Première partie

Les plans

Les plans sont des successions d'actions. Nous distinguons trois formes de plans :

- l'*autoplan* qui s'exécute en toile de fonds à chaque tick ; ils sert à décrire des plans relatifs à la physiologie de l'agent. Il a un label bleu dans l'interface,
- le *birthplan* s'exécute à la création de l'agent (soit à l'initialisation des peuplements, soit après la création par un agent ultérieurement). Il a un label rouge dans l'interface, *Remarque : autoplan et birthplan ne sont pas liés à des cognitons*
- les *plans* proprement dits, influencés et conditionnés par les cognitons. Un plan doit obligatoirement être conditionné par un ou plusieurs cognitons (qui doivent être dans l'esprit de l'agent pour pouvoir s'exécuter), il peut aussi être influencés par d'autres cognitons (l'influence ne s'exercera que si le cogniton origine de celle-ci est dans l'esprit de l'agent). Ils ont un label noir dans l'interface.

Deuxième partie

Les actions

Les actions sont exécutées à chaque pas de temps (*tick*) et sur le *patch* où se situe l'agent au moment de leur exécution.

Remarque : il existe cependant une action logique L_Instant qui permet d'exécuter plusieurs actions en un seul tick.

Les actions se spécialisent en actions proprement dites (préfixe A) ainsi qu'en actions logiques (préfixe L) qui regroupent des actions et d'autres actions logiques (cf figure 1).

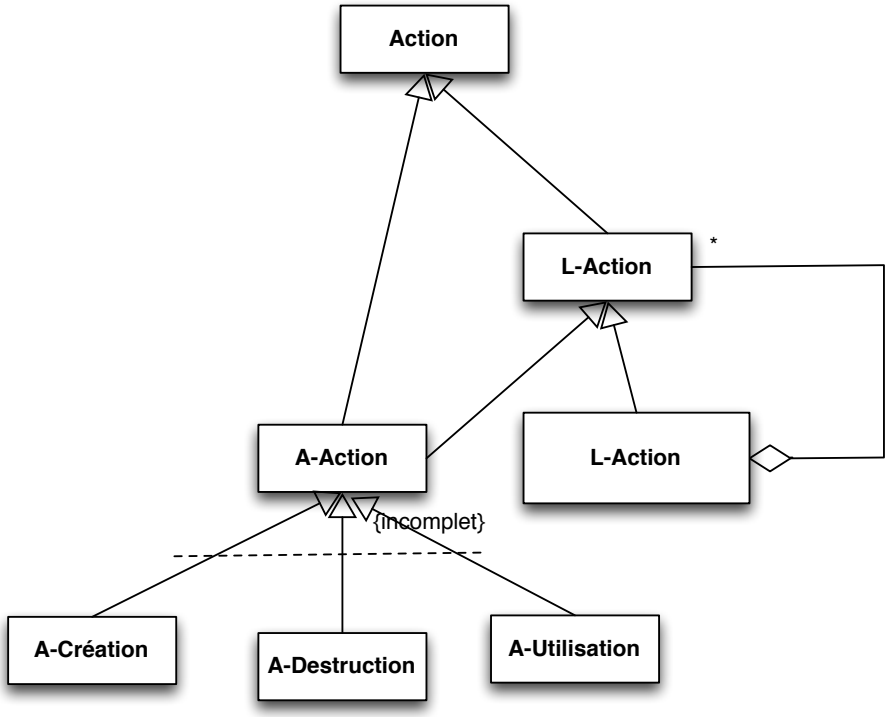

FIGURE 1 – Hiérarchie des actions

1 Les actions A

1.1 Les actions relatives aux objets

1.1.1 A_CreateObject

Cette action crée un objet conformément au processus de fabrication (recette ou *recipe*) défini au préalable.

La signature de l'action est A_CreateObject(Created Object) :

– **Created Object**, est l'objet à fabriquer.

