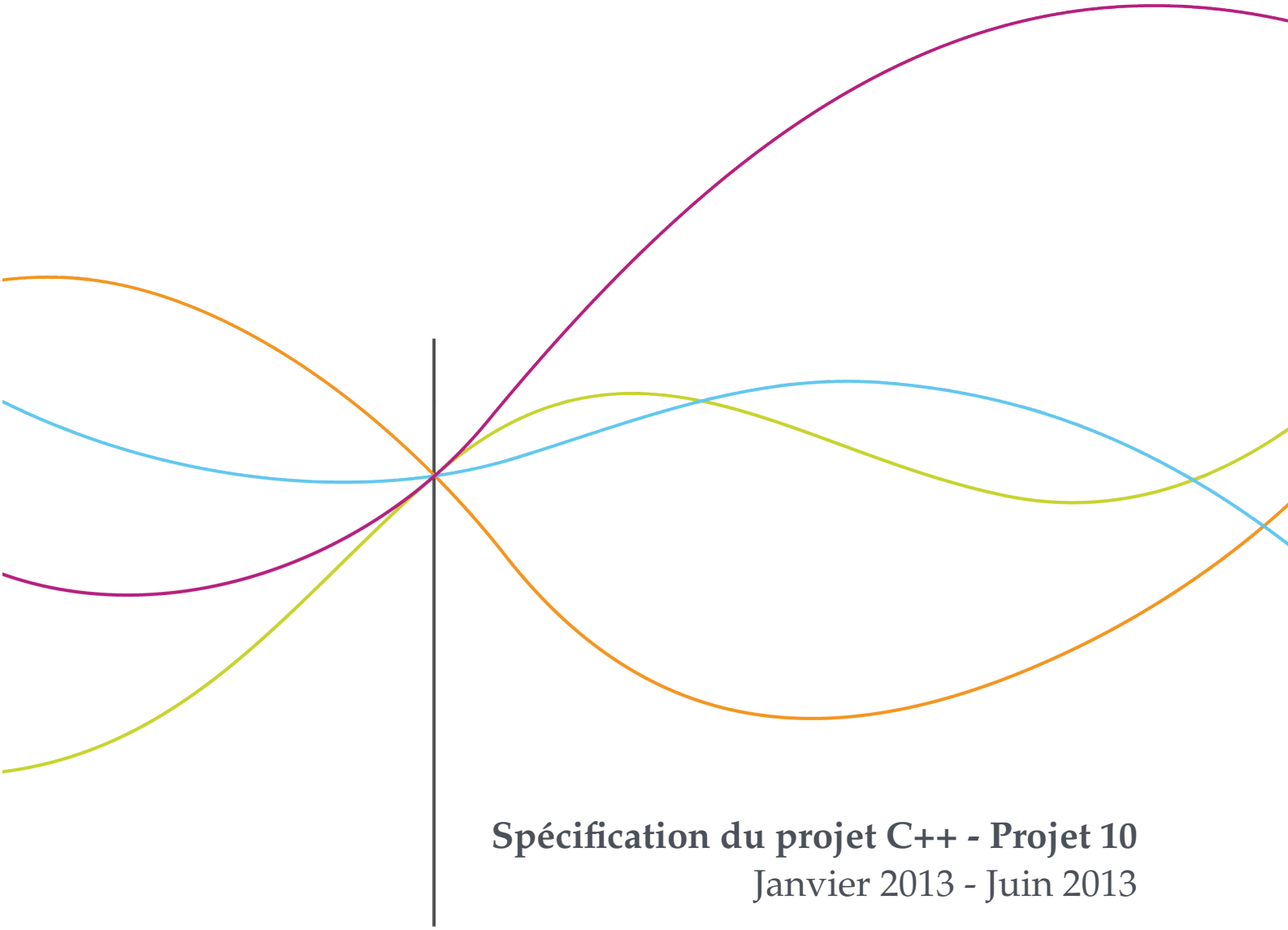
INSA ROUEN
À taille humaine,
à l'échelle du monde

Département Génie Mathématique

Affectation spatiale sous contraintes - Spécification fonctionnelle



Spécification du projet C++ - Projet 10

Janvier 2013 - Juin 2013

Adeline Bailly & Alexandre Quemy

Table des matières

Chapitre 1

Framework

1.1 Identification des briques logicielles

1.1.1 Module d'IA et d'environnement

Dans ce module, on distingue plusieurs entités :

- L'environnement, espace dans lequel évolue les objets. En pratique, généralement $\mathbb{R}^2$ ou $\mathbb{R}^3$.
- Les objets, étant de plusieurs types :
  1. Les ressources, que l'on peut affecter à une tâche et qui sont donc mobiles dans l'environnement.
  2. Les tâches, fixes dans l'environnement et qui permettent de modifier un ou plusieurs objectifs.
  3. Les obstacles, fixes dans l'environnement et qui modélisent la structure de l'espace.
- L'IA, qui va selon une ou plusieurs stratégies observer l'environnement et prendre des décisions pour résoudre le problème d'affectation.
- Les contraintes et objectifs, qui permettent de définir des seuils (minimal, maximal, proportion) par rapport à un type de tâche.
- Les stratégies, qui définissent un déroulement du processus d'affectation (fonction de coût, algorithmes utilisés)
- Les modèles d'apprentissage et de prise de décision (non étudié ici).

1.1.2 Module d'indexation spatiale et de partitionnement spatial

Les entités de ce module sont :

- Les arbres, servant à stocker l'information spatiale et y accéder rapidemment.
- Les techniques de construction de ces arbres. Il s'agit principalement d'algorithmes qui sont plus spécifiques à chaque structure de données et qui peuvent les améliorer. On peut citer par exemple le Z-order qui permet de construire de manière efficace des Quadtree ou des arbres de Hilbert.

1.1.3 Module d'algorithmes de plus court chemin

Les entités de ce module sont simplement les algorithmes travaillant sur l'environnement transformé grâce aux techniques d'indexation et de partitionnement spatial.

1.1.4 Module d'affectation

Les entités de ce module sont simplement les algorithmes qui doivent permettre la résolution de l'affectation.

