

## 8INF846 – Intelligence Artificielle – Hiver 2022

### Travail Pratique #1

### Création d'un agent aspirateur












## Le contexte

Vous êtes un programmeur engagé dans une entreprise qui construit des robots dotés d'intelligence artificielle pour les activités ménagères. Son nouveau projet est de développer un agent intelligent qui composera un aspirateur robot. L'agent/robot s'appellera *Aspirobot T-0.1* et ses objectifs sont d'aspirer la poussière d'une résidence et de récupérer les bijoux perdus sur le sol.

## Détails de l'environnement

Les maisons dans lesquelles le robot va travailler ont plusieurs pièces. De cette façon, pour simuler l'environnement d'une maison à partir de laquelle le robot travaillera, vous utiliserez une matrice 5x5, où chaque cellule représente une pièce de la maison, totalisant 25 pièces, comme l'illustre l'image ci-dessous. L'environnement sera chargé de contrôler la génération de poussière et des bijoux, de façon sporadique et dans des pièces aléatoires. Pour programmer l'environnement, vous devez utiliser un *thread* qui exécute une boucle infinie (selon le pseudocode ci-dessous) ou qui déclenche des événements sporadiquement. Vous pouvez choisir la fréquence et la probabilité de génération de poussière et des bijoux. Une pièce peut contenir de la poussière et/ou des bijoux en même temps.

```
while thread_is_running:
    if should_generate_dirty:
        generate_dirty()
    if should_generate_jewel:
        generate_jewel()
```

## Détails de l'agent

Les fonctions de l'agent Aspirobot T-0.1 sont les suivantes : aspirer (poussière ou poussière et bijou, lorsque les deux sont dans la même pièce), collecter le bijou (lorsque seule le bijou est dans la pièce), déplacer vers le haut, le bas, la gauche et la droite. Le robot ne se déplace pas en diagonale. A chaque cycle d'analyse (boucle), l'agent peut voir toutes les pièces. C'est-à-dire que l'agent est toujours au courant de l'état de l'environnement avant d'exécuter une exploration. Pour cela, l'agent doit observer l'environnement à travers ses capteurs puis effectuer des actions sur l'environnement à travers ses actionneurs. L'agent doit être de type objectif et avoir son état mental modélisé sous la forme d'un état BDI. De plus, pour programmer l'agent, vous devez utiliser un *thread* qui exécute une boucle infinie, comme indiqué dans le pseudocode ci-dessous.

```
while am_i_alive: #This is the analysis cycle
    environment = observe_environment_with_my_sensors()
    state_bdi = update_my_state_bdi(environment, metric)
    plan_action = execute_exploration(state_bdi, algo-informed or algo-not-informed)
    metric = execute_action_plan(plan_action)
```

L'agent doit mettre en œuvre **deux algorithmes d'exploration** pour réaliser le plan d'action : un **algorithme d'exploration non informé** et un **algorithme d'exploration informé**. Initialement, dans les premiers cycles (X cycles équivaut à un épisode), l'agent doit effectuer l'exploration non informée. Lors d'une exploration non informée, l'agent doit alimenter une métrique de performance (exemples de métriques : distance, nombre d'actions, électricité consommée, pénalités, entre autres que vous pouvez inventer) et la stocker dans son état mental BDI. A partir de X itérations et/ou d'un nombre Y de la métrique stockée dans le BDI, l'agent dispose d'un processus d'apprentissage et à partir de celui-ci il pourra exécuter une exploration informée en utilisant l'heuristique. Vous êtes libre de concevoir une heuristique. A titre de suggestion, l'heuristique peut être :

- Basé sur moins d'actions
- Basé sur des pénalités pour avoir mal passé l'aspirateur sur des bijoux
- Basé sur les distances (euclidienne, manhattan, etc.)
- Basé sur la performance par une consommation d'électricité plus faible
- Si vous le souhaitez, vous pouvez combiner les suggestions ci-dessus

