MASTER EN SCIENCES INFORMATIQUES

Le dilemme du prisonnier

Compère, Sébastien

Award date:
1998

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1997-1998

Le dilemme du prisonnier

Sébastien Compère

Mémoire présenté en vue de l'obtention
du grade de Licencié en Informatique

A mes parents

Avant-Propos

Tout au long de ce travail, j'ai eu l'avantage de bénéficier des conseils éclairés de mon directeur de mémoire, monsieur Habra. Je tiens à lui exprimer toute ma gratitude pour ses encouragements et la grande disponibilité dont il a fait preuve à mon égard.

Au cours de mes recherches, j'ai, à plus d'une reprise, eu l'occasion de recevoir des avis et des références bibliographiques de monsieur Schobbens. Je le remercie vivement pour l'aide qu'il m'a prodiguée.

Enfin, j'ai une pensée toute particulière pour mes parents qui m'ont toujours soutenu au cours de mes études universitaires.

A tous, je réitère mes plus vifs remerciements.

Tables des matières

Introduction

L'expression '*dilemme du prisonnier*' vient de R. Axelrod qui a imaginé le scénario suivant. Deux malfaiteurs sont arrêtés, l'arme à la main, au moment où ils préparent l'attaque à main armée d'une banque. Ils sont immédiatement séparés l'un de l'autre sans avoir eu le temps de se concerter. Comme l'agression n'a pas eu lieu, ils peuvent rester solidaires en niant toute intention coupable face à leurs interrogateurs. Le juge ne pourra alors que les inculper pour port d'arme prohibée. D'autre part, ils peuvent trahir leur complice dans l'espoir de bénéficier d'une remise de peine. Ils sont ainsi contraints de choisir entre deux options comportant chacune des inconvénients. D'une part, si personne ne trahit, chacun a la garantie de voir sa punition limitée à une amende pour port d'arme prohibée. D'un autre côté, si l'un des deux trahit, et l'autre pas, le premier aura, du fait de sa collaboration, une peine moins lourde que son complice, et ne devra même pas supporter de condamnation pour la détention de son armement. Enfin, si l'un et l'autre trahissent, ils supporteront ensemble une peine plus importante pour tentative d'attaque d'une banque. Le choix repose donc sur le degré de confiance ou de défiance réciproque.

Dans ce contexte, nous faisons référence à la '*théorie des jeux*' qui traite des décisions à prendre dans une situation rendue incertaine par les décisions possibles d'autres individus (la concurrence ou le partenariat). Cette théorie étudie différents types d'interactions entre personnes. Elle peut être formalisée par le triplet suivant: $\langle I, S, P \rangle$

où I représente l'ensemble des individus,

S se rapporte à l'ensemble des stratégies adoptées, et

P concerne les payoffs obtenus à l'issue de la confrontation. Ceux-ci peuvent également être assimilés aux règles du jeu.

Il existe une gamme infinie de jeux. Dans ce travail, nous nous focaliserons sur un ensemble de décisions possibles en fonction d'hypothèses de comportement dans le cadre du dilemme du prisonnier.

L'objectif de ce mémoire est de fournir une base solide pour des développements ultérieurs dans le domaine du dilemme du prisonnier notamment dans l'expérimentation des stratégies. Nous nous efforcerons de rencontrer ce but, tant au niveau rédactionnel qu'au niveau de la programmation. C'est la raison pour laquelle nous présentons, au sein du premier chapitre, un aperçu général des différentes variantes de dilemme du prisonnier et des méthodes de confrontation des stratégies.

Au chapitre II, nous développons en détail un concept crucial au sein de ce travail, à savoir celui de stratégie. A cet égard, nous nous attarderons à en donner les caractéristiques essentielles. Celles-ci seront utilisées ultérieurement.

En vue de constituer une base pour de futurs travaux en ce domaine, le troisième chapitre décrit, d'une part, les principaux éléments constitutifs de notre logiciel, et, d'autre part, formule l'énumération des données fondamentales. Nous donnons également les indications nécessaires permettant d'y adjoindre les compléments indispensables.

Les trois derniers chapitres fournissent différentes pistes de réflexion afin de compléter le logiciel en y insérant des techniques d'intelligence artificielle. Nous aborderons notamment les problèmes relatifs à la représentation de la connaissance et à la façon de reconnaître le comportement adopté par un interlocuteur. Soulignons que ce travail ne constitue pas une fin en soi mais est destiné à être repris et amplifié dans le cadre de recherches ultérieures. Dans le contexte de l'Intelligence Artificielle, l'objectif actuel constitue un premier pas permettant d'appréhender au mieux la stratégie de l'adversaire.

Par ailleurs, au niveau du logiciel, nous avons apporté une attention toute particulière à réaliser une programmation modulaire. Cette optique a pour but de faciliter les améliorations ultérieures. Ce point est notamment développé au sein du chapitre III consacré à la présentation du logiciel.

Chapitre I: Enoncé des variantes du dilemme du prisonnier et des méthodes de confrontation des stratégies

Introduction

Une modélisation de la vie de tous les jours (vie sociale, économique, politique,...) peut être interprétée par le biais d'agents qui interagissent via des 'actions'. L'objectif de la théorie des jeux est de modéliser les décisions susceptibles d'être prises par un partenaire, à partir d'une base de connaissances et d'un ensemble de critères. Le dilemme du prisonnier est un de ces modèles. Il vise à analyser l'interdépendance des actes posés par deux acteurs en compétition et représente la modélisation la plus simple puisqu'elle se réduit à un nombre limité d'alternatives. A chaque étape, les stratégies offrent le choix entre coopération et trahison. Dans certaines alternatives, on ajoute également la possibilité de renoncer. Enfin, cette théorie peut permettre de comprendre les sources de conflits et être utilisée dans leur résolution.

Dans une première partie, nous proposons un aperçu aussi précis et concis que possible du dilemme du prisonnier et de ses variantes. Dans la première section, nous exposons d'abord le problème dans sa perspective '*classique*'[1]. Par la suite, nous présentons différentes variantes du dilemme du prisonnier, à savoir le jeu avec possibilité de '*retour d'ascenseur*', les jeux asynchrones (ou séquentiels) et le jeu avec renoncement. Au niveau informatique, ces variations n'ont leur raison d'être que si les stratégies sous-jacentes peuvent être implémentées

[1] Nous abordons d'abord le dilemme du prisonnier portant sur un nombre fini de périodes. Par la suite, nous expliquons le problème dans une perspective itérée.

dans des outils de simulation. En ce qui nous concerne, nous nous sommes limités à la première variante.

La deuxième section introduit les différents types de confrontations envisageables dans un outil de simulation, à savoir: la '*confrontation simple*', la '*confrontation généralisée*' dans la '*compétition écologique*', l'introduction de perturbations via l'intrusion aléatoire des stratégies et le choix aléatoire des stratégies initiales.

Section I: Présentation du dilemme du prisonnier

Cette section est exclusivement axée sur la perspective '*classique*' du dilemme du prisonnier. Ultérieurement, nous abordons certains jeux plus '*complexes*'. Nous étudions à cet égard le '*retour d'ascenseur*', les jeux asynchrones et l'impact résultant de la possibilité de renoncer dans le chef des acteurs.

Article I: Le dilemme du prisonnier dans sa perspective classique

Avant d'entrer dans le vif du sujet, il nous semble opportun d'aborder cette problématique à l'aide d'un exemple tiré de la vie de tous les jours. Dans cette optique, nous nous référons au film de Gus Van Sant: '*Will Hunting*'. Celui-ci retrace la rencontre de deux personnes: Will Hunting et Sean Mc Guire. L'un et l'autre connaissent des problèmes relationnels. Pour les surmonter, ils doivent accepter de s'ouvrir l'un à l'autre. Mais chacun ignore si son interlocuteur lui rendra la réciproque. Bien que tous deux soient conscients qu'une franche coopération leur sera profitable, ils ne sont pas sans ignorer que le fait de coopérer, à titre personnel, sera uniquement salutaire à l'autre qui sera complètement guéri. Par contre, dans l'hypothèse de non-réciprocité, cette action n'apportera aucune amélioration à la situation de son auteur. Au contraire, elle ne sera bénéfique que s'il y a convergence des efforts de la part des interlocuteurs. Enfin, s'ils ne parviennent pas à se faire mutuellement confiance, alors la situation de chacun ne sera guère améliorée.

Nous avons choisi cet exemple dans le but de montrer que la théorie développée sous sa formulation classique dans le '*dilemme du prisonnier*' est un problème quotidien qu'on retrouve dans plusieurs disciplines. Celles-ci vont de la sociologie à l'économie en passant par la biologie, l'écologie et les sciences politiques. Les exemples ne manquent pas et se retrouvent dans les domaines les plus variés. Relevons entre autres:

- La course à l'armement et au désarmement.
- La concurrence commerciale par la baisse des prix ou la coopération susceptible de déboucher sur une situation de duopole comme c'est le cas en automobile.
- La coopération permettant de mettre au point des standards en télécommunication ou en télévision.
- ...

Dans la perspective classique nous nous limitons à un *nombre fini* et *connu à l'avance d'interactions* entre les différents agents. Afin de mieux cerner la problématique dont il est question, nous pouvons prendre l'exemple de la course aux armements entre deux nations rivales. Si elles décident ensemble de stopper les achats d'armes, elles peuvent réaffecter leur budget de défense à des objectifs qu'on peut qualifier de plus '*utilitaires*'. Par contre, si l'une des deux cesse ses commandes en la matière et l'autre pas, cette-dernière atteindra rapidement une force militaire susceptible d'anéantir celle de son voisin. Le risque d'être '*dupé*' est loin d'être négligeable et, de ce fait, la crainte qui en résulte peut être perçue comme un obstacle à la coopération.

Un danger analogue peut se présenter dans le cas d'une O.P.A. entre trois concurrents. Si chacun ne détient qu'une fraction minoritaire des actions de la société convoitée, l'alliance entre deux protagonistes devient inévitable. Mais la confiance des deux partenaires peut être altérée par le danger latent d'un retournement d'alliance.

Dans la présentation de A. Kehagias (1998), les payoffs sont négatifs. En ce qui nous concerne, et ce sans perte de généralité étant donné qu'il ne s'agit que d'un raisonnement à une constante près, nous ne considérerons que des payoffs positifs. Cette option implique que les agents essayent de maximiser leurs revenus et non pas de minimiser leurs coûts.

Sur base des différentes digressions faites ci-dessus, nous pouvons modéliser le dilemme comme suit:

- si les deux agents coopèrent, ils reçoivent un salaire de récompense C (C pour coopération),
- si tous les deux font défection, ils ne perçoivent qu'une rémunération de punition P sanctionnant leur égoïsme, et
- si seul, un des deux coopère, il reçoit un salaire de dupe D tandis que celui qui trahit obtient une rétribution de trahison T.

Ces résultats sont repris au sein de la *matrice des payoffs* donnée ci-dessous. Celle-ci détermine les rétributions de chaque intervenant en fonction des stratégies envisagées. Nous définissons une stratégie comme l'ensemble des décisions prises en fonction d'hypothèses de comportement des personnes intéressées dans une conjoncture déterminée. Le concept de stratégie est une des pierres angulaires de ce travail et fera l'objet d'un développement particulier au sein du chapitre suivant.

		Agent B	
		Coopération	Non-Coopération
Agent A	Coopération	(C, C)	(D, T)
	Non-Coopération	(T, D)	(P, P)

Figure 1: Matrice des Payoffs

Il importe de souligner que les différents participants décident ou non de coopérer de façon ***simultanée***[2]. Cette hypothèse est, sauf indication explicite, également en vigueur au sein des autres variantes présentées ultérieurement. Nous supposons également qu'à priori la matrice des payoffs est parfaitement connue par l'ensemble des interlocuteurs. Elle fait partie de la base de '*connaissances publiques*' et n'évolue pas.

Par ailleurs, pour qu'un dilemme prenne naissance, il faut que la tentation de ne pas coopérer offre certains avantages par rapport à la coopération, ce qui implique que $T > C$. La coopération doit elle-même avoir un salaire plus élevé que celui de la défection ($C > P$). En outre, lorsqu'un des interlocuteurs trahit, l'autre perçoit un salaire plus élevé s'il trahit à son tour, ce qui se traduit par $P > D$. Ceci nous permet d'écrire l'ordre des inégalités suivant:

$$T > C > P > D \qquad (1)$$

Enfin, nous prenons comme hypothèses que la confrontation ne comporte qu'un nombre limité de coups, que le facteur à optimiser concerne le score individuel et qu'il n'y a pas de communication entre agents. Dans un tel cas, les inéquations ont comme conséquence que quelle que soit la stratégie de l'autre, il est toujours plus avantageux de trahir que de coopérer. Afin d'éviter toute collusion entre les acteurs, d'une part, et, de ne pas accorder trop d'importance à la trahison par rapport à la coopération, d'autre part, nous posons que:

$$2 * C > T + D \qquad (2)$$

Cette hypothèse sera remise en question lorsque nous étudierons le cas du retour d'ascenseur. Pour les différents paramètres, nous prendrons les valeurs classiquement envisagées suivantes $T = 5$, $C = 3$, $P = 1$ et $D = 0$.

Sur base de ces éléments, il nous est possible de constater que le résultat obtenu par un participant à l'issue d'un duel dépend non seulement de la stratégie qu'il a adoptée mais aussi de celle mise en application par son adversaire[3].

---

2 En d'autres termes, quand un participant pose un acte, personne ne connaît quelle sera la décision de l'adversaire.

3 Il est plus exact de dire que la stratégie d'un agent dépend de ses convictions quant à la tactique de l'autre agent.

Comme nous pouvons le remarquer, le ***paradoxe*** résultant de ce type de problème est que ***s'il est vrai que collectivement chaque participant a intérêt à coopérer; il s'avère plus avantageux sur le plan individuel de trahir***[4].

Dans le cas du dilemme du prisonnier à une seule période, il est évident que la stratégie dominante de chaque joueur est la non-coopération[5]. En effet, sur base de l'inéquation (1), le choix le plus rationnel est la trahison étant donné qu'elle garantit au moins un payoff de 1 alors que la coopération peut engendrer un rendement égal à 0. Il s'agit d'un équilibre de Nash (c'est-à-dire qu'aucun agent n'a intérêt à modifier son comportement car il y perdrait). On dit que l'action de coopération est strictement dominée par la non-coopération.

Il en va de même lorsque le nombre d'interactions est connu à l'avance. Pour s'en convaincre, il suffit d'adopter un raisonnement par induction. En effet, chaque agent connaît la fin du jeu, et a intérêt à ne pas coopérer durant la dernière période car il n'y a plus de perspective de gain dans le futur. Etant donné qu'il n'y aura pas collaboration en dernière période, ils ne coopéreront pas non plus en avant-dernière période. Nous pouvons continuer de la sorte jusqu'à la première période. Cette configuration porte le nom de '*paradoxe du pendu*'. Nous pouvons en conclure que ***la coopération implique nécessairement la présence de stimuli***.

Pour clarifier les différentes hypothèses sous-jacentes au dilemme du prisonnier et à ses variantes, nous émettons les propositions suivantes:

1. Les agents sont *homo-économicus*.
2. Il n'y a pas de communication entre les agents. En d'autres termes, le *comportement* d'une personne est *indépendant* de celui posé par son interlocuteur au cours de la même période. Néanmoins, leurs payoffs respectifs dépendent des actions interactives.
3. Les agents ont une *parfaite connaissance des payoffs*.

---

4 Les exemples ne manquent pas en la matière. Citons notamment celui de la '*tragédie des communs*'. Il s'agit de l'accès à un bien indivisible mis à la disposition de l'ensemble de la communauté.

5 A ce sujet, Delahaye et Mathieu (1995) font remarquer que '*if the game is only played once, then each player gets a higher payoff from defecting than from cooperating, regardless of what the other player does*'.

4. Le *jeu* est parfaitement *symétrique*[6].

En outre, au sein de nos développements, nous avons pu constater que:

1. Le dilemme réside dans le fait que:
   - du point de vue collectif, on a intérêt à coopérer.
   - du point de vue individuel, on a intérêt à trahir.
2. La ***perspective collective*** engendre un ***cercle vertueux*** car elle renforce la coopération au fil du temps. Par contre, la ***perspective individuelle*** engendre un ***cercle vicieux*** car elle renforce la non-coopération.

Article II: Le dilemme répété du prisonnier

Par rapport aux hypothèses reprises ci-dessus, le dilemme répété du prisonnier se différentie par le fait qu'aucun participant ne connaît à l'avance le terme du jeu. Autrement dit, les joueurs peuvent répéter un même jeu un nombre indéfini de fois. Dès lors les joueurs peuvent décider de coopérer parce qu'ils espèrent que la coopération continuera indéfiniment dans le futur. Dans un jeu répété, chaque joueur se bâtit une certaine '*réputation*'. C'est sur base de celle-ci qu'il peut avoir la possibilité d'encourager l'autre joueur à coopérer ou non. On peut dès lors dire qu'un individu peut avoir la possibilité d'influencer, dans une certaine mesure, l'autre agent dans son comportement futur.

Dans une telle configuration, chaque intervenant a la possibilité de '*sanctionner*' son adversaire pour avoir adopté un '*mauvais*' comportement lors de la période précédente. Par '*mauvais*' comportement, nous envisageons le fait qu'un individu agisse de façon égoïste, et par conséquent trahisse.

A cet égard, nous pouvons mentionner A. Kehagias (1998) qui souligne que '... *the individual-rational way to play PD* [Prisoner's Dilemma] *is for both players to defect. This happens in the first round of the PD and consequently both players are punished. This may convince then to be a little more co-operative in the next round* (page 9)'.

---

[6] Le logiciel élaboré dans le cadre du mémoire permet qu'il n'en soit pas toujours ainsi. Cela signifie que nous acceptons des revenus différents pour des situations similaires en fonction de l'agent. Ce choix est lié à notre souci de présenter un logiciel permettant de servir de base à l'analyse, mais aussi d'ébaucher des simulations dont l'évolution de chaque stratégie est fonction d'éléments externes à celle-ci.

Article III: Dilemme avec possibilité de retour d'ascenseur

Le '*retour d'ascenseur*' est une forme d'altruisme réciproque. Celui qui coopère en premier tandis que l'autre trahit prend des risques mais il attend un '*feedback*' durant la prochaine période. Dans ce contexte, les termes '*trahir*' et '*coopérer*' ne sont plus à prendre au sens propre. Une telle situation peut être qualifiée de '*méta-coopération*'. La défiance initiale des protagonistes cède alors lentement la place à la confiance réciproque. Beaucoup de conflits ardus ont été résolus par une évolution lente où chaque geste d'une partie en direction de l'autre sous-entendait une réponse positive de l'adversaire en contre-partie.

En d'autres termes, une situation de méta-coopération signifie '*si j'adopte un comportement qui lui est particulièrement avantageux lors de cette période, alors il est prié d'adopter un comportement qui me soit favorable lors de la période suivante*'. Dans cette configuration, la deuxième équation devient: $C < \left(\frac{D+T}{2}\right)$. Dans la suite de ce travail, lorsque nous raisonnerons dans le cadre du '*retour d'ascenseur*', les valeurs adoptées pour les différents paramètres seront: T = 8, C = 3, P = 1 et D = 0.

Du fait que les premières inéquations sont toujours d'actualité, le dilemme subsiste dans cette variante. En effet, l'intérêt collectif s'oppose toujours à l'intérêt individuel. On peut dès à présent constater que, dans ce cas, deux niveaux de coopération sont maintenant envisageables, nous les explicitons ci-dessous:

1. ***La coopération classique***: les deux intervenants coopèrent à chaque période. En d'autres termes, les deux agents jouent simultanément la coopération[7]. On a continuellement (C, C), (C, C), (C, C), (C, C), (C, C),... On peut dès lors parler d'une coopération de premier niveau, ou, comme la qualifient certains auteurs de '*non aggression pact*'.

---

[7] Les deux agents sont donc en '*phase*'.

2. ***La méta-coopération***: afin d'obtenir les meilleurs résultats possibles, les deux intervenants peuvent alterner à tour de rôle coopération et trahison. En d'autres termes, ils doivent tenter d'arriver à la situation suivante: (D, T), (T, D), (D, T), (T, D), (D, T), (T, D),... Par conséquent, les agents doivent créer un contexte de confiance permettant de converger vers une situation où chaque stratégie accepte alternativement de recevoir un rendement faible puis un rendement élevé. Dans ce type de coopération, il importe de coordonner les stratégies de telle sorte qu'elles se trouvent dans une situation de déphasage. Etant donné que tout joueur ignore lorsqu'il prend sa décision quelle sera celle de son adversaire (autrement dit, si aucune entente préalable n'est possible), il y a dans cette alternative une prise de risque importante.

D'emblée on peut constater qu'en plus du paradoxe '*classique*' du dilemme du prisonnier, s'ajoute celui lié au '*retour d'ascenseur*'. Celui-ci est le suivant: ***Comment interpréter le fait qu'une personne décide de trahir?*** Refuse-t-elle toute forme de coopération ou essaie-t-elle d'instaurer une forme de méta-coopération.

Selon les études menées par Delahaye et Mathieu (1996 a), '*only probabilistic strategies can make a high score when they play against themselves*[8]'. Ces auteurs soulignent également que '*more complex dynamics can appear (the 'edge of chaos'?) as soon as three strategies are confronted*', et font remarquer que '*This creates an iterated game much more difficult to analyse than the classical Iterated Prisoner's Dilemma*'. En outre, il ressort de leurs travaux que ce type de jeu arrive à une situation où '*a collectively rational strategy is defined as a strategy which is able to obtain a maximum score against itself even if the rewards are not equally distributed between its representatives*' (Delahaye et Mathieu (1996 c)).

---

[8] Mr. Mathieu explique ceci par le fait que '*pour obtenir une bonne stratégie au retour d'ascenceur, il faut, ..., que la stratégie soit capable de tirer au maximum profit d'une rencontre avec ses congénères. ...* [La difficulté réside dans le fait qu'il y a *opposition de phase avec son congénère. Il s'ensuit que seules les stratégies non déterministes au départ sont à même d'arriver à cette situation.*' (Mathieu (1996 a) page 9).

Article IV: Les jeux asynchrones (ou séquentiels)

A la différence des alternatives présentées ci-dessus, nous considérons le scénario selon lequel un agent choisit sa stratégie en connaissant celle de l'autre. En d'autres termes, un joueur pose un acte en connaissant la réponse de l'autre joueur. La matrice des payoffs deux fois deux devient dès lors une matrice quatre fois quatre.

			Agent B		
		L'agent B joue en premier		***L'agent B joue en deuxième***	
		Coopération	Non-Coopération	Coopération	Non-Coopération
L'agent A joue en premier	Coopération			(C_1, C_2)	(D_1, T_2)
	Non-Coopération			(T_1, D_2)	(P_1, P_2)
Agent A ***L'agent A joue en deuxième***	Coopération	(C_2, C_1)	(D_2, T_1)		
	Non-Coopération	(T_2, D_1)	(P_2, P_1)		

représente des possibilités non pertinentes dans le cas du dilemme du prisonnier avec renoncement.

Figure 2: Matrice des Payoffs dans le cas asynchrone

Dans chaque cas, il importe peu que A joue en premier ou en second. Les contraintes présentées précédemment restent les mêmes. On retrouve ce canevas dans la vie de tous les jours. En effet, lorsqu'une entreprise décide de s'implanter sur un marché, les entreprises déjà présentes disposent d'un avantage évident, celui de l'expérience accumulée. Elles adaptent leur stratégie commerciale en fonction de celle adoptée par leur nouveau concurrent.

Article V: Le renoncement au sein du dilemme du prisonnier[9]

Dans cette variante, si l'un des participants considère que la confrontation avec un adversaire ne lui est pas bénéfique, il peut décider de se retirer du jeu. Il s'agit d'un '*renoncement définitif* [10]', c'est-à-dire que la personne qui préfère ne plus faire partie du jeu ne peut revenir sur sa décision par la suite.

Le rendement de la renonciation (R) doit être attractif. C'est la raison pour laquelle le salaire de celui qui s'est retiré doit être plus élevé que ce qu'il toucherait si l'un et l'autre trahissaient ($R > P$) mais doit être inférieur au payoff de coopération mutuelle ($R < C$). La première inégalité peut se traduire comme suit: il faut qu'il existe un stimulant à sortir du jeu sinon personne ne choisira cette alternative. Par ailleurs, le salaire de la renonciation ne peut être trop important car sinon il n'y aurait aucun attrait à coopérer. C'est pourquoi nous fixons la valeur de R à 2. Ajoutons enfin que si une personne décide de se retirer du jeu, cette décision aura un impact sur le comportement de l'autre joueur puisque que ce dernier sera de facto considéré comme se retirant également du jeu.

Par ailleurs, dans cette variante, trois alternatives s'offrent à tout participant, à savoir: coopérer, ne pas coopérer ou renoncer. Il est faux de considérer que les deux dernières attitudes sont équivalentes, et ce pour deux raisons. D'une part, comme nous venons de le souligner, en raison du niveau différent des payoffs, et d'autre part du fait qu'alors que le renoncement est définitif, par contre après avoir opté pour la non-coopération, un joueur peut vouloir recréer une situation de confiance.

---

9 L'idée sous-jacente à la renonciation est que lorsqu'une personne décide de se retrancher dans sa solitude, elle obtient un résultat plus élevé que lorsqu'elle est en conflit avec autrui.

10 En d'autres termes, le fait de renoncer à participer à un jeu lors d'une période est un acte irréversible. Dans ce cas, chaque joueur impliqué dans l'interaction avec la personne qui s'est retirée reçoit un payoff de deux. Ceci est une hypothèse simplificatrice.

Section II: Types de confrontations de stratégies

Après avoir défini les différentes variantes du dilemme du prisonnier, il nous faut encore déterminer les méthodes permettant d'apprécier la ***robustesse*** d'une stratégie. Une telle expression mesure l'ensemble des payoffs récoltés à l'issue des confrontations. A ce sujet, la valeur d'une stratégie est définie eu égard aux résultats engendrés par rapport aux coups joués par le joueur adverse. Nous abordons successivement le '*single confrontation*', le '*round-robin tournament*', l''*ecological evolution*', l'introduction de certaines perturbations via l'intrusion aléatoire de stratégies au cours de la confrontation et le choix aléatoire des stratégies initiales.

Article I: La confrontation simple

Durant un certain nombre de rounds défini préalablement, une stratégie est confrontée à une autre. Au terme de la confrontation, les résultats obtenus par chacune d'entre elles sont additionnés, et la stratégie victorieuse est celle qui a accumulé le plus de points. Ce type de confrontation ne permet pas de mettre en évidence les caractéristiques d'une bonne stratégie. En effet, une stratégie comme 'MECHANT[11]', remporte toutes les confrontations par rapport aux autres stratégies mais réalise toutefois des résultats fort moyens quand elle est confrontée à d'autres plus performantes. C'est la raison pour laquelle, pour avoir des résultats intéressants, il semble indiqué d'opter pour des confrontations généralisées.

Article II: La confrontation généralisée

On considère 'x' stratégies différentes. Chacune est opposée à toutes les autres, en ce compris elle-même et durant un nombre fixe de fois. Son payoff final est la somme de tous les payoffs réalisés lors de chaque confrontation. In fine, la robustesse d'une stratégie nous est donnée par le rang qu'elle occupe à la fin du tournoi. Ainsi la stratégie 'MECHANT' remporte ses confrontations contre toute autre stratégie sans pour autant réaliser le meilleur résultat accumulé.

---

11 On ne coopère jamais quel que soit le comportement de l'adversaire.

Article III: La compétition écologique

Dans cette partie, on considère une population, c'est à dire un ensemble d'individus, qui interagissent les uns avec les autres en utilisant des stratégies diverses. Au début de la simulation, on détermine dans quelle proportion chaque stratégie est présente au sein de la population[12]. Un tournoi similaire à celui présenté ci-dessus est alors réalisé. La différence avec celui-ci réside dans le fait que la proportion d'une stratégie en période '*n*' (n > 1) est proportionnelle au résultat obtenu lors de la période n-1. Ainsi, la population des '*mauvaises*' stratégies[13] décroît alors que les '*bonnes*' stratégies voient le nombre de leurs représentants augmenté. Dans '*Introduction: Symposium on Evolutionary Game Theory*', G. Mailath (1992) relève que '*strategies that are «good» replies to the distribution of actions chosen by the current population will be played by a larger fraction of the population in the next period. Thus, players learn from the experience of the population*' (page 259). Pour leur part, Binmore et Samuelson (1992) définissent la théorie des jeux '*évolutionnaires*' de la façon suivante: '*Nature chooses its players from a population whose composition changes over time in function of the past*' (page 286).

Normalement, ce type de simulation est répété jusqu'au moment où il y a stabilisation[14] des différentes populations. Le calcul permettant d'évaluer la population d'une stratégie est le suivant[15]:

$$T_i^n = \sum_{j=1}^{q} \left(P_j^n \,.\, S_i^j \right) - S_i^i , \tag{3}$$

$$P_i^{n+1} = P^{n+1} \,.\, \left(\frac{P_i^n \,.\, T_i^n}{\sum_{j=1}^{q} T_j^n \,.\, P_j^n} \right), \text{ et} \tag{4}$$

$$P_p^{t_d} = \sum_{i=1}^{n} P_{s_i}^{t_d} \tag{5}$$

---

12 L'absence d'une stratégie au sein de la population initiale est représentée par une proportion de 0%.

13 Nous entendons par ce terme les stratégies ayant abouti aux moins bons résultats.

14 Le terme de stabilisation marque le fait que l'ensemble des différentes stratégies ne se modifie plus.

15 Notons que ces équations sont basées sur l'hypothèse d'un affrontement entre stratégies non probabilistes. Si nous intégrons les stratégies probabilistes, cela entraîne un formalisme un peu plus complexe sans pour autant accroître la compréhension du problème.

où T_i^n représente le payoff obtenu par un individu appliquant la stratégie i à la n^{ime} période,

q est le nombre de stratégies différentes définies au début de la simulation,

P_j^n donne le nombre d'individus de la stratégie i à la n^{ime} période,

S_i^j représente le score obtenu par la stratégie i quand elle est confrontée à la stratégie j,

P^n concerne la population totale présente à la n^{ime} confrontation,

x donne le taux de croissance de la population[16], et

C représente la population de départ.

L'équation (3) peut s'interpréter de la manière suivante: le payoff d'un individu appliquant la stratégie 'i' est égal à l'ensemble des rémunérations qu'il reçoit quand il est confronté à une autre personne. Le nombre d'individus d'une stratégie à la période 'n' (n>1) est, quant à lui, défini par la proportion, au sein de la population totale à la période 'n', des rendements obtenus par la stratégie concernée par rapport au cumul des rendements obtenus. Ceci nous est indiqué par les formules (4) et (5).

Ce type de simulation, nous permet de déterminer les stratégies dites '***robustes***'. Il s'agit de stratégies dont le comportement peut être qualifié comme satisfaisant[17] dans un grand nombre de contextes écologiques.

Article IV: Le dilemme du prisonnier face à des intrusions aléatoires de stratégies

Dans cette version, certaines stratégies sont introduites, dans des proportions variables, au cours de la simulation. Pour ce, il faut utiliser un générateur aléatoire. Comme nous le verrons au chapitre suivant, une telle option permet de déterminer les stratégies évolutionnairement stables.

---

16 Il importe de remarquer que l'expression '$x = 0$' signifie qu'on raisonne à population constante (autrement dit, que le taux de croissance est égal à zéro).

17 Une stratégie a un comportement satisfaisant quand elle réalise un payoff final élevé.

Commentaires

Pour être complet, nous tenons à souligner que les différents commentaires développés ci-dessus ne prennent en considération que la confrontation de deux agents. Cependant, elles peuvent être généralisées au cas de N-joueurs[18]. Néanmoins, dans un but de simplification, et ce sans perte de généralité, nous ne raisonnons que dans le cas de deux joueurs.

Après cette présentation générale du dilemme du prisonnier et des types de confrontation de stratégies, nous abordons au chapitre suivant l'une des pierres angulaires de la théorie des jeux, à savoir le concept de stratégie.

---

[18] Pour de plus amples commentaires à cet égard, nous renvoyons le lecteur à l'article de A. Kehagias (1998).

Chapitre II: Quid des stratégies?

Introduction

Lorsqu'une personne décide de ne plus coopérer durant une période, faut-il pour autant écarter irrévocablement toute coopération future? Ou faut-il, au contraire, donner l'impression de ne pas l'avoir remarqué? N'existe-t-il pas entre ces deux comportements extrêmes, un '*juste milieu*' qui génère des résultats plus performants? Pour illustrer nos propos, nous pouvons envisager deux types d'individus. D'un côté ceux qui ne coopèrent jamais, et de l'autre ceux qui adoptent le comportement que leur adversaire avait précédemment et optent systématiquement pour la coopération en première période. Désignons respectivement les premiers par '*Méchant*' et les seconds par '*œil pour œil* '. S'il est vrai que '*Méchant*' bat individuellement chacun de ses adversaire[1], il obtient in fine, si on répète le jeu sur un grand nombre de périodes, un revenu accumulé moins élevé du fait qu'il n'a pas été capable d'établir une coopération mutuelle.

Ces différentes réflexions nous amènent à la question suivante: 'Comment faut-il réagir?' ou plus précisément 'Quelle stratégie est-il préférable d'adopter?'.

Ce chapitre n'a pas la prétention de définir la meilleure réaction à adopter. En effet, étant donné, d'une part, les situations fort disparates liées aux multiples configurations des confrontations possibles et, d'autre part, les buts propres que poursuit chaque individu (du style: maximiser son revenu, ne jamais avoir de salaire inférieur à son adversaire,...), un tel objectif est irréalisable. Par contre, nous souhaitons insister sur quelques points qui nous semblent importants dans le cas du dilemme du prisonnier.

Dans un premier temps, nous donnons une définition d'un des concepts essentiels de la théorie des jeux, à savoir: la notion de stratégie. A partir de là, nous articulons quelques-unes de ses caractéristiques. Enfin, nous terminons en illustrant nos propos par la présentation de plusieurs stratégies. De façon schématique, nous pouvons dire que nous adoptons la structure suivante:

Section I: Le concept de stratégie

Section II: Les caractéristiques d'une stratégie

Section III: Illustration à l'aide de quelques stratégies '*classiques*'

Avant d'aller plus en avant dans la matière, il importe de remarquer qu'on parlera fréquemment dans ce mémoire de type de stratégie. Cette expression comprend l'ensemble des individus au sein de la population ayant adopté un type bien déterminé de comportement.

Enfin, relevons également le fait que les éléments présentés dans ce chapitre sont destinés à susciter des pistes de réflexion permettant, dans le futur, d'aborder le domaine de l'Intelligence Artificielle.

---

[1] En d'autres termes, '*Méchant*' obtient un payoff au moins égal à celui de son adversaire.

Section I: Le concept de stratégie

Une stratégie peut être définie de la façon suivante: les agents sont des êtres autonomes doués d'un état mental. Chaque individu détermine les actes à poser, sur base de l'analyse de l'environnement dans lequel il se trouve, de façon à atteindre le meilleur niveau de satisfaction. L'analyse de cet environnement peut entraîner une modification de son état mental interne.

Mailath précise que 'a strategy is a programmed pattern of behaviour, not an object of choice' [Mailath, G. (1992), page 265]. Autrement dit, ***une stratégie est une procédure par laquelle on établit un ensemble systématique de réactions sur base des comportements des différents agents***, et non pas une matière sujette à options. En d'autres termes, on peut dire qu'une stratégie modélise le cheminement du raisonnement d'un individu, lui permettant d'aboutir à une décision.

Quelle que soit la variante du dilemme du prisonnier pour laquelle on opte, le jeu ainsi choisi n'est pas à somme nulle. En effet, l'ensemble des payoffs est réparti entre les différents intervenants en fonction des actions posées. Notons toutefois que le score global obtenu dépend de la capacité des différents participants de créer une situation de coopération (voire, dans certains cas, de méta-coopération).

En outre, il importe de remarquer qu'il existe une infinité de stratégies différentes sur une période déterminée. En effet, le fait que deux personnes adoptent exactement le même comportement n'implique pas pour autant qu'elles aient opté pour la même stratégie. Prenons trois individus A, B et C ayant choisi respectivement ***GENTILLE***, ***DONNANT-DONNANT*** et ***GRADUAL***[2]. On peut envisager une durée de confrontation aussi longue qu'on veut, ils auront toujours la même attitude, à savoir la coopération, alors que leurs caractéristiques propres, et donc leurs motivations, sont fort différentes.

---

[2] Etant donné leur coté typique, ces différentes stratégies, quoiqu'assez explicites, seront sommairement présentées dans la suite de ce chapitre.

Section II: Les caractéristiques d'une stratégie?

La terminologie utilisée ci-dessous est principalement basée sur celle définie par R. Axelrod (1984). On retrouve fréquemment les mêmes éléments caractéristiques au sein des stratégies. Parmi ceux-ci, nous trouvons notamment la *sérénité*, la *réaction graduelle*, le *pardon*, le *simplicité* et le *seuil*. A ces différentes caractéristiques, reprises également par messieurs Delahaye et Mathieu, permettant à une stratégie de perdurer au sein du dilemme répété du prisonnier[3], nous adjoindrons les *stratégies évolutionnairement stables*.

La sérénité: les stratégies ont comme souci de ne pas prendre l'initiative de la trahison, de ne pas être agressif. On parle également de stratégies '*gentille*'.

