Guillaume DUBUISSON DUPLESSIS
Pauline REQUENA
Arnaud FAURE
Département GM - 4ème année

Rapport de mini-projet C++

Sujet : Résolution du monde des cubes par éco-résolution

A l'attention de M. Jean-Philippe Kotowicz

Année scolaire 2008/2009 – 2ème semestre GM4

Sommaire

Introduction

blabla

Partie 1 : Analyse des besoins

1.1 Préliminaires

1.1.1 Principe de l'éco-résolution

L'éco-résolution est utilisée pour la résolution des problèmes en Intelligence Artificielle.
Elle se compose de 2 parties :

- Un protocole suivi par l'ensemble des agents, qui est un noyau indépendant du problème à résoudre
- Un code de comportements des éco-agents spécifiques au problème à résoudre

Les éco-agents sont les entités qui constituent le système. Leur particularité est d'être en quête perpétuelle d'un état de satisfaction. Les éco-agents peuvent se gêner mutuellement ce qui donne naissance à deux comportements : l'agression des gêneurs et la fuite de ceux-ci. Ils sont également caractérisés par :

- Un but : il s'agit d'un autre éco-agent avec lequel il dit être en relation de satisfaction
- Un état interne : satisfait, en recherche de satisfaction, en fuite ou en recherche de fuite
- Des actions élémentaires : elles dépendent du domaine et correspondent aux comportements de satisfaction ou de fuite
- La perception des gêneurs : Il s'agit de la perception des éco-agents qui empêchent l'éco-agent courant d'être satisfait ou de fuir
- Des dépendances : les éco-agents dont l'éco-agent courant est le but. Elles sont satisfaites uniquement si cet éco-agent est satisfait.

Un éco-agent a la volonté d'être satisfait. Il cherche à se trouver dans un état de satisfaction.
S'il est empêché par des gêneurs alors il les agresse.
Un éco-agent a l'obligation de fuir. Si un éco-agent est agressé, il doit trouver une place ou fuir.

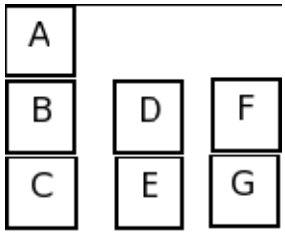
Enfin un éco-agent peut effectuer 3 opérations :

- Agresser
- FaireSatisfaction
- FaireFuite

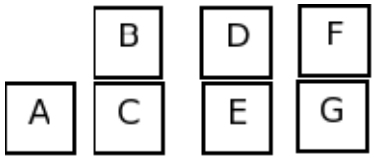
1.1.2 Le monde des cubes

Le monde des cubes consiste en le problème suivant : des cubes sont disposés sur une table formant des piles et l'objectif de pouvoir bouger les cubes suivant des contraintes (poser le cube A sur le cube H par exemple).

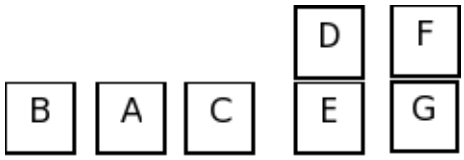
Prenons l'exemple de la situation suivante :



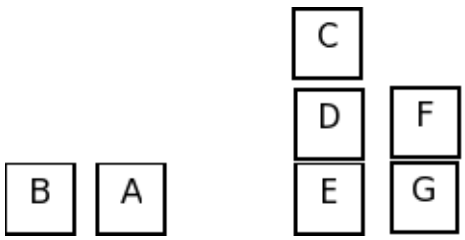
L'objectif est de déplacer le cube C et de le mettre sur le cube D. Cette opération sera réalisée selon les étapes suivantes :



Le cube A est posé sur la table.



Le cube B est posé sur la table, ainsi le cube C est libre.



Finalement, le cube C est déplacé sur le cube D.

Plusieurs options existent pour la résolution de ce problème : l'utilisation de robots qui déplaceraient les cubes et l'éco-résolution avec les cubes et la table comme éco-agents. Nous avons choisi cette dernière option même si de premier abord elle paraît être moins instinctive car nous pensons que ce problème illustre parfaitement l'utilisation de l'éco-résolution.

1.2 Composition du système

L'objectif affiché est donc de résoudre le problème du déplacement des cubes par l'éco-résolution. Tout problème possède néanmoins une base commune : la plateforme d'éco-résolution qui permet l'exécution des éco-agents et les éco-agents eux-mêmes.

1.2.1 Plateforme d'éco-résolution

La plateforme d'éco-résolution qui permet :

- l'ajout d'éco-agent
- la suppression d'éco-agent
- l'exécution des éco-agents

1.2.2 Les éco-agents

Les éco-agents précédemment décrits qui sont capables de réaliser principalement trois opérations:

- Agresser
- FaireSatisfaction
- FaireFuite

Les états successifs des éco-agents peuvent être décrits par l'automate suivant :

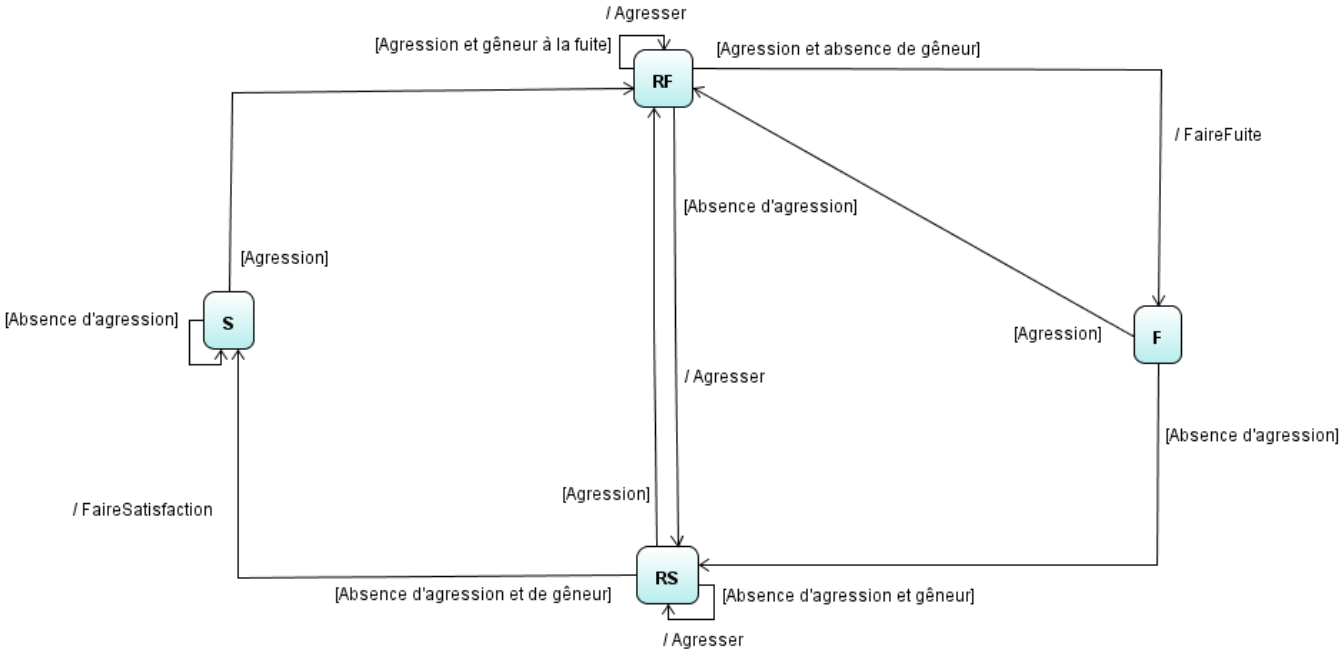


Diagramme d'états d'un EcoAgent

1.3 Les grandes fonctionnalités

On peut distinguer les grandes fonctionnalités suivantes :

- Créer une situation initiale : positionnement initial des cubes sur la table
- Déterminer la situation finale : positionnement final des cubes sur la table
- Démarrer la résolution
- Trace de la résolution (affichage graphique, log, etc.)

1.4 Les étapes pour démarrer la résolution

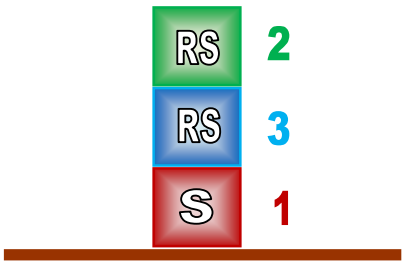
Nous allons maintenant résumer les différentes étapes nécessaires pour réaliser la résolution:

1. Création de l'éco-agent table
2. Création des éco-agents cubes
3. Donner aux éco-agents des conditions de satisfactions ainsi que les relations de dépendance qui en découlent
4. Démarrer la résolution

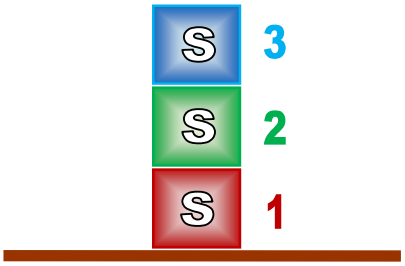
1.5 Exemple de résolution

La première étape consiste à initialiser les éco-agent Table et Cubes. Ensuite, on leur attribue une situation initiale, des conditions de satisfactions, et les relations de dépendances qui en découlent.

Situation initiale



Objectif final



Dans la situation initiale, le cube 1 est posé sur la table, le cube 3 est posé sur 1 et le cube 2 est posé sur le cube 3.

On donne aux cubes les conditions de satisfaction suivantes :

→ { sur(Cube 1 , Table), sur(Cube 2, Cube 1), sur(Cube 3, Cube 2) }

On en déduit donc les relations de dépendances suivantes :

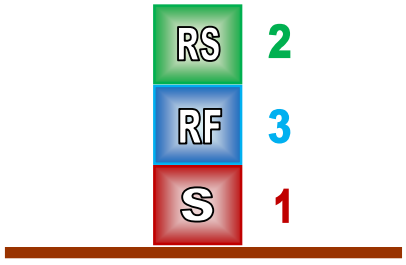
→ { Cube 1 dépend de Table, Cube 2 dépend de Cube 1, Cube 3 dépend de Cube 2 }

L'éco-agent Table est toujours satisfait. Dans la situation initiale, le cube 1 est donc satisfait, et les cubes 2 et 3 sont en recherche de satisfaction.

Une fois que tous ses éléments sont déterminés, la résolution en elle-même peut commencer.