Algorithmes d'exploration non informés pouvant être utilisés :

- *Tree-Search*
- *Graph Search*
- *Breadth-first search* (largeur)
- *Uniform-cost search* (coût uniforme)
- *Depth-first search* (profondeur)
- *Depth-limited search* (profondeur limitée)
- *Iterative deepening search* (itérative en profondeur)
- *Bidirectional search* (bidirectionnelle)

Algorithmes d'exploration informé pouvant être utilisés :

- *Best-First Search*
- *Greedy Search*
- *A\* Search*
- *IDA\**
- *Recursive Best-First Search*
- *Simplified Memory Bounded A\**

## Des contraintes à respecter

- Le robot doit être mis en œuvre en utilisant les principes des agents vus en classe, c'est-à-dire en utilisant des classes pour modéliser des capteurs et des classes pour modéliser des actionneurs.
- L'agent et l'environnement doivent s'exécuter sur deux *threads* distincts dans le même programme.
- La poussière et les bijoux doivent être générées sporadiquement et dans des pièces aléatoires par le *thread* d'environnement.
- Si une pièce contient à la fois des bijoux et de la poussière, le robot doit aspirer les bijoux à tort.
- L'agent doit exécuter l'exploration pour générer le plan d'action. Votre objectif est d'atteindre un état où chaque pièce est propre et sans bijoux.
- La modélisation de l'état mental interne de l'agent doit se faire sous forme de BDI (*Beliefs-Desires-Intentions*). Cette modélisation doit être répartie dans la structure de l'arbre d'exploration.
- Les explorations doivent être effectuées en utilisant la structure de données de l'arbre, où une nouvelle action implique l'expansion du nœud de l'arbre. Le plan d'action qui en résulte est la branche de l'arbre, c'est-à-dire la séquence de nœuds de la racine de l'arbre à la feuille qui atteint l'objectif.
- Dans une exploration informée, l'agent doit utiliser une heuristique.
- L'heuristique doit être alimentée par une métrique générée lors d'une exploration non informée.
- L'agent doit choisir quand exécuter l'exploration informée à partir de l'apprentissage obtenu par la métrique collectée lors de l'exploration non informée.

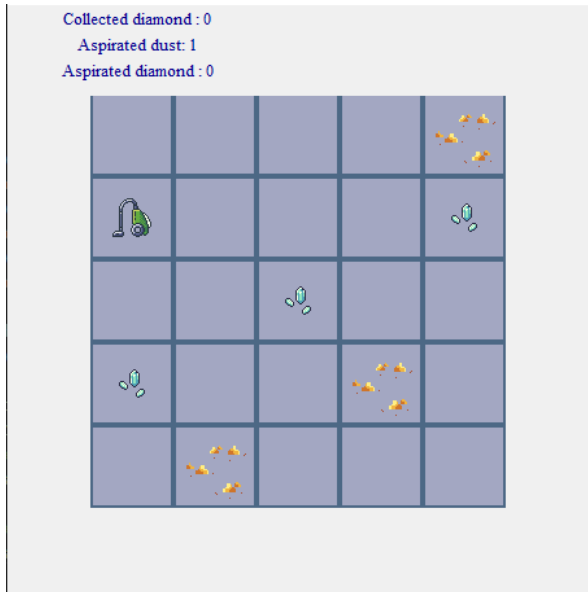
## Entrée et sortie

**Entrée :** Aucune entrée n'est requise. Une fois le programme exécuté, l'environnement commence à générer de la poussière et des bijoux et le robot commence à effectuer la procédure de nettoyage.

**Sortie :** Vous pouvez créer une interface graphique ou présenter une sortie de texte dans la console. L'écran de présentation doit contenir la matrice 25 positions, le robot, la poussière, les bijoux, le plan d'action en cours d'exécution, l'action réalisée et la métrique générée, et quelle

exploration a été effectuée. L'écran doit se mettre à jour à chaque action effectuée par le robot (aspirateur, ramasser ou déplacer) et à chaque action de l'environnement (génération de bijou ou de poussière).

