“Localisation d’un robot dans un environnement discret”

Sujet n°18

Tuteur : M.Thomas Vincent

IUT Nancy Charlemagne - 2018

Table des matières

Description du projet

Il s'agit de réaliser une application en Java, qui aura pour but d'initier des étudiants aux problématiques de localisation et de cheminement d'un robot dans un environnement discret.

L'environnement dans lequel évoluera le robot sera, dans le cadre de ce projet, un labyrinthe et l'application sera utilisée pour des démonstrations dans des événements du type foire de la science.

Il est important de préciser que le robot ne sera pas un robot physique et que celui-ci ne sera géré que logiquement dans l'application, puisqu'il s'agit d'introduire simplement des notions clés du système de fonctionnement d'un robot.

L'application n'est pas non plus un jeu comme nous le pensions au départ, mais un réel outil pédagogique permettant de montrer à des étudiants comment fonctionnent les systèmes de localisation et de déplacement d'un robot (avec néanmoins des interactions entre les étudiants et le logiciel).

L'application ne fonctionne que si un enseignant est présent avec l'étudiant pour pouvoir l'amener à se poser les bonnes questions et comprendre parfaitement les différents algorithmes.

Vision du projet

Les personnes qui vont être intéressées par le produit sont des enseignants ou des personnes faisant des salons scientifiques ou informatiques.

La cible de notre produit est les étudiants qui ont pour but d'apprendre ou de comprendre différents types d'algorithmes comme la localisation et peut être la recherche de chemins.

Le produit a pour but de permettre d'initier des étudiants aux problématiques de localisation et de recherche de chemins (optionnel) d'un robot dans un environnement discret.

Les fonctionnalités critiques par rapport au produit sont :

- Le module "Création de labyrinthes" doit permettre de créer un labyrinthe de taille donnée et positionner différents éléments sur celui-ci (Exemples : murs, entrée, sortie...)
- Le module "Localisation" affiche une vue globale du labyrinthe sans afficher le robot, et une vue locale qui correspond à ce que voit le robot (Vue 3D, vue cases adjacentes...). L'utilisateur de l'application pourra déplacer le robot et celle-ci devra mettre à jour les emplacements possibles du robot en fonction des déplacements et orientations passées.
- Le réajustement de la fenêtre est obligatoire pour notre tuteur pour qu'il puisse changer la taille de sa fenêtre pendant sa présentation.

Le projet n'a pas vraiment de but commercial, on connaît cependant un vrai robot (blue-bot) ayant un but éducatif assez similaire et un jeu mobile dont le but est de déplacer un personnage grâce à des algorithmes (lightbot).

Liste des fonctionnalités

Les différentes fonctionnalités clés de la première itération sont: la création de labyrinthe, les outils de création, affichage adaptable, l'interpréteur de fichiers (fichiers avec l'extension .maze représentant un labyrinthe), la mise en place de l'architecture en MVC (Modèle, Vue, Contrôleur), le déplacement du robot et l'algorithme de localisation.

Voici la répartition des tâches :

	Valentin	Lucas	Johan	Matthieu
Mise en place MVC				
Création du labyrinthe				
Outils de génération (panneau Création)				
Outils de déplacement (panneau Création)				
Affichage « création » adaptable à la fenêtre				
Localisation dans le labyrinthe				
Affichage « localisation» adaptable à la fenêtre				
Interpréteur de fichiers (Sauvegarder/Ouvir) en .maze				
Ouvrir un labyrinthe en .png				
Déplacement du robot				
Algorithme de localisation				
Tests unitaires				
Javadoc				

La sous-tâche du « MVC » est :

- Mise en place du pattern

Les différentes sous-tâches de « la création du labyrinthe » sont :

- Modifier les dimensions du labyrinthe de taille AxB
- Créer une case « vide »
- Créer une case « mur »
- Créer une case « entrée »
- Créer une case « sortie »
- Créer un nouveau labyrinthe en conservant l'existant

Les différentes sous-tâches de « outils de génération » sont :

- Faire un ilot
- Vider le labyrinthe
- Remplir de murs le labyrinthe
- Générer un labyrinthe parfait / imparfait

La sous-tâche de « outils de déplacement » est :

- Déplacement l'ensemble du labyrinthe vers le haut/bas/droite/gauche

Les différentes sous-tâches de « localisation du robot dans le labyrinthe » sont :

- Basculer entre l'affichage des positions possibles et la position du robot
- Afficher la vue des cases adjacentes
- Repositionner le robot sur l'entrée / aléatoirement

Les différentes sous-tâches de « l'interpréteur de fichiers » sont :

- Sauvegarder un labyrinthe
- Charger un labyrinthe

Les différentes sous-tâches du « déplacement du robot » sont :

