

# Learning Efficiencies Using Multi-Agent Based Game Simulations

Gordon Barker<sup>a</sup>, Barry Lee Reynolds<sup>b</sup>, Andrew Chan-Chio Lao<sup>b</sup>, Frank Wu<sup>b</sup>

<sup>a</sup>*Athabasca University, Alberta, Canada*

<sup>b</sup>*Graduate Institute of Network Learning Technology  
National Central University, Jhongli, Taiwan*

gordon@barker.name

**Abstract:** Learning effectiveness is a function of effective pedagogical practices. Accordingly, the question for learning technology designers should be “What combination of instructional strategies and delivery media will best produce the desired learning outcome for the intended audience?” This paper reviews four successful products that incorporate solid instructional strategies with agent based game simulations to provide a positive learning environment for students.

**Keywords:** game based learning, simulation, learning technology, end-user programming

## Introduction

The first introduction to basic computer science concepts that students receive occurs in elementary school. The students learn how to use some Windows applications such as PowerPoint, Word and Paint. Computer programming courses however, occur less frequently and are taught with lesser quality than that of other software packages. Research has revealed that children can improve their logical thinking, organization, judgment and problem solving ability through computer programming.

Sedighian [1] studied various design strategies of simulations on children’s learning of and attitudes towards complex mathematical concepts. He researched the effectiveness of teaching two-dimensional geometry using different strategies. He developed three approaches including “direct object manipulation” (DOM) in which students manipulate a geometric object directly; “direct concept manipulation” in which the students manipulate the mathematical representation of the shape and “reflective direct concept manipulation” (RDCM) in which visual feedback aids were gradually removed, requiring students to think carefully about the object [2].

The conclusions were that simple manipulation of an object in a game or simulation environment had a slight negative impact on learning. The greatest increase in understanding came from those students using the RDCM method accentuated with involvement by the instructor.

The main reason for this result is that learning how to play a game does not necessarily imply learning the target instructional domain [3]. Learning happens only when students actively build the connections between game moves and underlying knowledge.

In order to solve this problem, computer scientists and programmers have developed the concept of programming through demonstration. End-users, taking advantage of the

advances in GUI interfaces, interact with software agents by “showing” or demonstrating to programmable agents how to behave.

This is a characteristic common amongst the successful game based learning systems analyzed in this report. Students cannot just interact with a game; they must construct it and have enough underlying knowledge to program the game agents to perform appropriately in the game environment.

## 1. Early Developments at Apple

The idea of end-user programming became an especially hot topic in the 1980s at Apple Computer. Apple was most interested in how they could encourage young children to continue learning computer programming after class was over. Most children (and novice adult programmers) that learn computer programming do not want to continue to independently program after class [4]. Apple released a number of child-oriented programming languages, KidSim, Cocoa, and StageCast Creator.

### 1.1. *KidSim*

Apple determined that the reason children often failed at excelling at programming languages such as Logo and Smalltalk was because they had trouble abstracting their mental representations to something that a computer could understand [5]. Therefore, children would often struggle with writing of the simplest programs [7]. Pheasey, Underwood, & Underwood [9] were among the first that proclaimed that all end-users could grasp the concept of computer program abstractions if they approached them in a more concrete way.

This idea progressed to allowing end-users to program simulations through direct manipulation of agents instead of keying in programming code; this became the focus of about ten years of study with KidSim/Cocoa. Simulations allowed children to construct worlds and characters (agents) to interact within these worlds [8]. In a matter of minutes, an end-user programmer can begin to construct a simulation inside of KidSim without any knowledge of formal programming languages.

After KidSim’s initial development, researchers conducted several studies involving children between the ages of 5-15 years to explore how students could grasp programming concepts through demonstration when utilizing a graphical programming environment [7]. The Gilmore, et al [9] study involved 56 children between the ages of 10-13 years old working together in groups of 2-3 over a period of 2 days to 3 weeks between 2-12 hours in total. The results of their study showed that children were readily able to program simulations inside KidSim. The problems that children faced involved the ability to have a deep understanding of the programming abstractions that the interactions with the agents and the simulated environments represented.

Brand & Rader [10] showed that students mastered some skills often utilized by professional programmers such as decomposing of mental pictures into objects; they often tried to draw entire scenes as one object instead of breaking down the different parts to express their relationships. Furthermore, they had difficulties in discarding any of their work.