Exemple de capture d'écran avec interface graphique :



Exemple d'écran avec du texte dans la console :

```
Agent : Remaining energy 100
Manoir :
[1, 0, 1, 0, 0]
[0, 0, 0, 0, 0]
[2, 0, 10, 1, 0]
[0, 0, 0, 0, 0]
[0, 1, 3, 0, 0]
Agent : Observation
Agent : Temps avant scan 15
Agent : Planification
Recherche NonInformé start
AVANT WHILE
Recherche Noninformé end
Agent : Plan d'action
[13, 13, 12, 7, 2, 2, 1, 0, 0, 5, 10, 10, 11, 16, 21, 21, 22, 22]
Agent : Réalisation
Agent : déplacer 13
Environnement : move [2, 3]
Manoir :
[1, 0, 1, 0, 0]
[0, 0, 0, 0, 0]
[2, 0, 0, 11, 0]
[0, 0, 0, 0, 0]
[0, 1, 3, 0, 0]
Agent : aspirer
Environnement : aspire
Manoir :
[1, 0, 1, 0, 0]
[0, 0, 0, 0, 0]
[2, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 1, 3, 0, 0]
Agent : déplacer 7
Environnement : move [1, 2]
Manoir :
[1, 0, 1, 0, 0]
[0, 0, 10, 0, 0]
[2, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 1, 3, 0, 0]
Agent : déplacer 2
Environnement : move [0, 2]
Manoir :
[1, 0, 11, 0, 0]
[0, 0, 0, 0, 0]
[2, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 1, 3, 0, 0]
Agent : aspirer
Environnement : aspire
Manoir :
```

## Autres informations importantes

- La modélisation et la mise en œuvre doivent être cohérentes avec les concepts vus en classe. Vous perdrez des points si les modèles sont inadaptés ou si l'algorithme est copié d'internet, même s'il fonctionne parfaitement.
- Le langage de programmation le plus recommandé est **Python**. Si vous souhaitez utiliser une autre langue, vous devez demander l'approbation du professeur.
- Vous devez former un groupe de **3 ou 4 personnes**.

## Soumission du travail

- Vous devez soumettre une **copie électronique de votre code source**.
- Un **rapport** (en format .pdf) contenant les sections suivantes :
  - Page de titre avec **les noms, les code permanents et le titre du cours** ;
  - Section expliquant **comment exécuter votre programme** : quelle version de langue (Python, Java, etc.) doit être installée, quelles bibliothèques supplémentaires doivent être installées, paramètres supplémentaires, etc. Indiquez quel fichier .py exécuter. Attention : si le professeur a besoin de modifier une ligne de votre code pour exécuter votre programme, vous perdrez des points.
  - Section expliquant **comment vous avez modélisé la solution**, comment les *threads* communiquent, etc. Vous pouvez utiliser des schémas si vous préférez. Essayez d'utiliser le standard (par exemple *Unified Modeling Language* - UML) pour faire vos diagrammes.
  - Section avec **une capture d'écran du code source** de chacun des éléments ci-dessous et une brève explication de chacun :
    - L'algorithme d'exploration informée
    - L'algorithme d'exploration non informée
    - Le test de décision consistant à choisir entre une exploration informée et non informée à partir de l'apprentissage
    - La collecte/génération de métriques pour l'apprentissage
    - L'heuristique
    - Le test (IF, FOR) de l'objectif
    - Modélisation de l'état mental BDI dans l'arbre d'exploration
    - Modélisation de capteurs et d'actionneurs.
- La **soumission** doit être **sur Moodle** uniquement.
- Une **seule soumission par groupe**.

**Date limite de remise : 14 février 2022 à 23h59 (heure de Chicoutimi, QC, Canada)**

**Bon travail !**