1.2 Cas d'utilisation

1.2.1 Environnement et IA

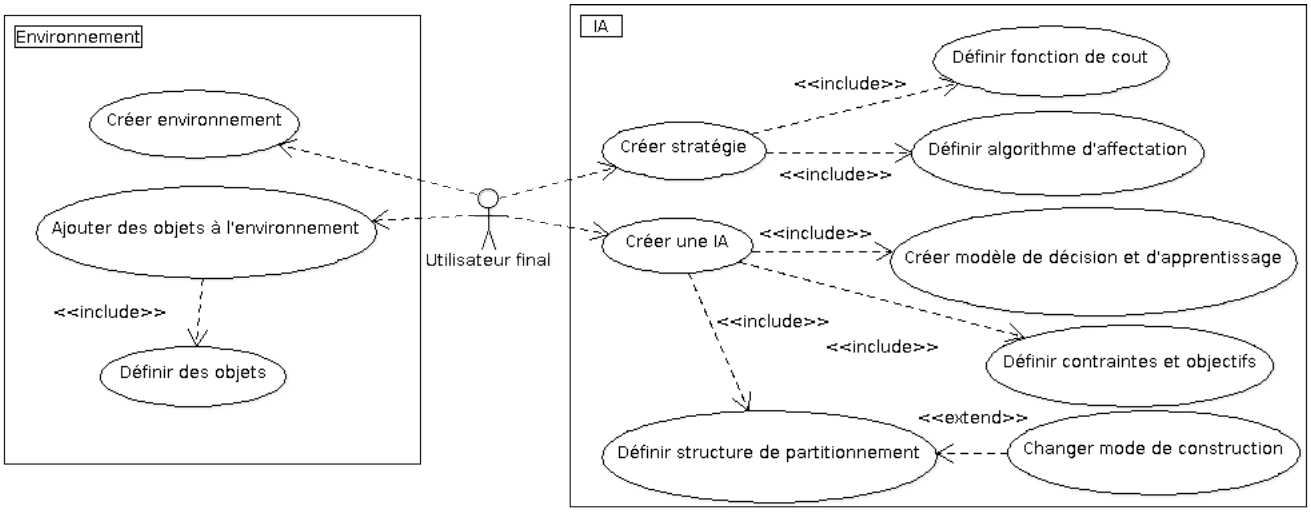


FIGURE 1.1 – Cas d'utilisation généraux

1.2.2 Ressources

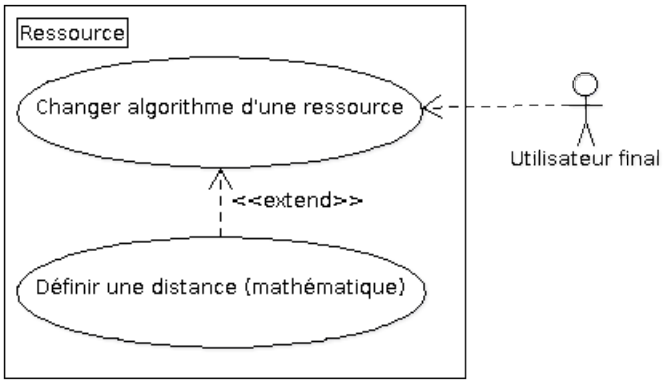


FIGURE 1.2 – Cas d'utilisation liés aux ressources

1.2.3 Framework

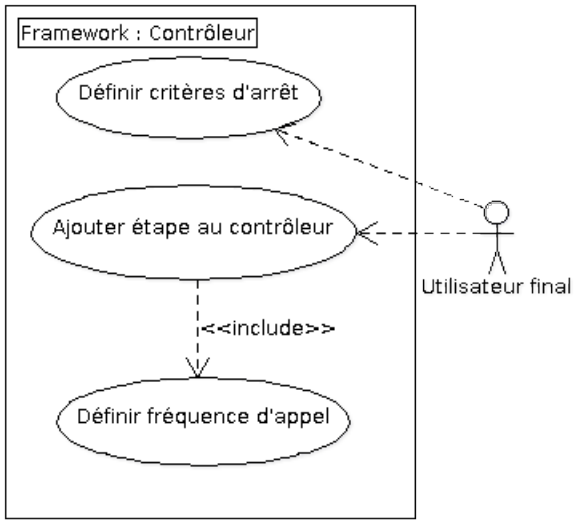


FIGURE 1.3 – Cas d'utilisation liés au framework

1.3 Modélisation des entités

1.3.1 Tâche

Une tâche est définie par une représentation numérique. Ne doit pas être confondue avec l'emplacement de travail (TaskSpot) qui est un objet concret de l'environnement qui peut modifier la tâche.

1.3.2 Contraintes

Une contrainte est une assertion et sera modélisée sous forme de foncteur renvoyant vrai ou faux. Il existe plusieurs types de contraintes ayant plusieurs fonctionnements. Chacune possède une priorité qui pourra représenter soit un ordre de traitement par l'IA soit une pénalité lors de l'évaluation.

Note : Il serait intéressant de pouvoirs détecter les contraintes impossibles à tenir en fonction de l'environnement ou tout simplement parce que deux contraintes s'opposent. Le problème de satisfaction de contraintes est à lui tout seul un problème NP-difficile dans la plupart des cas.

On supposera durant ce projet que ce point ne pose pas de soucis et que l'utilisateur n'entrera pas des contraintes qui pourront amener le système à se bloquer ou à ne pouvoir satisfaire l'ensemble des contraintes dans un temps raisonnable.

Par la suite il pourra néanmoins être intéressant de proposer des algorithmes de déduction de contraintes et de résolution du problème CSP (Constraint satisfaction problem).

Contraintes de seuils

Une contrainte de seuil (appliquée à une tâche) est composée d'un opérateur $\leq$ ou $\geq$, et d'une valeur numérique.

Ces contraintes possèdent deux fonctions de callback qui doivent se déclencher lorsque l'on franchi le seuil défini, d'un côté et de l'autre (ie, cela implique de garder en mémoire la

valeur lors de la vérification précédente).

Optionnellement, on peut définir un intervalle de tolérance non nécessairement symétrique.

Contraintes de de proportionnalité

Défini une relation de proportionnalité entre 2 ou plusieurs tâches. On peut imaginer par exemple vouloir 30 % de pierres contre 70 % d'or dans le cadre d'un jeu de statégie.

Là encore, on peut définir de manière optionnelle un intervalle de tolérance et deux fonctions de rappel qui seront appelées en cas de changement de statut de la contrainte (passage du respect au non respect et inversement).

Contraintes personnalisées

Une contrainte personnalisée est une contrainte modélisée par une fonction renvoyant un booléen : vrai si la condition est vérifiée, faux sinon. Elles servent à exprimer des contraintes diverses sur les tâches mais aussi sur d'autres éléments de l'environnement en fonction du problème de l'utilisateur.

Ces contraintes portent également deux fonctions de rappel pour le changement de statut de la contrainte.

1.3.3 Système de contraintes

Il s'agit d'un système comprenant les tâches ainsi que les contraintes. Les tâches / contraintes sont potentiellement indépendantes de l'IA qui va résoudre le problème puisqu'elles appartiennent au problème lui même.

Ainsi le but du système de contraintes est de centraliser les tâches et les contraintes pour en faciliter l'accès par l'utilisateur et l'IA.

Le système pourra également proposer des méthodes de résolution du CSP évoquées plus haut.

Note : La description des contraintes peut se faire de manière plus générale et plus souple par l'utilisateur grâce à un design pattern interpréteur (que nous n'implémenterons pas dans l'immédiat).

1.3.4 L'environnement

Il est composé d'un espace au sens mathématique, ainsi que de 3 types d'objets : les obstacles, les points de travail et les ressources. Il a donc principalement un rôle de conteneur.

Étant donné que la création de l'environnement peut très vite devenir complexe, un design pattern builder est envisagé, permettant par exemple de charger un fichier d'environnement depuis différents types de fichiers. Ce sera d'ailleurs le cas pour l'application d'exemple.

L'environnement est à la fois un observateur et un observable. Il est observé par l'IA et observe les objets dynamiques qu'il contient. Ainsi il s'agit d'un médiateur entre les objets qu'il contient et l'IA.

1.3.5 L'espace

L'espace est simplement défini par un nombre de dimensions et le type de ses coordonnées. Il contient également les limites de l'espace. L'espace peut donc être imaginé comme un parallélotope droit.

1.3.6 Les coordonnées

Les coordonnées permettent de localiser l'ensemble des objets dans l'espace. Elles présentent deux caractéristiques : le nombre de composantes et le type de ces composantes. Il s'agit d'un vecteur stockant n composantes d'un type numérique.

1.3.7 Les objets

Les objets évoluent dans l'environnement et plus particulièrement dans l'espace qu'il contient. Ainsi, tous les objets possèdent des coordonnées qui doivent être du même type que ceux de l'espace considéré.

Il existe deux types d'objets (pour le moment) : les objets statiques et les objets dynamiques. Chacun de ces deux types donnent lieu à une hiérarchie d'objets.

Tous les objets implémentent une géométrie qui permettra d'effectuer des calculs de collision selon diverses techniques. Par exemple on peut définir un objet par un polygone convexe, un polygone quelconque, un carré, un cercle. Chaque représentation à ses avantages et inconvénients sur les méthodes de collisions : algorithme du point dans un polygone (convexe), Bounding-Box, rayon de collision, etc.

Tous les objets possèdent également une fonction update qui sert à mettre à jour leur état interne régulièrement.

