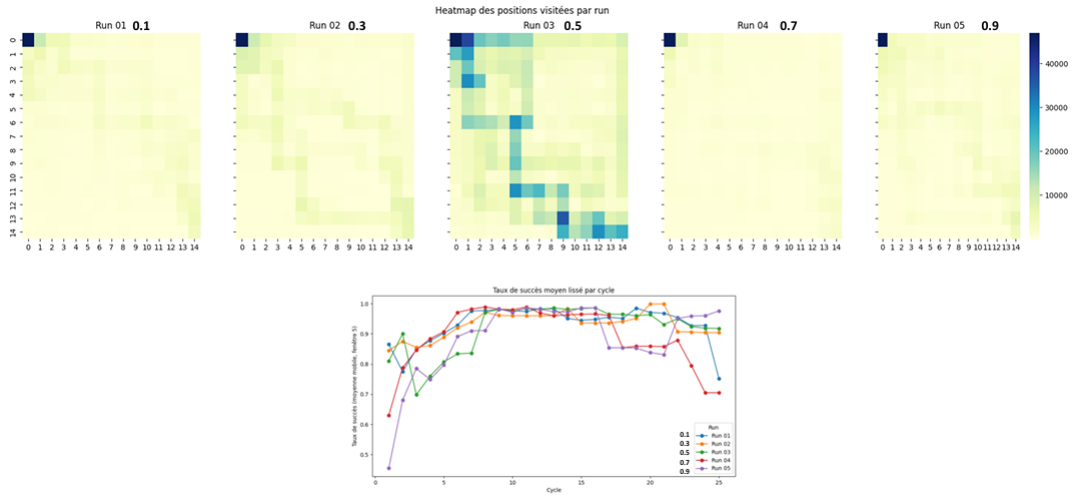


FIGURE 6.3 – Comparaison Epsilon - Vision globale

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

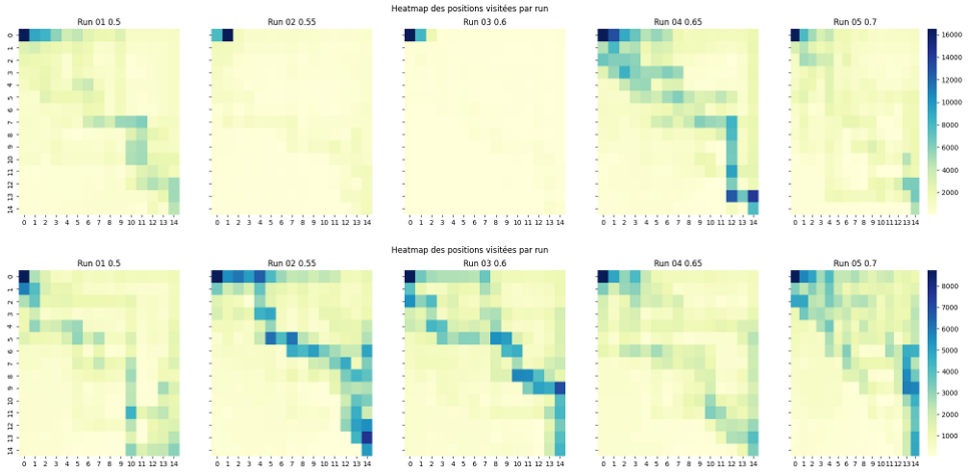


FIGURE 6.4 – Comparaison Epsilon - Heatmap

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

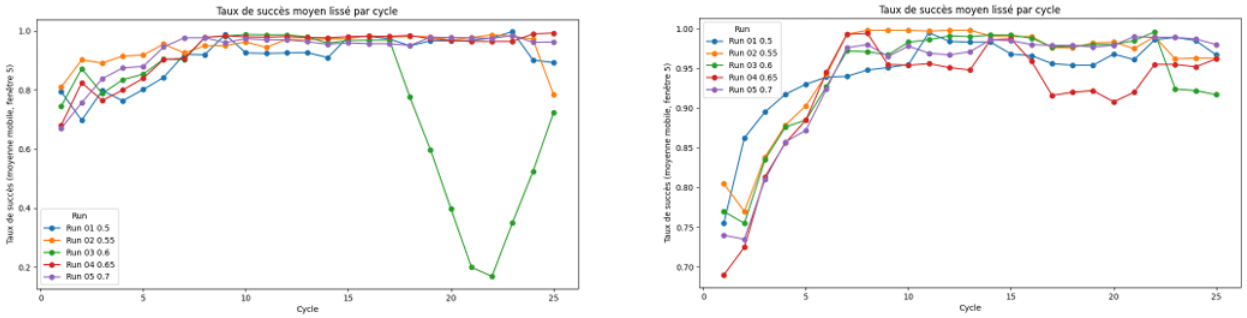


FIGURE 6.5 – Comparaison Epsilon - Taux de réussite

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.

