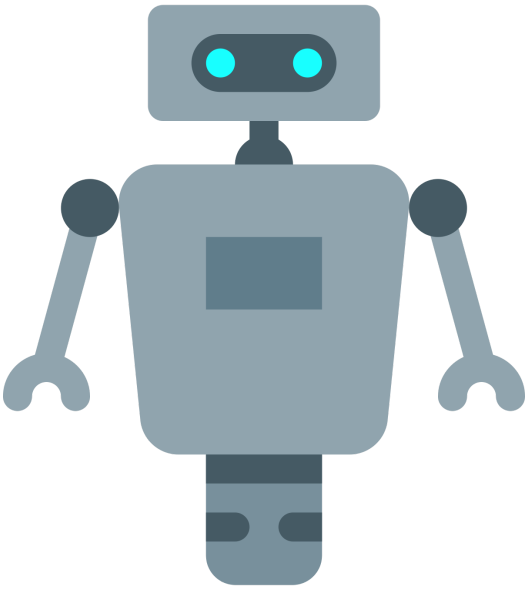
8INF846 – Intelligence Artificielle

Travail No. 1
Création d'un agent aspirateur



16 octobre 2017

Esmé James JAME15539504
Wilfried Pouchous POUW04069501

Sommaire

1. Organisation des fichiers du projet

Le projet est constitué des fichiers d'en-tête suivants:

- **targetver.h**
- **stdafx.h** qui contient toutes les importations de bibliothèques, les includes du projets et les defines
- **Environnement.h** qui contient la classe Environnement
- **AgentAspirateur.h** qui contient la classe AgentAspirateur et une structure servant à définir un noeud (utilisé pendant l'exploration)
- **InterfaceGraphique.h** qui contient la classe InterfaceGraphique permettant de mettre en place l'affichage SDL

Le projet est constitué des fichiers sources suivants:

- **stdafx.cpp**
- **AgentAspirateur.cpp** qui contient la définition des fonctions de la classe AgentAspirateur
- **Environnement.cpp** qui contient la définition des fonctions de la classe Environnement
- **InterfaceGraphique.cpp** qui contient la définition des fonctions mettant en place l'affichage SDL
- **main.cpp** qui est le point d'entrée du programme et qui contient la définition des fonctions update des threads agent et environnement

Le projet est constitué des ressources externes suivantes :

- **bijou.bmp** qui est l'image bitmap d'une case avec un bijou
- **both.bmp** qui est l'image bitmap d'une case avec une poussière et un bijou
- **case.bmp** qui est l'image bitmap d'une case vide
- **poussiere.bmp** qui est l'image bitmap d'une case avec une poussière
- **robot.bmp** qui est l'image bitmap d'une case avec un robot

Un **READ.me** est également présent dans le dossier contenant les sources du projet.

2. Environnement

2.1. Propriétés

L'environnement dans lequel évolue l'agent est :

- **Complètement observable**
- **Stochastique** car le prochain état de l'environnement ne dépend pas de son état courant
- **Épisodique** car le prochain épisode ne dépend pas des actions précédentes effectuées
- **Dynamique** car la carte de l'environnement peut changer pendant le cycle de vie de l'agent
- **Continu** car l'environnement est mis à jour en temps réel
- **Un seul agent**

2.2. Description

L'environnement est une carte quadrillée composée de 10*10 cases comme indiqué dans l'énoncé du travail. Il est lancé dans une boucle de vie infinie dans laquelle il génère des poussières et des bijoux aléatoirement sur la carte.

2.3. Génération aléatoire des éléments sur la carte

Lors de la première génération de poussière et/ou de bijoux sur la carte, l'environnement génère de la poussière avec une probabilité de **1/30** et un bijou avec une probabilité de **1/50**. Au tours suivants, l'environnement vérifie que la carte n'est pas déjà pleine puis ajoute des poussières avec une probabilité de **1/200** et des bijoux avec une probabilité de **1/300**. Cette différence entre le premier tour et les tours suivants a été mise en place pour que le robot ne soit pas débordé par une génération trop intense de poussière et de bijoux.