Les objets statiques

Il s'agit des objets de l'environnement qui ne peuvent pas bouger. Leur intérêt principal est de définir des obstacles dans l'espace, bloquant le passage aux ressources.

↪ **Les obstacles**

Il s'agit du seul objet statique concret dont nous avons perçu l'intérêt pour l'instant. Il est là pour modéliser, comme son nom l'indique, un obstacle dans l'espace.

Les objets dynamiques

Ce sont des objets qui sont capables d'évoluer.

↪ **Les ressources**

Elles représentent des objets dont l'IA dispose pour pouvoir les affecter à diverses tâches en vu de satisfaire les contraintes du système de contraintes. Il s'agit encore d'une classe abstraite dont le but est de clarifier l'architecture des objets. On retrouvera différents types de ressources ayant des comportements différents.

Ainsi on peut imaginer une ressource qui n'est pas réaffectable lorsqu'elle commence une tâche qui lui est donnée par l'IA, on peut imaginer une ressource qui n'accepte qu'un seul type de tâche, etc. Toutes ses variantes dépendent du problème et seront rajoutées à mesure que le besoin s'en fera sentir.
Dans notre cas on n'envisagera qu'une unité simple qui pourra être réaffectée n'importe quand, et une unité qui ne pourra pas être réaffectée.

Chaque ressource est un agent au sens où il est partiellement autonome. Il évolue indépendamment de l'IA qui ne fait que lui communiquer un ordre (une affectation). C'est la ressource qui se déplace de son propre chef pour se déplacer vers l'objectif. Ainsi chaque ressource possède un algorithme de plus court chemin ainsi qu'une vitesse de déplacement.

Comme l'intérêt du partionnement spatial et de l'indexation astucieuse des objets réside dans la diminution du temps de calcul des algorithmes en général, cela implique que les ressources doivent accéder aux structures de partitionnement et d'indexation. Ainsi ce n'est pas l'IA qui s'occupe de stocker ces structures (on peut imaginer dans un jeu vidéo que l'IA soit remplacé par un joueur).

↪ **Les emplacements de travail**

Ce sont les représentations physiques d'une tâche. C'est à dire des objets qui vont pouvoir modifier une tâche (son « compteur ») en fonction de certaines conditions.
Ainsi, ce sont des observateurs d'une tâche particulière qu'elle notifie en cas de changement.

Là encore, on peut imaginer beaucoup de variantes : des emplacements temporaires – l'emplacement est détruit au bout de tant de temps d'utilisation, des emplacements avec un compteur – l'emplacement se détruit après tant d'utilisation, etc.

1.3.8 Les politiques de chevauchement

Les objets sont paramétrés par une politique de chevauchement : certains objets peuvent ne pas en chevaucher d'autres.
Il s'agit concrètement d'un entier qui permet de définir une priorité sur le chevauchement. Un objet ne peux pas surpasser un objet ayant une valeur plus haute que la sienne.
Des constantes sont prévues pour définir des objets « fantômes » et des objets que l'on ne peut pas chevaucher, comme les obstacles par exemple.

1.3.9 Les fonctions de coûts

Il existe plusieurs types de fonctions de coûts dont le rôle est similaire : évaluer un objet ou une quantité en fonction de différents paramètres. Classiquement, on peut évaluer la satisfaction d'une contrainte, les paires (ressource / emplacement de travail), la situation

globale.
La manière dont l'évaluation est faite dépend de la stratégie adoptée.

L'implémentation est très libre et souple au vu de ce que le C++ permet de faire, surtout avec le nouveau standard.

1.3.10 Les stratégies

Il s'agit d'une manière de résoudre le problème d'affectation pour satisfaire les contraintes. Elle comporte une manière d'évaluer la situation et un algorithme d'affectation.

Une stratégie très simple consiste à ne traiter la satisfaction que d'une contrainte à la fois. Il s'agira de la seule stratégie que nous implémenterons.

L'évaluation de la situation est faite en fonction de la stratégie. Voici la stratégie que nous implémenterons (visible également sur le diagramme de séquence du contrôleur de l'IA) :

- Evaluation des contraintes
- Evaluation des paires ressource / emplacement
- Evaluation globale des ressources
- Evaluation de la situation

Voici un exemple concret des fonctions de coûts associés : La contrainte x non satisfaite : cette contrainte portait sur la tâche y dont la valeur est trop faible.

- Evaluation de l'intérêt des taskSpots
  # Le coût des taskSpots rattachés à y est 1, sinon 0.
- Evaluation d'un individu
  # Distance des couples (Indi , TaskSpot)
- Evaluation globale d'individu
  # evalIndi * evalTaskSpot
- Evaluation situation
  # Somme des évaluations globales des individus

Dans cet exemple très simple on ne veut affecter des ressources qu'aux emplacements susceptibles de faire augmenter la quantité non satisfaite. Ainsi dans l'algorithme d'affectation, les couples (individu, emplacement) où l'emplacement n'est pas rattaché à cette quantité seront nuls et donc ne pourront être choisi.

L'évaluation de la situation est un indicateur pour l'IA afin qu'elle puisse décider si elle doit changer de stratégie ou non.

1.3.11 L'IA

Il s'agit d'un conteneur de stratégies qui sont organisées selon un modèle (couple d'un modèle de décision et d'observation), pour pouvoir plus facilement traiter le problème d'affectation.

1.3.12 Arbres

Utilisés pour indexer les objets, pour partitionner l'espace. On retrouve beaucoup d'arbres différents ayant chacun leurs avantages et désavantages. En voici une liste assez complète qui pourront être implémentés par la suite :

- AtlasGrid
- R-tree
- Quadtree
- Octree
- BSP
- KD-tree
- implicit KD-tree
- vantage KD-tree
- R+-Tree
- R-Tree
- X-Tree
- M-Tree
- Hilbert R-Tree
- UB-Tree

1.3.13 Données spatiales

Etant donné que plusieurs objets qui ne se connaissent pas doivent accéder aux structures de données spatiales (que ce soit la partition de l'espace ou l'index des objets dynamiques), il nous faut une classe dédié à cela.
Cet objet possède 3 structures de données qui indexent la structure de l'espace (donc les objets statiques), les tâches et les ressources à partir des conteneurs basiques (des tableaux) de l'environnement.

Lors de l'initialisation, le contrôleur créera cet objet automatiquement et passera une référence aux ressources (en passant par l'environnement) et à l'IA.

1.3.14 Les algorithmes

Chaque type d'algorithme définit une hiérarchie de classes héritant toutes d'une classe abstraite définissant l'interface de ce type d'algorithme.
On utilisera un design pattern Factory pour permettre une instanciation aisée pour l'utilisateur.

Algorithmes de plus court chemin

Les algorithmes de plus court chemin renvoient une liste de noeuds à parcourir pour une ressource pour arriver à son objectif. Il se peut qu'il y ait des déplacements à prévoir au sein d'un noeud mais c'est à la ressource de s'en occuper.

Algorithmes d'affectation

Les algorithmes d'affectation prennent en entrée les emplacements de travail

Etant donné que les algorithmes ne travaillent pas nécessairement sur le même type de données (par exemple la méthode hongroise travaille sur une matrice alors que la recherche du plus proche voisin peut faire appel à des structures d'arbre, etc.), une étape de transformation des données peut avoir lieu.

1.4 Architecture du framework

Cette section s'attele à décrire l'architecture globale du framework :

1.4.1 Contrôleur

Le contrôleur principal gère le déroulement globale de la simulation. Il s'agit d'une boucle principale avec des critères d'arrêt définis par l'utilisateur (nous proposerons un critère temporel).

La boucle principale va effectuer en permanence les même actions, dans le même ordre. Ces actions sont des appels à des contrôleurs locaux ou des actions définis par l'utilisateur parce qu'il en a besoin dans son application : interception et traitement des entrées claviers, affichage ...

