Apprentissage par renforcement

STEYER Julien N°SCEI : 10402

Sommaire

- Présentation générale

- Objectif

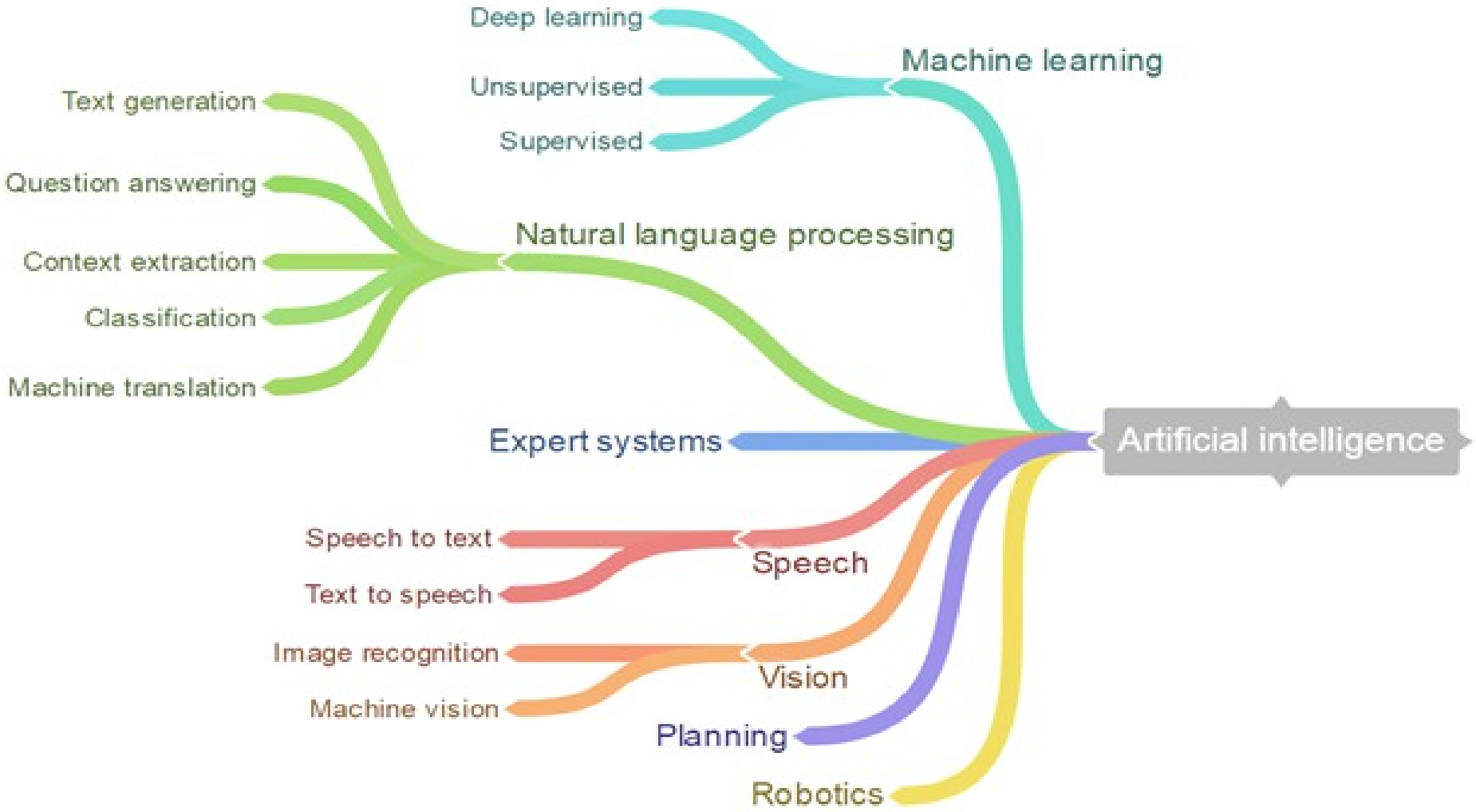
- L'algorithme du Q-learning

- Application simple : Le labyrinthe

- Application : Pendule inversé
  - Modèle de l'environnement
  - Résultats
- Limite et conclusion

Présentation générale

Vue d'ensemble



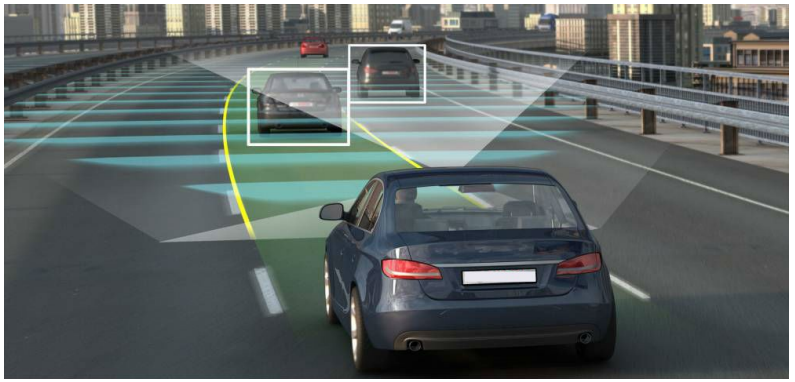
Définition : Intelligence artificielle

L'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence

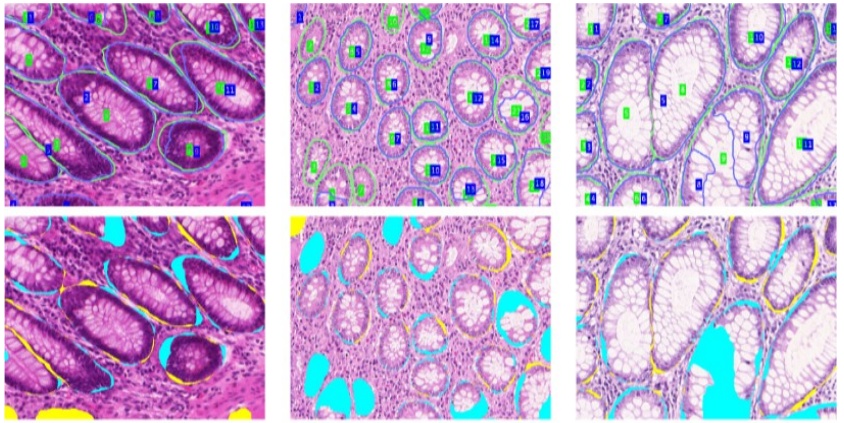
Pourquoi ?

Résoudre des problèmes où la programmation classique est insuffisante

Reconnaissance de caractère



Voiture autonome



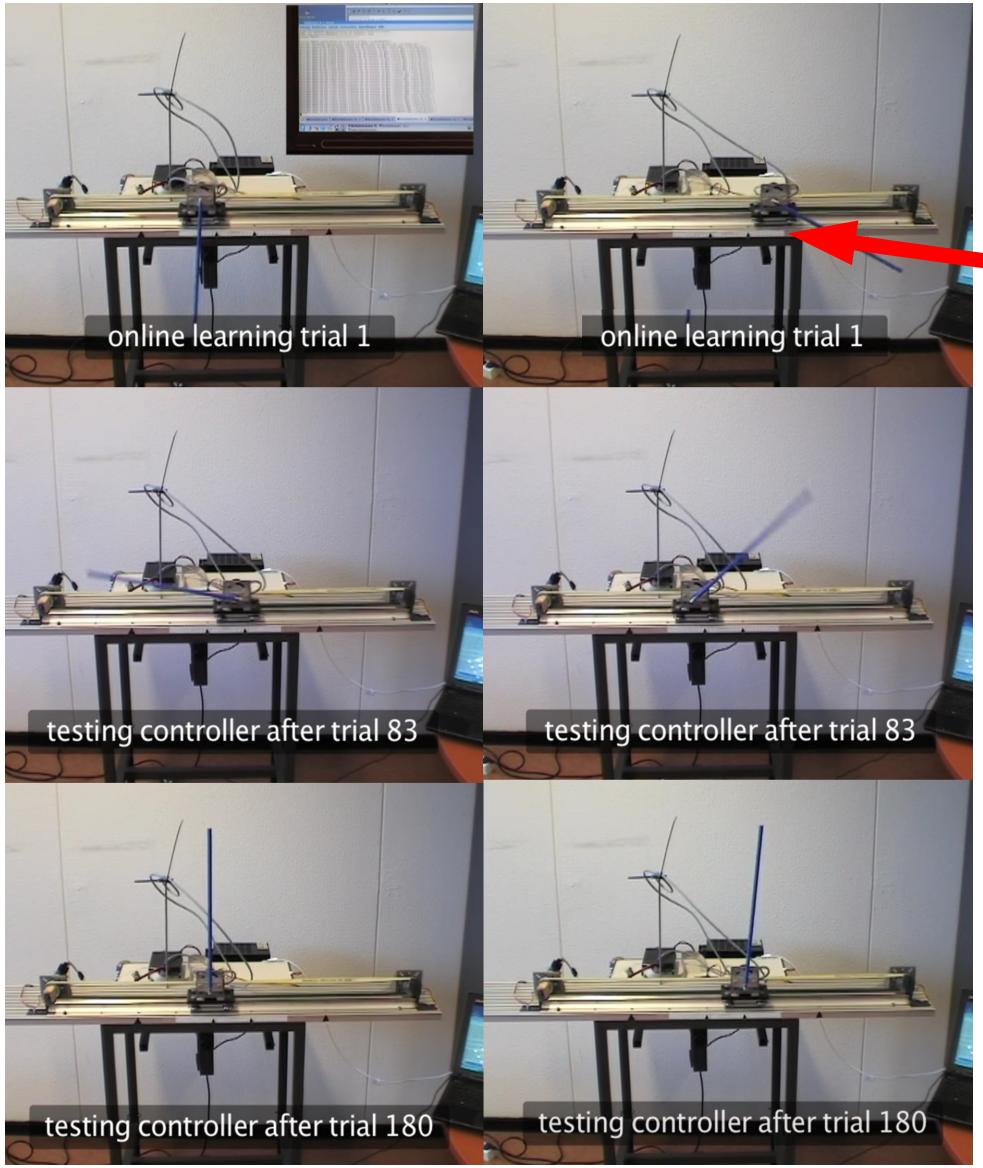
(d) benign (e) benign (f) malignant

Détection de cancer

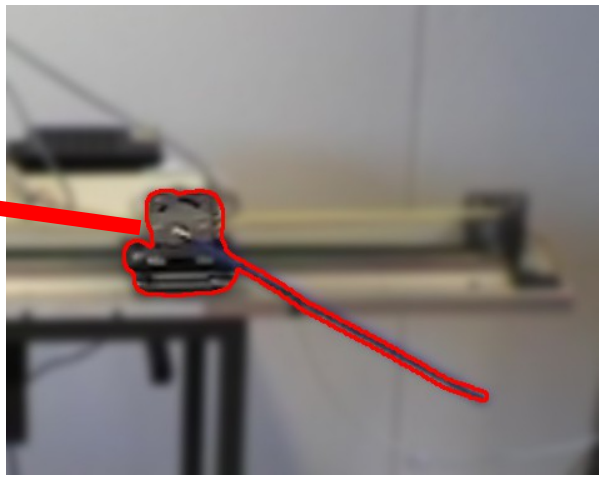
Optimisation procédé de production

L'apprentissage par renforcement

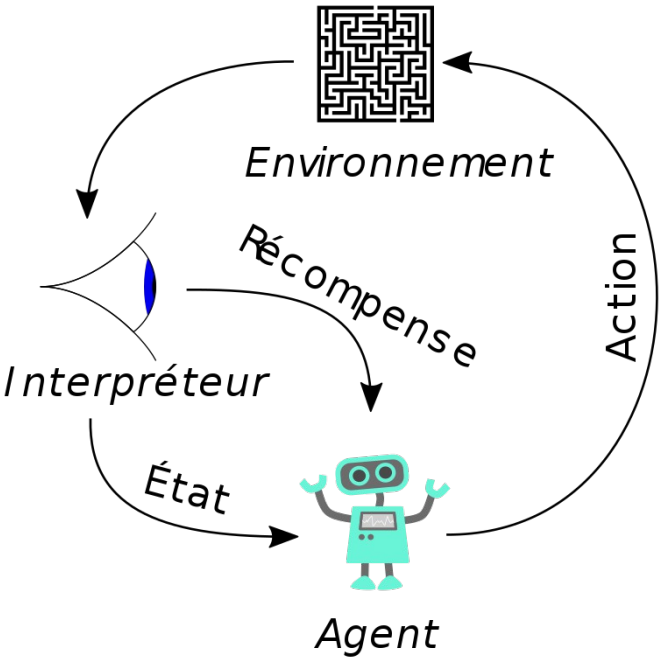
Plusieurs algorithmes basés sur un principe commun



