Transition de l'activité débranchée à la programmation avec AlgoTouch

Patrice Frison[1] and Moncef Daoud[2]

[1] Université de Bretagne Sud - Laboratoire IRISA
[2] Université de Bretagne Sud

Résumé L'apprentissage de la programmation peut se faire de différentes manières. On distingue habituellement les activités débranchées de l'activité branchée de programmation. Ce papier propose une démarche pour passer progressivement de l'activité débranchée à la programmation en utilisant AlgoTouch, un outil de programmation par démonstration.

Mots-clés Algorithmes, Programmation pour débutants, Programmation par démonstration, Activités débranchées, Visualisation d'algorithmes

Abstract There are different ways to learn programming. The unplugged activities are usually distinguished from the connected programming activity. This paper proposes a step-by-step approach to switching from unplugged activity to programming using AlgoTouch, a programming tool by demonstration.

Keywords Algorithms, Programming for novices, Programming by demonstration, Unplugged activities, Algorithms visualization.

1 Introduction

L'enseignement de la programmation doit désormais être dispensé depuis l'école primaire. L'approche la plus communément admise pour cet apprentissage consiste à développer d'abord des activités débranchées [3]. Elles consistent à réaliser des algorithmes sans utiliser de machines. L'idée étant de distinguer le concept d'algorithme, permettant de résoudre un problème, de celui de programme, exécuté par une machine, associé à un langage de programmation. Ces concepts (algorithme, machine, langage, information) étant les concepts de base pour l'enseignement de la discipline informatique d'après la SIF [6].

Cependant le passage à l'activité de programmation est difficile [3]. De nombreux obstacles d'ordre techniques sont à surmonter : l'apprentissage d'un langage de programmation, dont la syntaxe est parfois contraignante, la pratique d'un environnement de développement pour l'écriture du code, la compilation, l'exécution et le débogage. Pour surmonter ces difficultés des environnements particuliers ont été développés pour les débutants dont le plus connu est Scratch [4]. De plus, de nombreux systèmes de visualisation ont été proposés pour animer des algorithmes (voir [7] pour une étude bibliographique récente).

3. http ://csunplugged.org
4. http ://scratch.mit.edu

Mais, comme le souligne Boisvert [1], l'écriture d'un programme suit un processus qui est rarement décrit dans les manuels ou dans les cours. En effet, en général, le principe d'un algorithme est donné, puis son fonctionnement est illustré et enfin le code est donné directement. Par contre, l'expérience montre que des apprenants ont souvent le syndrome de la page blanche lorsqu'il s'agit d'écrire le programme dont on vient de décrire le principe de l'algorithme. Comme le note Hundhausen [4], même avec des environnements spécifiques, "les novices ont des difficultés à construire des boucles et à référencer correctement des éléments de tableau dans ces boucles avec l'usage d'index".

Il y a donc un vide à combler entre la réalisation d'une activité débranchée et l'écriture du programme associé. Dans cet article, nous proposons de montrer comment passer progressivement d'une activité débranchée à la réalisation d'un programme en utilisant un outil de programmation par démonstration appelé AlgoTouch. A partir d'une activité débranchée, nous montrons comment transformer progressivement les objets et actions de cette activité en objets numériques et opérations de la machine. Dans un premier temps, l'apprenant va réaliser l'algorithme sur lequel il travaillait avec les éléments de la programmation (variables, opérateurs, etc.) disponibles avec AlgoTouch. Dans un second temps, ayant constaté que certaines opérations se répètent, il va commencer à enregistrer ces opérations pour pouvoir les répéter automatiquement. AlgoTouch se chargera de créer les parties du programme associé. Ainsi, un algorithme complet pourra être transformé progressivement en programme.

Cet article présente d'abord brièvement le logiciel AlgoTouch. Puis, on expose comment passer de l'activité débranchée à un programme en illustrant par un exemple complet. Enfin, en conclusion, nous présenterons quelques perspectives.