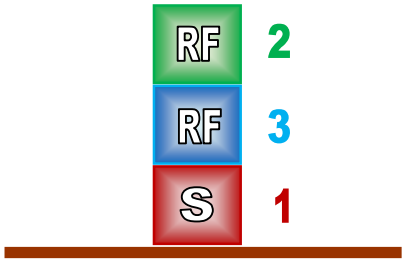
Etape 1

Parmi les cubes non satisfaits se trouvent les cubes 2 et 3. Le cube 2 est prioritaire car il doit être placé plus bas que le cube 3 dans la pile de cubes. On sélectionne donc le cube 2 comme cube courant. Dans la situation initiale, le cube 3 est un gêneur direct du cube 2 car il fait obstacle à la satisfaction directe du cube 3. Le cube 2 va donc agresser le cube 3, et le cube 3 va passer en recherche de fuite.



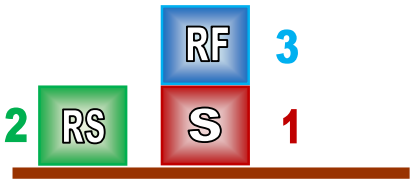
Etape 2

Le cube courant devient le cube en recherche de fuite, i.e. le cube 3. Le cube 2 est un gêneur du cube 3, car c'est un obstacle à la fuite de 3. Le cube 3 va donc agresser le cube 2 pour l'obliger à fuir, et le cube 2 va passer en recherche de fuite.



Etape 3

Le cube courant devient le cube 2, car il s'agit du cube le « plus haut placé » en recherche de fuite. Il n'est gêné par aucun autre cube, donc il peut fuir. Il fuit donc sur la table, et passe en recherche de satisfaction.

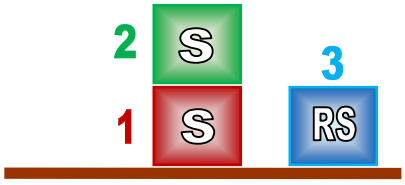


Etape 4

Le cube courant devient le cube 3, car il s'agit du seul cube en recherche de fuite. Il n'est gêné par aucun autre cube, donc il peut fuir. Il fuit donc sur la table, et passe en recherche de satisfaction.

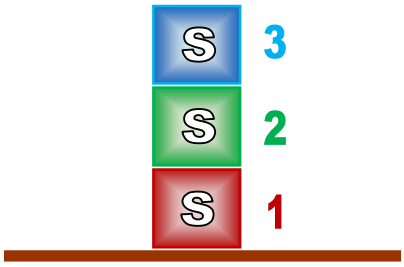
Etape 5

Le cube courant devient le cube 2, car parmi les 2 cubes en recherche de satisfaction, il doit être placé plus bas dans la pile de cube. Il n'y a aucun gêneur à sa satisfaction donc il peut se satisfaire et se poser sur le cube 1. Son état interne devient *S*.



Etape 6

Le cube courant devient le cube 3, car il s'agit du seul cube non satisfait. Il n'y a aucun gêneur à sa satisfaction donc il peut se satisfaire et se poser sur le cube 2. Son état interne devient *S*.



Partie 2 : Spécifications techniques

2.1 Cas d'utilisation

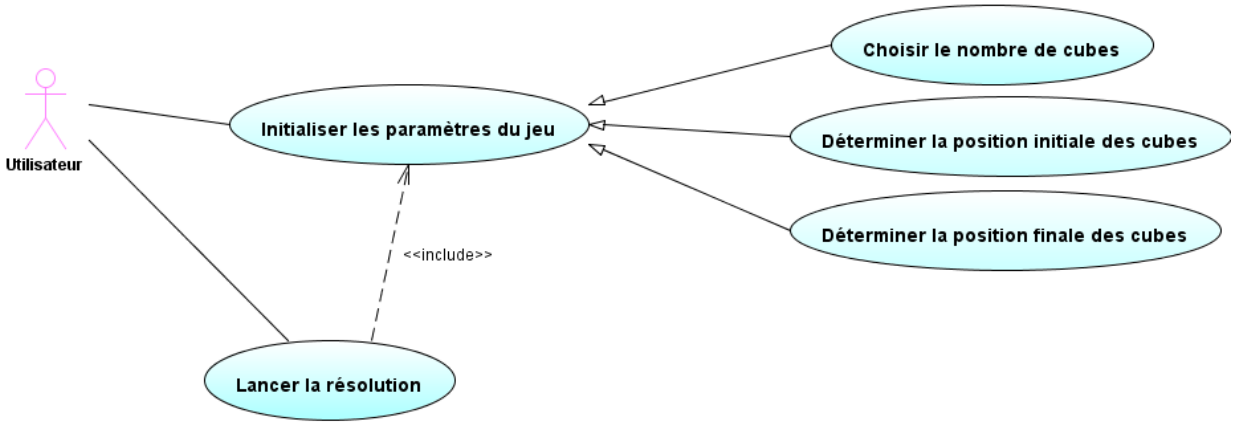
2.1.1 Identification des acteurs

Dans le cadre de cette application d'éco-résolution du monde des cubes, nous pouvons déterminer un acteur principal :

- l'utilisateur de l'application

Celui-ci utilise l'application dans le but de résoudre un problème de son choix. Il doit donc déterminer les paramètres du problème, à savoir le nombre de cubes ainsi que leurs positions initiale et finale, et ensuite il peut lancer la résolution du problème.

2.1.1 Diagramme des cas d'utilisation



2.2 Les scénarii

Dans cette partie, nous allons décrire les scénarii composant la résolution du monde des cubes par éco-résolution. Nous pouvons lister les scénarii suivants :

- Initialisation du jeu
- Résolution
- Cas d'un cube en recherche de satisfaction
- Cas d'un cube en recherche de fuite
- Cas d'une agression
- Satisfaction d'un cube
- Fuite d'un cube

2.2.1 Scénario « Initialisation du jeu »

Titre : Initialisation du jeu

Résumé : Ce scénario permet de mettre en place un problème dans le monde des cubes

Acteur : L'utilisateur

Pré conditions :

La résolution n'est pas lancée

Scénario nominal :

1. Le système demande à l'utilisateur le nombre de cubes.
2. L'utilisateur insère le nombre de cubes souhaités.
3. Le système vérifie le nombre.
4. Le système demande à l'utilisateur un identifiant pour la table.
5. Pour chaque cube, le système demande à l'utilisateur un identifiant.
6. Pour chaque cube :
   6.1 Le système demande sa position courante et sa position finale.
   6.2 Le système vérifie s'il y a une incohérence.
7. Les cubes se mettent en état interne « satisfaction » si les positions courante et finale sont identiques, en état interne « recherche de satisfaction » si les positions courante et finale sont différentes.

Enchaînement alternatifs :

A1 : le nombre de cubes n'est pas valide (n'appartient pas à N)*
L'enchaînement A1 démarre au point 3 du scénario nominal.
4. Le système indique à l'utilisateur que le nombre n'est pas valide.

Le scénario nominal revient au point 1.

A2 : les informations entrées sont incorrectes
L'enchaînement A2 démarre au point 6.2 du scénario nominal (la position courante n'existe pas ou le cube actuel possède la même position courante qu'un autre cube déjà entré, ou alors des positions finales forment un cycle, i.e. des cubes veulent tous finir les uns sur les autres).
6.3 Le système indique à l'utilisateur que les données sont incorrectes.

Le scénario nominal revient au point 6.1.

