Education Tech Research Dev (2014) 62:583–600
DOI 10.1007/s11423-014-9347-4

DEVELOPMENT ARTICLE

Learning problem-solving through making games at the game design and learning summer program

Mete Akcaoglu

Published online: 22 August 2014

Abstract Today's complex and fast-evolving world necessitates young students to possess design and problem-solving skills more than ever. One alternative method of teaching children problem-solving or thinking skills has been using computer programming, and more recently, game-design tasks. In this pre-experimental study, a group of middle school students ($n = 18$) with an age average of 12.6 attended a game-design summer program for 10 days. Students were assessed in their problem-solving skills, specifically in system analysis and design, decision-making, and troubleshooting domains, at the beginning and end of the program. The results indicated that there were significant improvements in students' problem-solving skills after attending the summer program, Wilks' $\Lambda = .258$, $F\ (3,\ 15) = 14.397$, $p < .001$, $\eta^2 = .742$. For system analysis and design, and decision-making follow-up t-tests pointed to large and medium effect sizes, while for troubleshooting the gains were not significant. This study is a contributes to the growing body of literature investigating the benefits of designing games for young children by adding that game-design activities can be suitable venues for young children to learn and practice problem-solving skills.

Keywords Problem-solving · Game-design · Constructionism · Programming · Kodu

Introduction

Our daily lives involve constant problem-solving (Funke and Frensch 1995). For this reason, problem-solving is considered to be one of the most important cognitive skills young students should possess to be successful in their future lives and careers (Jonassen 2000). Historically, schools have been expected to teach students methods of dealing with

M. Akcaoglu (✉)
Department of Leadership, Technology, and Human Development, College of Education, Georgia Southern University, Statesboro, GA 30458, USA
e-mail: makcaoglu@georgiasouthern.edu; mete.akca@gmail.com

the complex problems of the real world (Gagne 1980; Resnick 2010). They, however, have a different focus that contrasts with the nature of problem-solving in the real world, such as emphasizing symbolic thinking over direct engagement with objects, or teaching general skills and knowledge, as opposed to the situation-specific competencies that individuals benefit in out of school contexts (Resnick 1987). In addition, when students are given problems to solve at schools, they are generally well-structured (Jonassen 2000; Perkins 1986; Resnick and Rosenbaum 2013), not simulating the complex issues faced in the real world. This misalignment between what is emphasized in schools and what students face outside does not adequately prepare them for the challenges of the everyday and professional contexts (Jonassen 2000).

In the 1970s, the field of educational technology became interested in finding ways to use new technologies to foster children's thinking skills. Pioneering the field, Papert 1980 argued that, through computer programming (will be referred as programming henceforth), children showed improvements in their thinking skills. For example, it was believed that during the process of programming, students had opportunities to engage in deep thinking through debugging their codes, and persevere at this because they got chances to think about their own thinking and express their opinions through their creations (Guzdial 2004).

The promise of using programming to scaffold thinking skills has kept its prominence in the 21st century. Following the early work on the outcomes of learning simple textual programming languages, since the 1990 s, a plethora of new software has been developed to take advantage of the new graphical user interfaces. Today, with the advent of these new software applications, by simply dragging and dropping graphical programming commands on a computer screen, even young children can learn to code to create their own games or mobile applications.

In addition to being visually attractive, the new software has contextualized the abstract notion of programming into more concrete, inherently engaging, meaningful, and authentic contexts, such as game-design, story-telling, or animation-creation (Peppler and Kafai 2009). Among these contexts, game-design has stood out to be the most popular. For example, a computer game designed by a 13-year-old student was represented among many scientific projects at the White House Science Fair (Curtin 2013), pointing to the growing acceptance of game-design as an acceptable academic endeavor.

Recent research exploring the outcomes of learning game-design showed that through these tasks learners were able to engage in and learn programming (Baytak and Land 2010; Baytak and Land 2011) and computational thinking concepts (Denner et al. 2012). Other researchers showed that through game-design students improved in their learning and problem-solving perceptions (Hwang et al. 2013), and felt more positive about school subjects such as math (Ke 2014).

Researchers also argued that students could improve their problem-solving skills through game-design (e.g., Robertson 2012). This claim, however, did not find much empirical support. Given the importance of problem-solving for our children's future lives, the appeal and increasing availability of game-design activities for children, and that game-design process inherently involves design and problem-solving, using game-design to promote problem-solving skills can be considered as an ideal match. The purpose of this research, then, was to fill this gap by providing empirical support for the connection between learning game-design and students' problem-solving skills, especially in system analysis and design, troubleshooting, and decision-making. To this end, a pre-experimental study (one group pretest–posttest design) was conducted with the participation of students who attended a game-design summer program: Game Design and Learning (GDL)

program. As it will be detailed later, the GDL program sought to teach students digital game-design, while also offered activities that specifically targeted teaching of problem-solving skills. In this research, therefore, the improvement students showed in their problem-solving skills, after attending the GDL program, was of special interest.

Literature review

Benefits of learning game-design

Games are complex systems, made up of many interrelated variables (Fullerton 2008). In comparison to just playing games, the process of *creating* games is complex and cognitively demanding. This is because design tasks require bringing together many interrelated variables and parameters to create a complete and functional system (Denner et al. 2012; Fullerton 2008; Robertson 2012). Design tasks (including game-design) are, therefore, good examples of, and contexts to have practice in, solving ill-structured problems (Jonassen 2000).

Situating problem-solving within game-design tasks is appropriate for two main reasons. First, designing games is an intrinsically engaging task (Ke 2014). Children enjoy creating games, because they produce personally meaningful artifacts at the end of the game-design process. Second, being engaged in design requires constant problem-solving (Simon 1995). Engaging in ill-structured design problems, designers are frequently challenged to *make decisions*, create and analyze complex *systems*, come across new problems, and *troubleshoot* them as soon as they emerge. Game-design is, therefore, an ideal context to promote children's interest and development in important thinking skills, such as problem-solving (Prensky 2008).

