Transition didactique de l'activité débranchée à la programmation avec AlgoTouch

Patrice Frison, Moncef Daoud, Michel Adam

Patrice Frison[a,b], Moncef Daoud[b] & Michel Adam[b]

a. Laboratoire IRISA
b. Université de Bretagne Sud
patrice.frison@univ-ubs.fr, moncef.daoud@univ-ubs.fr,
michel.adam@univ-ubs.fr

Transition didactique de l'activité débranchée à la programmation avec AlgoTouch

Résumé

L'apprentissage de la programmation peut se faire de différentes manières. On distingue habituellement les activités débranchées de l'activité branchée de programmation. Cependant le passage à l'activité de programmation est difficile. De nombreux obstacles d'ordre technique sont à surmonter : l'apprentissage d'un langage de programmation, la pratique d'un environnement de développement, la compilation, l'exécution et le débogage. Ce papier propose une démarche pour passer progressivement de l'activité débranchée à la programmation en utilisant AlgoTouch, un outil de programmation par démonstration. Cet outil permet de manipuler directement les objets de la programmation (variables, tableaux, indices) pour réaliser un algorithme donné. De plus, il permet d'enregistrer un suite d'opérations et le programme est produit automatiquement. À 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.

Mots clés : algorithmes, programmation pour débutants, programmation par démonstration, activités débranchées, visualisation d'algorithmes

1 Introduction

L'enseignement de la programmation doit désormais être dispensé depuis l'école primaire. En France, depuis la réforme, la communauté scientifique s'intéresse à l'éducation de l'informatique pour tout cycle d'apprentissage (Baron & Drot-Delange, 2016). Il ne s'agit pas de proposer la programmation

et l'apprentissage d'un langage pour les débutants. L'objectif est de découvrir des concepts et des méthodes spécifiques à la science informatique. L'approche la plus communément admise pour cet apprentissage consiste à développer d'abord des activités débranchées[1]. 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 (voir Drot-Delange, 2013, pour une analyse de différentes expériences). Ces concepts (algorithme, machine, langage, information) sont les bases pour l'enseignement de la discipline informatique d'après la SIF (2016).

Cependant le passage à l'activité de programmation est difficile (Guzdial, 2004). De nombreux obstacles 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[2]. De plus, de nombreux systèmes de visualisation ont été proposés pour animer des algorithmes (voir Sorva, Karavirta & Malmi, 2013, pour une étude bibliographique récente).

Mais, comme le souligne Boisvert (2009), 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 montré 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 notent Hundhausen, Farley et Brown (2009), 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 développé par l'un des auteurs (Frison, 2015). À partir d'une activité débranchée, nous montrons comment transformer progressivement les objets et actions de cette activité en objets

1 <http://csunplugged.org>
2 <http://scratch.mit.edu>

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, nous exposerons 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 de la 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 (Frison, 2015).

AlgoTouch est dédié à la conception d'algorithmes manipulant uniquement des entiers ou des caractères. La structure de base d'un algorithme est une simple boucle. Lors de la construction d'un algorithme, le concepteur doit définir quatre parties : l'initialisation des variables, la répétition de la boucle, son corps et l'après-boucle. Il est possible de créer des sous-algorithmes définis sous forme de macro. Ils facilitent ainsi la création d'algorithmes non limités à une seule itération. La structure de boucle a été

définie dans un pseudo-langage inspiré par le langage Eiffel (Meyer, 2009). Cette structure est composée de 4 parties distinctes (figure 1).

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

Figure 1 : Structure d'une boucle.

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 cœur de l'itération. Elle est exécutée quand toutes les conditions de sortie sont fausses. Enfin la partie *Terminate* permet de définir les instructions éventuelles à effectuer après l'itération.

Dans la méthode proposée, l'utilisateur crée 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 : à chaque étape, l'outil sera utilisé. L'idée principale derrière la méthode est que l'utilisateur exécute les opérations de la séquence du corps avec des éléments réels de l'algorithme (variables, tableaux) sur un cas typique, puis il rejoue la séquence avec des données différentes (notamment pour traiter des instructions conditionnelles). Il utilise l'outil pour indiquer qu'il faut placer une condition de sortie, et les instructions correspondantes seront produites automatiquement dans la partie *Until*. Pour terminer la construction de l'algorithme, il va définir comment initialiser les variables dans la partie *From*. Ensuite, il peut vérifier que l'algorithme fonctionne correctement en exécutant le programme complet. Dans certains algorithmes, des opérations doivent être exécutées après la sortie de la boucle principale : il 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 règles 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. Un apprenant doit trier, par ordre croissant, des cartes contenant des nombres entiers, en présence d'un tuteur. Dans un premier temps, nous allons décrire plusieurs activités débranchées successives, puis les premières activités branchées avec AlgoTouch, enfin l'automatisation du tri et la création du programme.