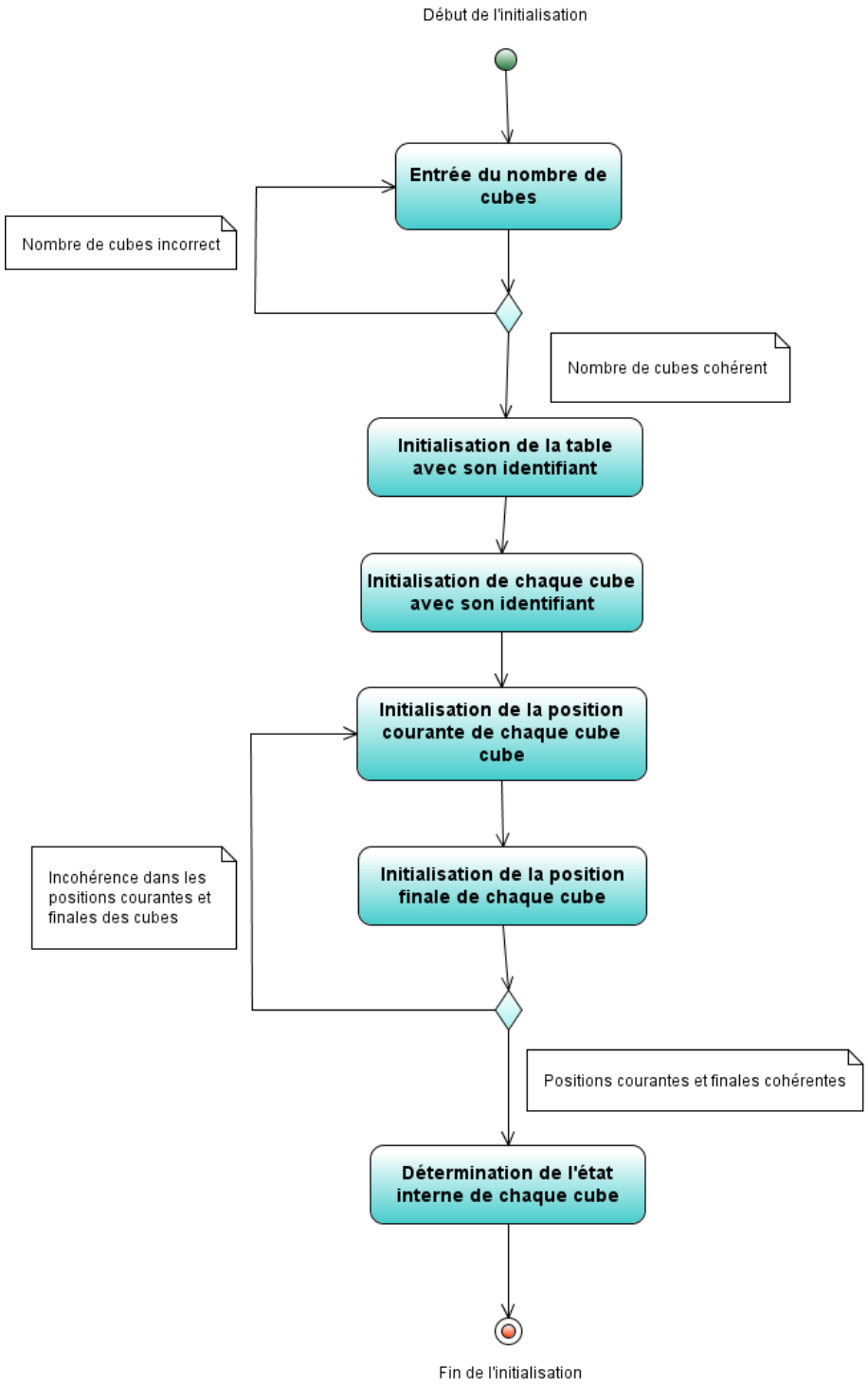


Diagramme d'activité – Initialisation du jeu

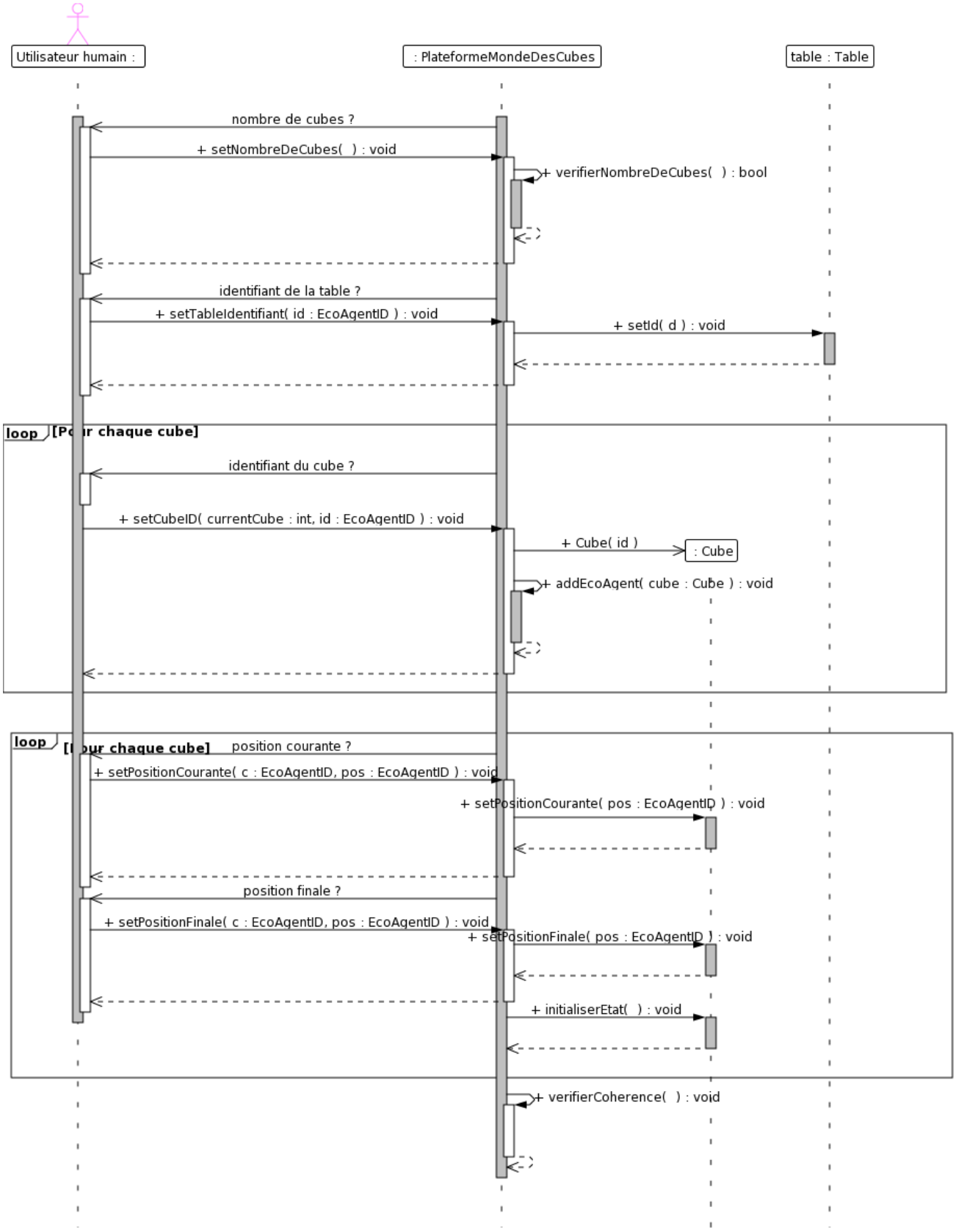


Diagramme de séquence – Initialisation du jeu

2.2.2 Scénario « Résolution du jeu »

Titre : Résolution du jeu

Résumé : Ce scénario permet d'effectuer la résolution des problèmes dans le monde des cubes

Acteur : N/A

Pré conditions :

L'initialisation s'est bien déroulée

Scénario nominal :

1. Tant que tous les éco-agents ne sont pas en état interne « satisfaction ».
   1.1 Le système sélectionne l'éco-agent prioritaire qui doit agir en fonction de ses critères propres.
   1.2 Le système demande à cet agent d'agir.
2. Fin de résolution.

Enchaînement alternatifs :

A1 : les éco-agents sont tous dans l'état interne « satisfait »
L'enchaînement A1 démarre au point 1 du scénario nominal.
2. Fin de résolution.

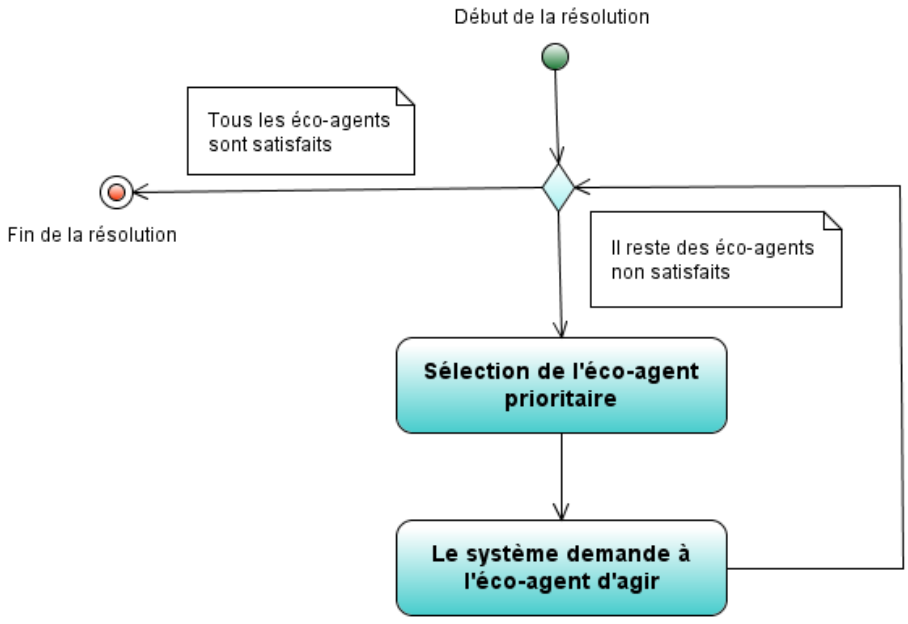


Diagramme d'activité – Résolution du jeu

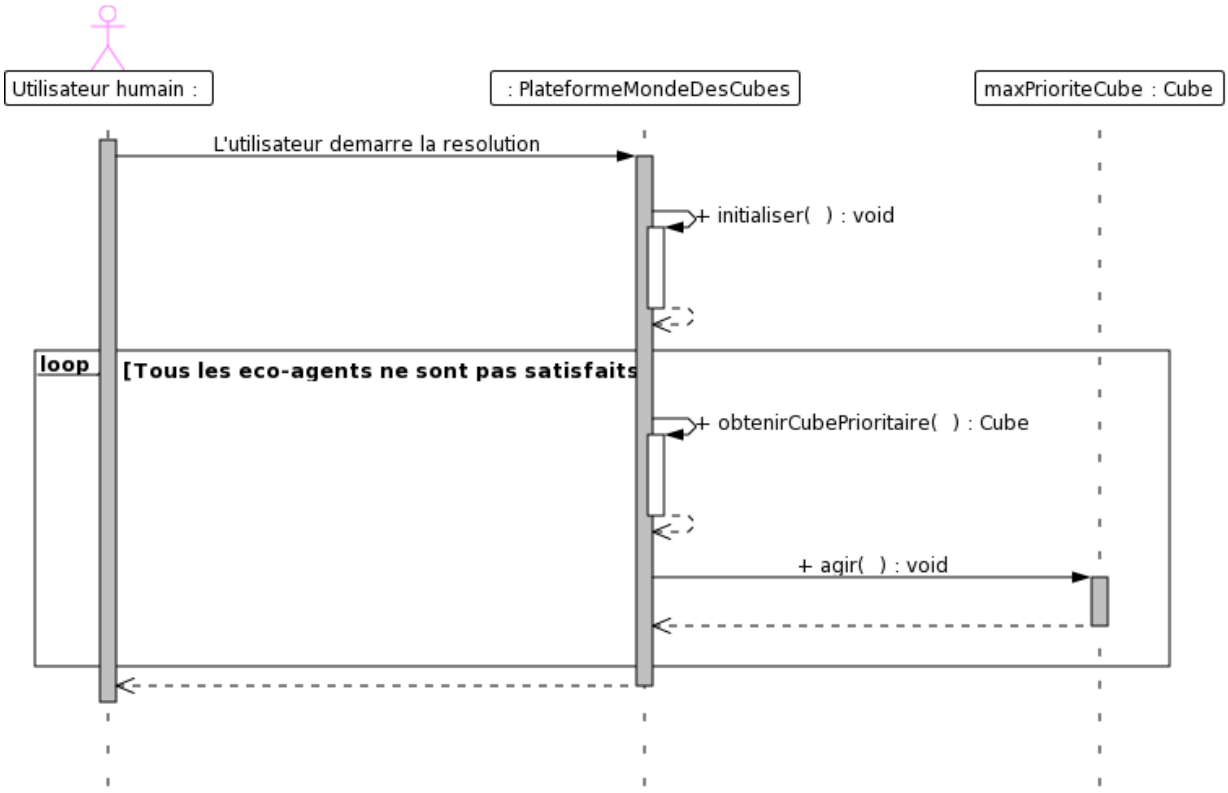


Diagramme de séquence – Résolution du jeu

2.2.3 Scénario « Cas d'un cube en recherche de satisfaction »

Titre : Recherche de satisfaction d'un cube

Résumé : Ce scénario traite le cas d'un cube dans l'état interne « recherche de satisfaction ».

