Human Processor!

Christian Blanvillain
UER-MI, HEP Vaud
christian.blanvillain@hepl.ch

Résumé. Nous avons conçu une formation complète de 20h pour découvrir l'algorithmie et pour développer les stratégies cognitives des élèves. Un support de cours accompagne un dispositif didactique débranché en bois, utilisé avec des élèves de 10 à 30 ans, qui permet de résoudre des problèmes algorithmiques. Après avoir résolu en petits groupes le problème donné, des débats réflexifs sont organisés autour des concepts abordés et sur les stratégies mobilisées par les élèves pour trouver les solutions. En sus de l'apprentissage de l'algorithmie, les élèves vont ainsi découvrir peu à peu des stratégies cognitives susceptibles de les aider durant les différentes phases de leur activité.

Mots-clés. Dispositif débranché, stratégies cognitives, intelligence.

1 Introduction

En classe, lors de nos séances d'enseignement de la programmation aux élèves débutants, nous avons pu observer chez la plupart des élèves en difficulté une tendance à éviter la phase de raisonnement qui conduit aux apprentissages. Ces élèves ont tendance à émettre leurs hypothèses "au hasard", appliquant la stratégie : je déplace une ligne de code sans réfléchir, j'exécute et je recommence jusqu'à ce que j'ai de la chance et que ça marche. Je "gagne", pas parce que je réfléchis bien, mais parce que je suis chanceux. Et surtout, ils ne semblent pas vouloir chercher à comprendre lorsque "ça marche". Ça fonctionne ? Génial, j'ai de la chance, passons vite au suivant ou sortons le smartphone pour faire autre chose. Ça ne fonctionne pas au bout de quelques minutes ? "Monsieur, je ne sais pas faire...". C'est un peu comme si le côté ludique de l'environnement masquait aux élèves le fait que derrière le jeu, pour accéder à la beauté de la pensée informatique, il fallait faire un effort de réflexion qui était considéré, par eux, comme anormal. Nous avons cru voir que, du moment où il faut commencer à réfléchir, les élèves s'arrêtent de travailler. Comme s'ils avaient peur d'apprendre ? Comme si le fait que l'activité présentée soit sous la forme d'un jeu, était un frein au processus de réflexion ? Comme si l'association effort intellectuel et jeu était incompatible…

Nous avons ainsi cherché à développer un artefact didactique qui éloigne le côté ludique des environnements numériques et qui introduit un effort de réflexion pour savoir si une solution proposée est juste ou pas, de manière à susciter une approche réfléchie et mobiliser un début de pensée informatique. Nous avons ainsi créé un langage de programmation par assemblage de blocs qui ne s'exécute pas, mais qui doit être interprété par un processeur humain (l'élève) pour savoir si, oui ou non, le code écrit fait bien ce qu'il était supposé faire. D'où le nom du dispositif : "*Human Processor!*".

2 Contexte

En nous positionnant de la sorte, un peu à contre-courant des outils modernes d'enseignement de la programmation, nous avons créé un environnement didactique de programmation débranché basé sur le langage machine, en nous inspirant du jeu *Human Resource Machine*. Ce jeu, développé en 2015 par Kyle Gabler, Allan Blomquist et Kyle Gray de la société Tomorrow Corporation[1], est lui-même inspiré de l'environnement didactique *Little Man Computer* (Andrew Eliasz, 2016) créé en 1965 par le Dr. Stuart Madnick, mais en beaucoup plus simple du fait qu'il propose un modèle mental explicite pour que l'élève puisse visualiser les déplacements de données entre l'entrée, la mémoire et la sortie, en passant par l'accumulateur (un petit bonhomme qui déplace physiquement les données avec ses mains à l'écran). Ces deux micro-mondes utilisent un jeu d'instructions que nous avons simplifié dans *hp!*. Le tableau 1 compare les jeux d'instructions des trois dispositifs, ainsi que le nombre d'emplacements mémoire disponibles.