3. Agent

3.1. Propriétés

L'agent aspirateur est un agent basé sur les buts. Il peut se formuler par un problème à simple état :

- **Etat initial** : Le robot est positionné en (0,0) soit dans le coin supérieur gauche, et à l'état « Ne rien faire ».
- **Opérateurs** : Ne rien faire, Aspirer Poussière, Ramasser bijou, Gauche, Droite, Haut, Bas
- **Fonction de succession {Action, État}** :
  { Droite, Robot déplacé d'une case à droite }
  { Haut, Robot déplacé d'une case en haut }
  { Bas, Robot déplacé d'une case en bas }
  { Gauche, Robot déplacé d'une case à gauche }
  { Aspirer, Carte beliefs mise à jour }
  { Ramasser, Carte beliefs mise à jour }
  { NeRienFaire, Robot attend }
- **Test de but** : pas de saleté et pas de bijou au sol
- **Coût du chemin** : 1 unité par action

3.2. Fonctionnement de l'agent

Le cycle de vie de l'agent est une boucle infinie qui se déroule comme suit :

1. Observation de l'environnement : l'agent lit la carte de l'environnement et l'affecte à ses croyances
2. Mise à jour de ses croyances : il se place sur la carte
3. Affichage de ses croyances
4. Effectue un test de désir : compare ses croyances à une carte de l'environnement vide
5. Choix d'une action (ne fait rien si le désir est respecté) : Appel d'un algorithme d'exploration
6. Récupération des intentions trouvées
7. Boucle de parcours des intentions trouvées
   a. Effectue l'action
   b. Mise à jour de ses croyances
   c. Affichage de ses croyances
8. Mesure de sa performance

3.3. Modélisation des actions

- **Action “Ramasser Bijou”**
  Prémisses :
    Case contenant un bijou
    Robot positionné sur cette case
  Conséquences :
    Bijou ramassé
    Case vidée
- **Action “Aspirer Poussière”**
  Prémisses:
    Case contenant une poussière
    Robot positionné sur cette case
  Conséquences :
    Poussière aspirée
    Case vidée
- **Action “Se déplacer” (Haut, Bas, Gauche, Droite)**
  Prémisses:
    Case suivante sur la carte
  Conséquences :
    Déplacement du robot d’une case
- **Action “Ne Rien Faire”**
  Prémisses :
    Toutes les cases de la carte sont vides
  Conséquences :
    Le robot attend

3.4. Modélisation de la perception

L’agent peut percevoir son environnement en observant une carte de l’environnement à un moment donné. Il utilise ensuite ses croyances pour explorer la carte et percevoir les éléments tels que la poussière et les bijoux.

3.5. BDI

L’agent aspirateur possède en tout temps trois attributs: beliefs, desire et intentions. Ces attributs sont implémentés de la manière suivante :

- **Beliefs** : l’agent possède en tout temps, une carte issue de son observation de l’environnement
- **Desire** : l’agent possède une carte de l’environnement vide qu’il utilisera afin d’évaluer son travail en le comparant à ses beliefs
- **Intentions** : l’agent possède une liste d’actions à effectuer pour arriver à son but

4. Exploration Non-Informée

Nous avons choisi de mettre en place une exploration **Breadth-First Search** (recherche en largeur) pour notre exploration non informée.

Étapes de notre algorithme d'exploration non informée :

1. Création d'un noeud racine contenant la position du robot

2. Création d'un file (FIFO) contenant les noeuds voisins

3. Parcours d'une boucle tant que la file n'est pas vide

4. A chaque tour, on regarde les cases voisines du noeud courant en gardant en mémoire le noeud parent et on ajoute, si possible, ces noeuds voisins à la file.

5. Si le noeud courant correspond à une case non vide (⇔ poussière ou bijou ou les deux) de la carte, on arrête le parcours de la file.

6. Parcours des noeuds parents à partir du noeud solution trouvé afin de créer une liste d'actions du robot pour arriver à la solution

7. Inversion de la liste d'action et envoi aux intentions du robot

5. Exploration Informée

Nous avons choisi de mettre en place une exploration **Best-First Search** pour notre exploration informée. Pour cela, nous utilisons la **distance de Manhattan** pour évaluer le noeud le plus désirable. A chaque tour dans l'exploration, on choisit le noeud le plus désirable, c'est-à-dire celui avec la distance de Manhattan la plus petite du but et on garde en mémoire une liste d'actions menant aux noeuds désirables choisis.

La distance de Manhattan correspond à la distance parcourue entre deux points sur un quadrillage. Elle se calcule avec la formule: $\Delta M = |xA - xB| + |yA - yB|$.

