Games and Culture
Volume 3 Number 3-4
July 2008 309-332

10.1177/1555412008317312
http://gac.sagepub.com
hosted at
http://online.sagepub.com

Making Computer Games and Design Thinking

A Review of Current Software and Strategies

Elisabeth R. Hayes
Arizona State University
Ivan Alex Games
University of Wisconsin–Madison

This article provides an overview of computer software and instructional strategies intended to engage young people in making computer games, to achieve a variety of educational goals. It briefly describes the most popular of such programs and compares their key features, including the kinds of games that can be created with the software, the types of communities and resources that are associated with each program, claims made for learning outcomes resulting from use of the software, and the results of empirical research (if any) on the application and outcomes of the software in formal or informal educational settings. A key finding is that existing software and educational applications stress the goal of teaching users about computer programming and place little or no emphasis on teaching concepts related to game design. It concludes by discussing the potential value of explicit attention to "design thinking" as goal of game making in education.

Keywords: *video games; learning; thinking; game design; software*

Over the past two decades, numerous educational scholars and practitioners have explored the potential benefits of computer games for enhancing learning in and out of formal educational settings. Research on the use of games for learning has addressed academic areas such as language and literacy (Gee, 2003), mathematics (Kafai, 1995), history (Squire, 2006), and science (Barab, Thomas, Dodge, Carteaux, & Tuzun, 2005). One of the central characteristics of good games, in the view of scholars such as Gee (2003), is that they allow their players to think about them as designed objects (p. 42).

The value of such a "designer mentality" has been recognized by a number of educational scholars, who see such a perspective as a fundamental ability required for full participation in the knowledge economy (Gee, Hull, & Lankshear, 1996; New London Group, 1996; Perkins, 1986; Wiggins & McTighe, 2005). Corporations in today's economic landscape look for employees capable of solving problems in creative and effective ways, as well as of producing new knowledge that can help them adapt and become competitive in the global market (Nussbaum, 2005). "Design thinking" has received growing attention from business leaders who seek to

improve the ability of their employees and companies to identify creative solutions to increasingly complex challenges (Hyer, 2006; Kelley & Littman, 2001; Mau, Leonard, & Institute Without Boundaries, 2004). Although some of this literature threatens to make design thinking into the latest educational and corporate fad, there is sound evidence that design thinking merits—and is receiving—serious attention. For example, the Hasso Plattner Design Institute at Stanford University (n.d.) was conceived as "a place for Stanford students and faculty in engineering, medicine, business, the humanities, and education to learn design thinking and work together to solve big problems in a human centered way."

Games, Learning, and Design Thinking

Kafai (2006) identifies two general approaches to using games for learning: instructivism and constructionism. Instructivism, the more common approach, involves the design and use of "educational" games in school or afterschool program curricula, based on the belief that games are inherently more motivating than traditional classroom activities (e.g., Kirriemuir & McFarlane, 2004; Rosas et al., 2003). This approach also includes the use of commercial off-the-shelf videogames in educational settings—for example, teaching aspects of world history through Age of Empires or Civilization. These approaches have the potential to elicit design thinking indirectly as players discover the design patterns that underlie the game (Gee, 2003).

Although instructivist approaches to using games for learning have dominated the literature, constructivist efforts are growing in popularity, partly because of the increasing availability of relatively easy-to-use programming and design tools. In contrast with instructivism, a constructionist approach stems from Seymour Papert's (1991) proposition that learning happens "especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe" (p. 1).

As applied to the use of games for learning, a constructionist perspective underlies efforts to engage young people in making their own games, to achieve a variety of educational goals (Kafai, 2006). This approach commonly entails providing learners with a set of game development tools, such as authoring software, game engines, or programming environments, and some kind of support for learning to use the tools as well as for constructing games. In this case, design thinking might be fostered through the actual experience of thinking through design problems in the course of making games.

Our goal in this article is twofold: (a) to provide an overview of different approaches to using game making and, (b) more specifically, to identify the implications of these approaches for understanding and facilitating young people's acquisition of a "designer mentality" through game making.

Four Approaches to Making Games for Learning

For the purpose of this review, we have identified four main purposes or goals for using game making in educational settings, based on the current literature. We describe examples of each overarching purpose in the following sections. By far the most common use of game creation has been for the purpose of helping students learn programming tools and concepts. A second, related approach has been to use game making in programs intended to attract girls to computer science and technical fields, with programming taking a secondary place to broader goals of confidence building and "empowerment." A third approach has been to use game making as a means of enhancing understanding of an academic domain, sometimes with learning programming as an additional goal, or in other cases, using software that minimizes or eliminates the need to learn programming. Last, a fourth general approach has focused specifically on facilitating learners' understanding of and ability to make games or features of games, such as types of game rules, with software specifically designed to support making games.

For each of the four general approaches, we will describe some more prominent examples of tools and educational strategies, including educational goals, features of the software, and documentation of learner outcomes. We will suggest what can be learned from each approach about how game making might be used most effectively, particularly in relationship to the goal of fostering design knowledge among learners.

Our review is limited in several ways. The examples were selected based on the availability of information about the software and strategies as well as published accounts of their use in formal or informal educational settings. Thus, some popular commercial programs, such as 3D Game Maker, The Games Factory, and RPG Maker (*Game Creation Resources*, n.d.), are not included because we could not locate examples of their use for explicitly educational purposes. Other popular software programs used by educators such as Alice (Cooper, Dann, & Pausch, 2000) and Scratch (n.d.) are not included because of lack of examples of their application to making games in particular. Even many examples we include have little research support for their claims of efficacy, an issue we return to in our final discussion. The goal of this article is not to provide an extensive or complete review of the research literature, and we do not discuss the empirical studies in-depth but rather use their findings to illustrate key issues and insights. Last, we did not include descriptions of courses or programs intended to train professional game designers.

Making Games as a Context to Learn Computer Programming

The earliest attempts at using computer game design within educational settings took place during the early 1990s. At that time, the software publishing industry was experiencing an unprecedented growth that would last nearly a decade and produce some of the most successful businesses in history (Cuban, 2001). It should come as no surprise that early attempts at using videogames for learning were heavily invested in helping students acquire the math and programming skills required for work in software production.

The Logo programming language developed by Seymour Papert, Wallace Feurzeig, and Daniel Bobrow in 1968 became a centerpiece of this line of research and educational strategies. Papert conceived Logo as an environment through which children could learn to "talk" to a computer (Papert, 1980). This conversation was carried out as children entered instructions that the computer would interpret and then enact through "turtles" on the screen (e.g., the turtle moves or draws a line). Various versions of Logo have been developed during the past several decades, with new capabilities and features. Interest in Logo waned in the late 1980s, but in the 1990s, new applications were developed, spearheaded by Mitch Resnick at the MIT Media Lab, including StarLogo, a specialized version of the Logo programming language for exploring the properties of decentralized systems.

Although Logo can be used for many different purposes, two versions have emphasized game construction as a core activity: MOOSE Crossing and StarLogo TNG.

MOOSE Crossing: Learning to program by collaborative world building. In the early 1990s, Amy Bruckman, at the time a student of Mitch Resnick, set out to create a multiuser virtual environment that would be accessible to children. The result was MOOSE Crossing, a text-based virtual world designed to allow players to collaboratively construct game-like elements such as virtual spaces and nonplayer characters and in the process learn reading, writing, and programming skills (Bruckman, 1997). MOOSE Crossing was based on a new programming language called MOOSE (based on the original MOO programming language) that Bruckman designed in collaboration with some of the original Logo designers. MOOSE was intended to be easily learned and usable by children, and a key design feature was the use of English-like syntax to leverage children's existing language knowledge. There was also a client program called MacMOOSE, which was designed to make the programming interface less awkward (Bruckman, 1997). For each object that users create with MOOSE, they write a combination of text and computer code to describe the properties and behaviors of the object.

