Projet d'UML
Dilemme Itéré des Prisonniers

(IUT, département informatique, 1re année)

Laurent AUDIBERT

L'objectif du projet consiste à proposer un modèle UML d'une petite application permettant de mettre en œuvre des tournois de stratégies jouant au *Dilemme Itéré des Prisonniers*, puis de proposer une implémentation en Java de cette application.

1 Le contexte : du dilemme du prisonnier à l'interaction de tribus

1.1 Le dilemme du prisonnier

Deux suspects porteurs d'armes ont été arrêtés devant une banque et mis dans deux cellules séparées. Les deux prévenus ne peuvent pas communiquer et doivent choisir entre avouer qu'ils s'apprêtaient à commettre un hold-up ou ne rien avouer. Les règles que le juge leur impose sont les suivantes :

- si l'un avoue et pas l'autre, celui qui avoue sera libéré en remerciement de sa collaboration et l'autre sera condamné à cinq ans de prison ;
- si aucun n'avoue, ils ne seront condamnés qu'à deux ans de prison, pour port d'arme illégal ;
- et si les deux avouent, ils iront chacun faire quatre ans de prison.

Dans cette situation, il est clair que si les deux s'entendent (pas d'aveu), ils s'en tireront globalement mieux (2×3 ans de remise de peine) que si l'un des deux dénonce l'autre (1×5 ans de remise de peine). Mais alors l'un peut être tenté de s'en tirer encore mieux en dénonçant son complice. Craignant cela, l'autre risque aussi de dénoncer son complice pour ne pas être le dindon de la farce. Le dilemme est donc : *faut-il accepter de couvrir son complice (donc de coopérer avec lui) ou le trahir ?*

Ce modèle très simple de la théorie des jeux semble appréhender en miniature les tensions entre cupidité individuelle et intérêts de la coopération collective. Pour cette raison, il est devenu un des modèles les plus utilisés en sociologie, biologie et économie.

1.2 Le dilemme du prisonnier itéré

Le dilemme du prisonnier devient plus intéressant et plus réaliste lorsque la durée de l'interaction n'est pas connue. On peut alors envisager de se souvenir du comportement d'un joueur à son égard et développer une stratégie en rapport. Par exemple, si je sais que mon adversaire ne coopère jamais, mon intérêt sera de ne pas coopérer non plus, sous peine d'être systématiquement floué. Par contre si je sais que mon adversaire coopérera toujours quoi qu'il arrive, j'aurai intérêt à être vicieux et ne jamais coopérer pour maximiser mon gain.

1.3 L'interaction de tribus

Considérons deux tribus d'indigènes partant à la chasse. Ces deux tribus peuvent choisir entre coopérer ou trahir la tribu adverse lors d'une confrontation. Lorsque la situation du dilemme est itérée,

le jeu devient très intéressant, car la question ne se pose plus sous la forme : *trahir ou coopérer ?*, mais sous la forme : *quelle stratégie faut-il adopter en fonction du comportement passé de l'entité adverse ?*.

Nous supposons que les deux tribus ne peuvent pas passer d'accord. La seule information qu'une tribu connaît sur l'autre est son comportement passé lors des coups précédents. Les décisions des deux tribus lors de la partie sont prises simultanément. Le nombre de parties n'est pas connu à l'avance. Par rapport au simple dilemme des prisonniers nous choisissons de rajouter la possibilité de renoncer à jouer, mais ce refus est définitif.

Décrivons la variante adoptée de manière plus abstraite : deux entités peuvent choisir entre coopérer (notation c), trahir (notation t) ou renoncer (notation n). Si l'une trahit et l'autre coopère (partie $[t, c]$), celle qui trahit obtient un gain de T unités et celle qui coopère (et s'est donc fait duper) obtient un gain de D unités. Lorsque les deux entités coopèrent (partie $[c, c]$), elles gagnent chacune C unités en récompense de leur association. Quand elles trahissent toutes les deux (partie $[t, t]$), elles gagnent P unités pour s'être laissées piéger mutuellement. Si une partie n'a pas lieu parce que l'une a refusé de jouer les deux entités gagnent N unités. Le choix des coefficients T, D, C, P et N n'est pas fortuit. Conformément aux n°181 de *POUR LA SCIENCE* nous prenons : $T = 5$, $D = 0$, $C = 3$, $P = 1$, $N = 2$.

