Actes du Congrès
COLLÈGES CÉLÉBRATIONS 92
Conference Proceedings

MONTRÉAL MAY 24 25 26 27 MAI 1992

Systèmes à base de connaissances et enseignement assisté par ordinateur

par

Pierre DION
Collège Régional Champlain
(Québec)

Atelier 3D40

Collèges créateurs d'avenir
Colleges creators of the future
Association des collèges communautaires du Canada
Association québécoise de pédagogie collégiale

7000, rue Marie-Victorin, Montréal (Québec) H1G 2J6 • (514) 325-0150, poste 2210 • Télécopieur: (514) 328-3830

Systèmes à base de connaissances et enseignement assisté par ordinateur

Pierre Dion
Collège Régional Champlain
Campus de Lennoxville
(819) -564-3680

1. INTRODUCTION.

Le but de cet article est de montrer les possibilités qu'offrent les techniques liées à l'intelligence artificielle pour le développement de logiciels éducatifs plus souples et plus efficaces.

Bien qu'il existe à l'heure actuelle un certain nombre de didacticiels fort intéressants, plusieurs enseignants n'en éprouvent pas moins l'impression que l'ordinateur n'a pas rempli sa promesse dans le domaine de l'éducation. La révolution promise il y a quelques années ne s'est pas matérialisée. Une des raisons de cet état de fait provient certainement de l'investissment considérable en ressources humaines nécessaire au développement de logiciels utiles. Une autre raison concerne les limites de la plupart des didacticiels actuels: rigidité dans la présentation de la matière et interaction limitée avec l'étudiant.

Les systèmes à base de connaissances (systèmes experts) sont de plus en plus utilisés dans l'entreprise. Ils servent généralement d'aides à la prise de décision. Nous croyons qu'il est possible de s'inspirer de l'architecture de ces systèmes experts pour les adapter au domaine du logiciel éducatif, permettant du fait même d'y insérer un peu d'intelligence et de souplesse.

Dans la sections suivante nous allons exposer les principales limites des systèmes tuteurs classiques (section 2). Ensuite, dans la section 3, nous décrirons l'architecture des systèmes tutoriels intelligents. L'essentiel de cet article portera sur la description d'un système tutoriel en particulier, soit AlgorithmiK, un système tutoriel intelligent utilisé pour l'enseignement de l'algorithmique auprès d'étudiants en technique informatique (section 4). Pour terminer, nous dresserons un court bilan des réalisations actuelles et nous nous interrogerons sur les perspectives d'avenir pour ces systèmes dits intelligents.

2. LIMITES DE L'ENSEIGNEMENT ASSISTÉ PAR ORDINATEUR.

L'enseignement assisté par ordinateur possède déjà une assez longue histoire. Dès le début des années soixante, l'utilisation de l'ordinateur pour individualiser l'enseignement suscita beaucoup d'intérêt et d'espoir. Cependant, malgré un certain nombre de réalisations non négligeables et le développement de langages et de systèmes auteur, la majorité des didacticiels réalisés jusqu'à présent n'en demeurent pas moins limités dans leurs performances.

Les systèmes tutoriels conventionnels sont basés sur l'utilisation de blocs de matériel (texte, questions, branchements) entrés à l'avance par l'auteur. Une leçon typique (figure 1) est constituée d'unités d'apprentissage ayant toutes la même structure: un texte explicatif, accompagné ou non d'exemples, est présenté à l'apprenant; une ou plusieurs questions servent ensuite à vérifier sa compréhension; dans le cas d'une bonne réponse, l'étudiant reçoit un renforcement positif et est dirigé vers l'unité suivante; dans le cas d'une mauvaise réponse, il reçoit au contraire une rétroaction lui expliquant son erreur et se trouve dirigé vers une unité d'enseignement correctif [8].

Déjà, au début des années 70, Carbonell [5] soulignait les limites principales de l'EAO *orienté par fenêtres* ("frame-oriented"): l'étudiant a peu ou pas d'initiative, il ne peut répondre aux questions du système en langue naturelle, et il fait face à un système qui lui semble relativement rigide; l'enseignant, quant à lui, a la tâche considérable de prévoir les questions, réponses et branchements de la leçon. Le système ne peut donc faire preuve d'initiative; il ne possède aucun pouvoir de décision ni de véritable connaissance.