Exemple (pour un agent agriculteur) A_CreateObject(Houe) lance la recette de fabrication de la Houe qui nécessite un bâton pour le manche et un soc (pièce métallique). Si l'agent ne possède pas les deux objets requis pour fabriquer la houe l'action ne fera rien, sinon, elle fabriquera 1 unité de cet objet et l'ajoutera à l'inventaire de l'agent exécutant l'action. Les objets élémentaires ayant servi à la fabrication seront supprimés de l'inventaire.

1.1.2 A_AddObject

Action permettant d'ajouter N objets à l'inventaire de l'agent.
La signature de l'action est A_AddObject(Modified Object, N) :
- **Modified Object**, précise le type d'objet utilisé pour créer directement l'objet à ajouter à l'inventaire de l'agent (sans utiliser la recette)
- **N**, le nombre d'objets à ajouter.

1.1.3 A_DropObject

Action permettant de supprimer N objets de l'inventaire de l'agent.
La signature de l'action est A_DropItem(Modified Object, N) :
- **Modified Object**, type de l'objet à retirer de l'inventaire de l'agent,
- **N**, le nombre d'objets à retirer.

1.1.4 A_Transform

Cette action transforme une ressource prélévée sur un patch en objet.
La signature de l'action est A_Transform (Resource to Collect, Modified Object) :
- **Resource to Collect**, le type de la ressource que l'agent prélève,
- **Changed Object**, l'objet correspondant (pour le modélisateur) à ajouter à l'inventaire de l'agent.

Exemple : un agent exécute A_Transform (Blé, Meule).

1.1.5 A_UseObject

Cette action correspond à l'usage d'un objet.
La signature de l'action est A_UseObject(Modified Object, N) :
- **Modified Object**, le type d'objet à utiliser,
- **N**, le nombre .

1.1.6 A_AddObjectXCogniton

La signature de l'action est A_AddObjectXCogniton(Modified Object, variation,base, Cogniton) :
- **Modified object**, le type d'objet à ajouter,
- **variation**, la variation du nombre d'objets à ajouter,
- **base**, le nombre d'objets à ajouter par défaut,
- **Cogniton**, le cogniton référence.

Cette action, suite à l'existence d'un Cogniton dans l'esprit de l'agent, ajoute des objets à l'inventaire de l'agent en fonction de paramètres contextuel :

- si le cogniton n'existe pas dans l'esprit de l'agent, l'action ajoute *base* objets à l'inventaire,
- si le cogniton existe dans l'esprit de l'agent, l'action ajoute *base* + (*variation* * poids de *Cogniton*) objets à l'inventaire.

Exemple : un agent qui possède le Cogniton Artisan avec un poids de 2, exécute l'action A_AddObjectXCogniton(Houe, 0.5,1, Artisan) ce qui lui permet d'ajouter 1+ 0.5*2 c'est-à-dire 2 Houes à son inventaire ; si l'agent ne possède pas le Cogniton Artisan l'exécution ajoute 1 seule Houe à son inventaire.

1.2 Les actions relatives aux aménagements

1.2.1 A_CreateFacility

Cette action permet à l'agent de construit l'aménagement sur le patch où il se trouve au moment de l'exécution ; la signature de l'action est A_CreateFacility(Facility) :

- **Facility**, le type d'aménagement à construire

1.2.2 A_EraseFacility

Cette action permet à l'agent de détruire l'aménagement du patch où il se trouve au moment de l'exécution, la signature de l'action est A_EraseFacility(Facility) :

- **Facility**, le type d'aménagement à détruire

1.2.3 A_GoToFacility

Cette action permet à l'agent d'atteindre le patch le plus proche disposant de l'aménagement (si celui ci existe) , la signature de l'action est A_GoToAmenagement(Facility) :

- **Facility**, le type d'aménagement à cibler

.

