International Journal of Game-Based Learning
Volume 6 • Issue 1 • January-March 2016

Computational Thinking in Constructionist Video Games

David Weintrop, Learning Sciences and Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, USA

Nathan Holbert, Mathematics, Science and Technology, Teachers College, Columbia University, New York, NY, USA

Michael S. Horn, Learning Sciences, Computer Science, and Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, USA

Uri Wilensky, Learning Sciences, Computer Science, Northwestern Institute on Complex Systems, and Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, USA

ABSTRACT

Video games offer an exciting opportunity for learners to engage in computational thinking in informal contexts. This paper describes a genre of learning environments called constructionist video games that are especially well suited for developing learners' computational thinking skills. These games blend features of conventional video games with learning and design theory from the constructionist tradition, making the construction of in-game artifacts the core activity of gameplay. Along with defining the constructionist video game, the authors present three design principles central to thier conception of the genre: the construction of personally meaningful computational artifacts, the centrality of powerful ideas, and the opportunity for learner-directed exploration. Using studies conducted with two constructionist video games, the authors show how players used in-game construction tools to design complex artifacts as part of game play, and highlight the computational thinking strategies they engaged in to overcome game challenges.

KEYWORDS

Computational Thinking, Constructionism, Design, Learner-Directed Exploration, Video Games

As our world continues to become progressively digital, the way in which we interact with and shape our environment is increasingly dependent on the ability to translate problems and ideas into forms that computers can interpret and execute. Recent calls by education researchers and computer scientists advocate for bringing this "computational thinking" into formal education spaces. "To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (Wing, 2006, p. 33). This enthusiasm has resulted in new curricula and tools that provide opportunities for students to engage with computational thinking in classrooms. While we applaud this effort to bring computational thinking into formal educational spaces, we think there is untapped potential in the many computational contexts of the everyday. Tools and environments such as video games, mobile applications, and online social communities are ubiquitous in youth culture and already provide compelling computational thinking experiences.

While the emphasis on the importance of computational thinking has received much recent attention (National Research Council, 2010, 2011), the ideas that underpin this movement have long been championed in constructionist theory and research circles, which advocate for engaging learners in meaningful design and has produced a long tradition of educational environments, tools,

DOI: 10.4018/IJGBL.2016010101

and interventions that foster computational thinking in young learners (Harel & Papert, 1991; Papert 1980, 1996). By incorporating constructionist design into video games and the digital world in which young learners live outside of the classroom, we can not only develop computational experiences that leverage the affordances of the medium and its position in youth culture, but also reimagine the forms that computational thinking can take and how, when, and where learners engage with it.

This paper describes a genre of learning environments called constructionist video games that are especially well suited for developing learners' computational thinking skills. We argue that by blending features of conventional video games with learning and design theory from the constructionist tradition, we can create compelling, motivating learning experiences that align with the skills and practices advocated by the computational thinking community. Using two constructionist video games of our own design, we demonstrate how constructionist design principles can be used to embed computational thinking in the activity of playing video games and provide evidence for the effectiveness of this approach.

This paper begins with a discussion of relevant constructs and design traditions. We then formally define constructionist video games and discuss the three central design principles of the genre. Our two constructionist games are then introduced, highlighting the design principles in use and reporting on a study conducted with each, providing evidence of the development and use of computational thinking skills by learners during game play. The paper concludes with a discussion of the potential of this design genre and a challenge to the educational video game design community to push on narrow views of computational thinking and further explore and experiment with the video game medium as a context for situating these critical skills.

RELEVANT CONSTRUCTS AND THEORY

Computational Thinking

The driving theme behind the computational thinking movement is the idea that knowledge and skills derived from the field of computer science have far reaching applications that can be beneficial to all learners. Central to this skillset is the ability to encode ideas into a form that can be interpreted and executed by a computational device. Though this idea, or close variants, have been proposed frequently under a variety of names over the last half century (diSessa, 2000; Guzdial & Soloway, 2003; Guzdial, 2008; Papert, 1980; Wilensky, 2001), Wing's (2006) recent call to make computational thinking a subject everyone should learn has brought renewed interest and excitement to the cause of bringing these skills into the mainstream.

Despite a long history of research to draw on, no clear consensus of where the boundaries of computational thinking lie has emerged (Grover & Pea, 2013). Wing defines computational thinking as: "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2011). The Computer Science Teachers Association succinctly captures both the central goals of computational thinking and the importance of the skills stating: "the study of computational thinking enables all students to better conceptualize, analyze, and solve complex problems by selecting and applying appropriate strategies and tools, both virtually and in the real world" (2011). A National Research Council report on the scope and nature of computational thinking detailed a lengthy list of skills including: heuristic reasoning, reformulation of difficult problems by reduction and transformation, parallel processing, testing, debugging, simulation, and search strategies. (NRC, 2010, p. 3). Replacing a constrained set of skills tightly coupled to programming with this broader set of concepts opens the door to a much wider set of possible designs that learners can engage with as part of developing computational thinking skills. This more inclusive view has motivated us to look beyond conventional computer science contexts to find opportunities to design novel, engaging computational thinking learning environments that build on existing digital practices of young learners.

Constructionism and Constructionist Design

From its inception, constructionism has explored the cognitive implications of learners generating their own computational artifacts and the potential of the computational medium as a context for expression and learning (Papert, 1980). The desire to allow the child to program the computer, rather than have the computer program the child, led to the development of the Logo language (Feurzeig, Papert, & Lawler, 2011; Papert, 1972). While Logo was designed most immediately to teach mathematical concepts, research revealed that through developing programming skills, learners could cultivate a range of powerful ideas that had purchase beyond the specific activity of programming (Papert & Harel, 1990), a position echoed in the computational thinking literature (Grover & Pea, 2013). Grounded in Piaget's constructivist theory, which characterizes learning as a learner-drive process of constructing understanding through the creation of internal cognitive structures through the processes of accommodation and assimilation (Piaget, 1952), constructionism applies this theory to the design of learning environments by creating learning experiences in which learners build their own cognitive structures through constructing public, sharable artifacts. This approach challenges the traditional instruction model of teachers as disseminators of knowledge and replaces it with learner-driven activities (Papert, 1980, 1993).

Constructionist learning designs have been successfully used in a variety of content domains, both to motivate learners to deeply explore content and to help them develop high levels of content understanding (Blikstein & Wilensky, 2009; Caperton, 2010; Harel & Papert, 1991; Kafai, 1995; Noss & Hoyles, 1996; Sengupta & Wilensky, 2009; Wilensky & Reisman, 2006; Wilensky, 1996). While numerous features contribute to the effectiveness of these environments, here we highlight three that are central to all constructionist tools. First is the idea that learners should have the opportunity to construct personally meaningful artifacts that can be publicly shared. Such designs allow learners to set their own agendas and pursue their own goals, empowering them to be the architects of their own learning. Through designing, constructing, and refining these external artifacts, internal cognitive structures are created and reorganized as the learner builds and explores. Second is the centrality of "powerful ideas" in the learning environment. Powerful ideas are concepts that are central to many domains and provide access to a large number of other useful concepts. Finally, constructionist environments should be "discovery rich." Learners should be encouraged to explore and the environment should be designed so that exploration within the space is fruitful with respect to the larger goals of the environment. These three design principles are central to constructionist video games and provide a roadmap for enabling deep, meaningful learning in video game contexts.