2 Le logiciel AlgoTouch

AlgoTouch utilise la métaphore du tableau noir, c'est-à-dire la manipulation directe sur l'écran des objets de la programmation (variables, tableaux, indices). Par des gestes simples, l'utilisateur peut déplacer ces éléments, glisser-déposer des valeurs dans des cases mémoire (variables ou tableaux), augmenter les valeurs d'index, lancer des comparaisons, effectuer les opérations de base (addition, soustraction, multiplication, division). De plus, il permet d'enregistrer une séquence d'actions, de rejouer la séquence enregistrée, et enfin d'achever la construction de l'algorithme (cas de sortie). En fait, lorsque le mode d'enregistrement est activé, un programme est créé et produit au fur et à mesure que l'on "joue" avec les éléments de l'algorithme. Le système gère également les instructions conditionnelles [2].

AlgoTouch est dédié à la conception d'algorithmes simples avec un seul niveau d'itération. Il permet aussi de créer des sous algorithmes (des tâches sous forme de macro) facilitant ainsi la création d'algorithmes non limités à une seule itération. Lors de la réalisation d'un algorithme comportant une simple boucle, le programmeur doit définir quatre parties : l'initialisation des variables, la répétition de la boucle, son corps et l'après boucle. Nous avons défini une

structure de boucle dans un pseudo langage inspiré par le langage Eiffel [5]. Cette structure est composée de 4 parties distinctes (figure 1).

```
From
  <Instructions>
Until
  <Conditions de sortie>
Loop
  <Instructions>
Terminate
  <Instructions>
End
```

La partie *From* permet de définir clairement les instructions nécessaires avant l'itération. La partie *Until* définit la ou les conditions de sortie de la boucle. Ces conditions sont placées en séquence sans opérateur. La première condition est évaluée, si elle est vraie, la boucle s'arrête. Sinon, la seconde condition est évaluée et ainsi de suite. La partie *Loop* constitue le coeur de d'itération. Enfin la partie *Terminate* permet de définir les instructions éventuelles à effectuer après l'itération.

Figure 1. Structure d'une boucle

Dans la méthode proposée, nous créons ces parties dans l'ordre inverse : le corps (partie *Loop*), la poursuite de la boucle (partie *Until*) et enfin l'initialisation (partie *From*) et la terminaison (partie *Terminate*). La raison est de définir progressivement l'algorithme du cas général jusqu'aux détails, en utilisant l'outil à chaque étape. L'idée principale derrière la méthode, est que l'on exécute les opérations de la séquence du corps avec des éléments réels de l'algorithme (variables, tableaux) sur un cas typique, puis on rejoue la séquence avec des données différentes (notamment pour traiter des instructions conditionnelles). On utilise l'outil pour indiquer qu'il faut placer une condition de sortie, et l'outil va générer les instructions correspondantes dans la partie *Until*. Pour terminer la construction de l'algorithme, on va définir comment initialiser les variables dans la partie *From*. Ensuite, on peut vérifier que l'algorithme fonctionne correctement en exécutant le programme complet. Dans certains algorithmes, il peut y avoir des opérations qui doivent être exécutées après la sortie de la boucle principale. On enregistre ces instructions qui sont codées dans la partie *Terminate*.

3 Transition du débranché vers la programmation

Dans de nombreuses activités débranchées, l'apprenant doit résoudre un problème par une série de manipulations et en utilisant des régles simples (peser des objets, déplacer des cartes, déplacer des jetons). Dans le cas de la résolution d'un problème par une machine, les "objets" disponibles sont des variables ou des tableaux et les opérations possibles sont limitées : affectation d'une variable, opérations simples (addition, soustraction, multiplication, division, reste de la division), comparaisons de valeurs pour prendre des décisions et répétition d'un ensemble d'opérations. La méthode que nous proposons consiste à transformer progressivement l'activité débranchée en ajoutant des règles permettant de remplacer les manipulations sur les objets de l'activité par des opérations sur les variables d'un algorithme. Pour illustrer notre propos, nous allons traiter un

exemple de tri. Nous allons trier, par ordre croissant, des cartes contenant des nombres entiers. Dans un premier temps nous allons décrire plusieurs activités débranchées successives, puis les premières activités branchées avec AlgoTouch et enfin l'automatisation du tri et la création du programme.

