Micro-mondes et concurrence par passage de messages : vers une nouvelle façon d'aborder la programmation ?

Cédric Libert*[1] and Wim Vanhoof[1]

[1]Université de Namur, Faculté d'Informatique – Rue Grandgagnage, 21, 5000-Namur, Belgique

Résumé

Le premier cours de programmation est un enjeu important pour les étudiants qui suivent un cours d'informatique. En effet, la programmation est au cœur de la discipline et permet de développer des compétences essentielles : résolution de problèmes, abstraction et décomposition[1, p.41]. Toutefois, le taux d'échec dans ce cours, bien que pas alarmant[15], est élevé et contribuerait à décourager beaucoup d'étudiants d'entreprendre les études d'informatique[6]. Nous partons de ce constat pour définir une question de recherche : une première approche de la programmation qui se base sur la concurrence permet-elle d'améliorer les aptitudes algorithmiques des néo-programmeurs ? Cette question prend à rebours l'apprentissage de la programmation séquentielle de plupart des cours d'introduction.

Ce que l'on appelle concurrence est la possibilité pour plusieurs actions d'avoir lieu en même temps[10]. La programmation concurrente permet ainsi l'exécution simultanée de plusieurs instructions. Aborder la concurrence dans un cours d'introduction est considéré comme plus complexe qu'une approche séquentielle[13, 2, 3]. On peut toutefois admettre que les étudiants vivent dans un monde concurrent et que cette façon de programmer, si elle est enseignée correctement, pourrait leur sembler naturelle[14]. Par ailleurs, elle offre, pour certains problèmes, une meilleure performance ou une décomposition plus simple en entités indépendantes[13]. De plus, l'ACM souligne qu'il faudrait faire de la concurrence dès le début[1, p.44].

Aborder le concept de concurrence dans un premier cours de programmation n'est pas neuf. En effet, Feldman et Bachus montrent qu'il est possible de le faire avec les novices[4]. Stein considère, quant à elle, qu'il faut revoir la programmation : ne plus considérer qu'un programme est une fonction, mais une communauté d'entités indépendantes qui communiquent. Cela correspond beaucoup mieux aux systèmes de haut niveau[12] (système d'exploitation, internet). Nous souhaitons l'enseigner au moyen de micromondes enrichis progressivement.

Un micromonde est un environnement d'apprentissage construit pour améliorer les mécanismes cognitifs liés à ce que l'on souhaite enseigner[8, p.204]. Cela favorise la découverte et l'assimilation de nouveaux concepts. En programmation, un micromonde est généralement composé de:
- un langage de programmation, avec une syntaxe et une sémantique simple;
- une visualisation graphique. Dans notre cas, elle consiste à montrer les entités qui agissent en fonction des instructions transmises.

*Intervenant

Ces deux éléments sont intégrés dans une interface qui allie un éditeur de texte et une fenêtre de visualisation. On retrouve cela dans des langages tels que Logo[5] et ObjectKarel[11]. Plus récemment, l'outil Programmer's Learning Machine[9] permet aussi de créer des micromondes. L'intérêt ces environnements est de favoriser un apprentissage progressif, grâce à un contrôle de l'enseignant sur les nouveaux concepts à aborder. Au début, l'étudiant dispose de peu d'outils pour résoudre les problèmes. Puis, l'enseignant le met face à un problème difficile à résoudre avec seulement ces outils. Suite à cela, il introduit un nouveau concept qui enrichit le micromonde. Cela n'est pas possible dans un langage traditionnel où tout est accessible tout de suite. Stelios Xinogalos[16] et Linda Mciver[7] relèvent ainsi un certain nombre de problèmes pédagogiques posés par les langages traditionnels, qui peuvent généralement être résolus par l'utilisation de micromondes.

Pour savoir si l'apprentissage de la concurrence permet de mieux développer les capacités algorithmiques des étudiants, nous devrons procéder à des expérimentations dans deux classes réputées égales et leur soumettre un questionnaire et un test initial. À l'une de ces classes, nous donnerons notre cours de programmation basé sur la concurrence ; à l'autre un cours basé sur la programmation séquentielle. Dans les deux cas, nous utiliserons une méthode pédagogique identique basée sur les micromondes. Au terme du cours de deux mois, un test de programmation aura lieu dans chacune des classes.

ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical eport, ACM Press and IEEE Computer Society Press, December 2013.

Claus Brabrand. Constructive alignment for teaching model-based design for concurrency (a case tudy on implementing alignment).

Manuel Carro, Ángel Herranz, and Julio Mariño. A model-driven approach to teaching concurrency. rans. Comput. Educ., 13(1) :5 :1–5 :19, February 2013.

Michael B. Feldman and Bruce D. Bachus. Concurrent programming can be introduced into the ower-level undergraduate curriculum. SIGCSE Bull., 29(3) :77–79, June 1997.

W. Feurzeig and Beranek Bolt. Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project [microform] / W. Feurzeig and Others. istributed by ERIC Clearinghouse [Washington, D.C.], 1969.

Caitlin Kelleher and Randy Pausch. Using storytelling to motivate programming. Commun. ACM, 0(7) :58–64, July 2007.

Linda Mciver. The effect of programming language on error rates of novice programmers. In 12th nnual Workshop of Psychology of Programmers Interest Group (PPIG), Corigliano, pages 181–192, 2000.

Seymour Papert. Computer-based microworlds as incubators for powerful ideas. The computer in the chool : Tutor, tool, tutee, pages 203–210, 1980.

Martin Quinson and Gérald Oster. A Teaching System To Learn Programming : the Programmer's earning Machine. In ACM Conference on Innovation and Technology in Computer Science Education 2015, ilnius, Lithuania, July 2015. ACM.

Caitlin Sadowski, Thomas Ball, Judith Bishop, Sebastian Burckhardt, Ganesh Gopalakrishnan, Joseph ayo, Madanlal Musuvathi, Shaz Qadeer, and Stephen Toub. Practical parallel and concurrent rogramming. In Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, IGCSE '11, pages 189–194, New York, NY, USA, 2011. ACM.

M. Satratzemi, S. Xinogalos, and V. Dagdilelis. An environment for teaching object-oriented programming : objectkarel. In Advanced Learning Technologies, 2003. Proceedings. The 3rd IEEE International onference on, pages 342–343, July 2003.

Lynn Andrea Stein. Challenging the computational metaphor : Implications for how we think, August 999.

Herb Sutter. The Free Lunch Is Over : A Fundamental Turn Toward Concurrency in Software. Dr. obb's Journal, 30(3), 2005.

Peter Van Roy, Joe Armstrong, Matthew Flatt, and Boris Magnusson. The role of language paradigms n teaching programming. SIGCSE Bull., 35(1) :269–270, January 2003.

Christopher Watson and Frederick W.B. Li. Failure rates in introductory programming revisited. In roceedings of the 2014 Conference on Innovation &#38 ; Technology in Computer Science Education, ITiCSE 14, pages 39–44, New York, NY, USA, 2014. ACM.

Stelios Xinogalos. An evaluation of knowledge transfer from microworld programming to conventional rogramming. Journal of Educational Computing Research, 47(3) :251–277, 2012.

Mots-Clés: programmation, concurrence, micromondes, passage de messages, didactique