RoboBuilder: A Program-to-Play Constructionist Video Game

David Weintrop, *dweintrop@u.northwestern.edu*
Learning Sciences, Northwestern University

Uri Wilensky, *uri@northwestern.edu*
Learning Sciences, Computer Science and Complex Systems, Northwestern University

Abstract

As the popularity of video games continues to grow, we see the medium as an increasingly important venue for giving young learners an opportunity to engage in constructionist learning. This paper introduces RoboBuilder, a blocks-based programming game that draws on constructionist design principles as well as video game norms to create a fun, challenging computational learning environment. After describing the game and discussing previous work that informed its design, we introduce two innovative features of RoboBuilder.

Keywords

Video games, Low-threshold Programming Environments, Visual Programming, Design

Introduction

The constructionist community has a rich history of creating innovative computer-based learning environments. These environments range from exploratory microworlds to story and game authoring environments to scientific modelling tools. In this paper we introduce RoboBuilder, a constructionist video game, and discuss some of the interesting features that arise when bringing constructionist design principles to the video game medium. RoboBuilder is a blocks-based programming game that seeks to introduce players to fundamental programming concepts through challenging and fun gameplay. This paper proceeds with a description of RoboBuilder and a review of prior work that inspired RoboBuilder's design. We then present two innovative aspects of the environment before discussing the next steps in our research agenda.

Meet RoboBuilder

RoboBuilder (figure 1) is a blocks-based programming game that challenges players to design and implement strategies to make their on-screen robot defeat a series of progressively more challenging opponents. The players' on-screen robot takes the form of a small tank, which competes in one-on-one battles against opponent robots. The objective of the game is to defeat your opponent by locating and firing at it while avoiding incoming fire from your adversary. Unlike a conventional video game where players control their avatars live during battle, in RoboBuilder, players must program their robot before the battle begins. To facilitate this interaction, RoboBuilder has two distinct components: a programming environment where players define their robot's strategy, and an animated robot battleground where their robot battles. Players first use the programming interface to implement their robot's behaviours before hitting the 'Go' button, which launches the battleground screen where the programmed strategies are enacted by their robot as it competes. To implement their strategy, players are provided with a set

of language primitives to program their robot; the language includes movement blocks (ex: *forward*, *turn left*, *turn gun right, fire*) to control their robot's motion, event blocks (ex: *When I See a Robot*, *When I Get Hit*) to control when instructions will execute, and control blocks (ex: *Repeat, If/Then*) that can be used to introduce logic into their robot's strategy. The battleground interface is read-only; once the battle starts, players cannot interact with or alter their robots.

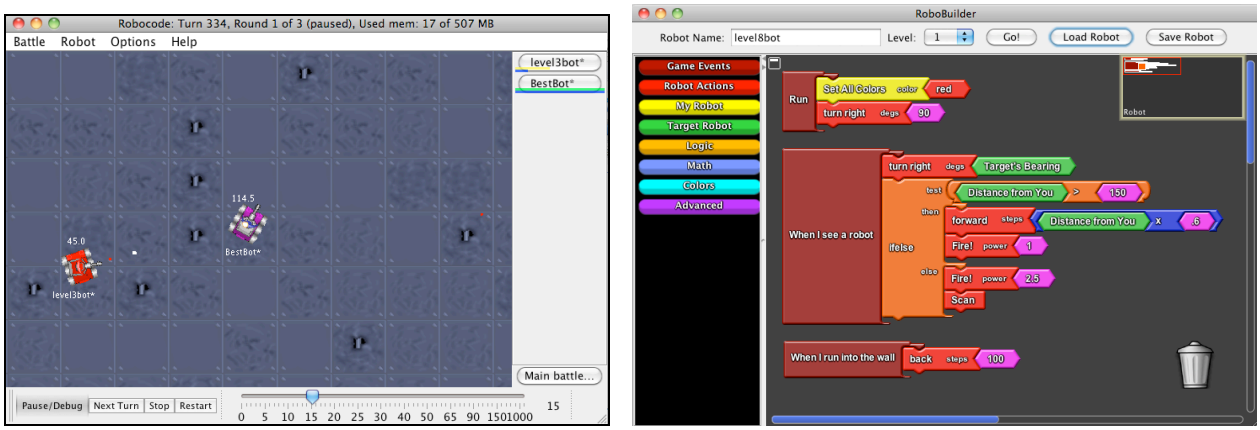

Figure 1. RoboBuilder's two screens. The battle screen, on the left, is where players watch their robot compete; the construction space, on the right, is where players implement their strategies.

The overarching learning goal of RoboBuilder is to give players the experience of developing and expressing ideas in a computational medium. The design of the game is also intended to enable learners to become comfortable using general programming strategies such as incremental development, breaking larger goals down into subtasks, and using feedback from prior runs of a program to inform revisions. Over the course of game play, we aim to give players a new understanding of what it means to program while fostering general-purpose programming skills that have broad application beyond our game.