2 Description du projet

2.1 L'application à réaliser

L'application à réaliser doit permettre de mettre en œuvre des tournois de stratégies jouant au dilemme itéré des prisonniers tel que décrit dans la section 1.3. Les stratégies sont implémentées en Java. Il faudra implémenter les stratégies décrites section 2.2 plus vos stratégies personnelles ou celles de vos collègues. Un tournoi est la confrontation d'un ensemble de stratégies. L'ensemble de stratégies est un sous-ensemble, choisi par l'utilisateur, des stratégies disponibles. Une rencontre entre deux stratégies se joue en n tours, n étant également défini par l'utilisateur. Dans un tournoi, une stratégie doit rencontrer toutes les stratégies, elle-même comprise, de l'ensemble sélectionné. Le score réalisé par une stratégie est la somme de ses points récoltés lors de chacune des confrontations.

Exemple

L'utilisateur demande un tournoi entre les stratégies *Gentille* et *Méchante* (cf. section 2.2) avec des confrontation en 20 tours.

L'application réalise le tournoi puis affiche le résultat sous la forme suivante :

```
         | Gentille | Méchante | TOTAL
---------+----------+----------+-------
Gentille |    60    |     0    |   60
---------+----------+----------+-------
Méchante |   100    |    20    |  120
---------+----------+----------+-------

Stratégie gagnante du tournoi : Méchante
```

2.2 Exemples de stratégie

Gentille : Je coopère toujours.

Méchante : Je trahis toujours.

Donnant-Donnant : Je coopère à la première partie, puis je joue ce qu'a joué l'autre à la partie précédente.

Donnant-Donnant-Dur : Je coopère à la première partie, puis je coopère sauf si mon adversaire a trahi lors de l'une des deux parties précédentes.

Méfiante : Je trahis à la première partie, puis je joue ce qu'a joué l'autre à la partie précédente.

Rancunière : Je coopère à la première partie, mais dès que mon adversaire trahit, je trahis toujours.

Périodique-Méchante : Je joue trahir, trahir, coopérer, trahir, trahir, coopérer, . . .

Périodique-Jentille : Je joue coopérer, coopérer, trahir, coopérer, coopérer, trahir, . . .

Majorité-Mou : Je joue ce que l'adversaire a joué en majorité, en cas d'égalité et à la première partie, je coopère.

Majorité-Dur : Je joue ce que l'adversaire a joué en majorité, en cas d'égalité et à la première partie, je trahis.

Dur : Je trahis tant que mon adversaire coopère. Dès qu'il trahit, je renonce.

Sondeur-4-coups : Aux quatre premiers coups, je joue coopérer, coopérer, trahir, trahir. Ensuite, si mon adversaire a trahi trois fois ou quatre fois, je renonce, sinon je coopère tout le reste du temps.

Donnant-Donnant-Avec-Seuil : je joue la stratégie Donnant-Donnant, mais, tous les cinq coups, je compte mon score et, si j'ai obtenu moins de deux points en moyenne par coup, je renonce.

Graine-de-champion : Je coopère au premier coup. Tous les 20 coups, j'évalue mon score. Si, en moyenne, il est inférieur à 1,5, je renonce. En règle générale, je coopère. A chaque fois que l'autre trahit, si je ne suis pas déjà dans une phase de punition, je rentre dans une phase de punition. Si mon adversaire m'a trahi n fois (en dehors des phases de punition), la phase de punition dure $\frac{n.(n+1)}{2}$ trahisons, et est suivie de deux coopérations.