Video Games and Constructionism

As we propose to bring computational thinking to young people's everyday practices, we turn now to video games, an activity shared by nearly all youth in economically developed countries (Lenhart et al., 2008; Rideout, Foehr, & Roberts, 2010). Because of the popularity of this activity, education researchers, policy makers, and funding agencies have sought to harness the motivational aspects of video games as a means to deliver skills and content taught in formal educational settings. Educational researchers have explored the learning that occurs when individuals play video games (Gee, 2003; Squire, 2013; Steinkuehler, Squire, & Barab, 2012; Stevens, Satwicz, & McCarthy, 2007). This work spans a variety of subject areas, ranging from the study of scientific practices (Barab et al., 2007; Clark et al., 2011; Holbert, 2013; Steinkuhler & Duncan, 2008) to how game play can be used to explore issues of player identity (Itō, 2010; Squire & Barab, 2004).

Constructionism has a long history of incorporating aspects of video games to achieve desired learning goals for ideas central to computational thinking. Early Logo projects and more recent constructionist programs utilize game design as an impetus for construction (Caperton, 2010; Kafai, 1995; Papert & Harel, 1991), while other constructionist initiatives draw on game design principles to structure the learner's experience (Goldstein et al., 2001; Kahn, 1999) or to teach "Computational Thinking Patterns" (Ioannidou et al., 2011; Repenning, Webb, & Ioannidou, 2010). Some projects have attempted to make *building* in the game the prime activity of *playing* a game. In these games, players progress through levels by creating and programming the behaviors and actions of game entities (Berland, Martin, & Benton, 2010; Games, 2010; Kazimoglu et al., 2012; MacLaurin, 2009; Weintrop & Wilensky, 2014). Another approach uses the mechanism of training virtual players to spur the development and use of computational thinking skills in a video game context (Lee et al., 2014). Finally, "microworlds," which are constructionist environments that instantiate rules and behaviors particular to a chosen domain, have been used for many years to provide game-like experiences within an constrained constructionist space to encourage learners to "play" with the rules of the system (Edwards, 1995).

While many projects, including those discussed above, have used game design as a context for practicing and developing computational thinking, fewer designs have attempted to make gameplay itself a constructionist endeavor. One reason relatively few video games have been designed for practicing computational thinking is the perception that traditional game design is at odds with the assumed territory of computational thinking—namely programming. Video games are often designed to carefully sequence challenges to ensure they remain appropriate for the player's skill level and constrain action to give players time to learn skills necessary for progression. While these design techniques are adept at ensuring players encounter targeted experiences (making them especially suitable for a particular paradigm of educational design), they are less appropriate for allowing the enactment of computational practices such as iterative development, recursion, or debugging, which often require an open space with a great degree of freedom to try a variety of solutions and to engage in productive failure (Kapur, 2008). In our own work we have explored the potential of this genre for learning about kinematics (Holbert & Wilensky, 2010; 2014), the particulate nature of matter (Brady et al., 2014; Holbert, 2013), systems feedback (Brandes & Wilensky, 1991) and programming (Weintrop & Wilensky, 2012; 2014). Others have utilized this approach to design games for exploring science and math (Klopfer et al., 2009), Newtonian mechanics (Clark et al., 2009), and robotics (Berland, Martin, & Benton, 2010). In this paper we demonstrate that by incorporating constructionist design principles, video game play can serve as an ideal space for developing computational thinking practices (Holbert, Penny, & Wilensky, 2010; Holbert & Wilensky, 2011).

DEFINING THE CONSTRUCTIONIST VIDEO GAME GENRE

Having articulated the constructs and theoretical foundations of this work, we now define constructionist video games and present three design principles to act as a framework for creating such games. Adopting the definition of *games* provided by Salen and Zimmerman (2004), we define constructionist video games as:

Designed computational environments in which players construct personally meaningful artifacts to overcome artificial conflict or obstacles resulting in quantifiable outcomes.

This definition of constructionist video games remains faithful to key aspects of traditional games, but also include the construction of in-game artifacts as the *central activity of gameplay*. By shifting gameplay towards construction (rather than reflex) to achieve in-game goals, the resulting game can remain true to video game conventions while adhering to constructionist principles. In constructionist video games, players might design and build in-game tools to accomplish a goal or "program" actions of the in-game character to fight off enemies. While construction activities within the game may take many forms, it is important that the resulting artifacts are identifiable by the player and useful for achieving in-game goals. In a constructionist video game, construction is not relegated to a mini-game that occurs outside of the main game action, nor should constructions be forgettable or artificial with respect to the larger game objective (such being able to customizing the "look" of a racecar without impacting its performance). Furthermore, the construction process itself should allow for a high degree of freedom—players are free to make the thing they *want* to make, rather than provided a false sense of freedom that leads them toward constructing the "correct" thing. This does not mean that there cannot be "correct" solutions—as an environment with "quantifiable outcomes" by definition requires some form of evaluation—but rather the game supports a variety of correct solutions, providing many different ways to arrive at these solutions.

While the design of individual games of this genre may vary, we propose three core design principles for creating constructionist video games that foster computational thinking: the construction of personally meaningful computational artifacts, the opportunity for learner directed explorations, and the centrality of powerful ideas. For each principle, we discuss its constructionist roots and link it to specific computational thinking skills and practices.

Principle 1: Constructionist video games include sufficiently expressive construction tools for players to engage with and build personally meaningful artifacts.

This paramount principle redefines gameplay to focus on having players build things they care about—things they wanted to show to their friends, their parents, and to keep for themselves. It is in this process of construction that play becomes linked to the development and refinement of knowledge (Harel & Papert, 1991; Papert, 1980). Creating this link in a video game context is dependent on players being given the opportunity to make authentic and consequential choices during play. Therefore, constructionist video games must provide a set of sufficiently expressive tools or mechanics for players to construct solutions to the in-game challenges in a way that promotes a feeling of ownership over the resulting artifact or gameplay.

Central to our conception of computational thinking is the practice of encoding ideas into forms that computational devices can interpret and execute. Designing video games that allow players to express their invented gameplay strategies, ideas, characters, etc. using tailored tools and representations ensures that players encounter and develop computational thinking skills such as testing and debugging computational constructions, identifying patterns, and iteratively developing and revising solutions. To achieve this goal, constructionist video games must provide thoughtfully constrained primitives, or building components, that are flexible enough to enable diverse constructions and support various epistemological perspectives, but also constrained enough so that learners can easily find meaning in their form. As game designers, we can adjust the expressiveness of the in-game representations through the breadth of control options provided, as well as through the granularity of the building blocks offered to players. If building blocks are too large, then the game may become too easy, or too restrictive in terms of expressiveness. At the same time, blocks that are too "small" might make the activity tedious or overly difficult (Wilensky, 1999). By providing carefully tailored and flexible construction systems that allow the player to create something personally meaningful that accomplishes the in-game objective, gameplay becomes fundamentally tied to encoding ideas to be acted on by a computational agent – a central computational thinking practice.