Recent research has provided support for the cognitive and motivational benefits of learning game-design. For example, Hwang et al. 2013 found that compared to a group of students who followed a regular game-development curriculum, students who attended a peer assessment-based game-development course for 10 weeks significantly improved in their knowledge of scientific topics (e.g., environmental issues, global warming), as well as in their perceptions of learning and problem-solving abilities. In another recent research, Ke 2014 found that students who followed a game-design course for 6 weeks, where they created games to teach younger students math concepts, showed significant improvements in their attitudes toward math. The author also noted that, during the design process, the students engaged and persevered in effortful thinking, which does not frequently happen in regular school learning. Finally, Vos et al. 2011 found in their quasi-experimental research that young students (ages 10–12) who developed games, compared to the ones who only played games, showed significantly increased levels of intrinsic motivation toward learning (reported as more competence, more interest, and more effort), as well as higher levels of deep-learning strategy use.

Based on the results of recent research, we now know how learning game-design can lead to increases in students' content knowledge (e.g., Ke 2014), or their motivation to learn new skills such as problem-solving (e.g., Hwang et al. 2013). Although it is argued that students improve in their problem-solving skills as a result of engaging in game-design tasks, empirical support for this connection has mostly been in terms of changes in students' attitudes or perceptions (e.g., Hwang et al. 2013; Robertson 2012), but not in their actual skills. More specifically, we do not know if learning game-design has any positive

effects on students' actual problem-solving abilities, specifically in system analysis and design, decision-making, and troubleshooting.

Defining problem and problem-solving

Teaching a skill (e.g., problem-solving) through game-design requires instructional activities to be informed by the theories in the targeted domain. For example, when the curricular aim is to teach problem-solving through game-design, instructional activities benefit from being informed by theories of teaching problem-solving (Akcaoglu 2014). At the GDL program, the instructional activities sought to teach students game-design and also problem-solving skills. The activities were, therefore, based on theories of problem and problem-solving.

Problem

A problem is a situation where the aim is to reach a desired goal state from a given state but there is not an obvious method of doing so, due to the barriers in between (Mayer and Wittrock 1996). According to this definition, a problem is composed of a given state, a desired goal state, and obstacles in between (Funke 2010). Game-design tasks can be considered as problem situations because there is a desired goal (i.e., the game), and a given state (i.e., the initial idea for a game), but not just one way of reaching the goal state (i.e., obstacles in between).

Problem-solving

From a cognitive perspective, the problem-solving process involves successful execution of specific component cognitive processes: (a) understanding, (b) representing, (c) planning/monitoring, and (d) executing (Mayer and Wittrock 2006; Polya 1957). According to this perspective, in order to successfully solve a problem, one needs to, first, understand the problem. *Understanding* involves making sense of a given problem by using background knowledge. This process usually requires decomposing the problem into its components (e.g., goals, rules, and constraints) to make sense out of it. The next step in solving a problem is representing the problem. *Representing* refers to transforming the external representations of a problem (the components identified in the previous step) into internal mental representations (Jonassen 2000). This process involves generating hypotheses based on the relationships among the variables of the problem scenario at hand. After understanding the problem and creating a mental representation of it, the next steps involve *planning* a solution, and then *executing* the plan to solve the problem (Polya 1957). Finally, one needs to check, or evaluate, whether the solution has worked, and if not, go back and plan a new alternative.

Types of problems

Problems can vary in terms of the underlying processes involved (Jonassen 2000). Three important problem types that were investigated in this study (i.e., dependent variables) were (a) system analysis and design, (b) decision-making, and (c) troubleshooting. These problems were chosen among many other problem types because they are frequently faced during people's daily lives. In addition, these problems are a natural part of most science,

technology, engineering, and math (STEM) related careers (Jonassen 2000; OECD 2004). Given the rising importance of STEM careers for nations all over the world, providing students with hands-on practice in tackling these problems is important for educational institutions around the world (Jonassen 2004; OECD 2004).

System analysis and design involves (a) understanding how systems work by analyzing them, and (b) designing new systems by bringing together different variables to make one complete unit. These problems are commonly found in real-life settings (Jonassen 2000; Nelson 2003). For example, to be successful at playing games (digital or real-life), one needs to, first, understand the system behind them. In *Pac-Man*, for instance, figuring out the systematic relation between the enemy characters (ghosts) and Pac-Man is the first task. Successful completion of this task, hopefully, leads to success in the game. In real-life settings, people usually engage in system *analysis* when they are placed into new contexts. For example, analyzing how the transportation system works in a newly visited location is an example system analysis task. The goal here is to find the best, shortest, and cheapest route. The success at this task is only possible through understanding the transportation system in this new context. Similarly, system *design* tasks can be frequently found in our daily lives. System design involves satisfying many variables to create a single unit. The result is often open-ended (Jonassen 2000). An example system design task faced frequently by teachers is creating syllabi for their courses. To meet the predefined objectives of the course, the system design task here requires creating one functional system (i.e., the course) by putting together a number of variables: a list of readings, assignments, activities, grading system, weight of each assignment, etc.

Decision-making involves making the best decision by "understanding the given information, identifying relevant alternatives and constrains involved, constructing or applying external representations, selecting the best solution from a set of given alternatives, evaluating, [and finally,] justifying or communicating the decision" (OECD 2004, p. 163). Decision-making tasks are dilemmas, and our lives are filled with personal, social, and ethical dilemmas (Jonassen 2000). For example, trying to decide on what type of car to purchase requires one to make a decision by taking several constraints into account: budgetary restrictions, environmental factors (e.g., city vs. the country), desired specifications (e.g., gas-mileage), purpose (e.g., snow-plowing vs. commuting), safety issues, size, etc. The dilemma is when one cannot satisfy all the requirements but needs to pick and choose among the given variables to make the best decision with the least amount of compromises. For example, cars are generally fuel-efficient when they are smaller. So, a person living in the country and requiring a large vehicle (because of environmental and work-related reasons) will probably have to be less demanding in terms of gas-mileage and select a larger but less fuel-efficient vehicle. The process of game-design also requires designers frequently make important decisions during the process concerning the outcome. For example, deciding whether, how much, or what type of a story to put in a game depends on the genre of the game. So, if designers want to have narrative-driven games, they need to make a decision and choose a genre that lends itself to narratives. In this case, for example, a sports game would probably not work well, while an adventure game can prove to be a good choice.