La réaction graduelle: elle a pour objet de punir immédiatement l'autre agent quand celui-ci a trahi précédemment. Au plus l'adversaire adopte un comportement non-coopératif, au plus il est nécessaire de réagir durement (d'où le terme de '*graduelle*'). Cette menace implicite peut inciter les autres agents à coopérer. On retrouve la notion '*d'un prêté pour un rendu*'. Dans cette optique, il importe de prendre en considération les choix stratégiques posés précédemment par l'adversaire.

Le pardon: ces stratégies sont déterminées par le fait qu'après avoir puni l'autre agent coupable d'une trahison, la riposte doit pouvoir prendre fin dans une période relativement courte de façon à recréer les conditions de coopération.

La simplicité: l'idée maîtresse à la base de cette attitude est qu'une stratégie doit pouvoir être intelligible. Messieurs Delahaye et Mathieu (1995) font remarquer que '*Too much complexity can appear total chaos. If you are using a strategy which appears random, then you also appear unresponsive to the other player. If you are unresponsive, then the other player has no incentive to cooperate with you. So being so complex as to be incomprehensible is very dangerous.... The trick is to encourage the cooperation*'. Binmore et Samuelson (1992) ajoutent toutefois que '*... evolutionary process will tend to select against complex machines*[4] *unless their complexity generates real gains compared with less complex machines*'.

---

[3] Cfr. Delahaye et Mathieu 1996 b.

[4] Il faut comprendre le terme '*machines*' comme étant une stratégie particulière.

Le seuil: dans la variante du dilemme du prisonnier avec renoncement, il représente le cas limite au delà duquel le renoncement devient souhaitable.

Enfin, l'expression ***stratégies évolutionnairement stables*** regroupe l'ensemble des stratégies qui malgré une perturbation ε à un moment donné connaissent une évolution stable au fil du temps. La perturbation dont il est question concerne un ensemble quelconque de stratégies choisies aléatoirement, de valeur maximale ε, injectées au sein de la population. Il faut remarquer qu'une stratégie évolutionnairement stable peut, lorsqu'on introduit d'autres stratégies, comporter une phase de déstabilisation passagère avant de retrouver une stabilité.

Formellement, une stratégie est évolutionnairement stable lorsqu'elle obéit à la règle suivante:

- si 'U' représente les préférences lexicographiques 'a' et 'b' concernent les stratégies envisagées, et
- si $\forall$ 'b', on a:

$$\begin{cases} U(a,a) > U(b,a) \\ ou \\ U(a,a) = U(b,a), \text{ et } U(b,b) > U(a,b) \end{cases}$$

alors 'a' est une stratégie évolutionnairement stable car elle est la meilleure réplique par rapport à elle-même.

Comme le font remarquer Binmore et Samuelson (1992), une caractéristique des stratégies évolutionnairement stables est que '*when mutations come at sufficiently infrequent intervals that the system can fully recover from the effects of repelling one mutation before the next appears*' (page 292).

Section III: Illustration à l'aide de quelques stratégies 'classiques'

Dans cette section, nous présentons quelques stratégies de base sous leur forme '*générique*'. Chacune d'ente elles peuvent être reformulées de façon mathématique, à l'aide de la formule ci-dessous. Celle-ci se présente de la façon suivante:

$$f\left(Prob\_Coop, C\_A\_P, N\_Trahison, Période, 2^{ième}\ Per,\ 3^{ième}\ Per, C\_P\right) = C, T\ ou\ R$$

les arguments entre parenthèses pouvant avoir une influence positive (+), négative(-) ou nulle (0) et, où C = coopérer, T = trahir, et R = renoncer.

La signification de chacun des différents paramètres est la suivante:

- ***Prob_Coop*** indique la probabilité avec laquelle l'agent a décidé de coopérer. Cette valeur est comprise entre zéro et l'unité.
- ***C_A_P*** fournit le comportement adopté par l'adversaire lors de la période précédente. Cette valeur peut donc être C (Coopérer), T (Trahir), et R (Renoncer).
- ***N_Trahison*** reflète le nombre de fois que l'adversaire a trahi lors des périodes précédentes.
- ***Période*** indique la période dans laquelle on se trouve.
- $2^{ième}$***Per*** donne le comportement de l'adversaire lors de la deuxième période par rapport à l'origine de la confrontation.
- $3^{ième}$***Per*** donne le comportement de l'adversaire lors de la troisième période par rapport à l'origine de la confrontation.
- ***C_P*** fournit le comportement de la stratégie analysée lors de la période précédente.

Comme le laisse prévoir la formule ci-dessus, une stratégie peut prendre en considération des critères très variés. Par ailleurs, ceux-ci peuvent être sans aucune relation entre eux.

Article I: Présentation de stratégies de base dans le cas du dilemme du prisonnier classique

ALEATOIRE [A, 1-A]

On coopère avec une probabilité '*a*'.

DONNANT-DONNANT

Lors de la première confrontation, on coopère. Par la suite, on calque son attitude sur adoptée par l'adversaire durant la période précédente.

GENTILLE

On coopère toujours quel que soit le comportement de l'adversaire.

GRADUAL

Cette stratégie est une généralisation de ***DONNANT-DONNANT***. En effet, lors la '$n^{ème}$' défection de l'adversaire, on réagit en trahissant '*n*' fois puis on revient à plus de modération en coopérant deux fois[5].

MAJO-MOU

Durant la première période on coopère puis on adopte le comportement appliqué le plus fréquemment par le concurrent. En cas d'égalité, on coopère.

MAJO-DUR

Durant la première période on coopère puis on adopte le comportement appliqué le plus fréquemment par le concurrent. En cas d'égalité, on trahit.

MECHANT

On trahit toujours quel que soit le comportement de l'adversaire.

---

5 La différence fondamentale de cette stratégie avec celle du ***DONNANT-DONNANT*** est qu'on dispose d'une mémoire du jeu depuis le début.

MEFIANT

Lors de la première confrontation on trahit. Par la suite, on adopte la même attitude que celle appliquée par l'adversaire lors de la période précédente.

PAVLOV

Lors de la première confrontation on coopère. Par la suite, on coopère seulement si les deux joueurs ont agi de la même façon durant la période précédente (Cette stratégie a été étudiée par M. Nowak et K. Sigmund (1993), '*A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game*', Nature, 364: 56-58).

PER-GENTIL

On joue périodiquement Coopération, Coopération, puis Trahison.

PER-MECHANT

On joue périodiquement Trahison, Trahison, puis Coopération.

PROBER

On commence par jouer Coopération, Trahison puis Trahison. Ensuite, si l'adversaire coopère lors des deuxième et troisième période on trahit toujours, sinon on joue la stratégie du ***DONNANT-DONNANT***.

RANCUNIERE (SPITE)

On trahit tout le temps dès que l'adversaire trahit, sinon on coopère.

REASON [A, 1-A]

On coopère avec une probabilité '*a*'. Lors de la première confrontation et jusqu'au moment où il y a déphasage, on choisit la coopération. Lorsqu'un déphasage[6] apparaît on choisit alternativement de trahir puis de coopérer.

---

[6] Il s'agit d'une situation où un individu décide de coopérer alors que son adversaire a choisi de trahir. A la période suivante, les deux interlocuteurs adoptent chacun un comportement inverse. Cet état se reproduit de période en période.

REASON DONNANT-DONNANT [A, 1-A]

On coopère avec une probabilité '*a*'. Lors de la première confrontation et jusqu'au moment où il y a déphasage, on choisit la coopération. Lorsqu'un déphasage apparaît on choisit d'adopter la stratégie ***DONNANT-DONNANT*** (autrement dit, on calque son attitude sur celle de l'autre joueur durant la période précédente).

SONDEUR

On joue Trahison, Coopération et Coopération. Ensuite, si l'adversaire coopère lors des deuxième et troisième période on trahit toujours, sinon on joue la stratégie du ***DONNANT-DONNANT***.

Article II: Présentation de stratégies dans le cas du dilemme du prisonnier avec renoncement

Les différentes stratégies proposées dans le cadre du dilemme du prisonnier classique (cfr. supra) sont évidemment encore applicables au sein de cette variante. En effet, chacune d'entre elles écarte de façon implicite l'alternative du renoncement.

DIFFICILE

On trahit si l'autre coopère, par contre si ce dernier trahit, on décide de se retirer du jeu.

DONNANT-DONNANT AVEC SEUIL

On joue ***DONNANT-DONNANT*** ; d'autre part, tous les cinq coups on calcule son payoff moyen par période. Si celui-ci est inférieur à une certaine valeur fixée sur base de la matrice des payoffs alors on se retire.

FIRST

On coopère durant les 20 premières périodes. Au terme de celles-ci, on calcule la moyenne des derniers payoffs. Si celle-ci est inférieure à 1.5 alors on se retire du jeu. Par ailleurs, chaque fois que l'autre joueur trahit à un moment qui n'est pas justifié par une réaction de riposte, on modifie la stratégie dans un sens analogue. La $n^{ième}$ période de riposte est la succession de $\frac{n.(n+1)}{2}$ défections suivies de deux coopérations.

SECOND

On échafaude successivement une stratégie en fonction de cinq ***DONNANT-DONNANT***, ***GENTILLE***, ***RANCUNIERE*** et ***PER-GENTIL***. Par la suite on calcule le rendement obtenu durant les quatre derniers mouvements de chaque stratégie. Si le meilleur résultat est inférieur à 1.5 alors on renonce à participer au jeu, sinon on joue douze fois d'affilée la stratégie ayant obtenu le meilleur résultat. Ensuite on recalcule la moyenne et on recommence comme précédemment.

TESTER 4

Durant les quatre premières périodes, on teste son adversaire en jouant Coopération, Coopération, Trahison et Trahison. Si l'autre joueur a trahi au moins trois fois, alors on renonce sinon on coopère jusqu'à la fin.

Commentaires

Après avoir présenté différentes stratégies, on se rend compte qu'il existe une infinité de stratégies, même s'il est vrai que certaines sont relativement proches. Par ailleurs, à l'issue des considérations sur les caractéristiques permettant de mieux cerner le concept de stratégie, nous aurions pu émettre quelques commentaires relatifs aux payoffs. En effet, nous aurions pu nous demander quelle auraient été les conséquences si un individu connaissait l'ensemble des payoffs associés à chacune des personnes composant la population. Nous nous contentons de noter à cet égard que les différentes stratégies que nous avons présentées ne prennent pas cet élément dans leur processus décisionnel. En d'autres termes, nous avons émis l'hypothèse selon laquelle le comportement des individus est le même qu'ils connaissent ou non chacun des payoffs. Néanmoins, le problème des revenus n'est pas plus amplement abordé dans le cadre de ce mémoire, et ce pour deux raisons: d'une part, nous avons émis l'hypothèse que les agents étaient homo-économicus[7]; et, d'autre part, ce point sort quel que peu du cadre de ce mémoire. Néanmoins, il importe de relever qu'il s'agit d'une piste intéressante pouvant être reprise dans le cadre d'extension possible dans un développement ultérieur.

Nous sommes maintenant en mesure de concevoir un programme susceptible de simuler la confrontation entre stratégies. L'objectif du chapitre suivant est de présenter les principaux éléments constitutifs d'un tel programme.

---

[7] Autrement dit, ils ne visent qu'à maximiser leurs revenus individuels. Dans cette perspective, les joueurs ont uniquement intérêt à isoler les succès possibles et non connaître les raisons de ces succès.

Chapitre III: Spécification des algorithmes et description des données

Introduction

Dans ce chapitre, nous décrivons les différents éléments qui représentent, à nos yeux, la clé de voûte du programme. A cet égard, ce chapitre est scindé en quatre parties. D'une part, nous exposons la procédure adoptée de façon à créer une structure de données qui soit la plus économique possible en termes d'espace mémoire et nous donnons également l'architecture logique de notre logociel. D'autre part, nous abordons les principaux algorithmes utilisés au sein du logiciel. Dans un troisième temps, nous exposons la '***règle d'évolution***' de la population considérée. Nous terminons en présentant les éléments à modifier de façon à implémenter de nouvelles stratégies et de nouvelles règles d'évolution.

Dans un souci de clarté, nous définissons d'abord les structures de données de façon rigoureuse puis les visualisons à l'aide d'un diagramme. En ce qui concerne les algorithmes, nous avons opté pour un pseudo-langage dans le but de rester indépendant de tout langage de programmation et d'accroître la lisibilité. Celui-ci permet d'exprimer sans ambiguïté le schéma de résolution. Ses caractéristiques sont, tout comme en Pascal, l'aspect structuré des traitements et des descriptions de données. Par ailleurs, les mots clés sont exprimés en français. Enfin, chaque bloc logique est indenté, et clairement délimité.

En outre, au risque de nous répéter, nous tenons à insister sur le fait que, tant pour les structures de données que pour les algorithmes, nous n'abordons que les éléments essentiels pour la bonne compréhension du fonctionnement du logiciel. Ce chapitre n'est cependant pas superflu car il s'inscrit dans le prolongement direct du but poursuivi dans ce mémoire, à savoir: le développement d'un logiciel de base qui fera l'objet ultérieurement d'ajouts de fonctionnalités nouvelles.

Section I: Les structures de données et l'architecture logique

Dans un premier temps, nous exposons de façon formelle les types de données que nous avons définis. Par la suite, nous présentons graphiquement les principaux éléments que nous utilisons au travers du programme dans le but de sauver ou de retrouver certaines informations. Nous terminons cette section en donnant l'architecture logique de notre logiciel.

Article I: Définition des structures de données

Point I: Les structures générales

Les principales structures sont au nombre de trois. Il s'agit d'une part de '*card*', de '*type_agent*', et de '*info*'. La première nous indique le comportement adopté par une personne. Par contre, '*type_agent*' nous renseigne sur la valeur des payoffs des individus représentant un type de stratégie. Ce champ nous permet de différencier trois niveaux de payoffs. Il importe de souligner que le fait d'admettre des niveaux de payoffs différents implique qu'on considère qu'***il existe des raisons de croissance exogène à la stratégie***. Enfin, nous avons défini l'union '*info*'. Cette dernière renferme les différents types de paramètres pouvant être transmis aux fonctions exprimant une stratégie particulière.

```
typedef enum {Cooperation, Defection, Abandon}card;

typedef enum {Val_Type, Agent_A, Agent_B} type_agent;

union info{
        float prob;
        card Last_Behaviour_Opp;
        ranc info_ranc;
        Nbre_Trahison Info_Nbre_Trahison;
        Sondage Info_Sondage;
        Behavior_Two_Last_Periods TFT_Dur;
        };
```

Point II: Les structures particulières

Par '*structures particulières*', nous entendons les structures que nous utilisons pour la sauvegarde ou la restauration des informations nécessaires pour l'exécution du logiciel. Celles-ci peuvent être classifiées en deux parties. D'une part celles qui n'évoluent pas au fil du temps et de l'autre celles qui sont régulièrement modifiées.

Pour les premières, nous avons considéré qu'une liste simplement chaînée était amplement suffisante. Il s'agit principalement de la structure '*info_init_strat*'. En ce qui concerne les structures qui subissent des évolutions, nous avons opté pour l'utilisation de listes doublement chaînées. Ce choix permet de simplifier la tâche lors de la navigation au sein des structures '*Info_Gen*' et '*Info_Indiv*'.

'*Info_init_strat*' a pour objet de mémoriser l'ensemble des informations propres à un type de stratégie sans prise en compte des autres stratégies auxquelles elle est confrontée.

```
typedef struct Info_Init_Strat_Type{
   ptr_fct Adr; /* 'Adr' pointe vers l'adresse où la stratégie est définie. */
   type_agent Agent; /* Indique s'il s'agit d'un agent de type A ou B, ou si on utilise les valeurs
         typiques. */
   unsigned int N_Strat; /* Indique le n° du type de stratégie dont il est question. */
   unsigned int Nb_Instance; /* 'Nb_Instance' indique le nombre d'instances de la stratégie
         concernée. */
   unsigned int Proba; /* 'Proba' nous informe sur le fait s'il s'agit d'une stratégie probabiliste ou
         non. Les stratégies probabilistes peuvent avoir une valeur comprise entre 0 et 100
         (étant donné qu'on exprime le fait qu'une stratégie a tendance à coopérer en
         pourcentage). Par défaut, ce champ reçoit, pour les stratégies probabilistes, 50
         pour valeur. Les stratégies non probabilistes ont comme valeur pour ce champ 200
         (de la sorte aucune confusion n'est possible). */
   int Tx_Croiss_Pop; /* Par défaut, on considère qu'une stragtégie n'a pas de croissance. La
         raison de ce choix s'explique par le fait que la survie d'une stratégie dépend de
         justifications endogènes. Rem: On considère que cette valeur doit être comprise
         entre -100 (on ne peut supprimer plus d'individus que la population totale de la
         stratégie concernée n'en possède) et 100. */
   unsigned int Payoff_acc_Preced; /* Ce champ nous sert pour enregistrer les payoffs
         accumulés précédemment par l'ensemble des individus de cette stratégie. */
   unsigned int Payoff_acc_Actuel; /* Ce champ nous sert pour enregistrer les payoffs accumulés
         par l'ensemble des individus de la stratégie en question. */
```

```
/* Les deux derniers paramètres permettent de définir les caractéristiques des traits que nous
allons tracer. */
TColor Couleur;
unsigned int Epaisseur;
struct Info_Init_Strat_Type *Next;
}info_init_strat;
```

'*Info_Gen*' permet, quant à elle, d'enregistrer les informations nécessaires pour le fonctionnement d'une stratégie. En d'autres termes, on se sert de cette structure pour disposer de certains renseignements relatifs au comportement de l'adversaire, et pour mémoriser l'action que pose l'agent pour la période durant laquelle il doit décider.

```
typedef struct Info_Gen_Type{
/* Informations à enregistrer pour le fonctionnement d'une stratégie. On utilise également cette
structure pour mémoriser l'action posée par l'agent durant la période de décision (il s'agit du
champ 'Action'). */
   struct Info_Gen_Type *Previous;
   unsigned int Ident; /* Identifiant de l'information stockée p.r. à l'adversaire. Il s'agit d'un type
         de stratégie ou de l'identifiant d'un agent particulier. */
   info Info_Strat_Adv; /* Information stockée p.r. à l'adversaire en ce qui concerne les périodes
         antérieures. */
   card Action; /* Action posée par l'individu pour la nouvelle période. */
   struct Info_Gen_Type *Next;
   }Info_Gen;
```

Enfin, nous utilisons la structure '*info_indiv*' afin de créer et de faire évoluer la population. Les champs '*Ident_Strat*' et '*Ident_Indiv*' nous permettent non seulement d'identifier un individu au sein de cette liste, mais servent également de clé pour disposer de certaines informations localisées à l'intérieur d'autres types de structures.

```
typedef struct Info_Indiv_Type{
   struct Info_Indiv_Type *Previous;
   unsigned int Ident_Strat;
   unsigned int Ident_Indiv;
   Info_Gen *Info_Non_Proba; /* Informations relatives aux strat. non probabilistes. */
   Info_Gen *Info_Proba; /* Informations relatives aux strat. probabilistes. */
   struct Info_Indiv_Type *Next;
   }info_indiv;
```

Point III: Présentation graphique des structures de données

La nécessité de disposer à tout moment de certaines informations est une contrainte très importante et il nous a semblé indispensable de représenter graphiquement les '*structures particulières*'. Chacune de ces structures peut être respectivement représentée comme suit:

- Info_Init_Strat:

Adr	Agen	N_Stra	Nb_Instance	Proba	Tx_Croiss_Po	Payoff_a	Next

- Info_Gen:

Previo	Ident	Info_Strat_Ad	Actio	Next

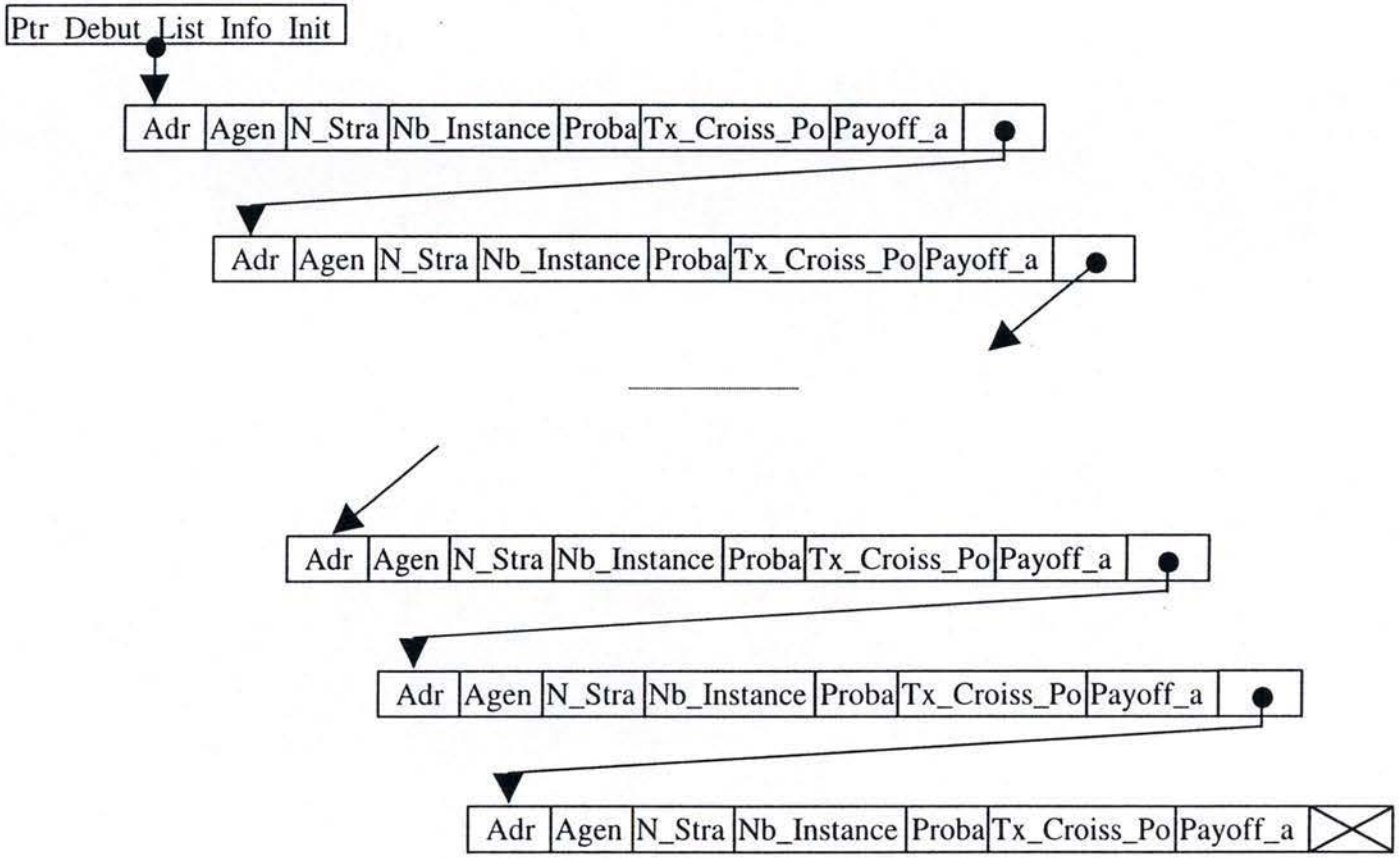
Remarquons que le champ '*Ident*' indique dans le cas de deux stratégies non probabilistes le numéro identifiant le type de stratégie de l'adversaire. Par contre, il fait référence à un numéro d'individu au sein de la population si, au moins, une des deux stratégies est probabiliste.

- Info_Indiv:

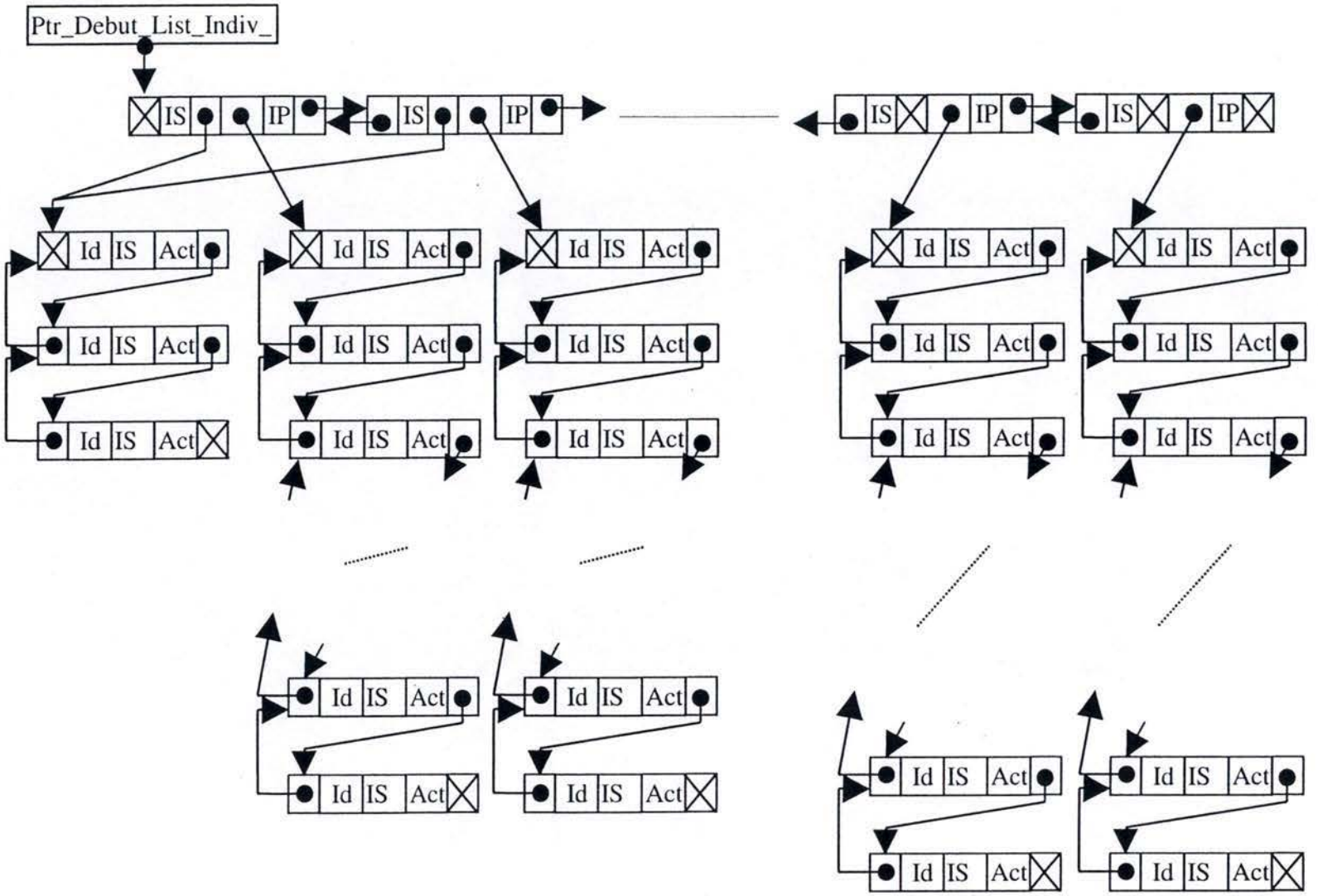
Previo	Ident_Str	Ident_Ind	Info_Non_Pro	Info_Pro	Next

Ci-dessous nous donnons la représentation de la liste reprenant l'ensemble des stratégies sélectionnées par l'utilisateur et les caractéristiques qui lui sont rattachées. Par la suite, nous donnons celles de la population. Soulignons que nous y avons décrit une population où les deux premiers individus de celle-ci ont adopté la même stratégie non probabiliste. Ceci est visible par le fait que leur champ '*Ident_Strat*' n'est pas nul et pointe vers une zone mémoire identique. Par ailleurs, leur champ '*Ident_Indiv*' nous permet d'affirmer qu'il y a au moins trois individus qui ont opté pour une stratégie probabiliste. C'est notamment le cas pour les deux derniers éléments représentés.

Représentation de la structure 'info_init_strat'

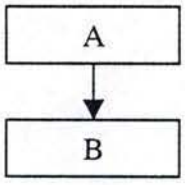


Représentation des structures 'Info_Gen' et 'Info_Indiv'



Article II: L'architecture logique

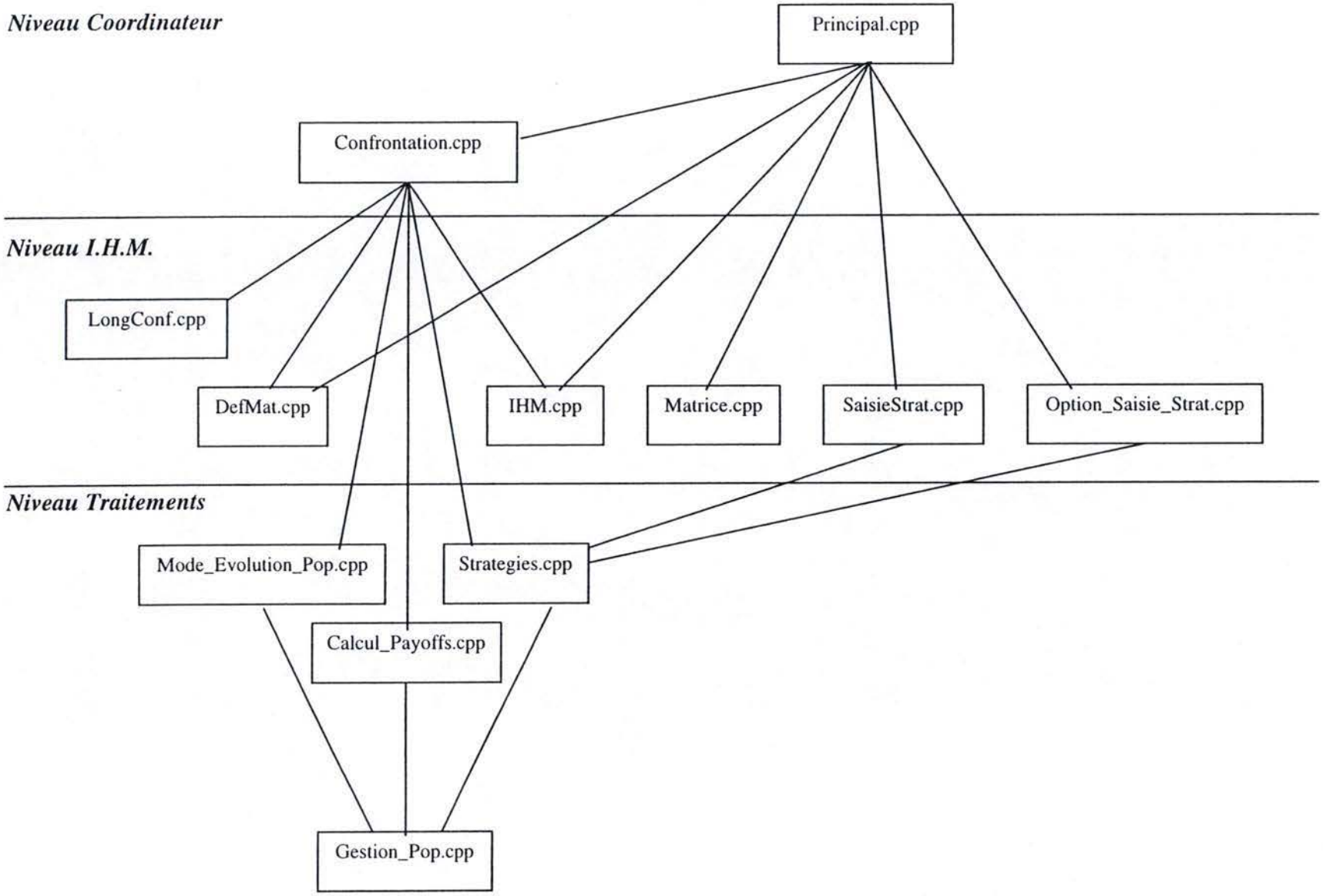
L'architecture logique est basée sur la décomposition du logiciel en modules. Dans le but d'alléger cet exposé, nous nous limitons volontairement aux éléments essentiels, et ne reprenons ci-dessous que le graphe acyclique. Celui-ci contient l'ensemble des fichiers constituant l'ossature de ce travail et les relations entre ceux-ci. La symbolique utilisée est la suivante:



Cette représentation signifie que le module '*A*' sert de coordinateur au module '*B*'. Dans cette phrasologie, il importe de noter qu'un module de niveau '*i*' ne peut coordonner que les modules de niveau inférieur. Pour la structure proprement dite, nous nous référons aux niveaux définis dans le cadre du cours de Méthodologie de Développement des Logiciels. En ce qui concerne notre logiciel, celui-ci se limite au trois premiers niveaux. Pour rappel, la structure des différents niveaux est caractérisée de la façon suivante:

Niveau 6:	Coordinateur
Niveau 5:	I.H.M.
Niveau 4:	Traitements
Niveau 3:	Données persistantes
Niveau 2:	Outils systèmes
Niveau 1:	Système d'exploitation

Le graphe acyclique est donné à la page suivante.


Niveau Coordinateur
Principal.cpp
Confrontation.cpp
Niveau I.H.M.
LongConf.cpp
DefMat.cpp
IHM.cpp
Matrice.cpp
SaisieStrat.cpp
Option_Saisie_Strat.cpp
Niveau Traitements
Mode_Evolution_Pop.cpp
Strategies.cpp
Calcul_Payoffs.cpp
Gestion_Pop.cpp

Section II: Les principaux algorithmes utilisés

Nous nous limitons ci-dessous aux principaux algorithmes du logiciel. Les fonctions auxquelles nous nous référons sans les exposer sont celles dont le nom est suffisamment explicite pour la compréhension du lecteur.

Les algorithmes que nous développons ici sont au nombre de trois. Dans le premier, pour une période donnée, nous déterminons le comportement d'un individu par rapport à tous les autres, et répétons cette opération pour tous les individus composant la population. Dans le second, nous calculons le payoff moyen par stratégie, et définissons également le payoff moyen de la population. Enfin, dans le troisième, compte tenu des résultats obtenus dans le second, nous comparons pour chaque stratégie son payoff moyen par rapport au payoff moyen de la population, et en fonction de la règle d'évolution exposée ci-après (Cfr. '*Section III: La règle d'évolution de la population*') déterminons le nombre d'instances de chaque stratégie. Nous répétons cette procédure jusqu'au moment où nous arrivons à la fin de la période de la confrontation.

Article I: Algorithme de confrontation pour une période

```
DEBUT DE L'ALGORITHME
  DEBUT DE L'INITIALISATION
    info_indiv *tmp = liste_populace ;
    tab_strat *tab_tmp = &tableau_strat;
  FIN DE L'INITIALISATION
  DEBUT DE LA CONFRONTATION PROPREMENT DITTE
  TANT QUE (tmp != NULL) REPETER
    SI (individu[1] != tmp)
      ALORS
        unsigned int i := individu→Ident_Strat;
        unsigned int j := tmp→Ident_Strat;
        SI ( (tab_tmp[i].Flag==Strat_Non_Prob) ET (tab_tmp[j].Flag==Strat_Non_Prob) )
          ALORS
            /* On traite le cas où les deux stratégies sont non probabilistes. */
            Individu→Info_Non_Proba = Affrontement(Individu, j, false);
            TANT QUE ( (tmp != NULL) ET (tmp→Ident_Strat == j) )
            /* Cette boucle a pour effet de passer au-delà des stratégies non probabilistes qui
            sont identiques à celle qui vient d'être traitée. */
```

[1] Il s'agit du paramètre passé à la fonction.

```
            tmp=tmp→Next;
          FIN DU TANT QUE
        SINON
          /* On traite le cas où l'une des deux stratégies au moins est non probabiliste. */
          Individu→Info_Proba = Affrontement(Individu, tmp→Ident_Indiv, true);
          tmp = tmp→Next;
        FIN DU SI
      SINON
        tmp = tmp→next;
    FIN SI
  FIN TANT QUE
  FIN DE L'ALGORITHME
```

Article II: Algorithme permettant de calculer les résultats par stratégie

```
DEBUT DE L'ALGORITHME
  PHASE D'INITILISATION
    unsigned int Nb_eff_indiv = 0; /* 'Nb_eff_indiv' donne le nombre effectif de personnes
    constituant la population. */
    info_indiv * liste_populace_tmp = liste_populace;
    unsigned int payoff_acc_totaux = 0;
    info_gen * ptr_info_gen;
    card behavior; /* 'behavior' indique quel sera le comportement de l'individu au cours de la
    période analysée. */
    unsigned int entier_tmp; /* 'entier_tmp' donne le nombre d'individus ayant adopté la même
    stratégie */
    info_init_strat * ptr_elem_list_init_strat; /* Cette variable nous permettra d'aller
    rechercher certaines informations relatives au type de stratégie étudié. */
    unsigned int val_interm;
  FIN DE LA PHASE D'INITILISATION

  TANT QUE (liste_populace_tmp != null) REPETER
    unsigned int i := liste__populace_tmp→Ident_Strat;  /* Identifiant pour les stratégies non
                      probabilistes. */
    unsigned int j := liste_populace_tmp→Ident_Indiv;  /* Identifiant pour les stratégies
                      probabilistes. */

    ptr_info_gen := liste_populace_tmp→Info_Non_Proba;
    TANT QUE (ptr_info_gen != null) REPETER
      SI (ptr_info_gen→Action != abandon)
        ALORS
          behavior := search behavior_adv(liste_populace, true, ptr_info_gen→Ident, i);
          ptr_elem_list_init_strat := search_ptr_elem_list_init_strat(i);
          val_interm := calcul_payoff(ptr_elem_list_init_strat, agant, ptr_info_gen→Action,
behavior);
        SINON
```

```
            val_interm := calcul_payoff_renoncement(type_agant, payoff);
        entier_tmp := nbr_people_same_strat(liste_pop, ptr_info_gen→Ident);
        val_interm := entier_tmp * val_interm;
        ptr_elem_list_init_strat→payoff_acc = ptr_elem_list_init_strat→payoff_acc +
    val_interm;
        ptr_info_gen = ptr_info_gen→next;
    FIN TANT QUE

    ptr_info_gen := liste_populace_tmp→info_proba;
    TANT QUE (ptr_info_gen != null) REPETER
        ptr_elem_list_init_strat := search_ptr_elem_list_strat(i);
        SI (ptr_info_gen != abandon)
            ALORS
                behavior := search_behavior_ado (liste_populace, false, ptr_info_gen→Ident, j);
                val_interm = calcul_payoff(ptr_elem_list_init_strat→agent, valeur_payoffs);
            SINON
                val_interm := calcul_payoff_renoncement(ptr_elem_list_init_strat→agent,
    valeur_payoffs);
        ptr_elem_list_init_strat→payoff_acc := ptr_elem_list_init_strat→payoff_acc +
    val_interm;
        ptr_info_gen := ptr_info_gen→next;
    FIN TANT QUE

    list_populace_tmp := list_populace_tmp→next;
    Inc(Nb_eff_indiv);
  FIN TANT QUE
FIN DE L'ALGORITHME
```

Cette fonction réévalue le nombre effectif de la population. En outre, elle met à jour le champ payoff_acc de la liste simplement chaînée.