Principle 2: Game goals and construction tools encourage exploration and discovery during play.

Constructionism places a premium on learner-directed exploration, which at first might seem counter to the goal of learners engaging with specific content. However, narrowing the provided construction tools to fit within the desired domain need not compromise the learners' ability to explore. "Although there are constraints on the materials, there are no constraints on the exploration of combinations…the power of the environment is that it is 'discovery rich'" (Papert, 1980, p. 162). This feature of constructionist environments, when brought into contexts in which the constructions are computational, provides a productive context for the development of computational thinking skills.

One characteristic of computational thinkers is their ability to work with computational tools to make progress towards solutions despite uncertainty of what form the final solution will take. Instead of expecting learners to know the solution at the outset, the expectation is that learners work their way towards it, trying solutions, learning from previous trials, and incorporating lessons learned in prior attempts into future iterations, a common practice in well designed video games (Gee, 2003; Squire, 2005). By designing games that encourage exploration and discovery, players become the architects of their own learning as they are challenged to learn, revise, and adapt to advance in the game.

If a game requires "defined rules and quantifiable outcomes" as we have stated in our definition, how then can we also reward exploration and self-directed discovery? First, by not limiting the player to a single or a small set of winning strategies, the game can support an epistemological pluralism (Turkle & Papert, 1990) that rewards various approaches that can accomplish the task. Allowing for a variety of solutions also introduces a qualitative aspect to constructions; players can decide if they prefer one construction to another or prioritize certain features of their constructions over others. Alternatively, such games can be designed to allow players to choose how and when they engage with challenges. Finally, creating a low-stakes environment where there is little risk associated with experimentation allows players to explore the construction space without worrying about negative consequences. In these types of designs, players are free to engage in trial and error, should be able to quickly iterate and modify play, and be allowed to explore alternative solutions even after they have achieved success without serious repercussion.

Principle 3: Learners engage with and employ powerful ideas to advance through the game.

Not all concepts are created equal—while some ideas have only narrow applications, others are much more *powerful*, providing purchase for reasoning across a diverse set of problem spaces. Powerful ideas are seen to be immediately useful, connect to many other productive ideas, and are rooted in the learner's intuitive understanding about the world (Papert 1980, 2000). An idea's power, from an epistemological standpoint, is tied to its ability to "lead to an understanding of a large class of phenomenon" (Papert, 2000, p. 727). Powerful ideas are not limited to the case in which they are discovered. Rather, they are central hubs in a vast network of knowledge. In this respect, there is much overlap between powerful ideas and computational thinking as computational thinking is frequently championed as being broadly applicable and useful across a wide variety of contexts (NRC, 2010; Wing, 2006).

Papert suggests that ideas are powerful if they are syntonic to the learner – that is, if they connect to the learner's personal and intuitive understanding of the world. The intuitive nature of powerful ideas makes them highly accessible to learners of all ages and cognitive ability, further democratizing access to knowledge once thought to require years of formal instruction. The Logo turtle, perhaps Papert's most famous contribution, was said to be body-syntonic (1980) in that the child was able to use his own understanding about how his body functions in the world to command and direct the turtle as it "walked-out" shapes and animations on a computer screen. Similarly, learning experiences that align to aspects of the learner's identity and way of life are culturally-syntonic (Papert, 1980), as the understandings that are being cultivated fit within the practices and activities the learner regularly engages in. Because video game play is a central part of youth culture, situating computational thinking within video games provides a culturally syntonic way for learners to experience these ideas.

COMPUTATIONAL THINKING IN CONSTRUCTIONIST VIDEO GAMES

Having described the constructionist video game genre and outlined design principles central to their creation, we now introduce two examples and present data on students' developing computational thinking skills as they play. We intentionally use two very different games to highlight the flexibility of the genre and challenge a narrow view of computational thinking that tightly couples the skills with conventional computer science learning environments. In evaluating these games we utilize a mixed methods approach. To explore how the environment shapes learners' emerging understanding of computational practices, we rely on semi-clinical interviews and observational data. To identify patterns in game play behaviors and to gain insight into the typicality of these behaviors and the outcomes observed in the case studies, we use computational methods such as learning analytics and cluster analysis. By triangulating the observations, interviews, and computational data, we gain insight into how the design of the games—and more specifically the constructionist features—provide a space for learners to explore and experiment with computational practices.

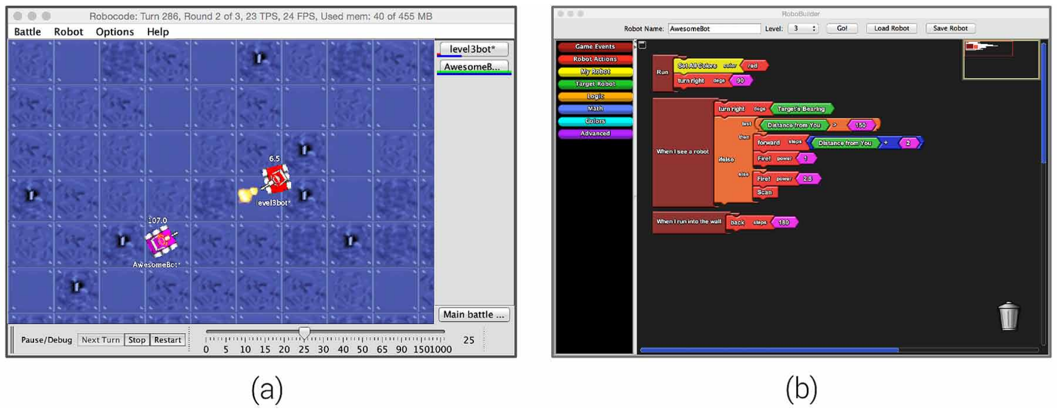
Developing Iterative Solutions in RoboBuilder

RoboBuilder (Figure 1), is a program-to-play game that challenges learners to design and implement combat strategies to make their virtual robot defeat a series of progressively more challenging opponents in one-on-one battle (Weintrop & Wilensky, 2012). Unlike a conventional game where the player would control the robot live as the game unfolds, in RoboBuilder, the player gives instructions to the robot before it competes. The design was initially conceptualized as a way to situate programming in a motivating context while also providing the designer a way to scaffold and motivate learning in moving from simple to more complex programs. Research into the design rational of other successful introductory programming environments led us to take a constructionist design approach. Figure 1 shows RoboBuilder's interface.

