LECOURTIER
Frédérique

27/05/2021

Projet : Apprentissage par renforcement

(en binôme avec Salomé Oswald)

1. Organisation du code

Toutes les classes créées sont des templates de classe dépendant de 2 paramètres, le type des états et le type des actions. Dans une première partie, nous avons créé 2 classes de processus de décision markovien, *MDP* et *MDP_parfait* qui hérite de *MDP*. Cette classe fille possède deux attributs supplémentaires correspondant aux probabilités et aux récompenses associées aux états et aux actions possibles.

Dans un second temps, nous avons cherché à déterminer la meilleure politique permettant de maximiser nos gains futurs. Pour cela, nous avons créé 2 classes *ProgDym* et *IterationValeur* qui hérite de *ProgDym*. On a ensuite pu comparer nos résultats entre des exécutions avec politique et sans politique (de manière aléatoire).

Dans une troisième partie, nous avons cherché à déterminer une politique dans le cas d'un MDP non parfait, c'est-à-dire un MDP ne possédant pas de probabilités des états suivants. Les méthodes de programmation dynamique ne s'appliquant qu'au MDP parfait, nous avons dû utiliser dans cette partie des algorithmes dit « stochastiques ». Le but étant de faire apprendre à notre robot les meilleures actions possibles en fonctions des états. Nous avons pour cela créé 2 classes *ModelSto* et *Q_learning* qui hérite de *ModelSto*. La première classe est une classe abstraite, l'avantage est qu'on ne peut pas déclarer un objet de cette classe ce qui d'ailleurs n'aurait pas réellement d'intérêt et cela nous permet également de forcer la déclaration des méthodes virtuelles dans les classes filles.

2. Résultats

Pour la deuxième partie, nous avons dans un premier temps utilisé notre politique et avons obtenu une récompense autour de 33. Par la suite, on a effectué une série d'actions de manières aléatoires afin de comparer les résultats. Pour être un peu plus précis, nous avons décidé d'effectuer une moyenne sur 10 000 cas et avons obtenu un résultat proche de 21, nettement inférieur à la récompense totale lorsque l'on utilise la politique. Voici ce que l'on a obtenu lors d'une exécution :

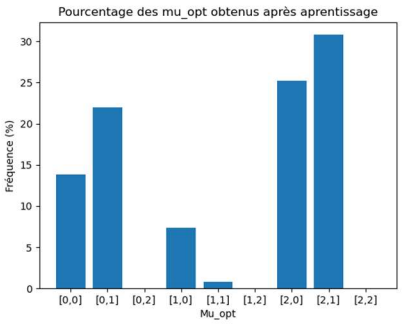
```
Récompense avec politique : 33
Moyenne des récompense sans politique : 21.2997
```

Pour la troisième partie, nous avons choisi d'effectuer comme pour la partie 2 une moyenne des récompenses sur 500 essais mais également de déterminer quelles politiques optimales apparaissent le plus. Pour cela nous avons décidé de récupérer le nombre de fois où chaque μ_{opt} possible apparait sur ces 500 essais et d'en faire un pourcentage. A noter que l'on constate bien que les μ_{opt} avec la valeur 2 en deuxième position ont tous un pourcentage nul, ce qui est correct car il n'est pas possible de recharger dans le cas où la batterie est haute.
Le dossier résultat que nous avons rajouté ne fait pas, à proprement parler, parti du travail demandé, nous voulions avoir une représentation des politiques les plus fréquentes. Pour cela, nous avons redirigé la sortie du terminal vers un fichier "sortie" afin de récupérer au

travers d'un script python "resultat.py" les pourcentages des μ_{opt} obtenus. Nous avons ensuite généré un graphique à barres dans "graphique_resultats.png" où l'on peut voir que la meilleure action pour un niveau de batterie bas est de recharger et que pour un niveau haut, c'est soit d'aller chercher soit d'attendre (ce qui correspond aux μ_{opt} [2,0] et [2,1]). En effet, il paraît logique de recharger la batterie si elle est faible et de faire d'autres actions si elle est haute. Voici ce que l'on a obtenu lors d'une exécution :

> Pourcentage des mu_opt :
> [0,0] : 13.8 %
> [0,1] : 22 %
> [0,2] : 0 %
> [1,0] : 7.4 %
> [1,1] : 0.8 %
> [1,2] : 0 %
> [2,0] : 25.2 %
> [2,1] : 30.8 %
> [2,2] : 0 %
> Moyenne des récompenses par apprentissage : 26.234

On obtient alors le graphique suivant :



Pour aller plus loin, on pourrait maintenant voir ce que notre robot ferait si on lui ajoutait un état supplémentaire correspondant par exemple à un niveau de batterie intermédiaire. En effet, il peut sembler incohérent que le robot attende alors que son niveau de batterie est élevé. Chaque action aurait alors la possibilité d'être la meilleure pour un état.