Sujet n°4 Conception et développement d'un simulateur de robot avec UML et Java3d

Environnement de travail

- Java SE 6.0 de Oracle
- Java 3D 1.5.1 disponible ici http ://www.oracle.com/technetwork/java/javamail/index-jsp-138252.html
- Java 3D 1.5.1 API disponible ici http ://download.java.net/media/java3d/javadoc/1.5.2/index.html
- Simbad disponible ici http ://simbad.sourceforge.net/
  - ant pour compiler
  - java -jar lib/simbad.jar pour démarrer l'application

1 Objet

Le contexte est l'automatisation de la logistique interne d'un environnement industriel simulé. Celui-ci est constitué d'un atelier et d'une zone de stockage attenante. Il est demandé d'automatiser les flux internes en utilisant un robot mobile de manutention. Le présent document décrit les besoins initiaux à satisfaire par le robot de manutention destiné à assurer le transport de produits dans l'atelier.

La mission nominale du robot consiste à transporter des produits dans un atelier, depuis deux files d'encours, en sortie respectivement de chaque centre d'usinage (CU1, CU2), pour les déposer dans une zone de stockage où les produits seront triés selon leur taille.

Initialement le robot est en position veille, sur sa zone de repos, à l'intérieur de la zone de stockage. Il n'y a pas d'orientation imposée au robot dans sa zone de repos. Il y a exactement 3 produits à transporter depuis chaque centre d'usinage. La composition des files d'encours est connue avant le début de la mission élémentaire. Chacune de ces files contient soit uniquement des petits produits, soit uniquement des grands produits. Pour ses déplacements dans l'atelier, le robot empruntera des allées de circulation délimitées, prévues à cet effet et matérialisées en leur centre par une bande de marquage au sol. Les zones machines (CU1, CU2) sont interdites à toute circulation. Pour ses déplacements à l'intérieur de la zone de stockage, la circulation du robot est totalement libre.

Le robot est mis en marche par un opérateur en indiquant où sont les petits produits (CU1 ou 2). En moins de 4 minutes, le robot doit sans intervention manuelle :

- aller chercher les produits dans l'atelier ;
- les apporter dans le stock correspondant à leur taille ;
- revenir en position de veille et s'arrêter à l'intérieur de la zone de repos.

Le robot porte sans contact avec le sol de 1 à 6 petits ou grands produits suivant le déroulement de la mission.

Dans le cadre de sa mission, le robot se déplace en toute autonomie (sans téléguidage), néanmoins si durant ses déplacements dans l'atelier, le robot rencontre un obstacle le gênant dans son parcours, il devra alors s'arrêter et attendre que l'obstacle soit évacué avant la poursuite de la mission.

Les produits à transporter sont composés d'un ensemble solidaire de pièces et ont les caractéristiques suivantes :

- Dimension du grand produit : 48mmL * 55mmH * 15mmE d'encombrement
- Dimension du petit produit : 31mmL * 38mmH * 15mmE d'encombrement

Les produits sont posés verticalement dans les zones prévues à cet effet, en appui comme indiqué sur la figure 1.

Le robot est équipé d'un capteur ultrasonique, d'un capteur de couleur et de trois servomoteurs interactifs pour les roues et le préhenseur. La vitesse de déplacement du robot est de 2000 cm/min à vide. La vitesse de déplacement du préhenseur est de 500cm/min. Ces vitesses sont ralenties de 5 pour cent par pièce portée.

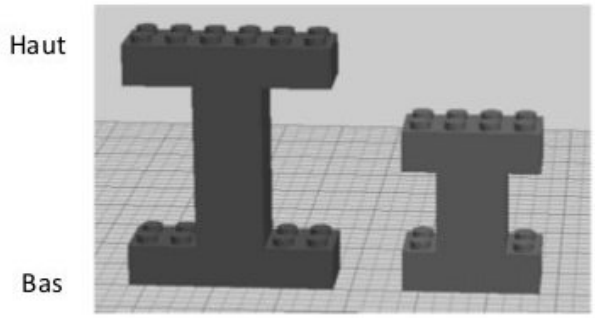


FIGURE 1 – Grandes et petites pieces

2 Travail à faire

2.1 Modélisation de l'atelier

Dans un premier temps vous modéliserez l'atelier dans lequel le robot évolue. Le diagramme de classe de la figure 2 montre l'architecture de simbad relative à la scène et aux objets qui la constituent. Les dimensions et couleurs de l'atelier et de la zone de stockage sont spécifiées sur les schémas de l'annexe. Pour cela vous devez implémenter une classe dans le paquetage *demo* qui hérite de la classe *EnvironmentDescription*. Ajouter cette classe dans le tableau *classNames* de la classe *DemoManager* pour qu'elle apparaisse dans le menu de l'application.