Article III: Algorithme de confrontation généralisée

```
DEBUT DE L'ALGORITHME
  PHASE D'INITILISATION
    info_indiv *liste_pop = Constitue_Pop_Init (list_Stat_Selec);
    unsigned int période = 0;
    info_indiv *liste_pop_tmp, first_elem = liste_pop;
  FIN DE LA PHASE D'INITILISATION
  TANT QUE !(période > max_length) REPETER
    liste_pop_tmp = first_elem;
    TANT QUE !(liste_pop_tmp == NULL) REPETER
      Confrontation_One_Periode (liste_pop_tmp);
      liste_pop_tmp = liste_pop_tmp→next;
    FIN TANT QUE
```

```
/* Pour chaque individu, on a déterminé quel était son comportement. Dès lors il nous faut
regarder l'évolution des stratégies (un type de stratégie est considéré comme formant un tout
homogène). */
  Calcul_résult_par_strat (liste_pop);
  liste_pop = Evol_pop (liste_pop, liste_strat_selec);
  Inc(période);
FIN TANT QUE
FIN DE L'ALGORITHME
```

Section III: La règle d'évolution de la population

Toute règle d'évaluation est exogène. Dans nos recherches, nous avons adopté une variante de '*Replicator dynamic*'. Cette règle est la plus fréquemment rencontrée à travers la littérature. Elle a l'avantage de dresser une comparaison entre les payoffs moyens accumulés lors de la période précédente de chacune des stratégies avec l'entièreté de la population. L'augmentation de fréquence d'une stratégie est, en effet, proportionnelle à l'écart entre le payoff de cette stratégie et celui de la population.

Ce mode d'évaluation favorise des payoffs élevés obtenus aussi souvent possible et non les payoffs élevés essentiellement concentrés en début de confrontation. Ainsi, on relève le différentiel moyen de la dernière période et non le différentiel moyen accumulé jusqu'à cet instant.

Afin de permettre l'extinction d'un type de stratégie tout en ne conservant que des payoffs positifs, nous avons fixé une règle d'évolution de la population. On retrouve celle-ci à la fin de l'algorithme de confrontation généralisée. Sur base des payoffs accumulés au cours d'une période et du nombre d'individus d'un type de stratégie, on calcule le *revenu par personne pour une stratégie déterminée*. Si celui-ci est inférieur au *revenu moyen (au cours de la période) de la population*, alors on retire le nombre d'instances souhaité de la stratégie concernée jusqu'au moment où cette situation disparaît. Par contre, si le revenu moyen de la population est supérieur ou égal à celui de la stratégie, alors cette dernière verra sa population croître comme prévu initialement.

A chaque période, nous devons effectuer les calculs ci-dessous. Le début et la fin de la période 't' sont respectivement notés 't_d' et 't_f'. De façon formelle et séquentielle, nous avons les trois étapes suivantes:

Première étape: Calcul du payoff moyen de la population au cours de la période 't'.

$$P_p^{t_f} = \sum_{i=1}^{n} P_{s_i}^{t_f}$$

où $P_p^{t_f}$ représente le payoff accumulé au cours de la période 't' par l'ensemble de la population,

$P_{s_i}^{t_f}$ désigne le payoff accumulé au cours de la période 't' par un type de stratégie, et

n indique le nombre de stratégies qui font l'objet de la confrontation.

$$P_p^{t_d} = \sum_{i=1}^{n} P_{s_i}^{t_d}$$

où $P_p^{t_d}$ représente le payoff accumulé en début de période 't' par l'ensemble de la population, et

$P_{s_i}^{t_d}$ désigne le payoff accumulé en début de période 't' par un type de stratégie.

$$P_{\mu_p}^{t_f} = \frac{P_p^{t_d} - P_p^{t_f}}{I_p^{t_d}}$$

où $P_{\mu_p}^{t_f}$ donne le payoff accumulé moyen au cours de la période t par l'ensemble de la population, et

$I_p^{t_d}$ est le nombre d'individus au sein de la population au début de la période 't'.

Deuxième étape: Calcul du payoff moyen obtenu par une stratégie au cours de la période 't'.

$$P_{\mu_{s_i}}^{t_f} = \frac{P_{s_i}^{t_d} - P_{s_i}^{t_f}}{I_{s_i}^{t_d}}$$

où $P_{\mu_{s_i}}^{t_f}$ donne le payoff accumulé moyen au cours de la période t par l'ensemble de la population,

$P_{s_i}^{t_d}$ désigne le payoff accumulé au début de la période 't' par un type de stratégie, et

$I_{s_i}^{t_d}$ est le nombre effectif d'individus de la stratégie 's_i' présents en début de période 't'.

Troisième étape: Détermination de la variation ***effective*** du nombre d'individus d'une stratégie particulière.

Pour ce faire, nous allons comparer $P_{\mu_p}^{t_f}$ et $P_{\mu_{s_i}}^{t_f}$. Il nous faut distinguer deux situations:

a. $P_{\mu_p}^{t_f} \leq P_{\mu_{s_i}}^{t_f}$

Dans ce cas, le nombre d'individus composant la population de la stratégie 's_i' évolue de $\left(I_{s_i}^{t_f} - I_{s_i}^{t_d}\right)$, où $I_{s_i}^{t_f} = I_{s_i}^{t_d} * Tc_{s_i}$ et Tc_{s_i} est égal au taux de croissance de la population que l'utilisateur a fixé au préalable.

b. $P_{\mu_p}^{t_f} > P_{\mu_{s_i}}^{t_f}$

Dans ce cas, le nombre d'individus composant la population de la stratégie 's_i' diminue de 'm' unités. 'm' est le plus grand nombre entier positif tel que $P_{\mu_p}^{t_f} \leq \frac{P_{s_i}^{t_f}}{I_{s_i}^{t_d} - m}$.

Formulation générale: Nous pouvons résumer nos différentes digressions par la formule suivante:

$$I_{s_i}^{t_f} = f\left(\underset{-}{I_{s_i}^{t_d}}, \underset{-}{P_{s_i}^{t_d}}, \underset{+}{P_{s_i}^{t_f}}, \underset{+}{I_p^{t_f}}, \underset{+}{P_p^{t_d}}, \underset{-}{P_p^{t_f}}\right)$$

Après avoir ainsi mis sous forme logarithmique les différents propos tenus aux chapitres précédents, il nous reste à les insérer dans le domaine réservé de l'''*Intelligence Artificielle*''. C'est ce que nous faisons au chapitre suivant.

Section IV: Indications concernant les améliorations susceptibles d'être apportées au logiciel

Il importe de cerner au mieux les méthodes, les algorithmes et les nouveaux paramètres à développer pour:

- fixer les critères de décision permettant l'implémentation de nouvelles stratégies et donner les règles élémentaires à la base de leur création, et
- déterminer de nouveaux modes de calcul des payoffs permettant de les adapter aux modifications proposées.

Il est certain que le logiciel pourra faire l'objet d'une multitude de modifications. Il nous a néanmoins semblé opportun d'expliciter les deux améliorations mentionnées ci-dessus, et ce pour trois raisons. Premièrement, il est impossible de définir l'ensemble des stratégies possibles. Deuxièmement, tout individu peut fixer une règle d'évolution pertinente. Enfin, les '*upgrates*' que nous proposons dans le cadre de cette section représentent une méthodes excessivement simple permettant d'avoir une première prise de contact avec le code source du logiciel. Dans les développements effectués ci-dessous, nous nous référons fréquemment aux fichiers: '*Confrontation.cpp*', '*GestionPop.cpp*', '*TypeDef.h*' et '*VarGlobales.h*'. Examinons à présent la méthode concernant l'implémentation d'une nouvelle stratégie.

Article I: Commentaires relatifs à l'implémentation d'une nouvelle stratégie

L'insertion d'une nouvelle stratégie peut être décomposée en deux étapes. En effet, il faut distinguer le cas de stratégies nécessitant des informations particulières pour l'exécution de leur fonction de celles qui se satisfont des structures déjà définies.

Point I: Création d'un nouveau champ à l'union 'info'

Si cela s'avère nécessaire, il se peut qu'on soit dans l'obligation de modifier le fichier '*TypeDef.h*'. En effet, on peut devoir créer une nouvelle structure correspondant aux diverses informations nécessaires permettant de faire exécuter de façon optimale la fonction relative à une nouvelle stratégie. Ce nouvel élément est ensuite introduit dans l'union '*info*'. Il nous faut alors adjoindre un nouvel élément au type énuméré '*Param*'.

On effectue ensuite la mise à jour de la fonction '*Affrontement*' qui se trouve dans le fichier '*Confrontation.cpp*'. Enfin, on apporte les compléments indispensables à la fonction '*MAJ_Champs*' dans le fichier '*GestionPop.cpp*'.

Point II: Insertion d'une nouvelle stratégie proprement dite

Nous avons déjà énuméré des exemples de stratégies. Il est bon de souligner que les fonctions susceptibles d'être écrites ne requérent aucune connaissance poussée du langage C. Après avoir inséré une nouvelle stratégie dans le fichier '*Stratégies.cpp*', il nous faut compléter, dans le fichier '*VarGlobales.h*', la variable globale '*Tab_Strat*' en nous référant adéquatement à la nouvelle stratégie. Pour terminer, dans ce même fichier, on incrémente d'une unité la variable '*Taille_Tab_Strat*'.

Article II: Création de nouvelles règles d'évolution de la population

Toute règle d'évolution de la population peut être considérée comme résultant d'un choix arbitraire. Comme nous l'avons indiqué précédemment, celle-ci constitue un ***paramètre exogène***[2]. On peut dès lors imaginer que dans certains domaines de recherche[3] l'opérateur doive adapter le logiciel et prendre en considération un nouveau mode d'évolution de la population. Dans cette optique, nous avons conçu des modules de programmation aisément adaptable à une telle modification. Il suffit, en effet, de créer un nouvel élément au sein des types énumérés '*Type_Mode_Calcul*' et '*Type_Evolution_Population*'. Par la suite, il ne reste plus qu'à transformer de façon adéquate la fonction '*Mode_Evol_Pop( )*' figurant dans le fichier '*Mode_Evolution_Population*'.

---

2 Cfr. Chapitre III, Section IV.

3 Notons que notre logiciel a pour ambition d'être utilisé dans des domaines de recherche les plus divers: l'économie, la biologie, la sociologie, l'informatique,...

Commentaires

Les logiciels existants sont essentiellement des modèles destinés à permettre l'observation du comportement des différentes stratégies. L'avantage de notre logiciel est qu'il cumule les propriétés du ***modèle*** aux facultés d'un ***simulateur*** qui fait également intervenir des variables exogènes. On peut amplifier cette qualité de simulateur en définissant notamment d'autres règles d'évaluation de la population.

Une dernière piste consiste à explorer les différentes orientations relevant du domaine de l'I.A.. Nous abordons ce champ de façon théorique dans les chapitres suivants.

Chapitre IV: Appréhension de la notion d'intelligence artificielle

Introduction

Il est toujours possible de concevoir des programmes qui posent des actions '*intelligentes*'. Encore faut-il définir ce que nous entendons par cette expression. En fait, il n'existe pas de définition précise et incontestable à ce sujet. Nous pouvons considérer qu'***une action est intelligente dans la mesure où elle est difficile à mettre en œuvre***[1]. Dans ce chapitre, nous expliciterons quelque peu nos motivations basées, entre autre, sur le test de Turing. Nous nous attacherons par la suite à mieux cerner la notion d'Intelligence Artificielle en explicitant ses deux caractéristiques essentielles, d'une part son caractère statique lié à la connaissance, et, de l'autre, son caractère dynamique associé au raisonnement. Dans les deux dernières sections, nous nous étendrons sur les deux conceptions qui s'affrontent actuellement quant à la place de l'Intelligence Artificielle à l'intérieur des sciences exactes, en la rattachant soit aux sciences cognitives, soit à une discipline de l'informatique.

1 Il est bon de remarquer que la notion d'intelligence est définie en utilisant des termes flous.

Section I: Motivation

La motivation sous-jacente à cette deuxième partie s'inscrit dans la philosophie du test de Turing. Ce dernier propose de déterminer '*l'intelligence*' d'une machine sur base d'un test appelé '*imitation game*'. Celui-ci consiste à placer une machine et un être humain dans une pièce séparée d'une autre personne appelée '*l'interrogateur*'. Celle-ci est incapable d'entamer directement un dialogue ou de voir son interlocuteur. Elle ne peut communiquer qu'en recourant à l'utilisation d'un terminal. Turing considère que la machine fait preuve d'intelligence si l'interrogateur est dans l'impossibilité de déceler si son interlocuteur est un être humain ou non[2]. Deux objections furent émises à l'encontre du critère de Turing, à savoir: '***Lady Lovelace's Objection***' et '***Argument from Informality of Behavior***'.

1. '***Lady Lovelace's Objection***': Ada Lovelace argumente que l'ordinateur ne peut faire que ce qu'on lui dit et, de ce fait, est incapable de poser des actions originales, ce qui est l'essence même de l'intelligence.

2. '***Argument from Informality of Behavior***': selon cet argument il est impossible de définir un ensemble exhaustif de règles permettant d'agir dans toutes les circonstances. En fait, cette argumentation est un plaidoyer implicite pour l'application de raisonnements aussi abstraits que possibles. Pour distinguer, l'être humain, cette dialectique sous-entend que l'interrogateur peut miser sur la nature émotionnelle de ce dernier. En d'autres termes, toutes les réactions, et entre autres les réactions affectives, sont propres à chaque individu et, de ce fait, difficilement quantifiables.

En ce qui nous concerne, l'objectif final est de disposer d'un automate qui soit capable d'adopter un comportement optimal en fonction de la stratégie choisie par l'adversaire. Mais ceci implique la nécessité de reconnaître la stratégie de ce dernier[3]. Cette reconnaissance constitue l'objet principal du reste de ce mémoire.

---

2 Il importe de souligner que le critère proposé par Turing ne permet pas de définir ce qu'est l'intelligence mais uniquement de l'observer.

3 Ce type de raisonnement est, comme nous le verrons ultérieurement, similaire à celui adopté par une personne lorsqu'elle est confrontée à une autre.

Section II: La notion d'intelligence artificielle

Dans l'intelligence artificielle, on essaie de donner aux ordinateurs la faculté de comprendre une situation donnée et de s'y adapter de façon optimale. Il importe donc de mettre à la disposition des automates un vocabulaire (autrement dit un ensemble de symboles) et des règles permettant de représenter les différentes situations en fonction du contexte. D'emblée, on peut distinguer deux caractéristiques essentielles au sein du concept d'intelligence:

- l'***aspect statique*** lié à la ***connaissance***. En Intelligence Artificielle (I.A.) cet élément fera l'objet de mémorisations au sein des bases de connaissances et des bases de données. Il est bon de constater que, dans la vie de tous les jours, les représentations et les significations d'un objet sont étroitement liées au contexte dans lequel on se trouve. Nous pouvons prendre comme exemple l'adjectif '*léger*'. Lorsqu'on dit qu'une personne a une attitude légère cette affirmation englobe un côté péjoratif alors que si cet adjectif est associé au physique d'une personne ce point de vue revêt un aspect positif. On peut donc se rendre compte que pour aboutir à une représentation qui prenne en considération l'environnement dans lequel on se trouve, il faut tenir compte des pressions '*bottom-up*' et '*top-down*'. Comme le soulignent Luger, F. et Stubblefield, A. (1997) '*Rather than growing in the dark like mushrooms, intelligence seems to depend on an interaction with a suitably rich environment*'(page 13). Dans le cadre du cours de psychologie cognitive, nous avons déjà abordé ce problème. En effet, le modèle connectionniste implémente une mémoire de travail en utilisant les propriétés des différents noeuds qui la composent. Chaque concept distinct est représenté par un regroupement de noeuds en synchronie. La discrimination entre les différents concepts est rendue possible par l'examen des synchronies successives. Quand un noeud est activé en synchronie avec un autre, ceux-ci sont temporairement reliés ensemble. Lorsque l'automate lit les données d'un problème, il construit des modèles mentaux permettant de représenter les différents états possibles et cohérents avec l'information disponible. Il importe alors d'élaborer une inférence provisoire sur base des propositions du modèle. Si aucun contre-exemple ne permet de rejeter le modèle, alors l'inférence est tenue pour vraie.

Nous avons donc présenté ci-dessus une méthode basée sur les raisonnement par induction permettant d'étendre la connaissance. Nous aborderons le problème de la représentation au chapitre suivant, mais auparavant nous exposons l'aspect purement dynamique de l'intelligence.

- l'***aspect dynamique*** provoquant le ***raisonnement***. En Intelligence Artificielle, on créera des algorithmes permettant de regrouper les problèmes en différentes classes de façon à les résoudre plus facilement et plus efficacement. Le programme recherche la représentation de la situation proposée en élaborant des analogies avec sa base de représentation. Le choix des différentes séquences est basé sur des mécanismes probabilistes permettant la détermination progressive de l'espace de travail (nous reviendrons ultérieurement sur cet aspect crucial). In fine, l'ordinateur sera dans la possibilité d'adopter le comportement optimal compte tenu du contexte dans lequel il évolue.

La façon de considérer l'I.A. a fait s'opposer deux thèses. Il nous semble intéressant de définir la place de l'I.A. à l'intérieur des sciences exactes en examinant les arguments de ceux qui la considèrent comme faisant partie des sciences cognitives, et ceux qui la considèrent comme une branche issue de l'informatique.

Section III: L'I.A. considérée comme science cognitive

Selon cette approche, l'objectif de l'I.A. est de pouvoir appréhender les phénomènes et les processus intervenant lors du raisonnement. Cette définition est en concordance avec la finalité des sciences cognitives qui étudient les différents mécanismes par lesquels on acquiert des informations sur l'environnement. Par analogie, on peut paraphraser le slogan de l'UNICEF, à savoir: '*Donnez des poissons aux habitants du tiers monde, ils mangeront pendant une année; apprenez leur à pêcher, il mangeront toujours*', en disant que l'I.A. a comme ambition non seulement de comprendre les mécanismes de compréhension de l'être humain mais également d'apporter des solutions à certains problèmes. Il s'agit notamment d'élaborer des algorithmes performants permettant de résoudre certaines difficultés. Ainsi, les scientifiques recourent de plus en plus à la programmation de leurs modèles sur ordinateur. En fonction du comportement du programme ils peuvent par la suite invalider ou non le modèle analysé. Il est toutefois faux de considérer qu'on tente d'assimiler la puissance de réflexion de l'homme à la force de calcul de l'ordinateur.

Comme nous l'avons mentionné précédemment, la vision cognitive de l'I.A. met en avant deux aspects importants de la connaissance qu'il importe de prendre en considération, à savoir:

1. la représentation symbolique des données, et
2. la manipulation de celles-ci.

S'il est vrai que ces deux points sont omniprésents au travers de la science informatique, il n'empêche qu'ils sont essentiels dans le domaine particulier de l'I.A..

Farreny, H et Ghallab, M. (1987) font remarquer que la thèse cognitive repose sur un postulat central selon lequel '***le mécanisme de raisonnement ne s'effectue pas chez l'homme de façon aléatoire*** (...)***, mais d'une manière organisée, c'est-à-dire selon un algorithme*** (...) ***qu'il s'agit de découvrir***'. Par ailleurs, les auteurs ajoutent que leur démarche implique trois orientations:

1. il existe '***un certain niveau de déterminisme (deux masses ne s'attirent pas au hasard),***
2. il importe '***d'élaborer une théorie (ensemble de lois ou d'algorithmes)***', et
3. il faut également se prêter au jeu de '***l'expérimentation***'.

Ces considérations ne peuvent nous faire oublier que le monde extérieur est caractérisé par son ouverture sur des domaines en perpétuelle mutation. A cet égard, les champs inexplorés sont particulièrement vastes. Dans ce contexte, la puissance de l'ordinateur se prête particulièrement bien à de nouvelles expériences. Mais, par ces propos, on reconnaît implicitement le manque de '*fiabilité*' résultant en premier lieu du recours à des règles figées, comme c'est notamment le cas lorsqu'on considère l'I.A. comme partie intégrante des sciences cognitives. Une autre lacune provient du fait qu'on doit se limiter à une modélisation (représentation) finie de l'environnement, et celui-ci est toujours plus vaste que sa représentation. Cette évidence renforce la thèse de ceux qui perçoivent l'I.A. comme une branche faisant intégralement partie de l'informatique. Nous en parlons brièvement dans la section suivante.

Section IV: L'I.A. comme branche de l'informatique

Pour certains, l'I.A. n'est qu'une discipline parmi d'autres de l'informatique. Selon cette approche, son objet est de rendre l'ordinateur '*plus intelligent*' en lui inculquant des modes de raisonnement permettant à l'automate d'accomplir des tâches de plus en plus complexes, comme c'est notamment le cas dans le monde médical ou dans le domaine aérien. En d'autres termes, l'I.A. vise à concevoir des programmes en mesure de traiter des problèmes pour lesquels on ne connaît pas de résolution directe et certaine. Elle apparaît alors comme un outil permettant d'exploiter de la façon la plus performante les possibilités des ordinateurs afin de leur faire réaliser des fonctions requérant certaines spécificités de l'intelligence humaine. Ces fonctions relèvent plus du raisonnement symbolique que des calculs arithmétiques purs. Dans cette perspective, la préoccupation première est de trouver une réponse aux problèmes posés et non de calquer le programme implémenté dans la machine sur la solution trouvée éventuellement par l'esprit humain. Au demeurant les deux solutions peuvent avoir suivi une démarche tout à fait différente.

Dans cette approche, l'I.A. apparaît comme une discipline en perpétuel développement. Elle suscite des recherches continuelles de la même manière que l'analyse à l'intérieur des sciences mathématiques ou la chirurgie au sein de la médecine. Du fait de nombreux progrès, l'Informatique est en plein bouleversement. Les logiciels deviennent de plus en plus performants tant sur le plan du temps de calcul que sur le plan de la masse des données manipulées. Leur rôle devient prépondérant par rapport aux informations transmises. Dans ce contexte, il n'est pas dénué de sens de considérer l'I.A. comme partie intégrante de l'Informatique.

Commentaires

Sur base des différents points de vue exposés ci-dessus, on peut dire que l'approche cognitive essaie de comprendre l'intelligence existante alors que l'approche technique tente de développer des choses intelligentes.

Malgré cette deuxième vision de l'I.A., il nous semble qu'elle peut être considérée comme une science à part entière. En effet, sa raison d'être est de permettre de mieux appréhender le monde réel. Cette discipline se penche sur des problèmes tels que le rôle de l'interprétation, les idées abstraites, la mémorisation des données, l'association de différents phénomènes, la déduction qui en découle ou encore la synthèse de plusieurs situations,... bref ***elle (l'I.A.) se réfère directement aux divers éléments constitutifs de l'intelligence de l'être humain.*** Mais en même temps, elle diffère considérablement de celle-ci étant donné que leurs champs d'investigation respectifs sont relativement différents. Ainsi l'I.A. résoudra sans trop de difficultés des problèmes mathématiques, comme le binôme de Newton ou les équations différentielles, ou impliquant l'intervention d'une multitude de variables, comme c'est notamment le cas au sein des systèmes experts des centrales nucléaires. Par contre, il est beaucoup plus ardu de modéliser les problèmes peu ou pas structurés. Ceux-ci ressortent davantage des compétences exclusives de l'esprit humain.

Comme susmentionné, les systèmes experts font apparaître l'émergence d'une forme d'I.A. dans la mesure où ils permettent des prises de décisions analogues à celles de l'homme. Cependant de tels systèmes mettent en avant un des problèmes majeurs auquel on est confronté: l'***acquisition de connaissances***. Celle-ci requiert une structuration rationnelle. C'est notamment la-dessus que repose le '***Case Base Reasoning***'. A l'inverse, l'expert humain organise l'ensemble des informations qui lui parviennent sans effort apparent. En ce

qui nous concerne, nous abordons cette problématique en tentant d'apporter des solutions permettant à l'ordinateur d'appréhender le comportement de l'adversaire.

En dehors de l'acquisition de connaissances, une deuxième difficulté réside dans la ***représentation symbolique*** de l'environnement dans lequel on raisonne. Elle présuppose un découpage pertinent du monde qui nous entoure. Dans un tel contexte, l'analyse du monde est essentiellement un problème d'analogie. Dans la mesure où la représentation du monde délimite notre espace de travail[4], il importe de choisir un modèle de notre environnement où toutes les structures sont construites. Ce dernier point fait l'objet du chapitre suivant.

En guise de conclusion, il importe de noter que, comme le laissait sous-entendre nos propos relatifs à la définition d'I.A., son domaine d'application est en perpétuelle évolution. Ainsi, au début de l'apparition de l'I.A., on considérait les compilateurs comme faisant partie de l'I.A.. Aujourd'hui, ils échappent au domaine réservé à l'I.A.. Comme nous le verrons dans les deux chapitres suivants, l'I.A. est maintenant nettement axée sur des algorithmes et des cheminements proches du raisonnement humain.

---

[4] Encore appelé, espace d'états. L'espace d'états est l'ensemble des états générables par application d'opérateurs de transition.

Chapitre V: La problématique de la représentation de la connaissance: Application au dilemme du prisonnier

Introduction

Nous avons signalé précédemment qu'un aspect important de la problématique de l'I.A. concerne la représentation du monde réel. Au plus cette dernière est représentée de façon convenable et efficace, au plus on parviendra à élaborer une véritable I.A.. En effet, il importe de trouver la meilleure représentation permettant, d'une part, de bien poser le problème et, d'autre part, d'en faciliter la résolution. En d'autres termes, la représentation adoptée influence la performance et les chances de succès du '*résolveur de problème*'.

Comme le soulignent K. Binmore & L. Samuelson, '*Players are identified with the finite automaton that represents what they are using*' (J.E.T., page 284). L'objet de ce chapitre découle directement de ce postulat. Il s'agit, en effet, de donner une représentation adéquate d'un concept à la base de tous les jeux, à savoir: la stratégie adoptée par un individu.

A cette fin, nous avons adopté la représentation de l'automate fini (A.F.). Dans un premier temps nous définissons les concepts nécessaires; par la suite, nous nous attachons aux problèmes des stratégies ne pouvant être représentées sous cette forme. Il importe de souligner que le recours à un A.F. équivaut à représenter le problème par une grammaire. Dès lors, pour un jeu

particulier[1], on l'analyse afin de déterminer s'il s'agit ou non d'une phrase acceptée par notre grammaire.

Le choix de la représentation d'un A.F. se justifie par le fait qu'il s'agit du modèle informatique le plus simple et le plus répandu. Celui-ci permet de prendre en considération un ensemble fini d'états[2]. Par l'introduction d'un input, l'A.F. opère ce qu'on appelle une '*transition*' d'un état vers un autre, voire vers lui-même.

Il est bon de remarquer que l'emploi d'un A.F. conduit à utiliser les concepts vus dans le cadre de la théorie des graphes. En effet, chaque état représente un domaine du monde, et chaque transition indique une relation entre deux représentations de celui-ci. En d'autres termes, on peut dire que l'utilisation d'un graphe permet non seulement de donner une représentation aisée et efficace du problème, mais surtout de le structurer. Comme le font remarquer Luger, G. et Stubblefield, W. (1997): '*The formalisation of graph theory afforted the possibility of state space search a major conceptual tool of Artificial Intelligence*' (page 7).

Dans ce qui suit, nous utiliserons indifféremment les termes A.F. et son graphe associé. Par ailleurs, pour la rédaction de ce chapitre, nous nous basons sur les travaux de K. Binmore & L. Samuelson, et de D. Mandrioll & C. Ghezzi.

Section I: Une stratégie vue comme un A.F.

Une stratégie vue comme un A.F. peut être représentée comme un quadruplet: $\langle S,I,E,\delta \rangle$,

où S = l'ensemble des états possibles (S = **S**tates). Etant donné que la confrontation peut s'arrêter à tout moment, il s'agit également de l'ensemble des états finaux possibles.

I = l'ensemble des états initiaux (I = **I**nitial states), il est évident qu'on a: $(I \subseteq S)$. Par ailleurs, à l'exception des stratégies probabilistes, l'ensemble '*I*' n'est composé que d'un et un seul état.

---

1 Ce qui revient à analyser une phrase particulière.

2 Comme nous le verrons, il s'agit à la fois d'une force et d'une faiblesse dans ce modèle.

δ = l'ensemble des caractères indiquant l'action pouvant être adoptée par une personne. En ce qui nous concerne, il ne peut s'agir que de C pour Coopérer, de T pour Trahir et de R pour Renoncer[3]

E = l'ensemble des arcs reliant deux états (E = **E**dges).

Cette composante peut être définie comme une fonction ayant la signature suivante: $S * \delta \rightarrow S$. Par référence à l'étoile de Keyne, E^* définit tous les '*mots*' pouvant être obtenus par l'automate, et ce en partant de l'état initial I. Simultanément, cet élément permet de définir l'ensemble des états accessibles d'une stratégie.

De façon générale, on peut dire que résoudre un problème[4] équivaut à déterminer une suite d'états intermédiaires S_i $(0 \leq i \leq n)$ tels que $\begin{cases} S_0 \in I, \\ S_n \in S, et \\ S_i \in E(S_{i-1}, \delta) \end{cases}$. Dans le cadre des méthodes proposées ultérieurement nous adapterons quelque peu ce processus de résolution.

Dès lors, nous pouvons graphiquement caractériser un automate de la façon suivante:

1. un ***cercle*** représente un état possible de l'automate. La lettre à l'intérieur de celui-ci indique l'action posée par la personne ayant adopté une stratégie déterminée à un moment du jeu. Par définition, cette action fait partie de l'ensemble δ.
2. un ***arc*** indique comment la personne passe d'un état à un autre. La lettre reprise au-dessus reflète la raison qui la pousse à changer de comportement. En fait, ce caractère indique le comportement de l'adversaire lors de la période précédente. Ici aussi, celui-ci fait partie de l'ensemble δ.
3. l'***état initial*** de la machine est indiqué par une flèche n'ayant pas d'origine.

Il est bon de remarquer que si on a un cercle avec la lettre R reprise en son sein, cela implique qu'aucun arc n'a celui-ci comme point de départ. On peut donc considérer cette situation comme un état final.

---

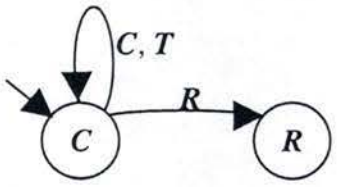
3 Au travers de toutes les représentations abordées ci-après, nous respecterons cette convention.

4 En d'autres mots, reconnaître si une suite de coups correspond bien à une stratégie donnée, ce qui revient à reconnaître une phrase via un analyseur syntaxique.

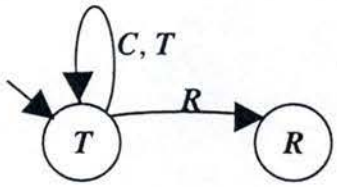
Dès à présent, on peut constater qu'à la différence des automates finis vus dans le cadre de la syntaxe et de la sémantique des langages de programmation, nous ne devons pas définir de façon explicite l'ensemble des états finaux. Ceci se justifie par le fait que tout état de l'automate représente potentiellement un état final étant donné que la longueur de la confrontation est inconnue[5].

Afin d'expliciter quelque peu ces différents propos, nous donnons ci-dessous la représentation via l'A.F. de certaines stratégies présentées dans le cadre du chapitre relatif aux stratégies (chapitre II).

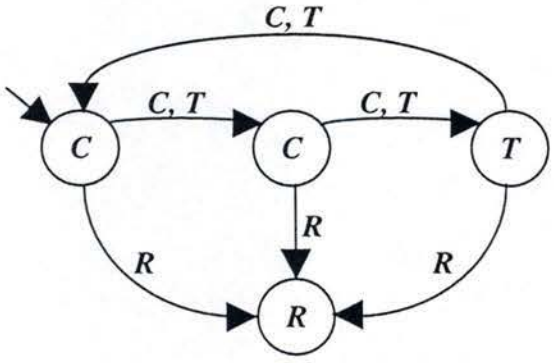
GENTILLE



MECHANT

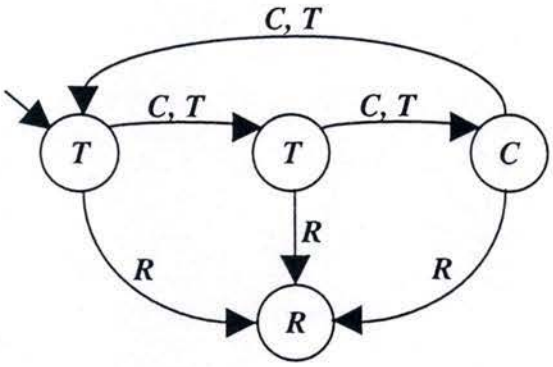


PER-GENTIL

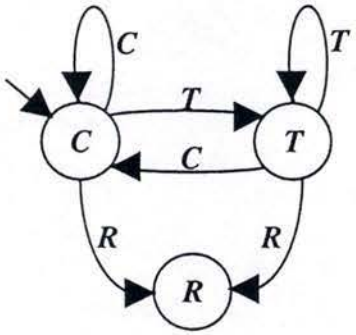


5 Sur le plan conceptuel, comme nous l'avons souligné précédemment, nous raisonnons sur une période de longueur infinie (Cfr. Chapitre I).

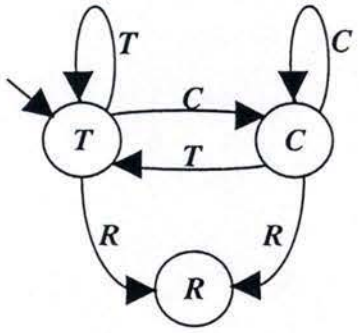
PER-MECHANT



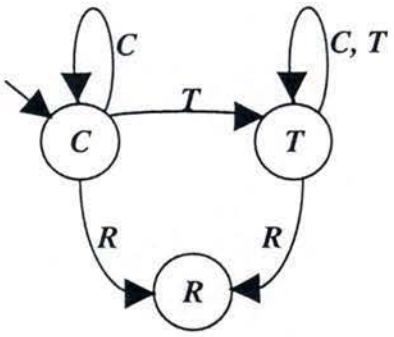
DONNANT-DONNANT



MEFIANT



RANCUNIERE



L'objectif de cette partie n'est certes pas de donner une vue exhaustive de la représentation de l'ensemble des stratégies possibles, car un tel développement est non seulement de peu d'intérêt mais aussi totalement impossible étant donné qu'il existe une infinité de stratégies. En fait, notre ambition se limite à trouver une représentation possible des stratégies via l'utilisation d'automates finis. Pour être complet, il importe de souligner que certaines stratégies (ceci est notamment le cas pour la stratégie '*Graduel*') ne peuvent être représentées via un automate fini.

Section II: Cas des stratégies complexes

Par stratégies complexes, nous entendons celles nécessitant, dans le cadre de la représentation d'un A.F. un nombre infini d'états ou du moins un nombre excessivement important[6]. Ainsi dans le cas de Graduel, les puristes pourraient argumenter, à juste titre, que sa représentation nécessite l'utilisation d'un automate à pile. Néanmoins, en ce qui nous concerne, et comme nous le verrons ultérieurement, nous pouvons nous contenter de l'utilisation d'un A.F. même si celui-ci ne reflète qu'imparfaitement la stratégie[7].

A cet égard, il est bon de noter que toutes les tentatives de représentation parfaite du monde réel sont vaines. Elles se heurtent, en effet, à des problèmes particulièrement complexes, à fortiori en informatique.

---

6 L'utilisation du terme '*important*' n'est pas fortuite. En effet, il nous est particulièrement ardu de le quantifier car il dépend de la configuration envisagée lors de la confrontation, c'est-à-dire de la longueur de la confrontation et des interactions avec les différentes stratégies sélectionnées.

7 C'est-à-dire en acceptant que l'A.F. représente une '*approximation*' suffisante de la stratégie.

Sans anticiper sur ce qui suit, et dans le but d'illustrer notre propos, nous pouvons envisager la stratégie suivante: '*Adopter la stratégie* ***TIT-FOR-TAT*** *sur 10.000 périodes puis toujours coopérer*'. Une telle stratégie est composée de $\left(1+\sum_{i=0}^{10.000} i*3+1\right)$ états, ce qui nous donne 150.015.002 possibilités.