### 1.2. *Cocoa*

Although the developers of Cocoa still thought it was useful for child end user programs to program simulations, they no longer considered it a cure for the lack of end user programmers. Promoters began to admit that Cocoa was more domain specific and that it

would work best in educational settings with children in grades K-8 [6]. Students could still use Cocoa for programming agents and then debug their simulations when agents performed differently than expected. Additionally, with the support of a well-informed teacher, students could grasp concepts of sub-domain programming.

Rader, Brand, and Lewis's work was a year-long study, which yielded some surprisingly *negative* findings. Most of the younger students spent too much time drawing and not enough time trying to simulate the science concepts that the teachers wanted students to explore. Even with the ease of using Cocoa, most of the students created programs that were unsophisticated and failed to grasp many concepts of the program's operation. Students needed much guidance to get their agents to perform even the simplest of tasks. The basic conclusion was that an interface may be created that is very easy for students to interact within but without additional support by teachers, it would be too difficult for students to understand the underlying mechanisms involved.

### 1.3. *Stagecast Creator*

Stagecast Creator allows students not only to discover implicit knowledge by manipulating tangible objects on the screen but also offers them the ability to create a fantastic game by themselves. The software tools are easy to combine past experiences with new knowledge while promoting conceptual learning through the broadcasting of dynamic images. Stagecast Creator is designed to allow children and novice programmers to develop interactive games and simulations without the use of conventional programming languages. Students often present their opinions and creativity in game-based learning environments through diagramming, writing, multi-media, video and simulations. The environment of Stagecast Creator echoes those that constructivists value as rich for student learning. "The educational philosophy of constructivism states that children learn best when they actively create" [11].

Stagecast Creator is also a visual programming language especially designed for children. During programming, one needs to know nothing about programming language because of two technologies: programming by demonstration (PBD) and visual before-after rules [12]. Programming by demonstration is a method for teaching a computer by demonstrating the task to transfer directly instead of programming through machine commands. The benefits of PBD are that users do not need to learn an obscure programming language; a program is automatically created corresponding to the user's actions. Therefore, "Stagecast Creator is an extremely open-ended software program based on the concept of programming by example." [12].

### 1.4. *AgentSheets*

AgentSheets is an agent-based authoring tool that allows non-programmers to create agents with behaviors and missions, teach agents to react to information and process it in personalized ways, and combine agents to create sophisticated interactive simulations and models [13]. The semantic meaning of the AgentSheets paradigm is based on spatial reasoning primitives like neighborhood and distance [14]. It consists of agents, spreadsheets, and Java authoring technologies such as "Ristretto".

In AgentSheets, students through a point-and-click interface can program agents, gain knowledge by creating simulations that are practical and connected to real experiences, and learn through designing. By exchanging the role of students and teachers, students can create their own applications by using their creativity. Although running simulations is an effective means of learning, creating a simulation challenges students to develop a thorough knowledge and understanding about a subject. The goal of AgentSheets is to allow students

to learn through the support of technology. Designing an agent's behavior can prompt the student to ask questions they might otherwise have overlooked.

### 1.5. *Scratch*

Scratch is one of the current most popular programming languages among teens. Scratch was designed with a constructionist theory of learning in mind, which assumes that people learn best when they are active participants in design activities [15]. Scratch is a networked, media rich programming environment designed to enhance the development of technological fluency at afterschool centers in economically disadvantaged communities [16]. Scratch adds programmability to the media rich and network based (Net 2.0) activities that are most popular among youth. With the current level of computer power available on a cheap desktop, Scratch supports a new programming paradigm that has made programming activities that were previously impossible now possible.

Scratch is different from other novice-friendly programming environments in that it removes the debugging process and any risk of syntax errors [17]. In the Scratch environment, you simply snap together graphical blocks (much like LEGO blocks) without worrying about semi-colons, square brackets or other syntactical styles. The blocks are designed to fit together only in the ways that make sense, so there are no "syntax error" messages as in traditional programming languages. Scratch also allows the user to add blocks as the program is running, making it very easy to "play with the code," testing out new ideas incrementally and iteratively.

