TP 1 : Découverte de l'Outil Gym

(a rendre à l'adresse : santucci@univ-corse.fr)

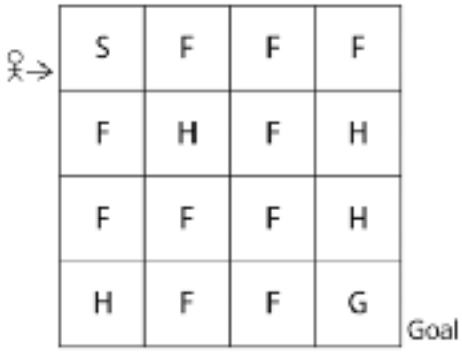
Découverte de l'outils OpenAI Gym

1.Installation de Gym

2. Exercices à partir de l'exemple Frozen Lake

2.1. Creation d'un environnement Gym.

Exemple l'environnement Frozen Lake : but atteindre l'état but G à partir de l'état initial S
S état initial
G état but
F état Frozen
H état Goal



Chaque case de la grille de l'environnement est appelée un. État.
Donc 16 stats de S à G et 4 actions possibles (up, down, left, right).
Objectif : atteindre G à partir de A sans passer par les etats H.
Rewards : +1 pour l'état G, 0 pour les autres états (par exemple mais pas le plus performant mais le but est de découvrir gym).

a-Pour avoir accès à gym sous python: import gym.
b- Ensuite créer l'environnement par : env = gym.make(''FrozenLake-v0'')
c- Voir à quoi ressemble l'environnement par la fonction render : env.render()

Nous avons vu qu'un environnement d'apprentissage automatique était modélisé par un MDP (Markov decision Process) impliquant:

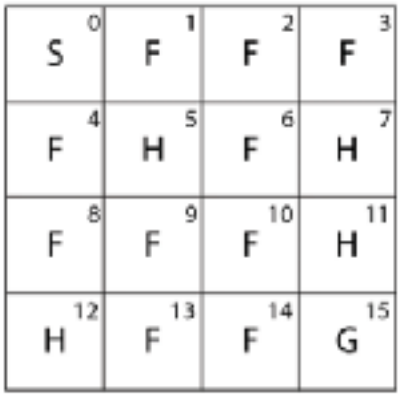
- l'ensemble des états
- l'ensemble des actions
- les probabilités de transition notées P(s' I (s,a) - la probabilité de passer d'un état s à un état s' en executant l'action a.
- les rewards notées R(s,a,s') : la récompense renvoyée à l'agent en passant de l'état s à l'etat s' en executant l'action a.

d- Visualiser les états avec Gym :

```
>>> print (env.observation_space)
Discrete(16)
>>>
```

Discrete(16) veut dire que nous avons 16 états discrets dans l'espace d'états à partir d l'état S à l'état G.

Dans Gym, les états sont co- dées sous forme de nombre :



L'ensemble des actions possible est donné par : env.action_space

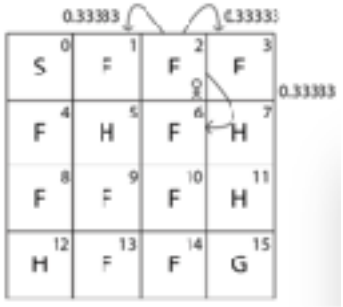
e- Visualiser les actions avec Gym

```
>>> print(env.action_space)
Discrete(4)
>>>
```

Dans Gym, les actions sont codées sous forme de nombres :

Number	Action
0	Left
1	Down
2	Right
3	Up

L'environnement FROZEN LAKE est stochastique



Pour obtenir les probabilités de transition et les rewards, il suffit de taper : env.P[state][action].