Finally, *troubleshooting* requires "comprehending the main features of a system and to diagnose a faulty, or under-performing, feature of the system or mechanism" to bring it back to a working state again (OECD 2004, p. 168). The essence of successful troubleshooting is understanding how a designed system works (National Research Council 1999). Similar to system analysis and design, and decision-making problems, troubleshooting is also regularly faced in people's daily lives. Some example troubleshooting scenarios can

be: "inoperative modem; why won't car start; determine chemicals present in qualitative analysis; determine why newspaper article is poorly written; identify communication breakdowns in a committee…" (Jonassen 2000, p. 76). During game-design, troubleshooting skills are of great importance, as well. For example, in order to find out why a certain character does not perform the desired actions, the designers need to open the code page for that specific character and go through its code, line by line, to see where the problem might be stemming from. Having identified the problem, and making the necessary changes, the designer can easily run the code to see if the problem is fixed. If the problem persists, the designer needs to go back to the code page again and look for other lines that might be causing the problem.

Game-design and learning (GDL) program

The GDL program was offered to middle school students to teach them how to design digital games and also give them practice in solving complex problems. Although reaching the first goal was straightforward; reaching the second goal–giving students practice in solving complex problems–required different types of activities to be embedded in the game-design process. Therefore, at GDL, in addition to game-design activities that aimed to teach students game-design and basics of computer programming, activities that openly sought to teach students problem-solving skills were also offered (Akcaoglu 2014). This way, one of the biggest affordance of game-design tasks (i.e., student engagement) was used to make these other activities that originally lacked appeal (i.e., problem-solving) more engaging and appealing. GDL activities were, therefore, designed based on theories of teaching of problem-solving, as it will be detailed in the later sections.

The idea of creating activities that openly aim teaching students problem-solving skills is based on the previous research showing that students, especially young ones, have difficulty in abstracting patterns or thinking skills (e.g., Goldstone et al. 2005), especially when they are buried in or hidden under "fun" and engaging tasks. For example, in an early research, Pea and Kurland 1984 found that students failed to abstract important higher-order thinking skills when they were left alone to learn programming. Similarly, recent research found that students often failed to meaningfully embed content (e.g., math) into their games (Ke 2014), or failed to understand advanced programming commands (e.g., "while" or "if") while designing games (Denner et al. 2012). This shows that subtle references are both difficult for the students to identify, and the students often cannot think beyond the immediate task at hand, especially when it is such an engaging task like game-design.

In order to reach the curricular goals (i.e., teach game-design and teach problem-solving), four different types of activities were offered during the GDL program: (a) game-design, (b) problem-solving, (c) troubleshooting, and (d) free-design (Fig. 1). As it can be seen in Fig. 1, these activities were offered in a specific sequence: game-design activities were offered first, which were followed by problem-solving, troubleshooting, and free-design activities. The reason for this choice was reducing cognitive-load (Mayer and Wittrock 1996; Mayer and Wittrock 2006) by teaching students the basic skills in game-design and programming first, before they were offered more complex activities such as problem-solving. Accordingly, game-design activities served to teach students basics of the game-design software, as well as the basics of game-design and programming. This way, when introduced to problem-solving or troubleshooting activities later on, the students did not have to deal with the cognitive burden of simultaneously struggling with learning the basic skills of game-design and programming.

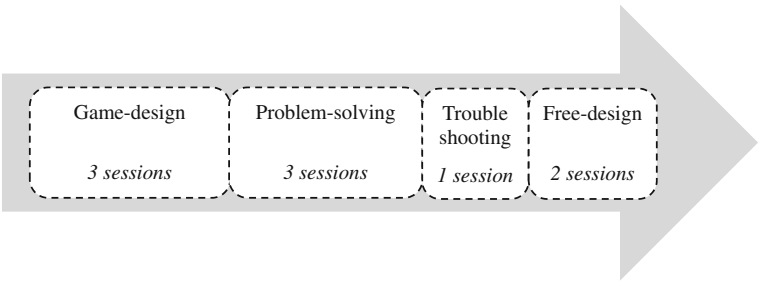

Fig. 1 Progression of activities during the GDL program

Game-design activities

Game-design activities served the goals of teaching students basics of game-design and programming, as well as teaching them how to use the game-design software: Microsoft Kodu. To this end, during the first three sessions of the GDL program, the students were guided in designing games, from simple to more complex. For example, while in the first session the students created a simple game called Apple Hunter, where the goal was to eat five apples; in the following sessions, the students were asked to design more complex games, such as Pac-Man or a tower-defense game.

During the game-design activities, instructors guided students, step-by-step, in programming their first games. The reason for this was the effectiveness of guided-discovery methods over pure discovery (Kirschner et al. 2006; Mayer 2004). Through the guidance of the instructor in this process, the students were able to learn the required basic game-design skills, while also getting chances to explore new game-design and programming concepts during the free time they had at the end of each session. During game-design activities, students also engaged in tasks that helped them to understand the systematic complexities of games, such as creating flowcharts. These tasks specifically aimed to provide students with practice in algorithmic thinking and system analysis and design (Klopfer et al. 2009).

Problem-solving activities

Problem-solving activities openly aimed to teach students skills in solving complex problems. To this end, these activities were built to utilize some of the effective methods of teaching problem-solving as detailed by Mayer and Wittrock in their extensive reviews (1996, 2006).