1.2.4 A_UseFacility

Cette action permet à l'agent l'usage de l'aménagement (à condition qu'il soit sur le patch sur lequel se trouve l'agent), la signature de l'action est A_UseAmenagement(Facility) :

- **Facility**, le type d'aménagement à utiliser

1.3 Les actions relatives aux cognitons

1.3.1 A_AddCogniton

Cette action ajoute un cogniton à l'esprit de l'agent.
La signature de l'action est A_AddCogniton(Cogniton) :
– **Cogniton**, le cogniton à ajouter

1.3.2 A_ChangeCognitonWeight

Cette action modifie le poids d'un cogniton.
La signature de cette action est A_ChangeCognitonWeight(Cogniton,N) :
– **Cogniton**, le cogniton à modifier
– **N**, la valeur à ajouter au poids du cogniton

1.3.3 A_DoubleCognitons

Cette action ajoute deux cognitons à l'esprit de l'agent.
La signature de cette action est A_DoubleCognitons(Cogniton1,Cogniton2) :
– **Cogniton1**, le premier cogniton à ajouter,
– **Cogniton2**, le deuxième cogniton à ajouter.

1.3.4 A_EmitCogniton

Cette action modifie le poids du cogniton de tous les agents se trouvant sur le même patch que l'exécutant.
La signature de l'action esr A_EmitCogniton(Cogniton, n)
– **Cogniton**, le cogniton à modifier,
– **n**, la valeur à ajouter au poids du cogniton.

1.4 Les actions relatives aux déplacements

1.4.1 A_MoveTowards

L'action permet à l'agent de se déplacer vers une cible spécifiée.
Remarque : la cible aura été définie par l'action A_ GetAnotherSettlementPatch.

1.4.2 A_MoveRandomly

L' action permet à l'agent d'effectuer un pas dans une direction aléatoire.
Remarque : la longueur du pas dépend de la facilité de traversée du patch (cf. passability)

1.4.3 A_DoNothing

Cette action permet à l'agent de n'exécuter aucune action.

1.4.4 A_ CreateSettlement

Cette action permet à l'agent de se déplacer aléatoirement jusqu'à ce qu'il trouve un ensemble de patchs (les 8 patchs voisins immédiats) dépeuplé de manière à créer un nouveau *Settlement*.

1.4.5 A_GetAnotherSettlementPatch

Cette action définie la cible utilisée dans l'action A_MoveTowards comme étant un des *Settlement* pris au hasard dans l'ensemble des *Settlement* existants.

1.4.6 A_GoBackHome

Cette action permet à l'agent de se déplacer d'un pas vers son lieu de création s'il s'agit d'un des agents des peuplements initiaux, sinon vers le lieu où se trouvait l'agent qui l'a engendré.

1.4.7 A_Move

Cette action déplace l'agent d'un pas dans une direction donnée.
La signature de cette action est A_Move(String) :

- **String**, "NORTH","SOUTH","WEST","EAST" correspond à la direction que prendra l'agent

1.4.8 A_SearchForResources

Cette action permet à l'agent de rechercher autour de lui (dans son champ de vision) la ressource passée en paramètre, pour cela il se dirigera vers le patch le plus proche et possédant cette ressource en quantité maximum. S'il ne trouve pas de patch contenant cette ressource, il se déplacera aléatoirement de un pas.

La signature de cette action est A_SearchForResources(Resource To Collect) :

- **Resource To Collect**, correspond à la ressource recherchée.

1.4.9 A_SmellAndMove

Cette action permet à l'agent de chercher dans son voisinage immédiat le patch contenant la ressource passée en paramètre en plus grande quantité et de faire un pas dans sa direction.

La signature de cette action est A_SmellAndMove(Resource To Collect) :

– **Resource To Collect**, correspond à la ressource recherchée.

1.5 Les actions relatives aux agents

1.5.1 A_GiveBirth

L'action permet à l'agent qui l'exécute de créer un nouvel agent sur le patch où il se situe : cet agent sera identique à ceux du peuplement initial.

1.5.2 A_Die

L'action permet à l'agent qui l'exécute de se supprimer.

1.5.3 A_DieIfAttributeUnderZero

L'action supprime l'agent qui l'exécute si la valeur de l'attribut (ou caractéristique) passé en paramètre descend en dessous de zéro. La signature de cette action est A_DieIfAttributeUnderZero(attributeToCompare) :

– **attributeToCompare**, correspond à l'attribut à vérifier

1.5.4 A_ChangeAttribute

L'action modifie une caractéristique de l'agent.