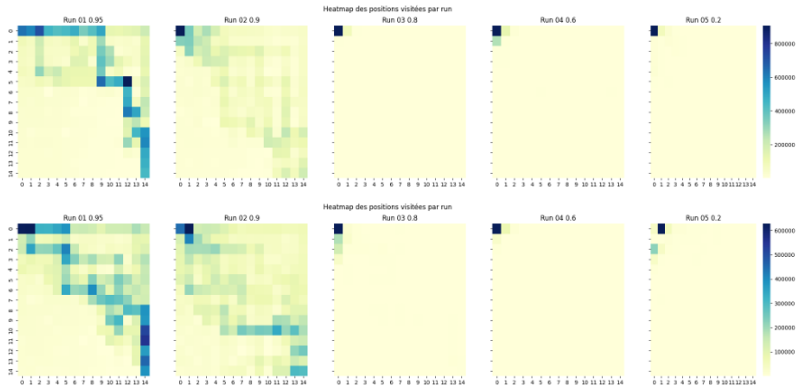


FIGURE 6.6 – Comparaison Discount_factor - Heatmap

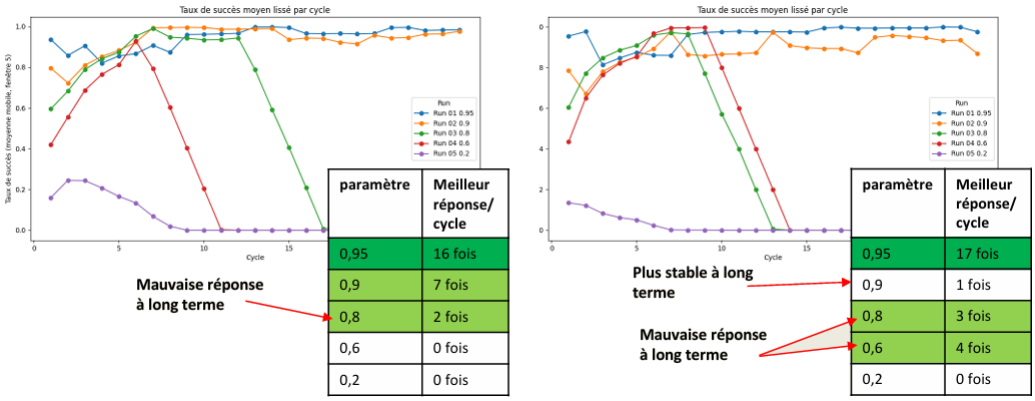


FIGURE 6.7 – Comparaison Discount_factor - Taux de réussite

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

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
```

```
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 ===’
    ↪
```

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

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

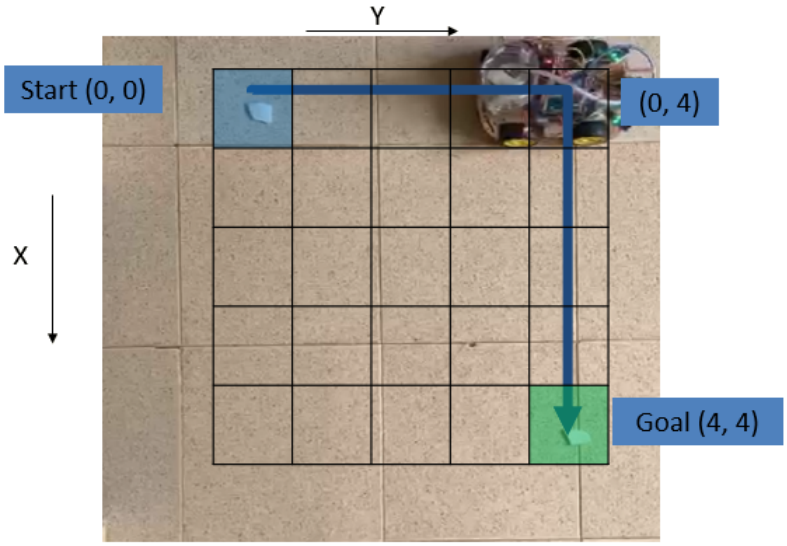


FIGURE 7.1 – Chemin préféré - Simulation ARDUINO

décisions aléatoires avec la probabilité ϵ pour explorer de nouvelles solutions plus optimales.

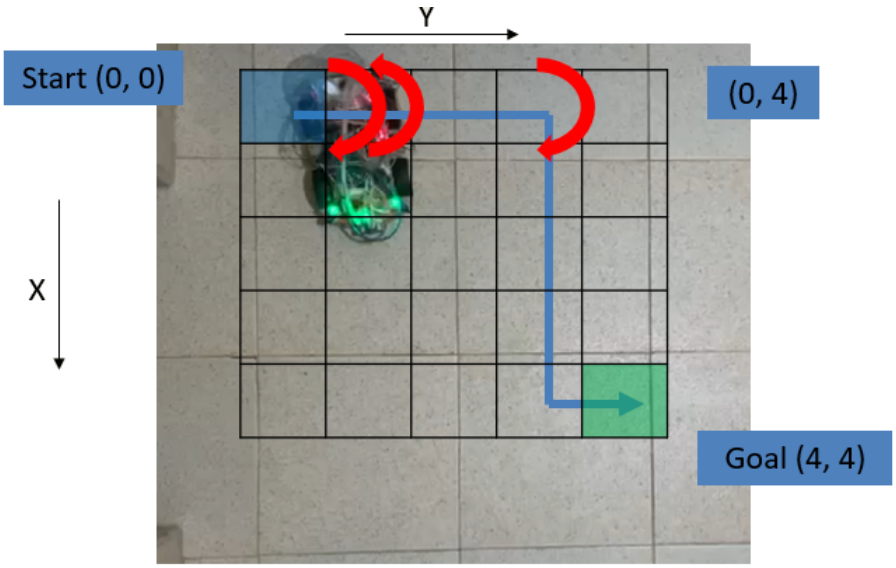


FIGURE 7.2 – Test 1 - Simulation ARDUINO

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

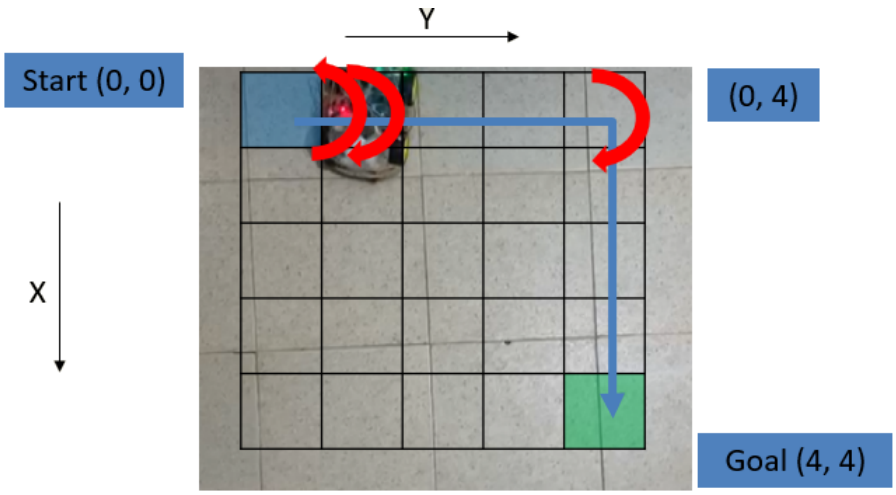


FIGURE 7.3 – Test 2 - Simulation ARDUINO

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

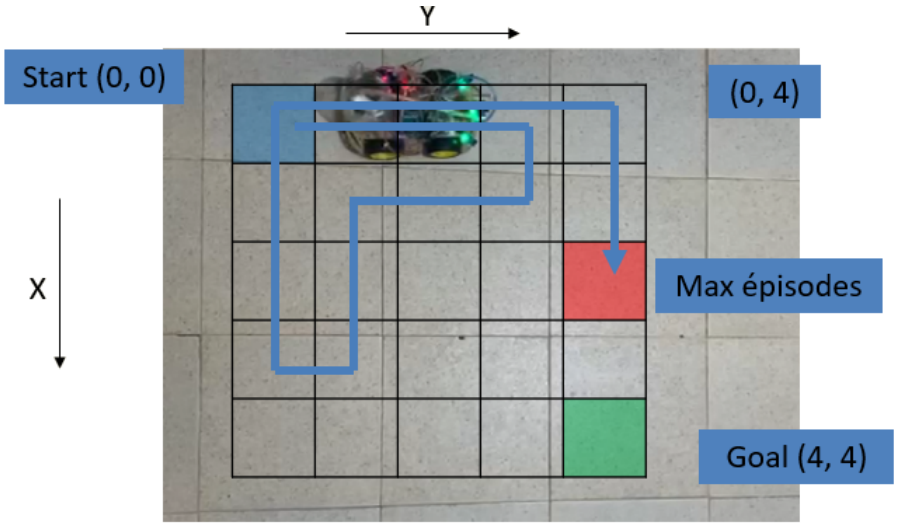


FIGURE 7.4 – Test 3 - Simulation ARDUINO

Annexes

Annexe A : Code complet simulation Python

```python
import pygame
import sys
import numpy as np
import random
import pandas as pd
import time
from collections import deque
import argparse
import uuid

# -------------------------
# Hyperparameters
# -------------------------
#
learning_rate = 0.3
discount_factor = 0.9
epsilon = 0.7
epsilon_min = 0.01
epsilon_decay = 0.997
num_episodes = 5000
grid_size = 15
obstacle_change_interval = 200
max_steps_per_episode = 250
cell_size = 40
screen_size = grid_size * cell_size
tick_time = 240
episodes_output = "episode_data.xlsx"

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (200, 30, 30)
PINK = (255, 170, 170)
GREEN = (0, 200, 0)
BLUE = (30, 144, 255)
GREY = (200, 200, 200)

# Directions: 0-North (up), 1-East (right), 2-South (down), 3-West (left)
deltas = [(-1, 0), (0, 1), (1, 0), (0, -1)]
```

```
dir_names = ['North', 'East', 'South', 'West']

# Actions: 0-forward, 1-turn_left, 2-turn_right
action_names = ['forward', 'turn_left', 'turn_right']
act_chars = ['F', 'L', 'R']
action_to_idx = {name: idx for idx, name in enumerate(action_names)}

