PÉGASE, UN SYSTÈME POUR L'APPRENTISSAGE DES CONCEPTS DE BASE DE LA PROGRAMMATION

François HEUZE

PRÉAMBULE

J'enseigne l'option informatique depuis six ans... Ainsi commençait cet article et je le mettais sous enveloppe lorsqu'on annonça la suppression de l'option ! Le bouleversement que provoque cette décision va évidemment bien au-delà de la brusque obsolescence de ce texte. Reste-t-il un intérêt quelconque à le lire ? Oui, je le pense, pour deux raisons :

- il peut encore être utile à ceux qui auraient à enseigner la programmation au lycée (l'option continue malgré tout encore un peu) ou dans d'autres structures.
- il est un témoignage de l'engagement profond de bon nombre d'entre nous en pédagogie de l'informatique. Souhaitons que l'acquis des recherches dans ce domaine ne soit pas perdu, en attendant que l'on redécouvre les vertus formatrices de la programmation.

Je reprends donc l'exposé en espérant que, s'il ne parlait plus au lecteur en tant que sujet d'avenir, celui-ci y trouve au moins le charme des choses anciennes ...

Introduction

J'enseigne l'option informatique depuis six ans. Pour présenter la programmation aux élèves de seconde, j'ai élaboré successivement diverses progressions utilisant d'abord le LSE, puis le BASIC, puis le PASCAL avec ou sans langage de description algorithmique ; je n'en ai jamais été vraiment satisfait, ne pouvant focaliser l'attention des élèves sur les choses importantes (dans mon lycée, pas de sélection, pas d'élitisme, nous accueillons tous les élèves volontaires). Je souhaitais un langage beaucoup plus simple permettant de bien mettre en place les

concepts de base sans trop formaliser et qui soit une étape dans l'étude d'un des langages précédents, incontournables pour des raisons institutionnelles. J'ai réalisé à cet effet un ensemble logiciel (PEGASE). Je le présente dans cet article, après un exposé des éléments qui ont guidé ma réflexion. Je montrerai ensuite comment il sert de support à ma démarche pédagogique.

1 - A propos de BASIC, PASCAL et LSE

Il n'est pas étonnant que les élèves de l'option aient du mal avec ces langages, appelons les professionnels. Ils supposent d'emblée que leurs utilisateurs aient parfaitement intégré la finalité de l'action de programmation, ce qui n'est pas le cas de nos élèves. De plus, leurs performances informatiques incontestables impliquent des environnements complexes qui peuvent devenir une gêne pour dégager l'essentiel. Enfin, certaines de leurs caractéristiques sont rédhibitoires pour l'apprentissage.

Jeu : retrouver le(s) langage(s) au(x) quel(s) s'appliquent les reproches suivants :

Pas d'erreur si une variable n'a pas de valeur

Variables non déclarables

Obligation de numéroter des lignes

Messages d'erreur en anglais

Difficultés pour imbriquer des choix sur une ligne logique

Usage d'une syntaxe anglaise

Transformation obligatoire de toutes les boucles en TANT QUE

Editeur de texte précolombien

Ecrire *l''informatique* pour obtenir *l'informatique*

Limitation des identificateurs à 5 caractères

Mettre des points-virgules où il faut

Ne pas mettre des points-virgules où il ne faut pas

2 - Le langage PEGASE

2.1 Un programme PEGASE est un texte écrit dans une syntaxe dont les mots-clés peuvent être redéfinis.

Le parenthésage des blocs d'instructions s'exprime par indentation dans le texte : un décalage à droite ouvre un bloc, un retour à gauche le ferme. Cela réduit la taille des programmes en supprimant les marques

de début et de fin de bloc. L'élève est en outre contraint à des écritures claires.

Les variables peuvent être déclarées, c'est à dire que le système ne produira une erreur que s'il ne peut pas déduire le type à partir du contexte. Ainsi la ligne

```
Message <- "Bonjour"
```

déclare la variable Message comme chaîne de caractères si elle ne l'a pas été auparavant. Un message explique à l'élève le sens de la correction.

Remarque : le symbole <- peut être redéfini.

Une variable doit avoir une valeur avant d'être utilisée. Cette contrainte a évidemment ses limites mais est très utile pour les programmes (simples) faits par des débutants. PEGASE n'est évidemment pas un outil de développement de logiciels !