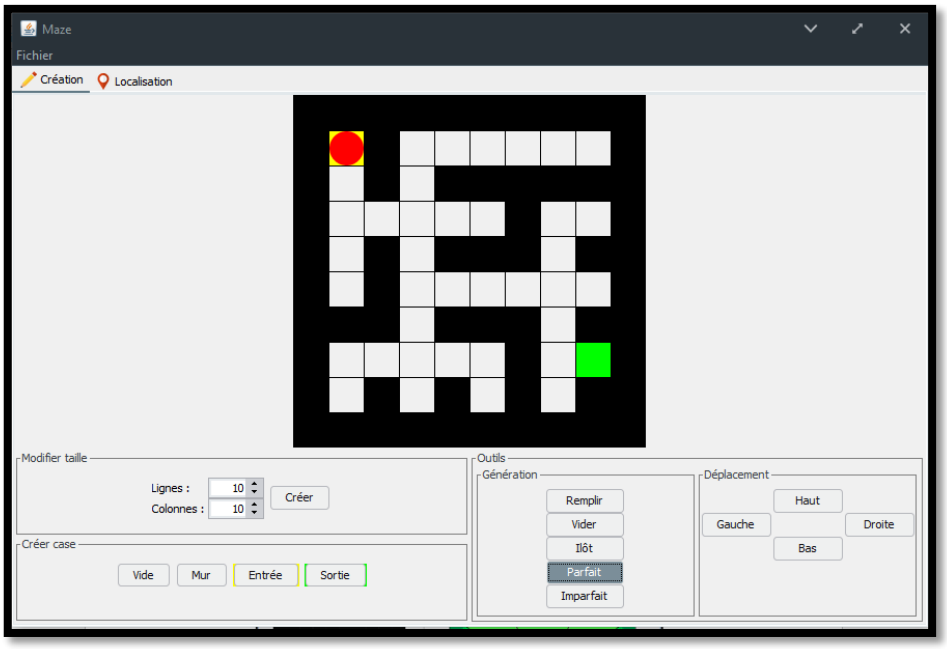
- Déplacer le robot vers l'avant
- Rotation du robot de 90° vers la gauche
- Rotation du robot de 90° vers la droite

Les différentes sous-tâches de « l'algorithme de localisation » sont :

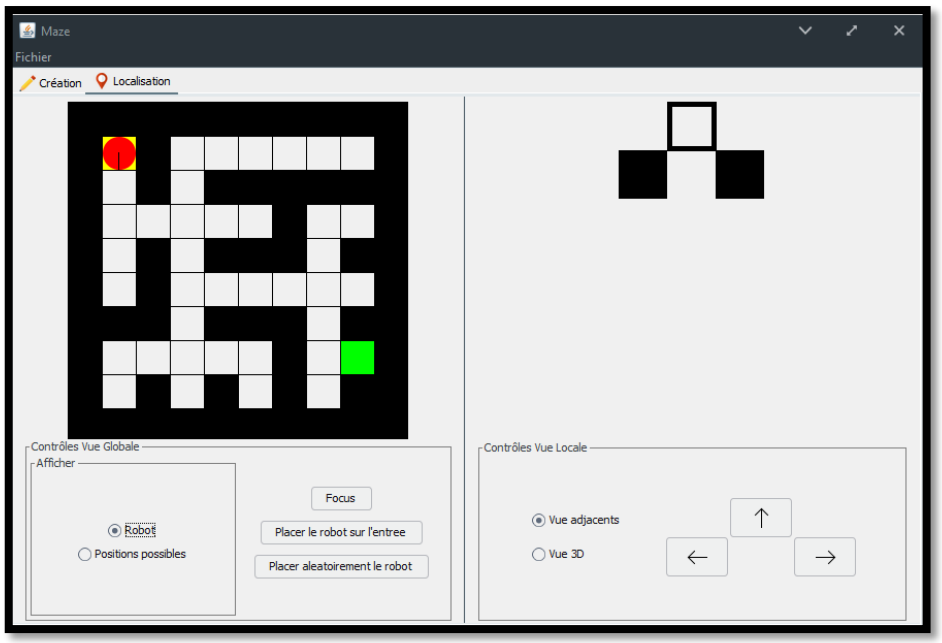
- Proposer les positions possibles du robot en fonction de sa vision

Maquettes

Panneau « Création » :



Panneau « Localisation » :



Cas d'utilisations, scénarios et conditions de validation

Les différents cas d'utilisation que nous allons développer sont : la création de labyrinthe avec des outils et sans, afficher et masquer la position du robot et déplacer le robot

- ***Création de labyrinthe à la main:***
  - ***Descriptif textuelle*** : Création du labyrinthe en utilisant une interface graphique grâce à une grille représentant ce labyrinthe
  - ***Critère de validation :***
    - Dans l'onglet Création
    - Création d'un labyrinthe fermé
    - Affichage d'une grille de 10x10 initialement
    - Possibilité de modifier la taille de la grille
    - Boutons en dessous de l'affichage du labyrinthe pour créer un mur, une case vide, une entrée et une sortie.
  - ***Scénario :*** L'enseignant lance le programme et le panneau de création de labyrinthe s'affiche. Par défaut, un labyrinthe vide de 10 par 10 est créé avec une entrée et une sortie positionnées automatiquement. Il clique sur le bouton « Mur » puis, l'aide du clic droit, crée des murs aux endroits où il le souhaite sur le labyrinthe (Tant que l'enseignant ne change pas le type de case à créer, « Mur » reste sélectionné). Il clique ensuite sur le bouton « Entrée » pour repositionner la case d'entrée du robot en cliquant sur la case souhaitée du labyrinthe. Enfin, il décide de supprimer certains murs qu'il vient de créer. Pour cela, il clique sur le bouton « Vide » et clique sur les murs qu'il souhaite supprimer.