Acteur : éco-agent cube

Pré conditions :

La résolution est en cours
Le cube doit être dans l'état « recherche de satisfaction »

Scénario nominal :

1. Le cube vérifie si aucun autre cube ne le gêne.
2. Le cube se déplace et change sa position courante.
3. Le cube se met en état interne « satisfaction ».

Enchaînement alternatifs :

A1 : Le cube est gêné par un autre cube positionné sur lui
L'enchaînement A1 démarre au point 1 du scénario nominal.
2. Le cube agresse le cube gêneur.

A2 : Le cube est gêné par un autre cube positionné sur sa position finale
L'enchaînement A1 démarre au point 1 du scénario nominal.
2. Le cube agresse le cube gêneur.

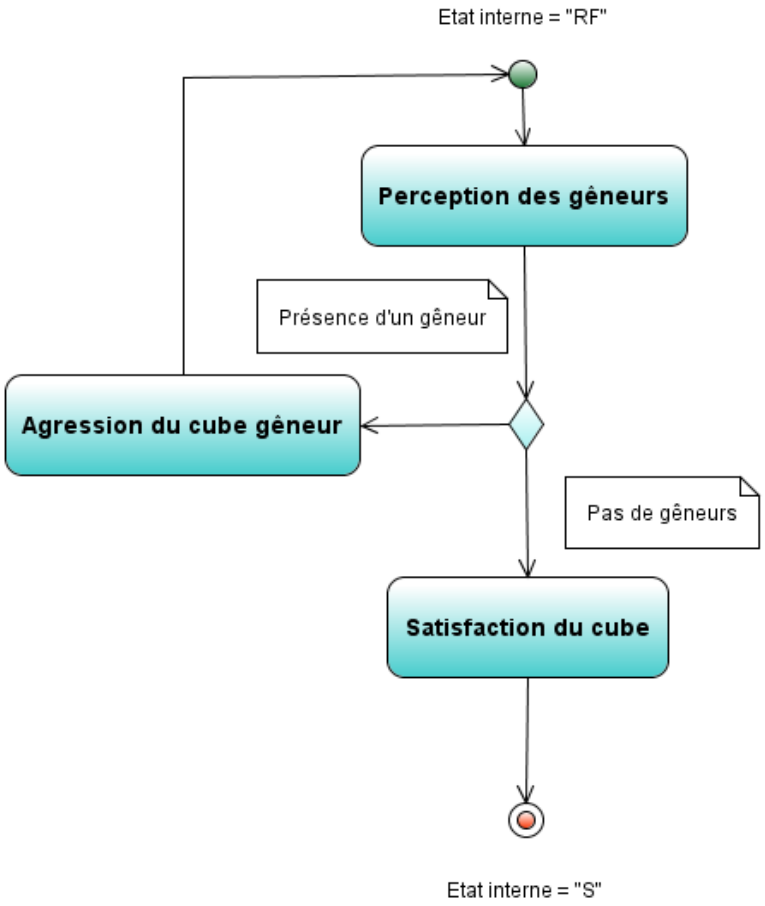


Diagramme d'activité – Recherche de satisfaction

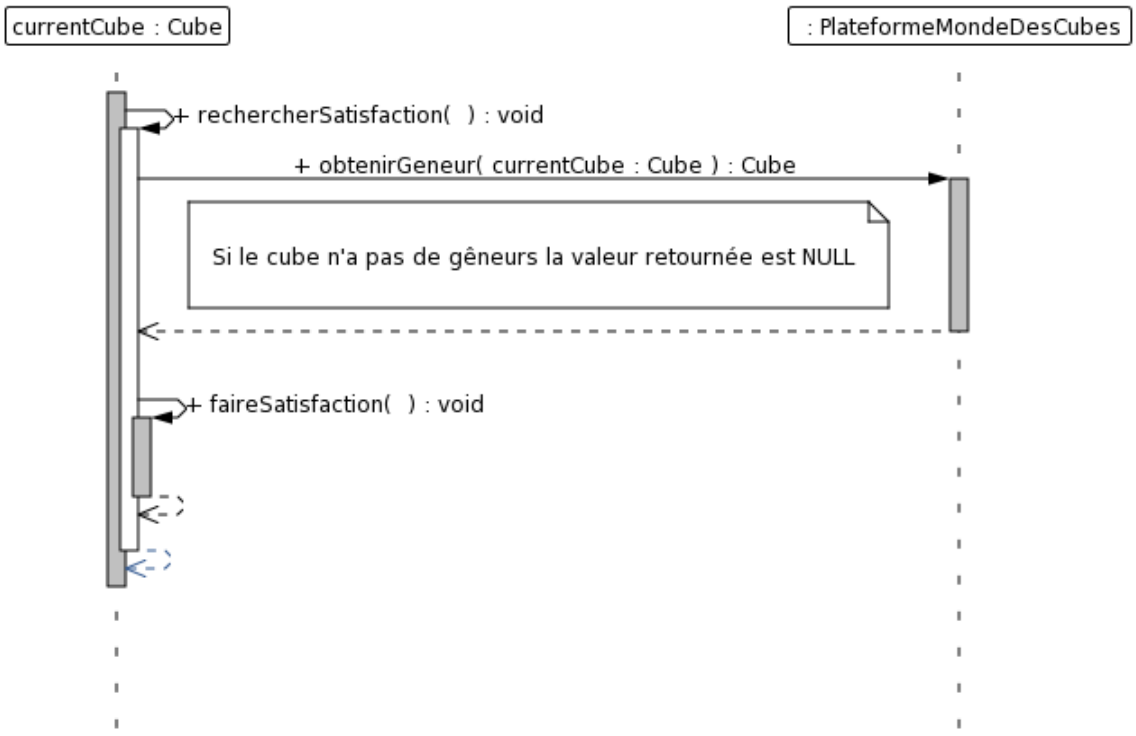


Diagramme de séquence – Recherche de satisfaction

2.2.4 Scénario « Cas d'un cube en recherche de fuite »

Titre : Recherche de fuite d'un cube

Résumé : Ce scénario traite le cas d'un cube dans l'état interne « recherche de fuite ».

Acteur : éco-agent cube

Pré conditions :

La résolution est en cours.
Le cube doit être dans l'état interne « recherche de fuite ».

Scénario nominal :

1. Le cube vérifie si aucun autre cube ne le gêne dans sa fuite.
2. Le cube se déplace et change sa position finale.
3. Le cube se met dans l'état interne « recherche de satisfaction».

Enchaînement alternatifs :

A1 : Le cube est gêné par un autre cube
L'enchaînement A1 démarre au point 1 du scénario nominal.