Donc pour obtenir la probabilité de passer de l'état S aux autres états en executant l'action right : env.P[S][right].
Bien sur dans Gym le codage se fait à l'aide des nombres donc: env.P[0][2]

```
>>> print (env.env)
<FrozenLakeEnv<FrozenLake-v0>>
>>> print (env.env.P[0][2])
[(0.3333333333333333, 4, 0.0, False), (0.3333333333333333, 1, 0.0, False), (0.3333333333333333, 0, 0.0, False)]
>>>
```

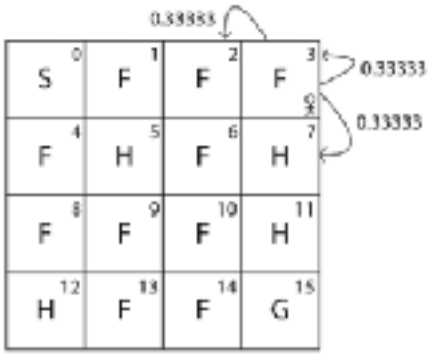
f- Visualiser les probabilités:
REMARQUE : Attention env.env (????).

En tapant env.P[state][action], on a comme résultat quelque chose de ce format:
[(probabilité de transition, état suivant, reward, est-ce l'état but?)].

Donc résultat

Transition Probability	Next State	Reward	Is terminal state
0.33333	4(F)	0.0	False
0.33333	1(F)	0.0	False
0.33333	0(S)	0.0	False

g-Autre exemple: Supposons que nous effectuons l'action 1 (down) dans l'état 3 (F).
Ecrire l'instruction permettant de visualiser la probabilité de transition de l'état 3 (F) en effectuant l'action 1
On devrait obtenir :



On peut exprimer le résultat sous forme de table:

Transition Probability	Next State	Reward	Is terminal State
0.33333	2(F)	0.0	False
0.33333	7(H)	0.0	True
0.33333	3(F)	0.0	False

Remarque : dans la deuxième ligne de la sortie on trouve (0.33333, 7, 0.0, True). Cette dernière valeur True indique que l'etat 7 est un état terminal.
Donc si on effectue l'action 1 (down) dans l'état 3 alors on atteint l'état 7 avec une probabilité de 0.33333 et puisque 7 est un trou , c'est un état terminal (l'agent meurt).

Résumé ; on a vu comment obtenir l'espace d'état, l'espace d'action, la probabilité de transition et la fonction de recompense en utilisant Gym. Voyons comment générer un episode.

2.2. Générer une série d'épisodes avec Gym

Rappel : un épisode est l'interaction agent-environnement depuis l'état initial jusqu'à l'état terminal : l'agent interagit avec l'environnement en effectuant une action dans chaque état.

Nous allons générer un episode en prenant des actions aléatoires dans chaque état.

avant de commencer, il faut initialiser l'état en réinitialisant l'environnement/ La réinitialisation remet l'agent à l'état initial.
a-Pour réinitialiser l'environnement utiliser la fonction reset: state = env.reset()

b-Selection d'une action: Pour que l'agent interagisse avec l'environnement, il doit effectuer une action dans l'environnement.
Exemple : on est dans l'etat 3 (F)

On veut effectuer l'action 1 (down) et passer à l'état 7(H).
-Vous allez utiliser la fonction step.: env.step(1); Verifier que l'action à été exécutée.

Remarque : la sortie de env.step est du type (next_state, reward, done,info) où next_state est l'état suivant, reward la récompense associée à la transition , done est false ou true suivant le fait que l'état suivant est un état terminal ou non et info extant la probabilité associée avec la transition et l'action, et l'état.
On peut donc récupérer les informations en faisant : next_state, reward, done,info = env.step(1)

c- On peut aussi effectuer une action aléatoire à partir de espace d'action avec la fonction sample.
-récupérer une action aléatoire par : random_action = env.action_space.sample()
-effectuer l'action aléatoire par : next_state, reward, done,info = env.step(random_action)

d-générer un épisode.
Rappel ; un épisode est l'interaction agent-environnement depuis l'état initial jusqu'à un état terminal. L'agent interagit avec l'environnement en effectuant une action dans chaque état.
Un épisode se termine si l'agent atteint un état terminal: dans Frozen Lake un état terminal est l'état but G ou un état de type H(trou).
Unepolitique aléatoire sélectionné une action aléatoire dans chaque état.
Il va falloir générer un episode en prenant des actions aléatoires dans chaque état.