# -------------------------
# Environment class
# -------------------------
class Environment:
    def __init__(self):
        self.start_pos = (0, 0)
        self.start_facing = 1  # East
        self.goal = (grid_size - 1, grid_size - 1)
        self.true_obstacles = set()
        self.known_obstacles = set()
        self.position = self.start_pos
        self.facing = self.start_facing

    def reset(self, keep_known=False):
        self.position = self.start_pos
        self.facing = self.start_facing
        if not keep_known:
            self.known_obstacles = set()
        self.reveal_neighborhood()
        return (self.position[0], self.position[1], self.facing)

    def add_random_obstacles(self, num):
        self.true_obstacles = set()
        attempts = 0
        while len(self.true_obstacles) < num and attempts < num * 30:
            attempts += 1
            x = random.randint(0, grid_size - 1)
            y = random.randint(0, grid_size - 1)
            coord = (x, y)
            if coord == self.start_pos or coord == self.goal:
                continue
            self.true_obstacles.add(coord)
        while not self.is_path_possible():
            if not self.true_obstacles:
                break
            self.true_obstacles.pop()

    def is_path_possible(self):
        visited = set()
        queue = deque([(self.start_pos, self.start_facing)])
        visited.add((self.start_pos[0], self.start_pos[1], self.start_facing))
        while queue:
            (x, y), f = queue.popleft()
            if (x, y) == self.goal:
```

```
                return True
            # Try forward
            dx, dy = deltas[f]
            nx, ny = x + dx, y + dy
            if 0 <= nx < grid_size and 0 <= ny < grid_size and (nx, ny) not in
    ↪ self.true_obstacles:
                if (nx, ny, f) not in visited:
                    visited.add((nx, ny, f))
                    queue.append(((nx, ny), f))
            # Try turn_left
            nf = (f - 1) % 4
            if (x, y, nf) not in visited:
                visited.add((x, y, nf))
                queue.append(((x, y), nf))
            # Try turn_right
            nf = (f + 1) % 4
            if (x, y, nf) not in visited:
                visited.add((x, y, nf))
                queue.append(((x, y), nf))
        return False

    def reveal_neighborhood(self):
        x, y = self.position
        dx, dy = deltas[self.facing]
        nx, ny = x + dx, y + dy
        if 0 <= nx < grid_size and 0 <= ny < grid_size:
            coord = (nx, ny)
            if coord in self.true_obstacles:
                self.known_obstacles.add(coord)

    def step(self, action):
        done = False
        if action == 1:  # turn_left
            self.facing = (self.facing - 1) % 4
            reward = -1
            self.reveal_neighborhood()
            return (self.position[0], self.position[1], self.facing), reward, done
        elif action == 2:  # turn_right
            self.facing = (self.facing + 1) % 4
            reward = -1
            self.reveal_neighborhood()
            return (self.position[0], self.position[1], self.facing), reward, done

        # action == 0: forward
        dx, dy = deltas[self.facing]
        tx = max(0, min(self.position[0] + dx, grid_size - 1))
        ty = max(0, min(self.position[1] + dy, grid_size - 1))
        target = (tx, ty)

        if target == self.position:  # hit wall
            reward = -5
```

```
            self.reveal_neighborhood()
            return (self.position[0], self.position[1], self.facing), reward, done

        if target in self.true_obstacles:
            reward = -10
            self.reveal_neighborhood()
            return (self.position[0], self.position[1], self.facing), reward, done

        self.position = target
        self.reveal_neighborhood()
        if self.position == self.goal:
            reward = 100
            done = True
        else:
            reward = -1
        return (self.position[0], self.position[1], self.facing), reward, done

# -------------------------
# Utilities
# -------------------------
def get_obstacle_config_key(obstacles):
    return tuple(sorted(obstacles))

def draw_grid(screen, env, episode, cycle):
    screen.fill(WHITE)
    for x in range(0, screen_size, cell_size):
        pygame.draw.line(screen, GREY, (x, 0), (x, screen_size))
    for y in range(0, screen_size, cell_size):
        pygame.draw.line(screen, GREY, (0, y), (screen_size, y))

    for ox, oy in env.true_obstacles:
        pygame.draw.rect(screen, RED, (oy * cell_size, ox * cell_size, cell_size,
    ↪ cell_size))
    for ox, oy in env.known_obstacles:
        pygame.draw.rect(screen, PINK, (oy * cell_size + 6, ox * cell_size + 6,
    ↪ cell_size - 12, cell_size - 12))

    pygame.draw.rect(screen, BLUE, (env.start_pos[1] * cell_size, env.start_pos[0] *
    ↪ cell_size, cell_size, cell_size))
    pygame.draw.rect(screen, GREEN, (env.goal[1] * cell_size, env.goal[0] *
    ↪ cell_size, cell_size, cell_size))

    font = pygame.font.SysFont(None, 22)
    txt = font.render(f"Episode: {episode}/{num_episodes}  Cycle: {cycle}  Known:
    ↪ {len(env.known_obstacles)}", True, BLACK)
    screen.blit(txt, (8, 6))

def get_learned_path(env, Q, max_steps=max_steps_per_episode):
    sim_env = Environment()
    sim_env.start_pos = env.start_pos
    sim_env.start_facing = env.start_facing
```