[1] Tomorrow Corporation. (2015). Human Ressource Machine. En ligne : <https://tomorrowcorporation.com/about>

Notre dispositif ne comporte finalement que deux types d'instructions : les déplacements d'information et les sauts dans la séquence d'instructions. Chacun décliné en deux versions : lire et écrire pour les déplacements d'informations et sauts conditionnels ou inconditionnels. Dans le jeu *Human Ressource Machine* ces mêmes instructions sont représentées par des couleurs différentes avec une syntaxe et une logique de fonctionnement également différente. La Figure 1 présente, pour un même problème, les différences entre *hp!* et *Human Ressource Machine*. Vous pouvez comparer ligne à ligne chacune des instructions. Dans l'interface du jeu, les premières lignes de code utilisent des instructions de couleurs différentes qui ont une action similaire : déplacer une information d'un endroit à un autre. C'est ce qui est explicité dans la version débranchée : les quatre instructions sont similaires et illustrent de la même manière le déplacement des données de l'entrée (représentée par un point d'interrogation) vers l'accumulateur (représenté par un rond – un jeton transparent joue le rôle du petit bonhomme dans le jeu, c'est l'accumulateur qui se déplace d'une instruction à l'autre en transportant les informations, et qui est capable d'exécuter des additions ou des soustractions). Nous travaillons uniquement avec des données de type entiers signés. Le dispositif est livré avec des pièces présentant les entrées sorties, le contenu de l'accumulateur et l'historique de l'évolution des valeurs des cases mémoire. Le code conçu à l'aide de ce dispositif est interprété par l'élève qui peut vérifier l'exactitude de son algorithme en le déboguant à la main. Nous avons adapté les exercices proposés en conséquence, de manière à couvrir un semestre de formation pour débutants (17 exercices), sans aborder les défis plus difficiles proposés par ces deux autres environnements qui nous ont inspiré. Les plans pour découper les pièces du jeu, ainsi que des instructions de réalisation, sont disponibles sur le site du FabLab[2] de la HEP. Le dispositif est partagé sous licence *creative commons* avec le manuel pour l'enseignant contenant la liste des problèmes ainsi que la liste des stratégies cognitives.

Tableau 1. Comparaisons du nombre d'instructions

Little Man Computer (1965) **11 instructions + 100 cases mémoire**	**Human Ressource Machine (2015)** **11 instructions + 25 cases mémoire**	**Human Processor! (2019)** **7 instructions + 3 cases mémoire**
add	add	read and add
substract	sub	read and sub
load	copy from	read
store	copy to	write
branch	jump	jump
branch if zero	jump if zero	jump if zero
branch if positive	jump if negative	jump if negative
input	inbox	
output	outbox	
halt	bump+	
data	bump-	

3 Stratégies cognitives

Il n'y a que deux concepts à comprendre pour savoir programmer en *hp!* : déplacer une valeur (read, read and add, read and sub, write) ou sauter à un autre endroit du code (jump, jump if zero, jump if negative), nous écartons ainsi toute la complexité d'apprentissage d'un langage de programmation et pouvons très vite nous concentrer sur l'essentiel : c'est-à-dire aider les élèves en difficulté à développer une pensée algorithmique, tout en restant dans un environnement complet au sens de Turing. Cette aide se fait au travers d'apport de stratégies cognitives qui se traduisent concrètement par une alternance de séances pratiques de résolution de problèmes en petits groupes, avec des séances réflexives de partage avec toute la classe durant lesquelles les élèves peuvent parler des stratégies utilisées. C'est dans ces moments d'échange que nous aidons les élèves à expliciter leurs stratégies cognitives et que nous en proposons de nouvelles.

La durée prévue de la formation est de 20h (une heure par semaine sur un semestre). Le dispositif *hp!* a été testé durant un semestre dans trois classes : 6 élèves de 10 à 11 ans, 11 élèves de 11 à 12 ans, 15 élèves apprentis informaticiens de 20 à 30 ans. Les jeunes élèves étudient dans l'École Active de Malagnou à Genève et font de l'informatique une heure par semaine uniquement. Les apprentis informaticiens sont en formation à l'École Supérieure d'Informatique de Gestion de Genève <et suivent un cours facultatif intitulé "apprendre à apprendre", basé sur le dispositif *hp!*, sur leur temps de pause à midi. Ils apprennent l'informatique à plein temps le reste de la semaine. Notre établissement envisage de rendre ce cours obligatoire dès l'année prochaine et de l'inclure dans le cours d'initiation à l'algorithmie et la programmation.

[2] FabLab de la HEP Vaud : <https://fablab-hepl.ch/human-processor/> [CC-BY-NC-SA].

Étonnamment les jeunes élèves ont réussi à comprendre les exercices initiaux (tutoriel) plus rapidement que les apprentis informaticiens. Par la suite, les informaticiens ont rattrapé leur retard et ont dépassé les jeunes élèves vers la fin de la formation.

Le dispositif est accompagné d'un support de cours. Ce support comprend un cahier d'exercices et un manuel à l'intention de l'enseignant avec le tutoriel du jeu, des exercices corrigés avec leur version équivalente en Python et des suggestions d'activités réflexives complémentaires pour développer la pensée informatique et l'intelligence algorithmique des élèves. Enfin, une liste de stratégies cognitives est fournie à l'enseignant qui devra les faire redécouvrir par les élèves ou bien les introduire lui-même en fonction des obstacles rencontrés.