Par défaut, seules deux actions sont présentes : l'appel au contrôleur local de l'IA et au contrôleur local de l'environnement.

Ces actions seront modélisées sous forme d'une liste de fonctions à appeler dans l'ordre. Chaque fonction sera assortie d'un délais d'activation : toutes les 100ms par exemple.

1.4.2 Logger

Un logger est absolument nécessaire pour un framework afin de repérer le déroulement du flux d'exécution et de localiser d'éventuel(s) problème(s) ou simplement visualiser le processus de calcul. C'est d'ailleurs très utile pour faire des tests unitaires ou d'intégration.

Nous mettrons en place un logger sous forme de Design Pattern Chaine de Responsabilité. Chaque maillon de la chaîne sera un logger avec un niveau de log différent (DEBUG, INFO, ERROR ...) et le message transite de maillon en maillon jusqu'à ce que le bon niveau soit atteint.

On pourra utiliser une sérialisation dans un fichier texte ou non.

Il existera une instance statique du logger pour permettre une utilisation globale. On n'utilisera pas de DP Singleton d'une part parce qu'il est très dur à rendre thread-safe et d'autre part parce pour une raison ou une autre, l'utilisateur final trouvera peut être une utilité à avoir plusieurs loggers.

1.4.3 Le modèle multi-thread

Nous n'implémenterons pas le modèle multithread pour des raisons de temps mais nous le rendrons facilement intégrable, notamment avec la fonction du contrôleur principal. En effet, les actions du contrôleur sont, normalement, toutes indépendantes et peuvent donc être lancées en parallèle.

Pour rendre l'application thread-safe à un niveau de granularité plus fin, on utilisera le paramétrage par politique qui permettra de découpler totalement le modèle multi-thread du reste et d'activer / désactiver le support multi-thread à la volée.

1.4.4 Les observateurs / observables

Le design pattern Observateur occupe une place importante notamment lors de la récupération d'information sur l'environnement par l'IA mais également par la notification de mise à jour de plus petites entités au sein de l'environnement (les objets). C'est pourquoi nous avons imaginés deux types de politiques liées à l'observation : une politique active et une politique passive.

Politique active

La politique active est le comportement classique lorsque l'on parle du patron Observateur. Dès qu'un changement apparaît sur un observé, il notifie l'observateur qui s'est abonné aux observés.
On retrouve ce comportement au niveau du couple Emplacement / Tâche où l'emplacement, sous certaines conditions, va notifier la tâche qui devra se mettre à jour.

Politique passive

La politique passive consiste pour l'observé à simplement changer son état indiquant qu'il a changé et préparer le message qui doit être transmis à l'observateur. C'est donc l'observateur qui est actif et va explicitement demander la notification d'état et récuperer les messages des observables qui ont indiqués qu'ils étaient modifiés.

Ce comportement apparaît par exemple au niveau de l'observation de l'environnement par l'IA. L'IA va périodiquement choisir de regarder l'état de l'environnement.

1.5 Diagramme de classes

1.5.1 Algorithmes

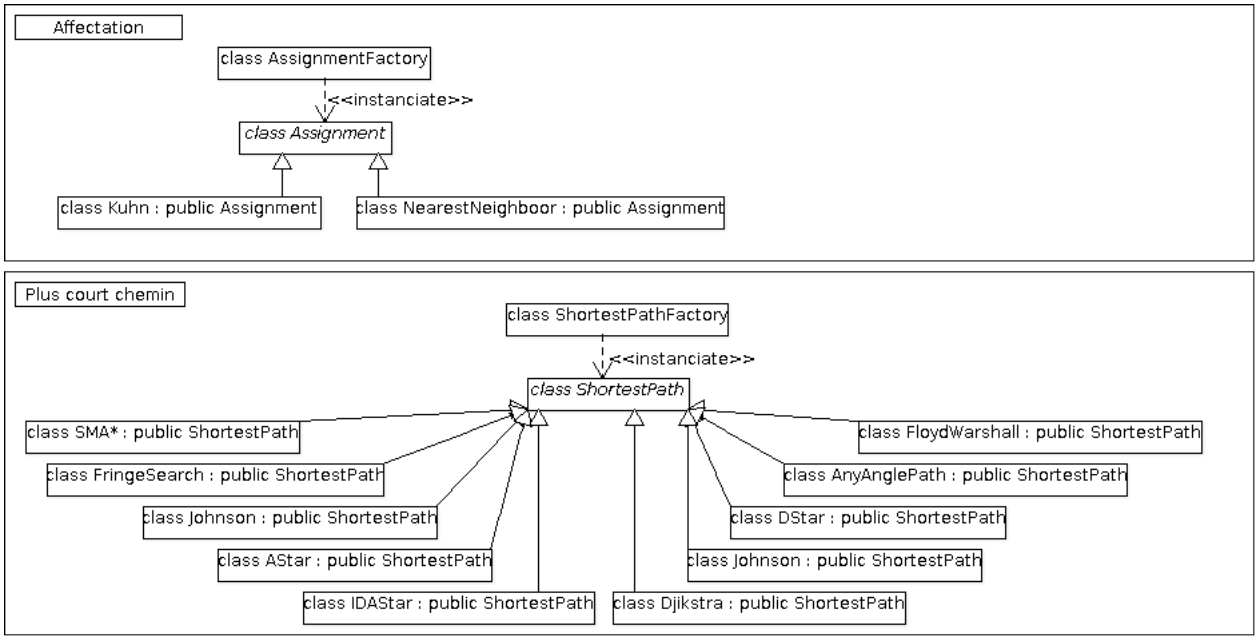


FIGURE 1.4 – Diagramme de classes des algorithmes

Pour faciliter l'instanciation des algorithmes de plus court chemin et d'affectation, un Design Pattern Factory est prévu. Il instanciera un objet dérivant de la classe abstraite Assigment ou ShortestPath qui représentent l'interface de manipulation des algorithmes.

