Visualisation de l'exécution des programmes pour l'enseignement de la programmation

Inggriani Liem

▶ To cite this version:

Inggriani Liem. Visualisation de l'exécution des programmes pour l'enseignement de la programmation. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1989. Français. NNT : . tel-00333142

HAL Id: tel-00333142
https://theses.hal.science/tel-00333142

Submitted on 22 Oct 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

TU 6498

THESE

présentée par

Inggriani LIEM

pour obtenir le titre de DOCTEUR
de L'UNIVERSITE JOSEPH FOURIER - GRENOBLE I

(arrêté ministériel du 5 Juillet 1984)
Spécialité : INFORMATIQUE

Visualisation de l'exécution des programmes pour l'enseignement de la programmation

Thèse soutenue le 22 septembre 1989

Composition du Jury :

Président :	Jean-Claude BOUSSARD
Rapporteurs :	Bernard FILLIATRE
	Laurent TRILLING
Examinateurs :	Jean-Pierre PEYRIN
	Pierre-Claude SCHOLL

Thèse préparée au sein du Laboratoire de Génie Informatique
IMAG - Grenoble

En confiant ce travail à la reprographie, mes remerciements vont tout d'abord

A Monsieur Jean-Pierre PEYRIN, Maître de Conférences à l'Université Joseph Fourier, responsable du projet ARCADE, qui a accepté de diriger cette thèse, et auprès de qui j'ai su trouver un trésor d'expérience, de réflexion ... et de patience attentive et chaleureuse. Ses deux adorables filles, Laure et Juliette ont été mon meilleur réconfort dans les innombrables moments d'incertitude.

A Monsieur Pierre-Claude SCHOLL, Professeur à l'Université Joseph Fourier, Directeur de l'U.F.R. d'Informatique et de Mathématiques Appliquées, qui m'a accueilli dès mon arrivée en France, pour ensuite suivre avec attention mon itinéraire universitaire français et enfin suggérer de nombreuses idées et propositions qui enrichissent et raffinent ma thèse. Sa participation au jury de ma thèse est un aboutissement pour lequel je lui exprime toute ma gratitude.

A Monsieur Jean-Claude BOUSSARD, Professeur à l'Université de Nice, qui porte le plus vif intérêt au projet ARCADE. Il me fait l'honneur d'être le Président de mon jury.

A Monsieur Bernard FILLIATRE, Professeur à l'Université des Sciences et Techniques du Languedoc qui a fait beaucoup pour le Département d'Informatique de l'Institut de Technologie de Bandung et qui a suivi avec intérêt mon travail. Je lui sais gré d'avoir accepté de rapporter sur mon travail et d'être présent à mon jury.

A Monsieur Laurent TRILLING, Professeur à l'Université Joseph Fourier de Grenoble qui a accepté d'être rapporteur de cette thèse, et m'a offert remarques et conseils me permettant de peaufiner cette thèse.

A Monsieur Bertrand BISSON, qui a de nombreuses fois trébuché sur ma prose, mais qui, grâce au soutien de sa fille Camille, a pu à chaque fois reprendre la lecture attentive de cette thèse. Toutes ses observations ont-elles permis à mon texte de respecter la syntaxe, la sémantique et l'esthétique de la langue française ?

Aux autres membres du groupe ARCADE : Jean-Michel CAGNAT, Viviane GUERAUD et Sylvie PAINVIN, avec lesquels j'ai partagé tant d'entretiens enrichissants.

A l'ensemble du personnel de la médiathèque et du Laboratoire de Génie Informatique pour leur indispensable travail et aux membres du service de reprographie pour le soin et l'efficacité apportés au tirage de cette thèse.

A Monsieur Tonny Ungerer et Monsieur Sumantri Slamet I. Santosa, du Centre Inter-Universitaire Informatique Indonésien, pour l'appui financier qu'ils m'ont permis d'obtenir pendant toute la durée de ce travail de thèse.

A Madame Elisabeth MANG du Comité d'Etudes sur la Formation d'Ingénieurs (CEFI) et à Monsieur Georges PIERRON du Centre d'Etudes Supérieures en Electricité, Electronique et Informatique (CESELEC), qui se sont occupés avec soin et bienveillance de la gestion de ma bourse d'étude.

A mes différents responsables de l'Institut de Technologie de Bandung, particulièrement Monsieur Harsono, responsable du Département d'Informatique, pour la confiance qu'ils m'ont accordée en me laissant quitter mon poste pour faire cette thèse.

A la mémoire de Monsieur Bernard VAUQUOIS, pour son dévouement et son aide au développement du Département d'Informatique de l'Institut de Technologie de Bandung, plus particulièrement pour son invitation à poursuivre mes études en France et pour son accueil...

à Camille,

à Juliette,

à Laure.

PLAN

Introduction

PREMIERE PARTIE : ETUDES PRELIMINAIRES

1. Motivations pédagogiques

1.1 Le savoir faire (faire) du programmeur

1.1.1 Observation du savoir faire

1.1.2 La contradiction entre le souci méthodologique et le besoin de concrétisation

1.2 Un moment pédagogique propre à l'étude de l'exécution des programmes

1.2.1 Difficultés face aux phénomènes dynamiques

1.2.2 Visualisation de l'exécution des programmes

2. Contexte de l'étude

2.1 L'enseignement support de l'étude

2.1.1 Notation algorithmique et schémas d'analyse

2.1.2 Machines abstraites

2.1.3 Un ensemble d'exemples types

2.1.4 Activités pratiques et leurs supports

2.2 Un environnement logiciel d'accueil des activités pratiques : le laboratoire ARCADE

2.2.1 Chronologie

2.2.2 Etat actuel du laboratoire ARCADE

3. Analyse d'expériences de visualisation

3.1 Méthodes et outils de visualisation

3.1.1 Image et texte

3.1.2 Supports statiques ou dynamiques

3.1.3 Visualisation et programmatioh

3.1.4 Visualisation et manipulation

3.1.5 Visualisation de programmes

3.2 Panorama de logiciels de visualisation

3.2.1 Assistance à la production de programmes

3.2.2 Animation de programmes

3.2.3 Un système d'animation : BALSA

3.3 Synthèse

DEUXIEME PARTIE : COMPTE RENDU DE REALISATION

4. Conception et description des logiciels réalisés

4.1 Caractéristiques communes

4.1.1 Fonctionnalités

4.1.2 Méthode de travail

4.2 Visualisation et tableaux

4.2.1 Tri interne

4.2.2 Recherche dans un tableau

4.2.3 Reconnaissance d'un motif

4.3 Visualisation et récursivité

4.3.1 Dessins récursifs

4.3.2 Jeu de Baguenaudier

4.3.3 Huit reines

4.3.4 Méthodes de visualisation

4.4 Visualisation et machines abstraites

4.5 Visualisation et graphes

4.5.1 Tri topologique

4.5.2 Graphes non-orientés

5. Premier bilan pédagogique

5.1 Expérimentations à l'Institut de Technologie de Bandung

5.1.1 Profil des étudiants

5.1.2 Déroulement de l'expérimentation

5.1.3 Bilan pédagogique

5.2 Discussion

5.2.1 Une seule fenêtre active

5.2.2 Visualisation d'exécution et analyse

5.2.3 Diversité des données

5.2.4 Plusieurs algorithmes pour un même énoncé

5.2.5 Nécessité de plusieurs vues

5.2.6 Exécution pas à pas

5.2.7 Temps d'exécution

5.2.8 Limites

5.2.9 Contrôle de l'interaction

5.2.10 Aspect esthétique

6. Aspects techniques

6.1 Annotation des programmes à visualiser

6.1.1 Evénements et boîte à outils

6.1.2 Boucle d'événements

6.1.3 Annotation dans le cas de vues multiples

6.1.4 Squelette

6.2 Fonctions de visualisation

6.2.1 Changement de vitesse

6.2.2 Exécution pas à pas

6.3 Visualisation des structures de données

6.3.1 Mémorisation ou calcul

6.3.2 Association entre l'image écran et le programme annoté

6.3.3 Rafraîchissement de la fenêtre

6.4 Autres problèmes

6.4.1 Erreur de système...

6.4.2 Difficulté du test

6.4.3 Convention standard de l'interface utilisateur

6.4.4 Retouche des dessins

6.4.5 Lightspeed Pascal comme outil de développement

6.5 Résumé

Conclusion

Références bibliographiques

Annexes

Annexe A. Le laboratoire ARCADE

Annexe B. Synopsis des maquettes réalisées

Introduction

Enseignante au département d'informatique à l'Institut de Technologie de Bandung (ITB), en Indonésie, j'ai choisi l'enseignement de la programmation comme thème de recherche. En effet, le coût, l'adaptabilité, et la fiabilité des programmes posent de tels problèmes dans le monde professionnel, qu'il est encore très important de réfléchir à la formation des programmeurs et de proposer des démarches pédagogiques plus performantes.

L'objectif évident de l'enseignement de la programmation est de former de bons programmeurs. S'il est déjà difficile de donner une définition de ce qu'est un bon programmeur [You 75], le produire est encore plus difficile : "Je ne sais toujours pas enseigner la programmation" reste la phrase d'un expert dans ce domaine [Pai 88].

La spécification et l'analyse sont essentielles dans l'enseignement de la programmation. A côté des travaux pratiques classiques, où l'étudiant doit produire un programme pour un énoncé donné, il est nécessaire de diversifier les formes d'activités pratiques proposées à l'étudiant.

Faisons une analogie avec un apprenti-chanteur dont l'objectif est de savoir bien chanter. Il est évident que, pour arriver à ce savoir faire, il ne lui suffit pas de chanter. Il doit aussi faire des exercices de respiration, connaître le solfège, apprendre la théorie de la musique, écouter de la musique, étudier les chants des autres, etc. S'il débute, il ne doit surtout pas chanter seul. Sans s'en rendre compte, il peut prendre de mauvaises habitudes qu'il lui sera difficile de corriger ensuite.

De la même façon, un apprenti-programmeur ne doit pas seulement écrire des programmes. Il doit s'exercer à plusieurs activités diversifiées pouvant

enrichir ses capacités : lire, compléter ou modifier un programme, utiliser une bibliothèque de composants, observer le fonctionnement d'un programme, étudier le mécanisme de la preuve formelle, travailler en groupe, etc.

Dans ce contexte, une question importante est la contradiction entre d'une part l'acquisition de méthodes de spécification et d'analyse des problèmes et d'autre part la compréhension de l'exécution des programmes obtenus.

Lors de son apprentissage, l'étudiant a une tendance naturelle, due à un besoin certain de concrétisation, à aborder la construction d'un programme en référence forte à une prévision de l'exécution du programme visé. Cette tendance s'oppose à la volonté des enseignants d'axer leur enseignement sur l'analyse. Certains auteurs rejettent le problème, comme Dijkstra :

".. I was recently exposed to a demonstration of what was pretended to be educational software for an introductory programming course. With its "visualisations" on the screen it was such an obvious case of curriculum infantilization that its author should be cited for "contempt of student body", but this was only a minor offence compared with what the visualisations were used for : they were used to display all sorts of features of computations involving under control of the student's program ! The system highlighted precisely what the student has to learn to ignore, it reinforced precisely what the student has to unlearn. Since breaking out bad habits, rather than acquiring a new ones, is the thoughest part of learning, we must expect from that system permanent mental damage for most students exposed to it.

Needless to say, that system completely hid the fact that, all by itself, a program is no more than half a conjecture. The other half of the conjecture is the functional specification the program is supposed to satisfy. The programmer's task is to present such complete conjectures as proven theorem. ....." [Dij 89].

Ceci s'oppose au point de vue préconisant le développement de logiciels pour la visualisation. L'expérience la plus significative est sans doute celle de l'université Brown [Van 84], [Bro 87] que nous présentons au §3.2.3.

Pour notre part, nous pensons important d'étudier la place de l'interprétation opératoire des programmes dans l'enseignement de la programmation.

Notre hypothèse est de satisfaire les besoins d'expérimentation concrète de l'étudiant en dissociant les moments pédagogiques concernant cette expérimentation et l'enseignement méthodologique. Il faut ainsi offrir des logiciels adéquats. Nous centrons notre travail sur le thème de la visualisation de l'exécution des programmes.

Le projet ARCADE, dans lequel s'inscrit cette thèse, a plus généralement pour objet d'étudier un environnement logiciel mis à disposition des enseignants et des étudiants pour expérimenter concepts et techniques de programmation au travers d'activités diversifiées. Manipulation, observation et jeu y sont privilégiés.

Dans le **chapitre 1** de cette thèse, nous présentons nos réflexions sur **l'interprétation opératoire** des programmes dans un enseignement de la programmation. Nous étudions l'intérêt potentiel d'une visualisation de l'exécution des programmes dans un tel enseignement.

Le **chapitre 2** décrit **le contexte** de notre travail : le type d'enseignement dans lequel s'insère l'étude et le projet **ARCADE**.

L'étude bibliographique, résumée dans le **chapitre 3**, essaye de cerner la question de la visualisation de l'exécution de programmes. Cette étude m'a permis de préciser les voies possibles à approfondir, et de déterminer mes choix de réalisation de logiciels.

Le **chapitre 4** rend compte **des caractéristiques communes** des logiciels réalisés : les sujets traités, la méthode de travail, les principes de conception et le fonctionnement.

Le **chapitre 5** décrit **deux expérimentations** effectuées, dans un contexte normal d'enseignement, à l'Institut de Technologie de Bandung. Ce travail "sur le terrain" m'a permis d'établir un premier bilan pédagogique.

Le laboratoire ARCADE est implémenté sur Macintosh. Mes logiciels ont été réalisés en Pascal. Dans le **chapitre 6**, je relate les difficultés rencontrées dans un tel environnement de programmation et je propose des solutions. Plus généralement, je présente **l'aspect technique** de mes réalisations.

Première partie

ETUDES PRELIMINAIRES

1. Motivations pédagogiques
2. Contexte de l'étude
3. Analyse d'expériences de visualisation

1. Motivations pédagogiques

Ce chapitre est centré sur l'interprétation opératoire des programmes dans l'enseignement de la programmation. Il s'agit d'étudier en quoi la compréhension de l'exécution des programmes interfère avec l'acquisition des méthodes d'analyse des problèmes. L'objectif est de compléter les traditionnels travaux pratiques dont l'objectif essentiel est l'acquisition d'une pratique d'un environnement de programmation.

Nous montrons qu'une certaine idée sur un processus séquentiel possible de résolution d'un problème peut être un point de départ pour concevoir un programme. Mais cette démarche peut ne produire que certaines solutions qui sont souvent difficile à prouver et rarement les plus performantes.

Nous pensons que l'enseignement ne peut pas ignorer la notion d'exécution. Nous cherchons donc un moyen précis qui permette d'intégrer au mieux cette notion dans la pédagogie.

1.1. Le savoir faire (faire) du programmeur

L'art du programmeur consiste à **faire faire**. Qui plus est, le programmeur doit faire faire un travail à une machine qui ne peut pas comprendre le problème qu'elle résout. Par conséquent, cette machine ne peut en aucune façon lever la moindre ambiguïté ou détecter la moindre erreur algorithmique.

1.1.1. Observation du savoir faire

On peut opposer deux manières pour analyser un problème et produire un algorithme. La première est descendante et la deuxième est ascendante.

L'approche descendante, déductive, conduit à définir le faire faire à partir d'une réduction descendante du problème, par exemple une analyse récurrente. Le programmeur suppose (hypothèse de récurrence) savoir faire faire un problème réduit. Il établit la solution dans des cas triviaux (base de la récurrence). Il doit alors définir la solution du problème complet en termes de la solution du problème réduit et des solutions des cas triviaux. Cette approche montre qu'un programmeur peut faire faire à une machine quelque chose qu'il ne sait pas nécessairement faire lui-même. Un exemple classique est celui des tours de Hanoï. La solution récursive est simple ; faire le jeu, à la main, sans se tromper, l'est beaucoup moins.

L'approche ascendante, inductive, consiste à analyser le faire faire à partir du **faire**. Le programmeur étudie le problème dans différents cas bien choisis : il le résout dans chaque cas, et établit, pour chacun d'eux, le **compte rendu** de la succession d'opérations qui conduit à la solution. L'observation des différents comptes rendus permet de les généraliser sous une forme unique qui est la base d'un algorithme.

Cette démarche est couramment utilisée lors de l'introduction à la programmation pour sensibiliser à la notion d'algorithme. L'exemple de Dijkstra sur le thème de "l'épluchage de pommes de terre" [Dij 71] a été suivi par de nombreux enseignants pour introduire les éléments de base algorithmique et notamment les trois formes de composition de programmes : séquentielle, conditionnelle et itérative.

Lucas, Peyrin et Scholl ([LuS 75], [Sch 79], [LPS 83], [ScP 88]) proposent d'introduire l'itération en employant une machine abstraite, la machine-caractères, qui modélise l'accès séquentiel à un texte. Les actions élémentaires, figurées par des boutons de modification de l'état de la machine, sont désignées par de simples noms : ***DEM****arrer {accès au premier caractère du texte},*

AVancer {accès au caractère suivant} ... correspondent à des procédures sans paramètres.

A partir de problèmes simples, on introduit l'itération sur la base de la notion de compte rendu. L'étudiant dispose, outre la machine caractères, d'autres machines permettant d'écrire des traitements, par exemple, une machine-compteur comportant deux boutons : *Mettre-A-Zéro {mise à zéro du compteur}* et *INCrémenter {incrémentation de 1 du compteur}*.

Ainsi pour étudier le calcul de la longueur d'un texte, on peut procéder comme suit :

Dans un premier temps, on établit différents comptes rendus dans des cas significatifs, par exemple, pour un texte de 2 caractères, puis 5 caractères, puis 0 caractères :

{2} *MAZ ; DEM ; INC ; AV ; INC ; AV*

{5} *MAZ ; DEM ; INC ; AV ; INC ; AV ; INC ; AV ; INC ; AV ; INC ; AV*

{0} *MAZ ; DEM*

Figure 1.1 Exemples de comptes rendus.

On peut remarquer que dans ces comptes rendus, les tests n'apparaissent pas, car il s'agit d'une observation a posteriori pour un cas précis.

Dans un deuxième temps, on fait rechercher à l'étudiant une forme unique pour tous les comptes rendus, par exemple :

MAZ ; DEM ; ***Long fois*** *(INC ; AV)*

{ où "Long" désigne la longueur cherchée }

ou

MAZ ; DEM ; (INC ; AV)$^{\textbf{Long}}$

...

Figure 1.2 Unification des comptes rendus.

Enfin, on recherche une expression algorithmique, par exemple :

MAZ ; DEM
tantque *CC ≠ marque* ***faire***
{où "CC" désigne le caractère courant }
{et "marque" un caractère spécial de terminaison du texte}
INC ; AV
ftantque

Figure 1.3 Expression algorithmique de la longueur d'un texte.

Un exemple non trivial d'approche ascendante : le Baguenaudier

Lors de la synthèse du Colloque Francophone sur la Didactique de l'Informatique de Septembre 1988 [Did 88], le professeur Jacques Arsac a présenté un exemple provocateur pour remettre en cause l'hégémonie excessive de la pensée et de l'expression algorithmique récursives. Cet exemple n'a pas été

publié mais nous souhaitons le présenter pour renforcer notre propos. Il illustre assez bien les deux manières décrites précédemment.

Il s'agit de construire un programme illustrant la solution du jeu de Baguenaudier (Figure 1. 4, 1.5, 1.6) [Coi 74], [Ger 77].

Un baguenaudier est un "casse-tête" formé de deux parties : d'une part, des anneaux imbriqués et reliés à une réglette par des tiges coulissantes et d'autre part, une barre mobile appelée navette. Au début du jeu, la navette est enchevêtrée dans les anneaux ; le jeu consiste à séparer les deux parties.

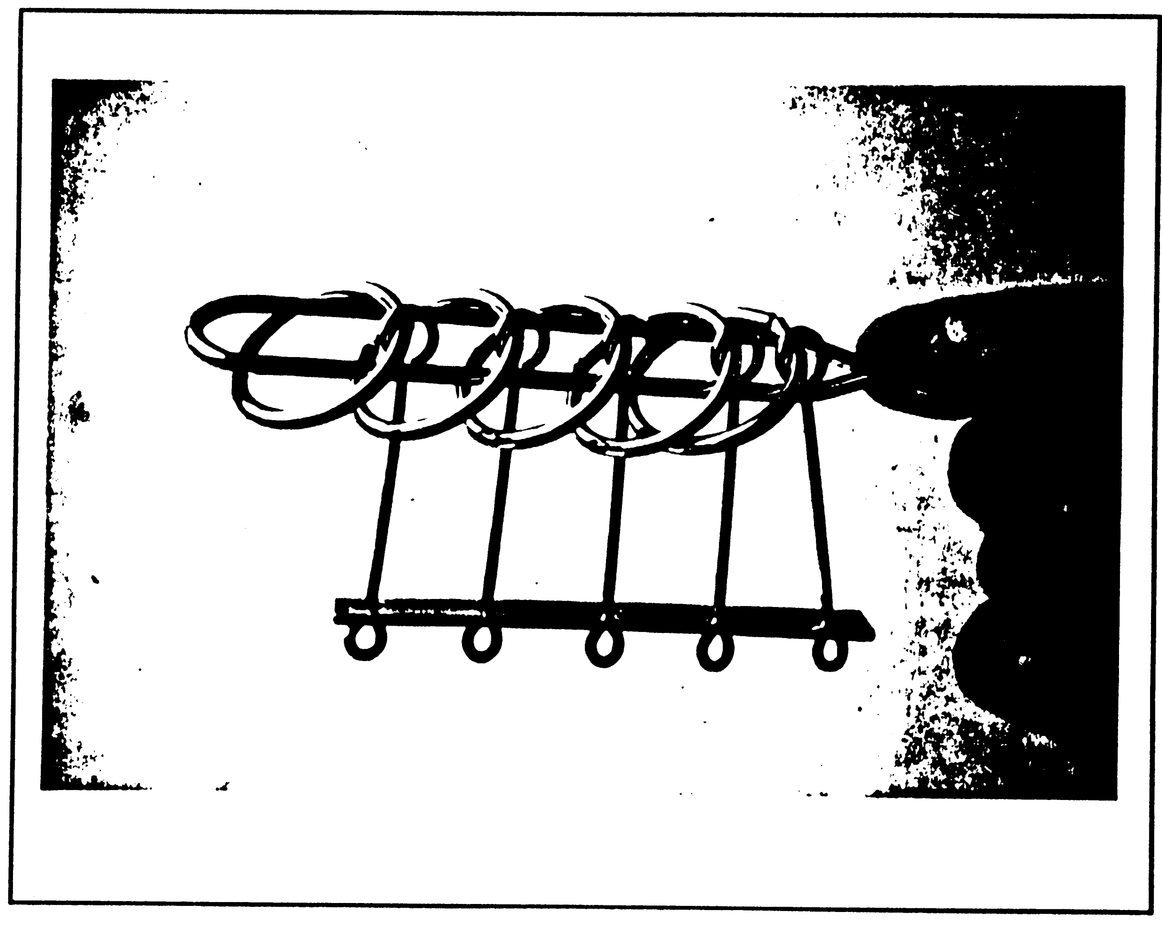


Figure 1.4 Etat initial du Baguenaudier.

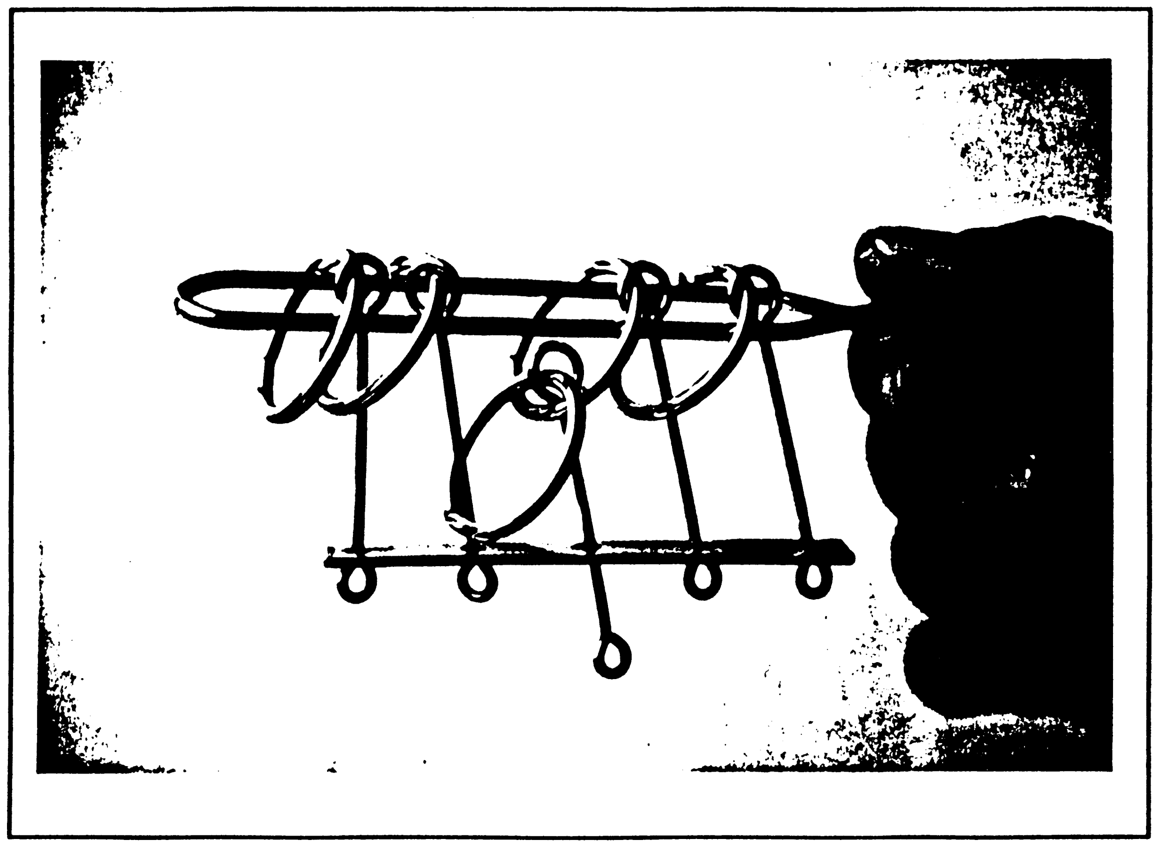


Figure 1.5 Etat intermédiaire du Baguenaudier.

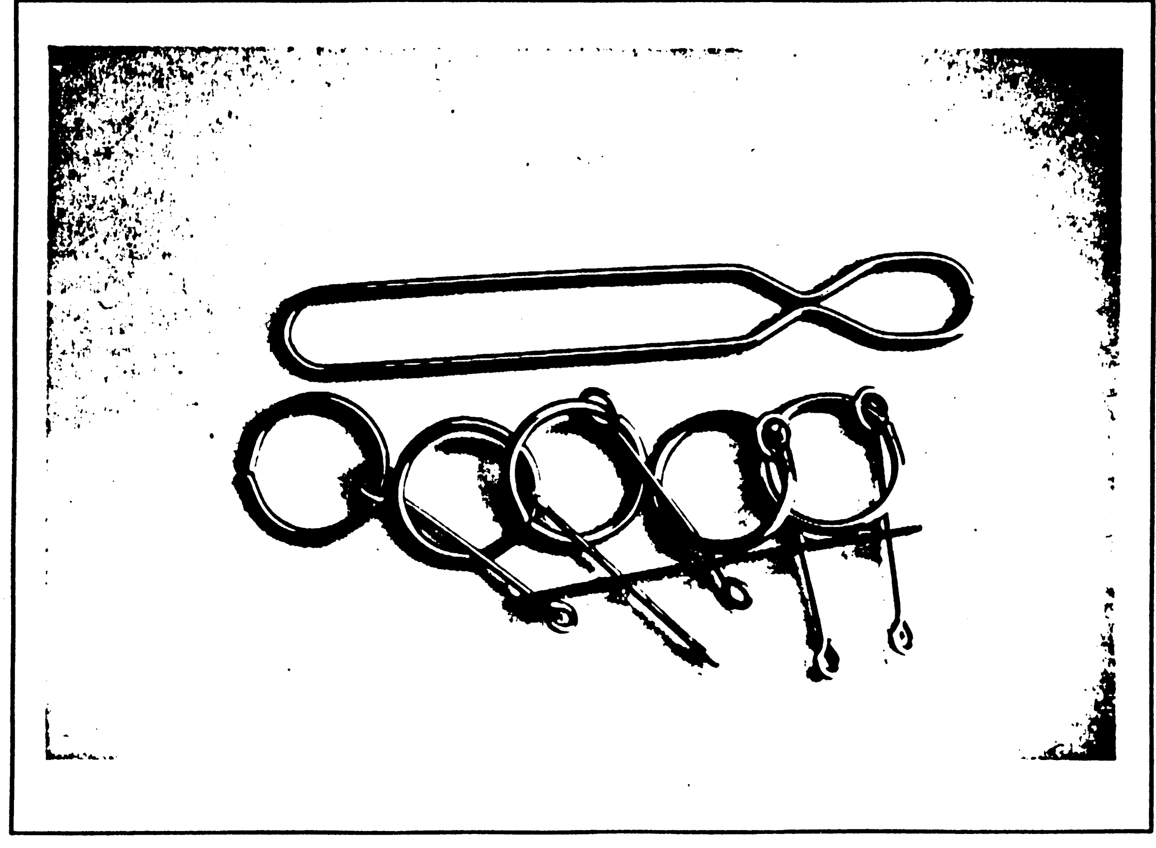


Figure 1.6 Etat final du Baguenaudier.

Chaque anneau a deux états : accroché sur la navette (état 1) ou bien décroché (état 0). L'état initial d'un jeu de cinq anneaux peut être représenté par 11111 et l'état final par 00000. On découvre, au bout d'un certain temps de tâtonnement, que l'on peut à chaque instant modifier seulement l'état de deux anneaux : le plus à gauche et celui qui suit le premier accroché. Ainsi, après l'état 00110 on peut passer à l'état 10110 ou bien à l'état 00100.

Le programme doit imprimer la succession des déplacements d'anneaux à effectuer.

Un premier algorithme procède d'une analyse récurrente et comporte deux procédures récursives (Figure 1.7).

```
Déplacer : l'action (i : un entier ≥ 1) {change l'état de l'anneau i}
   écrire ("Bougez l'anneau n° ", i)

Décrocher : l'action (n : un entier ≥ 1){défait un jeu de n anneaux}
   selon n
      n = 1 :  Déplacer(1)
      n = 2 :  Déplacer(2)  ;  Déplacer(1)
      n ≥ 3 :  Décrocher(n-2)
               Déplacer(n)
               Accrocher(n-2)
               Décrocher(n-1)

Accrocher : l'action (n : un entier ≥ 1) {refait un jeu de n anneaux}
   selon n
      n = 1 :  Déplacer(1)
      n = 2 :  Déplacer(1)  ;  Déplacer(2)
      n ≥ 3 :  Accrocher(n-1)
               Décrocher(n-2)
               Déplacer(n)
               Accrocher(n-2)
```

Ainsi le jeu de X anneaux est résolu par l'appel :

Décrocher (X)

Figure 1.7 Solution récursive du problème de Baguenaudier.

On peut observer la relative complexité de cette solution. En effet, elle nécessite deux procédures croisées, chacune d'elles comportant, en outre, deux appels récursifs utilisant des paramètres différents n-1 et n-2.

Une deuxième analyse s'appuie sur une démarche ascendante et conduit directement à un algorithme itératif. Présentons cette approche avec plus de détails.

Le point de départ est une observation du **savoir faire**. En effet, à toute étape, le jeu ne permet que deux déplacements possibles, soit le premier anneau, soit celui qui suit le premier anneau en place. L'un de ces deux déplacements consistant à revenir à l'étape précédente, il n'en reste qu'un seul intéressant, en réalité. L'idée est donc de décrire cette succession d'étapes à l'aide d'une expression itérative.

L'algorithme dépend de la parité initiale du nombre d'anneaux, puisqu'il y a, au début également, deux coups possibles. Pour simplifier, nous ne présentons que l'algorithme dans le cas d'un nombre impair d'anneaux.

```
Déplacer(1)
tantque non terminé :
   {recherche du premier anneau accroché}
   k ← 1 ; tantque EstDécroché(k) : k ← k+1
   {déplacement de l'anneau suivant}
   Déplacer(k+1)
   {déplacement du premier anneau}
   Déplacer(1)
```

Figure 1.8 Solution itérative du problème de Baguenaudier.

Jacques Arsac concluait que l'on ne doit pas refuser une analyse cohérente conduisant à cette solution itérative et que l'on devrait relativiser l'impérialisme de la récursivité.

Cette solution itérative est sans doute très simple et très élégante (elle permet, en particulier, de faire très aisément le jeu, manuellement), mais remarquons qu'elle est moins évidente à prouver que la solution récursive, et que son principe n'est pas généralisable. En effet, il n'est pas aussi simple de construire une solution itérative au problème des tours de Hanoï, ou au tracé d'une courbe de Sierpinski, par exemple.

Dans l'exemple des tours des Hanoï, on trouve dans [Bac 86] une démarche analogue pour construire une solution itérative du problème, à partir d'une observation de savoir faire.

Ce problème du Baguenaudier illustre bien qu'il est concevable de construire un algorithme basé sur l'observation d'une "exécution" du problème et qu'inversement, l'exécution d'un algorithme ne se comprend pas aisément à partir d'une expression récursive. En effet, cette exécution correspond à une interprétation itérative de la récursivité qui n'est pas toujours triviale.

Si l'on recherche un algorithme de tri à partir d'une pratique manuelle, on trouvera probablement le tri par minimums successifs ou le tri par insertions. Il est assez peu probable de découvrir ainsi le tri par segmentations ou le tri par tas. Inversement, on aura beaucoup de mal à appliquer, manuellement, un algorithme de tri récursif.

Faire un algorithme déduit de "comptes rendus" n'est possible que si le nombre de données manipulées est petit. La puissance d'un algorithme réside dans le fait qu'un même texte permet de provoquer un très grand nombre possible d'exécutions différentes. Le texte de l'algorithme représente donc une abstraction de toutes ces exécutions qui se distinguent par les jeux de données.

Et encore faut-il être capable d'exprimer cet algorithme : combien de nos étudiants sont-ils capables de rédiger correctement l'algorithme usuel de la division de deux nombres entiers, par exemple, alors qu'ils effectuent couramment des divisions depuis une quinzaine d'années ?

Pirolli constate [Pir 86] que l'être humain a des difficultés pour exécuter une formulation récursive. Par exemple, il est difficile de comprendre cette phrase : *"The fact that the sheperd said that the farmer had given the book to the child to the police was to be expected"*. Il constate qu'un être humain ne peut pas suspendre un processus pour effectuer un calcul avant de retourner au processus suspendu, surtout si le calcul est récursif.

Wiedenbeck [Wie 88] tente d'expérimenter la relation itératif-récursif dans l'enseignement de la programmation. Il constate que la récursivité est fondamentale en informatique (concept mathématique, méthode d'analyse, technique d'expression algorithmique, technique de programmation).

Une expérimentation a été faite sur 200 étudiants. Ses conclusions sont contradictoires quant à l'utilité de la connaissance de l'itération pour l'apprentissage de la récursivité. Mais il semble enseigner la récursivité à l'aide d'exemples du style "factorielle" ou "somme de n entiers", qui s'écrivent si bien itérativement, mais rien n'est dit sur ce qui est enseigné au titre de l'itération (y parle-t-on de récurrence et d'invariant ?).

1.1.2. La contradiction entre le souci méthodologique et le besoin de concrétisation

La fameuse phrase [Wir 73] : *"Testing a program can proof the presence of errors, but never their absence"* montrait la voie de la méthodologie de la programmation. L'effort devait être mis sur les spécifications et sur l'analyse

par décompositions pour oublier la tyrannie du processus d'exécution des programmes. La synthèse des programmes reste un sujet de recherche et est peu réaliste comme objectif à court terme [RiW 88]. Par contre, l'écriture d'un programme à partir d'une spécification, à l'aide de règles de transformation et de schémas est un principe établi. L'analyse est faite par le programmeur, et l'exécution par la machine.

Il est parfois difficile d'inculquer à nos étudiants les "bons principes". Comment les pousser à concentrer leur effort sur l'analyse au lieu de passer beaucoup d'heures à la mise au point empirique de leurs programmes, à l'aide d'un nombre inconsidéré d'exécutions. Nous avons tous nos exercices fétiches. Nous voulons citer celui d'un professeur d'un lycée grenoblois [Dou 90] : les programmes qu'il fait réaliser consistent à faire faire des dessins sur une table traçante : l'exécution est tellement longue par rapport au temps de conception du programme que les élèves préfèrent réfléchir deux fois plutôt que de risquer un échec.

Pour faire comprendre cette idée d'analyse des problèmes, les enseignants tentent d'éviter, autant que possible, le recours à une explication opératoire, en cherchant une formulation plus abstraite. D'ailleurs, le plus souvent, ils s'intéressent d'abord aux dossiers d'analyse et beaucoup moins aux programmes.

Mais les étudiants ont l'habitude, discutable, mais compréhensible de n'admettre un algorithme qu'après l'avoir "fait tourner" à la main. Par exemple, la plupart d'entre eux exécutent manuellement, le plus discrètement possible, un algorithme récursif pendant que l'enseignant s'évertue à prouver sa correction par récurrence. Ce n'est pas tant le manque de confiance dans le raisonnement par récurrence, qu'un besoin profond de "réaliser" le fonctionnement de l'algorithme.

Ainsi la principale volonté des enseignants va à l'encontre d'un besoin ressenti par les étudiants. Et ce besoin ne peut qu'être déclaré normal puisque le but d'un programme est bien d'être exécuté.

1.2. Un moment pédagogique propre à l'étude de l'exécution des programmes

Si l'enseignant veut être complet en présentant un algorithme, il doit donc à la fois en étudier la conception et en expliquer l'exécution. Mais ainsi, dans un même moment pédagogique, le cours, avec une même ressource pédagogique, l'enseignant, sont présentés deux éléments complémentaires mais conflictuels.

L'idée que nous souhaitons développer est de placer l'étude de l'exécution des programmes dans un moment pédagogique propre.

- Le cours et les travaux dirigés sont les moments adéquats pour travailler sur les méthodes d'analyse ; c'est là que sont le mieux exploitées les compétences des enseignants.
- Les travaux pratiques usuels restent, bien entendu, le moment privilégié pour que les étudiants expérimentent la construction des programmes. Dans le cadre de ces travaux pratiques, les étudiants utilisent des outils de mise au point, dont des traces, qui sont une certaine manière de travailler sur l'exécution. Mais ceci n'est pas tout à fait l'idée présentée ci-dessus, puisque la mise au point fait partie d'un processus de construction de programmes dont l'analyse initiale faisait partie.
- Il reste donc à développer un nouveau moment pédagogique dont l'exécution des programmes serait l'objet principal.

1.2.1. Difficultés face aux phénomènes dynamiques

L'enseignant, avec sa craie et son tableau noir, est très démuni pour présenter des phénomènes dynamiques. Il doit jouer avec des représentations des données et organiser des réécritures de valeurs qui rendent les choses relativement complexes. En fait, il cherche à établir une certaine représentation visuelle des données et des instructions pour gérer le dynamisme en utilisant son propre temps. Par exemple, dessiner un arbre nécessite un calcul des positions des noeuds à chaque instant [WeS 79]. Imaginons l'explication de l'insertion d'un nouvel élément (noir) comme fils gauche d'un certain noeud (gris) :

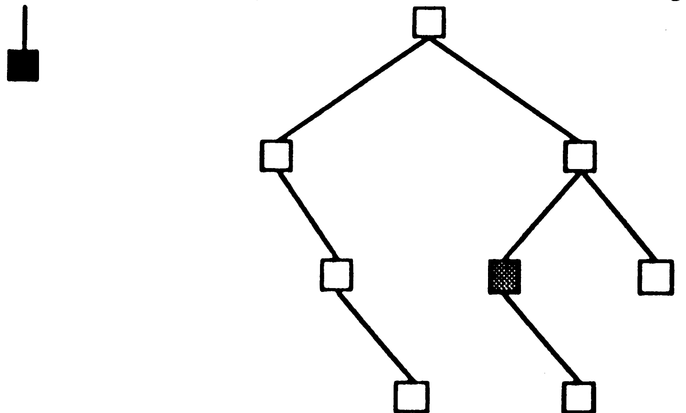


Figure 1.9 Insertion d'un dans un arbre.

Soit l'enseignant insère le noeud où il peut, et la figure risque, au bout d'un certain nombre d'insertions et de suppressions, de ne plus représenter un arbre :

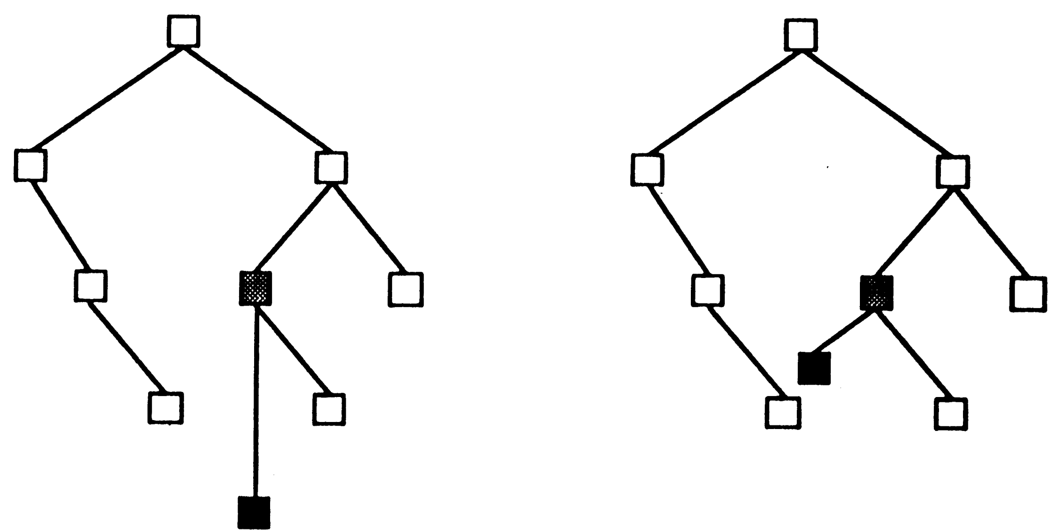


Figure 1.10 Etats possibles de l'arbre après insertion.

Soit il redessine l'arbre au fur et à mesure, mais c'est très vite fastidieux.

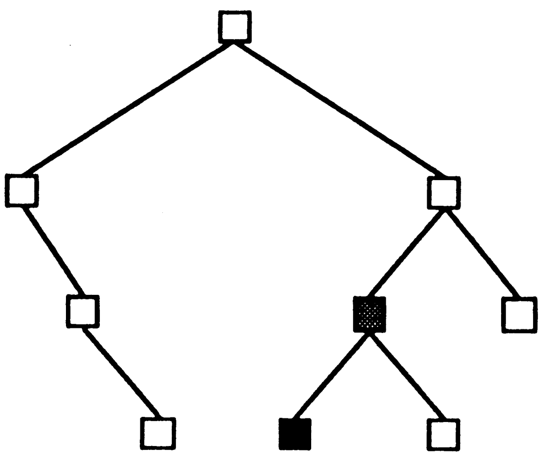


Figure 1.11 Etat de l'arbre après insertion.

Dans le discours d'un enseignant, certaines notions liées au temps, au mouvement et à la quantité deviennent abstraites.

Par exemple :

- l'évaluation du nombre de comparaisons dans différents algorithmes de recherche, séquentielle, dichotomique...
- le choix de données particulières qui rendent peu performant un tri rapide.
- l'inefficacité d'un tri par bulles.
- l'évaluation du temps d'exécution des algorithmes en log(n), n, n^2 ou n!.
- ...

1.3. Visualisation de l'exécution des programmes

Le rêve serait d'assister au déroulement de l'exécution d'un programme comme à un spectacle. La notion de temps appartiendrait au spectacle et la quantité et les mouvements d'informations seraient rendus visuels. Le cinéma pourrait être utilisé avec tout l'enrichissement pédagogique du trucage ; mais seuls quelques programmes particuliers pourraient ainsi être montrés étant donné le temps de fabrication d'un film.

L'ordinateur paraît idéal pour assurer une telle fonction puisque le programme concerné pourrait être moteur de la visualisation. Le programme se présenterait lui-même, moyennant la définition et la réalisation de procédures appropriées.

Selon le Petit Robert [Rob 85], visualiser est rendre visible un phénomène qui ne l'est pas. La visualisation de l'exécution des programmes dépend du mode de programmation utilisé.

En programmation impérative, il s'agit de rendre visible la succession d'états intermédiaires significatifs entre l'état initial et l'état final. La conception d'une visualisation s'appuie alors sur :

- le repérage des états intermédiaires significatifs.
- le choix d'une illustration cohérente pour l'ensemble de ces états.
- la définition de jeux de données caractéristiques.

Ces trois parties correspondent au modèle conceptuel d'un système d'animation d'algorithme proposé dans le système BALSA [Bro 87] : *"algorithm annotation (interesting event), view implementation, input generator"*.

En programmation fonctionnelle, il s'agit de rendre visible l'application des fonctions et les valeurs intermédiaires obtenues. En programmation logique, il s'agit de rendre visible l'unification, la déduction et la succession des règles choisies. En programmation orientée-objet, il s'agit de rendre visible la hiérarchie, les propriétés des objets, la succession d'héritages des valeurs et la communication entre objets.

La visualisation de l'exécution des programmes impératifs et orientés-objets correspond plus directement au fonctionnement de l'ordinateur, alors que pour les programmes fonctionnels et logiques, elle s'appuie sur un modèle de calcul intermédiaire.

L'enseignement élémentaire est traditionnellement celui de la programmation impérative, pour des raisons autant historiques que professionnelles. L'énorme majorité des applications informatiques est réalisée dans des langages impératifs. Aucun étudiant ne peut à l'heure actuelle échapper à cette forme de programmation qui, malgré les réticences que cela provoque, reste dominatrice, ne serait-ce que pour son efficacité.

C'est pourquoi nous avons placé dans un premier temps notre travail dans un contexte de programmation impérative.

Face à un logiciel de visualisation, un étudiant peut ne rester que quelques minutes sans y trouver un intérêt quelconque. Un autre étudiant peut y consacrer plusieurs heures car il observe attentivement les phénomènes, il essaye plusieurs jeux de données, il analyse chaque cas, il compare plusieurs solutions proposées, etc.

Il est difficile d'évaluer l'efficacité de tels outils, surtout lorsqu'ils sont proposés en libre service, car leur utilisation dépend de la volonté de travail et de la motivation personnelle de chaque étudiant.

Le rôle de l'enseignant doit rester important, surtout lors des premières observations, principalement pour proposer une méthode de travail et pour aider à découvrir les différents points de vue. Son rôle est analogue à celui d'un guide dans une visite touristique.

De même qu'écouter quelqu'un chanter est insuffisant pour apprendre le chant, regarder ou observer l'exécution ne donne pas un savoir faire.

Cette activité d'observation est nécessaire, mais ne peut donc qu'être complémentaire à beaucoup d'autres.

Ainsi l'observation de l'exécution d'un programme n'est qu'un premier niveau dans l'apprentissage. Elle ne montre que **comment** un programme fonctionne. Par exemple, la visualisation du passage de paramètres peut montrer l'effet du passage par référence mais ne montre pas quand et comment l'utiliser [Van 84].

2. Contexte de l'étude

Nous présentons d'une part le type d'enseignement que notre travail sur la visualisation doit enrichir, et d'autre part l'environnement logiciel, auquel notre travail a contribué, qui doit permettre d'organiser les activités pratiques des étudiants.

2.1. L'enseignement support de l'étude

Il s'agit d'un enseignement de la programmation qui a été élaboré dans le cadre de diverses formations à l'informatique de deuxième cycle assurées à l'Université Joseph Fourier de Grenoble. Plusieurs ouvrages jalonnent la réflexion pédagogique menée depuis plusieurs années : [LuS 75], [Sch 79], [LPS 83], [Sch 84], [ScP 88].

2.1.1. Notation algorithmique et schémas d'analyse

La notation algorithmique n'est pas uniquement une forme d'écriture d'algorithmes mais aussi le reflet d'une démarche de conception. Elle fixe la manière de décrire les informations, les actions mises en œuvre et l'organisation de ces actions dans le temps.

Par exemple, la notation "**selon**" [ScP 88] est éloignée des formes de composition conditionnelle des langages courants. C'est en fait un schéma reflétant une approche systématique de l'analyse par cas.

```
classer_a,b,c : l'action
{a, b, c ont des valeurs distinctes 2 à 2}
selon a,b,c {selon les valeurs de a, b, c : domaine de l'analyse}
   a < b < c : p ← a ; s ← b ; t ← c ;
   a < c < b : p ← a ; s ← c ; t ← b ;
   b < a < c : p ← b ; s ← a ; t ← c ;
   b < c < a : p ← b ; s ← c ; t ← a ;
   c < a < b : p ← c ; s ← a ; t ← b ;
   c < b < a : p ← c ; s ← b ; t ← a ;
{le triplet p, s, t est une permutation du triplet a, b, c et p < s < t }
```

Figure 2.1 Exemple d'utilisation de la notation **selon**.

La structure "**case**" en Pascal s'approche de cette notation mais ne permet pas d'exprimer une condition sous forme d'expression logique. La notation **selon** induit par sa disposition une lisibilité. La même action (Figure 2.1) écrite à l'aide des expressions "**si-alors-sinon**" emboîtées (Figure 2.2) est plus difficile à lire car elle ne reflète plus immédiatement l'analyse en 6 cas.

```
classer_a,b,c : l'action
{a, b, c ont des valeurs distinctes 2 à 2}
si a < b < c alors p ← a ; s ← b ; t ← c
sinon  si a < c < b alors p ← a ; s ← c ; t ← b
       sinon  si b < a < c alors p ← b ; s ← a ; t ← c
              sinon  si b < c < a  alors
                          p ← b ; s ← c ; t ← a
                     sinon  si c < a < b alors
                                 p ← c ; s ← a ; t ← b
                            sinon p ← c ; s ← b ; t ← a
{le triplet p, s, t est une permutation du triplet a, b, c et p < s < t }
```

Figure 2.2 L'action de la figure 2.1, écrite avec la notation **si-alors-sinon**.

La notation **selon** favorise une analyse par cas systématique par rapport à une analyse au coup par coup. Elle rend compte d'un travail systématique : repérage d'un ensemble de cas cohérents, énumération de ces cas, description des actions correspondant à chacun d'eux.

La notion de **schéma** est une partie intégrante de la notation algorithmique. Un schéma est un outil d'**analyse**, outil de **décomposition** ou de **réduction** d'un problème à des composants plus élémentaires. L'analyse d'un problème s'appuie ainsi sur un ensemble de schémas correspondant à diverses méthodes de décomposition.

Par exemple, on trouve dans [ScP 88] différents schémas de traitement de séquences organisés autour de deux classes de problèmes : parcours et recherche. Les formes itératives classiques "**tantque**" ou "**répéter**" sont liées à ces schémas : ainsi elles n'interviennent pas directement dans l'analyse d'un problème.

Par exemple, les étapes d'écriture d'un algorithme de traitement de séquence sont les suivantes :

- identification d'une classe de problème (parcours ou recherche),
- choix d'un schéma par rapport à la représentation de la séquence,
- application du schéma, en identifiant chacune des actions le composant : accès séquentiel à la séquence, traitement des éléments de la séquence.

Ainsi la notation algorithmique permet de guider l'étudiant, plus ou moins fortement, dans son approche des problèmes. Les expériences vécues nous montrent que les étudiants, notamment ceux qui ont déjà appris quelques éléments de programmation, refusent dans un premier temps cette approche méthodique. Il faut un certain temps pour qu'ils acquièrent (heureusement) ce mode de travail. Ce souci de rigueur, qui peut paraître "dogmatique", sera compensé par la liberté des activités proposées par le laboratoire ARCADE (voir §2.2) !

L'expression écrite est difficile, même dans un langage courant. L'étudiant a beaucoup de mal à écrire un programme, surtout s'il cherche à écrire directement dans un langage de programmation plus proche des besoins de la machine que de l'expression humaine : l'écriture du "**si-alors-sinon**" par rapport au "**selon**" en est un exemple. La notation algorithmique se place à un autre niveau d'expression et de plus libère l'étudiant de contraintes syntaxiques.

2.1.2. Machines abstraites

Une machine abstraite fixe un niveau d'abstraction pour libérer l'analyse de certaines contraintes imposées par la machine réelle. Elle est caractérisée par la définition de l'état et des primitives de modification et consultation des états. Sur le plan pédagogique, ce concept est utile pour "cacher" des difficultés à un certain moment pédagogique, notamment dès la phase de sensibilisation. On trouve dans [ScP 88] plusieurs exemples de telles machines.

La machine **Compteur** aide l'étudiant à raisonner sur l'initialisation et l'incrémentation d'une valeur. Son utilisation permet de différer une discussion sur la notion de variable et sur la signification d'expressions de type X := X + 1.

Dans un contexte de tracés de dessins, concrétisé par une machine **Tracé**, l'étudiant se familiarise avec la notion de spécification, découvre l'idée de paramètre et concrétise la notion d'état.

La machine **Caractères** est le support pour l'introduction de l'itération. Elle modélise un accès séquentiel à un texte et permet de reporter à une phase ultérieure l'étude du problème de terminaison. On centre ainsi la sensibilisation sur la recherche de l'invariant. Par ailleurs cette machine permet de lier la structure de séquence et la composition itérative.

2.1.3. Un ensemble d'exemples types

L'ossature du cours est une succession d'exercices modèles ou d'exemples types, ce qui permet d'aborder les difficultés les unes après les autres, d'isoler les notions essentielles, et de leur attacher un exemple concret et significatif.

Le tableau suivant résume à travers quelques exemples, les notions présentées pour introduire l'analyse des problèmes itératifs :

Données : un texte
Résultat : un entier ou un texte

Exemple type	Notions
Calculer la longueur	- découverte d'un schéma.
Compter les occurrences de la lettre "A"	- application d'un schéma (sur la base de la séquence des caractères données).
Compter les occurrences du couple de lettres "LE"	- introduction des variables intermédiaires. - application d'un schéma (sur la base de la séquence intermédiaire de couples).
Compter les mots	- résolution d'un problème par analogie avec un autre problème. - introduction d'une séquence intermédiaire (itération emboîtée).
Calculer la longueur moyenne des mots	- découverte de nouveaux schémas.
Supprimer les espaces redondants	- synthèse.

Figure 2.3 Une succession d'exemples types sur l'itération et le traitement de séquences.

Chaque exemple type présente plusieurs solutions aux problèmes posés. La diversité des solutions permet de mettre en évidence concrètement les divers choix qui sont faits lors de l'écriture d'un algorithme. Leur explicitation est à la base d'une programmation méthodique. La maîtrise de la programmation est alors l'aptitude à expliciter les choix, à organiser les différentes solutions possibles d'un problème et à identifier un problème ou partie de problème.

Un exercice modèle donne une certaine culture informatique qui permet à l'étudiant d'aborder dans de bonnes conditions la résolution d'un grand nombre de problèmes. Par exemple, la plupart des exercices du thème "traitement de séquences" portent sur le traitement de textes. L'étudiant comprendra facilement que les techniques décrites à cette occasion s'appliquent aussi bien à des séquences numériques.

"I try to keep in mind that teaching by setting a good example is often the most effective method and sometimes the only one available ... " [Wir 85].

2.1.4. Activités pratiques et leurs supports

a. Sensibilisation

Dans une phase de sensibilisation, l'enseignant cherche à susciter un intérêt qui servira de base à l'élaboration d'une solution algorithmique. Cette phase permet à l'étudiant de tâtonner à sa façon pour découvrir des éléments de solution. Une telle sensibilisation peut s'appuyer sur un jeu ou une simulation, par exemple : "l'épluchage des pommes de terre" de Dijkstra [Dij 71], la manipulation physique des cartes pour introduire un travail sur le tri, l'assemblage d'un document de dix pages [BiC 81]...

Au cours de cette phase, donner un jeu réel nécessite du matériel. Le jeu sur ordinateur est une simulation économique et efficace. L'étudiant observe d'abord le fonctionnement (l'exécution) du jeu, et ensuite pratique l'exercice. L'étudiant peut se servir de l'ordinateur comme d'un brouillon, garder les traces et vérifier certaines règles du jeu. Le programme permet aussi le guidage, par rapport au jeu réel.

Exemples des logiciels élaborés dans l'équipe pour la sensibilisation :
- jeu de balance pour introduire l'analyse par cas [Agu 85],
- jeu de cartes pour faire le tri par distribution [Lie 85],
- "Tours de Hanoï" [Gué 85] pour aborder la récursivité,
- manipulation des cubes [Arc 88e] pour aborder les tris.

b. Spécification et analyse

Les activités liées à la spécification et l'analyse prennent place dans le cours ou les travaux dirigés sous la direction de l'enseignant. Polycopiés et ouvrages de référence sont les supports essentiels de ces activités. La présence d'un manuel de cours permet à l'enseignant de se concentrer sur la discussion de l'analyse : le temps consacré par l'étudiant à prendre des notes peut être utilisé pour la discussion ou pour enrichir l'éventail des solutions proposées.

Les ouvrages américains [HoS 84], [Sed 86], [AHU 87] sont plus pragmatiques, leurs algorithmes sont écrits directement dans un langage et chaque procédure ou programme correspond à un énoncé très spécifique.

Les ouvrages du type [PMS 88], [ScP 88], [Gra 86] proposent une démarche différente : l'énoncé du problème initial, moins précis au niveau algorithmique entraîne une analyse plus riche, une discussion plus ouverte sur la mise en place des structures de données que l'on veut appliquer. L'accent est mis sur l'analyse des problèmes. La notion de spécification et la reconnaissance des schémas ont un

rôle primordial. Dans cet enseignement, l'analyse récurrente prend une place importante comme pour l'analyse fonctionnelle du traitement de séquence [Pey 88b]. "La récurrence est la base commune de la programmation itérative et récursive", [Ars 83].