```
    sim_env.goal = env.goal
    sim_env.true_obstacles = set(env.true_obstacles)
    state = sim_env.reset(keep_known=False)
    path = [state]
    done = False
    steps = 0
    while not done and steps < max_steps:
        x, y, f = state
        action = int(np.argmax(Q[x, y, f]))
        new_state, _, done = sim_env.step(action)
        path.append(new_state)
        state = new_state
        steps += 1
    return path

def animate_path(screen, path, env, episode, cycle, tick_time=60):
    index = 0
    clock = pygame.time.Clock()
    font = pygame.font.SysFont(None, 22)
    while index < len(path):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False
        draw_grid(screen, env, episode, cycle)
        x, y, f = path[index]
        cx = y * cell_size + cell_size // 2
        cy = x * cell_size + cell_size // 2
        radius = cell_size // 3
        pygame.draw.circle(screen, BLUE, (cx, cy), radius)
        drow, dcol = deltas[f]
        pygame.draw.line(screen, BLACK, (cx, cy), (cx + dcol * radius, cy + drow *
    ↪ radius), 3)
        txt = font.render(f"Step: {index}/{len(path)-1}", True, BLACK)
        screen.blit(txt, (10, 30))
        pygame.display.flip()
        clock.tick(tick_time)
        index += 1
    time.sleep(0.3)
    return True

def print_policy_to_console(Q, env):
    for f in range(4):
        print(f"\nPolicy facing {dir_names[f]} (S=start, G=goal, X=obst,
    ↪ F/L/R=actions):")
        for x in range(grid_size):
            row = []
            for y in range(grid_size):
                pos = (x, y)
                if pos == env.start_pos:
                    row.append('S ')
                elif pos == env.goal:
```

```
                    row.append('G ')
                elif pos in env.true_obstacles:
                    row.append('X ')
                else:
                    a = np.argmax(Q[x, y, f])
                    row.append(act_chars[a] + ' ')
            print(''.join(row))

def initialize_q_with_preferred_path(Q, preferred_path, env):
    sim_env = Environment()
    sim_env.start_pos = env.start_pos
    sim_env.start_facing = env.start_facing
    sim_env.goal = env.goal
    sim_env.true_obstacles = set()
    state = sim_env.reset(keep_known=False)
    q_bonus = 50.0
    for action_name in preferred_path:
        if action_name not in action_to_idx:
            print(f"Invalid action '{action_name}' in preferred path. Skipping.")
            continue
        action_idx = action_to_idx[action_name]
        x, y, f = state
        Q[x, y, f, action_idx] += q_bonus
        new_state, _, done = sim_env.step(action_idx)
        state = new_state
        if done:
            break

# -------------------------
# Excel/CSV append helper
# -------------------------
from openpyxl import load_workbook

def save_policies_append(policy_db, filename="policy_database.xlsx", use_csv=False):
    if use_csv:
        for idx, (config_key, q_table) in enumerate(policy_db.items()):
            rows = []
            for x in range(q_table.shape[0]):
                for y in range(q_table.shape[1]):
                    for f in range(4):
                        pos = (x, y)
                        if pos == (0, 0):
                            action = "Start"
                        elif pos == (q_table.shape[0] - 1, q_table.shape[1] - 1):
                            action = "Goal"
                        elif pos in config_key:
                            action = "Obstacle"
                        else:
                            action_idx = int(np.argmax(q_table[x, y, f]))
                            action = action_names[action_idx]
                        rows.append({
```

```
                            "x": x, "y": y, "facing": dir_names[f],
                            "action": action, "obstacle_config": str(config_key)
                        })
            df = pd.DataFrame(rows)
            csv_file = f"policy_config_{idx}_{len(config_key)}obs.csv"
            df.to_csv(csv_file, index=False)
            print(f"Policy saved to '{csv_file}'.")
    else:
        try:
            book = load_workbook(filename)
            existing_sheets = [ws.title for ws in book.worksheets]
            with pd.ExcelWriter(filename, engine="openpyxl", mode="a",
↪ if_sheet_exists="new") as writer:
                start_idx = len(existing_sheets)
                for idx, (config_key, q_table) in enumerate(policy_db.items(),
↪ start=start_idx):
                    rows = []
                    for x in range(q_table.shape[0]):
                        for y in range(q_table.shape[1]):
                            for f in range(4):
                                pos = (x, y)
                                if pos == (0, 0):
                                    action = "Start"
                                elif pos == (q_table.shape[0] - 1, q_table.shape[1]
↪ - 1):
                                    action = "Goal"
                                elif pos in config_key:
                                    action = "Obstacle"
                                else:
                                    action_idx = int(np.argmax(q_table[x, y, f]))
                                    action = action_names[action_idx]
                                rows.append({
                                    "x": x, "y": y, "facing": dir_names[f],
                                    "action": action, "obstacle_config":
↪ str(config_key)
                                })
                    df = pd.DataFrame(rows)
                    sheet_name = f"Config_{idx}_{len(config_key)}obs"
                    df.to_excel(writer, sheet_name=sheet_name[:31], index=False)
        except FileNotFoundError:
            with pd.ExcelWriter(filename, engine="openpyxl", mode="w") as writer:
                for idx, (config_key, q_table) in enumerate(policy_db.items()):
                    rows = []
                    for x in range(q_table.shape[0]):
                        for y in range(q_table.shape[1]):
                            for f in range(4):
                                pos = (x, y)
                                if pos == (0, 0):
                                    action = "Start"
                                elif pos == (q_table.shape[0] - 1, q_table.shape[1]
↪ - 1):
```