Notre micro-monde sert d'incubateur à stratégies, dans le sens où c'est en s'observant que les élèves sont supposés redécouvrir par eux-mêmes quelques-unes des stratégies cognitives. Les stratégies découvertes par les élèves ou introduites par l'enseignant au fil des exercices, couvrent 4 moments clés :

- Avant : stratégies pour apprendre à se concentrer et se mettre en condition de travail.
- Pendant : stratégies pour apprendre à résoudre des problèmes.
- En cas de blocage : stratégies pour apprendre à être créatif.
- Après : stratégies pour apprendre à apprendre.

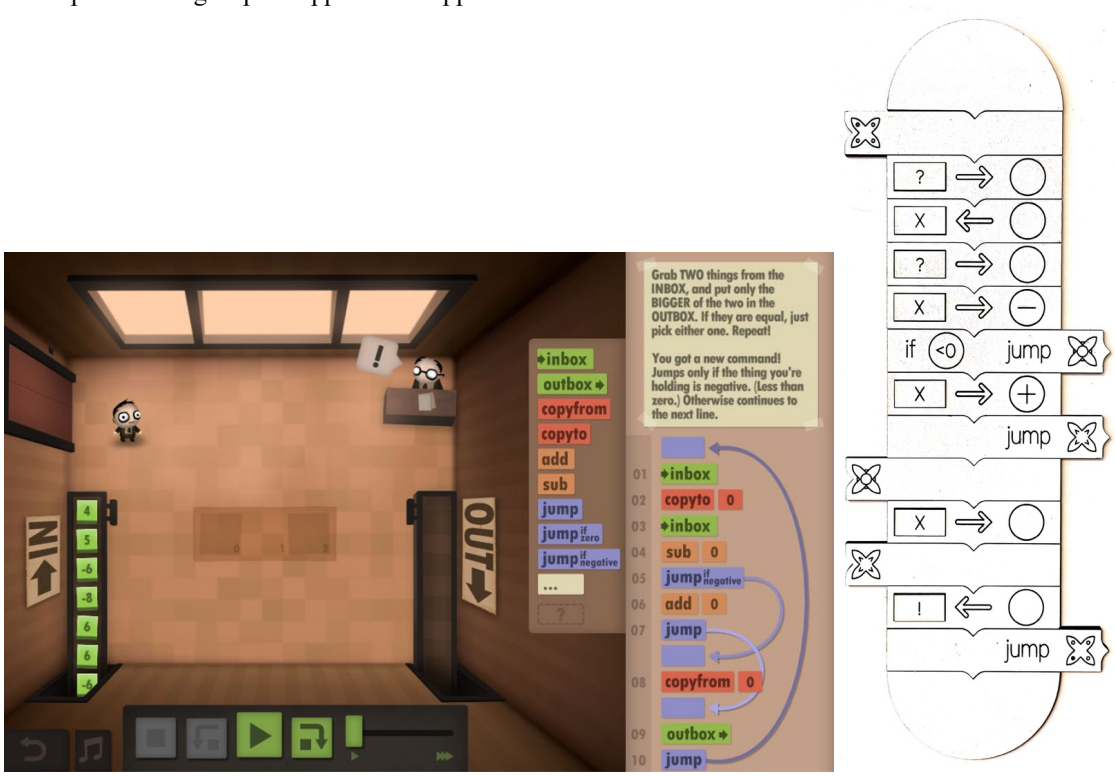

Figure 1. Interface Human Ressource Machine .vs. Human Processor!

4 Résultats et discussion

Chez les jeunes élèves, les différences de niveaux sont un frein à la motivation. Ainsi, bien que toute la classe ait pu apprendre les concepts fondamentaux de la programmation et développer un début de pensée algorithmique, tout le monde n'a pas pu résoudre l'ensemble des problèmes proposés. L'enseignement des stratégies cognitives n'a pas bien fonctionné. Susciter la métacognition chez les enfants de 10 à 12 ans n'est pas simple. Nous avons eu du mal à les motiver durant les séances de résolution de problème, ce qui fait que les moments de mise en commun réflexifs étaient assez courts et nous n'avons qu'exceptionnellement eu l'opportunité de parler des stratégies cognitives. Nos discussions étaient essentiellement centrées sur la concentration et la motivation. Nous avons mis en place une méthode pour indiquer les états du groupe : piste verte "ça roule", piste bleu "ça bosse", piste rouge "c'est dur", piste noire "on décroche"... La consigne était de consulter les stratégies cognitives lorsque l'on se retrouvait en piste rouge, mais nous n'avons pu observer aucun groupe d'élèves parvenir à se débloquer en appliquant la consigne.