3.1 Activités débranchées

Tri de cartes : faces visibles. Nous disposons huit cartes, faces visibles, devant un élève. Il doit les ranger par ordre croissant. En fait, en général, l'élève effectue ce travail assez rapidement sans qu'on ait vraiment besoin de l'aider. Si on lui demande comment il a procédé, les explications sont assez floues, il peut difficilement décrire l'algorithme qu'il a utilisé.

Tri de cartes : faces cachées. Nous définissons maintenant de nouvelles régles. Nous disposons les cartes, faces cachées. Et on ne peut consulter que deux cartes à la fois, puis les cacher à nouveau. Cette fois, l'élève est un peu perturbé, il retourne les cartes, deux par deux, au hasard, en tentant de mémoriser leur contenu. Nous apportons une aide pour organiser les choses en indiquant un principe simple : comparer la première carte avec la deuxième, si la deuxème est plus petite, échanger. Puis comparer, la première carte avec la troisième, échanger si besoin. Recommencer jusqu'à comparer la première carte avec la dernière en échangeant si besoin. A la fin de ce passage, la première carte est la plus petite. On recommence maintenant en comparant la seconde carte avec les suivantes pour placer correctement le second minimum. Et on recommence ce processus pour placer chaque carte.

Les élèves comprennent vite le principe et effectuent les opérations pour trier les huit cartes. Ils se rendent compte que cela prend du temps (notion de *complexité*).

3.2 Activités branchées

Dans cette étape, nous allons passer dans le monde du numérique en utilisant AlgoTouch. Les cartes seront remplacées par des nombres stockés dans des cases de la mémoire (la notion de *variable* est introduite). Nous disposons donc maintenant de 8 variables `a, b, ..., h` contenant un entier.

Tri de variables par déplacement. Le but du jeu consiste à trier les variables de la plus petite à la plus grande, c'est à dire, placer les variables en les ordonnant par ordre croissant de la gauche vers la droite. Pour compliquer le jeu, on cache le contenu des variables (mode aveugle). Comment alors savoir si le contenu d'une variable est plus petit que celui d'une autre ? On montre alors qu'un ordinateur possède un *comparateur*. Avec AlgoTouch, il s'utilise comme une balance : on place sur chaque plateau le contenu à comparer. On sait donc si le contenu de la variable `a` est plus petit que celui de la variable `b`, ou plus grand, ou égal.

Disposant de ces nouvelles règles du jeu, les élèves comprennent qu'ils peuvent utiliser le même algorithme que celui défini pour le tri de cartes faces cachées. Ils procèdent au tri des variables et pour finir, l'enseignant rend visible à nouveau le contenu des variables pour vérifier que le résultat est correct.

Tri du contenu de variables. A l'issue de l'étape précédente, l'enseignant fait remarquer qu'on a bien trié les variables `a, b, ..., h` en les replaçant de gauche à droite, par exemple `f, a, h, d`, etc. Cependant, il serait souhaitable que ce soit le contenu des variables `a, b, ..., h` qui soit réorganisé par ordre croissant. La raison en est assez simple, du point de vue de l'ordinateur, on ne peut pas déplacer les variables puisque ce sont des cases à des emplacements (adresses) fixes en mémoire. Par contre, on peut modifier le contenu d'une variable.