2.2 Voici un exemple de programme PEGASE, celui du jeu où il faut trouver le nombre choisi au hasard par l'ordinateur :

```
° * Jeu : trouver le nombre choisi par l'ordinateur
° atrouver,compt,n:entiers
° atrouver <- hasard(100)
° compt <- 0
° boucle
°   compt <- compt + 1
°   écris "Propose un nombre entre 0 et 100 "
°   lis n
°   quand n=atrouver sors
°   teste n>atrouver
°      sivrai
°         ? "TROP GRAND"
°      sifaux
°         ? "TROP PETIT"
° ? "tu as trouvé en ",compt," coups"
```

les mots-clés utilisés ici : entier, boucle, écris, lis, quand, sors, teste, sivrai, sifaux *peuvent être redéfinis.*

2.3 Comment ce programme est-il exécutable ?

Je n'ai pas écrit un compilateur mais un traducteur (PEGASE.EXE). Pour une exécution, le texte précédent doit d'abord être traduit dans un des langages professionnels suivants : PASCAL, BASIC, LSE ou C. L'envoi du résultat de la traduction sur un compilateur (ou un

interpréteur) permet d'obtenir l'exécution. Dans un premier temps, cet enchaînement de tâches est réalisé par un fichier de commandes, ce qui rend la traduction totalement transparente à l'élève. Ensuite, la possibilité de disposer de cette traduction peut être un support pour l'apprentissage du langage professionnel.

2.4 Autres caractéristiques de PEGASE :

- Messages en français, voyelles accentuées acceptées.
- Sous-programmes (sans paramètres mais avec variables locales : celles-ci ne le seront évidemment plus dans la traduction BASIC).
- Il est possible de demander la traduction en mode "débogage". Chaque affectation est alors suivie de l'instruction d'affichage de la valeur de la variable affectée.
- Diverses instructions graphiques et musicales reconnues. Pour une traduction en TURBO PASCAL, la présence d'une instruction graphique génère les instructions de basculement dans le mode graphique adéquat (initgraph, closegraph ...).

3 - Encore un langage !

A ceux qui relèveront le possible aspect superfétatoire d'un nouveau langage, je dirai que PEGASE n'a d'autre ambition que de faciliter les premiers pas en programmation. Il permet d'écrire dès le départ des algorithmes et d'en obtenir des exécutions. Le langage de description algorithmique ainsi mis en place peut être utilisé jusqu'en terminale.

PEGASE tente de réduire le formalisme comme il a été montré précédemment et permet une unification de certains concepts à travers une démarche pédagogique spécifique.

4 - De l'abstraction

4.1 D'une manière générale une bonne démarche pédagogique impose de présenter les concepts un à un dans un ordre de dépendance. Ainsi on devra placer mémoire informatique avant affectation, affectation avant variable, condition avant choix, portée d'une instruction avant choix et avant boucle, séquentialité avant portée d'une instruction ; cette notion de portée d'instruction est contenue en fait dans celle d'arbre.

Dans ce souci de dépendance des concepts, une difficulté se présente : faut-il parler des boucles avant ou après les vraies variables

(j'appelle vraie variable celles qui changent de valeur pendant l'exécution) ? En effet une vraie variable n'existe que dans un contexte de boucle et pour écrire une boucle il faut disposer au moins d'une vraie variable, celle justement dont le changement de valeur fera sortir de la boucle !

J'ai résolu ce problème en proposant une structure *répète n fois* qui permet deux choses :

- l'expression des boucles "au nombre de fois connu" sans recours à la notion de variable. La structure habituelle for *i:=1 to n* ne me paraît pas bonne du point de vue pédagogique ; en effet, c'est le système qui gère la vraie variable i et non l'élève alors qu'il a été placé dès le départ dans la situation où le comportement de ses variables était de sa responsabilité. De plus cette variable (qui doit être déclarée) n'est pas forcément utilisée dans la boucle (voir par exemple plus loin le programme sur l'échiquier).
- la manipulation de variables en dehors du contexte de condition d'arrêt.

4.2 Du concept d'arbre

L'unification des concepts est à la fois un bon exercice d'abstraction et un outil pouvant la simplifier. Dans cette idée, il me semble que le concept d'arbre doit être mis très tôt en évidence. Il est en effet partout présent en informatique. On le rencontre, on vient de le voir, dans l'écriture d'un programme (qui est une arborescence d'instructions), dans l'organisation des répertoires d'un disque ou des commandes d'un logiciel. D'autre part, concevoir un choix c'est concevoir un arbre.