2. Le cube agresse le cube gêneur.

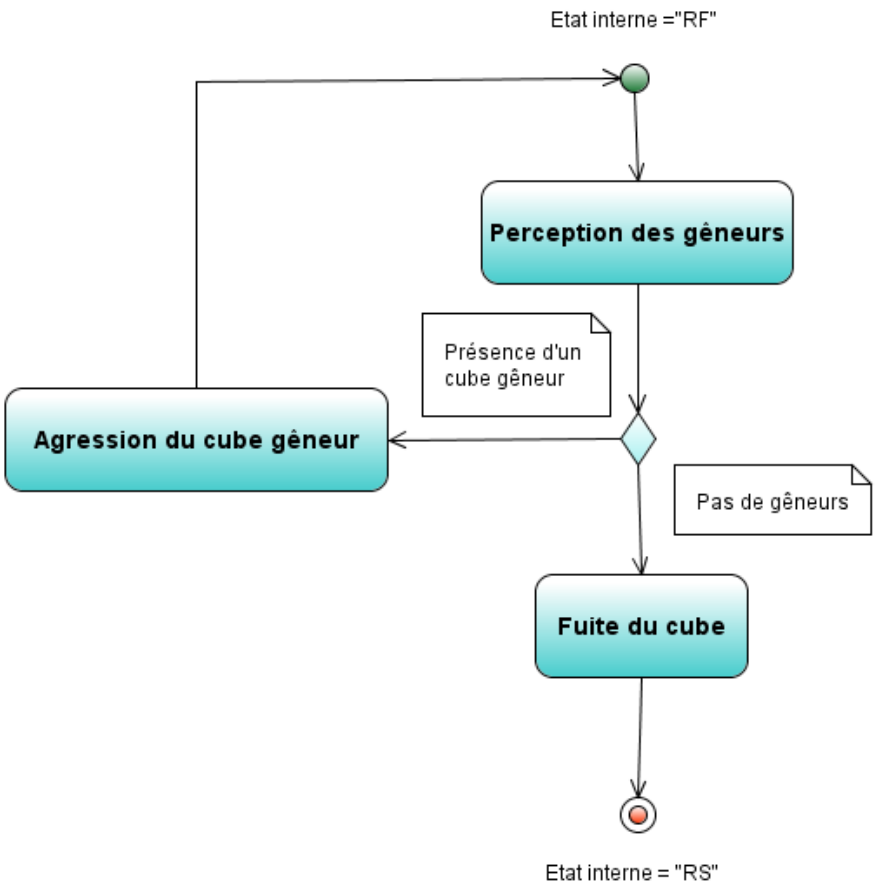


Diagramme d'activité – Recherche de fuite

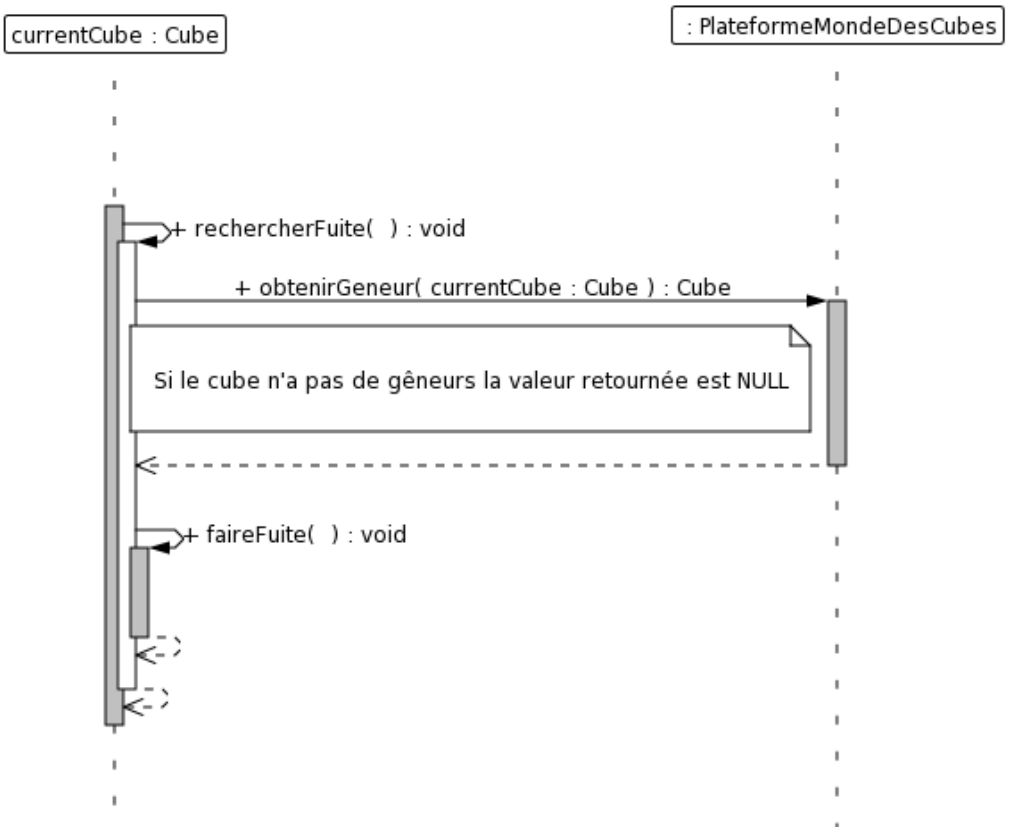


Diagramme de séquence – Recherche de fuite

2.2.5 Scénario « Cas d'une agression »

Titre : Agression d'un cube par un autre cube

Résumé : Ce scénario permet de traiter le cas d'une agression d'un cube par un autre cube.

Acteurs : éco-agents cubes

Pré conditions :

La résolution est en cours.
Un cube doit être gêné par un autre cube.

Scénario nominal :

1. Un cube dans l'état interne « recherche de satisfaction » ou « recherche de fuite » agresse un cube le gêneur.
2. L'état interne du deuxième cube change en mode « recherche de fuite ».

Enchaînement alternatifs :

N/A

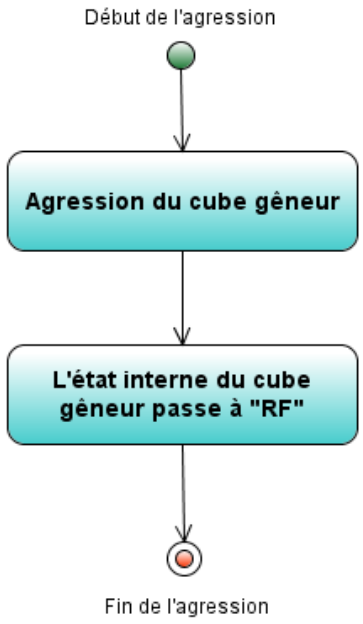


Diagramme d'activité – Agression d'un cube

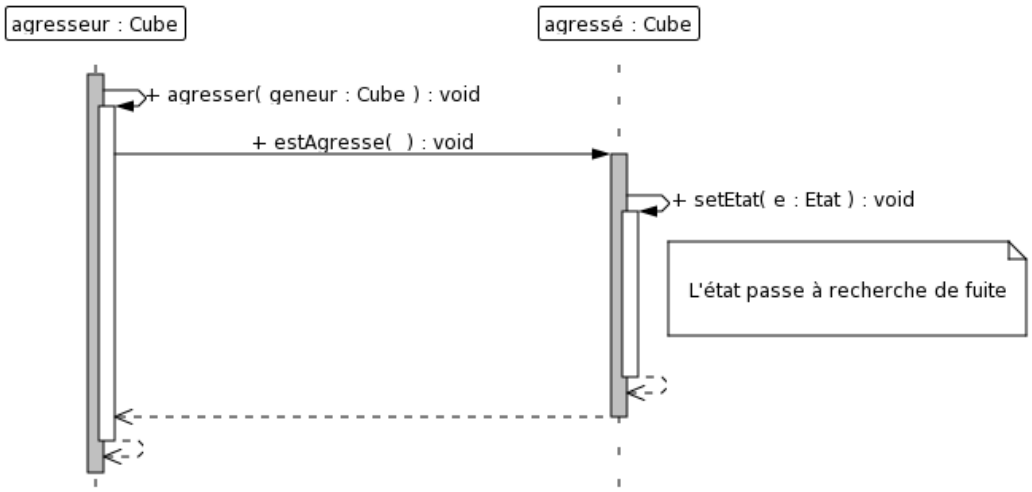


Diagramme de séquence – Agression d'un cube

2.2.6 Scénario « Satisfaction d'un cube »

Titre : Satisfaction d'un cube

Résumé : Ce scénario décrit la satisfaction d'un cube. Après avoir été en recherche de fuite ou en recherche de satisfaction, si le cube a l'occasion de se satisfaire, il le fait.

Acteurs : éco-agents cubes

Pré-conditions

La résolution est en cours.
Le cube doit être en état interne de « recherche de fuite » ou de « recherche de satisfaction ».

Scénario nominal

1. La position courante du cube devient sa position finale, i.e. sa position de satisfaction.
2. L'état interne du cube devient « *S* ».

Enchaînement alternatifs :

N/A

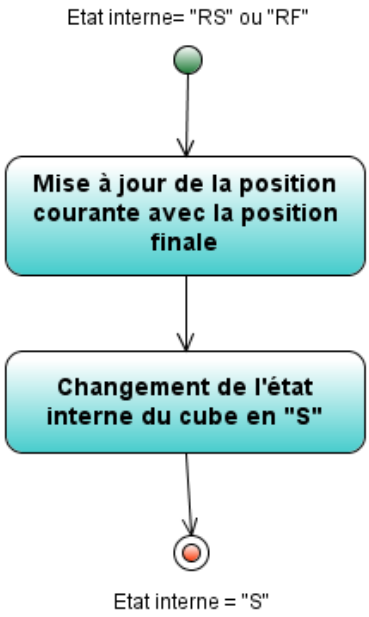


Diagramme d'activité – Satisfaction d'un cube

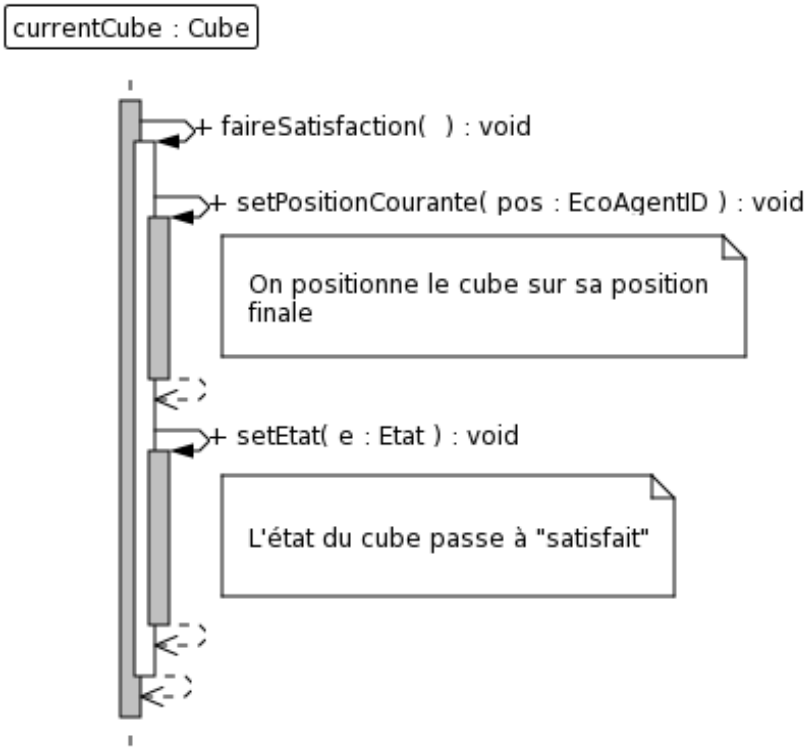


Diagramme de séquence – Satisfaction d'un cube

2.2.7 Scénario « Fuite d'un cube »

Titre : Fuite d'un cube

Résumé : Ce scénario décrit la fuite d'un cube. Après avoir été en recherche de fuite, si le cube a l'occasion de fuir, il est dans l'obligation de le faire.

Acteurs : éco-agents cubes

Pré-conditions

La résolution est en cours.
Le cube doit être en état interne de « recherche de fuite ».

Scénario nominal

1. La position courante du cube devient la table.
2. L'état interne du cube devient « RS ».

Enchaînement alternatifs :