Commentaires

Nous avons proposé une représentation symbolique. Celle-ci a l'avantage d'être exacte (c'est-à-dire qu'il n'y aura pas de dépassement de capacité) mais elle engendre comme désavantage d'être relativement coûteuse en temps et en mémoire. Après avoir jeté les bases de la représentation de la connaissance, l'étape suivante en découle logiquement. Elle concerne la faculté de raisonner en fonction des données perçues au sein de l'environnement. Nous développons cette problématique au chapitre suivant.

Chapitre VI: Algorithmes permettant d'appréhender le comportement de l'adversaire

Introduction

Le fait de vouloir déterminer la stratégie de l'adversaire est comparable au problème rencontré par un joueur d'échec ou un médecin. En effet, '*... when a chess player examines the effects of different moves or a doctor considers a number of alternative diagnoses, they are searching among alternatives*' (page 18)[1].

Il est évident qu'on ne peut demander à un automate d'élaborer des raisonnements qui ne découlent pas des algorithmes de base mis à sa disposition. L'objectif de ces derniers est de permettre à l'ordinateur d'apprendre à identifier '*la*' stratégie de l'adversaire. Pour ce, il doit être capable de reconnaître la structure de l'information (des données) qui se présente à lui. Vue sous cet angle, l'expression '*Intelligence Artificielle*' reflète la tendance à faire en sorte qu'un ordinateur soit capable de se comporter ***comme*** un être humain, au point de finir par se confondre avec ce dernier.

Au chapitre IV, nous avons déjà mis en exergue l'utilité de l'Intelligence Artificielle dans ce mémoire. Pour bien cerner le problème, nous évoquons ci-dessous l'histoire de Bruce & Sheila présentée par MAILATH, G. R. (1992).

---

[1] Luger, G. et Stubblefield, W.

Ce dernier donne l'exemple suivant: '*Suppose two players, Bruce and Sheila,... assume that Bruce and Sheila play this game several times before, and that Sheila has always played the same way. Then it seems reasonable, as a first approximation, for Bruce to believe that Sheila will continue to play this way in the game with him*[2], *and so Bruce should play a best reply to what Sheila has done historically*'. Mais on peut se demander ce qui peut se produire si Sheila a également pu observer le comportement de Bruce dans le passé (Cfr. infra). Afin d'éviter ce problème, on considère, dans un premier temps, qu'on raisonne dans un environnement constant à travers le temps. Nous entendons par '*environnement constant à travers le temps*' le fait qu'une fois qu'un individu opte pour une stratégie, il ne peut en changer.

L'auteur ajoute également, '*Suppose that Bruce does not know Sheila. ... He does know only how people have played this game,.... He will then* ***presumably*** *use the history of play in these games to help in evaluating the relative merits of his choices in this round,... , he will choose a best reply to that strategy. If all players behave in this manner and play reaches a steady state, then this play must constitute a Nash equilibrium. This is a learning story for Nash equilibrium* '.

Au moyen de cette illustration, on se rend compte que ***lorsqu'une personne doit interagir avec une autre, elle s'efforce de faire évoluer son comportement par rapport à celui de son concurrent en fonction des informations dont elle dispose sur ce dernier***. Autrement dit, elle puise dans sa mémoire, l'algorithme susceptible de reconstituer l'attitude passée du concurrent ou celle d'une personne semblable dans la même situation. Insistons sur le fait qu'avant de réagir, il importe de collecter suffisamment de données sur le comportement et sur les réactions possibles de son adversaire.

Il faut dès lors envisager un automate dont le rôle est dans un premier temps de déterminer la stratégie de l'adversaire. Par la suite, sur base des informations accumulées, il devra déterminer la stratégie optimale à adopter. Mais ce dernier point sort des objectifs que nous nous sommes fixés dans le cadre de ce chapitre. En effet, nous nous limitons à tracer quelques pistes de réflexions permettant de cerner au mieux la stratégie de l'adversaire.

---

2 Nous tenons à souligner le fait que cette hypothèse n'est pas à dédaigner.

Notons enfin que certains auteurs[3] considèrent qu'il est possible d'appréhender la problématique de l'I.A. en concevant des stratégies qui tentent de prendre en considération l'environnement dans lequel la confrontation évolue. Selon nous, il ne s'agit pas là d'une voie susceptible de saisir pleinement la complexité de l'I.A.. En effet, ces stratégies sont limitées à la connaissance accumulée au moment de leur conception. Elles sont de ce fait figées. Par conséquent, toute modification de la base de connaissances n'aurait aucun impact sur l'une d'entre elles puisqu'elles ne peuvent évoluer. En d'autres termes, une telle approche ne permet pas de mettre en œuvre les techniques de 'Case Base Reasoning' (Cfr. Infra).

Section I: Cadre de réflexion

Le cadre dans lequel nous évoluons est le suivant.

Face à une stratégie particulière[4], l'automate doit tenter de la distinguer. Dans ce but, il dispose d'un nombre maximum de périodes déterminées à l'avance que nous représentons par la lettre '*p*'.

Dans un premier temps, nous supposons que l'ensemble des stratégies potentielles pouvant être choisies par un concurrent fait partie de l'ensemble $\mathcal{E}^s$ des stratégies mémorisées dans l'ordinateur. Dans sa recherche de la stratégie adoptée par un concurrent, l'algorithme va procéder par élimination en enlevant de $\mathcal{E}^s$ les stratégies improbables.

A ce niveau, on peut signaler le fait qu'il peut s'avérer utile de regrouper les différentes stratégies en les numérotant de façon intelligente. Par intelligente, nous voulons dire que la numérotation les classifie dans des familles distinctes, chaque famille étant caractérisée par un '*tronc commun*[5]'. Dans la numérotation, le tronc commun caractérisant une famille sera déterminé par les deux derniers chiffres. Ainsi, nous pouvons définir la famille 01 et retrouver les stratégies 101, 201,..., la famille 02 et les stratégies associées 202, 302,... et ainsi de suite jusqu'à

---

3 Cfr. l'article de Kehagias (1998).

4 Parmi l'ensemble des stratégies enregistrées au sein de la base de connaissances de l'automate.

5 Par tronc commun, on entend les stratégies ayant des comportements similaires; en d'autres termes des caractéristiques identiques. Comme nous l'avons déjà souligné, il existe quatre critères fondamentaux caractérisant une stratégie. De ce fait, il faut envisager 2^4 classes différentes.

la famille 16. L'intérêt de cette façon d'aborder le problème sera réellement perceptible dans le cas de la deuxième méthode proposée ci-dessous.

Du fait de cette hypothèse simplificatrice, il importe de souligner qu'au terme de la période 'p':

1. L'ensemble $\mathcal{E}^s$ doit contenir au moins une stratégie puisque l'utilisateur opère ses choix à l'intérieur de la base de connaissances de l'ordinateur.
2. L'ensemble $\mathcal{E}^s$ peut contenir plusieurs stratégies potentielles.

Il est bon de rappeler qu'on suppose que l'ordinateur connaît l'ensemble des stratégies susceptibles d'être adoptées par un individu. Cette hypothèse implique que la représentation du monde n'est pas, pour notre automate, sujette à évolution[6]. Dans ce contexte, l'objectif de certains algorithmes proposés ci-dessous sera de réduire au maximum l'ensemble '$\mathcal{E}^s$' sur l'intervalle 'p'.

A contrario, lorsqu'on envisagera que l'ordinateur ne dispose pas de l'ensemble des stratégies possibles[7]*, on admettra de facto qu'il ne dispose d'aucune vue d'ensemble*[8]. De ce fait, il faudra adopter une approche qui ne soit plus basée sur '$\mathcal{E}^s$' (Cfr. Infra.).

Néanmoins, quelle que soit l'option choisie pour identifier la stratégie du concurrent nous allons tenter de construire un automate représentant la stratégie de l'adversaire ou plusieurs caractéristiques de celle-ci. Au terme de la période impartie, on écartera les automates qui ne sont pas équivalents à celui obtenu.

Enfin, quel que soit l'algorithme utilisé, il est évident que lorsqu'un agent pose une action, plusieurs concepts peuvent y être associés simultanément. Ainsi, si l'adversaire opte pour la trahison, cette attitude peut refléter d'une part une réaction (sanction) à la stratégie adoptée précédemment par son adversaire, et d'autre part une certaine complexité liée à la stratégie adoptée. Il faut dès lors envisager de faire ce que les psychologues appellent du '*JUNKING*', c'est-à-dire faire en sorte de différencier les concepts différents.

---

6 Ce qui implique un certain déterminisme.

7 En d'autres termes, on accepte le fait que la stratégie d'une personne puisse ne pas appartenir à la base de connaissances dont dispose l'automate.

Section II: Algorithmes de détection de la stratégie de l'adversaire

Ci-dessous nous présentons deux méthodes permettant de déceler la stratégie de l'adversaire. Après les avoir exposées sommairement, nous expliquons les avantages et les limites qui leur sont associés.

Article I: Détermination d'une signature d'exclusion permettant le rejet du plus grand nombre de stratégies

Point I: Présentation de la méthode

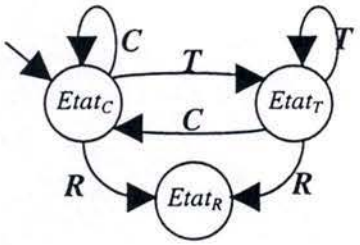
Comme il faut déterminer une stratégie parmi 's' options potentielles sur une période 'p', cela signifie qu'il existe 2^p signatures différentes. Pour s'en rendre compte, il importe de souligner qu'au départ trois alternatives sont proposées à chaque agent: coopérer, trahir ou renoncer. Cependant nous nous trouvons face à 2^p signatures différentes et non 3^p. En effet, dans la recherche de la stratégie de l'adversaire, nous pouvons écarter la possibilité de renoncer car cette décision, par son caractère irréversible ne nous apporte aucun renseignement supplémentaire.

Sur base des 2^p signatures proposées, il est essentiel d'identifier celle qui réduit au maximum l'ensemble initial $\mathcal{E}^s$. Formellement, on peut dire que chaque stratégie 'i' est perçue comme un langage particulier L_i. Autrement dit, on considère que chaque stratégie peut être représentée par une grammaire G_i qui lui est propre. L'analyseur de stratégie doit être dans la possibilité de dire si un item[9] reçu en entrée fait partie de la structure du langage de L_i. Pour ce, il faut définir une grammaire G_i pour L_i , qui peut s'écrire comme suit: $L(G_i)$. En d'autres termes, on peut dire que les règles de grammaire définissent la structure des mots de la stratégie 'i'. Prenons la stratégie ***DONNANT-DONNANT***. Comme nous l'avons déjà vu au chapitre précédent, les différents états de cet automate peuvent être représentés sur le graphe ci-dessous:

---

8 Dans la vie de tous les jours, nous n'avons qu'une vue partielle du monde qui nous entoure.

9 L'item sera ici composé de la signature d'exclusion et de l'action (réaction) de l'individu.



Si on utilise une formulation mettant en œuvre les règles de la logique du premier ordre, on peut la traduire sur base des huit règles suivantes:

- DébutDonnantDonnant($Etat_C$).
- FlècheDonnantDonnant(T, $Etat_C$, $Etat_T$).
- FlècheDonnantDonnant(T, $Etat_T$, $Etat_T$).
- FlècheDonnantDonnant(C, $Etat_C$, $Etat_C$).
- FlècheDonnantDonnant(C, $Etat_T$, $Etat_C$).
- FlècheDonnantDonnant(R, $Etat_C$, $Etat_R$).
- FlècheDonnantDonnant(R, $Etat_T$, $Etat_R$).
- FlècheDonnantDonnant(R, $Etat_R$, $Etat_R$).

Force est de constater que la méthode adoptée doit nous permettre de déterminer aussi précisément que possible la tactique de l'adversaire. Néanmoins, elle est grande consommatrice en termes d'espace mémoire et de temps de calcul.

En conclusion, on peut dire que cette méthode raisonne sur base d'éléments connus. Elle fonctionne suivant le ***plan de raisonnement*** suivant:

1. Etant donné l'ensemble des stratégies constituant la base de connaissances et le temps imparti pour découvrir la stratégie de l'adversaire, le premier objectif est de déterminer la signature d'exclusion qui permettra d'écarter le plus grand nombre de stratégies.

2. On confronte ensuite chacune des stratégies constituant la base de connaissance à la signature d'exclusion, et on regarde si l'action adoptée par celle-ci est conforme à celle de l'adversaire. Si ce n'est pas le cas on peut l'écarter

définitivement de l'ensemble $\mathcal{E}^s$. On répète cette étape pour les stratégies restantes dans $\mathcal{E}^s$ jusqu'au moment où, soit on arrive à la période 'p' soit il ne reste plus qu'une seule stratégie au sein de $\mathcal{E}^s$.

Point II: Critiques de la méthode

La méthode proposée ci-dessus peut aisément être améliorée en adoptant ***une signature d'exclusion évolutive***. En effet, au fil du temps, le choix de la signature d'exclusion fixé en début de période doit être remis en cause au vu des résultats récoltés et du laps de temps encore disponible. Si, par exemple, pour 'p' valant 6, au début de la période d'observation la meilleure signature d'exclusion était '*C T C C T C*', il se peut qu'après deux tours, et compte tenu des stratégies susceptibles d'être encore utilisées au sein de $\mathcal{E}^s$, la meilleure signature d'exclusion devienne '*T C T C*'.

Néanmoins, le processus proposé est peu efficace dans la mesure où il ne fait pas suffisamment preuve d'abstraction[10]. En effet, notre automate est limité à $\mathcal{E}^s$. Or s'il est vrai que cette hypothèse permet de simplifier une première approche, il n'empêche qu'elle limite exagérément le cadre de réflexion. Etant donné que la base de connaissances dont dispose l'ordinateur n'est pas infinie, il est évident que l'utilisateur peut décider d'adopter un nouveau comportement. De ce fait, il se peut qu'au terme de la période, l'ensemble $\mathcal{E}^s$ soit vide.

Enfin, comme nous l'avons déjà envisagé au sein de l'exemple de Bruce & Sheila, l'adversaire pourrait réévaluer sa stratégie et l'adapter en fonction des circonstances. Il faut dès lors concevoir les réactions de l'automate si suite à une action posée par l'adversaire l'ensemble des stratégies conservées au sein de $\mathcal{E}^s$ sont définitivement écartées. Faut-il redémarrer à zéro (en d'autres termes, considérer qu'on ne dispose d'aucune information pertinente par rapport à la stratégie de l'adversaire) dans le processus de détection de la stratégie de l'adversaire? Il peut alors s'avérer plus pertinent de considérer une procédure utilisant les caractéristiques des stratégies. En effet, lorsqu'une personne change son

---

10 Or le but de l'I.A. est de fournir des méthodes générales de résolution de problèmes. En d'autres termes des méthodes faisant preuve du plus d'abstraction possible.

comportement, il ne le modifie généralement pas complètement mais remet en cause certaines de ses caractéristiques.

L'objection sous-jacente commune à ces considérations peut être résumée en disant qu'*il est quasi impossible de fournir une base de connaissances qui soit suffisamment exhaustive*. Nous retrouvons ainsi ce qu'on appelle '*Argument from Informatily of Behavior*'.

Article II: L'utilisation des caractéristiques des stratégies

Point I: Présentation de la méthode

L'algorithme que nous proposons ici se base sur ***les caractéristiques des stratégies*** exposées précédemment pour déterminer la stratégie de l'adversaire. Pour rappel, ces caractéristiques sont: la sérénité, la réaction graduelle, la simplicité et le seuil. Plutôt que de réduire au maximum l'ensemble '$\mathcal{E}^6$', il semble plus pertinent de déterminer les caractéristiques de la tactique adoptée par l'adversaire. Plusieurs raisons plaident en faveur de cette solution:

1. plus la base de connaissances devient importante, plus le nombre de stratégies semblables augmente[11],
2. on peut avoir une base de données aussi importante qu'on le souhaite, rien n'empêche l'utilisateur d'opter pour une stratégie qui n'y soit pas reprise,
3. comme l'ont montré nos différentes considérations relatives à l'exemple de Bruce & Sheila, toute personne peut adapter son comportement en fonction de celui de son concurrent. Il est cependant raisonnable d'admettre que cette adaptation ne provoquera pas pour autant une révision fondamentale débouchant sur une nouvelle stratégie. En d'autres termes, il est raisonnable de supposer que la nouvelle stratégie aura plusieurs caractéristiques communes avec les stratégies utilisées auparavant,

---

[11] Par stratégies semblables, nous entendons des comportements relativement proches. Nous reviendrons ultérieurement sur ce concept.

4. le raisonnement sous-jacent à cet algorithme est beaucoup plus général que celui développé précédemment étant donné qu'on ne considère plus, comme c'était implicitement le cas, qu'une personne puisse pas avoir un comportement '*évolutionniste*', et

5. comme nous l'avons déjà mentionné au sein du chapitre II, quelle que soit la longueur de la période considérée, il existe une infinité de stratégies ayant des comportements similaires sans pour autant être complètement identiques.

Pour les raisons mentionnées ci-dessus, au terme de nos 'p' périodes, il est possible que l'ensemble '$\mathcal{E}^6$' soit vide, mais ceci n'a pas d'importance car, en ce cas, nous nous focalisons sur les caractéristiques de la stratégie de l'adversaire. En d'autres termes, s'il est clair que si la finalité est la même, la méthode pour y arriver est complètement nouvelle. Dans cette méthode, le langage de l'automate est constitué par les règles associées aux différentes caractéristiques des stratégies.

Notons enfin que cet algorithme se base sur les principes de l'algèbre de Boole. En effet, quand un événement est reconnu comme vrai, sa répétition n'augmente pas la connaissance. De même, dans le cadre de l'algorithme, lorsqu'on a découvert une caractéristique, on ne tente pas de la redécouvrir à nouveau mais par contre on essaye de mettre en exergue de nouvelles particularités.

Dans une première phase, il s'agit de classifier les différentes stratégies sur base de la méthode proposée au sein de la section I (Cadre de réflexion) de ce chapitre. Pour chacune des classes ainsi définies, on détermine un ensemble de règles logiques[12] permettant d'identifier les caractéristiques des stratégies la composant. Au fur et à mesure du comportement de l'adversaire, l'ordinateur exclut les familles dont le comportement adopté par l'individu enfreint les règles à la base de sa composition.

---

12 L'utilisation de langages de programmation de type Prolog semble constituer une alternative intéressante.

A la différence du formalisme adopté dans le cadre de la signature d'exclusion, il nous faut définir pour chacune des caractéristiques mentionnées l'ensemble des clauses la qualifiant. Dès lors, on peut dire que chaque caractéristique '*i*' est perçue comme un langage particulier L_i. Autrement dit, on considère que chaque caractéristique peut être représentée par une grammaire G_i qui lui est propre. L'analyseur de caractéristiques doit être dans la possibilité de dire si un item[13] reçu en entrée fait partie de la structure du langage de L_i. Pour ce, il faut définir une grammaire G_i pour L_i , qui peut s'écrire comme suit: $L(G_i)$. En d'autres termes, on peut dire que les règles de grammaire définissent la structure des mots de la caractéristique '*i*'.

Comme mentionné précédemment (Cfr. Farreny, H., et Ghallab, M. (1987)), il importe de définir un ensemble de lois (d'algorithmes) pour chacune des caractéristiques présentées dans le cadre du chapitre II. Nous passons ci-dessous en revue les quatre caractéristiques qui sont les plus significatives au niveau de la façon d'appréhender le comportement d'un individu. Pour chacune d'elles, après en avoir rappelé la définition, nous tenterons d'en donner dans un premier temps une représentation graphique respectant les conventions définies dans le cadre du chapitre précédent, par la suite nous essayerons de la traduire en Prolog[14]. L'adaptation des critères au Prolog se justifie pour deux raisons: d'une part, on ne crée que des relations logiques excessivement simples, et, d'autre part, ce langage adopte une méthode qu'on peut qualifier de '*complète*[15]'.

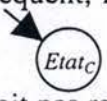
---

13 L'item sera ici composé de la signature d'exclusion et de l'action (réaction) de l'individu.

14 Il s'agit d'un langage de programmation logique.

15 Cela signifie que l'exécution d'un programme Prolog correspond à une tentative de construire un arbre. De ce fait, si on disposait de l'entièreté de l'arbre, toutes les conséquences logiques d'un programme seraient trouvées.

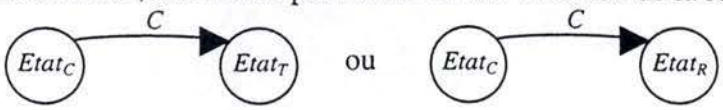
La sérénité

Celle-ci se définit par le souci de ne pas prendre l'initiative de la trahison, de ne pas être agressif. Par conséquent, le comportement d'un agent sera qualifiée de '*serein*' si à la première période on a: 

,et non 

ou 

. Par ailleurs, durant la confrontation, on ne doit pas retrouver une situation de la forme suivante:

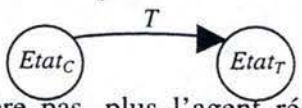


Si on utilise une formulation mettant en œuvre les règles de la logique du premier ordre, on peut traduire cette expression sur base des deux règles suivantes:

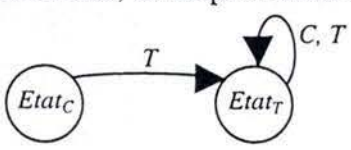
- DébutSérénité($Etat_C$).
- FlècheSérénité(C, $Etat_C$, $Etat_C$).

La réaction graduelle

Elle a pour objet de punir immédiatement l'autre agent quand celui-ci a trahi précédemment. Au plus l'adversaire adopte un comportement non-coopératif, au plus il est nécessaire de réagir durement. Cette caractéristique contient dès lors en elle deux particularités.

1. elle réagit au comportement non-coopératif de l'interlocuteur, ce qui peut être représenté comme suit: 

2. plus l'interlocuteur ne coopère pas, plus l'agent réagira de façon '*violente*'. Cette attitude n'est pas représentable à l'aide d'un automate fini. Toutefois, afin de prendre un tant soit peu cette notion en considération, on peut remarquer qu'adopter une réaction '*graduelle*' ne signifie pas qu'on soit rancunier. En d'autres termes, on ne peut avoir la situation suivante:



A défaut de pouvoir définir la fonction indiquant dans quelle mesure un comportement est graduel, et afin de prendre en considération cette deuxième particularité, nous sommes contraint de nous contenter de dire qu'une réaction graduelle n'est pas une réaction rancunière[16].

Si on utilise une formulation mettant en œuvre les règles de la logique du premier ordre, on peut traduire cette situation sur base des deux règles suivantes:

- FlècheRéactionGraduelle(T, $Etat_C$, $Etat_T$).
- FlècheRéactionGraduelle(C, $Etat_T$, $Etat_C$).

La simplicité

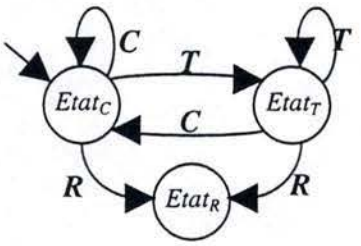
L'idée à la base de cette attitude est qu'une stratégie doit pouvoir être intelligible. Afin de pouvoir appréhender cette notion, nous optons pour la règle de Binmore, K. et Samuelson L. (1992) qui consiste à dire '*In measuring complexity, we simply count states*' (page 293). Toutefois, comme le font remarquer Abreu et Rubinstein (1988), ***cette complexité est endogène à chaque stratégie***. Ceci implique que ce n'est pas parce qu'une stratégie semble simple au premier abord qu'elle l'est réellement[17].

---

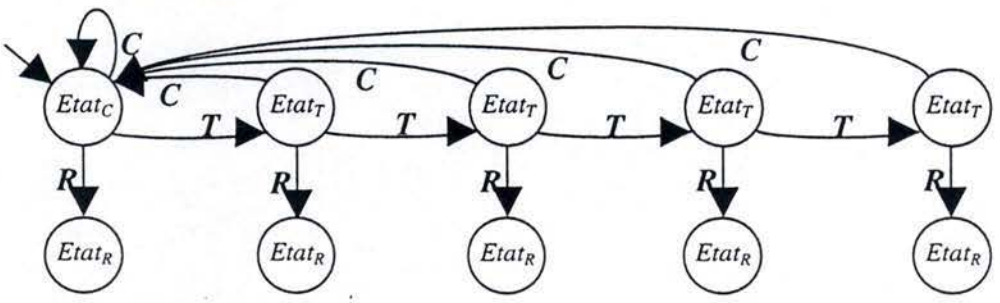
16 Il s'agit donc d'une définition de '*second best*'.

17 A cet égard, nous renvoyons le lecteur aux commentaires que nous avons émis au sein du chapitre V, section II (Cas des stratégies complexes).

Par conséquent, afin de déterminer la simplicité d'une stratégie, il est nécessaire, dans un premier temps, de déterminer son langage, et par la suite tenter de le transformer en un automate capable de constituer un ***nombre minimal*** d'états qui accepte le même langage. En effet, à partir d'un automate quelconque on peut en dériver une infinité qui reconnaissent le même langage soit en lui ajoutant un nombre quelconque d'états inaccessibles[18] soit en lui adjoignant des états accessibles. Prenons le cas de ***DONNANT-DONNANT***, normalement nous devons obtenir l'automate suivant:



Néanmoins, l'automate ci-dessous accepte exactement le même langage tout en contenant un nombre plus important d'états:



Par conséquent, on devra travailler en parallèle sur deux automates. Le premier, qualifié d'*automate primaire*, sera constitué par l'ensemble des états décelés lors des différentes confrontations. Le deuxième sera quant à lui l'*automate minimal* du premier. Ce dernier sera donc périodiquement recalculé[19]. Sur cette base, il suffira de fixer une règle

---

18 Il s'agit des états qui ne peuvent être atteints à partir de l'état initial. En d'autres termes, il n'existe pas de transition permettant d'y arriver. Etant donné la démarche que nous avons adoptée, une telle situation est totalement impossible.

19 On peut donc dire que l'automate primaire sera continuellement complété, alors que l'automate minimal est constamment remis en cause.

selon laquelle une stratégie sera considérée comme complexe si elle dépasse un nombre limité d'états, ce qui donne en formulation de règles de la logique du premier ordre[20]:

- CalculNombreEtats($Automate_A$, 0):- PremierEtat($Automate_A$) = ε.
- CalculNombreEtats($Automate_A$, N+1):- $Automate_B$ = $Automate_A$ - PremierEtat($Automate_A$), CalculNombreEtats($Automate_B$, N).
- NbInf(0, M):- M>0.
- NbInf(N+1, M+1):- NbInf(N, M).
- AutomateSimple($Automate_A$, Règle):- CalculNombreEtats($Automate_A$, N), NbInf(N, Règle).

En fait, cette règle semble quelque peu simpliste dans la mesure où elle est trop rigide. En effet, imaginons qu'on fixe la simplicité à 20 états au maximum. Peut-on considérer qu'un automate minimal qui possède 15 états après 30 coups soit aussi simple qu'un automate minimal qui possède 15 états après 3.000 coups? En d'autres termes, il semble plus pertinent de considérer qu'un automate soit simple en fonction de la longueur de la confrontation. Par conséquent, le paramètre '*Règle*' de la dernière clause est fonction de la période dans laquelle on se trouve.

Le seuil

Dans la variante du dilemme du prisonnier avec renoncement, il représente le cas limite au delà duquel le renoncement devient souhaitable. Pour ce faire, il faut calculer le payoff de l'individu et regarder si après une certain laps de temps il importe de renoncer du fait que celui-ci est inférieur à un certain niveau. Si on utilise une formulation mettant en œuvre les règles de la logique du premier ordre, on peut l'exprimer sur base des trois règles suivantes:

- PayoffAcc($Automate_A$, 0):- PremierEtat($Automate_A$) = ε.
- PayoffAcc($Automate_A$, N+M):- Act = DernierAction($Automate_A$), ActAdv = DernierActionAdversaire($Automate_A$), $Automate_B$ = $Automate_A$ - Act, Payoff(Act, ActAdv, M),PayoffAcc($Automate_B$, N).
- Seuil($Automate_A$, Période, S):- Période>= période, PayoffAcc($Automate_A$, Revenu), Revenu < S, Action($Automate_A$) = Renoncer.

---

20 Le littéral Premier nous donne l'état initial dans la liste des états d'un automate. Il renvoie ε si cette liste est vide.

De nouveau, cette façon d'agir semble simpliste dans la mesure où elle est trop rigide. En effet, il apparaît plus pertinent d'estimer qu'une stratégie doit tenir compte d'un seuil décisionnel si son comportement change en fonction du niveau de payoff obtenu par rapport à un niveau de référence, celui-ci évoluant au cours du temps.

Point II: Critique de la méthode

Comment faire face à une personne qui adopte un comportement qui est lui-même adaptatif par rapport à l'environnement dans lequel il évolue? Ce point a déjà été abordé dans le cas de la réaction graduelle mais est en fait également d'actualité pour les autres caractéristiques. Afin de résoudre ce problème, on peut pratiquer une politique acceptant de remettre en cause ce qu'on considérait comme acquis. On peut en effet décréter qu'*un individu puisse opter pour une certaine modification de sa stratégie tous les laps de temps de longueur 't'; par contre, s'il change plusieurs fois de comportement au sein de cette période 't', on peut prendre le parti qu'il s'agit d'une personne relativement instable.*

Par ailleurs, la détection de certains critères peut se révéler excessivement coûteuse en termes de temps de calcul et d'espace mémoire utilisé[21]. Néanmoins cette méthode a l'avantage de ne pas être limitée à la base de connaissances. Par ailleurs, si on classifie les stratégies, on peut obtenir un ordinateur qui soit en mesure d'accroître la capacité de celle-ci.

---

21 Il s'agit particulièrement de savoir si la stratégie est simple ou non.

Commentaires

Quelle que soit la méthode adoptée (la signature d'exclusion ou l'identification des caractéristiques d'une stratégie), on utilise les fondements de l'algèbre de Boole. En effet, on considère que lorsqu'un événement est réputé vrai, sa répétition n'augmente pas la connaissance. Comme nous l'avons déjà souligné précédemment, ce raisonnement nous pose quelques problèmes au niveau d'un adversaire s'adaptant au contexte dans lequel il évolue.

Une fois qu'on est parvenu à identifier le raisonnement adopté par l'adversaire, il faut mettre en œuvre des processus d'apprentissage afin que l'automate puisse calculer la meilleure riposte possible[22]. Pour ce faire, on peut envisager d'adopter des méthodes du type '***Case Based Reasoning***' (C.B.R.). L'idée centrale de cette méthode est que le raisonnement est élaboré par analogie à des situations antérieures plutôt que par la mise en application de règles de base. Par conséquent, l'apprentissage est l'élément central du C.B.R.. Cette méthode consiste à suivre le processus suivant:

1. retrouver, au sein de la base de connaissances, les situations les plus proches de celle qu'on est en train d'analyser[23],
2. sélectionner les cas les plus appropriés,
3. construire (élaborer) une solution de '*baseline* [24] ', et
4. évaluer la solution ainsi obtenue par rapport à la situation actuelle.

Mais nous touchons là l'étape ultime d'un projet dont ce travail ne constitue qu'un premier jalon. Nous n'aborderons dès lors pas davantage ce dernier point.

---

22 Dans ce contexte, on émet l'hypothèse que les joueurs sont dans la possibilité de calculer la meilleure réplique compte tenu de la population.

23 D'où l'intérêt de créer une classification entre les différentes stratégies sur base de leurs critères respectifs.

24 Il s'agit de la meilleure solution lorsqu'on prend en considération seulement les caractéristiques les plus importantes de la situation dans laquelle on se trouve.

Conclusion

Après avoir mis sous forme algorithmique les différentes propositions présentées dans les deux premiers chapitres, nous avons esquissé plusieurs pistes de réflexion permettant d'entrer dans le domaine de l'Intelligence Artificielle.

Dans cette optique, nous avons concentré nos recherches sur la détermination de la stratégie utilisée par son adversaire. Dans une première méthode, nous avons assimilé chaque stratégie à un langage particulier. Par l'utilisation d'une signature d'exclusion nous nous sommes efforcé de déterminer la signature permettant d'en exclure le plus grand nombre. Mais ce raisonnement comportait un certain nombre de lacunes et nous a amenés à rechercher un processus plus approprié.

Dans la deuxième méthode, nous avons considéré les caractéristiques des stratégies comme un langage particulier. En plus d'être plus pertinente, cette logique nous a permis d'introduire la notion de classe de famille. Cette classification nous a laissé la possibilité d'aborder de façon pertinente la problématique de l'Intelligence Artificielle, à savoir: concevoir un logiciel qui soit susceptible de déceler le comportement optimal en fonction de l'environnement.

Notre travail s'inscrit dans un projet s'étendant sur plusieurs années. Dans cette optique nous avons apporté un intérêt particulier à concevoir un programme modulaire suffisamment souple pour être adapté aux développements ultérieurs. Ceux qui prendront la relève auront la faculté d'implémenter des fonctionnalités nouvelles comme la sauvegarde et le chargement du résultat d'une confrontation, la visualisation du taux de coopération, de trahison ou de renoncement au sein de la population[1]. De même, ils pourront développer la mise en œuvre du concept de stratégies évolutionnairement stables et la définition de nouveaux modes de calcul de payoffs permettant de faire évoluer la population.

Dans une deuxième étape, après avoir maîtrisé ces fonctionnalités, ils pourront envisager l'objectif ultime de ces recherches: d'une part implémenter la fonction de recherche de la stratégie adoptée par l'adversaire, de l'autre déterminer le comportement optimal face à une telle stratégie.

---

[1] Notons que ces différentes fonctionnaités, quoique relativement simples, ont le mérite de permettre une prise de contact plus aisée avec le code source existant.

Annexe

Annexe: Le code source

Remarque préliminaire

Cette annexe a pour but de permettre une meilleure compréhension du code source constituant notre logiciel. Afin d'accroître la lisibilité de cette partie, nous reprenons ci-dessous la liste des fonctions[1] et des variables globalesutilisées en y indiquant la référence du fichier dans lequel elles sont implémentées. Nous donnons ensuite le code des différents fichiers, classés par ordre alphabéthique.