Question : Ecrire les instructions permettant d'implémenter la génération d'un episode avec une politique aléatoire.
Indication: Il faut tout d'abord définir le nombre de pas : (timesteps)
num_timesteps = 20

Puis boucler sur le nombre de pas en
- générant une action aléatoire à executer avec : random_action = env.action_space.sample()

- -executant l'action sélectionnée par : new_state,reward,done,info = env.step(random_action)
- -sortir de la boucle quand l'état terminal est obtenu

Il faudra obtenir une sortie de ce type (en faisant afficher les états intermédiaires et le pas de temps courant (time step):

e-générer une série d'episodes (en prenant une action aléatoire chaque état).
Indication : utiliser une variable permettant de fixer le nombre d'episodes.

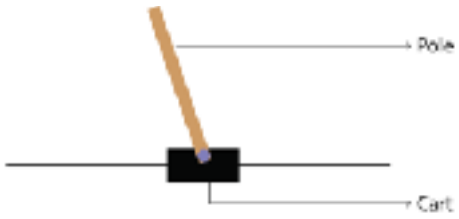
Dans l'exemple étudié, on a juste pris des actions aléatoires dans chaque état sur tous les épisodes. Evidemment nous avons besoin d'un algorithme d'apprentissage comme vu précédemment pour trouver la politique optimale (c-a-d la politique qui dit quelle action effectuée dans chaque état). Nous verrons dans le cours comment faire cela.
Nous allons voir avant d'autres fonctionnalités de l'outil Gym a travers l'exemple classique du Cart-Pole.

3. Le Cart-Pole

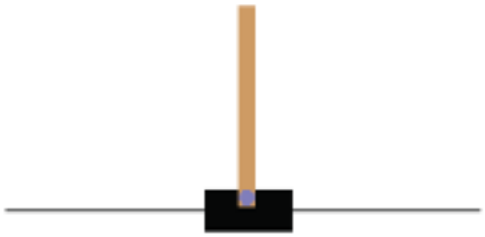
Gym fournit des environnements pour plusieurs tâches de contrôle classiques telles que l'équilibrage de Caart-Pole, le balancement d'un pendule inversé, l'escalade en voiture d montage,etC..

3.1. Création d'un environnement Gym pour une tache d'équilibrage Cart-Pole

L'environnement Cart-Pole peut être décrit ainsi :



Une perche (pole) est attachée au chariot (cart) et l'objectif de l'agent est d'équilibrer la perche (pole) sur le chariot (cart) (c-a-d que l'objectif de l'agent est de maintenir la perche (pole) debout sur le chariot (cart) ainsi



L'agent essaie donc de pousser le chariot à gauche et à droite pour maintenir la perche droite sur le chariot. Donc l'agent effectue 2 actions (à gauche et à droite) qui poussent le chariot vers la gauche ou vers la droite.

a - Manipulation de l'environnement Cart-Pole avec Gym
Creation de l'environnement : env = gym.make("CartPole-v0")

b -Espace d'état
L'etat du système est dans ce cas décrit par les valeurs continues suivantes:
- la posiiton du chariot (entre -4.8 et 4.8)
- la vitesse du chariot (entre -Inf et +Inf)
- L'angle de la perche (entre -0.418 radians et 0.418 radians)
- La vitesse de la perche à la pointe (entre -Inf et + Inf)

Donc l'espace d'état est un tableau de valeurs continues.
Pour obtenir l'espace d'état : print (env.observation_space) qui retourné Box(4,).
Box indique que l'espace d'état est constitué de valeurs continues et pas de leurs discrètes et 4 indique qu'il s'agit d'un tableau de 4 valeurs.
Reinitialisons l'environnement et voyons quel est l'état initial.
- print(env.reset()) qui retourne quelque chose du type:
[0.01663276 0.00942562 -0.01541287 0.00452822] (initialisation aléatoire)

on retrouve (position du chariot, la vitesse du chariot, l'angle de la perche, la vitesse de la perche).

