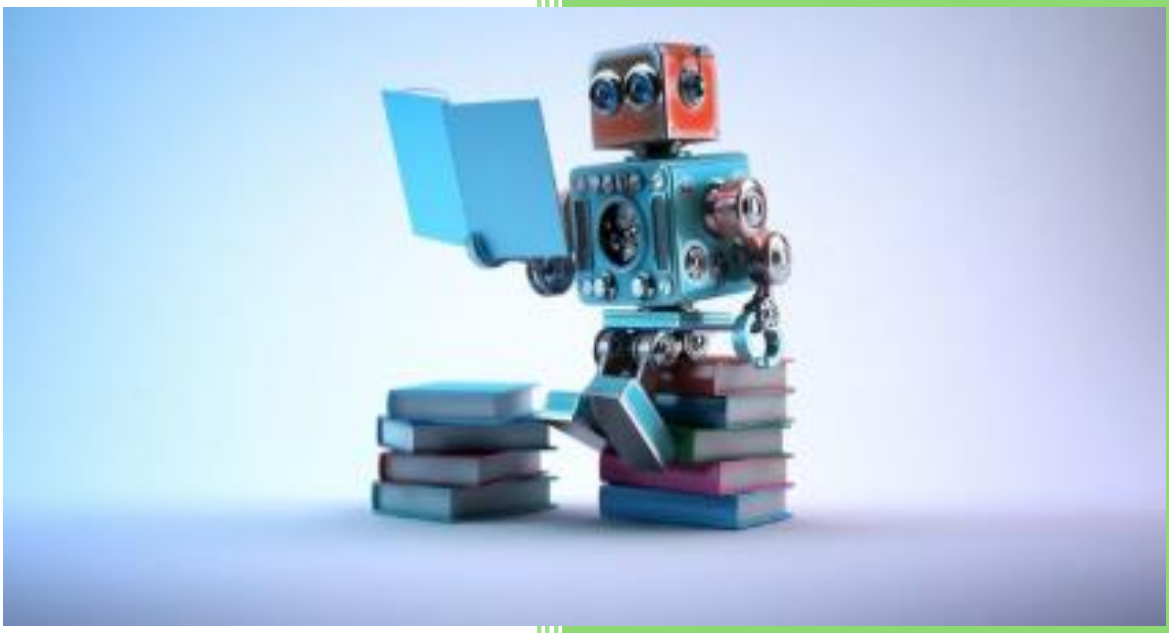
université de BORDEAUX

2024/2025

Reinforcement Learning: Projet



Ali Rachiq & Meriem Kadiri
Université de Bordeaux M2 IA
31/12/2024

Table de matière

1. Introduction 2

2. Modélisation de l'Environnement 3

2.1 Structure des Grid Worlds 3

.2 Modélisation des Agents 4

3. Approches Implémentées 4

3.1 Single Agent Model-Based 4

3.2 Single Agent Model-Free 6

3.2.1 Q-Learning 6

3.2.2 Double Q-Learning 6

3.2.3 Monte Carlo 7

3.3 Multi-Agent Model-Based 9

4. Analyse Comparative 10

5. Défis et Solutions 10

6. Améliorations Possibles 11

7. Conclusion 11

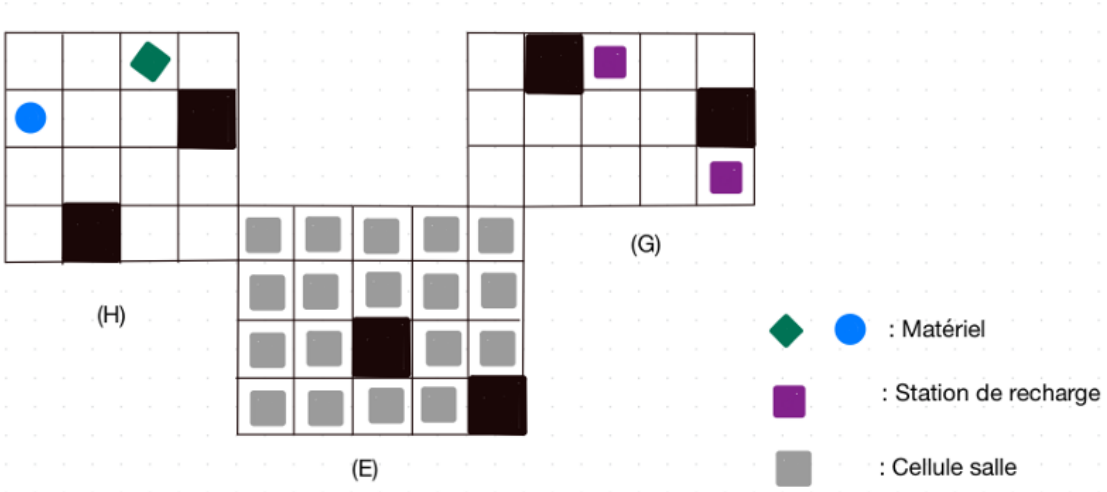
1. Introduction

Dans le cadre de ce projet, nous avons mis en œuvre différentes approches de Reinforcement Learning (RL) et de Multi-Agent Reinforcement Learning (MARL) pour résoudre un problème de nettoyage d'un environnement dangereux pour les humains. L'objectif principal est de développer des agents capables de naviguer dans trois environnements distincts : un hangar (H), un entrepôt (E) et un garage (G). Chaque jour, les robots doivent récupérer du matériel dans le hangar, nettoyer l'entrepôt, puis se recharger dans le garage. Les positions du matériel et des stations de recharge changent chaque jour, nécessitant une adaptation continue des agents.

Description générale de l'environnement et des défis

L'environnement simulé est composé de trois grid worlds de tailles potentiellement différentes :

- Hangar (H) : Contient des obstacles, des cellules vides ou du matériel. Une cellule particulière permet de sortir de l'environnement.
- Entrepôt (E) : Contient des obstacles, des cellules vides ou de la saleté. Une cellule particulière permet de sortir de l'environnement.
- Garage (G) : Contient des obstacles, des cellules vides ou des stations de recharge.