Calcul_Payoffs.cpp

info_init_strat *Search_Ptr_Elem_List_Strat(unsigned int Ident)
card Search_Behavior_Adv(bool Collectif, unsigned int Num_Adv, unsigned int Num_Moi)
unsigned int Calcul_Payoff(type_agent Type_Indiv, card Behavior_Myself, card Behavior_Adv, Alternative Valeur_Payoffs)
unsigned int Calcul_Payoff_Renoncement(type_agent Type_Indiv, Alternative Valeur_Payoffs)
void Calcul_result_par_strat(Alternative Valeur_Payoffs)

Confrontation.cpp

unsigned int Valeur_Prob_Coop(unsigned int i)
Info_Gen *Affrontement(info_indiv *Individu, unsigned int Num, bool Confront_indiv)
void Confrontation_One_Periode(info_indiv *Individu)
void Confrontation(Alternative Payoffs)

Gestion_Pop.cpp

info_indiv *Create_One_Individual(unsigned int Id_Strat)
Nb_Instance_Strat *Create_One_Box_Nb_Instance_Strat(info_init_strat *list_tmp)
info_indiv *Constitute_Pop_Init()
Nb_Instance_Strat *Constitute_List_Nb_Instance_Strat()
Info_Gen *Create_Info_Box(unsigned int Numero, card acte)
void MAJ_Box_Info_Strat(bool Confrontation_Indiv, unsigned int Numero_Adv, unsigned int Numero_Indiv)

[1] Pour être plus exacte, nous nous contentons de donner, au sein de cette liste, les fonctions qui ne sont pas directement liées à une composante de notre interface.

void MAJ_Champs(bool Probabiliste, info_indiv *Strat_Etud, Info_Gen *Strat_Adv)
void MAJ_Info_Adv()
int Estimation_Var_Pop_Strat(info_init_strat *List_Strat_Selec_Tmp, float Max, float Payoff_Medium)
unsigned int Nbr_People_Tot()
unsigned int Nbr_Strat()
unsigned int Num_Strat_In_Table(info_init_strat *Type_Strat)
unsigned int Nbr_Strat_Rest()
Info_Gen *Kill_list_Info_Gen(Info_Gen *list)
void Kill_list_indiv()
void Kill_list_Strat_Selec()
void Kill_Box_Indiv(info_indiv *Elem_To_Kill)
Info_Gen *Kill_One_Box(Info_Gen *Ptr, unsigned int Num_Strat)
void Kill_Box_Superflous_Non_Proba(unsigned int Num_Strat)
void Kill_Box_Superflous_Info_Proba(unsigned int Num_Indiv, unsigned int Num_Strat)
info_indiv *Kill_Ind(unsigned int Num_Strat, int Destructor)
void Kill_Elem_list_Strat_Selec(AnsiString s)
Info_Gen *Copy_Box_Info_Non_Proba(Info_Gen *Ptr)
Info_Gen *Copy_List_Info_Non_Proba(Info_Gen *Ptr)
void Add_Ind(unsigned int Num_Strat, int variation)
void Ajustement_Pop_Var(float Payoff_Medium, unsigned int period, unsigned int Nb_Tot_Indiv_Pop)
unsigned int Estimation_Pop_End_Period_Var_Medium(float Payoff_Per_Indiv)
void MAJ_List_Nb_Instance_Strat(info_init_strat *Ptr, unsigned int Nb_Instance_End)

IHM.cpp

void Aff_Couleur_Strat()
void IHM_Selection_Strat_Incorrecte()
void IHM_Selection_Strat_Correcte(unsigned int i)
void MAJ_Plot_Evol_Pop(unsigned int Beginning_Period, unsigned int Proportion_Ind_Beginning, info_init_strat void MAJ_Affich_Result(info_init_strat *Ptr_Tmp, float Proportion_End, unsigned int Nb_Instance_End)
void MAJ_StatusBarPrincipal(int Period)
void Aff_Result_Confront_Var_Medium(float Payoff_Medium_Per_Indiv, int Period, unsigned int Nb_Eff_Pop_Tot_Beginning, unsigned int Nb_Eff_Pop_Tot_End)
void Aff_Fin_Confrontation(unsigned int Period, unsigned int Nb_Strat_Diff_Rest)
void Progression_Pas_A_Pas()
void Affichage_Tableau_Resultat()

Matrice.cpp

bool Verif_Param_DP_Clas(int *C, int *T, int *D, int *P)

Mode_Evolution_Population.cpp

void Mode_Evol_Pop(int periode)

OptionSaisieStrat.cpp

void Save_Val_Ptr_Elem_List()
int Num_Strat_In_Combo_Box(AnsiString s)
bool Verification_Valeurs_Introduites()

SaisieStratConfr.cpp

void Find(AnsiString s)
unsigned int Long_List_Strat_Selec()
int Selection_Couleur()
void Insert_Elem(int i)
Alternative Initialisation_Payoff()

Strategies.cpp

card Gentille(info Pas_info)
card Mechant(info Pas_info)

```
card Donnant_Donnant(info Behaviour_Opp)
card Rancunier(info Info_Ranc)
card Per_Mechant(info Pas_Info)
card Per_Gentille(info Pas_Info)
card Maj_Mou(info Par_Maj_Mou)
card Maj_dur(info Par_Maj_Mou)
card Sondeur_1(info Info_Sonde)
card Tit_For_Tat_Dur(info Par_TFT_Dur)
card Mefiant(info Last_Behaviour_Opp)
card Lunatique(info prob)
```

Calcul_Payoffs.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Calcul_Payoffs.h"
#include "TypeDef.h"
#include "Confrontation.h"

extern info_init_strat *list_Strat_Selec;
extern info_indiv* Liste_Pop;
```

```
info_init_strat *Search_Ptr_Elem_List_Strat(unsigned int Ident)
{
info_init_strat *Ptr_tmp_list_strat_selec = list_Strat_Selec;
while (Ptr_tmp_list_strat_selec→N_Strat != Ident)
    {
    Ptr_tmp_list_strat_selec = Ptr_tmp_list_strat_selec→Next;
    }
return (Ptr_tmp_list_strat_selec);
}
```

```
card Search_Behavior_Adv(bool Collectif, unsigned int Num_Adv, unsigned int Num_Moi)
{
/* But: Renvoyer le comportement d'un individu ou d'un type de stratégie (identifier par 'Num_Adv') par rapport à un
individu ou par rapport à un type de stratégie (identifier par 'Num_Moi'). */
info_indiv *Liste_Pop_Tmp = Liste_Pop;
Info_Gen *Ptr_Info_Gen;
if (Collectif == true)
      {
      while ( (Liste_Pop_Tmp != NULL) && (Liste_Pop_Tmp→Ident_Strat != Num_Adv) )
          {
          Liste_Pop_Tmp = Liste_Pop_Tmp→Next;
          }
      Ptr_Info_Gen = Liste_Pop_Tmp→Info_Non_Proba;
      }
else  {
      while ( (Liste_Pop_Tmp != NULL) && (Liste_Pop_Tmp→Ident_Indiv != Num_Adv) )
          {
          Liste_Pop_Tmp = Liste_Pop_Tmp→Next;
          }
      /* Ce qui est ci-dessous devrait être simplifiable. */
      if (Liste_Pop_Tmp != NULL)
            {
            Ptr_Info_Gen = Liste_Pop_Tmp→Info_Proba;
            }
      else  {
            Ptr_Info_Gen = NULL;
            }
      }
/* 'Liste_Population_Tmp' pointe vers la boîte (mère) désirée. Il nous faut à présent passer en revue la liste des sous-
boîtes. Le pointeur 'Ptr_Info_Gen' se positionne alors au début de la sous-liste devant être parcourue. */
while ( (Ptr_Info_Gen!=NULL) && (Ptr_Info_Gen→Ident != Num_Moi) )
    {
```

```
    Ptr_Info_Gen = Ptr_Info_Gen→Next;
    }
if (Ptr_Info_Gen!=NULL)
    {
    return (Ptr_Info_Gen→Action);
    }
else {
    ShowMessage("Problème au niveau de la recherche du comportement d'un individu");
    return(Abandon);
    }
}
```

```
unsigned int Calcul_Payoff(type_agent Type_Indiv, card Behavior_Myself, card Behavior_Adv, Alternative
Valeur_Payoffs)
{
/* But: Sur base des payoffs déterminés par l'utilisateur, du comportement d'un individu et de celui de son
homologue, on renvoie le payoff obtenu par l'individu sous analyse. */
unsigned int resultat;
switch (Type_Indiv)
    {
    case Val_Type:    if (Behavior_Myself == Cooperation)
                          {
                          if (Behavior_Adv == Cooperation)
                              {
                              resultat = Valeur_Payoffs.Valeur_Type.Coop;
                              }
                          else {
                              resultat = Valeur_Payoffs.Valeur_Type.Naive;
                              }
                          }
                      else {
                          if (Behavior_Adv == Cooperation)
                              {
                              resultat = Valeur_Payoffs.Valeur_Type.Betrayal;
                              }
                          else {
                              resultat = Valeur_Payoffs.Valeur_Type.Punishment;
                              }
                          }
                      break;
    case Agent_A:     if (Behavior_Myself == Cooperation)
                          {
                          if (Behavior_Adv == Cooperation)
                              {
                              resultat = Valeur_Payoffs.Individu_A.Coop;
                              }
                          else {
                              resultat = Valeur_Payoffs.Individu_A.Naive;
                              }
                          }
```

```
                    else {
                        if (Behavior_Adv == Cooperation)
                            {
                            resultat = Valeur_Payoffs.Individu_A.Betrayal;
                            }
                        else {
                            resultat = Valeur_Payoffs.Individu_A.Punishment;
                            }
                        }
                    break;
    case Agent_B:   if (Behavior_Myself == Cooperation)
                        {
                        if (Behavior_Adv == Cooperation)
                            {
                            resultat = Valeur_Payoffs.Individu_B.Coop;
                            }
                        else {
                            resultat = Valeur_Payoffs.Individu_B.Naive;
                            }
                        }
                    else {
                        if (Behavior_Adv == Cooperation)
                            {
                            resultat = Valeur_Payoffs.Individu_B.Betrayal;
                            }
                        else {
                            resultat = Valeur_Payoffs.Individu_B.Punishment;
                            }
                        }
                    break;
    default:    ShowMessage("Type d'individu inconnu!");
                break;
    }
return (resultat);
}
```

```
unsigned int Calcul_Payoff_Renoncement(type_agent Type_Indiv, Alternative Valeur_Payoffs)
{
/* But: Sur base des payoffs déterminés par l'utilisateur, on renvoie le payoff obtenu lorsque les deux individus ont
choisi d'abandonner. */
unsigned int resultat;
switch (Type_Indiv)
    {
    case Val_Type:      resultat = Valeur_Payoffs.Valeur_Type.Abandonment;
                        break;
    case Agent_A:       resultat = Valeur_Payoffs.Individu_A.Abandonment;
                        break;
    case Agent_B:       resultat = Valeur_Payoffs.Individu_B.Abandonment;
                        break;
    default:    ShowMessage("Type d'individu inconnu!");
                break;
    }
return (resultat);
}
```

```
void Calcul_result_par_strat(Alternative Valeur_Payoffs)
{
/* Le but de cette fonction est de mettre à jour les champs 'Payoff_acc_Preced' et 'Payoff_acc_Actuel' de la variable
globale 'list_Strat_Selec'. */
info_indiv *Liste_Pop_Tmp = Liste_Pop;
Info_Gen *Ptr_Info_Gen;
card Behavior;
unsigned int entier_tmp; /* Cette variable permettra de déterminer le nombre d'instances d'une stratégie
particulière.*/
unsigned int i, j, Val_interm;
info_init_strat *Ptr_Elem_List_Init_Strat, *Ptr_Tmp_Elem_List_Init_Strat = list_Strat_Selec;
while (Ptr_Tmp_Elem_List_Init_Strat != NULL)
    {
    Ptr_Tmp_Elem_List_Init_Strat→Payoff_acc_Preced =
    Ptr_Tmp_Elem_List_Init_Strat→Payoff_acc_Actuel;
    Ptr_Tmp_Elem_List_Init_Strat = Ptr_Tmp_Elem_List_Init_Strat→Next;
    }
while (Liste_Pop_Tmp != NULL)
    {
    i = Liste_Pop_Tmp→Ident_Strat; /* Identifiant pour les stratégies non probabilistes. */
    j = Liste_Pop_Tmp→Ident_Indiv; /* Identifiant pour les stratégies probabilistes. */
    Ptr_Elem_List_Init_Strat = Search_Ptr_Elem_List_Strat(i);
    /* Cette séquence concerne les stratégies qu'on peut traiter en groupe. */
    Ptr_Info_Gen = Liste_Pop_Tmp→Info_Non_Proba;
    while (Ptr_Info_Gen != NULL)
        {
        if (Ptr_Info_Gen→Action != Abandon)
            {
            Behavior = Search_Behavior_Adv(true, Ptr_Info_Gen→Ident, i);
            Val_interm =
                Calcul_Payoff(Ptr_Elem_List_Init_Strat→Agent, Ptr_Info_Gen→Action, Behavior, Valeur_Payoffs);
            }
        else {
            Val_interm =
                Calcul_Payoff_Renoncement(Ptr_Elem_List_Init_Strat→Agent, Valeur_Payoffs);
            }
        /* La variable 'Ptr_Elem_List_Init_Strat' permet de pointer vers la boîte de la stratégie de l'adversaire dans
        le but de déterminer le numéro de sa stratégie */
        Ptr_Elem_List_Init_Strat = Search_Ptr_Elem_List_Strat(Ptr_Info_Gen→Ident);
        if (i == Ptr_Info_Gen→Ident)
            {
            entier_tmp = Ptr_Elem_List_Init_Strat→Nb_Instance - 1;
            }
        else {
            entier_tmp = Ptr_Elem_List_Init_Strat→Nb_Instance;
            }
        Val_interm *= entier_tmp;
        /* La variable 'Ptr_Elem_List_Init_Strat' permet de pointer vers la boîte de la stratégie de l'individu examiné
        pour enregistrer le résultat obtenu. */
        Ptr_Elem_List_Init_Strat = Search_Ptr_Elem_List_Strat(i);
        Ptr_Elem_List_Init_Strat→Payoff_acc_Actuel += Val_interm;
        Ptr_Info_Gen = Ptr_Info_Gen→Next;
        }
    /* A présent on se charge des stratégies qu'il faut traiter séparément. */
    Ptr_Info_Gen = Liste_Pop_Tmp→Info_Proba;
```

```
    while (Ptr_Info_Gen != NULL)
        {
        if (Ptr_Info_Gen→Action != Abandon)
            {
            Behavior = Search_Behavior_Adv(false, Ptr_Info_Gen→Ident, j);
            Val_interm =
                Calcul_Payoff(Ptr_Elem_List_Init_Strat→Agent, Ptr_Info_Gen→Action, Behavior, Valeur_Payoffs);
            }
        else {
            Val_interm =
                Calcul_Payoff_Renoncement(Ptr_Elem_List_Init_Strat→Agent, Valeur_Payoffs);
            }
        if (j != Ptr_Info_Gen→Ident)
          {
          Ptr_Elem_List_Init_Strat→Payoff_acc_Actuel += Val_interm;
          }
        Ptr_Info_Gen = Ptr_Info_Gen→Next;
        }
    Liste_Pop_Tmp = Liste_Pop_Tmp→Next;
    }
}
```

Confrontation.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Confrontation.h"
#include "TypeDef.h"
#include "Gestion_Pop.h"
#include "Strategies.h"
#include "SaisieStratConfr.h"
#include "Principal.h"
#include "IHM.h"
#include "Calcul_Payoffs.h"
#include "VariablesGlobales.h"
#include "Mode_Evolution_Population.h"

extern info_indiv *Constitute_Pop_Init();
extern Nb_Instance_Strat *Constitute_List_Nb_Instance_Strat();
extern info_indiv *Create_One_Individual(unsigned int Id_Strat);
extern Info_Gen *Create_Info_Box(unsigned int Numero, card acte);
extern void MAJ_Box_Info_Strat(bool Confrontation_Indiv, unsigned int Numero_Adv, unsigned int Numero_Indiv);
extern void MAJ_Info_Adv();
extern unsigned int Nbr_People_Tot();
extern void Calcul_result_par_strat(Alternative Valeur_Payoffs);
extern unsigned int Nbr_Strat();
extern unsigned int Nbr_Strat_Rest();
extern void IHM_Selection_Strat_Incorrecte();
extern void IHM_Selection_Strat_Correcte(unsigned int i);
extern void Aff_Fin_Confrontation(unsigned int periode, unsigned int Nb_Strat_Diff_Rest);
extern void Progression_Pas_A_Pas();
extern void Mode_Evol_Pop(int i);

extern info_init_strat *list_Strat_Selec;
extern info_indiv *Liste_Pop;
extern Nb_Instance_Strat *List_Nb_Instance_Strat;
extern Tab_Strat Tableau_Strat;
extern int Max_Conf;
extern int Last;
extern int Nb_Individu;
extern bool PasSuivant;
extern bool Confront_In_The_Past;
```

```
unsigned int Valeur_Prob_Coop(unsigned int i)
{
/* Dans le cas d'une stratégie probabiliste, on recherche la valeur de la probabilité de coopérer. */
info_init_strat *ptr_elem_list = list_Strat_Selec;
unsigned int val_prob_coop;
while ( (ptr_elem_list!=NULL) && (ptr_elem_list→N_Strat!=i) )
    {
    val_prob_coop = ptr_elem_list→Proba;
    ptr_elem_list = ptr_elem_list→Next;
    }
return (val_prob_coop);
}
```

```
Info_Gen *Affrontement(info_indiv *Individu, unsigned int Num, bool Confront_indiv)
{
card action;
info inf_param;
Tab_Strat *tab_tmp = &Tableau_Strat;
unsigned int i = Individu→Ident_Strat;
/* 'i' donne l'indice de la stratégie au sein du tableau reprenant l'ensemble de celles-ci. */
typedef card (*type_fct) (info);
type_fct fct = tab_tmp[i].Adresse;
Info_Gen *Ptr_tmp;
Info_Gen *Result;
if (Confront_indiv == false)
     {
     Ptr_tmp = Individu→Info_Non_Proba;
     }
else {
     Ptr_tmp = Individu→Info_Proba;
     }
Result = Ptr_tmp;
/* On se positionne à l'endroit souhaité (si c'est possible). */
while ( (Ptr_tmp!=NULL) && (Ptr_tmp→Ident!=Num) )
   {
   /* Dans un premier temps, on regarde s'il n'existe pas de boîte relative à la stratégie de l'adversaire. On sort de
   la boucle si 'Ptr_tmp' vaut NULL, ce qui signifie qu'aucune boîte ne se rapporte à la stratégie de l'adversaire;
   sinon 'Ptr_tmp' est différente de NULL. */
   Ptr_tmp = Ptr_tmp→Next;
   }

if (Ptr_tmp == NULL)
     {
     /* On examine ici le cas où deux stratégies n'ont pas encore été confrontées l'une à l'autre. */
     switch (tab_tmp[i].Param)
        {
        case Pas_Param:     /* En réalité, 'inf_param' a n'importe quelle valeur, mais cela n'a pas d'importance car
                            ce type de stratégies n'utilisent pas de paramètres lors de leur exécution. */
                            action = (*fct)(inf_param);
                            break;
        case Ranc:    inf_param.info_ranc.adv = Cooperation;
                      inf_param.info_ranc.moi = Cooperation;
                      action = (*fct)(inf_param);
                      break;
        case Last_Behaviour_Opponent:        inf_param.Last_Behaviour_Opp = Cooperation;
                                             action = (*fct)(inf_param);
                                             break;
        case Param_Nb_Trahison:      inf_param.Info_Nbre_Trahison.Nb_Trahison = 0;
                                     action = (*fct)(inf_param);
                                     break;
        case Param_Sondage:      inf_param.Info_Sondage.Behavior_Second_Perriod = Cooperation;
                                 inf_param.Info_Sondage.Behavior_Third_Perriod = Cooperation;
                                 inf_param.Info_Sondage.Last_Behaviour_Opp = Cooperation;
                                 action = (*fct)(inf_param);
                                 break;
        case Param_TFT_Dur:        inf_param.TFT_Dur.Beavior_Two_Period_Before = Cooperation;
                                   inf_param.TFT_Dur.Last_Behaviour_Opp = Cooperation;
                                   action = (*fct)(inf_param);
                                   break;
        case Proba:   inf_param.prob = Valeur_Prob_Coop(i);
                      action = (*fct)(inf_param);
```

```
                    break;
            default:    ShowMessage("Problème au niveau de l'exécution d'une stratégie!");
                        break;
            }
        /* Cette séquence est nécessaire pour enregistrer, au sein d'une nouvelle boîte le comportement de la stratégie
        analysée par rapport à la stratégie (ou au type de stratégie) à laquelle elle est confrontée. */
        Info_Gen *Tmp = Create_Info_Box(Num, action);
        Tmp→Next = Result;
        if (Result != NULL)
            {
            Result→Previous = Tmp;
            }
        Result = Tmp;
        }
    else {
        /* Traitement de la deuxième hypothèse, à savoir le fait que les deux stratégies ont déjà été confrontées l'une à
        l'autre. */
        if (Ptr_tmp→Action != Abandon)
            {
            /* Si lors de tours précédents, l'individu n'a pas opté pour l'abandon et si son adversaire n'a pas encore
            opté pour cette même alternative, alors l'individu traité a encore le choix entre 'Cooperation', 'Defection'
            et 'Abandon'.*/
            switch (tab_tmp[i].Param)
                {
                case Pas_Param:     /* En réalité, 'inf_param' a n'importe quelle valeur, mais cela n'a pas
                                    d'importance car ce type de stratégie n'utilisent pas de paramètres lors de son
                                    exécution. */
                                    action = (*fct)(inf_param);
                                    break;
                case Ranc:      inf_param.info_ranc.adv =
                                        Ptr_tmp→Info_Strat_Adv.info_ranc.adv;
                                inf_param.info_ranc.moi = Ptr_tmp→Info_Strat_Adv.info_ranc.moi;
                                action = (*fct)(inf_param);
                                break;
                case Last_Behaviour_Opponent:
                    inf_param.Last_Behaviour_Opp = Ptr_tmp→Info_Strat_Adv.Last_Behaviour_Opp;
                    action = (*fct)(inf_param);
                    break;
                case Param_Nb_Trahison:      inf_param.Info_Nbre_Trahison.Nb_Trahison =
                                    Ptr_tmp→Info_Strat_Adv.Info_Nbre_Trahison.Nb_Trahison;
                                        action = (*fct)(inf_param);
                                        break;
                case Param_Sondage:      inf_param.Info_Sondage.Behavior_Second_Perriod =
                                    Ptr_tmp→Info_Strat_Adv.Info_Sondage.Behavior_Second_Perriod;
                                        inf_param.Info_Sondage.Behavior_Third_Perriod =
                                    Ptr_tmp→Info_Strat_Adv.Info_Sondage.Behavior_Third_Perriod;
                                        inf_param.Info_Sondage.Last_Behaviour_Opp =
                                    Ptr_tmp→Info_Strat_Adv.Info_Sondage.Last_Behaviour_Opp;
                                        action = (*fct)(inf_param);
                                        break;
                case Param_TFT_Dur:      inf_param.TFT_Dur.Beavior_Two_Period_Before =
                                    Ptr_tmp→Info_Strat_Adv.TFT_Dur.Beavior_Two_Period_Before;
                                        inf_param.TFT_Dur.Last_Behaviour_Opp =
                                    Ptr_tmp→Info_Strat_Adv.TFT_Dur.Last_Behaviour_Opp;
                                        action = (*fct)(inf_param);
                                        break;
                case Proba:     inf_param.prob = Valeur_Prob_Coop(i);
```

```
                        action = (*fct)(inf_param);
                        break;
            default:    ShowMessage("Problème au niveau de l'exécution d'une stratégie.");
                        break;
                }
            }
        else {
            action = Abandon;
            }
        Ptr_tmp→Action = action;
        }
/* Si la stratégie de l'agent est d'abandonner, alors son adversaire devra adopter le même comportement qu'il le
veuille ou non. C'est ce qu'on met en oeuvre ci-dessous. */
if (action == Abandon)
    {
    if (Confront_indiv == true)
        {
        MAJ_Box_Info_Strat(true, Num, Individu→Ident_Indiv);
        }
    else {
        MAJ_Box_Info_Strat(false, Num, Individu→Ident_Strat);
        }
    }
return(Result);
}
```

```
void Confrontation_One_Periode(info_indiv *Individu)
{
Tab_Strat *tab_tmp = &Tableau_Strat;
info_indiv *tmp = Liste_Pop;
unsigned int i,j;
i = Individu→Ident_Strat;
while (tmp != NULL)
    {
    if (tmp→Ident_Indiv != Individu→Ident_Indiv)
        {
        j = tmp→Ident_Strat;
        if ( (tab_tmp[i].Flag==Strat_Non_Prob) && (tab_tmp[j].Flag==Strat_Non_Prob) )
            {
            /* Traitement du cas où les deux stratégies sont non probabilistes. */
            Individu→Info_Non_Proba = Affrontement(Individu, j, false);
            while ((tmp!=NULL)&&(tmp→Ident_Strat==j))
                {
                /* Cette boucle a pour effet de ne pas tenir compte des stratégies non probabilistes identiques à
                celle qui vient d'être traitée. */
                tmp=tmp→Next;
                }
            }
        else {
            /* Cas où au moins l'une des deux stratégies est non probabiliste. */
            Individu→Info_Proba = Affrontement(Individu, tmp→Ident_Indiv, true);
            tmp = tmp→Next;
            }
        }
```

```
    else {
        tmp = tmp→Next;
        }
    }
}

void Confrontation(Alternative Payoffs)
{
unsigned int Nb_Strategies = Nbr_Strat();
if (Nb_Strategies < 2)
    {
    IHM_Selection_Strat_Incorrecte();
    }
else {
    // Instance de confrontation pour deux stratégies différentes.
    Confront_In_The_Past = true;
    // Adaptation de l'interface.
    IHM_Selection_Strat_Correcte(Nb_Strategies);
    // Constitution de la population.
    Nb_Individu = 0; /* Remise à zéro du compteur permettant de déterminer le numéro identifiant d'un individu au
        sein de la population.*/
    Liste_Pop = Constitute_Pop_Init();
    List_Nb_Instance_Strat = Constitute_List_Nb_Instance_Strat();
    int periode = 1;
    info_indiv *Liste_Pop_Tmp; /* Liste_Pop_Tmp est nécessaire pour 'voyager' au sein de la population. */
    unsigned int Nb_Strat_Diff_Rest = Nbr_Strat_Rest();
    // Début de la confrontation générale.
    while ( (periode <= Max_Conf) && (Nb_Strat_Diff_Rest > 1) )
        {
        /* Les différentes instructions ci-dessous prennent en compte la confrontation lors d'une période déterminée.
        En outre, on met à jour les payoffs accumulés par stratégie et le nombre d'individus composant chaque
        stratégie. */
        Last = periode - 1; /* On initialise la variable à la valeur de 'periode' moins 1 (car 'periode' a pour valeur
        initiale 1 or 'Last' doit débuter à 0). */
        // Début de la confronation pour une période déterminée.
        Liste_Pop_Tmp = Liste_Pop;
        while (Liste_Pop_Tmp != NULL)
            {
            Confrontation_One_Periode(Liste_Pop_Tmp);
            Liste_Pop_Tmp = Liste_Pop_Tmp→Next;
            }
        // Enregistrement des informations nécessaires relatives à chaque adversaire ou à chaque type d'adversaire.
        MAJ_Info_Adv();
        Calcul_result_par_strat(Payoffs);
        Mode_Evol_Pop(periode); /* Cette fonction donne l'affichage des différentes stratégies en fonction du mode
                d'évolution de la population choisi par l'utilisateur. */
        // Gestion du fait que la progression de la confrontation se fait pas à pas ou en continu.
        Progression_Pas_A_Pas();
        periode++;
        Nb_Strat_Diff_Rest = Nbr_Strat_Rest();
        }
    Aff_Fin_Confrontation(periode-1, Nb_Strat_Diff_Rest);
    }
}
```

DefMat.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "DefMat.h"
#include "Matrice.h"
#include "VarParDefaut.cpp"
#include "MatriceValType.h"
#pragma resource "*.dfm"
```

```
TDefMatForm *DefMatForm;

extern int N_CC_A, N_TC_A, N_CT_A, N_PP_A, N_CC_B, N_TC_B, N_CT_B, N_PP_B, Renonc_A, Renonc_B;
extern int DA_CC_A,   DA_TC_A,  DA_CT_A,  DA_PP_A,  DA_CC_B,  DA_TC_B,  DA_CT_B,  DA_PP_B,
Renonc_A_DA, Renonc_B_DA;
```

```
__fastcall TDefMatForm::TDefMatForm(TComponent* Owner): TForm(Owner)
{
}
```

```
void __fastcall TDefMatForm::FormResize(TObject *Sender)
{
DefMatForm->Width = 363;
DefMatForm->Height = 223;
}
```

```
void __fastcall TDefMatForm::BtnDefMatClick(TObject *Sender)
{
if (RG_Type_Confrontation->ItemIndex==0)
    {
    MatriceForm->Caption = "Le dilemme du prisonnier itéré classique";
    MatriceForm->ME_CC_A->Text = N_CC_A;
    MatriceForm->ME_CC_B->Text = N_CC_B;
    MatriceForm->ME_CT_A->Text = N_CT_A;
    MatriceForm->ME_CT_B->Text = N_CT_B;
    MatriceForm->ME_TC_A->Text = N_TC_A;
    MatriceForm->ME_TC_B->Text = N_TC_B;
    MatriceForm->ME_PP_A->Text = N_PP_A;
    MatriceForm->ME_PP_B->Text = N_PP_A;
    MatriceForm->Ed_Renonc_A->Text = Renonc_A;
    MatriceForm->Ed_Renonc_B->Text = Renonc_B;
    MatriceForm->Hide();
    MatriceForm->ShowModal();
    }

if (RG_Type_Confrontation->ItemIndex==1)
    {
    MatriceForm->Caption = "Le retour d'ascenseur";
    MatriceForm->ME_CC_A->Text = DA_CC_A;
    MatriceForm->ME_CC_B->Text = DA_CC_B;
    MatriceForm->ME_CT_A->Text = DA_CT_A;
```

```
    MatriceForm->ME_CT_B->Text = DA_CT_B;
    MatriceForm->ME_TC_A->Text = DA_TC_A;
    MatriceForm->ME_TC_B->Text = DA_TC_B;
    MatriceForm->ME_PP_A->Text = DA_PP_A;
    MatriceForm->ME_PP_B->Text = DA_PP_A;
    MatriceForm->Ed_Renonc_A->Text = Renonc_A_DA;
    MatriceForm->Ed_Renonc_B->Text = Renonc_B_DA;
    MatriceForm->Hide();
    MatriceForm->ShowModal();
    }
}
```

```
void __fastcall TDefMatForm::BtnValTypeClick(TObject *Sender)
{
char *Str_CC_A, *Str_CC_B, *Str_TC_A, *Str_TC_B, *Str_CT_A, *Str_CT_B, *Str_PP_A, *Str_PP_B;
int n = sizeof(int);
Str_CC_A = new char[n+1];
Str_CC_B = new char[n+1];
Str_TC_A = new char[n+1];
Str_TC_B = new char[n+1];
Str_CT_A = new char[n+1];
Str_CT_B = new char[n+1];
Str_PP_A = new char[n+1];
Str_PP_B = new char[n+1];
int val= 2*(n+3);
char *valeur;
valeur= new char[val];
if (RG_Type_Confrontation->ItemIndex==0)
    {
    extern int DP_Clas_N, DP_Clas_P, DP_Clas_R, DP_Clas_T;
    MatriceValTypeForm->Caption = "Le dilemme du prisonnier itéré classique";
    itoa(DP_Clas_R, Str_CC_A, 10);
    itoa(DP_Clas_R, Str_CC_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_CC_A,Str_CC_B);
    MatriceValTypeForm->PanelCC->Caption = valeur;
    itoa(DP_Clas_T, Str_TC_A, 10);
    itoa(DP_Clas_N, Str_TC_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_TC_A,Str_TC_B);
    MatriceValTypeForm->PanelTC->Caption = valeur;
    itoa(DP_Clas_N, Str_CT_A, 10);
    itoa(DP_Clas_T, Str_CT_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_CT_A,Str_CT_B);
    MatriceValTypeForm->PanelCT->Caption = valeur;
    itoa(DP_Clas_P, Str_PP_A, 10);
    itoa(DP_Clas_P, Str_PP_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_PP_A,Str_PP_B);
    MatriceValTypeForm->PanelPP->Caption = valeur;
    MatriceValTypeForm->Hide();
    MatriceValTypeForm->Show();
    }
if (RG_Type_Confrontation->ItemIndex==1)
    {
    MatriceValTypeForm->Caption = "Le retour d'ascenseur";
    itoa(DA_R, Str_CC_A, 10);
    itoa(DA_R, Str_CC_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_CC_A,Str_CC_B);
    MatriceValTypeForm->PanelCC->Caption = valeur;
```

```
    itoa(DA_T, Str_TC_A, 10);
    itoa(DA_N, Str_TC_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_TC_A,Str_TC_B);
    MatriceValTypeForm->PanelTC->Caption = valeur;
    itoa(DA_N, Str_CT_A, 10);
    itoa(DA_T, Str_CT_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_CT_A,Str_CT_B);
    MatriceValTypeForm->PanelCT->Caption = valeur;
    itoa(DA_P, Str_PP_A, 10);
    itoa(DA_P, Str_PP_B, 10);
    wsprintf(valeur, "(%s, %s)",Str_PP_A,Str_PP_B);
    MatriceValTypeForm->PanelPP->Caption = valeur;
    MatriceValTypeForm->Hide();
    MatriceValTypeForm->ShowModal();
    }
if (RG_Type_Confrontation->ItemIndex==2)
    {
    MatriceValTypeForm->Caption = "Les jeux asynchrones";
    MatriceValTypeForm->Hide();
    MatriceValTypeForm->Show();
    }
}
```

Gestion_Pop.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Gestion_Pop.h"
#include "TypeDef.h"
#include "Principal.h"
#include "math.h"
#include "Confrontation.h"
#include "Strategies.h"
#include "Calcul_Payoffs.h"
#include "VariablesGlobales.h"
#include <values.h>

extern card Search_Behavior_Adv(bool Collectif, unsigned int Num_Adv, unsigned int Num_Moi);

extern info_indiv *Liste_Pop;
extern info_init_strat *list_Strat_Selec;
extern Tab_Strat Tableau_Strat;
extern Nb_Instance_Strat *List_Nb_Instance_Strat;

extern int Last;
extern int Nb_Individu;
```

Constitution de la population initiale

```
info_indiv *Create_One_Individual(unsigned int Id_Strat)
{
/* En plus de créer la boîte associé à un individu de la population en y assignant les différentes valeurs par défaut,
cette fonction se charge également de la mise à jour de la variable globale 'Nb_Individu'. Il est bon de remarquer
que c'est la seule fonction permettant de modifier la valeur de cette dernière. */
info_indiv *list_tmp;
list_tmp = (info_indiv*) malloc(sizeof(info_indiv));
list_tmp→Previous = NULL;
list_tmp→Ident_Strat = Id_Strat;
Nb_Individu++;
list_tmp→Ident_Indiv = Nb_Individu;
list_tmp→Info_Non_Proba = NULL;
list_tmp→Info_Proba = NULL;
list_tmp→Next = NULL;
return (list_tmp);
}
```

```
Nb_Instance_Strat *Create_One_Box_Nb_Instance_Strat(info_init_strat *list_tmp)
{
Nb_Instance_Strat *Tmp;
Tmp = (Nb_Instance_Strat*) malloc(sizeof(Nb_Instance_Strat));
Tmp→N_Strat = list_tmp→N_Strat;
Tmp→Nb_Instance_Beginning = list_tmp→Nb_Instance;
Tmp→Nb_Instance_End = list_tmp→Nb_Instance;
return (Tmp);
}
```

```
info_indiv *Constitute_Pop_Init()
{
/* On utilise 'list_tmp' afin de pouvoir parcourir la liste (intitulée 'list') contenant l'ensemble des informations
nécessaires pour la composition de la population initiale. De la sorte on est certain que la variable list pointe
toujours sur le premier élément de la liste concernée. */
info_init_strat *list_tmp = list_Strat_Selec;
info_indiv *First = NULL;
info_indiv *Last = NULL;
info_indiv *Tmp;
while (list_tmp != NULL)
    {
    unsigned int i = 0;
    while (i < list_tmp→Nb_Instance)
        {
        Tmp = Create_One_Individual(list_tmp→N_Strat);
        if (First == NULL)
              {
              First = Tmp;
              }
        else  {
              Last→Next = Tmp;
              }
        Tmp→Previous = Last;
        Last = Tmp;
        i++;
        }
    list_tmp = list_tmp→Next;
    }
return (First);
}
```

```
Nb_Instance_Strat *Constitute_List_Nb_Instance_Strat()
{
info_init_strat *list_tmp = list_Strat_Selec;
Nb_Instance_Strat *First = NULL;
Nb_Instance_Strat *Last = NULL;
Nb_Instance_Strat *Tmp;
while (list_tmp != NULL)
    {
    Tmp = Create_One_Box_Nb_Instance_Strat(list_tmp);
    if (First == NULL)
          {
          First = Tmp;
          }
    else  {
          Last→Next = Tmp;
          }
    Last = Tmp;
    list_tmp = list_tmp→Next;
    }
return (First);
}
```