On peut aussi obtenir les valeurs maximum et minimum de l'espace d'état avec :
−print (env.observation_space.high)
−print (env.observation_space.low
c -Espace d'actions
Dans l'environnement Cart-Pole: 2 actions. qui poussent le chariot vers la gauche ou vers la droite.
Donc l'espace d'action est discret
−a avec env.action_space vous obtenez des infos sur l'espace d'actions.
Evidemment on a Discrete (2) car deux actions discrètes.
Les actions sont codées ainsi :

Number	Action
0	Push cart to the left
1	Push cart to the right

3.2. Equilibrage du Cart-Pole avec une politique aléatoire

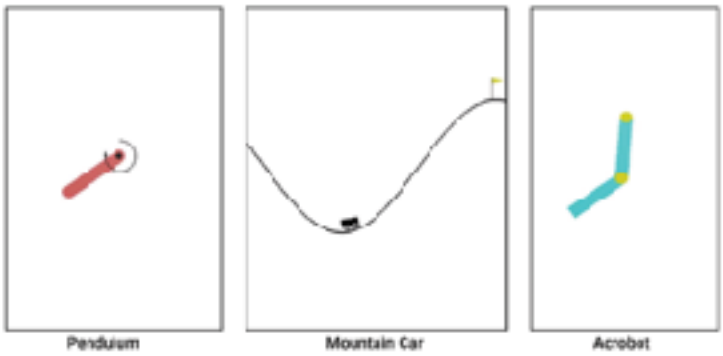
a-Vous allez créer un agent avec la politique aléatoire. Donc vous allez créer un agent qui sélectionne une action aléatoire dans l'environnement et essaie d'équilibrer la perche.
L'agent recevra une récompenses + 1 chaque fois que la perche se dresse sur le chariot.
Vous allez générer 100 épisodes et verrez le retour (somme des récompenses) obtenu sur chaque épisodes.
Pour créer l'environnement :
import gym
env = gym.make('CartPole-v0')
num_episodes = 100
num_timesteps = 50 (nombre de pas de temps dans un épisode)

Ecrire le code en vous inspirant de ce que vous avez fait précédemment.
b- Vous écrirez des instructions afin d'obtenir comme résultats :
Episode: 0, Return: 12.0
Episode: 10, Return: 22.0
Episode: 20, Return: 21.0
Episode: 30, Return: 14.0
Episode: 40, Return: 16.0
Episode: 50, Return: 17.0
Episode: 60, Return: 19.0
Episode: 70, Return: 33.0
Episode: 80, Return: 19.0
Episode: 90, Return: 10.0
Donc vous afficherez le Return tous les dix épisodes.

Evidemment les actions sont sélectionnées aléatoirement.
Nous allons lors du prochain TP voir comment écrire un algorithme d'apprentissage.

4. Autres environnements Gym

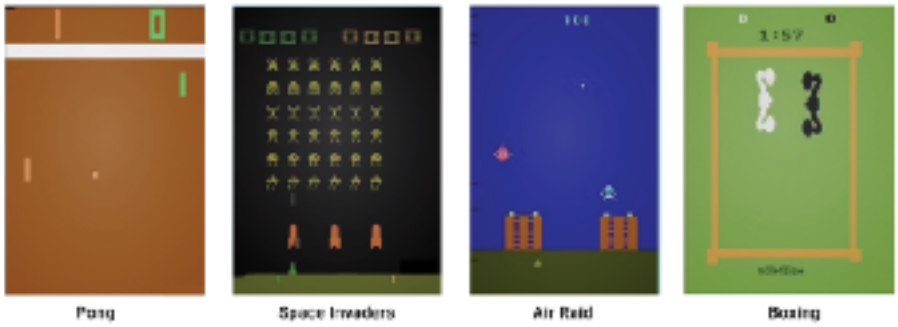
D'autres environnements du même type que CartPole sont disponibles comme:



Vous pouvez visualiser les environnements existants avec:
from gym import envs
print(envs.registry.all())

4.1 Environnements de jeux Atari

De nombreux jeux Atari sont disponibles (tels que Pong, space Invaders, Air Raid, Asteroids, Centipede, Pac-Man, etc..