Les défis principaux incluent la gestion des obstacles, la coordination entre les agents, et l'adaptation aux changements quotidiens dans la disposition des éléments clés (matériel, saleté, stations de recharge).

Aperçu des différentes approches implémentées

Nous avons implémenté et testé trois approches différentes :

1. Single Agent Model-Based : Utilisation de la méthode de Value Iteration pour déterminer la politique optimale pour un agent unique.

2. Single Agent Model-Free : Utilisation de méthodes de Q-learning, Double Q-learning et Monte Carlo pour apprendre la politique optimale sans modèle explicite de l'environnement.
3. Multi-Agent Model-Based : Utilisation de la méthode de Value Iteration pour deux agents qui doivent coordonner leurs actions pour maximiser la récompense cumulée.

2. Modélisation de l'Environnement

2.1 Structure des Grid Worlds

Description détaillée des trois environnements (H, E, G)

- Hangar (H) : Environnement simple où les cellules peuvent être des obstacles, des cellules vides ou contenir du matériel. Une cellule particulière permet de sortir de l'environnement.
- Entrepôt (E) : Environnement plus complexe où les cellules peuvent être des obstacles, des cellules vides ou contenir de la saleté. Une cellule particulière permet de sortir de l'environnement.
- Garage (G) : Environnement simple où les cellules peuvent être des obstacles, des cellules vides ou contenir des stations de recharge.

Représentation des états et des transitions

Chaque environnement est représenté par une grille de cellules. Les états sont définis par la position des agents et l'état des tâches (matériel collecté, saleté nettoyée et rechargement). Les transitions sont déterminées par les actions des agents et les règles de l'environnement.

Système de récompenses

Les récompenses sont attribuées en fonction des actions des agents :

- Collecte de matériel ou nettoyage de saleté : récompense positive.
- Collision avec un obstacle : pénalité.
- Atteinte de l'objectif final : récompense élevée.