N/A

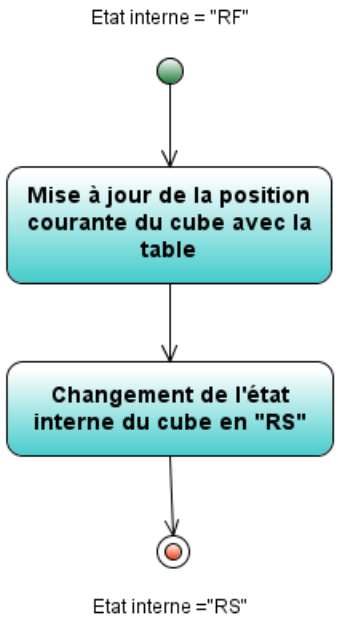


Diagramme d'activité – Fuite d'un cube

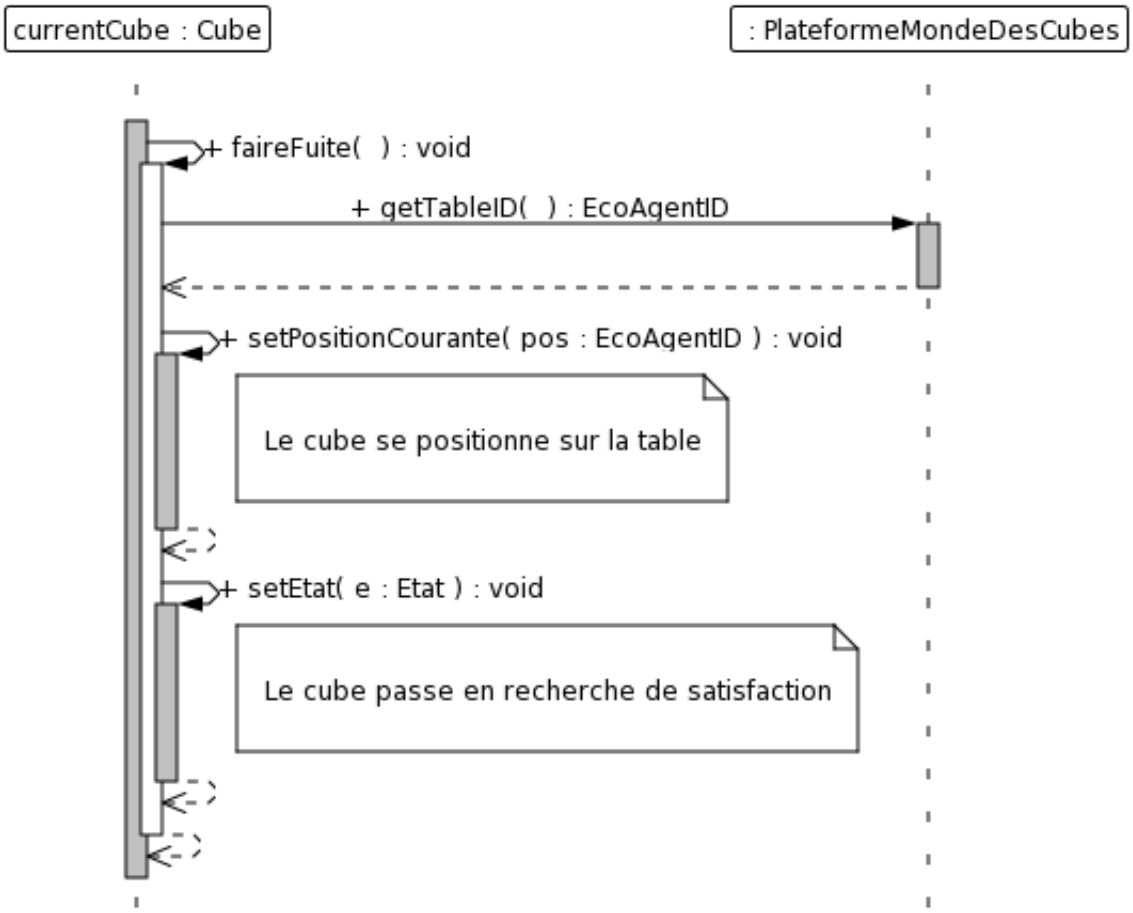


Diagramme de séquence – Fuite d'un cube

2.2.8 Scénario « Vérifier cohérence des informations »

Lors de l'initialisation, le système utilise la fonction vérifierCohérence(), pour vérifier que les positions initiale et finales des cubes sont possibles.

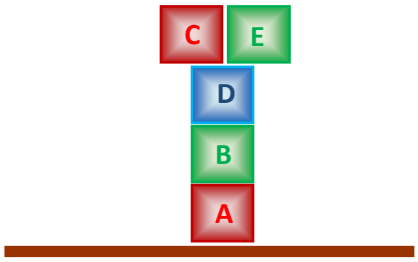
Cette fonction teste en fait si les règles suivantes sont respectées :

Règle 1 :
Les positions courante et finale du cube existent.

Règle 2 :
Il faut que tous cubes soient rattachés directement ou indirectement à la table (positions initiale et finale).
I.E : aucun cube en lévitation.

Règle 3 :
Aucun cube ne partage sa position courante et sa position final avec un ou plusieurs autres cubes.
I.E. : On liste les occurrences des cubes en tant que positions courantes et positions finales. Aucun décompte ne doit être strictement supérieur à 1 :

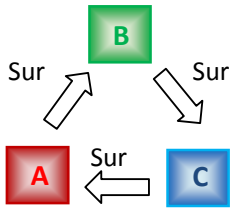
Exemple :



Cubes	Cube A	Cube B	Cube C	Cube D	Cube E
Occurrences en tant que positions courantes	1	1	0	2	0
Occurrences en tant que positions finales	1	1	1	1	1

Règle 4 :
Il n'existe pas de cycle.
I.E : les cubes ne peuvent pas faire une boucle.

Exemple :



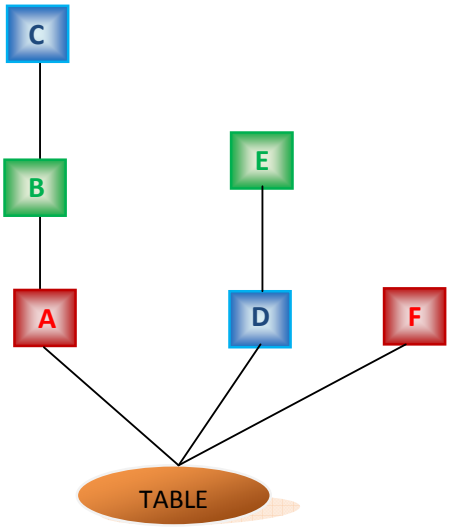
Remarque : ce problème revient au problème de lévitation posé plus haut. En effet, si des cubes sont positionnés les uns sur les autres, ils ne pourront pas être sur la table.

Réflexion :

Le respect de l'ensemble de ces règles revient à vérifier qu'on obtient bien un seul arbre initial et final avec :
- Comme unique racine la table.
- Plusieurs branches pouvant partir de la racine table.
- Au plus une seule branche part et arrive d'un cube.

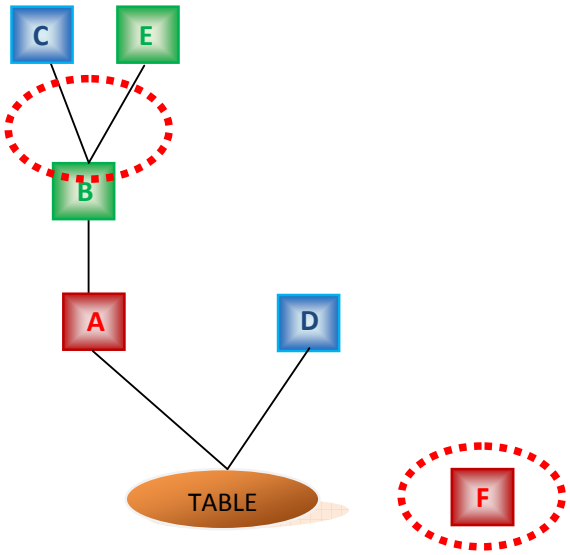
Exemples :

Bon graphe respectant les règles :



Mauvais graphe ne respectant pas 2 règles :

On observe deux arbres et en plus il y a deux branches qui arrivent à B.



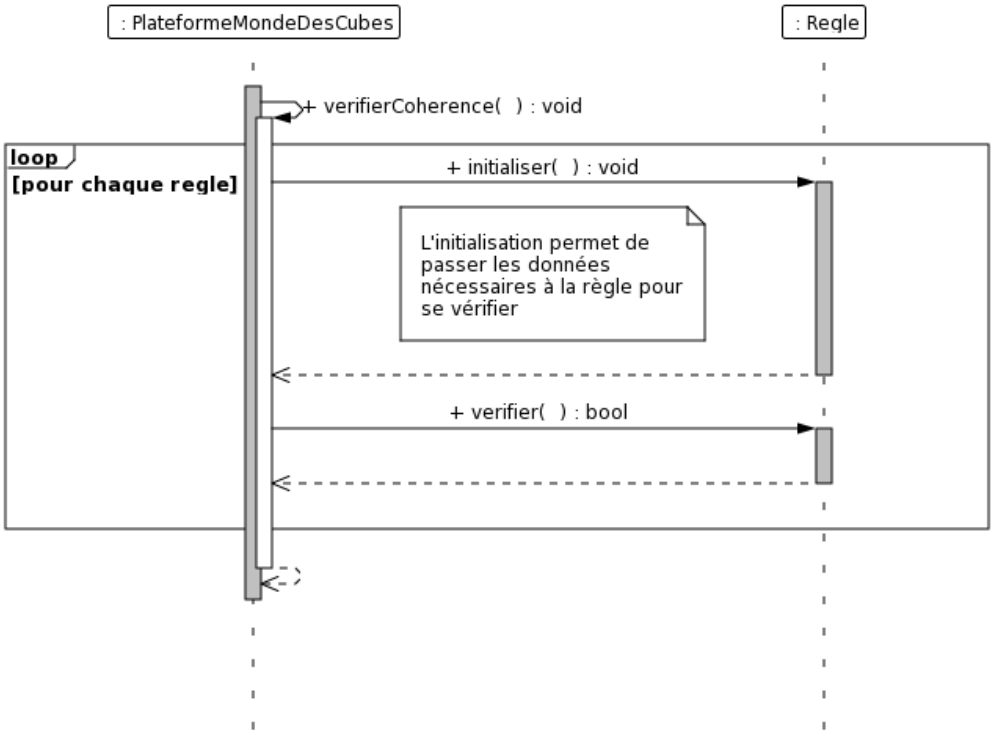


Diagramme de séquence – Vérifier cohérence

En résumé…

Nous avons donc deux règles principales :

- **Les cubes forment un arbre dont la racine est la table.**

  Donc les cubes ont tous des positions courantes et finales définies (ID différents de NULL) et valides (il existe bien un éco agent possédant cet ID), liant directement ou indirectement à l'éco agent Table (c'est-à-dire qu'il n'y a pas non plus de boucle).

- **Aucun cube ne supporte plus de un cube.**

  Donc si on met en place un compteur du nombre de cubes positionnés sur chaque cube au début et à la fin, ce compteur doit être égal à 0 ou 1 pour chaque cube.