La rigidité du système se manifeste principalement par la démarche linéaire des leçons ne permettant qu'une *pseudo-individualisation* : "... après avoir éventuellement fait quelques détours personnalisés grâce à telle ou telle mauvaise réponse explicitement prévue, l'apprenant, dans le cheminement de la leçon, se retrouve finalement soit à un point où il est censé avoir appris ou compris l'élément précédent, soit à un point où il est censé ne pas avoir appris ou compris cet élément, et c'est alors le même type de question qui lui sera posé (avec peut-être des variantes dans les messages d'encouragement)" [11, p. 12]. Les conséquences de cette progression linéaire sont que le logiciel est incapable de prendre en compte l'individualité psychologique de l'apprenant (ses straté-

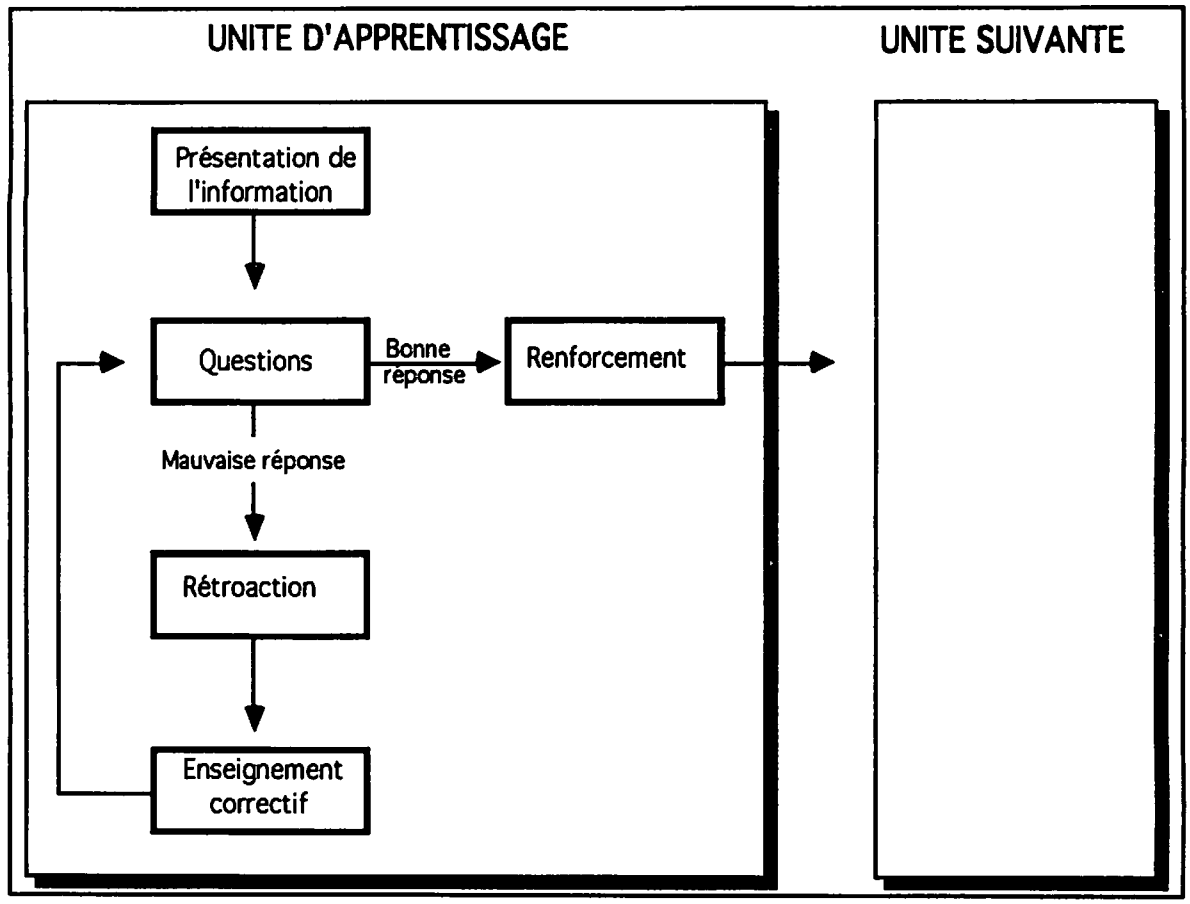

Figure 1. L'enseignement tutoriel classique

gies cognitives), son cheminement personnel et ses erreurs de compréhension.

L'EIAO veut remédier à ces lacunes: l'utilisation de techniques propres à l'intelligence artificielle facilite l'instauration d'un système d'enseignement à initiative partagée, favorisant un véritable dialogue personne-machine et simulant les pouvoirs d'inférence d'un tuteur humain. Devant l'ampleur manifeste de la tâche, les efforts de recherche se sont surtout portés sur le développement de systèmes fournissant un environnement de support à l'apprentissage dans des domaines restreints plutôt que sur le développement de cours complets.

En général, ces systèmes, désignés habituellement par *systèmes tutoriels intelligents* selon l'appellation de Sleeman et Brown [13], cherchent à combiner l'attrait d'une approche de type découverte ("discovery-learning") où l'apprenant est placé dans une situation de résolution de problèmes, et l'apport d'interactions tutorielles venant guider l'apprentissage. Les systèmes tutoriels intelligents offrent un environnement d'apprentissage *réactif* ("reactive learning environment") dans lequel l'étudiant est activement engagé avec le système d'enseignement et où ses intérêts et ses incompréhensions guident le dialogue tutoriel [3]. Un système intelligent peut ainsi mettre en évidence des possibilités diagnostiques bien supérieures aux systèmes conventionnels qui ne peuvent diagnostiquer que les fautes explicitement prévues.

