Rapport de TER
Extension NetLogo (interaction)

Équipe **Teutons** :
Arnault LEMMEL, Alexandre LOUVEAU,
Kiefer RONCOLI, Joseph STICH

Encadrants :
Jacques FERBER,
François SURO

22 mai 2018

Résumé

Ce projet vise à définir un framework d'interaction dans un système multi-agents en se basant sur les travaux existants de CogLogo et IODA. Deux modèles sont utilisés afin d'appuyer nos choix de conceptions tout au long du rapport : un modèle de combat et un modèle de reproduction. En comparant les implémentations de ces modèles dans les extensions CogLogo et IODA, nous avons défini une implémentation idéale, accessible et réaliste de notre schéma d'interaction.

Sujet : http://www.lirmm.fr/~ferber/TER/metaciv_m1.htm

Git du projet : https://github.com/cpttrvs/M1_TER

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 POA - qu'est-ce qu'un système multi-agents

Les systèmes multi-agents sont une approche de l'intelligence artificielle qui cherche à solutionner des problèmes plus où moins complexes en utilisant la force d'une multitude d'intelligences limitées et en considérant leurs apports personnels comme une intelligence collective, plutôt qu'une somme de ces intelligences. [1]

Nous nous intéresserons dans ce TER à la problématique de l'interaction dans les systèmes multi-agents ainsi que les intégrations déjà existantes.
Nous chercherons alors à définir ce qui serait, selons nous, une approche efficace et pratique d'implémentation des interactions entre agents, cognitif ou non.
Pour cela, nous nous appuierons sur l'environnement de modélisation multi-agents Netlogo et sur les extensions qu'il propose.
Il faut donc commencer par définir quelques termes qui reviendront plus tard dans ce rapport :

- agent : unité autonome du système, doté d'un système réactif et cognitif, qui peuvent dialoguer et, ce qui nous intéresse, interagir les uns avec les autres.
- breed : catégorie ou type d'agent. Dans Netlogo, une classe d'agent.
- tick : unité atomique de temps dans une modélisation. Chaque tick est l'instant dans lequel un agent va pouvoir réaliser une action.

1.2 Principe d'interaction

1.2.1 Qu'est-ce qu'une interaction ?

Afin de simuler des comportements plus réalistes, il faut permettre aux agents de créer des situations d'interaction, comme l'échange, le combat ou la diffusion d'information. Nous définirons une interaction entre seulement deux agents et dans un environnement donné. En effet, une interaction globale peut être réduite à un ensemble d'interaction un à un : un orateur devant une foule ne fait d'interagir indirectement avec chaque personne, faisant appel aux capacités personnelles de compréhension et d'analyse de chacun.
Pour simuler ces comportements, il faut définir les composantes essentielles à tout type d'interaction, et donc définir un framework d'interaction.

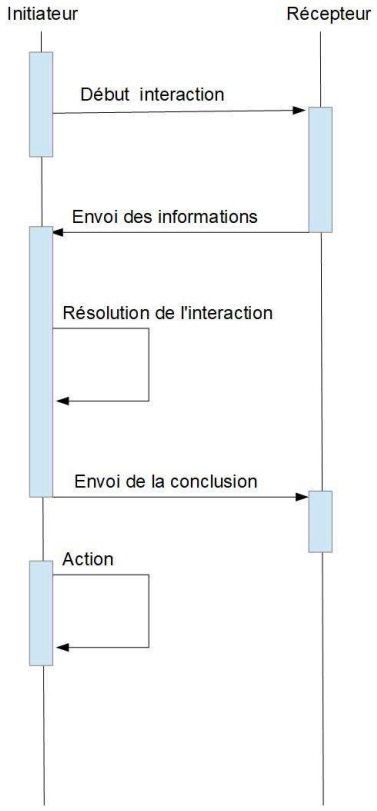

FIGURE 1.1 – Diagramme séquence d'une interaction généraliste

Dans le cas général d'une interaction, l'initiateur débute l'interaction quand les conditions sont satisfaites, puis récupère les informations utiles de l'agent destinataire utiles. Le modèle est alors résolu afin de n'avoir qu'une seule issue, puis l'agent destinataire en est informé afin qu'il puisse agir en conséquence. Le comportement des agents revient à la normale suite à l'interaction.

1.2.2 Initiateur et destinataire

Tout d'abord il faut définir un agent initiateur de l'interaction. Il s'agit de l'agent qui va effectuer l'action initiale, comme proposer un échange, tenter de frapper ou transférer une information ; cette action vise un autre agent, l'agent destinataire, qui, en fonction de ses capacités et celles de l'initiateur, va définir l'issue de l'interaction de l'agent initiateur vers l'agent destinataire. [2]

1.2.3 Habilitons

Il faut ensuite définir un moyen de représentation des capacités de l'agent, initiateur comme destinataire, à réagir à une interaction donnée. Une capacité, qu'on appellera habiliton[1], se définit donc par une valeur numérique afin d'influencer positivement ou négativement, et donc d'évaluer, les différentes issues possibles d'une interaction.