Les livres [Wir 76] et [Wir 86] ont une position intermédiaire entre ces deux catégories.

En plus des supports écrits, l'ordinateur peut être utilisé pour visualiser des schémas, un arbre d'analyse, etc. On peut penser à une visualisation du type "film" pour expliquer le processus de conception d'un programme. Par exemple, nous avons produit un tel logiciel dans le projet ARCADE sur le thème du tri [Arc 88c].

Le jeu de rôle [Arc 88f] développé dans le projet ARCADE pour sensibiliser aux problèmes de spécification et d'analyse est un autre exemple concret de voie possible d'utilisation de l'ordinateur.

L'écriture d'un algorithme se finalise par un texte "séquentiel", mais sa construction n'est pas séquentielle. Dans un livre ou sur un listing, on voit le texte "tout prêt", séquentiel. Pourtant, la construction d'un algorithme ne se fait pas ligne par ligne, de la première à la dernière. L'écriture d'un algorithme est la conséquence de l'analyse, et notamment de l'application de schémas : on remplit composant par composant.

Une bonne visualisation peut montrer que l'écriture n'est pas séquentielle. Par exemple un film sur l'analyse du tri par minimum (cf. chapitre 3 et [Arc 88c]).

Un éditeur de composants proposant des schémas et des notations algorithmiques est certainement un outil d'écriture d'algorithme qui permettrait à l'étudiant d'avoir un meilleur cahier de brouillon par rapport à un outil

traditionnel (crayon, papier et gomme). Par exemple le travail décrit dans [Luc 89].

c. Passage de l'algorithme au programme

La notation algorithmique donne une certaine indépendance vis à vis des particularités de tel ou tel langage de programmation. Elle est conçue pour un interlocuteur humain alors qu'un langage de programmation doit être interprété par la machine réelle. Le passage de la notation algorithmique au langage de programmation doit être fait quasi-automatiquement à partir d'un ensemble de règles de traduction.

Pour concrétiser le programme dans un langage "réel", on choisit de démarrer par une programmation impérative mais on aborde aussi tôt après la programmation fonctionnelle. Dans ce cadre, le choix du langage (Pascal et Lisp) est peu significatif car il dépend ici de contraintes liées aux objectifs de la formation ou aux logiciels disponibles.

Faisons une analogie : le génie de l'écrivain Franz Kafka ne vient pas de la langue allemande avec laquelle il écrivait ses textes, même si cette langue a un rôle non négligeable dans l'expression de son génie. De même, l'analyse d'un problème ne dépend pas, pour l'essentiel, du langage dans lequel sera exprimé sa solution [Pey 88a].

Un traducteur d'une notation algorithmique à un langage de programmation "réel" serait un bon outil pour cette phase d'écriture de programme. La première version d'un outil de ce type [Luc 89] concerne les langages Pascal et BASIC.

d. Etude de l'exécution des programmes

Nous avons présenté cette activité au chapitre I. Le rôle de l'ordinateur est double :

- support d'exécution des programmes écrits par l'étudiant (TP traditionnel),
- outil de visualisation de l'exécution qui permet à l'étudiant de comprendre la sémantique d'un programme, d'étudier son comportement dynamique, de sentir la performance "expérimentale". La visualisation est dérivée de l'exécution en temps réel.

e. Lecture de programmes

On oublie souvent cette activité, pourtant lire est aussi important qu'écrire dans l'apprentissage. Le paradoxe : les enfants apprennent à lire avant d'écrire à l'école ; mais pas dans la programmation. [KDP 83] fait un rapport de recherche sur l'étude de cette activité sans utilisation de l'ordinateur, en testant la compréhension des programmes écrits sur papier.

De plus, dans la pratique professionnelle, un programmeur passe plus de temps à lire qu'à écrire des programmes. Lire un programme, pour le modifier éventuellement, doit permettre de comprendre :

- la structure du programme, la logique de sa conception,
- le détail syntaxique de chaque composant,
- la nature et le rôle des différentes structures de données.

La plupart des programmeurs détestent écrire la documentation, très peu de programmes sont bien commentés [You 75]. L'activité de lecture permet à l'étudiant de se rendre compte de l'importance de la documentation : la lecture d'un programme non commenté et non documenté encourage l'étudiant à

accompagner ses propres programmes d'une documentation précise et pertinente.

L' activité de lecture est souvent mise de coté dans l'enseignement pour certaines raisons :

- les contraintes de temps atteignent aussi les enseignants !
- la lecture de documents écrits (livres ou polycopiés) est individuelle en dehors des cours,
- la genèse du programme n'est pas décrite,
- l'exécution du programme est impossible.

L'ordinateur est un bon support pour la lecture :

- lecture linéaire instruction par instruction dans l'ordre du listing ou dans l'ordre d'exécution,
- lecture structurée (par procédure, par composant syntaxique, par schéma),
- mise en place du manuel sous forme d'un livre électronique ou d'un Hypertexte (réseau sémantique des idées pédagogiques) [YMV 85], [Sav 87].

f. En conclusion

Les différentes modalités d'observation de programmes, de sa construction à l'exécution, sont des éléments importants dans l'apprentissage de la programmation. Actuellement, ces activités sont peu supportées par l'ordinateur, faute de logiciels de visualisation.

"...The difficulty in future years will not be a lack of hardware ; it will be lack of software, and the lack of creatif visual thinking to drive new software" [Bro 87].

2.2. Un environnement logiciel d'accueil des activités pratiques : le laboratoire ARCADE

2.2.1. Chronologie

1984 ... Les démarches

Le projet DIDALP [Sch 85] était le projet initial développé au sein de l'équipe de Méthodologie de la programmation de l'IMAG. Ce projet avait pour but de fournir à l'étudiant en algorithmique et programmation des ressources EAO complémentaires à l'enseignement traditionnel. Les apports spécifiques attendus d'une activité EAO dans ce domaine portaient sur la compréhension dynamique qui sous-tend une description algorithmique, la compréhension de principes de modélisation informatique, et la compréhension des divers niveaux de lecture d'un algorithme. L'enseignement de base était fondé sur un ensemble d'exemples types [LPS 83]. Les didacticiels étaient structurés à partir de ces mêmes exemples types auxquels étaient associées plusieurs activités (ou leçons) et une activité de simulation. Les modes d'interaction offerts à l'étudiant dans une leçon étaient : exemple, dialogue ou requête. Les didacticiels devaient être produits sous le système auteur DIANE. La méthode générale de développement a été définie sur la base d'un ensemble de maquettes réalisées depuis 1983 et décrites dans [Zam 84].

En 1984/1985, deux recherches parallèles menées en DEA [Gué 85], [Lie 85] et un projet de 3° année de l'ENSIMAG [Agu 85] ont conduit à la réalisation de plusieurs logiciels de soutien à l'enseignement de la programmation. Ces recherches portaient sur 3 thèmes différents.

[Agu 85] réalise des didacticiels qui illustrent l'analyse par cas (par un jeu de balances) et la manipulation de la machine caractères. Ce travail confirme qu'un

système auteur comme DIANE fonctionnant sur micro-ordinateur Micral 90/50 est inadapté.

[Gué 85] présente deux didacticiels d'aide à l'enseignement de la récursivité . Le premier, réalisé sur Plessey, visualise l'exécution de la fonction factorielle. Le second, développé sur Macintosh, est un "jeu" sur les tours de Hanoï. Cette seconde réalisation a permis d'étudier également la contribution du système Macintosh, nouveau venu sur le marché des micro-ordinateurs, et de confirmer que ses modalités d'interaction pouvaient convenir à l'ensemble de nos expérimentations.

Mon rapport de DEA [Lie 85] était l'embryon de cette thèse. Je travaillais sur la structure de séquence avec le matériel Plessey disposant du système "Voyelles" développé par l'équipe [Luc 83]. Le résultat est un ensemble de didacticiels traitant le tri par distribution (*radix sort*) comme exemple type sur la structure séquence. Ces didacticiels permettent à l'étudiant de manipuler la structure, d'observer l'exécution du tri, de lire l'algorithme et de faire des exercices sur le tri par distribution. La visualisation de la structure séquence est étudiée et réalisée sur la représentation simulée d'un jeu de cartes, logique et physique (la représentation contiguë des paquets de taille fixe et la représentation contiguë des paquets de taille variable).

Les moyens de communication offerts par ce matériel ne permettent pas une bonne interaction avec l'étudiant : le clavier est le seul périphérique de saisie possible, le fenêtrage est impossible, l'écran semi-graphique est souvent surchargé par des informations présentées simultanément. Certaines conceptions de visualisation (surtout la représentation de chaînage) et de manipulation n'ont pas abouti à cause de la pauvreté du graphisme et de la lenteur du temps de réponse du système. L'exercice est peu individualisé, les données sont toujours fournies par le système.

Malgré ce matériel rudimentaire, cette recherche a donné des idées sur :

- la visualisation à partir de l'exécution d'un algorithme,
- la nécessité de différents niveaux de représentation de données pour mieux visualiser,
- la définition de la visualisation des opérations de base d'une structure particulière,
- les différentes activités proposés pour un sujet/thème : l'observation de l'exécution sur plusieurs représentations et la manipulation libre des primitives d'une structure sur d'autres objets (dans ce cas, la structure séquence est simulée par des cartes),
- les caractéristiques du matériel.

1985 ... La spécification du projet ARCADE

Les expériences tirées de ces travaux antérieurs permettent de définir plus précisément le "laboratoire" à réaliser. Un système auteur comme DIANE ne répond pas à nos attentes. La lourdeur de la programmation, la lenteur du système et la difficulté de l'interaction comme celle du système Plessey ne nous convient pas. L'interface Macintosh répondait bien à nos besoins d'interface élève-ordinateur.

Le projet s'intitule "ARCADE" et fait partie du laboratoire de Génie Informatique de Grenoble (LGI) de l'IMAG. Les deux rapports [Arc 86a], [Arc 86b] décrivent les premières spécifications du projet.

ARCADE en résumé...

Il est paradoxal de constater que les pratiques courantes de l'enseignement de la programmation au niveau universitaire réservent un rôle très limité à l'ordinateur. L'ordinateur et son emploi constituent certes le thème de l'enseignement, mais l'étudiant n'utilise concrètement la machine que pour écrire un petit nombre de programmes dans le cadre des classiques travaux pratiques. La présentation de certains aspects dynamiques, tels que l'exécution d'un programme ou l'évolution des structures de données reste la pierre d'achoppement de l'enseignement traditionnel.

Nous voulons étudier et proposer plusieurs utilisations possibles de l'ordinateur comme aide et assistance à l'enseignement de la programmation. Nous ne souhaitons pas développer un cours complet et autonome car nous n'envisageons pas que l'étudiant puisse partir avec une disquette et revenir quelques mois après passer l'examen ! Notre but est de réaliser **un laboratoire de programmation** totalement informatisé dont les logiciels viendraient enrichir l'enseignement traditionnel : cours, travaux dirigés, travaux pratiques. Les logiciels réalisés constituent les étages, les salles et le matériel du laboratoire.

La conception de notre laboratoire de programmation s'inspire de plusieurs modèles connus dans des domaines divers : laboratoires de physique et de chimie, Cité des Sciences et de l'Industrie, **arcades** de jeux, jeux d'aventures. Certes ces modèles ont leurs limites et doivent être adaptés : leurs buts ne sont pas identiques, ils ne visent pas le même type d'apprentissage, ils ne s'adressent pas au même public, etc.

Mais nous souhaitons que notre laboratoire soit une synthèse de ces modèles avec :

- un abord plaisant, attrayant, ludique.
- une grande liberté laissée à l'étudiant pour se déplacer dans le laboratoire, en fonction des difficultés rencontrées.
- une implication forte dans des activités nombreuses et variées.
- **la manipulation, le jeu** et **l'observation** comme modes de travail privilégiés.

Le rôle premier et fondamental du laboratoire est d'être **un lieu d'accueil** pour :

- les logiciels qui y sont placés,
- l'étudiant qui vient y pratiquer un ensemble d'activités,
- l'enseignant qui y cherche des compléments de cours.

Pour remplir au mieux ce rôle, le laboratoire fournit :

- une bonne organisation d'ensemble permettant au visiteur de se diriger en fonction de ses intérêts, de découvrir l'existence des logiciels appropriés et de les utiliser facilement,
- de bonnes conditions de travail qui facilitent l'apprentissage et augmentent le plaisir d'apprendre.

Même si notre laboratoire privilégie la liberté de l'étudiant, les besoins de **communication** entre enseignant et étudiant ne disparaissent pas totalement. Quelques mécanismes simples sont utilisés à cet effet :

- des listes d'exercices suggérés stimulent l'imagination de certains étudiants, ou attirent leur attention sur des difficultés précises,
- un mécanisme de type "courrier des lecteurs" ou "forum" permet des échanges informels, redonne une certaine chaleur humaine à l'environnement en recueillant facilement les remarques et les suggestions des utilisateurs.

La **liberté** laissée à l'étudiant d'organiser ses activités s'oppose à l'idée de "contrôler" le cheminement de chacun. Rien n'empêche toutefois l'enseignant d'aiguiller ses étudiants vers telle ou telle activité en fonction de l'avancement de son cours, l'étudiant restant libre d'explorer, de sa propre initiative, des "régions" inconnues. L'idée de fournir des plans de parcours, guides "touristiques" ou "thématiques", est à l'étude.

Les logiciels développés par notre groupe sont évidemment placés dans le laboratoire, mais le laboratoire reste un lieu **ouvert** aux logiciels commerciaux ou du domaine public, qui illustrent des concepts en rapport avec le thème de notre laboratoire. Nous les désignons par le terme de "logiciels invités", par analogie avec les "professeurs invités" ou les "conférenciers invités". On retrouve bien ici l'aspect fondamental de lieu d'accueil. Les logiciels invités sont signalés comme tels. Leur origine est citée (si elle est connue).

1986... La réalisation

Chaque chercheur du projet ARCADE réalise des logiciels selon ses préoccupations. Dans un premier temps, l'équipe n'a pas défini le regroupement des logiciels. La sortie du logiciel Hypercard en 1987 nous permet d'intégrer les applications réalisées. Dès mars 1988, la première version du laboratoire est "opérationnelle" [Arc 88g] L'aspect de jeu est étudié particulièrement comme sujet de thèse [Gué 89].

2.2.2. Etat actuel du laboratoire ARCADE

La structure du laboratoire : bâtiments, salles, activités

Le laboratoire se présente comme un paysage semé de bâtiments (cf. Annexe A). Chaque bâtiment héberge diverses activités : son nom renseigne sur le contenu. Il faut désigner un bâtiment pour pénétrer à l'intérieur.

Les bâtiments sont les suivants :

- le **bâtiment "Accueil"** permet à un nouvel utilisateur de consulter quelques écrans d'informations sur les principales conventions employées,
- des **bâtiments standards** contiennent chacun pour l'instant une seule salle. Toutes les salles ont le même aspect général : les noms des différentes activités proposées accompagnés d'icônes dont le symbolisme deviendra plus significatif avec la pratique des activités,
- le **bâtiment "Langages"**, pour la mise en œuvre des programmes. Illustration parfaite de notre politique d'invitations, la salle "Langages" propose :

  a) l'interpréteur Pascal (Mac Pascal de Think Technologies),

  b) un interpréteur Basic (de Microsoft).

  Pour faciliter son travail, l'étudiant accède aux fichiers sources de plusieurs programmes étudiés dans les autres salles. Avec l'interpréteur, il peut ainsi examiner leur texte, suivre leur exécution normalement ou en pas à pas, les modifier, écrire des variantes, etc.

  L'expérience menée dans une université américaine [BST 86] confirme l'efficacité de l'interpréteur MacPascal dans la mise en place de petits programmes, surtout pour les débutants.
- des **bâtiments spéciaux** : bibliothèque, cinéma, ...

Activité type

Chaque salle propose des activités types [Arc 88g]. Le choix d'une activité conduit à un écran de présentation, similaire pour toutes les activités.
On y trouve toujours :

- une brève description de l'activité pour donner au novice une idée de ce qu'elle contient. Il peut ainsi explorer rapidement la salle pour voir les ressources offertes. L'habitué trouve là une confirmation de l'activité qu'il veut pratiquer.
- une icône permettant de lancer effectivement l'activité. Le mode d'emploi propre au logiciel choisi entre alors en vigueur. Lorsque l'utilisateur "quitte" le logiciel, il revient sur ce même écran de présentation.
- selon les activités, une ou plusieurs autres icônes donnent accès à des conseils d'utilisation, à des idées d'exercices, au courrier des utilisateurs, etc.

Le contenu actuel du laboratoire (au moment de la rédaction de cette thèse)

La Figure 2.4 résume les salles du laboratoire Arcade et leurs logiciels ; un même logiciel peut appartenir à plusieurs salles. Les logiciels précédés du signe sont mes réalisations. Les salles sont présentées par copie d'écrans en annexe A.

Salles	**Description des logiciels**
Salle de Tris	
La Gare de Triage	Dessin animé illustrant le tri par minimum.
Tri en BD	Exercice sur la Gare de Triage.
Tri Guidé	Apprentissage guidé du tri par minimum.
Meccano de Tri	Pratique libre des algorithmes de tri, sur des cubes.
Tris internes	Visualisation et confrontation de 7 algorithmes de tri.

Figure 2.4 Salles du laboratoire Arcade et leurs logiciels.

Salles	Description des logiciels
Salle Récursivité	
Dessins récursifs	Travail sur la récursivité.
Baguenaudier	Visualisation du jeu et exercice.
Les 8 reines	Visualisation du problème de 8 reines et exercice.
Arbres Binaires	Pratique libre des algorithmes classiques.
Tours de Hanoï	Découverte guidée, du jeu à l'algorithme récursif.
Salle Tables	
Exploration de tables	Illustration des algorithmes de recherche.
Recherches Min et Max	Illustration de recherche du minimum et du maximum.
Pattern matching	Etude et visualisation de 4 algorithmes de la reconnaissance d'un motif.
Salle Caractères	
Compter les "A"	Exécution libre de l'algorithme compter les "A".
Compter les "LE"	Exécution libre de l'algorithme compter les "LE".
Machines caractères	Manipulation libre des machines caractères.
Salle Cinéma	
La Gare de Triage	cf. Salle de Tris.
5 films	Réalisés avec Videoworks de Micromind.
Salle Graphes et Arbres	
Arbres binaires	cf. Salle Récursivité.
Tri topologique	Visualisation du tri topologique et exercice.
Parcours de Graphes	Visualisation des opérations de base et exercice.
Logiciel invité	*Binary Tree.*
Salle Spécifications	
Toi, Moi et Lui	Jeu de rôles sur la construction des programmes modélisée par la création de bandes dessinées.
Anagramme	Exemple d'une page de bande dessinée.
Salle Langages	
Interpréteur Macintosh Pascal	Think Technologies, Inc.
Interpréteur Microsoft Basic	Microsoft.

Figure 2.4 Salles du laboratoire Arcade et leurs logiciel (suite et fin).

3. Analyse d'expériences de visualisation

Cette étude bibliographique concerne la visualisation de l'exécution des programmes. Elle prolonge les recherches présentées dans la partie bibliographie du premier rapport d'Arcade [Arc 86a].

Nous analysons d'abord les différentes méthodes et outils de visualisation et leur rôle dans l'apprentissage, ainsi que l'utilisation d'images graphiques dans la programmation.

Nous étudions ensuite les logiciels existants dans le domaine de la visualisation (de l'exécution) des programmes. Nous décrivons brièvement les principales fonctionnalités de ces logiciels sous forme d'un tableau de référence.

Enfin, nous précisons la nature des logiciels que nous voulons réaliser pour le laboratoire ARCADE.

3.1. Méthodes et outils de visualisation

3.1.1. Image et texte

Pendant des siècles le texte a été l'outil de communication le plus largement utilisé. L'apparition de nouvelles techniques et de nouveaux média donne maintenant aux images une place prépondérante. Plusieurs congrès dans le domaine de l'EAO témoignent de l'importance des thèmes image et audiovisuel dans l'enseignement : Formav 86, 6° Congrès de l'EAO [Arc 86c], Congrès EAO-87 [EAO 87], 7° et 8° Congrès de l'EAO et Communication Interactive 88 [JFC 88], [JFC 89]. Pourquoi ?

Le cerveau humain traite différemment l'acquisition, le stockage et la recherche des informations selon la nature de l'information présentée : image ou texte. L'hémisphère gauche fonctionne plutôt de façon logique, verbale ou analytique comme l'ordinateur. Par contre, l'hémisphère droit du cerveau est spécialisé pour la perception "gestalt", spatiale ou synthétique.

"Those in [AI] loved the left side of the brain, and [those in graphics] love the right. And what we've done is fund the left, but not the right" [Fre 88].

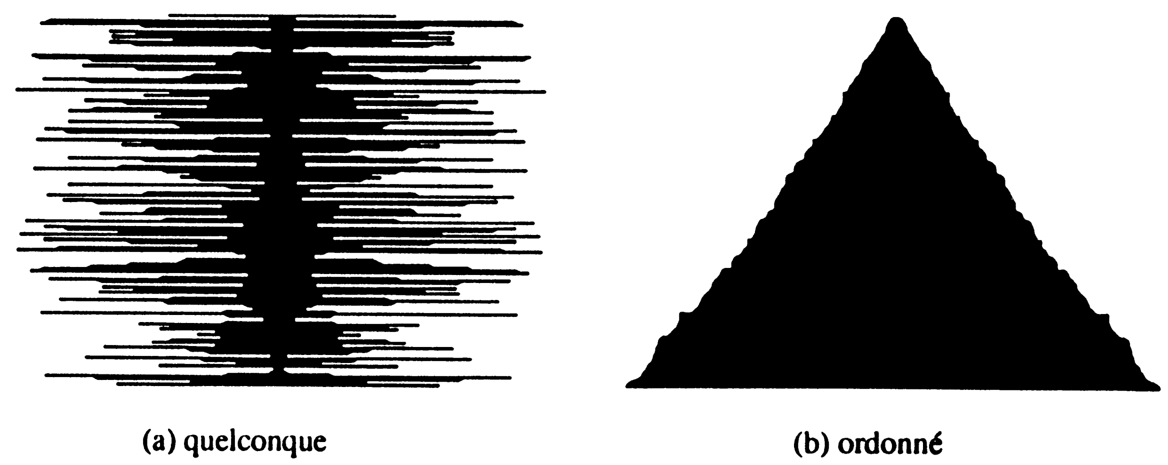
L'image et le texte sont deux langages différents. La compréhension d'une image repose sur la perception aléatoire de ses différents constituants et de leur mise en relation. La compréhension d'un texte relève d'un processus discontinu et complexe : la succession des éléments de base (lettres et espaces) forme une suite de mots analysée au fur et à mesure.

La rétention des images est rapide car celles-ci sont mémorisées directement dans la mémoire à long terme. Pendant l'acquisition, le texte est stocké dans la mémoire à court terme (qui est limitée), et ensuite transféré dans la mémoire à long terme. Une expérimentation [Hab 70] a montré que les sujets d'expérience reconnaissaient 85-95% des images présentées rapidement auparavant.

L'image a plus de dimensions (forme, taille, couleur, texture, direction, distance) que le texte. Dans un espace identique, une image est plus riche et plus compacte mais parfois moins précise. Elle peut présenter plus d'informations, mais l'abstraction peut être d'une interprétation délicate. De plus, la représentation visuelle favorise souvent l'intuition et la découverte. Il est intéressant d'exploiter cet aspect car la visualisation sur la surface d'un écran est toujours limitée.

Par exemple, les graphiques ci-dessous représentent le même tableau de valeurs entières. Cette représentation matérialise chaque élément par un trait

horizontal. Un écran Macintosh de 512 x 342 points peut alors présenter un tableau d'environ 300 éléments maximum.



(a) quelconque (b) ordonné

Figure 3.1 Représentation d'un tableau.

Il manque des détails dans l'acquisition des informations d'une image. Le texte scientifique est précis parce qu'il est écrit. La précision de reconnaissance est importante pour établir une interaction. L'image seule ne raconte jamais une histoire : le contexte est important pour la reconnaissance et la compréhension car il lève l'ambiguïté de l'image. Si le lien entre une image et le contexte est inexistant ou incomplet, la communication est inefficace.

Reprenons l'exemple ci-dessus. Si l'étudiant ne connaît pas la convention de la proportionnalité entre la longueur d'un trait horizontal et la valeur de l'élément du tableau, l'image n'a pas le sens voulu par l'émetteur (concepteur). Il pense peut-être à une montagne ou à un sapin de noël ! Par contre, sachant qu'un trait horizontal est proportionnel à la valeur de l'élément, l'étudiant peut comparer l'aspect aléatoire de l'image de gauche et l'aspect ordonné de l'image de droite.

Une image est le plus souvent un langage "universel", elle permet une référence directe et immédiate à un objet. Dans un texte, la référence est

indirecte et culturelle car on doit donner un "nom" ou un mot du langage pour désigner cet objet.

Les icônes du Macintosh, les symboles utilisés aux Jeux Olympiques 1968 [Gom 72] ou dans les aéroports internationaux [Lod 83] sont de bonnes illustrations de l'universalité.

En pratique, une visualisation est rarement indépendante d'une langue car l'image et le texte sont souvent associés. Le système Macintosh permet au programmeur de séparer le texte et l'image dans le code des programmes, ce qui facilite la traduction d'une langue à l'autre.

Dans le domaine de la communication écrite, la combinaison, l'association des mots aux images est une technique efficace pour améliorer la mémorisation et la compréhension du concept ou des informations.

Dans le domaine de l'enseignement de la programmation, plusieurs expériences ont été menées [Sca 87], [Sca 88] pour comparer ces deux modes de représentations (texte et graphique) de programmes. La plupart des hypothèses de ces expériences sont dérivées des fonctionnements différents de l'hémisphère droit et gauche du cerveau dans le traitement de l'information. D'après ces recherches, le pseudocode est traité seulement par l'hémisphère gauche, tandis que l'organigramme est traité par les deux hémisphères. L'organigramme est donc un outil plus accessible et plus efficace, car les deux hémisphères sont impliqués dans le processus de compréhension de l'algorithme associé. Les résultats de ces deux expérimentations confirment qu'il y a deux types d'étudiants : 75,1% à 89,1% préfèrent l'organigramme (méthode graphique), les autres le pseudocode (méthode verbale). La complexité croissante d'un algorithme accentue encore cette préférence. Malgré cette préférence marquée, les deux représentations doivent coexister car elles contribuent différemment à la compréhension. L'auteur ne veut pas généraliser le résultat de ces recherches à la conception et la documentation de programmes volumineux.

Dans [AcP 89], les chercheurs séparent leurs étudiants en 2 groupes : un groupe utilise le langage Pascal textuel et l'autre utilise le diagramme graphique du langage Pascal développé au "Naval Surface Warfare Center", USA. Ils ont mesuré les heures passées par les étudiants pour la conception, le codage et l'implémentation des programmes dans les travaux pratiques. Les expérimentations ont été effectuées pendant 2 semestres sur un grand nombre d'étudiants. Le résultat a montré que dans toutes les phases des travaux pratiques, les étudiants qui travaillent en représentation graphique gagnent du temps, 26,5% en moyenne, par rapport à ceux qui travaillent en Pascal textuel.

A notre avis, le résultat de ces expérimentations est vérifié pour la compréhension d'un programme par son exécution. Par contre comment comprendre l'analyse à l'origine d'un programme et sa structure, uniquement en terme d'exécution ?

3.1.2. Supports statiques ou dynamiques

Nous avons parlé de la visualisation indépendamment du support. La visualisation peut être effectuée sur un support statique (papier, livre, tableau noir, transparents) ou sur un support dynamique (film, écran de télévision ou d'ordinateur). L'image est prédéfinie sur un support statique, ou elle est enrichie au fur et à mesure de l'évolution du contexte à l'aide d'un support dynamique.

L'utilisation d'un support statique rend difficile la visualisation de l'exécution d'un programme : contrôle et évolution des valeurs de données. Il serait possible de décrire cette exécution en terme de succession d'états sur un petit nombre de jeux d'essais prédéfinis au prix d'une extension démesurée du texte. Mais il est de toutes façons impossible de montrer les aspects dynamiques et on ne peut pas exploiter des situations non prévues par l'auteur.

Par contre, un support dynamique comme le film est capable d'illustrer le dynamisme. Citons quelques exemples de films pour l'enseignement de la programmation :

- le film couleur "Sorting Out Sorting" [Bae 75], [Bae 81] montre l'exécution des tris internes. La visualisation d'un tableau de grande taille est la base de travail de divers groupes de recherches [Bro 88], [LoD 85].
- le film sur l'adressage dispersé [Hop 74] est une illustration de l'occupation mémoire et de la collision sur un tableau. Le film visualise dynamiquement le nombre d'essais pour arriver à occuper chaque case mémoire, le nombre maximal de collisions à tout instant et un graphique résume la performance de l'algorithme. Le film montre aussi les conséquences d'un changement de taille du tableau, et le comportement de l'algorithme selon différentes données.
- les films audiovisuels [Hau 88] et [Vau 85] illustrent la métaphore suivante : l'homme se déplace vers une boîte (programme), prend une feuille de papier (l'instruction y est écrite), lit l'instruction, exécute la tâche (processeur) en manipulant des valeurs dans les cases (mémoire) ou par les dispositifs d'Entrée/Sortie. Les auteurs soulignent le rôle important de la métaphore dans l'apprentissage.
- le film réalisé dans le cadre du projet ARCADE sur l'exécution de Tri par minimum [Arc 88c] procède par analogie : deux cheminots, le chef et Henri, trient des wagons de chemin de fer (un tableau : le numéro des wagons désignent ses valeurs). Le chef effectue les comparaisons en parcourant les wagons pour chercher un wagon ayant le numéro minimum (le programme principal), et Henri échange les deux wagons indiqués par le chef (la procédure d'échange). Les constructions de l'algorithme et du programme Pascal sont illustrées ensuite par deux personnages : Monsieur Algo repère les actions de base composant l'algorithme et Monsieur Pascal codifie l'algorithme dans un langage de programmation.

La réalisation de ces films demande des moyens et du temps ! Par exemple, le beau film "Sorting Out Sorting" [Bae 81] a nécessité 3 ans de production, pour 30 minutes de projection. Le film sur le tri par minimum [Arc 88c] a nécessité 1 an de travail pour 15 minutes de déroulement.

Parfois l'ordinateur facilite la visualisation dynamique puisque le programme génère les éléments de visualisation à l'exécution. La technique du film est alors trop lourde car le volume d'images à stocker est trop important.

L'ordinateur permet l'**interactivité**, alors qu'un support statique ou même la visualisation filmée déroule une information figée et préconçue. L'interactivité permet une observation active et individualisée.

"A picture is worth a thousand words, a movie even more. But interactive movies, available on workstations to virtually everybody, are even more valuable" [Bro 87].

3.1.3. Visualisation et programmation

Les différents outils de représentation graphique (symboles, images, schémas) sont souvent utilisés dans la documentation : structure de programme, architecture d'un système informatique, symbole de composant électronique.

Dans le domaine de la programmation, on utilise l'organigramme, le diagramme *Nassi-Shneiderman* (diagramme NS) [NaS 73], le diagramme d'état.

Pour illustrer les données, on utilise le diagramme des flots de données (*dataflow diagram*) [Rae 85] ou la représentation classique de chaque structure (arbre, graphe, tableau, liste).

Dans le domaine de la programmation, les images figuratives ne sont pas nécessaires pour la visualisation. La plupart des images peuvent être présentées clairement sous forme de graphiques en deux dimensions sans texture, ni relief, ni perspective. Des attributs graphiques supplémentaires (couleur) peuvent rendre certaines visualisations plus attrayantes et esthétiques (comme dans le cas des courbes fractales).

Les images produites par ordinateur sont un outil pour interpréter les données. Ceci est utile dans d'autres domaines scientifiques, par exemple en médecine ou en chimie (visualisation en couleurs des structures de molécules). Un système d'interprétation graphique permet la synthèse rapide des données sous différentes formes : histogramme, diagramme camembert, courbe plane [Lau 76], [Tha 84].

Evoquons l'époque de la programmation par lots. Un programme est une boîte noire à laquelle on fournit des données et qui produit, après l'exécution, un résultat. Il n'y a aucune visualisation, s'il y a une erreur, on recommence.

La "visualisation" de programme était alors très rudimentaire car effectuée sur papier :

- pour lister le programme source (listing),
- pour la mise au point du programme, par impression de traces (valeurs de variables utilisées) ou à partir d'un *"dump"*.

Cette technique de programmation a pratiquement disparu. La programmation est plus interactive, la visualisation est plus riche, plus dynamique.

La visualisation est un élément essentiel des recherches informatiques dans le domaine du calcul scientifique [IEEE 87], [SIG 87]. Pendant l'exécution d'un long programme de calcul, la visualisation permet de déceler l'apparition

d'erreurs ou d'anomalies dans les données. Le programmeur peut arrêter l'exécution au point d'erreur, corriger puis laisser l'exécution reprendre. La visualisation peut éviter ainsi l'exécution complète d'un programme erroné.

La visualisation permet de comprendre le cheminement d'un programme et son fonctionnement. Parfois, le programme simule et visualise le fonctionnement d'un processus, par exemple un processus industriel [FoM 86], des microprocesseurs comme le système MIDAS [GFV 81] ou MICRO3 [PLM 87], la machine de Turing (le logiciel "Turing" sur Macintosh).

La visualisation peut être utilisée avec des objectifs divers [Rae 85] :

- **mise au point** : un système graphique visualise l'exécution d'un programme, dans le but de faciliter sa correction,
- **performance** : le temps d'exécution et l'occupation de l'espace mémoire peuvent être bien visualisés et animés, ce qui aide au travail d'optimisation,
- **parallélisme** : la visualisation des actions concurrentes d'un programme en exécution est un moyen pour mesurer le parallélisme du programme [ZPS 88],
- **typage** : il est facile de représenter plusieurs types par les couleurs ou les formes,
- **abstraction** : la lecture séquentielle d'un programme n'est pas appropriée pour comprendre la structure abstraite d'un programme. Un système de visualisation est un bon moyen pour naviguer dans un programme module par module ou par niveau d'abstraction.

La visualisation ne s'applique pas uniquement dans le domaine des programmes d'application, mais aussi dans d'autres domaines, par exemple pour faciliter la navigation *("browsing")* dans un système de base de données [Lar 86]

ou dans le prototypage des logiciels de systèmes concurrents et répartis [CoA 87].

3.1.4. Visualisation et manipulation

La diffusion plus large de matériels graphiques performants a donné naissance à des systèmes graphiques interactifs où la visualisation prend une place prépondérante.

Les modalités de communication homme-machine évoluent dans le sillage des matériels [Rae 85] :

- les systèmes de manipulation directe d'objets graphiques remplacent peu à peu l'utilisation des langages de commande textuels,
- l'application de l'intelligence artificielle au traitement du langage naturel améliore la communication textuelle,
- les outils de traitement graphique améliorent la communication visuelle.

Apport de la visualisation à l'utilisateur

Grâce aux menus déroulants, aux fenêtres, à la souris, qui sont faciles à manipuler, l'utilisateur peut "se déplacer" facilement d'une partie à l'autre dans le système. L'image dynamique visualise en temps réel l'évolution du système. La technologie graphique permet aussi une représentation commune du texte et de l'image. Grâce à la visualisation, l'interface homme-machine s'est beaucoup améliorée.

Shneiderman dans [Shn 83] analyse l'approche par manipulation directe d'objets dans un système informatique. Les caractéristiques de ce type de système sont : l'interactivité, la facilité d'utilisation, la présentation permanente d'objets. Des matériels simples comme souris ou manche à balai permettent un dialogue

avec l'ordinateur en manipulant directement les représentations graphiques des objets informatiques utiles. Le résultat de l'opération demandée est visualisé immédiatement. La possibilité d'effectuer une action pas à pas permet, en cas d'erreur de défaire ou de refaire.

Un novice est opérationnel avec quelques actions simples et faciles, il progressera et apprendra à utiliser des actions plus complexes, en combinant les premières.

Ce type de système a les aspects positifs suivants :

- l'utilisateur maîtrise le système facilement,
- il acquiert par lui-même une compétence sur le système,
- il a confiance, il désire explorer le système,
- le système est agréable à utiliser,
- l'appropriation d'un langage de commande est inutile.

Rappelons tout de même les contraintes de visualisation que le concepteur doit respecter :

- l'image graphique est efficace si la représentation des objets et des relations est condensée. La sélection des informations présentées doit éviter le superflu.
- une image ambiguë mal conçue peut donner lieu à plusieurs interprétations différentes, fausses ou non. Il est possible que pour la même image, la signification construite par le concepteur ne soit pas la même que celle de l'utilisateur. Plusieurs essais avant la diffusion du système sont indispensables pour faire coïncider des références culturelles différentes.
- ne pas surcharger l'espace limité du dessin (l'écran).

La manipulation, la visualisation et le jeu sont des activités privilégiées du laboratoire ARCADE. En réalité, les trois activités sont fortement imbriquées :

la **visualisation** permet la **manipulation** directe d'objets. Les **jeux** vidéo comme "Load Runner" ou "Dark Castle" l'illustrent avec succès.

Apport de la visualisation au programmeur

Les langages de programmation évoluent en permanence : les langages de "haut niveau" sont très utilisés, mais aussi les langages de programmation visuelle [Cha 87]. Ce type de langage offre à l'utilisateur une programmation sans frappe de texte qui s'effectue par manipulation directe d'objets, d'entités graphiques ou d'images. Le programmeur sélectionne et compose des icônes pour construire un programme. Ensuite, il commande l'exécution et observe se qui se passe.

3.1.5. Visualisation de programmes

Il subsiste encore des confusions sur l'emploi des termes : visualisation de programme et programmation visuelle (même dans l'introduction de la revue IEEE Computer du mois d'août 1985 qui est consacrée à la programmation visuelle). Myers dans [Mye 86], [Mye 88] tente d'éclairer ces confusions en proposant une taxonomie des logiciels concernant la visualisation.

Un programme est un objet informatique, donc un objet abstrait matérialisé par la visualisation **de ses instructions** et **de ses données**. La visualisation de programme peut-être statique ou dynamique.

Visualisation statique

Pour les codes, la visualisation statique n'est pas systématiquement graphique, par exemple l'affichage du programme dans l'environnement du Mac Pascal, ou l'indentation se fait automatiquement et les mots clés sont affichés en caractères gras. Sous forme graphique, les codes du programme sont représentés par

organigramme ou par composants selon le vocabulaire du langage (par exemple, le langage IBOL dans Chipwits du Macintosh [ShJ 84]).

La visualisation statique des données permet de montrer les objets traités dans leurs états initiaux et finals. L'exemple de cette visualisation est la saisie ou la visualisation des données du programme avant et après l'exécution (éventuellement graphique)

Visualisation dynamique

La mise en évidence **des instructions** en cours d'exécution permet au programmeur de suivre le cheminement du programme. Par exemple la visualisation sur les composants dans le langage IBOL du Chipwits ou la petite main pointant l'index sur l'instruction exécutée par l'interpréteur Mac Pascal. La visualisation dynamique des instructions de programme peut être très primitive (l'indication instruction par instruction) ou structurée (procédure par procédure).

La visualisation dynamique **des données**, automatique ou non, est en général plus compliquée à réaliser que la visualisation des instructions.

Raeder identifie 3 méthodes possibles pour visualiser un programme [Rae 85] :

- créer une bibliothèque de visualisation, écrite en langage source. Le programmeur insère les procédures de cette bibliothèque dans son programme pour obtenir la visualisation aux points d'arrêt voulus. Dans cette méthode, il existe deux modules indépendants : le programme source et la bibliothèque.
- travailler au niveau le plus bas du logiciel : on insère les ordres de visualisation dans le code exécutable. Chaque modification de l'état du programme déclenche la visualisation automatique de l'image correspondante. L'aspect automatique de cette technique la rend inadaptée pour une animation pédagogique de l'algorithme.

- disposer d'un "moniteur" qui extrait les informations intéressantes du programme et les visualise pour obtenir des images-écran instantanées. On traite ensuite ces images pour obtenir une animation.

3.2. Panorama de logiciels de visualisation

Malgré les efforts portés sur l'analyse (cf. §2.1.4.6), selon [PIN 81], la réalité montre qu'un programmeur passe la moitié de son temps de production dans des activités liées à l'exécution des programmes : test ou mise au point. C'est pourquoi le "moniteur" de programme est un outil appréciable : la visualisation du fonctionnement et des données pendant l'exécution du programme permet au programmeur de détecter des anomalies et de corriger des erreurs, de comparer ce que "signifie" le programme avec ce qu'il devrait signifier.

La visualisation d'un programme en cours d'exécution recouvre **deux objectifs** différents :

1. faciliter la mise au point du programme (**assistance à la production**)
2. étudier le programme (**assistance à la compréhension**).

Les deux objectifs sont parfois complémentaires car la compréhension du programme accélère sa production. Mais le principe de visualisation dans les deux cas est fondamentalement différent, et ce pour trois raisons :

- un système d'assistance à la production travaille sur la visualisation automatisée à partir de l'exécution de n'importe quel programme, alors que l'assistance à la compréhension est parfois indépendante de l'exécution,
- un système d'assistance à la compréhension incite le concepteur à une interprétation sémantique et un travail pédagogique préalable pour placer au sein du programme des annotations à vocation pédagogique,
- un système d'assistance à la production est fait surtout pour la mise au point et la recherche d'erreurs, il doit donc associer en permanence les données visualisées aux variables correspondantes (déclarations, noms).

3.2.1. Assistance à la production de programmes

L'assistance la plus primitive concerne la mise en page du texte du programme par visualisation de la structure (indentation, mise en évidence des mots clés ou réservés), par la typographie et le formatage. Il s'agit de la visualisation "statique" des programmes sources : éditeurs, production de beaux listings.

Les outils d'édition de programmes évoluent vers des éditeurs dirigés par la syntaxe comme l'éditeur du Mac Pascal, éditeurs graphiques, éditeurs par composants comme "Cornell program synthesizer" [TeR 81].

L'éditeur se trouve de plus en plus intégré à un environnement sophistiqué :

- les vérifications faites à la saisie : cohérences de types, contraintes, etc. Si les programmes sont sous forme graphique, les éditeurs graphiques manipulent des symboles.
- les liaisons entre compilateur et éditeur.

a. Environnements de programmation

Plusieurs systèmes interactifs de développement de programmes sont des environnements intégrés et visuels, par exemple :

- un moniteur graphique interactif de programme est présenté dans [ClR 83],
- PIGS [PoN 83] permet l'édition et l'exécution du programme sous forme de diagrammes NS,
- Programming In Picture (PiP) [Rae 85] est un environnement de programmation fonctionnel sous forme d'images,
- PROGRAM VISUALISATION (PV) [BHK 85] est un prototype d'un environnement de programmation visuelle,
- PECAN [Rei 84] visualise plusieurs vues de programme à l'édition et à l'exécution,
- CEDAR [Tei 84] est un environnement graphique interactif qui intègre l'éditeur, le compilateur et plusieurs outils de mise au point,
- PICT [GlT 84] permet de programmer par organigrammes,
- PEGASYS [MoH 85] permet de programmer et de documenter un programme sous forme d'images,
- THINKPAD [RGR 85] est un système visuel pour la programmation par démonstration,
- Xerox Interlisp-D [BSS 86] est un environnement interactif et graphique de programmation pour le Lisp,
- GARDEN [ReP 86] est un environnement de programmation graphique,
- PROVIDE [Moh 88] visualise le programme pour faciliter la mise au point.

Dans ces environnements, le programmeur peut observer l'exécution de son programme et l'évolution des données.

Nous résumons dans la figure 3.2 les caractéristiques des logiciels que nous avons examinés (quelques logiciels ne sont pas présentés dans le texte).

Système Référence	Visualisation	Outils	Réalisation
PV [BHK 85]	- Structure de données (graphique) - Programme	- Editeur graphique	C VAX 11/780
PECAN [Rei 84]	- Structure de données (graphique) - Programme (édition, exécution) - Diagramme NS - Déclaration de variable - Type de variable - Arbre d'expression - Table de symbole - Organigramme de l'exécution - Pile d'exécution	- Editeur graphique - Editeur de texte - Gestionnaire de visualisation. - Env. de programmation	Poste de travail de l'univ. Brown Apollo
THINKPAD [RGR 85]	- Structure de données - Programme en Prolog -	- Editeur de données - Editeur de condition - Editeur d'opération	C, Prolog Apollo(UNIX)
GARDEN [ReP 86]	- Programme Structure de données -	- GELO: *graphics layout package* - APPLE: éditeur de dessin - PEAR : éditeur de vue	
PIG [PoN 83]	- Diagramme NS (édition, exécution)	- Editeur de diagramme NS	Pascal PDP 11/70

Figure 3.2 Environnement de programmation, assistance à la production de programmes.

Système Référence	Visualisation	Outils	Réalisation
PEGASYS [MoH 85]	- Programme (dessin) - Programme (texte) (en ADA)	- Editeur graphique - Editeur de texte	Interlisp-D Xerox-1100
PICT [GIT 84]	- Valeurs de variable - Organigramme (édition, exécution)	- Editeur graphique	Pascal VAX 11/780
[ClR 83]	- Diagramme NS (exécution)	- Générateur du diagramme NS - Editeur.	Pascal
REHEARSAL [FiG 84]	- Didacticiel, leçon (graphique) - Outils d'exécution	- Editeur de dessin - Outil de mise au point	Smalltalk-80 Xerox1132 (Dorado)
CEDAR [Tei 84]	- Document - Programme (Pascal, Lisp)	- Traitement de texte - Env. de programmation	Mesa Xerox1132 (Dorado)
PROVIDE [Moh 88]	- Programme - Module - Fonction	- Visualisation de programmes - Outil de mise au point	Macintosh lié au réseau avec VAX 11/780 sous BSD.42 UNIX
PVS [FoM 86]	Processus dynamique interactif : - création des icônes - définition des vues - lien - tableau de vues - animation des vues	- Editeur	
CAPS [Yao 85]	- Texte de programme - Message d'erreur - Conseils - Visualisation graphique des valeurs numériques	- Création, - Edition - Exécution - Mise au point	
GANDALF [HaN 86]		- ALOE : éditeur dirigé par la syntaxe	

Figure 3.2 Env. de programmation, assistance à la production de programmes (suite).

Système Référence	Visualisation	Outils	Réalisation
POE [FJM 84]	Beau listing du programme PASCAL	- Editeur, reconnait : la syntaxe et la sémantique	
CORNELL Progr synthesizer [TeR 81]		- Editeur par composant	
MAGPIE [DMS 84]	- Code de programme source : procédure/programme en exécution - Déclaration des variables - Valeurs des variables - Exécution immédiate des instructions	- Editeur dirigé par la syntaxe - Env. interactif de prog. Pascal - Demon : moniteur d'événement	
Smalltalk ENV. [Tes 81]	Env. interactif pour programmer en Smalltalk		
Starlite [CoA 86]	Simulation visuelle pour la programmation du système concurrent et réparti		Modula 2 Lilith

Figure 3.2 Env. de programmation, assistance à la production de programmes (fin).

L'analyse de ces expériences permet de dégager deux approches de la visualisation dans un système d'assistance à la production :

- l'utilisation du langage visuel pour présenter un programme : un diagramme NS [Rei 84], [PoN 83], [CIR 83], un organigramme [GIT 84] ou un autre graphique en 2 dimensions. Ceci ne rentre pas dans notre propos parce que notre enseignement ne recourt pas au langage visuel.
- le programme peut être un texte ou un diagramme, mais visualise plusieurs choses pendant l'exécution. La mise au point ou la recherche d'erreurs en sont facilitées. Ceci nous intéresse.

b. Vues multiples sur un programme : le système PECAN

Dans ce domaine, le système PECAN semble le plus riche [Rei 84] : il utilise un système graphique haute-résolution APOLLO dans l'environnement système de l'université BROWN aux Etats Unis. Les informations qu'il visualise concernent la syntaxe, la sémantique, et l'exécution d'un programme. Ses caractéristiques principales sont :

- plusieurs vues de programme sur l'écran, sous différentes formes : le programme (listing, diagramme NS), les données (schéma de la structure de données), la sémantique (arbre de l'expression, le diagramme des types de données, le diagramme des flots de données, la table de symbole), l'exécution (organigramme, pile d'exécution, instruction), le dialogue entrée/sortie. Ces différentes vues sont générées automatiquement par le système et représentées chacune dans une fenêtre.
- plusieurs éditeurs sont disponibles : éditeur dirigé par la syntaxe, éditeur de déclaration, éditeur de *structured flow graph.* L'édition d'un programme peut être textuelle ou graphique. Le système assure la cohérence entre le texte du programme et son diagramme.
- l'exécution du programme peut s'effectuer en mode pas à pas, avec la vitesse prédéterminée par l'utilisateur, en marche avant ou en marche arrière.
- la mise à jour des vues pendant l'exécution est utile pendant la phase de recherche d'erreurs.

L'évolution du projet prévoit l'intégration du système PECAN dans BALSA, système d'animation d'algorithme que nous détaillons plus loin (§3.2.3).

c. Visualisation de structures de données : INSENCE et GRAPHTRACE

Quelques logiciels s'intéressent uniquement à la visualisation des structures de données. Ils visualisent l'évolution des valeurs de variables et aident aussi à la recherche d'erreurs.

Les langages de programmation évolués permettent aux programmeurs de définir des types complexes. Cependant très peu de langages donnent la souplesse de visualisation des types complexes qui facilitent la mise au point, l'observation et la documentation du programme [Mye 83]. INSENCE, système de visualisation, génère automatiquement les images des structures de données d'un programme écrit en MESA, dans l'environnement de programmation CEDAR. Les identificateurs et les valeurs des informations traitées (variables simples) sont présentés textuellement. Pour les types complexes : un rectangle représente un enregistrement ou un tableau, une flèche représente un pointeur. Une même structure peut avoir plusieurs formats de visualisation différents. L'utilisateur définit les variables du programme qu'il souhaite visualiser, et INSENCE visualise leurs valeurs en temps réel sans modification du programme. L'utilisateur sélectionne, déplace, efface, modifie la taille de la visualisation en mode interactif.

La réalisation la plus complexe du système INSENCE concerne la visualisation d'une liste chaînée. Dessiner un pointeur n'est pas difficile, mais comment déterminer l'endroit où afficher l'enregistrement pointé ?
Trois techniques d'affichage sont possibles :

- l'utilisateur spécifie la position de chaque enregistrement pointé. Cette technique est la plus simple à programmer, mais lourde pour l'utilisateur si la liste est très ramifiée.
- la position de chaque objet pointé est calculé par rapport à la position de l'objet père sans respecter la disposition des objets existants. Ceci nécessite un

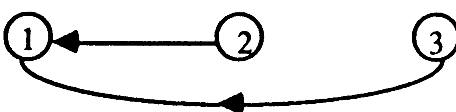
algorithme de placement automatique. Ainsi un nouveau pointeur peut traverser un ancien objet.

- l'écran est géré par le système comme une mémoire à deux dimensions et l'enregistrement est affiché dynamiquement. Cette technique permet l'utilisation maximale de l'écran et d'éviter toute superposition. Mais la performance du système est faible car la gestion de mémoire à deux dimensions nécessite un algorithme complexe.

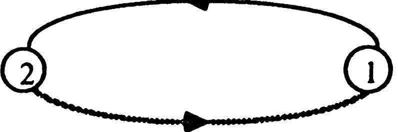
INCENCE combine les deux premières techniques : l'utilisateur définit le rectangle, et ensuite le système génère les champs, les pointeurs et les successeurs.

Le système GRAPHTRACE [GKS 83] visualise des structures de type *"record"* dynamique en Pascal et la relation entre les éléments. L'utilisateur peut donc "voir" ces structures au niveau conceptuel (graphe orienté, arbre). Ce système interactif permet à l'utilisateur de définir plusieurs combinaisons de pointeurs, négliger quelques enregistrements, sélectionner un enregistrement ou imprimer la structure dans un format proche de sa déclaration.

Le système se compose de trois modules : un précompilateur pour traiter les commandes d'affichage de GRAPHTRACE, un module de calcul des coordonnées pour l'affichage et un module pour tracer le dessin. Les nœuds colinéaires sont difficiles à visualiser car les arcs sont superposés. Dans ces cas, le système les affiche comme suit :



Cas I : Nœud 1,2,3 sont colinéaires.



Cas II : Nœud 1 pointe sur nœud 2 et réciproquement. Les deux pointeurs ne sont pas de même type.

Figure 3.3 Représentation des nœuds colinéaires.

3.2.2. Animation de programmes

L'animation de programmes est une forme de visualisation : l'utilisation de la technologie graphique interactive, la conception graphique et la typographie, l'animation, la cinématographie mettent en valeur la présentation du programme afin de le comprendre. La visualisation de programmes est liée mais distincte de la programmation visuelle. La visualisation dans l'animation de l'algorithme doit non seulement présenter la structure des données mais en plus exprimer les opérations effectuées.

Il s'agit donc de deux visualisations : l'exécution dynamique du programme et la modification dynamique des données. Il est impossible de comprendre un programme par l'exécution dynamique instruction par instruction. La visualisation des opérations sur les structures de données exprimée par la modification dynamique des données permet de mieux comprendre le programme. C'est la raison pour laquelle ce type de logiciels se concentre surtout sur la visualisation des structures de données. Plusieurs vues montrent mieux l'exécution, mais les images dans ce domaine restent conventionnelles : arbre, graphe, liste ont la représentation classique que l'on connaît.

Nous distinguons aussi ce type de logiciel d'autres didacticiels "classiques" de programmation, par exemple :

- l'apprentissage d'un langage comme LISP Tuteur [Pir 86], [AnR 85],
- le système PROUST, un détecteur de recherche d'erreurs non syntaxiques qui aide l'étudiant dans la programmation en Pascal [Sel 88],
- le système BRIDGE, un tuteur intelligent pour la programmation en Pascal [Sel 88],
- les cinq didacticiels de programmation observés à La Cité des Sciences et de l'industrie [Arc 86c].

Ces didacticiels "classiques" ne sont pas conçus pour comprendre l'algorithme à travers la visualisation de son exécution, même s'ils visualisent des images.

Nous présentons ici diverses expériences dans lesquelles on trouve l'animation ou la visualisation d'algorithmes à l'aide d'un compilateur général et de modules de visualisation. La figure 3.4 en résume les caractéristiques (quelques références ne sont pas détaillées dans le texte) :

Système Référence	Sujet de Visualisation	Utilisation	Langage Machine
[Hop 74]	- Adressage dispersé	- Pour enseigner	Film généré par ordinateur
[Bae 75] [Bae 81]	- Tris	- Film couleur+son	
[MTT 83]	- Parcours d'arbre - Algo. Boyer-Moore - B-arbre - Algo. ramasse miette - Adressage dispersé - Tris	- Visualisation	IBM-PC 24x80 caractères
BALSA-I [Bro 84] [BrS 84] [BrS 85]	- Les programmes du livre [Sed 83]	- Dans le cours - Activités libres	Smalltalk APOLLO 1042 x 800 bitmaps couleur
BALSA-II [Bro 87] [Bro 88]	- Idem BALSA I	- Idem BALSA I	LightspeedPascal Macintosh
[Owe 85]	- Arbre	- Bibliothèque à appeler par le programme d'étudiant	Turbo Pascal IBM-PC 24 x 80 caractères.
[LoD 85]	- Drapeau hollandais - Producteur consommateur - Tris internes - Huit reines	- Visualisation d'algorithme	Smalltalk Ecran couleur

Figure 3.4 Animation de programmes.

Système Référence	Sujet de Visualisation	Utilisation	Langage Machine
[Ram 85]	- Liste chaînée	- Manipulation de liste du LISP	IBM-PC-XT 24 x 80 caractères
[AuL 86]	- Tableau, pile, file - Arbre - Liste chaînée	- Visualisation - Bibliothèque à appeler par étudiant	Pascal IBM-PC 24 x 80 caractères
[BaB 86]	- Tableau - Pile, file - Liste chaînée - Arbre, graphe	- Autoformation	BASIC IBM-PC 24x80 caractères
[BMS 86]	- Tris internes - Récursivité : factorielle recherche dichotomique tours de Hanoï coefficients binomiaux - Liste chaînée - Arbre binaire - Arbre B - Tris externes - Gestion mémoire - Pile, file - Adressage dispersé - Fusion des fichiers - *CPU scheduling* - Mémoire virtuelle	- Visualisation d'algorithme	IBM-PC 24x80 caractères
[Kem 86]	- Programme - Traces de variables	- Edition de programme Pascal - Observation	Gamme des micros
CABTO [BaK 87]	- Parcours d'arbre : - préfixé - infixé - postfixé - Insertion (arbre) - Suppression (arbre)	- Outil d'enseignement	IBM PC
VISAL [Gia 87]	- Animation d'exécution (sujets ne sont pas précisés) - Visualisation des variables	- Enseignement - Exercices de recherche d'erreurs	

Figure 3.4 Animation de programmes (suite).