3. ARCHITECTURE D'UN SYSTÈME TUTORIEL INTELLIGENT.

Pour pouvoir enseigner efficacement, un tuteur —humain ou mécanique— doit posséder trois types de connaissances portant sur (1) le domaine enseigné, (2) les caractéristiques de l'étudiant, et (3) comment enseigner [14]. Les composantes principales d'un STI idéal (figure 2) sont donc un expert du domaine, un modèle de l'étudiant, représentant l'état des connaissances de l'étudiant, et un module de tutorat incorporant un certain nombre de stratégies d'enseignement.

En pratique, il est rare qu'un STI possède ces trois composantes parfaitement articulées. En général, l'effort de

recherche se concentre sur une ou deux des composantes. De plus, chaque composante ne constitue pas nécessairement un module distinct à l'intérieur du système.

3.1 L'expert du domaine.

L'expert du domaine contient la connaissance de la matière enseignée. L'idée de base est qu'un système tutoriel ne peut être vraiment efficace que dans la mesure où il possède une certaine compétence dans le domaine qu'il enseigne. Cette expertise permet au logiciel de décider si la réponse de l'apprenant est juste ou fausse, de conseiller un étudiant en difficulté sur la méthode à utiliser, d'évaluer les différentes méthodes proposées par l'élève, de justifier son propre mode de raisonnement en expliquant ses choix, de détecter chez l'étudiant une erreur que celui-ci a faite [7].

L'expert peut servir à générer des problèmes ou des sujets de discussion. Il ne peut pas toujours résoudre les problèmes qu'il pose, mais il peut au moins reconnaître une solution fausse. Doté d'heuristiques intelligentes, l'expert peut interpréter les réponses non-traditionnelles d'un étudiant, c'est-à-dire une réponse correcte mais exprimée de façon inhabituelle. Pour ce faire, il est important que le système puisse résoudre les problèmes de plusieurs façons différentes.

3.2 Le modèle de l'étudiant.

Le modèle de l'étudiant représente la compréhension qu'a l'étudiant de la matière enseignée. Le but du modèle est de poser des hypothèses sur les erreurs de compréhension de l'étudiant et sur ses mauvais choix stratégiques, de façon à ce que le module de tutorat puisse les lui indiquer, lui expliquer pourquoi ils sont faux, et lui suggérer des corrections [3].

Deux types principaux de modèles sont utilisés. Dans certains systèmes (WEST [4], GUIDON [6], WUSOR [9]), le modèle de l'étudiant est formé par comparaison du comportement de l'étudiant avec celui de l'expert dans le même environnement. La connaissance de l'étudiant est alors perçue comme un *recouvrement* ("overlay") de celle de l'expert. La fonction du modèle consiste à déterminer quelles sont les connaissances ou habiletés de l'expert connues et maîtrisées par l'étudiant.

Dans le deuxième type de modèle, le *modèle déviant* ("buggy model"), les connaissances de l'étudiant sont perçues plutôt comme une déviation ou une perturbation de celles de l'expert. Le modèle cherche à déterminer les erreurs et les confusions de l'étudiant.

3.3 Le module de tutorat.

Le module de tutorat contient les stratégies et les règles gouvernant les interactions du système avec l'apprenant. Ce module doit déterminer *quand* interrompre l'étudiant, *quoi* lui dire et *comment*. Il doit sélectionner les problèmes à résoudre par l'étudiant, le superviser, critiquer sa performance, lui fournir de l'aide sur demande et choisir du matériel correctif.