Donnant un exemple de Single Agent Model-Based dans le code, les récompenses sont définies dans la méthode « set_rewards_and_transitions » de la classe « GridEnvironment » et donc les récompenses seront :

- Récompense positive (**move_penalty = 10**).
- Collision avec un obstacle : pénalité (**move_penalty = -1**).
- Atteinte de l'objectif final : récompense élevée (**move_penalty = 20**).

.2 Modélisation des Agents

Définition des actions possibles

Les agents peuvent effectuer quatre actions principales : se déplacer vers le haut, vers le bas, vers la gauche ou vers la droite.

États observables

Les états observables incluent la position actuelle de l'agent et l'état des tâches (matériel collecté, saleté nettoyée et rechargement).

Objectifs spécifiques pour chaque environnement

- Hangar (H) : Trouver et collecter tout le matériel, puis sortir de l'environnement.
- Entrepôt (E) : Nettoyer toute la saleté, puis sortir de l'environnement.
- Garage (G) : Trouver et se recharger aux stations de recharge.

3. Approches Implémentées

3.1 Single Agent Model-Based

Principe théorique de Value Iteration

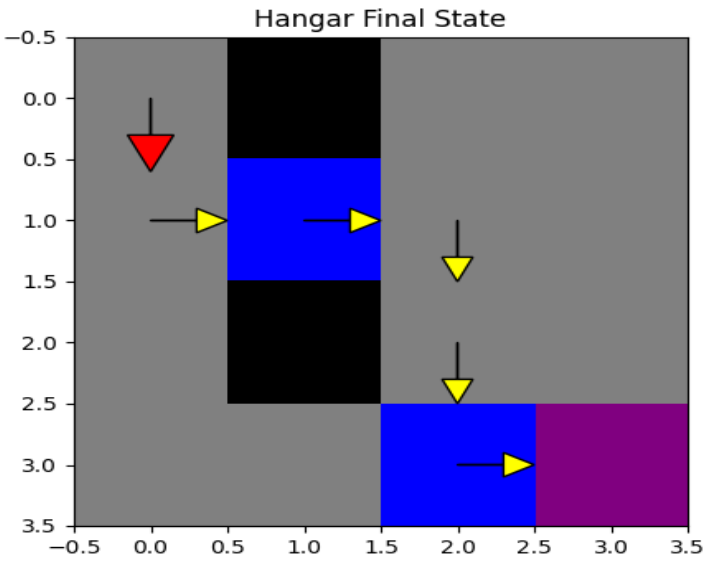
Value Iteration est une méthode de programmation dynamique utilisée pour trouver la politique optimale en résolvant l'équation de Bellman. Elle itère sur les valeurs des états jusqu'à convergence.

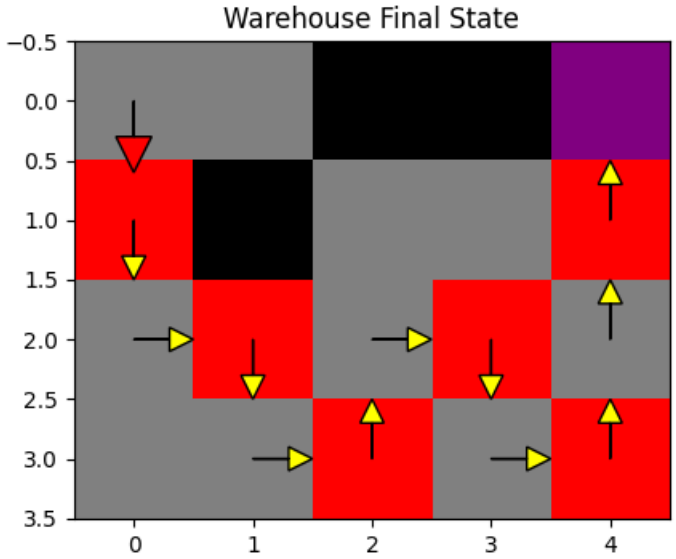
Implémentation spécifique pour notre cas