- ***Création de labyrinthe avec des outils de génération :***
  - ***Descriptif textuelle :*** Création de l'objet Labyrinthe grâce à des outils de génération (Vider, Remplir, Ilot, Parfait, Imparfait) avec un bouton sur l'interface graphique
  - ***Critère de validation :***
    - Dans l'onglet Création
    - Affichage d'une grille de 10x10 initialement
    - Possibilité de modifier la taille de la grille (Adaptation de l'affichage)
    - Création d'un labyrinthe fermé
    - L'outil **vider** : Toutes les cases du labyrinthe deviennent des cases vides (sauf le contour qui reste des cases murs)

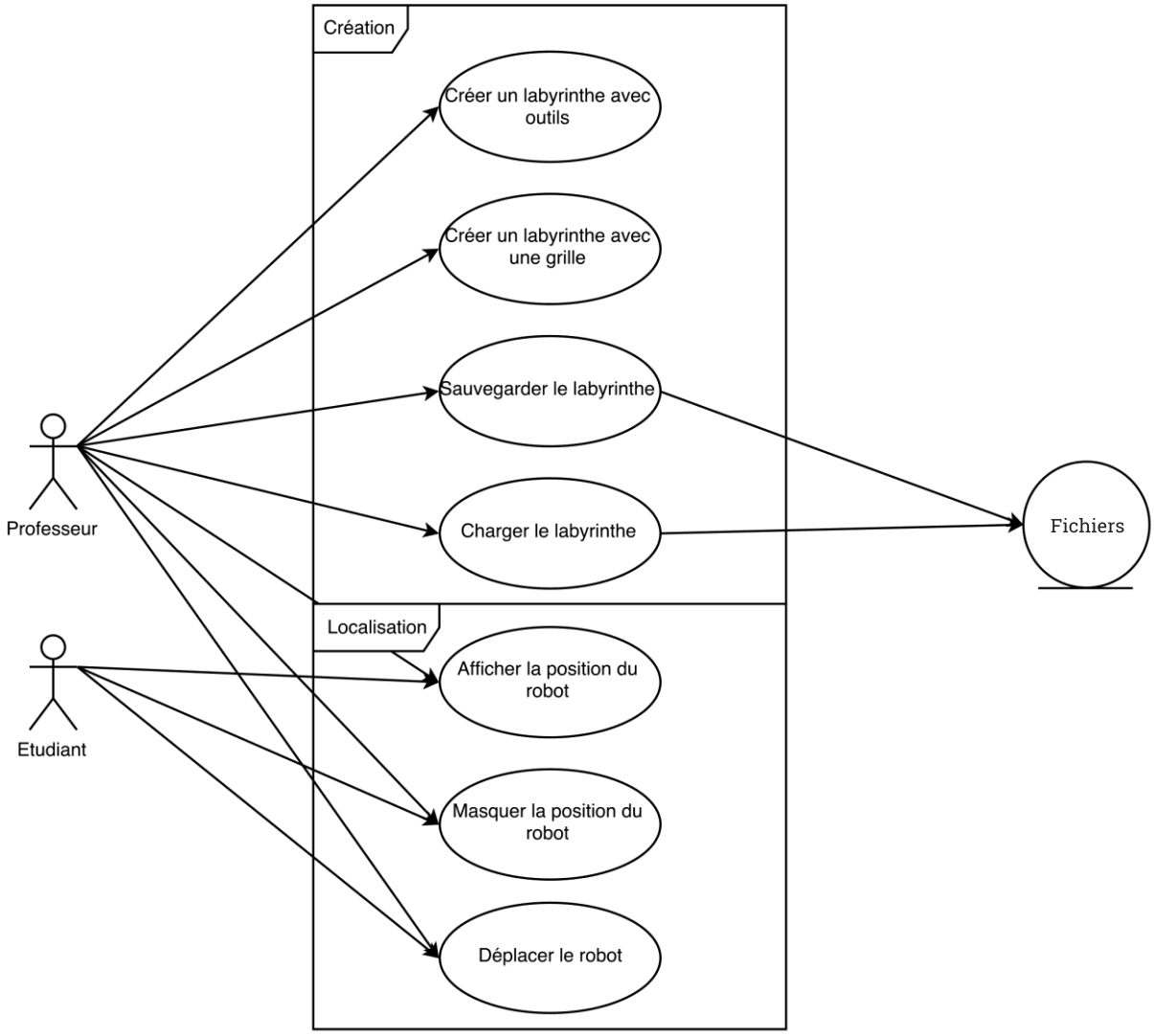
  - L'outil **remplir** : Toutes les cases du labyrinthe deviennent des cases murs (sauf les cases correspondant à l'entrée et la sortie qui reste des cases vides)
  - L'outil **ilot** : Construit un couloir de 2 cases vides
  - L'outil **Parfait** : Génère un labyrinthe parfait
  - L'outil **Imparfait** : Génère un labyrinthe imparfait

- ***Scénario :*** L'enseignant lance le programme et le panneau de création de labyrinthe s'affiche. Par défaut, un labyrinthe vide de 10 par 10 est créé avec une entrée et une sortie positionnées automatiquement. L'enseignant modifie la taille du labyrinthe en 20x20 et clique sur le bouton « Créer » : Le labyrinthe s'actualise avec sa nouvelle taille. Il décide de générer un labyrinthe parfait, il clique alors sur le bouton « Parfait ». Un labyrinthe parfait de taille 20x20 est alors généré et fonctionnel.

- ***Afficher et masquer la position du robot :***
  - ***Descriptif textuelle*** : Affichage et masquage de la position du robot grâce à un bouton
  - ***Critère de validation :***
    - Dans l'onglet localisation
    - Utilisation d'une checkbox pour afficher et masquer la position
  - ***Scénario :*** L'enseignant lance le programme et le panneau de création de labyrinthe s'affiche. Il charge un labyrinthe qu'il avait créé auparavant. Il va sur le panneau « Localisation ». Le robot est masqué par défaut. L'enseignant déplace le robot puis il décide de l'afficher pour révéler sa véritable position. Pour cela, il coche la case « Afficher robot » et le robot apparaît dans le labyrinthe. Il décoche ensuite cette case pour cacher le robot.