1.2.4 Environnement

Une interaction se définit aussi par une troisième entité : l'environnement dans lequel l'action se déroule. En effet l'agent initiateur et destinataire se trouvent tous deux dans un espace lors de l'interaction, ce qui influence, positivement ou négativement, leurs habilitons respectifs. Dans le cadre d'un échange par exemple, si l'environnement est oppressant pour le destinataire (pluie, insécurité, ...), sa vigilance vis-à-vis de la marchandise ne sera peut-être pas la même que dans un environnement favorable (en plein jour, en lieu publique, ...) et aura donc plus de chance de se faire escroquer.

1.2.5 Plans

Un plan définit une issue de l'interaction. Chaque habiliton de l'agent initiateur et destinataire ainsi que l'environnement influencent la valeur numérique d'un plan. Un plan symbolise donc une action de l'agent initiateur sur l'agent destinataire, comme celui de toucher dans le cadre d'un combat, ou de réussir la tentative d'escroquerie lors d'un échange.

1. Viens du latin habilitas (habilité) et suffixe -ton (unité atomique), représente la capacité d'un agent à réaliser une interaction

Chapitre 2

Modèles d'interactions

2.1 Modèles généraux

Nous allons maintenant illustrer ce concept d'interaction défini en introduction avec deux modèles distincts. Le modèle de combat illustre l'usage de l'environnement dans une interaction, et le modèle de reproduction prouve l'indépendance de notre schéma d'interaction puisqu'il n'utilise pas d'agents cognitifs.

2.1.1 Combat

Deux équipes d'agents s'affrontent. Les agents se promènent sur une carte et l'interaction débute lorsque deux agents ennemis sont sur le même patch. Quand c'est le cas, l'agent dont c'est le moment d'agir entame l'interaction, il va prendre en compte la connaissance du terrain et la force de son adversaire. Si celles-ci sont supérieures aux siennes, alors il rate son coup et perd de l'énergie. Sinon son adversaire a une chance de bloquer le coup ce qui lui donne un bonus à la contre attaque. Enfin si l'initiateur arrive à toucher le destinataire, ce dernier perd un point de vie, et le premier perd un point d'énergie.

Les schémas ci-dessous représentent le modèle d'interaction.

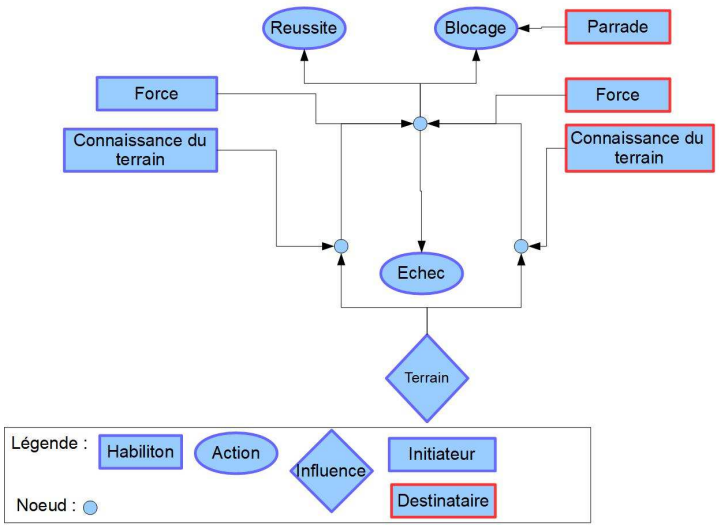

FIGURE 2.1 – Schéma d'interaction de combat.

Ici on compare la nature du terrain et la connaissance qu'en ont chaque agent. Ensuite on compare le résultat du noeud précédent additionné à la force de l'initiateur, avec celui du destinataire. Si le résultat est positif pour l'initiateur alors on compare le score de parade avec le résultat du noeud précédent, si la parade est supérieure, alors on fait l'action blocage. Sinon c'est une réussite.

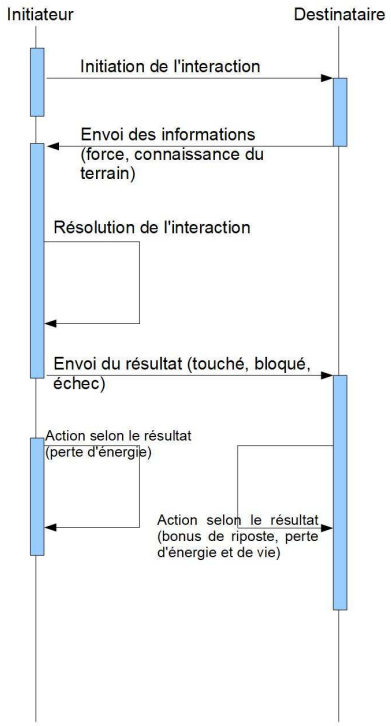

FIGURE 2.2 – Diagramme séquence de l'interaction de combat.

2.1.2 Reproduction

Le modèle représente la reproduction des paons. On a donc deux types d'agents : les mâles et les femelles. Dans ce modèle ce ne sont que les femelles qui initient les interactions lorsqu'elles sont sur le même patch qu'un mâle. Lorsque c'est le cas, la femelle compare la taille du mâle et la taille de ses plumes avec le plus gros mâle qu'elle a en mémoire. Si le mâle qu'elle a en face d'elle est plus gros et a de plus longue plumes alors elle accepte la reproduction et garde en mémoire ses mensurations. Sinon elle s'en va toute seule sans rien faire.