Cependant, chez les apprentis informaticiens l'enseignement des stratégies cognitives a porté ses fruits. Dans les témoignages des élèves, dont des extraits ont été résumés dans le tableau 2 ci-dessous, on peut constater que des apprentissages ont bien été réalisés et que même des transferts dans d'autres matières ont eu lieu. Nous pensons que le dispositif atteint pleinement son but avec le public d'élèves plus âgés.

Tableau 2. Témoignages des apprentissages réalisés

Apprendre à réfléchir de manière différente. Avec *hp!* c'est plus ludique, plus visuel que de travailler sur écran. Ça développe une autre capacité. Dans notre tête ça se fait plus rapidement qu'avec du code. Notre cerveau il réfléchit et il assemble le truc.	L'exercice de la multiplication par 40 m'a bien fait comprendre comment décomposer un problème en petits problèmes. C'est ce qui m'a le plus aidé durant les épreuves.
Ce que j'ai le plus appris c'est à chaque fois que j'ai réussi à faire quelque chose, de faire le chemin inverse. De réfléchir comment je suis arrivé au résultat. Ça m'a permis de bien comprendre et de mieux structurer les choses : de savoir pourquoi on fait ça.	J'ai compris comment ça marche un ordinateur en soit : c'est pas magique, il y a rien de magique, en fait c'est logique. J'ai compris ça au tout début. Tout ce qui était binaire j'avais pas compris en cours. J'ai trouvé l'algorithme de la multiplication extraordinaire.
Prendre plus le temps de réfléchir avant de se mettre à écrire le code. Maintenant je prends le temps de lire plusieurs fois une épreuve avant de la commencer et de temporiser alors qu'avant c'était pas le cas, je me précipitais à coder et après je me rendais compte que j'avais perdu du temps. Maintenant je réfléchis à ce que je vais faire, ça crée une structure dans ma tête avant de commencer.	Les stratégies cognitives m'ont bien aidé durant les épreuves. Je les ai essayées. J'ai fait des pauses durant l'examen. Je suis sorti pour aller aux toilettes même si j'avais pas besoin, juste pour penser à autre chose et c'est comme ça que j'ai trouvé des idées et des solutions aux problèmes qu'on avait.

5 Conclusion

L'enseignement des concepts de l'architecture d'un ordinateur a réellement bien fonctionné, même auprès d'élèves particulièrement en difficulté. Tous les élèves ont fait des progrès, quel que soit leur âge, et nous avons pu observer qu'ils ont tous développé un début d'intelligence algorithmique. En ce sens, nous considérons que le dispositif didactique débranché a rempli son rôle : les élèves réussissent bien à se construire un modèle mental leur permettant de se représenter les déplacements d'informations au sein de l'architecture de l'ordinateur. Chez les élèves plus âgés, non seulement la prise de conscience et le développement de stratégies cognitives ont été observés, mais des transferts effectifs dans d'autres disciplines se sont produits.

Ces travaux font l'objet d'une thèse en didactique de l'informatique en codirection à l'Université de Patras en Grèce avec le Professeur Vassilis Komis et à la Haute École Pédagogique de Lausanne avec le Professeur Bernard Baumberger. A l'heure actuelle, nous cherchons à identifier les facteurs intra personnels et interpersonnels qui pourraient justifier les différences de niveau d'apprentissage observées entre les jeunes élèves et les élèves plus âgés, sur le plan du développement de la méta-cognition et du développement des stratégies cognitives. Nous aimerions, à terme, proposer aux enseignants des jeunes élèves des solutions pour leur permettre de faire émerger du groupe classe des stratégies cognitives concrètes facilitant la résolution de problèmes. Notre objectif final serait de fournir à tous les élèves le moyen de développer leurs intelligences triarchiques (Robert J. Sternberg, 1985), c'est-à-dire créative, logico-mathématique et pratique. En effet, relever les challenges proposés dans notre micro-monde, nécessite de pouvoir mobiliser ces trois intelligences et développer ce que nous appelons l'intelligence algorithmique.

Références

Eliasz, A. (2016). *Little Man computer Programming: for the perplexed from the ground up.* Great Britain: The Internet Technical Bookshop.

Sternberg, R. J. (1985). *Beyond I.Q: A triarchic theory of human intelligence.* New York: Cambridge University Press.