The goal of MOOSE Crossing was to create "a context for learning through community-supported collaborative construction" (Bruckman, 1997, p. 3). Bruckman conducted a 2-year ethnographic study of children who used MOOSE Crossing in an afterschool program as well as children who participated online from home. Overall, she concluded that the collaborative design and construction process that took place in MOOSE Crossing helped facilitate the children's acquisition of programming skills, and in turn, the opportunity to collaborate and share knowledge helped to form a supportive community. She attributes the successes of MOOSE Crossing to the community's ability to provide:

- role models;
- situated, ubiquitous project models;
- emotional support to overcome technophobia;
- technical support; and
- an appreciative audience for completed work (Bruckman, 1998)

However, in a later study of 50 participants' actual scripting ability, Bruckman, Edwards, Elliott, and Jensen (2000) found considerable variation in participants' learning, with a small number of children who spent large amounts of time in MOOSE Crossing and who gained considerable programming ability, while the majority spent much less time and learned only the most basic concepts. The authors concluded that an incentive for participation, such as a "merit badge" system, might help to encourage children's achievement without spoiling the open-ended, self-motivated nature of the learning environment (Bruckman, 2004; Bruckman et al., 2000).

MOOSE Crossing is no longer accepting new participants, and its text-based environment is now dated in comparison to the graphical virtual worlds now widely available. MOOSE Crossing did not involve children in the design of a "game" per se, though as Bruckman points out, the environment resembled a text-based adventure game (Resnick, Bruckman, & Martin, 1998), and MUDS certainly were a precursor of contemporary massively multiplayer online role-playing games. The value of MOOSE Crossing lies primarily in Bruckman's documentation of the role of community in fostering learning and her insights into the uneven distribution of such learning among participants. The importance of "situated, ubiquitous project models" (Bruckman, 1997, p. 126) is also worth noting. Virtually every object present in MOOSE Crossing was created by participants and thus provided inspiration for new users. In addition, the objects' creators were often readily available to answer questions and provide guidance (Bruckman, 1997, 2000). This approach offers one example of the value of access to peer tutors and diverse examples of digital creations, in an informal educational context, at the same time suggesting the limitations of entirely self-directed learning environments.

StarLogo TNG: Making games to motivate novice programmers. Throughout the years, Logo has undergone major changes to provide greater accessibility and functionality to learners of programming languages. One of the most fundamental changes has been the extension of the programmer's ability to issue commands to more than one turtle (the core building blocks of the system) at the same time. Newer versions of Logo, such as StarLogo (Resnick, 1994) and NetLogo (Tisue & Wilensky, 2004), allow the learner to create contained simulations representing complex phenomena that can be understood by observing the interactions between the turtles at a systematic level (e.g., a model of population growth dynamics) (Tisue & Wilensky, 2004).

StarLogo TNG (TNG stands for "The Next Generation"), one of Logo's latest incarnations, has a somewhat different goal than its predecessors. According to the project Web site (http://education.mit.edu/starlogo-tng/), although TNG retains the overall purpose of serving as a tool to create and understand simulations of complex systems, it has the more specific goal of making programming easier to learn and more appealing by incorporating tools to make games in 3D environments.

TNG differs from prior versions of StarLogo in adopting a visual approach to programming. TNG incorporates two major innovations from previous versions of StarLogo.

Programming is accomplished with "blocks" rather than text commands, and the blocks are colored according to their function (e.g., movement, traits, interface), assisting students in understanding similarities among functions. The blocks are puzzle-piece shaped and can be put together only in meaningful ways, making it easier for students to create functional programs. The second innovation, Spaceland, offers a 3D worldview as well as first-person view (through the eyes of a turtle avatar) (Wang, McCaffrey, Wendel, & Klopfer, 2006). Work with StarLogo TNG is ongoing and has included an afterschool program intended to introduce participants to basic programming concepts (Wang et al., 2006), a set of math lessons using TNG (available at http://education.mit.edu/starlogo-tng/Math/index.html), and the design of instructional games by teachers (Klopfer & Yoon, 2005).

In a pilot project focused on gaming, Wang et al. (2006) introduced StarLogo TNG to a group of eight students (Grades 7 to 9 with diverse backgrounds) in an afterschool class. The class met once a week for 90 min, and by the end of five sessions, students were able to create an interactive maze that provided a first-person view and score keeping. The researchers noted that the initial instructional approach moved from largely teacher directed, as the instructors introduced the basic set of programming blocks, to more student directed as the participants created simple games, with new information provided as needed to scaffold the incorporation of new game elements. Preliminary findings indicated that engaging in a game-making task with this authoring environment did appeal to this group and increased their interest in programming. Feedback from this and other applications is being used to modify the StarLogo TNG software.

Work with StarLogo TNG indicates the value and viability of using simplified programming tools in game-making activities. The initial implementation suggests how constructionist methods can follow from a more instructionist approach. Lesson overviews (available at http://education.mit.edu/starlogo-tng/resources/) indicate how the instructors shifted from presenting information to providing guided activities to open-ended game-making activity. Although game design was not overtly "taught" (and even programming concepts were implicit rather than explicit in the presentation materials), design principles were inherent in how the software features were presented; for example, participants were taught to make a maze, and then, to add interest to the game, the concept of bonus blocks was introduced. It would not be difficult to develop a more overt process for supporting learners' design learning around the use of StarLogo TNG.

Making Games as a Way to Interest Girls in Computer Programming

Women continue to enroll in formal computer science education courses in much smaller numbers than men (Dean, 2007). The National Science Foundation (NSF) has funded hundreds of educational projects intended to support the participation of girls and women in computer science and related fields through targeted funding such as the Program for Gender Equity in Science, Mathematics, Engineering, and

Technology. In this section, we describe two such NSF-funded projects that used game making as a core activity: Rapunsel and Girls Creating Games (GCG).

Rapunsel: Programming in a girl-friendly game environment. The goal of the Rapunsel Project (Real-time Applied Programming for Underrepresented Students' Early Literacy) was to help girls learn computer programming in a more "girl-friendly" environment (see Rapunsel, n.d.). The project was funded by NSF from 2003 to 2006 and was intended to be a way to ameliorate the shortage of women in technology-related careers and degree programs by designing an appealing game environment that would motivate children, particularly girls, to master programming concepts (Flanagan, 2005). The designers characterize the project as "activist" given their ultimate goals of promoting "equity, empowerment, and access to technology" (Flanagan, Howe, & Nissenbaum, 2005b, p. 758).

Most available descriptions of the project focus on the development rather than use of the Rapunsel software, particularly how the designers attempted to make explicit their own values as well as incorporate the values of potential users into the game environment (e.g., Flanagan, Howe, & Nissenbaum, 2005a; Flanagan & Nissenbaum, 2007). Based on these analyses of values, Flanagan et al. (2005b) report, the focus of the project shifted from "how to teach programming" to "how to create an immersive, socially oriented game environment in which programming was an important and valued activity, central to achieving goals of the game" (p. 754).

Rapunsel is a game in itself, different from software such as StarLogo TNG, which are tools for creating games (and other things). The core game mechanic in Rapunsel is dance; the user is assigned a central character that can be programmed to move, dance, and behave in various ways. Players are introduced to the Java programming language through guided exercises that allow them to create new moves and dance sequences. The reward structure accommodates both competitive and cooperative play. Players can compete with each other in dance competitions, or they can choose to collect and swap codes, decorate their "homes" in the game, create music to accompany dances, and contribute code to a shared Library, where it can be used and rated by other players. They are rewarded for completing lessons that introduce increasingly complex programming concepts and procedures as well as for creating more sophisticated and original codes (Flanagan et al., 2005a). Thus, rather than using programming to create entire games, players are expected to learn new skills through a combination of interactive lessons and modding or creating new within-game actions and content.