Règle 1 : Les cubes forment un arbre dont la racine est la table

```
reliesATable(): boolean
Var: b boolean
Début
Pour un i parcourant la map <EcoAgentID,Cube> obtenue par PlateFormeMondeDesCubes.getCubes()
        Pour un j parcourant la map
                « Cube associé à j ».setVisite(faux)
        FinPour
        b <- « Cube associé à i ».estRelieATable()
        Si b=faux Alors retourne b
        FinSi

        Pour un j parcourant la map
                « Cube associé à j ».setVisite (faux)
        FinPour
        b <- « Cube associé à i ».seraRelieATable()
        Si b=faux Alors retourne b
        FinSi
FinPour
retourne b
Fin
```

```
estRelieATable(): boolean
Début
Si getVisite()=vrai
        Alors retourne faux
Sinon
        setVisite(vrai)
        Si getPositionCourante()=NULL
                Alors retourne faux
        Sinon
                Si getPositionCourante()=PlateFormeMondeDesCubes.getTableID()
                        Alors retourne vrai
                Sinon   Si PlateFormeMondeDesCubes.getEcoAgent(getPositionCourante())=NULL
                                Alors retourne faux
                        Sinon
                          retourne estRelieATable(PlateformeMondeDesCubes.getEcoAgent(getPositionCourante()))
                        FinSi
                FinSi
        FinSi
FinSi
Fin



seraRelieATable(): boolean
Début
Si getVisite()=vrai
        Alors retourne faux
Sinon
        setVisite(vrai)
        Si getPositionFinale()=NULL
                Alors retourne faux
        Sinon
                Si getPositionFinale()=PlateFormeMondeDesCubes.getTableID()
                        Alors retourne vrai
                Sinon
                        Si PlateFormeMondeDesCubes.getEcoAgent(getPositionFinale())=NULL
                                Alors retourne faux
                        Sinon
                                retourne seraRelieATable(PlateformeMondeDesCubes.getEcoAgent(getPositionFinale()))
                        FinSi
                FinSi
        FinSi
FinSi
Fin
```

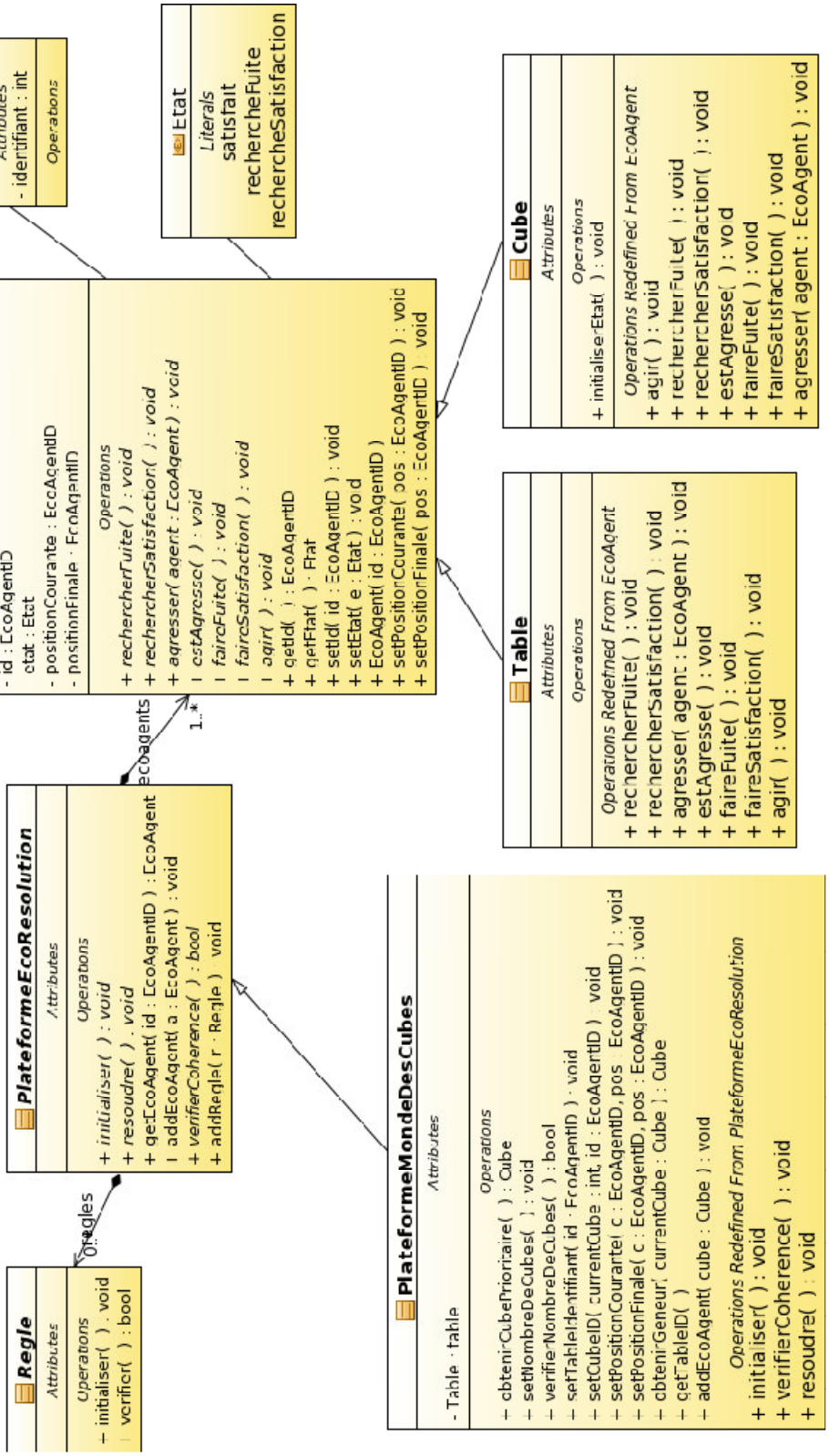
Règle 2 : Aucun cube ne supporte plus de 1 cube

AucuneSurcharge() : boolean
Début
Si PasSurcharges() = faux alors retourne faux
FinSi
Si SerontPasSurcharges() = faux alors retourne faux
FinSi
retourne vrai
Fin

PasSurcharges(): boolean
Var: i = Map de <EcoAgentID,int>
Début
Pour j parcourant la map <EcoAgentID,Cube> obtenue par PlateFormeMondeDesCubes.getCubes()
    Si l'EcoAgentID positioncourante du cube associé à j n'est pas dans i
        Alors on ajoute l'EcoAgentID dans i et l'entier associé = 0
    FinSi
    On incrémente de 1 l'entier associé à l'EcoAgentID positioncourante du cube associé au j actuel
    Si l'entier associé à l'EcoAgentID est > 1 alors retourne faux
    FinSi
FinPour
retourne vrai
Fin

SerontPasSurcharges(): boolean
Var: i = Map de <EcoAgentID,int>
Début
Pour j parcourant la map <EcoAgentID,Cube> obtenue par PlateFormeMondeDesCubes.getCubes()
    Si l'EcoAgentID positionfinale du cube associé à j n'est pas dans i
        Alors on ajoute l'EcoAgentID dans i et l'entier associé = 0
    FinSi
    On incrémente de 1 l'entier associé à l'EcoAgentID positionfinale du cube associé au j actuel
    Si l'entier associé à l'EcoAgentID est > 1
        Alors retourne faux
    FinSi
FinPour
retourne vrai
Fin

2.3 Diagramme de classes



Partie 3 : Conception

3.1. Fichiers hpp et documentation Doxygen

Voici la liste des fichiers hpp :

- ecoAgent.hpp
- cube.hpp
- table.hpp
- ecoAgentID.hpp
- etat.hpp
- singleton.hpp
- plateformeEcoResolution.hpp
- plateformeMondeDesCubes.hpp

Un fichier zippé contenant tous les fichiers hpp ainsi que la documentation Doxygen générée en PDF sera joint au rapport.

3.2. Test unitaires

a) Classe EcoAgentID

Classe	Méthode	Test	Donnees		Résultat attendu	Principe
EcoAgentID	operator==	egalité	EcoAgentID(identifiant=5)	EcoAgentID(identifiant=5)	TRUE	-
		inégalité	EcoAgentID(identifiant=5)	EcoAgentID(identifiant=2)	FALSE	-
	getId	Obtention de l'ID correct	EcoAgentID(identifiant=1)	-	1	-
	generateID	Bonne incrémentation du nombre de generations	-	-	0 puis 1 puis 2 puis 3 etc.	On affiche l'attribut statique nombre de générations avant et après création d'EcoAgentID

b) Classe EcoAgent

Classe	Méthode	Test	Donnees		Résultat attendu
EcoAgent	getEtat	état déterminé	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	Etat = RS
		état indéterminé	EcoAgent e (EcoAgentID(id=13), etat = NULL, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	NULL
	getId	ID déterminé	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	EcoAgentID(id=13)
		ID indéterminé	EcoAgent e (EcoAgentID(id=NULL), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	NULL
	setEtat	test sur un EcoAgent quelconque	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	Etat = RF	Etat prend la valeur RF
	setId	test sur un EcoAgent quelconque	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	EcoAgentID(id=9)	l'identifiant de l'EcoAgentID prend la valeur 9
	getPositionCourante	positionCourante déterminée	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	EcoAgentID(id=3)
		positionCourante indéterminée	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : NULL, positionFinale : EcoAgentID(id=11))	-	NULL
	setPositionCourante	test sur un EcoAgent quelconque	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	positionCourante : EcoAgentID(id=7)	modification de positionCourante : l'identifiant de l'EcoAgentID prend la valeur 7
	getPositionFinale	positionFinale déterminée	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	-	EcoAgentID(id=11)
		positionFinale indéterminée	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : NULL)	-	NULL
	setPositionFinale	test sur un EcoAgent quelconque	EcoAgent e (EcoAgentID(id=13), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=11))	positionFinale : EcoAgentID(id=10)	modification de positionFinale : l'identifiant de l'EcoAgentID prend la valeur 10