In RoboBuilder, players build strategies for their robots using a custom designed blocks-based, visual programming environment. The programming primitives were chosen to closely map on to in-game events and can support a wide variety of strategies (principle 1). While the core purpose of the RoboBuilder language is to provide a toolset for constructing strategies to defeat progressively more difficult opponents, players are free to interact with the tools, and consequently play the game, in whatever way they see fit (principle 2). The types of strategies that players compose reflect their own strategic sensibilities, be they assertive and aggressive, passive and defensive, or emphasize aesthetics along with effectiveness. The combination of a single, high-level game goal (defeat your opponent) paired with the expressive language and the visual enactment of players' constructed strategies, encourages players to explore various robot designs.

A number of powerful ideas that align with computational thinking are made central to the gameplay of RoboBuilder (principle 3). First, the act of constructing a winning strategy using a blocks-based programming language challenges players to encode their ideas into a format that the

Figure 1. RoboBuilder's two windows: (a) the battle screen where players watch their robot compete and (b) the construction space where players implement their strategies.



```
Run
    Set All Colors  color  red
    turn right  degs  90
When I see a robot
    turn right  degs  Target's Bearing
    ifelse
        test  Distance from You > 150
        then
            forward  steps  Distance from You ÷ 2
            Fire!  power  1
        else
            Fire!  power  2.5
    Scan
When I run into the wall
    back  steps  180
```

computer can interpret and execute, a practice central to our conception of computational thinking. In doing so, players must work with abstractions, debug unintended behaviors in the program, and interpret the feedback they received in order to advance in the game. In the analysis we present below, we focus on two computational thinking skills: encoding ideas in a computational medium and iteratively developing solutions to a problem.

Methods and Participants

The data presented in this section were collected through a series of hour-long, one-on-one interviews in which a researcher sat alongside participants as they played the game. The interviews followed a three-phase, iterative protocol. First, players discussed their strategies with the interviewer. Next, the players were given the chance to interact with the game and implement their strategy. Finally, the interviewer and player watched the robot compete, with the interviewer asking the player to describe what he or she had seen and whether or not it matched expectations. After each battle, players were asked how they wanted to improve or change their strategy, thus beginning the next iteration. This iterative interview-play design allowed us to observe the formation and evolution of an individual's computational thinking practices and to continually probe how the learner made sense of the game mechanics and concepts embedded into the game. Along with recording the interview, we collected and analyzed the robot strategies the players constructed and conducted brief post-gameplay interviews. In this analysis we first focus on characteristics of student-authored programs looking for evidence of productive computational thinking practices, before referring to the qualitative interviews to support the findings from our computational analysis.

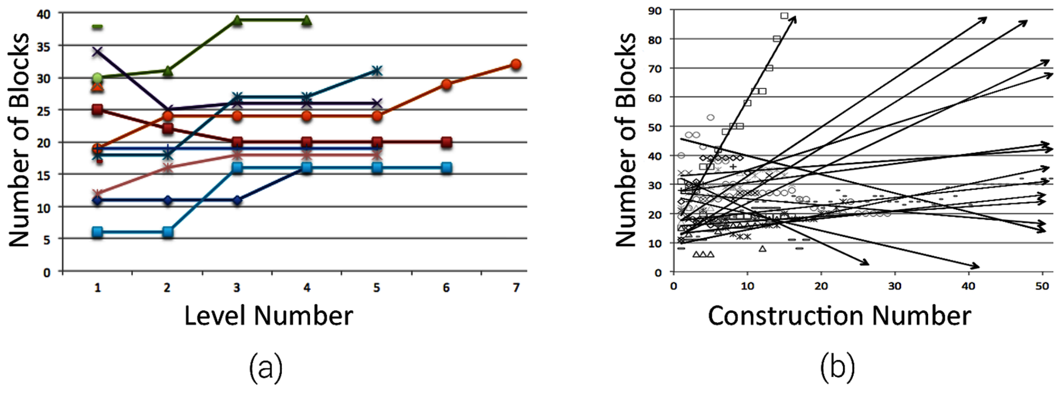
Fifteen subjects with little or no prior programming experience were recruited to participate in this study. The study took place in a large American city. Participation in the study was voluntary and included students ranging in age from seventh grade to graduate school. Convenience sampling was used to recruit participants, finding novice programmers affiliated with educational institutions the research had relationships with. Participants played for an average of 48 minutes and 43 seconds (SD 8 minutes 39 seconds) and constructed an average of 11.5 unique robot strategies (SD 4.9). This resulted in a total of over 200 robot strategies being constructed and roughly 19 hours of RoboBuilder gameplay footage.

Results

The central gameplay mechanism in RoboBuilder is to construct robot strategies using RoboBuilder's blocks-based programming language to defeat a series of opponents. Of the 15 programming novices who participated in this study, all but one were able to defeat the level-one opponent. Put another way, through playing our constructionist video game, 14 out of 15 novices were able to successful construct a program that a computer could execute to accomplish a task—an activity central to computational thinking. These programs went beyond a single command; the average winning level-one program contained 22.6 blocks (SD 9.5), with nine participants constructed programs that defeated the first three opponents in the game. As players advanced through the game, and defeated successively more challenging opponents, their constructions generally grew in size and complexity (Figure 2).

RoboBuilder players' also generally utilized an iterative and incremental approach when designing winning constructions, an important computational thinking practice. An iterative and incremental approach is often cited as a best practice when developing computational solutions to problems (Larman & Basili, 2003). We analyzed the video footage collected and coded each interview, looking for iterative development practices. Specifically, we looked for players repeating a cycle of making a small number of changes to their program, running the battle to see the effect of those changes, then returning to the construction space to make another round of small changes in quick succession. We found that each participant added an average of 1.6 blocks to their program for each level they advanced. If we only consider the cases where participants revised their robot strategies between successful battles, the number of blocks added per level increases to 3.3 blocks for each new successful

Figure 2. Charts showing the size of RoboBuilder constructions: (a) the size of winning robot strategies and (b) the trend lines of the changing size of constructions (projected forward).



robot construction. Only two players' robot constructions got smaller as they progressed through the game; a third player's program remained at a fixed size; the remaining players' constructions grew as they progressed in the game. This fact is highlighted in Figure 2b where we see mostly upward sloping trend lines, the gradual slopes denote small, incremental changes being made.