1.5.2 Arbres

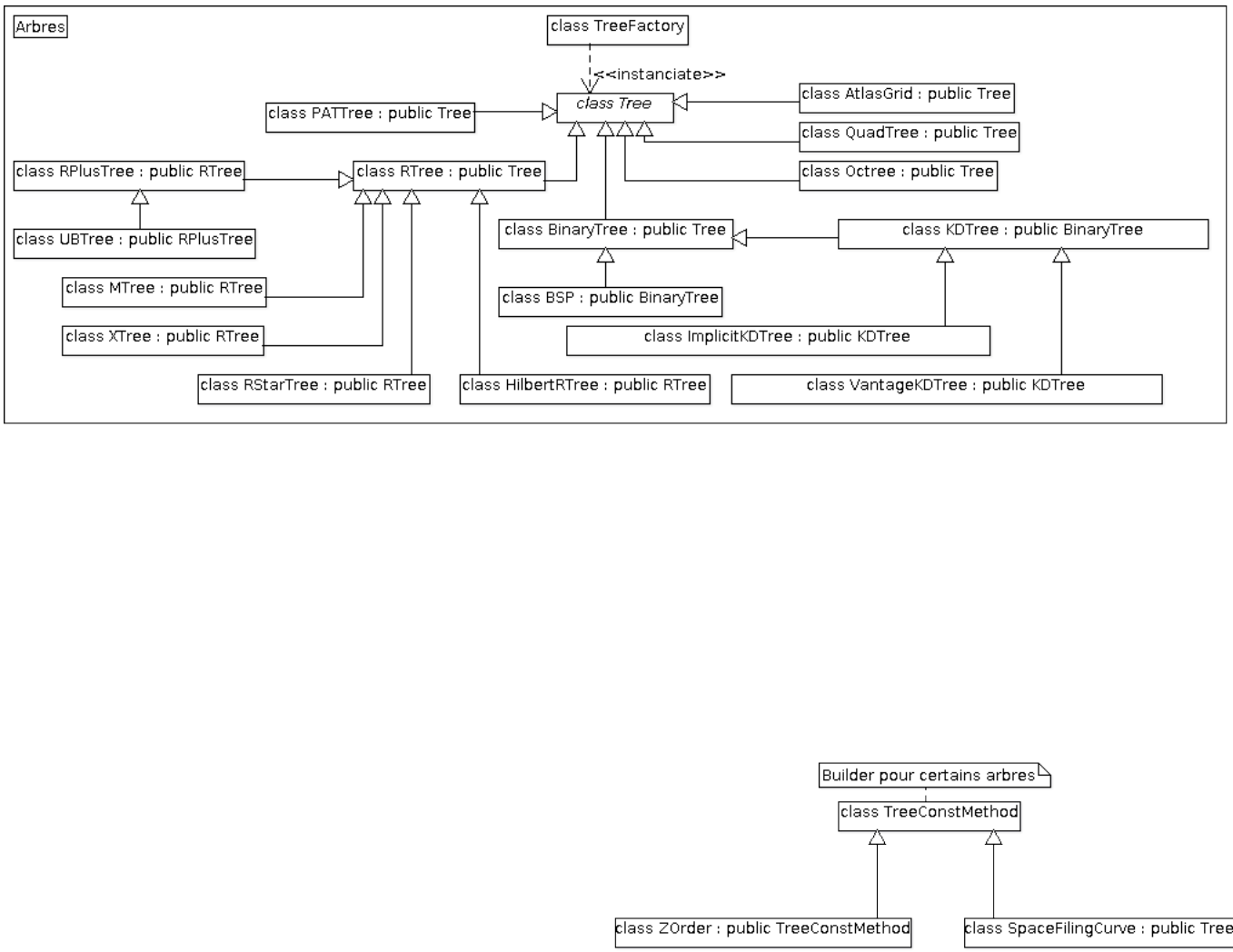


FIGURE 1.5 – Diagramme de classes des arbres

De la même manière qu'avec les algorithmes, une Factory permettra d'instancier un arbre concret.
Certains arbres peuvent se constuire de plusieurs manières, c'est pourquoi on prévoira un Design Pattern Builder pour permettre de monter différemment ces arbres.