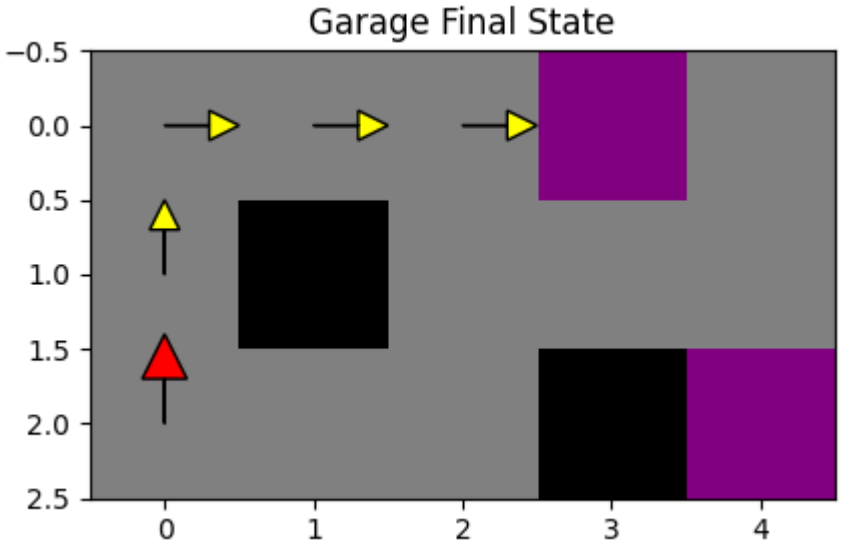
Nous avons implémenté Value Iteration pour calculer la valeur optimale de chaque état dans les trois environnements.

Résultats expérimentaux

- Courbes de convergence : Les valeurs convergent vers la politique optimale après un certain nombre d'itérations.
- Performances dans chaque environnement : Les agents atteignent leurs objectifs de manière optimale.
- Analyse des politiques apprises : Les politiques apprises sont déterministes et optimales.







Les récompenses pour Hangar :

```
Simulating Hangar Environment with Optimal Policy...
Environment reset: hangar. Starting state: (0, 0, 0)
Task completed with total reward: 37
```

Celles du Warehouse :

```
Simulating Warehouse Environment with Optimal Policy...
Environment reset: warehouse. Starting state: (0, 0, 0)
Task completed with total reward: 75
```

Pour Garage :

```
Simulating Garage Environment with Optimal Policy...
Environment reset: garage. Starting state: (2, 0, 0)
Task completed with total reward: 16
```

3.2 Single Agent Model-Free

3.2.1 Q-Learning

Implémentation et paramètres

Nous avons implémenté Q-learning avec une table Q pour estimer la valeur des actions dans chaque état. Q-learning est une méthode de Reinforcement Learning model-free qui utilise une table Q pour stocker les valeurs des paires état-action. La table Q est mise à jour en utilisant l'équation de Bellman.Résultats et analyse

```python
def q_learning(env, episodes=1000, max_steps=200, alpha=0.1, gamma=0.9, epsilon=0.2):
    q_table = np.zeros((*env.grid_size, 4))

    for episode in range(episodes):
        state = env.reset()
        total_reward = 0

        for _ in range(max_steps):
            x, y = state
            if np.random.rand() < epsilon:
                action = np.random.randint(4)  # Explore
            else:
                action = np.argmax(q_table[x, y])  # Exploit

            next_state, reward, done = env.step(action)
            total_reward += reward
            nx, ny = next_state
            q_table[x, y, action] += alpha * (
                reward + gamma * np.max(q_table[nx, ny]) - q_table[x, y, action]
            )
            state = next_state
            if done:
                break

        epsilon = max(0.01, epsilon * 0.995)
        print(f"[{env.env_type}] Episode {episode + 1}: Total Reward = {total_reward}")

    return q_table
```

- Résultats : Les agents apprennent à atteindre leurs objectifs après un certain nombre d'épisodes.
- Analyse : Q-learning est efficace mais peut nécessiter un grand nombre d'épisodes pour converger. La méthode d'exploration ε-greedy permet un bon équilibre entre exploration et exploitation.