Fin de la constitution de la population initiale

```
Info_Gen *Create_Info_Box(unsigned int Numero, card acte)
{
/* But: Créer une boîte d'information générale relative à une stratégie particulière ou à un type de stratégie. Il
importe de remarquer qu'on ne se préoccupe pas, à ce niveau, de savoir ce que contient le champ 'Info_Strat_Adv'.*/
Info_Gen *tmp;
tmp = (Info_Gen*) malloc(sizeof(Info_Gen));
tmp→Previous = NULL;
tmp→Ident = Numero;
tmp→Action = acte;
tmp→Next = NULL;
return (tmp);
}
```

```
void MAJ_Box_Info_Strat(bool Confrontation_Indiv, unsigned int Numero_Adv, unsigned int Numero_Indiv)
{
info_indiv *Tmp = Liste_Pop;
Info_Gen *Ptr_Info_Tmp;
if (Confrontation_Indiv == true)
        {
        while ( (Tmp != NULL) && (Tmp→Ident_Indiv != Numero_Adv) )
                {
                Tmp = Tmp→Next;
                }
        }
else    {
        while ( (Tmp != NULL) && (Tmp→Ident_Strat != Numero_Adv) )
                {
                Tmp = Tmp→Next;
                }
        }
while ( (Tmp != NULL) && (Tmp→Ident_Strat == Numero_Adv) )
        {
        if (Confrontation_Indiv == true)
                {
                Ptr_Info_Tmp = Tmp→Info_Proba;
                }
        else    {
                Ptr_Info_Tmp = Tmp→Info_Non_Proba;
                }
        while ((Ptr_Info_Tmp != NULL)&&(Ptr_Info_Tmp→Ident != Numero_Indiv))
                {
                Ptr_Info_Tmp = Ptr_Info_Tmp→Next;
                }
        Ptr_Info_Tmp→Action = Abandon;
        Tmp = Tmp→Next;
        }
}
```

```
void MAJ_Champs(bool Probabiliste, info_indiv *Strat_Etud, Info_Gen *Strat_Adv)
{
/* But: Mise à jour du champ 'Info_Strat_Adv' de la boîte 'Strat_Adv'. */
Tab_Strat *tab_tmp = &Tableau_Strat;
card Behavior;
/* Recherche du comportement choisi par l'adversaire. */
if (Probabiliste == true)
      {
      Behavior = Search_Behavior_Adv(false,Strat_Adv→Ident,Strat_Etud→Ident_Indiv);
      }
else  {
      Behavior = Search_Behavior_Adv(true,Strat_Adv→Ident,Strat_Etud→Ident_Strat);
      }
switch (tab_tmp[Strat_Etud→Ident_Strat].Param)
    {
    case Pas_Param: break;
    case Ranc:    Strat_Adv→Info_Strat_Adv.info_ranc.adv = Behavior;
                  Strat_Adv→Info_Strat_Adv.info_ranc.moi = Strat_Adv→Action;
                  break;
    case Last_Behaviour_Opponent:          Strat_Adv→Info_Strat_Adv.Last_Behaviour_Opp = Behavior;
                                           break;
    case Param_Nb_Trahison:
                  if (Last == 0)
                      {
                      Strat_Adv→Info_Strat_Adv.Info_Nbre_Trahison.Nb_Trahison = 0;
                      }
                  if (Behavior == Defection)
                      {
                      Strat_Adv→Info_Strat_Adv.Info_Nbre_Trahison.Nb_Trahison++;
                      }
                  break;
    case Param_Sondage:
          if (Last == 1)
                {
                Strat_Adv→Info_Strat_Adv.Info_Sondage.Behavior_Second_Perriod = Behavior;
                }
          else  {
                if (Last == 2)
                    {
                    Strat_Adv→Info_Strat_Adv.Info_Sondage.Behavior_Third_Perriod = Behavior;
                    }
                }
          Strat_Adv→Info_Strat_Adv.Info_Sondage.Last_Behaviour_Opp = Behavior;
          break;
    case Param_TFT_Dur:
          Strat_Adv→Info_Strat_Adv.TFT_Dur.Beavior_Two_Period_Before =
             Strat_Adv→Info_Strat_Adv.TFT_Dur.Last_Behaviour_Opp;
          Strat_Adv→Info_Strat_Adv.TFT_Dur.Last_Behaviour_Opp = Behavior;
          break;
    case Proba: break;
    default:    ShowMessage("Type de stratégie non reconnue.\n
                De ce fait, on est dans l'impossibilité d'enregistrer des informations sur l'adversaire.");
    }
}
```

```
void MAJ_Info_Adv()
{
/* But: Mise à jour du comportement de l'adversaire. Cette étape est utile pour pouvoir déterminer le comportement
futur de chaque individu. */
info_indiv *Tmp = Liste_Pop;
Info_Gen *Ptr_Info_Tmp;
while (Tmp != NULL)
    {
    Ptr_Info_Tmp = Tmp→Info_Non_Proba;
    while (Ptr_Info_Tmp != NULL)
        {
        MAJ_Champs(false,Tmp,Ptr_Info_Tmp);
        Ptr_Info_Tmp = Ptr_Info_Tmp→Next;
        }
    Ptr_Info_Tmp = Tmp→Info_Proba;
    while (Ptr_Info_Tmp != NULL)
        {
        MAJ_Champs(true,Tmp,Ptr_Info_Tmp);
        Ptr_Info_Tmp = Ptr_Info_Tmp→Next;
        }
    Tmp = Tmp→Next;
    }
}
```

```
int Estimation_Var_Pop_Strat(info_init_strat *List_Strat_Selec_Tmp, float Max, float Payoff_Medium)
{
float Difference_Payoff = ( (List_Strat_Selec_Tmp→Payoff_acc_Actuel) - (List_Strat_Selec_Tmp→Payoff_acc_Preced) );
float Payoff_tmp = ( Difference_Payoff / Max );
int Nb_Instance = Max;
int Result;
while ( (Payoff_tmp < Payoff_Medium) && (Max > 1) )
    {
    Max--;
    Payoff_tmp = (Difference_Payoff)/Max;
    }
if (Payoff_tmp < Payoff_Medium)
      {
      Result = 0 - Nb_Instance;
      }
else  {
      Result = Max - Nb_Instance;
      }
return (Result);
}
```

```
bool Search_Proba_Strat(unsigned int Ident_Strat)
{
/* But: indiquer si une stratégie est probabiliste ou non. Cette fonction renvoie true si la stratégie est probabiliste, et
false sinon. */
info_init_strat *Ptr_Elem_list = list_Strat_Selec;
while ((Ptr_Elem_list != NULL)&&(Ident_Strat != Ptr_Elem_list→N_Strat))
    {
    Ptr_Elem_list = Ptr_Elem_list→Next;
    }
if (Ptr_Elem_list→Proba == 200)
```

```
        {
        return (false);
        }
else    {
        return (true);
        }
}
```

```
unsigned int Nbr_People_Tot()
{
info_init_strat *Tmp = list_Strat_Selec;
unsigned int Nbr_Instance = 0;
while (Tmp != NULL)
    {
    Nbr_Instance += Tmp→Nb_Instance;
    Tmp = Tmp→Next;
    }
return (Nbr_Instance);
}
```

```
unsigned int Nbr_Strat()
{
info_init_strat *Liste_Tmp = list_Strat_Selec;
unsigned int Nbr = 0;
while (Liste_Tmp != NULL)
    {
    Nbr++;
    Liste_Tmp = Liste_Tmp→Next;
    }
return (Nbr);
}
```

```
unsigned int Num_Strat_In_Table(info_init_strat *Type_Strat)
{
unsigned int Numero = 1;
info_init_strat *Ptr_Tmp_List_Strat_Selec= list_Strat_Selec;
while (Type_Strat→N_Strat != Ptr_Tmp_List_Strat_Selec→N_Strat)
    {
    Numero++;
    Ptr_Tmp_List_Strat_Selec = Ptr_Tmp_List_Strat_Selec→Next;
    }
return (Numero);
}
```

```
unsigned int Nbr_Strat_Rest()
{
info_init_strat *Liste_Tmp = list_Strat_Selec;
unsigned int Nbr = 0;
while (Liste_Tmp != NULL)
    {
    if (Liste_Tmp→Nb_Instance != 0)
        {
        Nbr++;
```

```
        }
    Liste_Tmp = Liste_Tmp→Next;
    }
return (Nbr);
}
```

The KillerFunctions

```
Info_Gen *Kill_list_Info_Gen(Info_Gen *list)
{
Info_Gen *tmp=list;
while (list!=NULL)
    {
    list = list→Next;
    free(tmp);
    tmp = list;
    }
return (list);
}
```

```
void Kill_list_indiv()
{
info_indiv *tmp=Liste_Pop;
info_indiv *Tmp_Info_Non_Proba; /* Cette variable se charge de mettre à NULL les pointeurs 'Info_Non_Proba' des
autres boîtes relatives aux individus ayant adopté la même stratégie à NULL. En agissant de la sorte, on risque de
désallouer une adresse mémoire qui aurait déjà été désallouée => problème. */
while (Liste_Pop != NULL)
    {
    Tmp_Info_Non_Proba=Liste_Pop;
    Liste_Pop→Info_Non_Proba = Kill_list_Info_Gen(Liste_Pop→Info_Non_Proba);
    While   (     (Tmp_Info_Non_Proba != NULL)
            &&  (Tmp_Info_Non_Proba→Ident_Strat == Liste_Pop→Ident_Strat) )
        {
        Tmp_Info_Non_Proba→Info_Non_Proba = NULL;
        Tmp_Info_Non_Proba = Tmp_Info_Non_Proba→Next;
        }
    Liste_Pop→Info_Proba = Kill_list_Info_Gen(Liste_Pop→Info_Proba);
    Liste_Pop = Liste_Pop→Next;
    free(tmp);
    tmp = Liste_Pop;
    }
}
```

```
void Kill_list_Strat_Selec()
{
info_init_strat *Tmp=list_Strat_Selec;
while (list_Strat_Selec!=NULL)
    {
    list_Strat_Selec = list_Strat_Selec→Next;
    free(Tmp);
    Tmp = list_Strat_Selec;
    }
}
```

```
void Kill_Box_Indiv(info_indiv *Elem_To_Kill)
{
Elem_To_Kill→Info_Non_Proba = Kill_list_Info_Gen(Elem_To_Kill→Info_Non_Proba);
Elem_To_Kill→Info_Proba = Kill_list_Info_Gen(Elem_To_Kill→Info_Proba);
free(Elem_To_Kill);
}
Info_Gen *Kill_One_Box(Info_Gen *Ptr, unsigned int Num_Strat)
{
Info_Gen *First = NULL;
Info_Gen *Box_Previous;
Info_Gen *Box_Next;
if (Ptr != NULL)
    {
    if (Ptr→Ident == Num_Strat)
        {
        First = Ptr→Next;
        if (First != NULL)
            {
            First→Previous = NULL;
            }
        }
    else {
        First = Ptr;
        while ( (Ptr != NULL) && (Ptr→Ident != Num_Strat) )
            {
            Ptr = Ptr→Next;
            }
        if (Ptr != NULL)
            {
            Box_Previous = Ptr→Previous;
            Box_Next = Ptr→Next;
            if (Box_Previous != NULL)
                {
                Box_Previous→Next = Box_Next;
                }
            if (Box_Next != NULL)
                {
                Box_Next→Previous = Box_Previous;
                }
            free(Ptr);
            }
        }
    }
return(First);
}
```

```
void Kill_Box_Superflous_Non_Proba(unsigned int Num_Strat)
{
info_indiv *List_Pop_Tmp = Liste_Pop;
while (List_Pop_Tmp != NULL)
    {
    if (List_Pop_Tmp→Ident_Strat != Num_Strat)
        {
        List_Pop_Tmp→Info_Non_Proba =
          Kill_One_Box(List_Pop_Tmp→Info_Non_Proba, Num_Strat);
```

```
        }
    List_Pop_Tmp = List_Pop_Tmp→Next;
    }
}
void Kill_Box_Superflous_Info_Proba(unsigned int Num_Indiv, unsigned int Num_Strat)
{
Tab_Strat *Tab_Tmp = &Tableau_Strat;
info_indiv *List_Pop_Tmp = Liste_Pop;
unsigned int i;
if (Tab_Tmp[Num_Strat].Flag == Strat_Prob)
      {
      while (List_Pop_Tmp != NULL)
          {
          List_Pop_Tmp→Info_Proba = Kill_One_Box(List_Pop_Tmp→Info_Proba, Num_Indiv);
          List_Pop_Tmp = List_Pop_Tmp→Next;
          }
      }
else  {
      while (List_Pop_Tmp != NULL)
          {
          i = List_Pop_Tmp→Ident_Strat;
          if (Tab_Tmp[i].Flag == Strat_Prob)
              {
              List_Pop_Tmp→Info_Proba =
                    Kill_One_Box(List_Pop_Tmp→Info_Proba, Num_Indiv);
              }
          List_Pop_Tmp = List_Pop_Tmp→Next;
          }
      }
}
```

```
info_indiv *Kill_Ind(unsigned int Num_Strat, int Destructor)
{
info_indiv *List_Pop_Tmp = Liste_Pop;
info_indiv *Elem_To_Kill;
info_indiv *Tmp_Previous;
info_indiv *Tmp_Next;
info_indiv *First = Liste_Pop;
while ( (List_Pop_Tmp != NULL) && (List_Pop_Tmp→Ident_Strat != Num_Strat) )
    {
    List_Pop_Tmp = List_Pop_Tmp→Next;
    }
if (List_Pop_Tmp == NULL)
      {
      ShowMessage("Problème au niveau de la suppression d'un individu (1)!");
      }
else  {
      while (     (Destructor != 0)
            &&  (List_Pop_Tmp != NULL)
            &&  (List_Pop_Tmp→Ident_Strat == Num_Strat) )
          {
          if (First→Ident_Strat == Num_Strat)
              {
              First = First→Next;
              }
          Elem_To_Kill = List_Pop_Tmp;
```

```
        Tmp_Previous = Elem_To_Kill→Previous;
        Tmp_Next = Elem_To_Kill→Next;
        if (Tmp_Previous != NULL)
            {
            Tmp_Previous→Next = Tmp_Next;
            }
        if (Tmp_Next != NULL)
            {
            Tmp_Next→Previous = Tmp_Previous;
            }
        Destructor++;
        List_Pop_Tmp = List_Pop_Tmp→Next;
        if (List_Pop_Tmp != NULL)
            {
            if (List_Pop_Tmp→Ident_Strat != Elem_To_Kill→Ident_Strat)
                {
                Kill_Box_Superflous_Non_Proba(Elem_To_Kill→Ident_Strat);
                }
            }
        else {
            Kill_Box_Superflous_Non_Proba(Elem_To_Kill→Ident_Strat);
            }
        Kill_Box_Superflous_Info_Proba(Elem_To_Kill→Ident_Indiv, Elem_To_Kill→Ident_Strat);
        }
    }
if (Destructor > 0)
    {
    ShowMessage("Problème au niveau de la suppression d'un individu (2)!");
    }
return(First);
}
```

```
void Kill_Elem_list_Strat_Selec(AnsiString s)
{
/* Sur base d'un argument de type AnsiString, cette fonction repère l'élément à extraire de la liste existante, et réalise
cette opération. */
info_init_strat *Ptr_Elem_Courant_List = list_Strat_Selec;
info_init_strat *Ptr_Elem_Prec_List;
Tab_Strat *Tab_Tmp = &Tableau_Strat;
int j = list_Strat_Selec→N_Strat;
char *Buf;
Buf = new char [100];
AnsiString Tmp;
lstrcpy(Buf, Tab_Tmp[j].Nom_Strat);
Tmp = Buf;
if (Tmp.AnsiCompare(s) == 0)
    {
    list_Strat_Selec = list_Strat_Selec→Next;
    }
else {
    while ( (Ptr_Elem_Courant_List == NULL) && (Tmp.AnsiCompare(s) != 0) )
        {
        Ptr_Elem_Prec_List = Ptr_Elem_Courant_List;
        Ptr_Elem_Courant_List = Ptr_Elem_Courant_List→Next;
        j = Ptr_Elem_Courant_List→N_Strat;
        lstrcpy(Buf, Tab_Tmp[j].Nom_Strat);
```

```
            Tmp = Buf;
            }
        if (Tmp.AnsiCompare(s) == 0)
            {
            Ptr_Elem_Prec_List→Next = Ptr_Elem_Courant_List→Next;
            }
        else {
            char Text[200];
            wsprintf(Text,"Erreur au niveau de la liste de sélection des stratégies.\n
            Veuillez recommencer votre sélection, merci.\n");
            ShowMessage(Text);
            }
        }
if (Tmp.AnsiCompare(s) == 0)
    {
    free(Ptr_Elem_Courant_List);
    }
}
```

Insertion de nouveaux individus au sein de la population

```
Info_Gen *Copy_Box_Info_Non_Proba(Info_Gen *Ptr)
{
Info_Gen *Tmp;
Tmp = (Info_Gen *) malloc(sizeof(Info_Gen));
Tmp→Previous = NULL;
Tmp→Ident = Ptr→Ident;
Tmp→Info_Strat_Adv = Ptr→Info_Strat_Adv;
Tmp→Action = Ptr→Action;
Tmp→Next = NULL;
return(Tmp);
}
```

```
Info_Gen *Copy_List_Info_Non_Proba(Info_Gen *Ptr)
{
Info_Gen *First;
Info_Gen *Last;
Info_Gen *Tmp;
if (Ptr == NULL)
    {
    First = NULL;
    }
else {
    First = Copy_Box_Info_Non_Proba(Ptr);
    Last = First;
    Ptr=Ptr→Next;
    while (Ptr != NULL)
        {
        Tmp = Copy_Box_Info_Non_Proba(Ptr);
        Tmp→Previous = Last;
        Last→Next = Tmp;
        Last = Last→Next;
        Ptr = Ptr→Next;
        }
```

```
    }
return (First);
}
```

```
void Add_Ind(unsigned int Num_Strat, int variation)
{
info_indiv *List_Pop_Tmp = Liste_Pop;
info_indiv *Ptr;
info_indiv *Temporaire;
info_indiv *Ptr_Tmp;
while ( (List_Pop_Tmp != NULL) && (List_Pop_Tmp→Ident_Strat != Num_Strat) )
    {
    List_Pop_Tmp = List_Pop_Tmp→Next;
    }
Ptr = List_Pop_Tmp;
if (List_Pop_Tmp == NULL)
      {
      ShowMessage("Problème au niveau de l'insertion d'un nouvel individu!");
      }
else  {
      while (variation > 0)
          {
          Temporaire = Create_One_Individual(Num_Strat);
          Temporaire→Previous = List_Pop_Tmp;
          Temporaire→Next = List_Pop_Tmp→Next;
          List_Pop_Tmp→Next = Temporaire;
          Ptr_Tmp = Temporaire→Next;
          if (Temporaire→Next != NULL)
              {
              Ptr_Tmp→Previous = Temporaire;
              }
          Temporaire→Info_Non_Proba = Copy_List_Info_Non_Proba(Ptr→Info_Non_Proba);
          variation--;
          }
      }
}
```

Ajustement de la population

```
void Ajustement_Pop_Var(float Payoff_Medium, unsigned int period, unsigned int Nb_Tot_Indiv_Pop)
{
/* Nous devons obtenir par stratégie: Payoff_Medium < (Payoff_acc_Actuel)/Nb_Instance).
En dautres termes, si cette inégalité n'est pas respectée, nous limiterons 'Nb_Instance' de telle sorte qu'elle satisfasse
l'inégalité sus-mentionnée. */
info_init_strat *List_Strat_Selec_Tmp = list_Strat_Selec;
Nb_Instance_Strat *Ptr_Tmp_List_Nb_Instance_Strat;
float Nb_Instances_Beginning, Nb_Instances_End;
int Variation; /* 'Variation' indique la valeur de la variation de la population concernée. */
while (List_Strat_Selec_Tmp != NULL)
    {
    Ptr_Tmp_List_Nb_Instance_Strat = List_Nb_Instance_Strat;
    while (Ptr_Tmp_List_Nb_Instance_Strat→N_Strat != List_Strat_Selec_Tmp→N_Strat)
        {
        Ptr_Tmp_List_Nb_Instance_Strat = Ptr_Tmp_List_Nb_Instance_Strat→Next;
```

```
        }
    Nb_Instances_Beginning = Ptr_Tmp_List_Nb_Instance_Strat→Nb_Instance_Beginning;
    Nb_Instances_End = Ptr_Tmp_List_Nb_Instance_Strat→Nb_Instance_End;
    Variation = Nb_Instances_End - Nb_Instances_Beginning;
    if (Variation != 0)
        {
        if (Variation < 0)
            {
            Liste_Pop = Kill_Ind(List_Strat_Selec_Tmp→N_Strat, Variation);
            }
        else {
            Add_Ind(List_Strat_Selec_Tmp→N_Strat, Variation);
            }
        }
    List_Strat_Selec_Tmp→Nb_Instance = Ptr_Tmp_List_Nb_Instance_Strat→Nb_Instance_End;
    List_Strat_Selec_Tmp = List_Strat_Selec_Tmp→Next;
    }
}
```

Dans le cas d'un effectif total variable, nous utilisons 'Estimation_Pop_End_Period' permettant d'estimer la population en fin de période

```
unsigned int Estimation_Pop_End_Period_Var_Medium(float Payoff_Per_Indiv)
{
info_init_strat *Ptr_Tmp = list_Strat_Selec;
unsigned int Nb_Instance_End, Payoff_Period;
unsigned int Pop_Eff_End = 0;
int Variation;
float Payoff_Strat, Numerateur, Denominateur, Interm_Pop, Proportion, Unite;
while (Ptr_Tmp != NULL)
    {
    if (Ptr_Tmp→Nb_Instance != 0)
        {
        Payoff_Period = Ptr_Tmp→Payoff_acc_Actuel - Ptr_Tmp→Payoff_acc_Preced;
        Numerateur = Payoff_Period;
        Denominateur = Ptr_Tmp→Nb_Instance;
        Payoff_Strat = (Numerateur / Denominateur);
        if (Payoff_Strat < Payoff_Per_Indiv)
            {
            Variation =
                Estimation_Var_Pop_Strat(Ptr_Tmp, Ptr_Tmp→Nb_Instance, Payoff_Per_Indiv);
            Nb_Instance_End = Ptr_Tmp→Nb_Instance + Variation;
            }
        else {
            Numerateur = Ptr_Tmp→Tx_Croiss_Pop;
            Denominateur = 100;
            Unite = 1;
            Proportion = Numerateur / Denominateur;
            Interm_Pop = Ptr_Tmp→Nb_Instance;
            Nb_Instance_End = floor((Interm_Pop) * (Unite + (Proportion)));
            }
        }
    else {
        Nb_Instance_End = 0;
        }
```

```
    Pop_Eff_End += Nb_Instance_End;
    Ptr_Tmp = Ptr_Tmp→Next;
    }
return(Pop_Eff_End);
}
void MAJ_List_Nb_Instance_Strat(info_init_strat *Ptr, unsigned int Nb_Instance_End)
{
Nb_Instance_Strat *Ptr_Tmp = List_Nb_Instance_Strat;
while (Ptr_Tmp→N_Strat != Ptr→N_Strat)
    {
    Ptr_Tmp = Ptr_Tmp→Next;
    }
Ptr_Tmp→Nb_Instance_Beginning = Ptr→Nb_Instance;
Ptr_Tmp→Nb_Instance_End = Nb_Instance_End;
}
```

IHM.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "IHM.h"
#include "Principal.h"
#include "SaisieStratConfr.h"
#include "TypeDef.h"
#include "Strategies.h"
#include "Progression.h"
#include "NextPeriode.h"
#include "VariablesGlobales.h"
#include <math.h>

extern int Estimation_Var_Pop_Strat(info_init_strat *List_Strat_Selec_Tmp, float Max, float Payoff_Medium);
extern unsigned int Num_Strat_In_Table(info_init_strat *Type_Strat);
extern void MAJ_List_Nb_Instance_Strat(info_init_strat *Ptr_Tmp, unsigned int Nb_Instance_End);
extern unsigned int Nbr_Strat_Rest();
extern unsigned int Nbr_Strat();

extern info_init_strat *list_Strat_Selec;
extern Tab_Strat Tableau_Strat;
extern int Max_Conf;
extern bool Pas_A_Pas;
extern Type_Affichage Affich_Result;
extern bool Confront_In_The_Past;
```

```
void Aff_Couleur_Strat()
{
info_init_strat *Ptr_Tmp = list_Strat_Selec;
Tab_Strat *Tab_Tmp = &Tableau_Strat;
unsigned int i = Nbr_Strat();
unsigned int j = 1;
if ( i <= 12)
      {
      FormPrincipal→SB_Left→Visible  = false;
      FormPrincipal→SB_Rigth→Visible  = false;
      }
else  {
      FormPrincipal→SB_Left→Visible  = true;
      FormPrincipal→SB_Rigth→Visible  = true;
      }
while ((j <= 12) && (j <= i))
    {
    switch (j)
        {
        case 1:    FormPrincipal→LabStrat1→Visible = true;
                   FormPrincipal→LabStrat1→Color = Ptr_Tmp→Couleur;
                   FormPrincipal→LabStrat1→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                   break;
```

```
        case 2:   FormPrincipal→LabStrat2→Visible = true;
                  FormPrincipal→LabStrat2→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat2→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 3:   FormPrincipal→LabStrat3→Visible = true;
                  FormPrincipal→LabStrat3→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat3→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 4:   FormPrincipal→LabStrat4→Visible = true;
                  FormPrincipal→LabStrat4→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat4→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 5:   FormPrincipal→LabStrat5→Visible = true;
                  FormPrincipal→LabStrat5→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat5→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 6:   FormPrincipal→LabStrat6→Visible = true;
                  FormPrincipal→LabStrat6→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat6→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 7:   FormPrincipal→LabStrat7→Visible = true;
                  FormPrincipal→LabStrat7→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat7→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 8:   FormPrincipal→LabStrat8→Visible = true;
                  FormPrincipal→LabStrat8→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat8→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 9:   FormPrincipal→LabStrat9→Visible = true;
                  FormPrincipal→LabStrat9→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat9→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 10:  FormPrincipal→LabStrat10→Visible = true;
                  FormPrincipal→LabStrat10→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat10→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 11:  FormPrincipal→LabStrat11→Visible = true;
                  FormPrincipal→LabStrat11→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat11→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        case 12:  FormPrincipal→LabStrat12→Visible = true;
                  FormPrincipal→LabStrat12→Color = Ptr_Tmp→Couleur;
                  FormPrincipal→LabStrat12→Caption = Tab_Tmp[Ptr_Tmp→N_Strat].Nom_Strat;
                  break;
        }
    Ptr_Tmp = Ptr_Tmp→Next;
    j++;
    }
while (j <= 12)
    {
    switch (j)
        {
        case 1:   FormPrincipal→LabStrat1→Visible = false;
                  break;
        case 2:   FormPrincipal→LabStrat2→Visible = false;
                  break;
```

```
        case 3:   FormPrincipal→LabStrat3→Visible = false;
                  break;
        case 4:   FormPrincipal→LabStrat4→Visible = false;
                  break;
        case 5:   FormPrincipal→LabStrat5→Visible = false;
                  break;
        case 6:   FormPrincipal→LabStrat6→Visible = false;
                  break;
        case 7:   FormPrincipal→LabStrat7→Visible = false;
                  break;
        case 8:   FormPrincipal→LabStrat8→Visible = false;
                  break;
        case 9:   FormPrincipal→LabStrat9→Visible = false;
                  break;
        case 10:  FormPrincipal→LabStrat10→Visible = false;
                  break;
        case 11:  FormPrincipal→LabStrat11→Visible = false;
                  break;
        case 12:  FormPrincipal→LabStrat12→Visible = false;
                  break;
      }
   j++;
   }
}
```

```
void IHM_Selection_Strat_Incorrecte()
{
FormPrincipal→PanelPrincipalClient→Color = clGray;
FormPrincipal→Plot_Evol_Pop→Visible = false;
FormPrincipal→SG_Result→Visible = false;
FormPrincipal→LabelPriode→Visible = false;
FormPrincipal→GB_Selcet_Color→Visible = false;
FormPrincipal→AxeVertical→Visible = false;
if (Confront_In_The_Past == true)
   {
   FormPrincipal→StatusBarPrincipal→Panels→Items[0] →Free();
   }
MessageDlg("Vous devez choisir au moins deux types différents de stratégies.",mtError,TMsgDlgButtons()<<mbOK,
0);
}
```

```
void IHM_Selection_Strat_Correcte(unsigned int i)
{
FormPrincipal→PanelPrincipalClient→Color = clSilver;
FormPrincipal→GB_Selcet_Color→Visible = true;
Aff_Couleur_Strat();
FormPrincipal→Plot_Evol_Pop→Visible = true;
FormPrincipal→Plot_Evol_Pop→Canvas→Pen→Color = clWhite;
FormPrincipal→Plot_Evol_Pop→Canvas→Brush→Color = clWhite;
FormPrincipal→Plot_Evol_Pop→Canvas→Rectangle(0,0,
   FormPrincipal→Plot_Evol_Pop→Width,FormPrincipal→Plot_Evol_Pop→Height);
FormPrincipal→Plot_Evol_Pop→Canvas→Pen→Color = clBlack;
FormPrincipal→Plot_Evol_Pop→Canvas→Pen→Width = 2;
FormPrincipal→Plot_Evol_Pop→Canvas→MoveTo(0,0);
```

```
FormPrincipal→Plot_Evol_Pop→Canvas→LineTo(0,FormPrincipal→Plot_Evol_Pop→Height);
FormPrincipal→Plot_Evol_Pop→Canvas→MoveTo(0,FormPrincipal→Plot_Evol_Pop→Height);
FormPrincipal→Plot_Evol_Pop→Canvas→
   LineTo(FormPrincipal→Plot_Evol_Pop→Width,FormPrincipal→Plot_Evol_Pop→Height);
FormPrincipal→SG_Result→Visible = true;
FormPrincipal→LabelPriode→Visible = true;
FormPrincipal→SG_Result→Cells[0][0] = "Nom de la stratégie";
switch(Affich_Result)
   {
   case Prorata:      FormPrincipal→SG_Result→Cells[1][0] = "Part dans la population";
                      break;
   case Nb_Indiv:     FormPrincipal→SG_Result→Cells[1][0] = "Nombre d'individus";
                      break;
   case Revenu_Acc:   FormPrincipal→SG_Result→Cells[1][0] = "Revenu accumulé";
                      break;
   }
FormPrincipal→AxeVertical→Visible = true;
unsigned int Nb_Strategies = i + 1; /* On ajoute 1 car il faut prendre en considération l'intitulé de chaque colonne.
*/
if (Nb_Strategies <= 18)
/* Pourquoi fixer la valeur 18?
18 * 28 (la taille d'une cellule de SG_Result) = 504 (la taille de Plot_Evol_Pop). */
      {
      FormPrincipal→SG_Result→Height = ((Nb_Strategies) * 28);
      FormPrincipal→SG_Result→ScrollBars = ssNone;
      }
else  {
      FormPrincipal→SG_Result→Height = 504;
      FormPrincipal→SG_Result→ScrollBars = ssVertical;
      }
FormPrincipal→SG_Result→RowCount = Nb_Strategies;
/* Initialisation du tableau de résultats. */
info_init_strat *List_Tmp = list_Strat_Selec;
unsigned int index = 0;
char *buf;
buf = new char[100];
float Nb_Indiv_Strat;
float Nb_Indiv_Pop = 0;
while (List_Tmp != NULL)
   {
   Nb_Indiv_Pop += List_Tmp→Nb_Instance;
   List_Tmp = List_Tmp→Next;
   }
List_Tmp = list_Strat_Selec;
while (List_Tmp != NULL)
   {
   index++;
   Tab_Strat *tab_tmp = &Tableau_Strat;
   FormPrincipal→SG_Result→Cells[0][index] = tab_tmp[List_Tmp→N_Strat].Nom_Strat;
   switch(Affich_Result)
      {
      case Prorata:      Nb_Indiv_Strat = List_Tmp→Nb_Instance;
                         Nb_Indiv_Strat = (Nb_Indiv_Strat / Nb_Indiv_Pop) *100;
                         wsprintf(buf,"%f",Nb_Indiv_Strat);
                         FormPrincipal→SG_Result→Cells[1][index] = buf;
                         break;
      case Nb_Indiv:     wsprintf(buf,"%d",List_Tmp→Nb_Instance);
```

```
                        FormPrincipal→SG_Result→Cells[1][index] = buf;
                        break;
        case Revenu_Acc: wsprintf(buf,"%d",List_Tmp→Payoff_acc_Actuel);
                        FormPrincipal→SG_Result→Cells[1][index] = buf;
                        break;
        }
    List_Tmp = List_Tmp→Next;
    }
/* Initialisation de la StatusBar ('StatusBarPrincipal'). */
char *Texte;
Texte = new char[200];
wsprintf(Texte, "Aucune période n'a encore été traitée");
FormPrincipal→StatusBarPrincipal→Panels→Add();
FormPrincipal→StatusBarPrincipal→Panels→Items[0]→Text = Texte;
FormPrincipal→StatusBarPrincipal→Panels→Items[0]→Width = 300;
/* Initialisation de la barre de progression. */
FormProgression→Show();
FormProgression→ProgressBar→Max = Max_Conf;
}
```

```
void MAJ_Plot_Evol_Pop(unsigned int Beginning_Period, unsigned int Proportion_Ind_Beginning, info_init_strat
*Ptr_Tmp, unsigned int End_Period, unsigned int Proportion_Ind_End)
{
FormPrincipal→Plot_Evol_Pop→Canvas→MoveTo(Beginning_Period, Proportion_Ind_Beginning);
FormPrincipal→Plot_Evol_Pop→Canvas→Pen→Color = Ptr_Tmp→Couleur;
FormPrincipal→Plot_Evol_Pop→Canvas→Pen→Width = Ptr_Tmp→Epaisseur;
FormPrincipal→Plot_Evol_Pop→Canvas→LineTo(End_Period, Proportion_Ind_End);
}
```

```
void MAJ_Affich_Result(info_init_strat *Ptr_Tmp, float Proportion_End, unsigned int Nb_Instance_End)
{
char *Buf;
Buf = new char[200];
unsigned int Index = Num_Strat_In_Table(Ptr_Tmp);
int Result;
switch(Affich_Result)
    {
    case Prorata:       Result = Proportion_End * 100;
                        wsprintf(Buf, "%d", Result);
                        FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                        break;
    case Nb_Indiv:      wsprintf(Buf,"%d",Nb_Instance_End);
                        FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                        break;
    case Revenu_Acc:    wsprintf(Buf,"%d",Ptr_Tmp→Payoff_acc_Actuel);
                        FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                        break;
    }
}
```

```
void MAJ_StatusBarPrincipal(int Period)
{
char *Texte;
Texte = new char[100];
```

```
if (Period == 1)
    {
    wsprintf(Texte, "La dernière période traitée est %d ière", Period);
    }
else {
    wsprintf(Texte, "La dernière période traitée est %d ième", Period);
    }
FormPrincipal→StatusBarPrincipal→Panels→Items[0] →Text = Texte;
}
```

```
void Aff_Result_Confront_Var_Medium(float Payoff_Medium_Per_Indiv, int Period, unsigned int Nb_Eff_Pop_Tot_Beginning, unsigned int Nb_Eff_Pop_Tot_End)
{
/* Cette fonction concerne la mise à jour des différents paramètres relatifs à l'IHM. Ceux-ci englobent le graphe de confrontation des stratégies, le tableau des résultats de chaque stratégie, la barre de progression et la StatusBar 'StatusBarPrincipal'. ATTENTION, on effectue également la mise à jour du champ 'Nb_Instance' de la variable globale 'list_Strat_Selec'. */
info_init_strat *Ptr_Tmp = list_Strat_Selec;
unsigned int Nb_Instance_End, Payoff_Period, Beginning_Period, End_Period, Proportion_Ind_Beginning, Proportion_Ind_End;
int variation;
float period, Payoff_Strat, Proportion_Breadth, Proportion, Proportion_End, Proportion_Beginning, Nb_Instance_Beginning, Numerateur, Denominateur, Interm_Heigth, Interm_Pop, Unite;
Numerateur = FormPrincipal→Plot_Evol_Pop→Width;
Denominateur = Max_Conf;
Proportion_Breadth = Numerateur / Denominateur;
period = Period;
Beginning_Period = (period - 1) * Proportion_Breadth;
End_Period = period * Proportion_Breadth;
while (Ptr_Tmp != NULL)
    {
    Nb_Instance_Beginning = Ptr_Tmp→Nb_Instance;
    if (Nb_Instance_Beginning != 0)
        {
        Payoff_Period = Ptr_Tmp→Payoff_acc_Actuel - Ptr_Tmp→Payoff_acc_Preced;
        Numerateur = Payoff_Period;
        Denominateur = Nb_Instance_Beginning;
        Payoff_Strat = (Numerateur / Denominateur);
        if (Payoff_Strat < Payoff_Medium_Per_Indiv)
            {
            variation = Estimation_Var_Pop_Strat(Ptr_Tmp, Nb_Instance_Beginning, Payoff_Medium_Per_Indiv);
            Nb_Instance_End = Nb_Instance_Beginning + variation;
            }
        else {
            Numerateur = Ptr_Tmp→Tx_Croiss_Pop;
            Denominateur = 100;
            Unite = 1;
            Proportion = Numerateur / Denominateur;
            Interm_Pop = Nb_Instance_Beginning;
            Nb_Instance_End = floor((Interm_Pop) * (Unite + (Proportion)));
            }
        }
    else {
        Nb_Instance_End = 0;
        }
    if (Nb_Eff_Pop_Tot_Beginning != 0)
        {
```

```
            Numerateur = Nb_Instance_Beginning;
            Denominateur = Nb_Eff_Pop_Tot_Beginning;
            Proportion_Beginning = Numerateur / Denominateur;
            Interm_Heigth = FormPrincipal→Plot_Evol_Pop→Height;
            Proportion_Ind_Beginning = ceil(Interm_Heigth - (Interm_Heigth * Proportion_Beginning));
            Numerateur = Nb_Instance_End;
            Denominateur = Nb_Eff_Pop_Tot_End;
            Proportion_End = Numerateur / Denominateur;
            Proportion_Ind_End = ceil (Interm_Heigth - (Interm_Heigth * Proportion_End));
            }
        else {
            Proportion_Ind_Beginning = FormPrincipal→Plot_Evol_Pop→Height;
            Proportion_Ind_End = FormPrincipal→Plot_Evol_Pop→Height;
            }
        // Mise à jour du graphe de confrontation des stratégies
        MAJ_Plot_Evol_Pop(Beginning_Period, Proportion_Ind_Beginning, Ptr_Tmp, End_Period,
            Proportion_Ind_End);
        // Mise à jour du tableau des résultats de chaque stratégie
        MAJ_Affich_Result(Ptr_Tmp, Proportion_End, Nb_Instance_End);
        // Mise à jour de la variable globale 'List_Nb_Instance_Strat
        MAJ_List_Nb_Instance_Strat(Ptr_Tmp, Nb_Instance_End);
        Ptr_Tmp = Ptr_Tmp→Next;
        }
    // Mise à jour de la barre de progression
    FormProgression→ProgressBar→Position = Period;
    // Mise à jour de la StatusBar 'StatusBarPrincipal'
    MAJ_StatusBarPrincipal(Period);
    }
```

```
void Aff_Fin_Confrontation(unsigned int Period, unsigned int Nb_Strat_Diff_Rest)
{
char *buf1;
buf1 = new char[100];
char *Texte;
Texte = new char[200];
FormProgression→Close();
if(Nb_Strat_Diff_Rest < 2)
        {
        MessageDlg("Arrêt prématuré de la confrontation\n
            suite à la disparition de toute stratégie antagoniste.",
        mtWarning,TMsgDlgButtons()<<mbOK,0);
        if (Period == 1)
            {
            wsprintf(Texte, "La confrontation s'est terminée prématurément à la %d ière période",
                Period);
            }
        else {
            wsprintf(Texte, "La confrontation s'est terminée prématurément à la %d ième période",
                Period);
            }
        }
```

```
else {
      if (Period == 1)
            {
            wsprintf(Texte, "La confrontation s'est terminée normalement à la %d ière période",
                  Period-1);
            }
      else {
            wsprintf(Texte, "La confrontation s'est terminée normalement à la %d ième période",
      Period);
            }
      wsprintf(buf1,"Fin de la confrontation.\n
            Il reste %d types différents de stratégies.",Nb_Strat_Diff_Rest);
      MessageDlg(buf1,mtWarning,TMsgDlgButtons()<<mbOK,0);
      }
FormPrincipal→StatusBarPrincipal→Panels→Items[0] →Text = Texte;
}
```