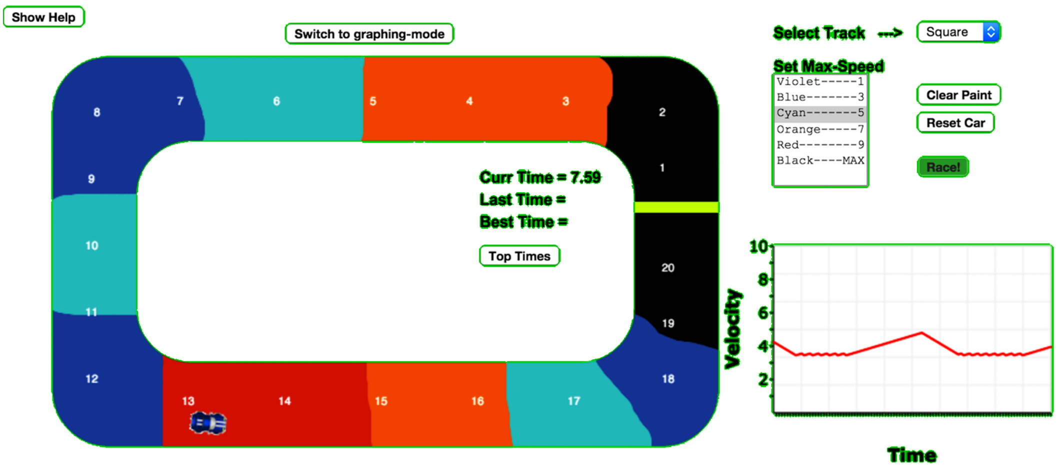
At the conclusion of the interviews, the participants were asked if they had any advice for future RoboBuilder players. In their responses, seven of the participants advised future players to use an iterative, incremental approach, or to start with simple constructions and build up from there. For example, one participant advised future players: "don't try and do too much in the beginning," and followed up that advice by saying "it is better to build up," recommending that strategies need not be built all at once. Another participant reflected: "I think what helped me was the iterations...I would recommend that [future players] don't change too much each time." The approach of taking an iterative, incremental approach is a powerful idea for problem solving and is an important, broadly applicable computational thinking strategy. By providing players with an accessible and sufficiently expressive toolset and the freedom to develop the strategies the wanted to, RoboBuilder, a constructionist video game, enabled players to have meaningful interactions with powerful ideas central to computational thinking.

PROCEDURAL THINKING IN FORMULAT RACING

Our second illustration of a constructionist video game for computational thinking is FormulaT Racing (Holbert & Wilensky, 2010). FormulaT Racing (FTR) is a racing game where, rather than drive a car with a joystick or controller, players direct a racecar by painting the track with various colors, each indicating a different velocity (Figure 3). Players are free to paint the track in any way they choose, though they must be careful to consider track features such as sharp turns and race time limits. After the player has painted the entire track, the race begins and the car drives around the track automatically, turning when necessary and dynamically adjusting its velocity as it moves over each painted segment.

While players do not engage in an activity that would be generally recognized as programming, as we will show in this section, the encoding of a "race" representation in FTR incorporates our three design principles and engages players in the development of computational thinking strategies. Though there are only a limited number of colors (corresponding to specific velocities), because players choose the location and combination of colors placed on the track, this small set of primitives can be highly expressive (principle 1) and result in great diversity of racetrack designs. Some players may choose to paint every inch of the track, while others may only apply colors sparingly in strategic locations; still

Figure 3. In FormulaT Racing players paint the track various colors to encode behaviors that the car follows as it races around the track



others may focus on the aesthetics of their construction, trying to make a track that is both beautiful and fast. Likewise, the ability to easily reset the player car after a failed run encourages players to quickly prototype and iterate race constructions to explore the range of designs possible (principle 2).

Rather than making computational thinking the central goal of the game, like RoboBuilder, FTR was designed to facilitate thinking and reasoning about kinematics *through* the enactment of computational strategies (Holbert & Wilensky, 2011). Specifically, FTR was designed to encourage players to reason about the relationship between velocity and acceleration by allowing them to encode these kinematic concepts into systematic representations (Holbert & Wilensky, 2014). By requiring players to articulate these relationships into a form that can be executed by the computer, and supporting them in debugging and iteratively refining these representations, we argue that players interact deeply with the powerful idea of kinematics (principle 3). In the remainder of this section we present a study of players painting and graphing in FTR and show how they not only used computational thinking during play, but how these skills became more advanced as they progressed in the game.

METHODS AND PARTICIPANTS

In a study exploring the ways in which players enacted computational strategies during FTR play, two unique but complementary forms of data were collected. First, six players aged 7-13 recruited from various informal organizations in a Midwestern American city were interviewed and observed as they played FTR at their homes or in an after-school program they attended. Pre-game interviews lasted approximately 45-minutes. Participants played the game for 1 hour 2-4 days after the first interview. Finally, a 45-minute post-game interview was conducted approximately 1 week after playing the game. The second source of data comes from anonymous logs collected from online play that recorded player constructions. The motivation for using these two data sources matches the previous study's rational: the interview data provided detailed insight into the effects of different aspects of the game's design on learning, specifically its constructionist features, while the computational data allowed for some level of generalizability across the full set of participants. In the following section, we present evidence from video recorded observations of gameplay as well as online logging data that show players utilizing powerful computational thinking strategies during play and that these strategies increase in sophistication as players become more experienced with game mechanics.

Results

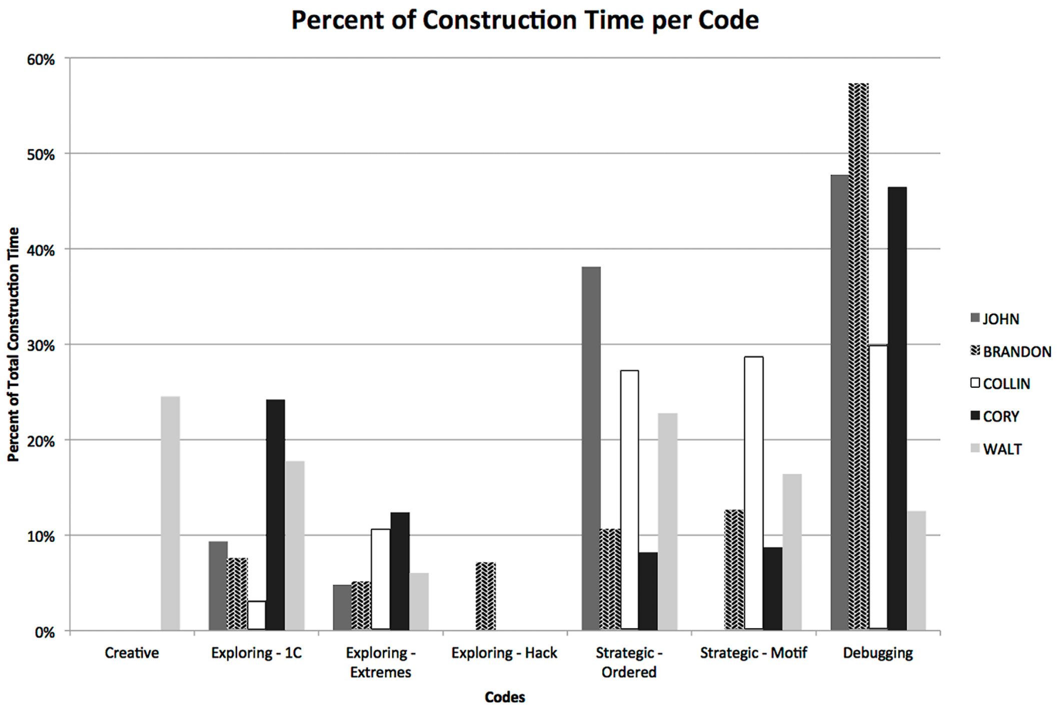
To analyze the way participants interact with FTR, video of the participant playing the game was synced to screen recording of game action and placed side-by-side (Stevens et al., 2008). Video data was split into analytic units according to instances of strategy switching (often corresponding to each race attempt). Analytic units were coded using a scheme that emerged from the data informed by the computational thinking literature and included codes such as creatively designing a representation, exploring the limits of the construction tools, planning construction chronologically, breaking constructions into repeating motifs, and debugging constructions (the full coding scheme can be found in Holbert, 2013). An independent researcher verified the game-play codes. Conflicts were discussed and resolved resulting in agreement on 97% of video time. Our analysis of participants' gameplay found that players usually begin by exploring the painting system. This exploration often takes the form of "painting" the entire track one very fast color. After getting a sense of the scale of the painting system, players begin to systematically debug constructions. Debugging occurs when players make very small changes to their painting—not unlike making minor and iterative modifications to "code"—before attempting another run. As play progresses, many players begin to notice and reuse patterns of motion and track features to paint and graph successful solutions. For example, finding a pattern that works for some feature of the course, like repeating a color pattern of fast->slower->slow in the corners, then systematically using it throughout the track. These motifs are a relatively sophisticated computational thinking strategy akin to breaking a program up into many smaller "procedures" or "methods" that can be called upon and executed at the appropriate time.