3.2.2 Double Q-Learning

Motivation pour l'utilisation du Double Q-Learning

Double Q-learning est utilisé pour réduire le biais de surévaluation des valeurs Q. En utilisant deux tables Q, Double Q-learning permet de sélectionner et d'évaluer les actions de manière indépendante, réduisant ainsi le biais de surévaluation.

```python
def double_q_learning(env, episodes=1000, max_steps=200, alpha=0.1, gamma=0.9, epsilon=0.2):
    q_table_1 = np.zeros((*env.grid_size, 4))
    q_table_2 = np.zeros((*env.grid_size, 4))

    for episode in range(episodes):
        state = env.reset()
        total_reward = 0

        for _ in range(max_steps):
            x, y = state
            if np.random.rand() < epsilon:
                action = np.random.randint(4)
            else:
                combined_q = q_table_1[x, y] + q_table_2[x, y]
                action = np.argmax(combined_q)

            next_state, reward, done = env.step(action)
            total_reward += reward
            nx, ny = next_state

            if np.random.rand() < 0.5:
                best_action = np.argmax(q_table_1[nx, ny])
                q_table_1[x, y, action] += alpha * (
                    reward + gamma * q_table_2[nx, ny, best_action] - q_table_1[x, y, action]
                )
            else:
                best_action = np.argmax(q_table_2[nx, ny])
                q_table_2[x, y, action] += alpha * (
                    reward + gamma * q_table_1[nx, ny, best_action] - q_table_2[x, y, action]
                )
```

Comparaison avec le Q-Learning simple

Double Q-learning converge plus rapidement et de manière plus stable que le Q-learning simple. En utilisant deux tables Q, Double Q-learning permet de réduire le biais de surévaluation et d'améliorer la stabilité de l'apprentissage.

Résultats expérimentaux

- Résultats : Les agents apprennent des politiques optimales plus rapidement.
- Analyse : Double Q-learning est plus efficace que le Q-learning simple en termes de convergence et de stabilité.

3.2.3 Monte Carlo

Spécificités de l'implémentation

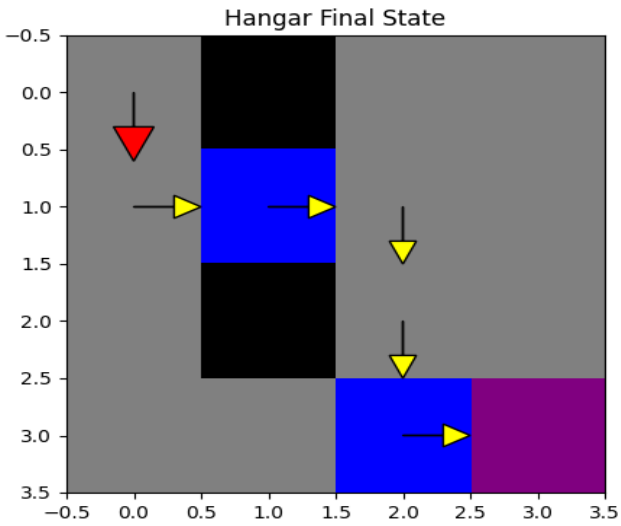
Nous avons implémenté Monte Carlo control pour mettre à jour les valeurs Q en utilisant les retours moyens des épisodes complets. Monte Carlo est une méthode de Reinforcement Learning model-free qui utilise des épisodes complets pour mettre à jour les valeurs Q.