```
void Progression_Pas_A_Pas()
{
if (Pas_A_Pas == true)
    {
    NextPeriodeForm→Hide();
    NextPeriodeForm→ShowModal();
    }
}
```

```
void Affichage_Tableau_Resultat()
{
info_init_strat *Ptr_Tmp = list_Strat_Selec;
char *Buf;
Buf = new char[200];
unsigned int Index;
int Result;
float Pop_Totale = 0;
float Proportion_End, Pop_Strat;
if (Ptr_Tmp != NULL)
    {
    switch(Affich_Result)
        {
        case Prorata:   FormPrincipal→SG_Result→Cells[1][0] = "Part dans la population";
                        while (Ptr_Tmp != NULL)
                            {
                            Pop_Totale += Ptr_Tmp→Nb_Instance;
                            Ptr_Tmp = Ptr_Tmp→Next;
                            }
                        Ptr_Tmp = list_Strat_Selec;
                        while (Ptr_Tmp != NULL)
                            {
                            Index = Num_Strat_In_Table(Ptr_Tmp);
                            Pop_Strat = Ptr_Tmp→Nb_Instance;
                            Proportion_End = (Pop_Strat / Pop_Totale);
                            Result = Proportion_End * 100;
                            wsprintf(Buf, "%d", Result);
                            FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                            Ptr_Tmp = Ptr_Tmp→Next;
```

```
                    }
                break;
        case Nb_Indiv:      FormPrincipal→SG_Result→Cells[1][0] = "Nombre d'individus";
                            while (Ptr_Tmp != NULL)
                                {
                                Index = Num_Strat_In_Table(Ptr_Tmp);
                                wsprintf(Buf,"%d",Ptr_Tmp→Nb_Instance);
                                FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                                Ptr_Tmp = Ptr_Tmp→Next;
                                }
                            break;
        case Revenu_Acc:    FormPrincipal→SG_Result→Cells[1][0] = "Revenu accumulé";
                            while (Ptr_Tmp != NULL)
                                {
                                Index = Num_Strat_In_Table(Ptr_Tmp);
                                wsprintf(Buf,"%d",Ptr_Tmp→Payoff_acc_Actuel);
                                FormPrincipal→SG_Result→Cells[1][Index] = Buf;
                                Ptr_Tmp = Ptr_Tmp→Next;
                                }
                            break;
        }
    }
}
```

LonConf.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "LongConf.h"
#pragma resource "*.dfm"
```

```
TLongConfForm *LongConfForm;

extern int Max_Conf;
```

```
__fastcall TLongConfForm::TLongConfForm(TComponent* Owner): TForm(Owner)
{
}
```

```
void __fastcall TLongConfForm::FormResize(TObject *Sender)
{
LongConfForm→Width = LongConfForm→Width;
LongConfForm→Height = LongConfForm→Height;

}
```

```
void __fastcall TLongConfForm::BitBtn_LongConfClick(TObject *Sender)
{
int Tmp_Length = atoi(LongConfForm→ME_LengthConf→Text.c_str());
if (Tmp_Length > 0)
      {
      Max_Conf = Tmp_Length;
      Close();
      }
else  {
      MessageBox(Handle, "La durée de la confrontation doit être supérieure à zéro",
          "Durée de la confrontation",MB_OK|MB_ICONEXCLAMATION);
      }
}
```

Matrice.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Matrice.h"
#pragma resource "*.dfm"

TMatriceForm *MatriceForm;

---

__fastcall TMatriceForm::TMatriceForm(TComponent* Owner): TForm(Owner)
{
}

---

bool Verif_Param_DP_Clas(int *C, int *T, int *D, int *P)
{
unsigned long int tmp1=2*(*C);
unsigned long int tmp2=(*T)+(*D);
long int tmpC = (*C);
long int tmpT = (*T);
long int tmpD = (*D);
long int tmpP = (*P);
return ((tmpT>tmpC) && (tmpC>tmpP) && (tmpP>tmpD) && ((tmp1)>(tmp2)));
}

---

void __fastcall TMatriceForm::BitBtn_Ok_ModifMatClick(TObject *Sender)
{
extern int N_CC_A,      N_TC_A, N_CT_A, N_PP_A, N_CC_B,   N_TC_B, N_CT_B, N_PP_B;

int /* Capture temporaire des payoffs de l'agent A. */
    Tmp_N_CC_A = atoi(MatriceForm→ME_CC_A→Text.c_str()),
    Tmp_N_TC_A = atoi(MatriceForm→ME_TC_A→Text.c_str()),
    Tmp_N_CT_A = atoi(MatriceForm→ME_CT_A→Text.c_str()),
    Tmp_N_PP_A = atoi(MatriceForm→ME_PP_A→Text.c_str()),
    /* Capture temporaire des payoffs de l'agent B. */
    Tmp_N_CC_B = atoi(MatriceForm→ME_CC_B→Text.c_str()),
    Tmp_N_TC_B = atoi(MatriceForm→ME_TC_B→Text.c_str()),
    Tmp_N_CT_B = atoi(MatriceForm→ME_CT_B→Text.c_str()),
    Tmp_N_PP_B = atoi(MatriceForm→ME_PP_B→Text.c_str());
    If  (     (Verif_Param_DP_Clas(&Tmp_N_CC_A, &Tmp_N_TC_A, &Tmp_N_CT_A, &Tmp_N_PP_A)==true)
        &&  (Verif_Param_DP_Clas(&Tmp_N_CC_B, &Tmp_N_CT_B, &Tmp_N_TC_B, &Tmp_N_PP_B)==true))
            {
            N_CC_A = Tmp_N_CC_A;
            N_TC_A = Tmp_N_TC_A;
            N_CT_A = Tmp_N_CT_A;
            N_PP_A = Tmp_N_PP_A;
            N_CC_B = Tmp_N_CC_B;
            N_TC_B = Tmp_N_TC_B;
            N_CT_B = Tmp_N_CT_B;
            N_PP_B = Tmp_N_PP_B;
            Close();
            }
```

```
    else    {
            MessageBox(Handle, "Les paramètres introduits pour le traitement du dilemme du prisonnier
                classique sont incorrects", "Problème de paramètres",MB_OK|MB_ICONEXCLAMATION);
            }
}
```

```
void __fastcall TMatriceForm::FormResize(TObject *Sender)
{
MatriceForm→Width = MatriceForm→Width;
MatriceForm→Height = MatriceForm→Height;
}
```

```
void __fastcall TMatriceForm::BitBtn_Annul_ModifMatClick(TObject *Sender)
{
Close();
}
```

Mode_Evolution_Population.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Mode_Evolution_Population.h"
#include "TypeDef.h"
#include "VariablesGlobales.h"
#include "IHM.h"
#include "Gestion_Pop.h"

extern void Aff_Result_Confront_Var_Medium(float i, int j, unsigned int k, unsigned int l);
extern void Ajustement_Pop_Var(float Payoff_Medium, unsigned int period, unsigned int Nb_Tot_Indiv_Pop);
extern unsigned int Estimation_Pop_End_Period_Var_Medium(float Payoff_Medium_Per_Indiv);
extern Type_Evolution_Population Regle_Evol_Pop;
extern Type_Mode_Calcul Regle_Mode_Calcul_Var;
extern info_init_strat *list_Strat_Selec;
```

```
void Mode_Evol_Pop(int periode)
{
info_init_strat *List_Strat_Selec_Tmp = list_Strat_Selec; /* De même, List_Strat_Selec_Tmp va permettre d'obtenir
les informations nécessaires au sein de la liste contenant les renseignements relatifs à un type de stratégie
particulier. */
unsigned int Diff_Payoff, Payoff_Acc_Global, Nb_Eff_Pop_Tot_Beginning, Nb_Eff_Pop_Tot_End;
float Payoff_Per_Indiv, Numerateur, Denominateur;
Payoff_Acc_Global = 0;
Nb_Eff_Pop_Tot_Beginning = 0;
while (List_Strat_Selec_Tmp != NULL)
    {
    if (Regle_Mode_Calcul_Var == Payoff_Medium)
        {
        Diff_Payoff = (List_Strat_Selec_Tmp→Payoff_acc_Actuel - List_Strat_Selec_Tmp→Payoff_acc_Preced);
        Payoff_Acc_Global += Diff_Payoff;
        }
    else {
        if (List_Strat_Selec_Tmp→Nb_Instance > 0)
            {
            Payoff_Acc_Global += List_Strat_Selec_Tmp→Payoff_acc_Actuel;
            }
        }
    Nb_Eff_Pop_Tot_Beginning += List_Strat_Selec_Tmp→Nb_Instance;
    List_Strat_Selec_Tmp = List_Strat_Selec_Tmp→Next;
    }
/* Pour l'ensemble de la population, on détermine le payoff moyen par individu. */
if (Nb_Eff_Pop_Tot_Beginning == 0)
      {
      ShowMessage("Problème au niveau du calcul du nombre d'individus en début de période!");
      Payoff_Per_Indiv = Payoff_Acc_Global;
      }
else {
      Numerateur = Payoff_Acc_Global;
      Denominateur = Nb_Eff_Pop_Tot_Beginning;
      Payoff_Per_Indiv = Numerateur / Denominateur;
      }
```

```
if (Regle_Evol_Pop == Variable)
    {
    Nb_Eff_Pop_Tot_End = Estimation_Pop_End_Period_Var_Medium(Payoff_Per_Indiv);
    switch (Regle_Mode_Calcul_Var)
        {
        case Payoff_Medium:     Aff_Result_Confront_Var_Medium(Payoff_Per_Indiv,
            periode, Nb_Eff_Pop_Tot_Beginning, Nb_Eff_Pop_Tot_End);
            Ajustement_Pop_Var(Payoff_Per_Indiv, periode, Nb_Eff_Pop_Tot_Beginning);
            break;
        }
    }
else {
    MessageDlg("Cette alternative doit encore être implémentée.",
            mtWarning,TMsgDlgButtons()<<mbOK,0);
    /* A faire
    switch (Regle_Mode_Calcul_Const)
        {
        case: break;
        case: break;
        }*/
    }
}
```

OptionSaisieStrat.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "OptionSaisieStrat.h"
#include "SaisieStratConfr.h"
#include "Strategies.h"
#include "VariablesGlobales.h"
#include "TypeDef.h"
#pragma resource "*.dfm"
```

```
TOptionSaisieStratForm *OptionSaisieStratForm;

extern Tab_Strat Tableau_Strat;
extern info_init_strat *list_Strat_Selec;
extern Info_Initiale Val_Temp;
extern info_init_strat *Ptr_Elem_List;
extern info_init_strat *Ptr_Elem_List;
```

```
__fastcall TOptionSaisieStratForm::TOptionSaisieStratForm(TComponent* Owner): TForm(Owner)
{
}
```

```
void Save_Val_Ptr_Elem_List()
{
Ptr_Elem_List→Nb_Instance = OptionSaisieStratForm→MEPopInit→Text.ToInt();
Ptr_Elem_List→Tx_Croiss_Pop = OptionSaisieStratForm→MECroissPop→Text.ToInt();
Ptr_Elem_List→Proba = OptionSaisieStratForm→MEPourcentCoop→Text.ToInt();
if (OptionSaisieStratForm→METypeAgent→Text == "T")
      {
      Ptr_Elem_List→Agent = Val_Type;
      }
else  {
      if (OptionSaisieStratForm→METypeAgent→Text == "A")
            {
            Ptr_Elem_List→Agent = Agent_A;
            }
      else  {
            Ptr_Elem_List→Agent = Agent_B;
            }
      }
Ptr_Elem_List→Couleur = OptionSaisieStratForm→ShapeColor→Brush→Color;
Ptr_Elem_List→Epaisseur = OptionSaisieStratForm→EditTaille→Text.ToInt();
}
```

```
int Num_Strat_In_Combo_Box(AnsiString s)
{
Tab_Strat *tab_tmp = &Tableau_Strat;
unsigned int j=0;
char *buf;
int n = s.Length();
buf = new char [n+10];
lstrcpy(buf, tab_tmp[j].Nom_Strat);
AnsiString tmp;
tmp = buf;
while (tmp.AnsiCompare(s)!=0)
    {
    j++;
    lstrcpy(buf, tab_tmp[j].Nom_Strat);
    tmp = buf;
    }
return (j);
}
```

```
void __fastcall TOptionSaisieStratForm::ComboBoxNameStratChange(TObject *Sender)
{
/* Sauvegarde des différents champs de l'ancienne boîte. */
Save_Val_Ptr_Elem_List();
info_init_strat *tmp_list = list_Strat_Selec;
/* Initialisation des différents paramètres associés à la nouvelle stratégie sélectionnée. Dans un premier temps, on
recherche l'identifiant de la stratégie. Par la suite, on positionne un pointeur sur l'élément sélectionné. */
// Première phase:
AnsiString tmp = ComboBoxNameStrat→Text;
unsigned int i = Num_Strat_In_Combo_Box(tmp);
// Deuxième phase:
while (tmp_list→N_Strat != i)
    {
    tmp_list = tmp_list→Next;
    }
Ptr_Elem_List = tmp_list;
// Troisième phase: sauvegarde des valeurs initiales.
Val_Temp.Agent = tmp_list→Agent;
Val_Temp.N_Strat = tmp_list→N_Strat;
Val_Temp.Nb_Instance = tmp_list→Nb_Instance;
Val_Temp.Proba = tmp_list→Proba;
Val_Temp.Tx_Croiss_Pop = tmp_list→Tx_Croiss_Pop;
Val_Temp.Couleur = tmp_list→Couleur;
Val_Temp.Epaisseur = tmp_list→Epaisseur;
// Nous arrivons enfin à la phase d'initialisation proprement dite des différents champs.
int j = tmp_list→Nb_Instance;
AnsiString s1(j);
MEPopInit→Text = s1;
j = tmp_list→Tx_Croiss_Pop;
AnsiString s2(j);
MECroissPop→Text = s2;
if (tmp_list→Proba == 200)
      {
      LabPourcentageCoop→Visible = false;
      MEPourcentCoop→Visible = false;
      LabPourcentCoop→Visible = false;
      UpDown2→Visible = false;
```

```
        LabStratNonProb→Visible = true;
        }
else    {
        LabPourcentageCoop→Visible = true;
        MEPourcentCoop→Visible = true;
        j = tmp_list→Proba;
        AnsiString s3(j);
        MEPourcentCoop→Text = s3;
        LabPourcentCoop→Visible = true;
        UpDown2→Visible = true;
        LabStratNonProb→Visible = false;
        }
if (tmp_list→Agent == Val_Type)
        {
        METypeAgent→Text = "T";
        }
else    {
        if (tmp_list→Agent == Agent_A)
                {
                METypeAgent→Text = "A";
                }
        else    {
                METypeAgent→Text = "B";
                }
        }
ShapeColor→Brush→Color = tmp_list→Couleur;
j = tmp_list→Epaisseur;
AnsiString s4(j);
EditTaille→Text = s4;
BitBtnRestaure→→Enabled = false;
}
```

```
void __fastcall TOptionSaisieStratForm::METypeAgentDblClick(TObject *Sender)
{
if (METypeAgent→Text == "T")
        {
        METypeAgent→Text = "A";
        }
else    {
        if (METypeAgent→Text == "A")
                {
                METypeAgent→Text = "B";
                }
        else    {
                METypeAgent→Text = "A";
                }
        }
BitBtnRestaure→Enabled = true;
}
```

```
void __fastcall TOptionSaisieStratForm::METypeAgentKeyUp(TObject *Sender,WORD &Key, TShiftState Shift)
{
char *buf;
buf = new char[200];
wsprintf(buf,"Vous devez choisir entre l'agent A, l'agent B ou les valeurs typiques.\nPour ce, vous devez introduire
respectivement le caractère 'A', 'B' ou 'T'.");
AnsiString caractere = METypeAgent→Text.Trim();
caractere = caractere.UpperCase();
if ((caractere == "A") || (caractere == "B") || (caractere == "T"))
      {
      BitBtnRestaure→Enabled = true;
      }
else  {
      MessageBox(Handle,buf,"Caractère introduit erroné", MB_ICONSTOP);
      }
}
```

```
void __fastcall TOptionSaisieStratForm::BitBtnRestaureClick(TObject *Sender)
{
Ptr_Elem_List→Agent = Val_Temp.Agent;
Ptr_Elem_List→N_Strat = Val_Temp.N_Strat;
Ptr_Elem_List→Nb_Instance = Val_Temp.Nb_Instance;
Ptr_Elem_List→Proba = Val_Temp.Proba;
Ptr_Elem_List→Tx_Croiss_Pop = Val_Temp.Tx_Croiss_Pop;
Ptr_Elem_List→Couleur = Val_Temp.Couleur;
Ptr_Elem_List→Epaisseur = Val_Temp.Epaisseur;
// Phase de mise à jour des différents champs de l'affichage
int i;
int j = Ptr_Elem_List→Nb_Instance;
AnsiString s1(j);
MEPopInit→Text = s1;
j = Ptr_Elem_List→Tx_Croiss_Pop;
AnsiString s2(j);
MECroissPop→Text = s2;
if (Ptr_Elem_List→Proba == 200)
      {
      LabPourcentageCoop→Visible = false;
      MEPourcentCoop→Visible = false;
      LabPourcentCoop→Visible = false;
      UpDown2→Visible = false;
      LabStratNonProb→Visible = true;
      }
else  {
      LabPourcentageCoop→Visible = true;
      MEPourcentCoop→Visible = true;
      j = Ptr_Elem_List→Proba;
      AnsiString s3(j);
      MEPourcentCoop→Text = s3;
      LabPourcentCoop→Visible = true;
      UpDown2→Visible = true;
      LabStratNonProb→Visible = false;
      }
if (Ptr_Elem_List→Agent == Agent_A)
      {
      METypeAgent→Text = "A";
      }
```

```
else {
    if (Ptr_Elem_List→Agent == Agent_B)
        {
        METypeAgent→Text = "B";
        }
    else {
        METypeAgent→Text = "T";
        }
    }
ShapeColor→Brush→Color = Val_Temp.Couleur;
i = Val_Temp.Epaisseur;
AnsiString s4(i);
OptionSaisieStratForm→EditTaille→Text = s4;
BitBtnRestaure→Enabled = false;
}
```

```
void __fastcall TOptionSaisieStratForm::SpeedButton1Click(TObject *Sender)
{
info_init_strat *tmp_list = list_Strat_Selec;
if (ColorDialog→Execute())
    {
    BitBtnRestaure→Enabled = true;
    Ptr_Elem_List→Couleur = ColorDialog→Color;
    ShapeColor→Brush→Color = tmp_list→Couleur;
    }
}
```

```
bool Verification_Valeurs_Introduites()
{
int i;
bool Result = true;
// Vérification de la valeur de 'MECroissPop'
AnsiString v0(OptionSaisieStratForm→MEPopInit→Text);
if (v0.ToIntDef(0) < 1)
    {
    i = Ptr_Elem_List→Nb_Instance;
    AnsiString s2(i);
    OptionSaisieStratForm→MEPopInit→Text = s2;
    }
else {
    Ptr_Elem_List→Nb_Instance = Val_Temp.Nb_Instance;
    }
// Vérification de la valeur de 'MECroissPop'
AnsiString v1(OptionSaisieStratForm→MECroissPop→Text);
if ((v1.ToIntDef(0)<(-99)) || (v1.ToIntDef(0)>100))
    {
    i = Ptr_Elem_List→Tx_Croiss_Pop;
    AnsiString s2(i);
    OptionSaisieStratForm→MECroissPop→Text = s2;
    Result = false;
    }
else {
    Ptr_Elem_List→Tx_Croiss_Pop = Val_Temp.Tx_Croiss_Pop;
    }
// Vérification de la valeur de 'METypeAgent'
```

```
AnsiString caractere = OptionSaisieStratForm→METypeAgent→Text.Trim();
caractere = caractere.UpperCase();
if ((caractere == "A") || (caractere == "B") || (caractere == "T"))
      {
      Ptr_Elem_List→Agent = Val_Temp.Agent;
      }
else  {
      Result = false;
      if (Ptr_Elem_List→Agent == Agent_A)
            {
            OptionSaisieStratForm→METypeAgent→Text = "A";
            }
      else  {
            if (Ptr_Elem_List→Agent == Agent_B)
                  {
                  OptionSaisieStratForm→METypeAgent→Text = "B";
                  }
            else  {
                  OptionSaisieStratForm→METypeAgent→Text = "T";
                  }
            }
      }
// Vérification de la valeur de 'EditTaille'
AnsiString v3(OptionSaisieStratForm→EditTaille→Text);
if (v3.ToIntDef(0)< 1)
      {
      Result = false;
      i = Ptr_Elem_List→Couleur;
      AnsiString s2(i);
      OptionSaisieStratForm→MECroissPop→Text = s2;
      }
else  {
      Ptr_Elem_List→Couleur = Val_Temp.Couleur;
      }
// Vérification de la valeur de 'MEPourcentCoop'
AnsiString v4(OptionSaisieStratForm→MEPourcentCoop→Text);
if ((v4.ToIntDef(0)<0) || (v4.ToIntDef(0)>100))
      {
      Result = false;
      i = Ptr_Elem_List→Proba;
      AnsiString s2(i);
      OptionSaisieStratForm→MEPourcentCoop→Text = s2;
      }
else  {
      Ptr_Elem_List→Proba = Val_Temp.Proba;
      }
return(Result);
}
```

```
void __fastcall TOptionSaisieStratForm::BitBtnOkClick(TObject *Sender)
{
char *Buf;
Buf = new char[200];
wsprintf(Buf,"Un ou plusieurs paramètres introduits sont erronés.");
bool No_Problem = Verification_Valeurs_Introduites();
if (No_Problem == true)
      {
      Save_Val_Ptr_Elem_List();
      }
else  {
      MessageBox(Handle,Buf,"Problème au niveau des paramètres", MB_ICONSTOP);
      }
Close();
}
void __fastcall TOptionSaisieStratForm::UpDown1Click(TObject *Sender,TUDBtnType Button)
{
Ptr_Elem_List→Agent = Val_Temp.Agent;
Ptr_Elem_List→N_Strat = Val_Temp.N_Strat;
Ptr_Elem_List→Nb_Instance = Val_Temp.Nb_Instance;
Ptr_Elem_List→Proba = Val_Temp.Proba;
Ptr_Elem_List→Tx_Croiss_Pop = Val_Temp.Tx_Croiss_Pop;
Ptr_Elem_List→Couleur = Val_Temp.Couleur;
Ptr_Elem_List→Epaisseur = Val_Temp.Epaisseur;
}
```

```
void __fastcall TOptionSaisieStratForm::ShapeColorDragDrop(TObject *Sender,TObject *Source, int X, int Y)
{
Ptr_Elem_List→Couleur = Val_Temp.Couleur;
}
```

```
void __fastcall TOptionSaisieStratForm::MEPopInitExit(TObject *Sender)
{
int i;
char *buf;
buf = new char[200];
wsprintf(buf,"Toute stratégie doit avoir au moins un individu.");
AnsiString s(MEPopInit→Text);
if (s.ToIntDef(0) < 1)
      {
      MessageBox(Handle,buf,"Problème de format", MB_ICONSTOP);
      i = Ptr_Elem_List→Nb_Instance;
      AnsiString s2(i);
      MEPopInit→Text = s2;
      }
else  {
      BitBtnRestaure→Enabled = true;
      Ptr_Elem_List→Nb_Instance = Val_Temp.Nb_Instance;
      }
}
```

```
void __fastcall TOptionSaisieStratForm::MECroissPopExit(TObject *Sender)
{
int i;
```

```
char *buf;
buf = new char[200];
wsprintf(buf,"Vous devez introduire un entier compris entre -99 et 100.");
AnsiString s(MECroissPop→Text);
if ((s.ToIntDef(0)<(-99)) || (s.ToIntDef(0)>100))
      {
      MessageBox(Handle,buf,"Problème de borne", MB_ICONSTOP);
      i = Ptr_Elem_List→Tx_Croiss_Pop;
      AnsiString s2(i);
      MECroissPop→Text = s2;
      }
else  {
      BitBtnRestaure→Enabled = true;
      Ptr_Elem_List→Tx_Croiss_Pop = Val_Temp.Tx_Croiss_Pop;
      }
}
```

```
void __fastcall TOptionSaisieStratForm::METypeAgentExit(TObject *Sender)
{
char *buf;
buf = new char[200];
wsprintf(buf,"Vous devez choisir entre l'agent A, l'agent B ou les valeurs typiques.\nPour ce, vous devez introduire
respectivement le caractère 'A', 'B' ou 'T'.");
AnsiString caractere = METypeAgent→Text.Trim();
caractere = caractere.UpperCase();
if ((caractere == "A") || (caractere == "B") || (caractere == "T"))
      {
      BitBtnRestaure→Enabled = true;
      Ptr_Elem_List→Agent = Val_Temp.Agent;
      }
else  {
      MessageBox(Handle,buf,"Caractère introduit erroné", MB_ICONSTOP);
      if (Ptr_Elem_List→Agent == Agent_A)
            {
            METypeAgent→Text = "A";
            }
      else  {
            if (Ptr_Elem_List→Agent == Agent_B)
                  {
                  METypeAgent→Text = "B";
                  }
            else  {
                  METypeAgent→Text = "T";
                  }
            }
      }
}
```

```
void __fastcall TOptionSaisieStratForm::MEPourcentCoopExit(TObject *Sender)
{
int i;
char *buf;
buf = new char[200];
wsprintf(buf,"Vous devez introduire un nombre entier compris entre 0 et 100.");
AnsiString s(MEPourcentCoop→Text);
if ((s.ToIntDef(0)<0) || (s.ToIntDef(0)>100))
```

```
        {
        MessageBox(Handle,buf,"Problème de borne", MB_ICONSTOP);
        i = Ptr_Elem_List→Proba;
        AnsiString s2(i);
        MEPourcentCoop→Text = s2;
        }
else    {
        BitBtnRestaure→Enabled = true;
        Ptr_Elem_List→Proba = Val_Temp.Proba;
        }
}
```

```
void __fastcall TOptionSaisieStratForm::EditTailleExit(TObject *Sender)
{
int i;
char *buf;
buf = new char[200];
wsprintf(buf,"Vous devez introduire une valeur entière strictement positive.");
AnsiString s(EditTaille→Text);
if (s.ToIntDef(0)< 1)
        {
        MessageBox(Handle,buf,"Problème de borne", MB_ICONSTOP);
        i = Ptr_Elem_List→Couleur;
        AnsiString s2(i);
        MECroissPop→Text = s2;
        }
else    {
        BitBtnRestaure→Enabled = true;
        Ptr_Elem_List→Couleur = Val_Temp.Couleur;
        }
}
```

Principal.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "Principal.h"
#include "DefMat.h"
#include "LongConf.h"
#include "SaisieStratConfr.h"
#include "VariablesGlobales.h"
#include "Gestion_Pop.h"
#include "IHM.h"
#include "TypeDef.h"
#include "VariablesGlobales.h"
#pragma link "Grids"
#pragma resource "*.dfm"
```

```
TFormPrincipal *FormPrincipal;

extern void Kill_list_Strat_Selec();
extern void Aff_Couleur_Strat();
extern void Aff_Memo_Regle_Calc(Type_Mode_Calcul Choice);
extern void Affichage_Tableau_Resultat();

extern Tab_Strat Tableau_Strat;
extern Type_Affichage Affich_Result;
extern int Max_Conf;
extern int Max_Confrontation;
extern bool PasSuivant;
extern bool Pas_A_Pas;
extern Type_Evolution_Population Regle_Evol_Pop;
extern Type_Mode_Calcul Regle_Mode_Calcul_Var;
extern Type_Mode_Calcul Regle_Mode_Calcul_Const;
```

```
__fastcall TFormPrincipal::TFormPrincipal(TComponent* Owner): TForm(Owner)
{
}
```

```
void __fastcall TFormPrincipal::Matricedespoints1Click(TObject *Sender)
{
DefMatForm→Hide();
DefMatForm→Show();
}
```

```
void __fastcall TFormPrincipal::Longueurdujeu1Click(TObject *Sender)
{
AnsiString s(Max_Conf);
LongConfForm→ME_LengthConf→Text = s;
LongConfForm→Hide();
LongConfForm→Show();
}
```

```
void __fastcall TFormPrincipal::QuitterClick(TObject *Sender)
{
FormPrincipal→Close();
}
```

```
void __fastcall TFormPrincipal::FormResize(TObject *Sender)
{
FormPrincipal→Width = Screen→Width;
FormPrincipal→Height = Screen→Height;
}
```

```
void __fastcall Initialisation_ListBox()
{
SaisieStratConfrForm→ListBoxTableauDeSelection→Items→Clear();
Tab_Strat *Tab_Tmp = &Tableau_Strat;
for (int i=0; i<Taille_Tableau_Strat; i++)
    {
    SaisieStratConfrForm→ListBoxTableauDeSelection→Items→Add(Tab_Tmp[i].Nom_Strat);
    }
}
```

```
void __fastcall TFormPrincipal::DP_ClasClick(TObject *Sender)
{
SaisieStratConfrForm→BitBtnEnlever→Enabled=false;
SaisieStratConfrForm→BitBtnToutEnlever→Enabled=false;
SaisieStratConfrForm→BitBtnOk→Enabled=false;
SaisieStratConfrForm→BitBtnOption→Enabled=false;
SaisieStratConfrForm→BitBtnAjouter→Enabled=true;
SaisieStratConfrForm→BitBtnToutAjouter→Enabled=true;
Kill_list_Strat_Selec();
Initialisation_ListBox();
SaisieStratConfrForm→ListBoxTab_Strat_Selectionnees→Items→Clear();
SaisieStratConfrForm→Caption = "Sélection de stratégies dans le cas du dilemme du prisonnier classique";
SaisieStratConfrForm→Hide();
SaisieStratConfrForm→Show();
}
```

```
void __fastcall TFormPrincipal::Retourdascenseur1Click(TObject *Sender)
{
SaisieStratConfrForm→BitBtnEnlever→Enabled=false;
SaisieStratConfrForm→BitBtnToutEnlever→Enabled=false;
SaisieStratConfrForm→BitBtnOk→Enabled=false;
SaisieStratConfrForm→BitBtnOption→Enabled=false;
SaisieStratConfrForm→BitBtnAjouter→Enabled=true;
SaisieStratConfrForm→BitBtnToutAjouter→Enabled=true;
Kill_list_Strat_Selec();
Initialisation_ListBox();
SaisieStratConfrForm→ListBoxTab_Strat_Selectionnees→Items→Clear();
SaisieStratConfrForm→Caption = "Sélection de stratégies dans le cas du retour d'ascenseur";
SaisieStratConfrForm→Hide();
SaisieStratConfrForm→Show();
```

```
}
```

```
void __fastcall TFormPrincipal::Paspas1Click(TObject *Sender)
{
Pas_A_Pas = true;
Paspas1→Checked = true;
Encontinu1→Checked = false;
PasSuivant = false;
}
```

```
void __fastcall TFormPrincipal::Encontinu1Click(TObject *Sender)
{
Pas_A_Pas = false;
Paspas1→Checked = false;
Encontinu1→Checked = true;
PasSuivant = true;
}
```

```
void __fastcall TFormPrincipal::BitBtnPasSuivantClick(TObject *Sender)
{
PasSuivant = true;
}
```

```
void __fastcall TFormPrincipal::ProportionClick(TObject *Sender)
{
Proportion→Checked = true;
Nbre_indiv→Checked = false;
Revenu_Accumule→Checked = false;
Affich_Result = Prorata;
Affichage_Tableau_Resultat();
}
```

```
void __fastcall TFormPrincipal::Nbre_indivClick(TObject *Sender)
{
Proportion→Checked = false;
Nbre_indiv→Checked = true;
Revenu_Accumule→Checked = false;
Affich_Result = Nb_Indiv;
Affichage_Tableau_Resultat();
b}
```

```
void __fastcall TFormPrincipal::Revenu_AccumuleClick(TObject *Sender)
{
Proportion→Checked = false;
Nbre_indiv→Checked = false;
Revenu_Accumule→Checked = true;
Affich_Result = Revenu_Acc;
Affichage_Tableau_Resultat();
}
```

SaisieStratConfr.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "SaisieStratConfr.h"
#include "Strategies.h"
#include "OptionSaisieStrat.h"
#include "TypeDef.h"
#include "Confrontation.h"
#include "Principal.h"
#include "Gestion_Pop.h"
#include "VariablesGlobales.h"
#pragma resource "*.dfm"
```

```
TSaisieStratConfrForm *SaisieStratConfrForm;

extern void Confrontation(Alternative Payoffs);
extern void Kill_list_indiv();
extern unsigned int Nbr_Strat();
extern void Kill_list_Strat_Selec();
extern void Kill_Elem_list_Strat_Selec(AnsiString s);

extern Tab_Strat Tableau_Strat;
extern info_indiv *Liste_Pop;

extern int N_CC_A, N_TC_A, N_CT_A, N_PP_A, N_CC_B, N_TC_B, N_CT_B, N_PP_B;
extern int DP_Clas_R, DP_Clas_T, DP_Clas_N, DP_Clas_P;
extern long Renonc_A, Renonc_B, Renoncement;
extern long DA_N, DA_P, DA_R, DA_T, Renonc_DA;
extern unsigned int DA_CC_A, DA_TC_A, DA_CT_A, DA_PP_A, DA_CC_B, DA_TC_B, DA_CT_B, DA_PP_B,
Renonc_A_DA, Renonc_B_DA;
extern unsigned int Nbr_Strat();
extern Info_Initiale Val_Temp;
extern info_init_strat *Ptr_Elem_List;
extern info_init_strat *list_Strat_Selec;
```

```
__fastcall TSaisieStratConfrForm::TSaisieStratConfrForm(TComponent* Owner): TForm(Owner)
{
}
```

```
void __fastcall TSaisieStratConfrForm::FormResize(TObject *Sender)
{
SaisieStratConfrForm→Height = 560;
SaisieStratConfrForm→Width = 634;
}
```

```
void Find(AnsiString s)
{
int k = 0;
while (s.AnsiCompare(OptionSaisieStratForm→ComboBoxNameStrat→Items→Strings[k])==0)
    {
    k++;
    }
OptionSaisieStratForm→ComboBoxNameStrat→Items→Delete(k);
}
```

```
unsigned int Long_List_Strat_Selec()
{
info_init_strat *list = list_Strat_Selec;
unsigned int i = 1;
while (list!=NULL)
    {
    list = list→Next;
    i++;
    }
return(i);
}
```

```
int Selection_Couleur()
{
int i = Nbr_Strat();
int Result;
switch (i)
    {
    case 0:    Result = (int) RGB(0,0,128);
               break;
    case 1:    Result = (int) RGB(0,128,0);
               break;
    case 2:    Result = (int) RGB(0,128,128);
               break;
    case 3:    Result = (int) RGB(128,0,0);
               break;
    case 4:    Result = (int) RGB(128,0,128);
               break;
    case 5:    Result = (int) RGB(128,128,0);
               break;
    case 6:    Result = (int) RGB(128,128,128);
               break;
    case 7:    Result = (int) RGB(192,192,192);
               break;
    case 8:    Result = (int) RGB(0,0,255);
               break;
    case 9:    Result = (int) RGB(0,255,0);
               break;
    case 10:   Result = (int) RGB(0,255,255);
               break;
    case 11:   Result = (int) RGB(255,0,0);
               break;
    case 12:   Result = (int) RGB(255,0,255);
               break;
    case 13:   Result = (int) RGB(255,255,0);
```

```
            break;
    default:  Result = (int) RGB(0,0,0);
    }
return (Result);
}
```

```
void Insert_Elem(int i)
{
/* L'argument 'i' indique la position de la stratégie au sein de la table. */
Tab_Strat *tab_tmp = &Tableau_Strat;
info_init_strat *list;
list = (info_init_strat*) malloc(sizeof(info_init_strat));
list→N_Strat = i;
list→Agent = Val_Type;
list→Nb_Instance = 100; /* Par défaut, on initialise le nombre d'instances d'une stratégie à 100 unités. */
if (tab_tmp[i].Flag == Strat_Non_Prob)
      {
      list→Proba = 200; /* Les stratégies non probabilistes reçoivent une valeur égale à 200. */
      }
else  {
      list→Proba = 50; /* Par défaut, les stratégies probabilistes reçoivent la valeur 50. */
      }
list→Tx_Croiss_Pop = 0; /* Par défaut, on considère qu'une stragtégie n'a pas de croissance exogène.*/
list→Payoff_acc_Preced = 0; /* Avant de commencer la confrontation, les payoffs accumulés d'un type de stratégie
sont nuls.*/
list→Payoff_acc_Actuel = 0; /* Avant de commencer la confrontation, les payoffs accumulés d'un type de stratégie
sont nuls.*/
list→Next = list_Strat_Selec;
list_Strat_Selec = list;
list→Couleur = Selection_Couleur();
list→Epaisseur = 1;
OptionSaisieStratForm→ComboBoxNameStrat→Items→Add(tab_tmp[i].Nom_Strat);
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnAjouterClick(TObject *Sender)
{
int j;
Tab_Strat *tab_tmp;
char *buf;
while (ListBoxTableauDeSelection→SelCount>0)
    {
    if (ListBoxTableauDeSelection→Items→Count==1)
    /* Si 'ListBoxTableauDeSelection→Items→Count' vaut 1. Cela signifie que l'utilisateur a sélectionné au moins
    une instance de chaque stratégie proposée. */
        {
        BitBtnAjouter→Enabled=false;
        BitBtnToutAjouter→Enabled=false;
        }
    int i=0;
    while (ListBoxTableauDeSelection→Selected[i]!=true)
        {
        i++;
        }
    AnsiString s = ListBoxTableauDeSelection→Items→Strings[i];
    ListBoxTab_Strat_Selectionnees→Items→Insert(0,s);
```