Details of the project evaluation are difficult to locate. According to presentation notes from Mary Flanagan, one of the lead investigators, 90 sixth graders in an urban school voluntarily participated in a study of the game's outcomes. Pre- and postsurveys were used to ascertain whether playing the game affected general self-efficacy, self-esteem, computer self-efficacy, programming knowledge, and confidence level about programming knowledge. Data were also collected through interviews, program tracking, and

blogs. According to Flanagan, playing Rapunsel increased female students' sense of self-efficacy; self-esteem and programming-related self-efficacy increased after playing the game for both boys and girls (see Grand Text Auto, 2007).

The Rapunsel software is not readily available, and it does not seem to be currently in use. This is unfortunate, as Rapunsel represents an interesting variation on the notion of using game design to teach programming. Because students are not actually making games, it might seem inaccurate to characterize the focus as game design per se. However, creating new material for existing games, or modding games, is frequently a preliminary step toward designing games anew and might be more appropriate for novices. In addition, the game structure provides a perhaps more clear set of goals and compelling context for learning new skills than, say, a task to construct a simulation or a game in its entirety. Last, using the core mechanic of dance offers a distinctive alternative to the typical emphasis on shooting or maze-type game designs.

GCG: Programming interactive stories. GCG was another 3-year (2002 to 2005) demonstration project supported by NSF and aligned with the NSF's core value of encouraging young women to build leadership and to pursue advanced education and careers in information technology (IT). The more specific goals of the program were

1. to increase girls' interest, confidence, and competence in IT and to encourage them to pursue educational and career paths that would keep them in the "technology pipeline," and
2. to help girls develop resiliency toward gender barriers through a variety of strategies (YouthLearn, n.d.)

The rationale for using game construction as a focal activity was that games are an attractive way to develop interests and engagement in IT, and games are usually not designed for or marketed to girls and young women (Denner, Werner, Bean, & Campe, 2005).

An extensive curriculum was developed as part of the GCG program, with 23 sessions, each lasting 2 hours. Detailed lesson plans and guides are still available on the project Web site (see http://programservices.etr.org/gcgweb/); clearly, how the girls were introduced to programming and game construction was as important as the actual content. There are very explicit instructions throughout the teacher's guides on how to foster a supportive, nonjudgmental environment. The instructional model was based on a model proposed by the Cognition and Technology Group at Vanderbilt University (2003), which includes the principles of learning by design, scaffolding and modeling, collaborative learning, and identity formation (Denner, 2007; Denner et al., 2005). Program participants were taught to create computer games with Macromedia's Flash program. The concept of "game" was somewhat loosely applied; the games were actually interactive story narratives in which players select a path at key decision points in the story to create their own series of events. Because collaboration was stressed, the

participants worked in pairs to write stories about themselves or their interests and produce Flash games to tell the stories. They also play-tested each other's games and viewed final versions in a culminating "gallery walk" in which the pairs demonstrated their games to other participants. The detailed instructional guides available on the project Web site offer a very structured, step-by-step approach to creating the stories and the use of Flash; for example, flowcharts are provided and sequences are specified for stories. Design is addressed briefly, using the metaphors of creating a movie or sculpting with clay. Examples of the participants' games are posted on the Internet and reflect a relatively standard format with variations in artwork and storyline.

The GCG program was implemented in its entirety six separate times over 2 years, after school and during the summer, with 126 middle school-age girls (Denner, 2007). Extensive survey data on program outcomes was collected from 90 girls, with a comparison group of 71 students, along with qualitative data from a smaller sample. The findings indicated significant changes in participants' perceptions of their computer knowledge and computer skill level as well as in social support for using computers. However, there were no significant differences in the participants' stereotypes of computer workers, intentions to take computer courses, or confidence in their computer skills. The project directors attribute this lack of change to the girls' relatively high initial confidence levels and their lack of endorsement of gender stereotypes. When asked what they liked least about the program, participants most frequently reported the amount of direct instruction and need to work with a partner (Denner, 2007).

The GCG program is noteworthy for the extent of guidance provided to participants and teachers. Although the interactive story model might seem overly narrow and prescriptive, such a structured approach can be useful for novice game makers to reduce the number and complexity of choices available to them. Similar to the StarLogo pilot project, this instructivist approach preceded a somewhat more open-ended constructivist activity. The predetermined narrative framework allowed the participants to acquire basic skills with Flash and some opportunity for creativity in the content of their stories and their use of graphics. The use of Flash is notable for giving the girls experience with a program used by real game designers (not a program designed for "kids"), which has applications beyond game design. As with Rapunsel, the NSF program funding approach does not seem to lend itself to sustainability, though curricular materials are still available online. Also funded by NSF, The Girl Game Company, an extension of GCG, is intended to involve rural, Latina, middle-school girls in creating Web-based digital games about life in outer space. The project will use the online virtual world Whyville combined with the SETI Institute's astrobiology curriculum (see ITEST: Learning Resource Center, n.d.).

Making Games as a Route to Learning in Other Academic Domains

The software and instructional approaches presented in this section are characterized by their emphasis on using game making as a way to enhance learning in content areas such as mathematics and history.

Making games to elicit mathematical thinking. Yasmin Kafai pioneered one of the earliest uses of computer game making in education. *In Minds in Play: Computer Game Design as a Context for Children's Learning*, Kafai (1995) describes an educational intervention in which 16 fourth-graders were introduced to Logo and for 6 months were given the task of producing games to teach fractions to younger students. Students who created games performed better on average than a control group on measures of fraction knowledge and Logo programming sophistication. Kafai (1996) also found some intriguing gender differences in the design of the games. For example, girls tended to locate their games in realistic places, while boys tended to use fantasy spaces; girls tended to use nonviolent feedback, while boys tended to have the game end and the player killed in response to an incorrect answer. She noted that the boys were far more likely to play games regularly, while the girls reported less game play and less interest in gaming, particularly because of a dislike of their themes and violence. It is not surprising that the boys' games more often were based on or showed the influence of popular commercial games.

Although students significantly increased their understanding of fractions, Kafai, Franke, Ching, and Shih (1998) later noted that almost all students created games in which the game ideas and fraction content were unrelated. Subsequently, the researchers conducted two studies that included more explicit attention to what the researchers describe as "conceptual design tools" (Kafai et al., 1998, p. 157) and their impact on the integration of mathematical content with the game design. In the first study, teams of elementary school students again designed computer games to teach fractions to younger students in a 50 minute afterschool session. The session included three phases: (a) sharing, (b) identification of an "emergent design tool," and (c) posing a challenge. Using fractions to describe a real-life situation in the students' games was a design parameter that emerged from students' discussion and initial game designs. In the second study, preservice teachers participated in three sessions over the course of 6 weeks. Each session addressed the following: (a) initial game design, (b) introduction of a conceptual design tool, and (c) extending conceptual design tools (i.e., introducing a new design tool that was used to modify prior game designs).

Conceptual design tools, as identified by the researchers, consisted of guiding questions or challenges that were intended to guide and focus the participants' thinking about design. These included using fractions to describe a real-life situation and creating a game without asking questions; with the preservice teachers, the design tools included a blank page of empty computer screen frames on which they could draw and annotate their game designs, the challenge "Can you create a game without asking questions?" and an example of a dynamic representation from the students' game designs (Kafai et al., 1998, p. 157).