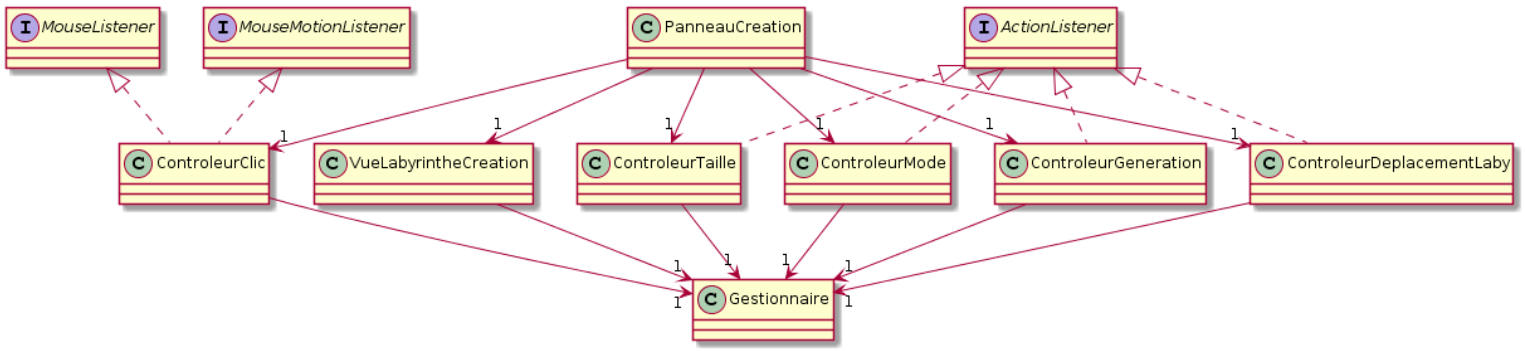
- ***Déplacer le robot :***
  - ***Descriptif textuelle :*** Déplacer le robot dans le labyrinthe grâce à plusieurs outils
  - ***Critère de validation :***
    - Dans l'onglet localisation
    - Déplacement grâce à des boutons sur l'interface, au clavier (exemple : flèches directionnelles ou ZQSD/WASD), manettes, …
    - Déplacement case par case du robot à chaque clic si c'est possible (pas de mur)
    - Met à jour la vue utilisée

Le diagramme de cas d'utilisation :



Différents diagrammes

Le diagramme de classe de la création :



Le panneau création est le panneau sur lequel il est possible de modifier le labyrinthe ainsi que d'en créer un nouveau.

C'est dans la classe **PanneauCreation** que l'on va créer l'interface graphique.

On a tout d'abord la classe **ControleurMode** qui change le mode de création en cours : elle permet de dire au modèle que l'on est en train de créer une case vide, pleine, entrée ou sortie.

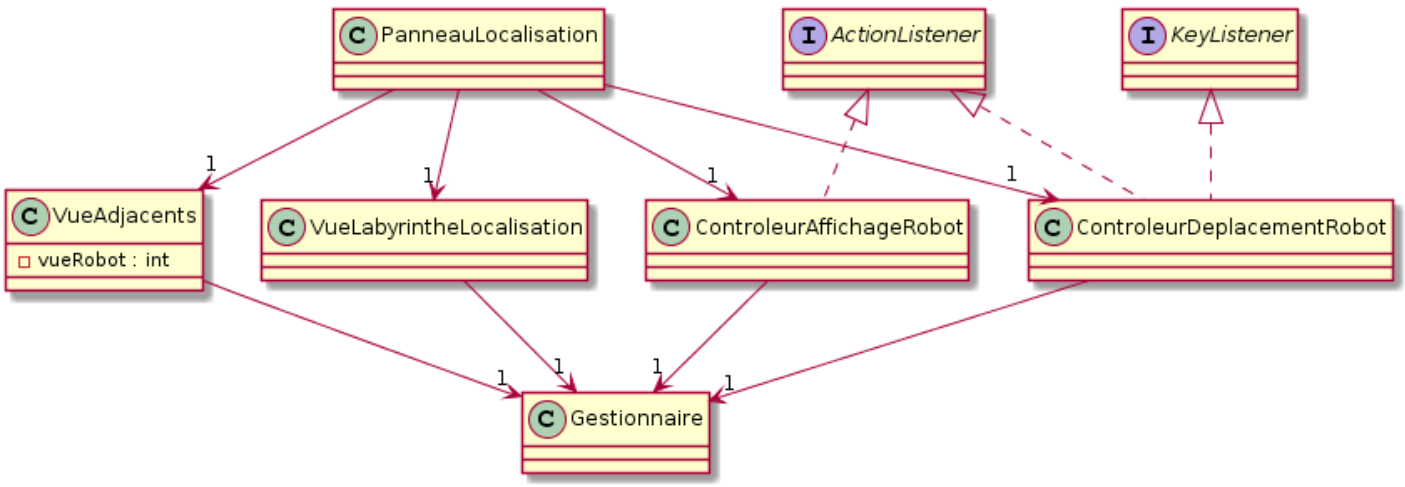
Parallèlement à celle-ci, la classe **ControleurClic** se charge de créer, lors d'un clic sur le labyrinthe, la case correspondante au mode sélectionné.

Si l'on souhaite ensuite changer le nombre de lignes ou de colonnes du labyrinthe, le **ControleurTaille** aura pour mission d'agrandir ou de rétrécir celui-ci tout en en conservant le labyrinthe actuellement en construction.

Il est également possible de déplacer le labyrinthe, par exemple de remonter tout le labyrinthe d'une case: c'est le travail de la classe **ControleurDeplacementLaby**.

Enfin, le **ControleurGeneration** donne la possibilité de créer un nouveau labyrinthe en utilisant un algorithme de génération (vide, plein, ilôt, parfait, imparfait).

Le diagramme de classe de la localisation :



Le panneau localisation est le panneau dans lequel l'algorithme de localisation du robot s'exécute.