On peut maintenant revoir l'algorithme de tri des variables en échangeant le contenu des variables sans les déplacer en utilisant une variable intermédiaire, notée `tmp` par exemple (ils viennent de découvrir le motif classique en programmation pour échanger deux variables). Après cela, les élèves réalisent aisément le tri des valeurs des variables `a, b,..., h` en reprenant l'algorithme précédent mais en échangeant les contenus de deux variables si besoin au lieu de les déplacer.

Tri manuel d'un tableau. Lors de l'étape précédente, on a montré qu'il était possible de réorganiser les données de 8 variables pour les classer par ordre croissant en utilisant les opérations de l'ordinateur : *l'affectation* et la *comparaison*. La question qui se pose alors est la suivante : comment faire pour que l'algorithme fonctionne indépendemment du nombre de variables ? En d'autres termes, comment pourrait-on appliquer le même principe de tri avec 20, 100 ou 1000 variables ?

Nous introduisons alors la notion de *tableau*. L'idée est la suivante : pourquoi ne pas associer un indice à un nom de variable. Les variables porteront ainsi le même nom et on pourra les différencier par leur indice. Ces variables forment ce qu'on appelle habituellement un tableau. Avec AlgoTouch, on va ainsi créer un tableau, nommé `t`, par exemple, de 8 cases. AlgoTouch ajoute une constante appelée `t.length` contenant la taille du tableau, soit la valeur 8.

On demande aux élèves d'indiquer ce qui change par rapport au tri des variables. En fait, a priori c'est pareil sauf que `a` est identifié par `t[0]`, `b` par `t[1]`, etc. Après avoir remarqué qu'on doit comparer `t[0]` à `t[1]`, puis à `t[2]` etc, on indique alors qu'il serait judicieux de remplacer l'indice par une variable (`j`) qui serait incrémentée pour repérer la variable suivante du tableau `t`.

Avec AlgoTouch, on introduit alors la notion de *variable d'index* associée à un tableau. On crée la variable d'index `j`. AlgoTouch identifie par une flèche l'indice du tableau repéré par `j`. De plus, AlgoTouch utilise un icône pour représenter `t[j]`, c'est une case comme une variable mais dont le contenu est calculé en fonction de la valeur de `j`. Lorsqu'on manipule cet icône, on manipule en fait `t[j]` (cf figure 2).

3.3 Automatisation

Dans cette partie, nous montrons que nous disposons maintenant de toutes les notions pour pouvoir automatiser l'algorithme de tri des données d'un tableau. Pour le problème qui nous occupe, on remarque qu'il est souhaitable de disposer d'un deuxième indice, par exemple `i`, pour mémoriser l'endroit où on va placer le prochain minimum. Dans notre exemple, on compare le contenu de `t[0]` avec `t[1]` puis `t[2]` etc. Pour généraliser, on va maintenant comparer `t[i]` successivement avec `t[j]` pour `j` allant de `i+1` au dernier indice du tableau.

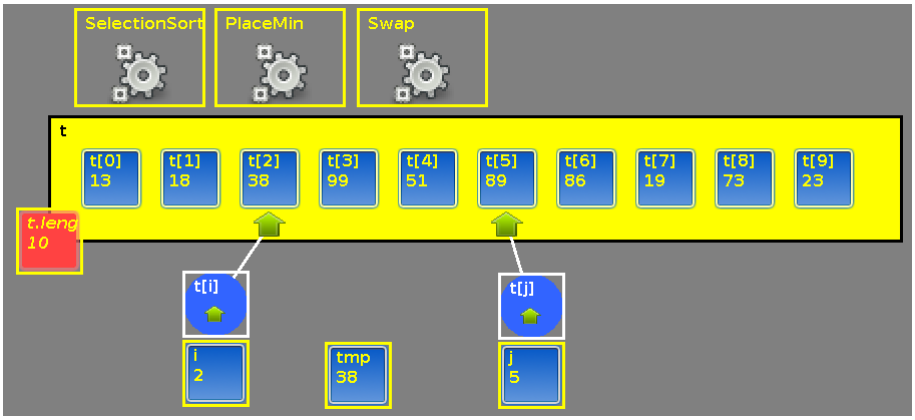

Figure 2. Tableau en cours de tri. Les données à gauche de l'indice `i` sont triées. On parcourt le tableau avec l'indice `j` pour placer la plus petite valeur à l'indice `i`.