A useful aspect of this research is the analytic framework that the researchers applied to the game design. They analyzed the games in relation to three broad features: (a) integration of fraction content and game design, (b) types of fraction representations available within the game, and (c) consideration of user thinking. Overall, the introduction

of design tools enhanced both student and preservice teachers' games in all three areas. The researchers noted that all participants started with quite narrow conceptions of educational games (e.g., games as drill and practice) and did not make use of their informal knowledge. The design tools encouraged them to think more expansively about both the design of their games as well as what they wanted players to learn. One implication stressed by the researchers was the importance of discussions among the teachers and students as a means of improving the design process and outcomes.

Kafai et al.'s (1998) work demonstrates the need for explicit attention to design in educational uses of game making. Even the relatively simple conceptual design tools introduced by the facilitators made observable differences in the games designed by participants. Although the analytic framework they derived from analyses of these games is specific to fractions, it can serve as a starting point for the development of similar ways of analyzing games developed in other content areas and for other educational goals.

Learning history by game modding. One of the popular trends in commercial videogames during the past 10 years has been for game studios to license the software upon which their videogames are built to allow others to create their own modded (modified) versions of the game. This software commonly consists of a game engine (the core software system upon which the game runs) and a set of mod tools (level and graphic editors and software code libraries) that allow modifications to the original game ranging from surface changes in appearance to deeper changes in rules and game play.

The Civilization games, including the most current version, Civilization IV, have been popular with educators because of their incorporation of historical facts and concepts associated with school history curriculum. Civilization is a turn-based strategy game in which the player assumes the role of a nation's leader, making strategic decisions that will affect its historical evolution from a primitive culture to an advanced civilization through the centuries. The World Builder mod tools in Civilization IV allow players to customize the game and create their own historical scenarios (including the countries, events, characters, and rules of the game) in which the game will be played. Civilization has been modified by educators to teach history; a prominent example is the History Canada Game, a million-dollar project funded by Telefilm Canada and supported by Canada's National History Society, which is a Civilization III mod designed to help students learn about Canadian history.

Adopting a more constructionist approach. Squire, Giovanetto, Devane, and Durga (2005) designed an afterschool program around Civilization III in which participants moved from playing historically accurate scenarios (created by the researchers) to using mod tools to create their own scenarios. The researchers' findings indicate that, as more experienced players moved to designing mods, they were able to understand the game

as a system of designed rules (particularly in the multiplayer scenario) to use sophisticated language to communicate their knowledge of these designs to other players and to use this knowledge to their advantage during play.

A contribution of this approach is the integration of game play and game making. Squire et al.'s findings suggest that by first playing games and reflecting on features of these games, young people can first learn to see games as what Gee (2003) calls designed artifacts. In addition, this type of game modding does not require any programming and builds on young people's more intrinsic interest in modifying games that are already familiar and engaging to them. However, moving from player to designer, at least in Squire's work, can take a considerable amount of time (estimates upward of 30-plus hours). Still, the Civilization game allows for the creation of quite sophisticated and complex scenarios, and thus, this extended learning curve might be appropriate and necessary.

AdventureAuthor: Narrative development through game making. AdventureAuthor is a platform developed by Judy Robertson at the University of Edinburgh, based on the Neverwinter Nights Aurora toolkit. Taking advantage of the story-centered nature of the Neverwinter Nights role-playing game, AdventureAuthor was designed to promote the acquisition of literacy skills (with an emphasis on storytelling) through game design by children (Good & Robertson, 2003). With AdventureAuthor, students produce original storylines in a game form, using the Aurora toolset to modify the environments, characters, and story of the game. AdventureAuthor removes the need for learners to learn programming to design their games, with the tradeoff that players can only make game genres restricted to the Neverwinter Nights model.

Through a series of workshops and pilot testing with children, Robertson and her colleagues have made modifications in AdventureAuthor to support the development of narrative skills (Good & Robertson, 2003, 2004). By documenting and analyzing how students design games with the authoring tool, Robertson's work yields valuable insights into the way that children produce storylines in computer games (Robertson & Good, 2005, 2006). This work also illustrates how features of the Neverwinter Nights interface were modified to, for example, explicitly represent a story's plot or to encourage users to create personalities for characters. In addition, they describe how resources were integrated into workshops to support the design of narrative features; for example, a screenwriter was recruited to teach participants how to write dialogue (Robertson & Nicholson, 2007).

Robertson and her colleagues' work is perhaps most notable for the iterative process she used to modify AdventureAuthor to engage learners in a more explicit design process, which now includes exploratory play, idea generation, design, implementation, evaluation, and testing. In a recent article, Robertson and Nicholson (2007) describe plans for the development of a built-in "Designer's Notebook" that will provide direct software support for design. They note that

> children tend towards ambitious designs which are not always easily achievable, as they not only require complex scripting and programming work beyond their abilities, but also the intellectual discipline to think through every aspect of the design. By providing wizards that directly scaffold specific designs tasks, we can help the user to clarify their designs until they are in a workable state.

Although AdventureAuthor's focus on narrative does not encompass game design in a broader sense, the design process she has evolved certainly can be applied more generally. In addition, this focus (and perhaps the tool) is readily transferable to more traditional school-based activities, such as creative writing (Robertson & Nicholson, 2007).

Understanding Design Concepts Through Making Games

The examples in this section suggest how game making can be a basis for acquiring explicit understanding of some aspect of game design itself. The Playground Project investigated children's understanding of rules, whereas Stagecast Creator and Game Maker are software programs designed specifically to allow novices to create computer games.

Toontalk & The Playground Project: Understanding rules. Toontalk (TT), first released in 1998, is an interactive, animated programming environment for children. Ken Kahn, its creator, was inspired by his exposure to Logo while he was a graduate student at MIT, and although similar in philosophy and goals—to give children "computational thinking tools" (Kahn, 2001)—TT has quite different features. According to Kahn, the fundamental idea behind TT is to replace computational abstractions by concrete familiar objects. Rather than rely on text or pictures to represent programs, in TT, programs are built in an animated virtual world. Programs are constructed by training robots to manipulate tangible program elements such as birds and trucks. For the user to manipulate the environment, a virtual hand is used to control a number of different tools: a magic wand for copying objects, a vacuum cleaner for erasing and removing things (called sucking and spitting), a bicycle pump for changing the size of objects, and notebooks to store objects. Kahn suggests that TT features resemble those of a videogame and even describes it as a game in some publications, because of its graphics and animation, a virtual world to explore, animated characters to interact with and get help from (Kahn, 2004).

While TT, like Logo, can be used for many different educational purposes, including learning programming, Kahn has stressed repeatedly that although learning programming is valuable, the process of building, running, and debugging programs is central to TT. The TT Web site (ToonTalk, n.d.a) likens this process to playing an adventure game, suggesting that TT makes it fun for kids to build things, not just to play with the resulting creation. The site elaborates five types of thinking

that TT can foster, including problem decomposition, component composition, explicit representation, abstraction, and thinking about thinking (ToonTalk, n.d.b).

In addition to describing TT as a game, Kahn (2001) argues that TT is a better tool than Logo for making most kinds of games and simulations because of TT's underlying concurrency (multiple processes can be executed simultaneously) and its extensive support for giving behaviors to pictures. Perhaps the most extensive application of TT to game making in education was the Playground Project. The Playground Project was funded through a European Union initiative from 1998 to 2001 and involved a partnership among schools and universities in the United Kingdom, Sweden, Portugal, and Slovakia. The project's goal was the design and evaluation of a computational "playground" where children ages 4 to 8 could play and create their own games. This space contained game-building tools, games already built by children or developers, and subelements of a game, such as game objects, rules, parts of rules, and scenery. TT was one of two programming languages used in the project; the other was Imagine, a graphical version of Logo. Researchers partnered with local schools to develop playgrounds and study children's games, the game creation process, and learning outcomes. A core objective for the project was to have children learn, through game design, about rules, the different ways they can be expressed, how they can be changed, and the implications of modifications (see Institute of Education University of London. n.d.a).