Système Référence	Sujet de Visualisation	Utilisation	Langage Machine
IPEX1 [LeM 87]	- Affectation - Entrée/Sortie - Conditionnel - Boucle *"while"* - Boucle *"repeat"* - Boucle *"for"* - Procédures et fonctions	Visualisation dynamique des instructions en Pascal et traces des variables	IBM PC Turbo Pascal
[Max 87]	Opération sur - pile - appels récursifs - algorithmes de tris - représentation chaînée	- Démonstration dans la classe - Observation en libre service	IBM PC (Turbo Pascal + Turbo Graphics)
[AuL 88]	Préprocesseur pour générer la visualisation - tableau - pile, file - arbre - liste chaînée	- Moniteur de programme pour la mise au point	Pascal
APEX1 [MBW 89]	- pile- - file - liste chaînée - récursivité - recherche et tri - parcours d'arbre	- Visualisation des structures de données	IBM PC Turbo Pascal

Figure 3.4 Animation de programmes (fin).

a) Mincy, Tarp et Tai constatent que l'enseignement traditionnel de la programmation (cours, travaux pratiques) n'est pas efficace, surtout pour les algorithmes complexes [MTT 83]. La visualisation de l'algorithme permet à l'élève d'observer le processus effectué par cet algorithme, en l'exécutant pas à pas. Les auteurs ont réalisé et expérimenté deux modules pour leur enseignement des structures de données (le parcours de l'arbre binaire et la reconnaissance de motif). L'élève travaille dans trois modes : démonstration, pas à pas, question-réponse. La visualisation dans ce système est textuelle. L'expérimentation portait sur 2 groupes d'étudiants distincts : un groupe suit la méthode d'enseignement traditionnel, l'autre utilise ce système. Le résultat montre que ce système

améliore la qualité et la performance de l'enseignement. Les auteurs prévoient alors l'implémentation d'autres modules : ramasse-miette, adressage dispersé, tris, B-arbres. Ils suggèrent même que la production des logiciels d'enseignement soit intégrée dans les cours de génie logiciel.

b) **Rambally** présente un système interactif pour la manipulation de liste chaînée, écrit en LISP sur IBM-PC-XT [Ram 85]. Ce système s'applique au cours de programmation et à la mise au point du programme. L'écran est divisé en 5 zones : menu, visualisation graphique, interaction, algorithme et portrait d'un individu de la liste. L'algorithme (limité à quelques instructions) qui manipule la structure s'affiche dans la zone algorithme. La zone de visualisation affiche en temps réel l'image dynamique de la liste chaînée. La représentation graphique est classique : le rectangle matérialise l'objet de la liste, le texte sa valeur et les flèches les différents pointeurs.

c) **London et Duisberg** décrivent une expérience sur matériel haut de gamme pour visualiser des programmes et des algorithmes en créant les vues correspondant à son exécution [LoD 85]. Les animations des programmes présentent les images des structures de données dans les niveaux d'abstraction appropriés à chaque programme. Selon les auteurs, la représentation d'image mémoire est inadaptée pour montrer l'abstraction. Les auteurs dessinent des schémas, des diagrammes, des images, des esquisses conventionnelles.

Dans un premier temps, ces chercheurs développaient en PASCAL. Ils ont ensuite utilisé SMALLTALK car le modèle MVC (*"Model, View, Controller"*) de ce langage orienté objet est mieux adapté pour créer l'animation d'exécution de l'algorithme.

Dans cet article les auteurs résument l'application de ce modèle à leur système d'animation :

- représentation de la structure de données comme "un objet",
- l'algorithme comme "une méthode" exécutée sur cet objet,
- la visualisation d'objet comme la vue.

On y trouve les trois éléments de base d'une animation. En ajoutant des modules de visualisation de ces vues du MVC, les auteurs implémentent un système d'animation d'algorithme. L'insertion des points de vue dans un algorithme est équivalente à la mise en place des assertions dans l'activité de programmation.

Les trois vues importantes d'un programme sont :

- la représentation graphique de la structure de données,
- l'état initial et l'état final du programme : l'image de deux états différents est plus claire que l'évolution continue sur une seule image,
- la transition "douce" et continue des états.

Des animations de l'algorithme de la recherche d'une chaîne de caractères de Hunt et Szymanski, et du modèle de producteur et consommateur d'un moniteur du système d'exploitation sont illustrées en détails. Les autres algorithmes animés sont décrits brièvement : plusieurs algorithmes de tris (en adaptant la représentation de tableau dans le film de [Bae 81]), l'algorithme du drapeau hollandais [Dij 76], l'algorithme des huit reines [Wir 76].

Les points forts issus de cette recherche sont :

- les modules de visualisation qui s'enrichissent tout au long du projet (les anciens modules sont réutilisables dans l'animation suivante),
- l'approche expérimentale et itérative est inévitable pour l'animation de l'algorithme.

d) Le système IPEX1 [LeM 85] offre 7 modules de visualisation dynamique des instructions des programmes Pascal sur IBM PC : affectation, instruction entrée/sortie, instruction conditionnelle, boucle *while*, boucle *repeat*, boucle *for*, procédure et fonction.

Le système visualise le programme Pascal, le résultat et les traces des valeurs des variables sur l'écran. Les programmes sont figés, mais les valeurs des variables sont diversifiées.

e) Dans **[AuL 86]**, les auteurs présentent les modules de visualisation dynamique des structures de données (tableau, pile, file, arbre, liste chaînée), écrits en PASCAL. Ce système sert à l'enseignement des primitives manipulant les structures de données. L'élève manipule la structure en tapant des touches de commande. Les procédures de ces primitives sont stockées dans un fichier, qui peuvent ensuite être invoquées dans le programme d'application de l'élève (par exemple : dans le programme de tri du tableau, l'élève insère les instructions définies dans ce fichier pour visualiser le tableau de son programme. Ce système sert aussi comme outil de déboguage pour le programmeur.

La représentation des structures de données dans ce système est classique, comme les dessins dans les livres de programmation. Les valeurs sont représentées sous forme de texte. Un préprocesseur pour produire ces structures est présenté dans [AuL 88].

f) Dans **[BMS 86]**, les auteurs proposent un modèle d'un système de production des logiciels pour l'enseignement de l'informatique : les étudiants de dernière année sont des ressources potentielles pour la conception, la réalisation, la validation et le raffinement de qualité des didacticiels de l'informatique. La fabrication des didacticiels fait partie de l'enseignement lui-même : travaux pratiques, projet du cours de génie logiciel. Les étudiants du deuxième cycle sont de "bons" auteurs pour les didacticiels du premier cycle (ils maîtrisent le sujet,

car ils étaient élèves de cet enseignement). En même temps, en produisant les pièces du didacticiel, les étudiants approfondissent le sujet et créent un environnement réel dans le domaine du génie logiciel.

Ce système de production de didacticiels a été expérimenté à l'université de NORTHRIDGE. Un projet, intitulé ***"Visible Algorithms"***, a démarré pour produire des didacticiels d'informatique exécutables sur des matériels simples : un mini ordinateur, des écrans simples (24 lignes - 80 colonnes). Chaque module de ce projet contient 3 ou 4 leçons, d'environ 20 minutes chacune.

7 prototypes sont en phase de test et de raffinement :

- Tris internes : par échange, par sélection, à bulle, tri de Shell.
- Récursivité : factorielle, recherche dichotomique, tours de Hanoï, coefficients binomiaux.
- Liste chaînée : recherche, insertion, suppression.
- Arbre binaire : recherche, insertion, suppression.
- B-Arbre : recherche, insertion, suppression.
- Tri externes : *simple selection, replacement selection, natural selection.*
- Gestion mémoire : compression, *coalesce buddy blocks, first fit and best fit allocation .*

6 autres prototypes sont en phase de conception :

- Pile : insertion et suppression d'une pile statique.
- File : insertion et suppression d'une file circulaire.
- Adressage dispersé : fonction hashage, *linear probe*, technique de résolution pour l'espace de débordement par chaînage.
- Fusion : *balanced N-way, optimum N-way, polyphase merge.*
- *CPU scheduling : FIFO, Selfish round robin, multiqueue.*
- Mémoire virtuelle.

g) Le livre **[BaB 86]** présente une vingtaine de programmes d'animation d'algorithmes en BASIC. Les sujets traités sont le tri, la recherche dans un

tableau et dans un répertoire hiérarchique, les tableaux et les fichiers, le traitement séquentiel, les piles, les files, les arbres, les graphes. La visualisation reste classique : la plupart des représentations sont textuelles.

h) Un système de visualisation de programmes Pascal et des structures de données sur une gamme de micro ordinateurs est présenté dans [Kem 86]. Ce système :

- visualise la structure de contrôle, le programme source et les variables,
- met en évidence les valeurs non initialisées,
- utilise un langage de programmation standard,
- fournit un éditeur évolué.

La visualisation de la structure de données ne concerne que l'affichage des noms des variables et de leurs valeurs, sans image graphique. Cette représentation textuelle est affichée à côté de sa déclaration en Pascal sur l'écran.

i) Owen présente la visualisation des piles, files, graphes, et arbres. Les procédures de visualisation (ou plutôt d'affichage) sont écrites en TURBO PASCAL sur IBM PC [Owe 86]. Les étudiants peuvent appeler ces procédures de visualisation pour vérifier leurs programmes. Les représentations des données sont classiques, les valeurs sont présentées en texte.

j) Maxim et Elenbogen constatent la pénurie de logiciels d'enseignement de la programmation [MaE 88]. Ils notent l'intérêt d'une visualisation dynamique des opérations sur la structure de données pour une meilleure compréhension. Les logiciels cités sont réalisés en Turbo Pascal+Turbo Graphix Toolbox sur IBM PC et en Mac Pascal sur Macintosh.

Chaque logiciel visualise sur l'écran :

- la structure logique des données,
- la représentation physique,
- le texte des programmes,
- le changement dynamique des valeurs.

Ces logiciels sont destinés aux enseignants, et aussi aux étudiants pour l'observation individuelle. Les opérations de la structure de pile, les tours de Hanoï, quelques algorithmes de tris (on ne précise pas lesquels) et la structure chaînée ont été visualisés. La visualisation des files, arbres et de l'adressage dispersé sont en cours de développement.

k) Le système APEX1 [MBW 89] offre 6 modules de visualisation des structures de données et des algorithmes :

- pile (empiler et dépiler un élément),
- file (vider, supprimer et insérer un élément, tester l'état de la file : vide ou pleine). La file est implémentée sur un tableau et sur une liste chaînée,
- liste chaînée (insérer, supprimer, allouer et libérer un élément). L'étudiant peut manipuler une liste des valeurs entières, limitée à 5 éléments,
- récursivité (fonction factorielle, calcul de x^y, suite de Fibonacci, recherche dichotomique),
- recherche (séquentielle et dichotomique) et tri (tri à bulles, tri rapide, tri par tas),
- parcours d'arbre (préfixé, infixé, postfixé).

3.2.3. Un système d'animation d'algorithmes : BALSA.

Si un logiciel de visualisation de programmes est analogue à un didacticiel, un **système d'animation d'algorithmes** est lui un système auteur particulier. La

production du logiciel d'assistance à la compréhension de l'exécution des programmes par la visualisation peut être facilitée par un système d'animation d'algorithmes.

Pourquoi distinguons-nous cet environnement des outils classiques en EAO ? Parce que la visualisation d'un programme en cours d'exécution ne peut pas être traitée de la même façon que des didacticiels classiques. Habituellement, un didacticiel enchaîne des pages écrans plus au moins indépendantes et prédéfinies. Tous les enchaînements possibles sont présentés par un graphe déterministe. Les didacticiels de visualisation de programmes sont guidés par l'exécution même du programme sujet d'étude, et il est souhaitable d'avoir des outils de visualisation différents selon la nature des données fournies par l'étudiant. La succession d'images-écran donne des successions d'images innombrables et imprévues avant l'exécution réelle. Les valeurs sont modifiées à l'exécution et dépendent de données fournies à l'extérieur du didacticiel.

Le film réalisé par notre équipe sur la visualisation du tri par minimum à l'aide du logiciel VideoWorks [Arc 88c] illustre l'exécution sur un seul jeu de données. Ce jeu de données a été choisi soigneusement pour proposer un éventail maximum de possibilités, sachant qu'il est impossible de montrer tous les phénomènes avec un seul jeu de données. Les auteurs ont prévu toutes les images-écran à partir d'une exécution préalable de ces données. Ce film s'apparente donc à un didacticiel classique mais c'est une approche nécessaire vers les didacticiels plus dynamiques, qui visualisent à l'exécution n'importe quelles données.

Comme en EAO traditionnel où l'environnement de production et l'environnement élève sont différents, un système d'animation d'algorithmes fournit des **outils différents** selon le public utilisateur. Pour un même programme de référence, **l'enseignant** cherche un moyen d'animation et **l'étudiant** recherche des moyens d'observation.

Deux systèmes d'animation de programmes sont conformes à cette démarche : le système ANIMUS et le système BALSA.

Le système ANIMUS [Dui 88] est non seulement dédié à l'animation d'algorithmes, mais aussi à l'animation de n'importe quel phénomène chronologique. Le seul exemple d'animation d'algorithmes décrit concerne les algorithmes de tris simples.

Nous décrivons les systèmes **BALSA I** [Bro 84], [BrS 84], [BrS 85] et **BALSA II** [Bro 87], [Bro 88] plus longuement car ces systèmes sont proches des préoccupations et des recherches du projet ARCADE.

a. Généralités

Le système BALSA ***(Brown University Algorithm Simulator and Animator)*** est une réalisation issue du projet ***"Electronics classroom"*** de l'université de BROWN aux Etats Unis [Van 84]. Ce projet, lancé en juin 1980, est opérationnel depuis l'automne 1983. La salle de cours est équipée pour l'instant de 55 postes de travail Apollo DN300 en réseau. Le nombre de postes de travail à l'université doit atteindre 10000 dans la prochaine décade !

Les algorithmes du livre de SEDGEWICK [Sed 84] sont animés par le système BALSA. La récente réédition de ce livre [Sed 88] inclut certaines vues obtenues à l'aide du système BALSA. Actuellement, les auteurs du livre et du système BALSA travaillent sur l'intégration au sein d'un même produit multimédia (appelé *"exploratorium"*) des outils pédagogiques précédemment décrits (livre et logiciel d'animation d'algorithmes).

Les utilisations pédagogiques sont les suivantes :

- l'enseignant explique son algorithme à l'aide des animations disponibles, en transmettant par réseau les vues sur les postes de travail étudiants.
- en dehors du cours, l'étudiant travaille et observe individuellement.

BALSA I et II sont des systèmes d'animation d'algorithmes, dont les principales fonctionnalités sont les suivantes : exécution parallèle de plusieurs algorithmes, contrôle de la vitesse d'exécution, vues multiples des données, l'effet de zoom de représentation, exécution en arrière, traitement des scripts.

BALSA I est écrit dans un langage graphique orienté objet de type Smalltalk. Le comportement dynamique en temps réel du programme est primordial pour ce type de logiciel : les auteurs ont donc choisi de modifier un langage général pour diminuer le temps de réponse et améliorer sa performance.

BALSA II est plus interactif, écrit pour l'environnement Macintosh. Il utilise la boîte à outils de Macintosh et un compilateur Pascal "taillé sur mesure" : utilisation des procédures réentrantes et imbrication des modules sont par exemple implémentés.

b. Principe d'animation

Dans le système BALSA I, l'animation d'un algorithme nécessite plusieurs intervenants et plusieurs étapes :

- le concepteur (*algorithm designer*) propose un algorithme dont l'animation semble a priori intéressante,
- le concepteur et l'animateur (*animator)* se mettent d'accord sur le planning de visualisation. Ils définissent les points de vue, aux endroits où les changements qui affectent la structure de données sont intéressants. Il est toujours possible d'ajouter des points de vue à des algorithmes déjà animés.
- l'animateur programme le logiciel en tenant compte des images, associées aux points de vue. Le concepteur reprend son algorithme, transmet les informations nécessaires pour le logiciel graphique.

Le résultat de ce travail est un ensemble d'algorithmes et de vues, accessibles par l'élève ou par le réalisateur d'animation (*scriptwriter).*

Le *scriptwriter* réalise ensuite une animation, en mode interactif : il découpe des scénarios d'animation et arrange les vues sur leurs fenêtres. Il n'écrit pas les codes du programme, il n'est donc que l'auteur d'un livre dynamique. Son rôle principal est d'assembler les algorithmes et les vues dans une unité cohérente racontant une "histoire".

Toutes les animations dans le système BALSA I sont préprogrammées. Ce système ne permet à l'élève ni d'exécuter son propre algorithme ni de proposer un jeu de données original.

BALSA II définit également le rôle des différents utilisateurs :

- l'utilisateur (étudiant) obsèrve l'animation,
- le client-programmeur est soit un "*algorithmatician*" qui veut animer un algorithme (déterminer les états d'un algorithme à animer et les générateurs de données), soit un animateur qui travaille plus sur les vues des structures de données,
- le *script author* (enseignant) ou le *script reader* (étudiant) travaillent au niveau du cinéma,
- l'expert du système maîtrise l'implémentation, la maintenance, l'amélioration et la mise au point d'une animation.

L'animation d'un programme est spécifiée selon trois éléments :

- **les points de vue** par les annotations dans la procédure à animer, ce qui permet de visualiser les moments "intéressants". Les moments intéressants sont les états intermédiaires qui montrent le comportement d'un algorithme (analogie dans la programmation : insertion d'ordres d'écriture pour tracer l'exécution en phase de mise au point).
- **la représentation sur l'écran** (la vue) : objets à visualiser et leurs formes, choix des représentations des structures des données et des programmes sur l'écran. Ces vues évoluent dynamiquement, et sont

modifiées aux moments intéressants choisis (analogie dans la programmation : choix des variables tracées).

- **les données** pour montrer les caractéristiques d'un algorithme et les cas particuliers (analogie dans la programmation : jeux de test).

Ces trois paramètres d'animation sont fournis par le concepteur. Les paramètres qui contrôlent l'exécution (vitesse, échelle, etc) sont fournis par l'élève pendant l'observation.

c. Primitives

Les primitives de BALSA I :

- les primitives de gestion d'écran offrent à l'élève toutes les opérations standards d'un système de fenêtrage : ouverture, suppression, déplacement, agrandissement, réduction et superposition. La communication s'établit à l'aide de la souris et des menus déroulants.
- les primitives d'interprétation contrôlent l'exécution de l'algorithme : commencer, terminer, modifier la vitesse, exécuter en avant ou en arrière. Elles gèrent aussi les points d'arrêt dans un algorithme et l'exécution parallèle des algorithmes. Ces primitives enregistrent le résultat de l'exécution, ce qui nous permet de répéter l'exécution ou de comparer ultérieurement les différents résultats.
- les primitives de contrôle permettent à l'utilisateur de sauvegarder ou de réactiver une configuration enregistrée ainsi que d'insérer des scripts.

Le système BALSA II comprend deux modules : le préprocesseur et l'application.

Le préprocesseur transforme un programme annoté pour que ce programme soit compréhensible par un compilateur général. Pour l'exécution, il capture et

traite les informations fournies par le client-programmeur : l'événement, les algorithmes, les générateurs de données et des vues.

Le module application prépare un environnement élève pour exécuter une animation et traiter les scripts.

BALSA I et II soulignent l'importance du script. Le système BALSA II est capable de traiter les scripts plus interactivement. Au niveau le plus élevé, le système permet de créer une animation complète en éditant les scripts. L'enseignant utilise le résultat du travail de création des *scripts* pour enseigner, l'étudiant l'utilise pour observer dans le cours ou en accès libre. Le scripting permet à l'enseignant de donner le cours en mode *"broadcast"* et les étudiants peuvent s'échanger leurs "vues".

d. Fonctionnalités d'observation :

d1 différentes vues sur plusieurs fenêtres :

Les systèmes BALSA I et II permettent l'exécution d'un programme sur plusieurs "vues" (images) simultanées. Les étudiants peuvent manipuler les fenêtres et déterminer l'échelle de représentation. Une seule vue n'est pas suffisante [BrS 85]. [Bro 87] définit l'importance de la visualisation dynamique de l'algorithme selon : le contenu, la pérennité, la transformation.

La visualisation d'une l'information varie du mode direct au mode synthétique. La visualisation directe est purement déduite de la structure de données sans interprétation. Par contre, la visualisation synthétique ajoute des informations supplémentaires car elle peut visualiser les opérations qui modifient les données.

La pérennité est exprimée par la valeur courante ou l'histoire (traces des valeurs). L'histoire est importante pour comprendre les opérations faites par l'algorithme.

La transformation est visualisée de la façon discrète ou continue. La visualisation discrète consiste à effacer l'ancienne valeur et réafficher la nouvelle valeur. La visualisation continue, plus difficile à réaliser sur le plan technique, permet de montrer une transition "douce" et continue.

Les auteurs constatent qu'il est toujours nécessaire de représenter d'abord la "petite" structure de données dans sa représentation classique (comme dans le livre). Cependant, il est intéressant d'observer les caractéristiques d'un même algorithme sur un grand nombre de données. La visualisation d'un grand nombre de données nécessite une création d'abstraction de vues car la surface de l'écran est limitée : ces systèmes présentent diverses vues (d'un grand nombre de données) originales. Le dynamisme de l'algorithme est aussi un facteur déterminant pour la représentation (par exemple, selon la méthode de tri, ils visualisent le tableau différemment).

d2 **exécution simultanée :**

Les systèmes BALSA I et II montrent l'histoire de l'exécution et la visualisation simultanée des exécutions : plusieurs algorithmes sur une même structure de données ou un algorithme sur des données différentes. L'exécution simultanée des algorithmes permet à l'étudiant de comprendre leurs caractéristiques et de comparer leurs performances.

d3 **visualisation des instructions :**

L'utilisateur peut observer l'exécution des instructions de programme ligne par ligne en décidant d'ouvrir une fenêtre de code. A chaque appel de procédure, une nouvelle fenêtre s'ouvre sur l'appelant. Les concepteurs du BALSA I ont expérimenté plusieurs systèmes d'affichage de fenêtres, et constatent l'efficacité de ce modèle d'affichage pour la lecture du programme et le déboguage.

d4 **contrôle de la vitesse :**

Les algorithmes s'exécutent en fonction de la vitesse donnée par l'utilisateur.

d5 **exécution en marche avant ou en marche arrière :**

Le système BALSA II ne permet que l'exécution en marche avant.

d6 **disponibilité des générateurs de données** (uniquement dans BALSA II).

L'expérience de l'utilisation de ce système montre une utilisation efficace de l'animation dynamique dans l'enseignement de la programmation, car il est lié à la nature dynamique du programme lui même. En plus de l'enseignement de programmation, BALSA contribue aussi à l'enseignement des autres disciplines (mathématiques, neurologie) et à la programmation système.

En réalité, l'utilisation de ce système pour animer un algorithme n'est pas facile. La complexité du système exige des compétences en programmation et en environnement système. Un étudiant ne peut donc pas programmer dans cet environnement car celui-ci n'a pas la simplicité d'un interpréteur.

3.3. Synthèse

Nous avons étudié différents types :

- de représentations (par image, textuelle),
- d'interactions homme-machine, conséquences de la représentation,
- de logiciels qui "visualisent" : assistance à la production de programmes ou l'animation de programmes (cf. Figure 3.2 et 3.4).

Nous retenons la manipulation directe d'objets et la visualisation graphique dans nos logiciels car le système Macintosh avec lequel nous travaillons généralise ce principe.

Nous chercherons plusieurs façons de visualiser les objets de programme sans oublier la représentation habituelle, car la représentation dans le domaine de la programmation est limitée à une convention (classique). Toutes les références citent la visualisation classique des structures de données, seul les systèmes BALSA I et II utilisent différentes visualisations graphiques "non-classiques".

Plusieurs publications citées précédemment montrent l'efficacité de l'utilisation de l'animation des programmes dans l'enseignement de programmation. Quelques animations sont déjà expérimentées et validées. Une qualité graphique insuffisante n'empêche pas l'obtention de résultats positifs. La plupart des systèmes travaillent directement avec un compilateur général. BALSA est le seul système qui prépare son environnement particulier dans lequel les rôles et la catégorie des utilisateurs sont bien définis.

Il est aussi intéressant de fournir aux étudiants en programmation un environnement visuel, qui facilite la mise au point de leurs programmes. La facilité de mise en place du programme permet de mettre en évidence les erreurs de structuration donc d'analyse.

Enfin, quel type de logiciel de **visualisation** sera réalisé dans le laboratoire ARCADE ?

Nous pensons à deux thèmes principaux : l'assistance à la production (car nous formons des programmeurs) et à la compréhension (car en ce moment nos étudiants apprennent la programmation). Les deux sont complémentaires en enseignement, donc une approche parallèle semble idéale.

L'un des objectifs principaux du laboratoire ARCADE est de proposer aux étudiants en programmation des **activités différentes de celles d'écriture de programme**. Nous savons que les logiciels d'assistance à la compréhension sont utiles mais nous ne disposons d'aucun sur Macintosh. Nous réaliserons des logiciels d'assistance à la compréhension des programmes en priorité. Avec cette décision, nous nous sommes heurtés encore à des choix de réalisation :

- soit réaliser d'abord un système d'animation d'algorithmes (outil de production) pour pouvoir produire vite des animations de programmes.
- soit réaliser directement des animations de programmes avec un compilateur général, sans caractéristiques spéciales.

Nous choisissons le deuxième axe de développement et de recherche.

Nous résumons dans la figure 3.5 le thème "programme + visualisation". Dans cette figure, nous avons situé notre travail par des flèches en gras.

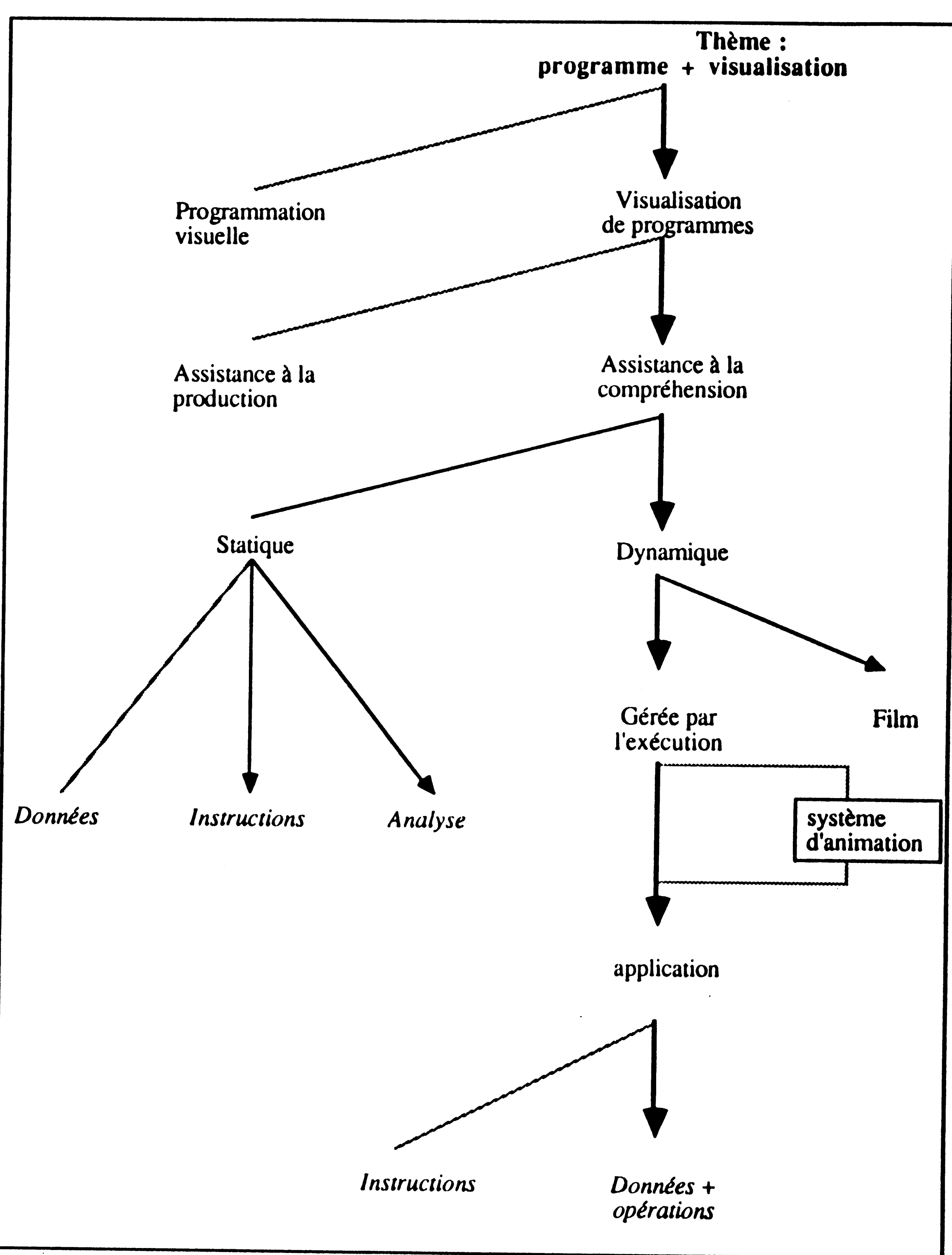


Figure 3.5 Résumé du thème "Programme + Visualisation".

Deuxième partie

COMPTE RENDU DE REALISATION

4. Conception et description des logiciels réalisés
5. Premier bilan pédagogique
6. Aspects techniques

4. Conception et description des logiciels réalisés

Dans la première partie de ce chapitre, je présente la conception des logiciels réalisés : les traits caractéristiques, la méthode de production et les sujets traités. Ensuite, je décris le rôle de la visualisation pour chaque logiciel.

Remarques :

- Je suggère au lecteur, s'il en a la possibilité, de visionner les logiciels cités avant la lecture de cette partie. En effet, les nombreux aspects dynamiques de mes logiciels les rendent plus intelligibles par leur exécution que par leur présentation écrite.
- Un texte précédé du signe et écrit en gras désigne le nom d'un logiciel, par exemple : **Tris internes.**

4.1. Caractéristiques communes

Il s'agit de construire des applications, accessibles par l'utilisateur directement dans l'environnement Macintosh. Chaque logiciel est accessible séparément sur disquette, ou intégré dans un cadre d'accueil propre au laboratoire ARCADE.

4.1. Fonctionnalités

Conformément à l'esprit du projet ARCADE, l'accent est mis sur l'activité de l'étudiant. Tout est fait pour que les étudiants puissent travailler seuls avec nos logiciels. Ce ne sont pas des logiciels d'autoformation mais plutôt des compléments d'un livre ou d'un cours traditionnel. De plus, ces logiciels servent aussi d'outils de démonstration ou d'illustration d'une exécution.

La figure 4.1 décrit les sujets ayant donné lieu à la réalisation de maquettes.

Sujet	Critères de choix
Tri interne	- richesse des algorithmes et des approches. - visualisation de tableau.
Recherche	- richesse des algorithmes et des approches. - visualisation de tableau.
Récursivité	- l'exécution correspond à une interprétation itérative.
Traitement séquentiel	- traitement de base dans les applications informatiques.
Reconnaissance d'un motif *(pattern matching)*	- algorithmes assez complexes.
Tri topologique	- algorithme complexe. - application riche concernant la gestion de la mémoire. - représentation des pointeurs.
Graphe	- opérations de base sur une structure de données.

Figure 4.1 Choix des sujets.

L'ergonomie et la convivialité sont des principes majeurs que j'applique à la conception des maquettes :

- l'utilisateur intervient par manipulation directe d'objets,
- l'utilisation du clavier se limite à la saisie de texte,
- l'aspect plaisant et ludique doit être présent autant que possible,

- l'observation de l'exécution est active puisqu'une requête élève est toujours possible,
- plusieurs points de vue présentent les algorithmes et les structures de données,
- pour chaque algorithme, l'utilisateur détermine lui-même les données à travers lesquelles il veut observer l'exécution. Il choisit l'un des générateurs de données fournis par le système, ou saisit ses propres données sous forme graphique, sauf pour le texte.
- le décor est standardisé. La même fonctionnalité (le contrôle de la vitesse, les boutons pour commencer l'exécution, l'arrêter, faire une pause) conserve, si possible, toujours les mêmes caractéristiques de forme et de position.

La figure 4.2 résume les fonctionnalités disponibles dans chaque maquette. Le nombre d'étoiles donne une indication qualitative. Les images-écran sont présentées dans l'annexe B. Les préalables sur lesquels nous nous appuyons sont les suivants :

- le contrôle de la vitesse est indispensable à une bonne observation,
- l'exécution pas à pas est nécessaire,
- l'exécution sur différentes vues, simultanée ou différée, est utile,
- l'exécution simultanée des algorithmes et l'effet de zoom sur les vues de données ne nous semblent pas primordiaux,
- le guidage dans la compréhension d'algorithme est présenté simplement par des images statiques sans animation,
- nous offrons une lecture structurée de l'algorithme, même si ce n'est pas une lecture dynamique,
- nous offrons la possibilité d'exécuter l'algorithme à la main pour l'exercice de compréhension de l'exécution.

Etat (Raffiné/Incomplet)	R	R	R	R	R	R	I	I	I	R	I
Fonctionnalités \ Maquettes	Tris internes	Exploration de tables	Pattern matching	Dessins récursifs	Baguenaudier	Les 8 reines	Machines caractères	Compter les "A"	Compter les "LE"	Tri topologique	Parcours de graphes
Observation d'exécution											
Vitesse	*		*	*	*	*				*	*
Arrêt	*		*	*	*	*				*	*
Pause		*		*	*	*				*	*
Pas à pas		*	* *							*	
Vues multiples			* *		* * *	* *				* * *	* * *
Algorithmes multiples	* * *	* *	* *		*						*
Lecture d'algorithme		*	*	*	*	*		*	*	*	*
Guidage d'analyse			*	* * *	*	*					
Exercice		*	*	*	*	*	*	*	*	*	*
Saisie/variation de données	* * *	* * *	* * *	*	*	*	* *	* *	* *	* * *	* * *

Figure 4.2 Récapitulation des réalisations

4.1.2. Méthode de travail

Notre méthode de travail met l'accent sur la production de maquettes [Arc 86a] comme étape importante et efficace dans le cycle de production de nos logiciels.

Une maquette est pour nous :

- un moyen rapide pour concrétiser une idée,
- un générateur d'idées qui suggère des améliorations, des raffinements, la mise en place de nouvelles fonctionnalités, etc...
- un moyen de vérifier la faisabilité technique de certaines hypothèses de travail,
- une source d'expérimentations possibles afin d'évaluer la validité des idées initiales.

Ceci correspond aux caractéristiques du maquettage [Boe 82] : ".... le produit peut être obtenu en développant une maquette du système après une analyse rapide, en le faisant essayer par les utilisateurs et en rédigeant ensuite le cahier des charges définitif ...".

La réalisation d'une maquette exige du matériel et prend du temps. Le temps nécessaire peut-être réduit par l'usage d'outils appropriés. De plus, des outils complémentaires de développement prennent forme au fur et à mesure de la réalisation de la maquette. La première réalisation est lente, puisque les outils doivent être mis au point, mais ensuite le rythme des réalisations s'accélère. Ce processus détermine progressivement les meilleurs outils de production et les met en place.

Nous comptons ainsi réaliser certains composants intéressants, jugés prometteurs, sous forme de maquettes plus au moins finalisées. Les autres seront

réétudiés, modifiés ou rejetés. Actuellement, mes maquettes se situent à plusieurs degrés de développement :

- quelques maquettes ont été raffinées et sont à l'état de logiciels commercialisables (cf. Figure 4.2),
- quelques maquettes restent imparfaites sur le plan technique et esthétique. D'autres sont encore incomplètes puisque quelques fonctionnalités ne sont pas encore implémentées (cf. Figure 4.2). Il sera toujours possible à l'avenir de les raffiner. Le laboratoire ARCADE les accueille tout de même dans cet état, car elles sont suffisamment démonstratives,
- d'autres non décrites ici, sont inachevées ou abandonnées pour deux raisons : le manque de temps ou le rejet après essai d'un sujet jugé moins intéressant à visualiser. Elles traitent de listes chaînées, adressage dispersé, machine-mots, traitement fonctionnel des séquences.

La situation réelle dans la classe n'est pas occultée. En dehors de mon activité de réalisation, j'ai observé les cours de la première année d'une maîtrise d'informatique à l'Université Joseph Fourier de Grenoble concernant les sujets développés dans les maquettes. Les réactions des étudiants, les questions posées, les difficultés rencontrées permettent par leur observation une meilleure compréhension des besoins réels d'outils d'enseignement. Je m'inspire aussi de mon expérience d'enseignement à l'Institut de Technologie de Bandung.

Les outils de production sont des logiciels commerciaux standards sur Macintosh :

- les logiciels MacPaint, MacDraw ou SuperPaint, pour créer les images de base,
- l'éditeur de ressource ResEdit,
- Lightspeed Pascal, comme langage de développement,
- Hypercard, pour intégrer les logiciels dans le laboratoire ARCADE.

4.2. Visualisation et tableaux

La structure de tableau est une structure primitive de bas niveau : elle est l'image de la mémoire. Dans un tableau, il est souvent représenté des structures logiques complexes qui n'ont que peu de rapport avec l'aspect séquentiel du tableau. Par exemple, dans le tri par tas (*heap sort*), on doit "plaquer" sur le tableau une structure arborescente.

La visualisation doit mettre en évidence la structure logique ainsi que la caractéristique des éléments selon le traitement effectué. Par exemple, dans un tri, la caractéristique d'un élément est qu'il est plus petit, égal, ou plus grand. Mais, dans une recherche, cette caractéristique est qu'il est égal ou différent.

4.2.1. Tri interne

Ce sujet est très souvent traité par les groupes qui travaillent sur l'animation d'algorithmes (cf. Figure 3.4).

Les algorithmes de tri sont nombreux, un livre entier [Knu 73] est consacré aux tris et recherches. ** Tris internes** anime des algorithmes de tris présentés dans [Wir 86] : tri par insertion, tri par minimums successifs, tri à bulles, tri shaker, tri de Shell, tri par tas, tri rapide. Nous avons choisi ce livre pour trois raisons : sa notoriété, l'exemplarité pédagogique des algorithmes proposés, même s'ils ne sont pas toujours optimisés, et la possibilité d'étudier immédiatement le rôle d'un tel logiciel comme complément à la lecture d'un livre scientifique.

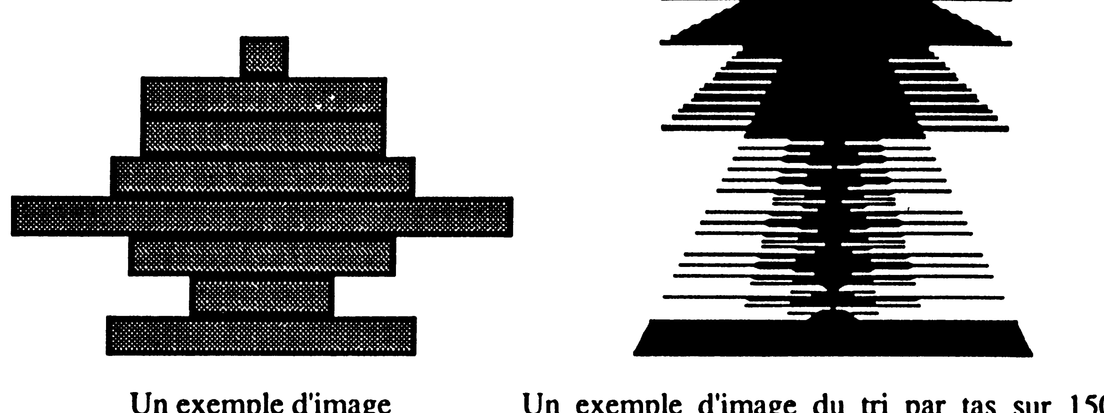
Le logiciel ** Tris internes** n'aborde que l'aspect opératoire du tri. Dans le laboratoire ARCADE, d'autres logiciels accompagnent et complètent ce logiciel : le film gare de triage [Arc 88c] (construction méthodique et aspect cinéma : animation, son), apprentissage guidé d'un algorithme de tri [Arc 88d]

et meccano de tri [Arc 88e] (aspect manipulatoire). Cet ensemble de logiciels formalise différentes approches, parmi lesquelles l'étudiant peut choisir selon ses goûts, ses erreurs, ses doutes, sa façon d'apprendre [Arc 88h].

La représentation visuelle d'un tableau reprend ce qui est proposée dans BALSA (§ 3.2.3). Chaque élément du tableau est symbolisé par un trait d'une longueur proportionnelle à sa valeur. Les déplacements successifs des traits matérialisent l'exécution du tri.

Le logiciel **Tris internes** réunit deux applications :

a) La première commande l'exécution d'un algorithme de tri : l'étudiant choisit un algorithme, la nature des données (aléatoires, ordonnées, inversées, égales ou bien saisies graphiquement par lui-même) et le nombre d'éléments. Les requêtes disponibles sont : la modification de la vitesse d'exécution, la reprise illimitée d'un tri sur un même jeu de données, le changement d'algorithme ou du jeu de données.



Un exemple d'image d'un tri de 8 éléments.

Un exemple d'image du tri par tas sur 150 éléments classés au départ par ordre croissant.

Figure 4.3 Exemples d'exécution de **Tris internes**.

Grâce à la visualisation, des phénomènes difficilement imaginables sont découverts à l'exécution, comme le tri par tas qui détruit et puis reconstruit les éléments déjà ordonnés (Figure 4.3).

J'ai observé, lors des premières expérimentations, deux attitudes principales d'utilisation :

- le travail sur une dizaine d'éléments avec une vitesse de l'ordre d'une opération élémentaire par seconde. Cette observation convient à l'étudiant qui désire comprendre le processus séquentiel de l'exécution dans ses moindres détails.
- le travail en vitesse rapide sur un tableau de 256 valeurs aléatoires (une trentaine de secondes). Cette observation convient à l'étudiant qui désire comprendre dans sa globalité le principe de transformation du tableau.

Ces observations confirment ce qui a été dit par [Bro 87] sur ces deux modes d'observation :

- un nombre réduit de données et une vitesse lente pour comprendre en détail le mécanisme du tri,
- un nombre important de données et une vitesse rapide pour donner une vision globale de ce que fait l'algorithme.

b) La seconde application compare et évalue les performances des algorithmes de tri sur les critères suivants : temps d'exécution, nombre de comparaisons et nombre d'affectations. L'évaluation est présentée par histogrammes. On observe soit différents algorithmes pour un seul type de données, soit un seul algorithme pour plusieurs types de données. Pour éviter un temps d'attente trop long, les évaluations sont pré-enregistrées. L'exécution des 7 algorithmes de tris pour un tableau de 2048 éléments et des données de nature différente (aléatoire, égale, ordonnée, inversée) nécessiterait plus de quatre heures sur MacPlus. Par contre, les tris sont exécutés en temps réel sans visualisation, pour des données saisies par l'étudiant.

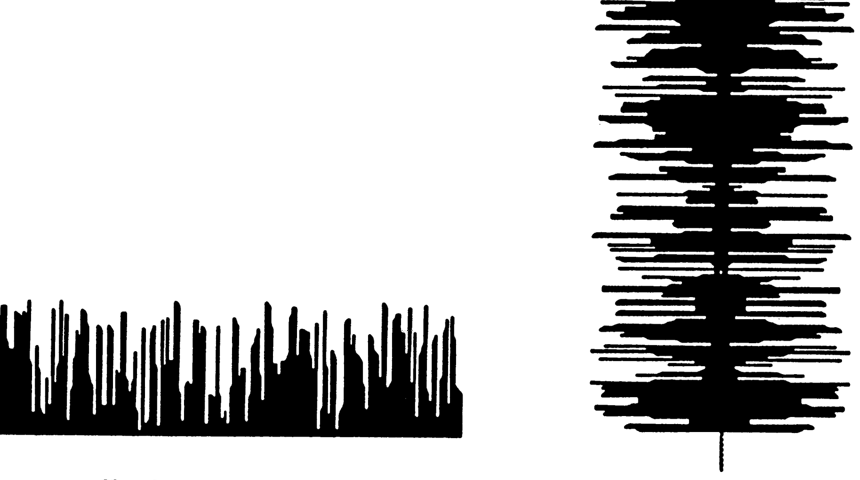
La visualisation semble utile lorsque l'on traite un algorithme de tri itératif. Par contre qu'en est-il des algorithmes complexes basés sur une approche récursive ? Dans ce cas il semble intéressant de se préoccuper plus de la visualisation du raisonnement de conception que de celle de l'exécution.

Voici encore quelques idées inexploitées, mais que nous pensons utiles pour l'avenir : le tri par tas sur un arbre même si ensuite l'arbre est représenté dans un tableau, et le tri rapide par groupe d'éléments traités à chaque appel.

4.2.2. Recherche dans un tableau

Au début, j'ai réalisé une maquette sur la recherche d'une valeur dans un tableau, appelée **Exploration de tables**. Elle correspond au chapitre 1.12 assez court de [Wir 86].

Comme pour **Tris internes**, j'ai choisi pour ce sujet simple, la même représentation des valeurs de tableau que celle utilisée dans le système BALSA [Bro 87]. Dans **Exploration de tables** les traits sont présentés verticalement, sans symétrie axiale. Cette représentation plus simple et moins encombrante, permet de visualiser plus d'éléments sur écran, et est plus claire pour la comparaison.



Exploration de tables. Tri.
Figure 4.4 Différentes représentations d'un tableau.

La même représentation ne produit pas le même résultat visuel puisque, cette fois, la différence de valeurs n'est pas significative. Nous aimerions rendre visuel l'existence de l'élément cherché. C'est un exemple où l'on doit visualiser les états, et non les modifications de valeurs.

Les éléments du tableau peuvent être aléatoires, ou ordonnés en ordre croissant/décroissant, tous générés par programme. L'étudiant peut également saisir son propre jeu de données graphiquement.

Exploration de tables propose l'observation d'exécution et la lecture des algorithmes suivants :

- recherche séquentielle, avec ou sans sentinelle,
- recherche dichotomique dans un tableau ordonné.

La visualisation de l'exécution apporte peu, si ce n'est la réduction extraordinaire du nombre de comparaisons dans la recherche dichotomique.

Dans cette maquette, on n'a pas donné la possibilité de régler la vitesse d'exécution, en pensant que le mode pas à pas pourrait suffire. Les expériences l'infirment : la modification de vitesse pendant l'exécution est une fonctionnalité indispensable d'un logiciel d'observation d'exécution parce que :

- l'exécution en mode continu avec une vitesse unique n'offre pas à chaque étudiant une observation individualisée. En général, un étudiant commence à observer en vitesse lente, et demande ensuite une vitesse plus élevée après avoir compris le principe.
- l'exécution en mode pas à pas ne donne pas une vue globale d'un algorithme car elle n'exprime qu'un pas.

L'extension de la maquette **Exploration de tables** s'est concrétisée rapidement par une nouvelle réalisation sur la recherche de la valeur maximale

et minimale : **Recherches Min et Max**. Ce sujet relève de la même idée que la recherche d'une valeur dans un tableau. Sa réalisation a servi de test pour la possibilité de réutilisation des modules existants.

L'exécution dans **Recherches Min et Max** visualise :

- plusieurs changements de valeurs de min/max,
- la nécessité d'une exploration exhaustive des éléments du tableau, quelques soient les données.

4.2.3. Reconnaissance d'un motif (*Pattern matching*)

Contrairement aux cas du tri ou de la récursivité, peu de didacticiels traitent ce sujet. [MTT 83] anime seulement l'algorithme Boyer-Moore.

La maquette **Pattern matching** propose un travail sur la recherche d'un motif *("pattern")* dans une chaîne de caractères. Deux formes de représentations visuelles sont proposées :

- la première correspond à celle utilisée dans **Exploration de tables**, où chaque caractère est codé et représenté par sa longueur,

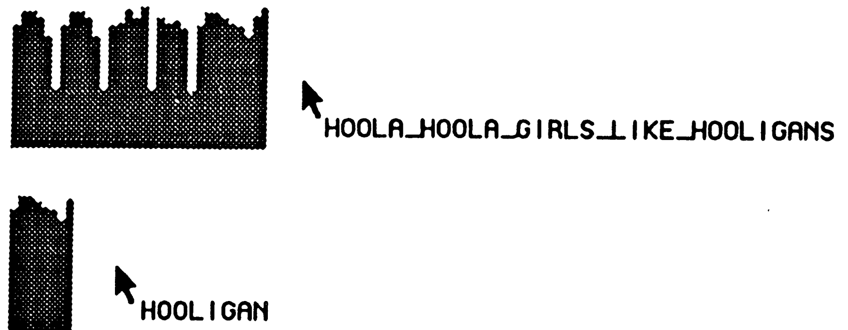


Figure 4.5 La représentation codée de la chaîne de caractères et du motif.

Cette représentation permet à l'étudiant d'observer la vue globale de l'exécution.

- la seconde représente la chaîne de caractères dans sa forme alphabétique. Cette représentation permet à l'étudiant une observation plus détaillée.

Exemple :

HOOLA_HOOLA_GIRLS_LIKE_HOOLIGANS
HOOLIGAN

Figure 4.6 La représentation exacte de la chaîne de caractères et du motif.

La comparaison est visualisée comme dans **Exploration de tables,** caractère par caractère. Il serait intéressant de visualiser "d'un seul coup" la comparaison du modèle sur la chaîne en jouant sur le ton de couleur de l'écran. Cette visualisation permet de mieux observer l'existence du modèle dans la chaîne à un autre niveau. Cette idée, non réalisée actuellement, peut être illustrée ainsi :

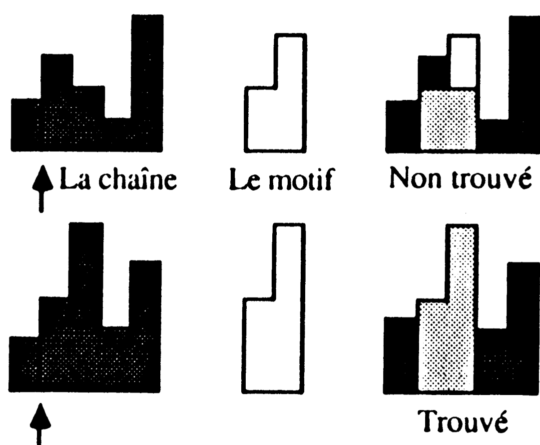


Figure 4.7 Comparaison du motif sans montrer le détail.

La maquette **Pattern matching** permet à l'étudiant de comparer quatre algorithmes de recherches traitant le même problème :

- Séquentiel [Wir 86],
- Rabin-Karp [Sed 83],
- Knuth-Morris-Pratt [KnP 77], [Wir 86],
- Boyer-Moore [BoM 77], [Wir 86].

L'étudiant peut créer ou modifier les données (la chaîne et le motif) sous forme de texte.

La maquette fournit aussi quelques exemples types du motif ("AAAAAB", "ABCABD", "ABCDEF", "ABCDEA", etc) tirés de [Wir 86]. Ces exemples types illustrent les cas particuliers de l'algorithme Knuth-Morris-Pratt.

Malgré la complexité du sujet, la maquette **Pattern matching** est capable de montrer les visions globales de différents fonctionnements de chaque algorithme :

- avancer d'un caractère ou de plusieurs caractères à chaque pas,
- comparer de gauche à droite ou de droite à gauche,
- comparer chaque caractère de la chaîne une ou plusieurs fois.

La visualisation d'exécution permet aussi de donner l'intuition sur l'efficacité et les performances des algorithmes (temps d'exécution et nombre de comparaisons par exemple).

Limites

A cause de la complexité de l'algorithme, l'utilisation isolée de **Pattern matching** semble inadaptée ou insuffisante : le seul aspect opératoire ne permet pas à l'étudiant de maîtriser les algorithmes Knuth-Morris-Pratt ou Boyer-Moore.

Malgré ses limites, ce logiciel reste le seul outil d'apprentissage sur ce thème dans le laboratoire ARCADE. Il est donc nécessaire de le compléter et de l'accompagner par des outils pédagogiques différents pour enrichir et diversifier les activités d'apprentissage.

Les algorithmes Boyer-Moore et Knuth-Morris-Pratt traitent le motif avant d'effectuer la reconnaissance. Ce traitement préliminaire est appelé "compilation" dans [Wir 86], il s'agit de la construction d'un tableau de référence. L'exercice dans **Pattern matching** sur la compilation du motif teste la compréhension par l'étudiant du contenu du tableau de référence, mais n'effectue aucun travail pédagogique sur la méthode de construction de ce tableau.

L'algorithme Knuth-Morris-Pratt est un algorithme "astucieux" et déjà optimisé. La visualisation de son exécution n'est pas claire car les éléments fondamentaux de l'analyse et les détails d'optimisation se situent au même niveau. La réalisation d'un nouvel outil présentant le processus d'optimisation à travers les algorithmes successifs serait un complément intéressant. Ce travail pédagogique sur l'élaboration d'une méthode ne fait pas partie pour l'instant de mes préoccupations.

4.3. Visualisation et récursivité

Qu'enseigne-t-on sur la récursivité ? Quelle est la contribution possible d'une visualisation gérée par l'exécution ?

- **l'analyse récurrente** conduit à l'écriture d'un algorithme, sans se préoccuper de son exécution. Cette partie importante du cours s'illustre difficilement par l'exécution. Mais une bonne visualisation sur l'analyse (qui n'est pas forcement générée par l'exécution), sous forme de dessins, de schémas ou d'animation peut aider la compréhension de l'élève.
- l'exécution de **sous-programmes** récursifs facilite grandement le travail de visualisation et permet l'observation des appels, de la gestion de la pile d'appels et du passage des paramètres.

- les opérations sur les **structures de données** récursives, arbre ou liste, sont intéressantes à montrer. Par exemple la visualisation sur les sous-arbres dans le traitement d'un arbre, car on raisonne sur le sous-arbre et non pas directement sur chaque nœud de l'arbre.
- la **transformation** d'un programme récursif en itératif et les règles de transformation. Cette transformation peut être générée semi-automatiquement [Gir 86]. La visualisation peut mettre en évidence ce processus.

L'enseignement de la récursivité est un sujet délicat et complexe. J'ai trouvé plusieurs articles didactiques sur ce thème, mais très peu parlent de réalisation de didacticiels. Les exemples types sur la récursivité sont peu nombreux. [LiM 88] constate de plus qu'il est difficile de trouver un sujet original pour des travaux pratiques, car les livres fournissent directement les programmes concernant les problèmes récursifs cités.

Le jeu des tours de Hanoï est l'exemple le plus cité dans les ouvrages traitant la récursivité. C'est un modèle exemplaire d'un problème où un être humain ne sait pas exécuter facilement les actions élémentaires (le déplacement d'un disque) à la main. Cet exemple montre bien le rôle de l'analyse récurrente [Bac 86].

Certaines recherches [Gué 85], [LoD 85], [BMS 86], [BaK 87], [MBW 89] conduisent aussi à la production de didacticiels sur ce thème.

4.3.1. Dessins récursifs

Les courbes fractales sont des exemples types souvent discutés, mais peu de didacticiels existent sur ce thème. Les fractales sont plutôt prises pour l'étude de la construction des algorithmes. [LiM 87] utilise les fractales avec la tortue graphique pour introduire la récursivité. [DuG 87] discute la découverte et l'introduction de la récursivité en LOGO dans une classe de 5ème, en prenant les dessins comme la croix, le flocon et l'arbre. Ces courbes sont souvent prises

comme sujet d'étude car le plaisir esthétique que leurs dessins apportent est une source de motivation supplémentaire pour les élèves.

[PrS 88] présente deux méthodes pour générer la courbe Koch (une famille de fractales) : *attracting method* and *repelling method.* L'article porte surtout sur les équations mathématiques des dessins, mais présente aussi les dessins en couleur générés par ces deux méthodes. [Man 84] est un livre dédié aux fractales.

Le logiciel **Dessins récursifs** offre à l'étudiant un certain nombre d'outils pour qu'il découvre, à sa manière, la définition récurrente d'un dessin géométrique et donc le programme récursif sous-jacent.

Dessins récursifs propose des dessins de difficultés variées :

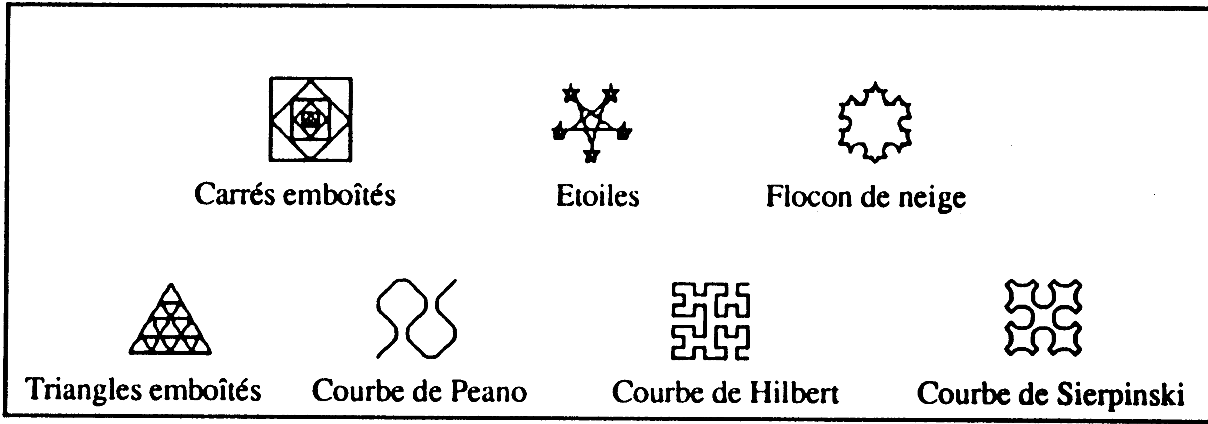


Figure 4.8 Différentes courbes dans **Dessins récursifs.**

Les propositions suivantes résument l'intérêt pédagogique de ces dessins :

- ils sont jolis à observer et agréables par les effets de symétrie générés,
- la variété d'appels récursifs des courbes montrent plusieurs modes de décomposition :
  a. les carrés emboîtés pour une récurrence simple.
  b. les étoiles, comme les carrés emboîtés, avec des appels multiples.
  c. le flocon de neige pour lequel la courbe n'est pas récursive, mais composée de trois segments récursifs.

d. les triangles emboîtés avec deux appels croisés (l'un appelle l'autre).

e. la courbe de Peano, celle de Hilbert ou de Sierpinski : les niveaux successifs ne sont pas superposables.