La classe *EnvironmentDescription* contient une méthode *add* qui permet d'ajouter des composants à la scène. Vous pourrez rapidement créer des murs et des boites non franchissables à partir des classes *Wall* et *Box* respectivement. Ces classes utilisent le parallélépipède (classe *Box*) de la bibliothèque Java 3D *com.sun.j3d.utils.geometry.Primitive*.

Ensuite il faut développer les pastilles, les zones de stockage et la bande de marquage au sol. La bibliothèque Java 3D ne propose pas de classe prédéfinies pour faire des cercles pleins, des arcs de cercles et des carrés. Il vous faudra réaliser vous même ses objets géométriques à partir de la classe *GeometryArray*. La classe Pastille qui hérite de la classe *SimpleAgent* vous est donnée à titre d'exemple, vous pourrez vous en servir pour instancier les pastilles jaunes et vertes. Cette classe utilise la classe *TriangleFanArray* pour construire un cercle plein. La classe *SimpleAgent* est la classe de base pour tout les éléments colorés qui permettent de localiser le robot et les pièces à transporter. Pour ajouter un objet géométrique à la scène il faut créer une instance de la classe *Shape3D* qui sera passée en paramètre de la méthode *addChild* de la classe *BaseObject*.

2.2 Modélisation du robot

Il n'y a pas de forme imposée pour le robot, vous pouvez utiliser celle proposée par simbad. Dans la classe de base qui hérite de *EnvironmentDescription* que vous avez créé précédemment vous construirez une classe interne qui hérite de la classe Agent. Le diagramme de classe de la figure 3 montre l'architecture de simbad relative aux agents robots.

Vous devez redéfinir les méthodes *initBehavior* et *performBehavior* pour définir le comportement du robot. *performBehavior* est appelée en boucle par le simulateur.Pour commander le robot la classe *Agent* propose les méthodes *setTranslationalVelocity* et *setRotationalVelocity*.

Tester votre robot en boucle ouverte.

2.3 Modélisation du capteur de couleurs et suivi de trajectoire

La classe *Eye* vous permettra de développer votre capteur de couleur. Simbad propose un capteur de luminosité et une camera qui peuvent vous servir d'exemple. La classe *Eye* utilise la classe *BufferedImage* de la bibliothèque *java.awt.Image*. Le diagramme de classe de la figure 4 montre l'architecture de simbad relative aux capteurs.