Ainsi, dès que l'on déplace le robot, les positions sur lesquelles il peut se trouver seront actualisées en fonction des positions possibles avant déplacement et de ce que le robot voit sur les cases adjacentes (classe **VueAdjacents**).

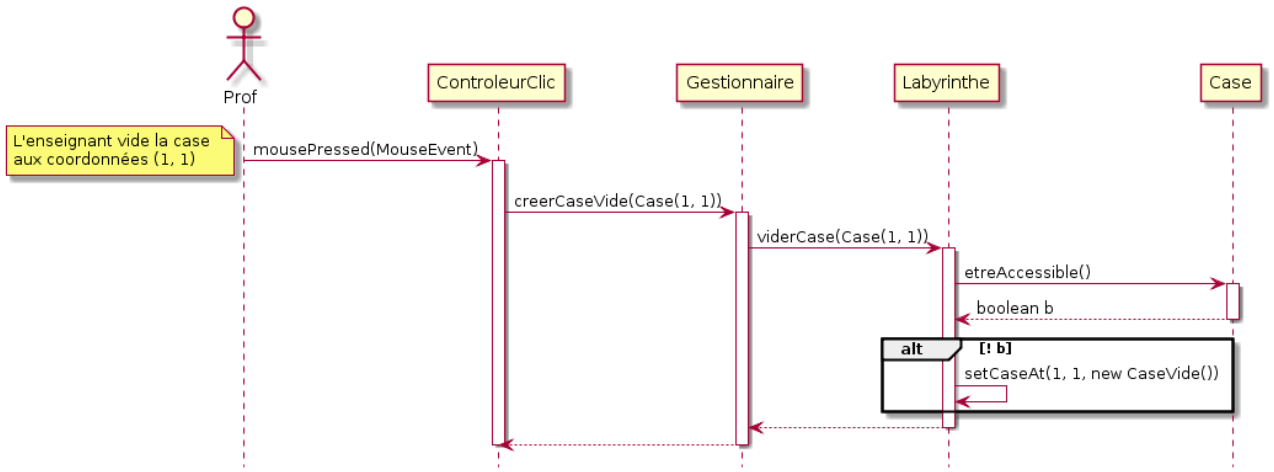
C'est dans la classe **PanneauLocalisation** que l'on va créer l'interface graphique et les deux vues(**VueAdjacents** et **VueLabyrintheLocalisation**).

On a tout d'abord la classe **ControleurAffichageRobot** qui fait la bascule entre l'affichage des positions possibles du robot et l'affichage de sa position effective.

Lorsqu'on déplace le robot, c'est le **ControleurDeplacementRobot** qui se chargera de le faire tourner à gauche, à droite ou encore de le faire avancer.

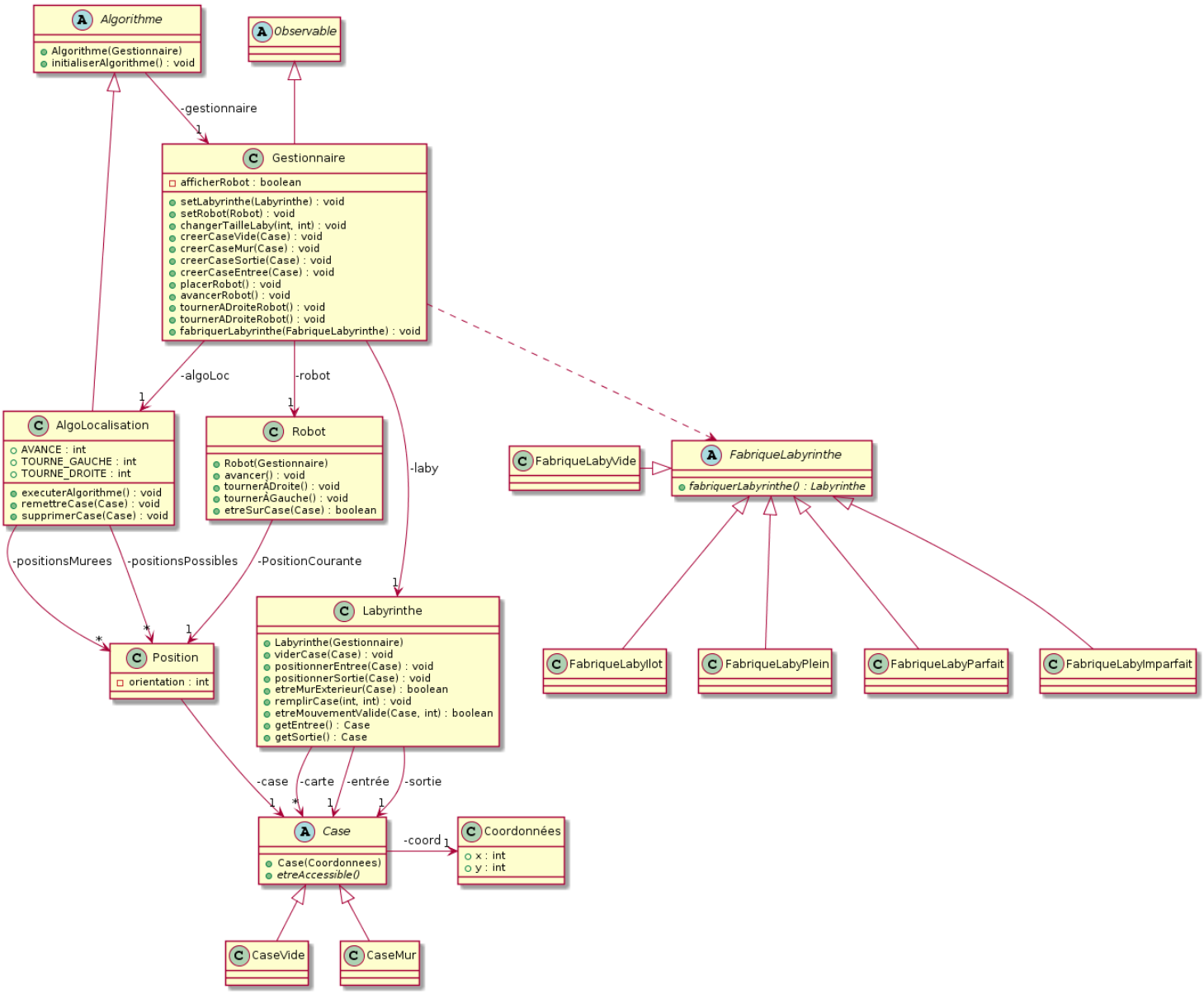
Le diagramme de classe du Modèle :