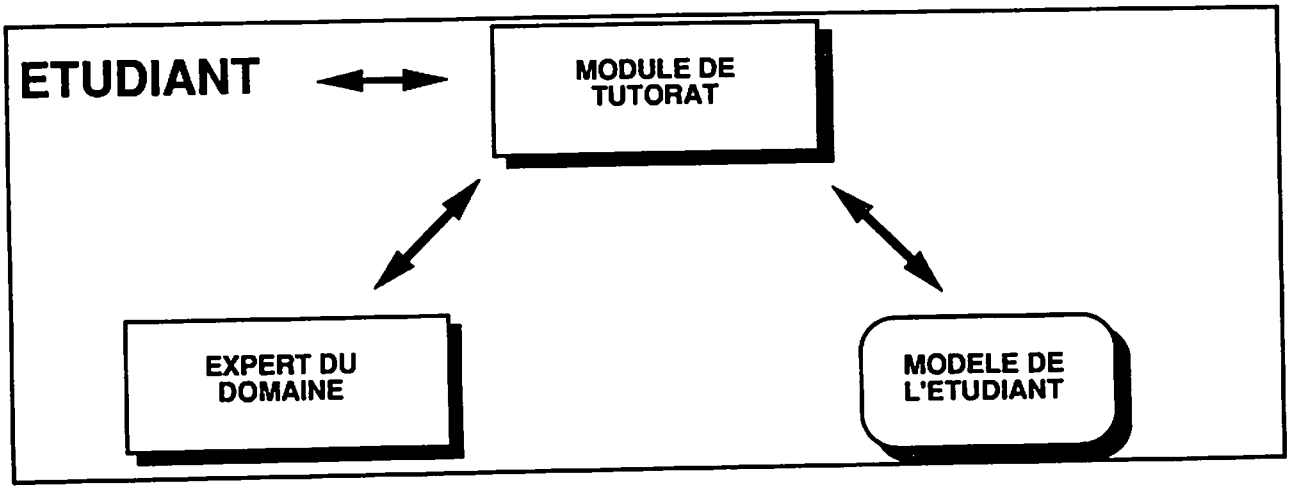

Figure 2. Les composantes d'un système tutoriel intelligent

Le module de tutorat constitue le coeur du système d'enseignement. Il utilise les connaissances de l'expert de même que l'information contenue dans le modèle de l'étudiant pour piloter le dialogue tutoriel. Le module de tutorat reflète le fait qu'un bon enseignant doit être un expert pédagogue et non seulement un expert de la matière enseignée.

Tout en laissant l'apprenant évoluer librement, le tuteur permet d'enrichir l'expérience de résolution de problèmes vécue par l'étudiant. Sans ce guide, l'étudiant risquerait de se débattre avec un problème conceptuel

qu'il ne peut résoudre ou d'escamoter des situations potentiellement très instructives.

4. ALGORITHMIK.

Le système AlgorithmiK est un système tutoriel intelligent conçu pour enseigner les notions de base de l'algorithmique et de la programmation à des étudiants de niveau collégial en technique informatique. Le développement d'AlgorithmiK a été rendu possible grâce à la participation financière et professionnelle de la Direction générale de l'enseignement collégial du ministère de l'Enseignement supérieur et de la Science.

AlgorithmiK est écrit en Pascal et en Prolog. Des versions pour environnement Macintosh et Windows seront disponibles en janvier '93 pour l'ensemble du réseau collégial.

AlgorithmiK n'enseigne pas comme tel l'algorithmique. Il est plutôt conçu pour être utilisé en laboratoire. La fonction du logiciel est de présenter des problèmes de programmation à l'étudiant et de l'aider dans le développement et la mise au point de son programme.

4.1 Philosophie de conception.

L'objectif à long terme de tout système tutoriel intelligent est de fournir l'équivalent d'un tuteur privé à l'étudiant. De façon plus modeste, nous voulons que notre système puisse indiquer et expliquer les erreurs conceptuelles trouvées dans le programme de l'étudiant et fournir un environnement convivial de développement de programmes

Pour pouvoir efficacement seconder un étudiant, le système doit être conçu selon les principes suivants:

- pour pouvoir évaluer la méthode qu'a utilisée l'étudiant pour atteindre un certain but de programmation, le système doit *comprendre* la démarche de l'étudiant, i.e. ce qu'il ou elle cherche à faire;

- puisque plusieurs algorithmes peuvent résoudre le même problème, le système doit pouvoir solutionner un problème en utilisant *plusieurs stratégies alternatives*;

- le système doit pouvoir affirmer si *le programme de l'étudiant est correct* ou non.

Pour toutes ces raisons, le système recherché doit être un *expert en algorithmique*.

Nous avons adopté une approche cognitive pour la conception du tuteur. Les premiers systèmes utilisés pour le diagnostic d'erreur en programmation [1,15] ont mis en évidence le besoin d'une telle approche. En effet, l'utilisation de connaissances et de techniques très générales en programmation seulement ne suffit pas pour localiser et expliquer des erreurs conceptuelles profondes des étudiants.

A l'opposé, le Tuteur LISP [2] et le système PROUST [10] ont clairement démontré l'importance de développer un module de diagnostic basé sur une théorie cognitive de la programmation. Pour comprendre la nature d'une erreur dans un programme informatique, il est essentiel de connaître les intentions de l'étudiant (donc la tâche à laquelle il s'attelle), c'est-à-dire la stratégie qu'il utilise. Une erreur n'a pas de signification en elle-même; elle est plutôt la manifestation d'un but de programmation mal réalisé. Un tuteur en programmation doit donc avoir accès à un ensemble de connaissances portant non seulement sur la programmation, mais aussi sur l'activité cognitive utilisée dans le développement et la mise au point de programmes.

4.2 Contexte d'enseignement.

L'outil pédagogique utilisé est *Karel le robot*, une méthode d'enseignement développée par Richard Pattis [12] pour fournir aux débutants "une introduction en douce à la programmation". Le langage de programmation robot est similaire à Pascal; cependant, il n'est pas possible de définir de variable ou de structure de données, ce qui permet à l'étudiant de porter son attention uniquement sur le concept de structure de contrôle, simplifiant ainsi le processus d'apprentissage.

La figure 3 illustre la structure du monde de Karel. Ce monde est constitué d'un quadrillage de rues et d'avenues bordés à l'ouest et au sud par un mur de longueur infinie, et contenant deux types d'objets: des *bippeurs* et des sections murales. Les bippeurs sont de petits cônes de plastique qui émettent un son très léger; ils sont situés aux intersections et peuvent être ramassés, transportés et déposés par Karel. Les sections murales, inamovibles, sont placées à mi-chemin entre deux intersections, bloquant ainsi la voie à Karel.

