Agents Intelligents

# TD1 :

Quand on ‘run’ mon code python, c’est la fonction choix() qui s’applique. Cette fonction est combinaison et le final de tout le TP.

Les premières lignes de cette fonction est de demander à l’utilisateur quels agents il veut utiliser (au fil du TP j’ai réalisé plusieurs agents : Reflexe Simple, Reflexe Simple Randomisé, Reflexe avec État et Reflexe avec Etat et Obstacle) ; après il demande la taille de la matrice ( mon environnement est toujours une matrice carre, une amélioration possible de mes agents serait de pouvoir avoir la taille non symétrique que l’on souhaite, ex : 2x5 ), il vérifie bien sûr que l’entrée de l’utilisateur soit un entier ; et enfin il demande le pourcentage de saleté que ma salle doit avoir et vérifie que ce soit un entier entre 0 et 100% .

Débute alors le code de la fonction de l’agent choisi.

Dans le cas de l’AgentReflexeAvecEtatAvecObstacle, il y’a une dernière demande a l’utilisateur : Le nombre d’obstacle souhaité.

## Explication de mes fonctions de base :

* MatriceP(TailleMatrice) => renvoie une matrice Propre (remplie de 0) de taille N.

Une image contenant texte

Description générée automatiquement

* plot\_grid(Matrice, TailleMatrice) => renvoie la matrice sous forme de grille a jouer.

Une image contenant texte

Description générée automatiquement

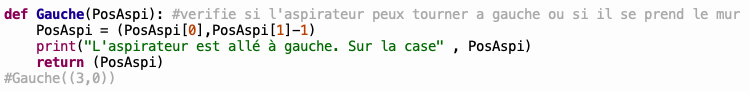
* Salete(Matrice,PourcentageSalete, TailleMatrice) => renvoie la matrice avec la saleté mis, aléatoirement, en fonction du pourcentage.

Une image contenant texte

Description générée automatiquement

* VerifAspiration(Matrice,PositionAspirateur) => renvoie True or False en fonction si la case de l’aspirateur dans la matrice est propre ou salle, si elle est salle l’aspirateur aspire et la case devient propre. Une image contenant texte

  Description générée automatiquement
* Droite(PositionAspirateur) => renvoie la nouvelle position de l’aspirateur après le déplacement. Respectivement pour Gauche, Bas et Haut.



* PremierCornerWithOutObstacle (PositionAspirateur,NbDeplacement) => Ne s’utilise que pour les agents sans obstacles, renvoie la nouvelle position de l’aspirateur (0,0) , le nombre de déplacement qu’il a fallu pour aller en position (0,0) et la Liste de toutes les positions où l’aspirateur est allé.

Une image contenant texte

Description générée automatiquement

* PremierCornerWithObstacle(PosAspi,NbDeplacement,ListePosObstacle,N) => Ne s’utilise que pour les agents avec obstacles, renvoie la même chose que la fonction précédente ; la différence entre les deux fonctions est que j’ai adapté celle-ci pour éviter les obstacles lors du chemin.

Une image contenant texte

Description générée automatiquement

* Afficher(Matrice, PositionAspirateur,TailleMatrice) => renvoie la matrice avec un croix rouge sur la case ou il y’a l’aspirateur.

Une image contenant texte

Description générée automatiquement

## Explication de mes Agents :

* AgentReflexeSimpleRandom(UniteTime,TailleMatrice,PourcentageS) =>
  + Créer une matrice propre
  + Met de la saleté
  + Génère une position aléatoire de l’aspirateur
  + Vérifie la ‘propreté’ de la case
  + Nettoie si sale
  + Tant que la matrice globale = ! de la matrice propre
  + Il se déplace aléatoirement, et recommence la vérification.
  + La matrice s’affiche seulement quand l’aspirateur aspire. C’est un Agent Reflexe Simple donc il se prend les murs. A chaque déplacements, aspirations, ou quand il se prend un mur => l’unité de temps prend -1.
  + Quand plus d’unité de temps => break
  + Print une jolie phrase de fin
* AgentReflexeAvecEtat(UniteT,N,PourcentageS) =>
  + Créer une matrice propre
  + Met de la saleté
  + Génère une position aléatoire de l’aspirateur
  + Si la matrice globale = ! de la matrice propre
  + L’aspirateur va dans le coin de position (0,0)
  + Vérifie la ‘propreté’ de la case
  + Nettoie si sale
  + Tant que la matrice globale = ! de la matrice propre
  + Il se déplace en mode ‘serpent’, et recommence la vérification.
  + La matrice s’affiche seulement quand l’aspirateur aspire. C’est un Agent Reflexe Avec Etat donc il sait être près des murs et l’évite sans prendre un ‘malus de temps’.
  + Et il retient toutes les cases où la été.
  + A chaque déplacements, aspirations, ou quand il se prend un mur => l’unité de temps prend -1.
  + Quand plus d’unité de temps => break
  + Print une jolie phrase de fin
* AgentReflexeSimple(UniteT,N,PourcentageS) =>
  + Comme l’AgentReflexeSimpleRandom(UniteTime,TailleMatrice,PourcentageS) sauf que ces deplacement sont en mode ‘serpent’. Donc il va à la case (0,0).
* AgentReflexeAvecEtatAvecObstacle(UniteT,N,PourcentageS) =>