Echange automatisé. Dans un premier temps, on constate qu'on compare toujours `t[i]` avec `t[j]`. Lorsque `t[j]` est inférieur, on doit échanger `t[i]` et `t[j]`. On peut donc automatiser cet échange.

Avec AlgoTouch, il suffit d'actionner le bouton "enregistrer", puis d'effectuer les actions nécessaires et enfin d'arrêter l'enregistrement. On vient de créer une *macro opération* que l'on peut renommer *Swap*. On peut alors ré-exécuter cette macro lorsqu'on souhaite échanger `t[i]` et `t[j]`.

On vient donc d'introduire une nouvelle notion, fondamentale en programmation, la notion de *sous-programme* : suite d'instructions identifiée par un nom que l'on peut ré-exécuter. Cette notion permet aussi d'introduire la notion de décomposition d'un programme complexe en un ensemble de programmes plus simples, eux-mêmes éventuellement décomposés en sous-programmes.

Placement du minimum automatisé. Pour illustrer concrètement la notion de décomposition d'un algorithme en sous algorithmes plus simples, nous reprenons l'exemple du tri du tableau de valeurs. Nous avons vu qu'à une certaine

étape, nous devons placer dans `t[i]` la plus petite valeur des données allant de l'indice `i` au dernier indice du tableau. Nous allons créer une macro appelée *PlaceMin* pour représenter cet algorithme.

Lorsqu'on exécute manuellement les différentes opérations de l'algorithme *PlaceMin*, on s'aperçoit que l'on répète toujours la même séquence d'instructions : (1) comparer `t[i]` avec `t[j]`, (2) si `t[j]` est inférieur à `t[i]`, échanger `t[i]` et `t[j]`, (3) incrémenter `j`.

On vient d'identifier la notion *d'itération ou de boucle*. Dans notre exemple, la séquence répétitive sera enregistrée dans le bloc *Loop*. Une fois cette séquence enregistrée, AlgoTouch offre la possibilité de l'exécuter à nouveau. On peut donc s'apercevoir que le corps de l'algorithme fonctionne correctement : à chaque exécution, la valeur de `t[i]` est échangée avec `t[j]` si besoin et l'indice `j` est incrémenté. AlgoTouch identifie visuellement par une flèche l'indice du tableau repéré par `j` (cf figure 2). On voit donc la flèche se déplacer vers la droite à chaque exécution du corps de boucle jusqu'à ce que la flèche devienne rouge et déborde du tableau.

On montre alors qu'il faut quitter la boucle et enregistrer la condition de sortie associée (*Until*). Un comparateur apparait, il faut donc comparer l'indice `j` avec la valeur de la taille du tableau (`t.length`). La condition de sortie est que `j` est supérieur ou égal à `t.length`. Il reste maintenant à enregistrer la séquence d'instructions à exécuter avant la première itération (*From*), soit `j = i+1`. Enfin, on doit éventuellement enregistrer la partie terminale (*Terminate*). Dans le cas présent, elle est vide. A l'issue de cette phase, nous avons construit les différentes séquences de la macro *PlaceMin*. Le code qui a été fabriqué automatiquement est illustré sur la figure 3.

```
Define Swap
From
Until
Loop
   tmp = t[j] ;
   t[j] = t[i] ;
   t[i] = tmp ;
Terminate
End
```

```
Define PlaceMin
From
  j = i + 1;
Until
  (j >= t.length)
Loop
  if (t[j] < t[i]){
    Swap
  }
  j = j + 1;
Terminate
End
```

```
Define SelectionSort
From
  i = 0 ;
Until
  (i>=t.length )
Loop
  PlaceMin
  i = i + 1;
Terminate
End
```