« Cart-Pole balancing » de Dr.Martin Riedmiller



Pendule inversé



Objectif

Problématique retenue : Comprendre et utiliser l'apprentissage par renforcement

Objectif : Reproduire l'exemple du pendule inversé

Sommaire

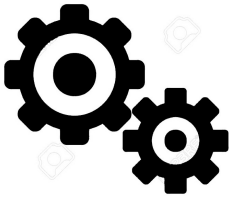
- Présentation générale

- Objectif

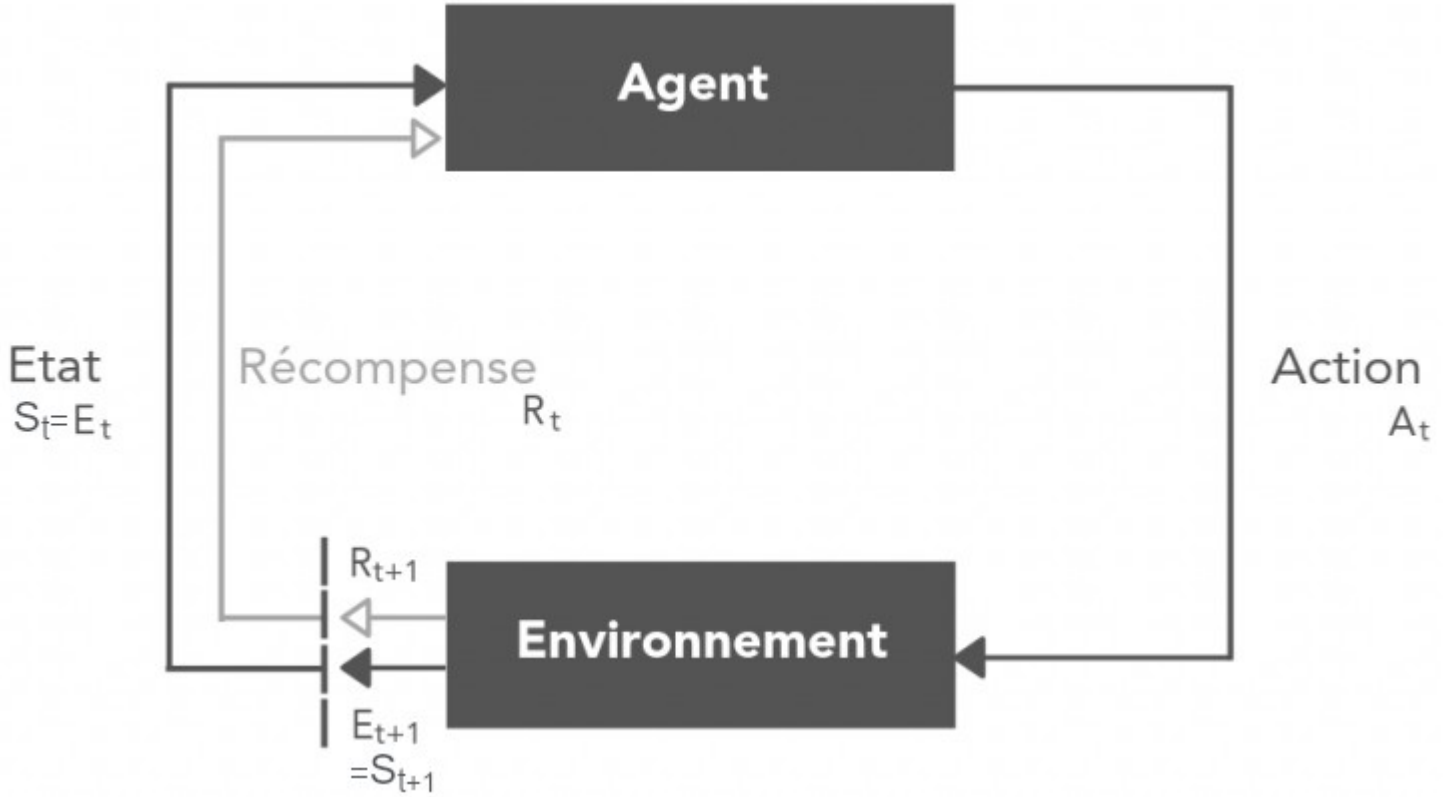
- L'algorithme du Q-learning

- Application simple : Le labyrinthe

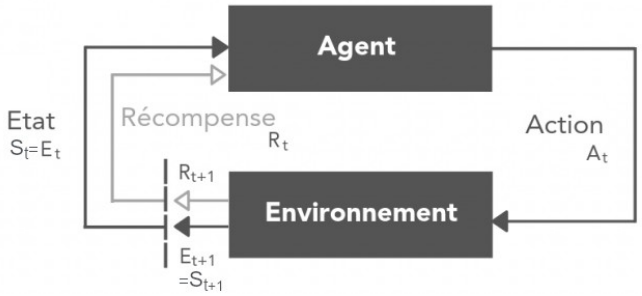
- Application : Pendule inversé
  - Modèle de l'environnement
  - Résultats
- Limite et conclusion



L'algorithme du Q-learning



Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

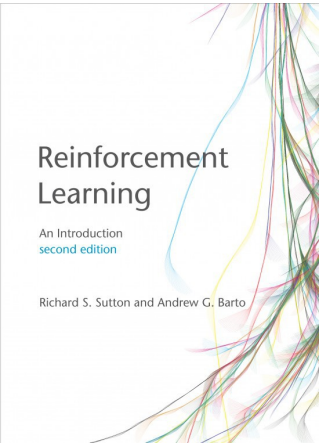


Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	327	0	0	0	0	0	0
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603



[1] R.S. SUTTON, A. BARTO : Reinforcement Learning: An Introduction, 1998, MIT Press

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Équation de Bellman

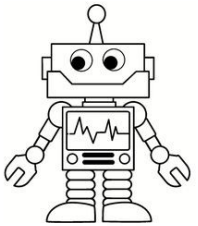
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

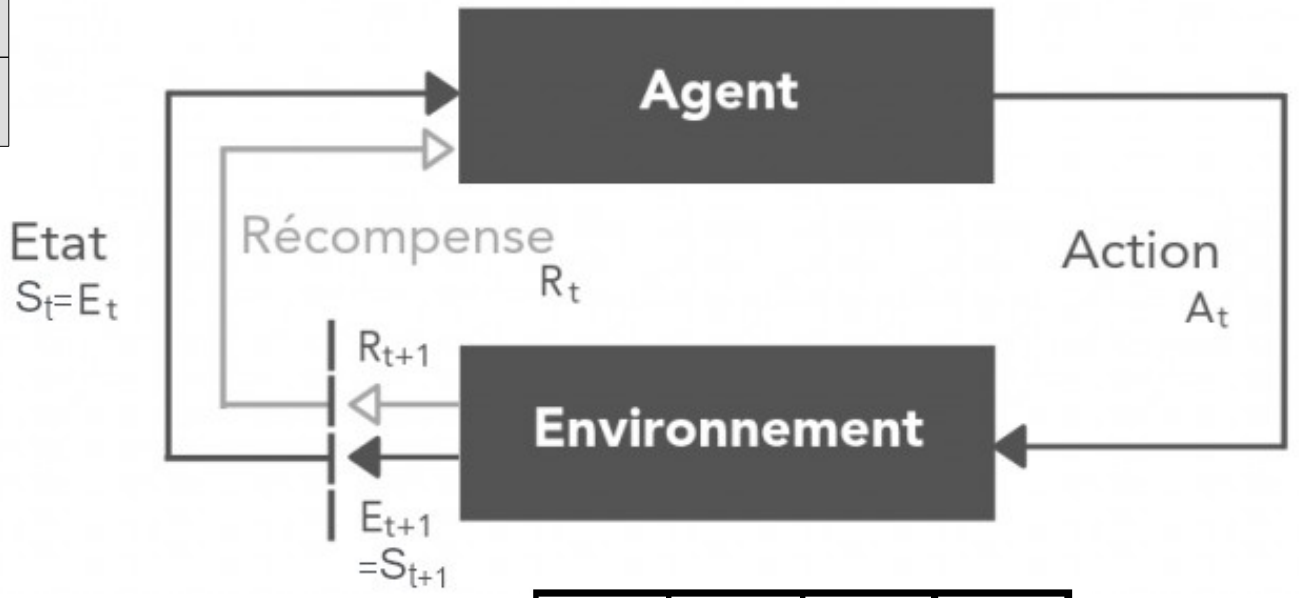
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

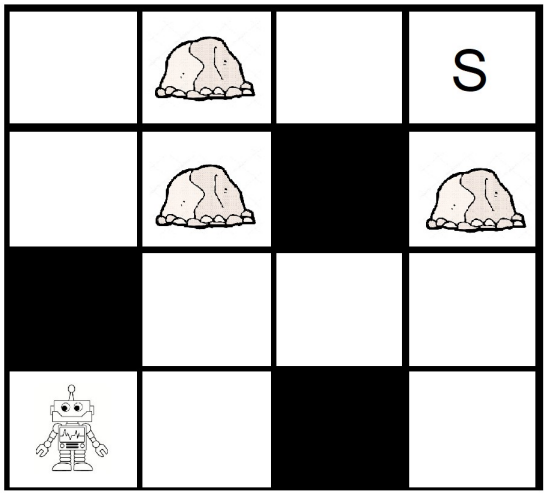
Application simple : Le labyrinthe

Élément	R_t
Vide	-1
Rocher	-10
Sortie	+10
Butée	-5



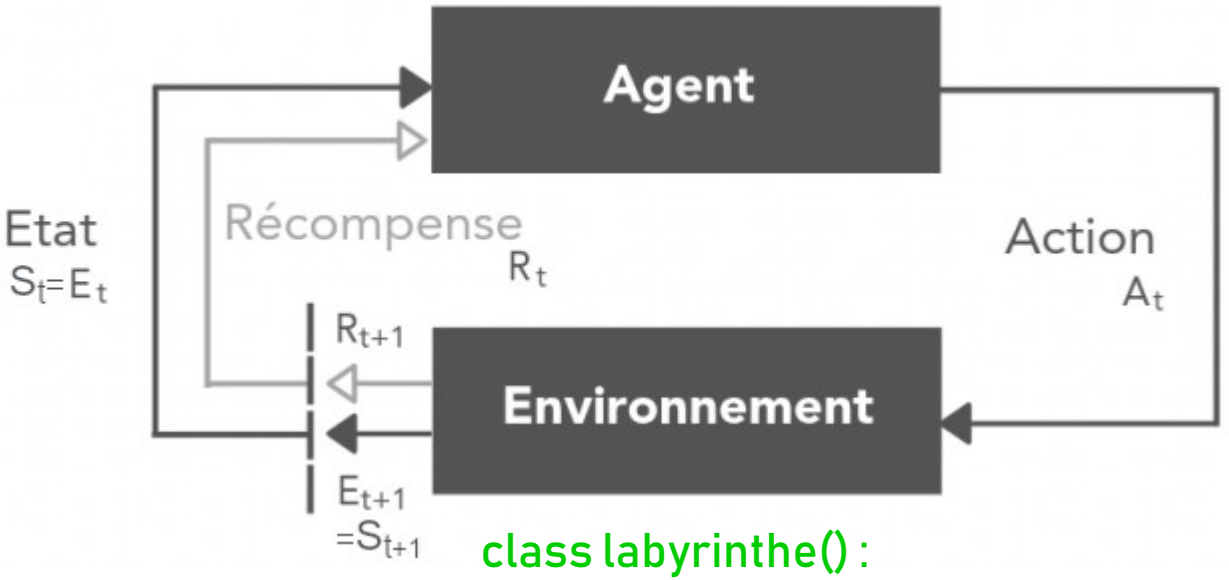


E_t ou S_t
0
1
...
15

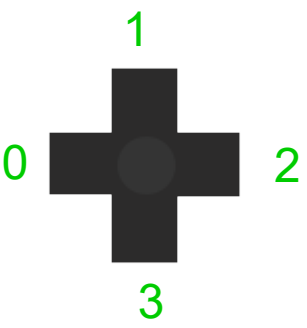


#

Variables :
s et r

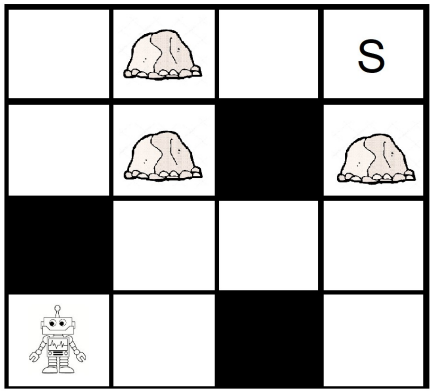


Variable d'action :
a



class labyrinthe() :

```
['-','O','-','S']
['-','O','x','O']
['x','-','-','-']
['#','-','x','-']
```



Application simple : Le labyrinthe

L’environnement

L'agent/ Le Q-learning

Action A

Environnement
class labyrinthe()

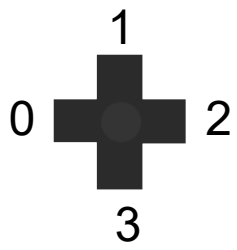
```
self.position=[1,1]
self.fin=False

self.obtenir_id(x,y)
    #retoune 5 pour [1,1]
self.deplacer(3)
```

État(=State) S

Récompense R

PPO
Programmation Orienté Objet

Action A



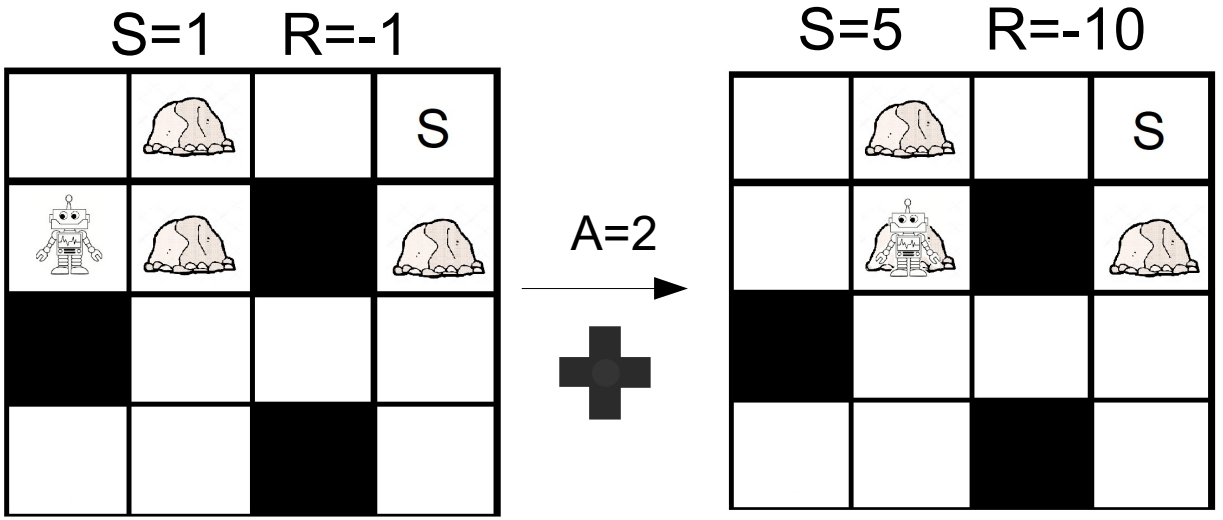
État(=State) S

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Récompense R

Élément	r
Vide	-1
Rocher	-10
Sortie	+10
Butée	-5

Exemple :



