1. Description (à lire après une lecture du support de cours)

L'objectif de ce travail est l'exploration de deux agents réflexe : avec et sans mémoire. Le problème est le suivant. Un robot aspirateur (*Vacuum Cleaner Robot*) est situé dans une pièce qui contient des obstacles et de la poussière sur le plancher. Le robot doit nettoyer la salle en aspirant toute la poussière qui se trouve sur le plancher, et ce en minimisant l'énergie dépensée.

Le monde est représenté de la manière suivante. La pièce est de forme carrée et toutes les positions possibles dans la pièce sont représentées par une grille de n x n. Les positions (0, 0) et (n, n) correspondent à la case supérieure gauche et la case inférieure droite, respectivement. Un obstacle occupe entièrement un certain nombre de cases. Le robot ne pourra jamais se déplacer à une case occupée par un obstacle et, évidemment, n'aura pas de poussière à aspirer à cet endroit. Il ne pourra pas non plus sortir de la grille (on suppose qu'un mur entoure toute la grille).

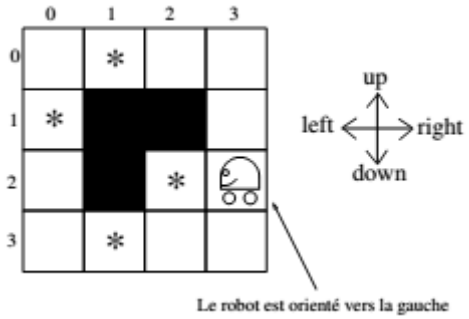

Figure 1 : Exemple de problème

2. Représentation du problème en JAVA

Une représentation générique en Java vous est fournie. La figure 2 présente le diagramme de classes.

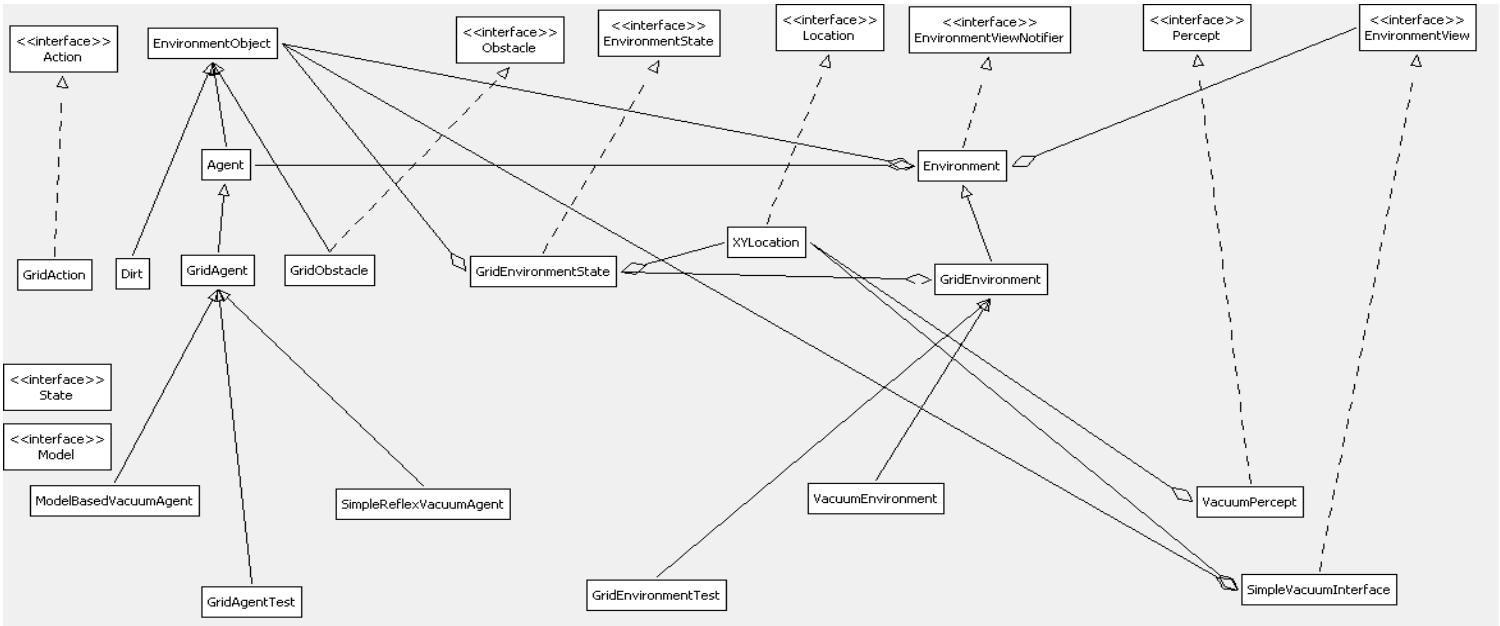

Figure 2. Diagramme de classe du projet Lab1

Les principales classes qui y sont définies sont les suivantes :

- **Environment** : Cette classe générique contient les attributs et méthodes auxquels on s'attend dans tout environnement. On peut associer à cet environnement un (ou plusieurs) objet observateur (de type **EnvironmentView**) qui sera en général une interface affichant l'état de notre environnement.

- **EnvironmentState** : Une variable de ce type sera utilisée pour indiquer un état possible d'un environnement.