3 Partie UML du projet (Responsable : Laurent Audibert)

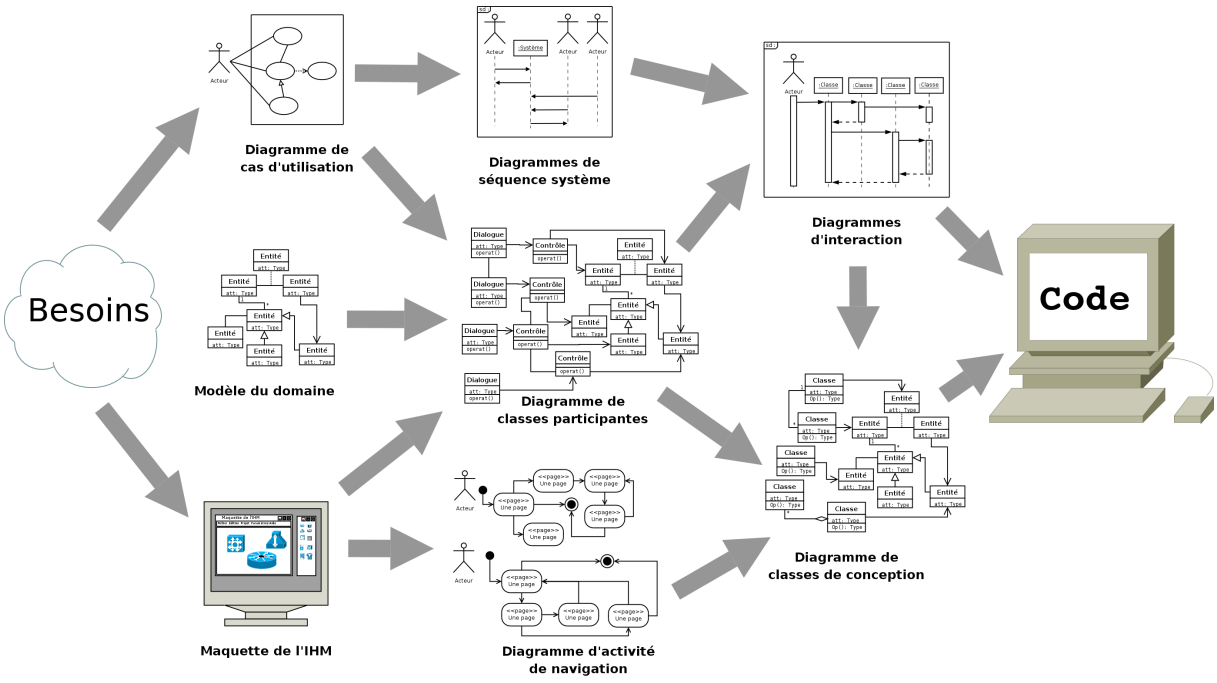

FIG. 1 – Chaîne complète de la démarche de modélisation du besoin jusqu'au code.

3.1 Méthode de conception

Pour modéliser ce système, vous devrez utiliser la méthode minimale décrite en cours et synthétisée par la figure 1.

Dans le cadre du projet d'UML, vous n'irez pas jusqu'à l'écriture du code : l'implémentation concerne le projet de java. Toutes les autres étapes devront être prises en compte. Une manière de procéder pourrait donc être la suivante.

1. Identifiez et spécifiez les besoins en réalisant un *diagramme de cas d'utilisation* accompagné de la description textuelle des cas d'utilisation.
2. Illustrez les cas d'utilisation à l'aide de *diagrammes de séquence système*.
3. Analysez et modélisez le domaine en proposant un *modèle du domaine* sous forme de diagramme de classes.
4. Proposez une *maquette d'interface* graphique ou textuelle de l'application.
5. A l'aide d'un *diagramme d'activité de navigation* montrez la navigation dans l'interface graphique ou textuelle de l'application.
6. La première version du diagramme de classes (le modèle du domaine) doit être reprise pour y adjoindre les classes de contrôle et d'interface. Vous obtiendrez ainsi le *diagramme de classes participantes*
7. Chaque diagramme de séquence système doit être détaillé en prenant en compte les classes figurant dans le diagramme de classes participantes. Ces *diagrammes d'interaction* éclatent le système, qui n'est plus considéré comme une boîte noire, en plusieurs classes participantes de manière à montrer comment elles interagissent pour réaliser chacun des cas d'utilisation.
8. Les messages échangés entre les classes participantes dans les diagrammes d'interaction permettent de définir les opérations de chacune des classes participantes. Ces opérations seront spécifiées dans la nouvelle version du diagramme de classes : le *diagramme de classes de conception*. On veillera à ce que toutes les opérations soient réalisables, et que tout ce qui a été défini dans les activités de navigation soit présent.