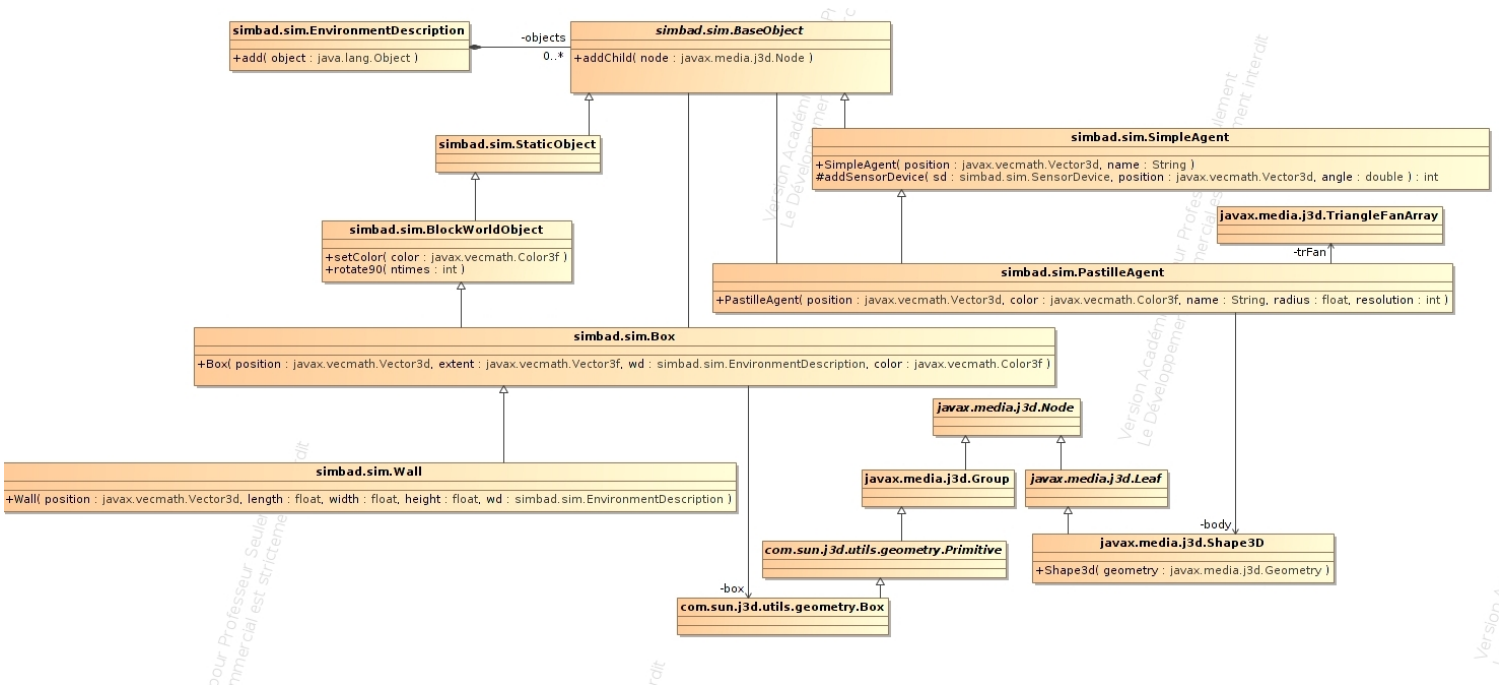


FIGURE 2 – Diagramme de classe relatif à la scène et aux objets qui la constituent

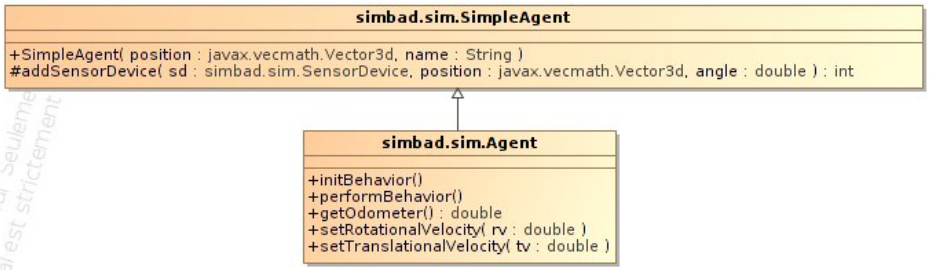


FIGURE 3 – Diagramme de classe relatif aux agents robots

Pour ajouter le capteur à votre robot il vous suffit de créer une méthode static dans la classe *RobotFactory* qui créera une instance de votre capteur, l'associera à l'agent passé en paramètre grâce à la méthode *addSensorDevice* et retournera cette instance. Attention à l'orientation du capteur, il s'agira peut être de l'orienter vers le sol. Puis vous n'avez plus qu'à appeler la méthode static à partir du constructeur de votre classe robot pour pouvoir manipuler l'instance du capteur dans la définition du comportement de votre robot.

Vous êtes libre d'implémenter l'algorithme de déplacement que vous voulez. Bien évidemment certains seront plus performants que d'autres.

2.4 Prise et dépose des pièces et réalisation de la mission

Il faut créer les petites pièces et les grandes pièces et les ajouter à la scène. Elles seront déposées dans les zones d'encours contre le mur.

Puis il faut mettre en place les fonctionnalités de prise et de dépose des pièces. Vous êtes libre de mettre en place la stratégie que vous voulez pour réaliser la mission (un aller-retour par produit, un aller-retour par centre d'usinage (même type de produit), un aller-retour pour les 2 centres d'usinage en faisant le tour de

