8INF878 – Intelligence Artificielle
Travail No. 1
Création d'un agent aspirateur

Travail à effectuer (sommaire):

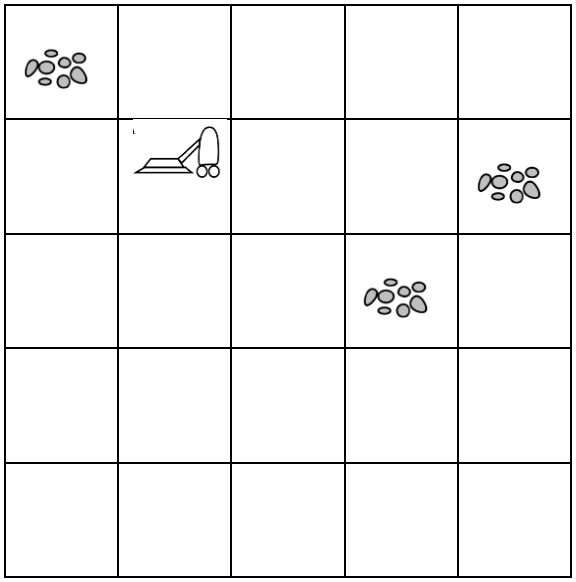
Vous possédez un grand manoir constitué de nombreuses pièces qui ne cessent de constamment se salir. Depuis que vos employés d'entretien se sont syndiqués, ils demandent à travailler des heures épouvantables (de 8 à 16 vous imaginez!) et d'obtenir des pauses durant la journée. Tant pis pour eux! Vous décidez de commander un Aspirobot T-0.1 enfin de remplacer tous ces paresseux. Après tout, la compagnie vous a garanti que celui-ci éviterait d'aspirer vos bijoux dispendieux (ce serait triste de perdre un diamant à 15 000$ pour un peu de poussière) et que de façon automatique il maintiendrait un maximum de pièces propres en tout temps sans gaspiller d'électricité!

Détails du travail à effectuer :

Environnement :

L'environnement dans lequel évolue l'agent Aspirobot T-0.1 sera constitué d'une carte contenant 25 pièces disposées telles qu'illustrées ci-dessous. Tel que défini au cours, l'environnement devrait contenir tous les éléments passifs de votre programme (carte, poussière, bijoux, etc.). L'environnement devrait aussi contenir minimalement une mesure de performance que l'agent peut consulter (via ses capteurs). L'environnement devrait être une boucle infinie (ou encore un événement programmé pour s'exécuter sporadiquement) qui aléatoirement génère soit de la poussière dans une case aléatoire ou encore un bijou. C'est à vous de décider des probabilités. Une case peut contenir à la fois de la poussière et un bijou.

```
While (gameIsRunning()){
    If (shouldThereBeANewDirtySpace())
        GenerateDirt()
    If (shouldThereBeANewLostJewel())
        GenerateJewel()
}
```



Agent :
L'agent Aspirobot T-0.1 possède un certain nombre de fonctionnalités. Il peut aspirer (ce qui a pour effet d'aspirer à la fois la poussière et les bijoux), ramasser un élément (pour ramasser un bijou idéalement!) et se déplacer (haut, bas, gauche et droite). Chacune de ses actions lui coûte 1 unité d'électricité (attention, rappelez-vous que vous êtes un propriétaire de manoir ~~cheap~~ heum… disons qui n'aime pas gaspiller de l'argent). Le robot peut voir toutes les pièces. Le robot doit être implémenté sous les principes agents vus en classe. Il doit donc observer l'environnement avec ses capteurs et agir sur celui-ci avec des effecteurs (je vous conseille carrément de faire des classes capteurs et effecteurs). Votre robot devrait être de type « basé sur les buts ». Même si le problème est très simple, il est important que celui-ci contienne les éléments de base (son état interne devrait contenir un état mental sous la forme BDI « Beliefs-Desires-Intentions »). Il devrait lui aussi être implémenté comme une boucle infinie respectant l'idée de base vue en cours (voir pseudo-code simple ci-dessous).

```
While (amIAlive()){
    ObserveEnvironmentWithAllMySensors()
    UpdateMyState()
    ChooseAnAction()
    justDoIt()
}
```

Finalement, votre agent doit implémenter **deux** algorithmes d'exploration (un non-informé et un informé) afin de planifier ses actions. L'exploration doit être implémentée en respectant les éléments de bases vues en classe et rester entièrement **générique** (sauf l'heuristique). Au départ, votre agent exécute son plan complet (à partir de ses *intentions*) sans percevoir de nouveau l'environnement. Puis, grâce à sa mesure de performance, après X itérations (à apprendre), il peut entrelacer l'exécution du plan de séquences de perception et d'une nouvelle phase d'exploration. L'agent devrait tenter d'optimiser la fréquence d'exploration.

Contraintes à respecter :

1. L'agent et l'environnement doivent s'exécuter sur deux fils d'exécution différents.
2. De la poussière et des bijoux doivent être générés sporadiquement par l'environnement.
3. L'agent dépense une unité d'électricité par action *exécutée*.
4. Si l'agent aspire et qu'un bijou se trouve à cet endroit, l'environnement devrait considérer que celui-ci a été aspiré (le robot n'a pas nécessairement à le savoir, puisqu'il ne l'avait pas perçu) – Il perd des points pour ça (dans sa mesure de performance)!
5. L'agent doit posséder un état mental BDI.
6. L'agent doit utiliser l'exploration afin de planifier ses actions. Son but est d'atteindre un état *propre*. Vous avez le droit d'utiliser une technique pour éviter les boucles.
7. L'agent doit avoir accès à une heuristique pour effectuer de l'exploration informée.
8. L'agent doit apprendre la meilleure fréquence d'exploration par rapport à sa mesure de performance (vous pouvez faire un apprentissage épisodique).

L'implémentation devra être cohérente avec les notions vues en classe. La modélisation des agents, de l'environnement, des actions, etc., devra s'appuyer sur les modèles vus en classe. ATTENTION! Vous perdrez des points si vos modèles sont inadéquats, même s'ils fonctionnent parfaitement.

Vous devez me remettre une copie électronique de votre code, ainsi qu'un exécutable Windows ou encore une vidéo montrant le fonctionnement du logiciel. De plus, vous devez inclure une page titre avec vos noms, votre code permanent et le titre du cours. Votre travail devra être accompagné d'un rapport expliquant votre modélisation et comment chacun des éléments est conforme à l'architecture présentée dans les notes de cours. Ce rapport devra positionner clairement votre implémentation : décrire le type d'environnement (propriétés), le type d'agent, la fonction d'agent, modélisation de l'action, de la perception, etc.

Si vous avez des indications particulières concernant l'exécution, la compilation ou autres, veuillez ajouter une page à cet effet juste après la page titre. Pour la remise, veuillez respecter la procédure expliquée dans le document « Procédure de remise ».

Date limite de remise : **6 octobre 2020 à 23h59**
Équipe de **deux** ou **trois**

Bon travail !