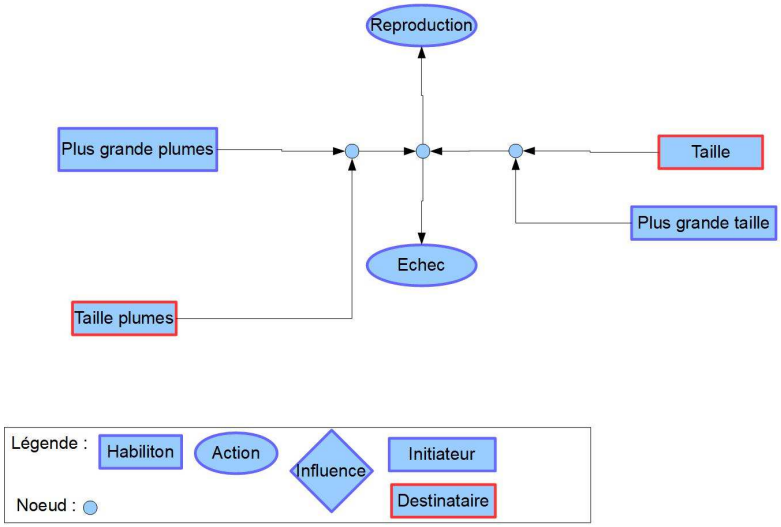

FIGURE 2.3 – Schéma d'interaction pour la reproduction.

Dans le schéma on compare juste les caractéristiques du mâles avec celle que la femelle a en mémoire. Si les deux caractéristiques sont plus grandes chez le mâle alors il y a reproductions. Sinon il ne se passe rien.

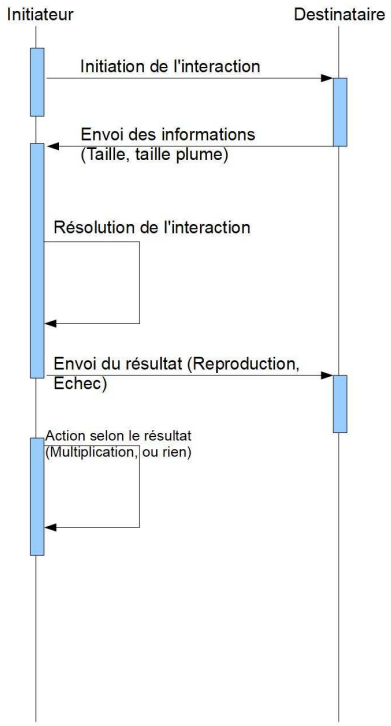

FIGURE 2.4 – Diagramme séquence de la reproduction.

2.2 Netlogo

2.2.1 Standard pour la simulation

Netlogo est un environnement créé dans le but de pouvoir y développer des programmes respectant la philosophie multi-agents. De ce fait il est possible, et même nécessaire pour certains modèles, d'y intégrer des interactions entre agents.
On peut envisager pour cela plusieurs méthodes :

— L'agent initiateur fait l'ensemble des calculs nécessaires à chaque interaction.
Cette méthode est la plus simple et elle permet de les intégrer dans un modèle où ils n'étaient pas prévus.
Elle nécessite néanmoins des calculs répétés.
— L'agent initiateur récupère pour les calculs des paramètres préparés par l'agent receveur pendant son tick.

Chaque breed pouvant être la cible d'une interaction doit être prévue pour recevoir ladite interaction.
Cela nécessite des champs et des calculs qui ne sont utiles que dans le cadre des potentielles interactions (et donc sont inutiles si elles n'ont pas lieu).

Dans le modèle de combat n'utilisant que netlogo, c'est la première solution qui a été privilégiée. Cela nécessite des champs et des calculs qui ne sont utiles que dans le cadre des potentielles interactions, et donc sont inutiles si elles n'ont pas lieu.

```
to-report hit? [x]
  let myhit random 10 * skill
  report (myhit > 80)
end

to-report block? [x]
  if([blocked] of x) [report false]
  let myhit (random blockChance) ^ 2 + [block] of x
  report (myhit > 20)
end

to hit [x]
  if(hit? x) [
    ifelse(not block? x) [ask x
      [set life (life - (maxNumber 1 (([strenght] of myself) - armor)))]]
    [set blocked true]
  ]
end
```

FIGURE 2.5 – définition de l'interaction de coup porté du modèle de combat, en netlogo.

2.2.2 Contraintes

Netlogo est fait pour les systèmes multi-agents, qui reposent souvent sur le fait que la solution à un problème compliqué peut venir de plusieurs individus qui effectuent des actions simples.
Le problème principal que cela pose c'est qu'il est vite difficile de réaliser des opérations complexes dans le langage.
En effet, on ne peut définir de structure et le seul moyen d'y avoir accès est de créer des "agents structures" qui vont servir de solveurs, qui ne seraient créés que pour accomplir leur tâche et mourir, ce qui contreviendrait rapidement à l'esprit du multi-agent.
On doit donc se contenter d'interactions simples sous peine d'avoir rapidement un code très verbeux, et des fonctions d'interaction qui ne seraient que des suites de calcul et de conditions imbriquées.
Le Netlogo n'est, de base, pas adapté pour des calculs complexes et c'est pour cela qu'il y a sur leur site de multiples extensions disponibles qui permettent de s'attaquer à des questions précises.