- ils sont intéressants parce qu'ils illustrent les étapes de l'analyse récurrente : la base peut être difficile à trouver (par exemple Peano), ou bien on peut avoir plusieurs solutions pour la récurrence (par exemple Sierpinski, triangles emboîtés).

4.3.2. Jeu de Baguenaudier

Nous avons déjà présenté ce jeu (§1.1).

Baguenaudier exploite un tableau de booléens. Un anneau accroché est visualisé comme un jeton noir, l'anneau décroché comme un jeton blanc.

Dans cette maquette, le nombre d'anneaux est compris entre 5 et 15 ce qui limite le nombre de combinaisons dans les jeux de données possibles.

L'intérêt de la réalisation du jeu sur l'ordinateur se justifie pour les raisons suivantes :

- mettre à disposition des étudiants un jeu utile et facilement manipulable,
- montrer comme pour les autres maquettes récursives, l'aspect opératoire inaccessible au raisonnement humain,
- permettre de raisonner par récurrence car la valeur de N qui exprime le nombre d'anneaux du jeu est une donnée dont dépend l'analyse,
- comparer des solutions itérative et récursive (le logiciel **Dessins récursifs** ne traite pas la solution itérative). La figure 4.9 schématise l'analyse itérative. La solution récursive est décrite dans la figure 4.10.

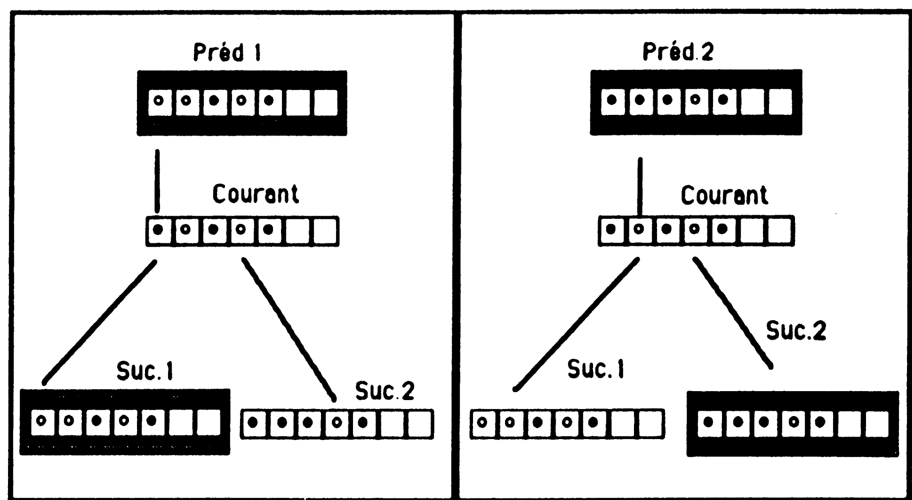


Figure 4.9 Solution itérative du jeu de Baguenaudier.

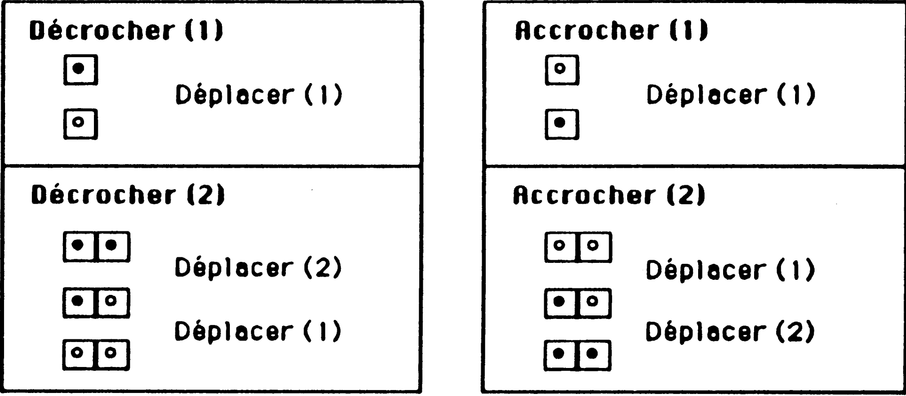


Figure 4.10a Base de la solution récursive du jeu de Baguenaudier.

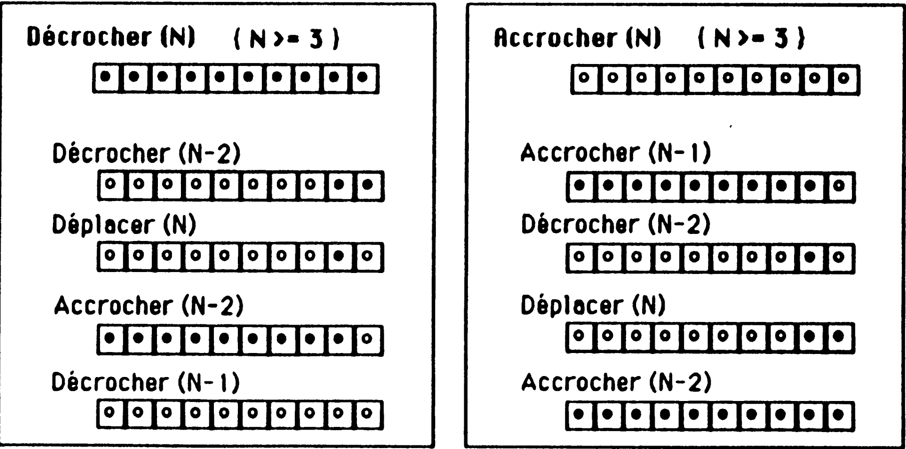


Figure 4.10b Récurrence de la solution récursive du jeu de Baguenaudier.

4.3.3. Huit reines

Le problème consiste à placer 8 reines sur un échiquier en respectant la règle suivante : aucune d'entre elles ne doit se trouver sur la même ligne, la même colonne ou la même diagonale.

Contrairement à la complexité de l'algorithme, l'évaluation de l'exercice dans **Les 8 reines** est facile, elle ne dépend que de l'état courant de l'échiquier. Ce n'est pas le cas dans l'évaluation du parcours de graphe. L'aspect ludique rend cette maquette "plaisante" et "attrayante".
L'analyse (cf. Figure 4.11) met en évidence la structure de données physique choisie dans [Wir 86]. Cette structure de données est simple, mais le choix de cette représentation pour faciliter la comparaison n'est pas intuitif.

Cette maquette visualise dans une seule fenêtre l'exécution de 2 représentations différentes : la structure logique sur l'échiquier et les structures physiques sur les tableaux (Figure 4.12). Ces deux représentations contiennent peu d'informations, donc occupent peu de place à l'écran. Mais l'exécution "simultanée" sur ces vues multiples est inobservable, ce qui confirme et justifie notre décision de fournir pour chaque maquette une fenêtre active par structure.

De plus, pour cet exemple précis, l'exécution est trop longue à observer. La vitesse maximale offre une solution au bout de 5 minutes environ sur Mac Plus, et dans ce cas l'attention prolongée rend la lecture impossible. J'ai donc créé le problème des N reines avec $N \leq 8$.

[LiM 88] utilise le problème de *"AMAZING"* (labyrinthe) comme exercice de travaux pratiques dans son enseignement sur la récursivité. Son animation pourrait être intéressante puisque cet exemple est de même type que les 8 reines.

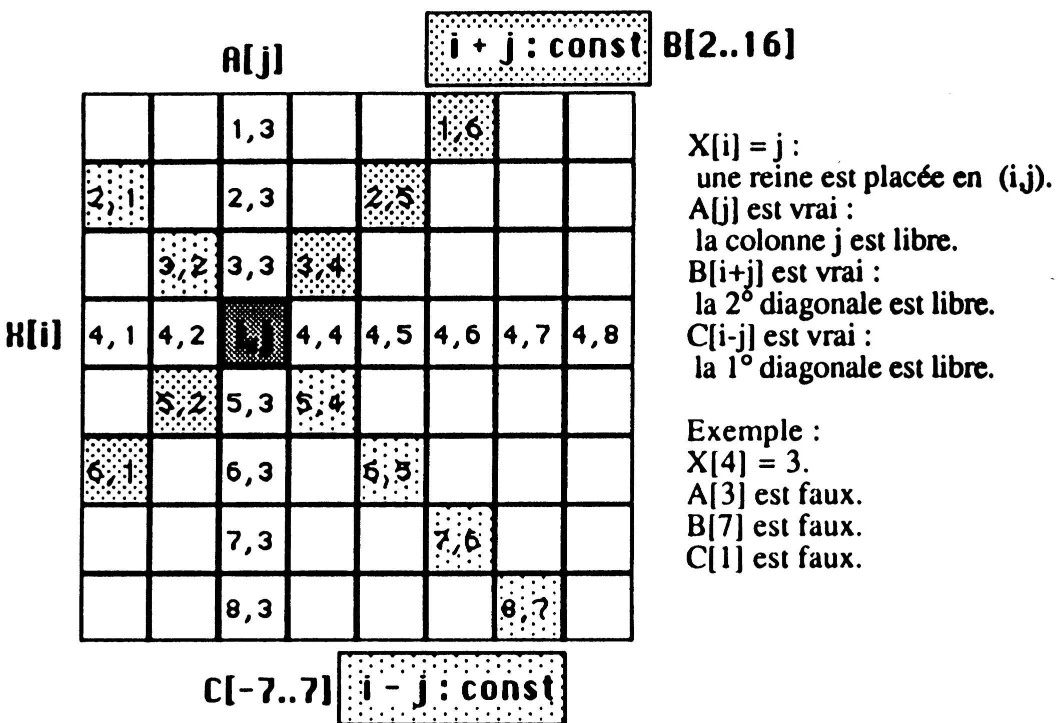


Figure 4.11 Représentations de données du problème des huit reines [Wir 86].

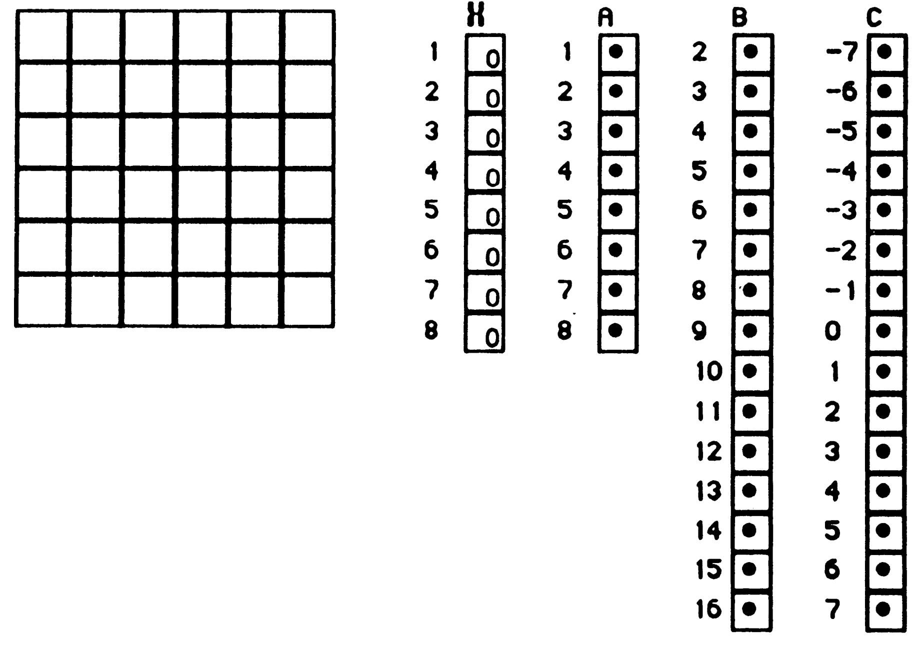


Figure 4.12 Images écran de l'exécution de **Les 8 reines.**

4.3.4. Méthodes de visualisation

Le mode "par niveau" est utile pour montrer l'exécution d'un programme récursif. Dans ce mode chaque élément du dessin est tracé sur un emplacement propre à la valeur du niveau de profondeur. Les traces dynamiques de **Baguenaudier** le montre :

IMAGES	APPEL
	N = 15
	N = 13
	N = 11
	N = 9
	N = 7
	N = 5
	N = 3, N = 1

Figure 4.13 Traces des appels récursifs dans **Baguenaudier.**

Nous proposons plusieurs modèles de visualisation des appels récursifs :

a. La visualisation entière se dessine dans un même espace.

   Les appels récursifs successifs ne sont pas visualisés. Dans ce modèle, seul le changement de valeur ou d'état est visualisé de manière quasi-automatique. C'est le cas du mode "normal" dans **Dessins récursifs**, ou "par déplacement" dans **Baguenaudier**.

Voici l'illustration pour les carrés emboîtés :

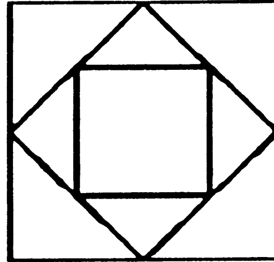


Figure 4.14 Visualisation sans mise en évidence des niveaux d'appels récursifs.

b. Chaque appel bénéficie d'un espace de visualisation.

Cette visualisation met en évidence l'appel et l'effet de chaque appel. C'est le cas du mode "par niveau" dans **Dessins récursifs** :

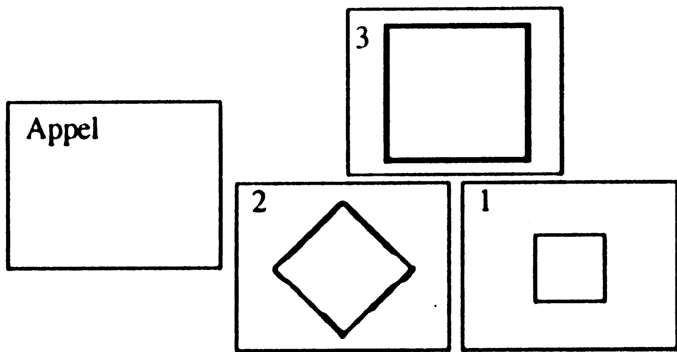


Figure 4.15 Visualisation avec mise en évidence de chaque niveau d'appel récursif.

c. Chaque appel bénéficie d'un espace de visualisation cumulant les effets des appels précédents.

Lorsque l'appel i est fini, la figure produite par l'appel i dans son propre espace est transférée dans la surface d'appel (i-1). **Baguenaudier** utilise ce modèle de visualisation, avec l'exécution en mode "par appel".

Si cette fonctionnalité existait dans **Dessins récursifs**, l'exécution des carrés emboîtés pourrait s'afficher comme suit :

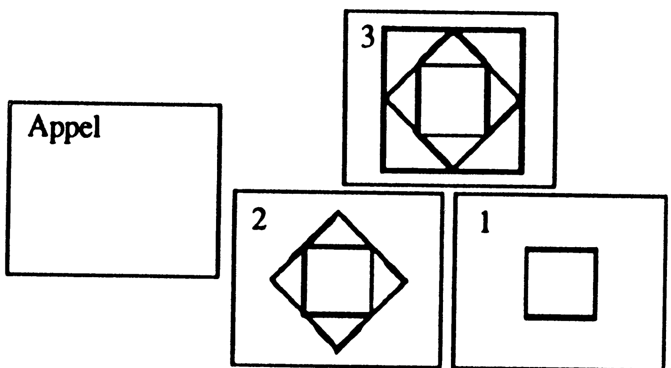


Figure 4.16 Visualisation qui montre chaque figure complète d'un niveau.

Ce modèle de visualisation permet de voir une image complétée à chaque niveau d'appel, ce que ne proposait pas le deuxième modèle.

Les dessins récursifs sont attrayants et incitent à programmer d'autres fractales. Plusieurs expériences menées avec des professeurs du second degré [Pey 88b] l'ont prouvé. C'est la confirmation de la complémentarité de l'outil de programmation avec l'outil d'observation.

D'après des expérimentations d'utilisation, l'aspect ludique rend ces exercices encore plus séduisants. Le système de comptage donne l'enjeu, incite l'étudiant à jouer comme dans les logiciels de jeux sur ordinateur [Mal 81].

L'exercice, lorsqu'il est par niveau, force l'étudiant à raisonner en terme de structure récursive, donc en terme d'analyse, ce qui est indispensable.

En travaillant avec ces logiciels, l'observateur appréhende la notion de temps d'exécution. En exécutant **Baguenaudier** pour N = 15 ou N = 10, l'étudiant sent bien la différence des deux temps d'exécution.

4.4. Visualisation et machines abstraites

J'ai réalisé plusieurs maquettes pour illustrer les machines abstraites du modèle de traitement de séquence (cf. [ScP 88], §2.1.2). Les maquettes **Machines caractères, Compter les "A"** et **Compter les "LE"** sont installées dans le laboratoire.

Machines caractères simule le fonctionnement des machines de consultation et de création ainsi que la mémorisation d'un ensemble très limité de variables.

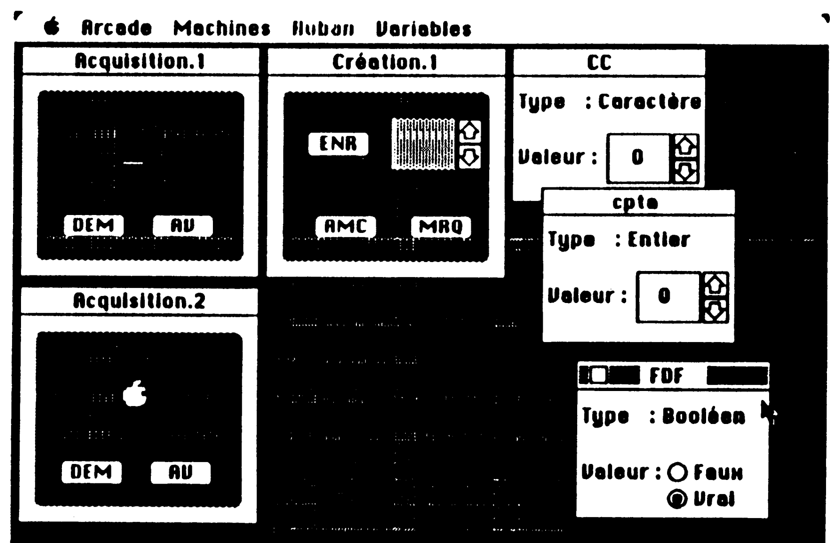


Figure 4.17 Les fenêtres manipulables dans **Machines caractères.**

L'étudiant peut exécuter à la main, en mode libre, des algorithmes simples tels que : compter les "A", compter les "LE", compter les mots, copie d'une séquence, union ou intersection de deux séquences...

Grâce à la fenêtre de traces d'actions, l'étudiant peut découvrir l'itération à partir du compte rendu de l'exécution [§ 1.1]. Le texte donné est généré par le logiciel, ou fourni par l'étudiant.

Les maquettes **Compter les "A"** et **Compter les "LE"** permettent à l'étudiant d'exécuter librement les algorithmes à l'aide des boutons de fonctions.

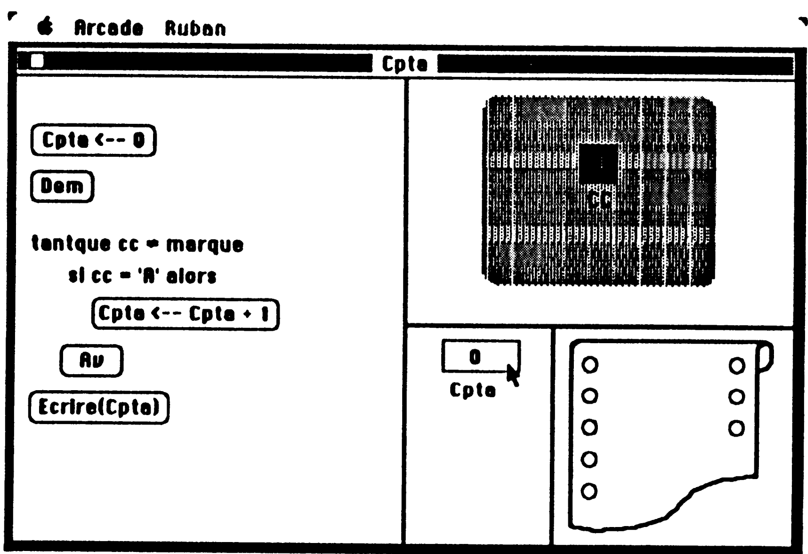


Figure 4.18 Compter les "A".

Ces maquettes sont purement manipulatoires, ce ne sont pas des animations d'algorithme gérées par l'exécution. L'étudiant déclenche librement les actions disponibles sur la machine. Les seuls contrôles effectués vérifient si l'action demandée respecte sa spécification fonctionnelle. Par exemple, faire **AV** avant **DEM**, ou **AV** après la marque de fin déclenche l'apparition d'un message d'erreur. C'est un outil de tâtonnement et de découverte pour aborder l'analyse itérative. Parmi des logiciels du laboratoire ARCADE, le meccano de tri [Arc 88e] relève d'une approche analogue.

Machines caractères exécute des algorithmes simples mais ne permet pas d'exécuter à la main, des algorithmes de traitement de séquence plus complexes, comme par exemple un éditeur de texte.

J'ai tenté de réaliser d'autres maquettes concernant la visualisation de plusieurs abstractions : Machine-Caractères, Machine-Couples, Machine-Mots qui délivrent un caractère, un couple ou un mot. Leur réalisation a été abandonnée car la visualisation devient trop lourde, et la notion des différents états importants à montrer pour la synchronisation des actions dans le traitement

séquentiel n'est pas claire sur les trois machines. Par contre, la visualisation sera plus claire en présentant uniquement le ruban sans les machines :

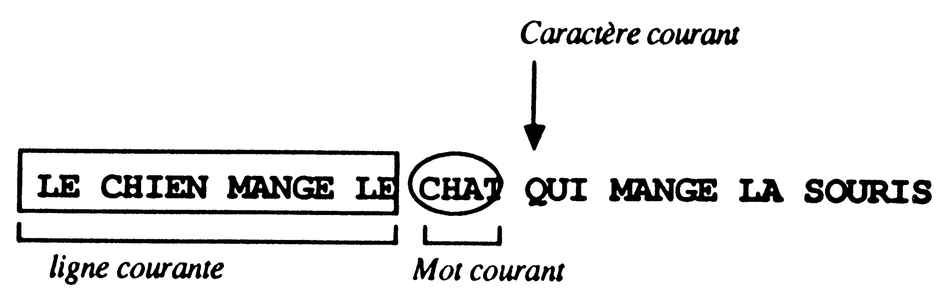


Figure 4.19 Représentation du ruban sans la machine.

La machine caractères est un modèle abstrait d'accès séquentiel à une séquence représentée par un ruban. Initialement, mon objectif était de concrétiser ce modèle car je croyais que la visualisation éclairerait cette machine abstraite. En réalité, j'ai beaucoup de mal à concrétiser l'abstraction car la visualisation détruit le lien entre le ruban et la machine.

En effet, le modèle simulé dispose d'un dispositif d'entrée/sortie, absent du modèle original puisque l'action "changer le ruban" n'est pas prévue. Une machine est associée à un ruban et à un ensemble d'actions. Pour cela la maquette visualise "changer le ruban" comme un changement de machine.

La réalisation de cet ensemble de maquettes apporte une conclusion importante au sujet des limites de la visualisation. Parfois la visualisation détruit une conception ou conduit à une interprétation fausse. La simulation d'un modèle abstrait nécessite manipulations et visualisations peu fidèles à l'abstraction initiale.

4.5. Visualisation et graphes

La représentation logique usuelle d'un graphe (nœuds et arcs) est très simple, aussi bien pour la lecture que pour la manipulation. Sa représentation physique en mémoire est, par contre, très complexe (enregistrements et pointeurs).

Une première difficulté de la visualisation de l'exécution d'un programme traitant un graphe est de rendre compte des modifications de la structure physique au cours de l'exécution. Une seconde difficulté est de montrer ces modifications en permettant à l'élève, en permanence, de relier les structures physique et logique.

4.5.1. Tri topologique

Le tri topologique consiste à définir un ordre total à partir d'un ordre partiel. C'est exemples type sur le thème de la gestion mémoire [Lie 85] intégrant des difficultés progressives : gestion des visiteurs d'une entreprise, gestion d'une salle d'attente d'un médecin, répertoire électronique, course de ski, tri par distribution, fréquences des mots d'un texte, tri topologique, références croisées.

Il s'agit donc de visualiser une structure de données complexe.

Deux types d'ouvrages complémentaires traitent ce sujet :

- [PMS 88] présente un enseignement basé sur l'analyse fonctionnelle qui conduit à la solution usuelle. La structure de données est déduite de cette analyse. Cette approche s'intéresse à l'élaboration de méthodes, elle n'est donc pas adaptée à nos recherches sur l'exécution.
- [Wir 86] et [HoS 84] se préoccupent plutôt de la mise en place de la structure des données. Ces ouvrages concernent plus l'exécution et servent donc de base de travail pour **Tri topologique**. Le "pourquoi" de la représentation chaînée, ici la gestion mémoire, n'est pas l'objet de ce logiciel.

●Tri topologique propose un travail sur l'algorithme appliqué au graphe **orienté**. Nous abordons la visualisation des pointeurs.

Les différentes représentations et algorithmes correspondants sont étudiés :

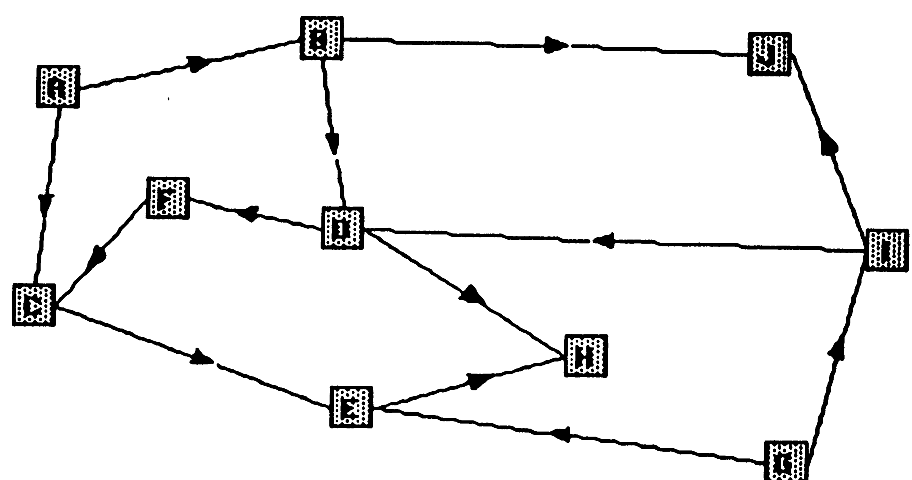


Figure 4.20 Représentation logique.

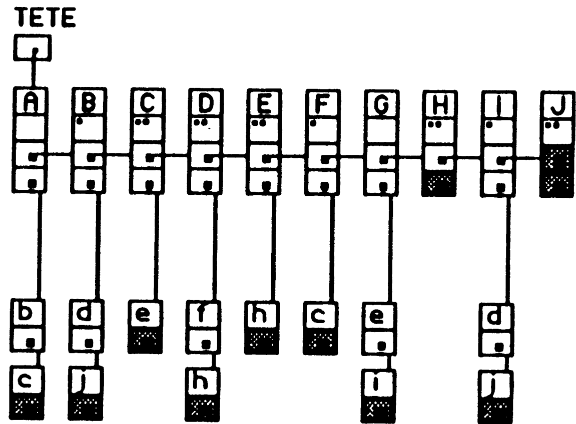


Figure 4.21 Représentation chaînée [Wir 86] : des nœuds et leurs successeurs.

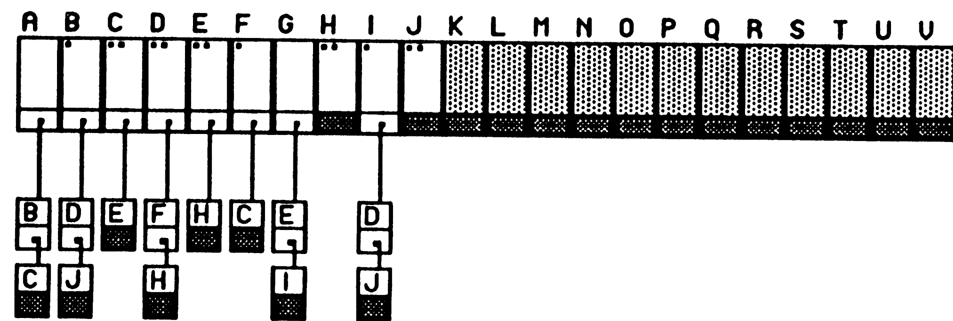


Figure 4.22 Liste d'adjacence [HoS 84] : l'ensemble des nœuds est présenté dans une table, et leurs successeurs sont chaînés.

La maquette propose un exemple de graphe par défaut, tiré de [Wir 86]. Cependant l'étudiant peut modifier ce graphe ou en créer un autre dans un environnement graphique interactif.

Remarques

Dans cet algorithme, le nombre important de données (nœuds et leurs successeurs) n'améliore pas la compréhension du fonctionnement de l'algorithme. **Tri topologique** limite donc volontairement le nombre de données pour la représentation physique.

La visualisation appelée "**physique simplifiée**" a nécessité un travail pédagogique préalable conséquent car l'exécution sur la représentation physique complète est inobservable, même avec la vitesse la plus lente. Cette visualisation masque certaines informations et ne conserve que les informations pertinentes à la compréhension de l'exécution.

Animer un pointeur oblige l'auteur à réfléchir sur le choix de l'objet pointé : un seul élément de la liste ou la liste elle même. L'animation doit mettre en évidence la sémantique visuelle (élément ou liste) correspondant au rôle du pointeur dans l'algorithme.

4.5.2. Graphes non-orientés

La maquette **Parcours de graphes** n'est pas liée à un problème ou à un exemple type. Elle illustre des opérations de bases sur une structure, ici un graphe **non-orienté** : recherche, parcours, arbre de recouvrement, composants connexes. L'exécution d'un algorithme est visualisée sur plusieurs représentations :

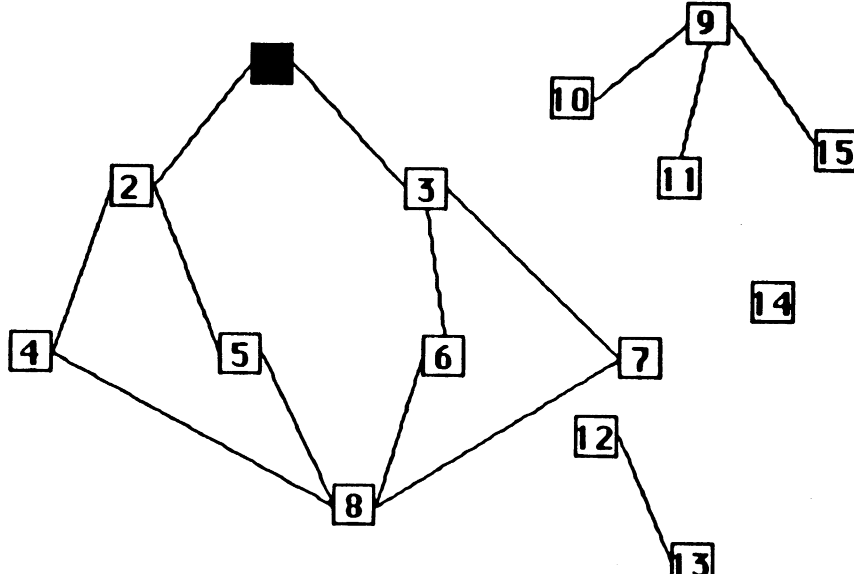


Figure 4.23 Représentation logique d'un graphe non-orienté.

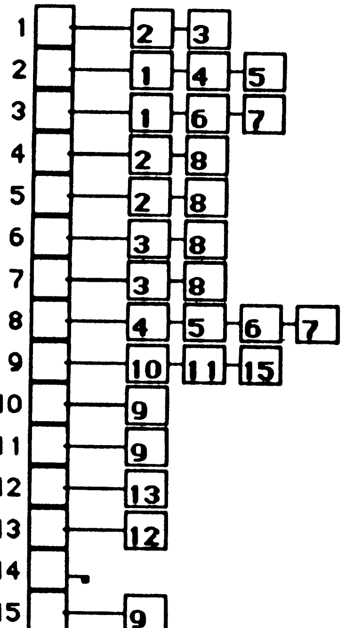


Figure 4.24 Représentation sous forme de "liste d'adjacence" du graphe précédent : les nœuds sont présentés dans un tableau et leurs successeurs sont chaînés.

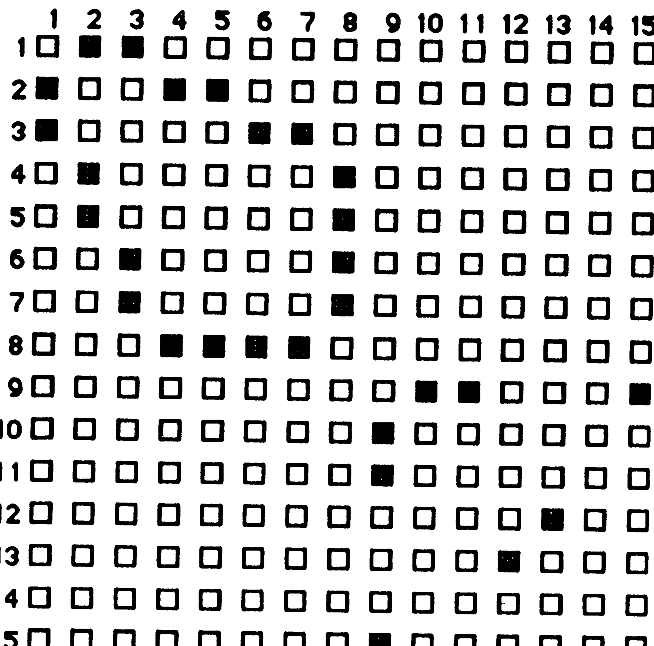


Figure 4.25 Représentation sous forme de "matrice d'adjacence" du graphe de la figure 4.23 : chaque booléen de la matrice représente un arc.

L'étudiant peut modifier ou créer interactivement le graphe d'entrée sous forme de représentation logique, dans un environnement graphique.

Concernant également la visualisation des opérations de base, nous pourrions ajouter au laboratoire ARCADE les thèmes suivants :

- parcours d'une structure arborescente,
- empiler/dépiler sur une pile,
- ajout, suppression sur une file,
- insertion, suppression, ajout sur une liste linéaire.

Ces primitives seraient relativement faciles à réaliser, de manière quasi automatique, surtout si la structure logique est seule présente.

5. Premier bilan pédagogique

Rentabiliser nos efforts successifs de maquettage nous oblige à réfléchir sur la démarche à suivre. Il ne suffit pas de formuler hâtivement une idée vague, puis de se précipiter dans une analyse rapide, pour enfin se satisfaire d'un programme qui tourne. D'ailleurs, à ce stade, quelles conclusions pédagogiques ou ergonomiques pourrions-nous faire ? La vie d'une maquette doit pouvoir se prolonger au delà de sa réalisation.

Il faut se donner les moyens d'exploiter la maquette pour en tirer les conclusions relatives à nos objectifs. Pour chaque maquette, il faut donc, dès le départ, formuler les questions que l'on se pose et la manière d'évaluer les réponses.

Une maquette concrétise une idée : laquelle ?

Elle doit permettre de la valider : comment ?

Plus la maquette est simple, plus son but sera facile à définir ; son évaluation en sera facilitée. Ensuite, la réalisation de maquettes plus ambitieuses s'appuiera sur les résultats cumulés des expériences précédentes.

Faudra-t-il nécessairement les valider sur une population d'élèves afin d'aller jusqu'à l'obtention de résultats statistiquement significatifs ?

Est-il vraiment souhaitable, ou même possible, de faire subir un tel banc d'essai à des maquettes qui, par définition, sont partielles et peu robustes ?

Les difficultés concrètes d'une évaluation en vraie grandeur sont loin d'être négligeables : temps, matériel, organisation, protocole d'expérimentation, groupe témoin, etc. Comme le montrent de nombreuses évaluations faites en psychopédagogie, les résultats obtenus sont parfois très minces et surtout très

sensibles à de minimes changements des conditions de l'expérience. On peut donc s'interroger sur l'intérêt d'aller très loin dans ce sens.

Nous retrouvons ici un grand sujet de polémique dans l'équipe ARCADE. Les précédentes maquettes n'ont pas été évaluées car les auteurs estimaient leurs réalisations trop imparfaites.

J'ai quand même expérimenté mes logiciels. Mais les expérimentations n'ont pas pour but l'obtention de statistiques significatives ou la mesure quantitative de l'efficacité des logiciels. Je souhaite plutôt tester leur convivialité, l'ergonomie des modalités d'interaction et bâtir quelques hypothèses pour leur intégration future à l'enseignement.

Les expérimentations ont eu lieu dans les conditions suivantes :

1. Quelques étudiants de l'IGEI (MIAG, MST Expert en Systèmes Informatiques) ont travaillé pendant quelques heures en libre service avec le laboratoire ARCADE.
2. Un groupe de 5 étudiants du DESS Informatique Double Compétence ont travaillé pendant une demi-journée en libre service avec le laboratoire ARCADE.
3. **Dessins récursifs** a servi d'outil d'enseignement pendant certaines séances de cours du CIAP [Pey 88b].
4. **Tris internes** a été utilisé pendant deux séances de cours au DEUG A de l'Université Joseph Fourier de Grenoble.
5. J'ai organisé deux séances d'expérimentations pour des enseignants et des élèves ingénieurs du département d'informatique de l'Institut de Technologie de Bandung (ITB), en Indonésie. Je décris plus particulièrement ces expérimentations.

5.1. Expérimentations à l'Institut de Technologie de Bandung

La rentrée de l'année universitaire 87/88 à l'ITB m'a donné l'occasion d'expérimenter **Tris internes**. A la rentrée suivante, j'ai pu présenter le laboratoire ARCADE et expérimenter mes maquettes.

Ces expériences m'ont permis d'une part de recenser les avis des enseignants sur l'utilité de ces maquettes dans leur enseignement de la programmation, et d'autre part d'observer les réactions des étudiants. Sur le plan technique, je teste la portabilité des logiciels d'une langue à l'autre, car les maquettes expérimentées à l'ITB sont écrites en anglais.

Les remarques des enseignants et des étudiants sont prises en compte dans la nouvelle version des maquettes.

5.1.1. Profil des étudiants

J'ai choisi 3 catégories d'étudiants, caractérisés par leur niveau de connaissance en algorithmique :

- ceux qui ont suivi un cours d'initiation à l'informatique (2 heures par semaine) pendant un semestre de leur première année d'études et qui rentrent en 2° année,
- ceux qui ont suivi pendant la 2° année, un semestre de cours de base d'algorithmique et de programmation. Ils connaissaient déjà certains algorithmes présentés par les maquettes.
- ceux qui rentrent en 4° année après avoir suivi des cours d'algorithmique "pointus" pendant les années précédentes.

Une vingtaine d'étudiants ont participé avec beaucoup d'enthousiasme à chaque expérimentation.

5.1.2. Déroulement de l'expérimentation

Pour les enseignants

J'ai organisé des démonstrations courtes. Ces maquettes semblent leur convenir en tant que complément de cours : comme outil d'enseignement et comme outil de travail individuel d'étudiant.

Ils aimeraient bien utiliser ces maquettes dans leur enseignement : l'absence actuelle de matériel Macintosh les en empêche.

Pour l'étudiant

La démarche d'expérimentation se déroule en quatre étapes :

- une démonstration préliminaire pour expliquer l'interface Macintosh, et plus particulièrement la manipulation des maquettes. Cette phase est nécessaire car les étudiants n'avaient jamais travaillé sur ce système auparavant.
- un travail individuel en binômes. Pour certains thèmes considérés comme connus, le tri ou la recherche par exemple, les étudiants travaillaient directement sur machine. Pour d'autres, comme la récursivité ou la reconnaissance d'un motif, une ou deux séances de cours magistral ont précédé le travail sur machine.
- des discussions libres pour noter remarques et observations pertinentes,
- une évaluation, écrite par l'étudiant, sur l'utilisation des maquettes.

5.1.3. Bilan pédagogique

Le dépouillement des questionnaires regroupe et classe les remarques les plus fréquentes.

Niveau de connaissance

Une même maquette peut être pratiquée à des niveaux différents d'apprentissage selon la connaissance que l'étudiant a du sujet :

- si l'étudiant ne connaît pas l'algorithme, il observe le fonctionnement global ou la performance de l'algorithme.
- si l'étudiant connaît l'algorithme, il essaie de retrouver ses caractéristiques, il comprend mieux le sujet et consolide ses connaissances.

Temps d'exécution

L'étudiant est très sensible à cet aspect quand il observe l'exécution. La première réaction enregistrée en cours d'expérimentation est : cet algorithme est lent ou cet algorithme est rapide !

Plusieurs vues présentant les valeurs

La visualisation abstraite et non-classique retenue dans **Tris internes** n'est pas naturelle pour une première approche : la plupart des étudiants souhaiteraient observer l'exécution avec la représentation exacte des valeurs du tableau à trier, sous forme de chiffres. L'abstraction des valeurs par les traits leur semble insuffisante et imprécise.

Ceci confirme aussi le constat de [BrS 85] : plusieurs vues de structure de données sont nécessaires, et la représentation classique sur un petit nombre de données est importante. Remarquons que le film "La gare de triage" [Arc 88c] présente précisément un tri à l'aide des valeurs chiffrées.

Observation détaillée ou appréciation globale

Un cours ou un livre est incapable de montrer l'exécution sur un grand jeu de données. Dans l'observation, l'étudiant a l'habitude de s'intéresser d'abord au détail de l'exécution. Après discussion, ils ont trouvé intérêt à exécuter sur un grand nombre de données. En effet, malgré la valeur par défaut des éléments à

trier (100 éléments) avec une vitesse rapide, la plupart des étudiants ont commencé à utiliser **Tris Internes** avec un petit nombre de données (15 - 20 éléments) en vitesse lente.

Clarté d'exécution

Ils ont affirmé que les algorithmes séquentiels sont plus faciles à observer. Par contre, les algorithmes arborescents sont difficiles à comprendre.
Ils commentent ainsi :

- "Nous voyons ce qui se passe, mais nous ne comprenons pas ce que fait exactement l'algorithme".
- "Nous comprenons cette exécution après avoir compris l'analyse".

Efficacité des différents algorithmes traitant le même problème

En exécutant les différents algorithmes traitant le même sujet (cf. Figure 4.2) les étudiants ont pris position sur l'efficacité des algorithmes. Le critère mesuré concernait le temps d'exécution sans exécution simultanée : ce n'est qu'une évaluation expérimentale.

L'occupation mémoire (**Tri topologique**) est un critère d'évaluation moins évident que la notion de temps.

Classification des algorithmes

Les étudiants désirent classifier les algorithmes selon les critères suivants : rapidité d'exécution, facilité d'observation, voire même parfois la qualité esthétique, ce qui ne correspond pas a priori à l'objectif initial avoué de notre recherche.

Compréhension d'un algorithme à travers son exécution

Les étudiants ont constaté la difficulté de comprendre l'algorithme complet après avoir vu son exécution. Ils arriveront quand même à voir les itérations

principales et à repérer les actions répétitives (par exemple : décalage, recherche) pour les algorithmes itératifs (tris simples). Ils ont repéré facilement l'invariant d'une boucle grâce à l'exécution.

La visualisation d'exécution est donc capable de donner une vue globale de l'algorithme, mais n'est qu'un élément de l'apprentissage !

Voir des choses dont le professeur avait parlé

Les étudiants voient mieux à travers la manipulation des différentes maquettes les différents algorithmes que le professeur avait présentés. Ce qui confirme que le guidage du professeur sur un outil de travail est important (cf. Chapitre I). Si le professeur explicite des choses pertinentes à observer avant la séance de travail, l'étudiant apprend plus de choses : les points importants à observer, les particularités d'un algorithme par rapport aux autres, etc ...

Analyser autrement

L'observation d'exécution permet à l'étudiant d'analyser le comportement d'un algorithme ou de comparer les comportements d'algorithmes concurrents en observant et en analysant les phénomènes visuels produits par l'exécution. Il est sûr que ce n'est pas une analyse de construction d'algorithme, mais l'expérimentation a montré que les étudiants sont plus actifs et plus critiques ensuite car ils ont acquis plus d'expérience dans l'observation. Pour les premières observations, le professeur doit expliciter chaque détail et comment observer. Après plusieurs observations, l'étudiant "voit" lui même des choses pertinentes, en analysant l'animation visuelle.

5.2. Discussion

5.2.1. Une seule fenêtre active

L'environnement Macintosh permet d'ouvrir simultanément plusieurs fenêtres. L'utilisateur est libre de manipuler ces fenêtres. Ce système est très pratique pour un travail documentaire puisque transférer un morceau de texte d'un document à l'autre est chose aisée.

L'observation et la visualisation sont conçues dans mes maquettes à l'intérieur d'une seule fenêtre. La volonté de ne pas disperser l'attention, de ne pas perturber la concentration et donc de faciliter la compréhension détermine ce choix.

5.2.2. Visualisation d'exécution et analyse

Au cours de l'utilisation de maquettes "purement" opératoires, plusieurs étudiants affirment : "L'exécution ne donne qu'une vue globale. Néanmoins, on voit ce qui se passe, on repère des choses. Mais l'observation de ces maquettes ne nous rend pas capable de construire un algorithme à partir d'un énoncé".

Un système de visualisation d'exécution, surtout si le système est basé purement sur l'exécution de l'algorithme est facile à réaliser, mais il ne sert pas à grand chose. Ce type de système peut montrer que l'algorithme est opérationnel, mais n'explique pas comment cet algorithme fonctionne. Il faut plus d'efforts pédagogiques (travailler, décortiquer, "tricher") pour obtenir une exécution observable et compréhensible. Malgré ces efforts, ce n'est qu'une "petite" contribution à l'apprentissage.

5.2.3. Diversité des données

Il est possible de classifier mes maquettes selon deux critères :

- l'algorithme s'exécute avec des données préétablies (cf. Figure 4.2, Saisie/variation de données "*")
  La variation d'images-écran à l'exécution est limitée par les choix restreints disponibles. La visualisation gérée par l'exécution est préférée, mais d'autres techniques comme celle de la reconstruction d'un film à partir des images-écran capturées par un "moniteur" de programme est possible.

- l'étudiant peut observer l'exécution d'un même programme sur des données différentes (cf. Figure 4.2, Saisie/variation de données "**" ou "***"). Le nombre et la nature des observations sont libres car l'utilisateur peut créer un ensemble illimité de données.
  La visualisation doit s'effectuer en temps réel, ce qui est une nécessité pédagogique. Les images dépendent exclusivement des données fournies par l'utilisateur avant l'exécution de l'algorithme à observer.

Dans les dessins récursifs, les données "réelles" sont des courbes ! Les données sont donc un programme ou un algorithme. Il est plus intéressant pour l'étudiant de programmer lui même une courbe dans le même environnement.

Si le logiciel traitait un programme comme donnée, donc permettait à l'étudiant de programmer son algorithme, le logiciel ne serait plus purement un logiciel de visualisation d'exécution, mais intégrerait un interpréteur. Cet environnement fournirait une assistance à la compréhension et à la production simultanément ; nous n'avons pas traité ce cas.

L'expérience d'utilisation du logiciel **Dessins récursifs** par des professeurs du secondaire (CIAP) [Pey 88b] a montré, qu'après la séance de travail avec ces logiciels, les professeurs souhaitaient programmer de nouvelles

fractales mais que **Dessins récursifs** ne leur permettait pas de programmer les dessins présentés, ni les autres fractales.

5.2.4. Plusieurs algorithmes pour un même énoncé

Compléments de cours

Le temps limité d'une séance de cours est une contrainte forte pour présenter toutes les solutions possibles. Les logiciels qui présentent plusieurs algorithmes, solutions d'un même problème (cf. Figure 4.2), permettent à l'étudiant d'étudier eux-mêmes les algorithmes écartés pendant le cours.

Ces logiciels sont des compléments de cours, ou plutôt un complément d'un complément de cours. Le professeur introduit quelques algorithmes dans le cours et indique d'autres algorithmes référencés dans un livre ou un polycopié. L'étudiant travaille ensuite individuellement avec le logiciel en suivant les démarches d'analyse présentés dans ces supports papier.

Les exécutions simultanées

L'exécution simultanée est une fonctionnalité importante : c'est le point fort du système BALSA [Bro 87]. Organiser des "courses" d'algorithmes est sûrement spectaculaire. Mais est-ce vraiment nécessaire ?

L'exécution simultanée est un élément important pour comparer les performances des algorithmes, mais secondaire pour comprendre leur exécution et encore moins leur construction ! C'est pour cela que cette fonctionnalité n'est pas introduite dans mes logiciels.

5.2.5. Nécessité de plusieurs vues

La création de vues est un travail pluridisciplinaire : pédagogique, artistique, graphique et informatique.

Pour la structure de données, mes recherches portent sur la visualisation de la représentation physique et logique des données [MeB 80].

La représentation physique détaille la mise en œuvre des données sur la machine : elle n'exprime pas le fonctionnement de l'algorithme au niveau de l'analyse. Elle correspond à l'exécution machine et ajoute parfois des variables de travail. Elle est incompréhensible.

La représentation logique exprime le fonctionnement de l'algorithme au niveau de l'analyse, indépendamment de la représentation physique. Ce qui rend l'exécution naturellement plus compréhensible.

Dans le parcours de graphe en profondeur par exemple, la descente d'un niveau impose la mémorisation explicite dans une pile. Sur la représentation logique, on "voit" le niveau courant, sans expliciter la pile. Par contre, l'exécution sur la représentation physique est inobservable sans visualisation de la pile.

Pour une structure complexe, nous masquons certaines informations afin que l'observation d'exécution ne soit pas confuse. Il n'est pas étonnant que l'exécution sur cette représentation physique simplifiée donne presque la même chose que la représentation logique car cette représentation extrait les informations nécessaires à la compréhension.

Parfois, une structure n'est compréhensible qu'à travers sa représentation logique. Par exemple, pour visualiser l'arbre de recouvrement d'un graphe non-orienté :

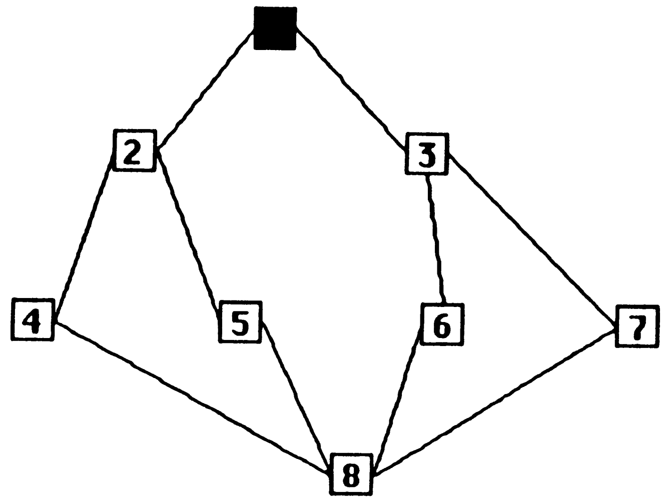


Figure 5.1 Le graphe.

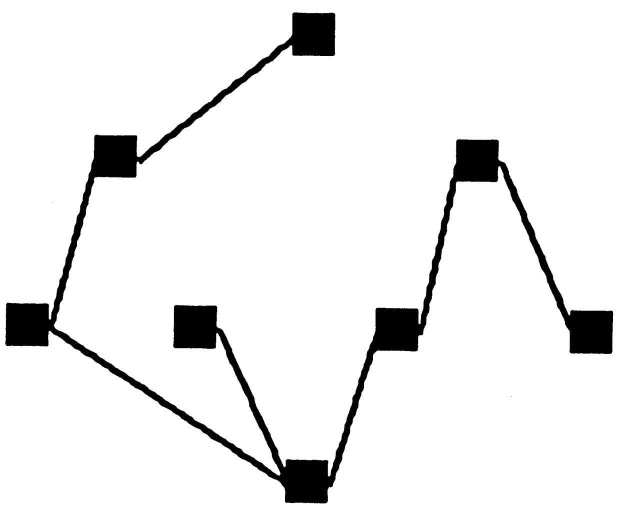


Figure 5.2 Un arbre de recouvrement sur la représentation logique du graphe précédent.

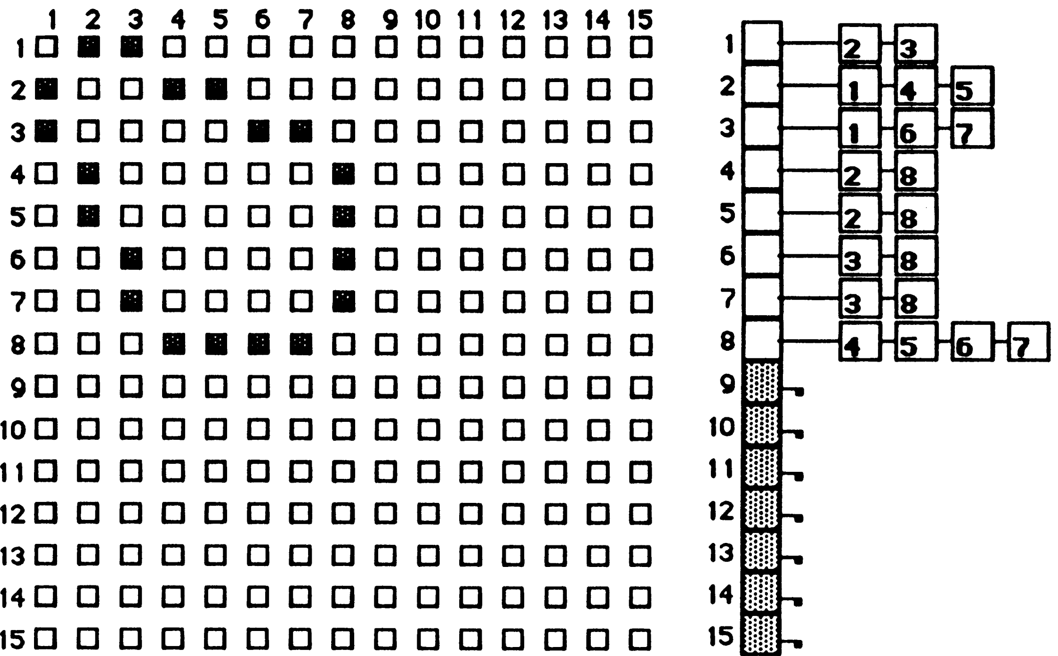


Figure 5.3 Représentations physiques du graphe de la figure 5.1.

Comment faire apparaître l'arbre au sein de ces représentations ?

5.2.6. Exécution pas à pas

La notion de "pas" correspond plus à la volonté d'une visualisation particulière, liée à la nature de l'algorithme. Quelle règle choisir pour déterminer ce qu'est un pas de programme ? Un pas peut être une instruction, un passage d'une itération, un appel récursif, ou une comparaison, deux comparaisons successives ...

Une telle attitude dans les choix nécessite une analyse pédagogique préalable de l'algorithme.

5.2.7. Temps d'exécution

Le même logiciel sur plusieurs machines

Le laboratoire ARCADE fonctionne sur 3 types de Macintosh : Mac Plus, Mac SE, Mac II. La puissance de ces machines est différente. Avec le même réglage, la vitesse d'exécution d'une visualisation en temps réel peut être trop rapide ou trop lente selon la vitesse de la machine.

La vitesse d'exécution d'un même logiciel doit donc être indépendante du matériel car :

- le temps d'exécution est un facteur primordial d'observation,
- un étudiant peut étudier un même logiciel au cours de plusieurs séances de travail, donc sur différentes machines.

La solution technique retenue pour traiter ce problème est décrite dans la partie §6.2.1.

Plusieurs algorithmes sur la même machine

L'observation de l'exécution des programmes donne une intuition forte sur la performance, surtout en terme de temps. L'expérimentation sur mes maquettes l'a prouvé. Il est donc important de ne pas tromper l'utilisateur. La visualisation de plusieurs algorithmes, solutions d'un même problème, doit s'effectuer dans des conditions identiques. Le rythme d'affichage doit donner une impression juste sur le temps d'exécution réel de chaque algorithme.

Par exemple, ****Tris internes** et ****Pattern matching** visualisent chaque affectation et chaque comparaison en appelant la même procédure.

5.2.8. Limites

Limite de l'exécution

La visualisation à partir de l'exécution en temps réel ne présente pas une bonne compréhension d'un algorithme, si les points de visualisation ne sont pas bien étudiés.

Pour certains cas, il est impossible d'obtenir une unité compréhensible à partir de l'exécution car l'exécution ne correspond pas à l'analyse.

Prenons l'exemple du flocon de neige de **Dessins récursifs**. L'unité compréhensible d'analyse est "un flocon" pour une valeur de N donnée. Or l'unité de l'exécution est "une ligne". L'exécution s'effectue en traçant coté par coté par juxtaposition des lignes. L'exécution ne génère qu'une visualisation au niveau le plus élémentaire (par ligne) et pas au niveau de l'analyse (flocon).

Comment peut-on produire un flocon d'un seul coup ? Il faut produire le film pour obtenir l'affichage flocon par flocon.

Le logiciel **Dessins récursifs** permet de produire des images d'un tel film. L'animation des images a été réalisé par un des membres de l'équipe avec le logiciel Videoworks.

Limite de l'espace de visualisation

La surface de l'écran est généralement trop restreinte pour dessiner la structure complète. La technique *zooming/panning* permettrait de résoudre ce problème. Je laisse de coté, pour le moment, cette technique graphique. Du point de vue pédagogique, notre enseignement insiste sur l'analyse : la limitation à un nombre réduit de données n'est pas un handicap et facilite l'occupation de l'espace écran.