The Playground Project generated a number of studies (e.g., Adamson, Hoyles, Tholander, & Noss, 2002; Goldstein, Noss, Kalas, & Pratt, 2001; Goldstein & Pratt, 2001; Hoyles, Noss, Adamson, & Lowe, 2001; Tholander, 2002) in the form of detailed case studies of how children of different ages create, talk about, and change rules in the process of making games. The Project's final report (Institute of Education University of London. n.d.b) provides a detailed overview of these studies and their findings. One useful observation from these studies is that rules function at different levels in games and must be understood in relation to their purposes. For example, there are rules for player behavior and rules that define conditions and actions in the game system (Adamson et al., 2002). The researchers found that children were most likely to describe player rules or constraints and had difficulty articulating system rules, or those that defined, for example, the core mechanic or the game environment. Overall, the researchers were able to document an increase in the children's ability to create and describe more system-related rules. However, the children were not always able to describe a rule they might have programmed correctly or predict the implied consequences of a formal rule they have programmed (Hoyles et al., 2001). Increased ability to identify rules did not result from unstructured game making and discussion; the participants were engaged in highly scaffolded activities, for example, modifying only one or two rules and predicting the consequences.

The Playground Project is well documented and provides very concrete examples of how young children make and talk about rules in games. The young age of the children limited the sophistication of the games they created, and most of the games used in the

research were relatively simple; however, they were discernable as games, unlike what children created in some projects in this review. Tholander (2002) points out that the concrete nature of the programming environment offered a crucial affordance in allowing the children to "talk" about programming elements without having developed the verbal skills of speaking about these elements. However, the language and actions they learned to use were perhaps too specific to the TT environment, potentially limiting transfer to other contexts and tools. The researchers also noted that their involvement played a key role in prompting and supporting the participants' learning, in ways that cannot be built into software alone, and that integrating use of the software with adult and peer support is crucial for learning (Tholander, 2002).

StageCast Creator: Making Games to Enrich Instruction A direct evolution of Apple's Cocoa/KidSim (Cypher & Smith, 1995) programming-by-example environment, Stagecast is a commercial software environment that lets kids develop simple games and simulations without programming. Creating a game in Stagecast involves the use of "agents." These agents are characters that can be edited and animated to have interactions with each other and with their surrounding microworld. To get agents to engage in certain behaviors (e.g., moving three pixels to the left if no other character is present), a set of visual rules can be assigned to them through a rule editor. Stagecast was conceived as a way to introduce children to programming without requiring them to understand any complex syntax. Instead, the visual rules used to define agent behaviors rely on concrete visual representations that were presumed to be easier for novices to grasp. The finished products can be uploaded onto a Web page and played over the Internet (Smith & Cypher, 1998; Smith, Cypher, & Tesler, 2000). Of the software reviewed in this article, Stagecast appears to be the easiest to learn and use but, at the expense of flexibility and power, letting learners create only a very limited variety of games.

The principal avenue for learning how to make games in Stagecast is following a set of tutorials. The tutorials aim to facilitate learning to use the Stagecast tool set as well as some fundamental concepts, such as associating character actions with rules. The tutorials consist of screens with some "broken" features that must be fixed to make them functional and built-in step-by-step instructions. Stagecast allows even young children to produce interactions between agents very easily; M. Habgood, Ainsworth, and Benford (2005) had 40 children between 7 and 11 years old develop their own games using Stagecast and showed evidence that even the youngest were able to create some form of a computer game.

Stagecast Software, the company behind Stagecast, has actively promoted it to educators as a means of achieving a wide range of educational goals, ranging from higher order thinking skills to deeper understanding of content across the curriculum (see http://www.stagecast.com/school.html). A variety of sample lessons are accessible through the Stagecast Web site and related sites, some of which include explicit

attention to elements of game design. For example, the materials developed and used by M. Habgood et al. (Game Learning, n.d.) include lesson plans along with simple templates for technical, level, and style design.

The visual rule definition system in Stagecast makes it a tool with a relatively shallow learning curve for novice designers. The tutorial is an excellent example of how to support users in mastering not only the tools but also the concepts underlying them. However, M. Habgood et al. (2005) observed that even after 6 weeks of instruction on the use of Stagecast, young learners could only achieve a low level of sophistication in the design of their games. When given the task of developing a game aimed at teaching some form of academic concept through game play, most of the games children made replicated rule systems and mechanics from entertainment games, and the learning content was only integrated as an afterthought, similar to Kafai et al.'s (1998) findings. Stagecast, while actively promoted as a game-making tool, still lacks sufficient attention to fostering an understanding of design principles that might improve the quality of user-created games.

Game Maker: *Making games as the goal*. Game Maker is a programming tool designed to facilitate the creation of 2D and 3D games by novices. The Game Maker interface follows the Microsoft Windows interface design style, and its look and feel is very similar to Microsoft development environments such as Visual Studio. Reflecting object-oriented software design, Game Maker allows users to make games by defining objects such as rooms (game screens), backgrounds, sprites (animated characters or objects), and sounds, which can be combined into game levels. These objects have properties (e.g., the color of a background) and behaviors (e.g., a sprite's ability to move) that can be customized by users. Game Maker uses an event-driven approach to the production of games, where events are "important things that happen in a game, such as when objects collide or a player presses a key on the keyboard" (J. Habgood & Overmars, 2006, p. 11). In Game Maker, users can give objects actions in response to these events and thus create a complex set of interactions. Basic games can be developed in Game Maker without any programming through a point-and-click interface and elements such as selection menus and check boxes for different game attributes. For more advanced users, a scripting language similar to the C programming language is available within Game Maker as well (Overmars, 2004).

A robust community has evolved around the use of Game Maker, and the Game Maker site (Game Maker Community, n.d.) has an extensive set of very active forums along with many games, a wiki, additional resources, and tutorials. Although the forums seem to be dominated by novice or aspiring professional game designers, technical information and support could be a resource for educators and students alike. The tutorials address making various game genres, ranging from maze games to first-person shooters, though the more sophisticated game tutorials require programming. However, the content of the tutorials focuses primarily on technical strategies, such as how to create a chat system for multiplayer games rather than the

overall design of each genre. The wiki includes a section with information for teachers, including course materials for primary through postsecondary education. There are active teacher communities around the use of Game Maker, particularly in Australia (see http://www.gamelearning.edu.au).

J. Habgood and Overmars (2006) have encouraged educators to think broadly about potential applications, arguing that Game Maker allows students to concentrate more on game design rather than the technical aspects of getting the game to work. However, little published research is readily available on how Game Maker has been used by educators; a scan of the online forums and posted materials suggest that it has primarily been used by teachers in the context of computer science education.

Game Maker embodies a number of important design principles used by game designers. The object-oriented approach is one of the most popular software development approaches in the industry, especially for 3D games where languages such as C++ or C# are in many ways the standard for game engines. The companion textbook *The Game Maker's Apprentice* nicely complements Game Maker by addressing at a basic level design principles such as rules, mechanics, challenges, and player goals. Perhaps most significant is the active user community, which supports considerable informal learning and might serve as a model for more deliberately educational efforts to use game making for learning.