Karel peut *exécuter* les commandes primitives suivantes: avancer jusqu'à l'intersection suivante, tourner d'un quart de tour à gauche, ramasser un beeper et déposer un beeper. Il peut mémoriser un programme et l'exécuter. Il peut aussi *apprendre* la définition d'une nouvelle commande (cet aspect nous permet d'introduire le concept de procédure).

Karel est aussi doté d'un certain nombre de senseurs qui lui permettent de déterminer l'état de son environnement: une caméra pour détecter les sections murales situées à une demi-intersection de lui, une boussole pour déterminer son orientation, une facilité audio pour entendre les

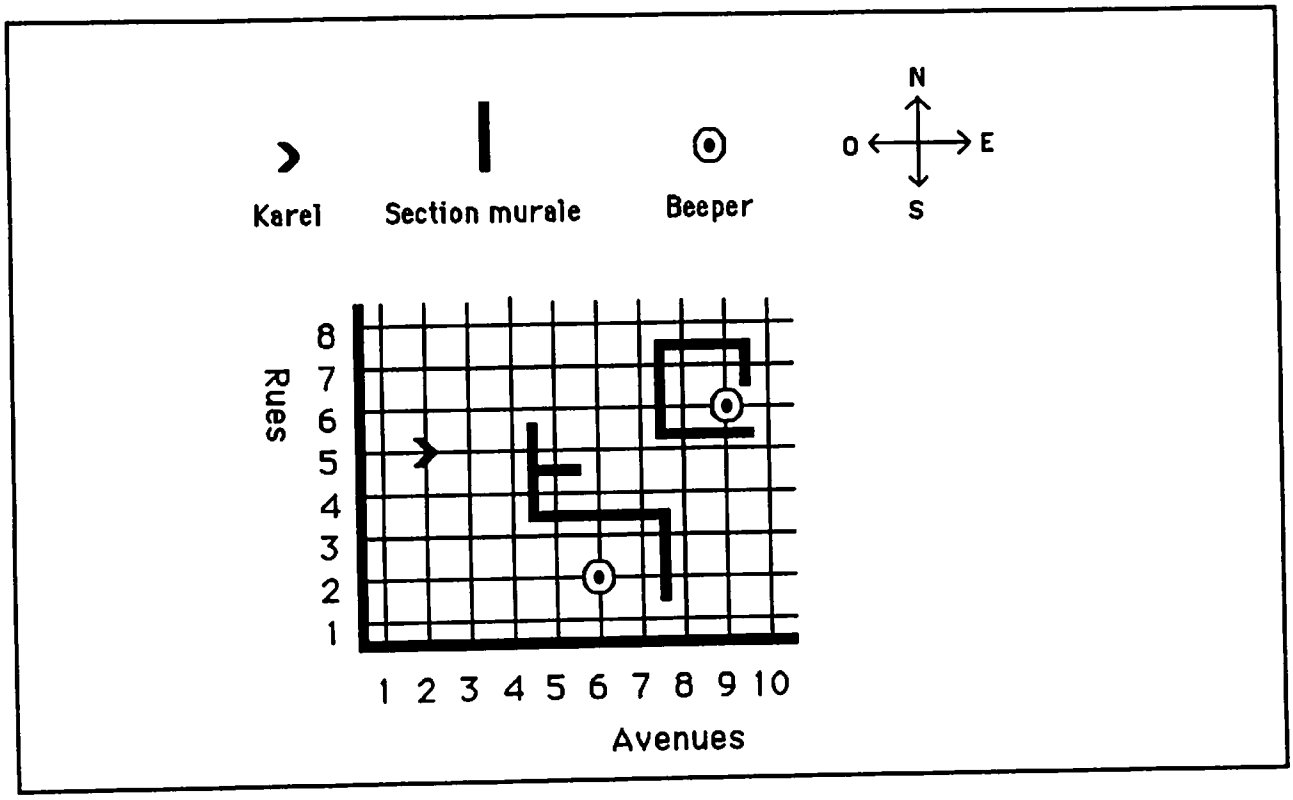

Figure 3. Le monde de Karel

bippeurs situés à la même intersection que lui. Karel peut donc répondre à certains *test* tels que: y-a-t-il un mur en face?, est-ce que je fais face au nord?, y-a-t-il un beeper à l'intersection?, etc...

Le robot peut ainsi être programmé pour accomplir diverses tâches, e.g. trouver l'issue d'une pièce, réaliser une course à obstacles, récolter un champ de bippeurs, ou se rendre à l'origine (intersection de la 1ère rue et de la 1ère avenue). Etant donné une situation initiale et une situation finale désirée, l'étudiant doit écrire la séquence d'instructions ordonnant au robot d'accomplir la tâche.

L'écriture de programmes simples mais non-triviaux requière l'utilisation de structures conditionnelles (`SI..ALORS..SINON`) et de répétition (`ITÉRER x FOIS`, `TANT_QUE..FAIRE`). L'étudiant est introduit au concept de procédure et à une méthodologie de conception de style *top-down*. Il apprend à procéder par raffinements successifs, en divisant un problème en sous-problèmes plus simples jusqu'à l'atteinte du niveau des commandes primitives de Karel.