Figure 4 shows the percentage of total time players enact a particular computational strategy. While players spend some time simply exploring the model—painting the track all one color, "just to see what will happen," or to see how fast the car could go—players engage in sophisticated computational strategies 76% of the time. These strategies include debugging, using repeated motifs, and enacting an ordered strategy across large portions of the track.

To understand if and how such computational thinking strategies were used among a larger population of users outside of one-on-one interviews, a second analysis of gameplay data was conducted using anonymous logging data from online play. Thirty logs were analyzed having an average of 27 changes made to the painted track per player. Numerical data was coded using computational analytic techniques to look for predefined patterns in player paintings. By identifying the number of changes made between each run, attempts could be automatically coded as either planning out complex strategies all at once (*plan*), enacting many changes between runs (*explore*), or making only a few changes between runs (*debug*). After automatically coding each run, a manual pass was made on the visual data to verify automatically generated codes and to also identify repeating patterns or *motifs* and evidence of *tinkering*—attempts to create a complex plan by enacting very small portions one at a time.

The process of painting and the forms the paintings took in the online version of FTR looked similar to what was observed during the interviews. While some players *planned* strategies from the beginning, more players tended to *tinker* towards a successful painting. Not surprisingly, tinkering tended to happen more at the beginning of a gameplay session and then dropped towards the end. *Debugging* was by far the most common activity enacted by players. Not surprisingly, debugging became more central to gameplay as players progressed in the game. Seventy percent of all online players also utilized repeating patterns or strategic *motifs* (such as slow->slower->fast around corners) during gameplay with each player creating an average of 4.9 motifs over the course of gameplay. Together with the analysis of interviewed participants' gameplay, these results indicate that FTR, and its employment of constructionist video game principles, allowed for and encouraged the use of sophisticated computational thinking strategies. The game's open-ended painting task, intuitive representation of kinematic concepts, and support of multiple play styles, allows players to *tinker*, *plan*, or *explore* their way towards a successful strategy.

Figure 4. This graph shows the breakdown of time each individual spent engaged in the coded activities. 76% of time spent using the construction tools was spent engaging in complex computational thinking (strategic ordered, strategic motif, or debugging).



DISCUSSION

In this paper we set out to both introduce constructionist video games—a genre blending common video game design techniques and learning theory rooted in the constructionist design paradigm—and argue that this new genre of game is particularly suited for supporting the development of complex computational thinking skills. Our proposal of the genre included three core design principles for creating these constructionist video games and examples of two such games. The presentation and analysis of these exemplar constructionist video games reveals the possible variety inherent in the genre, and indicates a relationship between the inclusion of the three core principles and observed play patterns and documented learning outcomes.

Both games provided accessible and highly expressive construction tools (principle 1). These tools allowed players to easily understand game mechanics and quickly implement strategies and actions to meet both their self-defined goals and the larger game objective. While certain ideas or strategies were common during robot construction, players implemented these ideas and strategies in unique ways throughout game play. Likewise, while some racing strategies were more successful than others, players approached track design in a variety of ways, with some planning ahead, others tinkering, and still others ensuring track constructions included aesthetic flair.

Despite vastly different game mechanics and designs, both RoboBuilder and FTR provided tools and mechanics that allowed for a high degree of personalization and encouraged exploration in construction (principle 2). Players freely experimented with previously unused programming blocks in RoboBuilder and thoroughly investigated the specifics behaviors of each color in FTR. By lowering the cost of failure and streamlining the test and debug cycle, players of both games created many constructions and engaged in tinkering and iterative design, two beneficial practices for working in computational contexts (Berland et al., 2013).

Both RoboBuilder and FTR players employed powerful ideas central to computational thinking practices throughout game play (principle 3). In both games, players successfully created abstractions of strategies in a form that could be read and acted on by a computational device. In RoboBuilder, this meant encoding behaviors and strategies for a robot to follow as it tried to defeat its competition. Similarly, in FTR, players painted colors to define the action of a vehicle as it moved around a track. Furthermore, we showed that in each game, players' "programs" became more complex, and their use of advanced computational thinking practices became more pronounced as they played the game, highlighting how the video game medium can scaffold learners in developing and refining computational think practices. RoboBuilder players utilized more blocks and advanced logic blocks as opponent difficulty increased and likewise, over time, FTR players were more likely to utilize motifs, or patterns of action, when building their race representation.

CONCLUSION

In this paper we weave together two successful, but disparate traditions—constructionism and video game design—in such a way as to take advantage of the strengths of each. In addition to defining the constructionist video game genre, we also provide guidelines in the form of three design principles for creating educational tools that fall within this space and argue that the games that result from this marriage are especially well situated to engage players in computational thinking ideas and practices. To support this claim, we present data from studies of two very different constructionist video game designs to offer concrete but diverse examples of games in this genre, as well as to show how interactions with the tools and features of these games facilitate the practice and development of computational thinking strategies. It is our hope that these exemplar games might offer new perspectives on what computational thinking learning environments might look like, and new insights into how learners can engage with ideas central to computational thinking.

Computational thinking, having primarily grown out of the computer science education community, remains closely associated with formal classrooms despite claims of broad applicability. If computational thinking is to achieve equal status in the larger educational and social space, we must challenge narrow conceptions of computational thinking, moving its locus out of computer labs and classrooms and into the larger, digital world in which learners live. By integrating computational thinking into self-directed play, as we have done with constructionist video games, learners begin to see these practices and ways of thinking as another way to interact with the world, rather than as skills only valued in a particular domain (see Berland & Lee, 2010 for a second, non-digital context). Continuing to explore the potential of these "everyday" contexts should be a central focus for the community as we work to introduce computational thinking to large and diverse audiences.