Figure 3. Programmes produits par AlgoTouch

Tri automatisé d'un tableau. A l'issue de cette étape, on sait placer la bonne valeur à l'indice `i` en appelant la macro *PlaceMin*. Dans un premier temps, nous allons jouer avec cette macro. D'abord, nous mélangeons les valeurs du tableau

`t` et on initialise `i` à `0`. Nous lançons l'exécution de la macro *PlaceMin*, la plus petite valeur se retrouve en première position. Incrémentons `i` et appelons à nouveau *PlaceMin*. La seconde valeur est correctement placée. Nous identifions la boucle comme étant la séquence suivante : (1) PlaceMin, (2) incrémenter `i`.

Nous allons donc créer le programme de tri (en fait la macro *SelectionSort*). Nous enregistrons d'abord le corps de la boucle vu précédemment (*Loop*). Puis exécutons cette séquence plusieurs fois pour vérifier que l'algorithme se déroule correctement. On arrête la boucle lorsque l'indice `i` déborde du tableau (*Until*). Pour la partie *From*, on initialise `i` à `0`. Pour la partie *Terminate*, elle est vide dans ce cas.

Le programme correspondant *SelectionSort* est donné dans la figure 3. Il utilise les différentes macros : *Swap* et *PlaceMin*. On constate que chacune d'elles est très simple et facile à comprendre. On illustre ainsi la notion vue précédemment qui consiste à décomposer un problème en sous-problèmes plus simples.

4 Conclusion et perspectives

Le logiciel AlgoTouch a été développé comme preuve de concepts. Il fonctionne sur plusieurs plateformes en utilisant la souris, mais l'usage d'un tableau interactif permet de manipuler directement les éléments d'un programme. Il facilite la création d'algorithmes simples sans avoir besoin de passer par un langage de programmation. Ainsi, il permet la transition d'une activité débranchée à la programmation (activité branchée). Par étapes successives, on remplace les objets et actions des activités débranchées, par des variables et des opérations de la programmation. Dans ce papier, nous avons montré comment traiter un algorithme de tri. Par ailleurs, nous avons aussi développé plusieurs scénarios de transitions d'activités débranchées vers des activités branchées : codage binaire, addition binaire, calcul d'une moyenne. Ils ont été mis au point et testés avec quelques élèves de collège en classe de troisième pendant quelques demi-journées. Pour ces exemples, AlgoTouch se révèle particulièrement bien adapté.

De nouvelles expérimentations doivent être réalisées afin d'avoir des retours d'enseignants, d'élèves et d'étudiants pour améliorer le système. Nous pensons en particulier que le logiciel permet de faire comprendre les mécanismes de la programmation à tout public en un séminaire d'une journée par exemple.

Références

1. Charles R. Boisvert. A visualisation tool for the programming process. *SIGCSE Bull.*, 41(3) :328–332, July 2009.
2. Patrice Frison. A teaching assistant for algorithm construction. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 9–14, New York, NY, USA, 2015. ACM.
3. Mark Guzdial. Programming environments for novices. *Computer science education research*, 2004 :127–154, 2004.

4. Christopher D Hundhausen, Sean F Farley, and Jonathan L Brown. Can direct manipulation lower the barriers to computer programming and promote transfer of training ? : An experimental study. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(3) :13, 2009.
5. Bertrand Meyer. *Touch of Class : learning to program well with objects and contracts*. Springer, 2009.
6. SIF. Enseigner l'informatique de la maternelle à la terminale. *1024, Bulletin de la Société informatique de France*, 9 :25–33, November 2016.
7. Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. *Trans. Comput. Educ.*, 13(4) :15 :1–15 :64, November 2013.