Application simple : Le labyrinthe

L’environnement

L'agent/ Le Q-learning

Le fonctionnement de l'algorithme

Agent

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,394	0,43	0,378	0,368
	...				
	15	0,351	0,157	0,954	0,545

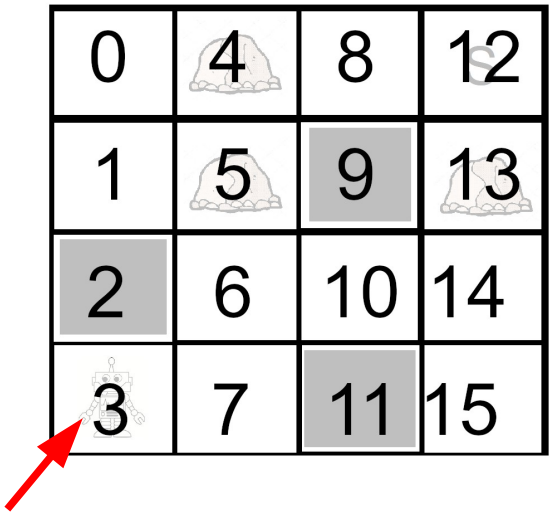
Le fonctionnement de l'algorithme

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,394	0,43	0,378	0,368
	...				
	15	0,351	0,157	0,954	0,545



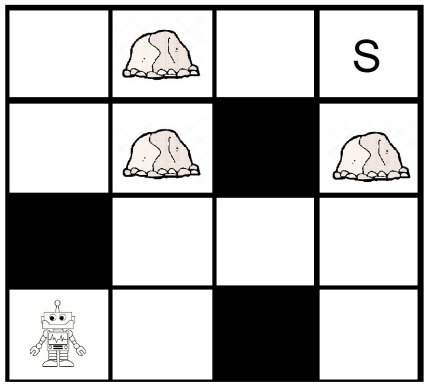
S=3

Le fonctionnement de l'algorithme

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal



S=3

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,394	0,43	0,378	0,368
	...				
	15	0,351	0,157	0,954	0,545

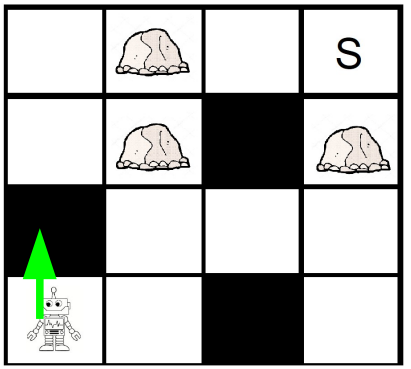
Le fonctionnement de l'algorithme

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

S=3



Le fonctionnement de l'algorithme

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

Variables de l'environnement

A_t=1 R_{t+1}=-5
S_t=3 S_{t+1}=3

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,378	0,43	0,394	0,368
	...				
	15	0,351	0,157	0,954	0,545

Paramètre de l'algorithme

α (Learning_rate) = 0.7
γ= 0.99

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Q(S=3,A=1) = 0,43 +0,7*[-5 + 0,99 * 0,43 - 0,43] =-9,85

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,378	-9,85	0,394	0,368
	...				
	15	0,351	0,157	0,954	0,545

Le fonctionnement de l'algorithme

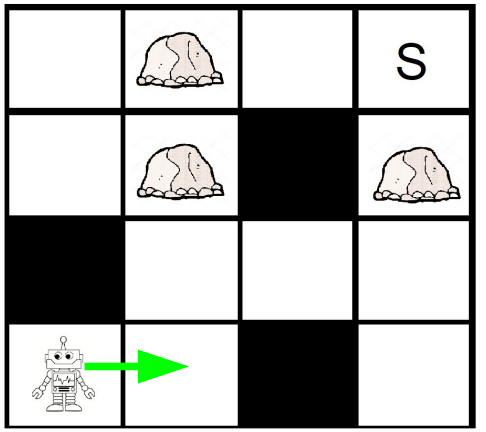
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

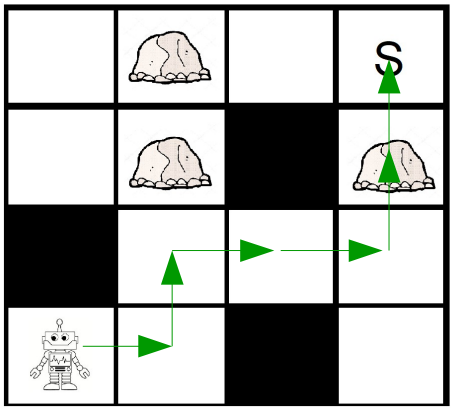
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
  → Choose A from S using policy derived from Q (e.g., ε-greedy)
  → Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

		Actions			
		0	1	2	3
Etats	0	0,846	0,657	0,214	0,981
	1	0,624	0,321	0,247	0,458
	2	0,148	0,854	0,312	0,174
	3	0,378	-9,85	0,394	0,368
	...				
	15	0,351	0,157	0,954	0,545

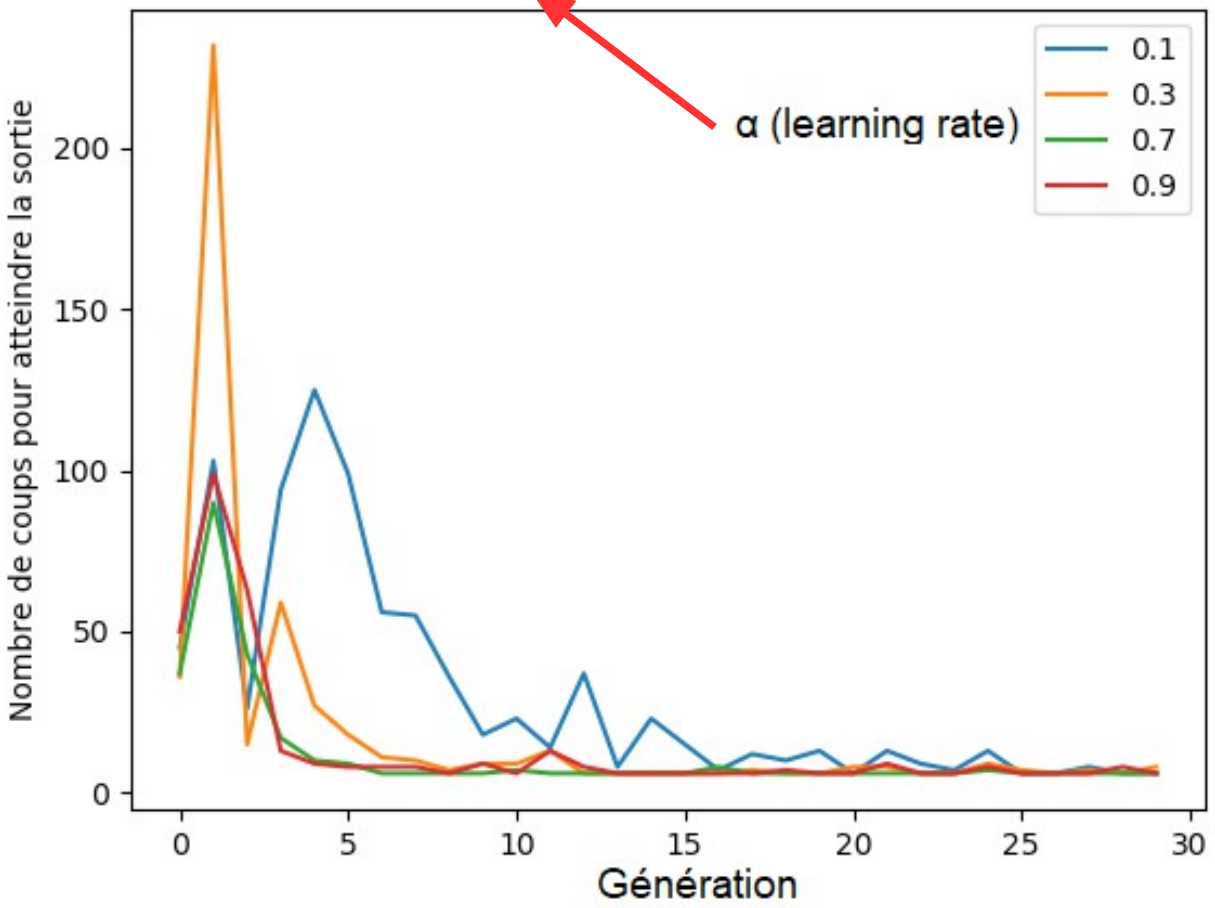
S=3





Conclusion :
L'agent atteint la sortie en un minimum de coup (6 coups)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$



Sommaire

- Présentation générale

- Objectif

- L'algorithme du Q-learning

- Application simple : Le labyrinthe

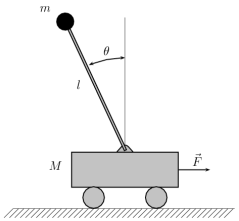
- Application : Pendule inversé
  - Modèle de l'environnement
  - Résultats
- Limite et conclusion

Application : Pendule inversé

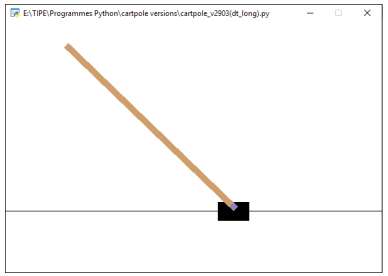
Objectif : Reproduire l'exemple du pendule inversé

Démarche pour accomplir l'objectif :

Observation et affinage du modèle

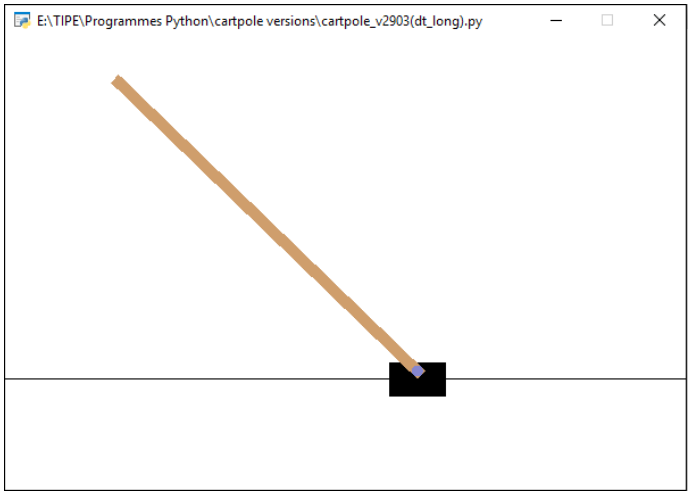


Modèle dynamique



Apprentissage sur le système simulé

Système réel



Système simulé

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

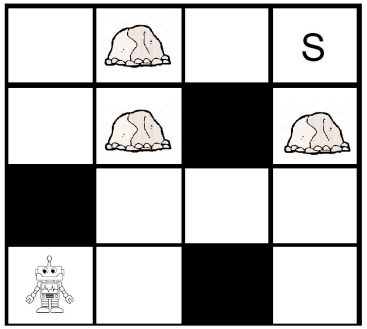
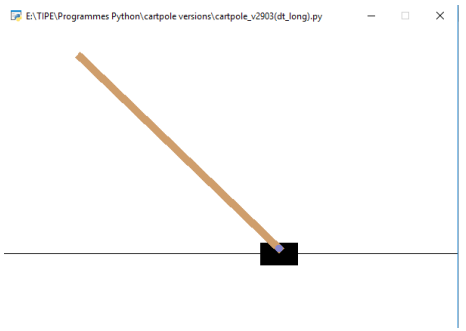
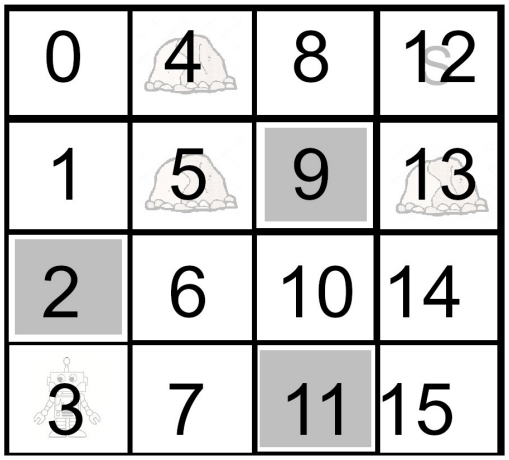
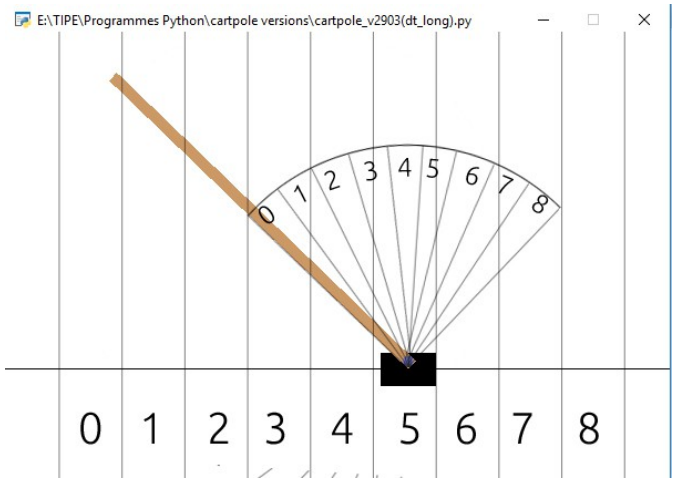
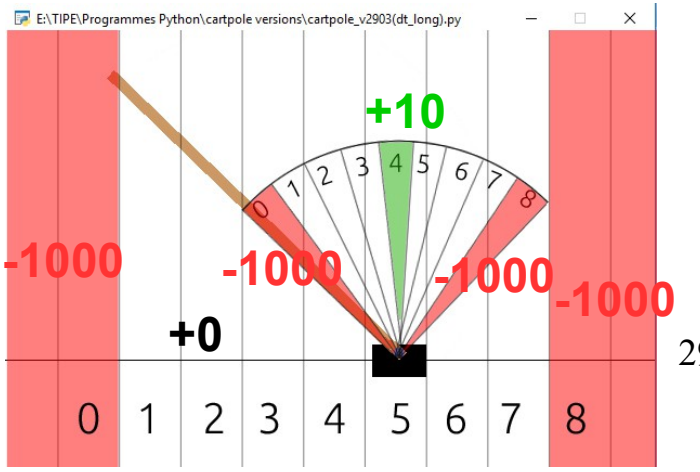
Q-Table