2.3 Coglogo

2.3.1 Coglogo pour l'interaction

L'extension CogLogo permet de modéliser des agents cognitifs à l'aide de cognitons[1], représentant les buts, opinions et désirs, influençant le déclenchement de plans. On retrouve une structure semblable à notre définition d'une interaction, ce qui nous permet de modéliser notre framework (figure ci dessous).

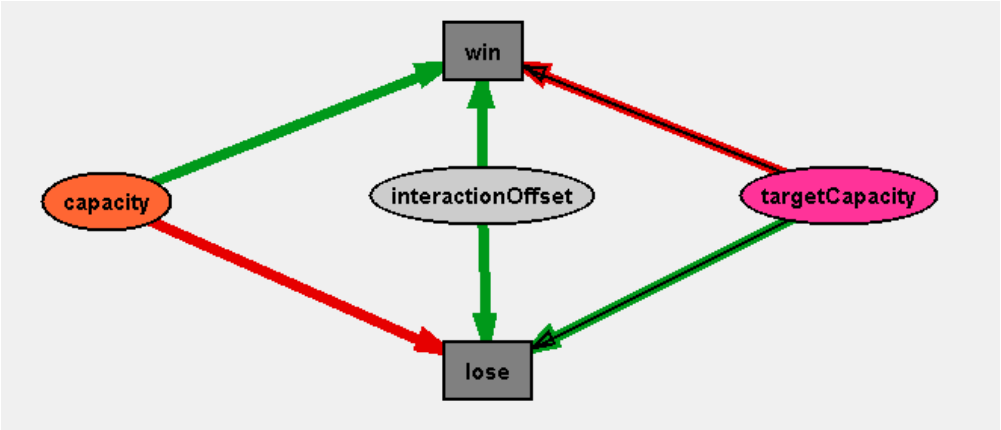

FIGURE 2.6 – modèle d'interaction dans CogLogo.

A gauche, « capacity » représente un habiliton de l'agent initiateur, et à droite, « targetCapacity » représente un habiliton de l'agent destinataire. Comme l'interaction se définit du point de vu de l'initiateur, « capacity » influence positivement le plan « win », et négativement le plan « lose » (et inversement pour « targetCapacity »). En effet, si la capacité de l'agent initiateur représente une valeur plus importante que celle de sa cible, alors il gagnera l'interaction. La prise de décision peut être maximale ou stochastique : pour y rajouter du réalisme, nous avons opté pour une prise de décision stochastique biaisée. Le plan choisi suite à la prise de décision appellera donc la fonction NetLogo du même nom.

2.3.2 Limites

CogLogo permet donc de prendre une décision en fonction des cognitons et des liens définis dans le modèle, mais l'extension ne permet pas de résoudre l'interaction directement. En effet, la valeur du cogniton n'est que le résultat de notre formule liant les différents habilitons et les modificateurs de l'environnement.
Dans la figure (schéma d'interaction), on remarque trois choses :

- les modificateurs d'environnement ne sont pas représentés puisqu'ils sont dans la formule, en amont, du cogniton.
- les cognitons n'influencent les plans que par addition ou soustraction. Or, nous souhaitons modéliser des comportements prenant en compte plusieurs critères qui ne peuvent parfois pas uniquement se représenter par ces opérateurs. Pour palier à ce problème, nous devons définir la formule dans la définition du cogniton, mais cette solution rend la définition invisible sur le schéma.

— la présence d'une valeur seuil « interactionOffset » qui permet de faire cohabiter le schéma cognitif classique de l'agent et notre schéma d'interaction. Cette valeur permet de toujours valoriser l'interaction aux autres plans lorsqu'une interaction se produit. En revanche, nous n'avons pas besoin de cette valeur si l'on souhaite modéliser des agents réactifs, comme dans notre modèle de reproduction (2.1.2).

CogLogo permet donc de simuler des interactions entre agents, réactifs ou cognitifs, mais en utilisant en parallèle des définitions en NetLogo lors de l'assignation de la valeur des habilitons. Comme nous souhaitons une extension où le framework permette d'y définir l'intégralité de l'interaction, CogLogo est insuffisant.

2.4 IODA

2.4.1 Fonctionnement des interactions

L'extension IODA repose sur deux principes, qui sont définis dans deux fichiers séparés.
Premièrement il y a une matrice d'interaction.
Elle est composée pour chaque breed d'une liste possible d'interactions accompagnées chacune :

- d'une priorité permettant de déterminer l'ordre dans lequel les interactions vont être considérées.
- de la breed destinataire possible, et de la distance maximum à laquelle ce destinataire doit se trouver.
- si nécessaire, on peut préciser les critères guidant le choix du destinataire si plusieurs sont possibles.

```
Swordmans Move 0
Swordmans Rest 5
Swordmans Rush 8 Swordmans 20
Swordmans Rush 8 Axemans 20
Swordmans Attack 10 Swordmans 5
Swordmans Attack 10 Axemans 5
Swordmans Rush 19 Bowies 20
Swordmans Attack 20 Bowies 5
Swordmans die 50 UPDATE
```

FIGURE 2.7 – Matrice d'interaction d'un épéiste du modèle de combat en IODA.