Limite des calculs et de la précision.

L'affichage en mode point nous condamne à travailler sur des valeurs entières. Si celles-ci sont le résultat d'un calcul tronqué, leur visualisation est alors erronée.

1° cas : l'affichage des valeurs utilisées dans l'algorithme Rabin-Karp.

Deux valeurs légèrement différentes peuvent être représentées par des objets identiques à cause de l'arrondi de calcul. L'affichage est donc corrigé pour que la différence réelle soit visible. C'est un des exemples de travail "pédagogique manuel" que l'animation d'algorithme exige !

Voici le résultat après correction :

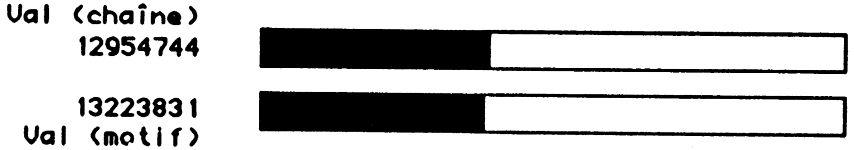


Figure 5.4 Visualisation de deux grandes valeurs.

2° cas : la notion de limite.

En théorie, la courbe fractale tend à couvrir toute la surface de dessin, mais se concentre en réalité sur un seul point à partir d'une certaine valeur de N. Pourquoi ce résultat paradoxal ?

Prenons la courbe de Hilbert comme exemple. L'unité de dessin d'un trait dépend de la valeur de N et de la surface totale du dessin. A partir d'une valeur de N, le calcul arrondi de cette unité donne 0 : la trace d'une ligne en avançant de 0 unité ne quitte pas le point de départ ! L'épaisseur de trait est aussi un facteur déterminant.

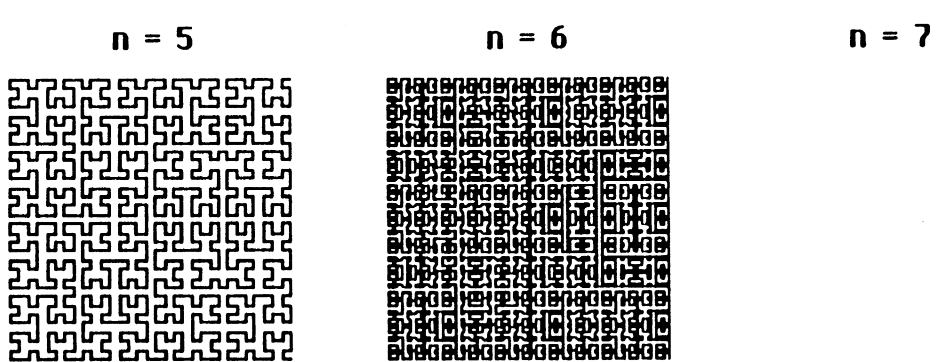


Figure 5.5 Courbes de Hilbert d'ordre 5, 6 et 7 avec une surface et épaisseur de trait identique.

3° cas : différentes tailles de dessins dans un logiciel.

Dessins récursifs détermine la taille optimale du rectangle englobant chaque courbe. Ce calcul diminue le nombre d'erreurs dans la représentation d'une même courbe à différents niveaux d'appels récursifs. De plus, chaque courbe a sa formule de calcul d'unité de dessin d'un trait, ce qui explique les différences de taille.

Malgré ces efforts d'ajustement de taille du dessin et l'unité, les dessins ne sont jamais très esthétiques. La visualisation des courbes de Sierpinski et de Hilbert nous a demandé un travail de préparation important dans le choix des dimensions :

- dessiner sur une surface constante avec l'unité de dessin d'un trait qui se raccourci,
- dessiner avec une unité de trait constante, ce qui agrandit très vite le dessin aux limites de l'espace.

J'ai finalement choisi la première manière de tracer la courbe à cause de la surface limitée de l'écran. Malheureusement, dans ce cas, la succession des calculs arrondis superpose ou décale certains tracés pour finalement afficher un dessin sans intérêt !

Voici ces deux courbes en taille réelle dans le logiciel :

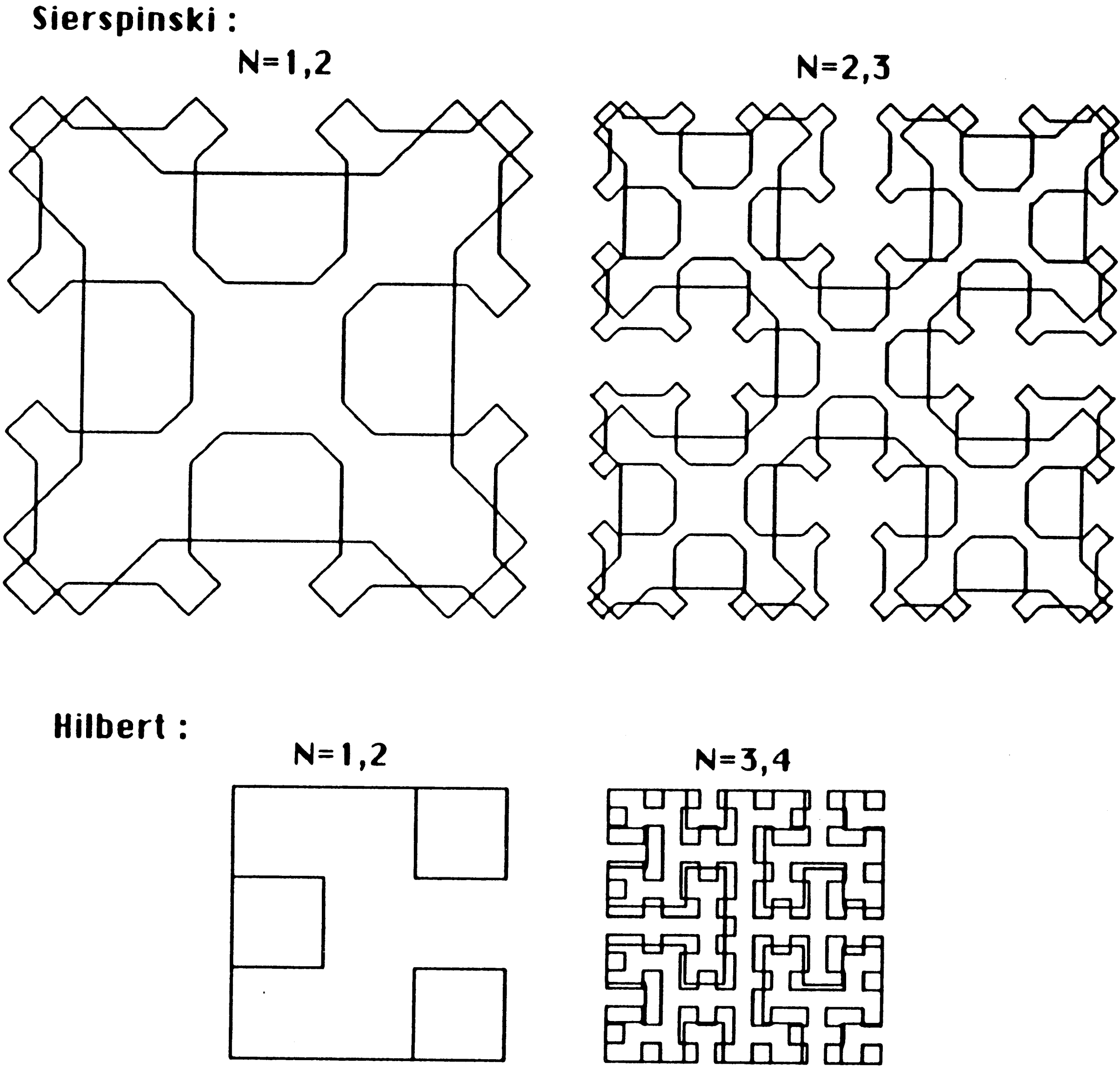


Figure 5.6 Courbes de Sierpinski et courbes de Hilbert superposées par 2 niveaux d'appels.

5.2.9. Contrôle de l'interaction

Les logiciels du laboratoire ARCADE orientés vers la manipulation (Meccano de tri, Arbre binaire) [Arc 88e] n'imposent pas de solution. L'objectif de ces maquettes est de mettre à disposition de l'utilisateur certains objets ou

opérateurs manipulables librement : aucune évaluation, aucune analyse de réponse, aucun jugement.

L'une des activités offertes par mes maquettes est l'exécution de l'algorithme à la main. L'étudiant propose, une à une, les différentes étapes successives de la construction de la solution.

Cet exercice peut être envisagé de deux manières :

- pour vérifier la compréhension d'une solution algorithmique d'un problème. Alors l'évaluation de la réponse de l'étudiant est simple : à chaque instant, il n'y a qu'une seule étape possible. Par exemple : **Dessins récursifs.**
- pour vérifier la compréhension d'un problème. Celui-ci peut admettre plusieurs solutions algorithmiques, l'évaluation de la réponse de l'étudiant est plus complexe : à chaque instant, il y a plusieurs étapes possibles. Par exemple : **Tri topologique**, **Parcours de graphes.**

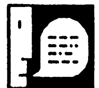
5.2.10. Aspect esthétique

Ce qui est inattendu, c'est l'apparition de dessins esthétiques et spectaculaires à l'exécution, comme pour **Tris internes** ou **Baguenaudier**. Il ne faut pas que la qualité esthétique de l'exécution d'un algorithme le fasse préférer à un autre dont l'exécution le serait moins.

6. Aspects techniques

L'objectif de cette partie n'est pas d'expliquer ou d'apprendre la programmation sur Macintosh : les explications complètes se trouvent dans "Inside Macintosh" [App 85]. J'essaie de faire part dans ce qui suit de mes expériences de programmation, des solutions choisies et de l'utilisation d'un compilateur général pour animer une exécution de programme.

Commençant par rappeler quelques termes utilisés dans le système Macintosh, je décris ensuite la mise en œuvre des fonctionnalités d'animation de programmes en Pascal. Enfin, j'énumère quelques problèmes de programmation survenus au cours de la réalisation.



Remarques :

- Plusieurs termes propres au Macintosh restent écrits en anglais, pour plus de justesse et de fidélité dans l'écriture : ils sont alors écrits en *italique*.
- Les exemples en Pascal sont incomplets : les déclarations de variables et de sous-programmes n'apparaissent pas. Leur présence surchargerait inutilement l'exemple proposé.

6.1. Annotation des programmes à visualiser

6.1.1. Evénements et boîte à outils

La programmation dans l'environnement Macintosh est pilotée par l'événement (*event driven system*). Le cœur d'une application est une boucle principale d'événement (*main event loop*). A chaque instant, l'application appelle le gestionnaire d'événement (*toolbox event manager*) qui lui même appelle le

gestionnaire d'événements du système d'exploitation pour examiner la requête de l'utilisateur. Cette boucle principale s'exécute jusqu'à ce que l'utilisateur quitte ce programme. Les événements sont gérés par le système dans une liste *FIFO* avec priorité.

La plupart des logiciels font appel au système d'exploitation ou à la ***"Toolbox"***, boîte à outils du Macintosh. Les procédures de la boîte à outils sont groupées par type d'objets à traiter au sein d'un même gestionnaire (*manager*). On trouve deux niveaux de gestionnaires :

- *high level* : accessible par l'application, stockés en *ROM* (*User interface toolbox*) ou en *RAM*. Ces gestionnaires font appel aux gestionnaires situés à un niveau plus bas du système d'exploitation.
- *low level* : procédures du système d'exploitation stockées en *ROM* ou procédures de bas niveau en *RAM*.

Décrivons très brièvement les fonctionnements des gestionnaires de haut niveau (high level) accessibles par l'application développée en Pascal :

- *Dialog manager* gère les dialogues en type mode (*modal*), sans mode (*modeless*) ou *alert*. Ce gestionnaire traite les items (contrôles, textes, icônes) associés à un dialogue. Le dialogue entre l'utilisateur et la machine s'établit par des fenêtres spécialisées.

- *Control manager* traite les contrôles standards (bouton, bouton radio, *check box*, ascenseur) : définition, affichage, reconnaissance, position, valeur, titre, activation, désactivation, etc.

- *Menu manager* gère les menus : définition, marquage, titre, activation, désactivation, reconnaissance d'un item.

- *Textedit* traite l'édition de texte : initialisation, définition, allocation, sélection, couper, coller, justification, saisie, *"scrolling"*, rafraîchissement automatique dans le cas d'une mise à jour de la fenêtre.

- *Window manager* gère les procédures standards de manipulation des fenêtres : définition, ouverture, fermeture, mouvement, changement de taille, superposition, activation/désactivation, détermination de la zone désignée par l'utilisateur, de la zone de dessin, de la zone visible et la porte graphique.

- *Toolbox utilities* contient les procédures des opérations standards : *fixed point arithmetic*, manipulation des chaînes de caractères, opérations logiques.

- *Toolbox event manager* traite les événements de l'utilisateur et du système. Ce gestionnaire transmet à l'application l'événement généré par le système qui nécessite une réponse, par exemple quand une fenêtre superposée devient active, il déclenche *update event*.

- *Desk manager* gère les accessoires de bureau (ouverture, fermeture). Un accessoire de bureau est une application qui s'exécute simultanément avec une application active (exemple: calculette, horloge, etc).

- *Scrapt manager* regroupe les procédures de traitement du presse papier (couper, coller). Le presse papier rend possible le transfert de différents types d'information (texte, graphique) d'une application à l'autre.

- *Quick Draw* gère les opérations graphiques.

- *Package manager* permet d'appeler les gestionnaires installés en RAM.

- *Font manager* gère le choix des polices ou jeux de caractères disponibles, édite ou la modifie une police existante ou non.

- *Resource manager* gère les procédures de définition et de modification des "ressources". Le concept de ressource est particulière à l'environnement Macintosh. La séparation entre les données et les codes de programme permet l'édition sans recompiler le programme. En principe, tous les messages textuels et tous les objets sont stockés dans la ressource : les fenêtres, les menus, les dialogues, les chaînes de caractères, les accessoires de bureau, les contrôles, les icônes, etc ....
  Programmer dans l'environnement Macintosh c'est : créer **d'abord** la ressource, **puis** programmer.

La volumineuse documentation des gestionnaires pose un problème d'utilisation : ils sont décrits dans les cinq volumes [App 85], comprenant au total environ 2000 pages.

6.1.2. Boucle d'événements

Généralement, l'écriture d'une application ne nécessite qu'une seule boucle d'attente, celle du programme principal (Figure 6.1). Le contrôle de programme revient à la boucle principale quand une tâche (une unité de programme) est complètement exécutée.

```
program MainProg;
{  Initmain : procédure d'initialisation                                   }
{  TermMain : procédure terminale                                          }
{  MaintainCursor : procédure d'affichage de curseur                       }
{  SystemTask : appel à toolbox                                            }
{  GetnextEvent : vérifier l'existence d'un événement                      }
{  MouseDown, KeyDown, UpdateEvt : type d'événement, prédéfini             }
{  Handlemouse, Handlekey, Miseajour : procédures de traitement            }
{     correspondant à l'événement. C'est dans ces procédures où            }
{     on "cache" l'application                                             }
var
   Finappl : boolean;    { vrai quand l'utilisateur finit l'application   }
   event : EventRecord; { les informations d'un événement                 }
begin
   Initmain;
   Finappl := false;
   repeat
      { obtenir prochain événement }
      repeat
         MaintainCursor;
         SystemTask;
      until GetnextEvent(EveryEvent, event);
      { traiter l'événement rangé dans "event" }
      case event.what of
         MouseDown : Handlemouse;
         KeyDown : Handlekey;
         UpdateEvt : Miseajour;
         otherwise
      end;
   until Finappl;
   TermMain;
end.
```

Figure 6.1 Structure générale d'un programme Pascal dans l'environnement Macintosh.

Dans la plupart des applications, on interdit l'intervention d'un événement pendant l'exécution d'une tâche, l'utilisateur doit attendre l'exécution complète d'une tâche, par exemple dans un traitement de texte : sauvegarder un fichier,

recherchercher un mot, etc. Mes logiciels doivent traiter plusieurs événements pendant le déroulement d'une procédure : réglage de vitesse, arrêt, pause. La structure élémentaire d'une boucle d'événements dans le programme principal est donc inapplicable.

En théorie, deux solutions sont possibles :

- "descendre" le niveau de la boucle d'événements à celui d'une procédure. Avec cette solution, l'appel génère des appels récursifs croisés (la boucle principale déclenche l'exécution d'une procédure, et la procédure appelle la boucle d'événements).
- emboîter les boucles d'événements ; la boucle d'événements principale contient d'autres boucles d'événements insérées dans les procédures.

Ces deux solutions ne sont pas satisfaisantes. En effet, une seule file est utilisée par le système et enregistre tous les événements en attente pour toutes les boucles emboîtées. L'exécution d'une boucle peut modifier l'enregistrement des événements en attente pour une boucle d'un niveau supérieur. Il faudrait pouvoir empiler les files d'attentes.

J'ai implémenté deux solutions.

Première solution

La première solution consiste à imposer une seule boucle d'événements dans le programme principal. Il est alors nécessaire de "découper" l'exécution d'une procédure en déterminant les "pas" (un pas est une tâche). Si une procédure doit effectuer une tâche F, on définit les pas i = 1, 2, 3, ... n, et le programme exécute successivement : f_1, f_2, f_3, ... f_n. Le contrôle revient au programme principal après l'exécution d'un pas. La boucle principale reçoit l'événement entre deux pas, et traite le changement des paramètres d'exécution (vitesse, arrêt, pause, etc).

Cette solution est cohérente sur le plan technique. Mais sur le plan pédagogique, le découpage de la procédure à étudier en une séquence linéaire de tâches détruit l'idée principale d'animation directe de programme. Ce découpage est encore pire pour la récursivité, la transformation d'une procédure récursive en procédures itératives (séquence d'appels) détruit complètement l'analyse récurrente.

A titre d'exemple, étudions comment annoter la procédure de recherche d'une valeur entière dans un tableau décrite dans la figure 6.2.

```
procedure TabseqPasapas (tab : tabel; X : integer;
                         var pos : integer; var found : boolean );
{ recherche d'une valeur entière (X) dans un tableau d'entiers (tab)      }
  var
    i : integer;
  begin
    i := 1;
    while (i < N) and (tab[i] <> X) do
        i := i + 1;
    found := (tab[i] = X);
    if (found) then
      pos := i
    else
      pos := 0;
  end;
```

Figure 6.2 Recherche séquentielle.

a) On modifie l'algorithme que l'on veut animer en introduisant la notion de pas. A chaque appel on doit retrouver tout le contexte mémorisé. Dans notre exemple, on obtient le programme de la figure 6.3.

```
procedure TabseqPasapas (tab : tabel; X, npas : integer;
                        var pos : integer; var found, finproc : boolean );
{ recherche d'une valeur entière (X) dans un tableau d'entiers (tab)       }
{ visuobserv : procédure qui visualise une comparaison                     }
  begin
    visuobserv(npas, xortab, yortab, black);
    finproc := (npas = N) or (tab[npas] = X);
    found := (tab[npas] = X);
    if (found) then
      pos := npas
    else
      pos := 0;
  end;
```

Figure 6.3 Procédure en pas à pas (seul le pas "npas" est exécuté et visualisé).

Cette solution ne nous satisfait pas car elle oblige à modifier la structure du programme que l'on veut annoter.

b) On intègre le numéro de pas dans la procédure initiale. Cette solution nécessite la possibilité de fonctionnement en coroutines. Malheureusement, cette fonctionnalité n'est pas offerte par Lightspeed. Alors, à chaque appel on exécute du premier pas jusqu'au pas courant. Pour donner l'impression de ne pas réexécuter, la visualisation de l'image écran est effectuée uniquement pour le numéro du pas courant.

L'absence de coroutines pénalise le temps d'exécution : plus le pas avance, plus l'exécution devient "lente" sur l'écran. Cette solution donne une impression

fausse de la performance de l'algorithme mais résout le problème d'une exécution pas à pas déclenchée par une action de l'utilisateur.

Cette méthode (Figure 6.4) n'est pas toujours possible, surtout si l'algorithme est complexe.

```
procedure TabseqPasapas (tab : tabel; X, npas : integer;
                         var pos : integer; var found, finproc : boolean );
{ visuobserv : procédure qui visualise une comparaison                    }
  var
    i : integer;
  begin
    i := 1;
    while (i < N) and (tab[i] <> X) and (i < npas) do
        i := i + 1;
    if (i = npas) then
      visuobserv(i, xortab, yortab, black);
    finproc := (i = N) or (tab[i] = X);
    found := (tab[i] = X);
    if (found)  then
      pos := i
    else
      pos := 0;
  end;
```

Figure 6.4 Recherche séquentielle, annotée (seul le pas "npas" est visualisé).

Le sous-programme TabseqPasapas (Figure 6.3 ou 6.4) est appelé dans la boucle intérieure du programme de la figure 6.5.

```
program mainprog;
var
   npascour : integer;    { numéro de pas                                   }
   finproc : boolean;     { vrai si on est à la fin de la procédure         }
                          { ou si l'élève arrête l'exécution                }
procedure Execotomatik;
begin
   { appel de la procédure annotée pour la visualisation, cf. Ex Figure 6.3 }
   Tabseqpaspas(tab, X, npascour, found, finproc);
   if (finproc) then
   begin
      { traitement de fin normal , à remplir ... }
   end else
      npascour := npascour + 1;
end;
begin
  Initmain;  Finappl := false;
  repeat
     repeat
        MaintainCursor;
        SystemTask;
        ExecOtomatik;
     until GetnextEvent(everyevent, event);
     case event.what of
        MouseDown : Handlemouse;
        KeyDown : Handlekey;
        UpdateEvt : Miseajour;
        otherwise
     end;
  until Finappl;
  TermMain
end.
```

Figure 6.5 Le programme principal de l'exécution d'une procédure en "pas à pas".

Deuxième solution

La deuxième solution consiste à introduire **2 boucles d'événements** différentes.

La première est la boucle principale qui attend l'événement, et **appelle** la procédure à animer. A ce moment là, cette boucle est terminée.

La deuxième boucle est un gestionnaire d'interruption et de temporisation. Elle **est appelée** à chaque fois qu'une interruption autorisée intervient pendant l'exécution (un changement de vitesse d'exécution, arrêt, pause). Cette boucle fixe en même temps la temporisation qui détermine la vitesse de l'exécution. Les appels à cette boucle sont insérés dans la procédure, de manière analogue aux annotations du système BALSA.

Le programme principal est modifié légèrement (Figure 6.6). La deuxième boucle est le gestionnaire d'interruption et de temporisation pendant l'exécution (Figure 6.7).

Cette solution est cohérente sur le plan pédagogique. Une idée forte du laboratoire ARCADE étant la modification possible des logiciels par les enseignants utilisateurs, ceux-ci doivent retrouver aisément le programme original tel qu'ils le connaissent sans gestion des points de visualisation.

```
program MainProg;
var
   InContextExec : boolean; { vrai pendant l'exécution d'un algorithme animé    }
begin
   Initmain;
   Finappl := false;
   InContextExec := false;
   repeat
                    { boucle principale d'application                          }
      repeat
                    { La première boucle d'événements :                        }
                    { ce n'est pas dans un contexte d'exécution                }
         repeat
            MaintainCursor;
            SystemTask;
         until GetnextEvent(everyevent, event);
         case event.what of
            MouseDown : HandleMouse;
            KeyDown : Handlekey;
            UpdateEvt : Miseajour;
            otherwise
         end;
      until InContextExec or Finappl;

      if InContextExec then
      begin
         ProcExec; { procédure qui déclenche l'exécution                      }
            { dans cette procédure on fait appel à la deuxième boucle        }
      end;
   until Finappl;
   TermMain;
end.
```

Figure 6.6 Programme principal d'une animation de programme.

```
procedure InteruptExec;
var
   passed : boolean;          { vrai si le temps d'attente qui          }
                              {  détermine "la vitesse"  est  atteint   }
   clickpause : boolean;      { vrai si le bouton "pause" est cliqué    }
   ctltick : controlhandle;   { ascenseur de la vitesse                 }
   interupt : boolean;        { vrai s'il y a une interruption          }
begin
   passed := false;     clickpause := false;
   repeat
      repeat
         debut := tickcount;        interupt := false;
         repeat
            SystemTask;
            if (tickcount - debut) > (GetCtlMax(ctltick)-GetCtlvalue(ctltick))
            then
               passed := true
            else if GetnextEvent(everyevent, event)
                  then interupt := true;
         until interupt or passed;
         if interupt then
         begin
            case event.what of
               MouseDown : HandleMouseExec;
               KeyDown : Handlekey;
               UpdateEvt : MiseajourExec;
               otherwise
            end;
            if (tickcount - debut) > (GetCtlMax(ctltick)-GetCtlvalue(ctltick))
            then passed := true;
         end;
      until (not InContextExec) or passed or clickpause;
   until not clickpause;
end;
```

Figure 6.7 Gestionnaire d'interruption et de temporisation pendant l'exécution.

La figure 6.8 est un exemple réel de l'animation dans **Tri topologique**. Nous avons pris tel quel l'algorithme du livre [Wir 86] (le texte du programme est imprimé en *italique*) et nous avons ajouté les gestions de visualisation et l'autorisation d'interruption pendant l'exécution qui sont équivalentes aux annotations dans le système BALSA.

Remarquons que les procédures de visualisation (DrawHeadLeader, InvertCounterAnElmtL, DrawLinkNextLeader, et VisuAelmtLeader) et d'autorisation d'interruption (Callinterrupt) ne sont pas systématiquement appelées à chaque changement de valeur ou chaque instruction. L'insertion de ces annotations a évolué au fil des essais de mise au point qui confortaient nos choix pédagogiques.

```
    procedure TriTopo;
{   "goto" est utilisé pour un arrêt immédiat pour éviter des multiples tests.      }
{   Lexique des procédures de visualisation                                          }
{      DrawHeadLeader pour dessiner ou effacer la tête d'une liste chaînée           }
{      InvertCounterAnElmtL pour mettre en inverse vidéo du champ "count"            }
{         d'un enregistrement                                                        }
{      DrawLinkNextLeader pour dessiner ou effacer le pointeur                       }
{      VisuAelmtLeader pour dessiner/effacer un enregistrement de la liste           }
   label
      999;
   var
      p, q : LPtr;
   procedure Callinterrupt;   { autorisation d'interruption et temporisation   }
   begin
      InteruptExec;
      if not InContextExec then
         goto 999;
   end;
{ à continuer }
```

Figure 6.8 Exemple d'un programme annoté pour la visualisation.

```
{ procedure TriTopo; (suite)                                    }
{ corps de la procédure                                         }
begin
   p := Head;
   DrawHeadLeader(Head, false);
   Callinterrupt;
   head := nil;
   DrawHeadLeader(Head, true);
   Callinterrupt;
   while (p <> tail) and (p <> nil) do
   begin
      q := p;
      InvertCounterAnElmtL(q);
      Callinterrupt;
      p := p^.Next;
      if (q^.count = 0) then
      begin
         DrawLinkNextLeader(Head, Tail, q, true, false);
         Callinterrupt;
         q^.next := head;
         InvertCounterAnElmtL(q);
         VisuAelmtLeader(Head, tail, q, true, true);
         VisuAelmtLeader(Head, tail, p, true, true);
         InvertCounterAnElmtL(q);
         DrawLinkNextLeader(Head, Tail, q, true, true);
         Callinterrupt;
         DrawHeadLeader(Head, false);
         head := q;
         DrawHeadLeader(Head, true);
         Callinterrupt;
      end;
      InvertCounterAnElmtL(q);
      Callinterrupt;
   end;
   999 :
end;
```

Figure 6.8 Exemple d'un programme annoté pour la visualisation (suite et fin).

6.1.3. Annotation dans le cas de vues multiples

Un logiciel de visualisation gère deux structures de données : structure de données internes (variables dans le programme) et structure de données externes (vues sur l'écran). Plusieurs modèles permettent de définir et de mettre en relation ces structures internes et externes. Par exemple :

- le modèle PAC : Présentation (vues sur l'écran) , Abstraction (structure interne), Contrôle (liaison entre une structure interne à une ou plusieurs présentations) [Cou 88],
- le modèle MVC : *Model, View, Controler* du langage orienté objet SmallTalk [LoD 85],
- le *"modeler"* (représentation interne et calcul des positions sur écran), *"renderer"* (visualisation des vues)" et *"adapter"* (liaison entre *modeler* et *renderer*) [Bro 87].

Un seul exemplaire de la structure interne pour plusieurs vues d'écran assure la cohérence. Ceci est efficace si les différentes vues se présentent simultanément sur l'écran, et si la structure est simple. Par exemple on peut visualiser un entier sous différentes formes (Figure 6.9).

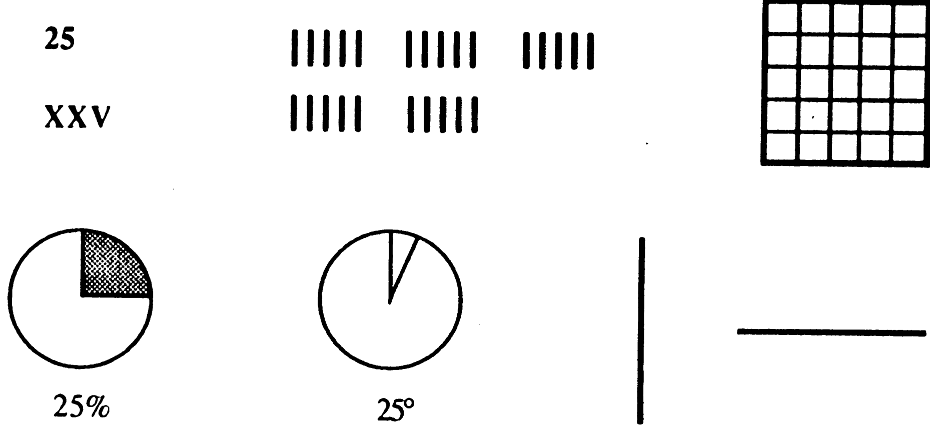


Figure 6.9 Diverses vues de la valeur 25.

La structure de données d'un problème peut être représentée par plusieurs structures physiques (implémentation sur la machine). Par exemple dans **Parcours de graphes**, la structure logique du graphe (Figure 4.23) a deux représentations physiques (Figure 4.24, Figure 4.25) et les algorithmes de parcours dépendent de ces dernières.

Quel compromis choisir pour la visualisation d'un sujet qui a plusieurs solutions algorithmiques ?

- exiger une seule représentation interne assure la cohérence et occupe peu de place mémoire mais condamne la performance puisque tous les algorithmes annotés utilise une déclaration de variables unique. Par contre la présence d'une seule structure interne nécessite la modification des algorithmes qui ne fonctionnent pas directement sur cette structure.
- permettre d'avoir plusieurs structures internes dans le programme.
  Chaque procédure annotée garde sa propre déclaration de variable. Dans ce cas, il faut assurer la cohérence des valeurs entre les structures pour l'ensemble des animations : il est important d'avoir un exemplaire de représentation de base pour assurer la cohérence. L'allocation d'une variable pour chaque structure interne est gourmande en place mémoire, mais l'animation d'algorithme est efficace puisqu'elle s'applique directement à la structure de données de l'algorithme.

Dans mes maquettes, j'ai appliqué plusieurs structures internes de données et j'ai associé pour l'instant chaque structure interne à une seule structure externe car :

- la plupart des sujets traités présentent plusieurs algorithmes traitant le même problème, chaque algorithme fonctionne sur une structure interne particulière. La structure de données de l'algorithme est donc une partie intégrante de la procédure annotée.
- chaque vue est basée sur une représentation interne complexe.

- les vues ne se présentent pas simultanément.
- il n'y a pas de vues différentes pour la même représentation interne, sauf pour les deux structures externes de **Pattern matching** (alphabétique ou codée). Dans ce cas, la liaison entre la représentation interne et la vue sur écran est exprimée par un *"case"* simple dans l'affichage.

Il serait intéressant dans le futur, d'enrichir chaque structure interne définie dans ce maquettage avec plusieurs structures externes.

6.1.4. Squelette

Mon expérience de réalisation rejoint celle décrite dans [Cou 88] au sujet de la difficulté d'apprentissage et de la multiplicité des efforts dans l'utilisation d'une boite à outils :

"...le programmeur doit découvrir le mode d'emploi des services, les assembler correctement et reproduire une partie de cette construction pour chaque système.

Une boîte à outils ne fournit pas nécessairement le niveau de service désiré, elle induit un modèle de contrôle conditionné par le mécanisme de réception des événements.

Si l'architecture du système doit être centrée sur le traitement des événements, alors il doit être possible de réaliser un noyau réutilisable qui serve de charpente minimale à toute une classe de système.

Un squelette d'application regroupe, sous la forme d'un logiciel réutilisable et extensible, les notions de noyau d'exécution et de service étendu. La tâche du programmeur consiste à remplir les trous du squelette et à redéfinir les parties prédéfinies inadaptées au cas particulier du système... "

Après avoir réalisé quelques logiciels, nous pouvons définir les fonctionnements de la visualisation de l'exécution. La conformité de décor et des fonctionnements offrent un ensemble de logiciels aisément manipulables par l'utilisateur et en même temps facilitent la programmation.

A partir de cet ensemble de fonctionnements et la réalisation de la visualisation de diverses structures de données, nous obtenons :

- une bibliothèque de composants pour visualiser : tableau, graphe, chaîne de caractères, liste chaînée.
- un squelette pour animer l'exécution d'un programme Pascal sur Macintosh. Ce squelette comprend les codes des fonctionnalités importantes d'une visualisation de programme, gérée par l'exécution.

  Je présente le schéma fonctionnel du squelette dans la figure 6.10, ses codes se trouvent dans [Arc 89e].
- les ressources utilisées dans le squelette sont décrites dans [Arc 89e]. Elles sont prêtes à l'emploi et permettent un gain de temps appréciable en programmation sur Macintosh.

Ces produits permettent d'une part à un enseignant de réaliser ses maquettes, et d'autre part, aux étudiants de programmer une animation dans le cadre de travaux pratiques.

Ce squelette à été testé et utilisé pour la production des maquettes de la même famille, par exemple :

- **Exploration de tables** et **Recherche Min/Max**
- **Compter les "A"** et **Compter les "LE"**.
- **Les 8 reines** et **Baguenaudier**
- **Tri topologique** et **Parcours de graphes**.

Schéma fonctionnel des modules

La figure 6.10 reflète l'organisation des modules du squelette. Normalement, une unité fonctionnelle est présentée par un module, mais la taille limitée d'un segment avec Lightspeed Pascal oblige parfois le découpage d'une unité fonctionnelle en plusieurs modules.

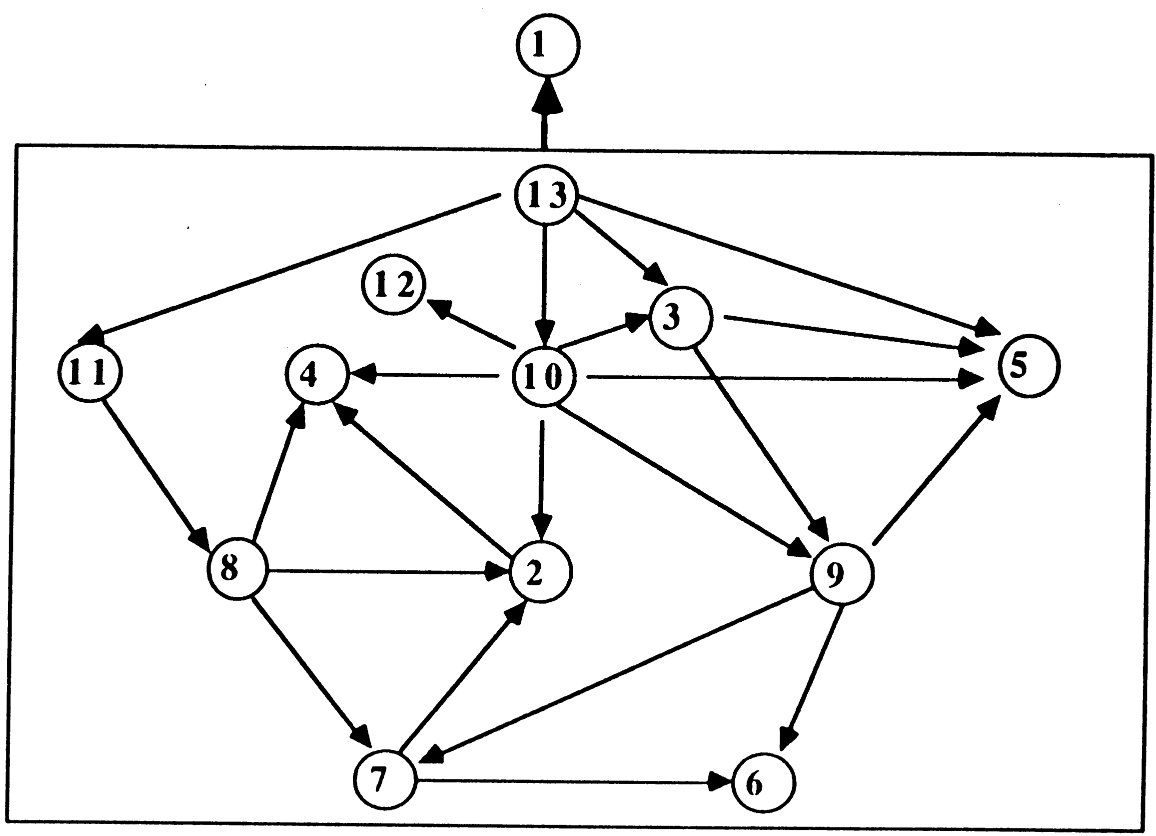


Figure 6.10 Modules fonctionnels du squelette
Note : une flèche de X vers Y indique que le module X utilise les procédures du module Y.
Tous les modules utilisent le module de déclaration des variables globales (module 1).

Voici brièvement la description du contenu de chaque module :

1. Déclaration des variables globales. Toutes les variables qui représentent le "décor" d'une animation d'un algorithme (fenêtres pour chaque activité, bouton, ascenseur, etc) sont déclarées dans ce module et disponibles dans la ressource. Le programmeur doit ajouter des variables représentant les structures des données propre à l'algorithme à animer.

2. Visualisation des structures de données. Ce module contient les sous-programmes de visualisation des structures de données d'une animation.

3. Entrées/sorties de l'utilisateur. Ce module permet de transférer les données saisies graphiquement vers la représentation interne de base de la structure de données (cf. §6.1.3).

4. Entrées/Sorties liées aux représentations internes. Ce module contient des procédures de transfert d'une représentation à l'autre facilitant le travail des programmes sur plusieurs structures internes (cf. §6.1.3).

5. Gestionnaire des fenêtres. Ce module contient les procédures pour préparer, tracer et fermer chaque fenêtre d'activité.

6. Gestionnaire de texte. Ce module est une procédure de préparation et d'affichage d'un texte (d'algorithme) mis dans la ressource.

7. Gestionnaire des événements. Ce module contient des procédures de traitement d'événements qui sont utilisables tout le temps : contexte exécution d'une animation et en dehors de ce contexte (cf. §6.1.2).

8. Animation. Ce module contient les algorithmes à animer avec les appels de la visualisation et l'autorisation d'interruptions pendant l'exécution (vitesse, arrêt, pause).

9. Gestionnaire des menus. Ce module regroupe les procédures de traitement des menus "standard" stockés dans la ressources (, Arcade, Activité, Algorithme, Vue).

10. Gestionnaire d'événement de la souris. Ce module contient toutes les procédures correspondant à l'événement "souris", exclusivement utilisé en dehors de l'exécution d'une animation (traitement d'ascenseur du texte de

l'algorithme dans la lecture, l'affichage des images dans la partie analyse, le clic des boutons recommencer, arrêt, pause et stop, etc).

11. Gestionnaire d'appel a une procédure animée. Ce module prépare le contexte de l'exécution et déclenche une animation (cf. §6.1.2).

12. Algorithme "pur". Ce module contient les algorithmes dans leur version originale, tels quels, sans annotations. Ce module est utile pour le travail en coulisse, par exemple pour comparer les réponses, pour vérifier la manipulation, pour vérifier la validité de données saisies par rapport à la spécification de l'algorithme, etc.

13. Programme principal. Il contient l'initialisation, la boucle principale d'événement et la procédure terminale d'une application.

6.2. Fonctions de visualisation

6.2.1. Changement de vitesse

D'abord, que veut dire "vitesse" d'exécution ? Ce n'est pas la puissance réelle de la machine, car je souhaite visualiser l'exécution du même algorithme sur les mêmes données avec des vitesses différentes.

La vitesse est simulée par un temps d'attente entre deux changements d'état sur l'écran. Si la maquette anime plusieurs algorithmes traitant le même problème, l'exécution avec la même "vitesse" doit exprimer le temps d'exécution réel de chaque algorithme.

Par exemple pour les algorithmes de tris : je mets toujours la temporisation (donc cela donne toujours le même temps d'attente) avant chaque affectation :

```
efface T[i];
attente;
T[i] := T[j];
affiche T[i];
```

Figure 6.11 Procédure qui "contrôle" la vitesse de l'exécution.

Une boucle d'attente se réalise de deux façons différentes :

1. Une boucle vide basée sur un compteur :

```
procedure attente;
{ ntemps est une variable globale qui détermine la vitesse          }
{ cette valeur peut être modifiée par l'utilisateur, traitée par    }
{     le gestionnaire d'événement                                   }
   var i: integer;
begin
   i := 1;
   while i <=ntemps do
         i := i + 1;
end;
```

Figure 6.12 Procédure qui "contrôle" la vitesse de l'exécution, basée sur un compteur.

Cette solution est bien connue, mais inadaptée car l'exécution du même programme dépend de la machine. Pour la même valeur de "ntemps" (Figure 6.12) l'exécution est inobservable (trop rapide) sur Mac II, mais ennuyante (trop lente) sur Mac Plus.

2. Une boucle vide basée sur le temps.

```
procedure attente;
{  ntemps est une variable globale qui détermine la vitesse           }
{  cette valeur peut être modifiée par l'utilisateur,                 }
{     traitée par le gestionnaire de l'événement                      }
{  tickcount est une fonction de la boite à outil, qui envoie le      }
{  temps écoulé depuis la mise sous tension de l'ordinateur           }

   var
      debut : longint;
begin
   debut := tickcount;
   while (tickcount-debut)<=ntemps do
      {      action vide          } ;
end;
```

Figure 6.13 Procédure qui "contrôle" la vitesse de l'exécution, basée sur l'horloge.

Cette solution est retenue car, grâce à l'accès à l'horloge, elle est indépendante de la machine.

A la réalisation, cette boucle d'attente est intégrée dans le gestionnaire d'interruption : le programme utilise cette temporisation pour enregistrer l'événement concernant le changement des paramètres de l'exécution (vitesse, arrêt, pause).

6.2.2. Exécution pas à pas

Le problème posé par l'exécution pas à pas d'une animation d'algorithme n'est pas sa programmation, mais la définition du pas. Pour un interpréteur, un pas est égal à une instruction, une unité syntaxique. Le pas dans une animation de

l'algorithme est une unité sémantique déterminé par l'enseignant. Si bien que le découpage d'un programme en pas se fait manuellement par l'annotation du programme à animer.

Avec Lightspeed Pascal, l'exécution pas à pas doit être simulée (cf. § 6.2.2). Il faut d'abord déterminer ce qu'est un pas (cf. § 5.2.6), puis visualiser l'exécution à chaque pas. Le programme déclenche le pas suivant après chaque demande.

Pas suivant

Deux implémentations sont possibles :

1. Comme dans la partie 6.2 : je découpe la procédure en plusieurs pas f_1, f_2, f_3, ... f_n, et réexécute f_1, f_2, ... f_i. La visualisation sur l'écran n'est effectuée que pour le pas courant i.
   Ceci est illustré bien sur l'écran, par exemple avec **Pattern matching.**
   Cette solution avait été choisie pour tester le problème des boucles d'événements décrit précédemment.

2. Le pas est analogue à la pause automatique :
   Le déclenchement du pas suivant est effectué par le programme dans la deuxième boucle d'événements, sans attendre la demande de l'utilisateur (le pas est déterminé par le programme). On reste dans la procédure tant qu'il n'y a pas d'arrêt ou tant que la fin de la procédure n'est pas atteinte.
   Cette solution est préférable car elle ne pénalise pas le temps d'exécution. Elle n'a pas été implémentée car nous avons préféré une "pause" qui laisse la détermination du pas à l'utilisateur. (cf. **Dessin récursifs**)

6.3. Visualisation des structures des données

6.3.1. Mémorisation ou calcul

Pour visualiser une structure de données qui se compose d'un ensemble de valeurs, il faut déterminer la position de chaque valeur sur l'écran, par le calcul ou par la mémorisation.

Le calcul est simple pour une structure interne contiguë (tableau, matrice), car les positions des différents éléments sont calculées à partir du premier élément. C'est le choix que j'ai effectué pour le tableau ou la matrice.

Le calcul est complexe pour une structure dynamique ou chaînée. Pour une liste chaînée ou un graphe, chaque enregistrement est attribué par sa position sur l'écran : la déclaration de variable inclue la mémorisation de l'emplacement.

La mémorisation nécessite de la place mémoire et la détermination des valeurs à mémoriser : soit manuellement (par exemple données par l'utilisateur), soit par un calcul (une fois).

Par exemple, un nœud du graphe est déclaré comme suit :

```
type
nœudptr = ^noeud;
nœud =  record
          id : char
          rnœud : rect; { position de rectangle qui          }
                        { représente le nœud sur l'écran     }
          next : nœudptr;
        end;
```

Figure 6.14 Déclaration d'un nœud d'un graphe.

L'environnement Macintosh permet la saisie de données en mode graphique. C'est esthétique, agréable, convivial pour l'utilisateur et évite l'utilisation systématique du clavier. La saisie récupère directement la position d'affichage des données sur l'écran. Pour l'affichage du graphe, par exemple, on laisse

l'utilisateur ranger les noeuds si bien qu'un calcul compliqué pour afficher le graphe [GKS 83], [GNV 88] est inutile.

Avec la mémorisation dans la déclaration, je peux utiliser directement les procédures de QuickDraw pour tracer une vue, mais l'inconvénient est que le module d'affichage dépend du module de déclaration.

Pour la plupart des structures de données, la mémorisation est plus simple : les procédures complexes d'affichage qui en général pénalisent la performance, sont inutiles.

Par contre la mémorisation est inapplicable pour l'arbre binaire : l'insertion ou la suppression d'un nœud modifie dynamiquement l'emplacement des autres nœuds, tout en conservant un arbre binaire "esthétique" [WeS 79]. Le recalcul des positions de tous les nœuds à chaque affichage est indispensable. La visualisation d'arbre binaire est étudiée dans notre équipe au travers de la maquette manipulatoire "Arbre Binaire" (cf. Chapitre 2). Je ne travaille pas sur la structure arborescente.

6.3.2. Association entre l'image écran et le programme annoté

La désignation sur l'écran ne donne que les coordonnées d'un point (X,Y) sur l'écran. C'est au programmeur d'associer ce point à une entité de son programme. Dans mes maquettes, ce point doit être associé à :

a. la structure de **données** interne du programme :

   la reconnaissance d'appartenance d'un point à une structure de données peut s'effectuer par la recherche d'une liste d'objets sur l'écran. Cette association est facilitée par notre représentation de la structure interne (cf. § 6.1.3).

Par exemple :

- comment vérifier la désignation d'un nœud ou d'un arc sur le graphe ? Il suffit de faire une recherche sur la liste des nœuds et des arcs (**Tri topologique** et **Parcours de graphes**).
- comment associer la position d'un trait à l'indice et la longueur du trait à la valeur d'un élément ? Il suffit de calculer sa position par rapport à l'origine (**Tri internes** et **Exploration de tables**).

b. une trace ou une image qui reflète un "**état**" d'une procédure.

Dans **Dessins récursifs**, la reconnaissance de désignation faite par l'élève réexécute la procédure de tracé d'une courbe et refait le calcul géométrique à chaque fois. Nous exigeons dans cet exercice, la désignation du milieu d'un segment de la ligne tracée, pour éviter la confusion d'évaluation antre AB ou AC. Un point (X,Y) est accepté ou erroné pendant que la procédure tracer_carré est en train de tracer le segment AB si le calcul de la distance de (X,Y) au milieu du segment AB convient. De plus, il faut prévoir une certaine tolérance que cette distance ≤ δ, δ dépend de la longueur du segment.

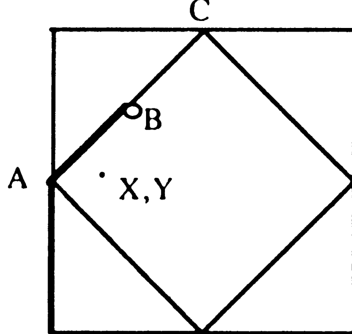


Figure 6.15 Evaluation du point (X,Y) par rapport au milieu de la ligne AB.

6.3.3. Rafraîchissement de la fenêtre

Le *Window manager* traite presque tous les traitements des fenêtres (superposition, déplacement, agrandissement, raccourcissement, ...), sauf le rafraîchissement de la fenêtre ! Cette tâche (bien sûr compliquée) est laissée au soin du programmeur.

La mise à jour du texte sur une fenêtre est automatique, faite par le système.

La mise à jour de dessin pose un problème plus délicat à gérer : soit retracer complètement la fenêtre, soit mémoriser le *bit map* de l'écran à tout moment. Quel choix effectuer sachant que la première solution prend du temps et la deuxième est gourmande en place mémoire ?

Retracer l'écran

L'image écran représente :

- soit l'état courant de la structure de données, par exemple : **Tri internes.** Retracer l'écran est facile, ce n'est qu'un appel à la procédure qui visualise la valeur courante de la structure.
- soit les traces de l'exécution (**Dessins récursifs**, **Pattern matching**), donc l'état courant d'une procédure.

  Le découpage de la procédure en pas à pas permet de retracer l'écran en réexécutant jusqu'au pas courant (**Pattern matching**).

  Dans **Dessins récursifs**, où la procédure n'est pas découpée en pas à pas, il faudrait mémoriser l'état courant : la dernière adresse d'instruction exécutée. En ce moment, je ne m'intéresse pas à cette technique : voilà pourquoi le menu est désactivé pendant l'exécution de ce logiciel, ce qui évite la mise à jour de fenêtre après la fermeture ou le déplacement d'un accessoire de bureau...

Mémorisation

La mémorisation d'une image bitmap de l'écran pour une sauvegarde rapide nécessite une variable de la taille de 22K octets. Cette variable doit être globale car la boucle d'événements principale se situe dans le programme principal. Or la zone de variables globales en Lightspeed Pascal est limité à 32K octets. Cette solution est donc irréalisable à cause du peu de place mémoire disponible pour les autres variables de l'application.

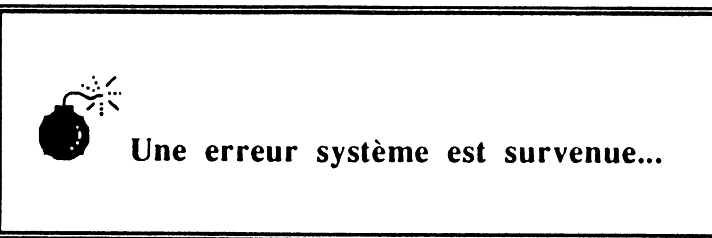
La mémorisation du bitmap de l'écran par allocation d'une variable du type pointeur sur le bitmap occupe très peu de mémoire, mais nécessite la copie de l'image écran bit par bit, ce qui pénalise la performance. Cette mémorisation est adaptée pour un logiciel qui met à jour l'image écran à partir de l'intervention de l'utilisateur (par exemple : MacPaint, MacDraw), car la manipulation de l'utilisateur est faible par rapport à la vitesse d'affichage. Cette mémorisation n'est pas une bonne solution pour l'animation des programmes en temps réel car :

- le temps de transfert des images bitmap (bit par bit) diminue la performance de l'exécution réelle et perturbe la véritable vitesse d'exécution,
- dans une animation de l'exécution, l'état de l'écran se change automatiquement avec une période plus courte que le transfert.

6.4. Autres problèmes

Des petits problèmes rencontrés pendant la réalisation. Parfois frustrants.

6.4.1. Erreur de système



La bombe bien connue est déclenchée à l'apparition d'une erreur système. Il faut alors réinitialiser le système : perte de temps, perte de travail éventuelle.

La mise au point des programmes dans cet environnement est beaucoup plus délicat que dans un environnement "traditionnel" [Wes 86]. Le travail obligé avec des pointeurs, ou même avec des pointeurs pointant sur un autre pointeur *(handle)*. La gestion mémoire est tellement dynamique. Il n'y a pas d'unité de gestion mémoire assurant la continuité si le programme fonctionne mal. Une petite erreur d'adressage suffit pour déclencher une bombe !

Le langage Pascal, connu comme un langage fortement typé et structuré, perd cette caractéristique dans l'environnement Macintosh. La boite à outils fournit des fonctions pour changer le type d'un *handle* à l'autre type ; on peut accéder et interpréter une zone mémoire comme des octets.

Le langage Pascal dans cet environnement est très paradoxal : langage machine ou langage évolué ?

6.4.2. Difficulté du test

En génie logiciel, la notion de jeu de test est importante. On détermine des jeux de données particuliers, pour pouvoir cheminer dans tous les cas intéressants du programme. Or, dans un *event driven system*, l'accès d'un chemin est déterminé par le système. Exemple : l'appel de la procédure qui

retrace une fenêtre pour une mise à jour de l'écran n'est pas fait par le programme mais par le *toolbox event manager* qui dit "quand" cette procédure doit être exécutée.

La programmation par événement oblige le programmeur à bien spécifier car le test du programme par la méthode exhaustive est très difficile à mettre au point.

Si un cas bizarre se produit à l'exécution, il est difficile à reproduire. On peut faire "exactement" les mêmes manipulations, sans être sûr que le système réagisse dans les mêmes conditions (rappelons que certains événements sont déclenchés automatiquement par le système et mis dans la même liste *FIFO* avec les événements extérieurs).

6.4.3. Convention standard de l'interface utilisateur

Toutes les applications dans l'environnement Macintosh ont le même style d'interface utilisateur. La programmation dans cet environnement exige des conventions standards (voir le chapitre "Guide de l'utilisateur" dans [App 85]). Par exemple, les menus , Fichier et Edit doivent apparaître comme les trois premiers (à gauche) de la barre de menu, un bouton équivalent à la touche "retour" doit être entouré, etc... Le respect rigoureux de ces règles permet une programmation conventionnelle sur Macintosh.

Pourtant, aucune de mes maquettes ne respecte cette conformité. Le menu "Fichier" est remplacé par "Arcade" pour revenir au laboratoire ARCADE car mes maquettes n'effectuent aucune opération sur fichier, le menu "Edit" est supprimé car mes applications n'utilisent jamais couper/coller. Pourquoi laisser ces deux menus grisés en permanence puisqu'ils sont inutiles ? Les règles standards sont utiles mais la spécificité d'une application justifie et oblige parfois la transgression de ces règles standards.

6.4.4. Retouche des dessins

Le travail en mode point à point s'il doit être fait soigneusement, prend du temps et demande de l'expérience.

Les spécifications des procédures de dessins dans QuickDraw sont parfois incohérentes. C'est au programmeur de corriger "à la main" sa visualisation pour faire disparaître ces incohérences. Illustrons au travers de ces deux exemples :

1. Regardons le résultat de l'exécution des instructions suivantes :

PROGRAMME	RESULTAT

procedure carreligne;
  var r : rect;
begin
{ définir le rectangle }
    setrect(r, 200, 200, 250, 250);
 	
{ tracer le rectangle }
    framerect(r);
 	
{ va au coin inférieur gauche }
    moveto(200, 250);
 	
{ tracer une ligne horizontale de gauche à droite}
    lineto(250, 250);
end;	

Figure 6.16 Tracé d'un carré et d'une ligne.

La vue agrandie du résultat est :

Figure 6.17 Agrandissement du carré de la figure 6.16.

La retouche est nécessaire car la ligne n'est pas ajustée aux dimensions du carré, et pourtant les extrémités sont bien définies de manière identique. Imaginer la complexité du problème de visualisation quand les points sont le résultat d'un calcul ! Ce genre de "petit" défaut graphique est jugé équivalent à une faute d'orthographe dans un texte [NeS 86].

2. La taille de la plume :

PROGRAMME	RESULTAT

procedure trypen;
var
x, y : integer;
longue : integer;
begin
x := 100;
y := 200;
longue := 50;
 	
pensize(4, 4);
penpat(black);
{ de bas en haut : A --> B }
moveto(x, y);
lineto(x, y - longue);
pensize(1, 1); { changer la taille }
penpat(black);
moveto(x + 12, y);
lineto(x + 12, y - longue);
 	
x := 200;
pensize(4, 4);
penpat(black);
{ de haut en bas : C --> D }
moveto(x, y);
lineto(x, y + longue);
pensize(1, 1); { changer la taille }
penpat(black);
moveto(x + 12, y);
lineto(x + 12, y + longue);
end;
	B
	A
	C
	D

Figure 6.18 Tracés de deux lignes de même longueur avec différentes tailles de la plume.

En agrandissant, si la longueur est égale à 3, on obtient :

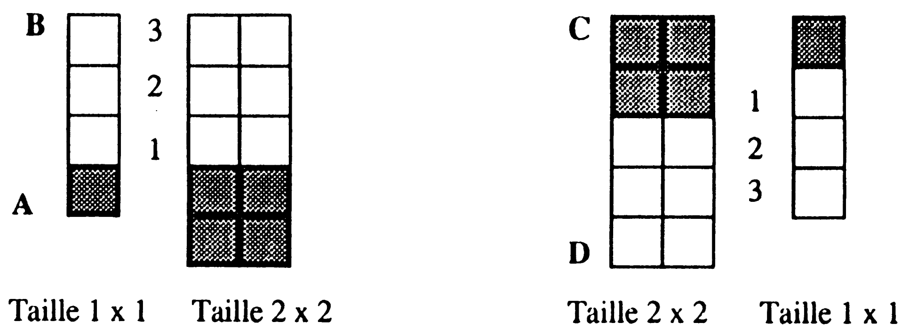


Figure 6.19 Effets de différentes tailles de la plume.