Système réel

Pilotage :

MATLAB

NATIONAL INSTRUMENTS
LabVIEW

Environnement		
États		
Actions		
Récompenses	Élément / R Vide / -1 Rocher / -10 Sortie / +10 Butée / -5	

Application : Pendule inversé

Modèle de l'environnement

Résultats

Améliorer les performances

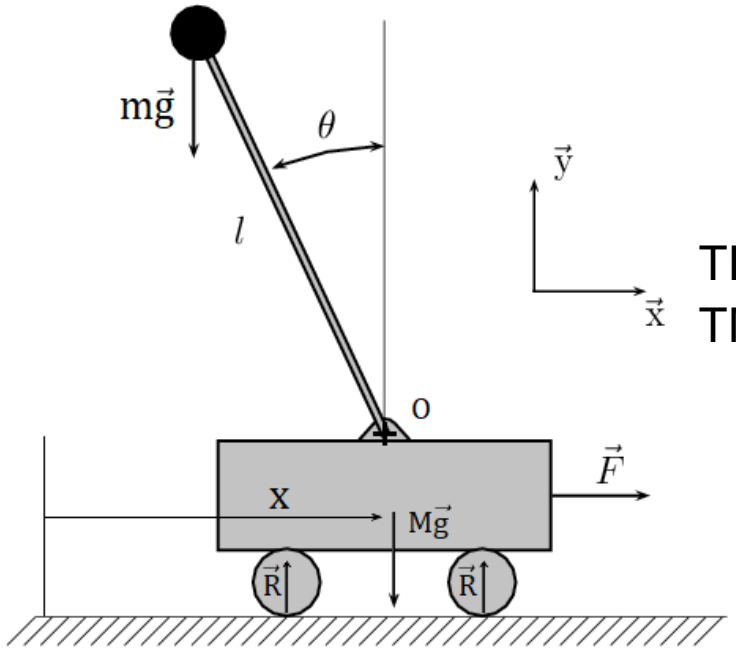
Environnement

Modèle mécanique

On isole {chariot+barre}

Bilan des Actions Mécaniques Extérieures :

Force de traction F	Poids de la barre
Poids du chariot	Réaction du support



On applique le Principe Fondamental de la Dynamique en O

TRD sur x
TMD sur z

$$\begin{cases} (M+m)\ddot{x} = F + ml\ddot{\theta}cos(\theta) - ml\dot{\theta}^2 sin(\theta) \\ ml^2\ddot{\theta} = mgsin(\theta)l + mlcos(\theta)\ddot{x} \end{cases}$$

On découple les variables

$$temp = \frac{F - ml\dot{\theta}^2 sin(\theta)}{M+m}$$

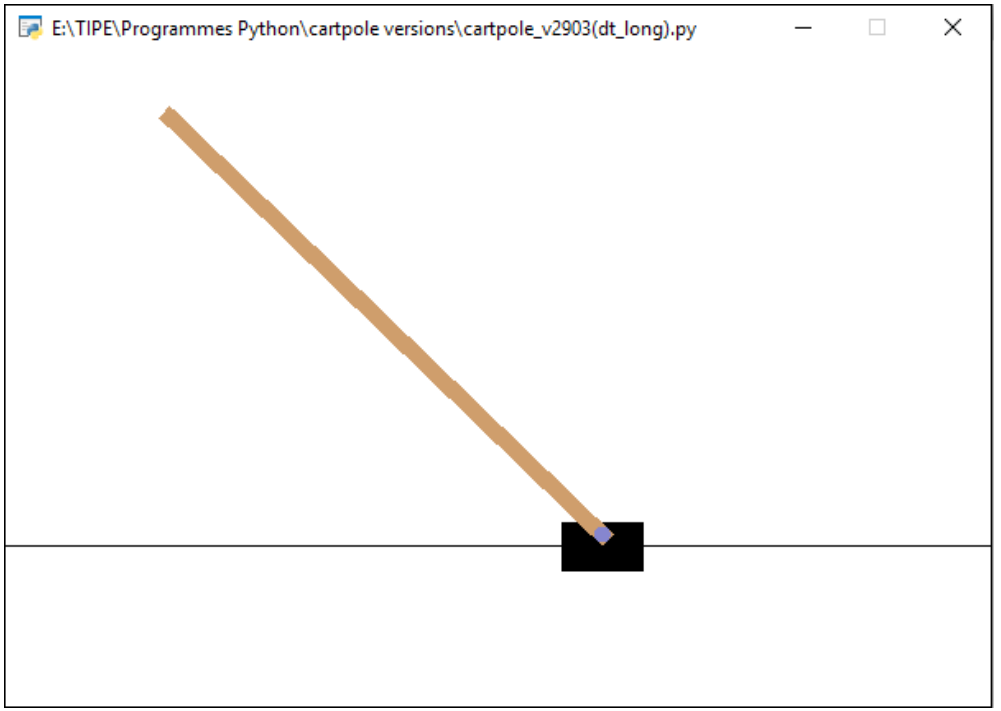
$$\ddot{\theta} = \frac{cos(\theta).temp + gsin(\theta)}{l(1 - \frac{mcos(\theta)^2)}{M+m})}$$

$$\ddot{x} = temp + \frac{ml\ddot{\theta}sin(\theta)}{M+m}$$

Hypothèse :
Liaisons parfaites

Environnement

Système simulé

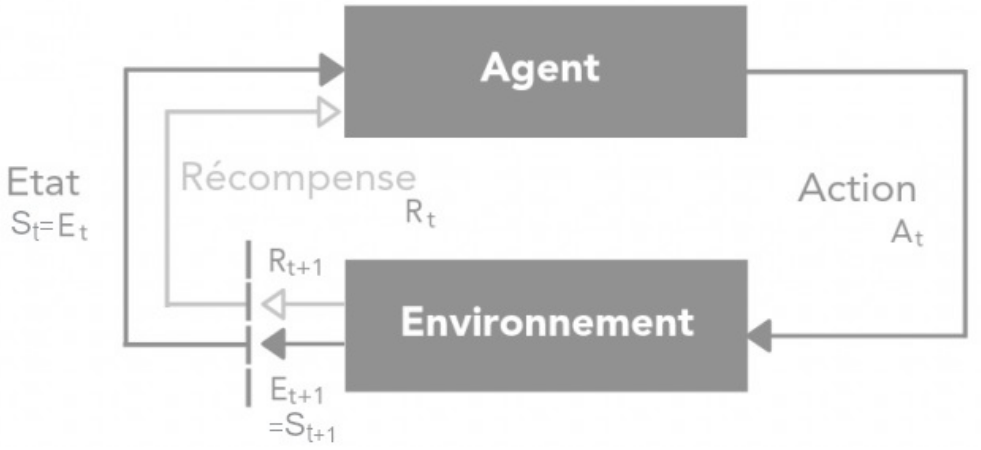


Fenêtre graphique de la bibliothèque pré-programmé

<u>Résolution numérique :</u>
Méthode d'Euler

```
x_dot = x_dot + self.tau * xacc
x  = x + self.tau * x_dot
theta_dot = theta_dot + self.tau * thetaacc
theta = theta + self.tau * theta_dot
```

Agent



Identique à l'application simple du labyrinthe

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

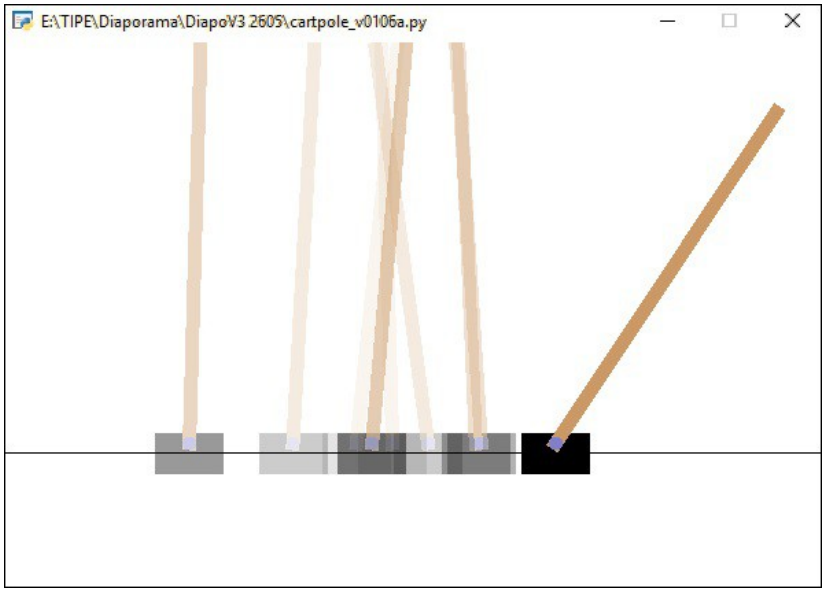
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

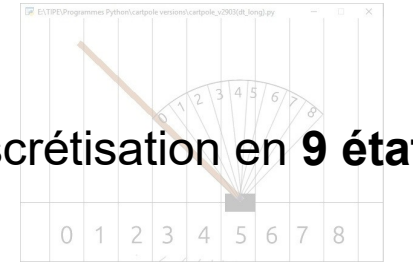
Application : Pendule inversé

Modèle de l'environnement

Résultats

Améliorer les performances



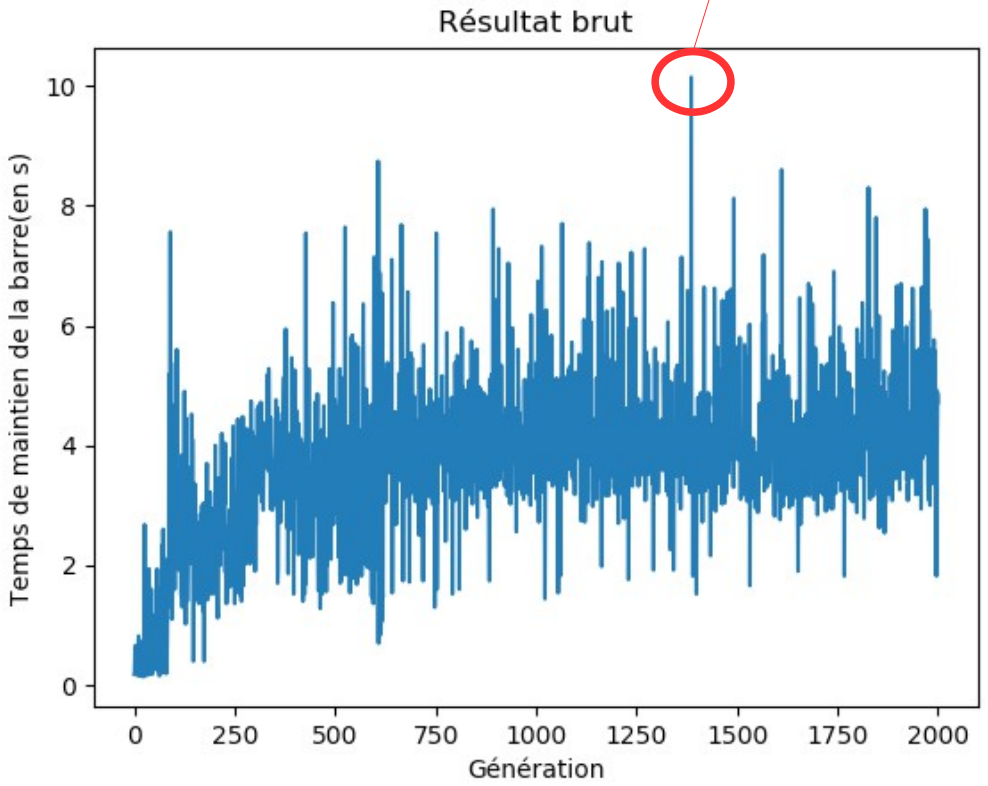


Discrétisation en **9 états**

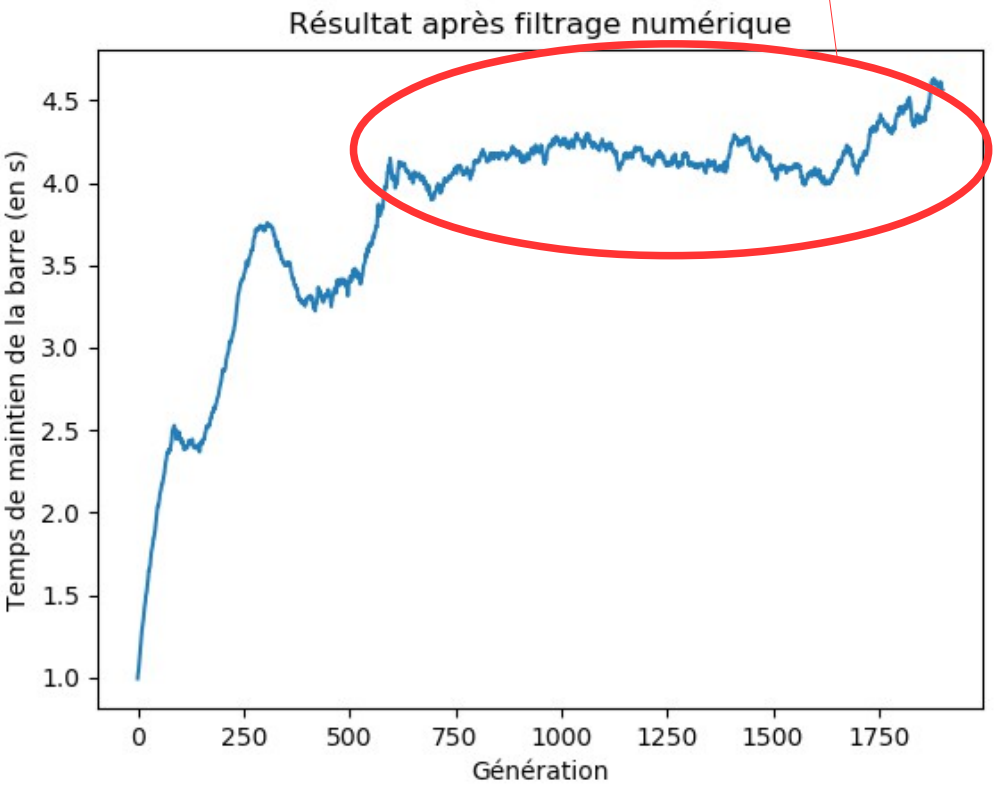
Paramètre de l'essai:

Apprentissage sur **2000 générations**

Performance maximale : 10s



Performance moyenne : 2s



Conclusion : Comportement voulu mais mauvaise performance

Application : Pendule inversé

Modèle de l'environnement

Résultats

Améliorer les performances

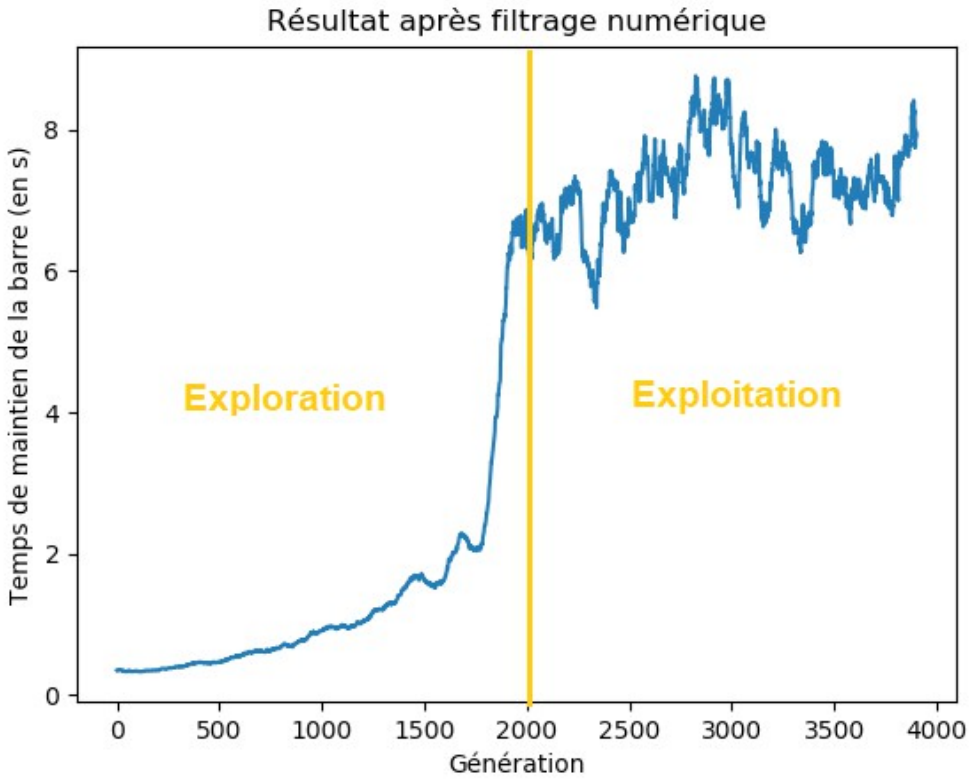
Dilemme Exploration/Exploitation Fonction d'exploration

Problème : L'agent tend à reproduire un comportement passé : il ne peut pas découvrir de nouvelles manières de résoudre le problème

```
## START Choose A from S using
a=np.argmax(Q[s[0],s[1],s[2],s[3],:])
## END Choose A from S using
```

→

```
## START Choose A from S using
import random as r
fct_exploration=lambda i: -1/(iteration*0.5)*i+1
if r.random()<fct_exploration(i):           #Choisir action aléatoire
    a=r.randint(0,n_actions-1)
else:                                       #Choisir action optimisée
    a=np.argmax(Q[s[0],s[1],s[2],s[3],:])
## END Choose A from S using
```



Conclusion : Amélioration observée mais les performances sont encore trop faibles

Problème : L'agent a un comportement trop grossier dû au découpage trop faible

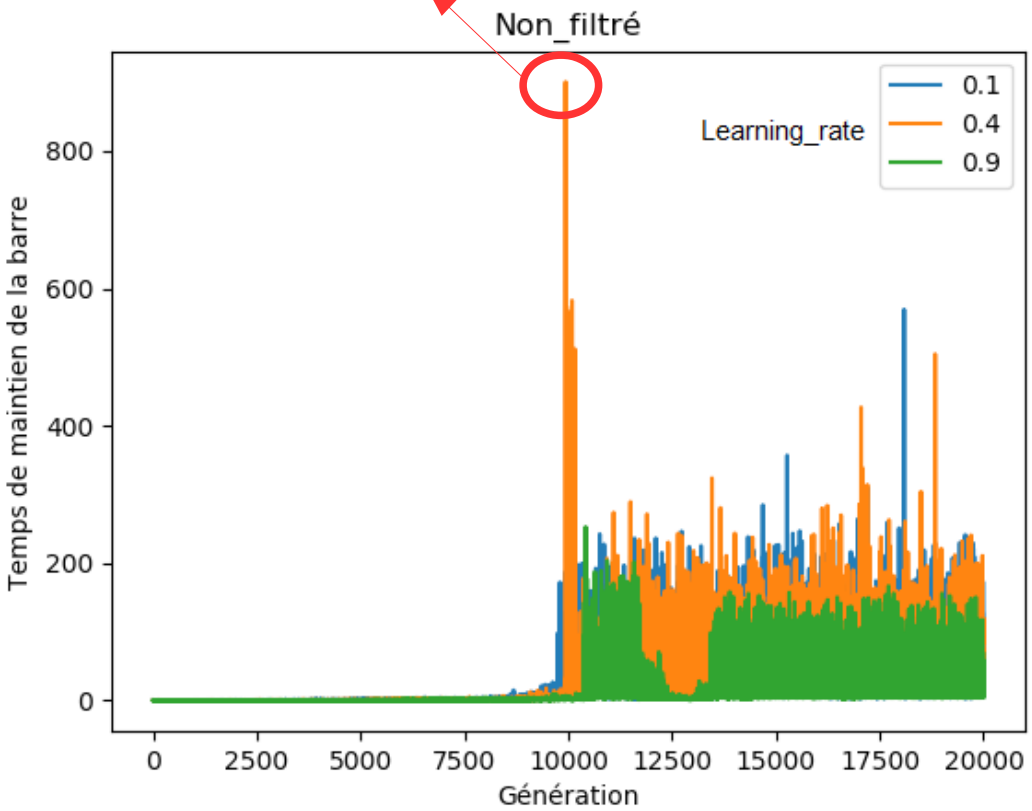
Paramètre de l'essai:

Discrétisation en **41 états**

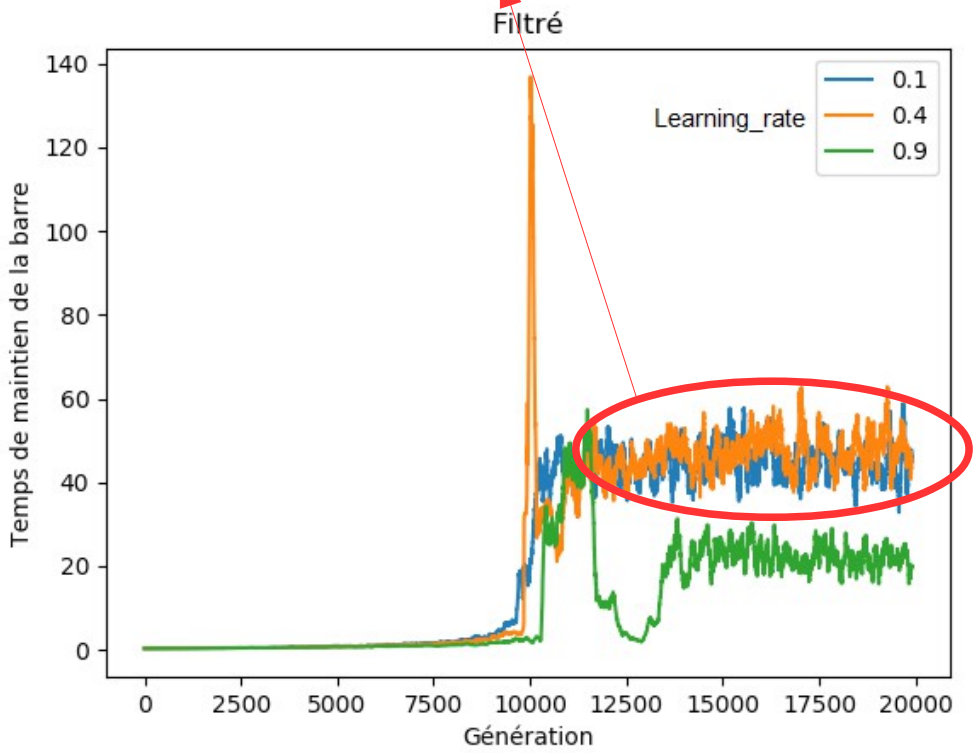
Apprentissage sur **20000 générations**

Observation secondaire : Influence du learning_rate

Performance maximale : 13 minutes



Performance moyenne : 50s



Conclusion : Comportement voulu et performance satisfaisante

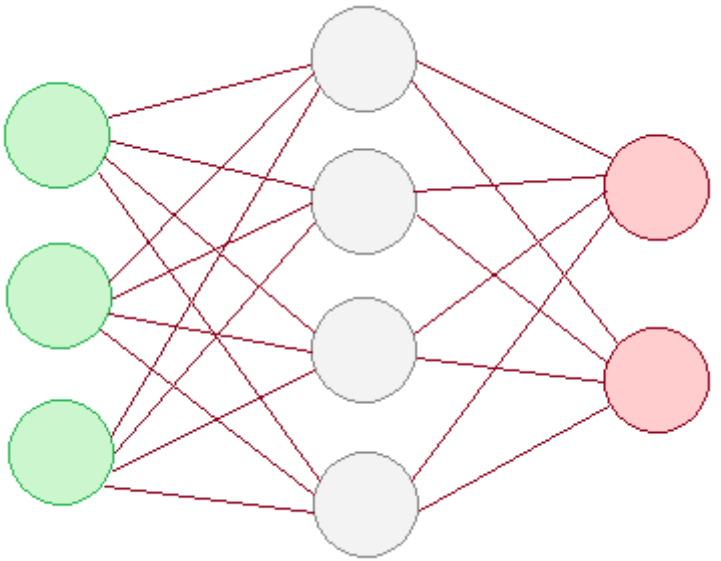
Limites

Augmenter le nombre d'états possibles => Amélioration des performances

<u>Problème :</u> Temps de calcul très important et limites des capacités machines atteintes

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Q-table

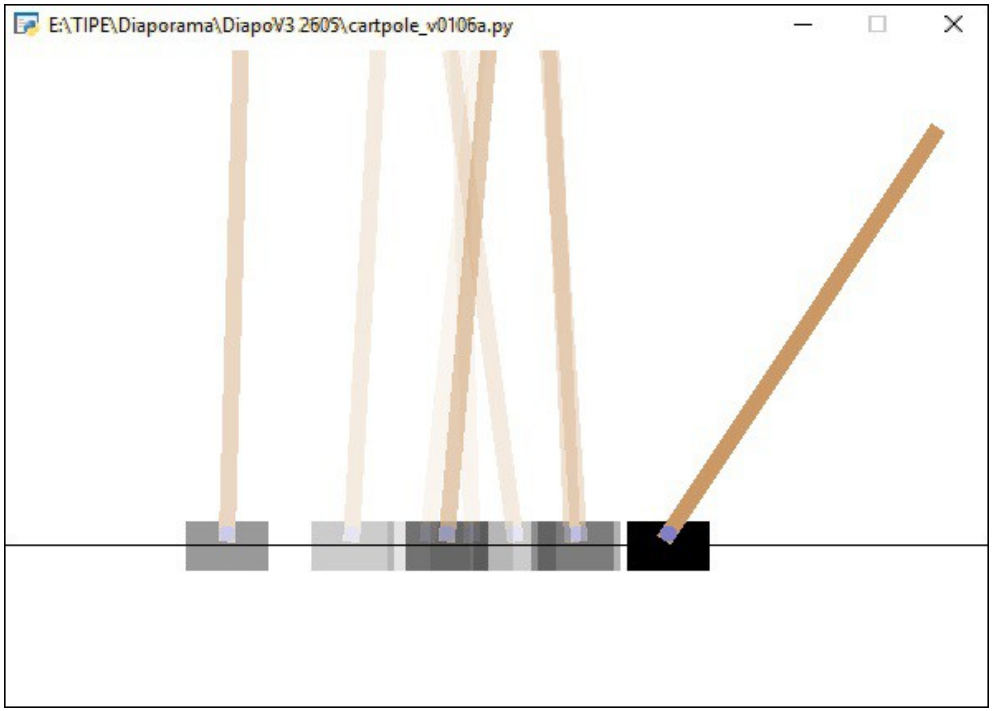


Réseaux de neurones

<u>Solution/ Ouverture :</u> Se libérer des contraintes de discrétisations pour traiter des états continus par les réseaux de neurones