On voit pour chaque ligne où elle sont nécéssaire :
Breed de l'initiateur, nom de l'interaction, définition de la priorité, Breed du destinataire, distance maximale de celui-ci.

Ensuite, il y a le fichier contenant la définition des interactions.
Celles-ci sont à mettre dans deux catégories : celles qui peuvent se dérouler en parallèles et celles dites exclusives, c'est à dire qu'elle sera, si choisie par la matrice et si les conditions sont respectées, la seule interaction initiée par l'agent pendant son tick.
On peut effectivement ajouter à ces interactions des conditions qui vont aider la matrice à déterminer l'action à choisir. Ainsi, si la condition n'est pas respectée, une autre interaction sera sélectionnée. De plus, on peut faire dans une même interaction différentes actions suivant le résultat de différentes conditions.

```
exclusive interaction Rest
  condition missingHP?
  actions   rest
end

exclusive interaction Rush
  actions   rest
end

exclusive interaction Attack
  actions   attack
end
```

FIGURE 2.8 – définition des interactions correspondantes à la matrice précente.

Enfin il faut redéfinir dans le code Netlogo certaines fonctions IODA, une seule étant réellement nécessaire dans le cadre d'un modèle basique, la fonction qui permet de déterminer le voisinage de l'agent.
Il faut également définir dans le code Netlogo les fonctions "'reporter"' utilisées dans les conditions et les méthodes appelées dans les interactions.
Il suffira ensuite de créer des agents IODA à la création des agents et d'appeler IODA à chaque tick qui lancera la cognition et les interactions.

2.4.2 Limites

Le premier problème de IODA vient de son typage. En effet, Netlogo n'est pas un langage fortement typé, et on peut ainsi définir une méthode qui sera appelée par des breeds multiples. Avec IODA on perd cet avantage.
En effet, le fait que IODA s'occupe de l'appel des méthodes entraîne une complexification de la syntaxe.Une simple fonction dead ? qui regarde si l'agent est encore en vie devra ainsi être redéfinie pour chaque breed.

```
to-report default::dead?
  report life <= 0
end

to-report Swordmans::dead?
  report default::dead?
end

to-report Lancers::dead?
  report default::dead?
end

to-report Axemans::dead?
  report default::dead?
end

to-report Bowies::dead?
  report default::dead?
end
```

FIGURE 2.9 – fonction dead ? en IODA.

```
to-report dead?
  report (life < 0)
end
```

FIGURE 2.10 – fonction dead ? en Netlogo.

Ensuite, si l'extension amène bien des choses pratiques pour réaliser des interactions (cognition,choix d'un destinataire, accès à celui-ci et aux autres destinataires potentiels), l'interaction en elle même doit toujours être codée en netlogo.
En effet, IODA est au final moins une extension qui permet de créer des interactions qu'une extension permettant de créer des comportements cognitifs basés sur les possibilités d'interaction. On retrouve donc les mêmes contraintes : l'interaction doit être simpliste sous peine d'être une suite de calculs fastidieux.
Elle ne répond donc pas à ce que nous cherchons : un modèle d'interaction ergonomique.

Chapitre 3

Comparaison CogLogo et IODA

3.1 Approche globale du problème

La première différence entre ces deux extensions se situent au niveau de leur philosophie.
Dans Coglogo, la cognition est définie en fonction du moi et du nous : on agit en fonction de ce que je sais ou de ce que savent les individus qui nous ressemblent.
Dans IODA, elle est définie en fonction du moi et de l'autre : que suis-je, qui est l'autre et quelles sont nos caractéristiques.
Nous trouvions que ces deux modèles manquaient d'un facteur essentiel qui est l'environnement. Pour rester dans le parallèle avec la psychologie : nous avons fait le choix de nous intéresser au moi, à l'autre et à ce qui nous entoure. Autrement dit, l'initiateur, le destinataire et l'environnement.

3.2 Résolution de l'interaction

La décision d'un schéma cognitif dans CogLogo peut être prise de différentes manières : en prenant la valeur maximale parmi les plans, en représentant chaque plan par une probabilité et donc de choisir de manière stochastique le plan à effectuer, ou en rajoutant un biais à la méthode stochastique afin d'accroître la probabilité des plans les plus probables selon le biais. Cette approche permet une simulation réaliste des comportements.
Avec IODA, l'interaction choisie est celle ayant la plus grande valeur et toutes ses conditions respectées.

Comme notre modèle n'intègre pas de cognition, on ne peut pas vérifier les conditions autre que celle de la validité de la breed des agents initiateur et destinataire. De plus, notre modèle vise à être une représentation réaliste d'une situation d'interaction. On pourra donc choisir, comme CogLogo, la prise de décision souhaitée parmi les plans entre un choix maximal, stochastique, ou stochastique avec biais.

3.3 Flexibilité du modèle

Avec CogLogo, on peut définir un schéma de cognition pour plusieurs breed. Cependant, une breed ne peut disposer que d'un seul schéma cognitif, comme l'indique la primitive « coglogo :choose-next-plan » qui ne prend pas de paramètre, ce qui correspond à l'utilisation classique du modèle.

Chaque ligne de matrice dans IODA correspond à une seule breed, et ayant comme cible une seule breed également. Il faut donc définir une interaction par breed initiatrice et par breed destinataire.