Discussion

Our review makes it clear that the majority of efforts to incorporate game making into education have emphasized learning computer programming as a rationale for using game making as well as the design of specialized software. Given the emphasis on programming, most software tools we reviewed were created to simplify programming concepts and methods without explicit consideration of how such tools would scaffold game design in particular. Even Game Maker as a stand-alone tool lacks a structure for supporting players in thinking through specific aspects of game design. Robertson et al. (Robertson & Good, 2005, 2006; Robertson & Nicholson, 2007), with their plans to create of a virtual Design Notebook, seem to have given the most attention to integrating support for a design process into the tool itself. Although there have been efforts to support design thinking through activities around the software, such efforts tend to be rather narrow in focus.

This stress on programming is not surprising given the history of (digital) game making in education. The impetus behind the original versions of Logo, for example, was to give children the opportunity to understand and interact with computers at a time when computers were not widely accessible and when mastery of programming concepts was essential for even basic applications. Even now, this emphasis on learning programming reflects broader efforts to make computer science education

more appealing to young people, a challenge with even more urgency given the decline in computer science enrollments at the postsecondary level. It is also not surprising that the teachers most likely to be interested in and feel competent in using the available software programs would be in computer science.

Design might also have been more intentionally left unaddressed. We can infer that at least some of the past approaches have assumed that design is an intuitive, creative act, an assumption that gives little or no attention to the very explicit approaches to game design that are now readily available (e.g., Adams & Rollings, 2006; Fullerton, Swain, & Hoffman, 2004; Salen & Zimmerman, 2003). In many cases, even the concept of a "game" was very loosely defined and applied to a wide variety of digital constructions. It is perhaps not surprising then that examples of what students created using these approaches often seem to reflect a limited sense of good (game) design.

Another assumption may be that design requires more sophisticated ways of thinking than most young people (or teachers) are prepared to undertake or, alternatively, that game design thinking specifically is not particularly relevant or valuable for anyone aside from game designers. We think there is plenty of evidence to the contrary in regard to the first point, though we acknowledge that current understanding of how young people think about design in general and game design in particular is quite limited and a topic sorely in need of further research. (We note that there is a body of literature on the use of design in science learning as well as in the training of design professions such as architects; this literature may have some relevant information but was beyond the scope of this review.) In terms of the second point, we can offer several arguments in support of the more general value of "thinking like a game designer" for those who have no aspirations to design games as a career. Although learning computer programming is no doubt an important goal, we would argue that design thinking should be recognized as an important goal of game making in education.

Our first argument concerns student motivation. Assuming that game making is what motivates students to learn programming, educators have used games as the "carrot" to be obtained after going through a sometimes arduous process of mastering programming tools. Yet if the games they ultimately create are not very good, the students may not have much motivation to continue making games or using their newly acquired programming skills. Indeed, the literature lacks any data on whether young people who have participated in game-making activities continue on to acquire more advanced skills or even sustain the skills they have already developed.

An alternative approach is to provide tools and strategies that enable learners to make games quickly and then support a transition to using programming tools to go beyond limitations of the game-making software. If participants are not given enough guidance to create games that are fun—or even playable—it is likely that they are not going to be motivated to take the next step into learning programming. Game Maker and some other commercial tools can support this approach, but even these tools tend to focus on game components rather than design and have a considerable learning curve. If game making

is foregrounded, it becomes crucial that principles of game design be explicitly addressed. Furthermore, students who acquire an understanding of design principles before they learn to program may well learn to program more rapidly and more effectively because they have a context in which to understand coding issues (Claypool & Claypool, 2005).

Our second argument is that design skills and knowledge are increasingly recognized as crucial to software engineering and other computer science/programming-related work. More than 20 years ago, software pioneer Fred Brooks argued that cultivating great designers was essential for the development of reliable software, though perhaps the most difficult task for educators and employers (Denning, 2004). Teaching game design, as an instance of software design, would result in a broader introduction to computer science (Claypool & Claypool, 2005). Students gain an appreciation of how programming accomplishes its goals when they learn about the design of a program. As Denning and Martell (2007) put it, great designers can "see" large systems at all levels of detail in their heads and can transform their vision into working code very quickly. Designing software involves concepts and strategies such as levels, prototyping, understanding user interests and contexts, and understanding systems, all of which may be illustrated and practiced in the context of making games (Denning, 2007; Denning & Martell, 2007).

Our third argument hinges on the value of learning to engage in professional practice, in particular learning to think like a professional. Shaffer (2006) has argued for the value of what he calls "epistemic games," games that introduce young people to the ways of thinking of creative professionals. Shaffer states that

> creative professionals learn innovative thinking through training that is very different from traditional academic classrooms because innovative thinking means more than just knowing the right answers on a test. It also means having real-world skills, high standards and professional values, and a particular way of thinking about problems and justifying solutions. (Williamson Shaffer, 2006)

This stance does not imply that game making be approached as vocational training for budding game designers. It does point to the potential value of attending to not only the technical aspects of game design (i.e., learning to use tools such as programming or even discrete concepts such as rules) but also to broader dimensions of "thinking like a game designer." These dimensions would include understanding the specialist language of game design, understanding ways of evaluating games through the lens of a particular value system, and broadly speaking, taking on the identity of a game designer.

Fourth, and perhaps most important, as we noted in our introduction, "design thinking" is increasingly viewed as a foundational capacity for engagement in professional practice beyond more obvious examples such as software engineering. Indeed, as Gee (2007) argues, design thinking can be viewed as a way of viewing the world:

> games designers have to think about how objects and actions (their nouns and verbs) combine to get effects from players when specific goals are assumed or given. In this sense, game design is a core way of thinking about the world. . . . Indeed, in our daily lives, when we are thinking proactively, we look at the world as if we could design the objects and actions around us to achieve certain goals—we "game" it. Game design is, thus, akin to the design of social life.

Design thinking, as the ability to think about—and influence—social systems, can thus be a precursor to learning how to negotiate the complexities of modern life.

So how can we shift from a focus on game making to game design in education?

Educators need to start by becoming more familiar with theories of design and game design in particular. There are multiple perspectives on game design (see Salen & Zimmerman, 2003, for an overview) and clarifying the particular stance toward design that will inform educational strategies is an important first step. Second is the development or selection of tools and strategies to support students' engagement in design activities. Fischer and Lemke (1987-1988) make a distinction between construction kits and design environments that can be useful in this process. As he suggests, construction kits are not sufficient; learners need to be immersed in environments that assist in the design process through features such as models, critics, and suggestions. Furthermore, there are many courses intended to prepare professional game designers, and although their content may not be appropriate for young students or more general student population, they offer useful examples of content and instructional strategies. In addition, there are examples of instruction based on more general design, such as the learning-by-design approach (e.g., Kolodner et al., 2003) that could inform more game-specific curricula.

Third is the mapping of design knowledge and skills to other valued domains in school and beyond. Although we found some suggestions of how game making might address academic skills, a focus on game design opens up a wealth of potential applications. However, as Tholander (2002) notes, we have evidence that without explicit scaffolding, children do not transfer what they have learned from one learning domain to another and such scaffolding needs to be incorporated into activities intended to promote connections to other domains.

Last, to inform these efforts, we need a better understanding of design thinking itself, how novices think about game design, and how they develop more sophisticated and useful understandings and practices. There may be different types of learning trajectories, based on students' prior experience with games, their gaming preferences, and the contexts of their learning.

Kafai (2006) calls for educators to investigate all potential ways to use games for learning, both playing and making games. We would add that educators should explore the full educational potential of making games for learning, which includes explicit attention to design. Why continue to overlook such a rich and valuable aspect of game-based learning?

References

Adams, E., & Rollings, A. (2006). *Fundamentals of game design* (Game Design and Development Series). Saddle River, NJ: Prentice Hall.