Conclusion

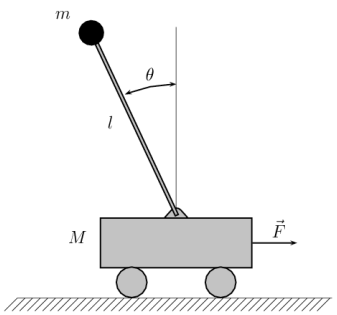
Problématique retenue : Comprendre et utiliser l'apprentissage par renforcement



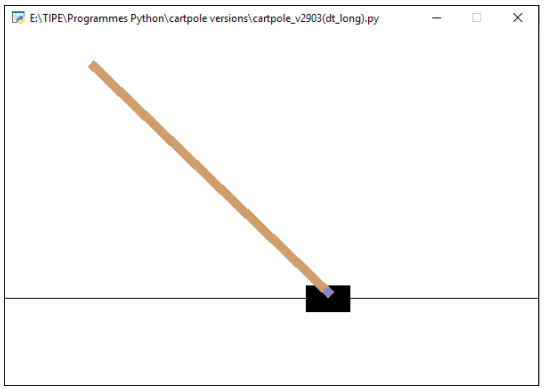
Conclusion

Problématique retenue : Comprendre et utiliser l'apprentissage par renforcement

Observation et affinage du modèle



Modèle dynamique



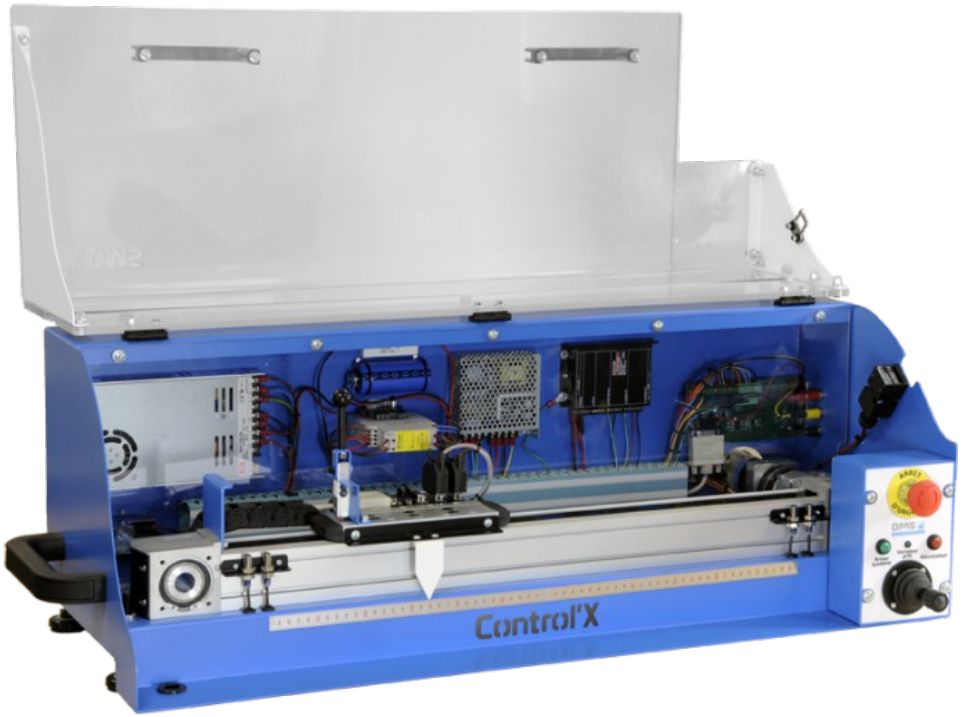
Apprentissage sur le système simulé

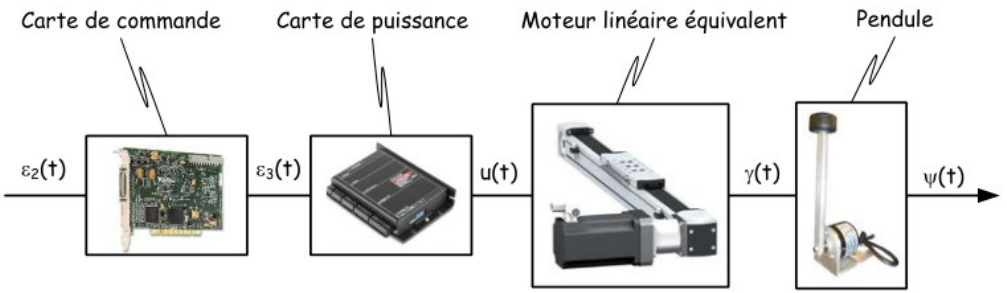
Implantation sur le système réel

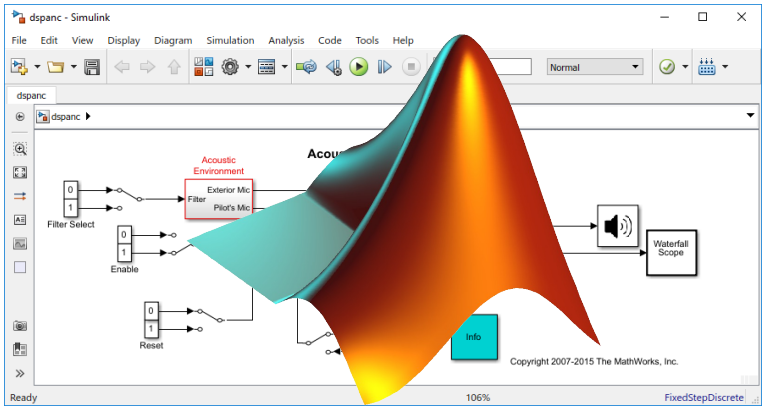
(Deuxième conclusion)

Annexe

Système réel



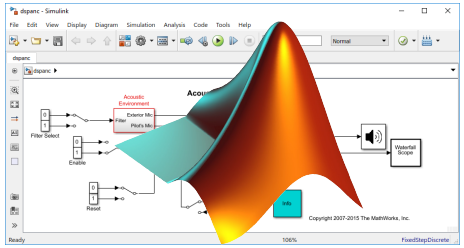




Pilotage par MATLAB

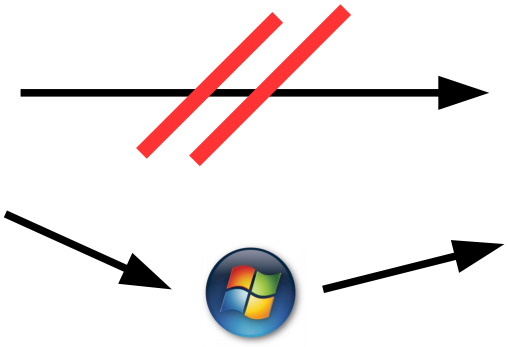
Système réel

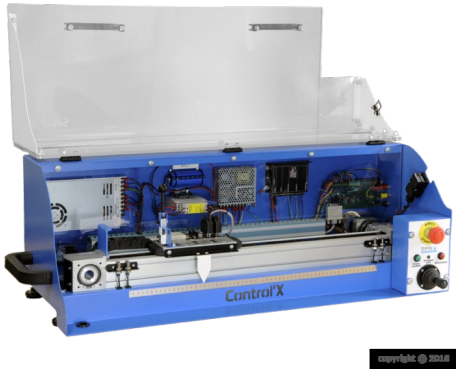
Problème 1 : Problème de communication entre le logiciel et le système



Pilotage par MATLAB

PC Propriétaire vérolé

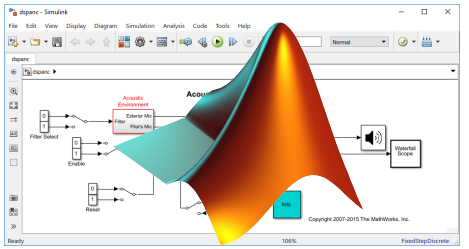




Solution : Réinstallation du système d'exploitation

Problème 2 : Communication du module impossible

Récupération de la position angulaire du chariot impossible

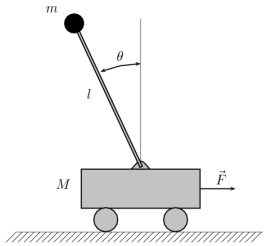


Pilotage par MATLAB

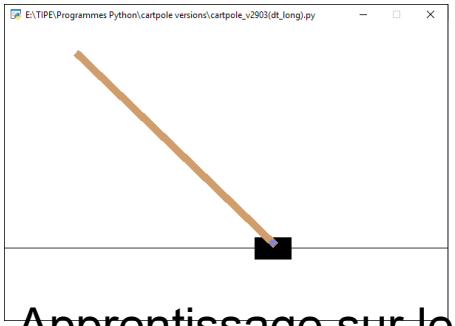
Solution : Non trouvée

Système réel

Observation et affinage du modèle



Modèle dynamique

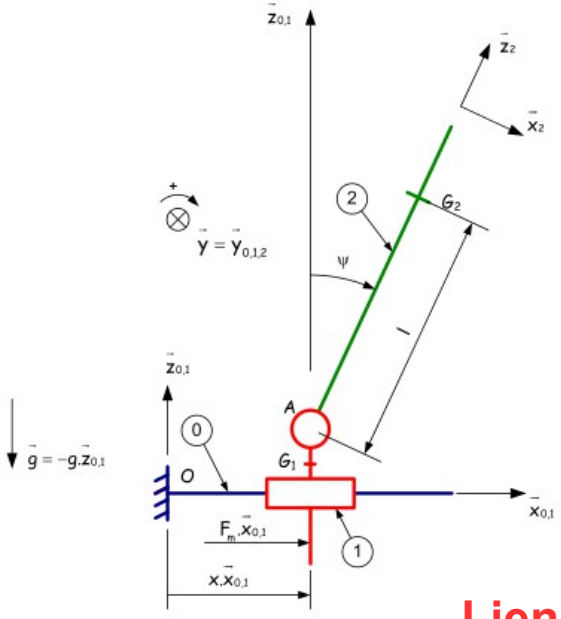


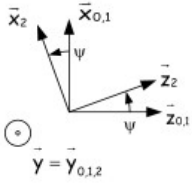
Apprentissage sur le système simulé

Implantation sur le système réel

Modèle plus fin prenant en compte les frottements

$$m_{tot}.\ddot{x} + m_2.l.\ddot{\psi}.\cos\psi - m_2.l.\dot{\psi}^2.\sin\psi = F_m - F_f.\text{sign}(\dot{x}) - f_v.\dot{x}$$
$$I.\ddot{\psi} + m_2.l.\ddot{x}.\cos\psi = m_2.g.l.\sin\psi - C_{f_2}.\text{sign}(\dot{\psi}) - f_{\omega_2}.\dot{\psi}$$





Composant	Modèle linéaire
CNA de la carte de commande	$\varepsilon_3(t) = \varepsilon_2(t)$
Carte de puissance	$u(t) = B.\varepsilon_3(t)$
Moteur linéaire équivalent	$m_{tot}.\dot{v}(t) = F_m(t) - f_v.v(t) - m_2.l.\ddot{\psi}(t)$ $e'(t) = k'.v(t)$ $F_m(t) = k'.i(t)$ $u(t) = r.i(t) + e'(t)$
Pendule	$I.\ddot{\psi}(t) + m_2.l.\ddot{x}(t) = m_2.g.l.\psi(t) - f_{\omega_2}.\dot{\psi}(t)$

Lien entre l'effort F appliqué au système et la modélisation de la MCC

```
## DEBUT Définition de l'environnement
##
import numpy as np
import random as r
import matplotlib.pyplot as pl

class labyrinthe():
    def __init__(self):
        self.Grille=np.array([
                ['-','O','-','S'],
                ['-','O','x','O'],
                ['x','-','-','-'],
                ['#','-','x','-']
            ])
        self.position=[]
        self.sortie=[]
        self.fin=False
        self.Mur=[]
        self.Rocher=[]
        self.Recompenses=[-1,-10,10,-5]

        self.index_elements()

    def index_elements(self):

        for i in range(len(self.Grille)):
            for j in range((len(self.Grille[0]))):
                if self.Grille[i][j]=='x':
                    self.Mur.append([i,j])
                if self.Grille[i][j]=='S':
                    self.sortie=[i,j]
                if self.Grille[i][j]=='O':
                    self.Rocher.append([i,j])
                if self.Grille[i][j]=='#':
                    self.position=[i,j]


    def obtenir_id(self,x,y):

        return x+y*len(self.Grille)

    def deplacer(self,action):

        nouvelle_position=self.position.copy()

        if action==0:
            nouvelle_position[1]-=1
        if action==1:
            nouvelle_position[0]-=1
        if action==2:
            nouvelle_position[1]+=1
        if action==3:
            nouvelle_position[0]+=1
```