Comme un agent peut avoir plusieurs types d'interactions, interactions qui peuvent cibler plusieurs breeds différentes, notre approche permettra donc de définir un schéma pour plusieurs breeds initiatrices et ayant plusieurs breeds destinataires. Une primitive de lancement d'interaction, du contexte de l'agent initiateur, prendrait donc comme paramètre le nom de l'interaction et l'agent destinataire. L'extension se chargerait ensuite de vérifier si les breeds des agents correspondent aux breeds définies dans le modèle.

3.4 Implémentation dans Netlogo

On a ici une problématique très intéressante.
En effet, au niveau de la façon dont s'insèrent ces deux extensions dans Netlogo nous avons deux approches très différentes.
La où Coglogo va faire du calcul en interne, IODA va préférer laisser le développeur gérer dans Netlogo la quasi intégralité du code. Les deux approches ont leurs avantages et leurs inconvénients.
Premièrement la façon dont il faut préparer l'interaction.
Dans Coglogo, il va falloir avant chaque interaction avoir une fonction qui remplit les champs définis dans le schéma, car il ne correspondent pas nécessairement à des valeurs utiles dans le reste du code. Pour les conditions d'effet d'une interaction, il est possible de définir en amont une variable (booléenne dans le principe) qui définit si une action est possible ou non.
Dans IODA, il faut définir un certain nombre de rapporteurs (ici, des fonctions renvoyant un booléen) qui serviront de conditions pour le choix de l'interaction. Ensuite, la différence majeure se situe au niveau de la gestion d'un agent lui même.
La où Coglogo peut être intégré simplement au comportement d'un agent (en remplaçant la partie s'occupant de la cognition par un modèle, et l'interaction par un autre), IODA demande d'intégrer le code au modèle. Ainsi, on peut facilement passer à Coglogo depuis un code Netlogo.
Dans le cas de IODA, une grande partie du code devra être réécrite ou renommée afin de l'adapter à la syntaxe nécessaire au bon comportement de l'extension. En fait, si on veut schématiser la programmation, le code Netlogo appelle le schéma Coglogo, tandis que le schéma IODA appelle le code Netlogo. Nous avons fait le choix ici de nous inspirer de Coglogo. Cela permet des architectures et donc des codes plus flexibles, et donc plus facilement adaptables, le coût n'étant au final qu'une redéfinition de certaines variables.
Dans un cadre théorique optimal, l'idée aurait été qu'une interface permette directement de lier les variables d'un agent Netlogo à ceux de sa représentation dans un schéma. Néanmoins nous n'avons trouvé aucun moyen de le mettre en place avec les outils que propose Netlogo.

3.5 Interface

CogLogo implémente une interface graphique schématique permettant de modéliser les cognitions d'un agent très facilement. En effet, il ne faut aucune compétence de programmation ni pour faire un schéma, ni pour le comprendre. Il y a des cognitons (ovales), des plans (rectangles) et des liens de différentes sortes (influences positives ou négatives, de condition ou de renforcement) (schéma ci-dessous).

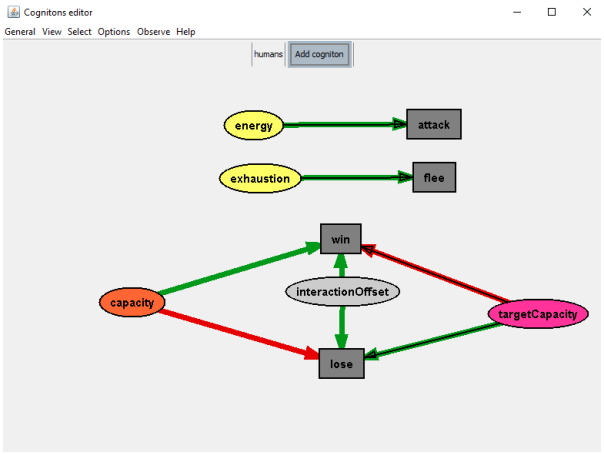

FIGURE 3.1 – Interface d'une interaction Coglogo.

IODA, en revanche, utilise une approche matricielle des interactions. Pour chaque breed, on définit l'action, la priorité, la breed cible et la distance pour que l'interaction se déclenche. Cette visualisation est intéressante puisqu'elle permet de centraliser les différentes informations utiles à une interaction. (figure ci-dessous).

```
Swordmans Move 0
Swordmans Rest 5
Swordmans Rush 8 Swordmans 20
Swordmans Rush 8 Axemans 20
Swordmans Attack 10 Swordmans 5
Swordmans Attack 10 Axemans 5
Swordmans Rush 19 Bowies 20
Swordmans Attack 20 Bowies 5
Swordmans die 50 UPDATE
```

FIGURE 3.2 – Définition d'une interaction IODA.

Pour intégrer au mieux notre modèle d'interaction, nous utiliserons l'approche CogLogo. Nous voulons une interface graphique afin de visualiser rapidement et sans notion de développement comment deux agents interagissent entre eux. Cependant, pour déclencher une interaction, ceci se définit à même le code NetLogo en utilisant des primitives, puisque notre framework s'intéresse à l'interaction elle-même et non aux déclenchements de celle-ci, ce qui relève de la cognition.(cf : 4.3)

Chapitre 4

Proposition d'extension

4.1 Spécification

Une interaction, bien qu'étant un concept défini, reste quelque chose de relativement large.
En partant de ce constat, nous avons décidé de créer un modèle à l'échelle atomique.
De fait, nous ne considérerons pas un combat comme une interaction mais comme une suite de micro-interactions, chacune correspondant à une tentative d'action exécuté sur un temps instantané : la granularité de nos schémas d'interaction est donc très fine.
De plus toute interactions entre n individus peut être résumée comme une somme d'interactions deux à deux entre chacun de ces individus.
Ainsi, et en variant l'échelle de temps, on peut considérer un maximum d'interactions en conservant une grande précision.