Dans Gym, chaque environnement d'un jeu Atari a 12 variantes.
Exemple Pong

- Pong-v0 et Pong-v4: on peut créer un environnement Pong avec Pong-v0 ou Pong-v4. L'etat est l'image de l'écran du jeu.
- Pong-ram-v0 et Pong-ram-v4: similaire à Pong-v0 et Pong-v4 sauf que l'état est l'état de la RAM de la machine Atari (128 bytes au lieu des valeurs pixel de l'écran du jeu.
- PongDeterministic-v0 et Pong-Deterministic-v4: la position initiale du jeu est toujours la même à chaque fois que l'on réinitialise l'environnement (letat de l'environnement est les valeurs pixel de l'écran du jeu).
- Pong-ramDeterminisitic-v0 et Pong-ramDeterministic-v4: identique aux précédents mais l'état est la RAM Atari.
- PongNoFrameskip-v0 et PongNoFrameskip-v4:tous les écrans de jeu sont visibles par l'agent (valeurs en pixels)
- Pong-ramNoFrameskip-v0 et Pong-ram NoFrameskip-v4:tous les écrans de jeu sont visibles par l'agent (etat RAM de la machine Atari)

a-Espaces d'Actions et d'états et
Créer un environnement Tennis comme d'habitude: env = gym.make("Tennis-v0")
L'écran de jeu est l'état de l'environnement.
Les instructions sont identiques à celles précédentes.

```
import gym
env = gym.make('Tennis-v0')
env.render()
```

b- Agent jouant au jeu Tennis

```
num_episodes = 100
num_timesteps = 50
```

Puis la boucle sur sur le nombre d'épisodes et de pas.
Resultats:
−L'environnement du jeu

−Les Return tous les dix épisodes comme :

```
Episode: 0, Return: -1.0
Episode: 10, Return: 0.0
Episode: 20, Return: -1.0
Episode: 30, Return: -1.0
Episode: 40, Return: 0.0
Episode: 50, Return: -1.0
Episode: 60, Return: -1.0
Episode: 70, Return: 0.0
Episode: 80, Return: -1.0
Episode: 90, Return: -1.0
```

c- Enregistrer le jeu
Nous avons vu comment créer un agent qui sélectionne au hasard une action dans l'espace d'action et joue au tennis.
Nous allons pouvoir enregistre le jeu joué et le sauvegarder sous forme de video.
Gym fournit une classe wrapper qu'on va utiliser pour enregistrer le gameplay de l'agent sous forme video.
Pour cela:

```
import gym
env = gym.make('Tennis-v0')
env = gym.wrappers.Monitor(env,'recording',force=True)
env.render()
```

num_episodes = 100
num_timesteps = 50

Le jeu est enregistrer en utilisant le wrapper Monitor comme indiqué dans le code précédent. il faut trois paramètres : l'environnement;le répertoire où vous voulez sauvegarder les videos, et l'option forcée. Si force = False cela implique que vous devez créer un nouveau répertoire chaque fois que vous voulez enregistrer de nouveaux enregistrements. Si force = True, les anciens enregistrements du répertoire sont effacés et remplacés par de nouveaux enregistrments.

Attention : Pour enregistrer le jeu, votre système doit prendre en charge FFmpeg. FFmpeg est un framework utilisé pour le traitement des fichiers multimédia .

4.2 Autres Environnements

A découvrir soi-même.