4.3 Interface.

La figure 4 donne un instantané d'une session de travail avec AlgoritmiK. L'interface est de type graphique avec fenêtres, menus déroulants, et utilisation de la souris. Tel que mentionné plus haut, deux outils importants sont mis à la disposition de l'étudiant: un éditeur de structures et un simulateur graphique.

L'éditeur de structures élimine toute possibilité d'erreur de nature syntaxique. L'étudiant construit son programme en choisissant dans le menu *Instructions* la structure de contrôle, la primitive ou le test élémentaire qu'il veut insérer dans le programme. Le menu *Édition* lui permet de réaliser des transformations sur le programme qui en préserve l'exactitude syntaxique.

Le simulateur permet à l'étudiant de vérifier son programme. Il fournit la trace des actions du robot à mesure qu'il exécute les commandes du programme. L'étudiant peut aussi modifier l'environnement du robot pour tester son programme dans différentes situations initiales.

4.4 Rôle du tuteur.

Une fois le problème soumis, l'étudiant garde le contrôle absolu sur le déroulement de la session d'apprentissage. Il peut développer, tester et corriger son programme sans faire appel au tuteur ni être interrompu par celui-ci.

Dans ce contexte, le tuteur n'intervient qu'à la demande de l'étudiant, et son rôle est triple.

1º Choix de la tâche. Le premier rôle du tuteur est de choisir dans une banque de problèmes le problème (suivant) à soumettre à l'étudiant.

2º Conseil. Au cours de son activité de programmation, l'étudiant en panne peut demander au tuteur de lui fournir un indice pour compléter la partie du programme en cours de réalisation. Conformément au modèle de programmation adopté, les conseils fournis par le tuteur seront de niveau plus élevé au début du développement (correspondant à des buts plus généraux), et deviendront de plus en plus proches du code final à mesure que l'on s'approche des feuilles de l'arbre de décomposition fonctionnelle. Si l'indice transmis n'est pas suffisant, l'étudiant peut demander qu'une certaine partie du code lui soit fournie, ce qui lui permettra ainsi de poursuivre le développement de son programme.

3º Verdict. A tout moment lors de la session de travail, mais en particulier lorsqu'il estime avoir terminé son programme, l'étudiant peut demander au tuteur de vérifier sa solution (finale ou partielle). Dans ce cas, le tuteur pourra apposer un des verdicts suivants:

(1) Le programme est correct;

(2) La partie du programme actuellement réalisée est correcte;

(3) Le programme est incorrect (le tuteur fournit la localisation de l'erreur et sa description);

(4) Le programme semble correct bien que je ne comprenne pas la stratégie employée;

(5) Je ne comprends pas le programme mais il ne fonctionne pas dans la situation initiale suivante (le tuteur présente un contre-exemple).

4.4 Expert-programmeur.

L'expert-programmeur constitue l'expert du domaine du système tutoriel. La compréhension de la solution de l'étudiant nécessite des connaissances à la fois sur le