L'élément central du modèle est le **Gestionnaire**. Inspiré du pattern Médiateur (https://en.wikipedia.org/wiki/Mediator_pattern), il permet de gérer toutes les interactions entre le robot et le labyrinthe. Par exemple, lorsque l'on souhaite créer une case vide à un endroit du labyrinthe, on va appeler la méthode **creerCaseVide** de Gestionnaire qui va se charger de demander au labyrinthe de créer une case vide à cet emplacement. (cf. diagramme de séquence ci-dessous).

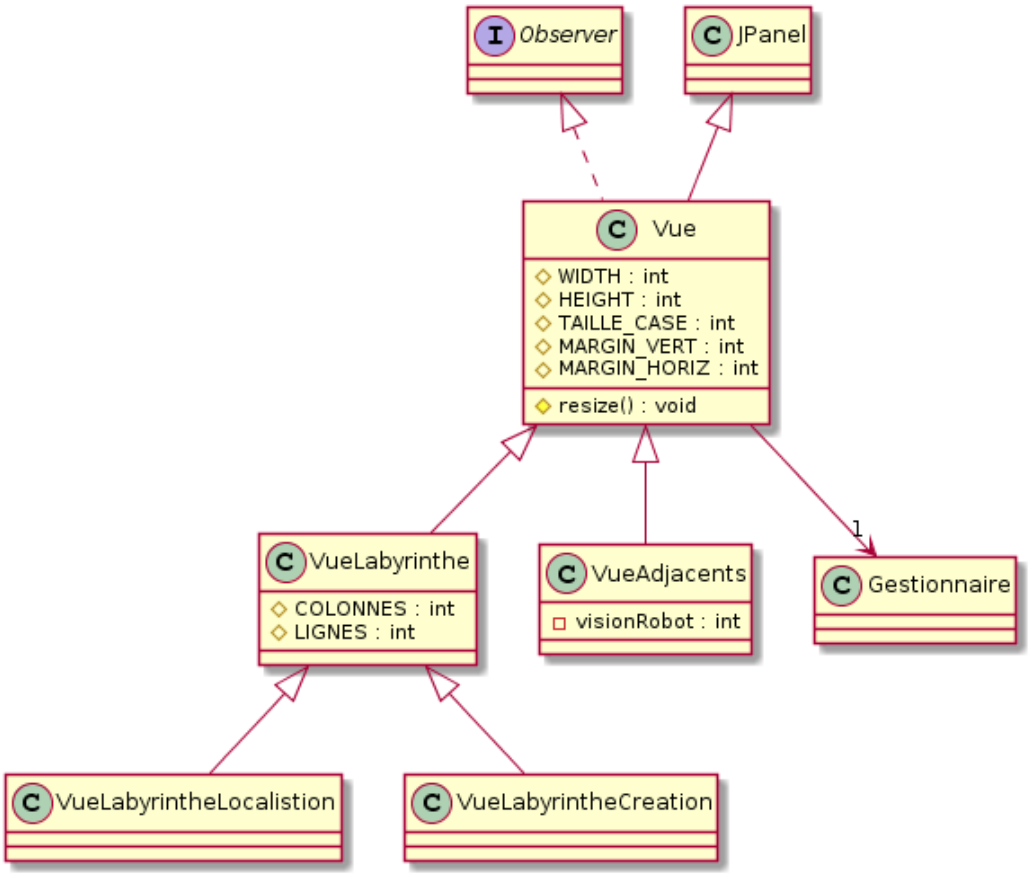


Dès que l'on va modifier le modèle, on va passer par la classe Gestionnaire, c'est pourquoi c'est celle-ci qui est **Observable**. C'est la porte d'entrée sur les données de l'application. C'est également lui qui entre en jeu lorsqu'on souhaite générer un labyrinthe (vide, plein, îlot, parfait, imparfait) : on appelle la méthode **fabriquerLabyrinthe** avec en paramètre la fabrique adéquate qui va fabriquer le labyrinthe souhaité et le Gestionnaire placera le robot sur ce nouveau labyrinthe.

Enfin, la classe Gestionnaire est celle qui se charge de réinitialiser les données de l'algorithme de localisation (positions possibles du robot) lorsqu'on change de labyrinthe.