J’ai definin mes obstacles de tel sorte qu’il ne peut en avoir 2 à côté.

* + Créer une matrice propre
  + Créer des obstacles (représenté par des petits carrées)
  + Met de la saleté
  + Génère une position aléatoire de l’aspirateur
  + Vérification qu’il n’arrive pas sur un obstacle
  + Si la matrice globale = ! de la matrice propre
  + L’aspirateur va dans le coin de position (0,0)
  + Vérifie la ‘propreté’ de la case
  + Nettoie si sale
  + Tant que la matrice globale = ! de la matrice propre
  + Commence son chemin en mode ‘serpent’
  + Si il capte un obstacle devant lui, il fait un détour.
  + Il évite les murs.
  + Et il retient toutes les cases où la été.
  + A chaque déplacements, aspirations, ou quand il se prend un mur => l’unité de temps prend -1.
  + Quand plus d’unité de temps => break
  + Print une jolie phrase de fin

## Questions :

1. Python
2. AgentReflexeSimple(UniteT,N,PourcentageS) =>

* Matrice 6x6, Salissure 100% en 80 Unités de Temps.
* Matrice 6x6, Salissure 75% en 72 Unités de Temps.
* Matrice 6x6, Salissure 50% en 68 Unités de Temps.
* Matrice 6x6, Salissure 25% en 51 Unités de Temps.
* Matrice 6x6, Salissure 0% en 0 Unités de Temps.

1. Considérer une version modifiée de l’environnement où l’agent est pénalisé́ d’un point pour chaque mouvement. => Déjà comme ceci au-dessus.

Est-ce qu’un agent réflexe simple peut être parfaitement rationnel pour cet environnement ? => Pénalisé ou pas, lui ne s’en rend pas compte mais ce n’est pas efficace.

1. AgentReflexeAvecEtat(UniteT,N,PourcentageS) =>

Est-ce qu'il peut être parfaitement rationnel pour l'environnement de la question 3 ? => Cette agent est plus efficace.

1. Pyhton
2. Est-ce qu’un agent réflexe simple peut être parfaitement rationnel pour l’environnement de la question 5 ? Justifiez. => Non encore moins car là dès qu’il se prend un mur sont nombre d’unité de temps diminue encore plus.
3. Est-ce qu’un agent réflexe simple avec une fonction randomisée peut faire mieux qu’un agent réflexe simple ? Réalisez-le et faites des simulations pour répondre à cette question. => AgentReflexeSimpleRandom(UniteTime,TailleMatrice,PourcentageS).
   * Matrice 6x6, Salissure 100% en 268 Unités de Temps.
   * Matrice 6x6, Salissure 75% en 334 Unités de Temps.
   * Matrice 6x6, Salissure 50% en 213 Unités de Temps.
   * Matrice 6x6, Salissure 25% en 290 Unités de Temps.
   * Matrice 6x6, Salissure 0% en 0 Unités de Temps.

On voit ici que cette fonction n’est pas du tout logique ou rationnel, elle n’est que aléatoire.