```
                                    action = "Goal"
                                elif pos in config_key:
                                    action = "Obstacle"
                                else:
                                    action_idx = int(np.argmax(q_table[x, y, f]))
                                    action = action_names[action_idx]
                                rows.append({
                                    "x": x, "y": y, "facing": dir_names[f],
                                    "action": action, "obstacle_config":
    ↪ str(config_key)
                                })
                    df = pd.DataFrame(rows)
                    sheet_name = f"Config_{idx}_{len(config_key)}obs"
                    df.to_excel(writer, sheet_name=sheet_name[:31], index=False)
        print(f"Policies appended to '{filename}'.")

# -------------------------
# Main training loop
# -------------------------
def main(preferred_path=None, no_render=False, use_csv=False):
    global epsilon
    pygame.init()
    screen = None if no_render else pygame.display.set_mode((screen_size,
    ↪ screen_size))
    if not no_render:
        pygame.display.set_caption('Q-Learning Partial Observability - Car-like
    ↪ Agent')

    env = Environment()
    Q = np.zeros((grid_size, grid_size, 4, 3))  # Updated for 3 actions
    policy_db_local = {}
    episode_data = []  # To store paths and rewards
    cycle = 0

    env.true_obstacles = set()
    env.reset(keep_known=False)
    if preferred_path:
        print(f"Applying preferred path for empty map: {preferred_path}")
        initialize_q_with_preferred_path(Q, preferred_path, env)

    for episode in range(num_episodes):
        if not no_render:
            for evt in pygame.event.get():
                if evt.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

        if episode % obstacle_change_interval == 0:
            cycle += 1
            if episode == 0:
                env.true_obstacles = set()
```

```
                print(f"[Cycle {cycle}] Starting with empty obstacle map.")
            else:
                env.add_random_obstacles(max(4, int(grid_size * 0.8)))
                print(f"[Cycle {cycle}] New true obstacles set
↪ ({len(env.true_obstacles)} obstacles).")

        state = env.reset(keep_known=False)
        done = False
        steps = 0
        total_reward = 0
        path = [state]

        while not done and steps < max_steps_per_episode:
            steps += 1
            x, y, f = state
            if random.random() < epsilon:
                action = random.randint(0, 2)  # Updated for 3 actions
            else:
                action = int(np.argmax(Q[x, y, f]))

            new_state, reward, done = env.step(action)
            total_reward += reward
            path.append(new_state)
            nx, ny, nf = new_state
            best_next = int(np.argmax(Q[nx, ny, nf]))
            td_target = reward + discount_factor * Q[nx, ny, nf, best_next]
            td_error = td_target - Q[x, y, f, action]
            Q[x, y, f, action] += learning_rate * td_error
            state = new_state

        if not done:
            last_x, last_y, last_f = state
            Q[last_x, last_y, last_f, :] -= 0.02

        # Store episode data
        episode_data.append({
            'episode': episode + 1,
            'cycle': cycle,
            'path': path,
            'total_reward': total_reward,
            'steps': steps,
            'reached_goal': done,
            'obstacle_config': str(get_obstacle_config_key(env.true_obstacles))
        })

        print(f"[Cycle {cycle}] Episode {episode+1}/{num_episodes}: steps={steps},
↪ reached={done}, eps={epsilon:.3f}, total_reward={total_reward:.2f}")

        if episode % obstacle_change_interval == obstacle_change_interval - 1 or
↪ episode == num_episodes - 1:
            cfg_key = get_obstacle_config_key(env.true_obstacles)
```

```
            policy_db_local[cfg_key] = Q.copy()
            print(f"[Cycle {cycle}] Snapshot saved for config (obs={len(cfg_key)}).")
            learned_path = get_learned_path(env, Q)
            print(f"[Cycle {cycle}] Learned path length: {len(learned_path)}")
            print_policy_to_console(Q, env)
            if not no_render:
                cont = animate_path(screen, learned_path, env, episode + 1, cycle,
    ↪ tick_time=int(tick_time / 2))
                if not cont:
                    break

        epsilon = max(epsilon_min, epsilon * epsilon_decay)

    print('Training finished. Saving all snapshots and episode data.')
    # if policy_db_local:
    #     save_policies_append(policy_db_local, use_csv=use_csv)

    # Save episode data to CSV
    episode_df = pd.DataFrame([
        {
            'episode': data['episode'],
            'cycle': data['cycle'],
            'path': ';'.join([f"({x},{y},{dir_names[f]})" for x, y, f in
    ↪ data['path']]),
            'total_reward': data['total_reward'],
            'steps': data['steps'],
            'reached_goal': data['reached_goal'],
            'obstacle_config': data['obstacle_config']
        } for data in episode_data
    ])
    params_df = pd.DataFrame({
        'learning_rate': [learning_rate],
        'discount_factor': [discount_factor],
        'epsilon': [epsilon],
        'epsilon_min': [epsilon_min],
        'epsilon_decay': [epsilon_decay],
        'num_episodes': [num_episodes],
        'grid_size': [grid_size],
        'obstacle_change_interval': [obstacle_change_interval],
        'max_steps_per_episode': [max_steps_per_episode]
    })
    with pd.ExcelWriter(episodes_output, engine='openpyxl', mode='w') as writer:
        params_df.to_excel(writer, sheet_name='Hyperparameters', index=False)
        episode_df.to_excel(writer, sheet_name='EpisodeData', index=False)
    print(f"Episode data saved to {episodes_output}.")

