To appear in: *Proceedings of the Eighth International Conference of Complex Systems (ICCS 2011).*

Design Principles for a Visual Programming Language to Integrate Agent-Based Modeling in K-12 Science

Pratim Sengupta
Mind, Matter & Media Lab, Vanderbilt University
pratim.sengupta@vanderbilt.edu

Despite arguments supporting the pedagogical benefits of using agent-based computer modeling and programming (ABMP) as the medium to learn science, ABMP remains yet to be well-integrated with K12 science. To address this issue, I present ViMaP, a new, agent-based visual-and-tactile programming language, its design principles, and also provide evidence for the effectiveness of these principles.

1 Introduction

Despite arguments supporting the pedagogical benefits of using agent-based computer modeling and programming (ABMP) as the medium to learn science [1][3][4], ABMP remains yet to be well-integrated with K12 science. To address this issue, I present ViMaP [2], a new, agent-based visual-and-tactile programming language, its design principles, and also provide evidence for their effectiveness.

2 ViMaP & Its Design Principles

Real-world constraints that prevent advanced computing technologies from being adopted in K12 settings include teachers' lack of technological and computational experiences, tension between learning programming syntax and learning domain concepts [4], centralized policies that often prevent installation of new software. To address these issues, we are developing ViMaP based on the following design principles: P1) *Close-coupling between primitives and parameters;* P2) *Support both domain-specific and domain-general primitives;* P3) *Multiple "liveness" factors to support debugging through algorithm visualization; and* P4) *Fewer core primitives to support language design and adaptation by end-users.* The fourth principle seeks to support a long-term learning progression of computational thinking, by enabling students to design programming languages. In this paper, I focus on the first three.

ViMaP is currently based in the NetLogo platform [3] and can be run either locally on a desktop or as an online applet. The ViMaP world is divided into two

parts: construction-world (C-World) where the learner constructs his or her program, and enactment world (E-World), where a protean computational agent – the classic Logo turtle (or multiple agents (turtles)) - carry out the learner's commands through movement in representational space. Programming commands in C-World are also represented visually as turtles, which can be dragged and moved by mouse-clicks.

3 Design Principles at Work: An User Study

We conducted a design-based research study where eight 5th grade students with no programming background, recruited from various public schools in Nashville, successfully completed a curricular unit in mechanics using ViMaP in a weekly, hour long after-school session over five weeks. In Phase 1, students generated geometric shapes using *domain-specific primitives* for controlling the movement of an agent (e.g., moving forward by a particular step-size; speed-up and slow-down - which automatically change the step-size by an argument chosen by the user – see P1), and *domain-general primitives,* for control structures (loops, conditionals), and aesthetics (color and drawing). In Phase 2, students were then asked to use these primitives to model different scenarios involving physical setups of a ball rolling down a ramp from rest and another ball being initially pushed on the floor by different amounts of force. Analysis of students' programs, and their interview-based explanations of the challenges they faced and how they overcame those challenges, indicate that a) all students were able to identify the relationships between constant speed, constant acceleration, distance travelled and time through several iterations of debugging in Phase 1, often without soliciting help from the teacher, and b) they used two specific debugging scaffolds (step-by-step breakpoints, and increasing delay between the execution of successive commands – see P3) to do so. In Phase 2, we found that learners' interactions with the real-world aided their debugging, as their programs iteratively progressed towards more sophisticated representations of kinematics. 90% of the learners also invented measures of acceleration and speed, indicating that they successfully transferred both domain-specific and domain-general elements of computational modeling from Phase 1 to Phase 2 without instructional prompts. This suggests that the design principles (P1-P3) led to a seamless integration of programming activities and domain concepts in the context of learning kinematics.

Selected Bibliography

[1] Repenning, A., & Sumner, T., 1995, Agentsheets: A medium for creating domain-oriented languages. *Computer,* 28(3), 17-25.

[2] Sengupta, P. & Wright, M, 2010, *ViMaP: Virtual Multi-Agent-based Programming Language,* [Computer Software], Vanderbilt University.

[3] Tisue, S. & Wilensky, U., 2004, *NetLogo: A Simple Environment for Modeling Complexity*. International Conference on Complex Systems, Boston, MA.

[4] Sherin, B, et al., 1993, Dynaturtle revisited: Learning Physics Through Collaborative Design of a Computer Model, *Interactive Learning Environments*