Algorithme
Algorithme(Gestionnaire)
initialiserAlgorithme() : void
Observable
-gestionnaire
1
Gestionnaire
afficherRobot : boolean
setLabyrinthe(Labyrinthe) : void
setRobot(Robot) : void
changerTailleLaby(int, int) : void
creerCaseVide(Case) : void
creerCaseMur(Case) : void
creerCaseSortie(Case) : void
creerCaseEntree(Case) : void
placerRobot() : void
avancerRobot() : void
tournerADroiteRobot() : void
tournerADroiteRobot() : void
fabriquerLabyrinthe(FabriqueLabyrinthe) : void
-algoLoc
-robot
-laby
AlgoLocalisation
AVANCE : int
TOURNE_GAUCHE : int
TOURNE_DROITE : int
executerAlgorithme() : void
remettreCase(Case) : void
supprimerCase(Case) : void
Robot
Robot(Gestionnaire)
avancer() : void
tournerÀDroite() : void
tournerÀGauche() : void
etreSurCase(Case) : boolean
FabriqueLabyVide
FabriqueLabyrinthe
fabriquerLabyrinthe() : Labyrinthe
-positionsMurees
-positionsPossibles
-PositionCourante
Labyrinthe
Labyrinthe(Gestionnaire)
viderCase(Case) : void
positionnerEntree(Case) : void
positionnerSortie(Case) : void
etreMurExterieur(Case) : boolean
remplirCase(int, int) : void
etreMouvementValide(Case, int) : boolean
getEntree() : Case
getSortie() : Case
Position
orientation : int
FabriqueLabyIlot
FabriqueLabyPlein
FabriqueLabyParfait
FabriqueLabyImparfait
-case
-carte
-entrée
-sortie
Case
Case(Coordonnees)
etreAccessible()
-coord
Coordonnées
x : int
y : int
CaseVide
CaseMur

Le diagramme de classe de la vue:



Il y a, pour l'instant, deux types de vues : la vue des cases adjacentes au robot (**VueAdjacents**) et la vue du labyrinthe vu de dessus (**VueLabyrinthe**), qui est redécoupée selon les besoins d'affichage des panneaux localisation et création.

Toute vue a une taille de case qui lui est propre et qui s'adapte automatiquement en fonction de la taille de son composant parent.(à chaque repaint, on appelle resize() pour adapter l'affichage).

C'est ainsi que le redimensionnement automatique s'effectue.

Le diagramme de séquence du déplacement de l'algorithme de Localisation :

Voici une description détaillée du fonctionnement de l'algorithme de localisation :

Tout d'abord, supposons que l'enseignant fait avancer le robot en appuyant sur la touche directionnelle haut. Cela va déclencher l'appel de la méthode **avancerRobot** du Gestionnaire qui va faire avancer le robot s'il n'a pas de mur devant lui.

Après cela, on va exécuter l'algorithme de localisation en précisant le déplacement que l'on vient d'effectuer (ici AVANCE, mais il y a aussi TOURNE_GAUCHE et TOURNE_DROITE).

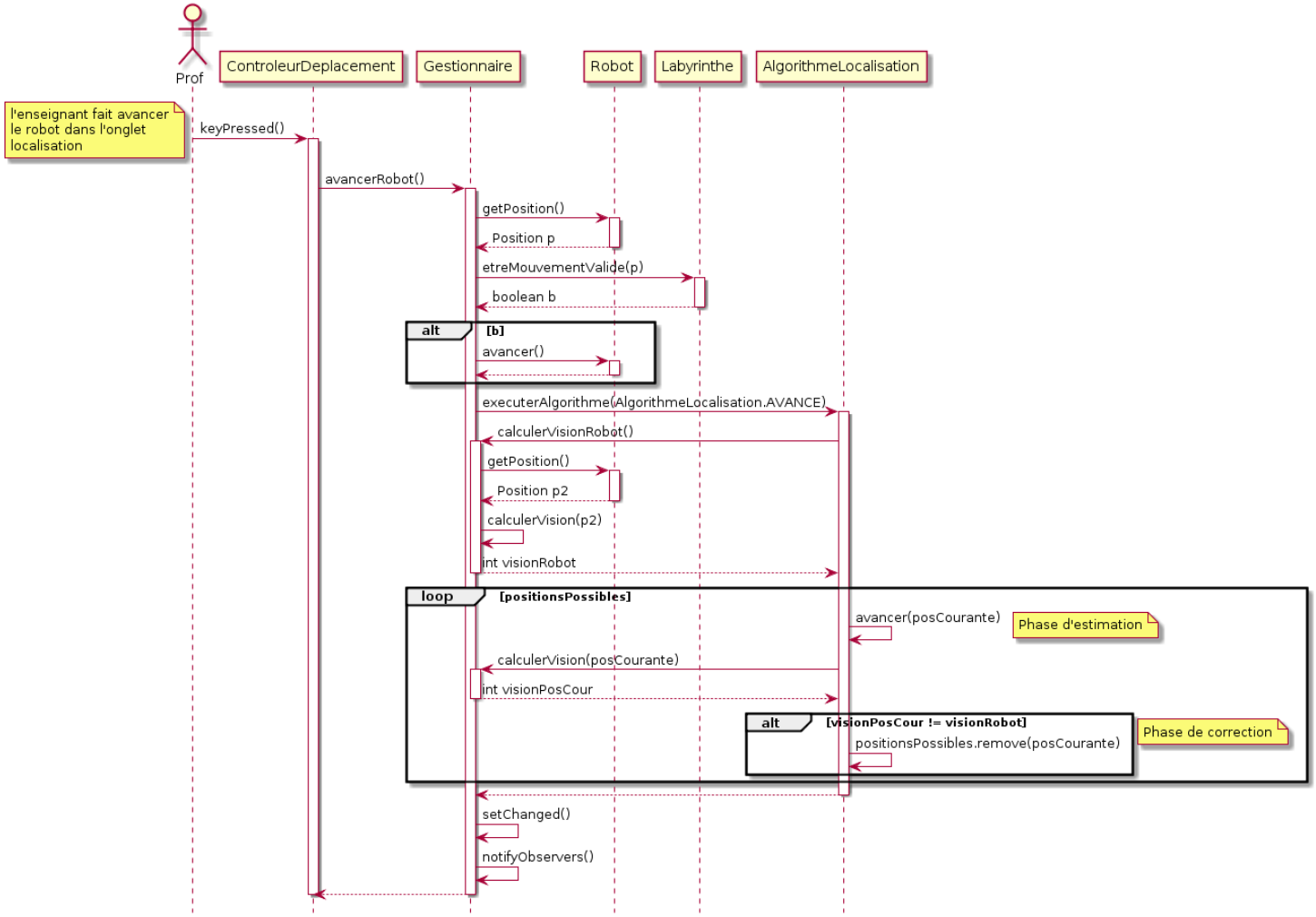
On va ensuite capturer ce que voit actuellement le robot (après déplacement).