Nous avons donc cherché à quoi pouvait ressembler un modèle d'interaction lorsqu'on voulait le représenter en un programme.
Nous sommes arrivés à la conclusion qu'une interaction pouvait n'être, une fois modélisée, qu'un modèle de calcul.
Il faut, pour permettre tous types d'interactions plusieurs paramètres :

- Les capacités des deux individus en présence (capacités de l'initiateur et capacités du destinataire, représentées par les habilitons).
- les modificateurs créés par l'environnement.
- une équation déterminant les probabilités de chaque issue en fonction des paramètres précédents.

Une autre composante qui nous semblait nécessaire était celle de la lisibilité.
En effet, une interaction complexe, symbolisée par une longue équation n'aurait été compréhensible que par la personne qui l'aurait définie.
Nous avons donc décidé que l'interaction devrait être représentée par un schéma, et non par du code. Ce facteur sera détaillé dans la présentation de l'interface. (4.3)

4.2 Formule par noeud

La seule structure qui semble nous permettre une représentation à la fois claire et précise d'une équation d'interaction est une structure de graphe.
Elle permet ainsi de regrouper des calculs dans des noeuds pour créer des interactions complexes tout en restant claires.
L'exemple le plus éloquent est celui d'un lancer quel qu'il soit.
A partir du moment où un objet va quitter l'initiateur pour se diriger vers le destinataire il y a plusieurs éléments à prendre en compte :

— du côté de l'initiateur : La vision, la force, l'expertise.
— du côté du destinataire : la vision, l'agilité, la vitesse.
— du côté de l'environnement : le vent, la distance.

Ces éléments ne sont pas tous dissociables les uns des autres. En effet, si la distance va avoir de facto un effet négatif sur les chances de toucher, elle va aussi avoir un effet multiplicateur sur l'impact du vent. Suivant le niveau de précision de l'interaction, l'expertise et la force auront également un effet sur l'impact du vent - Si l'individu est expérimenté, il saura gérer le vent, et si la vitesse de l'objet est grande, le vent modifiera moins sa trajectoire.

Pour ce type de problématique, nous avons décidé que le modèle d'interaction le plus intéressant était un modèle de calcul poussé, permettant tous les calculs basiques des mathématiques.
Ce qui nous paraît nécessaire :

— opérateurs (-,+,/,*,%)
— minimum
— maximum
— puissance,logarithme
— opérateurs booléens (=,<,>,|,&)

De plus, pour éviter de devoir définir une interaction par configuration possible, si certains champs peuvent être absents il faut pouvoir leur affecter une valeur par défaut.

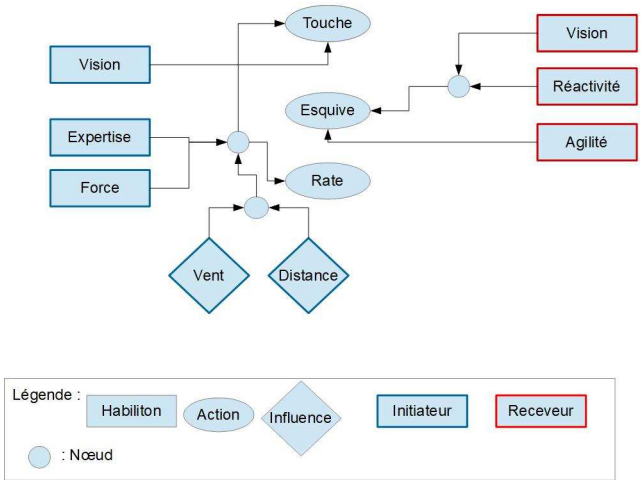

FIGURE 4.1 – Schéma d'interaction pour l'exemple du lancé.

4.3 Interface

4.3.1 Apparence générale

L'interface se divise en trois partie. Les deux onglets à gauche permettent de sélectionner l'élément que l'on souhaite rajouter au schéma d'interaction. La partie du milieu permet de visualiser le schéma, et la partie de droite nous donne les informations détaillées du dernier élément sélectionné.

Cette organisation a été choisi afin de bien séparer les différentes informations, et pour garder une interface complète mais pas trop surchargée. Les inspirations principales sont les interfaces de logiciel de type IDE et des moteurs de création de jeux (Unreal par exemple).