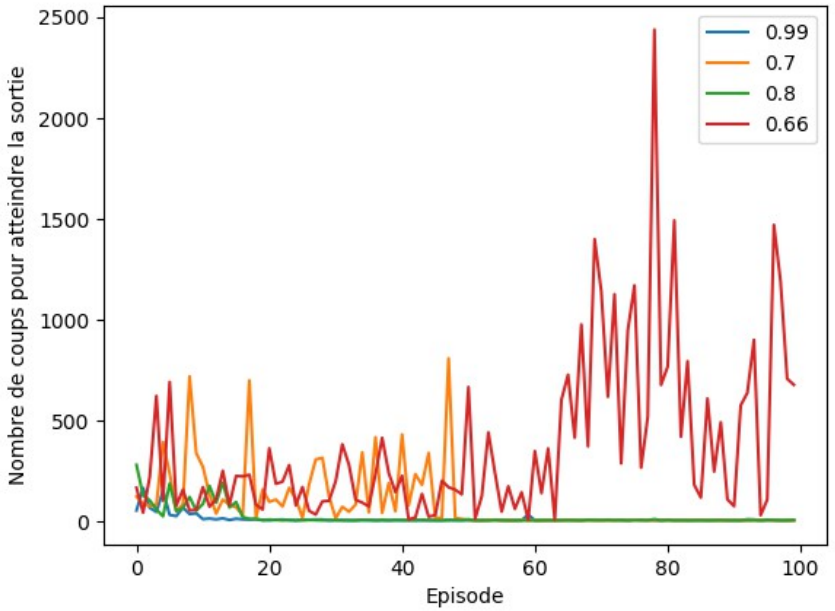
```
        ##Verifier si bordures
        if nouvelle_position[0]<0 or nouvelle_position[0]>=len(self.
            recompense=self.Recompenses[3]
            None
        ##Verifier si mur
        elif nouvelle_position in self.Mur:
            recompense=self.Recompenses[3]
            None
        ##Verifier si rocher
        elif nouvelle_position in self.Rocher:
            recompense=self.Recompenses[1]
            self.position=nouvelle_position
        ##Verifier si sortie
        elif nouvelle_position==self.sortie:
            recompense=self.Recompenses[2]
            self.fin=True
            self.position=nouvelle_position
        ##Aucun problème donc on autorise le déplacement
        else:
            self.position=nouvelle_position
            recompense=self.Recompenses[0]

        return recompense

    def afficher(self):
        ##Affiche l'état du labyrinthe
        for i in range(len(self.Grille)):
            for j in range(len(self.Grille[0])):
                if [i,j] in self.Mur:
                    self.Grille[i][j]='x'
                elif [i,j] in self.Rocher:
                    self.Grille[i][j]='O'
                elif [i,j]==self.position:
                    self.Grille[i][j]='#'
                elif [i,j]==self.sortie:
                    self.Grille[i][j]='S'
                else:
                    self.Grille[i][j]='-'
        print(self.Grille)


# FIN Définition de l'environnement
#
```

```
## Debut algorithme d'apprentissage
##_____________________________________________________

def Q_learning(env,learning_rate,gamma,iteration):
    import random as r

    ##START Algorithm parameters: step size...
    lr=learning_rate
    y=gamma
    iteration=iteration
    ##END Algorithm parameters: step size...

    ##START Initialize Q(s,a)...arbitrarly execpt that Q(terminal,.)=0
    partie=labyrinthe()
    n_states=len(partie.Grille)*len(partie.Grille[0])
    n_actions=4
    Q=np.zeros([n_states,n_actions])
    for i in range(len(Q)):
        for j in range(len(Q[0])):
            Q[i,j]=r.random()
    ##END Initialize Q(s,a)...arbitrarly execpt that Q(terminal,.)=0

    ##START Loop for each episode
    for i in range(iteration):
        partie=labyrinthe()

        #Initialize S
        s=partie.obtenir_id(partie.position[0],partie.position[1])

        ##START Loop for each step of epsiode
        while partie.fin==False:

            ## START Choose A from S using
            import random as r
            if r.random()<0.1:
                a=r.randint(0,n_actions-1)
            else:
                a=np.argmax(Q[s,:])
            ## END Choose A from S using

            #Take action  A , Observe R
            r=partie.deplacer(a)

            #Observe S'
            s1=partie.obtenir_id(partie.position[0],partie.position[1])

            #Refresh Q: Q(S,A)=Q(S,A)+...
            Q[s, a] = Q[s, a] + lr*(r + y * np.max(Q[s1,:]) - Q[s, a])

            #S becomes S'
            s=s1
        ##END Loop for each step of epsiode
    ##END Loop for each episode

    return Q

## FIN Algorithme d'apprentissage
##_____________________________________________________
```

1/1


```
Coups:  0
[['-' 'O' '-' 'S']
 ['-' 'O' 'x' 'O']
 ['x' '-' '-' '-']
 ['-' '#' 'x' '-']]

_____________________________________
Coups:  1
[['-' 'O' '-' 'S']
 ['-' 'O' 'x' 'O']
 ['x' '#' '-' '-']
 ['-' '-' 'x' '-']]

_____________________________________
Coups:  2
[['-' 'O' '-' 'S']
 ['-' 'O' 'x' 'O']
 ['x' '-' '#' '-']
 ['-' '-' 'x' '-']]

_____________________________________
Coups:  3
[['-' 'O' '-' 'S']
 ['-' 'O' 'x' 'O']
 ['x' '-' '-' '#']
 ['-' '-' 'x' '-']]

_____________________________________
Coups:  4
[['-' 'O' '-' 'S']
 ['-' 'O' 'x' 'O']
 ['x' '-' '-' '-']
 ['-' '-' 'x' '-']]

_____________________________________
Coups:  5
[['-' 'O' '-' '#']
 ['-' 'O' 'x' 'O']
 ['x' '-' '-' '-']
 ['-' '-' 'x' '-']]

_____________________________________
>>>
```

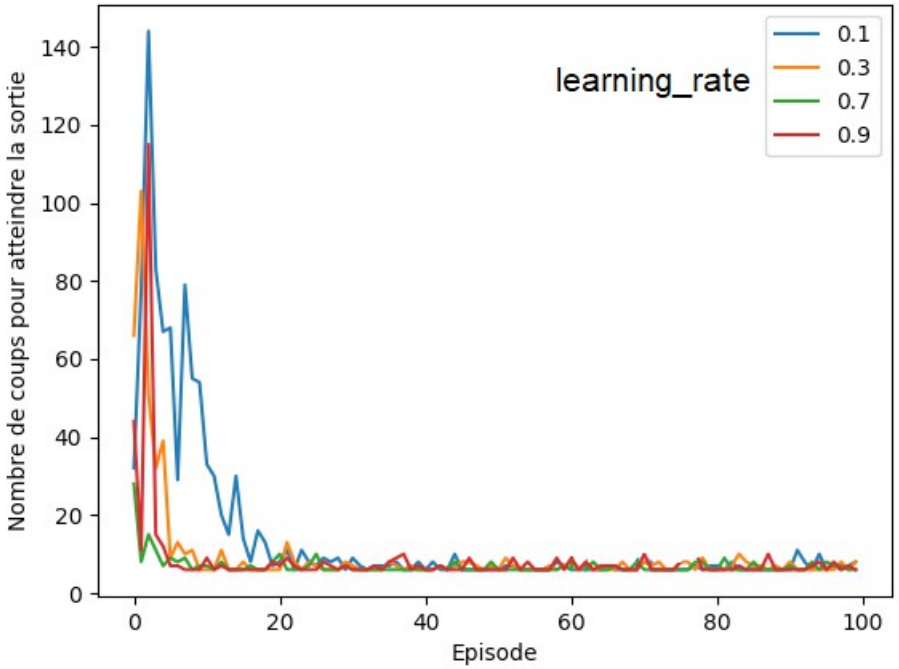
Résultat dans la console Python



Gamma pour Learning_rate fixé

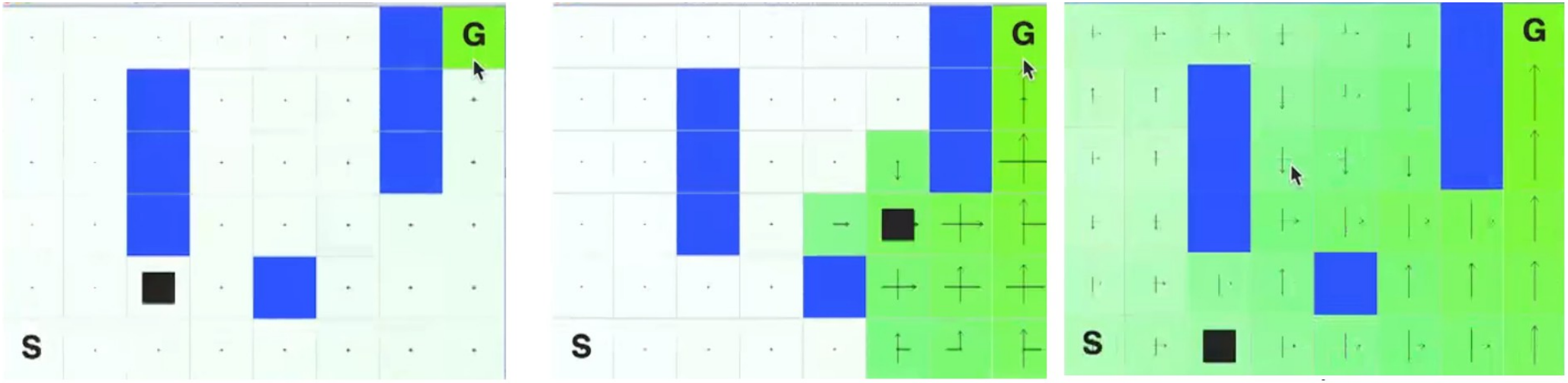
Mesure rendant compte de l'influence des deux paramètres de l'algorithme

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$



Learning_rate pour Gamma fixé

Visualisation de la propagation de la récompense de la sortie vers les états précédents



Capture d'écran de la conférence de R.S Sutton [2]

Code Python du pendule inversé
L'environnement

```
##_____________________
## DEBUT environnement
##_____________________

import math
import gym
from gym import spaces, logger
from gym.utils import seeding
import numpy as np
import math as m

class CartPoleEnv(gym.Env):

    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second' : 50
    }

    def __init__(self):

        #Paramètres de la modélisation:
        self.gravity = 9.8
        self.masscart = 1.0
        self.masspole = 0.1
        self.total_mass = (self.masspole + self.masscart)
        self.length = .5 # actually half the pole's length
        self.polemass_length = (self.masspole * self.length)
        self.force_mag = 10.0
        self.tau = 0.02  # dt pour euler
        self.dt_system= 0.02   # temps de réaction du système

        #Paramètres de l'instance
        self.Recompenses=[-10,0,1,5] #Récompenses associées à [Zône rouge, Neutre, Zô

        #Limite de fin d'instance et d'intervalle de dicrétisation
        self.x_limit= 1 #Zône rouge pour le chariot
        self.theta_limit= 30 * 2 * math.pi / 360 #Cône rouge pour la barre
        self.x_dot_limit=15
        self.theta_dot_limit=15

        #Paramètre de discrétisation
        self.nb_discretisation_x=12
        self.nb_discretisation_x_dot=self.nb_discretisation_theta=self.nb_discretisat

        #Intervalles de dicrétisation
        self.X=np.linspace(-self.x_limit, self.x_limit, self.nb_discretisation_x)
        self.X_dot=np.linspace(-self.x_dot_limit, self.x_dot_limit, self.nb_discretis
        self.THETA=np.linspace(-self.theta_limit, self.theta_limit, self.nb_discretis
        self.THETA_dot=np.linspace(-self.theta_dot_limit, self.theta_dot_limit, self.

        high = np.array([
            self.x_limit,
            np.finfo(np.float32).max,
            self.theta_limit,
            np.finfo(np.float32).max])

        #Autres paramètres
        self.action_space = spaces.Discrete(2)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)
        done=False  #Valeur déclarant la fin de l'instance

        self.seed()
        self.viewer = None
        self.state = None
        self.steps_beyond_done = None

    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]
```