By bringing computational thinking out of the classroom and into kids' daily lives, we also have the potential to fundamentally reshape the nature of computational thinking. For example, in RoboBuilder, which utilizes a blocks-based language aligned with typical programming conventions, computational thinking is pushed beyond the act of programming. While learners certainly practice computational thinking within the robot construction phase, they also hone these skills as they study and decipher their opponents' behavior, and as they interpret and refine their strategies. These two activities, distinct from programming, become central to the way that learners reflect on created code and gain knowledge of new strategies and programming concepts (Weintrop & Wilensky, 2013). In FTR, players engage in computational thinking using tools and representations lacking many defining features of conventional programming environments. Instead of manipulating text or numbers, players express instructions for the computer through painting—using colors and spatial arrangement to convey meaning. Learners continue to work with abstractions, search for patterns, and debug programs, but as part of an activity quite different than typing statements in a text editor. RoboBuilder and FTR serve as examples of learning environments that push beyond a narrow view of computational thinking as learning to program in a formal classroom. By reimagining what it looks like to practice computational thinking, we can broaden its reach and relevance, engaging and exciting learners in ways that are more familiar and inviting than conventional computational thinking settings.

While we think constructionist video games are especially well suited for engaging learners in computational thinking, the complementary features of the constructionist and video game design paradigms provide an ideal learning space for a range of domains and powerful ideas. The synthesis of these two design traditions is a relatively recent—and still underutilized—approach, due at least in part to the seemingly different motivations that underpin each genre. On the one hand, popular video games emphasize a low entry point and streamlined play in an effort to ensure players fully enjoy the designed experience. Conversely, constructionist environments privilege personalization and creativity, allowing learners to explore topics and ideas of personal interest. While these aims differ, they are not incompatible; attending to both can produce learning environments that have the engagement, enjoyment and cultural syntonicity (Papert, 1980) of video games, while producing the deep learning outcomes common to constructionist learning tools.

Further, incorporating video game design tropes into constructionist spaces can provide one way out of what Noss and Hoyles (1996) call the "play paradox"—a long-standing design challenge for constructionist environments. In a highly open-ended constructionist environment, it is difficult to ensure that learners encounter the target content. And yet, constraining the interaction space to ensure content coverage often sacrifices learner agency and the feeling of play. As constructionist video games require players to achieve specific, quantifiable outcomes, through carefully designing in-game objectives to align with learning outcomes, players can be motivated to engage with the desired content in ways that are learner-directed and empowering. In RoboBuilder, to defeat progressively more difficult opponents, learners must explore and utilize the full range of language features, while FormulaT, through its use of normal, repeating racetrack features and compelling visual representations, rewards players for finding and reusing productive patterns. As a result, the design of the in-game challenges serves as a means to encourage and reward leaners for engaging with and employing the ideas central to the designer's learning agenda. In bringing together these two design traditions, we see great promise in providing learners with meaningful learning experiences, be it with computational thinking or other powerful ideas.

On the surface, RoboBuilder and FormulaT Racing look like very different games. FTR's roots are in more traditional video game design, while RoboBuilder emerged from environments and tools designed to teach programming fundamentals. Likewise, RoboBuilder was designed specifically to teach more classic programming concepts, while FTR targets the development of kinematic intuitions. Despite these very different beginnings and goals, by focusing on allowing learners to create artifacts or solutions that are personally meaningful, allowing for low-stakes exploration of the design space, and focusing on exploring and using powerful ideas, both games provide players with an engaging construction experience that is rich with opportunities to engage in computational thinking. Our hope is these two games, and the broader genre of constructionist video games, can provide meaningful and powerful computational thinking learning experiences for young players that can be drawn on during whatever future endeavors they choose to pursue.

REFERENCES

Barab, S., Zuiker, S., Warren, S., Hickey, D., Ingram-Goble, A., Kwon, E., & Herring, S. C. et al. (2007). Situationally embodied curriculum: Relating formalisms and contexts. *Science Education*, *91*(5), 750–782. doi:10.1002/sce.20217

Berland, M., & Lee, V. R. (2011). Collaborative Strategic Board Games as a Site for Distributed Computational Thinking. *International Journal of Game-Based Learning*, *1*(2), 65–81. doi:10.4018/ijgbl.2011040105

Berland, M., Martin, T., & Benton, T. (2010). Programming standing up: Embodied computing with constructionist robotics. *Proceedings of Constructionism 2010*, Paris, France.

Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. *Journal of the Learning Sciences*, *22*(4), 564–599. doi:10.1080/10508406.2013.836655

Blikstein, P., & Wilensky, U. (2009). An Atom is Known by the Company it Keeps: A Constructionist Learning Environment for Materials Science Using Agent-Based Modeling. *International Journal of Computers for Mathematical Learning*, *14*(2), 81–119. doi:10.1007/s10758-009-9148-8

Brady, C., Holbert, N., Soylu, F., Novak, M., & Wilensky, U. (2014). Sandboxes for Model-Based Inquiry. *Journal of Science Education and Technology*, 24(2), 1–22.

Brandes, A., & Wilensky, U. (1991). Treasureworld: An Environment for the Study and Exploration of Feedback. In I. Harel & S. Papert (Eds.), *Constructionism*. Norwood, MA: Ablex Publishing.

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).

Clark, D., Nelson, B., Sengupta, P., & D'Angelo, C. (2009). Rethinking science learning through digital games and simulations: Genres, examples, and evidence. Paper commissioned for the national research council workshop on games and simulations. Washington, D.C.

Clark, D. B., Nelson, B., Chang, H., Martinez-Garza, M., Slack, K., & D'Angelo, C. M. (2011). Exploring Newtonian Mechanics in a conceptually-integrated digital game: Comparisons of learning and affective outcomes for students in Taiwan and the United States. *Computers & Education*, *57*(3), 2178–2195. doi:10.1016/j.compedu.2011.05.007

Computer Science Teachers Association. (2011). *K-12 computer science standards*. Retrieved from http://csta.acm.org/Curriculum/sub/K12Standards.html

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Edwards, L. D. (1995). Microworlds as representations. *Proceedings of the 2nd International NATO Symposium on Advanced Technology and Education*.

Feurzeig, W., Papert, S., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, *19*(5), 487–501. doi:10.1080/10494820903520040

Games, A., & Kane, L. (2012). *Examining Trends in Adolescents' Computational Thinking Skills within the Globaloria Educational Game Design Environment*. World Wide Workshop. Retrieved from http://www.worldwideworkshop.org/pdfs/ GlobaloriaExaminingTrendsAdolescentsComputSkillsGamesKaneAug2012.pdf

Games, I. A. (2010). Gamestar Mechanic: Learning a designer mindset through communicational competence with the language of games. *Learning, Media and Technology*, *35*(1), 31–52. doi:10.1080/17439880903567774

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

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

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. doi:10.3102/0013189X12463051

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27. doi:10.1145/1378704.1378713

Guzdial, M., & Soloway, E. (2003). Computer science is more important than calculus: The challenge of living up to our potential. *SIGCSE Bulletin*, *35*(2), 5–8. doi:10.1145/782941.782943

Harel, I., & Papert, S. (Eds.). (1991). *Constructionism*. Norwood, N.J.: Ablex Publishing.