## Conclusions

There is a common thread in the design of the programming systems reviewed in this paper. The design of each one has followed the rules outlined by Resnick and Silverman in their paper "Some Reflections on Designing Construction Kits for Kids" [15].

- **Design for designers:** The success of a programming system depends on its ability to engage the imagination of the user. It makes sense then, to design a system that allows users to design things in a meaningful way.
- **A Little bit of programming goes a long way:** The introduction of programming has had mixed effectiveness in schools. Many students see programming as a narrow, technical skill and often reach a "plateau" that they cannot easily go beyond. This is not really a problem and they can still learn a great deal while staying at their plateau.
- **Give People what they want:** It is more productive to watch users interact with a prototype and gain understanding of what they want to do from this interaction rather than asking them to explain their needs.

## References

- [1] Sedighian, K. (1997). Challenge-driven learning: A model for children's multimedia mathematics learning environments. Paper presented at the ED-MEDIA 97: World Conference on Educational Multimedia and Hypermedia
- [2] Klawe, M. (2000). When Does The Use of Computer Games And Other Interactive Multimedia Software Help Students Learn Mathematics? Paper presented at the NCTM Standards 2000 Technology Conference
- [3] Wolfe, J. (1997). The Effectiveness of Business Games in Strategic Management Course Work. *Simulation and Gaming*, 28(4), 360-376.

- [4] Smith, D.C., Cypher, A. & Spohrer, J. (1994). KidSim: Programming agents without a programming language [Electronic version]. *Communications of the ACM*, 37(7), 54-67. Retrieved December 6, 2007 from <http://doi.acm.org/10.1145/176789.176795>
- [5] Lieberman, H. (Ed.). (2001). *Your wish is my command: Programming by example*. San Francisco: Morgan Kaufmann.
- [6] Smith, D.C., Cypher, A. & Schmucker, K. (1996). Making programming easier for children [Electronic version]. *Interactions*, 3(5), 58-67. Retrieved December 6, 2007 from <http://doi.acm.org/10.1145/234757.234764>
- [7] Smith, D.C., Cypher, A. & Spohrer, J. (1994). KidSim: Programming agents without a programming language [Electronic version]. *Communications of the ACM*, 37(7), 54-67. Retrieved December 6, 2007 from <http://doi.acm.org/10.1145/176789.176795>
- [8] Cypher, A., & Smith, D. C. (1995). *KidSim: end user programming of simulations* (pp. 27-34): ACM Press/Addison-Wesley Publishing Co. New York, NY, USA.
- [9] Gilmore, D. D., Pheasey, K., Underwood, J., & Underwood, G. (1995). Learning graphical programming: An evaluation of KidSim. In *Proceedings of InteractN5*, 145-150.
- [10] Brand, C. & Rader, C. (1996). How does a visual simulation program support students creating science models [Electronic version]. *Proceedings: 11<sup>th</sup> IEEE International Symposium on Visual Languages*, 110-111. Retrieved December 6, 2007 from <http://doi.ieeecomputersociety.org/10.1109/VL.1996.545276>
- [11] Cypher, A. & Smith, D.C. (1995). KidSim: End user programming of simulations [Electronic version]. *Proceedings of the SIGCHI conference on human factors in computing systems*, 27-34. Retrieved December 6, 2007 from <http://doi.acm.org/10.1145/223904.223908>
- [12] Smith, D. C., Cypher, A., & Tesler, L. (2000). *Programming by example: novice programming comes of age* (Vol. 43, pp. 75-81): ACM Press New York, NY, USA.
- [13] Martin, C. K.(1999). *Teaching Basic Computer Science Concepts through Programming by Example: A study teaching middle school students computer science using Stagecast Creator*.
- [14] AgentSheets. (2007). Retrieved from <http://www.agentsheets.com/products/index.html>
- [15] Repenning, A. (1991). *Creating User Interfaces with AgentSheets*. IEEE Conference on Applied Computing
- [16] Resnick, M., Silverman, B. (2005) *Some reflections on designing construction kits for kids*. *Proceedings of the 2005 conference on Interaction design and children*, pp. 117-122
- [17] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. (2004). *Scratch: A sneak preview*, *Second International Conference on Creating, Connecting and Collaborating through Computing* (pp. 104-109). Kyoto, Japan