Imaginons que les deux lignes représentent deux valeurs à comparer (par leurs longueurs) ! La correction dépend de la direction de la trace. Dans la première trace (de bas en haut) la correction est faite au point de départ. Dans la deuxième trace (de haut en bas) la correction est faite au point d'arrivée.

Pour régler ce problème, **Tris internes** traite toujours une ligne avec une largeur variable par un rectangle puisqu'il possède toujours des bornes fixes.

6.4.5. Lightspeed Pascal comme outil de développement

Question sur Lightspeed Pascal

Pourquoi Lightspeed Pascal [Thi 86] et pas un langage orienté objet qui serait plus "proche" du *event driven system* et pas non plus le langage C qui serait plus proche du système ?

Parce que nous voulons animer directement des procédures Pascal et qu'un compilateur du même langage nous facilite l'annotation.

En résumé, parce que mon objectif était de réfléchir sur la visualisation de l'exécution des programmes, le choix d'outils de travail était secondaire dans un premier temps.

Les problèmes rencontrés

L'une des caractéristiques d'une application sur Macintosh est la séparation entre les codes de programmes et ses ressources. En théorie, la création et la modification des programmes sont indépendantes du travail sur des ressources et vice versa. La réalité est toute différente dans mon travail : les ressources sont créées à partir du programme.

Position exacte des items (bouton, texte, images, etc...)

Lightspeed Pascal ne permet pas l'édition des ressources, si bien que le travail avec un éditeur de ressource comme ResEdit est nécessaire. La création des ressources sous ResEdit, avec une disposition approximative des items (textes, boutons, choix, etc), permet de les intégrer sous Lightspeed Pascal. L'exécution pendant le maquettage détermine les dispositions exactes, remises ensuite dans la ressource par ResEdit.

Programmation des images

L'utilisation d'images en Lightspeed Pascal est très banale : prendre l'image dans la ressource et afficher à l'écran avec la procédure *"DrawPicture"*. L'itinéraire de production et d'édition d'une l'image est plus complexe puisque Lightspeed Pascal et ResEdit ne permettent pas ce travail. Il faut donc utiliser un logiciel de dessin (MacPaint, SuperPaint ou MacDraw), passer par l'album, mettre dans la ressource (ResEdit) et programmer (Lightspeed Pascal).

Pour la précision, de même que la disposition exacte des items, dans **Dessins récursifs**, les images stockées dans la ressource sont le résultat du programme ! La modification d'images est le résultat d'une chaîne de traitement longue : programmer avec Lightspeed Pascal, exécuter, capturer l'écran, retravailler avec SuperPaint, passer par l'album, remettre dans la ressource, réexécuter, etc....

Pourquoi dans ce cas, ne pas mettre une image dans la ressource directement par le programme ? Si l'image est stockée dans la ressource par le programme, le système mémorise l'image comme une série d'exécution des procédures ; en conséquence l'image se retrace dynamiquement pendant l'affichage ce qui n'est pas voulu pour une visualisation d'image "statique" et d'un seul coup. Si l'image est stockée par "couper/coller", elle est stockée comme une seule image entière, donc elle apparaît d'un seul coup. Pourtant, on affiche avec la même procédure (*"DrawPicture"*)...

Programmer avec Lightspeed Pascal prend du temps car le programmeur doit faire des aller-retour du Lightspeed Pascal aux autres logiciels.

6.5. Résumé

L'animation des programmes est une application qui est très liée à l'algorithme à animer ; l'automatisation est impossible. D'une part, le travail pédagogique dépend évidemment des choix pédagogiques de l'enseignant. D'autre part, le travail de programmation nécessite une analyse pour chaque algorithme. Enfin, les corrections apportées aux procédures de calcul ou d'affichage doivent se faire manuellement, ce qui est difficile à généraliser.

Une fois les primitives de visualisation des structures de données disponibles, le travail de maquettage consiste à construire et à tester plusieurs séquences d'animation pour que la sémantique visuelle de chaque variable de programme correspond à son rôle dans l'algorithme.

Conclusion

Nous avons montré en quoi l'exécution des programmes est un problème particulier dans l'enseignement de la programmation. La conception des algorithmes ne doit pas être basée principalement sur cette notion d'exécution. Cependant le programmeur doit avoir conscience, de manière précise, des phénomènes produits par les programmes qu'il conçoit.

Nous avons alors défini un moment pédagogique propre à ce problème. Ce moment pédagogique est consacré essentiellement à l'observation du processus séquentiel d'exécution des programmes. Cette observation doit être faite sur une représentation visuelle des valeurs et des états manipulés.

Ce moment pédagogique nécessite des outils. L'ordinateur en est manifestement le meilleur support. Notre travail concret a donc consisté à définir et à réaliser des logiciels de visualisation d'exécution de programmes. Nous avons voulu que ces logiciels recouvrent, le plus possible, les principaux thèmes d'un enseignement fondamental de la programmation impérative : familles d'algorithmes et techniques de représentation des données.

Une dizaine de logiciels ont ainsi été réalisés et expérimentés. Ils illustrent divers thèmes algorithmiques (itération, récursivité, recherche, tri interne, reconnaissance d'un motif, tri topologique, parcours de graphes) et diverses représentations des données (contiguïté, chaînage, tableau, liste, graphe). De ces logiciels, nous avons extrait une bibliothèque d'outils de visualisation et un squelette d'animation de programmes accompagné de ses ressources.

Ces logiciels s'intègrent dans un ensemble de logiciels destinés à l'enseignement de la programmation ; ceux-ci constituent le laboratoire ARCADE. En effet, la visualisation de l'exécution n'est qu'un élément d'apprentissage. Plusieurs outils et activités sont nécessaires pour aborder un

sujet. La diversité des outils permet à l'étudiant de privilégier ceux qui correspondent le mieux à sa manière d'apprendre.

Perspectives

Les logiciels réalisés doivent être intégrés à un enseignement existant pour en faire une évaluation plus complète. L'Institut Grenoblois d'Etudes Informatiques dispose maintenant d'équipements appropriés et le laboratoire ARCADE sera mis partiellement à la disposition des étudiants dès la rentrée universitaire 1989. Je compte utiliser ces logiciels dans mon propre enseignement à l'Institut de Technologie de Bandung.

Nous voulons permettre aux étudiants de réaliser eux-mêmes l'animation de leurs programmes. D'une part, pour fournir un exercice de lecture : les étudiants doivent annoter leurs programmes (déterminer les états significatifs et les données caractéristiques, choisir les vues). D'autre part, pour permettre une observation plus personnalisée. Il suffit de donner accès à la bibliothèque de visualisation et au squelette d'animation. Cette activité constituera une nouvelle forme complémentaire de travaux pratiques de programmation. En outre, les produits réalisés par les étudiants pourraient enrichir le laboratoire ARCADE.

Nous voulons également permettre aux enseignants d'animer les algorithmes de leurs cours sans pour autant alourdir leurs préparations. Un enseignant maîtrise les bases du langage Pascal. A l'aide de la bibliothèque de visualisation et du squelette, animer un programme demande surtout une réflexion pédagogique et peu de programmation.

S'appuyant sur la définition d'exemples types et de schémas algorithmiques, il serait intéressant de modéliser la visualisation des programmes. A partir de l'étude d'un grand nombre d'algorithmes tirés d'ouvrages tels que [AHU 87],

[HoS 84], [Knu 69], [Knu 73], [Sed 86], on doit pouvoir classifier les différents types de visualisation et en déduire un certain nombre de schémas.

Pour chaque schéma type de visualisation d'algorithmes, on recherche différentes vues. L'expérimentation a montré que pour une même structure de données, la vue significative dépend de l'algorithme. Pour le tableau, par exemple, la représentation par des traits est pertinente pour le tri, mais inadéquate pour la recherche. On pourrait alors imaginer une étude plus poussée associant les méthodes et outils de représentation graphique aux schémas algorithmiques et structures de données.

Nous aimerions définir un outil de production d'animations plus systématique sans perdre la précision pédagogique.

L'automatisation du processus de visualisation, au niveau de la compilation, gommerait les efforts de programmation et permettrait d'animer un programme sans insérer d'annotations. Mais cette automatisation ne permettrait pas d'obtenir une visualisation pédagogique individualisée car les valeurs et les états significatifs ne pourraient pas être déterminés à partir de la syntaxe du programme.

Un système d'animation d'algorithmes permet de générer les vues en traitant les annotations insérées manuellement, donc demande un investissement important pour la compréhension de cet environnement spécialisé. Un tel système ne peut pas être confié à des étudiants d'un niveau élémentaire.

Travailler avec un compilateur général pourrait un compromis satisfaisant, à condition de systématiser la démarche de production, d'enrichir la bibliothèque et le squelette et de formaliser un cahier des charges plus précis.

A partir d'une réflexion sur l'enseignement de la programmation, je me suis intéressée au seul problème de l'exécution des programmes et j'ai proposé un certain nombre de logiciels de visualisation. La conception de ces logiciels

résulte plus d'une recherche sur la pertinence des techniques de visualisation que sur un processus pédagogique particulier. Cette recherche étant faite et des premières réponses étant proposées, je souhaite replacer ces résultats dans un contexte pédagogique cohérent.

La base du projet ARCADE était l'étude des rôles pédagogiques possibles de l'ordinateur dans l'enseignement de la programmation. Je souhaite prolonger cette recherche en me situant dans le cadre d'un cours très précis, par exemple celui défini à l'Institut Grenoblois d'Etudes Informatiques dans [ScP 88]. Il s'agira alors de réaliser un environnement de soutien à tous les moments pédagogiques de l'enseignement. L'observation de l'exécution des programmes prendra sa place, naturellement, dans l'ensemble des activités de cet environnement.

Références bibliographiques

[AcP 89]
ACKERMANN E. C. et POPE W. R. : "Computer Aided Programme Design Experiments : Diagrammatic Versus Textual Method", ACM SIGCSE Bulletin, vol. 21, n° 1 (février 1989), pp. 117 - 121.

[Agu 85]
AGUIRRE J-L. : "Réalisation d'un didacticiel en vraie grandeur pour l'enseignement de l'algorithmique et de la programmation", Rapport de 3ème année ENSIMAG, juin 1985.

[AHU 87]
AHO A., HOPCROFT J. et ULLMAN J. : "Data structures and Algorithmes", Addison Wesley, 1983.

[AnR 85]
ANDERSON J. R. et REISER B. J. : "The LISP Tutor", Byte, (avril 1985), pp. 159 - 175.

[App 85]
Apple Computer Inc : "Inside Macintosh", volume I, II, III, IV, V, Addison Wesley, 1985.

[Arc 86a]
CAGNAT J-M. : "Présentation du projet Arcade", Rapport Arcade n° 1, avril 1986.

[Arc 86b]
CAGNAT J-M., GUERAUD V. et PAINVIN S. : "Vers un laboratoire de Programmation", Rapport Arcade n° 3, juillet 1986.

[Arc 86c]
GUERAUD V. et LIEM I. : "Les salons EDUCATEC ET FORMATION 86, la cité des Sciences et de l'Industrie, le CESTA, le sixième congrès de l'EAO", Rapport Arcade n° 6, février 1986.

[Arc 88a]
LIEM I. et PEYRIN J-P. : "Le logiciel Tri Internes", Rapport Arcade n° 7, février 1988.

[Arc 88b]
LIEM I. et PEYRIN J-P. : "Le logiciel Dessins Récursifs", Rapport Arcade n° 8, février 1988.

[Arc 88c]
PAINVIN S. : "La gare de triage : un dessin animé pour illustrer un algorithme", Rapport Arcade n° 9, mars 1988.

[Arc 88d]
CAGNAT J-M. : "Apprentissage guidé d'un algorithme : le tri par minimum *(le logiciel <<Tri Guidé>>)*", Rapport Arcade n° 10, août 1988.

[Arc 88e]
CAGNAT J-M. : "Meccano de Tri : une boîte d'outils pour trier des cubes", Rapport Arcade n° 11, août 1988.

[Arc 88f]
GUERAUD V. et PEYRIN J-P. : "Un jeu de rôles pour l'enseignement de la programmation", Communication présentée au Colloque Francophone sur la Didactique de l'informatique, Rapport Arcade n° 12, Paris 1-2-3 septembre 1988.

[Arc 88g]
CAGNAT J-M., GUERAUD V., PAINVIN S., LIEM I. et PEYRIN J. P. : "Un laboratoire pour l'enseignement de la programmation", Communication présentée au Colloque sur l'Evolution de l'Outil Informatique à l'université", Poitiers, 14-15-16 septembre 1988, Rapport Arcade n° 13.

[Arc 88h]
CAGNAT J-M., PAINVIN S., LIEM I. et PEYRIN J-P. : "Pour quelques approches de plus ... ", article paru dans le n° 52 (novembre 88) du Bulletin de l'EPI, Rapport Arcade n° 14, 1988.

[Arc 89a]
LIEM I. : "La recherche dans une séquence : Explorations de tables, Recherches Minimum et Maximum, Pattern matching", Rapport Arcade n° 15, 1989.

[Arc 89b]
LIEM I. : "Traitement séquentiel : Machines caractères, Compter Les "A", Compter les "LE"", Rapport Arcade n° 16, 1989.

[Arc 89c]
LIEM I. : "Les graphes : Tri topologique et Parcours de graphes", Rapport Arcade n° 17, 1989.

[Arc 89d]
LIEM I. : "Jeux récursifs : Jeu de Baguenaudier et Les huit reines", Rapport Arcade n° 18, 1989.

[Arc 89e]
LIEM I. : "Visualisation des programmes : Squelette d'animation des programmes et bibliothèque de visualisation des structures des données", Rapport Arcade n° 19, 1989.

[Ars 83]
ARSAC J. : "Les bases de programmation", Dunod Informatique, Paris, 1983.

[AuL 86]
AUGENSTEIN M. et LANGSAM Y. : "Graphique Display of Data Structures on the IBM PC", ACM SICSE Bulletin, vol. 18, n° 1 (février 1986), pp. 73 - 81.

[AuL 88]
AUGENSTEIN M. et LANGSAM Y. : "Automatic Generation of Graphic Display of Data Structures Through a Preprocessor", ACM SIGCSE Bulletin, vol. 20, n° 1 (février 1988), pp. 148 - 314.

[BaB 86]
BARNETT M. P. et BARNETT S. J. : "Animated Algorithms : A Self Teaching Course in Data Structures and Algorithms", Mac Graw Hill, 1986.

[Bac 86]
BACKHOUSE R. C. : "Program Construction and Verification", Prentice Hall, 1986

[Bae 75]
BAECKER R. : "Two Systems Which Produce Animated Representations of the Execution of Computer Program", ACM SICSE Bulletin, vol. 7, n° 1 (février 1975), pp. 158 - 167.

[Bae 81]
BAECKER R. : "Sorting Out Sorting" , ACM SIGGRAPH 1981, vol. 15, n° 4 (décembre 1981), p. 313.

[Bae 86]
BAECKER R. : "An Application Overview of Program Visualisation", Computer Graphics, vol. 20, n° 4 (juillet 1986), p. 325.

[BaK 87]
BARNES C. M. et KIND G. A. : "Visual Simulations of Data Structure during lecture", ACM SIGCSE Bulletin, vol. 19, n° 1 (février 1987), pp. 267 - 276.

[BHK 85]:
BROWN G. P., HEROT C. F., KRAMLICH D. A., SOUZA P. : "Program Visualisation : Graphical Support for software development", IEEE Computer, vol. 18, n° 8 (août 1985), pp. 27 - 35.

[BiC 81]
BIONDI J et CLAVEL G : "Introduction à la programmation", Masson, 1981.

[BMS 86]
BARNES, MICHAEL, SHU et all : "A Computer Courseware Factory", ACM SIGCSE Bulletin, vol. 18, n° 1 (février 1986), pp. 318 - 328.

[Boe 82]
BOEHM B. W. : "Les facteurs de coût du logiciel", T.S.I. vol. 1, n° 1, 1982, pp. 5 - 24.

[BoM 77]
BOYER R.S. et MOORE J.S. : "A Fast String Searching Algorithm", Communication of the ACM, vol. 20, n° 10 (octobre 1977), pp. 762 -772.

[Bro 84]
BROWN M. H. : "A System for Algorithm Animation", ACM Computer Graphics, vol 18, n° 3 (juillet 1984), pp. 177 - 185.

[Bro 87]
BROWN M. H. : "Algorithm Animation", thèse de Ph. D de l'Université de BROWN, USA, mai 1987.

[Bro 88]
BROWN M. H. : "Exploring Algorithms Using Balsa - II", IEEE Computer, vol. 21, n° 5 (mai 1988), pp. 14 - 36.

[BrS 84]
BROWN M. H. et SEDGEWICK R. : "Progress Report : Brown University Instructional Computing Laboratory", ACM SIGCSE Bulletin, vol. 16, n° 1 (février 1984), pp. 91 - 101.

[BrS 85]
BROWN M. H. et SEDGEWICK R. : "Techniques for Algorithm Animation", IEEE Software, vol. 2, n° 1 (janvier 1985), pp. 28 - 39.

[BSS 86]

BARSTOW D. R., SHROBE H. E., SANDEWALL E. : "Interactive Programming Environments", Mac Graw Hill International Edition, 1986.

[BST 86]

BULGREN W. G., SCHEPPE E. A. et THURMAN T. : "An Improved Introduction to Computing Emphasizing the Development of Algorithms and Using The Apple Macintosh Pascal", ACM SIGCSE Bulletin, vol. 18, n° 1 (février 1986), pp. 253 - 256.

[Cha 87]

CHANG S. K. : "Visual Languages : A Tutorial and Survey", IEEE Software, vol. 4, n° 1 (janvier 1987), pp. 29 - 39.

[ClR 83]

CLARC B. E. J. et ROBINSON S. K. : "A Graphically Interacting Program Monitor", The Computer Journal, vol. 26, n° 3 (1983).

[CoA 87]

COOK R. P. et AULETTA R. J. : "StarLite, A Visual Simulation Package for Software Prototyping", Proceeding of the ACM SIGSOFT/SIGPLAN, vol. 22, n° 1 (janvier 1987), pp. 102 - 110.

[Coi 74]

COITIER M. : "Le Baguenaudier", Le Pentamino, Irem de Grenoble, n° 2 (1974), pp. 85-94.

[Cou 88]

COUTAZ J. : " Interface Homme-Ordinateur, Conception et Réalisation", Thèse d'Etat de l'Université Joseph Fourier de Grenoble I, décembre 1988.

[Did 88]

Actes de "Colloque sur la Didactique de l'Informatique", Paris, 1 - 3 Septembre 1988.

[Dij 71]
DIJKSTRA E.W. : "A short introduction to the art of programming", Notes de cours, EWD 316, Department of Mathematics, Eindhoven, Pays Bas, 1971.

[Dij 76]
DIJKSTRA E.W. : "A Discipline of Programming", Prentice Hall, 1976.

[Dij 89]
DIJKSTRA E. W. : "On the Cruelty of Really Teaching Computing Science", The SIGCSE Award Lecture, ACM SIGCSE Bulletin, vol. 21, n° 1 (février 1989), pp. xxiv - xxxix.

[DMS 84]
DELISLE N. M., MENICOSY D. E., SCHWARTZ M. D. : "Viewing a Programming Environment as a Single Tool", Proceedings of the ACM SIGSOFT/SIGPLAN, Software Engineering notes (mai 1984), SIGPLAN Notices (mai 1984), pp. 49 - 56.

[Dou 90]
DOURIS : article à apparaître, IREM, Grenoble, 1990.

[DuG 88]
DUPUIS C. et GUIN D. : "Récursivité en LOGO dans une classe", Actes du premier colloque franco-allemande de didactique, (1988), pp. 267-274.

[Dui 88]
DUISBERG R.A. : "Animation Using Temporal Constraints : An Overview of the Animus System", Human Computer Interaction, vol 3, n°3, Lawrence Erlbaum Association Publishers, 1987 - 1988, pp. 275 - 307.

[EAO 87]
Actes de Congrès francophone sur l'enseignement assisté par l'ordinateur, Cap d'Agde, 23-24-25 mars 1987.

[EIO 88]
ELENBOGEN B.S. et O'KENNON M.R. : "Teaching Recursion Using Fractals in Prolog", ACM SIGCSE Bulletin, vol. 20, n° 1 (février 1988), pp. 263 -266.

[FiG 84]
FINZER W. et GOULD L. : "Programming By Rehearsal", Byte (juin 1984), pp. 187 - 210.

[FJM 84]
FISCHER C. N., JOHNSON G. F., MAUNEY J., PAL A., STOCK D. L. : "The POE Language Based Editor Project", ACM SIGSOFT/SIGPLAN Notes, SIGSOFT vol.9, n° 3, SIGPLAN vol. 19, n° 5 (mai 1984), pp. 21 - 29.

[FoM 86]
FOLEY J. et McMATH C. H. : "Dynamic Process Visualisation", IEEE Computer Graphics and Applications, vol. 6, n° 3 (mars 1986), pp. 16 - 25.

[Fre 88]
FRENKEL K.A. : "The Art and Science of Visualizing Data", Communication of ACM, vol. 31, n° 2 (février 1988), pp. 111 - 121.

[Ger 77]
GERBIER A. : "Mes Premières Constructions de Programmes", Springer Verlag, 1977.

[GFV 81]
GURWITZ R. F., FLEMING R. T., van DAM A. : "MIDAS : A Microprocessor Display and Animation System", IEEE Transactions on Education, vol. E-24, n° 2 (mai 1981), pp. 126 - 132.

[Gia 87]
GIANNOTTI E. I. : "Algorithm animator : a tool for programming learning", ACM SIGCSE Bulletin, vol. 19, n° 1 (février 1987), pp. 308 - 314.

[GKS 83]
GETZT S. L., KALLIGIANIS G, SCHACH S. R. : "A Very High-level Interactive Graphical Trace for the Pascal Heap", IEEE Transaction on Software Engineering , vol. SE-9, n° 2 (mars 83), pp. 179 - 185.

[Gir 86]
GIROD X. : "Etude d'un environnement de travaux pratiques en enseignement de la programmation", DEA INPG, 1986.

[GIT 84]
GLINERT E. P. et TANIMOTO S. L. : "PICT, An Interactive Graphical Programming Environment", IEEE Computer, vol. 17, n° 11 (novembre 1984), pp. 7 - 25.

[GNV 88]
GANSNER E. R., NORTH S.C. et VO K. P. : "DAG : A program that Draws Directed Graph", Software Practice and Experience, vol. 18, n° 11 (novembre 1988), pp. 1047 - 1062.

[Gom 72]
GOMBRICH E. H. : "The Visual Image", Scientific American (septembre 1972), pp. 82 - 94.

[Gra 86]
Anna GRAM alias JC Boussard, J-P Finance, C GResse, P Jaquet, G-R Perrin, A Quéré, P-C Scholl, M Sintzoff, L Trilling, J Voiron : "Raisonner pour programmer", Dunod Informatique, 1986.

[Gué 85]
GUERAUD V. : "Etude des fonctions EAO nécessaires pour un didacticiel de soutien à l'enseignement de a récursivité - Réponses du système Macintosh à des besoins E.A.O.", DEA INPG, 1985.

[Gué 89]
GUERAUD V. : "Un Jeu de Rôles dans le laboratoire Arcade : Une autre façon d'enseigner la programmation", Thèse de Doctorat de l'Université Joseph Fourier de Grenoble, février 1989.

[Hab 70]
HABER R. N. : "How We Remember What We See", Scientific American (septembre 1970), pp. 104 - 112.

[HaN 86]
HABERMANN A. N. et NOTKIN D. : "Gandalf : Software Development Environment", IEEE Transactions on Software Engineering, vol. SE-12, n° 12, (décembre 1986), pp. 1117 - 1127.

[Hau 88]
HAUGLUSTAINE-CHARLIER : "Images pour apprendre à programmer", Actes du Colloque francophone sur la didactique de l'informatique, Paris, 1,2,3 septembre 1988, pp.205 - 223.

[Hop 74]
HOPGOOD F. R. A. : "Computer Animation Used as a Tool in Teaching Computer Science", IFIP 1974, pp. 889 - 892.

[HoS 78]
HOROWITZ E. et SAHNI S. : "Fundamentals of Computers Algorithms", Pitman, 1978.

[HoS 84]
HOROWITZ E. et SAHNI S. : "Fundamentals of Data Structures in Pascal", Pitman, 1984.

[IEEE 87]
Special report : Visualisation in Scientific Computing, a synopsis, IEEE Computer Graphics and Applications, vol. 7, n° 7 (juillet 1987), pp. 61 - 70.

[JFC 88]
Le Journal de la Formation Continue et de l'EAO, janvier 1988.

[JFC 89]
Le Journal de la Formation Continue et de l'EAO, janvier 1989.

[Kem 86]
KEMPTON W. : "A System to Make Visible the Structure and Execution of Student Program", ACM SICSE Bulletin, vol. 18, n° 1 (février 1986), pp. 313 -317.

[KDP 83]
KOLTUN P., DEIMEL L. E. et PERRY J. : "Progress Report on the Study of Program Reading", ACM SIGCSE Bulletin, vol. 15, n° 1 (février 1983), pp. 168 - 176.

[KnP 77]
KNUTH D. E. et PRATT V. R. : "Fast Pattern Matching in String", SIAM J. Computer, vol. 6, n° 2 (juin 1977), pp. 323-349.

[Knu 69]
KNUTH D. E. : "Fundamental Algorithms", vol. 1: "The Art of Computer programming", 1969.

[Knu 73]
KNUTH D. E. : "Fundamental Algorithms", vol. 3: "Sorting and Searching", 1973.

[Lar 86]
LARSON J. A. : "Special feature : A Visual Approach to Browsing in a Database Environment", IEEE Computer, vol. 19, n° 6 (juin 1986), pp. 62 - 71.

[Lau 76]
LAUGIER C. : "Illustration dynamique de programmes a l'aide d'une console de visualisation", Congrès AFCET, "Panorama de la nouveauté Informatique en France", Tome II, Paris/GIF - Sur Yvette, Ecole Supérieure d'électricité (3,4,5 novembre 1976), pp. 703 - 711.

[LeM 85]
LEWIS L. et McGLINN R. J. : "IPEX1, A Library of Dynamic Pascal Programming Examples", ACM SIGCSE Bulletin, vol. 17, n° 1 (février 1985), pp. 72 - 77.

[Lie 85]
LIEM I. : "Etude des fonctions EAO necessaires pour un didacticiel de soutien à l'enseignement des structures de données : cas de la structure séquence", DEA INPG, 1985.

[LiM 87]
LISS I. B. et Mc.MILLAN T.C. : "Fractals with Turtle Graphics : A CS2 Programming Exercise for Introducing Recursion", ACM SIGCSE Bulletin, vol. 19, n° 1 (février 1987), pp. 141-147.

[LiM 88]
LISS I. B. et McMILLAN T.C. : "An AMAZING Exercise in Recursion for CS1 and CS2", ACM SIGCSE Bulletin, vol. 20, n° 1 (février 1988), pp. 270-274.

[Lod 83]
LODDING K. N. : "Iconic Interfacing", IEEE Computer Graphics and Application, vol. 3, n° 2 (mars/avril 1983), pp. 11 - 20.

[LoD 85]
LONDON R. L. et DUISBERG R. A. : "Animating Programs Using SmallTalk", IEEE Computer, vol. 18, n° 8 (août 85), pp. 61 - 71.

[LPS 83]
LUCAS M., PEYRIN J-P., SCHOLL P-C. : "Algorithmiques et Représentation des Données", tome 1 : "Files, automates d'états finis", Masson, 1983.

[Luc 83]
LUCCI A. : "Le système Voyelle : manuel d'utilisation, Rapport interne INP Grenoble, septembre 1983.

[Luc 89]
LUCCI A. : "Production de logiciels pour l'enseignement : une expérience de prototypage d'un système construit sur un environnement Prolog", Thèse de 3ème cycle en informatique, INPG, février 1989.

[LuS 75]
LUCAS M. et SCHOLL P-C. : "Proposition pour une initiation à l'algorithmique", polycopiés, USTMG et INPG, 1975.

[MaE 87]
MAXIM B. R. et ELENBOGEN B. S. : "Teaching Programming Algorithms Aided by Computer Graphics", ACM SIGCSE, vol. 19, n° 1 (février 1987), pp. 297 - 301.

[Mal 81]
MALONE T.W. : "What Makes Computer Games Fun ?", Byte, vol. 6, n° 12 (décembre 1981), pp. 258 - 277.

[Man 84]
MANDELBROT B. : "Objets fractals", Flammarion, 2° édition, 1984.

[MBW 89]
McGLINN R. J., BRITT M., WOOLARD L. : "APEX1, A Library of Dynamic Programming Examples", ACM SIGCSE Bulletin, vol. 21, n° 1 (février 1989), pp. 98 - 102.

[MeB 80]
MEYER B. et BAUDOIN C. : "Méthodes de programmation", Eyrolles, 2° éd., 1980.

[MoH 85]
MORICONI M. et HARE D. F. : "Visualizing Program Designs through Pegasys", IEEE Computer, vol. 18, n° 8 (août 85), pp. 72 - 85.

[Moh 88]
MOHER T. G. : "PROVIDE: A Process Visualisation and Debugging Environment", IEEE Transaction on Software Engineering, vol. 14, n° 6 (juin 1988), pp. 849 - 857.

[MTT 83]

MINCY J. W., THARP A. L. et TAI K. : "Visualizing Algorithms and Processes with The Aid of a Computer", ACM SICSE Bulletin, vol. 15, n° 1 (février 1983), pp. 106 - 111.

[Mye 83]

MYERS B. A. : "INCENSE : A System for Displaying Data Structures", Computer Graphics, vol. 17, n° 3 (juillet 1983), pp. 115 - 125.

[Mye 86]

MYERS B. A. : "Visual Programming, Programming by Example and Program Visualisation : A Taxonomy", CHI '86 proceedings (avril 1986), pp. 59 - 66.

[Mye 88]

MYERS B. A. : "The State of the Art in Visual Programming and Program Visualisation", Rapport de recherche, Computer Science Department, Carnegie Melon University, Pittsburg, PA, 15213-3890, février 1988.

[NaS 73]

NASSI I et Shneiderman B : "Flowchart techniques for structured programming", ACM SIGPLAN Notices, vol. 8 , n° 8 (août 1973), pp. 12 - 26.

[NeS 86]

NEWMAN W. S. et SPROULL R. F. : "Principles of Interactive Computer Graphics", Mac Graw Hill International Student Edition, 12th printing, 1986.

[Owe 86]

OWEN G. S. : "Teaching of Tree Data Structures Using Microcomputer Graphics", ACM SICSE Bulletin, vol. 18, n° 1 (février 1986), pp. 67 - 72.

[Pai 88]
PAIR C. : "Je ne sais toujours pas enseigner la programmation" (actes - version distribuée aux participants) ou "L'apprentissage de la programmation" (actes - version définitive), Colloque sur la Didactique de l'Informatique, Paris (1 - 3 septembre 1988).

[Pey 88a]
PEYRIN J-P. : "Programmation, réalités et promesses", Premier Colloque sur l'informatique, Bamako, Mali (février 1988).

[Pey 88b]
PEYRIN J-P. : "Algorithmique, expression fonctionnelle du traitement des séquences, actions récursives", UJF CIAP, Option Informatique - Stage Prélenfrey, mars 1988.

[Pir 86]
PIROLLI P. : "A Cognitive Model and Computer Tutor for Programming Recursion", Human Computer Interaction, vol. 2 (1986), pp. 319 - 355.

[PlN 81]
PLATTNER B. et NIEVERGELD J. : "Special feature : Monitoring Program Execution: A Survey", IEEE Computer vol. 14, n° 11 (novembre 1981), pp. 76 - 93.

[PLM 87]
de la PASSARDIERE B., LE CALVEZ F. et MADAULE F. : "Apports de l'EAO dans l'enseignement de l'informatique", actes du Congrès francophone sur l'enseignement assisté par ordinateur", Cap d'Agde (23-24-25 mars 1987), pp. 235 - 245.

[PMS 88]
PAIR C, MOHR R, SCHOTT R : "Construire les algorithmes : les améliorer, les connaître, les évaluer", Dunod, 1988.

[PoN 83]
PONG M. C. et NG N. : "PIGS : A System for Programming with Interactive Graphical Support", ACM Software Practice and Reference, vol. 13, n° 9 (septembre 1983), pp. 847-855.

[PrS 88]
PRUSINKIEWICZ P et SANDNESS G : "Fractals : Koch curves as Attractors and Repellers", IEEE Computer Graphics and Applications, vol. 8, n° 6 (novembre 1988), pp. 26 - 40.

[Rae 85]
RAEDER G. : "A Survey of Current Graphical Programming Technique", IEEE Computer, vol. 18, n° 8 (août 85), pp. 11 - 25.

[Ram 85]
RAMBALLY G. K. : "Real-time Graphical Representations of Linked Data Structures", ACM SICSE Bulletin, vol. 17, n° 1 (février 1985), pp. 41 - 48.

[Rei 84]
REISS S. P. : "Graphical Program Development with PECAN Program Development System", ACM SIGSOFT/SIGPLAN Notes, SIGSOFT vol.9, n° 3, SIGPLAN vol. 19, n° 5 (mai 1984), pp. 30 - 41.

[ReP 86]
REISS S. P. et PATO J. N. : "Displaying Program and Data Structure", Rapport de recherche, Department of Computer Science, Brown University, Providence RI 02912, avril 1986.

[RGR 85]
RUBIN R. V., GOLIN E. J., REISS S. P. : "THINKPAD : A Graphical System for Programming by Demonstration", IEEE Software, vol. 2, n° 2 (mars 1985), pp. 73 - 78.

[RiW 88]
RICH C. et WATERS R. C. : " Programmer's Apprentice Project, A Research Overview", IEEE Computer, vol. 21, n° 11 (novembre 1988), pp. 11 - 25.

[Rob 85]
ROBERT P. : "Le Petit Robert 1", Dictionnaire alphabétique et analogique de la langue française, rédaction dirigée par A. REY et J. REY-DEBOVE, 1985.

[Sav 87]
SAVOY J. : "Le livre électronique EBOOK 3", Actes du Congrès francophone sur l'enseignement assisté par ordinateur, Cap d'Agde (23, 24, 25 mars 1987), pp. 279 - 292.

[Sca 87]
SCANLAN D. : "Data-Structures Students May Prefer to Learn Algorithms Using Graphical Methods", ACM SIGCSE Bulletin, vol. 19, n° 1 (février 1987), pp. 302 -307.

[Sca 88]
SCANLAN D. : "Should Short, Relatively Complex Algorithms Be Taught Using Both Graphical and Verbal Method?", ACM SIGCSE Bulletin, vol. 20, n° 1 (février 1988), pp. 185-188.

[Sch 79]
SCHOLL P-C. : "Vers une programmation systématique : étude de quelques méthodes, techniques et outils", Thèse de Docteur d'Etat es-Sciences Mathématiques de l'USTMG - INPG, juin 1979.

[Sch 84]
SCHOLL P-C. : "Algorithmiques et représentation des données", tome 3 : "Récursivité et arbre", Masson, 1984.

[Sch 85]
SCHOLL P-C. : "Projet DIDALP, didacticiel en algorithmique et programmation", LGI-IMAG, Groupe Méthodologie de la programmation, mars 1985.

[ScP 88]
SCHOLL P-C. et PEYRIN J-P. : "Schéma algorithmique fondamentaux", Masson, 1987.

[Sed 83]
SEDGEWICK R. : "Algorithms", Addison Wesley, 1983.

[Sed 84]
SEDGEWICK R. : "Algorithms", Addison Wesley, 1984 (2° éd).

[Sed 86]
SEDGEWICK R. : "Algorithms", Addison Wesley, 1986.

[Sel 88]
SELF J. : "Artificial Intelligence and Human Learning", Intelligent computer-aided instruction, Chapman and Hall, 1988.

[ShJ 84]
SHARP D., JOHNSTON M.D. : "ChipWits, Robots You Teach to Think for Themselves", Brainpower Inc., 1984.

[Shn 83]
SHNEIDERMAN B. : "Direct Manipulation : A Step Beyond Programming Languages", IEEE Computer, vol. 16, n° 8 (août 83), pp. 57 - 69.

[SIG 87]
Special Issue on Visualisation in Scientific Computing, a publication of ACM SIGGRAPH, vol. 21, n° 6 (novembre 1987).

[Tei 84]
TEITELMAN W. : "A Tour through CEDAR ", Proceedings of 7-th International Conference on Software Engineering, mars 26 - 29, 1984, Florida, pp. 181 - 195.

[TeR 81]
TEITELBAUM T. et REPS T. : "The Cornell Program Synthesizer : A Syntax directed Programming Environment", Communication of the ACM, vol. 24, n° 9 (septembre 1981), pp. 563 - 573.

[Tes 81]
TESLER L. : "The Smalltalk Environment", Byte (août 81), pp. 90 - 147.

[Tha 84]
THALMANN D. : "An Interactive Data visualisation System", Software Practice and Experience, vol. 14, n° 3 (mars 1984), pp. 277 - 290.

[Thi 86]
THINK Technologies : "Lightspeed Pascal for the Macintosh", User's Guide and Reference manual, version 1, First edition, Think Technologies, Inc. 1986.

[Van 84]
van DAM A. : "The Electronic Classroom : Workstations for Teaching", Int. J. Man-Machine Studies, vol. 21, n° 4 (octobre 1984), pp. 353 - 363.

[Vau 85]
VAUDENE D. : "Système d'exploitation des ordinateurs", Vidéogramme, ADI, 1985.

[Wes 86]
WEST J. : "Debugging Macintosh Applications", Byte (décembre 1986), pp. 127 - 139.

[WeS 79]
WETHERELL C. et SHANNON A. : "Tidy Drawings of Tree", IEEE Transaction on Software Engineering , vol. SE-5, n° 1 (septembre 1979), pp. 514 -520.

[Wie 88]
WIEDENBECK S. : "Learning Recursion As a Concept and As a Programming Technique", ACM SIGCSE Bulletin, vol. 20, n° 1 (février 1988), pp. 275 - 266.

[Wir 73]
WIRTH N. : "Systematic Programming : An Introduction", Prentice Hall, 1973.

[Wir 76]
WIRTH N. : "Algorithms + Data Structures = Programs", Prentice Hall, 1976.

[Wir 85]
WIRTH N. : "From Programming Language Design To Computer Construction", Turing Award Lecture, Communication of the ACM, vol. 28, n° 2 (février 1985), pp. 160 - 164.

[Wir 86]
WIRTH N. : "Algorithms and Data Structures", Prentice Hall, 1986.

[Yao 85]
YAO N. F. : "The Computer-aided Programming System - A Friendly Programming Environment", IEEE MICRO, vol. 5, n° 2 (avril 1985), pp. 9 - 19.

[YMV 85]
YANKELOVICH N., MEYROWITZ N., van DAM A. : "Reading and Writing the Electronic Book", IEEE Computer, vol. 18, n° 10 (octobre 1985), pp. 15 - 30.

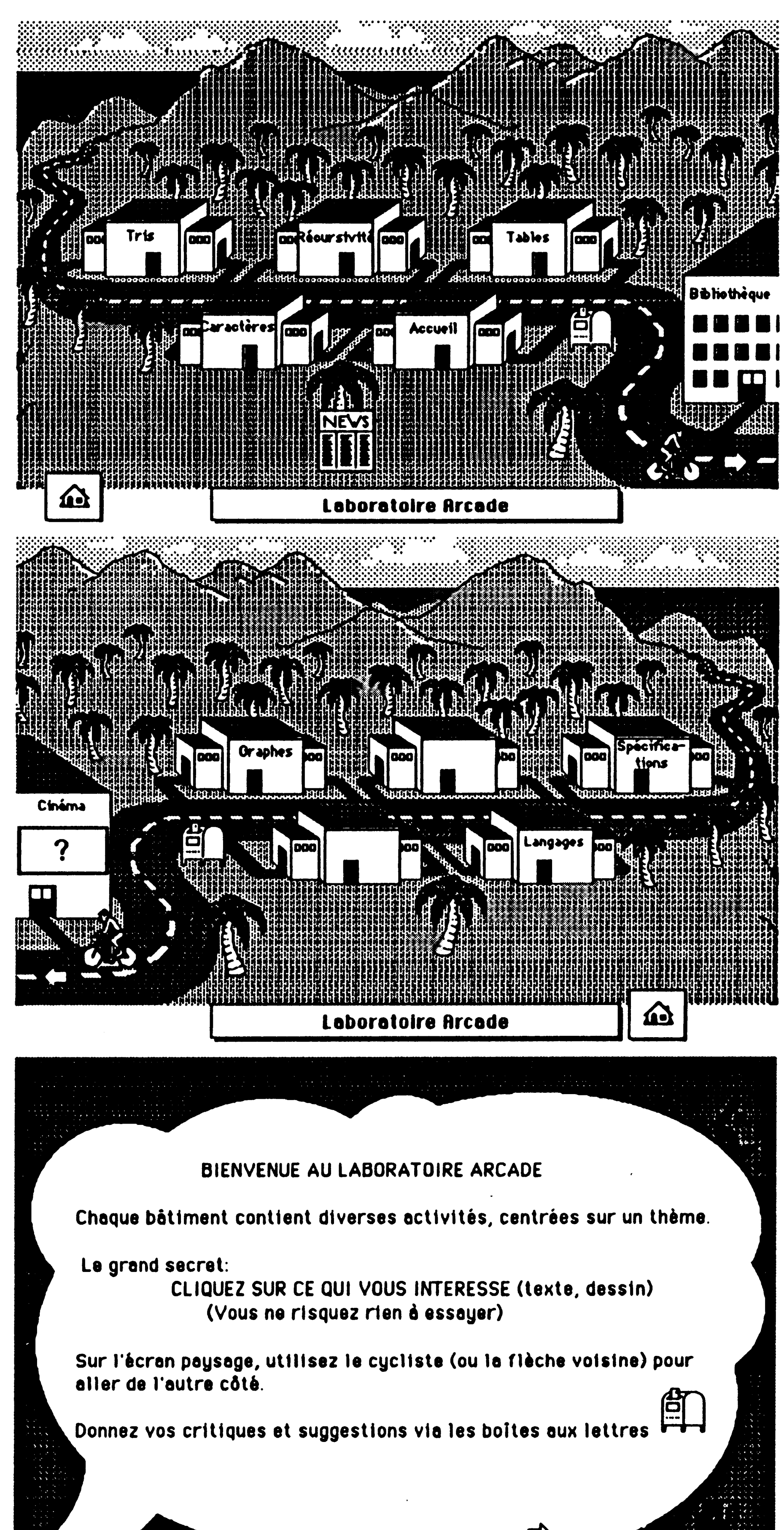
[You 75]
YOURDON E. : "Techniques of program structure and design", Prentice Hall, 1975.

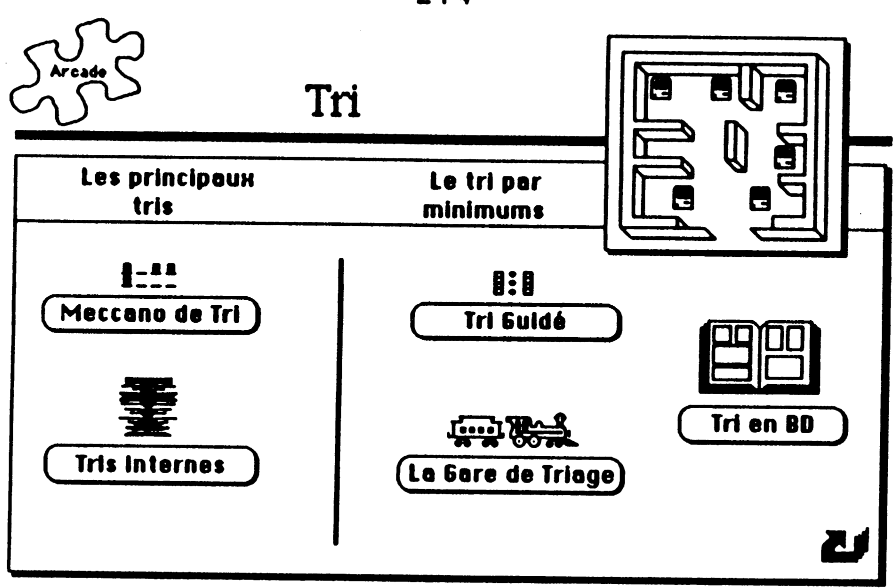
[Zam 84]
ZAMBRANO J. : "E.A.O. et enseignement de la programmation : une maquette de didacticiel", Thèse de 3ème cycle en informatique, INPG, octobre 1984.

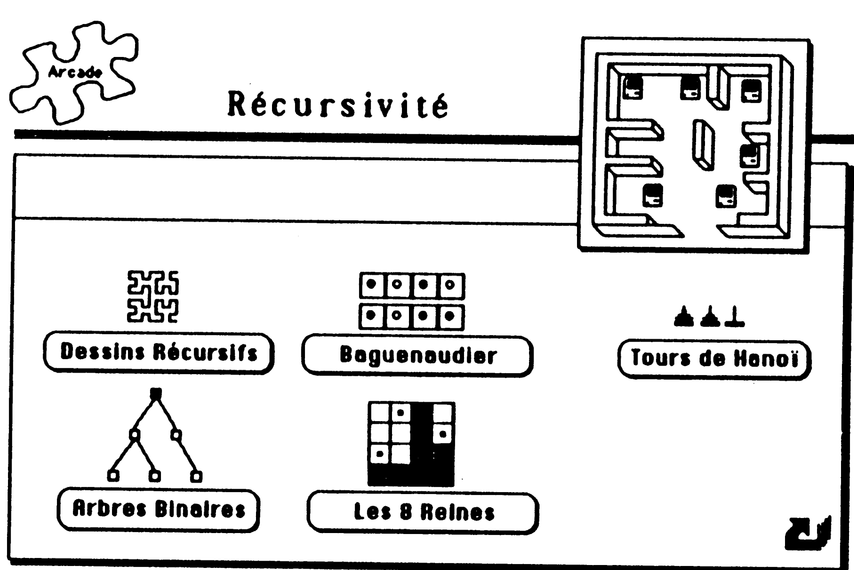
[ZPS 88]
ZIMMERMAN M., PERRENOUD F., SCHIPER A. : "Understanding Concurrent Programming Through Program Animation", ACM SICSE Bulletin, vol. 20, n° 1 (février 1988), pp. 27 - 31.

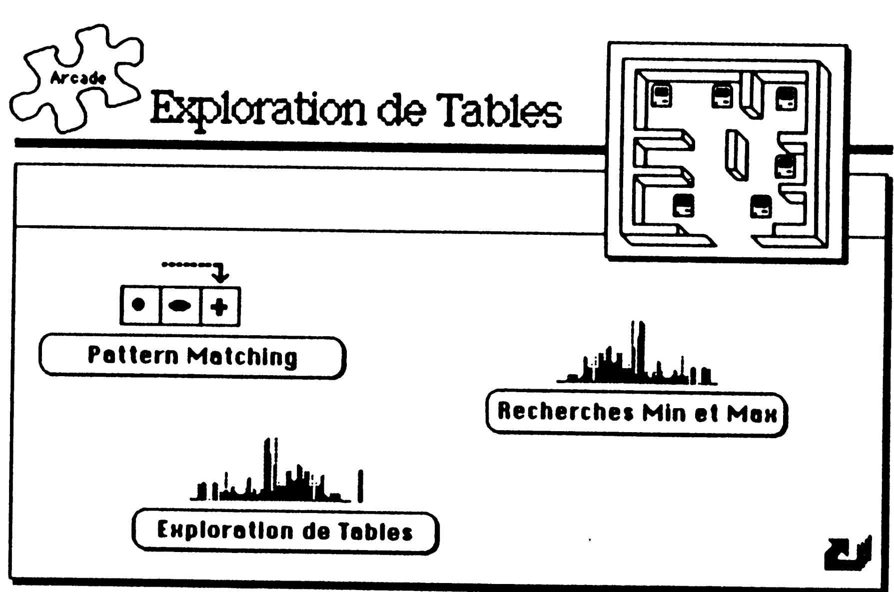
ANNEXE A

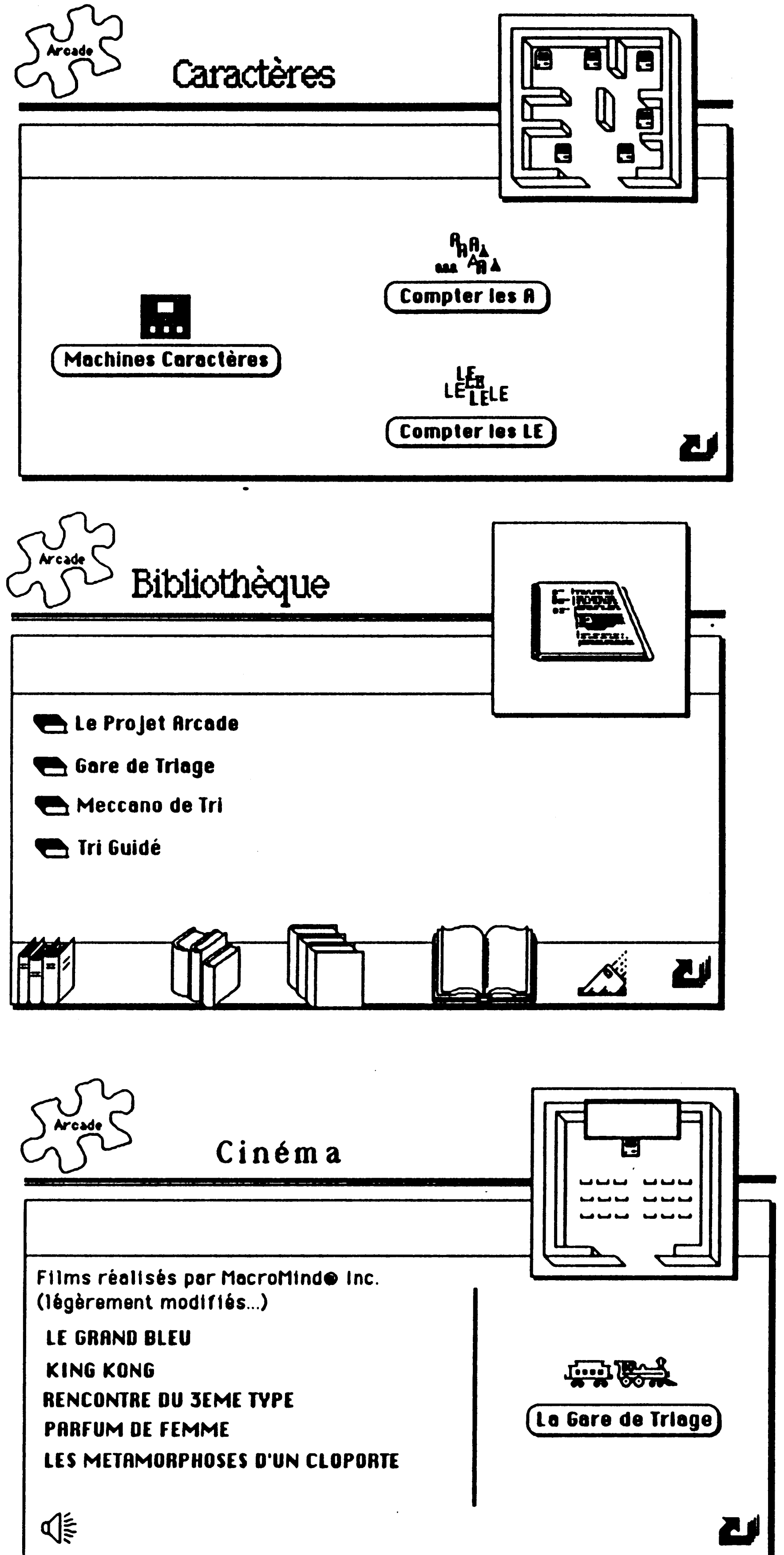
LE LABORATOIRE ARCADE

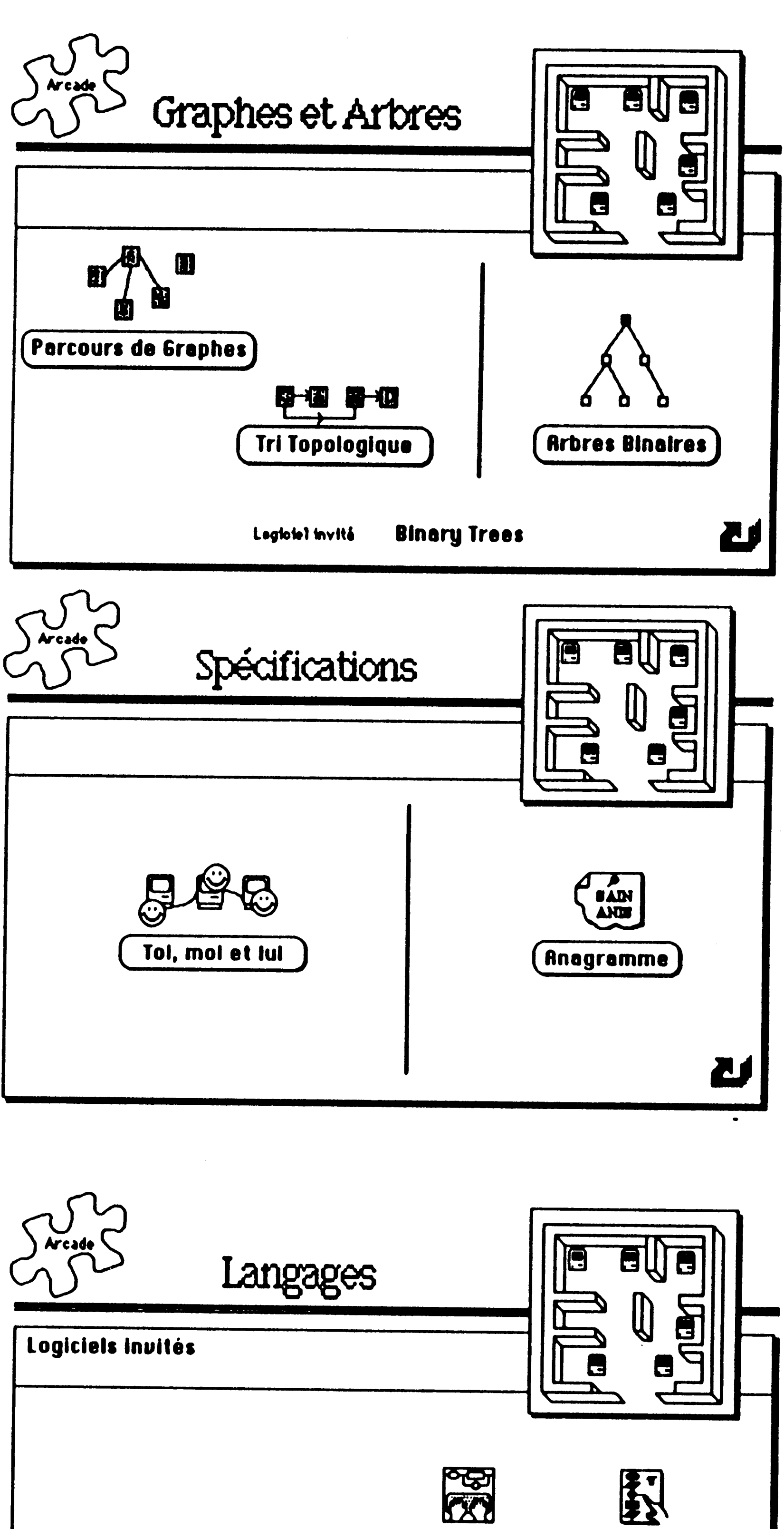

Tris
Récursivité
Tables
Bibliothèque
Caractères
Accueil
NEWS
Laboratoire Arcade
Graphes
Spécifications
Cinéma
?
Langages
Laboratoire Arcade
BIENVENUE AU LABORATOIRE ARCADE
Chaque bâtiment contient diverses activités, centrées sur un thème.
Le grand secret:
CLIQUEZ SUR CE QUI VOUS INTERESSE (texte, dessin)
(Vous ne risquez rien à essayer)
Sur l'écran paysage, utilisez le cycliste (ou la flèche voisine) pour aller de l'autre côté.
Donnez vos critiques et suggestions via les boîtes aux lettres


Arcade
Tri
Les principaux tris
Le tri par minimums
Meccano de Tri
Tris Internes
Tri Guidé
Tri en BD
La Gare de Triage


Arcade
Récursivité
Dessins Récursifs
Baguenaudier
Tours de Hanoï
Arbres Binaires
Les 8 Reines


Arcade
Exploration de Tables
Pattern Matching
Recherches Min et Max
Exploration de Tables


Arcade
Caractères
Compter les A
Machines Caractères
Compter les LE
Arcade
Bibliothèque
Le Projet Arcade
Gare de Triage
Meccano de Tri
Tri Guidé
Arcade
Cinéma
Films réalisés par MacroMind® Inc.
(légèrement modifiés...)
LE GRAND BLEU
KING KONG
RENCONTRE DU 3EME TYPE
PARFUM DE FEMME
LES METAMORPHOSES D'UN CLOPORTE
La Gare de Triage

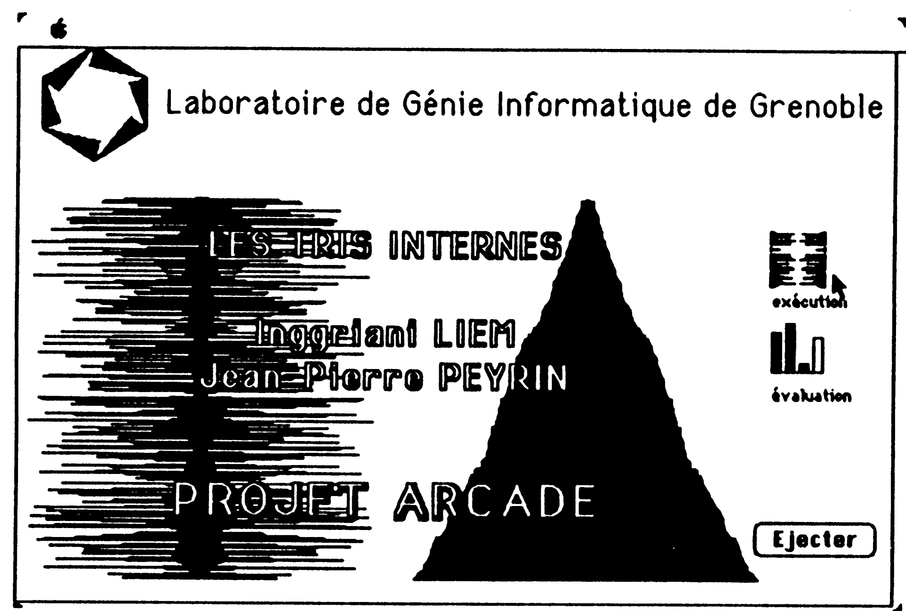

Arcade
Graphes et Arbres
Parcours de Graphes
Tri Topologique
Arbres Binaires
Logiciel invité
Binary Trees
Arcade
Spécifications
Toi, moi et lui
Anagramme
Arcade
Langages
Logiciels invités
Pascal
Basic

ANNEXE B

SYNOPSIS DES LOGICIES REALISES

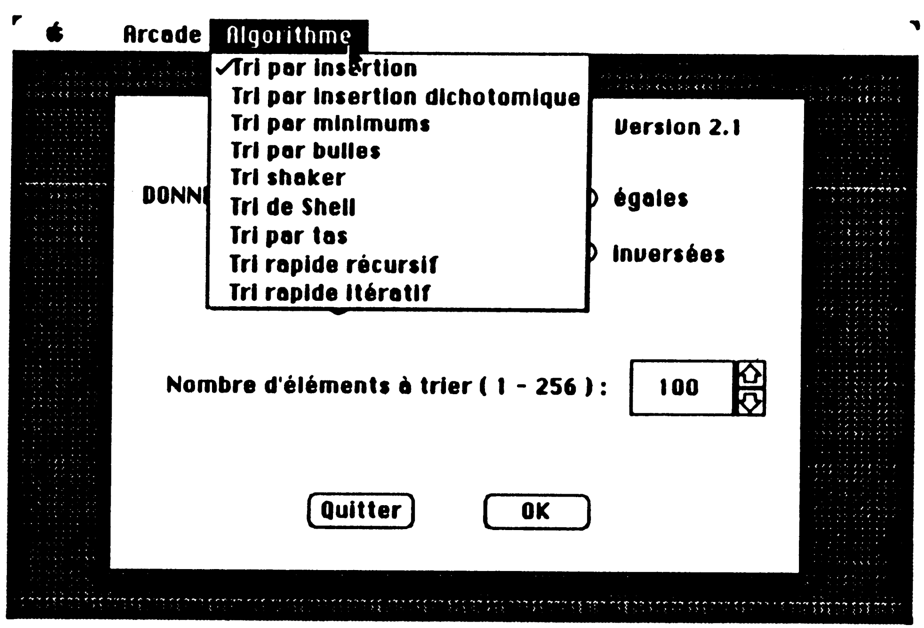
Cet annexe montre les fonctionnalités de chaque logiciel sous forme d'images d'écrans. Malheureusement, cette présentation papier donne une succession statique, incapable de montrer le dynamisme de l'exécution, ni toutes possibilités de cheminement. Nous invitons le lecteur à essayer les différents logiciels sur machine. Les manuels utilisateur se trouvent dans [Arc 88a], [Arc 88b], [Arc 89a], [Arc 89b], [Arc 89c], [Arc 89d].