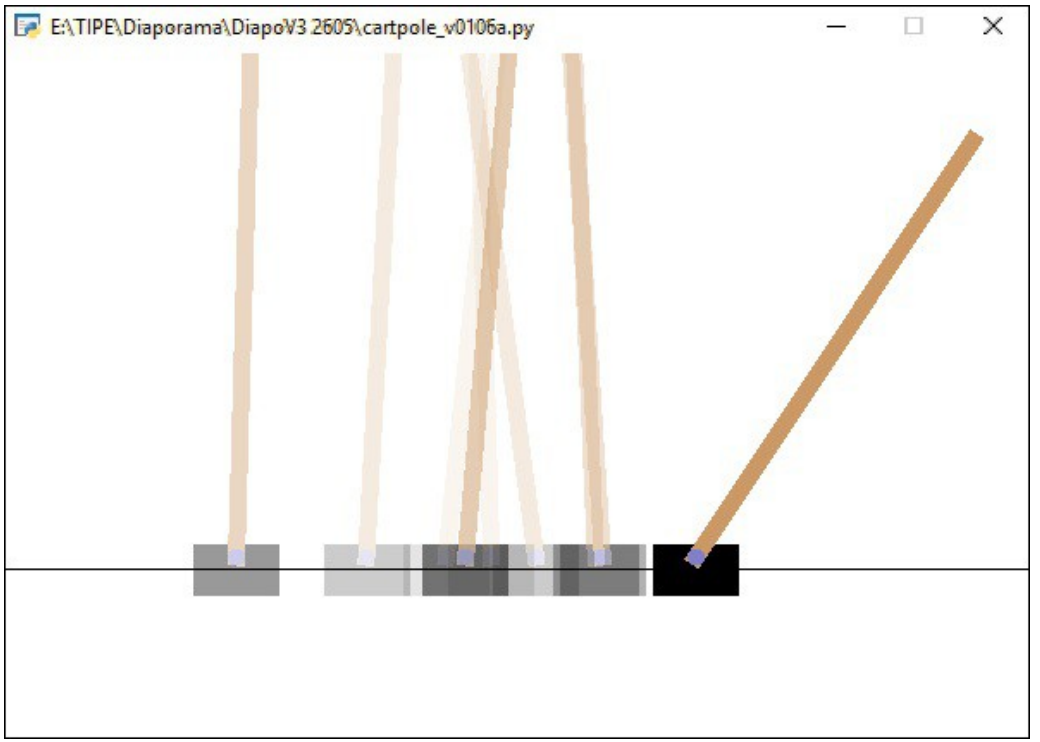


```
    def step(self, action):
        #Modélisation du pendule
        for i in range(int(self.dt_system/self.tau)):
            state = self.state
            x, x_dot, theta, theta_dot = state
            force = self.force_mag if action==1 else -self.force_mag
            costheta = math.cos(theta)
            sintheta = math.sin(theta)
            temp = (force - self.polemass_length * theta_dot * theta_dot * sintheta) / self.t
            thetaacc = (self.gravity * sintheta - costheta* temp) / (self.length * (1 - self
            xacc  = temp - self.polemass_length * thetaacc * costheta / self.total_mass

            x_dot = x_dot + self.tau * xacc
            x  = x + self.tau * x_dot
            theta_dot = theta_dot + self.tau * thetaacc
            theta = theta + self.tau * theta_dot

            self.state = (x,x_dot,theta,theta_dot)

            #Associer etat(state) continue à ceux discrets
            self.state_discrete=self.continu_to_discrete(self.state)

        pos_cart,vit_cart,pos_pole,vit_pole=self.state_discrete

        #Condition de fin de partie, si valeur sors des limites autorisés
        last_index_state=len(self.X)
        mid_index_state=len(self.X)//2
        if pos_pole==0 or pos_pole==last_index_state or vit_pole==0 or vit_pole==last_index_
            done=True
        else:
            done=False

        #Attribution des récompenses
        negative_reward,neutral_reward,positive_reward,outstanding_reward=self.Recompenses
        if (pos_pole==5 and (pos_cart in [mid_index_state-1,mid_index_state,mid_index_state+1
            reward=positive_reward
        elif pos_pole==mid_index_state:
            reward=outstanding_reward
        elif pos_pole==0 or pos_pole==last_index_state or vit_pole==0 or vit_pole==last_inde
            reward=negative_reward
        else:
            reward=neutral_reward

        return self.state_discrete, reward, done

    def reset(self):
        self.state = self.np_random.uniform(low=-0.05, high=0.05, size=(4,))
        self.steps_beyond_done = None
        self.state_discrete=self.continu_to_discrete(self.state)
        return np.array(self.state_discrete)

    def continu_to_discrete(self,state):
        state_discrete=[0,0,0,0]
        x,x_dot,theta,theta_dot=state
        for i in range(len(self.X)-1):
            if self.X[i] <= x < self.X[i+1]:
                state_discrete[0]=i
            if self.X_dot[i] <= x_dot < self.X_dot[i+1]:
                state_discrete[1]=i
            if self.THETA[i] <= theta < self.THETA[i+1]:
                state_discrete[2]=i
            if self.THETA_dot[i] <= theta_dot < self.THETA_dot[i+1]:
                state_discrete[3]=i
        return state_discrete
```

```
    def render(self, mode='human'):
        screen_width = 600
        screen_height = 400

        world_width = self.x_limit*2
        scale = screen_width/world_width
        carty = 100 # TOP OF CART
        polewidth = 10.0
        polelen = scale * (2 * self.length)
        cartwidth = 50.0
        cartheight = 30.0

        if self.viewer is None:
            from gym.envs.classic_control import rendering
            self.viewer = rendering.Viewer(screen_width, screen_height)
            l,r,t,b = -cartwidth/2, cartwidth/2, cartheight/2, -cartheight/2
            axleoffset =cartheight/4.0
            cart = rendering.FilledPolygon([(l,b), (l,t), (r,t), (r,b)])
            self.carttrans = rendering.Transform()
            cart.add_attr(self.carttrans)
            self.viewer.add_geom(cart)
            l,r,t,b = -polewidth/2,polewidth/2,polelen-polewidth/2,-polewidth/2
            pole = rendering.FilledPolygon([(l,b), (l,t), (r,t), (r,b)])
            pole.set_color(.8,.6,.4)
            self.poletrans = rendering.Transform(translation=(0, axleoffset))
            pole.add_attr(self.poletrans)
            pole.add_attr(self.carttrans)
            self.viewer.add_geom(pole)
            self.axle = rendering.make_circle(polewidth/2)
            self.axle.add_attr(self.poletrans)
            self.axle.add_attr(self.carttrans)
            self.axle.set_color(.5,.5,.8)
            self.viewer.add_geom(self.axle)
            self.track = rendering.Line((0,carty), (screen_width,carty))
            self.track.set_color(0,0,0)
            self.viewer.add_geom(self.track)

            self._pole_geom = pole

        if self.state is None: return None

        # Edit the pole polygon vertex
        pole = self._pole_geom
        l,r,t,b = -polewidth/2,polewidth/2,polelen-polewidth/2,-polewidth/2
        pole.v = [(l,b), (l,t), (r,t), (r,b)]

        x = self.state
        cartx = x[0]*scale+screen_width/2.0 # MIDDLE OF CART
        self.carttrans.set_translation(cartx, carty)
        self.poletrans.set_rotation(-x[2])

        return self.viewer.render(return_rgb_array = mode=='rgb_array')

    def close(self):
        if self.viewer:
            self.viewer.close()
            self.viewer = None
```

```
##_____________________________
## FIN environnement
##_____________________________
```

Code Python du pendule inversé
L'environnement



Code Python du pendule inversé
L'environnement

```
def step(self, action):
    #Modélisation du pendule
    for i in range(int(self.dt_system/self.tau)):
        state = self.state
        x, x_dot, theta, theta_dot = state
        force = self.force_mag if action==1 else -self.force_mag
        costheta = math.cos(theta)
        sintheta = math.sin(theta)
        temp = (force - self.polemass_length * theta_dot * theta_dot * sintheta) / self.total_mass
        thetaacc = (self.gravity * sintheta - costheta* temp) / (self.length * (1 - self.masspole * costheta * costheta / self.total_mass))
        xacc  = temp - self.polemass_length * thetaacc * costheta / self.total_mass

        x_dot = x_dot + self.tau * xacc
        x  = x + self.tau * x_dot
        theta_dot = theta_dot + self.tau * thetaacc
        theta = theta + self.tau * theta_dot

        self.state = (x,x_dot,theta,theta_dot)

        #Associer etat(state) continue à ceux discrets
        self.state_discrete=self.continu_to_discrete(self.state)
```

Résolution par Euler

```
##_____________________________________
## DEBUT Algorithme d'apprentissage
##_____________________________________

def Q_learning(env,learning_rate,gamma,iteration):
    import random as r

    ##START Algorithm parameters: step size...
    lr=learning_rate
    y=gamma
    iteration=iteration
    ##END Algorithm parameters: step size...

    ##START Initialize Q(s,a)...arbitrarly execpt that Q(terminal,.)=
    n_states=env.nb_discretisation_x
    n_actions=2
    Q=np.zeros([n_states,n_states,n_states,n_states,n_actions])
    for i in range(len(Q)):
        for j in range(len(Q[0])):
                for k in range(len(Q[0][0])):
                    for l in range(len(Q[0][0][0])):
                            for m in range(len(Q[0][0][0][0])):
                                    Q[i,j,k,l,m]=r.random()
    ##END Initialize Q(s,a)...arbitrarly execpt that Q(terminal,.)=0

    ##START Loop for each episode
    for i in range(iteration):
        #Initialize S
        end=False
        s=env.reset()

        ##START Loop for each step of epsiode
        while end==False:

            ## START Choose A from S using
            a=np.argmax(Q[s[0],s[1],s[2],s[3],:])
            ## END Choose A from S using

            #Take action  A
            observation=env.step(a)

            #Observe R,S'
            s1,r,end=observation

            #Refresh Q: Q(S,A)=Q(S,A)+...
            Q[s[0],s[1],s[2],s[3], a] = Q[s[0],s[1],s[2],s[3], a] + lr*(r + y * np.max(Q[s1[(

            #S becomes S'
            s=s1.copy()

        ##END Loop for each step of epsiode
    ##END Loop for each episode

    return Q
##_____________________________________
## FIN Algorithme d'apprentissage
##_____________________________________


env=CartPoleEnv()
learning_rate=0.8
gamma=0.99
iteration=201
Q_learning(env,learning_rate,gamma,iteration)
```

Code Python du pendule inversé
L'agent

Code Python du pendule inversé
L'environnement (alpha)

```
"""
Classic cart-pole system implemented by Rich Sutton et al.
Copied from http://incompleteideas.net/sutton/book/code/pole.c
permalink: https://perma.cc/C9ZM-652R
"""

import math
import gym
from gym import spaces, logger
from gym.utils import seeding
import numpy as np

class CartPoleEnv(gym.Env):
    """
    Description:
        A pole is attached by an un-actuated joint to a cart, which moves al
    Source:
        This environment corresponds to the version of the cart-pole problem
    Observation:
        Type: Box(4)
        Num     Observation               Min         Max
        0       Cart Position             -4.8            4.8
        1       Cart Velocity             -Inf            Inf
        2       Pole Angle                -24 deg         24 deg
        3       Pole Velocity At Tip      -Inf            Inf

    Actions:
        Type: Discrete(2)
        Num     Action
        0       Push cart to the left
        1       Push cart to the right

        Note: The amount the velocity that is reduced or increased is not fi
    Reward:
        Reward is 1 for every step taken, including the termination step
    Starting State:
        All observations are assigned a uniform random value in [-0.05..0.05
    Episode Termination:
        Pole Angle is more than 12 degrees
        Cart Position is more than 2.4 (center of the cart reaches the edge
        Episode length is greater than 200
        Solved Requirements
        Considered solved when the average reward is greater than or equal t
    """

    metadata = {
        'render.modes': ['human', 'rgb_array'],
        'video.frames_per_second' : 50
    }

    def __init__(self):
        self.gravity = 9.8
        self.masscart = 1.0
        self.masspole = 0.1
        self.total_mass = (self.masspole + self.masscart)
        self.length = 0.5 # actually half the pole's length
        self.polemass_length = (self.masspole * self.length)
        self.force_mag = 10.0
        self.tau = 0.02  # seconds between state updates
        self.kinematics_integrator = 'euler'

        # Angle at which to fail the episode
        self.theta_threshold_radians = 12 * 2 * math.pi / 360
        self.x_threshold = 2.4

        # Angle limit set to 2 * theta_threshold_radians so failing observat
        high = np.array([
            self.x_threshold * 2,
            np.finfo(np.float32).max,
            self.theta_threshold_radians * 2,
            np.finfo(np.float32).max])

        self.action_space = spaces.Discrete(2)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)

        self.seed()
        self.viewer = None
        self.state = None

        self.steps_beyond_done = None

    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]

    def step(self, action):
        assert self.action_space.contains(action), "%r (%s) invalid"%(action,
        state = self.state
        x, x_dot, theta, theta_dot = state
        force = self.force_mag if action==1 else -self.force_mag
        costheta = math.cos(theta)
        sintheta = math.sin(theta)
        temp = (force + self.polemass_length * theta_dot * theta_dot * sinthet
        thetaacc = (self.gravity * sintheta - costheta* temp) / (self.length *
        xacc  = temp - self.polemass_length * thetaacc * costheta / self.total_
        if self.kinematics_integrator == 'euler':
            x  = x + self.tau * x_dot
            x_dot = x_dot + self.tau * xacc
            theta = theta + self.tau * theta_dot
            theta_dot = theta_dot + self.tau * thetaacc
        else: # semi-implicit euler
            x_dot = x_dot + self.tau * xacc
            x  = x + self.tau * x_dot
            theta_dot = theta_dot + self.tau * thetaacc
            theta = theta + self.tau * theta_dot
        self.state = (x,x_dot,theta,theta_dot)
        done =  x < -self.x_threshold \
                or x > self.x_threshold \
                or theta < -self.theta_threshold_radians \
                or theta > self.theta_threshold_radians
        done = bool(done)

        if not done:
            reward = 1.0
        elif self.steps_beyond_done is None:
            # Pole just fell!
            self.steps_beyond_done = 0
            reward = 1.0
        else:
            if self.steps_beyond_done == 0:
                logger.warn("You are calling 'step()' even though this environ
            self.steps_beyond_done += 1
            reward = 0.0

        return np.array(self.state), reward, done, {}

    def reset(self):
        self.state = self.np_random.uniform(low=-0.05, high=0.05, size=(4,))
        self.steps_beyond_done = None
        return np.array(self.state)
```

```python
def render(self, mode='human'):
    screen_width = 600
    screen_height = 400

    world_width = self.x_threshold*2
    scale = screen_width/world_width
    carty = 100 # TOP OF CART
    polewidth = 10.0
    polelen = scale * (2 * self.length)
    cartwidth = 50.0
    cartheight = 30.0

    if self.viewer is None:
        from gym.envs.classic_control import rendering
        self.viewer = rendering.Viewer(screen_width, screen_height)
        l,r,t,b = -cartwidth/2, cartwidth/2, cartheight/2, -cartheight/2
        axleoffset =cartheight/4.0
        cart = rendering.FilledPolygon([(l,b), (l,t), (r,t), (r,b)])
        self.carttrans = rendering.Transform()
        cart.add_attr(self.carttrans)
        self.viewer.add_geom(cart)
        l,r,t,b = -polewidth/2,polewidth/2,polelen-polewidth/2,-polewidth/2
        pole = rendering.FilledPolygon([(l,b), (l,t), (r,t), (r,b)])
        pole.set_color(.8,.6,.4)
        self.poletrans = rendering.Transform(translation=(0, axleoffset))
        pole.add_attr(self.poletrans)
        pole.add_attr(self.carttrans)
        self.viewer.add_geom(pole)
        self.axle = rendering.make_circle(polewidth/2)
        self.axle.add_attr(self.poletrans)
        self.axle.add_attr(self.carttrans)
        self.axle.set_color(.5,.5,.8)
        self.viewer.add_geom(self.axle)
        self.track = rendering.Line((0,carty), (screen_width,carty))
        self.track.set_color(0,0,0)
        self.viewer.add_geom(self.track)

        self._pole_geom = pole

    if self.state is None: return None

    # Edit the pole polygon vertex
    pole = self._pole_geom
    l,r,t,b = -polewidth/2,polewidth/2,polelen-polewidth/2,-polewidth/2
    pole.v = [(l,b), (l,t), (r,t), (r,b)]

    x = self.state
    cartx = x[0]*scale+screen_width/2.0 # MIDDLE OF CART
    self.carttrans.set_translation(cartx, carty)
    self.poletrans.set_rotation(-x[2])

    return self.viewer.render(return_rgb_array = mode=='rgb_array')

def close(self):
    if self.viewer:
        self.viewer.close()
        self.viewer = None
```

3/3

Code Python du pendule inversé
L'environnement (alpha)

Code Python du pendule inversé
Mesures