1. Peut-on dessiner un environnement dans lequel l'agent randomisé aura une mauvaise performance (pire que celle de l'agent non-randomisé) ? => Déjà le cas.
2. Est-ce qu'un agent reflexe avec état peut faire mieux que l'agent reflexe simple ? Réalisez- le et faites des simulations pour répondre à cette question. => AgentReflexeAvecEtat(UniteT,N,PourcentageS)
   * Matrice 6x6, Salissure 100% en 75 Unités de Temps.
   * Matrice 6x6, Salissure 75% en 65 Unités de Temps.
   * Matrice 6x6, Salissure 50% en 54 Unités de Temps.
   * Matrice 6x6, Salissure 25% en 51 Unités de Temps.
   * Matrice 6x6, Salissure 0% en 0 Unités de Temps.

On voit que l’agent avec état est plus efficace que le simple si il ‘a bcp de salissure, sinon il s’équivaut.

1. Repetez les experiences precedentes en remplacant la capteur de position de l'agent avec un capteur d'obstacle, qui detecte si l'agent essaye de passer sur un obstacle ou de traverser le bord de l'environnement. => AgentReflexeAvecEtatAvecObstacle(UniteT,N,PourcentageS)
   * Matrice 6x6, Salissure 100% en 75 Unités de Temps.
   * Matrice 6x6, Salissure 75%, 5 obstacle en ? Unités de Temps.
   * Matrice 6x6, Salissure 50%, 5 obstacle en 58 Unités de Temps.
   * Matrice 6x6, Salissure 25%, 5 obstacle en 52 Unités de Temps.
   * Matrice 6x6, Salissure 0%, 5 obstacle en 0 Unités de Temps.

# TD2 :

## Questions 1 :

1. Décidez comment représenter les croyances de l'agent sur l'état de l'environnement. Ce qu'il faut représenter est la position à laquelle l'agent croit être et si chaque case est sale ou propre. Supposons pour l'instant que l'agent a une connaissance innée de la topographie de l'environnement. =>

Le problème est que dans l’introduction on nous dit que l’agent ne sais pas si une case est propre ou sale.

Faisons comme si l’agent connait la topographie et la saleté. Une image contenant texte

Description générée automatiquement

1. Codez la fonction (ou méthode) actualiser-état, qui, étant données la dernière action exécutée et la perception courante, actualise la représentation interne de l'état de l'environnement ; le code de cette fonction pourra être considérée le "modèle" qui décrit la façon dont l'état suivant dépend de l'état courant et de l'action effectuée. =>
   1. Une image contenant texte

      Description générée automatiquement
2. Modifiez votre agent pour que son comportement soit décrit par des règles condition- action : comment peut-on représenter ces règles dans notre cas ?
3. Codez la fonction (ou méthode) trouver-règle, qui, étant donné la représentation interne de l’état, renvoie la règle à utiliser.
4. Écrivez les règles qui déterminent le comportement de l’agent.

## Questions 2 = Non deterministe :

1. A chaque fois que l’action aspirer est effectuée, il y a une probabilité de 20% que cette action échoue (la propreté de la case ne change pas) => J’ai du modifie ma fonction verification() Une image contenant texte

   Description générée automatiquement

Ansi que la manière dont j’utilise cette fonction dans la fonction agent , tant que l’aspiration echoue l’aspirateur re-essaie.

Une image contenant texte

Description générée automatiquement

1. A chaque fois que l’action de bouger est effectuée, il y a une probabilité de 10% que les roues glissent et que la position de l’aspirateur reste inchangée => J’ai du modifier tous les fonctions de Deplacement , Droite, Gauche , Haut et Bas tel que :

Une image contenant texte

Description générée automatiquement

Le Nombre de deplacement prend +1 automatiquement et directement grace a commenet j’ai ecrit a ma fonction de l’agent .

Une image contenant texte

Description générée automatiquement

1. A chaque instant, chaque case propre a une probabilité de 5% de devenir sale. =>

Une image contenant texte

Description générée automatiquement

Du coup il faut modifier le fonctionnnement de l’agent ,

Si cette nouvelle case sale est sur une position antérieur a l’agent alors l’aspirateur va directement nettoyer cette case , avec une Verification() .

Une image contenant texte

Description générée automatiquement

## Questions 3&4 :

1. Modifiez les règles qui determinent le comportement de l’agent pour gérer de manière rationnelle ce nouvelle version de l’environnement.
2. Faisons tomber l’hypothèse que l’agent a une connaissance innée de la topographie de l’environnement, de sorte à ce qu’il doive la découvrir. Modifiez la représentation interne de l’agent e sa fonction (ou méthode) actualiser-état pour gérer ce cas.