3.1 Activités débranchées

Tri de cartes : faces visibles

Le tuteur dispose cinq cartes, faces visibles, devant un apprenant (*cf.* figure 2). Celui-ci doit les ranger par ordre croissant. En fait, en général, l'apprenant effectue ce travail assez rapidement sans vraiment besoin d'aide. Si le tuteur 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

Le tuteur définit maintenant de nouvelles règles. Il dispose les cartes, faces cachées, et il indique à l'apprenant qu'il ne peut consulter que deux cartes à la fois, puis les cacher à nouveau (*cf.* figure 3).

Figure 2 : Cartes à trier : faces visibles.

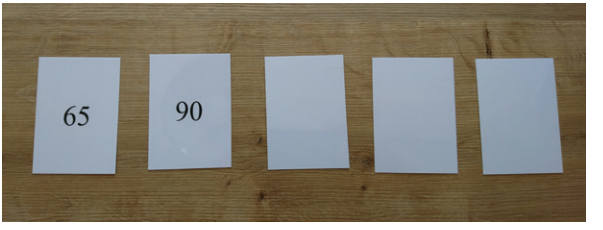

Figure 3 : Cartes à trier : faces cachées ; seules deux cartes peuvent être consultées à la fois.

Cette fois, l'apprenant est un peu perturbé, il retourne les cartes, deux par deux, au hasard, en tentant de mémoriser leur contenu.

Le tuteur apporte une aide pour organiser les choses en indiquant un principe simple : comparer la première carte avec la deuxième, si la deuxième 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. Les figures 4 et 5 illustrent la manipulation avec les cartes 1 et 4.

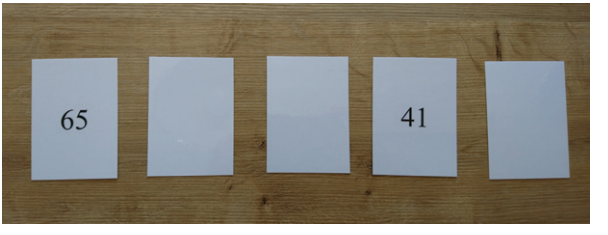

Figure 4 : Comparaison des cartes 1 et 4.

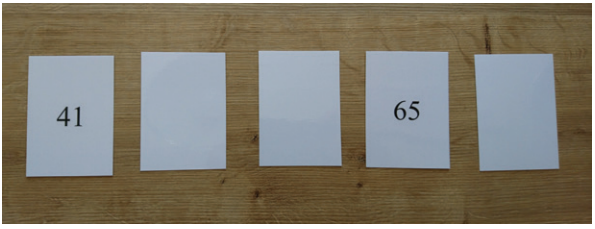

Figure 5 : Échange des cartes 1 et 4.

À la fin de ce passage, la première carte est la plus petite. Il recommence maintenant en comparant la seconde carte avec les suivantes pour placer correctement le second minimum. D'un point de vue pratique, le tuteur propose à l'apprenant de repérer avec l'index de sa main gauche, la position de la carte qui va contenir le minimum et avec l'index de la main droite, la position de la prochaine carte à comparer (*cf.* figure 6). En fait, cette manière de faire s'avère indispensable si le nombre de cartes est plus élevé ! Cette pratique sera mise à profit dans l'activité branchée.

Figure 6 : Itération du processus à partir de la carte 2. Ici, comparaison de la carte 2 avec la carte 4. L'apprenant repère la carte 2 avec l'index de la main gauche et la carte 4 avec l'index de la main droite.

Il recommence ce processus pour placer chaque carte des minimums successifs. L'apprenant comprend vite le principe et effectue les opérations pour trier les cinq cartes (*cf.* figure 7). Il se rend compte que cela prend du temps (notion de *complexité*). À ce stade, le tuteur peut en profiter pour lui demander combien de comparaisons ont été effectuées. La réponse est 4+3+2+1 soit 10 pour cet exemple avec 5 cartes. Le calcul du cas général avec n cartes peut être proposé.