Adamson, R., Hoyles, C., Tholander, J., & Noss, R. (2002). *Collaborative change to rules of the game: From player to system rules*. Unpublished Proceedings of Computer Support for Collaborative Learning 2002, Boulder, Colorado.

Barab, S., Thomas, M., Dodge, T., Carteaux, R., & Tuzun, H. (2005). Making learning fun: Quest Atlantis, A game without guns. *Educational Technology Research and Development*, *53*(1), 86-107.

Bruckman, A. S. (1997). *MOOSE Crossing: Construction, community, and learning in a networked virtual world for kids*. Unpublished doctoral dissertation, Boston, Massachusetts Institute of Technology.

Bruckman, A. (1998). Community support for constructivist learning. *Computer Supported Cooperative Work*, *7*, 47-86.

Bruckman, A. (2000). Situated support for learning: Storm's weekend with Rachael. *Journal of the Learning Sciences*, *9*(3), 329-372.

Bruckman, A. (2004). Co-evolution of technological design and pedagogy in an online learning community. In S. Barab, R. Kling, & J. Gray (Eds.), *Designing for virtual communities in the service of learning* (pp. 239-255). Cambridge, UK: Cambridge University Press.

Bruckman, A., Edwards, E., Elliott, J., & Jensen, C. (2000, June). *Uneven achievement in a constructionist learning environment*. Unpublished proceedings of International Conference for Language Studies 2000, Ann Arbor, MI.

Claypool, K., & Claypool, M. (2005). Teaching software engineering through game design. *ACM SIGCSE Bulletin*, *37*(3), 123-127.

Cognition and Technology Group at Vanderbilt University. (2003). Connecting learning theory and instructional practice: Leveraging some powerful affordances of technology. In H. F. O'Neil, Jr., & R. S. Perez (Eds.), *Technology applications in education: A learning view* (pp. 173-209). Mahwah, NJ: Lawrence Erlbaum.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107-116.

Cuban, L. (2001). *Oversold and underused: Computers in the classroom*. Boston: Harvard University Press.

Cypher, A., & Smith, D. C. (1995). KidSim: End user programming of simulations. In *Proceedings of CHI* (pp. 27-34). Denver, CO: ACM Press.

Dean, C. (2007, April 17). Computer science takes steps to bring women to the fold. *The New York Times*. Retrieved March 10, 2008, from http://www.nytimes.com/2007/04/17/science/17comp.html?ex=1185163200&en=9d6945

Denner, J. (2007). The Girls Creating Games Program: An innovative approach to integrating technology into middle school. *Meridian: A Middle School Computer Technologies Journal*, *1*(10). Available at http://www.ncsu.edu/meridian/win2007/girlgaming/index.htm

Denner, J., Werner, L., Bean, S., & Campe, S. (2005). The Girls Creating Games Program: Strategies for engaging middle school girls in information technology [special issue on gender and IT]. *Frontiers: A Journal of Women's Studies*, *26*(1), 90-98.

Denning, P. J. (2004). The field of programmers myth. *Communications of the ACM*, *47*(7), 15-20.

Denning, P. J. (2007). Computing is a natural science. *Communications of the ACM*, *50*(7), 13-18.

Denning, P., & Martell, C. (2007, April). *Design: Reliable and dependable form and function.* Unpublished paper. Available at http://cs.gmu.edu/cne/pjd/GP/overviews/ov_design.pdf

Fischer, G., & Lemke, A. (1987-1988). Construction kits and design environments: Steps toward human problem-domain communication. *Human-Computer Interaction*, *3*(3), 179-222.

Flanagan, M. (2005). *Troubling "games for girls": Notes from the edge of game design*. Unpublished proceedings of Digital Games Research Association 2005, Vancouver, British Columbia, Canada.

Flanagan, M., Howe, D. C., & Nissenbaum, H. (2005a). *New design methods for activist gaming*. Unpublished proceedings of Digital Games Research Association 2005, Vancouver, British Columbia, Canada.

Flanagan, M., Howe, D. C., & Nissenbaum, H. (2005b). Values at play: Design tradeoffs in socially-oriented game design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 751-760). Portland, OR: ACM Press.

Flanagan, M., & Nissenbaum, H. (2007). A game design methodology to incorporate social activist themes. In *Proceedings of CHI 2007* (pp. 181-190). New York: ACM Press.

Fullerton, T., Swain, C., & Hoffman, S. (2004). *Game design workshop: Designing, prototyping, and playtesting games*. San Francisco: CMP Books.

Game creation resources. (n.d.). Retrieved March 10, 2008, from http://www.ambrosine.com/resource.html

Game Learning. (n.d.). Retrieved March 10, 2008, from http://www.gamelearning.net/

Game Maker Community. (n.d.). Retrieved March 10, 2008, from http://gmc.yoyogames.com/

Gee, J. P. (2003). *What video games have to teach us about learning and literacy* (1st ed.). New York: Palgrave Macmillan.

Gee, J. P. (2007, July). *Getting young people to think like game designers*. Retrieved March 10, 2008, from http://spotlight.macfound.org/main/entry/gee_think_like_game_designers/

Gee, J. P., Hull, G. A., & Lankshear, C. (1996). *The new work order: Behind the language of the new capitalism*. Boulder, CO: Westview.

Goldstein, R., Noss, R., Kalas, I., & Pratt, D. (2001). Building rules. In M. Beynon, C. Nehaniv, & K. Dautenhahn (Eds.), *Proceedings of the 4th International Conference of Cognitive Technology* (CT2001, pp. 267-281). Kenilworth, UK: University of Warwick, Coventry.

Goldstein, R., & Pratt, D. (2001). Michael's computer game: A case of open modelling. In M. van der Heuval-Panhuizen (Ed.), *Proceedings of the Twenty Fifth Annual Conference of the International Group for the Psychology of Mathematics* (Vol. 3, pp. 49-56), Utrecht, the Netherlands. Available at www.ioe.ac.uk/playground/RESEARCH/papers/open_modelling.pdf

Good, J., & Robertson, J. (2003, June). *Children's contributions to new technology: The design of AdventureAuthor*. Paper presented at Interaction Design and Children 2004, College Park, MD.

Good, J., & Robertson, J. (2004). Computer games authored by children: a multi-perspective evaluation. In *Proceeding of the 2004 Conference on Interaction Design and Children: Building a Community* (pp. 123-124). Baltimore: ACM Press.

Grand Text Auto. (2007). *Emerging terrain in games and simulation*. Retrieved March 10, 2008, from http://grandtextauto.gatech.edu/2007/04/25/emerging-terrain-in-games-and-simulation

Habgood, J., & Overmars, M. (2006). *The game maker's apprentice: Game development for beginners*. Berkeley, CA: Apress.

Habgood, M., Ainsworth, S., & Benford, S. (2005). Endogenous fantasy and learning in digital games. *Simulation & Gaming*, *36*(4), 483.

Hasso Plattner Design Institute at Stanford University. (n.d.). *Our vision*. Retrieved March 10, 2008, from http://www.stanford.edu/group/dschool/big_picture/our_vision.html

Hoyles, C., Noss, R., Adamson, R., & Lowe, S. (2001). Programming rules: What do children understand? Unpublished proceedings of the Twenty Fifth Annual Conference of the International Group for the Psychology of Mathematics, Utrecht, the Netherlands.