There are three main reasons we decided to make a robot-battling task the objective of our game. First, programming robots to accomplish specific tasks has been found to be fun and motivating in both a video game context as well as in learning contexts (Berland & Wilensky, 2004; Hancock, 2001; Martin et al, 2000). Second, the body syntonic nature of controlling the robot and the direct mapping of programming blocks onto in-game behaviours makes gameplay accessible to players with no prior programming experience. Finally, as educational researchers conducting design research (Edelson, 2002), rapid prototyping and iterative development is important to our research agenda. Thus, we built our game on top of Robocode (Nelson, 2001), a problem-based learning environment. This allowed us to have a working prototype early in the research process.

Previous Work

Low-Threshold Programming Environments

A great deal of work has been done on the design and implementation of programming languages and environments for beginners (for a review see: Guzdial, 2004 or Kelleher & Pausch, 2005). One design strategy for novice languages that has gained popularity is a visual programming approach that presents language primitives as on-screen objects to be assembled using a drag-and-drop interface. These blocks-based environments have also been called "component-oriented microworlds", where components are autonomous, reusable computational objects of varying technical and behavioural complexity (Kynigos et al., 1997). In these environments, programming takes the form of combining components, or groups of components, to create

complex, composite structures that carry computational meaning.

RoboBuilder's programming interface is a modified version of the OpenBlocks framework (Roque, 2007), which is an open source Java library used to create blocks-based programming environments. This library allows the language designer to define the set of primitives as well as the shape of each block, which in turn dictates how and where the pieces can be used. Blocks-based programming languages, like LogoBlocks (Begel, 1996) with its snap-to-compile feature, and environments, like Scratch (Resnick et al., 2009) with its stage where programs are visually run, have influenced RoboBuilder's language and the design of its interactive construction space.

Video games as learning environments

Video games are becoming increasingly pervasive in youth culture. The potential of video games as learning environments was recognized by Papert who wrote that they empower children to "test out ideas about working within prefixed rules and structures in a way few other toys are capable of doing" (Papert, 1993, p. 4). A growing body of work has argued that video games are an effective medium for teaching and learning (Prensky, 2001). Gee (2003) argued that video games are the source of a powerful new literacy with benefits ranging from identity formation to reasoning skills. The benefits of video games extend beyond in-game learning, as social aspects of video games are similar to effective non-virtual learning environments (Stevens et al., 2007).

Making Game Play a Constructionist Activity

Marrying constructionism and video games is not new. A number of constructionist learning programs have made video games a central part of their learning agenda (Goldstein et al., 2001; Harel & Papert, 1990; Kahn, 1999). For example, Caperton, in her work with Globaloria, has been very successful appealing to kids by having video games be the output of constructionist learning environments (Caperton, 2010). RoboBuilder seeks to leverage video games to serve a similar motivational purpose, but does so in a very different way; instead of constructing games, in RoboBuilder the player-created constructions are actually used to play the game. This approach is not without its challenges. By making the learner-constructed artefact the mechanism for playing the game, we are faced with what Noss and Hoyles call the "play paradox" (1996). On the one hand, RoboBuilder offers an engaging exploratory environment that supports many ways of solving the proposed problem; on the other hand, there are specific learning objectives we have for RoboBuilder. How can we be sure that the players are learning what we have designed the environment to teach? One solution is to design the environment such that "the system carries with it elements of what is to be appreciated" (Noss & Hoyles, 1996, p. 132). This idea fits very naturally into a video game context. A central goal of RoboBuilder is to give players the experience of developing ideas with and expressing ideas in a computational medium. Because RoboBuilder has a clear goal (defeating your opponent) and easily identifiable notion of success (winning the match), the game rewards players who successfully encode their strategies with the provided computational form. In this way the player's goals (to win the match) are aligned with our learning goal (players learning to encode ideas in a computational medium).

Video Games for Situating Programming Abstractions

By framing RoboBuilder as a video game and making programming the central activity of gameplay, we seek to put the player in a problem-solving context that challenges them to computationally implement potential solutions to the in-game objective. Thus, we use the game as a way to situate the programmatic abstractions the player must use to participate. The language

primitives develop meaning for the player through the iterative, construction process central to gameplay. "These meanings become reshaped as learners exploit the available tools to move the focus of their attention onto new objects and relationships" (Noss & Hoyles, 1996, p. 122). In this way, players are "abstracting *within*, not *away from*, the situation" (Noss, Healy, & Hoyles, 1997, p. 228, emphasis in origial). By having players express their ideas in the computational medium and then witness their expressed ideas enacted, the video game provides an opportunity for the learner to interact with the representational system and form rich connections with the language elements and the computational behavior they embody. These once abstract language primitives undergo a process of "concretion" (Wilensky, 1991) for the learner, whereby they develop meaning in the context of the game.