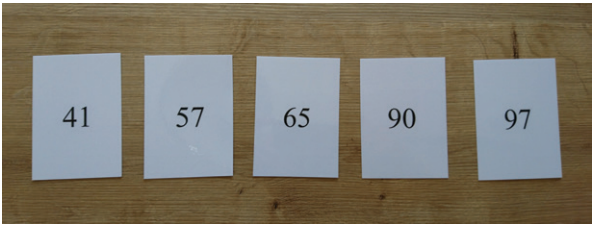

Figure 7 : Résultat du tri des cartes.

3.2 Activités branchées

Dans cette étape, il s'agit de 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. Le tuteur crée donc maintenant 5 variables (`a,b,c,d,e`) contenant chacune un entier, les mêmes valeurs que dans le cas débranché. Le résultat est affiché sur la figure 8.

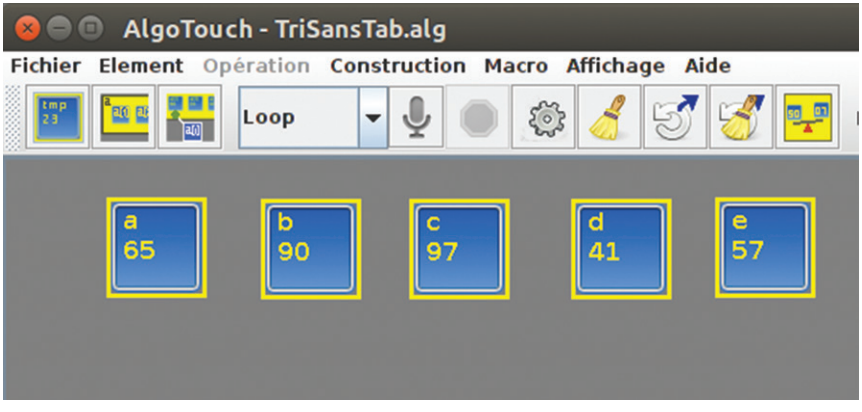

Figure 8 : Données à trier dans des variables simples.

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, le tuteur cache le contenu des variables (mode aveugle disponible dans AlgoTouch), comme dans l'activité débranchée.

Comment savoir si le contenu d'une variable est plus petit que celui d'une autre ? Le tuteur montre alors qu'un ordinateur possède un *comparateur* (*cf.* figure 9).

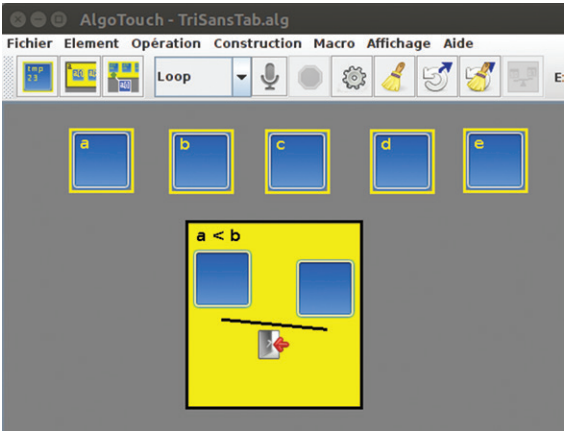

Figure 9 : Comparaison de variables en mode aveugle. La valeur de a, placée sur le plateau de gauche, est plus petite que la valeur de b, placée sur le plateau de droite.

Avec AlgoTouch, il s'utilise comme une balance : l'utilisateur place sur chaque plateau le contenu à comparer.
AlgoTouch indique 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, l'apprenant comprend qu'il peut utiliser le même algorithme que celui défini pour le tri de cartes faces cachées. Il procède au tri des variables et pour finir, le tuteur rend visible à nouveau le contenu des variables pour vérifier que le résultat est correct (voir figure 10).

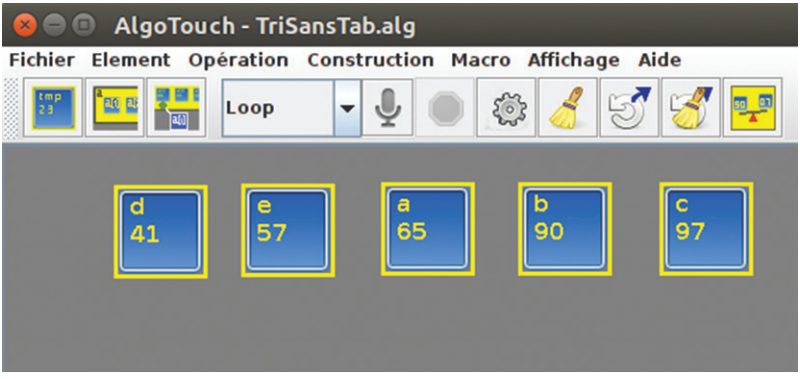

Figure 10 : Les variables sont rangées par ordre de valeur croissante. Remarquer que les variables ont été déplacées visuellement en fonction de leur contenu.