1.5.3 Framework

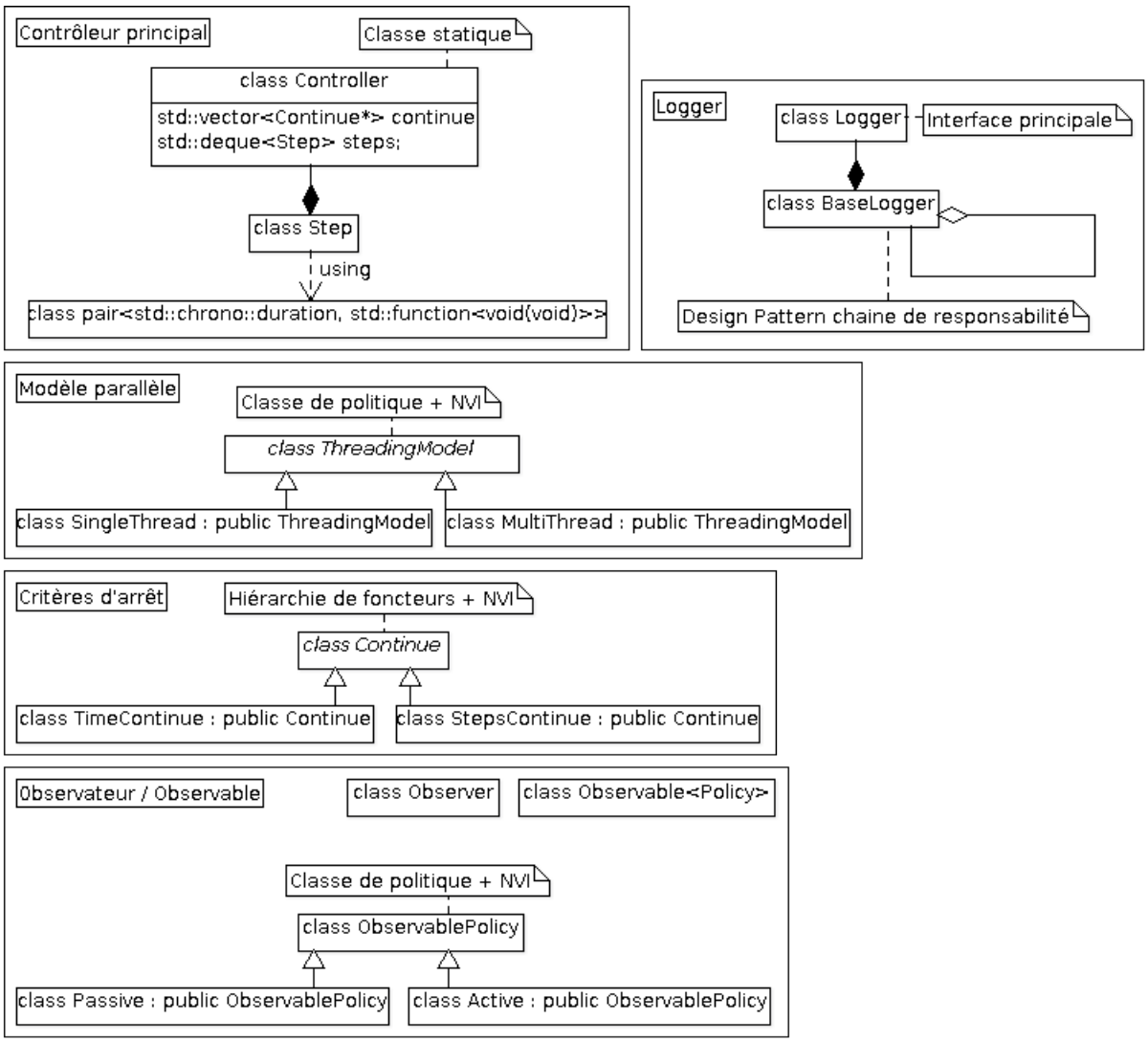


FIGURE 1.6 – Diagramme de classes du framework

1.5.4 IA & Contraintes

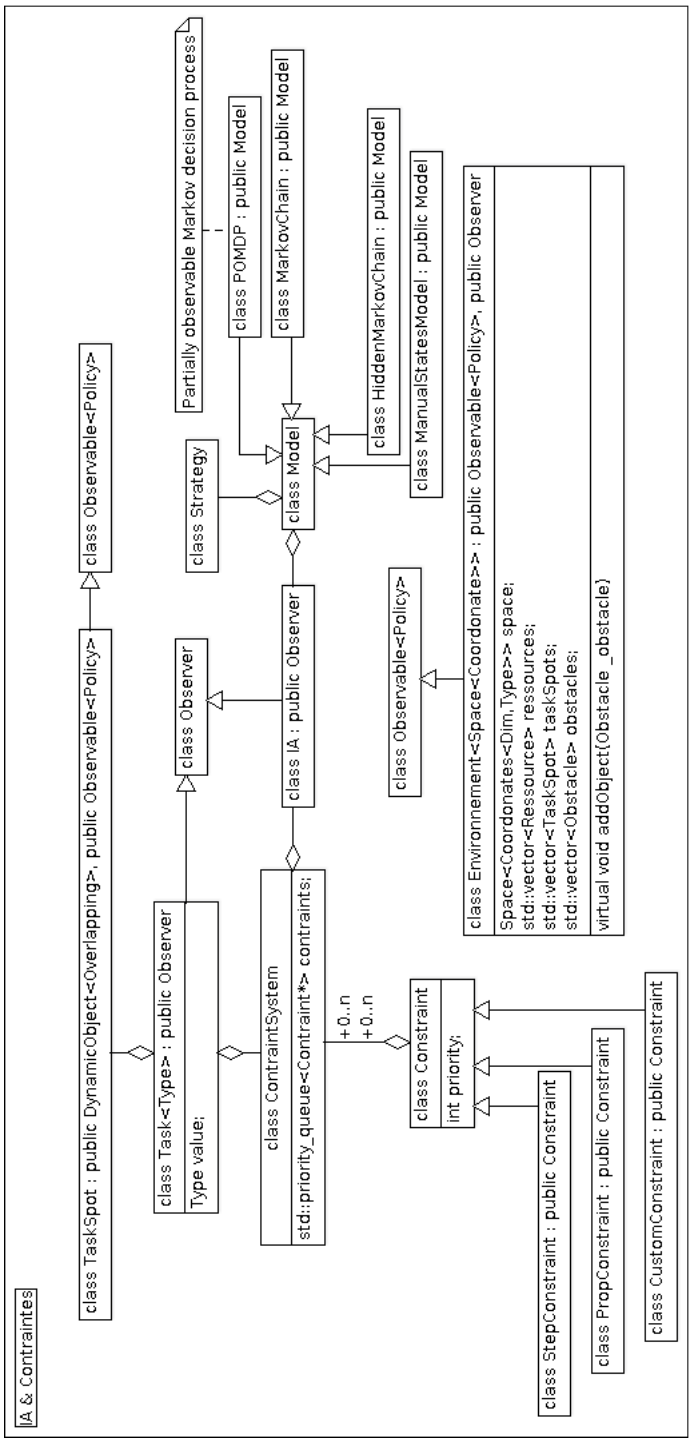


FIGURE 1.7 – Diagramme de classes d'IA et contraintes

1.5.5 Environnement

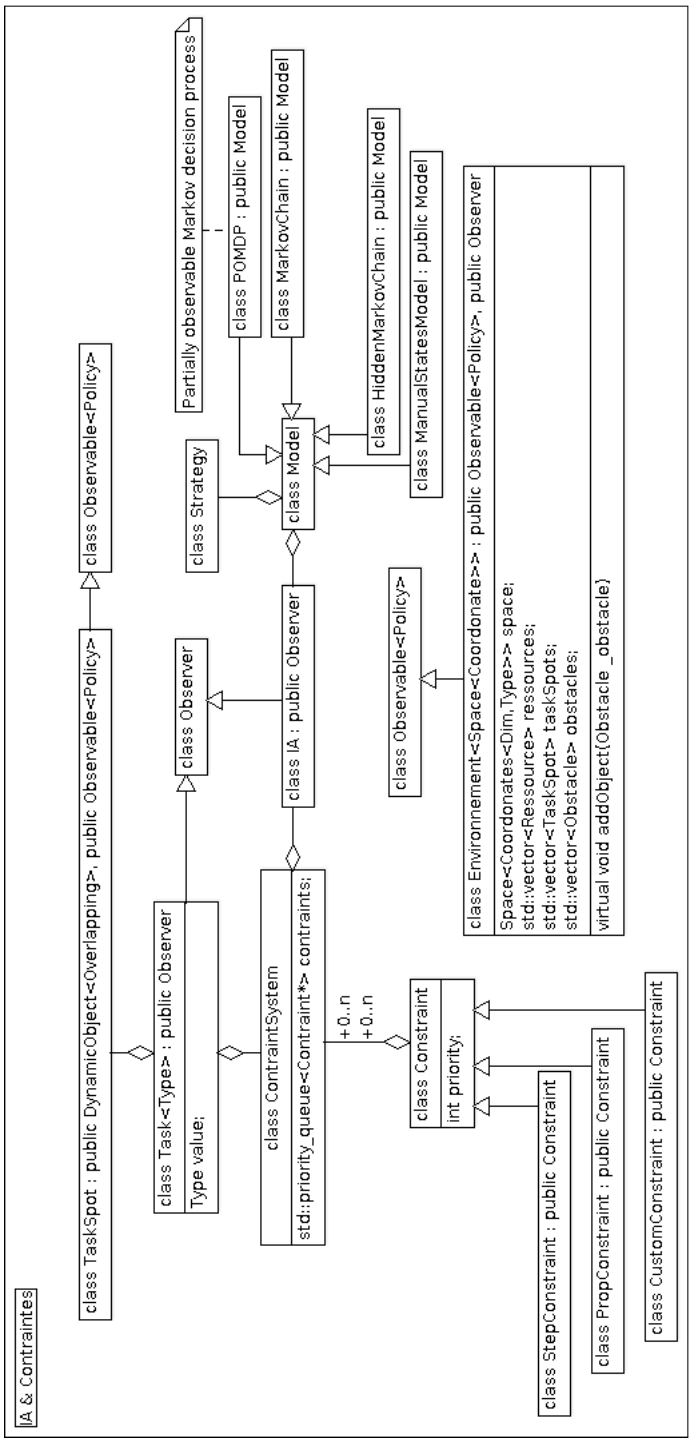


FIGURE 1.8 – Diagramme de classes d'IA et contraintes

1.6 Scénario et diagrammes de séquences

On peut distinguer plusieurs scénarios pour notre application. Le scénario utilisateur où l'acteur principal est l'utilisateur final, c'est à dire celui qui va utiliser le framework pour résoudre un problème en fonction de son contexte.

Des scénarios annexes existent, relatifs aux sous-systèmes du framework et à l'interaction entre ses composants. Le principal est celui qui intervient durant toute la simulation / résolution et dont l'acteur principal est le contrôleur du framework.

1.6.1 Scénario principal

- L'utilisateur définit ses tâches
  ↪ Définition des contraintes
- L'utilisateur définit l'environnement
  ↪ Définition de l'espace (dimensions, frontières)
  ↪ Définition des objets statiques (obstacles)
  ↪ Définition des objets dynamiques
    - Définition des ressources
      - Définition d'une fonction de coût
    - Définition des emplacements de travail
      - Observation de l'emplacement par un type de tâche

  ↪ Paramétrage de l'environnement
    - Délais de mise à jour
- Définition de l'algorithme de partitionnement de l'espace
- Définition de l'algorithme d'indexation des objets dynamiques
- Création des stratégies
  ↪ Définition des fonctions d'évaluation
  ↪ Définition de l'algorithme d'affectation
- Création de l'IA
- Définition du modèle de décision et d'observation
- Ajout des stratégies
- Paramétrage de l'IA
  ↪ Délais entre chaque cycle « observation – apprentissage – prise de décision - affectation »
- Lancement de la simulation / résolution par le contrôleur du framework.

1.6.2 Scénarios annexes

Initialisation

L'initialisation est la première étape effectuée lors de l'exécution. Elle se déroule comme suit :

- Création de l'objet contenant les données spatiales
- Partionnement de l'espace
- Indexation :
  ↪ Des ressources
  ↪ Des emplacements de travail
  ↪ Des obstacles
- Donne un accès en lecture aux données spatiales

(Voir Diagramme de séquence d'initialisation 1.9, page 19).

Contrôleur principal

- Le contrôleur principal appelle le contrôleur de l'IA si la contrainte de temps est respectée
- Le contrôleur de l'IA :
  ↪ Observation du monde par la stratégie courante
    - Récupération des changements
    - Evaluation de la situation
  ↪ Apprentissage (non étudié ici)
  ↪ Prise de décision (non étudié ici - peut être effectué manuellement par l'utilisateur)
  ↪ Affectation par la stratégie courante
  ↪ Notification aux ressources des nouvelles affectations
- Le contrôleur principal appelle le contrôleur de l'environnement si la contrainte de temps est respectée
- Le contrôleur de l'environnement :
  ↪ Mise à jour des ressources
    - Si pas d'affectation : ne rien faire
    - Si affectation : calcul du chemin le plus court
    - Si chemin connu : déplacement et calcul de collisions
    - ...
- La boucle recommence tant que les conditions sont respectées

(Voir Diagramme de séquence de simulation 1.10, page 20).