To teach students problem-solving skills in an engaging way, each problem-solving activity was designed and offered in the following manner: (a) students were introduced to a complex problem scenario, (b) they were, then, guided in solving it, and finally (c) they created a simulation replicating the scenario in the game-design software (Microsoft Kodu). During the initial steps, instructors modeled how students need to follow certain steps while solving complex problems (i.e., Polya 1957). This was based on *teaching thinking skills directly* method, where the goal is to support students in solving problems by explicitly teaching them important thinking skills (Mayer and Wittrock 1996; Mayer and Wittrock 2006). In the first problem-solving activity, for example, students were given a problem scenario regarding a trash situation in their school. By investigating some data first, students were tasked with finding the source of the problem, and then planning/

creating a simulation of it to share with an imaginary school management. During this task, students were constantly reminded to take their time to understand the source of the problem, before planning their solution. Instructor support was available when students had difficulty in reading the data or could not effectively follow the problem-solving steps.

Following identifying and understanding the source of the problem, students worked on planning how they could create the scenario in Kodu as a simulation. While creating their simulations students got chances to reflect on and evaluate their solutions, and also produce external visual representations of their solutions. Given the importance of simulations in science and scientific thinking, for example in testing hypotheses or making predictions (Klopfer et al. 2009), this process of simulation-creation carried additional importance. Through problem-solving activities, and simulations created, students had opportunities to openly see how complex problems can be solved, and they did this in an engaging manner through creating fun simulations.

Troubleshooting

After students got exposed to game-design, and problem-solving activities, during the final steps of the GDL program, students engaged in activities dedicated to troubleshooting. For example, they worked on troubleshooting broken games, where some codes or game elements were intentionally removed. Their task was, then, to go through the code of the game, identify the problems, and then, fix them to make the game work.

Free-design

Finally, in the last sessions of the GDL program, students were offered free-design activities where they created a game of their own choice by planning and programming it from scratch. During the free-design tasks, students first planned and sketched out their game ideas, created flowcharts of their games, and identified the elements (e.g., goals, rules, etc.) of their games. Then, they worked on creating their games in Kodu. During this process, although they worked individually, they were free to seek help from their peers and instructors.

In summary, at the GDL program, students were guided through a series of activities to teach them the basics of game-design and programming, while also giving them hands-on experiences in solving complex and meaningful problems, troubleshooting activities, and free-design. Readers interested in a broader discussion of the theory behind the design of the curriculum, or a more detailed account of instructional activities can find such detailed reports elsewhere (Akcaoglu 2014).

Course software: Microsoft Kodu

Microsoft Kodu is a game-design software that is specifically designed to provide an early entry to computer programming for children (MacLaurin 2011). The software was designed to "help users learn computer science concepts through game creation" (p. 100). It uses a simplified programming language, and allows creation of games as well as simulations.

Kodu was selected over other game or animation creation software for several reasons. First, as opposed to other available software (e.g., Scratch, Alice, Gamestar Mechanic),

Fig. 2 Screenshots of a Tower Defense Game in Scratch (*left*) and Kodu (*right*)

Kodu lets users create games in a three-dimensional (3D) environment. This feature makes Kodu visually more appealing for students, and allows for gaming experiences similar to their real-life ones (Fig. 2).

Second, Kodu uses a simplified programming language, which contains basic computer programming functions (Stolee and Fristoe 2011). Therefore, experience in Kodu environment provides students with an understanding of the basic concepts in computer programming. For example, Kodu introduces learners to basic programming concepts such as variables, Boolean logic, objects, and control flow, which are common and frequent programming functions.

Finally, Kodu programming is flexible enough to let users create simulations. As opposed to games, simulations can run by themselves, letting users observe and count user-defined events. This is especially useful when the instructional goal is to ask students to design or replicate real-life (or problem) scenarios, or visualize problem solutions (Klopfer et al. 2009). For example, a predator–prey relationship simulation can easily be programmed in Kodu, allowing learners to count and keep track of the populations of every type of character in the world. Simulations, in this sense, are ideal for creating external representations and reverse-engineering problem scenarios.

Purpose of the study

Programming, and recently game-design, has been lauded as popular methods of teaching thinking skills to young children. Empirical support for the cognitive benefits of game-design, however, has been slow to emerge. While some research was fraught with methodological weaknesses, such as only providing anecdotal evidence (e.g., Games and Kane 2011; Richards and Wu 2012; Wu and Richards 2011) others reported changes in students' content knowledge (e.g., Denner et al. 2012; Hwang et al. 2013), or their motivation to learn (e.g., Hwang et al. 2013; Ke 2014; Li 2013; Vos et al. 2011), but not in their actual skills, especially in problem-solving.

The purpose of this study was, therefore, to empirically examine the cognitive impacts of learning game-design on students' problem-solving skills. In this research, capturing the changes in students' abilities in solving three specific problem types (system analysis and design, decision-making, and troubleshooting) was the main goal. To this end, a pre-experimental study was conducted to investigate the following research question:

- Does attending the GDL program impact students' problem-solving skills in system analysis and design, troubleshooting, and decision-making domains?

Method

In order to answer the research question, this research was conducted with the participation of a group of students who attended the GDL summer program. The students were self-selected to attend the program. Achieving experimental conditions, such as randomization of the participants or the presence of another group to serve as the control group was not possible. In this research, thus, a pre-experimental design (i.e., one-shot pretest–posttest design—Campbell and Stanley 1963) was used.

Participants

Data for this study came from a group of students ($n = 21$) who attended the GDL summer program that was offered in Istanbul, Turkey. There were 13 male and eight female students. All students came from middle or upper-middle class families, and their age average was 12.6 (grades six through eight). All of the students were attending a private middle school. None of the students had previous knowledge of Microsoft Kodu or game-design. As indicated previously, these students self-selected to participate in the summer program.