Tri du contenu de variables

À l'issue de l'étape précédente, le tuteur fait remarquer que les variables (`a,b,c,d,e`) ont bien été triées en les replaçant de gauche à droite, par exemple (`d,e,a,b,c`) comme illustré sur la figure 10. Cependant, il serait souhaitable que ce soit le contenu des variables (`a,b,c,d,e`) qui soit réorganisé par ordre croissant. La raison en est assez simple, du point de vue de l'ordinateur, les variables ne peuvent pas être déplacées puisque ce sont des cases à des emplacements (adresses) fixes en mémoire. Par contre, il est possible de modifier le contenu d'une variable.

Le tuteur propose maintenant de 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. En fait, ils viennent de découvrir le motif classique, en programmation, pour échanger deux variables. Après cela, l'apprenant réalise aisément le tri des valeurs des variables (`a,b,c,d,e`) en reprenant l'algorithme précédent mais en échangeant les contenus de deux variables si besoin au lieu de les déplacer. La figure 11 montre le résultat de la manipulation.

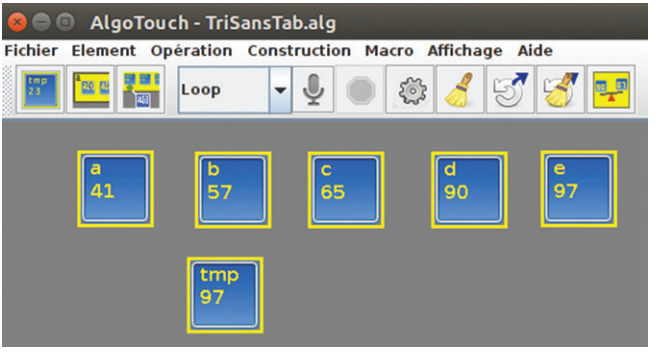

Figure 11 : Les variables sont triées par ordre de valeur croissante.

Tri manuel d'un tableau

Lors de l'étape précédente, le tuteur a montré qu'il était possible de réorganiser les données de 5 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 indépendamment du nombre de variables ? En d'autres termes, comment appliquer le même principe de tri avec 20, 100 ou 1000 variables ?

Le tuteur introduit 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 leur indice permettra de les différencier. Ces variables forment ce qu'on appelle habituellement un tableau. Avec AlgoTouch, le tuteur va alors créer un tableau, nommé `t`, par exemple, de 10 cases. AlgoTouch ajoute une constante appelée `t.length` contenant la taille du tableau, soit la valeur 10.

Le tuteur demande à l'apprenant ce qui change par rapport au tri des variables. L'apprenant indique, qu'*a priori*, c'est pareil sauf que `a` est identifié par `t[0]`, `b` par `t[1]`, etc. Il précise que cette fois, il doit comparer `t[0]` à `t[1]`, puis à `t[2]`, etc. Le tuteur 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`.

AlgoTouch définit la notion de *variable d'index* associée à un tableau. Le tuteur crée la variable d'index `j` associée au tableau `t`. AlgoTouch identifie par une flèche l'indice du tableau repéré par `j`. De plus, AlgoTouch crée une 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`. Lorsque l'utilisateur manipule cette icône, il manipule en fait `t[j]` (*cf.* figure 12).

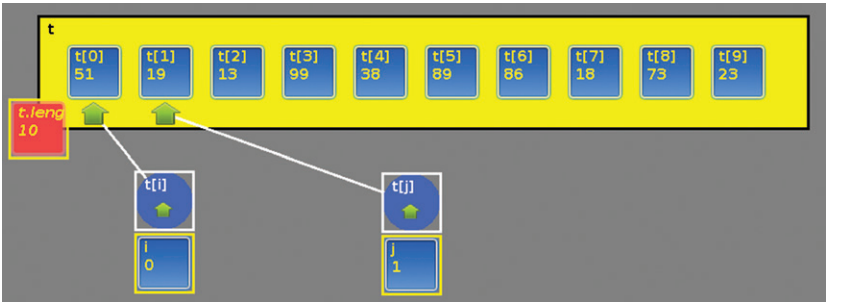

Figure 12 : Tableau de 10 valeurs à trier. À la création du tableau `t`, la constante `t.length` est définie automatiquement. Deux variables d'index `i` et `j` permettent d'accéder respectivement à `t[i]` et `t[j]` grâce aux icônes placées au-dessus. De plus, des flèches indiquent les éléments correspondants.