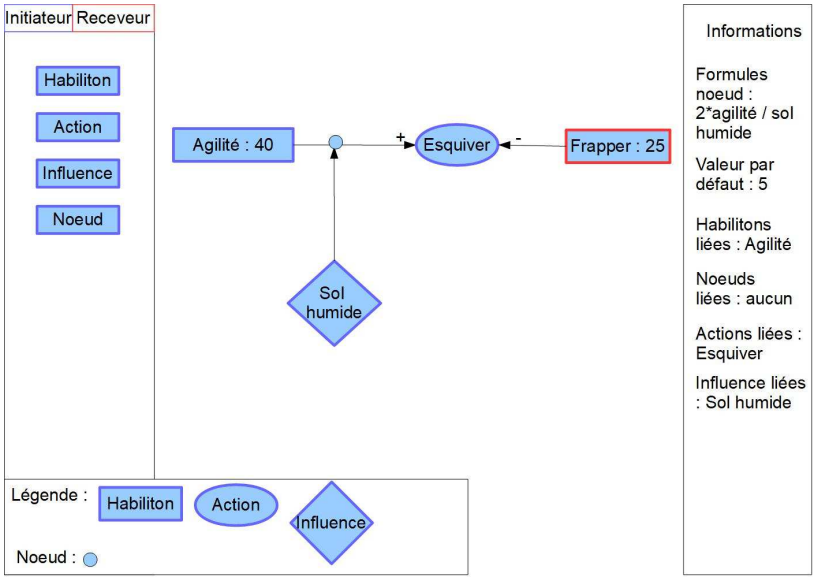

FIGURE 4.2 – Mock up de l'interface.

Dans l'interface, pour différencier les éléments appartenant à l'initiateur de l'interaction et du receveur, on utilise une couleur différente sur la bordure.

4.3.2 Fonctionnement

Un initiateur peut générer plusieurs éléments dans l'interface : un habiliton qui est liée à une variable interne de celui ci ; une action, qui est liée à une procédure ou une fonction dans le code de l'agent ; et enfin une influence qui est une variable liée au patch sur lequel se trouve l'agent. Les agents receveurs ne peuvent générer que des habilitons et des influences car le schéma ne leur fait pas prendre d'action en direct. Les schéma doivent prendre en compte le plus d'éléments possible, même s'ils ne sont pas applicables à tous les agents. Par exemple dans le schéma ci dessus, on utilise une influence "'Sol Humide"', mais il se peut que le terrain ne possède pas de variable Sol humide, dans ce cas on utilisera la valeur par défaut donnée dans l'interface. Ou alors il se peut que si un agent a une caractéristique spécifique il annule purement et simplement

une influence, dans ce cas c'est au concepteur du schéma de penser à faire un noeud pour tester la présence de cette caractéristique et agir en conséquence.
Les noeuds ne sont pas liés à l'initiateur ou au receveur, car ils prennent en entrée plusieurs habilitons et influences pour ne donner en sortie qu'une formule permettant de décider d'une action à effectuer, cette formule peut d'ailleurs renvoyer un booléen ce qui est très utile dans certains cas, comme le modèle de reproduction.

4.3.3 Utilisation

A l'utilisation lorsqu'on veut ajouter un habiliton ou une influence au schéma, il suffit d'appuyer sur le bouton correspondant dans la partie de gauche. Ensuite une fenêtre s'ouvre, nous demandant le nom de l'habiliton ou de l'influence, ainsi que sa valeur par défaut au cas où la valeur ne serait pas encore initialisée lors de l'appel du schéma. Pour les actions il suffit de donner le nom de la fonction ou de la procédure à utiliser lorsqu'on veut en placer une.
Pour lier les éléments entre eux, il suffit de sélectionner un élément et de remplir dans la partie information, le champs correspondant aux habilitons ou actions liées.
Enfin il est possible de bouger les éléments du schéma par drag'n'drop, afin de le rendre plus lisible.

Chapitre 5

Conclusion

Pour synthétiser, nous allons énumérer les aspects qui nous semblent nécessaires pour un framework d'interaction :

- Il faut qu'il s'appuie sur l'initiateur, le destinataire et leur environnement.
- Il faut que celui-ci soit simple à prendre en main et donc qu'il puisse s'insérer dans un code préexistant.
- Il faut qu'un modèle créé soit lisible, surtout par quelqu'un n'ayant pas de connaissance en programmation.
- Il faut qu'il permette de créer des modèles complexes.
- Il faut qu'il puisse, au besoin, intégrer une part contrôlée d'aléatoire, par exemple par la décision stochastique, pour se rapprocher du naturel.
- Les interactions doivent être autonomes ou indépendantes.

L'environnement existants étudiés dans ce rapport (Netlogo,Coglogo,IODA) ne permettent pas un tel modèle :

- Netlogo ne propose pas les structures pour le créer.
- Coglogo n'a pas été créé pour cela et est donc rapidement limité dans les fonctionnalités qu'il propose pour réaliser des interactions.
- IODA change la syntaxe du code, et n'est donc pas pratique pour le développeur, et ne permet pas une représentation simple et/ou schématique de l'interaction, ce qui est dommageable pour les néophytes.

De façon plus générale, les extensions existantes permettent de créer un modèle cognitif de l'agent mais ne s'occupent pas de résoudre l'interaction.

C'est pour cela que l'apparition d'une extension gérant exclusivement la résolution des interactions nous semble nécessaire pour les systèmes multi-agents.
La solution à base de graphe présentée nous semble être la méthode la plus efficace pour réaliser un tel framework.

Bibliographie

[1] J. Ferber. *Les Systemes Multi Agents : vers une intelligence collective.* InterEditions, 1995.

[2] Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations : From theory to implementation. University of Lille, France, 2008.