c) Classe Cube

Classe	Méthode	Test	Donnees		Résultat attendu
Cube	rechercherFuite	test pour un cube qui est gêné	Cube a (EcoAgentID(id=2),etat = RF, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	Gêneur : Cube b (EcoAgentID(id=5), etat = S, positionCourante : EcoAgentID(id=2), positionFinale : EcoAgentID(id=2))	lancement des méthodes : obtenirGeneur(a), qui retourne un pointeur sur le Cube b, puis agresser(b)
		test pour un cube qui peut fuir	Cube a (EcoAgentID(id=2),etat = RF, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	aucun cube n'a pour position courante EcoAgentID(id=2)	lancement des méthodes : obtenirGeneur(a), qui retourne NULL, puis faireFuite()
	rechercher Satisfaction	test pour un cube qui est gêné	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	Gêneur : Cube b (EcoAgentID(id=5), etat = S, positionCourante : EcoAgentID(id=2), positionFinale : EcoAgentID(id=2))	lancement des méthodes : obtenirGeneur(a), qui retourne un pointeur sur le Cube b, puis agresser(b)
		test pour un cube qui peut être satisfait	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	aucun cube n'a pour position courante EcoAgentID(id=2) ou EcoAgentID(id=3)	lancement des méthodes : obtenirGeneur(a), qui retourne NULL, puis faireSatisfaction
	agresser	agression sur un cube quelconque	Agresseur : Cube(EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	Agressé : Cube(EcoAgentID(id=9), etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=6))	le Cube agressé lance la méthode estAgressé()
	estAgressé	Etat du Cube = S	Cube(EcoAgentID(id=15),etat = S, positionCourante : EcoAgentID(id=6), positionFinale : EcoAgentID(id=6))	-	Etat devient RF
		Etat du Cube = RS	Cube(EcoAgentID(id=9),etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=6))	-	Etat devient RF
		Etat du Cube =RF	Cube EcoAgentID(id=2),etat = RF, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	-	Etat devient RF
	faireFuite	fuite d'un cube quelconque	Cube EcoAgentID(id=2),etat = RF, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	-	positionCourante devient la Table
					Etat devient RS
	faireSatisfaction	satisfaction d'un cube quelconque	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=12), positionFinale : EcoAgentID(id=3))	-	positionCourante devient positionFinale
					Etat devient S
	initialiserEtat	position courante != position finale	Cube(EcoAgentID(id=10), positionCourante : EcoAgentID(id=24), positionFinale : EcoAgentID(id=1))	-	Etat devient RS
		position courante = position finale	Cube(EcoAgentID(id=22), positionCourante : EcoAgentID(id=6), positionFinale : EcoAgentID(id=6))	-	Etat devient S

agir	test pour un cube en RF	Cube(EcoAgentID(id=2),etat = RF, positionCourante : EcoAgentID(id=17), positionFinale : EcoAgentID(id=8))	-	lancement de la méthode rechercheFuite()
	test pour un cube en RS	Cube(EcoAgentID(id=9),etat = RS, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=6))	-	lancement de la méthode rechercheSatisfaction()
	test pour un cube satisfait	Cube(EcoAgentID(id=15),etat = S, positionCourante : EcoAgentID(id=6), positionFinale : EcoAgentID(id=6))	-	Aucune action

d) Classe Table

Classe	Méthode	Test	Donnees	Résultat attendu
Table	rechercheFuite	test trivial	Table	Aucune action
	rechercheSatisfaction	test trivial	Table	Aucune action
	agresser	test trivial	Table	Aucune action
	estAgressé	test trivial	Table	Aucune action
	faireFuite	test trivial	Table	Aucune action
	faireSatisfaction	test trivial	Table	Aucune action
	agir	test trivial	Table	Aucune action

e) Classe Singleton

Classe	Méthode	Test	Donnees	Résultat attendu	Principe
template<typename> class Singleton	getInstance	obtention d'une instance unique	-	Les appels de plusieurs instances retournent la même instance unique	On appelle plusieurs fois getInstance() et on vérifie qu'on obtient bien la même adresse en mémoire
	kill	destruction de l'instance NULL	-	Aucune destruction puisqu'aucune instance	On appelle kill sans avoir initialisé d'instance
		destruction d'une instance existante	-	Destruction de l'instance	On crée une instance en faisant un getInstance puis on la détruit

f) Classe PlateformeEcoResolution

Classe	Méthode	Test	Donnees	Résultat attendu	Principe
Plateforme EcoResolution	getEcoAgent	Recherche d'un ecoAgent inexistant	EcoAgentID (identifiant=70)	Pointeur NULL en retour	On essaye d'obtenir un ecoAgent à partir d'un ecoAgentID non referencé
		Recherche d'un ecoAgent existant	EcoAgentID (identifiant=1)	Pointeur sur l'EcoAgent avec un EcoAgentID à 1	-
	addEcoAgent	Ajout d'un eco-agent déjà référencé	EcoAgent (EcoAgentID(identifiant=1))	Exception "ExceptionEcoAgentDejaEnregistre"	-
		Ajout d'un eco-agent non référencé	EcoAgent (EcoAgentID(identifiant=34))	Ajout avec succès de l'agent	-
	addRegle	Ajout d'une regle	Regle	La regle est ajoutée dans list<Regle*> regles	-
	verifierCoherence	Vérification de regles toutes vérifiées	4 x Regles (verifier() -> TRUE)	TRUE	-
		Verification de regles avec une regle non verifié	2 x Regles (verifier() -> TRUE) 2 x Regles (verifier() -> FALSE)	FALSE	
		Verification de regles sans aucune regles	Aucune règle	TRUE	-

g) Classe PlateformeMondeDesCubes

Classe	Méthode	Test	Donnees		Résultat attendu
Plateforme MondeDesCubes	obtenirCubePrioritaire	obtention du cube prioritaire tous les cubes en RS	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID (id=IDTABLE), positionFinale : EcoAgentID (id=3))	Cube b (EcoAgentID(id=3), etat = RS, positionCourante : EcoAgentID (id=2), positionFinale : EcoAgentID (id=IDTABLE))	Cube b
		obtention du cube prioritaire: mélange de RF et RS	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID (id=IDTABLE), positionFinale : EcoAgentID (id=3))	Cube b (EcoAgentID(id=3), etat = RS, positionCourante : EcoAgentID(id=2), positionFinale : EcoAgentID (id=IDTABLE))	Cube c
			Cube c (EcoAgentID(id=4),etat = RF, positionCourante : EcoAgentID(id=3), positionFinale : EcoAgentID(id=IDTABLE))	-	
		obtention du cube prioritaire alors que tous les cubes sont satisfaits	Cube a (EcoAgentID(id=2),etat = S, positionCourante : EcoAgentID(id=IDTABLE), positionFinale : EcoAgentID(id=IDTABLE))	Cube b (EcoAgentID(id=5), etat = S, positionCourante : EcoAgentID (id=12), positionFinale : EcoAgentID (id=12))	NULL
	setNombreDeCubes	insertion d'un nombre de cubes valide	entier n compris supérieur ou égal à 1	-	nombreDeCubes <- n
		insertion d'un nombre de cubes invalide	n = -5	-	nombreDeCubes <- NULL
	getNombreDeCubes	obtention du nombre de cubes	nombreDeCubes=12	-	12
	setTableIdentifiant	identifiant table valide	EcoAgentID(id=5)	-	Table table (EcoAgentID(id=5))
		identifiant table invalide	EcoAgentID(id=-45)	-	NULL
	getTableID	ID existant	Table table (EcoAgentID(id=IDTABLE))	-	id=IDTABLE
		ID non existant	Table table (EcoAgentID(id=NULL))	-	NULL
	verifierNombreDeCubes	Nombre de cubes valide	n = 3	-	VRAI
		Nombre de cubes invalide	n = -15	-	FAUX
	obtenirGeneurs	obtention d'un cube gêneur pour un cube existant	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=IDTABLE), positionFinale : EcoAgentID(id=4))	Cube b (EcoAgentID(id=3),etat = S, positionCourante : EcoAgentID(id=2), positionFinale : EcoAgentID(id=2))	pointeur sur le cube gêneur
		obtention d'un cube gêneur pour un cube non existant	Aucun cube	-	NULL
		obtention d'aucun cube gêneur pour un cube existant	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=IDTABLE), positionFinale : EcoAgentID(id=4))	Aucun cube ayant positionCourante : EcoAgentID(id=2)	NULL

Classe	Méthode	Test	Données		Résultat attendu
PlateformeMondeDesCubes	obtenirGeneurs	obtention d'un cube gêneur pour un cube existant	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=IDTABLE), positionFinale : EcoAgentID(id=4))	Cube b (EcoAgentID(id=3),etat = S, positionCourante : EcoAgentID(id=2), positionFinale : EcoAgentID(id=2))	pointeur sur le cube gêneur
		obtention d'un cube gêneur pour un cube non existant	Aucun cube	-	NULL
		obtention d'aucun cube gêneur pour un cube existant	Cube a (EcoAgentID(id=2),etat = RS, positionCourante : EcoAgentID(id=IDTABLE), positionFinale : EcoAgentID(id=4))	Aucun cube ayant positionCourante : EcoAgentID(id=2)	NULL
	addEcoAgent(Cube)	Ajout d'un cube déjà référencé	Cube a (EcoAgentID(identifiant=1))	-	Exception "ExceptionCubeDejaEnregistre"
		Ajout d'un cube non référencé	Cube a (EcoAgentID(identifiant=34))	-	Ajout avec succès du cube
	getCubes	ID déterminés	Un ou plusieurs Cubes dont les EcoAgentID sont déterminés	-	Map des cubes
		ID non déterminés	Un ou plusieurs Cubes avec au moins un EcoAgentID non déterminé	-	NULL
	setPositionFinale(EcoAgentID,EcoAgentID)	ID cube inexistant ID position finale déterminée	NULL	EcoAgentID(id=3)	Aucune action
		ID cube inexistant ID position finale non déterminée	NULL	NULL	Aucune action
		ID cube existant ID position finale déterminée	EcoAgentID(id=2)	EcoAgentID(id=3)	La position finale du cube d'EcoAgentID(id=2) devient EcoAgentID(id=3)
		ID cube existant ID position finale non déterminée	EcoAgentID(id=2)	NULL	Aucune action
	setPositionCourante(EcoAgentID,EcoAgentID)	ID cube inexistant ID position courante déterminée	NULL	EcoAgentID(id=3)	Aucune action
		ID cube inexistant ID position courante non déterminée	NULL	NULL	Aucune action
		ID cube existant ID position courante déterminée	EcoAgentID(id=2)	EcoAgentID(id=3)	La position courante du cube d'EcoAgentID(id=2) devient EcoAgentID(id=3)
		ID cube existant ID position courante non déterminée	EcoAgentID(id=2)	NULL	Aucune action
	setCubeID(Cube,EcoAgentID)	Cube existant, EcoAgentID invalide	Cube a (EcoAgentID(id=4),etat = NULL, positionCourante : NULL, positionFinale : NULL)	NULL	Aucune action
		Cube existant, EcoAgentID valide	Cube a (EcoAgentID(id=NULL),etat = NULL, positionCourante : NULL, positionFinale : NULL)	EcoAgentID(id=3)	Cube a (EcoAgentID(id=3),etat = NULL, positionCourante : NULL, positionFinale : NULL)
		Cube non existant, EcoAgentID invalide	NULL	NULL	NULL
		Cube non existant, EcoAgentID valide	NULL	EcoAgentID(id=IDENTIFIANT)	NULL

Conclusion

blabla