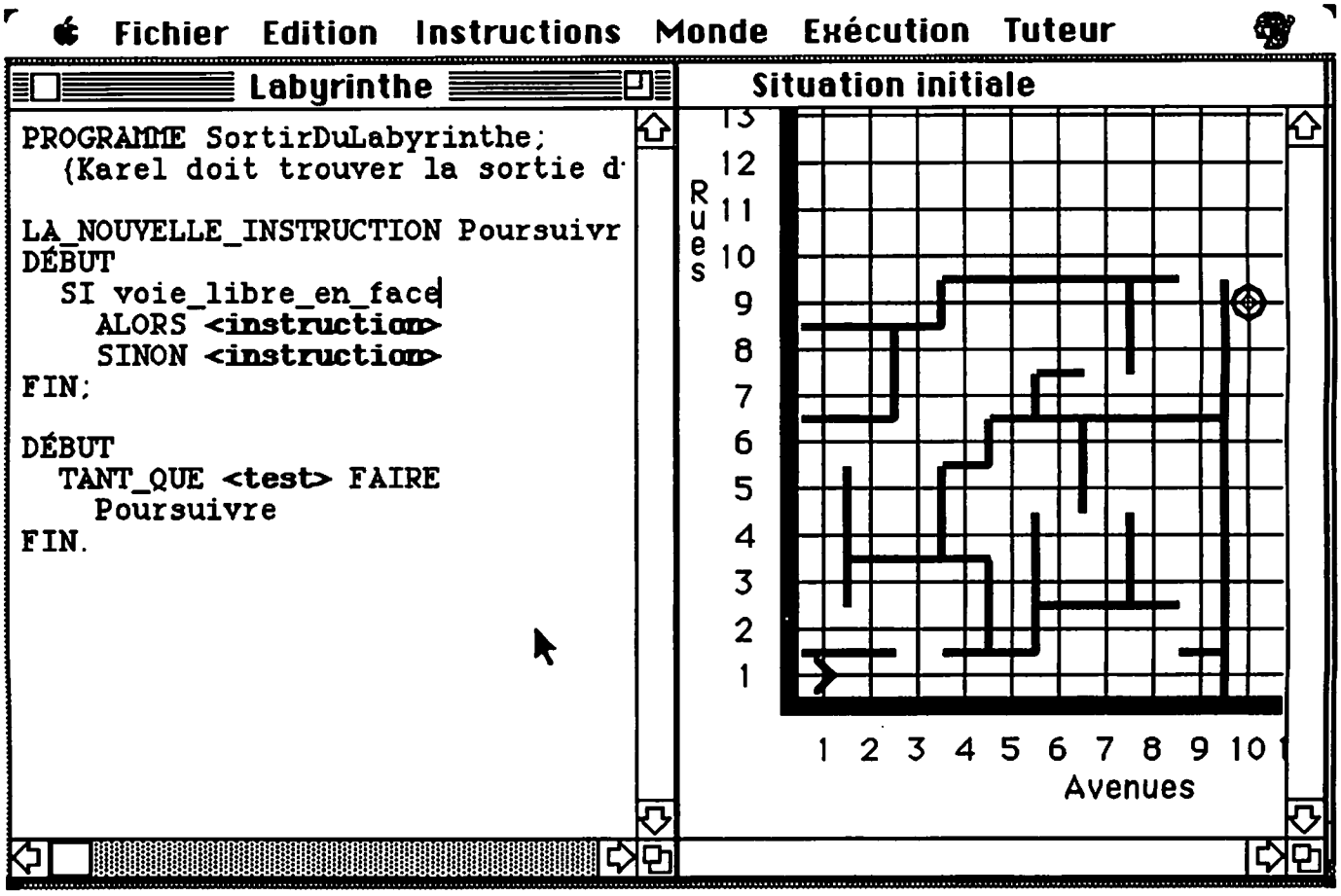

Figure 4. L'environnement d'AlgoritmiK

problème à résoudre et sur la programmation. Dans notre système, la connaissance du problème est fournie à l'expert au moyen d'un *langage interne de spécifications* servant à décrire la nature de la tâche à réaliser ainsi que le contexte dans lequel cette tâche s'inscrit. La connaissance de la programmation est distribuée entre deux spécialistes: l'*analyseur* et le *vérificateur*.

L'analyseur est responsable de la compréhension de *la structure statique du programme de l'étudiant*. Son but est d'identifier la stratégie de résolution utilisée par l'étudiant et de déterminer les erreurs (s'il y en a) commises dans l'application de cette stratégie. Sa connaissance est codifiée sous la forme de trois groupes de règles de production représentant trois aspects différents de l'expertise en programmation:

- des *règles de génération de code* permettant d'implanter les diverses stratégies qu'un programmeur humain pourrait utiliser pour solutionner le même problème;
- des *règles d'équivalence sémantique* permettant à l'analyseur de reconnaître et d'accepter des fragments de code sémantiquement équivalents;
- des *mal-règles* représentant des erreurs typiques de débutants;

Règles de génération de code.

Les règles de génération de code représentent la connaissance nécessaire à la conception et à l'écriture de programmes robots. Elles incorporent un ensemble de stratégies pouvant être utilisées pour obtenir la solution d'un problème. Ces règles permettent à l'analyseur de construire un programme correct à partir des spécifications internes en simulant le processus de développement adopté par un programmeur humain qui utiliserait une méthodologie top-down. Un but initial de programmation est systématiquement divisé en sous-buts plus élémentaires jusqu'à l'obtention du code du programme.

Voici un exemple de règle indiquant une décomposition de but:

```
sortir(pièce) ->    trouver-côté-pièce
                    tourner(gauche)
                    suivre-mur(droite)
                    aller-à(droite);
```

Le but de programmation est "sortir", et le contexte indique que l'endroit d'où le robot doit sortir est une "pièce". Une façon de réaliser ce but est de réaliser les quatre sous-buts suivants: (1) aller tout droit jusqu'à ce que l'on rencontre un mur, (2) tourner d'un quart de tour à gauche, (3) avancer tout en longeant le mur de droite, (4) sortir de la pièce lorsqu'une ouverture est détectée. Notons que ce n'est qu'une des stratégies possibles: l'analyseur contient aussi d'autres règles lui permettant d'implanter des stratégies alternatives que pourrait utiliser un étudiant(e).

Règles d'équivalence sémantique.

Pour traiter le problème des équivalences sémantiques, l'analyseur utilise un ensemble de règles d'équivalences constituées de connaissances propres au domaine enseigné soit la programmation robot. L'intérêt de ces règles réside dans leur généralité: elles permettent l'économie d'un nombre considérable de règles de production et simplifient la codification de ces règles.

Mal-règles.