Holbert, N. (2013). *Reimagining Game Design: Exploring the Design of Constructible Authentic Representations for Science Reasoning*. Northwestern University.

Holbert, N., Penney, L., & Wilensky, U. (2010). Bringing Constructionism to Action Game-Play. Proceedings of Constructionism 2010, Paris, France.

Holbert, N., & Wilensky, U. (2010). *FormulaT Racing*. Evanston, IL: Center for Connected Learning and Computer-based Modeling.

Holbert, N., & Wilensky, U. (2011). FormulaT Racing: Designing a game for kinematic exploration and computational thinking. *Proceedings of 7th Annual Games, Learning, and Society Conference*, Madison, WI, USA.

Holbert, N., & Wilensky, U. (2014). Constructible Authentic Representations: Designing Video Games that Enable Players to Utilize Knowledge Developed In-Game to Reason About Science. *Technology. Knowledge and Learning*, *19*(1-2), 53–79. doi:10.1007/s10758-014-9214-8

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). Computational Thinking Patterns. *Paper presented at the Annual Meeting of the American Educational Research Association*, New Orleans, LA.

Itō, M. (2010). *Hanging Out, Messing Around, and Geeking Out: Kids Living and Learning With New Media*. MIT Press.

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

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

Kapur, M. (2008). Productive failure. *Cognition and Instruction*, *26*(3), 379–424. doi:10.1080/07370000802212669

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning Programming at the Computational Thinking Level via Digital Game-Play. *Procedia Computer Science*, *9*(0), 522–531. doi:10.1016/j.procs.2012.04.056

Klopfer, E., Roque, R., Huang, W., Wendel, D., & Scheintaub, H. (2009). The Simulation Cycle: Combining games, simulations, engineering and science using StarLogo TNG. *E-learning*, *6*(1), 71. doi:10.2304/elea.2009.6.1.71

Lee, T. Y., Mauriello, M. L., Ahn, J., & Bederson, B. B. (2014). CTArcade: Computational thinking with games in school age children. *International Journal of Child-Computer Interaction*. doi:10.1016/j.ijcci.2014.06.003

Lenhart, A., Kahne, J., Middaugh, E., Macgill, A. R., Evans, C., & Vitak, J. (2008). *Teens, Video Games, and Civics*. PEW Internet & American Life Project.

MacLaurin, M. (2009). Kodu: end-user programming and design for games. *Proceedings of the 4th International Conference on Foundations of Digital Games* (p. 2). ACM. doi:10.1145/1536513.1536516

National Research Council. (2010). *Report of a Workshop on the Scope and Nature of Computational Thinking*. Washington, D.C.: The National Academies Press.

National Research Council. (2011). *Report of a Workshop of Pedagogical Aspects of Computational Thinking*. Washington, D.C.: The National Academies Press.

Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers*. Dordrecht, The Netherlands: Kluwer Academic Press. doi:10.1007/978-94-009-1696-8

Papert, S. (1972). Teaching Children to be Mathematicians Versus Teaching About Mathematics. *International Journal of Mathematical Education in Science and Technology*, *3*(3), 249–262. doi:10.1080/0020739700030306

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

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

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal, 39*(3.4), 720–729.

Papert, S., & Harel, I. (1991). Situating constructionism. In *Constructionism*. New York: Ablex Publishing.

Piaget, J. (1952). *The origins of intelligence in children*. International Universities Press, Inc. doi:10.1037/11494-000

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 265–269). doi:10.1145/1734263.1734357

Rideout, V. J., Foehr, U. G., & Roberts, D. F. (2010). *Generation M2: Media in the lives of 8- to 18-year olds*. The Henry J. Kaiser Family Foundation.

Salen, K., & Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*. Cambridge: The MIT Press.

Sengupta, P., & Wilensky, U. (2009). Learning Electricity with NIELS: Thinking with Electrons and Thinking in Levels. *International Journal of Computers for Mathematical Learning*, *14*(1), 21–50. doi:10.1007/s10758-009-9144-z

Squire, K. (2005). Changing the game: What happens when video games enter the classroom. *Innovate: Journal of Online Education, 1*(6).

Squire, K., & Barab, S. (2004). Replaying history: Engaging urban underserved students in learning world history through computer simulation games. *Proceedings of the 6th International Conference on the Learning Sciences* (pp. 505–512).

Squire, K. D. (2013). Video Game–Based Learning: An Emerging Paradigm for Instruction. *Performance Improvement Quarterly*, *26*(1), 101–130. doi:10.1002/piq.21139

Steinkuehler, C., & Duncan, S. (2008). Scientific habits of mind in virtual worlds. *Journal of Science Education and Technology*, *17*(6), 530–543. doi:10.1007/s10956-008-9120-8

Steinkuehler, C., Squire, K., & Barab, S. (2012). *Games, Learning, and Society: Learning and Meaning in the Digital Age*. Cambridge University Press. doi:10.1017/CBO9781139031127

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

Turkle, S., & Papert, S. (1990). Epistemological pluralism: Styles and voices within the computer culture. *Signs (Chicago, Ill.)*, *16*(1), 128–157. doi:10.1086/494648

Weintrop, D., & Wilensky, U. (2012). RoboBuilder: A program-to-play constructionist video game. In C. Kynigos, J. Clayson, & N. Yiannoutsou (Eds.), *Proceedings of the Constructionism 2012 Conference*, Athens, Greece.

Weintrop, D., & Wilensky, U. (2013). Know your enemy: Learning from in-game opponents. *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 408–411). New York, NY, USA: ACM. doi:10.1145/2485760.2485789

Weintrop, D., & Wilensky, U. (2014). Program-to-play videogames: Developing computational literacy through gameplay. *Proceedings of the 10th Games, Learning, & Society Conference*, Madison, WI, USA.

Weintrop, D., & Wilensky, U. (2014). Situating programming abstractions in a constructionist video game. In G. Futschek & C. Kynigos (Eds.), *Proceedings of Constructionism 2014*. Vienna, Austria.

Wilensky, U. (1996). Making sense of probability through paradox and programming: A case study in a connected mathematics framework. In *Constructionism in practice: Designing, thinking, and learning in a digital world*. Mahwah, NJ: Lawrence Erlbaum.

Wilensky, U. (1999). GasLab: An extensible modeling toolkit for exploring micro-and-macro views of gases. In *Modeling and Simulations in Science and Mathematics Education* (pp. 151–178). New York: Springer-Verlag. doi:10.1007/978-1-4612-1414-4_7

Wilensky, U. (2001). Modeling nature's emergent patterns with multi-agent languages. In Proceedings of EuroLogo Linz, Austria (pp. 1–6).

Wilensky, U., & Reisman, K. (2006). Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories— an embodied modeling approach. *Cognition and Instruction*, *24*(2), 171–209. doi:10.1207/s1532690xci2402_1

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. doi:10.1145/1118178.1118215

Wing, J. M. (2011). *Research Notebook: Computational Thinking—What and Why? The Link. Spring*. Pittsburgh: Carnegie Mellon University.