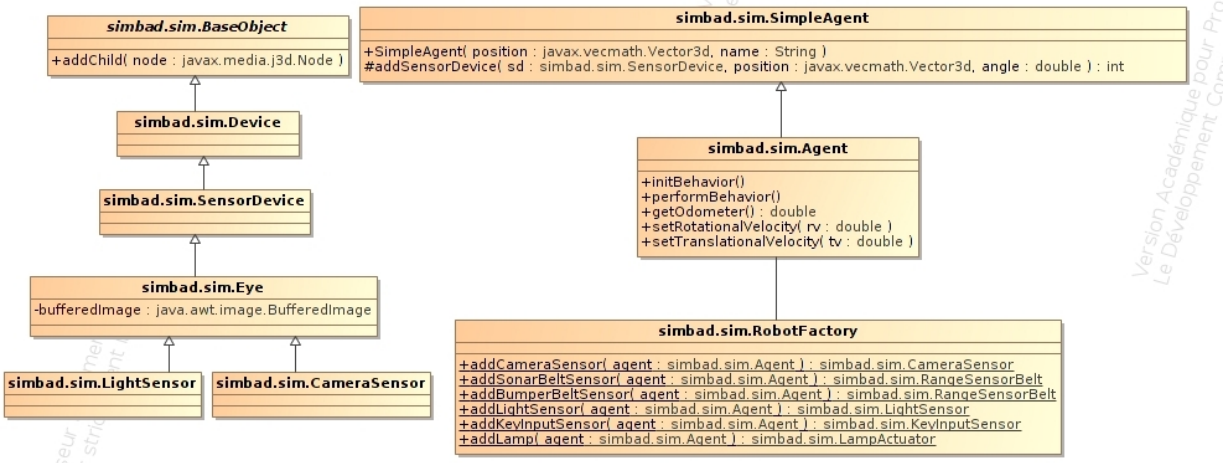


FIGURE 4 – Diagramme de classe relatif aux capteurs

l'atelier, un aller-retour pour les 2 centres d'usinage en repassant devant l'entrée du stock). Si vous décidez de dimensionner le préhenseur pour pouvoir prendre 6 pièces à la fois, alors le robot en prendra de 4 à 6 de manière aléatoire. Si vous décidez de le dimensionner pour 3 pièces alors il en prendra 2 ou 3 de manière aléatoire uniquement pour les grandes pièces.

Une prise ne pourra se faire que lorsque le robot se trouvera à une distance d'un cm maximum. Vous pouvez utiliser le capteur ultrason *RangeSensorBelt* fourni par Simbad pour localiser vos pièces.

2.5 Prise en compte des obstacles

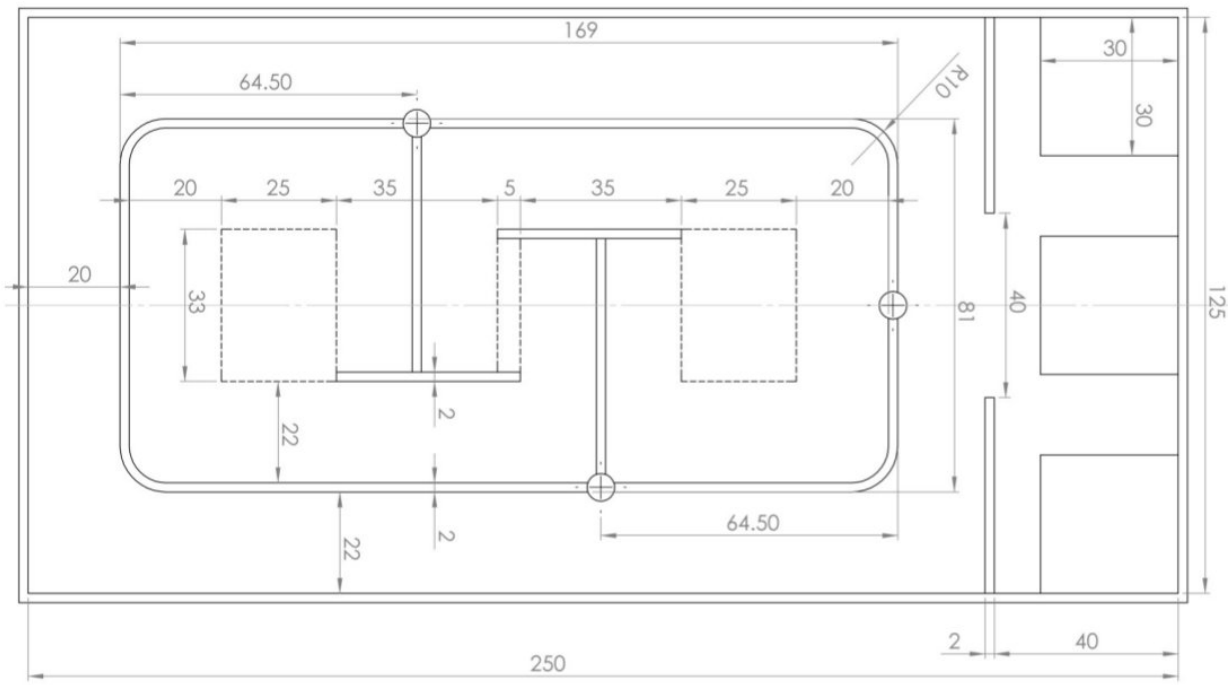
La simulation doit permettre de disposer par un clic souris un obstacle de type *Wall* à l'emplacement du clic souris. L'obstacle sera supprimé par un autre clic souris n'importe où sur la scène.