Étapes de notre algorithme d'exploration:

1. Création d'un noeud racine contenant la position du robot

2. Parcours de la carte en calculant la distance de Manhattan entre la racine et chaque élément non vide (⇔ poussière ou bijou)

3. Sauvegarde du noeud but (position et contenu) le plus proche et de sa distance

4. Parcours des voisins possibles pour arriver au noeud but : on évalue quel voisin permet de se rapprocher le plus du noeud but (dans le cas de deux voisins à la même distance du but, on prend le premier trouvé) et on ajoute l'action à faire pour l'atteindre dans une liste

5. Envoi de la liste d'actions aux intentions du robot

6. Mesure de performance

Nous avons également implémenté une mesure de performance. Comme énoncé dans les spécifications du travail, le robot aspirateur dépense 1 unité d'électricité à chaque action réalisée. Nous avons également décidé d'attribuer un système de points à l'agent, celui-ci fonctionne de la manière suivante :

- L'agent gagne un point lorsqu'il ramasse un bijou ou aspire une poussière
- L'agent perd un point s'il aspire un bijou par erreur

La mesure de performance s'effectue à chaque fois que **10 actions** de nettoyage ont été réalisées par l'agent.

Notions utilisées:

- *Rapport de performance* = Nombre de points / Quantité d'électricité dépensée
- *Pourcentage d'exploration* = Pourcentages des intentions du robot réalisées avant de re-évaluer son environnement. Il effectue 100% de ses intentions au départ.
- *Progression d'exploration* = Quantité des actions réalisées par le robot avant de re-évaluer son environnement. Il est calculé en multipliant le pourcentage d'exploration avec le nombre total d'intentions.
- *Précision de performance* = Pourcentage variable permettant d'affiner le rapport de performance. Elle influe sur le pourcentage d'exploration. Elle commence à 10%.

Étapes de la mesure de performance:

1. On vérifie que le robot a réalisé 10 actions de nettoyage
2. Si c'est le cas, on calcul le rapport de performance
3. On associe le rapport de performance à un pourcentage d'exploration (100% au début) et on l'ajoute dans un tableau qui garde en mémoire les deux derniers rapports de performance
4. Deux cas se distinguent ensuite :
    a. Le cas où le rapport est meilleur que le précédent. Dans ce cas, on diminue le pourcentage d'exploration d'une précision calculée (précision de performance)
    b. Le cas où le rapport est moins bon que le précédent. Dans ce cas, on augmente le pourcentage d'exploration d'une précision calculée (précision de performance)
5. En fonction du nombre d'actions déterminées par l'agent pour arriver au but, on calcule la quantité d'actions qu'il effectuera en multipliant le nombre d'actions par le pourcentage d'exploration déterminé plus tôt.

Exemple :

Au départ, le robot effectue 10 actions de nettoyage à 100%. Il compare son rapport de performance au précédent et le trouve meilleur. Il baisse donc son pourcentage d'exploration de 10% ; de ce fait, lors de sa prochaine exploration, il effectuera 90% de ses actions. Si la prochaine exploration indique 8 actions à effectuer, il en réalisera alors $8 * 0.9 \simeq 7$ avant de ré-observer son environnement.

Cas particuliers :

- Si le robot trouve 3 actions ou moins à réaliser lors de son exploration, même s'il trouve un meilleur rapport, il effectuera tout de même 100% de ses intentions afin d'éviter les boucles
- Nous avons également mis en place une variable permettant d'améliorer au fur et à mesure la précision du pourcentage d'exploration, plutôt que de l'augmenter ou de le diminuer de 10% à chaque fois.

Pour l'amélioration de la précision du pourcentage d'exploration, nous utilisons un booléen correspondant au sens de parcours du tableau. Ce booléen change à chaque changement de sens, c'est-à-dire à chaque fois qu'on varie entre rapport meilleur et rapport moins bon. Si le sens a changé, alors la précision du pourcentage d'exploration sera diminuée par 2. Ainsi, une progression du pourcentage d'exploration pourrait être : $100\% - 90\% - 80\% - 85\% - 82,5\% - 83,75\% - \ldots$.

7. Affichage graphique

Nous avons implémenté un affichage graphique en utilisant la librairie SDL 2. De ce fait, lors de l'exécution de notre programme, il est demandé à l'utilisateur s'il souhaite afficher le programme en console ou en graphique.

Il est également demandé à l'utilisateur quelle type d'exploration il souhaite effectuer.