3.3 Automatisation

Dans cette partie, l'apprenant dispose maintenant de toutes les notions pour pouvoir automatiser l'algorithme de tri des données d'un tableau. Pour avancer, le tuteur remarque qu'il est souhaitable de disposer d'un deuxième indice, par exemple `i`, pour mémoriser l'endroit où va être placé le prochain minimum. Le tuteur indique que les variables d'index jouent le même rôle que les index (les doigts) utilisés dans la version débranchée pour repérer les éléments à comparer.

Dans cet exemple, le contenu de `t[0]` est comparé avec `t[1]` puis `t[2]` etc. Pour généraliser, il faut maintenant comparer `t[i]` successivement avec `t[j]` pour `j` allant de `i+1` au dernier indice du tableau.

Échange automatisé

Dans un premier temps, le tuteur montre qu'il compare toujours `t[i]` avec `t[j]`. Lorsque `t[j]` est inférieur, il doit échanger `t[i]` et `t[j]`. Cet échange peut donc être automatisé.

Avec AlgoTouch, il suffit d'actionner le bouton « enregistrer », puis d'effectuer les actions nécessaires et enfin d'arrêter l'enregistrement. Le tuteur crée alors une *macro opération* qu'il renomme *Swap*. Il peut alors ré-exécuter cette macro lorsqu'il souhaite échanger `t[i]` et `t[j]`.

Le tuteur vient donc d'introduire une nouvelle notion, fondamentale en programmation, la notion de *sous- programme* : suite d'instructions identifiée par un nom que l'utilisateur 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, le tuteur reprend l'exemple du tri du tableau de valeurs. Il rappelle qu'à une certaine étape, il faut placer dans `t[i]` la plus petite valeur des données allant de l'indice `i` au dernier indice du tableau. Il va créer une macro appelée *PlaceMin* pour représenter cet algorithme.

Lorsqu'il exécute manuellement les différentes opérations de l'algorithme *PlaceMin*, il s'aperçoit qu'il 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`.

Le tuteur vient d'identifier la notion *d'itération ou de boucle*. Dans cet exemple, la séquence répétitive sera enregistrée dans le bloc *Loop*. Une fois cette séquence enregistrée, AlgoTouch offre la possibilité de l'exécuter à nouveau. Le tuteur 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 13). Il 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.

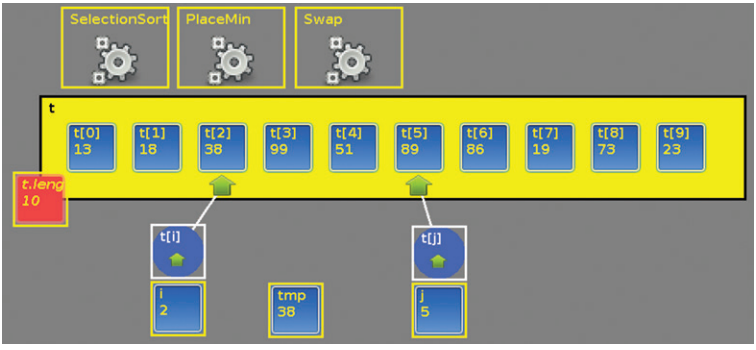

Figure 13 : Tableau en cours de tri. Les données à gauche de l'indice `i` sont triées. Le tableau est parcouru (indice `j`) pour placer la plus petite valeur en `t[i]`. La figure montre également les icônes des trois macros utilisées pour ce tri. Ces macros sont exécutables indépendamment par un double clic.

```
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 Selection-
Sort
From
  i = 0;
Until
  (i>=t.length)
Loop
  PlaceMin
  i = i + 1;
Terminate
End
```

Figure 14 : Programmes produits par AlgoTouch. Le premier, *Swap*, échange les contenus de `t[i]` et `t[j]`. À noter que puisque la clause *Until* est vide, le corps de la boucle ne s'exécute qu'une seule fois. Dans une version ultérieure d'AlgoTouch, la structure de la boucle sera remplacée par le corps du programme. Le second, *PlaceMin*, place en `t[i]` la valeur minimum de la suite. Le dernier, *SelectionSort*, trie tout le tableau en utilisant l'algorithme de sélection du minimum à chaque étape.