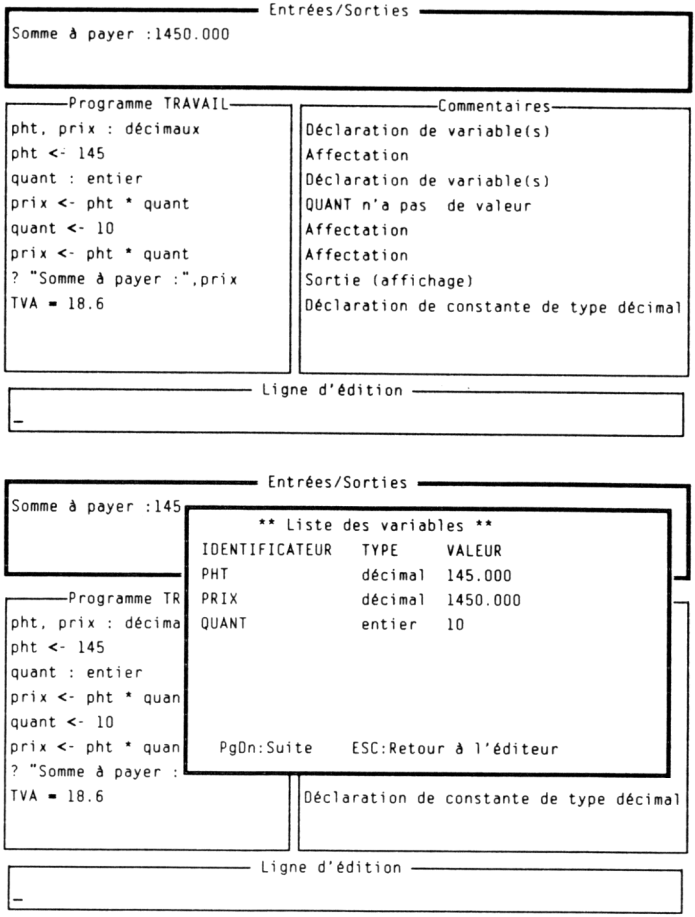
C'est pourquoi je fais d'abord travailler les élèves sur cette notion, en dehors de tout contexte de programmation : apprendre à reconnaître des structures d'arbres, les exprimer par des textes en utilisant la technique d'indentation, préparer ainsi l'écriture des programmes. Un éditeur disposant de l'indentation automatique est évidemment une aide précieuse.

5 - Une progression

Je présente maintenant la démarche pédagogique à laquelle PEGASE sert de support. Outre le traducteur (PEGASE.EXE), j'ai réalisé un logiciel (PEGINTER.EXE) qui permet de travailler en mode interprété avec tous les avantages pédagogiques de ce mode :

consultation des variables, exécution en pas à pas, correction aisée ... L'élève entre des instructions une à une ; celles-ci sont commentées par le logiciel.

5.1 A l'aide de cet environnement, j'aborde les concepts de mémoire informatique, de types de données, de réservation de mémoire (déclaration), de prise de valeur (affectation) et de sortie. Voici un exemple d'écran (l'élève a entré huit instructions) :



Après demande d'affichage de la liste des variables :

La fenêtre Entrées/Sorties se comporte comme l'écran de l'ordinateur pendant une exécution normale. En effet, un programme

écrit avec le logiciel PEGINTER peut, après sauvegarde, être traduit puis compilé pour une exécution normale. De nombreux programmes séquentiels simples peuvent alors être faits en utilisant les instructions de déclaration, d'affectation et d'affichage à l'écran ou à l'imprimante (j'ai prévu une instruction de sortie sur imprimante : *! expression*).

5.2 J'introduis ensuite l'instruction de lecture. Celle-ci est présentée comme une commodité évitant de reprendre la séquence édition-compilation pour une exécution avec une nouvelle valeur.

5.3 L'élève ayant appris à exprimer des arbres par indentation l'applique avec la structure de répétition "au nombre de fois connu" : répète n fois. Voici un exemple d'algorithme PEGASE qui dessine un échiquier ou un damier :

```
° * Dessine un échiquier ou un damier
° écris "Pour un échiquier entrez 4,
°        pour un damier entrez 5 "
° nb:entier
° lis nb

° répète nb fois
°   répète nb fois
°     caseblanche
°     casenoire
°   alaligne
°   répète nb fois
°     casenoire
°     caseblanche
°   alaligne
°
° procédure caseblanche
°   écris "  "
°
° procédure casenoire
°     écris "ÜÜ"
°
° procédure alaligne
°   ?
```