    if not no_render:
        last_path = get_learned_path(env, Q)
        index = 0
        clock = pygame.time.Clock()
        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

```
// Q-Learning Obstacle Avoidance Robot in 5x5 Grid World - Arduino Uno
// Avec enregistrement de toutes les donnes d'apprentissage
// Terminologie : Cycle = ensemble complet du dbut au goal, pisode = chaque action
#define ENA 2
#define IN1 22
#define IN2 23
#define IN3 24
#define IN4 25
#define ENB 3
#define TRIG_PIN 9
#define ECHO_PIN 10
#define GRID_SIZE 5
#define NUM_ACTIONS 3
#define CELL_DIST 10
#define MOVE_TIME 300

#define TURN_TIME 700
#define ACTION_WAIT 1000
#define RESET_DELAY 10000
#define MAX_STEPS 50

// Q-LEARNING HYPERPARAMETERS - OPTIMISS
#define LEARNING_RATE 0.5      // Apprentissage rapide
#define DISCOUNT 0.9           // Focus objectif proche
#define EPSILON 0.3
#define EPSILON_MIN 0.05       // 5% exploration minimale
#define EPSILON_DECAY 0.95     // Dcroissance rapide
#define NUM_CYCLES 50

// Tables Q-learning
int8_t Q[GRID_SIZE][GRID_SIZE][4][NUM_ACTIONS] = {0};

// Variables d'tat
int pos_x = 0, pos_y = 0, facing = 1;
int goal_x = 4, goal_y = 4;
bool known_obstacles[GRID_SIZE][GRID_SIZE] = {false};
float epsilon = EPSILON;

// TABLEAUX D'ENREGISTREMENT DES DONNES - OPTIMISS
int8_t cycle_steps[NUM_CYCLES];        // Nombre d'pisodes (actions) par cycle
int cycle_rewards[NUM_CYCLES];         // Rcompense totale par cycle (int au lieu de
    ↪ float)
byte cycle_success[NUM_CYCLES];        // 0=chec, 1=succs (byte au lieu de bool)

// Statistiques en temps rel (pas de stockage complet)
int current_cycle = 0;
int current_episodes = 0;
int current_reward = 0;
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 =
    ↪ action"));
77     Serial.println(F("Dbut de l'apprentissage...\n"));
78     initializeQWithPreferred();
79     blinkDuringReset();
80 }
81
82 void blinkOnAction() {
83     digitalWrite(LED_BUILTIN, HIGH);
84     delay(100);
85     digitalWrite(LED_BUILTIN, LOW);
86     delay(100);
87 }
88
89 void blinkDuringReset() {
90     unsigned long startTime = millis();
91     bool ledState = false;
92
93     Serial.println(F("=== FIN DE CYCLE === Repos 10s ==="));
94
95     while (millis() - startTime < RESET_DELAY) {
96         if ((millis() - startTime) % 1000 < 500) {
97             if (!ledState) {
98                 digitalWrite(LED_BUILTIN, HIGH);
```

```
                ledState = true;
            }
        } else {
            if (ledState) {
                digitalWrite(LED_BUILTIN, LOW);
                ledState = false;
            }
        }
        delay(10);
    }
    digitalWrite(LED_BUILTIN, LOW);
}

void initializeQWithPreferred() {
    int sim_x = 0, sim_y = 0, sim_f = 1;
    float bonus = 50.0;
    for (int i = 0; i < 4; i++) {
        Q[sim_x][sim_y][sim_f][0] += bonus;
        sim_y += deltas[sim_f][1];
    }
    Q[sim_x][sim_y][sim_f][2] += bonus;
    sim_f = (sim_f + 1) % 4;
    for (int i = 0; i < 4; i++) {
        Q[sim_x][sim_y][sim_f][0] += bonus;
        sim_x += deltas[sim_f][0];
    }
}

int getDistance() {
    digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    long duration = pulseIn(ECHO_PIN, HIGH);
    return duration * 0.034 / 2;
}

bool revealNeighborhood() {
    int dx = deltas[facing][0];
    int dy = deltas[facing][1];
    int nx = pos_x + dx;
    int ny = pos_y + dy;
    if (nx >= 0 && nx < GRID_SIZE && ny >= 0 && ny < GRID_SIZE) {
        int dist = getDistance();
        if (dist < CELL_DIST && !known_obstacles[nx][ny]) {
            known_obstacles[nx][ny] = true;
            Serial.print(F(">>> OBSTACLE en ("));
            Serial.print(nx); Serial.print(","); Serial.print(ny);
            Serial.println(F(")"));

            // VRIFIER SI OBSTACLE AU GOAL
            if (nx == goal_x && ny == goal_y) {
```

```
                Serial.println(F("!!!!! OBSTACLE AU GOAL (4,4) - CYCLE INVALIDE !!!!!"));
                return true;  // Signaler qu'il faut redmarrer le cycle
            }
        }
    }
    return false;  // Pas d'obstacle au goal
}

void reset() {
    pos_x = 0;
    pos_y = 0;
    facing = 1;
    memset(known_obstacles, 0, sizeof(known_obstacles));

    // Vrifier si obstacle au goal ds le dbut
    if (revealNeighborhood()) {
        // Obstacle au goal dtect - attendre repositionnement
        Serial.println(F(">>> Attente repositionnement obstacle du goal - nouveau cycle..."));
        blinkDuringReset();
        reset();  // Ressayer le cycle
    }
}

int chooseAction() {
    if (random(1000) / 1000.0 < epsilon) {
        return random(NUM_ACTIONS);
    } else {
        int best = 0;
        for (int a = 1; a < NUM_ACTIONS; a++) {
            if (Q[pos_x][pos_y][facing][a] > Q[pos_x][pos_y][facing][best]) best = a;
        }
        return best;
    }
}

float executeAction(int action, int step) {
    float reward = 0;

    // Affichage simplifi
    Serial.print(F("E"));
    Serial.print(step);
    Serial.print(F(": "));
    Serial.print(action_names[action]);
    Serial.print(F(" ("));
    Serial.print(pos_x); Serial.print(","); Serial.print(pos_y);
    Serial.print(F(")"));

    blinkOnAction();