Contrôleur IA

Contrôleur Environnement

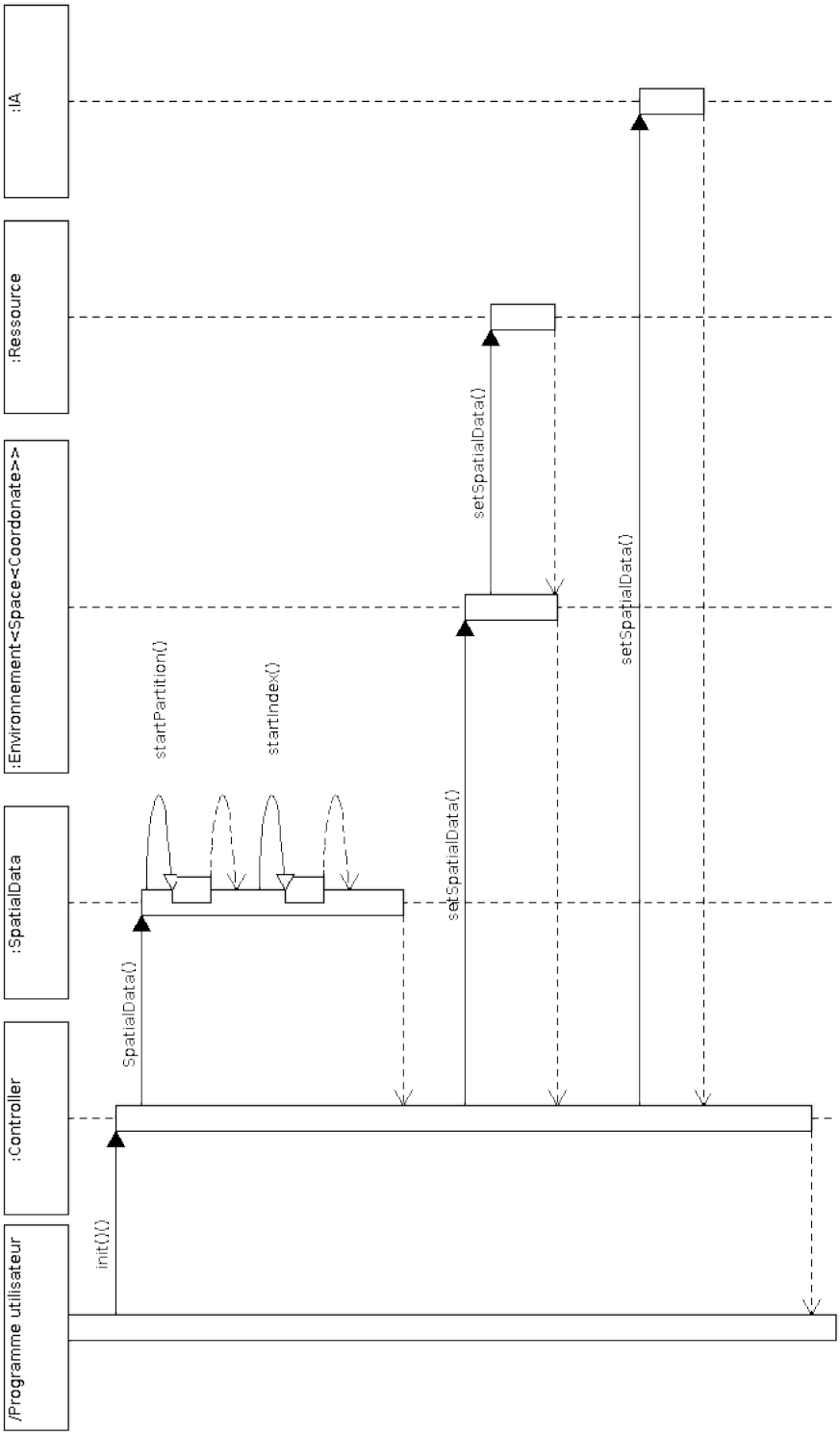


FIGURE 1.9 – Séquence d'initialisation

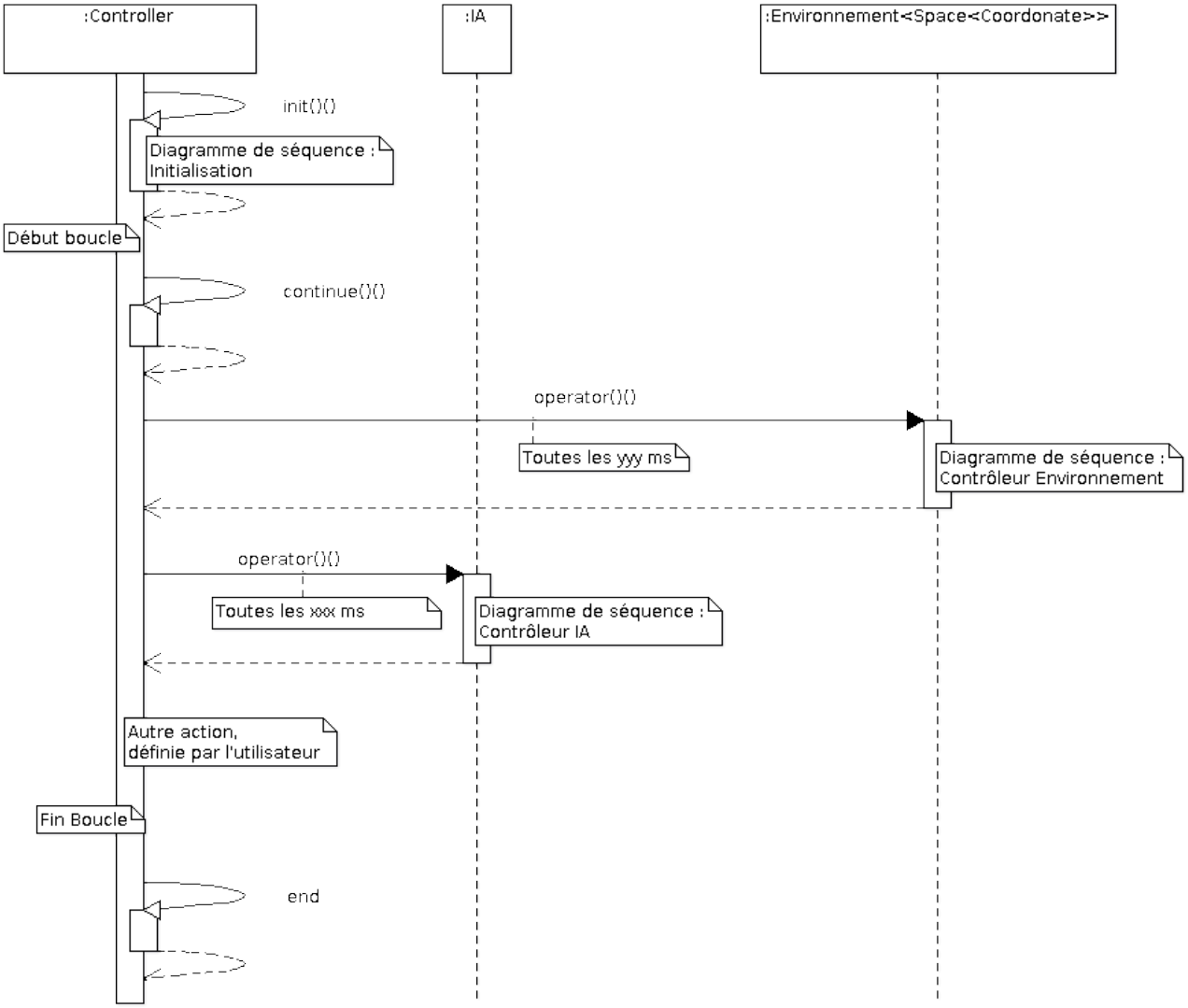


FIGURE 1.10 – Séquence de simulation

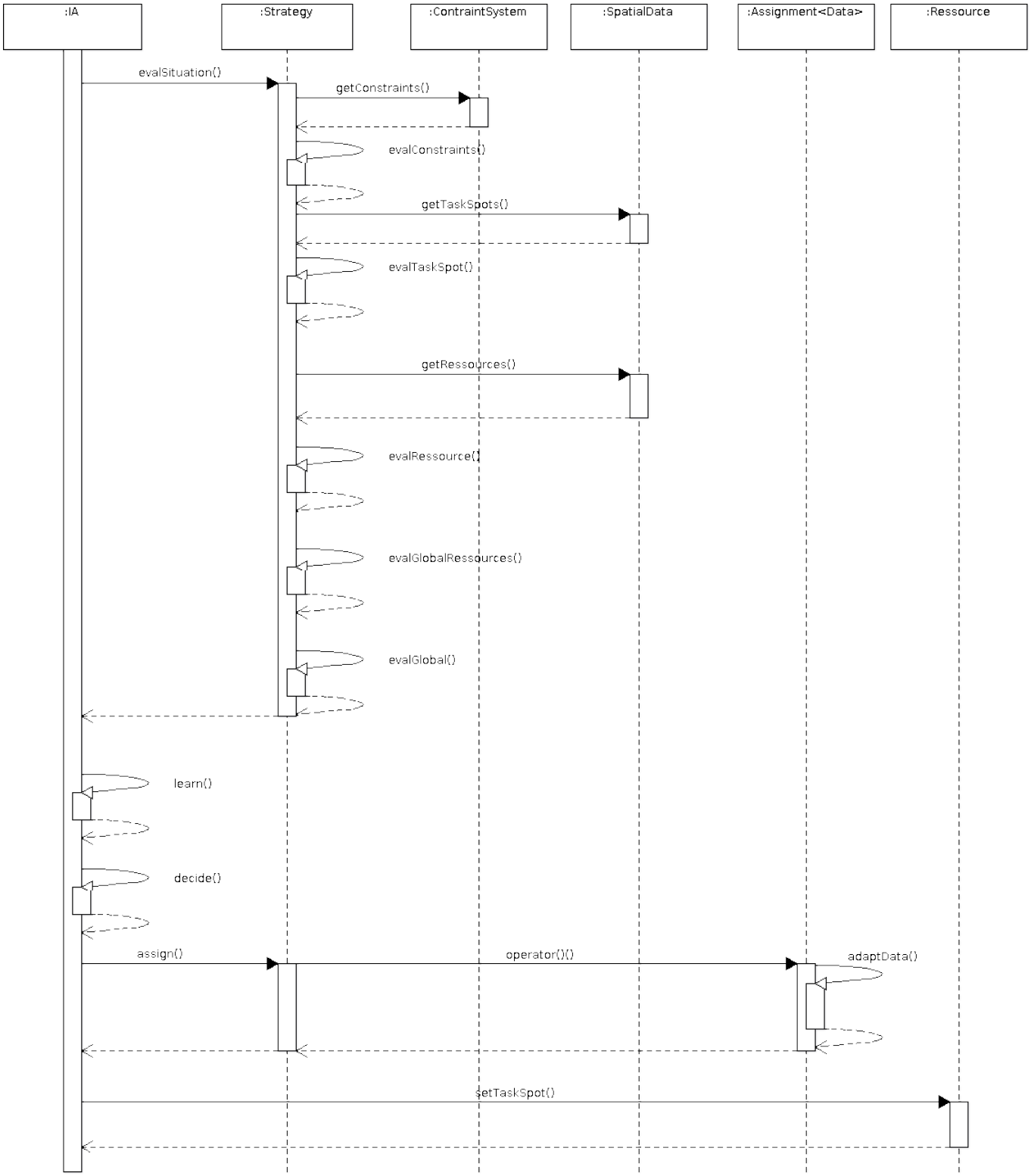


FIGURE 1.11 – Séquence du contrôleur de l'IA : cycle « observation – apprentissage – prise de décision - affectation »

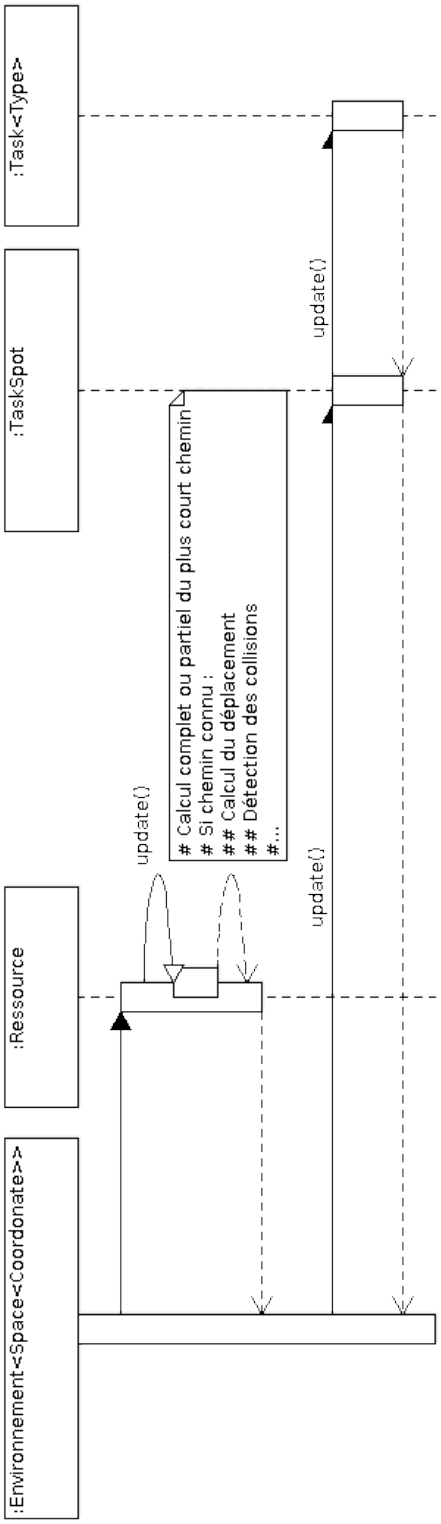


FIGURE 1.12 – Séquence de mise à jour de l'environnement

Chapitre 2

Application

2.1 Description du système et objectifs

L'application servira de démonstration à l'IA construite avec le Framework. Il s'agira d'un petit jeu de stratégie en temps réel.

L'environnement sera une carte 2D avec des murs (les obstacles), des ouvriers (les ressources), ainsi que des mines d'or et de pierres (les tâches sont donc au nombre de deux). L'action de miner durera un certain temps qui occupera un ouvrier et lui permettra de faire augmenter le compteur général d'or ou de pierres.

L'application permettra donc à l'utilisateur d'avoir un exemple concret d'utilisation du Framework.

Les fonctionnalités suivantes permettront à l'utilisateur d'interagir avec le monde pour voir la réaction de l'IA :

- Supprimer / Créer un ouvrier
- Augmenter / Diminuer le nombre d'or / de pierres en réserve
- Fixer un ratio entre or et pierres
- Fixer un objectif à atteindre pour l'or ou/et la pierre

Bibliothèque multimédia (affichage, gestion des saisies) : SFML
Outil de création des cartes : Tiled Map Editor
Bibliothèque d'import des cartes de Tiled : TMX Lib