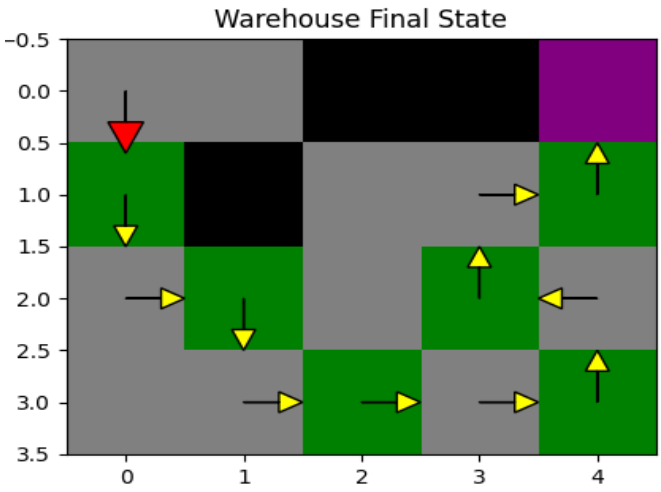
Avantages et limitations observés

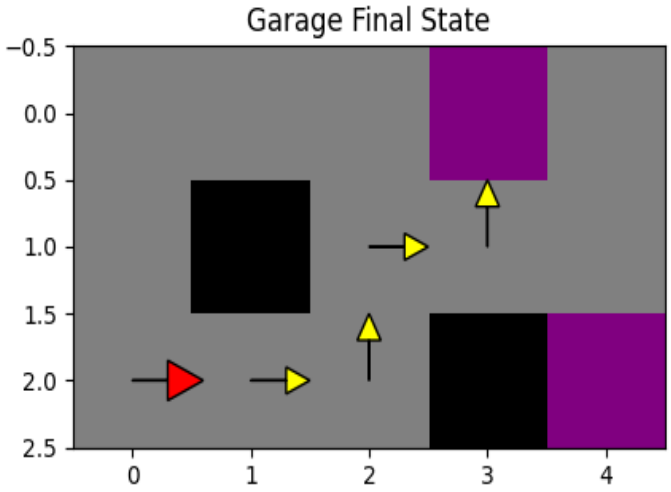
- Avantages : Monte Carlo est simple à implémenter et converge rapidement.

- Limitations : Nécessite des épisodes complets pour mettre à jour les valeurs Q, ce qui peut être coûteux en termes de temps et de ressources.

Résultats :







Les recompenses:

```
[garage] Episode 1000: Total Reward = 58.0
Simulating Hangar Environment...
Environment reset: hangar. Starting position: (0, 0)
[hangar] Collected material at (1, 1). Reward gained.
[hangar] Collected material at (3, 2). Reward gained.
[hangar] Task complete at step 6 with total reward 138.5.
[hangar] Final Total Reward: 138.5
Simulating Warehouse Environment...
Environment reset: warehouse. Starting position: (0, 0)
[warehouse] Cleaned dirty spot at (1, 0). Reward gained.
[warehouse] Cleaned dirty spot at (2, 1). Reward gained.
[warehouse] Cleaned dirty spot at (3, 2). Reward gained.
[warehouse] Cleaned dirty spot at (3, 4). Reward gained.
[warehouse] Cleaned dirty spot at (2, 3). Reward gained.
[warehouse] Cleaned dirty spot at (1, 4). Reward gained.
[warehouse] Task complete at step 12 with total reward 217.5.
[warehouse] Final Total Reward: 217.5
Simulating Garage Environment...
Environment reset: garage. Starting position: (2, 0)
[garage] Task complete at step 5 with total reward 58.0.
[garage] Final Total Reward: 58.0
```

3.3 Multi-Agent Model-Based

Adaptation de Value Iteration pour multiple agents

Nous avons adapté Value Iteration pour deux agents qui doivent coordonner leurs actions pour maximiser la récompense cumulée. La méthode de Value Iteration est étendue pour gérer les états et les actions conjointes des deux agents.

Stratégies de coordination entre agents

Les agents coordonnent leurs actions en utilisant des politiques jointes optimales. Les politiques sont dérivées de la fonction de valeur optimale calculée par Value Iteration.

Analyse des performances

- Performances : Les agents atteignent leurs objectifs de manière coordonnée.
- Analyse : La coordination entre les agents améliore les performances globales et permet d'atteindre des objectifs qui seraient impossibles à atteindre individuellement.