Treatment

The treatment in this research was the GDL program that the students attended, and it was considered holistically. In other words, because changes in students' skills were measured at only two points (before and after the program) understanding the impact of specific activities offered during the program was not possible in this research. Due to this design, it was beyond the scope of this research to identify the specific role of individual activity types offered during GDL on students' problem-solving skills.

Dependent variables

There were three dependent variables in this research: students' skills in solving (a) system analysis and design, (b) decision-making, and (c) troubleshooting problems. As described previously, system analysis and design requires analyzing and understanding complex systems, and being able to design them to solve problems. Decision-making is when individuals face choices and have to make a decision based on constraints. Finally, troubleshooting requires understanding how a system works, and then identifying the element(s) that stopped the system to get it back to a working state again.

Instruments

In order to measure the changes in students' problem-solving skills, an internationally validated problem-solving assessment, prepared and offered by *The Program for International Student Assessment* (PISA), was used (OECD 2004). The instrument had a total of 19 items: system analysis and design (seven items), troubleshooting (five items), and

decision-making (seven items). The questions were in the form of multiple-choice and short-answer items. The PISA assessment is publicly available and can be accessed through the *Organisation for Economic Co-operation and Development* (OECD) website (OECD 2013).

The questions in the PISA test were designed to measure students' skills in solving complex problems in a generic sense (OECD 2004). In other words, the system analysis and design, decision-making, and troubleshooting questions were not contextualized in game-design. Therefore, the test was designed to assess students' skills in solving these problems in a generic sense, as they appeared in real-life contexts.

Students' ability in solving system analysis and design problems were measured by questions that involved analyzing or designing systems that were not contextualized in game-design, but were based on real-life scenarios. For example, a sample system analysis and design question was an item called "library system." In this question, the students were asked to draw the library system flowchart of an imaginary school, based on a paragraph explaining how students or staff were able borrow books from this library.

The second dependent variable was students' ability in solving decision-making problems. An example question that measured this in the PISA test was an item about "energy needs" of people from different walks of life. In this problem scenario, for example, students were asked to calculate and decide whether a meal would be enough to satisfy the energy needs of a given person based on that person's daily energy needs. To inform their decisions, the students had to read through data tables and graphs explaining the energy needs of people with different jobs (e.g., an athlete, a teacher, etc.), and calories of different foods that these people can choose as a part of their daily diets.

Finally, the third dependent variable was students' ability in solving troubleshooting problems. This ability was measured by, for example, a question that required students to analyze an irrigation system to find the faulty gate that prevented the water from flowing throughout all the pipes around a garden. Giving the correct answer required students to understand how the system worked and find out what was causing the system to fail.

Procedures

In order to measure the changes in students' problem-solving skills, the PISA problem-solving test was administered on the first and the last days of the GDL program, as a paper-and-pencil test. The students received the same version of the test on both occasions (pre and post). The students did not receive any feedback on their performance neither at the pretest nor the posttest. Students completed the test in approximately 40 min. On the test days, activities were resumed as normal following the tests.

The GDL program took place in a computer lab where each student had a Windows PCs running Microsoft Kodu. Three instructors (one lead and two assistants) ran the program. The lead instructor had designed the GDL curriculum and previous experience in teaching game-design courses. The assistant instructors were also knowledgeable in designing games in Kodu, and their main duty was to provide assistance and guidance to students when they needed support.

The GDL program lasted for 10 days, and met for approximately for 5 h each day. As explained previously (Fig. 1), during the program, 3 days were dedicated to game-design, while another three were dedicated to problem-solving activities. Troubleshooting activities were offered on day seven, and the following 2 days were dedicated to free-design activities where students designed a game of their choice from scratch. The final day of the GDL program was dedicated to student presentations, as well as a closing ceremony to

celebrate the students' success by getting their parents to join them to play their games, as well as to share their stories and experiences.

Data analysis

Due to missing data (i.e., missing the pretest or post-test), data from three of the 21 students were excluded from further analyses, making the total $n = 18$. The first step of the data analysis was grading the pretest and the posttest. The grading of all student tests was done following the conclusion of the GDL program.

The lead instructor graded the tests according to the answer key provided by PISA (OECD 2013). All of the test items had only one correct answer, as well as clear guidelines as to what is considered a partially correct answer for some questions. The grading was, therefore, done in an objective manner and was not open to grader judgment or bias.

In order to calculate the students' abilities in problem-solving, *Item Response Theory* (IRT) methods were implemented, according to the guidelines provided by the PISA assessment framework (OECD 2004). To this end, an item matrix was created using the provided item-difficulty parameters. Because there were both dichotomous (i.e., items with two possible outcomes) and polytomous items (i.e., items with more than two possible outcomes, such as a partial credit), a student data matrix, composed of scores of 0 (incorrect), 1 (partial for polytomous, or correct for dichotomous), and 2 (correct for partial), was created.

As suggested by the assessment framework, Partial Credit Model (Masters 1982) was used to calculate students' abilities in problem-solving. Inputting the item parameters and student data matrix into Xcalibre 4.1.8 software (Assessment Systems Corporation 2012), each student's ability estimate (theta) in the system analysis and design, decision-making, and troubleshooting domains were obtained. The problem-solving abilities were reported on a scale ranging from -4 to 4. This scale can roughly be interpreted as a student's probability of answering all the items on a test correctly. Specifically, a student with an ability level of 0, for example, would have 50 % probability of answering a question correctly. After calculating problem-solving ability estimates for each student in both the pretest and posttest, the data was analyzed using SPSS statistical package.

Results

To understand whether the students showed improvements in their problem-solving skills from pretest to posttest, a repeated-measures multivariate analysis of variance (RM-MANOVA), having two levels of time (pre vs. post) as within subjects factors was conducted on the dependent variables of system analysis and design, decision-making, and troubleshooting. The multivariate omnibus for time was significant, Wilks' $\Lambda = .258$, $F(3, 15) = 14.397$, $p < .001$, $\eta^2 = .742$, indicating that when all problem types are combined, there was a significant change in students' problem-solving skills from pretest to posttest. The multivariate Wilks' Λ was quite strong at .26, indicating a large effect size for the change.