La signature de l'action est A_ChangeAttribute(Modified attribute,n) ;

– **Modified attribute**, correspond à la caractéristique à modifier

– **n**, donne la valeur à ajouter à la caractéristique

1.5.5 A_CreateGroup

L'action permet à l'agent de créer un groupe.

La signature de l'action est A_CreateGroup(GroupToCreate) :

– **GroupToCreate**, donne le type du groupe à créer.

1.5.6 A_HireForRole

L'action permet à l'agent exécutant d'ajouter un agent présent sur le même patch que lui à son groupe .

La signature de l'action est A_HireForRole(GroupToCreate) :

- **GroupToCreate**, donne le type du groupe dans lequel l'agent sera admis

Remarque : un agent peut appartenir à plusieurs groupes

2 Les actions L

2.1 Tests

La structure générale des actions logiques de type Test est

```
SI  condition ALORS action interne 1
              SINON action interne 2.
```

Remarque : ces actions logiques sont toutes composées de deux actions internes (qui peuvent elles-mêmes être composites).

Les diverses actions suivantes vont préciser le type de condition.

2.1.1 L_CompareAttribute

La condition porte sur la valeur d'un attribut de l'agent.

La signature de cette action est L_CompareAttribute(attributTocompare, comparator, n) :

- **attributTocompare**, est l'attribut concerné,
- **comparator** l'opérateur de comparaison <, >, <=,>=,==,
- **n** la valeur de comparaison.

2.1.2 L_CompareObject

La condition porte sur le nombre d'objets d'un type donné possédés.

La signature de cette action est L_CompareObject(objectTocompare, comparator, n) :

- **objectTocompare**, est le type d'objet concerné,
- **comparator** l'opérateur de comparaison <, >, <=,>=,==,
- **n** le nombre d'objets.

2.1.3 L_CompareResource

La condition porte sur la valeur de la ressource présente sur le patch où est situé l'agent.

La signature de cette action est L_CompareResource(resourceTocompare, comparator, n) :

- **resourceTocompare**, est le type de la ressource concernée,
- **comparator** l'opérateur de comparaison <, >, <=,>=,==,
- **n** la valeur de la ressource.

2.1.4 L_IsFacilityHere

La condition teste la présence de l'aménagement *(facility)* concerné sur le patch où est situé l'agent.

La signature de cette action est L_IsFacilityHere(facility) :

- **facility**, le type d'aménagement testé.

2.1.5 L_IsAnyFacilityHere

La condition teste la présence d'un aménagement de n'importe quel type sur le patch où est situé l'agent.

2.1.6 L_OwnFacility

La condition teste la possession d'un aménagement donné.

La signature de cette action est L_OwnFacility(facility) :

- **facility**, le type d'aménagement testé.

2.1.7 L_OwnCogniton

La condition teste la possession d'un cognition.

La signature de cette action est L_OwnCogniton(cogniton) :

- **cogniton**, le type de cogniton testé.

2.1.8 L_OwnObject

La condition teste la possession d'un objet.

La signature de cette action est L_OwnObject(object) :

- **object**, le type d'objet testé.

2.2 Algorithmiques

Ces actions logiques composites sont associées à des structures standard algorithmique.

2.2.1 L_Instant

Cette action composite effectue la globalité de ses actions internes en un seul tick.

2.2.2 L_Random

Cette action composite effectue une seule de ses actions internes choisie aléatoirement en un seul tick.

2.2.3 L_RandomPercent

Cette action composite n'a que deux actions internes et choisit selon le pourcentage l'action interne à exécuter.

La signature de cette action est L_RandomPercent(n) :

– **n**, le pourcentage de chance qu'à l'action interne 1 de s'exécuter.

2.2.4 L_Repeat

Cette action composite ne comporte qu'une action interne qu'elle va exécuter n fois en n tick.

La signature de cette action est L_Repeat(n) :

– **n**, le nombre d'itérations.