Récompenses cumulées Totale du Hangar :

```
Stopped after 6 steps. Total reward = 34.0
Converged after 16 iterations.
```

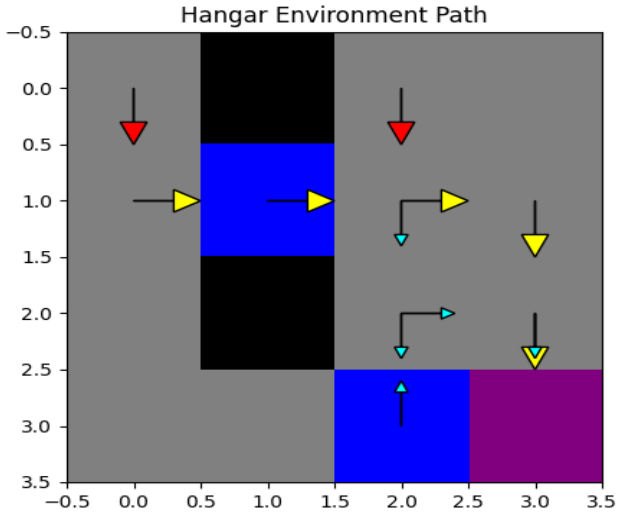
Celle du Warehouse :

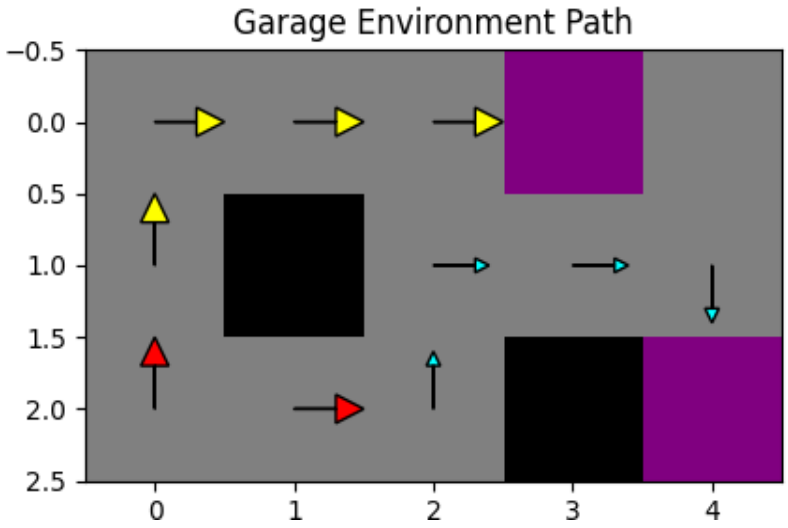
```
Stopped after 9 steps. Total reward = 96.0
Converged after 7 iterations.
```

Et enfin celle du garage :

```
Stopped after 5 steps. Total reward = 95.0
```

Résultats





4. Analyse Comparative

Comparaison des performances des différentes approches

- Forces et faiblesses de chaque méthode :
    - Value Iteration : Optimalité garantie mais nécessite une connaissance complète de l'environnement.
    - Q-learning : Flexibilité mais peut nécessiter un grand nombre d'épisodes pour converger.
    - Double Q-learning : Réduction du biais de surévaluation mais complexité accrue.
    - Monte Carlo : Simplicité mais nécessite des épisodes complets.
    - Multi-Agent Value Iteration : Coordination optimale mais complexité computationnelle élevée.