Le tuteur montre alors qu'il faut quitter la boucle et enregistrer la condition de sortie associée (*Until*). Un comparateur apparaît, 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, il doit éventuellement enregistrer la partie terminale (*Terminate*). Dans le cas présent, elle est vide. À l'issue de cette phase, le tuteur a construit les différentes séquences de la macro *PlaceMin*. Le code qui a été fabriqué automatiquement est illustré sur la figure 14.

Tri automatisé d'un tableau

À l'issue de cette étape, l'apprenant sait placer la bonne valeur à l'indice `i` en appelant la macro *PlaceMin*. Dans un premier temps, le tuteur propose de jouer avec cette macro. D'abord, il mélange les valeurs du tableau `t` et il initialise `i` à `0`. Il lance l'exécution de la macro *PlaceMin*, la plus petite valeur se retrouve en première position. Il incrémente `i` et appelle à nouveau *PlaceMin*. La seconde valeur est correctement placée. Le tuteur identifie la boucle comme étant la séquence suivante : (1) *PlaceMin*, (2) incrémenter `i`.

Le tuteur va alors créer le programme de tri (en fait la macro *SelectionSort*). Il enregistre d'abord le corps de la boucle vu précédemment (*Loop*). Puis exécute cette séquence plusieurs fois pour vérifier que l'algorithme se déroule correctement. Il arrête la boucle lorsque l'indice `i` déborde du tableau (*Until*). Pour la partie *From*, il initialise `i` à `0`. Pour la partie *Terminate*, elle est vide dans ce cas.

Le programme correspondant *SelectionSort* est donné dans la figure 14. Il utilise les différentes macros : *Swap* et *PlaceMin*. Le tuteur fait remarquer que chacune d'elles est très simple et facile à comprendre. Ceci 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, les objets et actions des activités débranchées, sont remplacées 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, etc. Ils ont été mis au point et testés avec quelques élèves de collège en classe de troisième pendant deux demi-journées. Pour ces exemples, AlgoTouch se révèle particulièrement bien adapté. Les élèves ont maîtrisé très rapidement l'outil. Ils ont vite compris comment réaliser des affectations, des opérations, des comparaisons. Ils ont aussi pris l'habitude d'utiliser des index sur des algorithmes opérant sur des tableaux. Enfin, ils ont pu apprendre progressivement les instructions du langage d'AlgoTouch puisque le système les produit automatiquement au fur à mesure des actions effectuées. À titre d'exemple, à la fin de l'expérience, ils ont réussi avec succès à réaliser un algorithme simple directement dans le langage de programmation.

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

Baron, G.-L., & Drot-Delange, B. (2016, novembre). L'éducation à l'informatique à l'école primaire. *1014, Bulletin de la Société informatique de France*, *8*, 73–79. Consulté sur <https://hal.archives-ouvertes.fr/hal-01403598>.

Boisvert, C. R. (2009). A visualisation tool for the programming process. In *Proceedings of the annual acm sigcse conference on innovation and technology in computer science education* (pp. 328–332). New York, NY, USA : ACM. Consulté sur <http://doi.acm.org/10.1145/1562877.1562976> doi : 10.1145/1562877.1562976.

Drot-Delange, B. (2013, août). Enseigner l'informatique débranchée : analyse didactique d'activités. In *AREF* (p. 1–13). France. Consulté sur <https://archivesic.ccsd.cnrs.fr/sic_00955208> (ACTI Axe1).

Frison, P. (2015). A teaching assistant for algorithm construction. In *Proceedings of the 2015 ACM conference on innovation and technology in computer science education* (pp. 9–14). New York, NY, USA : ACM. Consulté sur <http://doi.acm.org/10.1145/2729094.2742588> doi : 10.1145/2729094.2742588.

Guzdial, M. (2004). Programming environments for novices. *Computer science education research*, *2004*, 127–154.

Hundhausen, C. D., Farley, S. F. & Brown, J. L. (2009). 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.

Meyer, B. (2009). *Touch of class : learning to program well with objects and contracts*. Springer.

SIF. (2016, novembre). Enseigner l'informatique de la maternelle à la terminale. *1024, Bulletin de la Société informatique de France*, *9*, 25–33.

Sorva, J., Karavirta, V. & Malmi, L. (2013, novembre). A review of generic program visualization systems for introductory programming education. *Trans. Comput. Educ.*, *13*(4), 15 :1–15 :64. Consulté sur <http://doi.acm.org/10.1145/2490822> doi : 10.1145/2490822.