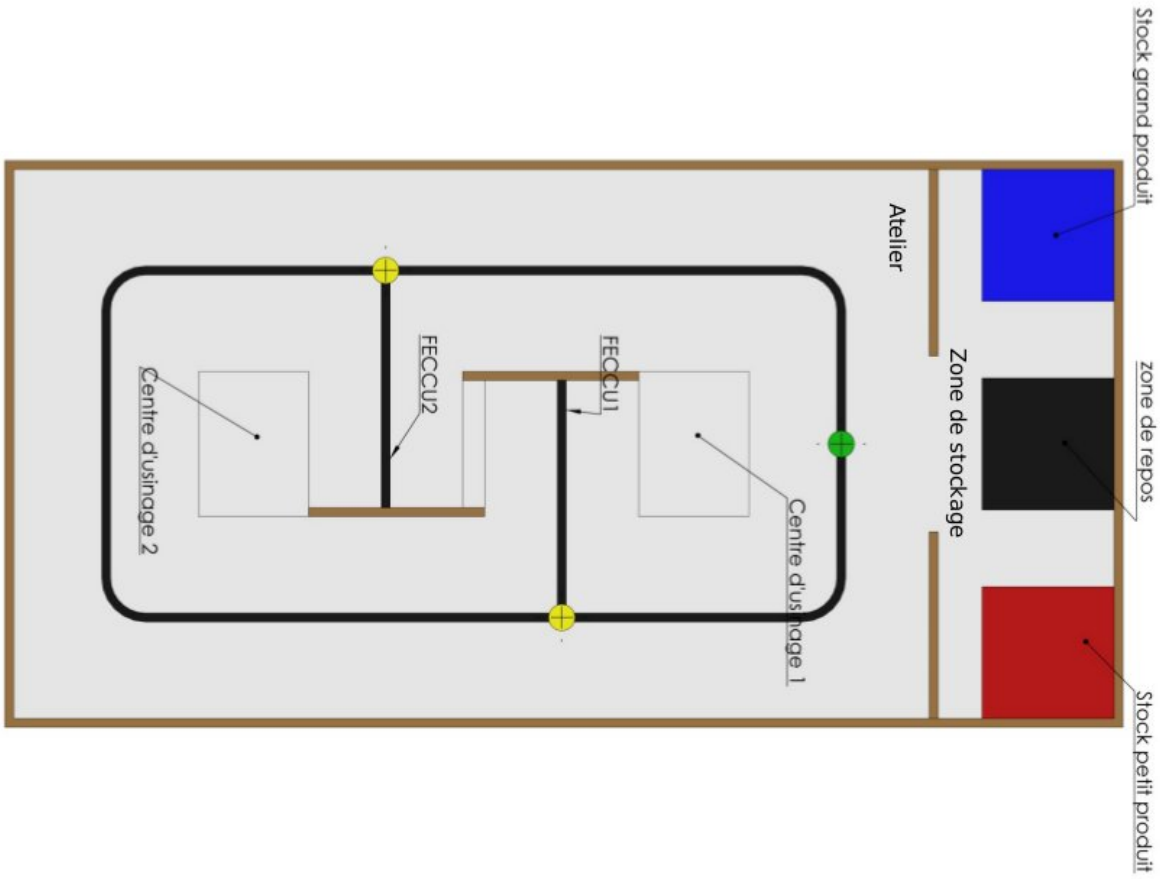
3 Documents à rendre

La livraison doit comprendre l'ensemble de votre programme ainsi qu'un document qui contiendra :
- Les diagrammes de classe qui modéliseront rigoureusement l'architecture de votre implémentation.
- Un diagramme d'objet qui modélisera rigoureusement la hiérarchie des objets de la scène.
- Les diagrammes de séquence correspondants aux différents scénarios d'appel de messages entre vos classes lors de la création de la scène et lors du déroulement de la mission.
- L'analyse détaillée et commentée de vos algorithmes de suivi de trajectoire.
- L'explication de la mise en oeuvre du capteurs de couleurs, des objets géométriques (bande de marquage), du préhenseur.
- La justification de vos choix stratégiques pour réaliser la mission.

4 Annexe 1

Unité cm


169
64.50
R10
30
30
20
25
35
5
35
25
20
20
33
81
40
125
22
2
64.50
22
2
2
40
250


Stock grand produit
zone de repos
Stock petit produit
Atelier
Zone de stockage
FECCU2
FECCU1
Centre d'usinage 2
Centre d'usinage 1