RoboBuilder's language primitives and interface were designed to facilitate this concretion process. When developing such environments, it is essential that the designed representational system "provide 'natural' expressive power - the right things to talk about, and ways to talk about them" (Hoyles, Noss, & Kent, 2004, p. 320). We achieve this "natural expressive power" by providing a language that blends blocks with clear in-game meaning (like: *Turn Right* and *My Energy*) with conventional programming blocks (*Repeat*, *While,* and *If/Then*) that could be used to bring the player's envisioned strategies to life. In this way, RoboBuilder is an example of a component-oriented microworld that gives the player the ability to "build and think in terms of objects that are close to their domain of interest" (Kynigos et al., 1997, p. 231). We use the video game context, and the dynamic interactive challenge that comes with it, to situate programming abstractions and motivate players to computationally reify their imagined strategies with a programming language designed to fit with the challenge at hand.

Where video games excel as contexts for situating programmatic abstractions is their ability to encourage and foster constructions of increasing complexity and size. Video games can reward small and simple programs, but also include incremental challenges that require the learner to create more advanced, complex programs. Thus, a video game can not only be a low-threshold, high-ceiling programming environment, it can also include a built in mechanism to encourage learners to progress from basic programs to more advanced, sophisticated constructions.

Conclusion and Next Steps

This paper introduces RoboBuilder, a program-to-play constructionist video game designed to give younger learners a chance to develop and apply programming skills in a fun and motivating context. We are currently conducting a pilot study where we record and clinically interview participants as they play the game. These preliminary sessions have shown RoboBuilder to be an engaging and motivating environment and we have observed some of the desired learning goals enacted by participants during gameplay. As for our larger research agenda, we see RoboBuilder as the first in a series of similar constructionist video game we will design. We recognize that the task of controlling a tank-like robot and shooting at enemies may only appeal to only a subset of learners; particularly boys, a population already over-represented in the field of computer science (Margolis & Fisher, 2003). Based on our findings from RoboBuilder, we hope to create more such environments that use the same program-to-play approach but have different in-game goals, use different language primitives and appeal to different (and hopefully even wider) audiences.

References

Begel, A. (1996). LogoBlocks: A Graphical Programming Language for Interacting with the World.

Berland, M., & Wilensky, U. (2004). *VBOT: Collaborative Constructionist Learning using a Virtual*

Robotics Environment. Presented at the Annual Meeting of AERA, San Diego, CA.

Caperton, I. H. (2010). Toward a theory of game-media literacy: Playing and building as reading and writing. *International Journal of Gaming and Computer-Mediated Simulations*, *2*(1), 1–16.

Edelson, D. C. (2002). Design research: What we learn when we engage in design. *The Journal of the Learning sciences*, *11*(1), 105–121.

Gee, J.P. (2003). *What Video Games Have to Teach Us About Learning and Literacy*.Palgrave Macmillan.

Goldstein, R., Kalas, I., Noss, R., & Pratt, D. (2001). Building rules. *Cognitive Technology: Instruments of Mind*, 267–281.

Guzdial, M. (2004). Programming environments for novices. *C. S. Education Research*, *2004*, 127–154.

Hancock, C. (2001). Children's understanding of process in the construction of robot behaviors. *Annual Meeting of the American Educational Research Association, Seattle, WA*.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments*, *1*(1), 1–32.

Hoyles, C., Noss, R., & Kent, P. (2004). On the integration of digital technologies into mathematics classrooms. *International Journal of Computers for Mathematical Learning*, *9*(3), 309–326.

Kahn, K. (1999). From prolog to Zelda to ToonTalk. *Proceedings of the International Conference on Logic Programming* (pp. 67–78).

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, *37*(2), 83–137.

Kynigos, C., Koutlis, M., & Hadzilacos, T. (1997). Mathematics with component-oriented exploratory software. *International Journal of Computers for Mathematical Learning*, *2*(3), 229–250.

Margolis, J., & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. The MIT Press.

Martin, F., Mikhak, B., Resnick, M., Silverman, B., and Berg, R. (2000). "To Mindstorms and beyond: Evolution of a construction kit for magical machines." In (eds) Hendler, J. and Druin, A., editors, *Robots For Kids*, chapter 1.

Nelson, M. (2001). Robocode. *IBM Advanced Technologies*.

Noss, R., Healy, L., & Hoyles, C. (1997). The construction of mathematical meanings: Connecting the visual with the symbolic. *Educational Studies in Mathematics*, *33*(2), 203–233.

Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings*. Springer.

Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. Basic Books.

Prensky, M. (2001). *Digital game-based learning*. New York: McGraw Hill.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., et al. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60.

Roque, R. V. (2007). *OpenBlocks: an extendable framework for graphical block programming systems*. Massachusetts Institute of Technology.

Stevens, R., Satwicz, T., & McCarthy, L. (2007). In-game, in-room, in-world: Reconnecting video game play to the rest of kids' lives. *MacArthur Series on Digital Media and Learning*, 41–66.

Wilensky, U. (1991). Abstract Meditations on the Concrete and Concrete Implications for Mathematics Education. In I. Harel & S. Papert (Eds.), *Constructionism*. Norwood N.J.: Ablex Publishing Corp.