5. Défis et Solutions

Problèmes rencontrés dans l'implémentation

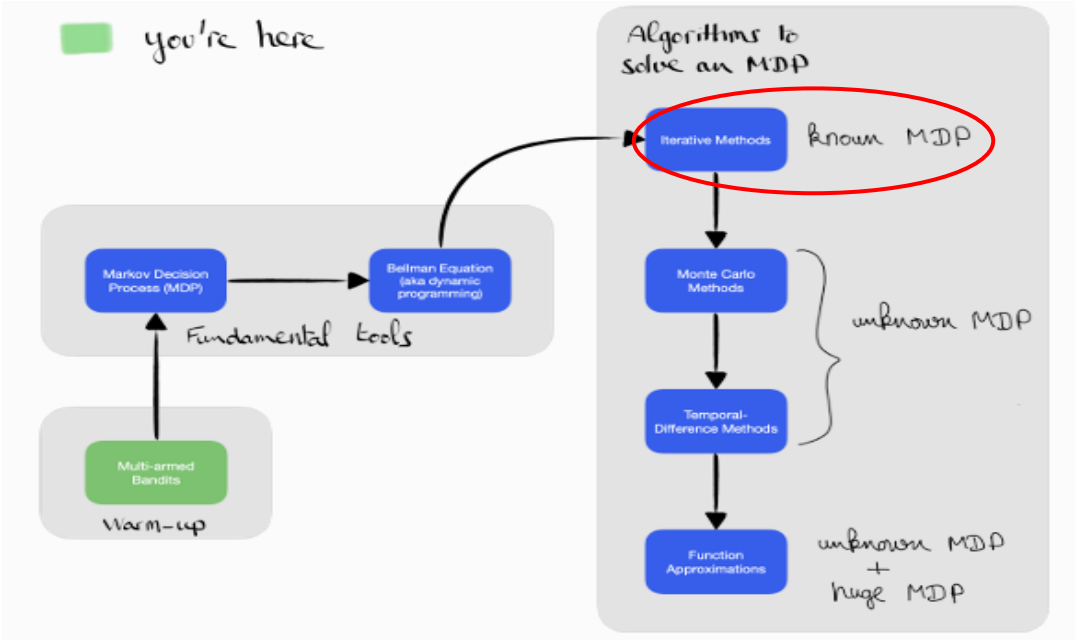
- Complexité computationnelle : Value Iteration et Multi-Agent Value Iteration sont computationnellement coûteux.
- Convergence lente : Q-learning peut nécessiter un grand nombre d'épisodes pour converger.

Solutions proposées

- Optimisation des algorithmes : Utilisation de techniques de parallélisation et de réduction de la dimensionnalité.
- Amélioration de l'exploration : Utilisation de stratégies d'exploration plus efficaces pour Q-learning.

Limitations actuelles

- Connaissance complète de l'environnement : Value Iteration nécessite une connaissance complète de l'environnement.
- Complexité des environnements dynamiques : Les méthodes model-free peuvent être moins efficaces dans des environnements très dynamiques.



6. Améliorations Possibles

Suggestions d'optimisation

- Utilisation de réseaux de neurones : Implémentation de Deep Q-Networks (DQN) pour améliorer les performances dans des environnements complexes.
- Techniques de réduction de dimension : Utilisation de techniques de réduction de dimension pour accélérer la convergence.

Pistes d'exploration futures

- Reinforcement Learning multi-agent avancé : Exploration de techniques de MARL plus avancées, telles que les réseaux de communication entre agents.
- Environnements plus réalistes : Test des algorithmes dans des environnements plus réalistes et dynamiques.

7. Conclusion

Synthèse des résultats

Dans ce projet, nous avons exploré différentes approches de Reinforcement Learning pour résoudre un problème de nettoyage d'un environnement dangereux pour les humains. Chaque

approche a ses avantages et inconvénients, et le choix de la méthode dépend des contraintes spécifiques de l'environnement et des ressources disponibles.

Leçons apprises

- Importance de la coordination : La coordination entre les agents améliore les performances globales.
- Flexibilité des méthodes model-free : Les méthodes model-free sont flexibles et adaptables à des environnements dynamiques.

Perspectives

- Amélioration des algorithmes : Continuer à optimiser les algorithmes pour améliorer les performances et la stabilité.
- Exploration de nouvelles techniques : Explorer de nouvelles techniques de Reinforcement Learning pour des environnements plus complexes et dynamiques.