Hyer, T. (2006, May). Intro to design thinking: An interview with David Burney. *Red Hat Magazine*, *19*. Available at http://www.redhat.com/magazine/019may06/features/burney/

Institute of Education University of London. (n.d.a). *Playground Project proposal*. Retrieved March 10, 2008, from http://www.ioe.ac.uk/playground/proposal/proposal1.pdf

Institute of Education University of London. (n.d.b). *Playground Project final report*. Retrieved March 10, 2008, from http://www.ioe.ac.uk/playground/RESEARCH/reports/finalreport/index.htm

I Test: Learning Resource Center. (n.d.). *Girl game company (formerly "Girls Creating Games")*. Retrieved March 10, 2008, from http://www2.edc.org/itestlrc/projects/youthbased/gcg_ca.asp

Kafai, Y. B. (1995). *Minds in play: Computer game design as a context for children's learning*. Mahwah, NJ: Lawrence Erlbaum.

Kafai, Y. B. (1996). Gender differences in children's constructions of video games. In P. M. Greenfield & R. R. Cocking (Eds.), *Interacting with video* (pp. 39-66). Norwood, NJ: Ablex.

Kafai, Y. B. (2006). Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and Culture*, *1*(1), 36.

Kafai, Y., Franke, M., Ching, C., & Shih, J. (1998). Game design as an interactive learning environment for fostering students' and teachers' mathematical inquiry. *International Journal of Computers for Mathematical Learning*, *3*(2), 149-184.

Kahn, K. (2001, August). *ToonTalk and Logo—Is ToonTalk a colleague, competitor, successor, sibling, or child of Logo?* Paper presented at the EuroLogo Conference, Linz, Austria. Available at http://www.toontalk.com/Papers/logott.pdf

Kahn, K. (2004). ToonTalk—Steps towards ideal computer-based learning environments. In M. Tokoro & L. Steels (Eds.), *A learning zone of one's own: Sharing representations and flow in collaborative learning environments* (pp. 253-270). Amsterdam: Ios Press.

Kelley, T., & Littman, J. (2001). *The art of innovation: Lessons in creativity from IDEO, America's leading design firm*. New York: Currency Books.

Kirriemuir, J., & McFarlane, A. (2004). *Literature review in games and learning* (Nesta Futurelab Series, Report 8). Bristol, UK: Nesta Futurelab.

Klopfer, E., & Yoon, S. (2005). Developing games and simulations for today and tomorrow's tech savvy youth. *TechTrends*, *49*(3), 33-41.

Kolodner, J. L., Camp, P. J., Crismond, D., Fasse, B., Gray, J., Holbrook, J., et al. (2003). Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning-by-design into practice. *Journal of the Learning Sciences*, *12*(4), 495-547.

Mau, B., Leonard, J., & Institute Without Boundaries. (2004). *Massive change*. London: Phaidon Press.

New London Group. (1996). A pedagogy of multiliteracies: Designing social futures. *Harvard Educational Review*, *66*(66), 60-92.

Nussbaum, B. (2005, March). The empathy economy. *Business Week*. Retrieved March 10, 2008, from http://www.businessweek.com/bwdaily/dnflash/mar2005/nf2005037_4086.htm

Overmars, M. (2004). Teaching computer science through game design. *Computer*, *37*(4), 81-83.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 1-12). Norwood, NJ: Ablex.

Perkins, D. N. (1986). *Knowledge as design*. Hillsdale, NJ: Lawrence Erlbaum.

Rapunsel. (n.d.). *About Rapunsel*. Retrieved March 10, 2008, from http://www.rapunsel.org/about.htm

Resnick, M. (1994). *Turtles, termites, and traffic jams*. Cambridge, MA: MIT Press.

Resnick, M., Bruckman, A., & Martin, F. (1998). Constructional design: Creating new construction kits for kids. In A. Druin (Ed.), *The design of children's technology* (pp. 149-168). San Francisco: Morgan Kaufmann.

Robertson, J., & Good, J. (2005). Children's narrative development through computer game authoring. *TechTrends*, *49*(5), 43-59.

Robertson, J., & Good, J. (2006). Supporting the development of interactive storytelling skills in teenagers. *Lecture Notes in Computer Science*, *3942*, 348.

Robertson, J., & Nicholson, K. (2007, June). *Adventure Author: A learning environment to support creative design*. Paper presented at the 6th International Conference on Interaction Design and Children, Aalborg, Denmark.

Rosas, R., Nussbaumb, M., Cumsillea, P., Marianovb, V., Correaa, M., Floresa, P., et al. (2003). Beyond Nintendo: Design and assessment of educational video games for first and second grade students. *Computers & Education*, *40*, 71-94.

Salen, K., & Zimmerman, E. (2003). *Rules of play: Game design fundamentals*. Boston: MIT Press.

Scratch. (n.d.). *About Scratch*. Retrieved March 10, 2008, from http://scratch.mit.edu/about

Shaffer, D. W. (2006). Epistemic frames for epistemic games. *Computers & Education*, *46*(3), 223-234.

Smith, D. C., & Cypher, A. (1998). Making programming easier for children. In A. Druin (Ed.), *The design of children's technology* (pp. 201-222). San Francisco: Morgan Kaufmann.

Smith, D. C., Cypher, A., & Tesler, L. (2000). Novice programming comes of age. *Communications of the ACM*, *43*(3), 75-81.

Squire, K. (2006). From content to context: Videogames as designed experience. *Educational Researcher*, *35*(8), 19-29.

Squire, K., Giovanetto, L., Devane, B., & Durga, S. (2005). From users to designers: Building a self-organizing game-based learning environment. *TechTrends*, *49*(5), 34-43.

Tholander, J. (2002, October). *Children's understanding and explanations of ToonTalk programs. Elaborations on views on objects and actions*. Paper presented at the International Conference of the Learning Sciences, Seattle, WA.

Tisue, S., & Wilensky, U. (2004, May). *NetLogo: A simple environment for modeling complexity*. Paper presented at the International Conference on Complex Systems, Boston.

ToonTalk. (n.d.a). Retrieved March 10, 2008, from http://www.toontalk.com/english/adultask.htm

ToonTalk. (n.d.b). *Thinking skills and ToonTalk*. Retrieved March 10, 2008, from http://www.toontalk .com/english/think.htm

Wang, K., McCaffrey, C., Wendel, D., & Klopfer, E. (2006). 3D game design with programming blocks in StarLogo TNG. In *Proceedings of the 7th International Conference on Learning Sciences* (pp. 1008-1009). Bloomington, IN: ACM Press.

Wiggins, G. P., & McTighe, J. (2005). *Understanding by design*. Alexandria, VA: Association for Supervision and Curriculum Development.

Williamson Shaffer, D. (2006, December 11). *Why "epistemic"?* Retrieved March 10, 2008, from http://epistemicgames.org/eg/?p=414#more-414

YouthLearn. (n.d.). *Girls Creating Games: About the program*. Retrieved March 10, 2008, from http://www.youthlearn.org/afterschool/GirlsCreatingGames.htm

Elisabeth R. Hayes is a professor of curriculum and instruction at Arizona State University and a founding member of the Games, Learning, & Society research group at the University of Wisconsin–Madison. Her current research interests focus on gender, digital technologies, and learning, particularly the development of IT fluency.

Ivan Alex Games is a doctoral student in curriculum and instruction at the University of Wisconsin–Madison. He holds a BS in computer systems engineering and has an extensive background in systems design, artificial intelligence, robotics, and learning science. Originally from Mexico, he has been involved in a variety of projects that explore the value of videogame technology for facilitating academic learning by children and young adults. During his time as a doctoral student and systems analyst at the University of Texas–Austin, he was in charge of the design and supervision of several videogame-based learning projects for higher education. He has been an avid videogame player for 25 years, and his current research interests involve the design of videogame-based tools to facilitate the acquisition of computer literacy, programming, and problem-solving skills by young people.