Choix d'une application :



Exécution ou évaluation.

Exécution :



Choix d'algorithme.

Exécution (suite) :

Arcade Algorithme

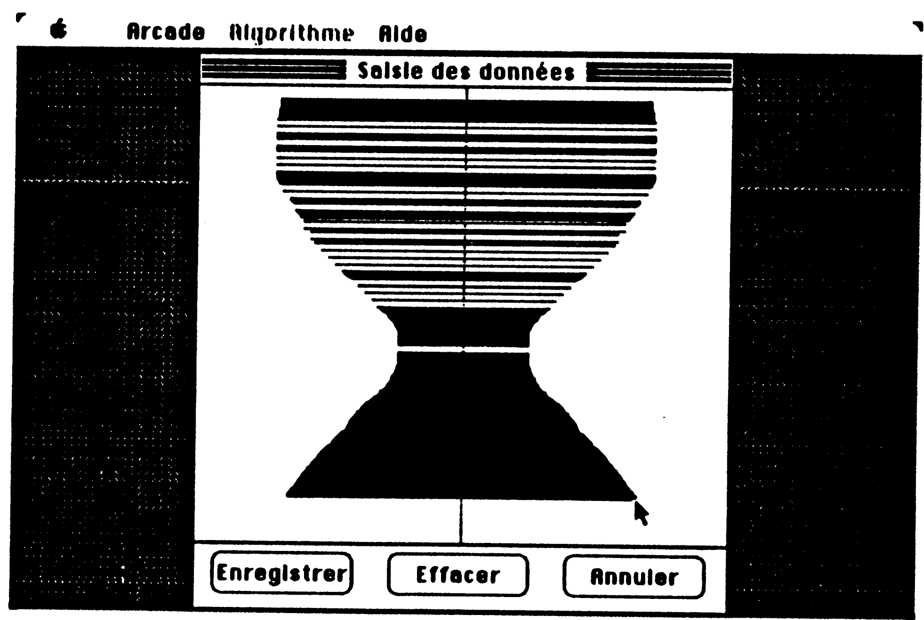
EXECUTION DE TRI Version 2.1

DONNEES : ◉ aléatoires ○ égales
○ ordonnées ○ inversées
○ à saisir

Nombre d'éléments à trier (1 - 256) : 100

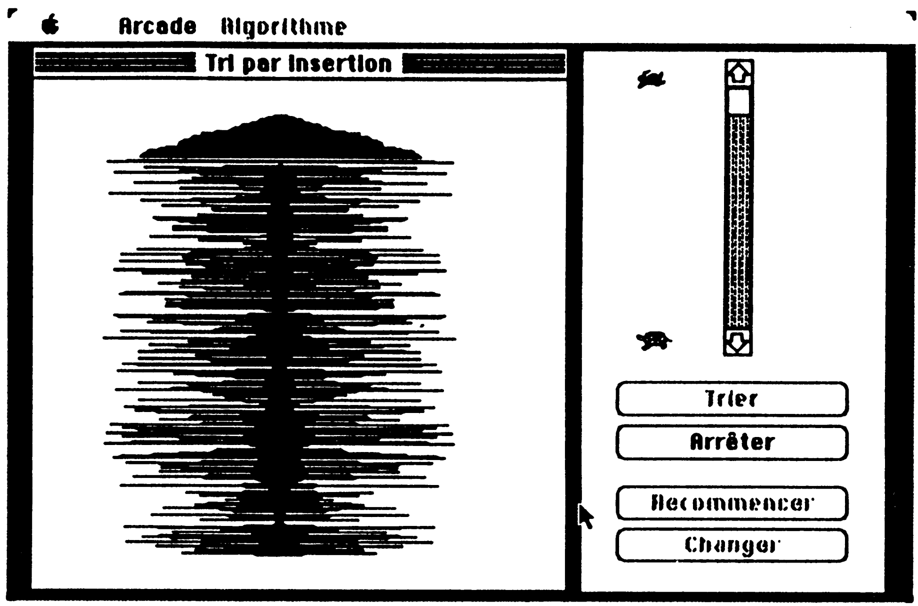
Quitter OK

Choix des données :
type des données et du nombre d'éléments,ou saisie des données.

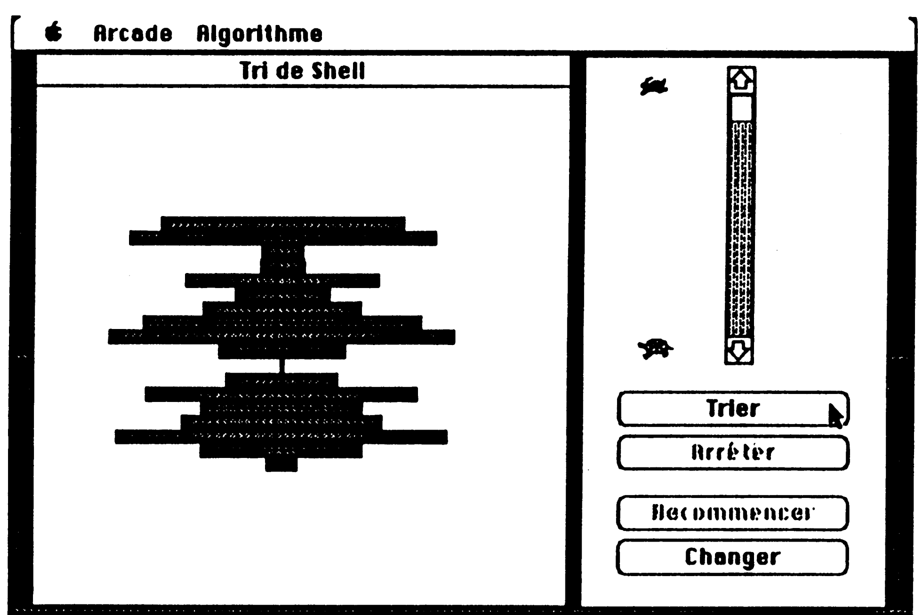


Exemple de saisie graphique.

Exécution (suite) :



Exemple d'exécution sur l'un des algorithmes de tri,
avec un grand nombre de données.



Exemple d'exécution sur l'un des algorithmes de tri,
avec un petit nombre de données.

Evaluation :

Arcade

COMPARAISON DES ALGORITHMES DE TRI

ALGORITHMES :

- ☒ Tri par insertion
- ☒ Tri par insertion dichotomique
- ☒ Tri par minimums
- ☒ Tri par bulles
- ☒ Tri shaker
- ☒ Tri de Shell
- ☒ Tri par tas
- ☒ Tri rapide récursif
- ☒ Tri rapide itératif

DONNEES :

- ☒ aléatoires
- ☒ inversées
- ☒ ordonnées
- ☒ égales
- ☐ à saisir

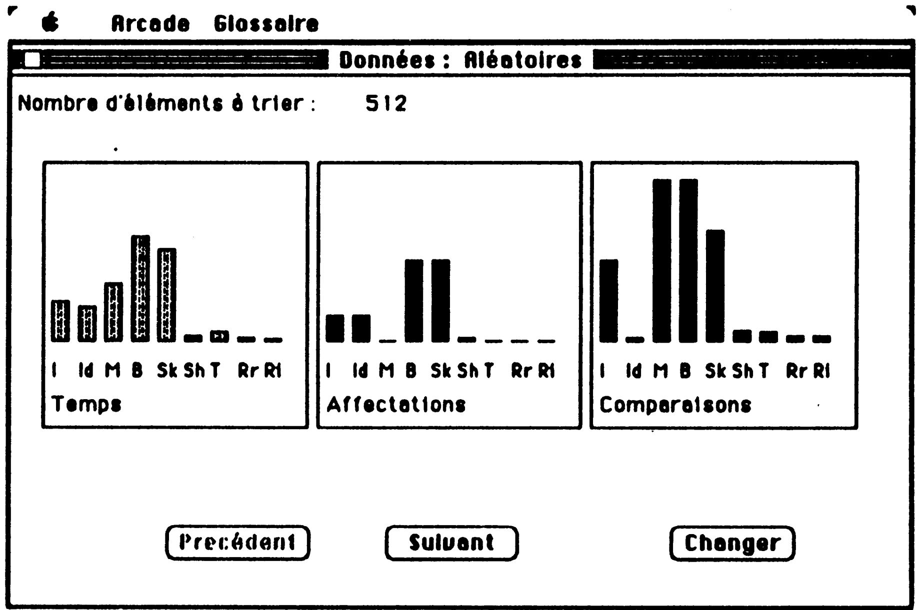
Evaluation pré-enregistrée

Taille :

○ 8	○ 64	◉ 512
○ 16	○ 128	○ 1024
○ 32	○ 256	○ 2048

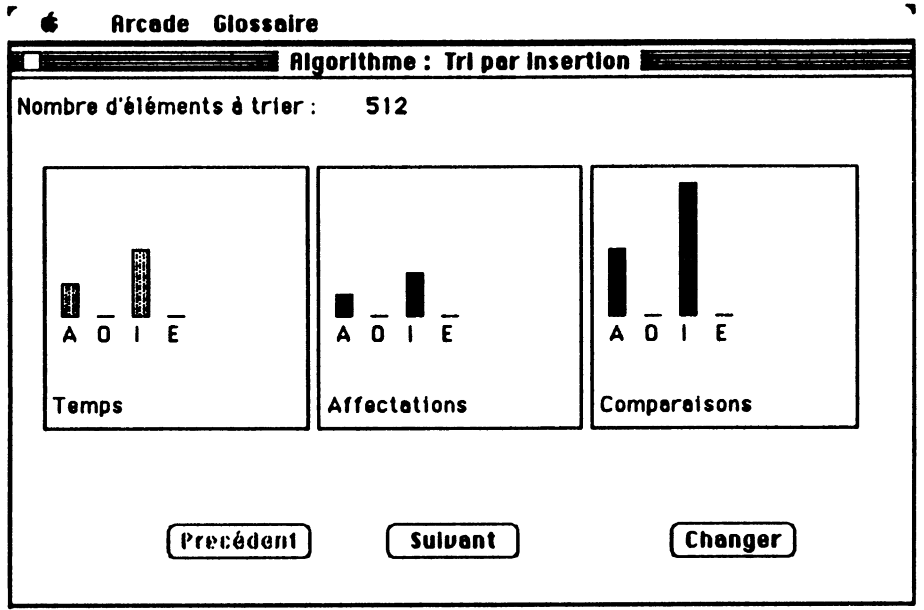
Quitter OK

Choix des algorithmes de tris, des types et de la taille des données.



Comparaison des algorithmes pour un type de données, ici "aléatoires".

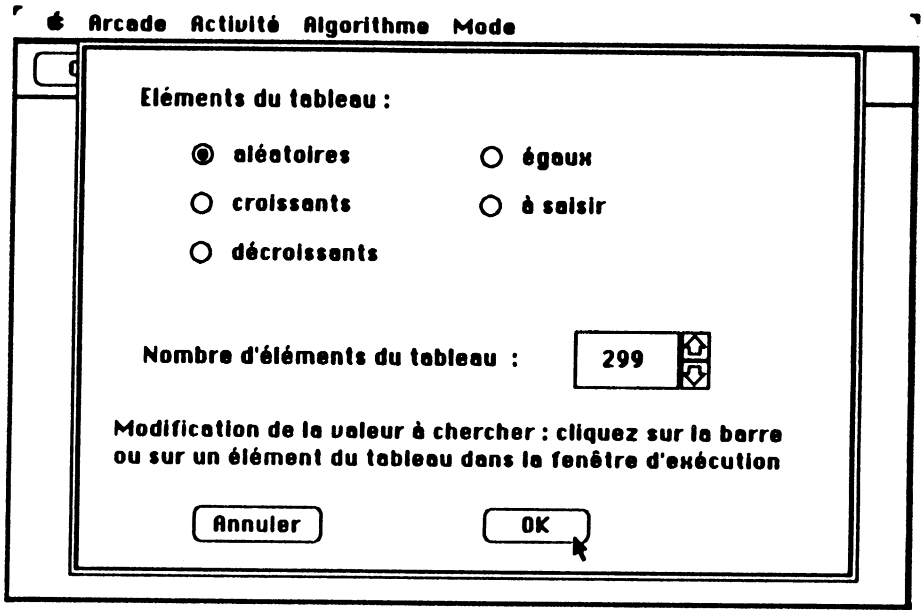
Evaluation (suite) :



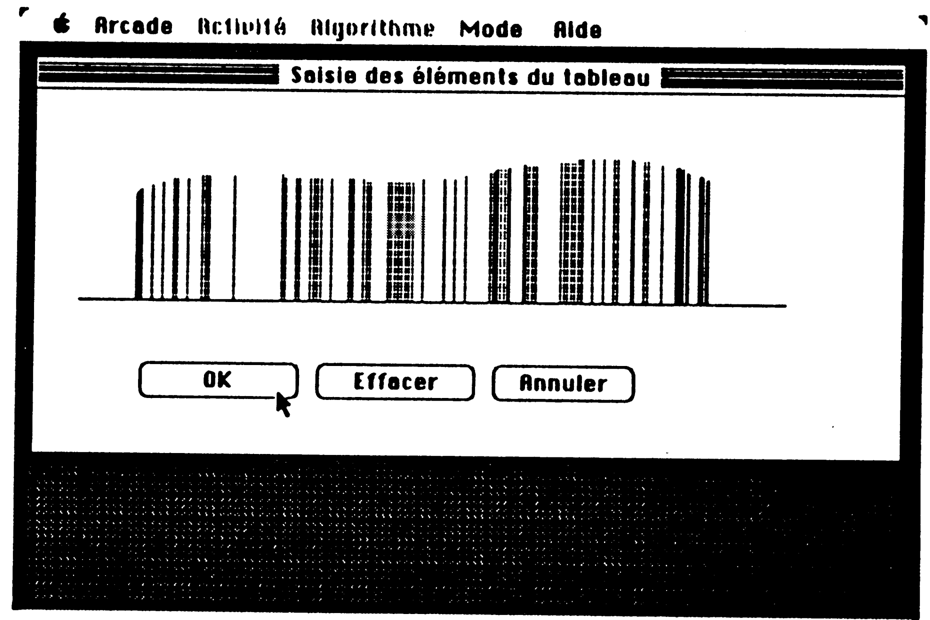
Comparaison des types de données pour un algorithme, ici le "Tri par insertion".

Exploration de tables

Choix des données :

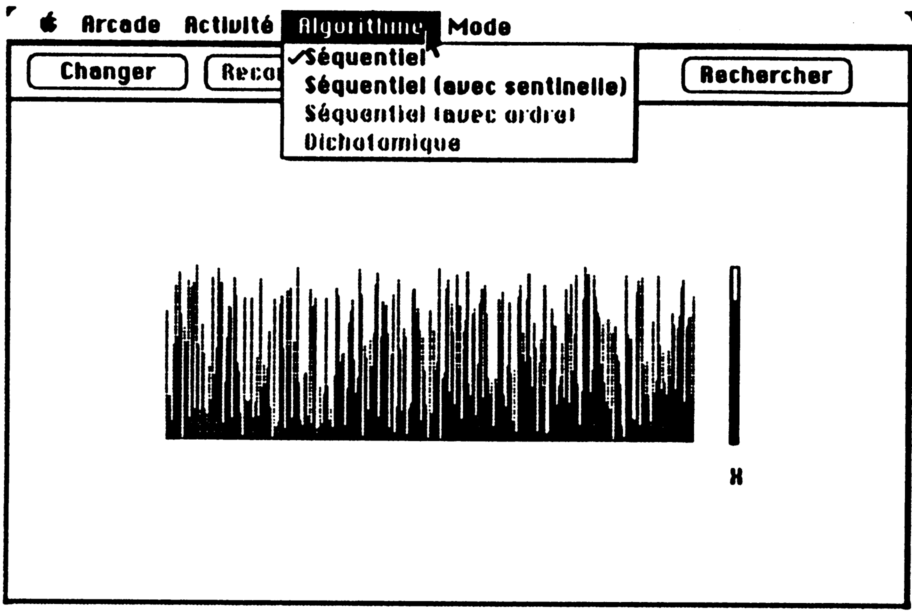


Choix du nombre d'éléments, ou saisie graphique des éléments.

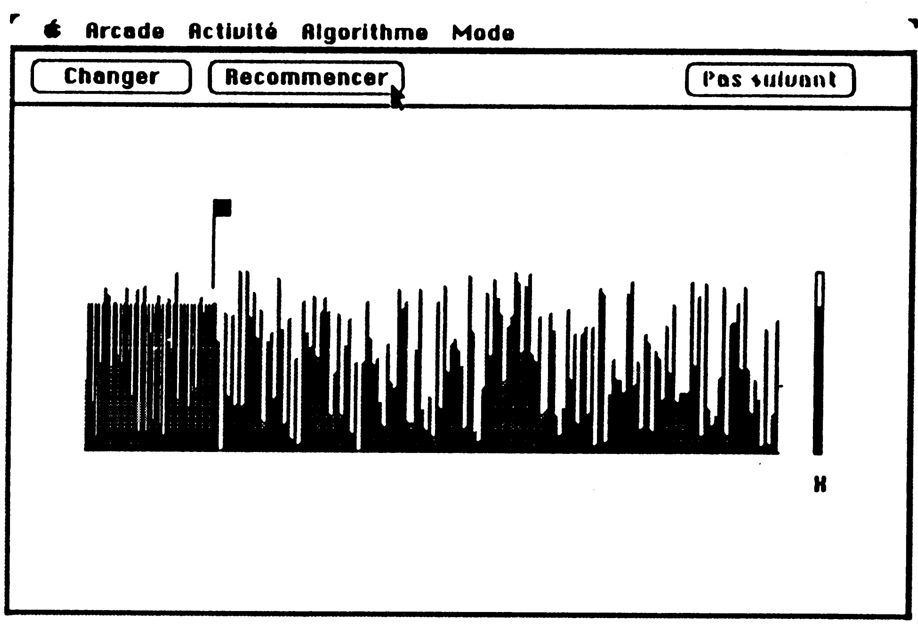


Saisie graphique.

Exécution :



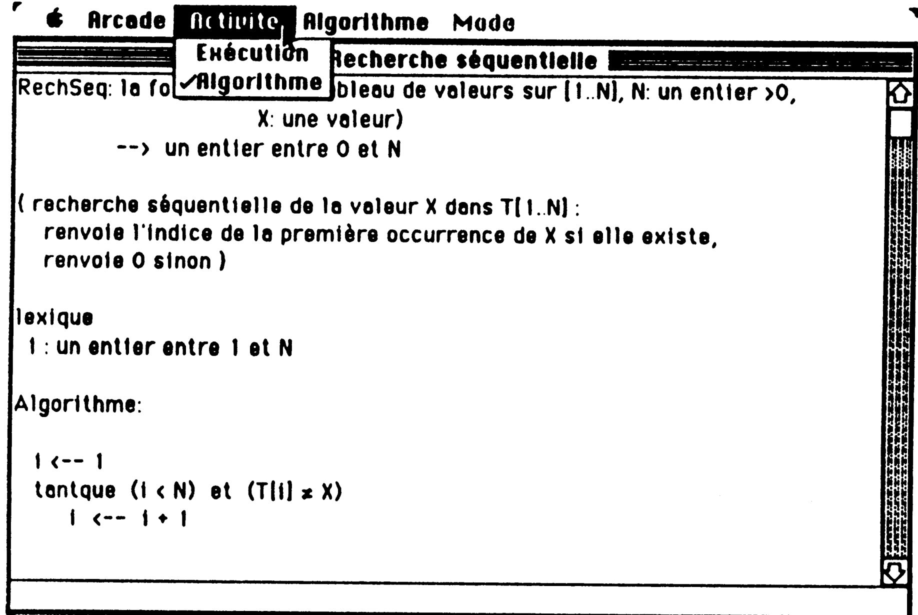
4 algorithmes sont proposés.



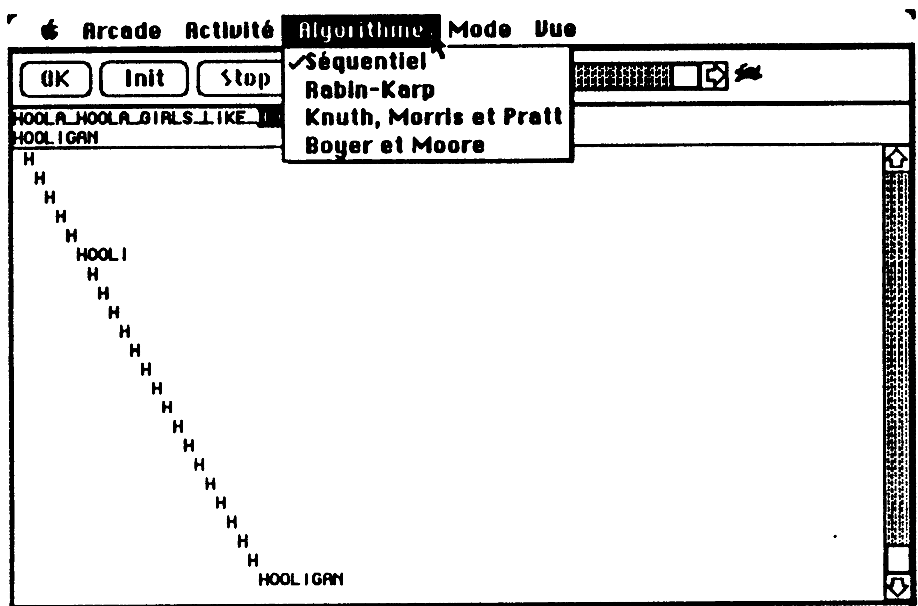
On peut recommencer l'exécution sur les mêmes données ou changer.les données.

Exploration de tables

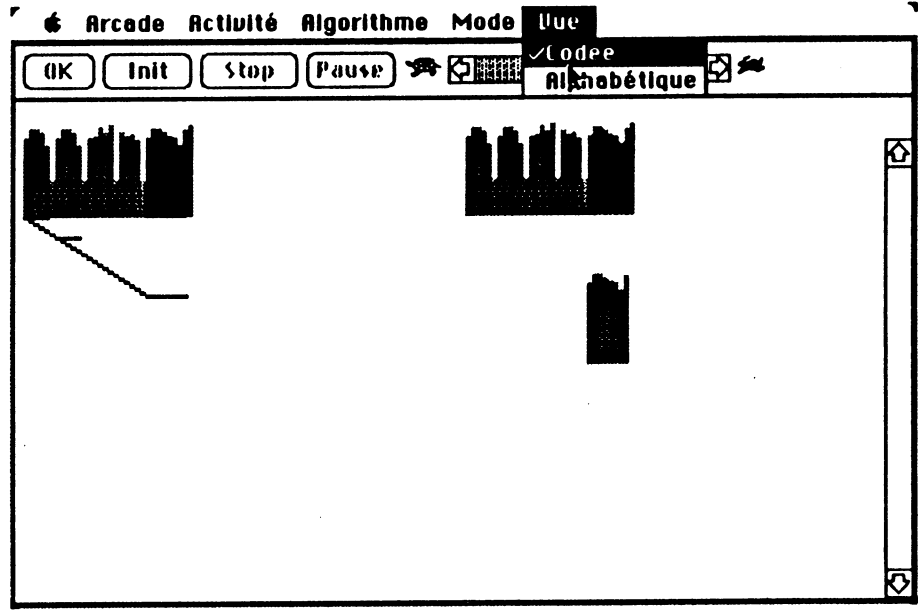
Lecture d'algorithme :



Exécution :



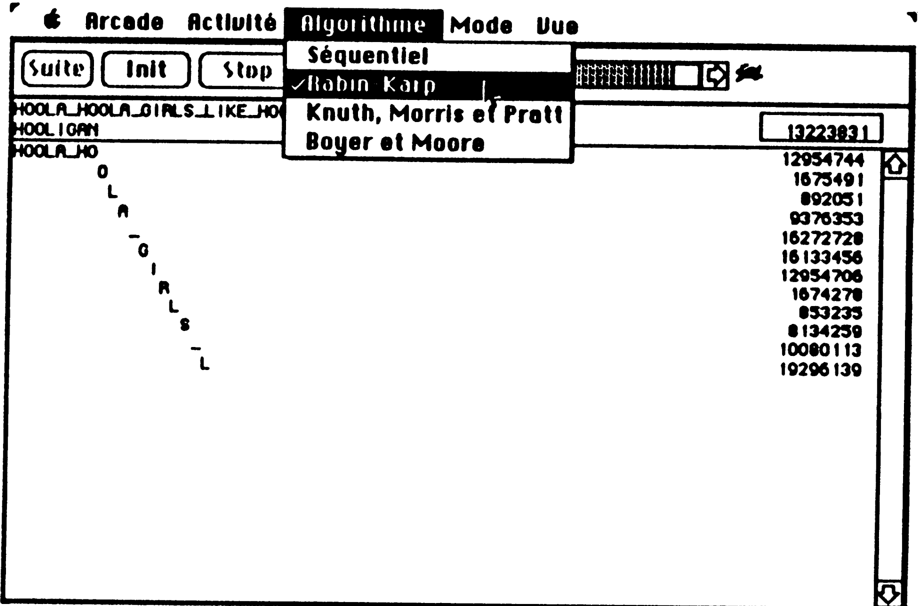
Exécution sur le texte.



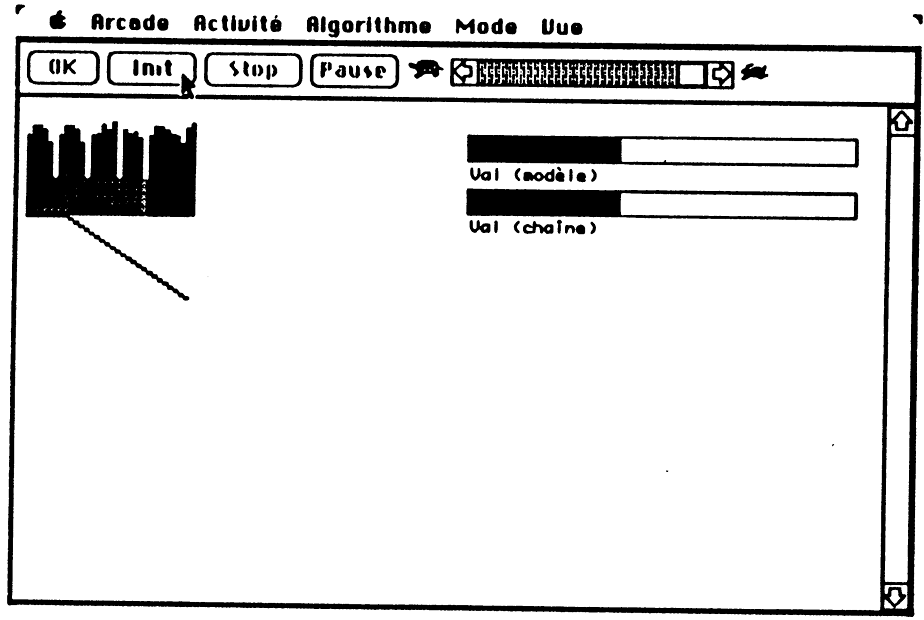
Exécution du texte sur un codage graphique.

Pattern matching

Exécution (suite) :

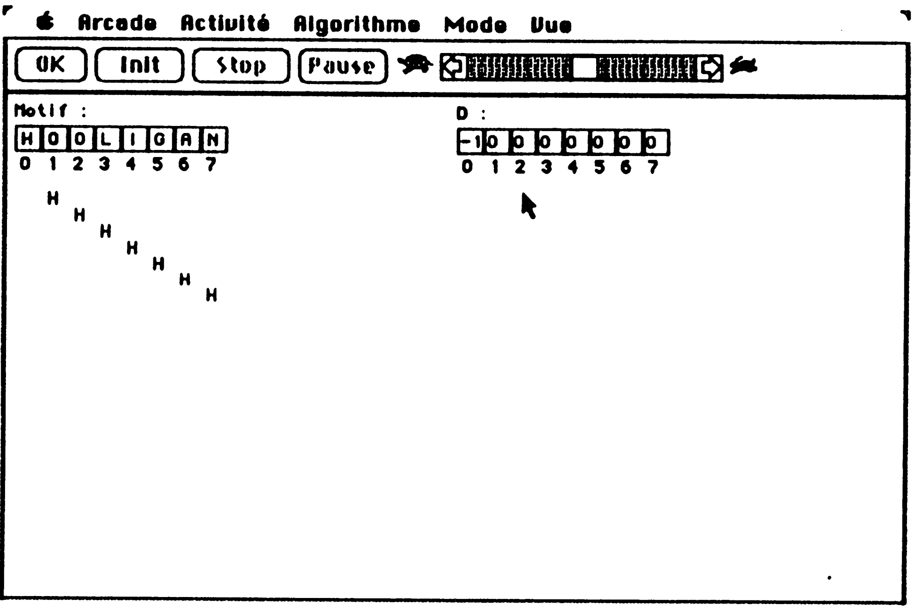


Exécution de l'algorithme Rabin Karp.

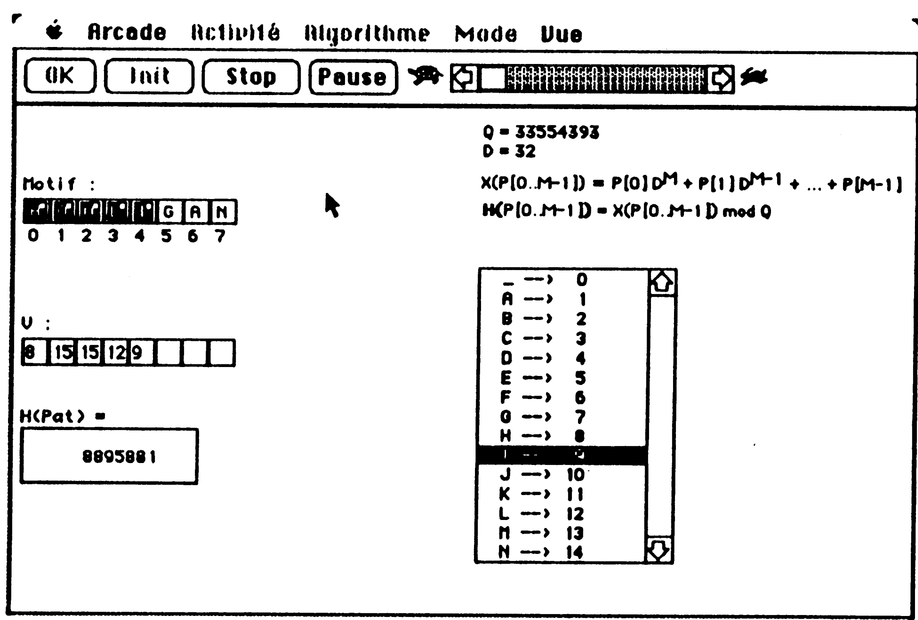


Exécution du texte sur un codage graphique
(algorithme Rabin Karp).

Exécution (suite) :



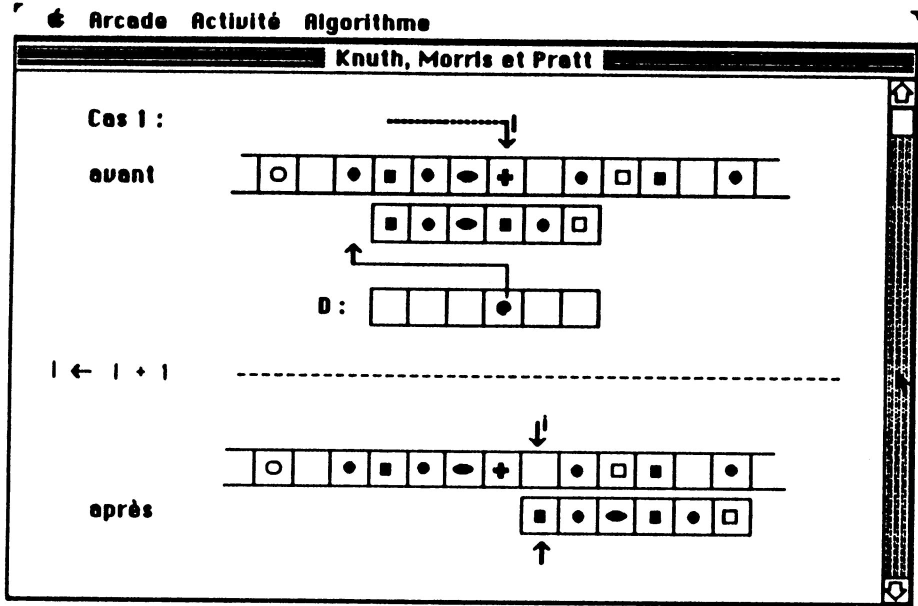
Compilation du motif (algorithme Knuth-Morris-Pratt).



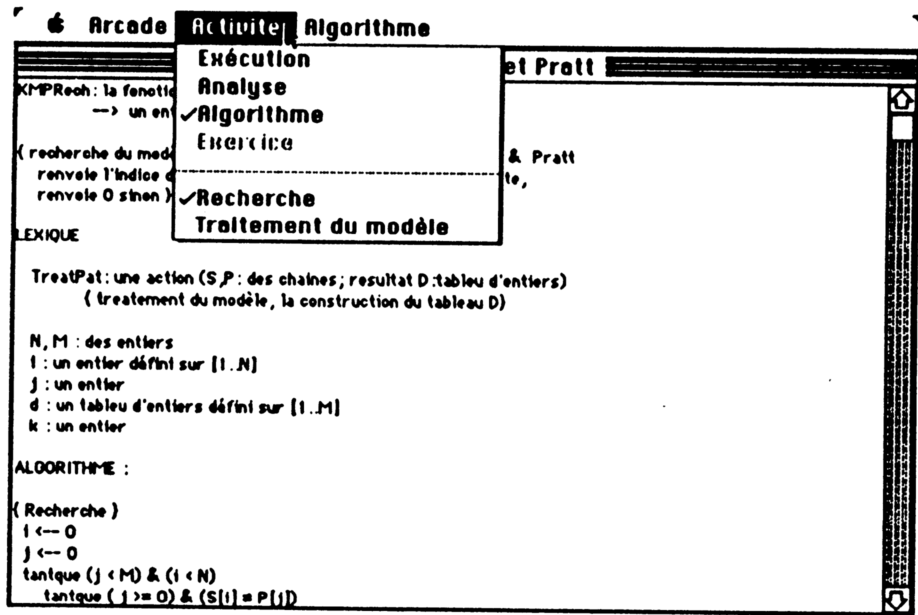
Compilation du motif (algorithme Rabin Karp).

Pattern matching

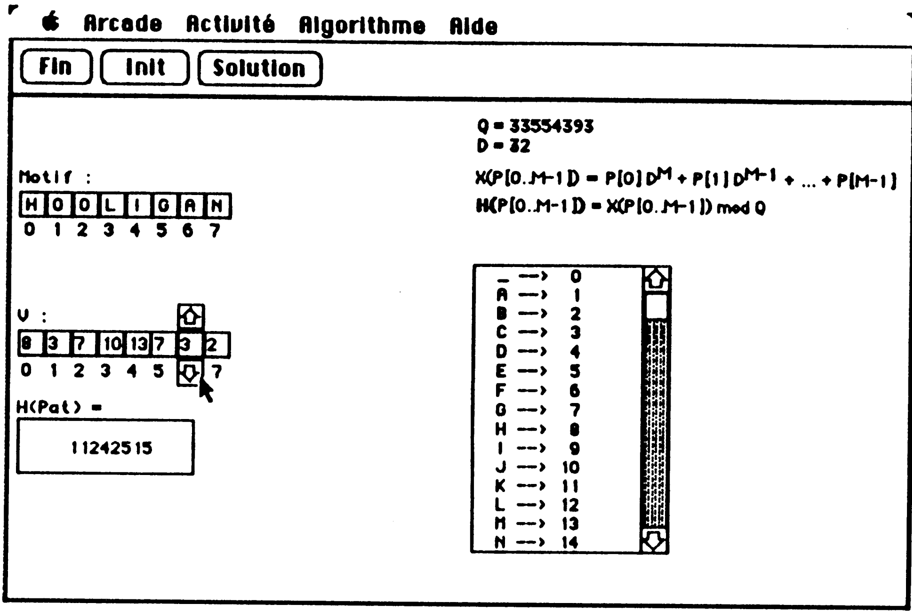
Analyse :



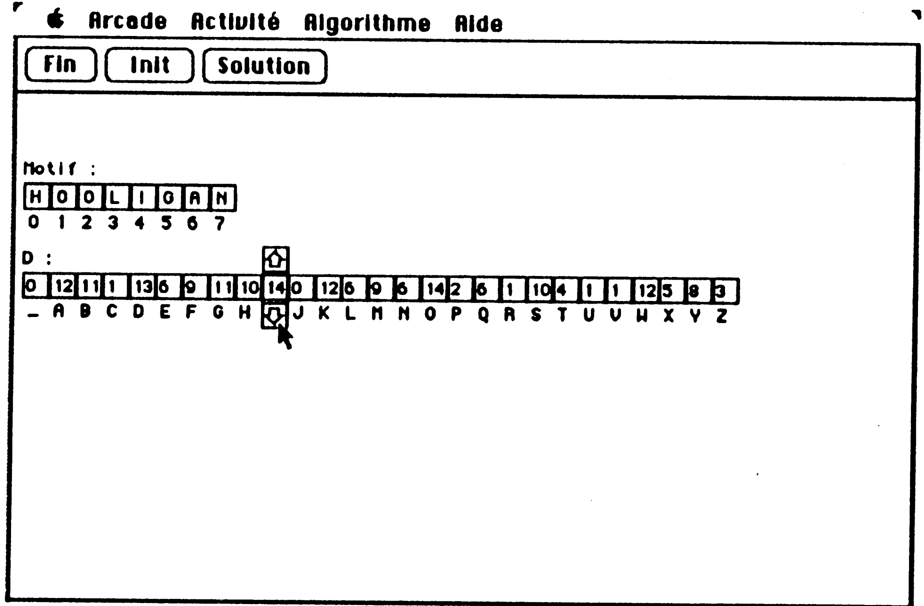
Lecture d'algorithme :



Exercice :



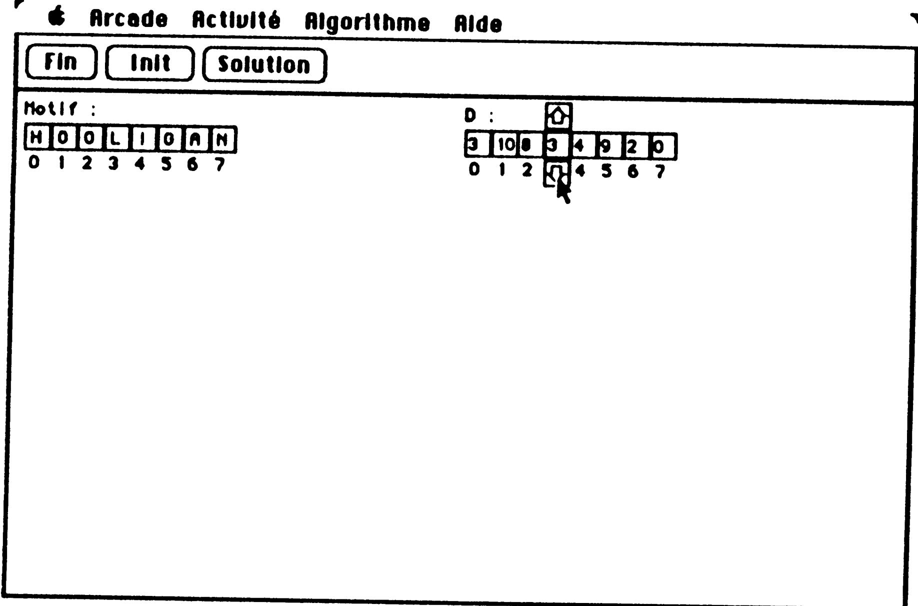
Exercice sur l'algorithme Rabin Karp.



Exercice sur l'algorithme Boyer Moore.

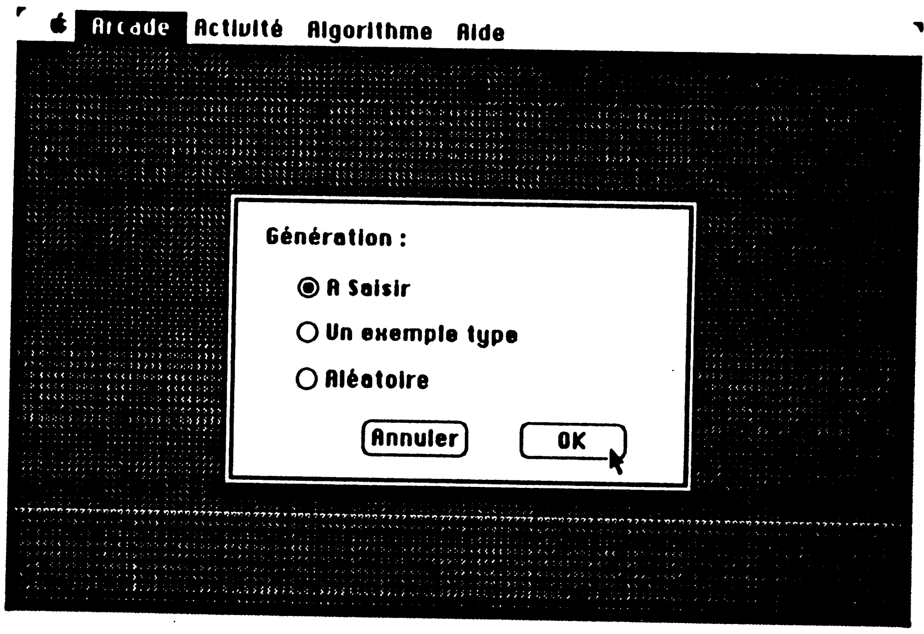
Pattern matching

Exercice (suite) :

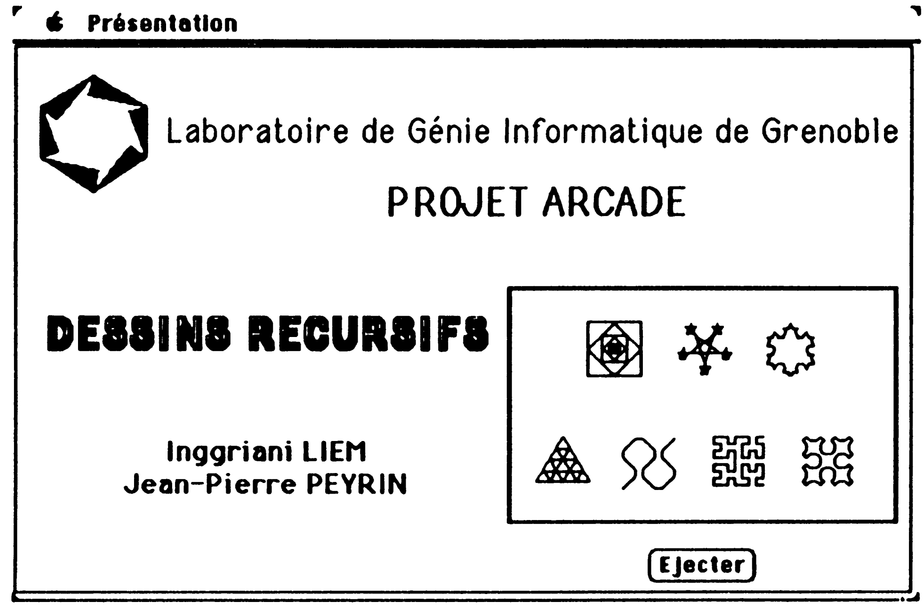


Exercice sur l'algorithme Knuth-Morris-Pratt.

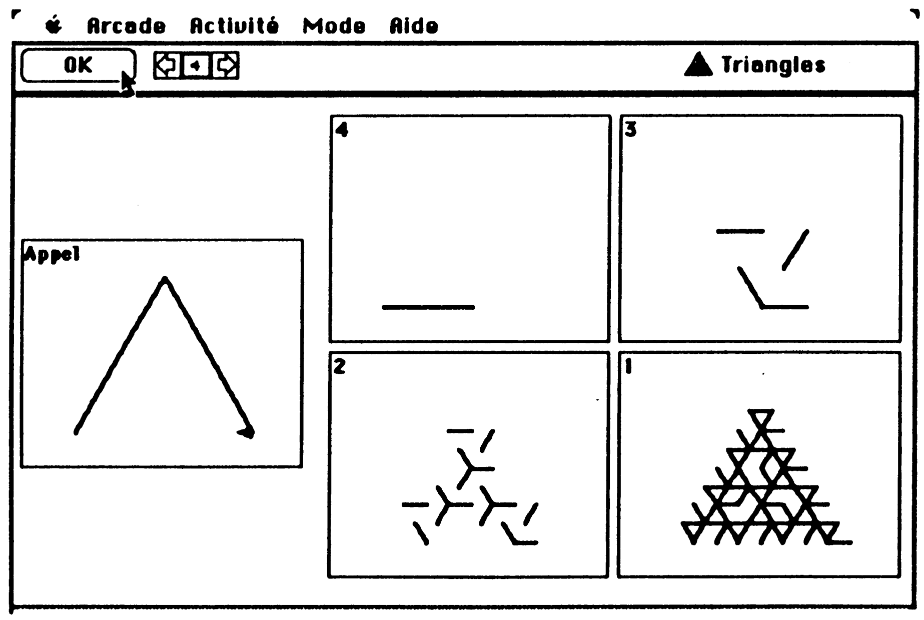
Choix des données :



Choix d'un dessin :



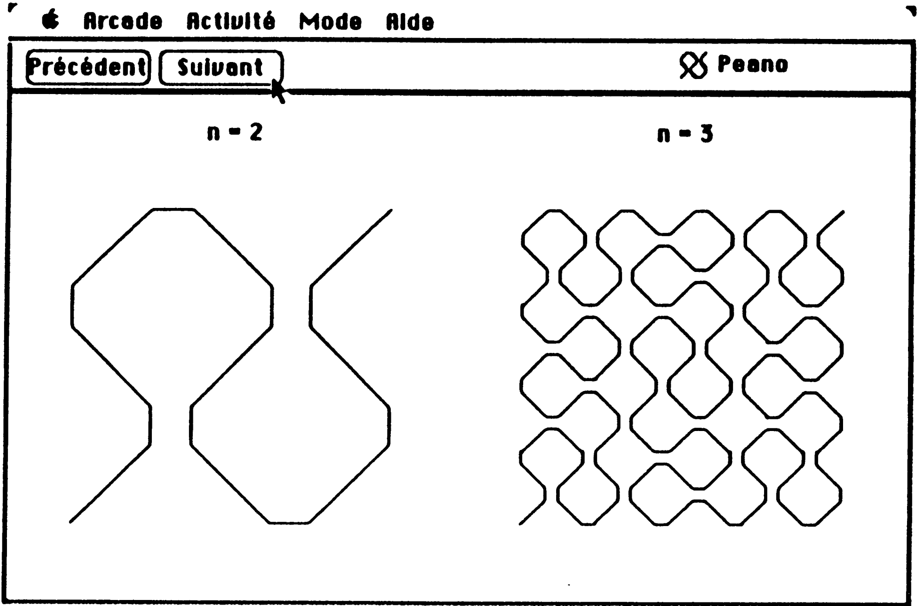
Exécution :



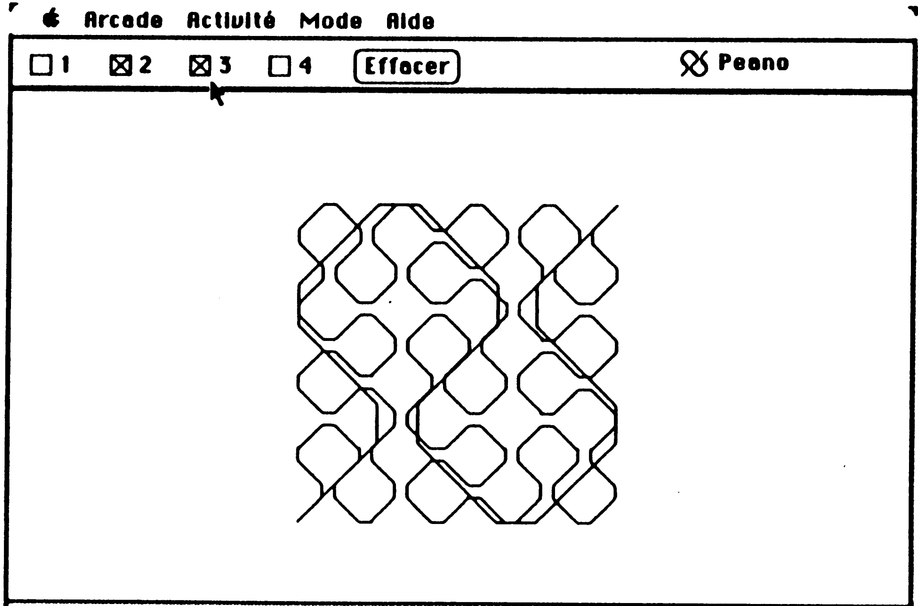
Exécution par niveau, sans trame.

Dessins récursifs

Analyse :

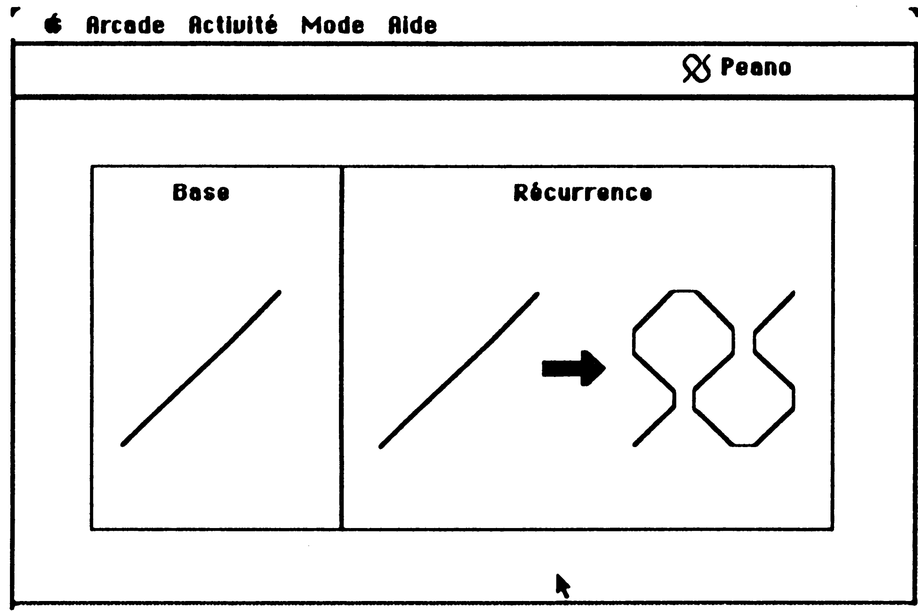


Mode "Succession".

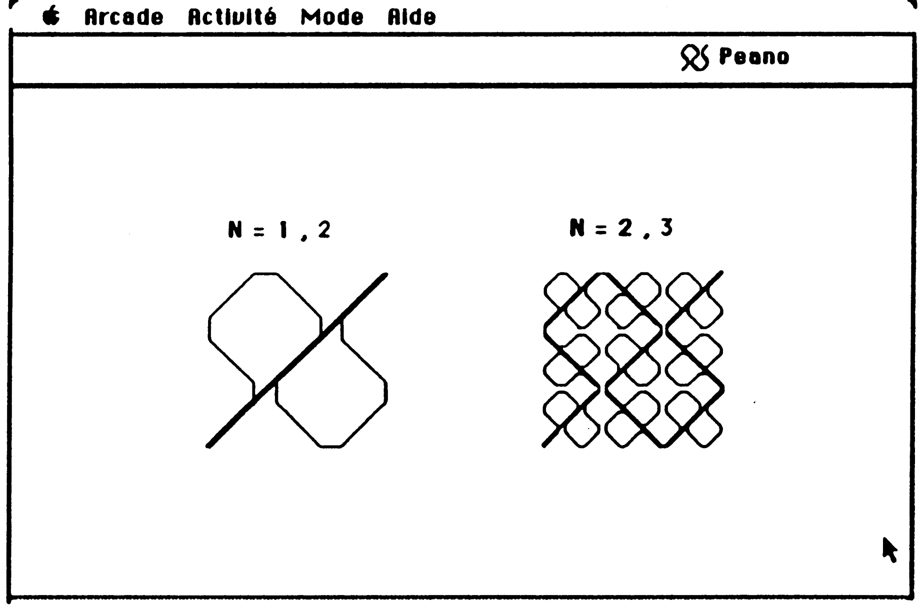


Mode "Superposition".

Analyse (suite) :



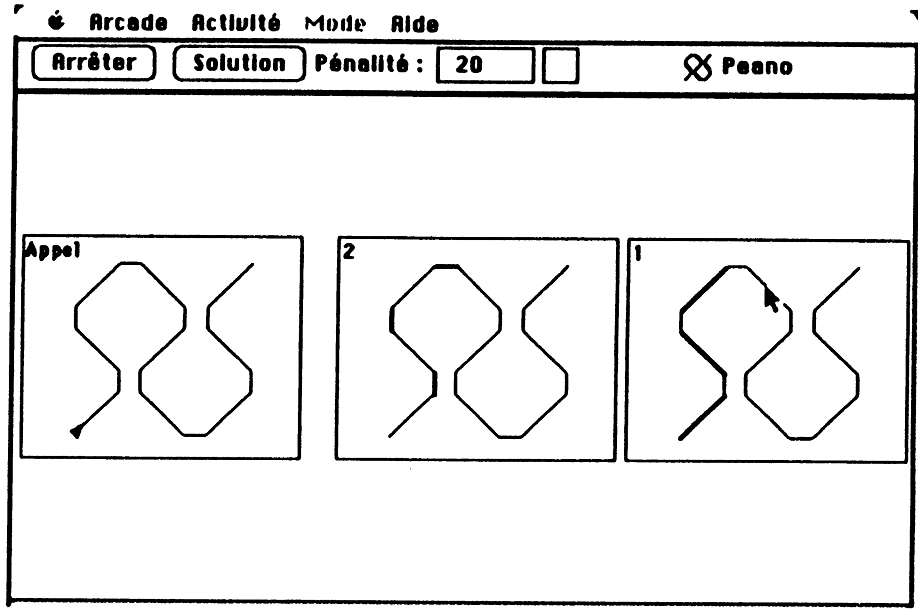
Mode "Construction".