As follow-up tests to the RM-MANOVA, and to understand the specific changes in system analysis and design, decision-making, and troubleshooting domains; paired-samples *t*-tests were conducted for each dependent variable. Using the Bonferroni method for controlling Type I error rates for multiple comparisons, each *t*-test was tested at the .017 level. This *p* value was calculated by dividing the critical alpha (.05) by the number

Table 1 Descriptive statistics based on the PISA test

	Items (n)	Pretest*		Posttest*	
		M	*SD*	*M*	*SD*
System analysis and design	7	−0.16	0.80	0.63	0.78
Decision-making	7	−0.27	1.30	0.50	0.83
Troubleshooting	5	0.00	1.16	0.38	0.78

* Scores range from −4 to 4, 4 is the highest skill level

($n = 3$) of the follow up tests (Field 2009). The *t*–tests for system analysis and design, $t\ (17) = 6.389$, $p < .001$; and decision-making, $t\ (17) = 3.034$, $p = .007$, were statistically significant. The *t*-test for troubleshooting, however, did not yield statistical significance, $t\ (17) = 1.383$, $p = .185$. This indicated that the students showed significant gains in two out of the three specific problem-solving skills (Table 1). As seen in Table 1, students' posttest scores were higher than their pretest scores in all of the three domains. In other words, compared to their ability in the pretest, in the posttest, the students improved in their ability to tackle the test questions in the three domains. For example, while in system analysis and design, the mean student ability was -0.16 in the pretest, the number was 0.63 in the posttest, indicating that students were significantly more than 50 % likely to get the questions correctly in this section.

Finally, Cohen's *d* (Cohen 1988) was calculated to understand the magnitude of the effect size for the changes in each dependent variable. The effect size for the change in system analysis and design was $d = 1.25$. For decision-making it was $d = 0.71$. These effect sizes indicate a large effect size for the changes in system analysis and design, and a medium effect size for decision-making.

In summary, the results indicated that the students showed significant gains in all three problem-solving skills, when combined. Detailed follow-up analyses showed that students made significant gains, with large effect sizes, in system analysis and design and decision-making, while their gains in troubleshooting did not reach statistical significance.

Discussion

Research providing empirical support for the impact of game-design on children's thinking skills have been slow to emerge (Denner et al. 2012), although the idea of using game-design in education has received a great amount of attention from both academics and practitioners over the past few years. This study was an important first step in understanding the impacts of learning game-design on students' actual problem-solving skills. More specifically, the results of this study indicated that the students who attended the GDL program showed significant improvements in their system analysis and design, and decision-making skills. These results point to the potential role of GDL curriculum in helping students develop their abilities in solving system analysis and design, and decision-making problems. This being said, it should be noted that this study incorporated a pre-experimental design, and therefore (as noted in more detail the limitations section) the results should be interpreted with caution.

The improvement in students' system analysis and design skills may be explained by the role of the GDL program and activities in teaching young students how to design and analyze complex systems. During the GDL program the students had numerous opportunities to tackle system analysis and design problems. More specifically, during the GDL

activities, students completed tasks that aimed to show them how games were complex systems. For example, during each activity at the GDL program, students were first asked to identify the elements of the given games or problem scenarios. They were, then, asked to create flowcharts of these complex systems, showing the relationship between the elements. Finally, they were asked to create games, and this final step helped students understand how games were complex systems. Through such tasks, students had hands-on practice in designing and analyzing systems, in a meaningful and engaging context like game-design. This engaging process of design may have had a role in improving students' skills in solving system analysis and design problems.

The results of this study also indicated that students who attended the GDL program showed improvements in their decision-making skills. One potential explanation for this is that during the game-design process, students often faced situations where they needed to make choices to make their games playable and fun. These engaging decision-making opportunities came naturally during the game-design process. These decisions sometimes came in the form of deciding on the genre of the game and what elements they needed to put in their games to conform to the nature of the selected genre. During this process, making sure that the choices did not damage the integrity of their games requires constant decision-making. Students were also forced to make decisions when programming in Kodu, especially when the software did not allow certain actions. In these cases, for example, the students needed to decide how to circumvent these shortcomings by finding alternative methods. Another frequent dilemma was when the students needed to decide how big their game-worlds should be, depending on the processing power of their computers. Depending on their computers' processing power, the students needed to make decisions to remove some elements from their games, while making sure that all necessary components were still there.

Despite the gains in system analysis and design and decision-making skills, the results indicated that the change in students troubleshooting skills was not at a statistically significant level, and the effect size was small. One possible explanation for this lack of improvement could be a potential misalignment between the items on the test and the actual troubleshooting situations faced during game-design. For example, while troubleshooting during game-design meant identifying a misplaced line of code, the questions at the test required the students to follow descriptions of unfamiliar systems and to identify parts causing the problem. In addition, during the game-design process students received immediate feedback (e.g., run the code to see if the game is fixed) for their troubleshooting attempts, while it was not possible during the test. In addition to the possible mismatch between test questions and troubleshooting during game-design, another reason for the lack of significant change in troubleshooting could be due to dedicating only one activity explicitly for this task. It may be argued that an addition of activities openly teaching students generalizable troubleshooting skills could lead to much improved performance in this domain.

Understanding the impact of specific GDL activities was beyond the scope of this research, and the GDL program was treated as one single intervention. Therefore, the results of this study need to be interpreted within the context of GDL curriculum and activities offered within the GDL program. At this point it is hard to understand the impact of specific instructional methods and their unique affordances on the outcomes. More specifically, we do not know how the individual instructional methods or activities might have contributed to the changes in students' system analysis and design and decision-making skills, and if these methods or activities are additive, interactive, or independent. From early research on programming, it is known that differences in instructional methods

used in teaching programming can lead to differences in learning outcomes (e.g., Lehrer et al. 1999). Important questions, therefore, remain unanswered about the details of the GDL program, and need to be scrutinized further in future research.