```
    ListBoxTableauDeSelection→Items→Delete(i);
    tab_tmp = &Tableau_Strat;
    j = 0;
    buf = new char [100];
    lstrcpy(buf, tab_tmp[j].Nom_Strat);
    AnsiString tmp = buf;
    while (tmp.AnsiCompare(s) != 0)
        {
        j++;
        lstrcpy(buf, tab_tmp[j].Nom_Strat);
        tmp = buf;
        }
    Insert_Elem(j);
    BitBtnEnlever→Enabled=true;
    BitBtnToutEnlever→Enabled=true;
    BitBtnOk→Enabled=true;
    BitBtnOption→Enabled=true;
    }
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnToutAjouterClick(TObject *Sender)
{
/* Cette fonction a pour objet de mettre l'ensemble des stratégies disponibles dans la
ListBoxTab_Strat_Selectionnees.
Attention: Les stratégies se trouvant dans cette dernière liste sont toutes initialisées avec des valeurs par défaut. */
Tab_Strat *tab_tmp = &Tableau_Strat;
ListBoxTableauDeSelection→Items→Clear();
ListBoxTab_Strat_Selectionnees→Items→Clear();
OptionSaisieStratForm→ComboBoxNameStrat→Items→Clear();
if (list_Strat_Selec!=NULL)
    Kill_list_Strat_Selec();
for (int i=0; i<Taille_Tableau_Strat; i++)
    {
    ListBoxTab_Strat_Selectionnees→Items→Add(tab_tmp[i].Nom_Strat);
    Insert_Elem(i);
    }
BitBtnEnlever→Enabled=true;
BitBtnToutEnlever→Enabled=true;
BitBtnOption→Enabled=true;
BitBtnOk→Enabled=true;
BitBtnAjouter→Enabled=false;
BitBtnToutAjouter→Enabled=false;
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnToutEnleverClick(TObject *Sender)
{
Tab_Strat *tab_tmp = &Tableau_Strat;
ListBoxTableauDeSelection→Items→Clear();
ListBoxTab_Strat_Selectionnees→Items→Clear();
OptionSaisieStratForm→ComboBoxNameStrat→Items→Clear();
if (list_Strat_Selec!=NULL)
        {
        Kill_list_Strat_Selec();
        }
list_Strat_Selec = NULL;
```

```
for (int i=0; i<Taille_Tableau_Strat; i++)
        {
        ListBoxTableauDeSelection→Items→Add(tab_tmp[i].Nom_Strat);
        }
BitBtnAjouter→Enabled=true;
BitBtnToutAjouter→Enabled=true;
BitBtnOk→Enabled=false;
BitBtnEnlever→Enabled=false;
BitBtnToutEnlever→Enabled=false;
BitBtnOption→Enabled=false;
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnEnleverClick(TObject *Sender)
{
int i=0;
while (ListBoxTab_Strat_Selectionnees→SelCount>0)
    {
    while (ListBoxTab_Strat_Selectionnees→Selected[i] != true)
        {
        i++;
        }
    ListBoxTableauDeSelection→Items→Insert(0,ListBoxTab_Strat_Selectionnees→Items→Strings[i]);
    Kill_Elem_list_Strat_Selec(ListBoxTab_Strat_Selectionnees→Items→Strings[i]);
    Find(ListBoxTab_Strat_Selectionnees→Items→Strings[i]);
    ListBoxTab_Strat_Selectionnees→Items→Delete(i);
    BitBtnAjouter→Enabled=true;
    BitBtnToutAjouter→Enabled=true;
    if (ListBoxTab_Strat_Selectionnees→Items→Count==0)
        {
        BitBtnEnlever→Enabled=false;
        BitBtnToutEnlever→Enabled=false;
        BitBtnOk→Enabled=false;
        BitBtnOption→Enabled=false;
        list_Strat_Selec = NULL;
        }
    }
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnOptionClick(TObject *Sender)
{
AnsiString s, tmp;
int i;
unsigned int j=0;
char *buf;
buf = new char [100];
s = ListBoxTab_Strat_Selectionnees→Items→Strings[0];
OptionSaisieStratForm→ComboBoxNameStrat→Text = s;
/* Dans un premier temps on recherche l'indice qui correspond à l'AnsiString dont on dispose. Pour rappel, cet
indice est l'identifiant de l'élément sélectionnée dans la liste composée. */
Tab_Strat *tab_tmp = &Tableau_Strat;
lstrcpy(buf, tab_tmp[j].Nom_Strat);
tmp = buf;
while (tmp.AnsiCompare(s)!=0)
    {
    j++;
```

```
    lstrcpy(buf, tab_tmp[j].Nom_Strat);
    tmp = buf;
    }
/* On vient d'obtenir le numéro souhaité. Il 'suffit' alors de se positionner sur la boîte disposant de l'ensemble des
informations relatives à la stratégie sélectionnée. */
info_init_strat *tmp_list = list_Strat_Selec;
while (tmp_list→N_Strat != j)
    {
    tmp_list = tmp_list→→Next;
    }
/* Mémorisation de l'adresse de la case mémoire de la stratégie qui va être affichée dans la ComboBox de la fiche
OptionSaisieStrat. */
Ptr_Elem_List = tmp_list;
/* Sauvegarde des valeurs initiales associées à la case mémoire tmp_list. */
Val_Temp.Agent = tmp_list→Agent;
Val_Temp.N_Strat = tmp_list→N_Strat;
Val_Temp.Nb_Instance = tmp_list→Nb_Instance;
Val_Temp.Proba = tmp_list→Proba;
Val_Temp.Tx_Croiss_Pop = tmp_list→Tx_Croiss_Pop;
Val_Temp.Couleur = tmp_list→Couleur;
Val_Temp.Epaisseur = tmp_list→Epaisseur;
/* 'tmp_list' pointe vers l'élément souhaité. On peut dès lors effectuer la MAJ des différents champs de la fiche
OptionSaisieStratForm. */
i = tmp_list→Nb_Instance;
AnsiString s1(i);
OptionSaisieStratForm→MEPopInit→Text = s1;
i = tmp_list→Tx_Croiss_Pop;
AnsiString s2(i);
OptionSaisieStratForm→MECroissPop→Text = s2;
if (tmp_list→Proba == 200)
      {
      OptionSaisieStratForm→LabPourcentageCoop→Visible = false;
      OptionSaisieStratForm→MEPourcentCoop→Visible = false;
      OptionSaisieStratForm→LabPourcentCoop→Visible = false;
      OptionSaisieStratForm→UpDown2→Visible = false;
      OptionSaisieStratForm→LabStratNonProb→Visible = true;
      }
else  {
      OptionSaisieStratForm→LabPourcentageCoop→Visible = true;
      OptionSaisieStratForm→MEPourcentCoop→Visible = true;
      i = tmp_list→Proba;
      AnsiString s3(i);
      OptionSaisieStratForm→MEPourcentCoop→Text = s3;
      OptionSaisieStratForm→LabPourcentCoop→Visible = true;
      OptionSaisieStratForm→UpDown2→Visible = true;
      OptionSaisieStratForm→LabStratNonProb→Visible = false;
      }
if (tmp_list→Agent == Agent_A)
      {
      OptionSaisieStratForm→METypeAgent→Text = "A";
      }
else  {
      if (tmp_list→Agent == Agent_B)
            {
            OptionSaisieStratForm→METypeAgent→Text = "B";
            }
      else  {
```

```
            OptionSaisieStratForm→METypeAgent→Text = "T";
            }
        }
OptionSaisieStratForm→ShapeColor→Brush→Color = tmp_list→Couleur;
i = tmp_list→Epaisseur;
AnsiString s4(i);
OptionSaisieStratForm→EditTaille→Text = s4;
OptionSaisieStratForm→BitBtnRestaure→Enabled = false;
OptionSaisieStratForm→Hide();
OptionSaisieStratForm→ShowModal();
}
Alternative Initialisation_Payoff()
{
Alternative tmp;
if (SaisieStratConfrForm→Caption == "Sélection de stratégies dans le cas du dilemme du prisonnier classique")
    {
    tmp.Individu_A.Coop = N_CC_A;
    tmp.Individu_A.Betrayal = N_TC_A;
    tmp.Individu_A.Naive = N_CT_A;
    tmp.Individu_A.Punishment = N_PP_A;
    tmp.Individu_A.Abandonment = Renonc_A;
    tmp.Individu_B.Coop = N_CC_B;
    tmp.Individu_B.Betrayal = N_CT_B;
    tmp.Individu_B.Naive = N_TC_B;
    tmp.Individu_B.Punishment = N_PP_B;
    tmp.Individu_B.Abandonment = Renonc_B;
    tmp.Valeur_Type.Coop = DP_Clas_R;
    tmp.Valeur_Type.Betrayal = DP_Clas_T;
    tmp.Valeur_Type.Naive = DP_Clas_N;
    tmp.Valeur_Type.Punishment = DP_Clas_P;
    tmp.Valeur_Type.Abandonment = Renoncement;
    }
if (SaisieStratConfrForm→Caption == "Sélection de stratégies dans le cas du retour d'ascenseur")
    {
    tmp.Individu_A.Coop = DA_CC_A;
    tmp.Individu_A.Betrayal = DA_TC_A;
    tmp.Individu_A.Naive = DA_CT_A;
    tmp.Individu_A.Punishment = DA_PP_A;
    tmp.Individu_A.Abandonment = Renonc_A_DA;
    tmp.Individu_B.Coop = DA_CC_B;
    tmp.Individu_B.Betrayal = DA_CT_B;
    tmp.Individu_B.Naive = DA_TC_B;
    tmp.Individu_B.Punishment = DA_PP_B;
    tmp.Individu_B.Abandonment = Renonc_B_DA;
    tmp.Valeur_Type.Coop = DA_R;
    tmp.Valeur_Type.Betrayal = DA_T;
    tmp.Valeur_Type.Naive = DA_N;
    tmp.Valeur_Type.Punishment = DA_P;
    tmp.Valeur_Type.Abandonment = Renonc_DA;
    }
return (tmp);
}
```

```
void __fastcall TSaisieStratConfrForm::BitBtnOkClick(TObject *Sender)
{
Close();
```

```
Alternative Valeur_Payoff = Initialisation_Payoff();
Kill_list_indiv();
FormPrincipal→Plot_Evol_Pop→Visible = true;
FormPrincipal→PanelPrincipalClient→Cursor = crHourGlass;
Confrontation(Valeur_Payoff);
Screen→Cursor = crArrow;
}
void __fastcall TSaisieStratConfrForm::BitBtnAnnulerClick(TObject *Sender)
{
Close();
}
```

```
void __fastcall TSaisieStratConfrForm::ListBoxTab_Strat_SelectionneesDblClick(TObject *Sender)
{
AnsiString s, tmp;
Tab_Strat *tab_tmp = &Tableau_Strat;
info_init_strat *tmp_list = list_Strat_Selec;
int i = 0;
unsigned int j=0;
char *buf;
buf = new char [30];
while (ListBoxTab_Strat_Selectionnees→Selected[i]!=true)
    {
    i++;
    }
s = ListBoxTab_Strat_Selectionnees→Items→Strings[i];
OptionSaisieStratForm→ComboBoxNameStrat→Text = s;
/* Dans un premier temps on recherche l'indice qui correspond à l'AnsiString dont on dispose. Pour rappel, cet
indice est l'identifiant de l'élément sélectionné dans la liste composée. */
lstrcpy(buf, tab_tmp[j].Nom_Strat);
tmp = buf;
while (tmp.AnsiCompare(s) != 0)
    {
    j++;
    lstrcpy(buf, tab_tmp[j].Nom_Strat);
    tmp = buf;
    }
/* On vient d'obtenir l'identifiant souhaité. Il 'suffit' alors de se positionner sur la boîte disposant de l'ensemble des
informations relatives à la stratégie sélectionnée. */
while (tmp_list→N_Strat != j)
    {
    tmp_list = tmp_list→Next;
    }
/* Mémorisation de l'adresse de la case mémoire de la stratégie qui va être affichée dans la ComboBox de la fiche
OptionSaisieStrat. */
Ptr_Elem_List = tmp_list;
/* Sauvegarde des valeurs initiales associées à la case mémoire tmp_list. */
Val_Temp.Agent = tmp_list→Agent;
Val_Temp.N_Strat = tmp_list→N_Strat;
Val_Temp.Nb_Instance = tmp_list→Nb_Instance;
Val_Temp.Proba = tmp_list→Proba;
Val_Temp.Tx_Croiss_Pop = tmp_list→Tx_Croiss_Pop;
Val_Temp.Couleur = tmp_list→Couleur;
Val_Temp.Epaisseur = tmp_list→Epaisseur;
/* 'tmp_list' pointe vers l'élément souhaité. On peut dès lors effectuer la MAJ des différents champs de la fiche
OptionSaisieStratForm. */
i = tmp_list→Nb_Instance;
```

```
AnsiString s1(i);
OptionSaisieStratForm→MEPopInit→Text = s1;
i = tmp_list→Tx_Croiss_Pop;
AnsiString s2(i);
OptionSaisieStratForm→MECroissPop→Text = s2;
if (tmp_list→Proba == 200)
        {
        OptionSaisieStratForm→LabPourcentageCoop→Visible = false;
        OptionSaisieStratForm→MEPourcentCoop→Visible = false;
        OptionSaisieStratForm→LabPourcentCoop→Visible = false;
        OptionSaisieStratForm→UpDown2→Visible = false;
        OptionSaisieStratForm→LabStratNonProb→Visible = true;
        }
else    {
        OptionSaisieStratForm→LabPourcentageCoop→Visible = true;
        OptionSaisieStratForm→MEPourcentCoop→Visible = true;
        i = tmp_list→Proba;
        AnsiString s3(i);
        OptionSaisieStratForm→MEPourcentCoop→Text = s3;
        OptionSaisieStratForm→LabPourcentCoop→Visible = true;
        OptionSaisieStratForm→UpDown2→Visible = true;
        OptionSaisieStratForm→LabStratNonProb→Visible = false;
        }
if (tmp_list→Agent == Agent_A)
      {
      OptionSaisieStratForm→METypeAgent→Text = "A";
      }
else  {
      if (tmp_list→Agent == Agent_B)
            {
            OptionSaisieStratForm→METypeAgent→Text = "B";
            }
      else  {
            OptionSaisieStratForm→METypeAgent→Text = "T";
            }
      }
OptionSaisieStratForm→ShapeColor→Brush→Color = tmp_list→Couleur;
i = tmp_list→Epaisseur;
AnsiString s4(i);
OptionSaisieStratForm→EditTaille→Text = s4;
OptionSaisieStratForm→BitBtnRestaure→Enabled = false;
OptionSaisieStratForm→Hide();
OptionSaisieStratForm→ShowModal();
}
```

Strategies.cpp

```
#include <vcl\vcl.h>
#include "TypeDef.h"
#pragma hdrstop
#include "Strategies.h"
```

```
extern int Last;
```

Stratégies non probabilistes

```
card Gentille(info Pas_info)
/* Stratégie acceptant de coopérer quelle que soit la tactique de l'adversaire. */
{
return (Cooperation);
}
```

```
card Mechant(info Pas_info)
/* Stratégie refusant toute coopération. */
{
return (Defection);
}
```

```
card Donnant_Donnant(info Behaviour_Opp)
/* Après une coopération initiale, cette stratégie calque son attitude sur celle adoptée par le concurrent durant la
période précédente. */
{
if (Last == 0)
      {
      return (Cooperation);
      }
else  {
      if (Behaviour_Opp.Last_Behaviour_Opp == Cooperation)
            {
            return (Cooperation);
            }
      else  {
            return (Defection);
            }
      }
}
```

```
card Rancunier(info Info_Ranc)
/* Cette stratégie est basée sur la coopération mais opte définitivement pour la 'non coopération' à la première
trahison de l'adversaire. */
{
if (Last == 0)
      {
      return (Cooperation);
      }
else  {
      if ((Info_Ranc.info_ranc.adv==Defection) || (Info_Ranc.info_ranc.moi==Defection))
            {
            Info_Ranc.info_ranc.moi=Defection;
            return (Defection);
            }
      else  {
            return (Cooperation);
            }
      }
}
```

```
card Per_Mechant(info Pas_Info)
/* Cette stratégie fait alterner des périodes de trahison et de coopération selon un ordre préalblement fixé, et qui est
le suivant: 'Trahison', 'Trahison' et 'Coopération'. */
{
if (Last == 0)
      {
      return (Defection);
      }
else  {
      if (Last%3 == 0)
            {
            return (Cooperation);
            }
      else  {
            return (Defection);
            }
      }
}
```

```
card Per_Gentille(info Pas_Info)
/* Cette stratégie fait alterner des périodes de trahison et de coopération selon un ordre préalblement fixé, et qui est
le suivant: 'Coopération', 'Coopération' et 'Trahison'. */
{
if (Last == 0)
      {
      return (Cooperation);
      }
else  {
      if (Last%3 == 0)
            {
            return (Defection);
            }
      else  {
            return (Cooperation);
            }
      }
```

```
}
card Maj_Mou(info Par_Maj_Mou)
{
/* Cette stratégie réagit de façon rétrospective par rapport aux tactiques précédentes adoptées par le concurrent.
Elle opte pour la coopération si celle-ci a été majoritairement appliquée par l'adversaire et la trahison dans le cas
contraire. En cas d'égalité, elle coopère toujours et agit de même initialement. */
if ( (Last == 0) || (2 * Par_Maj_Mou.Info_Nbre_Trahison.Nb_Trahison <= Last) )
      {
      return (Cooperation);
      }
else  {
      return (Defection);
      }
}
```

```
card Maj_dur(info Par_Maj_Mou)
{
/* Cette stratégie réagit de façon rétrospective par rapport aux tactiques précédentes adoptées par le concurrent.
Elle opte pour la coopération si celle-ci a été majoritairement appliquée par l'adversaire et la trahison dans le cas
contraire. En cas d'égalité, elle trahit et agit de même initialement. */
/* Stratégie jouant ce que l'adversaire a joué en majorité et trahit en cas d'égalité. Le coup initial est considéré
comme une égalité, c'est donc 'Defection' qu'on a initialement. */
if ( (Last == 0) || (2 * Par_Maj_Mou.Info_Nbre_Trahison.Nb_Trahison >= Last) )
      {
      return (Defection);
      }
else  {
      return (Cooperation);
      }
}
```

```
card Sondeur_1(info Info_Sonde)
/* Cette stratégie fixe son comportement en fonction de celui adopté par l'adversaire durant les trois premières
périodes. */
{
switch(Last)
    {
    case 0:    return(Defection);
    case 1:    return(Cooperation);
    case 2:    return(Cooperation);
    default:   if (   (Info_Sonde.Info_Sondage.Behavior_Second_Perriod == Cooperation)
               &&  (Info_Sonde.Info_Sondage.Behavior_Third_Perriod == Cooperation) )
                    {
                    return(Defection);
                    }
               else  {
                    /* On retrouve ci-dessous la fameuse stratégie du Donnant-Donnant.*/
                    if (Info_Sonde.Info_Sondage.Last_Behaviour_Opp == Cooperation)
                         {
                         return (Cooperation);
                         }
                    else  {
                         return (Defection);
                         }
                    }
    }
```

```
}
```

```
card Tit_For_Tat_Dur(info Par_TFT_Dur)
/* Cette stratégie coopère toujours sauf si l'autre a trahi au cours d'une des deux périodes précédentes. Elle est
analogue à 'Donnant-Donnant' avec une mémoire un peu plus longue puisqu'elle impose de trahir deux fois après
une trahison. */
{
if (Last == 0)
      {
      return(Cooperation);
      }
else  {
      if (   (Par_TFT_Dur.TFT_Dur.Last_Behaviour_Opp == Defection)
      ||     ( (Last > 1) && (Par_TFT_Dur.TFT_Dur.Beavior_Two_Period_Before == Defection) ) )
            {
            return(Defection);
            }
      else  {
            return(Cooperation);
            }
      }
}
```

```
card Mefiant(info Last_Behaviour_Opp)
/* Cette stratégie choisit d'aborder la trahison, puis calque son attitude sur celle adoptée par l'adversaire durant la
période précédente. Il s'agit d'un 'Donnant-Donnant' qui commence par trahir. */
{
if (Last == 0)
      {
      return (Defection);
      }
else  {
      if (Last_Behaviour_Opp.Last_Behaviour_Opp==Cooperation)
            {
            return (Cooperation);
            }
      else  {
            return (Defection);
            }
      }
}
```

Stratégies probabilistes

```
card Lunatique(info prob)
/* Stratégie coopérant avec une probabilité fixée dès le départ. */
{
float tmp = (rand()/RAND_MAX);
randomize();
if (tmp < prob.prob)
      {
      return (Defection);
      }
```

```
else {
    return (Cooperation);
    }
}
```

TypeDef.h

```
#ifndef TypeDefH
#define TypeDefH

typedef enum {Cooperation, Defection, Abandon}card;

typedef enum {Prorata, Nb_Indiv, Revenu_Acc}Type_Affichage;

typedef struct Vide_Type{
   /* Aucune information n'est utile pour l'exécution de la stratégie. */
   }vide;

typedef struct Ranc_Type{
   /* On mémorise son dernier comportement et si l'adversaire a, dans le passé, trahi une seule fois. */
   card adv; /* Information sur l'adversaire. */
   card moi; /* Information sur soi-même. */
   }ranc;

typedef struct Nbre_Trahison_Type{
   /* Mémorisation du nombre de trahison de l'adversaire. */
   int Nb_Trahison;
   }Nbre_Trahison;

typedef struct Sondage_Type{
   card Behavior_Second_Perriod;
   card Behavior_Third_Perriod;
   card Last_Behaviour_Opp;
   }Sondage;

typedef struct Behavior_Two_Last_Periods_Type{
   card Beavior_Two_Period_Before;
   card Last_Behaviour_Opp;
   }Behavior_Two_Last_Periods;

union info{
   float prob; /* Probabilité de coopérer dans le cas d'une stratégie probabilisite.*/
   card Last_Behaviour_Opp; /* Information nécessaire dans le cas du donnant-donnant.*/
   ranc info_ranc;
   Nbre_Trahison Info_Nbre_Trahison;
   Sondage Info_Sondage;
   Behavior_Two_Last_Periods TFT_Dur;
   };

typedef enum {Pas_Param, Ranc, Last_Behaviour_Opponent, Param_Nb_Trahison, Param_Sondage,
Param_TFT_Dur, Proba} TypeParam;

typedef card (*ptr_fct) (info); /* Toute fonction doit renvoyer un résultat du type défini 'card'.*/

typedef enum {Val_Type, Agent_A, Agent_B} type_agent;
```

```
typedef struct Info_Gen_Type{
/* Informations à enregistrer pour le fonctionnement d'une stratégie. On utilise également cette structure pour
mémoriser l'action posée par l'agent durant la période de décision (il s'agit du champ 'Action'). */
    struct Info_Gen_Type *Previous;
    unsigned int Ident; /* Identifiant de l'information stockée p.r. à l'adversaire. Il s'agit d'un type de stratégie ou de
                l'identifiant d'un agent particulier. */
    info Info_Strat_Adv; /* Information stockée p.r. à l'adversaire en ce qui concerne les périodes antérieures. */
    card Action; /* Action posée par l'individu pour la nouvelle période. */
    struct Info_Gen_Type *Next;
    }Info_Gen;

typedef struct Info_Indiv_Type{
    struct Info_Indiv_Type *Previous;
    unsigned int Ident_Strat;
    unsigned int Ident_Indiv;
    Info_Gen *Info_Non_Proba; /* Informations relatives aux strat. non probabilistes. */
    Info_Gen *Info_Proba; /* Informations relatives aux strat. probabilistes. */
    struct Info_Indiv_Type *Next;
    }info_indiv;

typedef struct Info_Init_Strat_Type{
    ptr_fct Adr; /* 'Adr' pointe vers l'adresse où la stratégie est définie. */
    type_agent Agent; /* Indique s'il s'agit d'un agent de type A ou B, ou si on utilise les valeurs typiques. */
    unsigned int N_Strat; /* Indique le n° du type de stratégie dont il est question. */
    unsigned int Nb_Instance; /* 'Nb_Instance' indique le nombre d'instances de la stratégie concernée. */
    unsigned int Proba; /* 'Proba' nous informe sur le fait s'il s'agit d'une stratégie probabiliste ou non. Les stratégies
                probabilistes peuvent avoir une valeur comprise entre 0 et 100 (étant donné qu'on exprime le fait
                qu'une stratégie a tendance à coopérer en pourcentage). Par défaut, ce champ reçoit, pour les
                stratégies probabilistes, 50 pour valeur. Les stratégies non probabilistes ont comme valeur pour ce
                champ 200 (de la sorte aucune confusion n'est possible). */
    int Tx_Croiss_Pop; /* Par défaut, on considère qu'une stragtégie n'a pas de croissance. La raison de ce choix
                s'explique par le fait que la survie d'une stratégie dépend de justifications endogènes. Rem: On
                considère que cette valeur doit être comprise entre -100 (on ne peut supprimer plus d'individus que la
                population totale de la stratégie concernée n'en possède) et 100. */
    unsigned int Payoff_acc_Preced; /* Ce champ nous sert pour enregistrer les payoffs accumulés précédemment par
                l'ensemble des individus de cette stratégie. */
    unsigned int Payoff_acc_Actuel; /* Ce champ nous sert pour enregistrer les payoffs accumulés par l'ensemble des
                individus de la stratégie en question. */
    /* Les deux derniers paramètres permettent de définir les caractéristiques des traits que nous allons tracer. */
    TColor Couleur;
    unsigned int Epaisseur;
    struct Info_Init_Strat_Type *Next;
    }info_init_strat;

typedef struct Nb_Instance_Strat_Type{ /* Cette structure permet d'enregistrer le nombre d'individus en début et en
                fin de période. */
    unsigned int N_Strat; /* Indique le n° du type de stratégie référencée. */
    unsigned int Nb_Instance_Beginning; /* 'Nb_Instance_Beginning' indique le nombre d'instances en début de
                période. */
    unsigned int Nb_Instance_End; /* 'Nb_Instance_End' indique le nombre d'instances en fin de période. */
    struct Nb_Instance_Strat_Type *Next;
    }Nb_Instance_Strat;
```

```
typedef struct Info_Initiale_Type{
/* La signification de chacun de ces différents champs est similaire à celle qui a été donnée pour les éléments
équivalents de la structure 'info_init_strat'. */
    type_agent Agent;
    unsigned int N_Strat;
    unsigned int Nb_Instance;
    unsigned int Proba;
    int Tx_Croiss_Pop;
    TColor Couleur;
    unsigned int Epaisseur;
    }Info_Initiale;

/* Définition de deux structures permettant de simplifier la recherche des payoffs des différentes stratégies. */

typedef struct Valeur_Type{
    int Coop;
    int Betrayal;
    int Naive;
    int Punishment;
    int Abandonment;
    }Valeur;

typedef struct Alternative_Type{
    Valeur Individu_A;
    Valeur Individu_B;
    Valeur Valeur_Type;
    }Alternative;

typedef struct Nb_Indiv_Par_Strat_Type{
    unsigned int Ident_Strat;
    unsigned int Nb_Instances;
    struct NB_Indiv_Par_Strat_Type *Next;
    }NB_Indiv_Par_Strat;
```

Définition du type 'Tab_Strat' qui sera utilisé pour regrouper l'ensemble des stratégies qui ont été définies. Pour chacune, on aura un nom, l'adresse de la fonction et un indice indiquant si elle est probabiliste ou non.

```
typedef enum {Strat_Non_Prob, Strat_Prob} Prob;

typedef struct Tab_Strat_Type{
    char *Nom_Strat; /* Ce champ donne l'intitulé de la stratégie. Il joue le rôle d'identifiant. */
    ptr_fct Adresse; /* Ce champ fournit l'adresse où la stratégie en question est déclarée. */
    TypeParam Param; /* Ce champ donne des renseigneents sur les paramètres qui devront être transmis à la
                stratégie pour qu'elle soit exécutée. */
    Prob Flag; /* Ce champ indique s'il s'agit d'une stratégie probabiliste ou non. */
    }Tab_Strat;
```

Définition des différents types nécessaires pour la sélection d'un mode de calcul particulier

```
typedef enum {Payoff_Medium} Type_Mode_Calcul;

typedef enum {Constante, Variable} Type_Evolution_Population;
#endif
```

VariablesGlobales.cpp

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "VariablesGlobales.h"
#include "TypeDef.h"
#include "Strategies.h"

// Variables utilisées pour déterminer le mode d'évolution de la confrontation

bool PasSuivant;

bool Pas_A_Pas = false;

// Variable utilisée pour déterminer le mode d'affichage des résultats obtenus pour chaque stratégie

Type_Affichage Affich_Result = Revenu_Acc;

/* Variable utilisée pour déterminer la boîte relative à une stratégie particulière. Elle permet d'en donner les
différents paramètres. */

info_init_strat *Ptr_Elem_List;

Info_Initiale Val_Temp;

/* Ci-dessous, nous avons les deux plus importantes variables globales utilisées au sein de ce programme. */

/* Cette variable pointe vers le premier individu de la population. */

info_indiv *Liste_Pop = NULL;

/* Cette variable pointe vers la première stratégie sélectionnée. */

info_init_strat *list_Strat_Selec = NULL;

/* Cette variable permet d'enregistrer, pour chaque stratégie, le nombre d'individus en début et en fin de période. */

Nb_Instance_Strat *List_Nb_Instance_Strat = NULL;

/* Déclaration des stratégies qui ont été définies au sein du fichier 'Stratégies.h'. */

extern card Gentille(info Pas_info);
extern card Mechant(info Pas_info);
extern card Donnant_Donnant(info Last_Behaviour_Opp);
extern card Rancunier(info info_ranc);
extern card Per_Mechant(info Pas_Info);
extern card Per_Gentille(info Pas_Info);
extern card Maj_Mou(info Par_Maj_Mou);
extern card Maj_dur(info Par_Maj_Mou);
extern card Sondeur_1(info Sondage);
extern card Tit_For_Tat_Dur(info TFT_Dur);
extern card Mefiant(info Last_Behaviour_Opp);
extern card Lunatique(info prob);
```

```
Tab_Strat Tableau_Strat[Taille_Tableau_Strat]=
    {
    {"Gentille", Gentille, Pas_Param, Strat_Non_Prob},
    {"Méchant", Mechant, Pas_Param, Strat_Non_Prob},
    {"Donnant Donnant", Donnant_Donnant, Last_Behaviour_Opponent, Strat_Non_Prob},
    {"Rancunier", Rancunier, Ranc, Strat_Non_Prob},
    {"Périodiquement méchant", Per_Mechant, Pas_Param, Strat_Non_Prob},
    {"Périodiquement gentille", Per_Gentille, Pas_Param, Strat_Non_Prob},
    {"Majorité-Mou", Maj_Mou, Param_Nb_Trahison, Strat_Non_Prob},
    {"Majorité-Dur", Maj_dur, Param_Nb_Trahison, Strat_Non_Prob},
    {"Sondeur (Version 1)", Sondeur_1, Param_Sondage,Strat_Non_Prob},
    {"Donnant Donnant Dur", Tit_For_Tat_Dur, Param_TFT_Dur,Strat_Non_Prob},
    {"Méfiant", Mefiant, Last_Behaviour_Opponent, Strat_Non_Prob},
    {"Lunatique", Lunatique, Proba, Strat_Prob}
    };

/* Cette variable sert pour indiquer quel est le numéro (identifiant) du dernier individu qui a été inséré au sein de la
liste de la population. */

int Nb_Individu = 0;

/* Durée par défaut de la confrontation.*/

int Max_Conf = 100;

/* La dernière période où il y a eu confrontation. On l'initialise à zéro pour indiquer qu'un nouveau jeu n'a pas
encore débuté. */

int Last = 0;

/* Variable indiquant si une confrontation opposant deux stratégies différentes a déjà eu lieu. */

bool Confront_In_The_Past = false;

/* Variables utilisées pour indiquer la règle adoptée pour l'évolution de la population. */

Type_Mode_Calcul Regle_Mode_Calcul_Var = Payoff_Medium;

Type_Evolution_Population Regle_Evol_Pop = Variable;
```

VarParDefaut.cpp

```
#ifndef VarParDefautH
#define VarParDefautH

/* Valeur par défaut pour le dilemme du prisonnier itéré classique. */
/*  DP_Clas_N < DP_Clas_P < DP_Clas_R < DP_Clas_T, et
    (DP_Clas_N + DP_Clas_T) < 2*DP_Clas_R
    N = Naïf, P = Puni, R = Reward, et T = Trahison. */

int DP_Clas_N = 0,
    DP_Clas_P = 1,
    DP_Clas_R = 3,
    DP_Clas_T = 5;

int N_CC_A = DP_Clas_R,
    N_TC_A = DP_Clas_T,
    N_CT_A = DP_Clas_N,
    N_PP_A = DP_Clas_P,
    N_CC_B = DP_Clas_R,
    N_TC_B = DP_Clas_N,
    N_CT_B = DP_Clas_T,
    N_PP_B = DP_Clas_P;

/* Dans le cas du renoncement, les deux joueurs obtiennent un payoff de 2 pour l'ensemble des parties restantes.*/

int Renoncement = 2;

int Renonc_A = Renoncement,
    Renonc_B = Renoncement;

/* Valeur par défaut pour le dilemme de l'ascenseur. */
/* Pour rappel, à la différence du dilemme du prisonnier itéré classique, les payoffs du dilemme de l'ascenseur sont
tels qu'ils pousssent les agents à adopter une stratégie de 'coopération' à phase décalée.*/
/* DA_N <= DA_P <= DA_R <= DA_T, mais à la différence du dilemme du prisonnier itéré classique, on a: (DA_N
+ DA_T)/2 > DA_R */

int DA_N = 0,
    DA_P = 1,
    DA_R = 3,
    DA_T = 8;

int DA_CC_A = DA_R,
    DA_TC_A = DA_T,
    DA_CT_A = DA_N,
    DA_PP_A = DA_P,
    DA_CC_B = DA_R,
    DA_TC_B = DA_N,
    DA_CT_B = DA_T,
    DA_PP_B = DA_P;

int Renonc_DA = 2;

int Renonc_A_DA = Renonc_DA,
    Renonc_B_DA = Renonc_DA;
#endif
```

Bibliographie

ABREU et RUBINSTEIN (1988), '***The Structure of Nash Equilibrium in repeated games with finite automate***', Econometrica n°56 (pg. 1259-1282).

AXELROD, R. (1984), '***The Evolution of Cooperation***', Basic Books, New York.

BINMORE, G. et SAMUELSON, L. (1992), '***Evolutionary Stability in Repeated Games Played by Finite Automata***', Journal of Economic Theory (pg. 278-305).

DELAHAYE, J-P., et MATHIEU, P.(1992), '***Experiences sur le dilemme itéré des prisonniers***'.

DELAHAYE, J-P., et MATHIEU, P.(1993), '***L'Altrisme perfectionné***'.

DELAHAYE, J-P., et MATHIEU, P.(1995), '***Complex Strategies in the Iterated Prisoner's Dilemma***'.

DELAHAYE, J-P., et MATHIEU, P.(1996 a), '***Random Strategies in a Two Levels Iterated Prisoner's Dilemma: How to Avoid Conflicts?***', 12th European Conference on Artificial Intelligence, édité par W. Wahlster.

DELAHAYE, J-P., et MATHIEU, P.(1996 b), '***Our Meeting With Gradual: A Good Strategy For The Iterated Prisoner's Dilemma***'.

DELAHAYE, J-P., et MATHIEU, P.(1996 c), '***The Iterated Lift Dilemma, or How to Establish Meta-Cooperation with your Opponent***'.

DELAHAYE, J-P., et MATHIEU, P.(1996 d), '***Etude sur les dynamiques du Dilemme Itéré des Prisonniers avec un petit nombre de stratégies: Y a-t-il du chaos dans le Dilemme pur?***'.

DRIX, Ph. (1990), '***Langage C norme ANSI***', Masson.

FARRENY, H., et GHALLAB, M. (1987), '***Traité des nouvelles techniques d'Intelligence Artificielle. Eléments d'Intelligence Artificiel***', Editeur Hernes, Paris - Londres - Lausanne.

HOWARD, J-V. (Avril 1988), '***Cooperation in the prisoner's dilemma***', Theoretical Economics.

KEHAGIAS, A. (1998), '***Probabilistic Learning Automata and the Prisoner's Dilemma***', non publié.

LEBLANC, G. (1996), '***Borland C++ Builder***', Eyrolles.

LUGER, G., et STUBBLEFIELD, W. (1997), '***Artificial Intelligence: Structures and Strategies for Complex Problem Solving***', Addison Wesley Longman, Third Edition.

MAILATH, G. R. (1992), '***Introduction: Symposium on Evolutionary Game Theory***', Journal of Economic Theory (pg. 259-277).

MATHIEU, P.(12 mars 1996 a), '***Dilemme Itéré des prisonniers***'.

SWINKELS, J. (1992), '***Evolutionary Stability with Equilibrium Entrants***', Journal of Economic Theory (pg. 259-277).

TELLES, M. (1997), '***Borland C++ Builder***', Coriolis Group Books.

VARIAN, H. (1992), '***Introduction à la micro-économie***', De Boeck Université, Bruxelles.

VELU, J. (1994), '***Méthodes mathématiques pour l'informatique***', Dunod.