5.4 J'aborde ensuite la notion de variable, en commençant par les compteurs. Ceux-ci sont gérés "à la main" de la façon suivante:

```
° * Compte de 1 à 10:
° compteur : entier
° compteur <- 0
° répète 10 fois
°   compteur <- compteur+1
°   ? compteur
```

5.5 Je montre ensuite la sensibilité de l'ordinateur aux tests. Avec PEGINTER, les élèves entrent des instructions du type : *teste condition*. Un message donnant la valeur du test est affiché dans la fenêtre Commentaires. Cette étape permet d'insister sur la notion de condition en dehors des choix ou des boucles. C'est l'occasion d'apprendre à formaliser par des expressions mathématiques des situations telles que "le nombre est nul", "la chaîne se termine par ER", "on a dépassé le nombre de coups autorisés"...

5.6 L'instruction *teste condition* est la racine d'un arbre (l'arbre de choix) qui s'exprime aussi par indentation. Voici un exemple d'algorithme qui envoie un texte à l'écran ou à l'imprimante:

```
° ? "Entre le texte à afficher"
° texte: chaîne
° lis texte
° ? "Ce texte doit s'afficher sur l'écran (E)
°    ou sur l'imprimante (I) ? "
° réponse : chaîne
° lis réponse
° teste tma(réponse)="I"
°  sivrai
°      ? "Le texte va sur l'imprimante"
°      ! texte
°   sifaux
°      ? "Le texte va sur l'écran:"
°      ? texte
```

On peut permuter les blocs *sivrai* et *sifaux*, ce qui montre bien la différence de nature entre l'arbre de choix et l'arbre syntaxique. Je me suis aperçu que les élèves tirent profit de cette possibilité ; en effet, ils ont tendance à placer en premier le bloc le plus facile à écrire en

appliquant le précepte "gardons le plus dur pour la fin". Pouvoir jouer sur l'inversion des blocs leur évite de chercher la négation de la condition, exercice sans intérêt à ce niveau de l'analyse.

5.7 Les notions sont maintenant en place pour aborder le concept général de boucle. La forme retenue permet d'exprimer tous les cas: boucle à une ou plusieurs conditions de sortie, placées au début, au milieu ou à la fin de la boucle :

```
° boucle
°   [Action1]
°   quand [Condition1] sors
°   [Action2]
°   quand [Condition2] sors
°   [Action3]
```

J'ai remarqué une nette tendance des élèves à placer la condition de sortie en fin de boucle. La mettre au début ne leur vient pas naturellement à l'esprit car ils expriment alors une action qui ne s'effectuera peut-être pas, ce qu'ils ont du mal à imaginer.

5.8 Arrivé à ce point, je présente le PASCAL en prenant appui sur PEGASE : il suffit d'appliquer les mêmes schémas de traduction. L'élève peut aussi apprendre en comparant son codage à celui donné par le logiciel. En fait, deux années de travail avec PEGASE m'ont montré que seuls les bons élèves acceptent de faire cet effort supplémentaire. Les autres préfèrent continuer avec PEGASE, ce qui d'ailleurs ne pose aucun problème en seconde. Le PASCAL sera vu ou revu en première.

A titre d'exemple, voici les schémas utilisés pour traduire la structure boucle en PASCAL avec une condition de sortie :

* boucle à une condition de sortie placée au début *

PEGASE	PASCAL
boucle	while not [condition] do BEGIN
quand [condition] sors	[Action];
[Action]	END;

* boucle à une condition de sortie placée à la fin *

PEGASE	PASCAL
boucle	repeat
[Action]	[Action];
quand [condition] sors	until [condition];

* autres cas *

PEGASE	PASCAL
boucle	fini := false; while not fini do BEGIN
[Action1]	[Action1];
quand [condition] sors	if [condition] then fini:=true else BEGIN
[Action2]	[Action2];
	END; END;

Je ne prétends évidemment pas avoir résolu tous les problèmes d'apprentissage. Les utilisateurs de ce système (actuellement huit professeurs de l'Académie de CAEN) ont cependant le sentiment que cette approche est bien adaptée aux élèves de seconde.

Les fichiers et une documentation du produit sont disponibles au CRDP de CAEN : rue du Moulin au Roy - 14000 CAEN CEDEX.

François HEUZE
Professeur coordonnateur
de l'option informatique
Lycée Allende
14200 HEROUVILLE SAINT CLAIR