Limitations

By design, this study had some important limitations, and results should be interpreted with caution. First, the participants in this study were self-selected, and because the GDL program was offered as a summer course finding other students to serve as a control group was not possible. This investigation, therefore, was not an experimental study and was prone to threats of internal and external validity (Campbell and Stanley 1963). More specifically, since there was not a control group it is hard to know if the gains students showed were confounded by other variables. One important concern here is the effect of testing (Campbell and Stanley 1963). *Testing*, in other words, having taken a test previously, might lead to better performance of the subject in the posttest. Future research should seek to eliminate these rival hypotheses to get a clearer picture of the effects of the GDL program. More specifically, there is a specific need for a study with a control group to eliminate the effects of testing as a rival hypothesis.

In addition, the participants in this study were all coming from upper middle class families. It might be argued that these students had important characteristics, due to their socio-economic status (e.g., access to technology), that helped them benefit from the GDL program at the levels that they did. Therefore, the study should be replicated in different contexts in order to understand if and how students' personal backgrounds (e.g., learning styles) impact how much they gain from the game-design courses.

The possibility of a misalignment between the problem-solving skills measured in the PISA test and the skills that may be utilized during game-design should also be considered as a limitation. More specifically, the problem-solving skills measured by PISA were domain-general, in that students solved problems that represented various problems that could be frequently faced during their daily lives. As suggested (e.g., Mayer 1992), some problem-solving skills are considered to be domain-specific, requiring the problem-solver to use contextualized problem-solving skills, and may not be easily executable in other domains. In a future study, students' domain-specific problem-solving skills (e.g., problem-solving skills used during game-design) should be measured. This may help capture improvement beyond what was possible by the domain-general PISA assessment.

Finally, as explained previously, the participants in this study were self-selected to attend the GDL program. Their motivations to learn, therefore, can be different than the students in a regular classroom, or the students who chose not to participate in the program. Therefore, readers should be cautious when generalizing the results reported here to other populations.

Conclusions and implications

The results of this study have important implications for theory and practice. First and foremost, the results indicate that, through a curriculum and instructional activities based on effective methods of teaching problem-solving, game-design could be a good context to teach complex problem-solving skills. It was also seen that, through the game-design process, teaching skills in solving multiple types of problems can be also possible. More

specifically, the results indicate that system analysis and design and decision-making may be practiced and improved through game-design tasks.

In terms of implications, it might be suggested that, in a similar manner to teaching problem-solving skills at GDL, teaching of content knowledge may also be possible by putting a content layer onto game-design tasks. For example, important STEM topics such as environmental problems (e.g., ecological literacy) may be integrated into game-design curriculum and activities. In fact, recent research has already showed that through game-design, students can be taught concepts in science (Baytak and Land 2011; Hwang et al. 2013), math (Ke 2014; Li 2013), or literature (Robertson 2012).

Learning though design is a powerful method of learning (Harel 1991). Through design, children get invaluable chances to play and tinker with objects and finally create something they value. They also learn about their own learning and thinking (Papert 1980). The present research supports this ethos and gives us indications that game-design can be a good platform to teach problem-solving skill. At the GDL program, activities were designed to carefully incorporate both theories of problem-solving and effective methods of teaching game-design. Future interventions seeking similar outcomes should take into account the design process and decisions in the making of the GDL curriculum and activities.

Acknowledgments This paper was based on the author's doctoral dissertation, completed at Michigan State University in 2013 under the supervision of Matthew J. Koehler, and with support from Cary J. Roseth, Carrie Heeter, and Christina Schwarz. I also would like to thank Matthew D. Boyer, Kristen DeBruler, and Tyler DeBruler in running the GDL programs.

References

Akcaoglu, M. (2014). Teaching problem solving through making games: Design and implementation of an innovative and technology-rich intervention. In M. Searson & M. Ochoa (Eds.), In *Proceedings of Society for Information Technology & Teacher Education International Conference 2014* (pp. 597–604). Chesapeake, VA: AACE. Retrieved from http://www.editlib.org/p/130820.

Assessment Systems Corporation. (2012). Xcalibre 4.1.6.1. Retrieved from http://www.assess.com/xcart/product.php?productid=569.

Baytak, A., & Land, S. M. (2010). A case study of educational game design by kids and for kids. *Procedia - Social and Behavioral Sciences, 2*(2), 5242–5246. doi:10.1016/j.sbspro.2010.03.853.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59*(6), 765–782. doi:10.1007/s11423-010-9184-z.

Campbell, D. T., & Stanley, J. C. (1963). *Experimental and quasi-experimental designs for research*. Boston: Houghton Mifflin Company.

Cohen, J. (1988). *Statistical power analysis for behavioral science* (2nd ed.). Hillsdale, New Jersey: Lawrance Erlbaum Associates Inc, Publishers.

Curtin, C. (2013, April 24). Young Kodu designer showcases at 2013 White House Science Fair. *Microsoft Citizenship Blog*. Retrieved February 13, 2014, from http://blogs.technet.com/b/microsoftupblog/archive/2013/04/24/2013-white-house-science-fair.aspx.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education, 58*(1), 240–249.

Field, A. (2009). *Discovering statistics using IBM SPSS Statistics*. London: SAGE Publications Ltd.

Fullerton, T. (2008). *Game design workshop: A playcentric approach to creating innovative games*. Boston, MA: Elsevier.

Funke, J. (2010). Complex problem solving: A case for complex cognition? *Cognitive Processing, 11*(2), 133–142.

Funke, J., & Frensch, P. (1995). Complex problem solving research in North America and Europe: An integrative review. *Foreign Psychology, 5*, 42–47.