```

```
    if (action == 1) { // turn left
        turnLeft(TURN_TIME-20);
        facing = (facing - 1 + 4) % 4;
        reward = -1;
        Serial.println(F(" -1"));
    }
    else if (action == 2) { // turn right
        turnRight(TURN_TIME);
        facing = (facing + 1) % 4;
        reward = -1;
        Serial.println(F(" -1"));
    }
    else { // forward
        int dx = deltas[facing][0];
        int dy = deltas[facing][1];
        int nx = pos_x + dx;
        int ny = pos_y + dy;
        if (nx < 0 || nx >= GRID_SIZE || ny < 0 || ny >= GRID_SIZE) {
            Serial.println(F(" MUR -5"));
            reward = -5;
        }
        else if (known_obstacles[nx][ny]) {
            Serial.print(F(" OBS("));
            Serial.print(nx); Serial.print(","); Serial.print(ny);
            Serial.println(F(") -10"));
            reward = -10;

            if (nx == goal_x && ny == goal_y) {
                reward = -999;
            }
        }
        else {
            int dist = getDistance();
            if (dist < CELL_DIST) {
                known_obstacles[nx][ny] = true;
                Serial.print(F(" NEW_OBS("));
                Serial.print(nx); Serial.print(","); Serial.print(ny);
                Serial.println(F(") -10"));
                reward = -10;

                if (nx == goal_x && ny == goal_y) {
                    Serial.println(F("!!! GOAL BLOQU !!!"));
                    reward = -999;
                }
            }
            else {
                moveForward(MOVE_TIME);
                pos_x = nx;
                pos_y = ny;
                reward = -1;
                Serial.println(F(" -1"));
```

```
            }
        }
    }

    if (pos_x == goal_x && pos_y == goal_y) {
        Serial.println(F(">>> GOAL +100 <<<"));
        reward = 100;
    }

    revealNeighborhood();
    //delay(ACTION_WAIT);
    return reward;
}

void moveForward(int ms) {
    analogWrite(ENA, 200); analogWrite(ENB, 200);
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    delay(ms);
    stopMotors();
}

void turnLeft(int ms) {
    analogWrite(ENA, 180); analogWrite(ENB, 180);
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    delay(ms);
    stopMotors();
}

void turnRight(int ms) {
    analogWrite(ENA, 180); analogWrite(ENB, 180);
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    delay(ms);
    stopMotors();
}

void stopMotors() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}

// FONCTION POUR AFFICHER LES STATISTIQUES - SIMPLIFI
void printStatistics() {
    Serial.println(F("\n===== STATISTIQUES ====="));
    Serial.println(F("Cycle,Episodes,Reward,Success"));

    int total_episodes = 0;
    long total_rewards = 0;
    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 pisodes: "));
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     }
356 
357     // DBUT DU CYCLE
358     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 =
    ↪ Q[pos_x][pos_y][facing][a];
392         }
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 +
    ↪ DISCOUNT * max_next - Q[old_x][old_y][old_f][action])),
396         -128, 127
397         );
398 
399         episodes++;
400     }
```

```
401
402     // ENREGISTRER LES DONNES
403     if (!cycle_invalid) {
404         cycle_steps[cycle] = episodes;
405         cycle_rewards[cycle] = total_reward;
406         cycle_success[cycle] = done ? 1 : 0;
407
408         Serial.print(F("Fin: "));
409         Serial.print(episodes);
410         Serial.print(F(" ep, R="));
411         Serial.print(total_reward);
412         if (done) Serial.println(F(" GOAL!"));
413         else Serial.println();
414
415         epsilon = max(EPSILON_MIN, epsilon * EPSILON_DECAY);
416         blinkDuringReset();
417         cycle++;
418     }
419 }
420
```

Bibliographie

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