Mode "Structure".

Dessins récursifs

Exercice :



Mode "Par niveau".

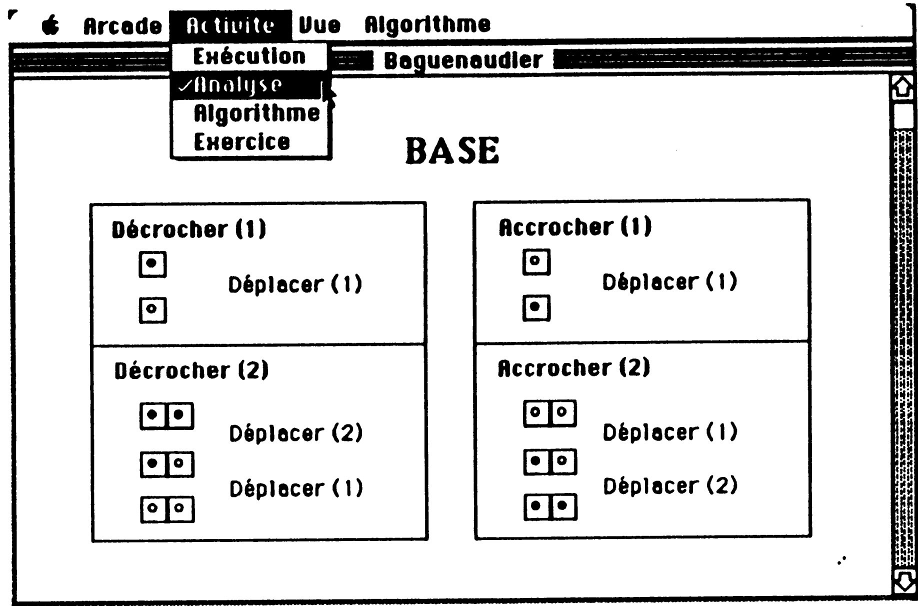
Lecture d'algorithme :

Arcade Activité Aide

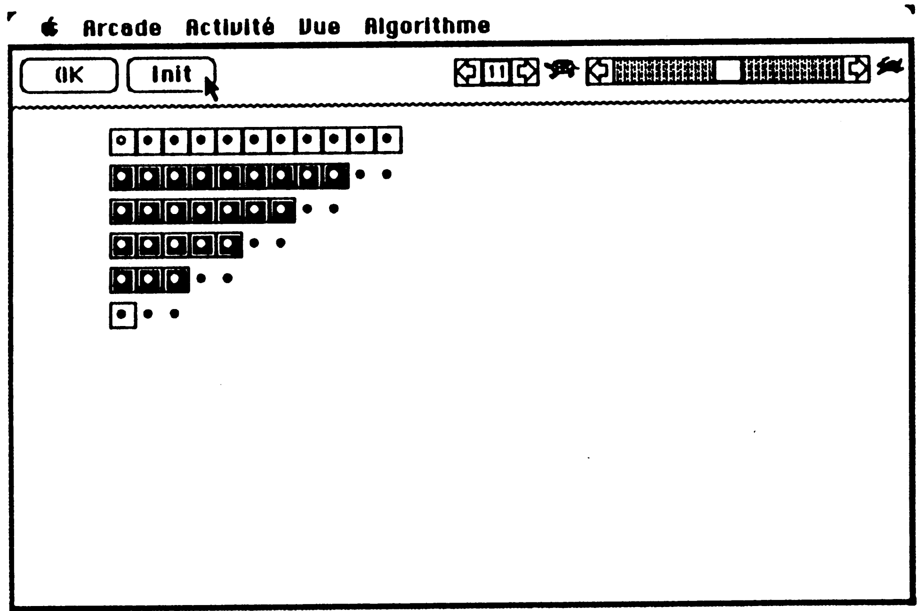
Programme | Peano | Peano

```
Péano : l'action (les données N : un entier > 0 ; C : un réel > 0 )
        ( N est la profondeur, C le côté du carré externe
          é.i.: pe= basse et pp = po et cap = do
          é.f.: la courbe définie par < C, N, po, do > est tracée
               et pe = basse, pp = p1, et cap = do )

      selon N :
         N = 1 :  Av(C*2*Racine(2))
         N > 1 :  Péano (N-1)
                  Ga(45) ; Av(K) ; Ga(45)
                  répéter 3 fois :
                    Péano(N-1)
                    Dr(45) ; Av(K) ; Dr(45)
                  répéter 3 fois :
                    Péano(N-1)
                    Ga(45) ; Av(K) ; Ga(45)
                  Péano(N-1)
```

Analyse :



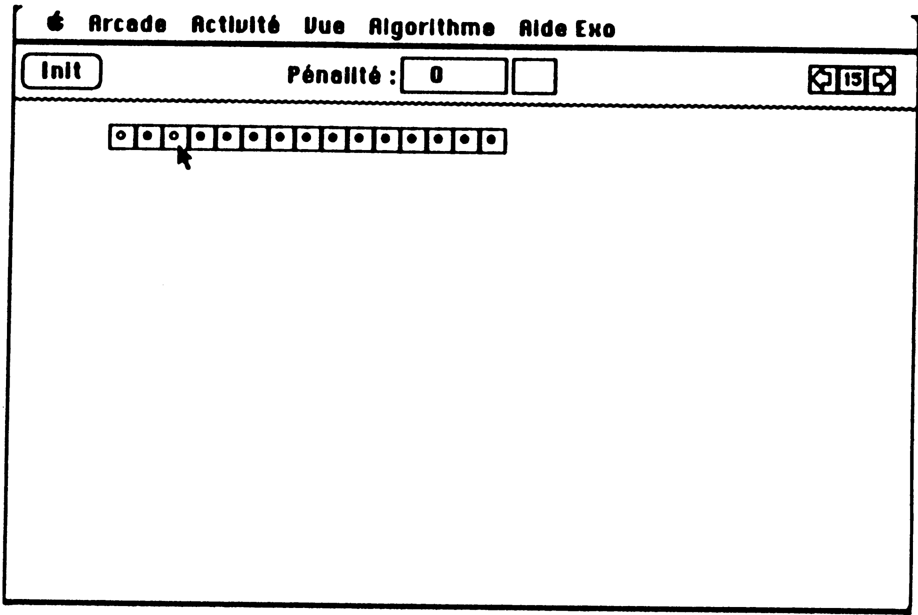
Exécution :



Exemple de l'exécution par appel.
Chaque appel est visualisé par une trace.

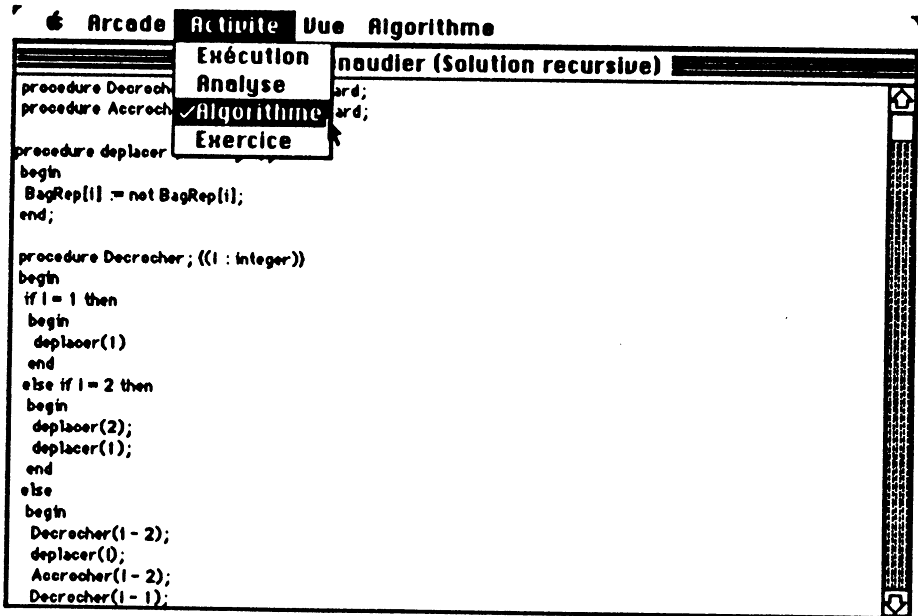
Baguenaudier

Exercice :

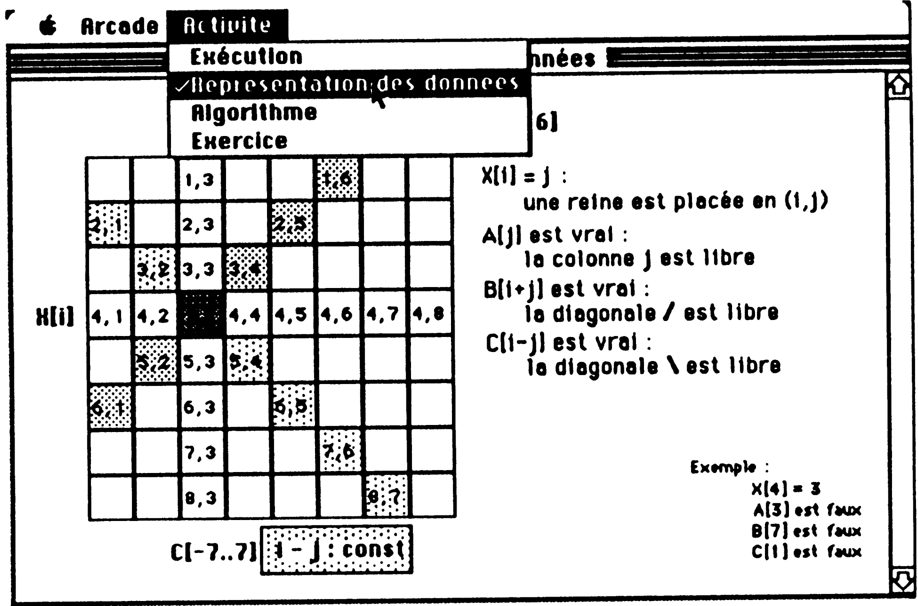


Pour décrocher tous les anneaux :
cliquer sur la case représentant l'anneau permet de changer sa position
(jeton noir = anneau accroché, jeton blanc = anneau décroché).

Lecture d'algorithme :

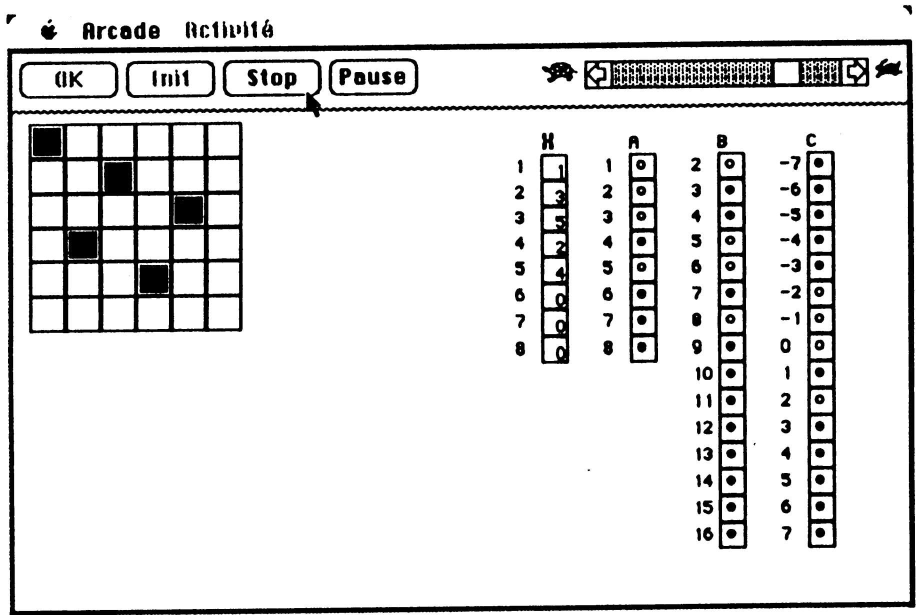


Représentation des données :



Choix des structures des données.

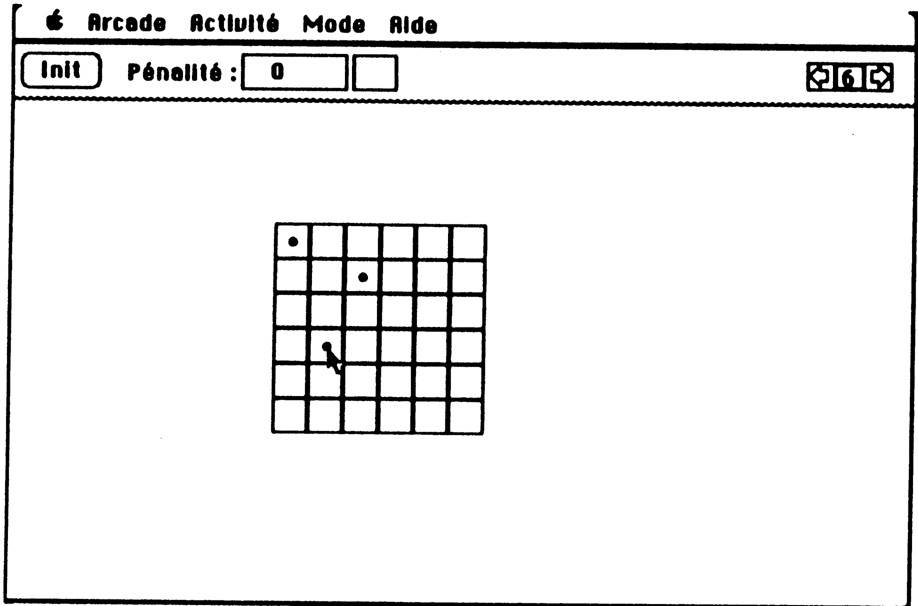
Exécution :



Exécution (6 reines) sur :
- échiquier (6 x 6),
- tableaux de booléens A, B, C,
- tableau d'entier X.

Les 8 reines

Exercice :



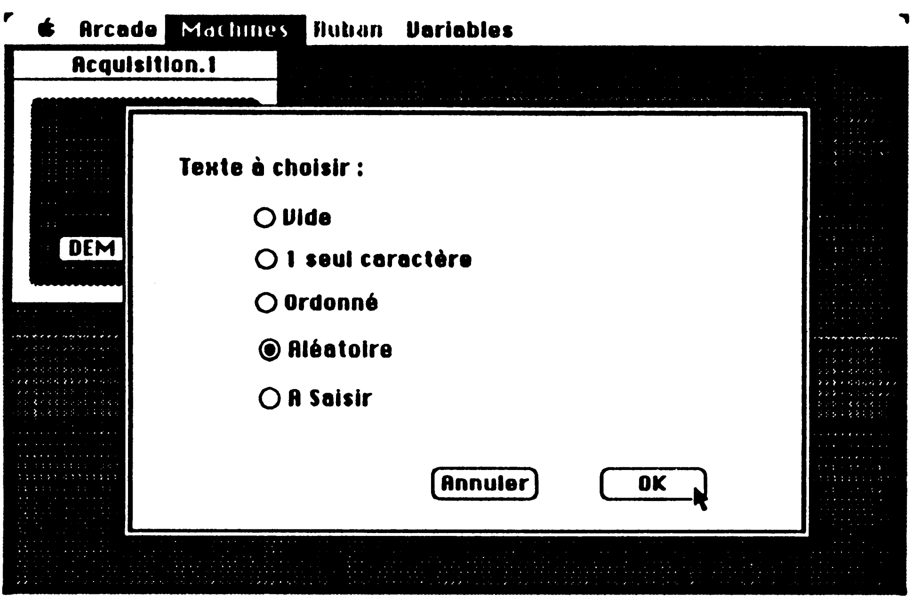
Placer les reines sur l'échiquier.

Lecture d'algorithme :

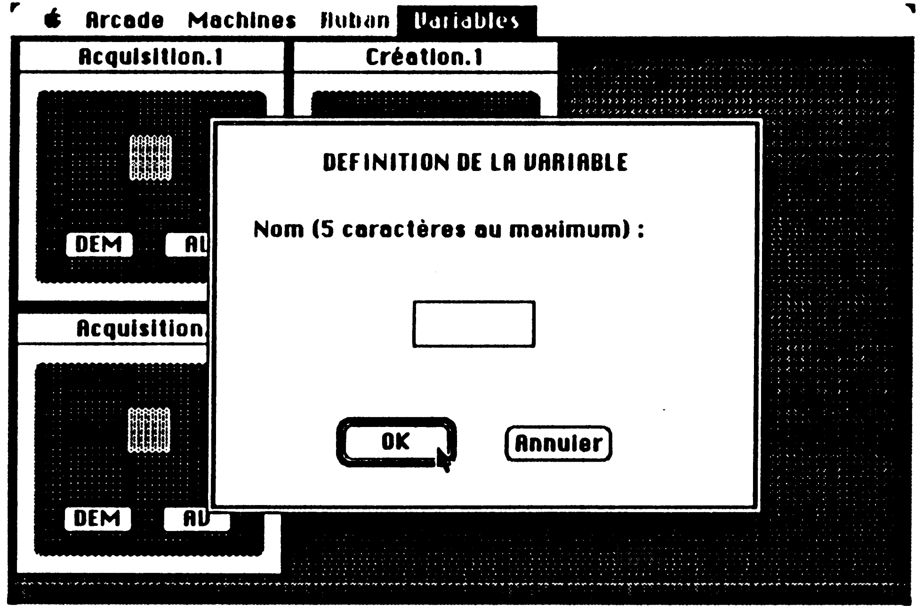
Arcade Activite
Exécution
Représentation des données
✓Algorithme
Exercice

```
procedure Queens;
 procedure Try (i :
   var q : boolean)
 var
  j : integer;
 begin
  j := 0;
  repeat
   j := j + 1;
   q := false;
   if (a[j] and b[i + j] and c[i - j]) then
   begin
    x[i] := j;
    a[j] := false;
    b[i + j] := false;
    c[i - j] := false;
    if (i < 8) then
    begin
     Try(i + 1);
     if not q then
     begin
      a[j] := true;
      b[i + j] := true;
      c[i - j] := true;
     end;
```

Choix d'un ruban :

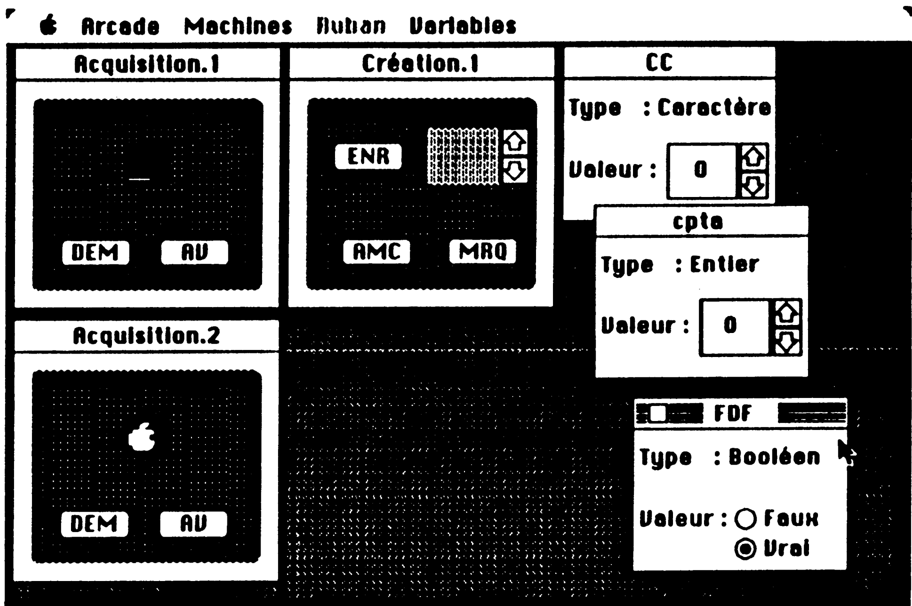


Choix d'une variable :



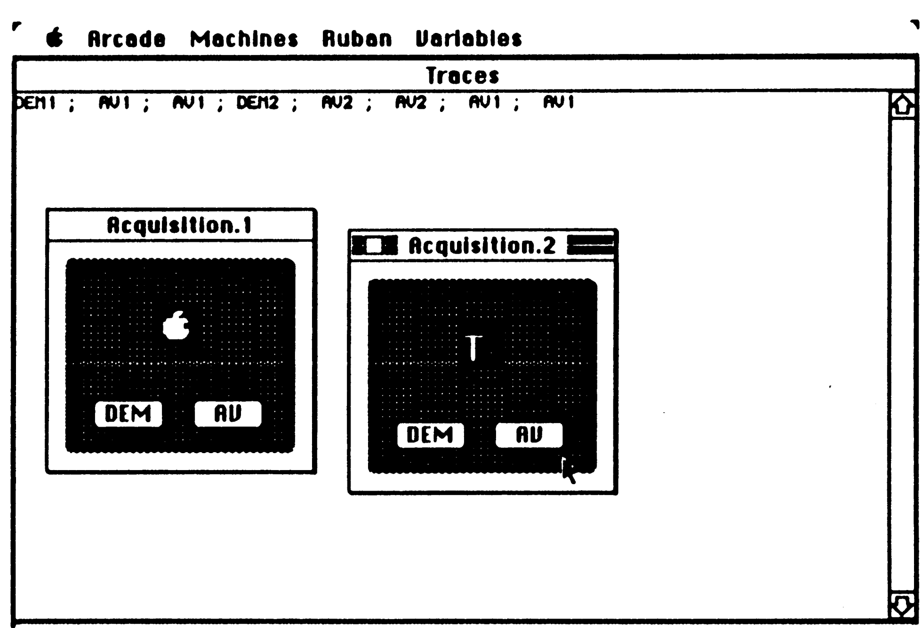
Machines caractères

Exemples des machines et des variables :



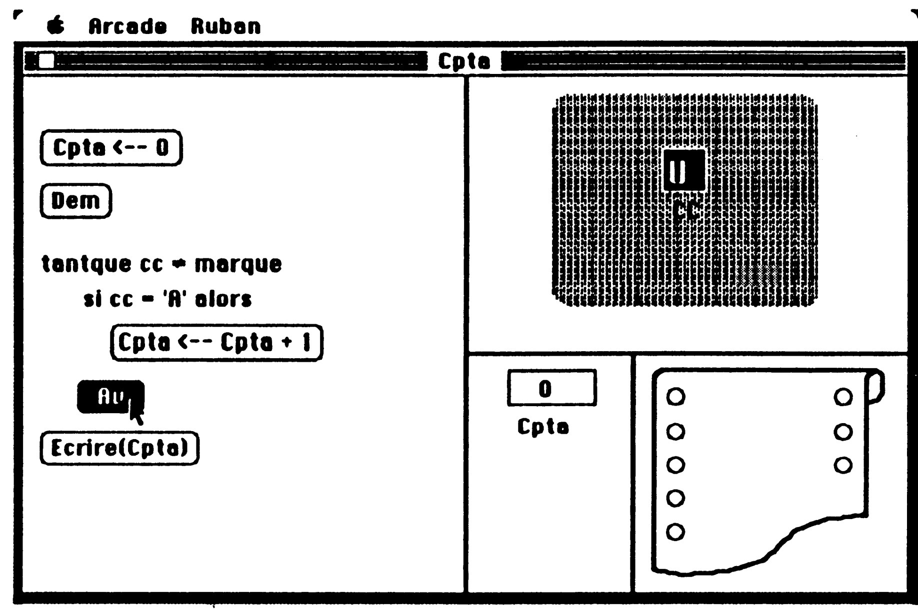
Machines d'acquisition, machines de création et fenêtres des variables.

La fenêtre "Traces" :

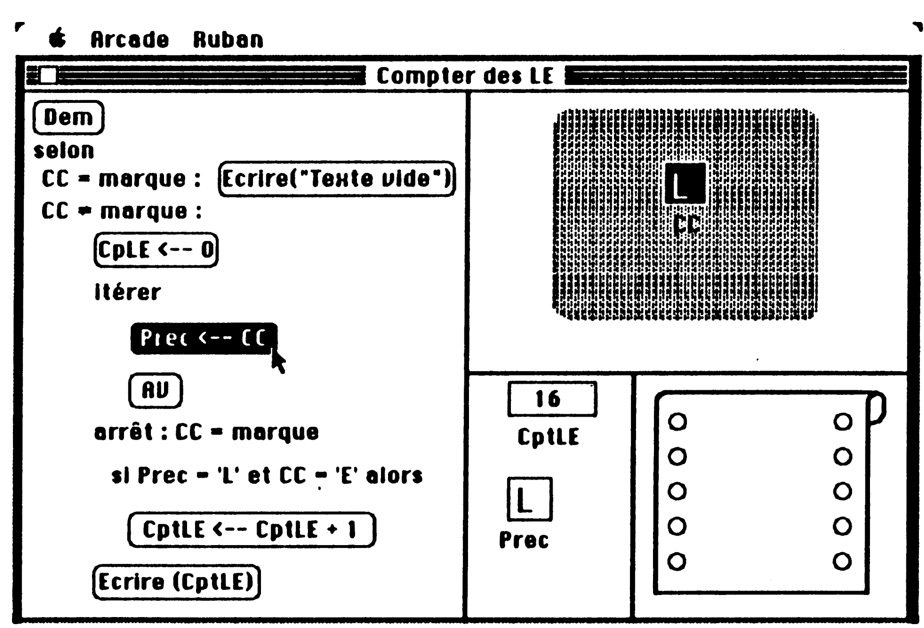


Relecture des actions faites par l'utilisateur.

Compter les "A"

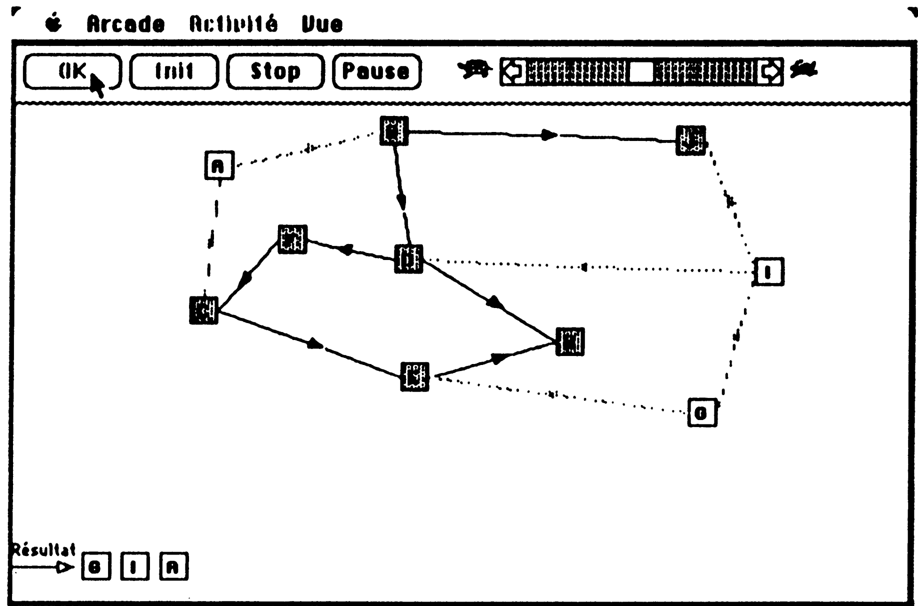


Compter les "LE"

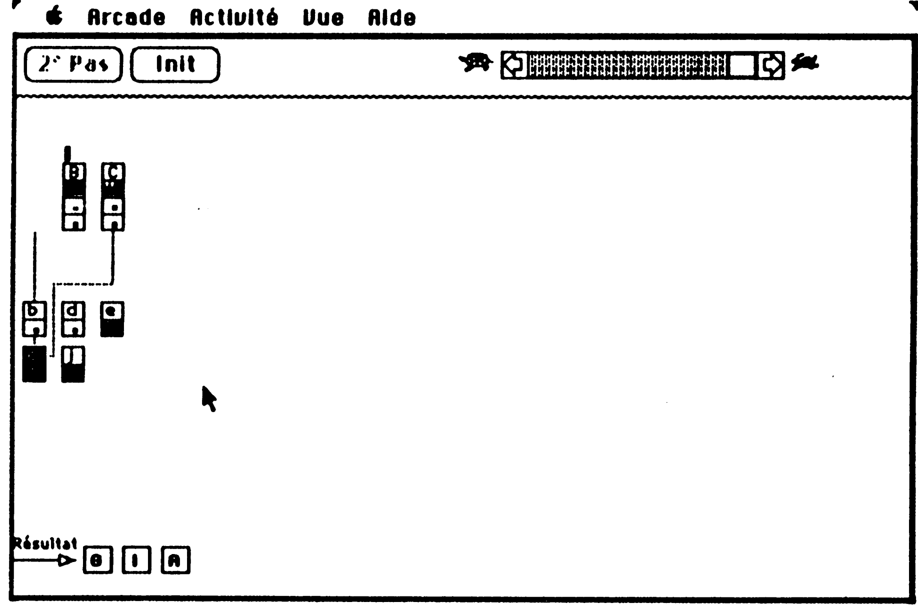


Tri topologique

Exécution :

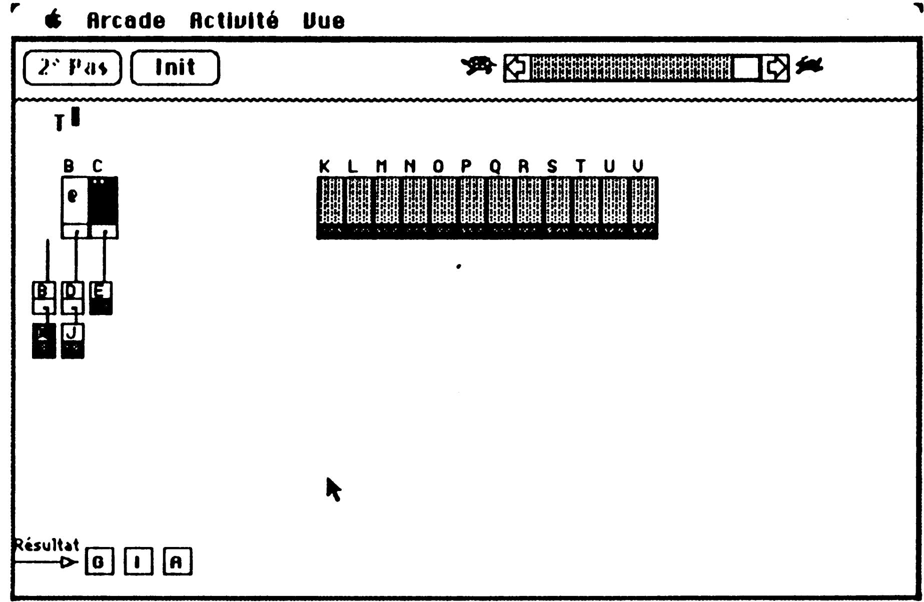


Exécution sur la structure logique.



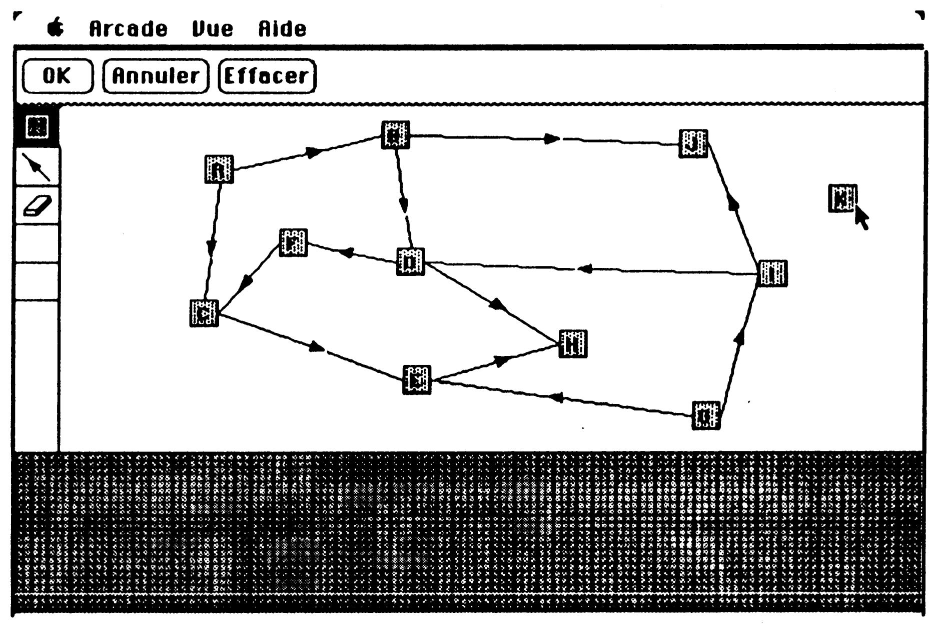
Exécution sur une structure physique simplifiée.

Exécution (suite) :



Exécution sur une autre structure physique simplifiée.

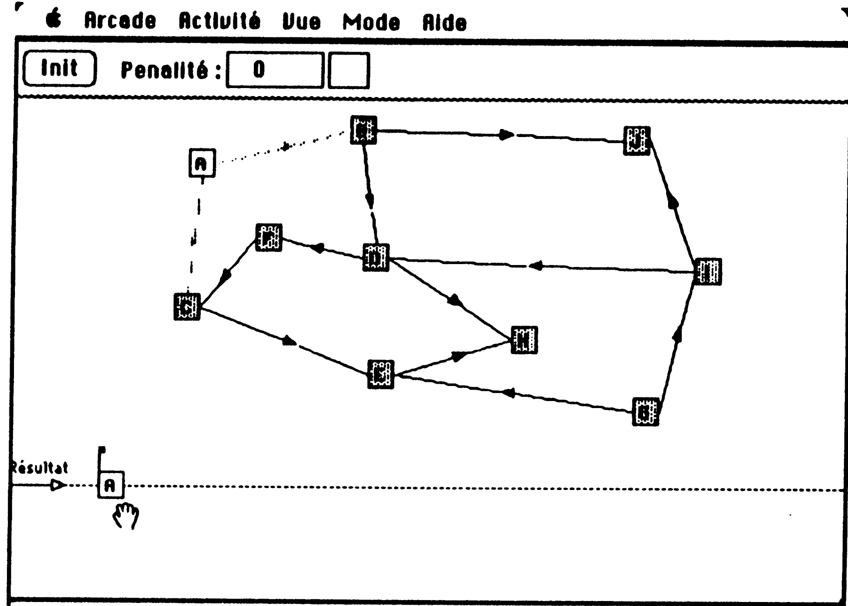
Environnement de saisie :



- Créer ou effacer un nœud.
- Créer ou effacer un arc.

Tri topologique

Exercice :



Amener les nœuds, un à un, dans la zone "Résultat".

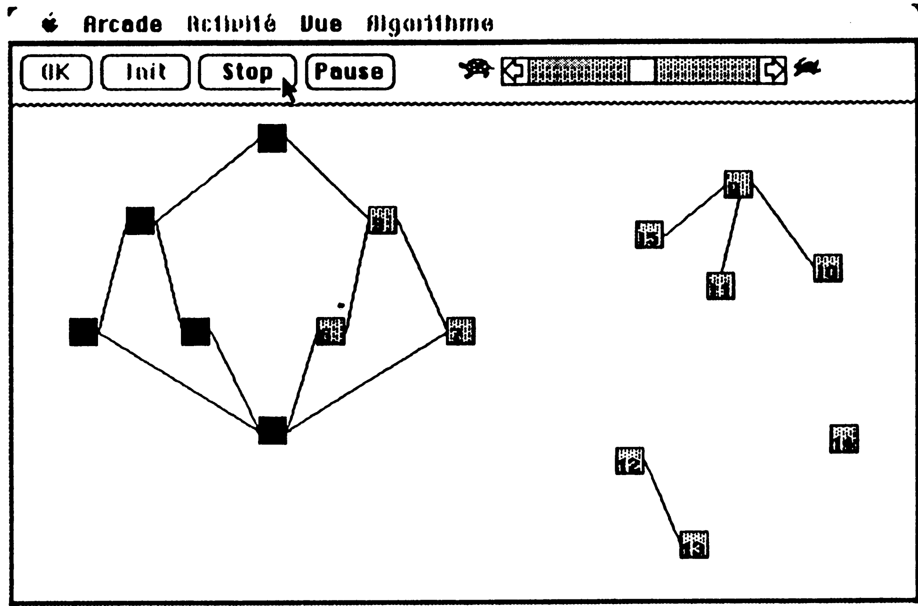
Lecture d'algorithme :

Arcade Activité Vue

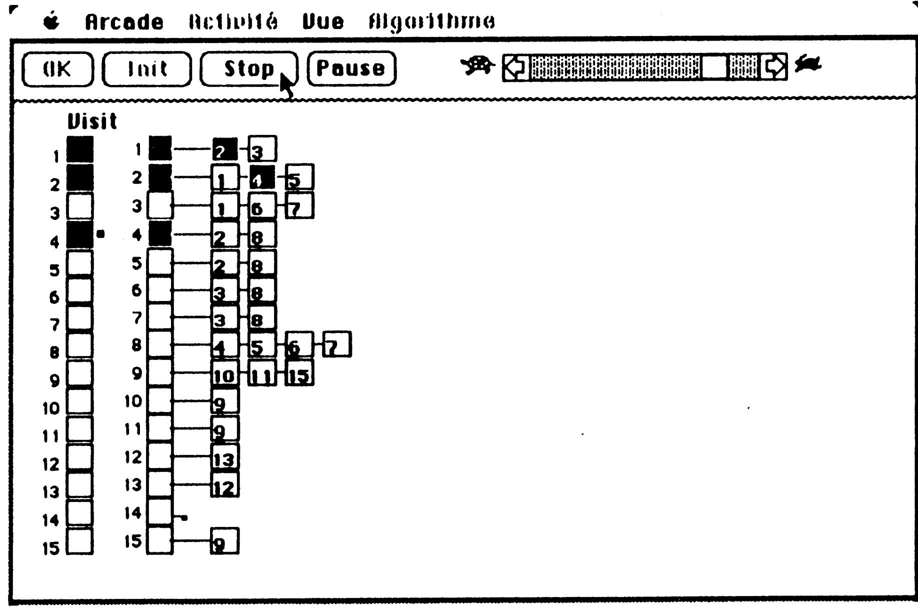
Programme Pascal d'après [Wirth-86]

```
program TopoSort;

Type
 Lptr= ^leader;
 Tptr= ^trailer;
 leader= record
        key,count:integer;
        trail : Tptr;
        next  : Lptr;
       end;
 trailer = record
         id:Lptr;
         next : Tptr;
        end;
var
 p,q,head,tail : Lptr;
 t : Tptr;
 x,y,n : integer;

function Find(w : integer) : LPtr;
 var
 h : LPtr;
begin
 h := head;
 tail^.key := w; (sentinel)
```

Exécution :



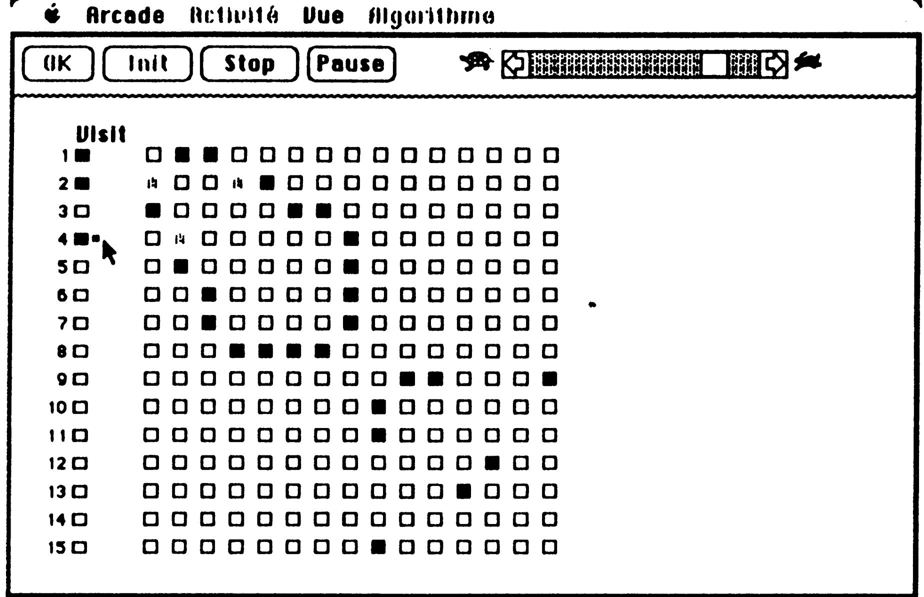
Exécution sur la représentation logique.



Exécution sur la représentation physique, liste d'adjacence.

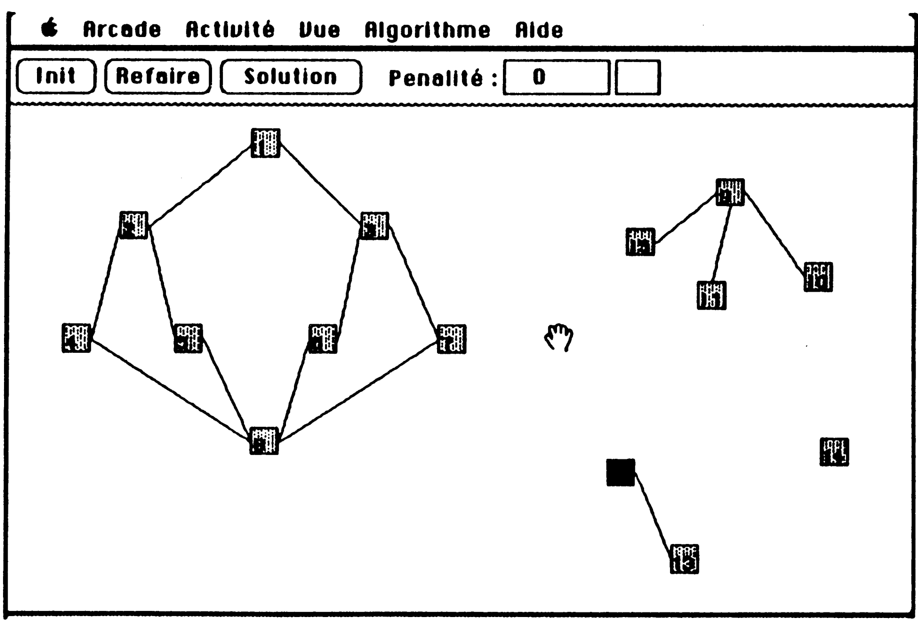
Parcours de graphes

Exécution (suite) :

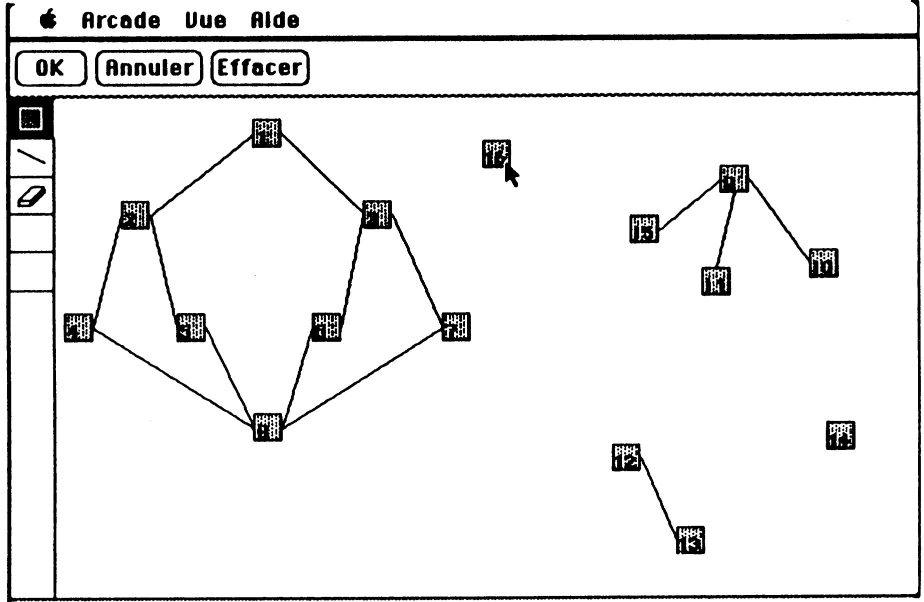


Exécution sur la représentation physique, matrice d'adjacence.

Exercice :



Environnement de saisie :



- Créer ou effacer un nœud.
- Créer ou effacer un arc.

Lecture d'algorithme :

Arcade Activite Vue Algorithme

Exécution
✓Algorithme
Exercice

dth First Search Algorithm

```
procedure DFSList(                    ylist;n : integer;v :integer);
{ Given an undirect                   by an adjacency list
  with n vertices a                   ] initially set to false,
  this subroutine  visits all vertices reachable from v
  visited is global and adjacencylist is a type global :
  type
    nextnode = ^node
    node = record
           vertex : integer;
           link : nextnode
         end;
    adjacencylist = array[1..n] of nextnode; }

 var
  w : integer;
  p : nextnode;

 begin
  visited[v] := true;
  p := adlist[v].link;
  while (p <> nil) do
   begin
    w := p^.v;
    if not visited[w] then
    DFSList(adlist, n, w);
```

TABLE DES MATIERES

Introduction 7

PREMIERE PARTIE : ETUDES PRELIMINAIRES 11

1. Motivations pédagogiques 13
1.1 Le savoir faire (faire) du programmeur 13
1.1.1 Observation du savoir faire 13
1.1.2 La contradiction entre le souci méthodologique et le besoin de concrétisation 22
1.2 Un moment pédagogique propre à l'étude de l'exécution des programmes 24
1.2.1 Difficultés face aux phénomènes dynamiques 25
1.2.2 Visualisation de l'exécution des programmes 26

2. Contexte de l'étude 31
2.1 L'enseignement support de l'étude 31
2.1.1 Notation algorithmique et schémas d'analyse 31
2.1.2 Machines abstraites 34
2.1.3 Un ensemble d'exemples types 35
2.1.4 Activités pratiques et leurs supports 36
2.2 Un environnement logiciel d'accueil des activités pratiques : le laboratoire ARCADE 42
2.2.1 Chronologie 42
2.2.2 Etat actuel du laboratoire ARCADE 48

3. Analyse d'expériences de visualisation 51
3.1 Méthodes et outils de visualisation 51
3.1.1 Image et texte 51
3.1.2 Supports statiques ou dynamiques 55
3.1.3 Visualisation et programmation 57
3.1.4 Visualisation et manipulation 60
3.1.5 Visualisation de programmes 62
3.2 Panorama de logiciels de visualisation 64
3.2.1 Assistance à la production de programmes 65
3.2.2 Animation de programmes 73
3.2.3 Un système d'animation : BALSA 82
3.3 Synthèse 91

DEUXIEME PARTIE : COMPTE RENDU DE REALISATION. 95

4. Conception et description des logiciels réalisés 97
4.1 Caractéristiques communes 97
4.1.1 Fonctionnalités 97
4.1.2 Méthode de travail 101
4.2 Visualisation et tableaux 103
4.2.1 Tri interne 103
4.2.2 Recherche dans un tableau 106
4.2.3 Reconnaissance d'un motif 108
4.3 Visualisation et récursivité 111
4.3.1 Dessins récursifs 112
4.3.2 Jeu de Baguenaudier 114
4.3.3 Huit reines 116
4.3.4 Méthodes de visualisation 118
4.4 Visualisation et machines abstraites 121
4.5 Visualisation et graphes 124
4.5.1 Tri topologique 124
4.5.2 Graphes non-orientés 127

5. Premier bilan pédagogique 129
5.1 Expérimentations à l'Institut de Technologie de Bandung 131
5.1.1 Profil des étudiants 131
5.1.2 Déroulement de l'expérimentation 132
5.1.3 Bilan pédagogique 132
5.2 Discussion 136
5.2.1 Une seule fenêtre active 136
5.2.2 Visualisation d'exécution et analyse 136
5.2.3 Diversité des données 137
5.2.4 Plusieurs algorithmes pour un même énoncé 138
5.2.5 Nécessité de plusieurs vues 139
5.2.6 Exécution pas à pas 141
5.2.7 Temps d'exécution 142
5.2.8 Limites 143
5.2.9 Contrôle de l'interaction 146
5.2.10 Aspect esthétique 147

6. Aspects techniques 149
6.1 Annotation des programmes à visualiser 149
6.1.1 Evénements et boîte à outils 149
6.1.2 Boucle d'événements 152
6.1.3 Annotation dans le cas de vues multiples 164
6.1.4 Squelette 166
6.2 Fonctions de visualisation 170
6.2.1 Changement de vitesse 170
6.2.2 Exécution pas à pas 172
6.3 Visualisation des structures de données 174
6.3.1 Mémorisation ou calcul 174
6.3.2 Association entre l'image écran et le programme annoté 175
6.3.3 Rafraîchissement de la fenêtre 177

6.4 Autres problèmes 178
6.4.1 Erreur de système 179
6.4.2 Difficulté du test 179
6.4.3 Convention standard de l'interface utilisateur 180
6.4.4 Retouche des dessins 181
6.4.5 Lightspeed Pascal comme outil de développement 183
6.5 Résumé 185

Conclusion 187

Références bibliographiques 191

Annexes
Annexe A. Le laboratoire ARCADE 211
Annexe B. Synopsis des logiciels réalisés 217

Table des matières 251

Table des illustrations 255

TABLE DES ILLUSTRATIONS

Figure 1.1 Exemples de comptes rendus 15
Figure 1.2 Unification des comptes rendus 16
Figure 1.3 Expression algorithmique de la longueur d'un texte 16
Figure 1.4 Etat initial du Baguenaudier 17
Figure 1.5 Etat intermédiaire du Baguenaudier 18
Figure 1.6 Etat final du Baguenaudier 18
Figure 1.7 Solution récursive du problème de Baguenaudier 19
Figure 1.8 Solution itérative du problème de Baguenaudier 20
Figure 1.9 Insertion d'un dans un arbre 25
Figure 1.10 Etats possibles de l'arbre après insertion 25
Figure 1.11 Etat de l'arbre après insertion 26

Figure 2.1 Exemple d'utilisation de la notation **selon** 32
Figure 2.2 L'action de la figure 2.1, écrite avec la notation **si-alors-sinon** 32
Figure 2.3 Une succession d'exemples types sur l'itération et le traitement de séquences 35
Figure 2.4 Salles du laboratoire Arcade et leurs logiciels 49

Figure 3.1 Représentation d'un tableau 51
Figure 3.2 Environnement de programmation, assistance à la production de programmes 67
Figure 3.3 Représentation des nœuds colinéaires 72
Figure 3.4 Animation de programmes 74
Figure 3.5 Résumé du thème "Programme + Visualisation" 93

Figure 4.1 Choix des sujets 98
Figure 4.2 Récapitulation des réalisations 100
Figure 4.3 Exemples d'exécution de ** Tris internes** 104
Figure 4.4 Différentes représentations d'un tableau 106
Figure 4.5 La représentation codée de la chaîne de caractères et du motif 108
Figure 4.6 La représentation exacte de la chaîne de caractères et du motif 109
Figure 4.7 Comparaison du motif sans montrer le détail 109
Figure 4.8 Différentes courbes dans ** Dessins récursifs** 113

Figure 4.9 Solution itérative du jeu de Baguenaudier.......................... 115
Figure 4.10a Base de la solution récursive du jeu de Baguenaudier.......... 115
Figure 4.10b Récurrence de la solution récursive du jeu de Baguenaudier. 115
Figure 4.11 Représentations de données du problème des huit reines [Wir 86].. 117
Figure 4.12 Images écran de l'exécution de **Les 8 reines**.................. 117
Figure 4.13 Traces des appels récursifs dans **Baguenaudier**............. 118
Figure 4.14 Visualisation sans mise en évidence des niveaux d'appels récursifs.. 119
Figure 4.15 Visualisation avec mise en évidence de chaque niveau d'appel récursif.................................. 119
Figure 4.16 Visualisation qui montre chaque figure complète d'un niveau................................ 120
Figure 4.17 Les fenêtres manipulables dans **Machines caractères**.... 121
Figure 4.18 **Compter les "A"**... 122
Figure 4.19 Représentation du ruban sans la machine.......................... 123
Figure 4.20 Représentation logique.. 125
Figure 4.21 Représentation chaînée [Wir 86] : des nœuds et leurs successeurs.. 125
Figure 4.22 Liste d'adjacence [HoS 84] : l'ensemble des nœuds est présenté dans une table, et leurs successeurs sont chaînés. 126
Figure 4.23 Représentation logique d'un graphe non-orienté................. 127
Figure 4.24 Représentation sous forme de "liste d'adjacence" du graphe précédent :les nœuds sont présentés dans un tableau et leurs successeurs sont chaînés.. 127
Figure 4.25 Représentation sous forme de "matrice d'adjacence" du graphe de la figure 4.23, chaque booléen de la matrice représente un arc................. 128

Figure 5.1 Le graphe.. 140
Figure 5.2 Un arbre de recouvrement sur la représentation logique du graphe précédent.. 140
Figure 5.3 Représentations physiques du graphe de la figure 5.1.......... 141
Figure 5.4 Visualisation de deux grandes valeurs.............................. 144
Figure 5.5 Courbes de Hilbert d'ordre 5, 6 et 7 avec une surface et épaisseur de trait identique.. 145
Figure 5.6 Courbes de Sierpinski et courbes de Hilbert superposées par 2 niveaux d'appels.. 146

Figure 6.1 Structure générale d'un programme Pascal dans l'environnement Macintosh.................................... 153
Figure 6.2 Recherche séquentielle... 155
Figure 6.3 Procédure en pas à pas (seul le pas "npas" est exécuté et visualisé)....................... 156
Figure 6.4 Recherche séquentielle, annotée (seul le pas "npas" est visualisé)................................... 157
Figure 6.5 Le programme principal de l'exécution d'une procédure en "pas à pas"...................................... 158
Figure 6.6 Programme principal d'une animation de programme....... 160
Figure 6.7 Gestionnaire d'interruption et de temporisation pendant l'exécution... 161
Figure 6.8 Exemple d'un programme annoté pour la visualisation...... 162
Figure 6.9 Diverses vues de la valeur 25... 164
Figure 6.10 Modules fonctionnels du squelette................................ 168
Figure 6.11 Procédure qui "contrôle" la vitesse de l'exécution............ 171
Figure 6.12 Procédure qui "contrôle" la vitesse de l'exécution, basée sur un compteur... 171
Figure 6.13 Procédure qui "contrôle" la vitesse de l'exécution, basée sur l'horloge... 172
Figure 6.14 Déclaration d'un nœud d'un graphe............................... 174
Figure 6.15 Evaluation du point (X,Y) par rapport au milieu de la ligne AB... 176
Figure 6.16 Tracé d'un carré et d'une ligne..................................... 181
Figure 6.17 Agrandissement du carré de la figure 6.16...................... 181
Figure 6.18 Tracés de deux lignes de même longueur avec différentes tailles de la plume................................ 182
Figure 6.19 Effets de différentes tailles de la plume.......................... 183

AUTORISATION DE SOUTENANCE

DOCTORAT 3ème CYCLE, DOCTORAT INGENIEUR,

DOCTORAT DE L'UNIVERSITE JOSEPH FOURIER - GRENOBLE 1

Vu les dispositions de l'Arrêté du **16 avril 1974**,

Vu les dispositions de l'Arrêté du **5 juillet 1984**,

Vu les rapports de **M** FILLIATRE Bernard

M TRILLING Laurent

Mme LIEM Inggriani est autorisé(e) à présenter une thèse en vue de l'obtention du doctorat de l'UJF mention Informatique

Grenoble, le 08 SEP. 1989

Le Président de l'Université
Joseph Fourier - Grenoble 1

A. NEMOZ

Visualisation de l'exécution des programmes pour l'enseignement de la programmation

Résumé :

Cette recherche s'insère dans le projet ARCADE dont l'objet est la construction d'une collection de logiciels de soutien à l'enseignement de la programmation. Chaque thème est illustré par diverses activités telles que la manipulation, le jeu et l'observation.

La première partie de cette thèse présente une réflexion sur l'interprétation opératoire des programmes dans cet enseignement. Cette réflexion conduit à définir un moment pédagogique particulier centré sur l'utilisation de logiciels de visualisation de l'exécution des programmes.

La deuxième partie présente la conception, la réalisation et une première évaluation d'une dizaine de logiciels de ce type qui recouvrent les principaux thèmes d'un enseignement fondamental de la programmation impérative. Une bibliothèque d'outils de visualisation et un squelette d'animation de programmes sont extraits des logiciels réalisés.

Mots clés :

Méthodologie de la programmation, Didactique de la programmation, Enseignement Assisté par Ordinateur, Animation des programmes, Visualisation des structures des données.

Visualisation of program execution for the teaching of programming

Abstract :

This research contributes to the ARCADE project, whose objective is to build a collection of programming courseware. Each theme is illustrated by various activities, such as manipulation, game and observation.

The first part of the thesis explores the role of execution in the teaching of programming. This leads to a definition of a precise pedagogical phase centered on the visualisation of program execution.

The second part presents the design and implementation, and a first evaluation of ten courseware which cover the main themes of a fundamental course on imperative programming. A library of visualisation tools and a program animation skeleton are extracted from these implementations.

Key words :

Programming methodology, Didactics of programming, Computer Assisted Instruction, Program animation, Visualisation of data stuctures.