Cette méthode est minimale et flexible. Si vous trouvez pertinent de spécifier certains aspects du logiciel en utilisant d'autres diagrammes, vous êtes naturellement libres de le faire.

Si vous pensez ne pas pouvoir tout faire, concentrez-vous sur les cas d'utilisation les plus importants et développez-les jusqu'au diagramme de classes de conception, quitte à négliger certaines fonctionnalités secondaires. En d'autres termes, un travail en profondeur d'abord sera préféré à un travail en largeur d'abord.

3.2 Modalités de remise du projet

Ce projet est à réaliser en binômes. Le rendu se fera impérativement sous forme de dossier papier à remettre le 7 janvier 2007 au secrétariat à l'attention de votre chargé de TD. Le 8 ou le 9, vous recevrez par mail une confirmation de la bonne réception du projet. Si ce n'est pas le cas, inquiétez-vous !

4 Partie Java du projet (Responsable : Bouchaïb Khafif)

4.1 Objectifs

L'objectif est d'implémenter le jeu du dilemme des prisonniers dont l'analyse et la conception ont été faites dans le cadre du projet UML

Objectif pédagogique : Mise en œuvre des concepts objets enseignés dans le cours java, en particulier l'héritage et le polymorphisme.

Implémentation demandée :

- L'affichage est en mode texte (ne pas faire de graphisme).
- Les résultats d'un tournoi doivent pouvoir être sauvegardés.

4.2 Méthodologie de programmation objet

Contrairement à la programmation procédurale, la programmation *objet* considère le problème à résoudre en dernier. Il faut créer et tester chaque classe indépendamment des autres classes (dans la mesure du possible) selon une méthode ascendante :

1. Créer et tester les classes de niveau 0 (classes n'utilisant pas d'autres classes conçues pour le projet)
2. Créer et tester les classes de niveau 1 (classes utilisant les classes de niveau 0, sans faire appel à d'autres classes de niveau 1).

On réitère le processus consistant à créer les classes de niveau $k + 1$ à partir des classes de niveau k, jusqu'à obtenir une classe résolvant le problème.

Exemple : pour programmer un jeu d'échecs on s'attachera d'abord à créer les classes représentant les pièces, puis les classes représentant l'échiquier, puis les classes gérant l'interaction des pièces sur l'échiquier, etc.

Remarque : dans le cas de relations d'héritage il faut créer les classes les plus générales en premier. Ainsi dans l'exemple du jeu d'échecs on créera d'abord une classe Piece, puis les classes Fou, Tour, ... qui en héritent. Ce principe n'est pas contradictoire avec l'approche ascendante : *A* hérite de *B* ne signifie pas que *B* utilise *A* mais que *B est un A* spécifique.

4.3 Organisation du travail

- Le travail s'effectue en binôme d'étudiants. Un unique trinôme sera toléré par groupe possédant un nombre impair d'étudiants. Aucun monôme ne sera admis.
- Chaque binôme remettra au plus tard le xxx[1] un dossier comprenant : un manuel utilisateur, un diagramme des classes actualisé, un bilan du projet, les listings et la documentation (javadoc) des classes. De plus chaque binôme enverra un fichier zip contenant les sources (.java) et ainsi que tous les fichiers objets (.class) permettant de tester l'implémentation (sans devoir compiler).
- Une soutenance individuelle, où chaque étudiant présentera le travail qu'il a fait, aura lieu à une date à préciser.

[1] M. Khafif vous communiquera cette date ultérieurement.