Gagne, R. M. (1980). *The conditions of learning*. New York: Holt, Rinehart, and Winston.
Games, I. A., and Kane, L. P. (2011). Exploring adolescent's STEM learning through scaffolded game design. In *Proceedings of the 6th International Conference on Foundations of Digital Games* (pp. 1–8). New York: ACM. doi:10.1145/2159365.2159366.
Goldstone, R., Son, J. Y. J. Y., & Robert, L. (2005). The transfer of scientific principles using concrete and idealized simulations. *The Journal of the Learning Sciences, 14*(1), 69–110.
Guzdial, M. (2004). Programming environments for novices. In S. Fincher & M. Petre (Eds.), *Computer science education research* (pp. 1–16). The Netherlands: Taylor & Francis.
Harel, I. (1991). *Children designers: Interdisciplinary constructions for learning and knowing mathematics in a computer-rich school*. New Jersey: Ablex Publishing Corporation.
Hwang, G.-J., Hung, C.-M., & Chen, N.-S. (2013). Improving learning achievements, motivations and problem-solving skills through a peer assessment-based game development approach. *Educational Technology Research and Development,*. doi:10.1007/s11423-013-9320-7.
Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development, 48*(4), 63–85.
Jonassen, D. H. (2004). *Learning to solve problems: An instructional design guide*. San Francisco, CA: Pfeiffer.
Ke, F. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, *73*(1), 26–39. doi: dx.doi.org/10.1016/j.compedu.2013.12.010.
Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41*(2), 75–86. doi:10.1207/s15326985ep4102_1.
Klopfer, E., Scheintaub, H., Huang, W., Wendel, D., & Roque, R. (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.
Lehrer, R., Lee, M., & Jeong, A. (1999). Reflective teaching of LOGO. *The Journal of the Learning Sciences, 8*(2), 245–289.
Li, Q. (2013). Digital game building as assessment: A study of secondary students' experience. *Developments in Business Simulation and Experiential Learning, 40*, 74–78.
MacLaurin, M. B. (2011). The design of Kodu: A tiny visual programming language for children on the Xbox 360. *ACM SIGPLAN Notices*, *46*(1).
Masters, G. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149–174.
Mayer, R. E. (1992). *Thinking, problem solving, cognition* (2nd ed.). New York: Freeman.
Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? The case for guided methods of instruction. *The American Psychologist, 59*(1), 14–19. doi:10.1037/0003-066X.59.1.14.
Mayer, R. E., & Wittrock, M. C. (1996). Problem-solving transfer. In D. C. Berliner & R. C. Calfee (Eds.), *Handbook of educational psychology* (pp. 47–62). New York: Macmillan Library Reference.
Mayer, R. E., & Wittrock, M. C. (2006). Problem solving. In P. A. Alexander & P. H. Winne (Eds.), *Handbook of educational psychology* (pp. 287–303). New Jersey: Lawrence Erlbaum Associates.
National Research Council. (1999). *Being fluent with information technology*. Washington, DC: National Academies Press.
Nelson, W. A. (2003). Problem solving through design. *New Directions for Teaching and Learning, 2003*(95), 39–44. doi:10.1002/tl.111.
OECD. (2004). *The PISA 2003 Assessment Framework: Mathematics, Reading, Science and Problem Solving Knowledge and Skills*. PISA: OECD Publishing. doi:10.1787/9789264101739-en.
OECD. (2013). Test questions - PISA 2003. Retrieved from http://www.oecd.org/education/school/programmeforinternationalstudentassessmentpisa/34009000.pdf.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books Inc.
Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137–168.
Peppler, K. A., & Kafai, Y. B. (2009). Gaming fluencies: Pathways into participatory culture in a community design studio. *International Journal of Learning and Media, 1*(4), 45–58.
Perkins, D. N. (1986). *Knowledge as design*. New Jersey: Lawrance Erlbaum Associates Inc, Publishers.
Polya, G. (1957). *How to solve it*. New Jersey: Princeton University Press.
Prensky, M. (2008). Students as designers and creators of educational computer games: Who else? *British Journal of Educational Technology, 39*(6), 1004–1019.
Resnick, L. B. (1987). The 1987 presidential address: Learning in school and out. *Educational Researcher, 16*(9), 13–20.

Resnick, L. B. (2010). Nested learning systems for the thinking curriculum. *Educational Researcher, 39*(3), 183–197. doi:10.3102/0013189X10364671.

Resnick, M., & Rosenbaum, E. (2013). Designing for tinkerability. In M. Honey & D. Kanter (Eds.), *Design, make, play: Growning the next generation of STEM innovators* (pp. 163–181). New York: Routledge.

Richards, K., & Wu, M. L. (2012). Learning with educational games for the intrepid 21st century learners. In P. Resta (Ed.), In *Proceedings of Society for Information Technology & Teacher Education International Conference* (pp. 55–74). Chesapeake, VA: AACE.

Robertson, J. (2012). Making games in the classroom: Benefits and gender concerns. *Computers & Education, 59*(2), 385–398. doi:10.1016/j.compedu.2011.12.020.

Simon, H. A. (1995). Problem forming, problem finding, and problem solving in design. In A. Collen & W. W. Gasparski (Eds.), *Design and systems: General applications of methodology* (Vol. 3, pp. 245–257). New Jersey: Transaction Publishers.

Stolee, K. T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu Game Lab. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 99–104).

Vos, N., van der Meijden, H., & Denessen, E. (2011). Effects of constructing versus playing an educational game on student motivation and deep learning strategy use. *Computers & Education, 56*(1), 127–137. doi:10.1016/j.compedu.2010.08.013.

Wu, M. L., & Richards, K. (2011). Facilitating computational thinking through game design. In M. Chang, W. Y. Hwang, M. P. Chen, & W. Müller (Eds.), *Edutainment technologies, educational games and virtual reality/augmented reality applications* (pp. 220–227). Heidelberg: Springer, Berlin.

Mete Akcaoglu is assistant professor of Instructional Technology at the College of Education at Georgia Southern University. His scholarly interests include the design and evaluation of technology-rich and innovative learning environments for K-12 children. He can be found at http://meteakcaoglu.com