- **EnvironmentObject**: Classe de base de tout objet de l'environnement (Agent, Obstacle, Dirt,..). Sert à fixer un identificateur unique.

- **Agent :** Représente l'agent. Il ne contient qu'un seul attribut indiquant s'il est vivant. Il s'agit d'une classe abstraite. Toute sous-classe concrète devra définir les méthodes *chooseAction*, qui détermine l'action choisie selon l'interprétation faite de l'environnement, qui lui est passée en paramètre. Un agent avec mémoire devra aussi redéfinir la méthode *updateModel*.

Notre robot évolue dans un environnement sous forme de grille (Grid environment). Les classes pour cet environnement sont définis dans le package grid :

- **GridEnvironment :** C'est cette classe qui représente notre grille. Dans cet environnement, ce sont les actions suivantes qui peuvent être exécutées :
    - ***Forward* :** L'agent essaie de se déplacer à la position suivante en face de lui.
    - ***Turn_90_Right :*** L'agent fait une rotation de 90 degrés vers la droite.
    - ***Turn_90_Left :*** L'agent fait une rotation de 90 degrés vers la gauche.
    - ***Stop :*** L'agent meurt.
    - ***NoOp* :** L'agent ne fait rien.
    - ***Error :*** L'agent à fait une action erronée.
- **GridAgent** : étendre **Agent** par l'ajout de deux attributs orientation et collision indiquant l'orientation de l'agent, ou s'il vient juste de foncer sur un obstacle ou un mur.
- **GridObstacle** : Représente un obstacle. Elle implante **Obstacle**. Aucun attribut et aucune méthode.
- **GridEnvironmentState** : l'état de cet environnement est entièrement définie par les objets et leurs position dans la grille.

Notre agent est un robot aspirateur (Vacuum Cleaner Robot). Les classe correspondant à notre robot sont définis dans le package vacuum :

- **Dirt** : Représente une poussière. Elle étende **EnvironmentObject**. Aucun attribut et aucune méthode.
- **VacuumPercept :** Repesent la perception de l'agent. Nous supposons ici que l'environnement est partiellement observable. L'agent ne pourra voir que sa localisation et son état (salle ou propre).
- La classe **VacuumEnvironment** possède les caractéristiques spécifiques suivantes :
- Une action supplémentaire, Suck, qui a pour effet de faire disparaître une poussière si elle se trouve au même endroit que l'agent. Une implémentation de la méthode getPerceptByAgent, qui retourne un percept (location, status, collision). Le premier item est la position où il se trouve dans la grille. La deuxième aura pour valeur Dirty ou Clean, selon qu'il y a de la poussière ou non à l'endroit où se trouve l'agent. Le troisième item a la valeur true si l'agent vient juste de foncer sur un obstacle ou un mur.
- Une implémentation de la méthode isDone, qui retourne true lorsque l'environnement ne contient aucune poussière.

- **SimpleVacuumInterface :** Une interface agit comme un observateur de l'environnement. Chaque fois que l'état de l'environnement change, il avise l'interface, par le biais des méthodes *thingAdded*, *thingDeleted* et *changedState*. L'interface mettra alors à jour l'affichage de la grille. Pour créer un objet de cette classe, il faut lui fournir la hauteur et la largeur de la grille.

Le package *vacuum.test* contient une classe de teste « **GridEnvironmentTest ».** Vous pouvez exécuter la fonction main de cette classe et tester les classes prédéfinies.

3. Travail à réaliser

Dans ce travail, on vous demande de compléter l'implantation deux classes d'agents aspirateurs qui évolueront dans notre environnement : **SimpleReflexVacuumAgent** et **ModelBasedVacuumAgent**. De plus, vous devrez compléter l'implantation de la classe **VacuumModel** qui implante l'interface Model.
La classe **ReflexVacuumAgent** est très simple. L'agent choisit une action au hasard. Dès qu'il perçoit de la poussière, il l'aspire.
La classe **ModelBasedVacuumAgent** est un peu plus complexe. L'agent aspire aussi la poussière dès qu'il en perçoit. Mais par contre il s'assure de ne pas foncer à nouveau sur un mur ou un obstacle lorsqu'il revient à une position qu'il a déjà visitée (ex : mémorisation de la topologie de l'environnement). Aussi, il essaie de ne pas revenir à une position déjà visitée. Plus particulièrement, s'il s'apprête à se déplacer à une position déjà visitée, il s'en empêchera si au moins une des trois autres positions adjacentes n'a pas été visitée mais accessible (ce n'est ni un mur, ni un obstacle).
Attention ! L'agent n'a aucune information sur les positions des objectifs (poussières). C'est un agent avec état interne et ce n'est pas un agent basé sur les buts (voir support de cours).

Le package *test1* contient une classe de teste « **TestVacuum ».** C'est le programme pour tester vos agents. On suppose que les mesures de performances d'un agent dépendent du résultat de l'action exécutée. Pour tourner gauche ou droite notre robot va consommer une unité d'énergie, deux unité pour avancer, trois unité s'il entre en collision avec un obstacle, et cinq unité pour aspirer :

```
TURN_90_RIGTH_MEASURE = -1;
TURN_90_LEFT_MEASURE  = -1;
 FORWARD_MEASURE      = -2;
COLLISION_MEASURE     = -3;
SUCK_MEASURE          = -5;
```