Les mal-règles représentent deux types d'erreurs de programmation: des conceptions erronées typiques de novices et de simples oublis. Par exemple, les mal-règles permettent à l'analyseur de reconnaître qu'il manque une instruction, que deux instructions ont été inversées, qu'il manque une certaine structure de contrôle, qu'un `SI` a été utilisé alors qu'il fallait un `TANT_QUE`, etc...

5. PERSPECTIVES D'AVENIR.

Comme on peut le voir, l'utlisation de méthodes propres au domaine de l'intelligence artificielle permet de réaliser des logiciels éducatifs doués de pouvoirs d'analyse assez sophistiqués. Malheureusement, il reste plusieurs étapes à parcourir avant d'assister à une utilisation sur une grande échelle de ce type de logiciel.

Tout d'abord, il importe de souligner que peu de systèmes tutoriels intelligents sont sortis des laboratoires de recherche pour être véritablement testés dans le champ. Il reste donc à démontrer l'efficacité réelle de ces tuteurs. Malgré leurs pouvoirs d'analyse plus grands, ces logiciels n'ont pas encore démontré de façon convaincante leur utilité pédagogique. En effet, il est important de vérifier si l'utilisation d'un système tutoriel intelligent améliore l'apprentissage de façon significative. Pour ce qui est d'AlgorithmiK, la prochaine étape de la recherche prévue portera justement sur une évaluation formelle de son efficacité pédagogique.

Par ailleurs, même si ce nouveau type de logiciel s'avère très utile pour l'étudiant, il n'en demeure pas moins que le coût de développement n'est pas négligeable, et qu'une analyse coûts-bénéfices s'imposera avant d'entreprendre un développement massif de nouveaux logiciels. Ce domaine de recherche est encore jeune et il existe peu de méthodes standards de développement de systèmes intelligents. Chaque concepteur de logiciel doit donc plus ou moins réinventer la roue. Il est prévisible, toutefois, qu'à

moyen terme nous assisterons à l'apparition de systèmes-auteurs venant facilité le développement de nouveaux tuteurs.

Il n'en demeure pas moins que malgré ces bémols, le potentiel des systèmes tuteurs intelligents est énorme, et nous assisterons dans les prochaines années à l'arrivée de didacticiels de plus en plus sophistiqués. Il est probable que les techiniques liées à l'intelligence artificielle s'imposeront progressivement dans le monde de l'éducation un peu comme elles sont en train de le faire dans le domaine de l'entreprise, et ce pour les mêmes raisons soit le besoin de logiciels toujours de plus en plus puissants.

6. BIBLIOGRAPHIE

1. Adam, A., et Laurent, J.P. LAURA, A System to Debug Student Programs. *Artificial Intelligence* 15, 1980, pp. 75-122.

2. Anderson, J.R., et Skwarecki, E. The automated tutoring of introductory computer programming. *Communications of the ACM* 29, 9 (Sept. 1986), pp. 842-849.

3. Barr, A., et Feigenbaum, E.A (eds.).The Handbook of Artificial Intelligence, Vol II, Chap. IX: Applications-oriented AI Research: Education. William Kaufman, Inc., Los Altos, Calif.,1982, pp. 223-294.

4. Burton, R.R., et Brown, J.S. An investigation of computer coaching for informal learning activities. Dans *Intelligent Tutoring Systems*, D. Sleeman, et J.S. Brown, Eds. Academic Press Inc., New York, 1982, pp. 79-98.

5. Carbonell, J.R. AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems MMS-11*, 4 (Décembre 1970), 190-202.

6. Clancey, W.J. Tutoring rules for guiding a case method dialogue. Dans *Intelligent Tutoring Systems*, D. Sleeman, et J.S. Brown, Eds. Academic Press Inc., New York, 1982, pp. 201-226.

7. Cordier, M.O. Les systèmes experts. *La Recherche 15*, 151 (Janvier 1984), pp. 60-70.

8. Gagné, R.M., Wager, W., et Rojas, A. Planning and Authoring Computer-Assisted Instruction Lessons. *Educational Technology* , Sept. 1981, pp. 17-26.

9. Goldstein, I.P. The genetic graph: a representation for the evolution of procedural knowledge. Dans *Intelligent Tutoring Systems*, D. Sleeman, et J.S. Brown, Eds. Academic Press Inc., New York, 1982, pp. 51-78.

10. Johnson, W.L. *Intention-Based Diagnosis of Novice Programming Errors*. Morgan Kaufman Publishers Inc., Los Altos, 1986.

11. Lelouche, R. et Huot, D. Fondements et objectifs d'un système intelligent pour l'apprentissage des langues. Rapport de recherche DIUL-RR-8506 . Université Laval, Québec, 1985.

12. Pattis, R.E. *Karel the Robot. A Gentle Introduction to the Art of Programming*. John Wiley & Sons, New York, 1981.

13. Sleeman, D., et Brown, J.S. *Intelligent Tutoring Systems.*, Academic Press Inc., New York, 1982.

14. Wenger, Etienne. *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers Inc., Los Altos, 1987.

15. Wertz, H. Stereotyped program debugging: an aid for novice programmers. *International Journal of Man-Machine Studies 16*, 1982, pp. 379-392.