Il est à noter que la « vision à une position » est un entier codé sur 3 bits qui correspond au type des cases adjacentes à cette position. Par exemple, si la vision vaut 3, c'est-à-dire 011 (car 011=3), cela veut dire, qu'à cette position, il n'y a pas de mur à gauche, mais il y en a un devant et à droite. Si la vision vaut 2 (010), cela veut dire qu'il y a seulement un mur devant.

À partir de là, on parcourt la liste des positions sur lesquelles le robot pouvait se trouver avant déplacement, on leur applique le déplacement et on calcule la vision à cette nouvelle position pour estimer ce que le robot verrait s'il s'était déplacé **(Phase d'estimation).**

Enfin, il ne nous reste plus qu'à regarder si ce que voit actuellement le robot correspond à la vision à chaque position. Si elles sont différentes, on oublie la position correspondante et on la supprime de la liste des positions possibles du robot **(Phase de correction)**

De plus, si l'on est dans une certaine situation et que l'on souhaite rajouter ou enlever quelques murs pour se retrouver dans une situation qui nous semble intéressante, il faut prendre en compte ces modifications dans l'estimation des cases sur lesquelles le robot peut se trouver. Pour cela, à chaque fois qu'un mur est placé à un endroit où le robot était susceptible d'être, on ajoute cette case à la liste des positions murées et on supprime les positions possibles correspondantes (méthode **supprimerCase**). Si, par la suite, on souhaite revider cette case, on réintègre les positions possibles que l'on avait supprimées et on supprime la case des positions murées. La liste des positions murées est vidée à chaque déplacement pour que l'état des estimations soit cohérent à chaque fois que le robot se déplace.


Prof
ControleurDeplacement
Gestionnaire
Robot
Labyrinthe
AlgorithmeLocalisation
l'enseignant fait avancer le robot dans l'onglet localisation
keyPressed()
avancerRobot()
getPosition()
Position p
etreMouvementValide(p)
boolean b
alt
[b]
avancer()
executerAlgorithme(AlgorithmeLocalisation.AVANCE)
calculerVisionRobot()
getPosition()
Position p2
calculerVision(p2)
int visionRobot
loop
[positionsPossibles]
avancer(posCourante)
Phase d'estimation
calculerVision(posCourante)
int visionPosCour
alt
[visionPosCour != visionRobot]
positionsPossibles.remove(posCourante)
Phase de correction
setChanged()
notifyObservers()

Recensement et évaluation des risques

Une des tâches critique qui est la plus logique et visible est bien entendu le chargement de labyrinthes. En effet, sans cette tâche, tout le projet ne peut pas fonctionner puisque sans labyrinthe, on ne peut pas utiliser d'algorithmes.

Par contre, nous ne trouvons qu'aucune des tâches présentes pour la première itération ne présente réellement de difficulté, à part peut-être l'algorithme de localisation.

Algorithmes actuels et futurs

Les algorithmes que nous allons développer dans cette itération sont :

- Génération de labyrinthe parfait (algorithme par fusion) : *existant et connu*
  - https://fr.wikipedia.org/wiki/Modélisation_mathématique_d%27un_labyrinthe#Fusion_al.C3.A9atoire_de_chemins
- Génération de labyrinthe imparfait (algorithme par fusion puis destruction aléatoire de cases murs)
- Algorithme de localisation (estimation des cases sur lesquelles le robot peut être en fonction de ses déplacements et vues) : *inconnu à implémenter*

Dans les prochaines itérations, nous développerons peut-être ces différents algorithmes

- **Main droite** (recherche d'un chemin quelconque pour sortir d'un labyrinthe) : *existant et connu*
  - https://www.wikihow.com/Find-Your-Way-Through-a-Maze
  - https://en.wikipedia.org/wiki/Maze_solving_algorithm#Wall_follower
- **Dijkstra** (recherche de chemin minimal) : *existant et inconnu*
  - https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- A* (recherche de chemin minimal basé sur une heuristique) : *existant et inconnu*
  - https://en.wikipedia.org/wiki/A*_search_algorithm
- **Pledge** (recherche de chemin quelconque pour sortir d'un labyrinthe basé sur un compteur de rotations) : *existant et inconnu*
  - https://en.wikipedia.org/wiki/Maze_solving_algorithm#Pledge_algorithm
  - https://interstices.info/jcms/c_46065/l-algorithme-de-pledge
- **Trémeaux** (recherche de chemin quelconque pour sortir d'un labyrinthe basé sur le balisage des endroits déjà parcourus ou backtracking) : *existant et inconnu*
  - https://en.wikipedia.org/wiki/Maze_solving_algorithm#Tr.C3.A9maux.27s_algorithm
  - http://blog.jamisbuck.org/2014/05/12/tremauxs-algorithm.html
  - http://www.cems.uvm.edu/~rsnapp/teaching/cs32/notes/tremaux_rules.pdf